i.MX23 Applications Processor Reference Manual

IMX23RM
Rev. 1
11/2009

freescale™
semiconductor

How to Reach Us:

Home Page:

www.freescale.com

Web Support:

http://www.freescale.com/support

USA/Europe or Locations Not Listed:

Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or
+1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:

Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:

Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064
Japan
0120 191014 or
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:

Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 010 5879 8000
support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
+1-800 441-2447 or
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale and the Freescale logo are trademarks or registered trademarks of Freescale Semiconductor, Inc. in the U.S. and other countries. All other product or service names are the property of their respective owners. ARM is the registered trademark of ARM Limited. ARM926EJ-S is the trademark of ARM Limited.

© Freescale Semiconductor, Inc., 2009. All rights reserved.

Document Number: IMX23RM
Rev. 1, 11/2009

Contents

Contents

Contents

Contents

Contents

Contents

Chapter 13
General-Purpose Media Interface (GPMI)

Contents

Contents

Contents

Chapter 18
LCD Interface (LCDIF)

Contents

Contents

Contents

Contents

Contents

Contents

Chapter 33
Low-Resolution ADC and Touch-Screen Interface

Contents

Contents

Contents

Contents

Contents

Figures

Figures

Figures

Figures

Figures

Figures

Tables

Tables

Tables

Tables

Chapter 1
Product Overview

The i.MX23 is an applications processor targeted at devices that require low power, high performance, high integration and quality audio and video playback.

This chapter provides a general overview of the i.MX23 product and describes hardware features, application capability, design support, and additional documentation. See Table 1-1, and the pinout information in Chapter 36, "Pin Descriptions," for more detailed information about which functions described later in this document are supported in which package and part number.

Table 1-1. i.MX23 Functions by Package

Function	LQFP128	BGA169
External memory interface (2.5 V DDR, 1.8 V mDDR)	64MB Maximum, 16-bit data, 13-bit address, 1 chip enable	128MB Maximum 16-bit data, 13-bit address, 2 chip enable
General-Purpose Media Interface (GPMI): • NAND data width • Number of external NANDs supported	 8-bit data 4 (2 dedicated, 2 muxed)	 16-bit data 4 (3 dedicated, 1 muxed)
LCD Interface (LCDIF): • Up to 24-bit full-color parallel RGB mode • Up to 18-bit parallel RGB mode • 8-bit serial RGB mode • ITU-R BT.656 8-bit+clock mode • 8-bit system mode • Up to 18-bit parallel system mode • Up to 24 bit parallel system mode	 No No Yes Yes Yes No No	 Yes /w 8-bit NAND Yes Yes Yes Yes Yes Yes
Serial Audio Interface (SAIF or I^2S): • Interfaces supported (Note SAIF1/SAIF2 are muxed with LCD_DATA[8:16])	0	2
SPDIF Transmitter	No	Yes
Low-Resolution ADC (LRADC): • Number supported • Touch-screen supported	 2 (or 3 without 2.5 V DDR) No	 6 Yes
Application UART2: • Supported via dedicated pins	 No	 Yes

Table 1-1. i.MX23 Functions by Package (continued)

Function	LQFP128	BGA169
Synchronous Serial Port 1 (SSP1): • Data width	4-bit data	8-bit data
Synchronous Serial Port 2 (SSP2): • Data width (Note SSP2 is muxed with GPMI/NAND)	8-bit data	8-bit data
Embedded Trace Macrocell (ETM)	No	Yes
Pulse Width Modulation (PWM) Channels	3	5
Rotary Encoder	Yes (Muxed with PWM / DEBUG UART pins)	Yes (Dedicated pins)
Mono speaker amplifier	No	Yes
Real-Time Clock (RTC)	24 MHz	32 kHz and 24 MHz
Power Supply	Li-Ion	Li-Ion
4.2 V Regulated Output	Yes	Yes
Single Channel 10-bit Video DAC (Composite output)	Yes	Yes

1.1 Hardware Features

- ARM926 CPU Running at 454 MHz
  - — Integrated ARM926EJ-S CPU
  - — 16-Kbyte data cache and 16-Kbyte instruction cache
  - — ARM Embedded Trace Macrocell (ETM CoreSight 9) (169BGA only)
  - — One-wire JTAG interface
  - — Resistor-less boot mode selection using integrated OTP values
- 32 Kbytes of Integrated Low-Power On-Chip RAM
- 64 Kbytes of Integrated Mask-Programmable On-Chip ROM
- 1 Kbit of On-Chip One-Time-Programmable (OCOTP) ROM
- Universal Serial Bus (USB) High-Speed (Up to 480 Mb/s), Full-Speed (Up to 12 Mb/s)
  - — Full-speed/high-speed USB device and host functions
  - — Fully integrated full-speed/high-speed Physical Layer Protocol (PHY)
  - — Mass storage host-capable (uncertified by USB-IF)
- Power Management Unit
  - — Single inductor DC-DC switched converter with multi-channel output supporting Li-Ion batteries.
  - — Features multi-channel outputs for VDDIO (3.3 V), VDDD (1.2 V), VDDA (1.8 V), VDDM (2.5V) and regulated 4.2V source.
  - — Direct power from 5-V source (USB, wall power, or other source), with programmable current limits for load and battery charge circuits.

  - Silicon speed and temperature sensors enable adaptive power management over temperature and silicon process.
- Audio Codec
  - Stereo headphone DAC with 99 dB SNR
  - Stereo ADC with 85 dB SNR
  - Stereo headphone amplifier with short-circuit protection and direct drive to eliminate bulky capacitors
  - Mono speaker amplifier (169-Pin BGA only) providing up to 2W rms output, running directly from the battery.
  - Amplifiers are designed for click/pop free operation.
  - Two stereo line inputs
  - Microphone input
  - SPDIF digital out
- 16-Channel Low-Resolution ADC
  - 6 independent channels and 10 dedicated channels
  - Resistive touchscreen controller
  - Temperature sensor controller
  - Absolute accuracy of 1.3%
  - Up to 0.5% with bandgap calibration
- Security Features
  - Read-only unique ID for digital rights management algorithms
  - Secure boot using 128-bit AES hardware decryption
  - SHA-1 hashing hardware
  - Customer-programmed (OTP) 128 bit AES key is never visible to software.
- External Memory Interface (EMI)
  - Provides memory-mapped (load/store) access to external memories
  - Supports the following types DRAM:
    - 1.8-V Mobile DDR
    - Standard 2.5V DDR1
- Wide Assortment of External Media Interfaces
  - Up to four NAND flash memories with hardware management of device interleaving
  - High-speed MMC, secure digital (SD)
  - Hardware Reed-Solomon Error Correction Code (ECC) engine offers industry-leading protection and performance for NANDs.
  - Hardware BCH ECC engine allowing for up to 20-bit correction and programmable redundant area.
- Dual Peripheral Bus Bridges with 18 DMA Channels
  - Multiple peripheral clock domains save power while optimizing performance.

  - Direct Memory Access (DMA) with sophisticated linked DMA command architecture saves power and off-loads the CPU.
- Highly Flexible Display Controller
  - Up to 24-bit RGB (DOTCK) modes
  - Up to 24-bit system-mode including VSYNC and WSYNC modes.
  - Up to VGA (640x480) resolution at 60Hz LCD panel support
  - 8-bit data ITU-R BT.656 D1 digital video stream output mode (PAL/NTSC), with on-the-fly RGB to YCbCr color-space-conversion.
  - Flexible input formats
- Pixel Processing Pipeline (PXP)
  - Provides full path from color-space conversion, scaling, alpha-blending to rotation without intermediate memory access
  - Bi-linear scaling algorithm with cropping and letterboxing
  - Alpha-blend, BITBLT, color-keying
  - Memory efficient block-based rotation engine
  - Supports up to eight overlays
- Integrated TV-Out Support
  - Integrated PAL/NTSC TV-encoder fully pipelined to display controller's D1 resolution output stream
  - Integrated low-power 10-bit Video DAC (VDAC) for composite analog video output.
- Data Co-Processor (DCP)
  - AES 128-bit encryption/decryption
  - SHA-1 hashing
  - High-speed memory copy
- Three Universal Asynchronous Receiver-Transmitters (UARTs)
  - Two high-speed application UARTs operating up to 3.25 Mb/s with hardware flow control and dual DMA.
  - Debug UART operates at up to 115Kb/s using programmed I/O.
- I^2C Master/Slave
  - DMA control of an entire EEPROM or other device read/write transaction without CPU intervention
- Dual Synchronous Serial Ports (for SPI, MMC, SDIO, Triflash)
  - Up to 52MHz external SSP clock for all modes, including SPI
  - 1-bit, 4-bit and 8-bit MMC/SD/SDIO modes
  - Compliant with SDIO Rev. 2.0
  - SPI with single, dual and quad modes.
- Four-Channel 16-Bit Timer with Rotary Decoder
- Five-Channel Pulse Width Modulator (PWM)

- Real-Time Clock
  - Alarm clock can turn the system on.
  - Uses the existing 24-MHz XTAL for low cost or optional low power crystal (32.768 kHz or 32.0 kHz), customer-selectable via OTP.
- SPDIF Transmitter
- Dual Serial Audio Interface (SAIF), Three Stereo Pairs
  - Full-duplex stereo transmit and stereo receive operations
  - Cell phone baseband processor connection and external ADCs and DACs
  - Bluetooth hands-free connection
  - Analog I/O for peripheral bus breakouts
  - I^2S, left-justified, right-justified, and non-standard formats
- Customer-Programmable One-Time-Programmable (OTP) ROM via Integrated eFuse Block
  - Resistor-less boot mode selection
  - 128-bit boot mode crypto key
  - Boot mode specification of NAND characteristics for device that the customer is soldering to the board. This means no more costly delays waiting for new device support in the boot ROM.
  - Fully software-programmable and accessible
- Flexible I/O Pins
  - All digital pins have drive-strength controls as described in Section 37.2.2.1, "Pin Drive Strength Selection."
  - Most non-EMI digital pins have general-purpose input/output (GPIO) mode.
- Offered in 128-Pin Low-Profile Quad Flat Pack (LQFP), and 169-Pin Ball Grid Array (BGA)

1.2 i.MX23 Product Features

The i.MX23 offers long battery life, minimal external components through integration, high processing performance, and excellent software development and debug support. The i.MX23 is especially suited for multi-media applications requiring audio/video decode and rich display support. These requirements are achieved via the high-performance CPU, pixel processing and integrated display and TV-Out hardware.

Figure 1-1 shows a block diagram of a typical system based on the i.MX23.

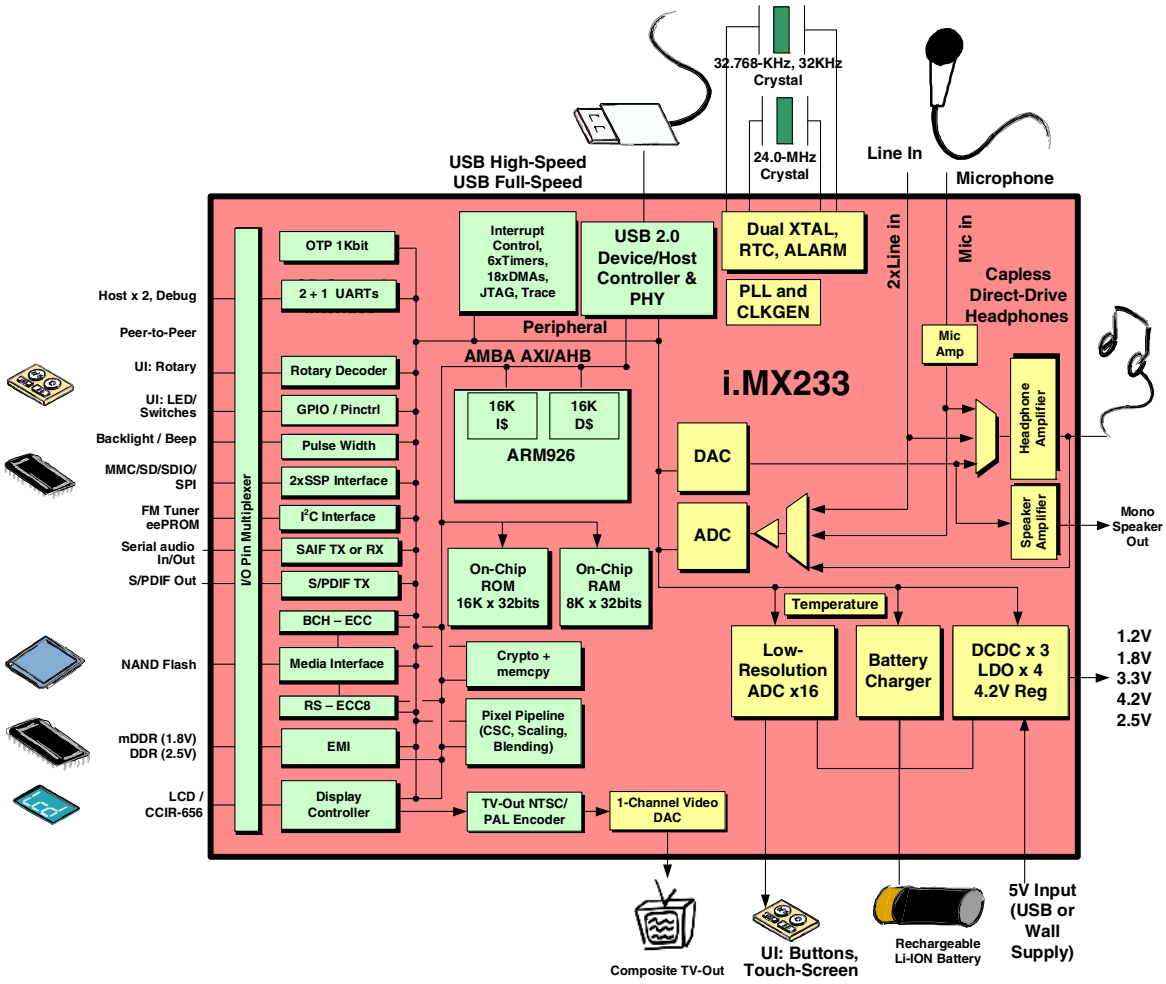


Figure 1-1. System Block Diagram

The i.MX23 features low power consumption to enable long battery life in portable applications. The integrated power management unit includes a high-efficiency, on-chip DC-DC converter. The power management unit also includes an intelligent battery charger for Li-Ion cells and is designed to support adaptive voltage control (AVC), which can reduce system power consumption by half. AVC also allows the chip to operate at a higher peak CPU operating frequency than typical voltage control systems. The DC-DC converters and the clock generator can be reprogrammed on-the-fly to trade off power versus performance dynamically.

To provide the maximum application flexibility, the i.MX23 integrates a wide range of I/O ports. It can efficiently interface to nearly any type of flash memory, serial peripheral bus, or LCD. It is also ready for advanced connectivity applications such as Bluetooth and WiFi via its integrated 4-bit SDIO controller and high-speed (3.25 Mb/s) UARTs.

The i.MX23 also integrates an entire suite of analog components, including a high-resolution audio codec with headphone amplifier, 16-channel 12-bit ADC, 10-bit Video DAC, Mono Speaker Amplifier, high-current battery charger, linear regulators for 5-V operation, high-speed USB Host PHY, and various system monitoring and infrastructure systems.

An ARM 926 EJ-S CPU with 32 Kbytes of on-chip SRAM and an integrated memory management unit provides the processing power needed to support advanced features such as audio cross-fading, as well as still picture and video decoding.

Execution always begins in on-chip ROM after reset, unless overridden by the debugger. A number of devices are programmed only at initialization or application state change, such as DC-DC converter voltages, clock generator settings, etc. Certain other devices either operate in the crystal clock domain or have significant portions that operate in the crystal clock domain, e.g., ADC, DAC, PLL, etc. These devices operate on a slower speed asynchronous peripheral bus. Write posting in the ARM core, additional write post buffering in the peripheral AHB, and set/clear operations at the device registers make these operations efficient.

1.2.1 ARM 926 Processor Core

The on-chip RISC processor core is an ARM, Ltd. 926EJ-S. This CPU implements the ARM v5TE instruction set architecture. The ARM9EJ-S has two instructions sets, a 32-bit instruction set used in the ARM state and a 16-bit instruction set used in Thumb state. The core offers the choice of running in the ARM state or the Thumb state or a mix of the two. This enables optimization for both code density and performance. The ARM CPU is described in Chapter 3, “ARM CPU Complex.”

The ARM RISC CPU is the central controller for the entire i.MX23 SOC, as shown in Figure 1-2. The ARM 926 core includes two AHB masters:

- AHB1—Used for instruction fetches
- AHB2—Used for data load/stores, page table accesses, DMA traffic, etc.

1.2.2 System Buses

The i.MX23 uses buses based on ARM’s Advanced Microcontroller Bus Architecture (AMBA) for the on-chip peripherals. The AMBA2 specification outlines two bus types: AHB and APB. The AMBA3 specification additionally outlines the AXI fabric.

- AXI is the highest-performance AMBA bus that supports de-coupled R/W channels, multiple outstanding transactions, and out-of-order data capability. This leads to higher performance and more efficient use of external memory.
- AHB is a higher-performance bus that supports multiple masters such as the CPU and DMA controllers.
- The APB is a lower-speed peripheral bus.

As shown in Figure 1-2, the i.MX23 uses a three-layer AHB, a high-performance AXI segment and two APBs: APBH and APBX. The APB buses are enhanced to include byte-write capability.

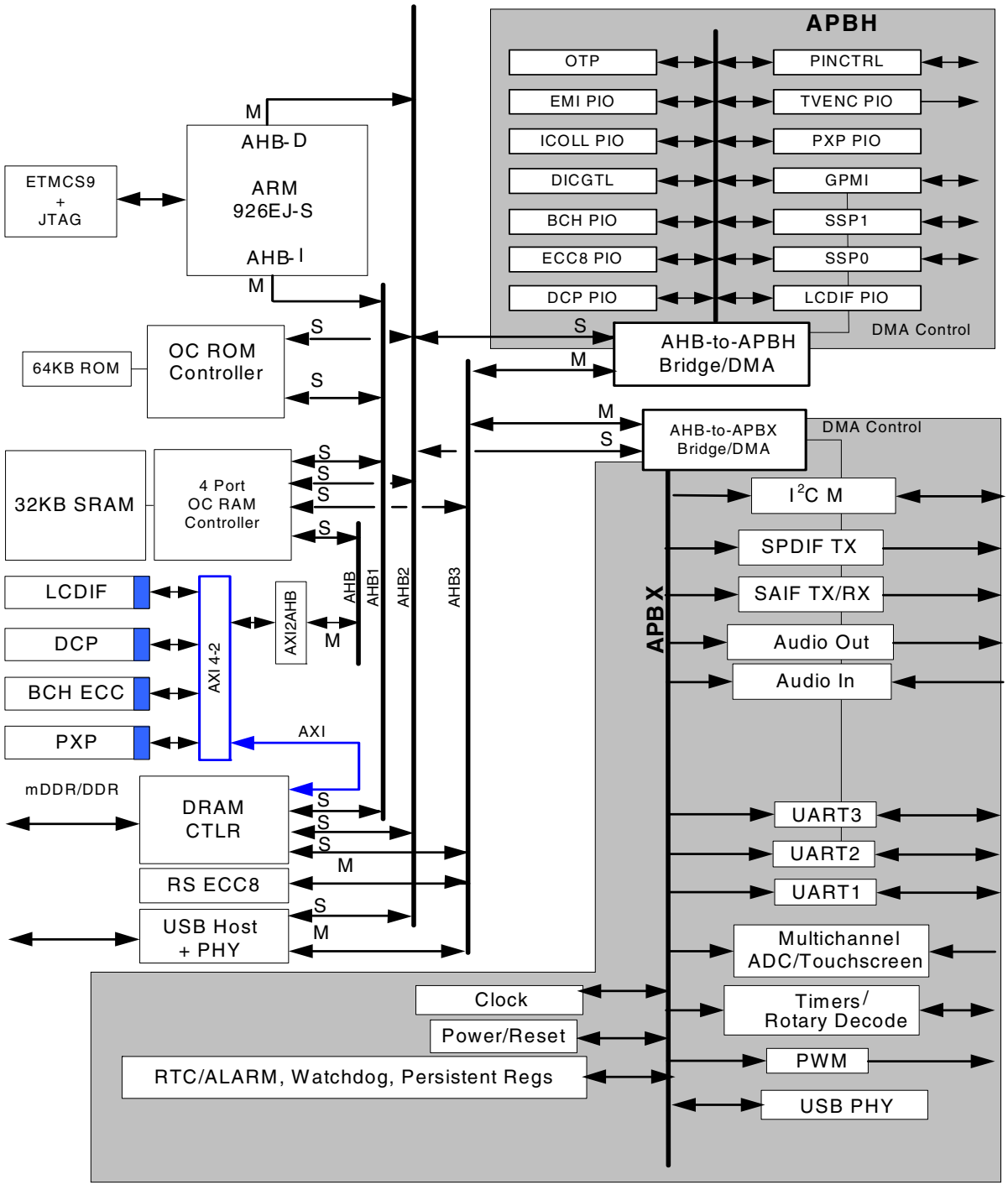


Figure 1-2. i.MX23 SoC Block Diagram

1.2.2.1 AXI Bus

The AXI bus-segment on i.MX23 provides several high-bandwidth/performance-critical peripherals a tightly-coupled and efficient interface to Port-0 of the external memory controller. The peripherals are as follows:

- DCP (Crypto/Memcpy)
- PXP (Pixel Processing Pipeline)
- LCDIF (Display Controller)
- BCH-ECC Engine

This connection also allows for a path to the On-Chip RAM and EMI as shown in the figure. The AXI bus-segment allows for each peripheral to issue multiple outstanding transactions allowing for higher-performance and more efficient memory bus usage.

1.2.2.2 AHB Bus

The AHB is the main high-performance system bus and is implemented in three layers, as follows:

- Layer 1 (AHB1)—CPU instruction access to OCRAM, OCROM, EMI
- Layer 2 (AHB2)—CPU data access to OCRAM, OCROM, all bridges, USB slaves, EMI
- Layer 3 (AHB3)—APBH DMA, APBX DMA, RS-ECC8 and USB masters, EMI

The ARM926 instruction bus (AHB1) is a single-master layer, as is the ARM926 data bus (AHB2). The other two layers have multiple masters, as shown in Figure 1-2.

The ARM926 data bus connects to the all slaves in the system, including RAMs, ROMs, bridge slaves, and USB slaves. The APB peripherals can act as AHB slaves through the AHB-APB bridge. The AHB has seven slaves:

- USB slave
- On-chip RAM
- On-chip ROM
- Two APB bridges
- External memory

Each layer of the AHB bus allows one active transaction at a time. A transaction is initiated by a master, controlled by an arbiter, and serviced by the slave at the corresponding address. A transaction can be as short as a single byte, or as long as a CPU cache line (32 bytes). For the USB, a transaction can be much longer, up to 512 bytes on its AHB layer. For more information, refer to the AMBA 2.0 specification.

1.2.2.3 APB Buses

There are two APB peripheral buses on the i.MX23:

- The APBH bus runs completely synchronously to the AHB's HCLK.

- The APBX bus runs in the independent XCLK clock domain that can be slowed down significantly for power reduction.

The “H” in APB*H* denotes that the APBH is synchronous to HCLK, as compared to APB*X*, which runs on the crystal-derived XCLK. Figure 1-2 shows which blocks are controlled by which bus.

See Section 1.2.7, “DMA Controller,” for more information about these peripheral buses and their DMA bridges.

1.2.3 On-Chip RAM and ROM

The i.MX23 includes 8Kx32-bit on-chip RAM implemented as a single physical bank with four AHB slave ports. Each access to the on-chip RAM requires at a minimum two HCLK cycles.

The i.MX23 also includes 16Kx32-bit words of on-chip mask-programmable ROM. The ROM contains intitialization code written by Freescale. to handle the initial boot and hardware initialization. Software in this ROM offers a large number of boot configuration options, including manufacturing boot modes for burn-in and tester operation.

Other boot modes are responsible for loading application code from off-chip into the on-chip RAM. It supports initial program loading from a number of sources:

- NAND flash devices
- I^2C master mode from EEPROM devices
- USB recovery mode

At power-on time, the first instruction executed by the ARM core comes from this ROM. The reset boot vector is located at 0xFFFF0000. The on-chip boot code includes a firmware recovery mode. If the device fails to boot from NAND flash, or hard drive, for example, the device will attempt to boot from a PC host connected to its USB port.

The on-chip RAM and ROM run on the AHB HCLK domain.

Figure 1-3 shows the memory map for the AHB2 devices.

1.2.4 External Memory Interface

The i.MX23 supports off-chip DRAM storage via the EMI controller, which is connected to the four internal AHB/AXI busses.

The EMI supports multiple external memory types, including:

- 1.8-V Mobile DDR
- Standard 2.5V DDR1

The DRAM controller supports up to two external chip-select signal for the i.MX23 platform. Programmable registers within the DRAM controller allow great flexibility for device timings, low-power operation, and performance tuning. Note the diffences between the two package options:

- The 128-pin LQFP has 1 chip enable.
- The 169-pin BGA has 2 chip enables.

The EMI uses two primary clocks: the AHB bus HCLK and the DRAM source clock EMI_CLK. The maximum specified frequencies for these two clocks can be found in Chapter 2, “Characteristics and Specifications. The memory controller operates at frequencies that are asynchronous to the rest of the i.MX23.

The EMI consists of two major components:

- DRAM controller
- Delay compensation circuitry (DCC)

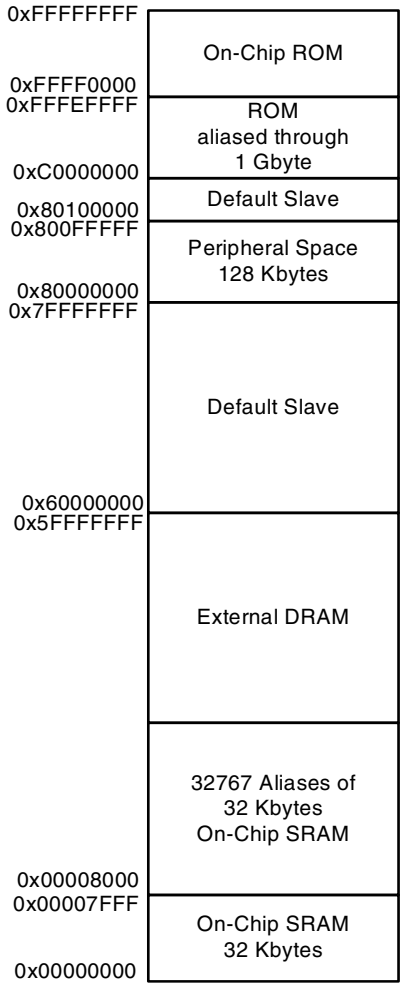


Figure 1-3. Physical Memory Map

1.2.5 On-Chip One-Time-Programmable (OCOTP) ROM

The i.MX23 contains 1024 bits (1Kb) of OTP ROM. The OTP is segmented into four distinct physical banks. Each bank is further divided logically into eight 32-bit words. The OTP serves several functions:

- Housing of hardware and software capability bits (copied into shadow registers)
- Housing of Freescale operations and unique-ID fields.
- Housing the customer-programmable cryptography key
- Four words for customer general use
- A 32-bit word is dedicated to controller read and write locking of the various OTP regions (copied into a shadow register)
- Storage of various ROM configuration bits

Access to the OTP is done through a memory-mapped APBH slave interface. Each of the 32 words is memory-mapped on APBH for the purposes of reading (requires a bank-opening sequence). Writing to the OTP is done through an address and data interface, where software provides the OTP word number (one of 32) and a programming mask.

For more information, see Chapter 7, “On-Chip OTP (OCOTP) Controller.”

1.2.6 Interrupt Collector

The i.MX23 contains a 128-bit vectored interrupt collector for the CPU’s IRQ input and a separate non-vectored interrupt collection mechanism for the CPU’s FIQ input. Each interrupt can be assigned to one of four levels of priority. The interrupt collector supports nesting of interrupts that preempt an interrupt service routine running at a lower priority level. Each of the 128 interrupts is assigned its own 32-bit programming register and can be set for HW source IRQ, SW source IRQ or HW source FIQ.

The interrupt collector is described in Chapter 5, “Interrupt Collector.”

1.2.7 DMA Controller

Many peripherals on the i.MX23 use direct memory access (DMA) transfers. Some peripherals, such as the USB controller, make highly random accesses to system memory for a large number of descriptor, queue heads, and packet payload transfers. This highly random access nature is supported by integrating a dedicated DMA into the USB controller and connecting it directly to the high-speed AHB bus.

Similarly, the RS-ECC8 error correction engine, the DCP (crypto/memcpy), BCH-ECC and LCD controller devices contain their own bus masters to allow for more random accesses to system memory.

Other peripherals have a small number of highly sequential transactions, for example the ADC or DAC streams, SPDIF transmitter, etc. These devices share a centralized address generation and data transfer function that allows them to share a single shared master on the AHB.

As mentioned previously, there are two AMBA peripheral buses on the i.MX23:

- The APBH bus runs completely synchronously to the AHB's HCLK.
- The APBX bus runs in an independent XLKC clock domain that can be slowed down significantly for power reduction.

See Chapter 10, "AHB-to-APBH Bridge with DMA," and Chapter 11, "AHB-to-APBX Bridge with DMA," for more detailed information. Note that the AHB HCLK can run up to 133 MHz.

The two bridge DMAs are controlled through linked DMA command lists. The CPU sets up the DMA command chains before starting the DMA. The DMA command chains include set-up information for a peripheral and associated DMA channel. The DMA controller reads the DMA command, writes any peripheral set up, tells the peripheral to start running and then transfers data, all without CPU intervention. The CPU can add commands to the end of a chain to keep data moving without interventions.

The linked DMA command architecture offloads most of the real-time aspects of I/O control from the CPU to the DMA controller. This provides better system performance, while allowing longer interrupt latency tolerances for the CPU.

1.2.8 Clock Generation Subsystem

The i.MX23 uses several different clock domains to provide clocks to the various subsystems, as shown in Figure 4-1. These clocks are either derived from the 24-MHz crystal or from the integrated high-speed PLL. The PLL output is fixed at 480 MHz.

More details about the system clock architecture can be found in Chapter 4, "Clock Generation and Control."

The system includes a real-time clock that can use either the 24-MHz system crystal or an optional low power crystal oscillator running at either 32.768 kHz or 32.0 kHz (customer-configurable via OTP). An integrated watchdog reset timer is also available for automatic recovery from errant code execution.

See Chapter 23, "Real-Time Clock, Alarm, Watchdog, Persistent Bits," for more information about these features.

1.2.9 Power Management Unit

The i.MX23 contains a sophisticated power management unit (PMU), including an integrated DC-DC converter, four linear regulators and a regulated 4.2V output. The PMU can operate from a Li-Ion battery using the DC-DC converter or a 5-V supply using the linear regulators and can automatically switch between them without interrupting operation. The PMU includes circuits for battery and system voltage brownout detection, as well as on-chip temperature, digital speed, and process monitoring.

The integrated PMU converter can be used to provide programmable power for the device as well as the entire application on up to five rails:

- VDDIO (nominal 3.3V) – DC-DC or linear-regulator from 5V
- VDDD (nominal 1.2V) – DC-DC or linear-regulator from VDDA
- VDDA (nominal 1.8V) – DC-DC or linear-regulator from VDDIO
- VDDM (nominal 2.5V) – linear-regulator from VDDIO
- VDD4P2 (nominal 4.2V) – when connected to 5V source

The 4.2V regulated output also allows for programmable current limits:

- Total load plus battery charge current (5V Limit)
- Battery Charge current
- Load current – for both on-chip and off-chip circuits

The 4.2V circuit is capable of adjusting distribution of current supply between the load and the battery-charger depending on programmed current-limits and load conditions. For example, when charging the battery, and exceeding the 5V current limit, the 4.2V regulator will steal current from the battery-charger circuit and divert it to the load circuit.

The converter can be configured to operate from standard Li-Ion battery chemistries up to 4.2 volts. These converters use off-chip reactive components (L/C) in a pulse-width or frequency-modulated DC-DC converter.

The real-time clock includes an alarm function that can be used to “wake-up” the DC-DC converters, which will then wake up the rest of the system.

The power subsystem is described in Chapter 32, “Power Supply.”

1.2.10 USB Interface

The chip includes a high-speed Universal Serial Bus (USB) version 2.0 controller and integrated USB transceiver macrocell interface (UTMI) PHY. The i.MX23 device interface can be attached to USB 2.0 hosts and hubs running in the USB 2.0 high-speed mode at 480 Mbits per second. It can be attached to USB 2.0 full-speed interfaces at 12 Mbits per second.

The USB controller and integrated PHY support high-speed Host modes for peer-to-peer file interchange. The i.MX23 has a high-current PWM channel that can be used with low-cost external components to generate up to 8 mA of 5 volts on the Host VBUS for Host session initiation. The USB controller can also be configured as a high-speed host.

The USB subsystem is designed to make efficient use of system resources within the i.MX23. It contains a random-access DMA engine that reduces the interrupt load on the system and reduces the total bus bandwidth that must be dedicated to servicing the five on-chip physical endpoints.

It is a dynamically configured port that can support up to five endpoints, each of which may be configured for bulk, interrupt, or isochronous transfers. The USB configuration information is read from on-chip memory via the USB controller's DMA.

See Chapter 8, "USB High-Speed Host/Device Controller," and Chapter 9, "Integrated USB 2.0 PHY," for more information.

1.2.11 General-Purpose Media Interface (GPMI)

The chip includes a general-purpose media interface (GPMI) controller that supports NAND devices (all packages).

The NAND flash interface provides a state machine that provides all of the logic necessary to perform DMA functions between on-chip or off-chip RAM and up to four NAND flash devices. The controller and DMA are sophisticated enough to manage the sharing of a single 16 bit wide data bus among four NAND devices, without detailed CPU intervention. This allows the i.MX23 to provide unprecedented levels of NAND performance.

The general-purpose media interface can be described as two fairly independent devices in one. The three operating modes are integrated into one overall state machine that can freely intermix cycles to different device types on the media interface. There are four chip selects on the media interface. Each chip select can be programmed to have a different type device installed.

The GPMI pin timings are based on a dedicated clock divider from the PLL, allowing the CPU clock divider to change without affecting the GPMI.

See Chapter 13, "General-Purpose Media Interface (GPMI)," for more information.

1.2.12 Hardware Acceleration for ECC for Robust External Storage

The hardware ECC accelerator provides a forward error-correction function for improving the reliability of various storage media that may be attached to the i.MX23. Modern high-density NAND flash devices presume the existence of forward error-correction algorithms to correct some soft and/or hard bit errors within the device, allowing for higher device yields and, therefore, lower NAND device costs.

The i.MX23 contains two separate Error Correction Code (ECC) hardware engines implemented the following algorithms:

- Reed-Solomon – Provides 4 or 8 bits/symbol correction (RS-ECC8). This is the same engine found in previous SoC products.
- Bose Ray-Choudhury Hocquenghem – Provides up to 20-bits correction (BCH-ECC)

Both engines are tightly coupled to the GPMI and are for mutually exclusive use with completely separate programming models and DMA structures. The BCH engine supersedes the RS-ECC8 except in

allowing for backwards compatibility of legacy NAND software drivers written specifically for the RS-ECC8.

1.2.12.1 Reed-Solomon ECC Engine

The RS-ECC module consists of two different error correcting code processors:

- Four-symbol error correcting (9 bits/symbol) Reed-Solomon encoder/decoder
- Eight-symbol error correcting (9 bits/symbol) Reed-Solomon encoder/decoder

The Reed-Solomon modes are used for storage elements that have a higher native defect probability, such as MLC NAND. It can correct up to four 9-bit symbols over a 512-byte block in a 2-Kbyte paged device or up to eight 9-bit symbols over a 512-byte block in a 4-Kbyte paged device.

Both of these error correction encoder/decoders use DMA transfers to move data from system memory completely in parallel with the CPU performing other useful work.

For storage read transfers, the ECC8 controller uses its AHB bus master to transfer data directly to system memory. In addition, the ECC8 automatically corrects errors in the read data buffers in system memory without CPU assistance.

The ECC8 includes one more significant enhancement, namely, it provides four-symbol error correction for the 9 or 16 byte metadata stored in the redundant area of the NAND device.

See Chapter 14, “8-Symbol Correcting ECC Accelerator (ECC8),” for more information.

1.2.12.2 Bose Ray-Choudhury Hocquenghem ECC Engine

The Bose, Ray-Chaudhuri, Hocquenghem (BCH) Encoder and Decoder module is capable of correcting from 2 to 20 single bit errors within a block of data no larger than about 900 bytes (512 bytes is typical) in applications such as protecting data and resources stored on modern NAND flash devices. The correction level in the BCH block is programmable to provide flexibility for varying applications and configurations of flash page size. The design can be programmed to encode protection of 2, 4, 8, 10, 12, 14, 16, 18, or 20 bit errors when writing flash and to correct the corresponding number of errors on decode. The correction level when decoding MUST be programmed to the same correction level as was used during the encode phase.

BCH-codes are a type of block-code, which implies that all error-correction is performed over a block of N-symbols. The BCH operation will be performed over GF(2^{13} = 8192), which is the Galois Field consisting of 8191 one-bit symbols. BCH encoding (or encode for any block-code) can be performed by two algorithms: systematic encoding or multiplicative encoding. Systematic encoding is the process of reading all the symbols which constitute a block, dividing continuously these symbols by the generator polynomial for the GF(8192) and appending the resulting ***t*** parity symbols to the block to create a BCH codeword (where ***t*** is the number of correctable bits).

The BCH sits on the AXI fabric with close coupling to both the GPMI and external memory controller.

See Chapter 15, “20-BIT Correcting ECC Accelerator (BCH).”

1.2.13 Data Co-Processor (DCP)—Memory Copy, Crypto, and Color-Space Converter

The i.MX23 SOC contains a data co-processor consisting of four virtual channels. Each channel is essentially a memory-to-memory copy engine. The linked list control structure can be used to move byte-aligned blocks of data from a source to a destination. In the process of copying from one place to another, the DCP can be programmed to encrypt or decrypt the block using AES-128 in one of several chaining modes. An SHA-1 hash can be calculated as part of the memory-copy operation.

See Chapter 16, “Data Co-Processor (DCP),” for more information.

1.2.14 Mixed Signal Audio Subsystem

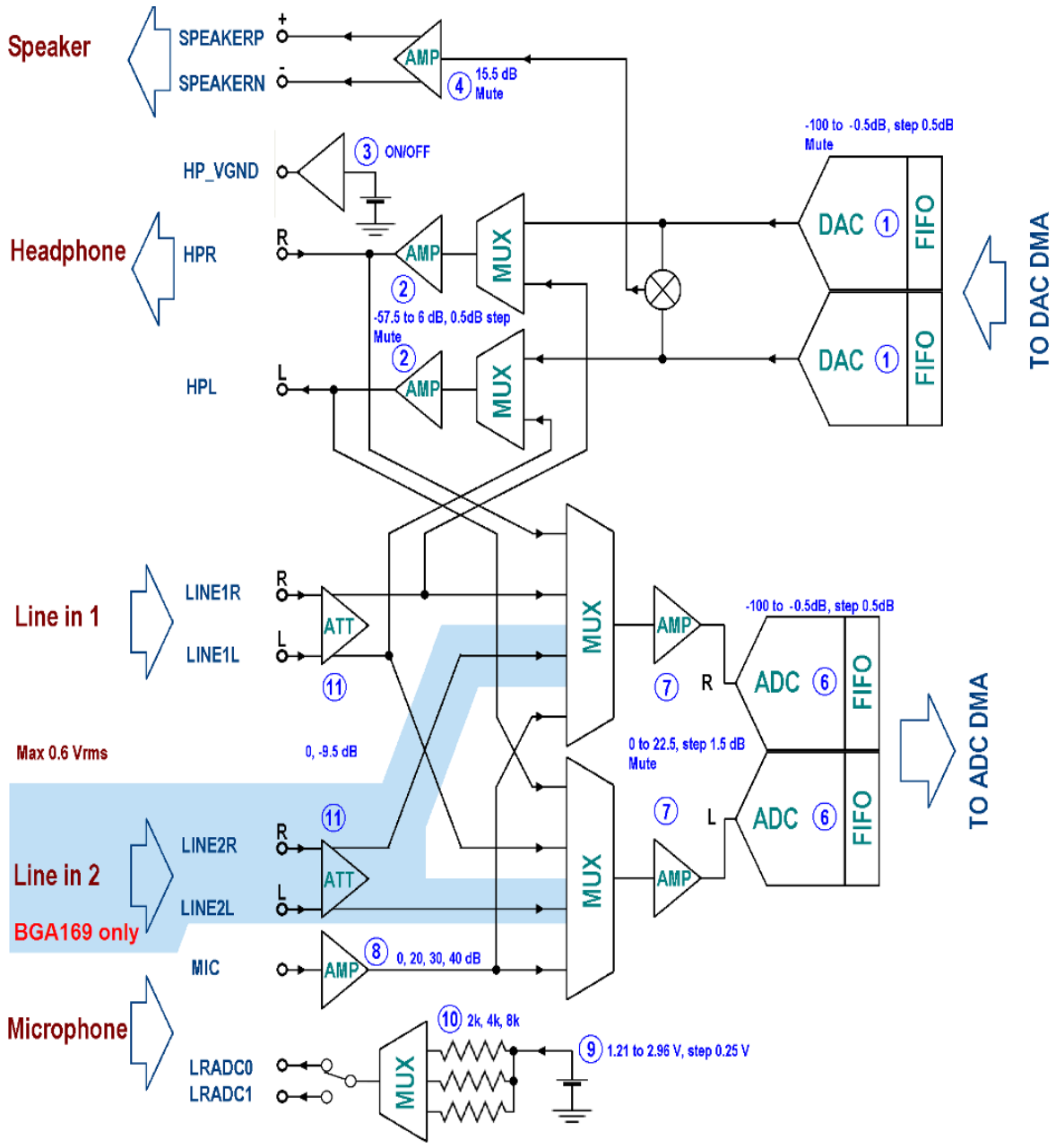
The i.MX23 contains an integrated high-quality mixed signal audio subsystem, including high-quality sigma delta D/A and A/D converters, as shown in Figure 1-4.

The chip includes a low-noise headphone driver that allows it to directly drive low-impedance (16Ω) headphones. The direct drive, or “capless” mode, removes the need for large expensive DC blocking capacitors in the headphone circuit. The headphone power amplifier can detect headphone shorts and report them via the interrupt collector. A digitally programmable master volume control allows user control of the headphone volume. Use of the headphone amplifier volume control is recommended as the digital control may reduce SNR performance. Annoying clicks and pops are eliminated by zero-crossing updates in the volume/mute circuits and by headphone driver startup and shutdown circuits.

The microphone circuit has a mono-to-stereo programmable gain pre-amp and an optional microphone bias generator.

Also integrated is a class A-B mono speaker amplifier which must be powered from a sufficiently high-enough current 4.2V source such as the battery. The speaker amplifier can support up to 2W rms of output assuming a 4.2V supply and a 4Ω speaker load.

These features are described in Chapter 28, “AUDIOIN/ADC,” and Chapter 29, “AUDIOOUT/DAC.”



Notes:
1. HW_AUDIOOUT_DACVOLUME: Digital volume control. -100 dB to -0.5 dB in 0.5 dB steps.
2. HW_AUDIOOUT_HPVOL: Analog volume control. -57.5 dB to 6 dB in 0.5 dB steps.
3. HW_AUDIOOUT_PWDN: Enable capless headphone common amplifier.
4. HW_AUDIOOUT_SPEAKERCTRL: Analog control for speaker amplifier, fixed gain of 9.5 dB from each DAC, 15.5dB total.
5. HW_AUDIOIN_ADCVOLUME: Digital volume control. -100 dB to -0.5 dB in 0.5 dB steps.
6. HW_AUDIOIN_ADCVOL: Analog volume control that controls the ADC gain block. 0 dB to 22.5 dB gain in 1.5 dB steps.
7. HW_AUDIOIN_MICLINE_MICGAIN: Analog volume control that controls the microphone amplifier. 0, 20, 30, or 40 dB gain.
8. HW_AUDIOOUT_MICLINE_MIC_BIAS: Mic bias voltage. 1.21V to 2.96V in 0.25V steps.
9. HW_AUDIOOUT_MICLINE_MIC_RESISTOR, HW_AUDIOOUT_MICLINE_MIC_SELECT.
10. HW_AUDIOIN_MICLINE_DIVIDE_LINE1/2.

Figure 1-4. Mixed Signal Audio Elements

1.2.15 Master Digital Control Unit (DIGCTL)

The master digital control unit (DIGCTL) provides control registers for a number of blocks that do not have their own AHB or APB slaves, notably the on-chip RAM and on-chip ROM controllers. In addition, it provides control registers for the DRAM controller output clock shifting. It also provides several security features, including an entropy register, as well as the JTAG shield.

See Chapter 6, "Digital Control and On-Chip RAM," for more information.

1.2.16 Synchronous Serial Port (SSP)

The i.MX23 SOC contains two integrated synchronous serial ports, SSPs. Each SSP supports a wide range of synchronous serial interfaces, including:

- 1-bit, 4-bit, or 8-bit high-speed MMC/SD/SDIO
- Motorola (1-bit) and Winbond (1, 2 and 4-bit) SPI with up to 3 slave selects
- TI SSI

Each SSP has a dedicated DMA channel and a dedicated clock divider from the PLL.

See Chapter 21, "Synchronous Serial Ports (SSP)," for more information about these features.

1.2.17 I^2C Interface

The chip contains a two-wire SMB/I^2C bus interface. It can act as a master on the SMB interface. The on-chip ROM supports boot operations from I^2C EEPROMs.

See Chapter 25, "I2C Interface," for more information.

1.2.18 General-Purpose Input/Output (GPIO)

The i.MX23 contains 95 GPIO pins in the 169-pin package and 64 GPIO pins in the 128-pin package. Most digital pins (except for EMI pins) that are available for specific functions are also available as GPIO pins if they are not otherwise used in a particular application.

See Chapter 37, "Pin Control and GPIO," for more information

1.2.19 Display Processing

The i.MX23 display processing and output consists of four distinct modules as shown in Figure 1-5. These are:

- Pixel Processing Pipeline - dedicated AXI bus master.
- Display Controller (LCDIF) - dedicated AXI bus master.
- PAL/NTSC TV-Encoder - direct feed from LCDIF output
- 10-bit Video DAC for analog composite output - direct feed from TV-Encoder

This allows for all post video-decode pixel processing to be handled in hardware with minimal CPU intervention. Multiple pixel formats and display configurations are also supported.

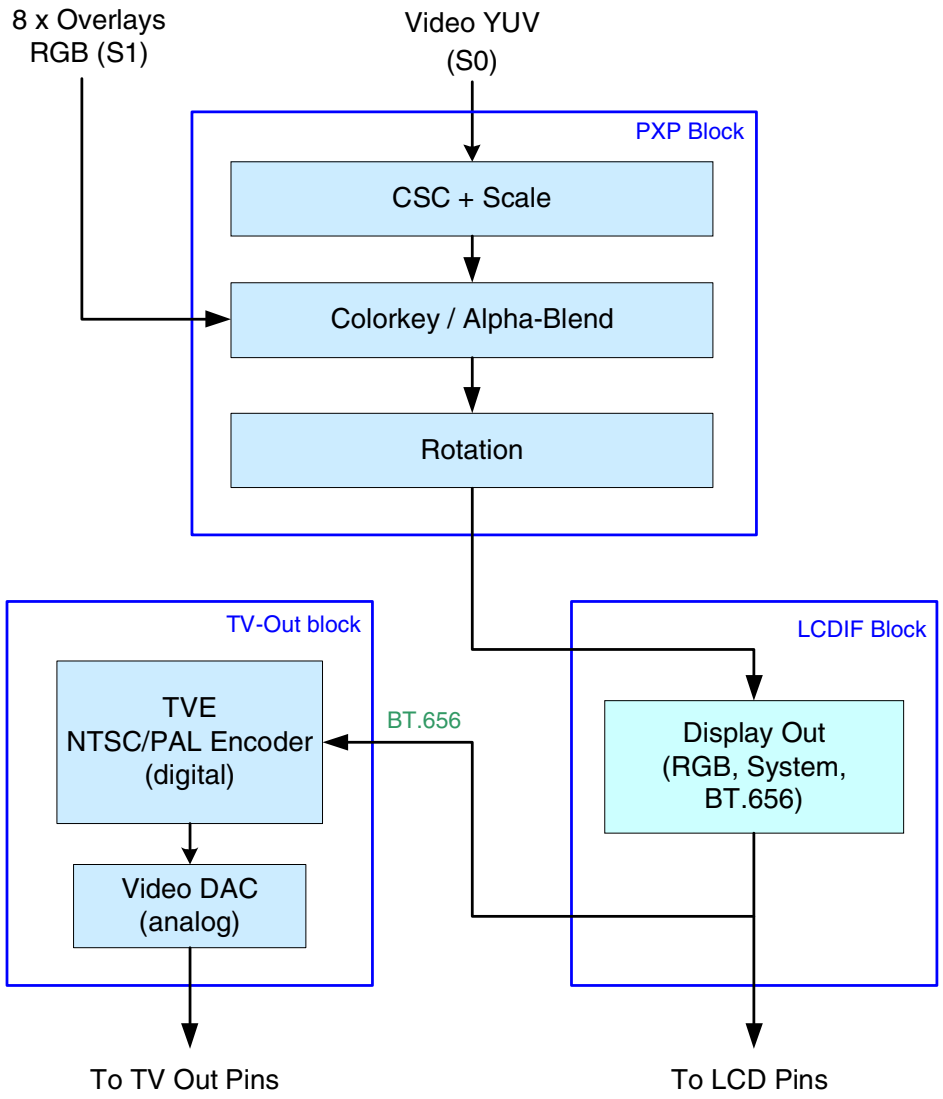


Figure 1-5. Display Processing Sub-System

1.2.19.1 Display Controller / LCD Interface (LCDIF)

The i.MX23 Display Controller (LCDIF) includes:

- AMBA AXI master mode allows for high-performance operation from external memory. This also includes an increase to a 128x32-bit internal latency buffer which features an under-flow recovery mechanism.
- Supports 24-bit full color parallel RGB (DOTCK) mode. Able to drive up to VGA (640x480) full color displays at refresh rates up to 60Hz.
- Supports full 24-bit system mode (8080/6080/VSYNC/WSYNC). Read-mode is not supported.

- ITU-R BT.656 compliant D1 digital video output mode with on-the-fly RGB to YCbCr color-space-conversion. This output also feeds the integrated TV-Encoder
- Supports wide variety of input and output formats allowing for conversion between input and output (e.g., RGB565 input to RGB888 output). Also supports packed pixel formats.

See Chapter 18, "LCD Interface (LCDIF)," for more information.

1.2.19.2 Pixel Processing Pipeline (PXP)

The PXP performs all necessary post display frame pre-processing in hardware with minimal memory overhead. In a video-centric system such as the i.MX23, this allows for the CPU to have maximum processing bandwidth for video-decode operation. The PXP operation and features can be described as follows:

- The *background* image (e.g. decoded video frames) is read from external memory into separate Y/U/V buffers as 8x8 pixel macroblocks. These buffers are then fed into a color-space converter (e.g. YUV to RGB) followed by the scaling engine which utilizes an advanced bi-linear weighted scaling algorithm. The scaling operation is defined in terms of the output image (via programmable offsets and cropping registers). The output of the scaler is fed into yet another internal buffer called S0. If the background image is already in the RGB color-space it is assumed to be scaled appropriately for the required output format and can thus be read directly into the internal S0 buffer. In order to maintain efficient use of external memory, only the relevant (visible) portion of the background image is fetched.
- The scaled RGB image (in the internal S0 buffer) can be blended with up to eight programmable *overlays*. The co-ordinates of the overlays can once again be described in terms of the resultant output image. Each overlay can have a either a global programmable opacity or a per-pixel resolution if constructed with ARGB color-space. In addition to this, each overlay can have a relative priority level such that when constructing the output image, the PXP only fetches the visible overlay in the current 8x8 macroblock. The overlays are fetched into the internal S1 buffer. Alpha blending is performed on the S0 and S1 buffers to generate the blended output into the internal S3 buffer. Other operations such as BITBLT and color-keying can also be performed at this stage.
- The final stage of the PXP operation is the rotator which can perform flips and 90, 180 and 270 rotations. The rotator operates on the 8x8 pixel macroblocks in the S3 buffer to maximize external memory fetch efficiency. It writes 8x8 macroblocks to external memory in this final stage.

It should be noted that the PXP supersedes all pixel operations of the DCP.

See Chapter 17, "Pixel Pipeline (PXP)," for more information on the PXP.

1.2.19.3 PAL/NTSC TV-Encoder

The PAL/NTSC TV-Encoder is part of the integrated TV-Out functionality of i.MX23. The encoder takes input directly from the LCDIF without intermediate memory access. In order to utilize the TV-Out path, the LCDIF must be configured to output the ITU-R BT.656/BT.601 D1 digital video stream mode. This stream is synchronized to the internal 108MHz clock of the TV-Encoder. After this point, the block

encodes the stream into a format suitable for the Video DAC. Before being sent to the video DAC, the output of the TV-Encoder is passed through a pixel interpolating filter which helps to lessen the requirements for off-chip video filtering.

See Chapter 19, "TV-Out NTSC/PAL Encoder." for more information on the TV-Encoder.

1.2.19.4 Video DAC

The i.MX23 includes a fully integrated low-power 10-bit Video DAC which takes the direct output from the TV-Encoder to generate a compliant analog composite analog video signal (CVBS). Also supported are optional source termination and automatic jack detection (via interrupt) allowing the Video DAC to be enabled/disabled automatically.

See Chapter 20, "Video DAC," for more information.

1.2.20 SPDIF Transmitter

The i.MX23 includes a Sony-Philips Digital Interface Format (SPDIF) transmitter. It supports sample rates independently from the A/D and D/A sample rates so that all three can run simultaneously. The SPDIF has a dedicated DMA channel. The SPDIF has its own clock divider from the PLL. See Chapter 30, "SPDIF Transmitter," for more information.

1.2.21 Dual Serial Audio Interfaces

The BGA169 package of the i.MX23 includes two serial audio interfaces (SAIF), each with three stereo pairs. The pin multiplexing scheme for i.MX23 allows a stereo transmitter on one device and a stereo receiver to be connected to external devices, either D/A and A/D converters or to a host processor, such as a cell phone or BlueTooth controller.

See Chapter 31, "Serial Audio Interface (SAIF) (BGA169 Only)."

1.2.22 Timers and Rotary Decoder

An automatic rotary decoder function is integrated into the chip. Two digital inputs are monitored to determine which is leading and by how much. In addition, the hardware automatically determines the period for rotary inputs. There are four timers to provide timer functionality based on different clock inputs.

See Chapter 22, "Timers and Rotary Decoder," for more information.

1.2.23 UARTs

Three UARTs, similar to a 16550 UART, are provided—two for application use and one for debug use. The application UARTs are a high-speed devices capable of running up to 3.25 Mbits per second with

16-byte receive and transmit FIFOs. The application UARTs supports DMA and flow control (CTS/RTS). The debug UART does not use DMA channels.

See Chapter 26, “Application UART,” and Chapter 27, “Debug UART,” for more information.

1.2.24 Low-Resolution ADC, Touch-Screen Interface, and Temperature Sensor

The LRADC provides 16 “physical” channels of 12-bit resolution analog-to-digital conversion. Only 8 “virtual” channels can be used at one time, but those 8 channels can be mapped to any of the 16 physical channels. Some physical channels have dedicated inputs:

- Channel 15—VDD5V
- Channel 14—Bandgap reference
- Channel 13—USB_DN
- Channel 12—USB_DP
- Channel 10 and 11—Reserved
- Channel 8 and 9—Internal temperature sensing
- Channel 7—Battery
- Channel 6—VDDIO

The USB_DN/DP inputs can only be sampled with the LRADC in non-USB mode (see HW_USBPHY_CTRL_DATA_ON_LRADC).

The remaining six channels are available for other uses and can be used for resistive button sense, touch-screens, or other analog input. Channels 0 and 1 have integrated current sources to drive external temperature monitor thermistors. Channels 2–5 have integrated drivers for resistive touch-screens. The LRADC provides typical performance of 12-bit no-missing-codes, 9-bit/~56dB SNR, and 1% absolute accuracy (limited by the bandgap reference).

See Chapter 33, “Low-Resolution ADC and Touch-Screen Interface,” for more information.

1.2.25 Pulse Width Modulator (PWM) Controller

The i.MX23 contains five PWM output controllers that can be used in place of GPIO pins. Applications include LED and backlight brightness control. Independent output control of each phase allows 0, 1, or high impedance to be independently selected for the active and inactive phases. Individual outputs can be run in lock step with guaranteed non-overlapping portions for differential drive applications.

See Chapter 24, “Pulse-Width Modulator (PWM) Controller,” for more information.

1.2.26 Real-Time Clock, Alarm, Watchdog, Persistent Bits

The i.MX23 supports real-time clock, alarm clock, watchdog reset, persistent bits and millisecond counter. The RTC system can be powered from the battery 5 V supply. The clock sources for these functions are selectable between 32 kHz, 32.768 kHz or 24 MHz crystals.

See Chapter 23, “Real-Time Clock, Alarm, Watchdog, Persistent Bits,” for more information.

Chapter 2
Characteristics and Specifications

This chapter describes the characteristics and specifications of the i.MX23 and includes sections on absolute maximum ratings, recommended operating conditions, and DC characteristics.

2.1 Absolute Maximum Ratings

Table 2-1. Absolute Maximum Ratings

Parameter	Min	Max	Units
Storage Temperature	–40	125	°C
Battery Pin - BATT, VDD4P2V	–0.3	4.242	V
5-Volt Source Pin - VDD5V (transient, t < 30ms, duty cycle < 0.05%)	-0.3	7.00	V
5-Volt Source Pin - VDD5V (static)	–0.3	6.00	V
PSWITCH (Note 1)	–0.3	VDDXTAL + 1.575	V
Analog Supply Voltage—VDDA	–0.3	2.10	V
Speaker Amplifier Supply Voltage—VDDS	–0.3	4.242	V
Digital Core Supply Voltage —VDDD	–0.3	1.575	V
Non-EMI Digital I/O Supply—VDDIO	–0.3	3.63	V
EMI Digital I/O Supply—VDDIO.EMI	–0.3	3.63	V
DC-DC Converter—DCDC_BATT (Note 2)	–0.3	BATT	V
Input Voltage on Any Digital I/O Pin Relative to Ground	–0.3	VDDIO+0.3	V
Input Voltage on USB_DP and USB_DN Pins Relative to Ground (Note 3)	–0.3	3.63	V
Input Voltage on Any Analog I/O Pin Relative to Ground	–0.3	VDDA+0.3	V

1 VDDIO can be applied to PSWITCH through a 10 kΩ resistor. This is necessary in order to enter the chip's firmware recovery mode. (The on-chip circuitry prevents the actual voltage on the pin from exceeding acceptable levels.)

2 Application should include a Schottky diode between BATT and VDD4P2.

3 USB_DN and USB_DP can tolerate 5V for up to 24 hours. Note that while 5V is applied to USB_DN or USB_DP, LRADC readings can be corrupted.

Table 2-2. Electro-Static Discharge Immunity

169-Pin BGA & 128-Pin LQFP Packages	Tested Level
Human Body Model (HBM)	2 kV
Charge Device Model (CDM)	500 V

2.2 Recommended Operating Conditions

Table 2-3. Recommended Power Supply Operating Conditions

Parameter	Min	Typ	Max	Units
Analog Core Supply Voltage—VDDA	1.62	-	2.10	V
Digital Core Supply Voltage—VDDD *Specification dependent on frequency.* (Notes **1, 4**)	1.00	-	1.55	V
Non-EMI Digital I/O Supply Voltage—VDDIO	2.90	-	3.575	V
EMI Digital I/O Supply Voltage—VDDIO.EMI	1.8	-	3.25	V
Battery / DCDC Input Voltage - BATT, DCDC_BATT (Note **2**)	2.6	-	4.242	V
Speaker Supply Voltage - VDDS	2.7	-	4.242	V
VDD5V Supply Voltage (5V current < 100 ma)	4.40	5.00	5.25	V
VDD5V Supply Voltage (5V current >= 100 ma)	4.75	5.00	5.25	V
Offstate Current (Note **3**): • 32-kHz RTC off, BATT = 4.2 V • 32-kHz RTC on, BATT = 4.2 V	 - -	 11 13.5	 30 30	 µA µA

1 For optimum USB jitter performance, VDDD = 1.35 V or greater.

2 This requires software to program the RTC_PERSIST0_SPARE<31:28> bits. Minimum without the software programming is 2.9V

3 When the real-time clock is enabled, the chip consumes additional current when in the OFF state to keep the crystal oscillator and the real-time clock running.

4 VDDD supply minimum voltage includes 75 mV guardband.

Table 2-4. Operating Temperature Conditions

Parameter	Min	Typ	Max	Units
Commercial Ambient Operating Temperature Range, T_A (Note **1, 2**)	–10	-	70	°C
Commercial Junction Temperature Range, T_J (Note **1, 2**)	–10	-	85	°C
Industrial Ambient Operating Temperature Range, T_A (Note **1, 2**)	–40	-	85	°C
Industrial Junction Temperature Range, T_J (Note **1, 2**)	–40	-	105	°C
Package Thermal Impedance, 128-Pin LQFP, Θ_{JA} (Note **3**)	-	-	43	°C/W
Package Thermal Impedance, 169-Pin BGA, Θ_{JA} (Note **3**)	-	-	44	°C/W

1 In most systems designs, battery and display specifications will limit the operating range to well within these specifications. Most battery manufacturers recommend enabling battery charge only when the ambient temperature is between 0° and 40°C. To ensure that battery charging does not occur outside the recommended temperature range, the player ambient temperature may be monitored by connecting a thermistor to the LRADC0 or LRADC1 pin on the i.MX23.

2 Maximum Ambient Operating Temperature may be limited due to on-chip power dissipation. $T_{A\ (MAX)} <= T_J - (\Theta_{JA} \times P_D)$ where:

- T_J = Maximum Junction Temperature
- Θ_{JA} = Package Thermal Impedance
- P_D = Total On-chip Power Dissipation = PSpeakerAmp + PVDD4P2 + PBatteryCharger + PDCDC + PLinearRegulators +PInternal. Note that depending on the application, some of these power dissipation terms may not appliy.
- PSpeakerAmp = Speaker Amp On-Chip Power Dissipation = ~1W (regardless of output amplitude)
- PVDD4P2 = VDD4P2 On-Chip Power Dissipation = (VDD5V - VDD4P2) x IDD4P2
- PBatteryCharger = Battery Charger On-Chip Power Dissipation = (VDD5V - BATT) x ICHARGE
- PDCDC = DC-DC Converter On-Chip Power Dissipation = (BATT x DCDC Input Current) x (1 - efficiency)
- PLinearRegulators = Linear Regulator On-Chip Power Dissipation = (VDD5V-VDDIO) x (IDDIO + IDDM + IDDA + IDDD) + (VDDIO - VDDM) x IDDM + (VDDIO - VDDA) x (IDDA + IDDD) + (VDDA - VDDD) x IDDD
- PInternal = Internal Digital On-Chip Power Dissipation = ~VDDD x IDDD

3 Assumes 4-layer PCB and still air. Actual thermal performance may vary based on board and enclosure composition and design.

Table 2-5. Recommended Analog Operating Conditions

Parameter	Min	Typ	Max	Units
Low Resolution ADC:				
• Input Impedance (CH0 - CH5)	>1	-	-	MΩ
• Absolute Accuracy		+/- 1.5		%
On-Die Temperature Sensor:				
• Absolute Accuracy		+/- 1.5		%
Microphone:				
• Full-Scale Input Voltage (MIC_GAIN=40 dB) (Note **4**, **5**)	0.0052	0.0055	0.0057	Vrms
• Input Resistance	75	100	125	kΩ
• Idle SNR (Note **8**)	59	66	-	dB FS
• THD+N at -3dBFS	-53	-63	-	dB
Line Inputs:				
• Full-Scale Input Voltage to ADC (Note **1, 4**, **5**)	0.52	0.54	0.56	Vrms
• Full-Scale Input Voltage to HP with 0dB Gain (Note **1, 4**)	0.52	0.59	0.61	Vrms
• Input Resistance (Line-to-Headphone mode) (Note **2**)	37.5	50	62.5	kΩ
• Input Resistance (Line-to-ADC mode) (Note **2**)	14.1	18.75	23.4	kΩ
• LineIn-to-HP SNR Idle Channel (Note **2**)	97	100	103	dB FS
• ADC SNR Idle Channel (Note **2**)	80	87	89	dB FS
• ADC –60 dB Dynamic Range (Note **2**)	80	87	89	dB FS
• ADC THD+N at -3dBFS	-73	-80	-	dB
Headphone:				
• Full-Scale Output Voltage (VDDA = 1.8 V, 16 Ω load) (Note **6**)	0.46	0.52	0.54	Vrms
• Output Resistance	0.1	0.3	0.5	Ω
• Crosstalk between Input Channels (16 Ω load) (Note **7**)	-65	-70	-	dB
• THD+N (16 Ω load) (Note **7, 9**)	-80	-84	-	dB
• THD+N (10 kΩ load) (Note **7, 9**)	-83	-86	-	dB
• DAC SNR Idle Channel (Note **2, 7, 8**)	95	97	-	dB FS
• DAC –60 dB Dynamic Range (Note **2, 7, 8**)	95	97	-	dB FS
• Output Frequency Response (20Hz - 20kHz, 1 kHz = 0dB)	-1	-	+1	dB FS
• Channel Balance (Level Difference between L-ch and R-ch)	-0.2	0.04	0.2	dB
Speaker Amplifier:				
• Full-Scale Output Voltage (VDDS = 4.2 V, 8 Ω load)	2.6	2.7	2.8	Vrms
• Output Offset Voltage	0	15	45	mV
• Maximum Out Power (VDDS = 4.2 V, 8 Ω load, 1% THD)	0.85	0.9	1	W
• Maximum Out Power (VDDS = 3.0 V, 8 Ω load, 1% THD)	0.425	0.45	0.55	W
• Maximum Out Power (VDDS = 4.2 V, 4 Ω load, 10% THD)	1.6	1.75	1.9	W
• Maximum Out Power (VDDS = 4.2 V, 4 Ω load, 1% THD)	1.275	1.45	1.6	W
• Maximum Out Power (VDDS = 3.0 V, 4 Ω load, 1% THD)	0.6	0.7	0.8	W
• SNR (VDDS = 4.2 V, A-Weighted, 8 Ω load)	90	95	-	dB
• Speaker VDDS Active Current (no signal) (Note **3**)	-	1.3 + I_{offset}	45 mV/Ω	mA
• Speaker VDDS Leakage Current (VDDS=4.2V, Speaker = OFF)	-	0.1	0.6	uA
• THD+N (4 Ω load, -2dB signal)	-50	-55	-	dB
• THD+N (8 Ω load, -2dB signal)	-55	-61	-	dB
• PSRR at 217-1000 Hz	60	75	-	dB

1 1 Vrms requires external resistor divider.

2 Measured “A weighted” over a 20-Hz to a 20-kHz bandwidth, relative to full scale output voltage (when VDDIO = 3.3 V and VDDA = 2.1 V).

3 I_{offset} = offset voltage/speaker impedance.

4 Maximum input that achieves -40dB THD+N

5 ADC gain = 0 dB

6 Maximum output that achieves at least -66dB THD+N into 16 Ω load

7 Measured at -1dB FS, where fullscale achieves at least -66dB THD+N into a 16 Ω load

8 SNR and Dynamic range measurements made in capless headphone mode with DCDC converters running and JTAG disconnected.

9 Applies for both LINE IN and DAC IN sources.

2.3 DC Characteristics

Table 2-6. PSWITCH Input Characteristics

Parameter	HW_PWR_STS _PSWITCH	Min	Max	Units
PSWITCH LOW LEVEL (See Note 3)	0x00	0.00	0.30	V
PSWITCH MID LEVEL & STARTUP (See Note 1, 3)	0x01	0.65	1.50	V
PSWITCH HIGH LEVEL (See Note 2, 3)	0x11	2.1	VDDXTAL + 1.575	V

1 A MID LEVEL PSWITCH state can be generated by connecting the VDDXTAL output of the SOC to PSWITCH through a switch.

2 PSWITCH acts like a high impedance input (>300 kΩ) when the voltage applied to it is less than 1.5V. However, above 1.5V it becomes lower impedance. To simplify design, it is recommended that a 10 kΩ resistor to VDDIO be applied to PSWITCH to set the HIGH LEVEL state.

3 Consult the reference schematics for recommended PSWITCH button circuitry.

Table 2-7. Power Supply Characteristics

Parameter	Min	Typ	Max	Units
VDDXTAL Voltage Reference				
Output Voltage		1.0		V
Linear Regulators	**Min**	**Typ**	**Max**	**Units**
Output Voltage Accuracy (VDDIO, VDDA, VDDM, VDDD) (See Note 1)	-1		+3	%
VDDIO Maximum Output Current (VDDIO=3.325V, VDD5V=4.75V) (See Note 2, 8)	240			mA
VDDIO Maximum Output Current (VDDIO=3.325V, VDD5V=4.40V) (See Note 2, 8)	175			mA
VDDM Maximum Output Current (VDDM=2.5V, VDD5V=4.75V) (See Note 2, 8)	240			mA
VDDM Maximum Output Current (VDDM=2.5V, VDD5V=4.40V) (See Note 2, 8)	230			mA
VDDA Maximum Output Current (VDDA=1.8V, VDD5V=4.75V) (See Note 2, 8)	235			mA
VDDA Maximum Output Current (VDDA=1.8V, VDD5V=4.75V) (See Note 2, 8)	225			mA
VDDD Maximum Output Current (VDDD=1.2V, VDD5V=4.75V) (See Note 2, 8)	260			mA
VDDD Maximum Output Current (VDDD=1.2V, VDD5V=4.75V) (See Note 2, 8)	245			mA
VDDIO Output Impedance (See Note 9)	-	115	-	mΩ
VDDM Output Impedance (See Note 9)	-	335	-	mΩ
VDDA Output Impedance (See Note 9)	-	105	-	mΩ
VDDD Output Impedance (See Note 9)	-	175	-	mΩ
DCDC Converters	**Min**	**Typ**	**Max**	**Units**
Output Voltage Accuracy (DCDC_VDDIO, DCDC_VDDA, DCDC_VDDD) (See Note 1)	-1		+3	%
DCDC Converters (3.0V < BATT < 4.242V)	**Min**	**Typ**	**Max**	**Units**
DCDC_VDDD Maximum Output Current (VDDD=1.55V) (See Note 3, 7)	250			mA
DCDC_VDDA Maximum Output Current (VDDA=1.8V) (See Note 3, 7)	200			mA

Table 2-7. Power Supply Characteristics (continued)

DCDC_VDDIO Maximum Output Current (VDDIO=3.30V, VDDD=1.55V) (See Note 3, 4, 7)	135			mA
DCDC_VDDIO Maximum Output Current (VDDIO=3.20V, VDDD=1.55V) (See Note 3, 4, 7)	120			mA
DCDC_VDDIO Maximum Output Current (VDDIO=3.15V, VDDD=1.55V) (See Note 3, 4, 7)	95			mA
DCDC_VDDIO Maximum Output Current (VDDIO=3.15V, VDDD=1.375V) (See Note 3, 4, 7)	175			mA
DCDC Converters (3.3V < BATT < 4.242V)	**Min**	**Typ**	**Max**	**Units**
DCDC_VDDD Maximum Output Current (VDDD=1.55V, 3.3V < BATT < 4.242V) (See Note 3, 7)	250			mA
DCDC_VDDA Maximum Output Current (VDDA=1.8V, 3.3V < BATT < 4.242V) (See Note 3, 7)	200			mA
DCDC_VDDIO Maximum Output Current (VDDIO=3.30V, VDDD=1.55V) (See Note 3, 4, 7)	215			mA
DCDC_VDDIO Maximum Output Current (VDDIO=3.20V, VDDD=1.55V) (See Note 3, 4, 7)	175			mA
DCDC_VDDIO Maximum Output Current (VDDIO=3.15V, VDDD=1.55V) (See Note 3, 4, 7)	135			mA
DCDC_VDDIO Maximum Output Current (VDDIO=3.15V, VDDD=1.375V) (See Note 3, 4, 7)	265			mA
DCDC Converters (4.0V < BATT < 4.242V)	**Min**	**Typ**	**Max**	**Units**
DCDC_VDDIO Maximum Output Current (VDDIO=3.30V, VDDD=1.55V) (See Note 3, 4, 7)	355			mA
DCDC_VDDIO Maximum Output Current (VDDIO=3.20V, VDDD=1.55V) (See Note 3, 4, 7)	340			mA
DCDC_VDDIO Maximum Output Current (VDDIO=3.15V, VDDD=1.55V) (See Note 3, 4, 7)	325			mA
DCDC_VDDIO Maximum Output Current (VDDIO=3.15V, VDDD=1.375V) (See Note 3, 4, 7)	420			mA
VDD4P2 Regulated Output	**Min**	**Typ**	**Max**	**Units**
VDD4P2 Output Voltage Accuracy (TARGET=4.2V) (Note 1)	-2		+1	%
VDD4P2 Maximum Output Current (VDD5V=5.00V, ILIMIT=780mA) (See Note 5)	725			mA
VDD4P2 Maximum Output Current (VDD5V=4.75V, ILIMIT=780mA) (See Note 5)	605			mA
VDD4P2 Maximum Output Current (VDD5V=4.40V, ILIMIT=780mA) (See Note 5)	325			mA
VDD4P2 Output Impedance (See Note 9)		90		mΩ
VDD4P2 Output Current Limit Accuracy (VDD5V=4.75V, ILIMIT=480mA) (See Note 6)	466	480	505	mA
VDD4P2 Output Current Limit Accuracy (VDD5V=4.75V, ILIMIT=100mA) (See Note 6)	95	100	105	mA
Battery Charger	**Min**	**Typ**	**Max**	**Units**
Final Charge Voltage Accuracy (TARGET=4.2V)	-2		+1	%

1 No Load.

2 Output regulated within 100 mV of target voltage.

3 DCDC Double FETs Enabled, Inductor Value = 15µH.

4 Assumes simultaneous load of IDDD = 250 mA@1.55V and IDDA = 200 mA@1.8V.

5 Output regulated within 300 mV of target voltage.

6 Untuned.

7 The DCDC Converter is a triple output buck converter. The maximum output current capability of each output of the converter is dependent on the loads on the other two outputs. For a given output, it may be possible to achieve a maximum output current higher than that specificed by ensuring the load on the other outputs is well below the maximum.

8 Because the internal linear regulators are cascaded, it is not possible to simultaneously operate the VDDIO, VDDA, VDDM, and VDDD linear regulators at the maximum specified load current. For example, the VDDIO linear regulator provides current to both the VDDIO 3.3 V supply rail as well as the VDDM and VDDA linear regulator inputs. Likewise, the VDDA linear regulator provides current to both the 1.8 V supply rail as well as the VDDD linear regulator input. The application designer should ensure the following two conditions are met:

- (VDDIO Load Current + VDDM Load Current + VDDA Load Current) < VDDIO Maximum Output Current
- (VDDA Load Current + VDDD Load Current) < VDDA Maximum Output Current

9) The output impedance value is measured on a PCB evaluation board and includes the PCB trace impedance.

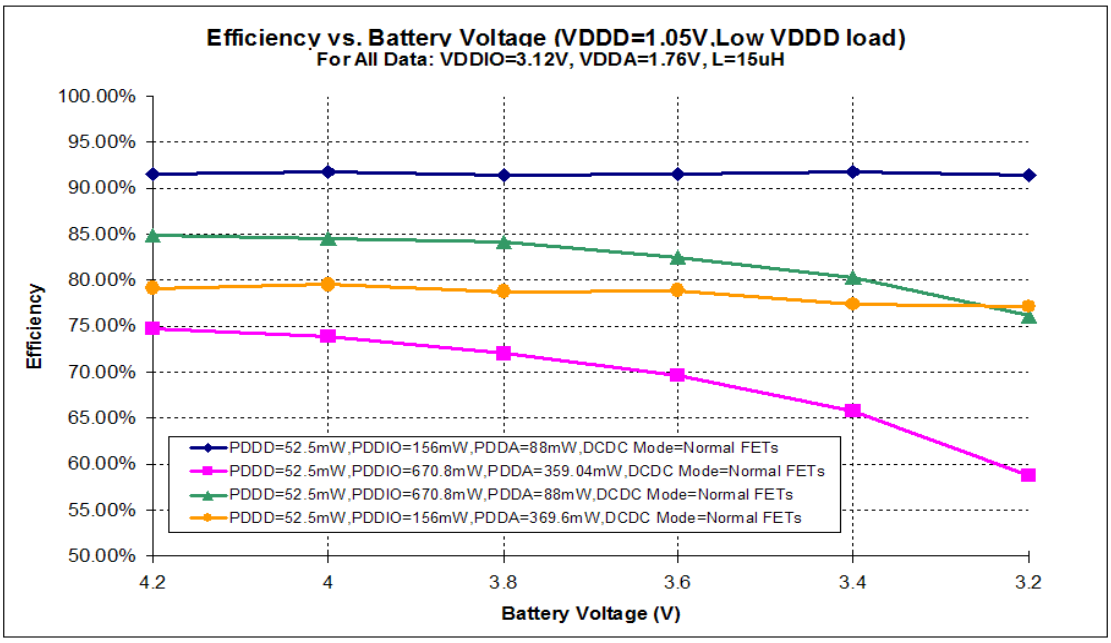


Figure 2-1. DCDC Efficiency vs. Battery Voltage (VDDD=1.05V, Low VDDD Load)

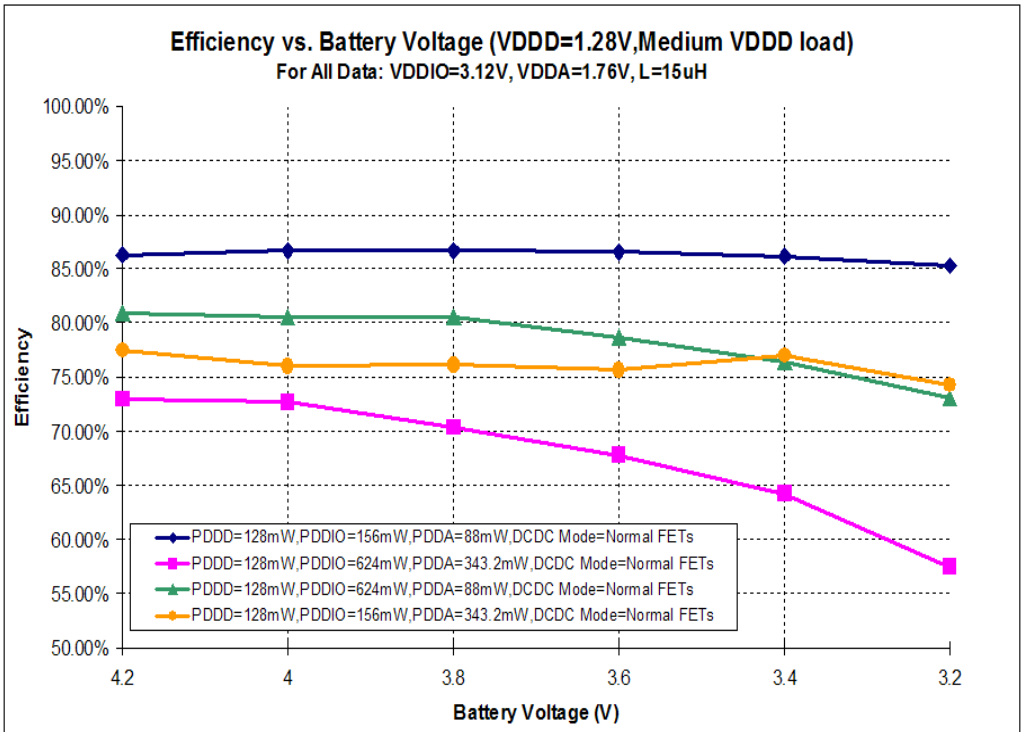


Figure 2-2. DCDC Efficiency vs. Battery Voltage (VDDD=1.28V, Medium VDDD Load)

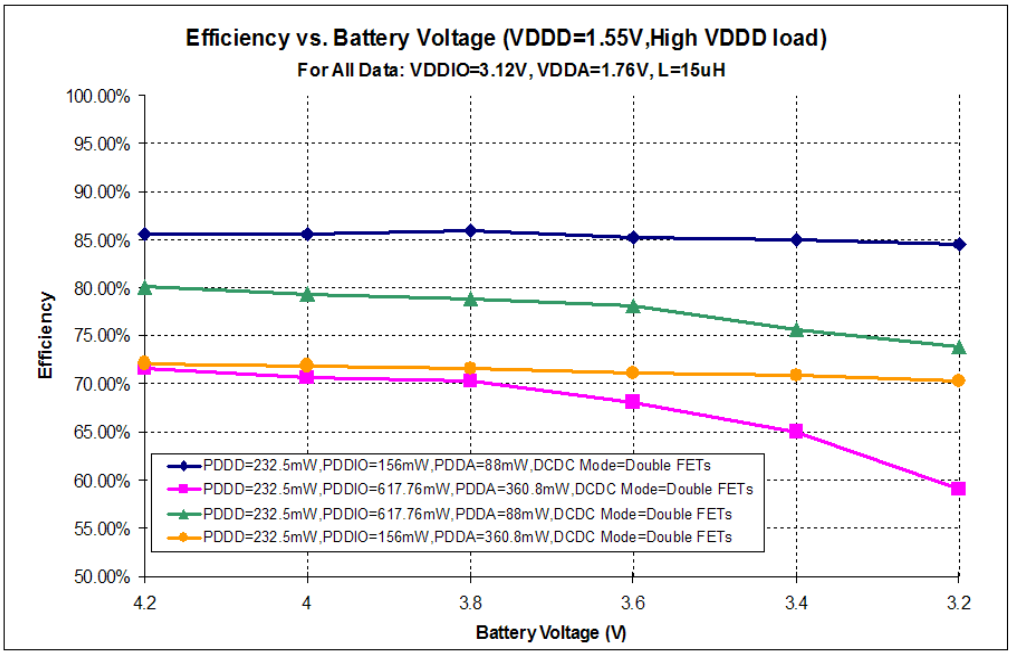


Figure 2-3. DCDC Efficiency vs. Battery Voltage (VDDD=1.55V, High VDDD Load)

Table 2-8. Non-EMI Digital Pin DC Characteristics

		VDDIO = 3.3 V		
Parameter	**Name**	**Min**	**Max**	**Units**
Non-EMI Regular & High Drive I/O Input Voltage	VIH	2.00	VDDIO	V
	VIL	-	0.80	V
Non-EMI Regular & High Drive I/O Output Voltage	VOH	0.8 * VDDIO	-	V
	VOL	-	0.40	V
Non-EMI Regular I/O Output Current (see notes 1 and 6 of Table 2-10)	IOH - 4mA	3.60	-	mA
	IOH - 8mA	7.20	-	mA
	IOH - 12mA	10.80	-	mA
	IOL - 4mA	-3.60	-	mA
	IOL - 8mA	-7.20	-	mA
	IOL - 12mA	-10.80	-	mA
Non-EMI High Drive I/O (PWM4) Output Current (see notes 2 and 6 of Table 2-10)	IOH - 8mA	-6.50	-	mA
	IOH - 16mA	-11.00	-	mA
	IOH - 24mA	-16.80	-	mA
	IOL - 8mA	-8.00	-	mA
	IOL - 16mA	-14.50	-	mA
	IOL - 24mA	-19.00	-	mA
External Pull-Up / Pull-Down Resistor Value Required to Overdrive Internal Gate Keeper		-	50	kΩ
Internal Pull-Up Resistor Accuracy	-20	-	+20	%

.

Table 2-9. EMI Digital Pin DC Characteristics

		VDDIO.EMI = 2.5 V		VDDIO.EMI = 1.8 V		Unit
Parameter	**Name**	**Min**	**Max**	**Min**	**Max**	
EMI I/O Input Voltage	VIH	(VDDIO.EMI / 2) + 0.2	VDDIO.EMI	(VDDIO.EMI / 2) + 0.2	VDDIO.EMI	V
	VIL	-	(VDDIO.EMI / 2) - 0.2	-	(VDDIO.EMI / 2) - 0.2	V
EMI I/O Output Voltage	VOH	0.7 * VDDIO.EMI	-	0.8 * VDDIO.EMI	-	V
	VOL	-	0.40	-	0.2 * VDDIO.EMI	V
EMI I/O Output Current (See notes 1 and 6 of Table 2-10)	IOH - 4 mA	3.00	-	3.00	-	mA
	IOH - 8 mA	6.00	-	5.00	-	mA
	IOH - 12 mA (see note 3 of Table 2-10)	10.00	-	8.50	-	mA
	IOH - 16 mA (see note 3 of Table 2-10)	14.00	-	11.00	-	mA
	IOL - 4 mA	-4.00	-	-3.50	-	mA
	IOL - 8 mA	-8.00	-	-7.00	-	mA
	IOL - 12 mA (see note 3 of Table 2-10)	-12.00	-	-10.50	-	mA
	IOL - 16 mA (see note 3 of Table 2-10)	-16.00	-	-13.50	-	mA

Table 2-10. External Devices Supported by the EMI

DRAM Device	**Max Load** (see notes 4 and 5)	**Pad Voltage**
DDR	15 pF	2.5 V
mDDR	15 pF	1.8 V

1 The stronger the driver mode, the noisier the on-chip power supply. The use of a stronger drive mode must be limited to only a few pins. The majority of GPIO drivers must be set in 4-mA mode.

2 High-drive I/O has a high current source/sink capability. However, it is not meant as high-speed I/O - the driver turns on slowly to reduce L*di/dt power supply noise.

3 The EMI I/O pad pre-drivers are powered from VDDIO rather than VDDIO.EMI. This causes the higher EMI I/O drive strengths at 2.5 V and 1.8 V to have a dependency on the VDDIO voltage. For 2.5 V and 1.8 V EMI I/O 12 mA & 16 mA drive strengths, VDDIO should equal 3.3 V or higher.

[4] Max load includes capacitive load due to PCB traces, pad capacitance and driver self-loading.

[5] Setting is for worst case. Freescale's EMI interface uses less powerful drivers than those typically used in mDDR devices. A possible transmission-line effect on the PC board must be suppressed by minimizing the trace length combined with Freescale's slower edge-rate drivers. The i.MX23 provides up to 16 mA programmable drive strength. However, the 16-mA mode is an experimental mode. With the 16-mA mode, the EMI function may be impaired by simultaneous switching output (SSO) noise. In general, the stronger the driver mode, the noisier the on-chip power supply. Freescale recommends not using a stronger driver mode than is required. Because on-chip power and ground noise is proportional to the inductance of its return path, users should make their best effort to reduce inductance between the EMI power and ground balls and the PC board power and ground planes.

[6] IOH is the maximum output current at which the VOH specification is met. IOL is the maximum input current at which the VOL specification is met.

2.3.1 Recommended Operating Conditions for Specific Clock Targets

NOTE

At this time, all data is preliminary and subject to change without notice.

Table 2-11. System Clocks

Name	Min. Freq. (MHz)	Max. Freq. (MHz)	Description
clk_gpmi	-	102.858	General Purpose memory interface clock domain
clk_ssp	-	102.858	Internal SSP Interface clock.
External SSP Clock	-	51.429	External SSP clock.

Table 2-12. Recommended Operating States - 169BGA Package

VDDD (V)	VDDD Brown-out (V)	HW_ DIGCTRL ARMCACHE (note 1)	CPUCLK / clk_p Frequency (MHz)	HW_ CLKCTRL CPU_DIV_CPU	HW_ CLKCTRL FRAC_ CPUFRC / PFD	AHBCLK / clk_h Frequency (MHz)	HW_ CLKCTRL HBUS_DIV	EMICLK / clk_emi Frequency (MHz)	HW_ CLKCTRL EMI_ DIV_EMI	HW_ CLKCTRL FRAC_ EMIFRAC	SUPPORTED DRAM
1.050	0.975		24.00			24.00	1	24.00			DDR, mDDR
1.050	0.975	11	64.00	5	27	64.00	1	64.00	5	27	DDR, mDDR
1.275	1.175	00	261.82	1	33	130.91	2	130.91	2	33	DDR, mDDR
1.375	1.275	00	360.00	1	24	120.00	3	120.00	3	24	DDR, mDDR
1.475	1.375	00	392.73	1	22	196.36	2	130.91	2	33	DDR, mDDR
1.550	1.450	00	454.74	1	19	151.58	3	151.58	3	19	mDDR
1.550	1.450	00	454.74	1	19	151.58	3	130.91	2	33	DDR

1 All timing control bit fields in HW_DIGCTRL_ARMCACHE should be set to the same value.

Table 2-13. Recommended Operating States - 128QFP Package

VDDD (V)	VDDD Brown-out (V)	HW_ DIGCTRL ARMCACHE (note 1)	CPUCLK / clk_p Frequency (MHz)	HW_ CLKCTRL CPU_DIV_CPU	HW_ CLKCTRL FRAC_ CPUFRC / PFD	AHBCLK / clk_h Frequency (MHz)	HW_ CLKCTRL HBUS_DIV	EMICLK / clk_emi Frequency (MHz)	HW_ CLKCTRL EMI_ DIV_EMI	HW_ CLKCTRL FRAC_ EMIFRAC	SUPPORTED DRAM
1.050	0.975		24.00			24.00	1	24.00			mDDR
1.050	0.975	11	64.00	5	27	64.00	1	64.00	5	27	DDR, mDDR
1.275	1.175	00	261.82	1	33	130.91	2	130.91	2	33	DDR, mDDR
1.375	1.275	00	360.00	1	24	120.00	3	120.00	3	24	DDR, mDDR
1.475	1.375	00	392.73	1	22	196.36	2	130.91	2	33	DDR, mDDR
1.550	1.450	00	454.74	1	19	151.58	3	130.91	2	33	DDR, mDDR

Table 2-14. Recommended Operating Conditions - CPU Clock (clk_p)

Minimum VDDD (V)	Minimum $VDDD_{Brown\text{-}out}$ (V)	HW_DIGCTRL ARMCACHE (note 1)	HW_CLKCTRL FRAC_CPUFRC / PFD	CPUCLK / clk_p Frequency max (MHz)
1.050	0.975	11	25 - 35	64.00
1.225	1.125	00	18 - 35	278.71
1.375	1.275	00	18 - 35	360.00
1.450	1.350	00	18 - 35	392.73
1.550	1.450	00	18 - 35	454.74

1 All timing control bit fields in HW_DIGCTRL_ARMCACHE should be set to the same value.

Table 2-15. Recommended Operating Conditions - AHB Clock (clk_h)

Minimum VDDD (V)	Minimum $VDDD_{Brown\text{-}out}$ (V)	HW_DIGCTRL ARMCACHE (note 1)	HW_CLKCTRL FRAC_CPUFRC / PFD	AHBCLK / clk_h Frequency max (MHz)
1.050	0.975	11	25 - 35	64.00
1.275	1.175	00	18 - 35	130.91
1.350	1.250	00	18 - 35	160.00
1.475	1.375	00	18 - 35	196.36
1.525	1.425	00	18 - 35	205.71

1 All timing control bit fields in HW_DIGCTRL_ARMCACHE should be set to the same value.

Table 2-16. Frequency vs. Voltage for EMICLK - 169-Pin BGA Package

Minimum VDDD (V)	Minimum $VDDD_{Brownout}$ (V)	EMICLK Fmax (MHz)	
		DDR (note 1)	mDDR (note 2)
1.55	1.45	130.91	151.58

Table 2-16. Frequency vs. Voltage for EMICLK - 169-Pin BGA Package (continued)

Minimum VDDD (V)	Minimum $VDDD_{Brownout}$ (V)	EMICLK Fmax (MHz)	
		DDR (note 1)	mDDR (note 2)
1.45	1.35	130.91	151.58
1.30	1.20	130.91	151.58
1.20	1.10	130.91	151.58
1.05	0.975	96.00	96.00

1) DDR EMICLK maximum is valid for the following conditions: Temp ≤ 105°, 2.4V ≤ VDDM ≤ 2.5V, 3.2V ≤ VDDIO ≤ 3.3V, Drive strength ≥ 12mA

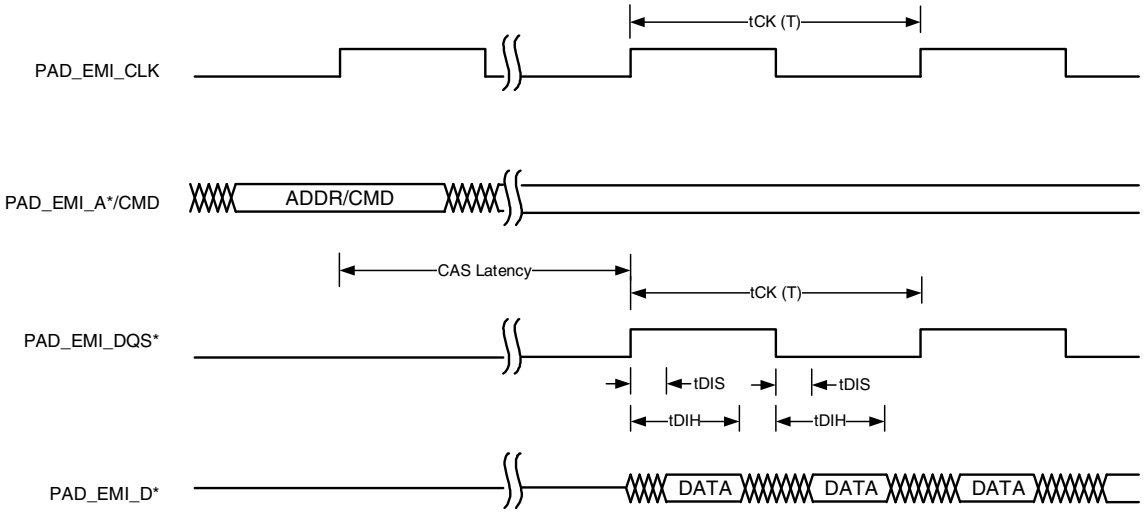
Table 2-17. Frequency vs. Voltage for EMICLK - 128-Pin LQFP Package

Minimum VDDD (V)	Minimum $VDDD_{Brownout}$ (V)	EMICLK Fmax (MHz)	
		DDR (note 1)	mDDR (note 2)
1.55	1.45	130.91	130.91
1.45	1.35	130.91	130.91
1.30	1.20	130.91	130.91
1.20	1.10	130.91	130.91
1.05	0.975	96.00	96.00

2) mDDR EMICLK maximum is valid for the following conditions: Temp ≤ 105°, 1.7V ≤ VDDA ≤ 1.9V, 3.2V ≤ VDDIO ≤ 3.3V, Drive strength ≥ 12mA

2.4 AC Characteristics

2.4.1 EMI Electrical Specifications



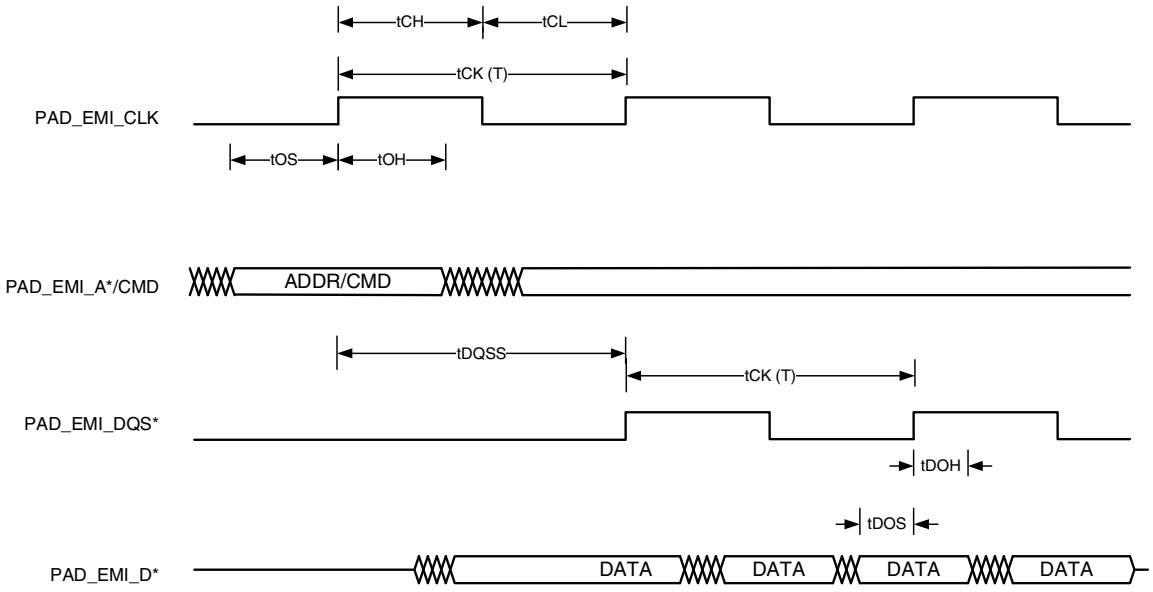
Assumptions
==========
VDDD PVT : 1.08V, SS, 125C Junction
VDDA IO PVT : 1.62V, SS, 125C Junction (1.8V setting, but WCS IO voltage)
IO Drive Strength = 4mA, Cap Load = 15pF on all pins
DQS has pull-downs on board (never goes high-Z), but DQ has keepers disabled.

DQS In Delay chain setting = 4 taps WCS, 13 taps BCS (approx ¼ cycle, ie approx 0x20)

Note that the SoC creates an internal delay on the DQS relative to DQ, so data launched from the DRAM on the rising edge of the DQS will set-up to the rising edge of the DQS, and will hold to the previous edge.

Legend
======
tDIS = Data Input Max Setup Time relative to DQS = 0.25T - 0.85 (e.g. at 151.58MHz, tDQSQ cannot exceed 0.25*(1000/151.58) - 0.85 = 0.8ns)
tDIH = Data Input Minimum Hold Time relative to DQS = 0.25T + 0.75 (e.g. at 151.58MHz, tQH must be at least 0.25*(1000/151.58) + 0.75 = 2.4ns)

Figure 2-4. i.MX23 EMI mDDR DRAM Input AC Timing



Assumptions
==========
VDDD PVT : 1.08V, SS, 125C Junction (unless otherwise noted)
VDDA IO PVT : 1.62V, SS, 125C Junction (1.8V setting, but WCS IO voltage)
IO Drive Strength = 4mA, Cap Load = 15pF on all pins
DQS has pull-downs on board (never goes high-Z), but DQ has keepers disabled.

DQS Out Delay chain setting = 0
DQS Write Clock Delay chain setting = 5 taps (approx ¼ cycle, ie approximately 0x20)
Clock Delay line setting = 5 (this also works at BCS PVT and gives best CK/DQS skew)

Legend
======
tCK = T = DRAM Clock Cycle Time = @ VDDD=1.55V 6.6ns (min), @ VDDD=1V 7.639ns (min)
tCH = DRAM Clock High Pulse = T/2 to T/2 - 0.37ns
tCL = DRAM Clock Low Pulse = T/2 to T/2 + 0.37ns
tOS = Addr/Cmd output setup to CK rising = T/2 - 0.96ns (min)
tOH = Addr/Cmd output hold to CK rising = T/2 - 1.51ns (min)

tDQSS = Write command valid to first DQS latching transition = T to T+0.1

tDOS = DQ to DQS setup time = T/4 - 0.485ns (min)
tDOH = DQ to DQS hold time = T/4 - 0.365ns (min)

Figure 2-5. i.MX23 EMI mDDR DRAM Output AC Timing

2.4.2 I²C Electrical Specifications

Figure 2-6 depicts the timing for the I²C module. Table 2-18 lists the I²C module timing parameters.

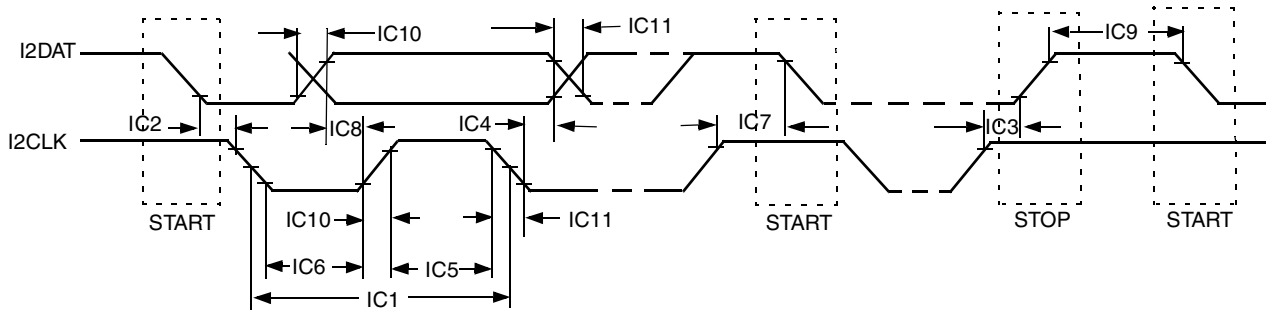


Figure 2-6. I²C Bus Timing Diagram

Table 2-18. I²C Timing Parameters

ID	Parameter	Min	Max	Unit
	I²C Input Timing			
IC1	I2CLK cycle time [a]	8	-	clk_x cycles
IC2	Hold time (repeated) START condition	1	-	clk_x cycles
IC3	Set-up time for STOP condition	1	-	clk_x cycles
IC4	Data hold time	1	-	clk_x cycles
IC5	HIGH Period of I2CLK Clock	4	-	clk_x cycles
IC6	LOW Period of the I2CLK Clock	4	-	clk_x cycles
IC7	Set-up time for a repeated START condition	1	-	clk_x cycles
IC8	Data set-up time	1	-	clk_x cycles
IC9	Bus free time between a STOP and START condition	4	-	clk_x cycles
	I²C Output Timing			
IC1	I2CLK cycle time [b]	high_count + low_count + 6	-	clk_x cycles
IC2	Hold time (repeated) START condition	leadin_count	-	clk_x cycles
IC3	Set-up time for STOP condition	rcv_count + 6	-	clk_x cycles
IC4	Data hold time	xmit_count	-	clk_x cycles
IC5	HIGH Period of I2CLK Clock	high_count	-	clk_x cycles
IC6	LOW Period of the I2CLK Clock	low_count	-	clk_x cycles
IC7	Set-up time for a repeated START condition	bus_free + 7	-	clk_x cycles
IC8	Data set-up time	low_count - xmit_count	-	clk_x cycles
IC9	Bus free time between a STOP and START condition	bus_free + 7 - rcv_count	-	clk_x cycles
IC10	Rise time of both I2DAT and I2CLK signals	0.15*Cb	0.17*Cb	ns

Table 2-18. I²C Timing Parameters (continued)

ID	Parameter	Min	Max	Unit
IC11	Fall time of both I2DAT and I2CLK signals	0.16*Cb	0.17*Cb	ns
IC12	Capacitive load for each bus line (C_b) [c]	5	100	pF

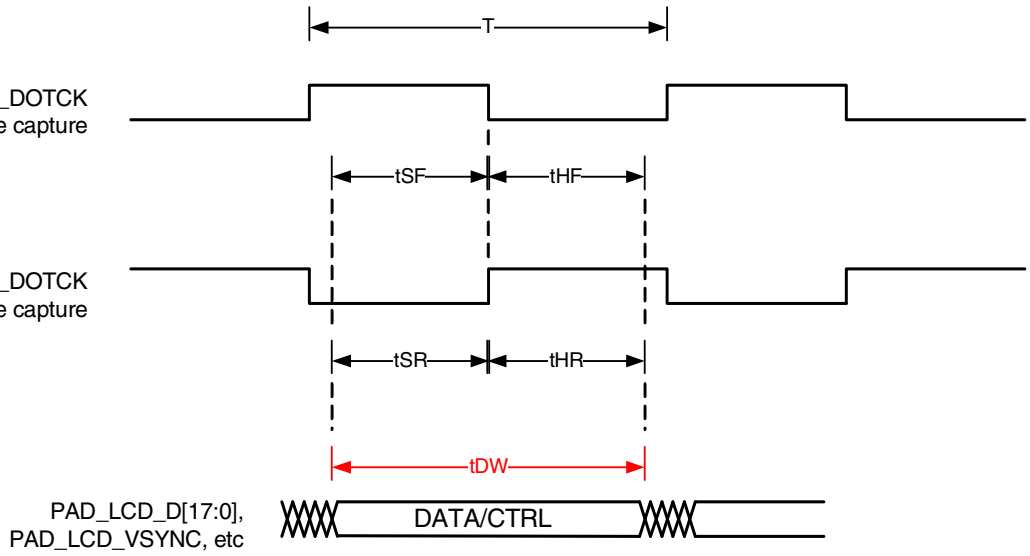
[a] The clk_x period is programmed as a divide with respect to the xtal clock. The divide value can be >= 1.

[b] All I2C output timings are determined by PIO register values. These values are multiplied by the programmable clk_x period.

[c] C_b = total capacitance of one bus line in pF.

2.4.3 LCD AC Output Electrical Specifications

Figure 2-7 depicts the AC output timing for the LCD module. Table 2-19 lists the LCD module timing parameters.



Notes:
T = LCD interface clock period
I/O Drive Strength = 4mA
I/O Voltage = 3.3V
Cck = Capacitance load on DOTCK pad
Cd = Capacitance load on DATA/CTRL pad

Figure 2-7. LCD AC Output Timing Digram

Table 2-19. LCD AC Output Timing Parameters

ID	Parameter	
tSF	Data setup for falling edge	DOTCK = T/2 – 1.97ns + 0.15*Cck – 0.19*Cd
tHF	Data hold for falling edge	DOTCK = T/2 + 0.29ns + 0.09*Cd – 0.10*Cck
tSR	Data setup for rising edge	DOTCK = T/2 – 2.09ns + 0.18*Cck – 0.19*Cd
tHR	Data hold for rising edge	DOTCK = T/2 + 0.40ns + 0.09*Cd – 0.10*Cck
tDW	Data valid window	tDW = T – 1.45ns

Chapter 3
ARM CPU Complex

This chapter describes the ARM CPU included on the i.MX23 and includes sections on the processor core, the JTAG debugger, and the embedded trace macrocell (ETM) interface.

3.1 ARM 926 Processor Core

The on-chip Reduced Instruction Set Computer (RISC) processor core is an ARM, Ltd. 926EJ-S. This CPU implements the ARM v5TE instruction set architecture, which includes enhanced DSP instructions.

The ARM9EJ-S has two instruction sets: a 32-bit instruction set used in the ARM state and a 16-bit instruction set used in Thumb state. The core offers the choice of running in the ARM state or the Thumb state or a mix of the two. This enables optimization for both code density and performance.

A block diagram of the ARM926EJ-S core is shown in Figure 3-1.

See http://www.arm.com/documentation/ARMProcessor_Cores/index.html to download the following ARM documentation on the ARM926EJ-S core:

- ARM926EJ-S Technical Reference Manual, DDI0198D
- ARM926EJ-S Development Chip Reference Manual, DDI0287A

The ARM9 core has a total of 37 programmer-visible registers, including 31 general-purpose 32-bit registers, six 32-bit status registers, and a 32-bit program counter, as shown in Figure 3-2. In ARM state, 16 general-purpose registers and one or two status registers are accessible at any one time. In privileged modes, mode-specific banked registers become available.

The ARM state register set contains 16 directly addressable registers, r0 through r15. An additional register, the current program status register (CPSR), contains condition code flags and the current mode bits. Registers r0–r13 are general-purpose registers used to hold data and address values, with R13 being used as a stack pointer. R14 is used as the subroutine link register (lr) to hold the return address. Register r15 holds the program counter (PC).

The Thumb state register set is a subset of the ARM register set. The programmer has access to eight general-purpose registers, r0–r7, the PC (ARM r15), the stack pointer (ARM r13), the link register (ARM r14), and the cpsr.

Exceptions arise whenever the normal flow of program execution has to be temporarily suspended, for example, to service an interrupt from a peripheral. Before attempting to handle an exception, the ARM

core preserves the current processor state, so that the original program can resume when the handler is finished.

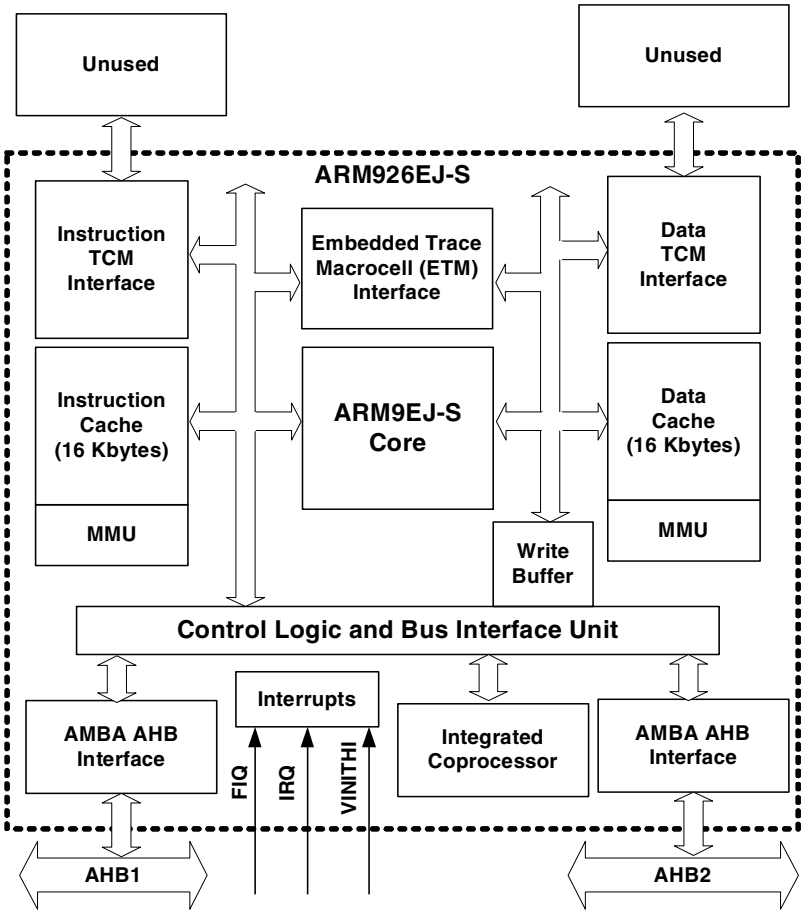


Figure 3-1. ARM926 RISC Processor Core

The following exceptions are recognized by the core:

- SWI—Software interrupt
- UNDEF—Undefined instruction
- PABT—Instruction prefetch abort
- FIQ—Fast peripheral interrupt
- IRQ—Normal peripheral interrupt
- DABT—Data abort
- RESET—Reset
- BKPT—Breakpoint

The vector table pointing to these interrupts can be located at physical address 0x00000000 or 0xFFFF0000. The i.MX23 maps its 64-Kbyte on-chip ROM to the address 0xFFFF0000 to 0xFFFFFFFF. The core is hardwired to use the high address vector table at hard reset (core port VINITHI =1).

The ARM 926 core includes a 16-Kbyte instruction cache and 16-Kbyte data cache and has two master interfaces to the AMBA AHB, as shown in Figure 3-1.

The i.MX23 always operates in little-endian mode.

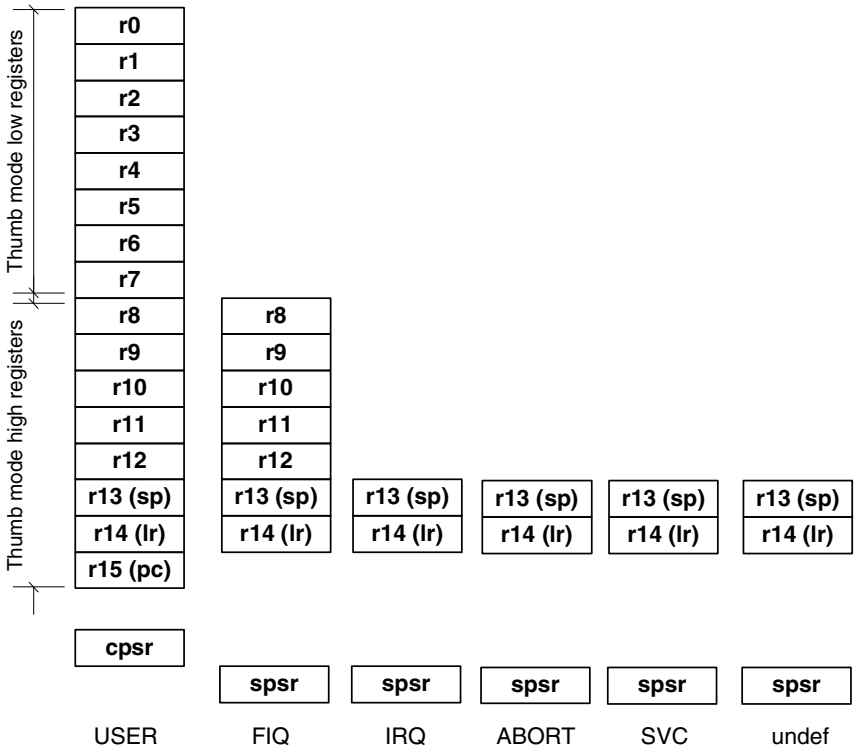


Figure 3-2. ARM Programmable Registers

3.2 JTAG Debugger

The TAP controller of the ARM core in the i.MX23 performs the standard debugger instructions.

3.2.1 JTAG READ ID

The TAP controller returns the following 32-bit data value in response to a JTAG READ ID instruction: 0x0792_64F3

3.2.2 JTAG Hardware Reset

The JTAG reset instruction can be accomplished by writing 0xDEADC0DE to ETM address 0x70. The ETM is on scan chain 6. The bitstream is 0xF0DEADC0DE.

The digital wide reset does not affect the DC-DC converters or the contents of the persistent registers in the analog side of the RTC.

3.2.3 JTAG Interaction with CPUCLK

Because the JTAG clock is sampled from the processor clock CPUCLK, there are cases in which the behavior of CPUCLK affects the ability to make use of JTAG. Specifically, the JTAG block will not function as expected if:

- CPUCLK is stalled due to an interrupt
- CPUCLK is less than 3x the JTAG clock
- CPUCLK is disabled for any reason

3.3 Embedded Trace Macrocell (ETM) Interface (169BGA-only)

The i.MX23 includes a stand-alone ARM CoreSight Embedded Trace Macrocell, ETM9CSSingle, which provides instruction trace and data trace for the ARM9 microprocessor. For more details see the CoreSight ETM9 Technical Reference Manual. Also, see the pin list in Chapter 36, "Pin Descriptions," for pinout information.

Chapter 4
Clock Generation and Control

4.1 Overview

The clock control module, or CLKCTRL, generates the clock domains for all components in the i.MX23 system. The crystal clock or PLL clock are the two fundamental sources used to produce all the clock domains. For lower performance and reduced power consumption, the crystal clock is selected. The PLL is selected for higher performance requirements but requires increased power consumption. In most cases, when the PLL is used as the source, a phase fractional divider (PFD) can be programmed to reduce the PLL clock frequency by up to a factor of 2.

The PLL and PFD clocks are used as reference clock sources to drive digital clock dividers in the clock control module. These reference clocks, or ref_<clock>, drive the digital clock dividers in CLKCTRL. The digital clock dividers have three modes of operation, integer divide mode, fractional divide mode, and gated clock mode. The details of these three modes will be described to understand which mode should be selected to achieve the desired frequency.

All programming control for system clocks are contained in the CLKCTRL module. All clock domains have a programmable clock frequency to meet application requirements. Also, all analog clock control programming is done indirectly through the CLKCTRL module. This contains the complexity of overall system clock selection to a single device. Also, the hardware used to generate all clock domains is replicated. Following is a description of all clock domains in the i.MX23 system.

4.2 Clock Structure

The reference clocks are used in CLKCTRL as fundamental clock sources to produce clock domains throughout the system. A reference clock can be either the crystal clock, 480Mhz PLL, or PFD output from the analog module. The selected reference clock is used by a digital clock divider to produce the desired clock domain. The table below summarizes all available reference clocks used within the CLKCTRL and all clock domains used in the system. The diagram that follows depicts all clock domains and how they are connected within the CLKCTRL module. This should provide a reference for how clocks are generated within the i.MX23 system.

4.2.1 Table of System Clocks

Table 4-1 summarizes the clocks produced by the clock control module.

Table 4-1. System Clocks

NAME	REFERENCE	DIVIDE /FREQ	DESCRIPTION
Reference Clocks.			
ref_xtal	xtal_24m /ring_24m	1	This is the muxed select between the internal ring oscillator and the external crystal.
ref_cpu	PLL	9 phase	The 9 phase fractional divider output used as the reference for the CPU clock divider.
ref_emi	PLL	9 phase	The 9 phase fractional divider output used as the reference for the EMI clock divider.
ref_io	PLL	9 phase	The 9 phase fractional divider output used as the reference for the GPMI, SSP, and IR clock dividers.
ref_pix	PLL	9 phase	The 9 phase fractional divider output used as the reference for the PIX clock divider.
ref_vid	PLL	9 phase	The 9 phase fractional divider output used as the reference for the clk_tv108m clock divider.
ref_pll	PLL	1	This is the raw PLL output used as the reference for the SAIF clock divider.
Divided clock domains referenced from PLL or Xtal clock.			
clk_p	ref_xtal /ref_cpu	10/6 bits	ARM core clock.
clk_h	clk_p	5 bits	AHB/APBH clock domain. clk_h is a gated branch of the clk_p domain.
clk_etm	ref_xtal /ref_cpu	6/6 bits	ARM etm clock.
clk_emi	ref_xtal /ref_emi /ref_cpu	4/6 bits	External DDR interface clock.
clk_ssp	ref_xtal /ref_io	8 bits	SSP interface clock.
clk_gpmi	ref_xtal /ref_io	8 bits	General purpose memory interface clock domain.
clk_irov/ir	ref_io /clk_irov	9/10 bits	Over sample IR clock and IR data bit clock. The IROV clock has the ref_io as its reference. The IR clock domain uses the clk_irov domain as its reference.
clk_spdif /clk_pcmspdif	ref_xtal /ref_io		Clk_spdif is an intermediate clock that drives the clk_pcmspdif fractional clock divider.
clk_pix	ref_xtal /ref_pix	DDA	External display interface clock. Its reference is the xtal or fractional divider output that drives a DDA fractional divider.
clk_saif	ref_xtal /ref_pll	DDA	Serial Audio Interface clock domain. Its reference is the PLL clock output which drives a DDA fractional divider.
Divided clock domains referenced from Xtal clock.			
clk_x	ref_xtal	10 bits	APBX clock domain.
clk_uart	ref_xtal	2 bits	UART clock domain.
Fixed clock domains.			
clk_xtal24m	ref_xtal	24Mhz	Used for the DRI, filter, and analog 24Mhz clock domains.
clk_32k	ref_xtal32k /ref_xtal	32khz	Fixed 32khz clock domain. The reference is either the 32kHz crystal or the 24Mhz crystal and divides by 768 to produce 32kHhz.

Table 4-1. System Clocks (continued)

NAME	REFERENCE	DIVIDE /FREQ	DESCRIPTION
clk_adc	ref_xtal	2khz	Fixed 2khz clock domain.
clk_tv108m_ng	ref_vid	108Mhz	Fixed 108Mhz clk domain.
clk_tv54m	int_108m	54Mhz	Fixed 54Mhz clk domain. The reference is a gated clock on the internal fixed 108Mhz clock.
clk_tv27m	int_108m	27Mhz	Fixed 27Mhz clk domain. The reference is a gated clock on the internal fixed 108Mhz clock.
clk_tvenc_fifo	Int_108m	54Mhz/27Mhz	Selectable between 54MHz and 27MHz with control bit from tvenc block. The reference is a gated clock on the internal fixed 108Mhz clock.

4.2.2 Logical Diagram of Clock Domains

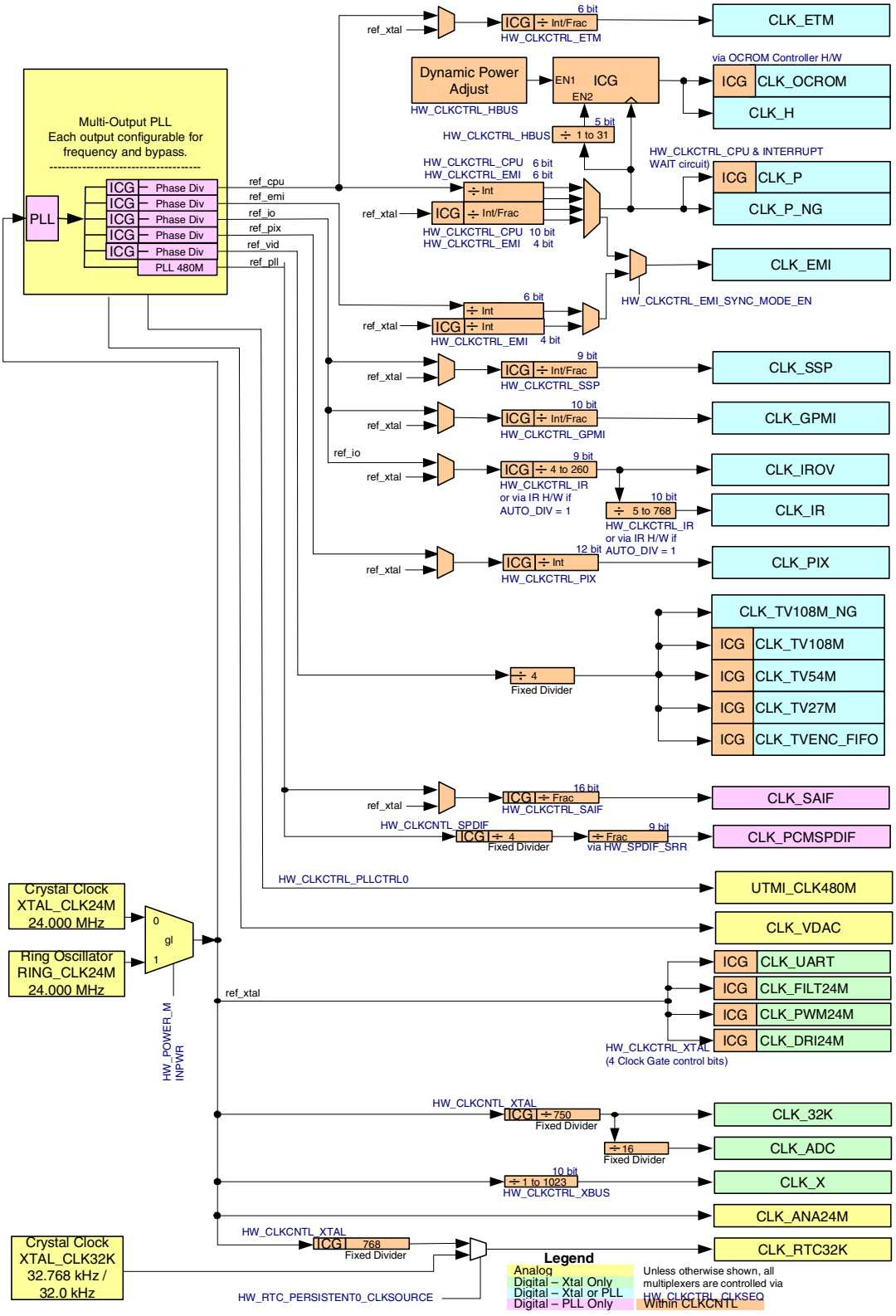


Figure 4-1. Logical Diagram of Clock Domains

4.2.3 Clock Domain Description

All major functional clock domains/branches have trunk level clock gating for power management. The intent is to gate clock domains off when modules for certain applications are not necessary. This clock gating is instantiated using an ICG element from the standard cell library. Software will have to enable the clock domain that drives on chip devices where trunk level clock gating is implemented.

The location of ICG elements to gate clock domains is not systematic. Since most of the clock structures throughout the system are unique, the location of ICGs for clock tree power reduction differs from one domain to the next. The location of these ICGs to gate off clock domains is apparent in the clock structure diagram.

All clock domains are asynchronous unless noted otherwise.

4.2.3.1 CLK_P, CLK_H

The clk_p domain is the used to drive the integrated ARM9 core. The reference for clk_p can be either ref_xtal or ref_cpu. The reference ref_cpu drives a 6 bit clock divider to provide a maximum divide down of the reference clock by 2^6. The reference ref_xtal drives a 10 bit clock divider to provide a maximum divide down of the selected reference clock by 2^10. All of the ARM core and SoC components on the clk_h branch are considered to be on the clk_p domain. clk_h is actually a branch of the clk_p domain. So, clk_h runs synchronous to clk_p.

The clk_h domain can be programmed to any divided ratio with respect to the clk_p domain depending on performance and power requirements. A dynamic clock frequency management controller monitors system performance requirements and scales the clk_h frequency to meet performance needs. When the CPU or support components require data transfer to/from system memory, the frequency manager scales the clk_h domain to meet the system performance requirements. Also, when the system is quiesed, the clk_h frequency is reduced to save power.

Clk_h has a 5 bit divider that divides the clk_p domain to produce the clk_h domain. The frequency for clk_h can be clk_p/32 <= clk_h <= clk_p. Two divide modes exist for the clk_h branch:

- Integer divide. In this mode, the value programmed in the hw_clkctrl_hbusclkctrl.div field represents an integer divide value.
- Fractional divide. In this mode, the value programmed in the hw_clkctrl_hbusclkctrl.div field represents a binary fraction. When the accumulation of the current count and the programmed divide value carry out of the most significant bit, a clk_h pulse is generated. For example, to achieve an 8:3 clk_p:clk_h clock ratio, set the div field to 0.01100 which represents (0*1/2) + (1*1/4) + (1*1/8) + (0*1/16) + (0*1/32). Note, fractional divide can not be used when clk_emi is synchronous with clk_h.

The clk_h branch can be further divided by the dynamic clock frequency adjustment logic, (hw_clkctrl_emi_sync_mode_en = 0 only). When all the system clk_h components are not busy and their respective busy signals are inactive, the clk_h branch is further divided down by the value in the

hw_clkctrl_hbusclkctrl register. The frequency reduction of the clk_h branch saves overall power con¬sumption. Note, the dynamic clock frequency adjustment logic should not be enabled when clk_emi is synchronous with clk_h.

4.2.3.2 CLK_EMI

The external memory interface domain is called clk_emi. This clock can be asynchronous to clk_h to achieve the highest possible clock rate for the EMI interface, or synchronously to minimize the incurred latency for CPU access to external DRAM. This option is provided to tradeoff the optimization of performace for systems that are dependent on memory access latency or throughput.

When the hw_clkctrl_emi_sync_mode_en bit is set to 1, clk_h is synchronous and edge aligned with the emi clock and clk_p. The emi clock dividers will set the frequency of clk_h and clk_emi domains when synchronous mode is selected. In synchronous mode, the dynamic clock frequency adjust logic should be disabled. This is required since DRAM devices cannot operate correctly with changing clock frequencies.

4.2.3.3 System Clocks

All reference clock domains used in the CLKCTRL are driven by replicated instances of the PFD pre dividers in the analog module. These PFD reference clocks drive replicated instances of a single digital clock divider design to create all system clocks. The following sections describe the digital clock dividers features and how they can be used to create clocks throughout the system. The CLKCTRL structural diagram should be used with the digital clock divider description to understand how clocks are generated in the i.MX23 system.

4.3 CLKCTRL Digital Clock Divider

The digital clock divider that is used to drive all functional clock domains has three modes of operation. These are:

- integer divide mode
- fractional divide mode
- gated clock divide mode

These modes are described in the following three sections.

4.3.1 Integer Clock Divide Mode

Each divider has the capability to divide an input reference frequency by a fixed integer value. This is the most common mode that will be used to select a particular clock frequency. For a desired clock frequency, first try to select a PFD reference clock frequency AND an integer clock divide value to achieve the desired clock domain frequency. This mode is selected when the respective “frac_en” field in the clock control register is logic 0. The divide value will be in the range of 1 to 2^N. When programming

the DIV field to 1, the reference clock for the domain is passed and the clock domain assumes the same frequency as the reference domain. When a value of 2 is programmed, the clock domain frequency will be half the reference clock frequency. The maximum divide value depends on the number of bits each digital clock divider implements. This is different for each digital clock divider. The number of bits implemented for each divider is indicated by each DIV field that controls each clock domain. Divide by zero is NOT a valid programming value for the DIV field of any clock control PIO register.

4.3.2 Fractional Clock Divide Mode

This mode is used to divide a reference clock in the range of 2 < div < 2^N. The fractional clock divider in the CLKCTRL module implements a fractional counter to approximate a divided clock with respect to the selected reference frequency. The accuracy of the output clock is depen¬dent on the extent of the bits used to implement the fractional counter. The reference clock frequency and the fractional divide value must both be selected to achieve the desired output frequency.

This mode is enabled when setting the FRAC_EN field of the respective clock domain control register to logic 1 AND the most significant bit of the DIV field is logic 0. Do NOT use this mode to divide the reference clock domain by an integer value (such as 4, 8, etc). Use the integer divide mode to achieve the best results for dividing by an integer.

NOTE

It is important to note that the nearest rising or falling edge of the input reference clock frequency is used to approximate the rising edge of the output clock domain. So, the output clock frequency will jitter based on the input reference clock frequency and the programmed fractional divide value.

4.3.2.1 Fractional Clock Divide Example, Divide by 3.5

As an example, if the desired divide value is 3.5, the digital approximation of 1/3.5 is 0.01001001 using an 8 bit fractional approximation. The most significant bit of the “div” field in this case is logic 0, so the fractional divide mode is selected. The following sequence indicates the first 8 values of the fractional clock divider. Notice the accumulated count is simply the current value incremented by the value programmed in the “div” field on each cycle.

1. 0.01001001
2. 0.10010010
3. 0.11011011
4. 1.00100100 (carry out of MSB initiates an output clock edge)
5. 0.01101101
6. 0.10110110
7. 0.11111111
8. 1.01001000 (output edge initiated)

When the carry out of the fractional count is one, a rising edge output pulse is initiated and the remainder of the accumulator is preserved. The sub fractional accumulated value is considered to determine if the output edge should occur on the falling edge of the reference clock or the rising edge of the reference clock to minimize output clock jitter

4.3.2.1.1 Fractional ClockDivide Example, Divide by 3/8

This example uses a 3 bit fractional accumulator to divide the reference clock input by 3/8. There are 3 output clock edges produced for every 8 input reference clock edges. An output edge is generated on every cycle that the fractional accumulator carries out of the most significant bit. Notice when the fractional component is .01, the output edge is shifted and generated off the falling edge of the input reference clock. This is done to produce the best output duty cycle that can be achieved based on the input reference clock frequency.

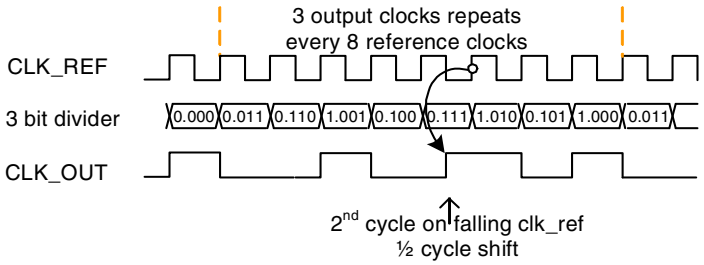


Figure 4-2. Fractional Clock divide; 3/8 example

4.3.3 Gated Clock Divide Mode

This mode is selected when the reference clock frequency is divided by a range of 1 < div < 2. To select this mode, program the FRAC_EN field to logic 1 and progarm the DIV field with the most significant bit set to logic 1. In this case, the reference clock is enabled/disabled on a cycle by cycle basis to pass to the output clock domain. Essentially, the reference clock is gated on or off depending on the cary out bit of the fractional count accumulator. This option is useful to divide the 24 MHz clock to a range between 12 to 24 MHz. The effective period is equal to the reference period since the output clock is a gated version of the reference clock. For example, a divide value of 4/3 will allow 3 consecutive pulses of the reference clock to propagate and will then gate off a single refer¬ence clock cycle. The edge to edge timing is effectively equal to the reference clock.

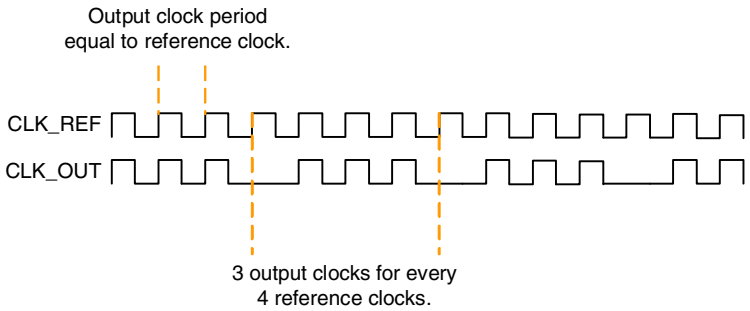


Figure 4-3. Divide Range 1 < div < 2

4.4 Clock Frequency Management

Clock frequency selection for some domains can be a function of multiple reference clock sources and divide parameters that are set in the CLKCTRL PIO control registers. The most extreme case is using a programmable fractional PLL clock divider, a multiplexer that selects either the xtal clock or fractional PLL clock as a source to drive the CLKCTRL divider, and the divide value for the CLKCTRL divider itself. When programming a selected frequency, the sequence of events to achieve a given frequency must maintain the integrity of the system as a whole. During a clock system context switch, intermediate clock frequencies for selected domains cannot be faster than the sub system or I/O interface is designed to support.

It is expected that the sequence of events when a clock domain is tuned to a desired frequency be managed by software using a hardware status polling mechanism. Each parameter has an associated enable bit so that all the divide parameters can be programmed in advance of the parameters taking effect. A single register, hw_clkctrl_clkseq, contains all the enable bits that cause the divide parameters to take effect. The enable bits can be set, the busy bits can be polled for each parameter, and thus the enable/busy sequencing via software control can manage the tuning of clock frequencies throughout the system.

4.5 Analog Clock Control

Analog clock control is performed indirectly through PIO accessable registers in the CLKCTRL module. The analog circuits that are controlled via CLKCTRL PIO access are the PLL and all instances of the phase fractional dividers, or PFDs.

4.6 CPU and EMI Clock Programming

A defined protocol is necessary for selecting clock frequencies and root sources for driving the clk_p and clk_emi domains. These two clock structures are unique in that they each implement a separate divider, one referenced by xtal clock and a second referenced by a PLL/PFD structure. The “roots” of these clocks must be programmed in order of the sources furthest from the trunk first. Elements in the clock roots should subsequently be configured along the root path up to the desired clock trunk. The programming sequence to go from a clock that is referenced from the xtal clock to the PLL is outlined below. This is the

case when the device is in low power operation and there exists the need for higher clock rates to meet the demands of a more compute-intensive application.

1. The crystal is the current source for the CPU or EMI clock domain.
2. Enable the PLL.
3. Wait for PLL lock.
4. Program and enable the PFD with the desired configuration.
5. Clear the PFD clock gate to establish the desired reference clock frequency.
6. Program the CLKCTRL clock divider register (EMI or CPU) that uses the PLL/PFD as its reference clock.
7. Switch the bypass to *off* (select PLL, not crystal).

The requirement is that the roots of the clock are configured and stable before elements higher up in the tree are programmed. This will allow the roots to stabilize before selected as a valid source to drive a clock trunk/tree. If this sequence is not honored, unpredictable frequencies can occur which may violate the maximum operating frequency of components on the respective clock trees. Be sure to gate off the clock paths directly downstream from the PLL before powering off the PLL.

When clk_emi is operating in synchronous mode, the following requirements must be maintained:

- The clk_p divide value is less than or equal to the clk_emi divide value.
- The clk_emi divide value must be divisable by the clk_p divide value. An example of possible clk_p:clk_emi divide values would be, but not limited, to 1:1, 1:2, 1:3, 2:2, 2:4, 2:6, 3:3, 3:6, 3:9.

4.7 Chip Reset

Two PIO accessible soft reset bits exists to establish the initial state of the device. These bits are called HW_CLKCTRL_RESET_CHIP and HW_CLKCTRL_RESET_DIG. Set¬ting these bits will result in a chip wide reset cycle. When setting the DIG software reset bit, the digital logic is reset with the exception of the power module and the DCDC converter control logic. The CHIP software reset bit also initiates the full reset cycle and the power and DCDC con¬verter logic are also reset. These two soft reset bits are themselves reset during a soft reset sequence.

See Figure 4-4 for reference of the functionality of these two reset bits.

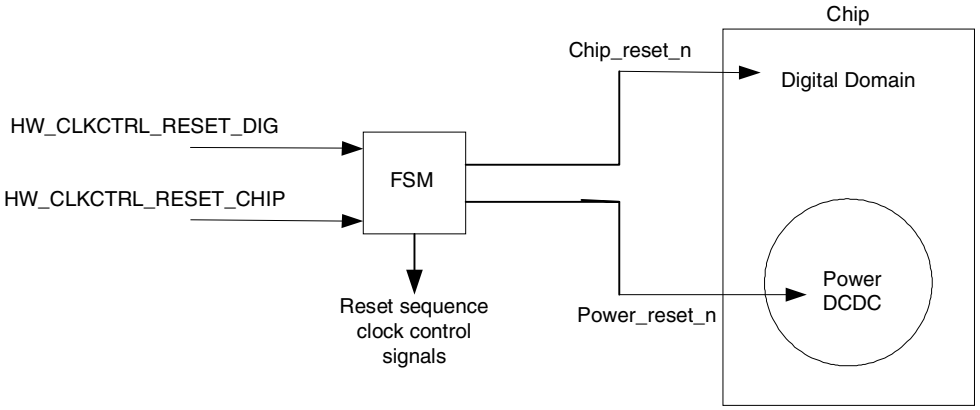


Figure 4-4. Reset Logic Functional Diagram

4.8 Programmable Registers

This section includes the programmable registers supported in the Clock Controller Module.

4.8.1 PLL Control Register 0 Description

The PLL Control Register 0 programs the 480 MHz PLL and the USB-clock enables.

HW_CLKCTRL_PLLCTRL0	0x000
HW_CLKCTRL_PLLCTRL0_SET	0x004
HW_CLKCTRL_PLLCTRL0_CLR	0x008
HW_CLKCTRL_PLLCTRL0_TOG	0x00C

Table 4-2. HW_CLKCTRL_PLLCTRL0

31 30	29 28	27 26	25 24	23 22	21 20	19	18	17	16	15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
RSRVD6	LFR_SEL	RSRVD5	CP_SEL	RSRVD4	DIV_SEL	RSRVD3	EN_USB_CLKS	RSRVD2	POWER	RSRVD1

Table 4-3. HW_CLKCTRL_PLLCTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD6**	RO	0x0	Always set to zero (0).
29:28	**LFR_SEL**	RW	0x0	TEST MODE FOR INTERNAL USE ONLY. Adjusts loop filter resistor. DEFAULT = 0x0 Default loop filter resistor TIMES_2 = 0x1 Doubles the loop filter resistor TIMES_05 = 0x2 Halves the loop filter resistor UNDEFINED = 0x3 Undefined

Table 4-3. HW_CLKCTRL_PLLCTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
27:26	**RSRVD5**	RO	0x0	Always set to zero (0).
25:24	**CP_SEL**	RW	0x0	TEST MODE FOR INTERNAL USE ONLY. Adjusts charge pump current DEFAULT = 0x0 Default charge pump current TIMES_2 = 0x1 Doubles charge pump current TIMES_05 = 0x2 Halves the charge pump current UNDEFINED = 0x3 Undefined
23:22	**RSRVD4**	RO	0x0	Always set to zero (0).
21:20	**DIV_SEL**	RW	0x0	TEST MODE FOR INTERNAL USE ONLY. This field is currently NOT supported. DEFAULT = 0x0 PLL frequency is 480 MHz LOWER = 0x1 Lower the PLL fequency from 480MHz to 384MHz LOWEST = 0x2 Lower the PLL fequency from 480MHz to 288MHz UNDEFINED = 0x3 Undefined
19	**RSRVD3**	RO	0x0	Always set to zero (0).
18	**EN_USB_CLKS**	RW	0x0	0: 8-phase PLL outputs for USB PHY are powered down. If set to 1, 8-phase PLL outputs for USB PHY are powered up. Additionally, the UTMICLK120_GATE and UTMICLK30_GATE must be deasserted in the UTMI phy to enable USB operation.
17	**RSRVD2**	RO	0x0	Always set to zero (0).
16	**POWER**	RW	0x0	PLL Power On (0 = PLL off; 1 = PLL On). Allow 10 us after turning the PLL on before using the PLL as a clock source. This is the time the PLL takes to lock to 480 MHz.
15:0	**RSRVD1**	RO	0x0	Always set to zero (0).

DESCRIPTION:

The PLL Control Register 0 programs the 480 MHz PLL and the USB-clock enables.

EXAMPLE:

```
HW_CLKCTRL_PLLCTRL0_WR(BF_CLKCTRL_PLLCTRL0_POWER(1)); // enable PLL
wait_10us; // Wait 10 us to let PLL lock before using it
```

4.8.2 PLL Control Register 1 Description

PLL Lock Control Register

HW_CLKCTRL_PLLCTRL1 0x010

Table 4-4. HW_CLKCTRL_PLLCTRL1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
LOCK	FORCE_LOCK	RSRVD1														LOCK_COUNT															

Table 4-5. HW_CLKCTRL_PLLCTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**LOCK**	RO	0x0	PLL Lock bit. 1=PLL Locked. 0=PLL Unlocked.
30	**FORCE_LOCK**	RW	0x0	Force the PLL Lock sequence to start. 1=Enable Force Lock. This bit is not self clearing.
29:16	**RSRVD1**	RO	0x0	Reserved - Always set to zero (0).
15:0	**LOCK_COUNT**	RO	0x0	Status of the PLL lock count. The PLL lock bit will assert when the count reaches 0x4B0. The lock count is driven off of xtal, so the 50us.

DESCRIPTION:

The lock count is driven off of xtal. So after the PLL is powered on, the PLL Lock should be asserted after 50us.

EXAMPLE:

```
HW_CLKCTRL_PLLCTRL1_WR(BF_CLKCTRL_PLLCTRL1_FORCE_LOCK(1)); // force pll lock sequence
HW_CLKCTRL_PLLCTRL1_WR(BF_CLKCTRL_PLLCTRL1_FORCE_LOCK(0)); // clear force pll lock
```

4.8.3 CPU Clock Control Register Description

The CPUCLK Clock Control Register provides controls for generating the ARM CPUCLK.

HW_CLKCTRL_CPU	0x020
HW_CLKCTRL_CPU_SET	0x024
HW_CLKCTRL_CPU_CLR	0x028
HW_CLKCTRL_CPU_TOG	0x02c

Table 4-6. HW_CLKCTRL_CPU

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD6		BUSY_REF_XTAL	BUSY_REF_CPU	RSVD5	DIV_XTAL_FRAC_EN	DIV_XTAL										RSVD4			INTERRUPT_WAIT	RSVD3	RSVD2	RSVD1				DIV_CPU					

Table 4-7. HW_CLKCTRL_CPU Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSVD6**	RO	0x0	Always set to zero (0).
29	**BUSY_REF_XTAL**	RO	0x0	This read-only bit field returns a one when the clock divider is busy transfering a new divider value across clock domains.
28	**BUSY_REF_CPU**	RO	0x0	This read-only bit field returns a one when the clock divider is busy transfering a new divider value across clock domains.
27	**RSVD5**	RO	0x0	Always set to zero (0).
26	**DIV_XTAL_FRAC_EN**	RW	0x0	1 = Enable fractional divide. 0 = Enable integer divide.

Table 4-7. HW_CLKCTRL_CPU Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
25:16	**DIV_XTAL**	RW	0x001	This field controls the divider connected to the crystal reference clock that drives the CLK_P domain when bypass is selected. NOTE: The divider is set to divide by 1 at power-on reset. Do NOT divide by 0.
15:13	**RSVD4**	RO	0x0	Always set to zero (0).
12	**INTERRUPT_WAIT**	RW	0x0	Gate off CLK_P while waiting for an interrupt.
11	**RSVD3**	RO	0x0	Always set to zero (0).
10	**RSVD2**	RW	0x0	Program this field to 0x0.
9:6	**RSVD1**	RO	0x0	Always set to zero (0).
5:0	**DIV_CPU**	RW	0x01	This field controls the divider connected to the ref_cpu reference clock that drives the CLK_P domain when bypass is NOT selected. For changes to this field to take effect, the ref_cpu reference clock must be running. NOTE: The divider is set to divide by 1 at power-on reset. Do NOT divide by 0.

DESCRIPTION:

Controls for the ARM 926 clock divider. Note: Do not write register space when busy bit(s) are high.

EXAMPLE:

```
HW_CLKCTRL_CPU_WR(BF_CLKCTRL_DIV_CPU(12)); // 480 MHz / 12 = 40 MHz
```

4.8.4 AHB, APBH Bus Clock Control Register Description

The AHB, APBH Bus Clock Control Register provides controls for CLK_H generation when HW_CLKCTRL_EMI_SYNC_MODE_EN = 0.

HW_CLKCTRL_HBUS	0x030
HW_CLKCTRL_HBUS_SET	0x034
HW_CLKCTRL_HBUS_CLR	0x038
HW_CLKCTRL_HBUS_TOG	0x03c

Table 4-8. HW_CLKCTRL_HBUS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD4		BUSY	DCP_AS_ENABLE	PXP_AS_ENABLE	APBHDMA_AS_ENABLE	APBXDMA_AS_ENABLE	TRAFFIC_JAM_AS_ENABLE	TRAFFIC_AS_ENABLE	CPU_DATA_AS_ENABLE	CPU_INSTR_AS_ENABLE	AUTO_SLOW_MODE	RSRVD2	SLOW_DIV			RSRVD1										DIV_FRAC_EN	DIV				

Table 4-9. HW_CLKCTRL_HBUS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD4**	RO	0x0	Reserved
29	**BUSY**	RO	0x0	This read-only bit field returns a one when the clock divider is busy transfering a new divider value across clock domains.
28	**DCP_AS_ENABLE**	RW	0x0	Enable auto-slow mode based on DCP activity. 0 = Run at the programmed CLK_H frequency.
27	**PXP_AS_ENABLE**	RW	0x0	Enable auto-slow mode based on PXP activity. 0 = Run at the programmed CLK_H frequency.
26	**APBHDMA_AS_ENABLE**	RW	0x0	Enable auto-slow mode based on APBH DMA activity. 0 = Run at the programmed CLK_H frequency.
25	**APBXDMA_AS_ENABLE**	RW	0x0	Enable auto-slow mode based on APBX DMA activity. 0 = Run at the programmed CLK_H frequency.
24	**TRAFFIC_JAM_AS_ENABLE**	RW	0x0	Enable auto-slow mode when less than three masters are trying to use the AHB. More than three active masters will engage the default mode. 0 = Run at the programmed CLK_H frequency.
23	**TRAFFIC_AS_ENABLE**	RW	0x0	Enable auto-slow mode based on AHB master activity. 0 = Run at the programmed CLK_H frequency.
22	**CPU_DATA_AS_ENABLE**	RW	0x0	Enable auto-slow mode based on with CPU Data access to AHB. 0 = Run at the programmed CLK_H frequency.
21	**CPU_INSTR_AS_ENABLE**	RW	0x0	Enable auto-slow mode based on with CPU Instruction access to AHB. 0 = Run at the programmed CLK_H frequency.
20	**AUTO_SLOW_MODE**	RW	0x0	Enable CLK_H auto-slow mode. When this is set, then CLK_H will run at the slow rate until one of the fast mode events has occurred. Note: The AUTO_SLOW_MODE bit must be cleared before writing to the SLOW_DIV bitfield.
19	**RSRVD2**	RO	0x0	Reserved
18:16	**SLOW_DIV**	RW	0x0	Slow mode divide ratio. Sets the ratio of CLK_H fast rate to the slow rate. Note: The AUTO_SLOW_MODE bit must be cleared before writing to the SLOW_DIV bitfield. BY1 = 0x0 Slow mode divide ratio = 1 BY2 = 0x1 Slow mode divide ratio = 2 BY4 = 0x2 Slow mode divide ratio = 4 BY8 = 0x3 Slow mode divide ratio = 8 BY16 = 0x4 Slow mode divide ratio = 16 BY32 = 0x5 Slow mode divide ratio = 32
15:6	**RSRVD1**	RO	0x0	Reserved
5	**DIV_FRAC_EN**	RW	0x0	1 = Enable fractional divide. 0 = Enable integer divide.
4:0	**DIV**	RW	0x01	CLK_P-to-CLK_H divide ratio. NOTE: The divider is set to divide by 1 at power-on reset. Do NOT divide by 0.

DESCRIPTION:

This register controls the clock divider that generates the CLK_H, the clock used by the AHB and APBH buses, when HW_CLKCTRL_EMI_SYNC_MODE_EN = 0.

Note: Do not write register space when busy bit(s) are high.

EXAMPLE:

```
HW_CLKCTRL_HBUS_WR(BF_CLKCTRL_HBUS_DIV(2)); // set CLK_H to half the ARM clock (CLK_P) frequency
```

4.8.5 APBX Clock Control Register Description

The APBX Clock Control Register provides control of the CLK_X clock divider.

HW_CLKCTRL_XBUS	0x040

Table 4-10. HW_CLKCTRL_XBUS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BUSY	RSVD2																				RSVD1	DIV									

Table 4-11. HW_CLKCTRL_XBUS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**BUSY**	RO	0x0	This read-only bit field returns a one when the clock divider is busy transfering a new divider value across clock domains.
30:11	**RSVD2**	RO	0x0	Always set to zero (0).
10	**RSVD1**	RW	0x0	Program this field to 0x0.
9:0	**DIV**	RW	0x001	This field controls the CLK_X divide ratio. CLK_X is sourced from the 24-MHz XTAL through this divider. Do NOT divide by 0.

DESCRIPTION:

This register controls the clock divider that generates the CLK_X, the clock used by the APBX bus.

Note: Do not write register space when busy bit(s) are high.

EXAMPLE:

```
HW_CLKCTRL_XBUS_WR(BF_CLKCTRL_XBUS_DIV(4)); // set apbx xbus clock to 1/4 the 24.0MHz crystal clock
frequency
```

4.8.6 XTAL Clock Control Register Description

The XTAL control register provides gating control for clocks sourced from the 24-MHz XTAL clock domain.

HW_CLKCTRL_XTAL	0x050
HW_CLKCTRL_XTAL_SET	0x054
HW_CLKCTRL_XTAL_CLR	0x058
HW_CLKCTRL_XTAL_TOG	0x05C

Table 4-12. HW_CLKCTRL_XTAL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
UART_CLK_GATE	FILT_CLK24M_GATE	PWM_CLK24M_GATE	DRI_CLK24M_GATE	DIGCTRL_CLK1M_GATE	TIMROT_CLK32K_GATE	RSRVD1																								DIV_UART	

Table 4-13. HW_CLKCTRL_XTAL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**UART_CLK_GATE**	RW	0x0	If set to 1, fixed 24-MHz clock for the UART, CLK_UART, is gated off.
30	**FILT_CLK24M_GATE**	RW	0x1	If set to 1, fixed 24-MHz clock for the Digital Filter, CLK_FILT24M, is gated off.
29	**PWM_CLK24M_GATE**	RW	0x1	If set to 1, fixed 24-MHz clock for the PWM, CLK_PWM24M, is gated off.
28	**DRI_CLK24M_GATE**	RW	0x1	If set to 1, fixed 24-MHz clock for the Digital Radio Interface (DRI), CLK_DRI24M, is gated off.
27	**DIGCTRL_CLK1M_GATE**	RW	0x0	If set to 1, fixed 1-MHz clock for DIGCTRL, CLK_1M, is gated off.
26	**TIMROT_CLK32K_GATE**	RW	0x0	If set to 1, fixed 32-kHz clock for the TIMROT block, CLK_32K, is gated off.
25:2	**RSRVD1**	RO	0x0	Always set to zero (0).
1:0	**DIV_UART**	RW	0x1	Reserved - Always set to one (1)

DESCRIPTION:

This register controls various fixed-rate divider clocks working off the 24.0-MHz crystal clock.

EXAMPLE:

```
HW_CLKCTRL_XTAL_WR(BF_CLKCTRL_XTAL_UART_CLK_GATE(0)|BF_CLKCTRL_XTAL_DRI_CLK24M_GATE(1));
```

4.8.7 PIX (LCDIF) Clock Control Register Description

The PIX control register provides control for LCDIF clock generation.

HW_CLKCTRL_PIX 0x060

Table 4-14. HW_CLKCTRL_PIX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CLKGATE	RSRVD2	BUSY	RSRVD1																DIV_FRAC_EN	DIV											

Table 4-15. HW_CLKCTRL_PIX Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**CLKGATE**	RW	0x1	CLK_PIX Gate. If set to 1, CLK_PIX is gated off. 0: CLK_PIX is not gated. When this bit is modified, or when it is high, the DIV field should not change its value. The DIV field can change ONLY when this clock gate bit field is low.
30	**RSRVD2**	RO	0x0	Always set to zero (0).
29	**BUSY**	RO	0x0	This read-only bit field returns a one when the clock divider is busy transfering a new divider value across clock domains.
28:13	**RSRVD1**	RO	0x0	Always set to zero (0).
12	**DIV_FRAC_EN**	RW	0x0	Reserved - Always set to zero (0).
11:0	**DIV**	RW	0x1	The Pixel clock frequency is determined by dividing the selected reference clock (ref_xtal or ref_pix) by the value in this bit field. This field can be programmed with a new value only when CLKGATE = 0. NOTE: The divider is set to divide by 1 at power-on reset. Do NOT divide by 0. Do not divide by more than 255.

DESCRIPTION:

This register controls the divider that generates the PIX (LCDIF) clock.

Note: Do not write register space when busy bit(s) are high.

EXAMPLE:

```
HW_CLKCTRL_PIX_WR(BF_CLKCTRL_PIX_DIV(40));
```

4.8.8 Synchronous Serial Port Clock Control Register Description

The SSP control register provides control for SSP clock generation.

HW_CLKCTRL_SSP 0x070

Table 4-16. HW_CLKCTRL_SSP

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CLKGATE	RSVD3	BUSY	RSVD2																			RSVD1	DIV								

Table 4-17. HW_CLKCTRL_SSP Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**CLKGATE**	RW	0x1	CLK_SSP Gate. If set to 1, CLK_SSP is gated off. 0: CLK_SSP is not gated. When this bit is modified, or when it is high, the DIV field should not change its value. The DIV field can change ONLY when this clock gate bit field is low.
30	**RSVD3**	RO	0x0	Always set to zero (0).
29	**BUSY**	RO	0x0	This read-only bit field returns a one when the clock divider is busy transfering a new divider value across clock domains.
28:10	**RSVD2**	RO	0x0	Always set to zero (0).
9	**RSVD1**	RW	0x0	Program this field to 0x0.
8:0	**DIV**	RW	0x1	The synchronous serial port clock frequency is determined by dividing the selected reference clock (ref_xtal or ref_io) by the value in this bit field. This field can be programmed with a new value only when CLKGATE = 0. NOTE: The divider is set to divide by 1 at power-on reset. Do NOT divide by 0.

DESCRIPTION:

This register controls the clock divider that generates the clock for the synchronous serial port (SSP), CLK_SSP.

Note: Do not write register space when busy bit(s) are high.

EXAMPLE:

```
HW_CLKCTRL_SSP_WR(BF_CLKCTRL_SSP_DIV(40));
```

4.8.9 General-Purpose Media Interface Clock Control Register Description

The GPMI control register provides control for GPMI clock generation.

HW_CLKCTRL_GPMI 0x080

Table 4-18. HW_CLKCTRL_GPMI

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CLKGATE	RSVD3	BUSY	RSVD2																		RSVD1	DIV									

Table 4-19. HW_CLKCTRL_GPMI Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**CLKGATE**	RW	0x1	CLK_GPMI Gate. If set to 1, CLK_GPMI is gated off. 0: CLK_GPMI is not gated. When this bit is modified, or when it is high, the DIV field should not change its value. The DIV field can change ONLY when this clock gate bit field is low.
30	**RSVD3**	RO	0x0	Always set to zero (0).
29	**BUSY**	RO	0x0	This read-only bit field returns a one when the clock divider is busy transfering a new divider value across clock domains.
28:11	**RSVD2**	RO	0x0	Always set to zero (0).
10	**RSVD1**	RW	0x0	Program this field to 0x0.
9:0	**DIV**	RW	0x1	The GPMI clock frequency is determined by dividing the selected reference clock (ref_xtal or ref_io) by the value in this bit field. This field can be programmed with a new value only when CLKGATE = 0. NOTE: The divider is set to divide by 1 at power-on reset. Do NOT divide by 0.

DESCRIPTION:

This register controls the divider that generates the General-Purpose Media Interface (GPMI) clock, CLK_GPMI.

Note: Do not write register space when busy bit(s) are high.

EXAMPLE:

```
HW_CLKCTRL_GPMI_WR(BF_CLKCTRL_GPMI_DIV(40));
```

4.8.10 SPDIF Clock Control Register Description

The SPDIF control register provides control for SPDIF clock generation.

HW_CLKCTRL_SPDIF 0x090

Table 4-20. HW_CLKCTRL_SPDIF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CLKGATE	RSRVD																														

Table 4-21. HW_CLKCTRL_SPDIF Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**CLKGATE**	RW	0x1	CLK_PCMSPDIF Gate. If set to 1, CLK_PCMSPDIF is gated off. 0: CLK_PCMSPDIF is not gated. When this bit is modified, or when it is high, the SPDIF rate change field should not change its value. The SPDIF rate change field can change ONLY when this clock gate bit field is low.
30:0	**RSRVD**	RO	0x0	Always set to zero (0).

DESCRIPTION:

This register controls the clock gate on the SPDIF clock, CLK_PCMSPDIF.

EXAMPLE:

```
HW_CLKCTRL_SPDIF_WR(BF_CLKCTRL_SPDIF_CLKGATE(1));
```

4.8.11 EMI Clock Control Register Description

The EMi control register provides control for External Memory Interface clock generation.

HW_CLKCTRL_EMI 0x0a0

Table 4-22. HW_CLKCTRL_EMI

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CLKGATE	SYNC_MODE_EN	BUSY_REF_XTAL	BUSY_REF_EMI	BUSY_REF_CPU	BUSY_SYNC_MODE	RSVD5								RSVD4	RSVD3	RSVD2					DIV_XTAL				RSVD1				DIV_EMI			

Table 4-23. HW_CLKCTRL_EMI Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**CLKGATE**	RW	0x1	CLK_EMI crystal divider Gate. If set to 1, the EMI_CLK divider that is sourced by the crystal reference clock, ref_xtal, is gated off. 0: CLK_EMI crystal divider is not gated
30	**SYNC_MODE_EN**	RW	0x0	If set to 1, EMI_CLK is synchronous with the APBH clock. If set to 0, EMI_CLK is aysnchronous. In synchronous operation, the EMI clock dividers control both EMI_CLK and APBH clock. Both xtal and ref_cpu must be active to switch between asynchronous/synchronous operation.
29	**BUSY_REF_XTAL**	RO	0x0	This read-only bit field returns a one when the clock divider is busy transfering a new divider value across clock domains. This bit is valid when HW_CLKCTRL_EMI_SYNC_MODE_EN = 0.
28	**BUSY_REF_EMI**	RO	0x0	This read-only bit field returns a one when the clock divider is busy transfering a new divider value across clock domains.
27	**BUSY_REF_CPU**	RO	0x0	This read-only bit field returns a one when the clock divider is busy transfering a new divider value across clock domains. This bit is valid when HW_CLKCTRL_EMI_SYNC_MODE_EN = 1.
26	**BUSY_SYNC_MODE**	RO	0x0	This read-only bit field returns a one when there is a change in HW_CLKCTRL_EMI_SYNCE_MODE_EN or when there is a change in HW_CLKCTRL_CLKSEQ_BYPASS_CPU and HW_CLKCTRL_EMI_SYNCE_MODE_EN is set. When this bit returns a one, do not change the CPU or EMI divider values.
25:18	**RSVD5**	RO	0x0	Always set to zero (0).
17	**RSVD4**	RO	0x0	Program this field to 0x0.
16	**RSVD3**	RW	0x0	Program this field to 0x0.
15:12	**RSVD2**	RO	0x0	Always set to zero (0).
11:8	**DIV_XTAL**	RW	0x1	This field controls the divider connected to the crystal reference clock, ref_xtal, that drives the CLK_EMI domain when bypass IS selected. NOTE: The divider is set to divide by 1 at power-on reset. Do NOT divide by 0.
7:6	**RSVD1**	RO	0x0	Always set to zero (0).
5:0	**DIV_EMI**	RW	0x1	This field controls the divider connected to the ref_emi reference clock that drives the CLK_EMI domain when bypass IS NOT selected. For changes to this field to take effect, the ref_emi reference clock must be running. NOTE: The divider is set to divide by 1 at power-on reset. Do NOT divide by 0.

DESCRIPTION:

This register controls the clock dividers that generate the External Memory Interface (EMI) clock.

Note: Do not write register space when busy bit(s) are high.

EXAMPLE:

```
HW_CLKCTRL_EMI_WR(BF_CLKCTRL_EMI_DIV_XTAL(1));
```

4.8.12 SAIF Clock Control Register Description

The SAIF control register provides control for SAIF clock generation.

HW_CLKCTRL_SAIF 0x0c0

Table 4-24. HW_CLKCTRL_SAIF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CLKGATE	RSRVD2	BUSY	RSRVD1												DIV_FRAC_EN	DIV															

Table 4-25. HW_CLKCTRL_SAIF Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**CLKGATE**	RW	0x1	CLK_SAIF Gate. If set to 1, CLK_SAIF is gated off. 0: CLK_SAIF is not gated. When this bit is modified, or when it is high, the DIV field should not change its value. The DIV field can change ONLY when this clock gate bit field is low.
30	**RSRVD2**	RO	0x0	Always set to zero (0).
29	**BUSY**	RO	0x0	This read-only bit field returns a one when the clock divider is busy transfering a new divider value across clock domains.
28:17	**RSRVD1**	RO	0x0	Always set to zero (0).
16	**DIV_FRAC_EN**	RW	0x0	Reserved - Always set to one (1) - Notice this is not the reset value.
15:0	**DIV**	RW	0x1	The SAIF clock frequency is determined by dividing the selected reference clock (ref_xtal or ref_pll) by the value in this bit field. This field can be programmed with a new value only when CLKGATE = 0. NOTE: The divider is set to divide by 1 at power-on reset. Do NOT divide by 0.

DESCRIPTION:

This register controls the divider that generates the Serial Audio Interface (SAIF) clock.

Note: Do not write register space when busy bit(s) are high.

EXAMPLE:

```
HW_CLKCTRL_SAIF_WR(BF_CLKCTRL_SAIF_DIV(40));
```

4.8.13 TV Encode Clock Control Register Description

The TV control register provides control for TV Encoder clock generation.

HW_CLKCTRL_TV 0x0d0

Table 4-26. HW_CLKCTRL_TV

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CLK_TV108M_GATE	CLK_TV_GATE	RSRVD																													

Table 4-27. HW_CLKCTRL_TV Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**CLK_TV108M_GATE**	RW	0x1	If set to 1, fixed 108-MHz clock for the TV Component Video is gated off. 0: CLK_TV108M, is not gated.
30	**CLK_TV_GATE**	RW	0x1	If set to 1, fixed 54-MHz and 27-MHz clocks for the TV Encoder are gated off. 0: CLK_TV54M and CLK_TV27M, are not gated.
29:0	**RSRVD**	RO	0x0	Always set to zero (0).

DESCRIPTION:

This register controls various video divider clocks.

EXAMPLE:

```
HW_CLKCTRL_CLK_TV108M_GATE_WR(BF_CLKCTRL_CLK_TV108M_GATE(0));
```

4.8.14 ETM Clock Control Register Description

The ETM control register provides control for ETM clock generation.

HW_CLKCTRL_ETM 0x0e0

Table 4-28. HW_CLKCTRL_ETM

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CLKGATE	RSRVD2	BUSY	RSRVD1																						DIV_FRAC_EN	DIV					

Table 4-29. HW_CLKCTRL_ETM Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**CLKGATE**	RW	0x1	CLK_ETM Gate. If set to 1, CLK_ETM is gated off. 0: CLK_ETM is not gated. When this bit is modified, or when it is high, the DIV field should not change its value. The DIV field can change ONLY when this clock gate bit field is low.
30	**RSRVD2**	RO	0x0	Always set to zero (0).
29	**BUSY**	RO	0x0	This read-only bit field returns a one when the clock divider is busy transfering a new divider value across clock domains.
28:7	**RSRVD1**	RO	0x00000	Always set to zero (0).
6	**DIV_FRAC_EN**	RW	0x0	Reserved - Always set to zero (0).
5:0	**DIV**	RW	0x01	The Pixel clock frequency is determined by dividing the selected reference clock (ref_xtal or ref_cpu) by the value in this bit field. This field can be programmed with a new value only when CLKGATE = 0. NOTE: The divider is set to divide by 1 at power-on reset. Do NOT divide by 0.

DESCRIPTION:

This register controls the divider that generates the ETM clock.

Note: Do not write register space when busy bit(s) are high.

EXAMPLE:

```
HW_CLKCTRL_ETM_WR(BF_CLKCTRL_ETM_DIV(4));
```

4.8.15 Fractional Clock Control Register Description

The FRAC control register provides control for PFD clock generation. NOTE: Only byte accesses are supported. When using DWORD accesses to this regiester, the PFD update sequence will commence for all four PFDs controlled by this register. This may be undersirable if only one of the PFD divide values need to be updated. Only access individual bytes within this register in a single PIO access.

HW_CLKCTRL_FRAC	0x0f0
HW_CLKCTRL_FRAC_SET	0x0f4
HW_CLKCTRL_FRAC_CLR	0x0f8
HW_CLKCTRL_FRAC_TOG	0x0fC

Table 4-30. HW_CLKCTRL_FRAC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CLKGATEIO	IO_STABLE	IOFRAC						CLKGATEPIX	PIX_STABLE	PIXFRAC						CLKGATEEMI	EMI_STABLE	EMIFRAC						CLKGATECPU	CPU_STABLE	CPUFRAC					

Table 4-31. HW_CLKCTRL_FRAC Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**CLKGATEIO**	RW	0x1	IO Clock Gate. If set to 1, the IO fractional divider clock (reference PLL ref_io) is off (power savings). 0: IO fractional divider clock is enabled.
30	**IO_STABLE**	RO	0x0	This read-only bitfield is for DIAGNOSTIC PURPOSES ONLY since the fractional divider should become stable quickly enough that this field will never need to be used by either device driver or application code. The value inverts when the new programmed fractional divide value has taken effect. Read this bit, program the new value, and when this bit inverts, the phase divider clock output is stable. Note that the value will not invert when the fractional divider is taken out of or placed into clock-gated state.
29:24	**IOFRAC**	RW	0x12	This field controls the IO clocks fractional divider. The resulting frequency shall be 480 * (18/IOFRAC) where IOFRAC = 1-35.
23	**CLKGATEPIX**	RW	0x1	PIX Clock Gate. If set to 1, the PIX fractional divider clock (reference PLL ref_pix) is off (power savings). 0: PIX fractional divider clock is enabled.
22	**PIX_STABLE**	RO	0x0	This read-only bitfield is for DIAGNOSTIC PURPOSES ONLY since the fractional divider should become stable quickly enough that this field will never need to be used by either device driver or application code. The value inverts when the new programmed fractional divide value has taken effect. Read this bit, program the new value, and when this bit inverts, the phase divider clock output is stable. Note that the value will not invert when the fractional divider is taken out of or placed into clock-gated state.
21:16	**PIXFRAC**	RW	0x12	This field controls the pixel clock fractional divider. The resulting frequency shall be 480 * (18/PIXFRAC) where PIXFRAC = 1-35.
15	**CLKGATEEMI**	RW	0x1	EMI Clock Gate. If set to 1, the EMI fractional divider clock (reference PLL ref_emi) is off (power savings). 0: EMI fractional divider clock is enabled.
14	**EMI_STABLE**	RO	0x0	This read-only bitfield is for DIAGNOSTIC PURPOSES ONLY since the fractional divide should become stable quickly enough that this field will never need to be used by either device driver or application code. This value inverts when the new programmed fractional divide value has taken effect. Read this bit, program the new value, and when this bit inverts, the phase divider clock output is stable. Note that the value will not invert when the fractional divider is taken out of or placed into clock-gated state.
13:8	**EMIFRAC**	RW	0x12	This field controls the EMI clock fractional divider. The resulting frequency shall be 480 * (18/EMIFRAC) where EMIFRAC = 1-35.
7	**CLKGATECPU**	RW	0x1	CPU Clock Gate. If set to 1, the CPU fractional divider clock (reference PLL ref_cpu) is off (power savings). 0: CPU fractional divider clock is enabled.

Table 4-31. HW_CLKCTRL_FRAC Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
6	**CPU_STABLE**	RO	0x0	This read-only bitfield is for DIAGNOSTIC PURPOSES ONLY since the fractional divide should become stable quickly enough that this field will never need to be used by either device driver or application code. The value inverts when the new programmed fractional divide value has taken effect. Read this bit, program the new value, and when this bit inverts, the phase divider clock output is stable. Note that the value will not invert when the fractional divider is taken out of or placed into clock-gated state.
5:0	**CPUFRAC**	RW	0x12	This field controls the CPU clock fractional divider. The resulting frequency shall be 480 * (18/CPUFRAC) where CPUFRAC = 1-35.

DESCRIPTION:

This register controls the 9-phase fractional clock dividers. The fractional clock frequencies are a product of the values in these registers.

EXAMPLE:

```
HW_CLKCTRL_FRAC_WR(BF_CLKCTRL_FRAC_CPUFRAC(4));
```

4.8.16 Fractional Clock Control Register 1 Description

The FRAC1 control register provides control for PFD clock generation.

HW_CLKCTRL_FRAC1	0x100
HW_CLKCTRL_FRAC1_SET	0x104
HW_CLKCTRL_FRAC1_CLR	0x108
HW_CLKCTRL_FRAC1_TOG	0x10C

Table 4-32. HW_CLKCTRL_FRAC1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CLKGATEVID	VID_STABLE	RSRVD1																													

Table 4-33. HW_CLKCTRL_FRAC1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**CLKGATEVID**	RW	0x1	432 MHz PLL Clock Gate. If set to 1, the 432 MHz fractional divider clock (reference PLL ref_vid) is off (power savings). 0: IO fractional divider clock is enabled.
30	**VID_STABLE**	RO	0x0	This read-only bitfield is for DIAGNOSTIC PURPOSES ONLY since the fractional divider should become stable quickly enough that this field will never need to be used by either device driver or application code. The value inverts when the new programmed fractional divide value has taken effect. Read this bit, program the new value, and when this bit inverts, the phase divider clock output is stable. Note that the value will not invert when the fractional divider is taken out of or placed into clock-gated state.
29:0	**RSRVD1**	RO	0x0	Always set to zero (0).

DESCRIPTION:

This register controls the 9-phase fractional clock dividers. The fractional clock frequencies are a product of the values in these registers.

EXAMPLE:

```
HW_CLKCTRL_FRAC1_WR(BF_CLKCTRL_FRAC1_CLKGATEVID(0));
```

4.8.17 Clock Frequency Sequence Control Register Description

The CLKSEQ control register provides control for switching between XTAL and PLL clock generation.

HW_CLKCTRL_CLKSEQ	0x110
HW_CLKCTRL_CLKSEQ_SET	0x114
HW_CLKCTRL_CLKSEQ_CLR	0x118
HW_CLKCTRL_CLKSEQ_TOG	0x11c

Table 4-34. HW_CLKCTRL_CLKSEQ

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																							BYPASS_ETM	BYPASS_CPU	BYPASS_EMI	BYPASS_SSP	BYPASS_GPMI	BYPASS_IR	RSRVD0	BYPASS_PIX	BYPASS_SAIF

Table 4-35. HW_CLKCTRL_CLKSEQ Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:9	**RSRVD1**	RO	0x0	Always set to zero (0).
8	**BYPASS_ETM**	RW	0x1	ETM bypass select. 1 = Select ref_xtal path to generate the ETM clock domain. 0 = Select ref_cpu path to generate the ETM clock domain. PLL and 9-phase fractional divider must be configured when this bit is cleared.
7	**BYPASS_CPU**	RW	0x1	CPU bypass select. 1 = Select ref_xtal path to generate the CPU and APBH clock domains. 0 = Select ref_cpu path to generate the CPU and APBH clock domains. PLL and 9-phase fractional divider must be configured when this bit is cleared.
6	**BYPASS_EMI**	RW	0x1	EMI bypass select. 1 = Select ref_xtal path to generate the EMI clock domain. 0 = Select ref_emi path to generate the EMI clock domain. PLL and 9-phase fractional divider must be configured when this bit is cleared.
5	**BYPASS_SSP**	RW	0x1	SSP bypass select. 1 = Select ref_xtal path to generate the SSP clock domain. 0 = Select ref_io path to generate the SSP clock domain. PLL and 9-phase fractional divider must be configured when this bit is cleared.
4	**BYPASS_GPMI**	RW	0x1	GPMI bypass select. 1 = Select ref_xtal path to generate the GPMI clock domain. 0 = Select ref_io path to generate the GPMI clock domain. PLL and 9-phase fractional divider must be configured when this bit is cleared.
3	**BYPASS_IR**	RW	0x1	IR bypass select. 1 = Select ref_xtal path to generate the IR clock domain. 0 = Select ref_io path to generate the IR clock domain. PLL and 9-phase fractional divider must be configured when this bit is cleared.
2	**RSRVD0**	RO	0x0	Always set to zero (0).
1	**BYPASS_PIX**	RW	0x1	PIX bypass select. 1 = Select ref_xtal path to generate the PIX clock domain. 0 = Select ref_pix path to generate the PIX clock domain. PLL and 9-phase fractional divider must be configured when this bit is cleared.
0	**BYPASS_SAIF**	RW	0x1	Reserved - Always set to zero (0) - Notice this is not the reset value.

DESCRIPTION:

This register controls the selection of clock sources (ref_xtal or ref_*) for various clock dividers.

EXAMPLE:

```
HW_CLKCTRL_CLKSEQ_WR(BF_CLKCTRL_CLKSEQ_BYPASS_IR(1));
```

4.8.18 System Software Reset Register Description

The RESET control register provides control for soft reset.

HW_CLKCTRL_RESET 0x120

Table 4-36. HW_CLKCTRL_RESET

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD																														CHIP	DIG

Table 4-37. HW_CLKCTRL_RESET Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:2	**RSRVD**	RO	0x0	Always set to zero (0).
1	**CHIP**	RW	0x0	Setting this bit to a logic one will reset the ENTIRE chip, no exceptions. This bit will also be reset after the full chip reset cycle completes.
0	**DIG**	RW	0x0	Setting this bit to a logic one will reset the digital sections of the chip. The DCDC and power module will not be reset. This bit will also be reset after the reset cycle completes.

DESCRIPTION:

This register controls full chip reset generation.

EXAMPLE:

```
HW_CLKCTRL_RESET_WR(BF_CLKCTRL_RESET_ALL(1));
```

4.8.19 ClkCtrl Status Description

The STATUS control register provides read only status of the CPU frequecy limits.

HW_CLKCTRL_STATUS 0x130

Table 4-38. HW_CLKCTRL_STATUS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CPU_LIMIT	RSRVD																														

Table 4-39. HW_CLKCTRL_STATUS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**CPU_LIMIT**	RO	0x00	CPU Limiting. 00: full cpu frequency, 01: limit cpu frequency to 411.43 MHz, 10: limit cpu frequency to 360 MHz, 11: limit cpu frequency to 320 MHz
29:0	**RSRVD**	RO	0x0	Always set to zero (0).

DESCRIPTION:

This register indicates the CPU Frequency limit.

EXAMPLE:

```
HW_CLKCTRL_STATUS_RD(BF_CLKCTRL_STATUS_CPU_LIMIT());
```

4.8.20 ClkCtrl Version Description

The VERSION control register is a read only status of the clkctrl block version.

HW_CLKCTRL_VERSION 0x140

Table 4-40. HW_CLKCTRL_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 4-41. HW_CLKCTRL_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x4	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x0	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
HW_CLKCTRL_VERSION_RD(BF_CLKCTRL_VERSION_MAJOR());
```

CLKCTRL Block v4.0, Revision 1.48

Chapter 5
Interrupt Collector

This chapter describes the interrupt control features of the i.MX23 and includes sections on interrupt nesting, FIQ generation, and CPU wait-for-interrupt mode. Table 5-1 lists all of the interrupt sources available on the i.MX23. Programmable registers for interrupt generation and control are described in Section 5.4, “Programmable Registers.”

5.1 Overview

The ARM9 CPU core has two interrupt input lines, IRQ and FIQ. As shown in Figure 5-1, the Interrupt Collector (ICOLL) can steer any of 128 interrupt sources to either the the FIQn or IRQn lines of the ARM9 CPU.

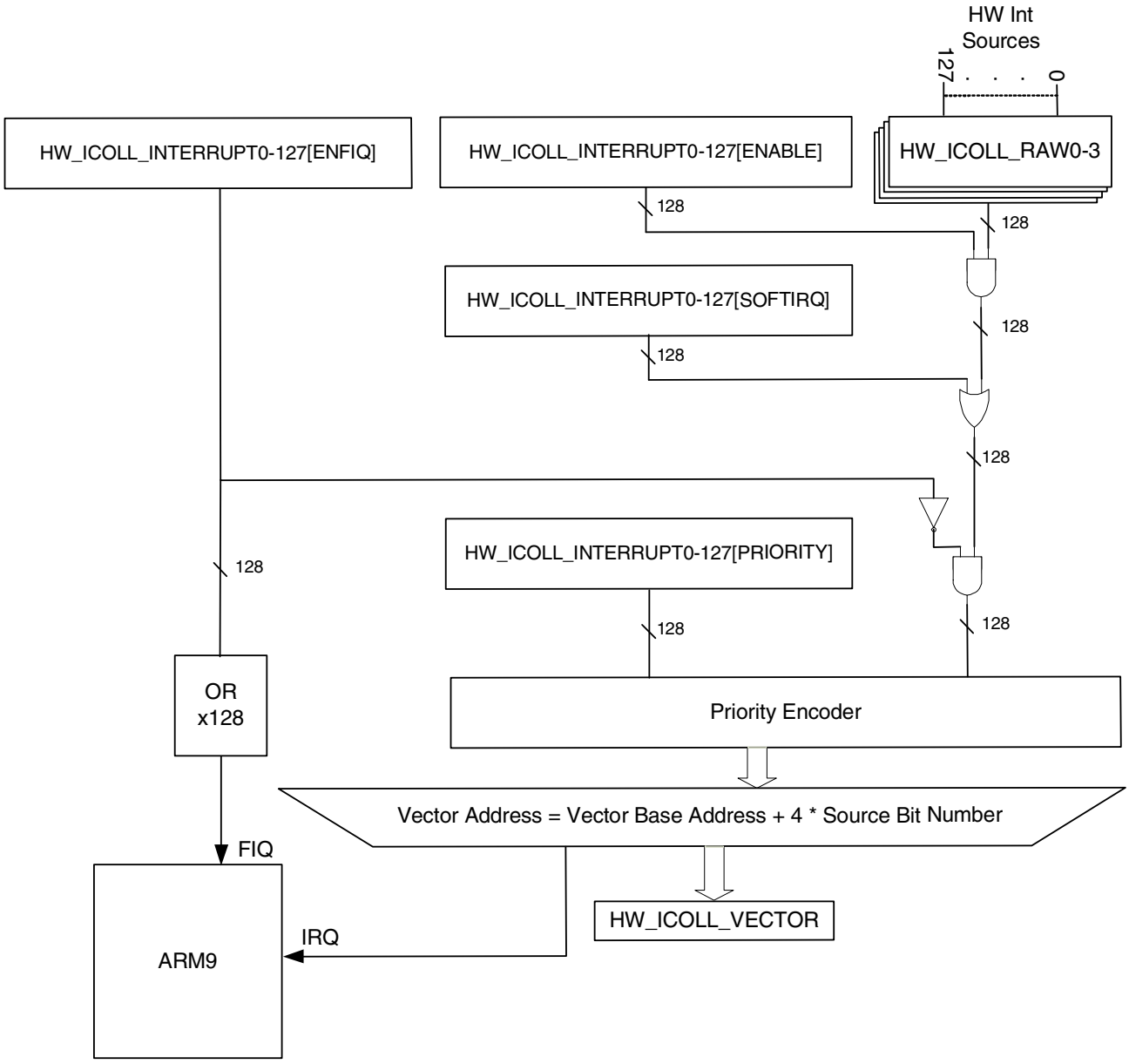


Figure 5-1. Interrupt Collector System Diagram

5.2 Operation

Within an individual interrupt request line (IRQ only), the ICOLL offers four-level priority (above base level) for each of its interrupt sources. Preemption of a lower priority interrupt by a higher priority is supported (interrupt nesting). Interrupts assigned to the same level are serviced in a strict linear priority order within level from lowest to highest interrupt source bit number. FIQ interrupts are not prioritized, nor are they vectorized. All interrupt lines can be configured as a FIQ. If more than one is routed to the FIQ, then they must be discriminated by software. It is highly recommended to reserve FIQ assignment to time critical events such as voltage brownouts or timers.

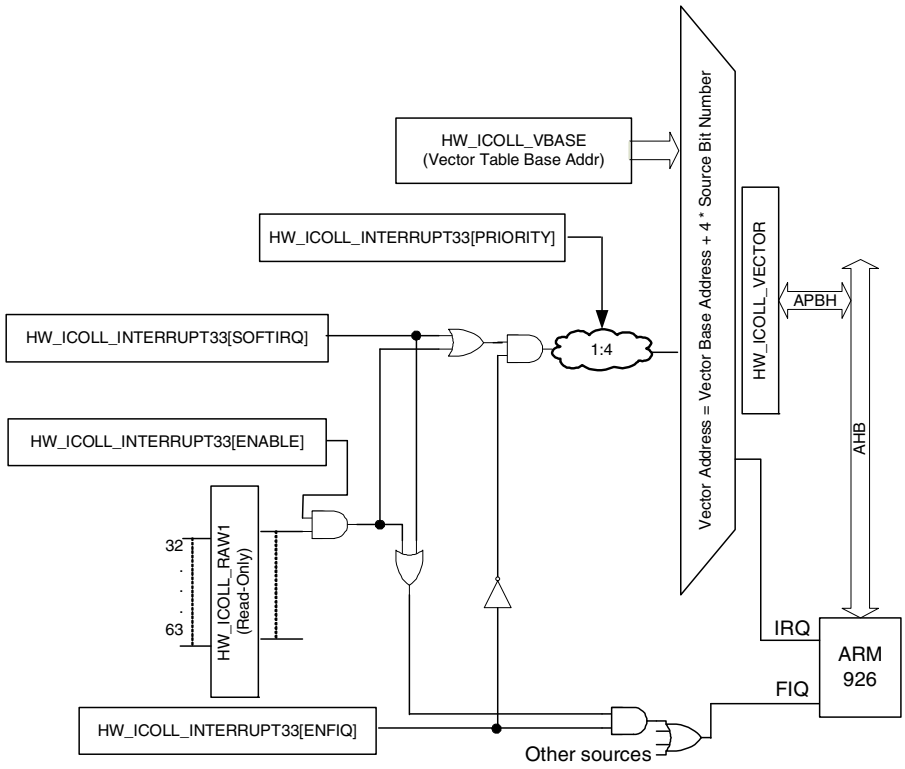


Figure 5-2. Interrupt Collector IRQ/FIQ Logic for Source 33

For a single interrupt source bit, there is an enable bit that gates it to the priority logic (HW_ICOLL_INTERRUPTn[ENABLE]). A software interrupt bit per source bit can be used to force an interrupt at the appropriate priority level directed to the corresponding vector address. Each source can be applied to one of four interrupt levels.

The enable bit, FIQ-enable, the software interrupt bit, and the two-bit priority level specification for each interrupt source bit are contained with a single programmable register for each interrupt. The path from any interrupt source to the FIQ or IRQ logic is shown in Figure 5-2 using HW_ICOLL_INTERRUPT33 as an example.

The data path for generating the vector address (readable by software) for the IRQ generation portion of the interrupt collector is implemented as a multicycle path, as shown in Figure 5-3. The interrupt sources are continuously sampled in the holding register until one or more arrive. The FSM causes the holding register to stop sampling while a vector address is computed. Each interrupt source bit is applied to one of four levels based on the two-bit priority specification of each source bit. When the holding register "closes," there can be more than one newly arrived source bit. Thus, the source bits could be assigned such that more than one interrupt level is requesting an interrupt. The pipeline first determines the highest level requesting interrupt service. All interrupt requests on that level are presented to the linear priority encoder. The result of this stage is a six-bit number corresponding to the source bit number of the highest priority requesting an interrupt. This six-bit source number is used to compute the vector address as follows:

VectorAddress = VectorBase + (Pitch * SourceBitNumber)

Pitch = 4,8, 12,16,20,24, or 28 as desired, see HW_CTRL_VECTOR_PITCH.

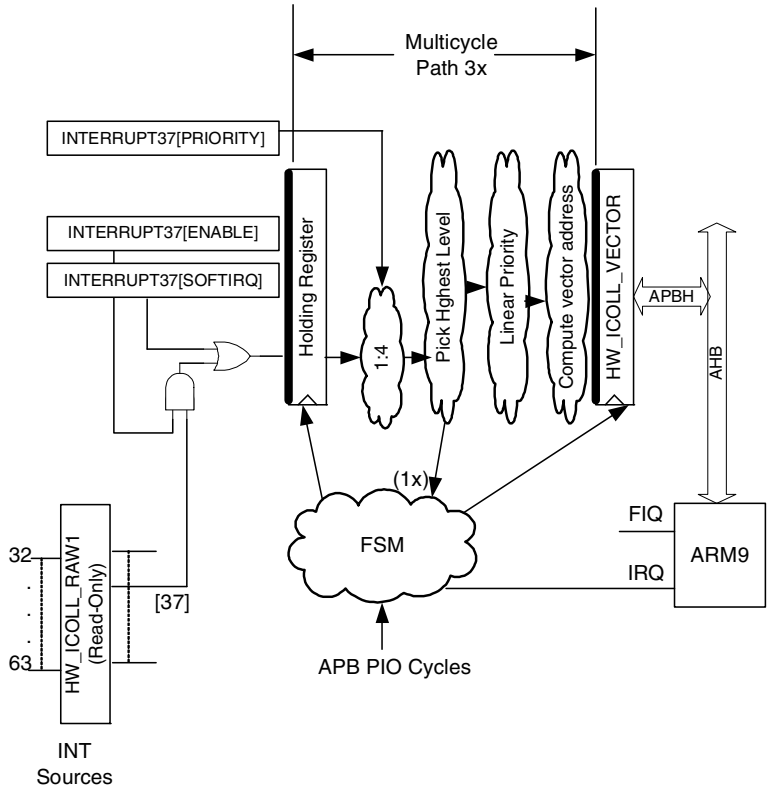


Figure 5-3. IRQ Control Flow

5.2.1 Nesting of Multi-Level IRQ Interrupts

There are a number of very important interactions between the interrupt collector's FSM and the interrupt service routine (ISR) running on the CPU. See Figure 5-4 for the following discussion.

As soon as the interrupt source is recognized in the holding register, the FSM delays two clocks, then grabs the vector address and asserts IRQ to the CPU. As soon as possible after the CPU enters the interrupt service routine, it must notify the interrupt collector. Software indicates the in-service state by writing to the HW_ICOLL_VECTOR register. The contents of the data bus on this write do not matter. Optionally, firmware can enable the ARM read side-effect mode. In this case, the in-service state is indicated as a side effect of having read the HW_ICOLL_VECTOR register at the exception vector (0xFFFF0018). At this point, the FSM reopens the holding register and scans for new interrupt sources. Any such IRQ sources are presented to the CPU, provided that they are at a level higher than any currently in-service level.

Whenever the ARM CPU takes an IRQ exception, it turns off the IRQ enable in the CPU status register (CSR), as shown in Figure 5-4. If a higher priority interrupt is pending at this point, then another IRQ exception is taken.

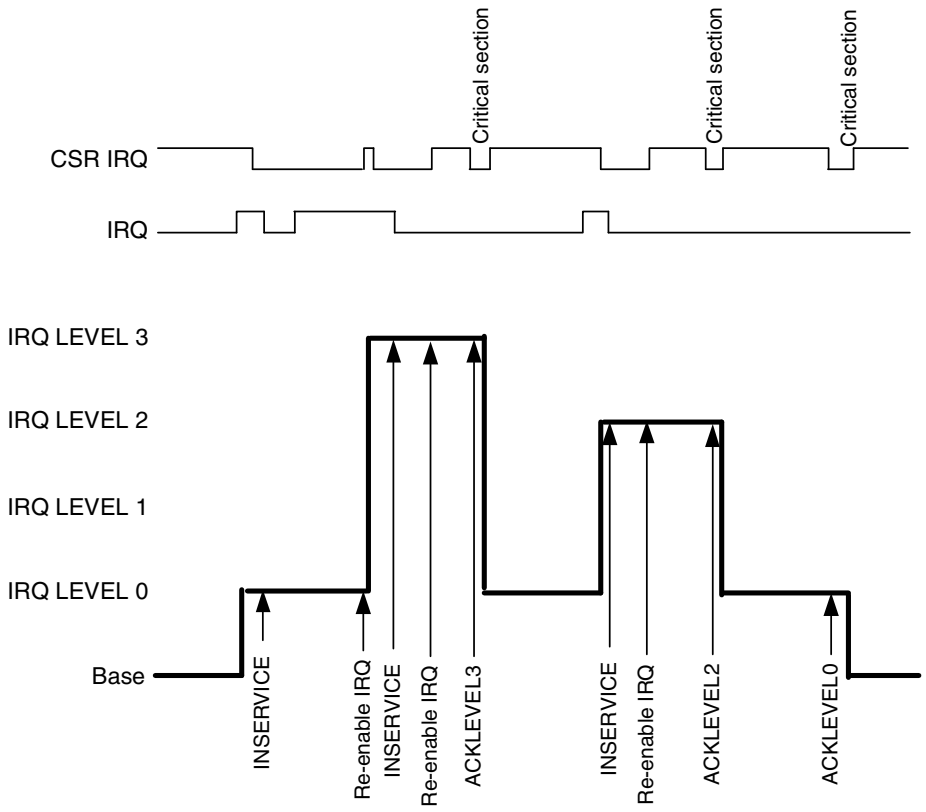


Figure 5-4. Nesting of Multi-Level IRQ Interrupts

The example in Figure 5-4 shows going from the base to a level 0 ISR. When the ISR at level 0 was ready, it enabled IRQ interrupts. At this point, it nests IRQ interrupts up to a level 3 interrupt. The level 3 ISR marks its in-service state, which causes the interrupt collector to open the holding register to search for new interrupt sources. In this example, none comes in, so the level 3 ISR completes. As part of the return process, the ISR disables IRQ interrupts, then acknowledges the level 3 service state. This is accomplished by writing the level number (3 in this case) to the interrupt collector's Level Acknowledge register. The interrupt collector resets the in-service bit for level 3. If this enables an IRQ at level 3, then it asserts IRQ and goes through the nesting process again. Since IRQ exceptions are masked in the level 3 ISR, this nesting does not take place until the level 3 ISR returns from interrupt. This return automatically re-enables IRQ exceptions. At this point, another exception could occur.

Figure 5-4 shows a second nesting of the IRQ interrupt by the arrival of a level 2 interrupt source bit. Finally, the figure shows the point at which the level 0 ISR enters its critical section (masks IRQ) and acknowledges level 0 to the interrupt collector and returns from interrupt.

The FSM reverts to its "BASE" level state waiting for an interrupt request to arrive in the holding register. The waveform for the IRQ mask in the CPU status register (CSR) and the waveform for the IRQ input to the CPU as they relate to the interrupt collector action are shown in Figure 5-4.

WARNING: There is an inherent race condition between notifying the interrupt collector that an ISR has been entered and having that ISR re-enable IRQ exceptions in the CSR. The in-service notification can take a number of cycles to percolate through the write buffer, through the AHB and APB bridge and into the interrupt collector where it removes the IRQ assertion to the CPU. This ICOLL IRQ must be deasserted before the CSR IRQ on the CPU is re-enabled or the CPU will see a phantom interrupt. This is why the ARM vectored interrupt controller provides this in service notification as a read side effect of the vector address read. Alternatively, the ISR can read the interrupt collector's CSR. The value received is unimportant, but the time required to do the read ensures that the write data has arrived at the interrupt collector. If firmware uses this method, it should allow clocks after the read for the FSM and for the CPU to recognize that the IRQ has been deasserted.

5.2.2 FIQ Generation

On i.MX23, all interrupt sources can be configured as FIQ. This is controlled via the HW_ICOLL_INTERRUPTn[ENFIQ] register bit as shown in Figure 5-2. When enabled to the FIQ, the software interrupt associated with these bits can be used to generate the FIQ from these sources for test purposes. When an interrupt source is programmed as an FIQ, and IRQ cannot be generated from that source.

5.2.3 Interrupt Sources

Table 5-1 lists all of the interrupt sources on the i.MX23. Use hw_irq.h to access these bits.

Table 5-1. i.MX23 Interrupt Sources

INTERRUPT SOURCE	SRC	VECTOR	DESCRIPTION
DEBUG_UART	0	0x0000	Non DMA on the debug UART
COMMS_RX,COMMS_TX	1	0x0004	JTAG debug communications port
SSP2_ERROR	2	0x0008	SSP2 device-level error and status
VDD5V	3	0x000C	IRQ on 5V connect or disconnect. Shared with DCDC status, Linear Regulator status, PSWITCH, and Host 4.2V
HEADPHONE_SHORT	4	0x0010	HEADPHONE_SHORT
DAC_DMA	5	0x0014	DAC DMA channel
DAC_ERROR	6	0x0018	DAC FIFO buffer underflow
ADC_DMA	7	0x001C	ADC DMA channel
ADC_ERROR	8	0x0020	ADC FIFO buffer overflow
SPDIF_DMA,SAIF2_DMA	9	0x0024	SPDIF DMA channel, SAIF2 DMA channel
SPDIF_ERROR, SAIF1_IRQ, SAIF2_IRQ	10	0x0028	SPDIF, SAIF1, SAIF2 FIFO underflow/overflow

Table 5-1. i.MX23 Interrupt Sources (continued)

INTERRUPT SOURCE	SRC	VECTOR	DESCRIPTION
USB_CTRL	11	0x002C	USB controller
USB_WAKEUP	12	0x0030	USB wakeup. Also ARC core to remain suspended.
GPMI_DMA	13	0x0034	From DMA channel for GPMI
SSP1_DMA	14	0x0038	From DMA channel for SSP1
SSP_ERROR	15	0x003C	SSP1 device-level error and status
GPIO0	16	0x0040	GPIO bank 0 interrupt
GPIO1	17	0x0044	GPIO bank 1 interrupt
GPIO2	18	0x0048	GPIO bank 2 interrupt
SAIF1_DMA	19	0x004C	SAIF1 DMA channel
SSP2_DMA	20	0x0050	From DMA channel for SSP2
ECC8_IRQ	21	0x0054	ECC8 completion interrupt
RTC_ALARM	22	0x0058	RTC alarm event
UARTAPP_TX_DMA	23	0x005C	Application UART1 transmitter DMA
UARTAPP_INTERNAL	24	0x0060	Application UART1 internal error
UARTAPP_RX_DMA	25	0x0064	Application UART1 receiver DMA
I2C_DMA	26	0x0068	From DMA channel for I^2C
I2C_ERROR	27	0x006C	From I^2C device detected errors and line conditions
TIMER0	28	0x0070	TIMROT Timer0, recommend to set as FIQ.
TIMER1	29	0x0074	TIMROT Timer1, recommend to set as FIQ.
TIMER2	30	0x0078	TIMROT Timer2, recommend to set as FIQ.
TIMER3	31	0x007C	TIMROT Timer3, recommend to set as FIQ.
BATT_BRNOUT	32	0x0080	Power module battery brownout detect, recommend to set as FIQ.
VDDD_BRNOUT	33	0x0084	Power module VDDD brownout detect, recommend to set as FIQ.
VDDIO_BRNOUT	34	0x0088	Power module VDDIO brownout detect, recommend to set as FIQ.
VDD18_BRNOUT	35	0x008C	Power module VDD18 brownout detect, recommend to set as FIQ.
TOUCH_DETECT	36	0x0090	Touch detection.
LRADC_CH0	37	0x0094	Channel 0 complete.
LRADC_CH1	38	0x0098	Channel 1 complete.
LRADC_CH2	39	0x009C	Channel 2 complete.
LRADC_CH3	40	0x00A0	Channel 3 complete.
LRADC_CH4	41	0x00A4	Channel 4 complete.

Table 5-1. i.MX23 Interrupt Sources (continued)

INTERRUPT SOURCE	SRC	VECTOR	DESCRIPTION
LRADC_CH5	42	0x00A8	Channel 5 complete.
LRADC_CH6	43	0x00AC	Channel 6 complete.
LRADC_CH7	44	0x00B0	Channel 7 complete.
LCDIF_DMA	45	0x00B4	From DMA channel for LCDIF.
LCDIF_ERROR	46	0x00B8	LCDIF error.
DIGCTL_DEBUG_TRAP	47	0x00BC	AHB arbiter debug trap.
RTC_1MSEC	48	0x00C0	RTC 1 ms tick interrupt.
RSVD	49	0x00C4	Reserved
RSVD	50	0x00C8	Reserved
GPMI	51	0x00CC	From GPMI internal error and status IRQ.
RSVD	52	0x00D0	Reserved
DCP_VMI	53	0x00D4	DCP Channel 0 virtual memory page copy.
DCP	54	0x00D8	DCP
RSVD	55	0x00DC	Reserved.
BCH	56	0x00E0	BCH consolidated Interrupt
PXP	57	0x00E4	Pixel Pipeline consolidated Interrupt
UARTAPP2_TX_DMA	58	0x00E8	Application UART2 transmitter DMA
UARTAPP2_INTERNAL	59	0x00EC	Application UART2 internal error
UARTAPP2_RX_DMA	60	0x00F0	Application UART2 receiver DMA
VDAC_DETECT	61	0x00F4	Video dac, jack presence auto-detect
RSVD	62	0x00F8	Reserved.
RSVD	63	0x00FC	Reserved.
VDD5V_DROOP	64	0x0100	5V Droop, recommend to be set as FIQ.
DCDC4P2_BO	65	0x0104	4.2V regulated supply brown-out, recommend to be set as FIQ.
RSVD	66-1278	0x0108-01FC	Reserved.

5.2.4 CPU Wait-for-Interrupt Mode

To enable wait-for-interrupt mode, two distinct actions are required by the programmer:

Set the INTERRUPT_WAIT bit in the HW_CLKCTRL_CPUCLKCTRL register. This must be done via a RMW operation. For example:

```
uclkctrl  = HW_CLKCTRL_CPUCLKCTRL_RD();
uclkctrl |= BM_CLKCTRL_CPUCLKCTRL_INTERRUPT_WAIT;
HW_CLKCTRL_CPUCLKCTRL_WR(uclkctrl);
```

8. After setting the INTERRUPT_WAIT bit, a coprocessor instruction is required.

```
asm (
            // Note: R0 is used in the following example, but any usual
            // <Rd> register may be used.
"mov R0, 0;" // Rd SBZ (should be zero)
"mcr p15,0,r0,c7,c0,4;"//Drain write buffers, idle CPU clock & processor,
                       // and stop processor at this instruction
"nop");      // The lr sent to handler points here after RTI
```

The coprocessor instruction sequence above enables an internal gating signal. This internal signal guarantees that write buffers are drained and ensures that the processor is in an idle state. On execution of the MCR coprocessor instruction, the CPU clock is stopped and the processor halts on the instruction—waiting for an interrupt to occur.

The INTERRUPT_WAIT bit can be thought of as a Wait-for-Interrupt enable bit. Therefore, it must be set prior to execution of the MCR instruction. It is recommended that, when the Wait-for-Interrupt mode is to be used, the INTERRUPT_WAIT bit be set at initialization time and left on.

With the INTERRUPT_WAIT bit set, after execution of the MCR WFI command, the processor halts on the MCR instruction. When an interrupt or FIQ occurs, the MCR instruction completes and the IRQ or FIQ handler is entered normally. The return link that is passed to the handler is automatically adjusted by the above MCR instruction, such that a normal return from interrupt results in continuing execution at the instruction immediately following the MCR. That is, the LR will contain the address of the MCR instruction plus eight, such that a typical return from interrupt instruction (e.g., subs pc, LR, 4) will return to the instruction immediately following the MCR (the NOP in the example above).

Whenever the CPU is stopped because the clock control HW_CLKCTRL_ CPUCLKCTRL_INTERRUPT_WAIT bit is set and the MCR WFI instruction is executed, the CPU stops until an interrupt occurs. The actual condition that wakes up the CPU is determined by ORing together all enabled interrupt requests including those that are directed to the FIQ CPU input. The ICOLL_BUSY output signal from the ICOLL communicates this information to the clock control. This function does not pass through the normal ICOLL state machine. It starts the CPU clock as soon as an enabled interrupt arrives.

5.3 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, "Correct Way to Soft Reset a Block, for additional information on using the SFTRST and CLKGATE bit fields.

5.4 Programmable Registers

The following registers provide interrupt generation and control for the i.MX23.

5.4.1 Interrupt Collector Interrupt Vector Address Register Description

This register is can be read by the Interrupt Service Routine using a load PC instruction. The priority logic presents the vector address of the next IRQ interrupt to be processed by the CPU. The vector address is held until a new ISR is entered..

HW_ICOLL_VECTOR	0x000
HW_ICOLL_VECTOR_SET	0x004
HW_ICOLL_VECTOR_CLR	0x008
HW_ICOLL_VECTOR_TOG	0x00C

Table 5-2. HW_ICOLL_VECTOR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IRQVECTOR																														RSRVD1	

Table 5-3. HW_ICOLL_VECTOR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:2	**IRQVECTOR**	RW	0x0	This register presents the vector address for the interrupt currently active on the CPU IRQ input. Writing to this register notifies the interrupt collector that the interrupt service routine for the current interrupt has been entered (alternatively when ARM_RSE_MODE is set, reading this register is required).
1:0	**RSRVD1**	RO	0x0	Always write zeroes to this field.

DESCRIPTION:

This register mediates the vectored interrupt collectors interface with the CPU when it enteres the IRQ exception trap. The exception trap should have a LDPC instruction from this address.

EXAMPLE:

```
LDPC HW_ICOLL_VECTOR_ADDR; IRQ exception at 0xffff0018
```

5.4.2 Interrupt Collector Level Acknowledge Register Description

The Interrupt Collector Level Acknowledge Register is used by software to indicate the completion of an interrupt on a specific level.

HW_ICOLL_LEVELACK	0x010

Table 5-4. HW_ICOLL_LEVELACK

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																												IRQLEVELACK			

Table 5-5. HW_ICOLL_LEVELACK Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:4	**RSRVD1**	RO	0x0	Any value can be written to this bitfield. Writes are ignored.
3:0	**IRQLEVELACK**	RW	0x0	This bitfield is written by the processor to acknowledge the completion of an interrupt. The value written must correspond to the priority level of the completed interrupt LEVEL0 = 0x1 level 0 LEVEL1 = 0x2 level 1 LEVEL2 = 0x4 level 2 LEVEL3 = 0x8 level 3

DESCRIPTION:

This register is written to advance the ICOLL internal irq state machine. It advances from an in-service on a level state to the next pending interrupt level or to the idle state. This register is written at the very end of an interrupt service routine. If nesting is used then the CPU irq must be turned on before writing to this register to avoid a race condition in the CPU interrupt hardware. WARNING: the value written to the level ack register is decoded not binary, i.e. 8, 4, 2, 1.

EXAMPLE:

```
HW_ICOLL_LEVELACK_WR(HW_ICOLL_LEVELACK__LEVEL3);
```

5.4.3 Interrupt Collector Control Register Description

The Interrupt Collector Control Register provides overall control of interrupts being routed to the CPU. This register is not at offset zero from the block base because that location is needed for single 32 bit instructions to be placed in the exception vector location.

HW_ICOLL_CTRL	0x020
HW_ICOLL_CTRL_SET	0x024
HW_ICOLL_CTRL_CLR	0x028
HW_ICOLL_CTRL_TOG	0x02C

Table 5-6. HW_ICOLL_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	RSRVD3						VECTOR_PITCH			BYPASS_FSM	NO_NESTING	ARM_RSE_MODE	FIQ_FINAL_ENABLE	IRQ_FINAL_ENABLE	RSRVD1																	

Table 5-7. HW_ICOLL_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	When set to one, this bit causes a soft reset to the entire interrupt collector. This bit must be turned off for normal operation. RUN = 0x0 Allow the interrupt collector to operate normally. IN_RESET = 0x1 Hold the interrupt collector in its reset state.
30	**CLKGATE**	RW	0x1	When set to one, this bit causes all clocks within the interrupt collector to be gated off. WARNING: Do not set this bit at the same time as SFTRST. Doing so, causes the softreset to have no effect. Setting SFTRST will cause the CLKGATE bit to set automatically four clocks later. RUN = 0x0 Enable clocks for normal operation of interrupt collector. NO_CLOCKS = 0x1 disable clocking within the interrupt collector.
29:24	**RSRVD3**	RO	0x0	Always write zeroes to this bitfield.
23:21	**VECTOR_PITCH**	RW	0x0	When an interrupt occurs one of the 128 input requests becomes the winning bit number, i.e. 0 to 127. This bit field selects one of eight constant multiplier values to multiply the winning bit number. The multiplied bit number is added to the vector table base to become the vector address. 0x0 and 0x1 yield a multiplier of 4 bytes. 0x2 yields a multiplier of 8 bytes while 0x3 yields a multiplier of 12 bytes, i.e. (8 + 4) bytes per step. DEFAULT_BY4 = 0x0 one word pitch BY4 = 0x1 one word pitch BY8 = 0x2 two word pitch BY12 = 0x3 three word pitch BY16 = 0x4 four word pitch BY20 = 0x5 five word pitch BY24 = 0x6 six word pitch BY28 = 0x7 seven word pitch
20	**BYPASS_FSM**	RW	0x0	Set this bit to one to bypass the FSM control of the request holding register and the vector address. With this bit set to one, the vector address register is continuously updated as interrupt requests come in. Turn off all enable bits and walk a one through the software interrupts, observing the vector address changes. Set to zero for normal operation. This control is included as a test mode, and is not intended for use by a real application. NORMAL = 0x0 Normal BYPASS = 0x1 no FSM handshake with CPU

Table 5-7. HW_ICOLL_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19	**NO_NESTING**	RW	0x0	Set this bit to one disable interrupt level nesting, i.e. higher priority interrupt interrupting lower priority. For normal operation, set this bit to zero. NORMAL = 0x0 Normal NO_NEST = 0x1 no support for interrupt nesting
18	**ARM_RSE_MODE**	RW	0x0	Set this bit to one enable the ARM-style read side effect associated with the vector address register. In this mode, interrupt inservice is signalled by the read of the HW_ICOLL_VECTOR register to acquire the interrupt vector address. Set this bit to zero for normal operation, in which the ISR signals inservice explicitly by means of a write to the HW_ICOLL_VECTOR register.
17	**FIQ_FINAL_ENABLE**	RW	0x1	Set this bit to one to enable the final FIQ output to the CPU. Set this bit to zero for testing the interrupt collector without causing actual CPU interrupts. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
16	**IRQ_FINAL_ENABLE**	RW	0x1	Set this bit to one to enable the final IRQ output to the CPU. Set this bit to zero for testing the interrupt collector without causing actual CPU interrupts. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
15:0	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.

DESCRIPTION:

This register handles the overall control of the interrupt collector, including soft reset and clock gate. In addition, it handles state machine variations like NO_NESTING and ARM read side effect processing on the vector address register.

EXAMPLE:

```
HW_ICOLL_CTRL_CLR(BM_ICOLL_CTRL_SFTRST | BM_ICOLL_CTRL_SFTRST );
```

5.4.4 Interrupt Collector Interrupt Vector Base Address Register Description

This register is used by the priority logic to generate a unique vector address for each of the 80 interrupt request lines coming into the interrupt collector. The vector address is formed by multiply the interrupt bit number by 4 and adding it to the vector base address.

HW_ICOLL_VBASE	0x040
HW_ICOLL_VBASE_SET	0x044
HW_ICOLL_VBASE_CLR	0x048
HW_ICOLL_VBASE_TOG	0x04C

Table 5-8. HW_ICOLL_VBASE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TABLE_ADDRESS																														RSRVD1	

Table 5-9. HW_ICOLL_VBASE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:2	**TABLE_ADDRESS**	RW	0x0	This bitfield holds the upper 30 bits of the base address of the vector table.
1:0	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.

DESCRIPTION:

This register provides a mechanism to specify the base address of the interrupt vector table. It is used in the compuation of the value supplied in HW_ICOLL_VECTOR register.

EXAMPLE:

```
HW_ICOLL_VBASE_WR(pInterruptVectorTable);
```

5.4.5 Interrupt Collector Status Register Description

Read only view into various internal states, including the Vector number of the current interupt.

HW_ICOLL_STAT 0x070

Table 5-10. HW_ICOLL_STAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																									VECTOR_NUMBER						

Table 5-11. HW_ICOLL_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:7	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
6:0	**VECTOR_NUMBER**	RO	0x7F	Vector number of current interrupt. Multiply by 4 * HW_ICOLL_CTRL[VECTOR_PITCH] and add to vector base address to obtain the value in HW_ICOLL_VECTOR.

DESCRIPTION:

This register is used to test interrupt collector state machine and its associated request holding register.

EXAMPLE:

```
if(HW_ICOLL_STAT_VECTOR_NUMBER_READ() == 0x00000017) ISR_vector_23(); // ISR for vector 23 decimal,
17 hex
```

5.4.6 Interrupt Collector Raw Interrupt Input Register 0 Description

The lower 32 interrupt hardware-source states are visible in this read-only register.

HW_ICOLL_RAW0	0x0A0
HW_ICOLL_RAW0_SET	0x0A4
HW_ICOLL_RAW0_CLR	0x0A8
HW_ICOLL_RAW0_TOG	0x0AC

Table 5-12. HW_ICOLL_RAW0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RAW_IRQS																															

Table 5-13. HW_ICOLL_RAW0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**RAW_IRQS**	RO	0x0	read-only view of the lower 32 hardware interrupt request bits.

DESCRIPTION:

This register provides a read-only view of the raw interrupt request lines coming from various parts of the chip. Its purpose is to improve diagnostic observability. Note that these only capture the state of hardware interrupt sources.

EXAMPLE:

```
ulTest = HW_ICOLL_RAW0.RAW_IRQS;
```

5.4.7 Interrupt Collector Raw Interrupt Input Register 1 Description

Interrupt hardware-source states 32-63 are visible in this read-only register.

HW_ICOLL_RAW1	0x0B0
HW_ICOLL_RAW1_SET	0x0B4
HW_ICOLL_RAW1_CLR	0x0B8

HW_ICOLL_RAW1_TOG 0x0BC

Table 5-14. HW_ICOLL_RAW1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RAW_IRQS																															

Table 5-15. HW_ICOLL_RAW1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**RAW_IRQS**	RO	0x0	read-only view of hardware interrupt request bits 32-63.

DESCRIPTION:

This register provides a read-only view of the raw interrupt request lines coming from various parts of the chip. The purpose is to improve diagnostic observability. Note that these only capture the state of hardware interrupt sources.

EXAMPLE:

```
ulTest = HW_ICOLL_RAW0.RAW_IRQS;
```

5.4.8 Interrupt Collector Raw Interrupt Input Register 2 Description

Interrupt hardware-source states 64-95 are visible in this read-only register.

HW_ICOLL_RAW2 0x0C0
HW_ICOLL_RAW2_SET 0x0C4
HW_ICOLL_RAW2_CLR 0x0C8
HW_ICOLL_RAW2_TOG 0x0CC

Table 5-16. HW_ICOLL_RAW2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RAW_IRQS																															

Table 5-17. HW_ICOLL_RAW2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**RAW_IRQS**	RO	0x0	read-only view of hardware interrupt request bits 64-95.

DESCRIPTION:

This register provides a read-only view of the raw interrupt request lines coming from various parts of the chip. The purpose is to improve diagnostic observability. Note that these only capture the state of hardware interrupt sources.

EXAMPLE:

```
ulTest = HW_ICOLL_RAW0.RAW_IRQS;
```

5.4.9 Interrupt Collector Raw Interrupt Input Register 3 Description

Interrupt hardware-source states 96-127 are visible in this read-only register.

HW_ICOLL_RAW3	0x0D0
HW_ICOLL_RAW3_SET	0x0D4
HW_ICOLL_RAW3_CLR	0x0D8
HW_ICOLL_RAW3_TOG	0x0DC

Table 5-18. HW_ICOLL_RAW3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RAW_IRQS																															

Table 5-19. HW_ICOLL_RAW3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**RAW_IRQS**	RO	0x0	read-only view of hardware interrupt request bits 96-127.

DESCRIPTION:

This register provides a read-only view of the raw interrupt request lines coming from various parts of the chip. The purpose is to improve diagnostic observability. Note that these only capture the state of hardware interrupt sources.

EXAMPLE:

```
ulTest = HW_ICOLL_RAW0.RAW_IRQS;
```

5.4.10 Interrupt Collector Interrupt Register 0 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT0	0x120
HW_ICOLL_INTERRUPT0_SET	0x124
HW_ICOLL_INTERRUPT0_CLR	0x128
HW_ICOLL_INTERRUPT0_TOG	0x12C

Table 5-20. HW_ICOLL_INTERRUPT0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY		

Table 5-21. HW_ICOLL_INTERRUPT0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT0_SET(0,0x00000001);
```

5.4.11 Interrupt Collector Interrupt Register 1 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT1	0x130
HW_ICOLL_INTERRUPT1_SET	0x134
HW_ICOLL_INTERRUPT1_CLR	0x138
HW_ICOLL_INTERRUPT1_TOG	0x13C

Table 5-22. HW_ICOLL_INTERRUPT1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-23. HW_ICOLL_INTERRUPT1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT1_SET(0,0x00000001);
```

5.4.12 Interrupt Collector Interrupt Register 2 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT2	0x140
HW_ICOLL_INTERRUPT2_SET	0x144
HW_ICOLL_INTERRUPT2_CLR	0x148
HW_ICOLL_INTERRUPT2_TOG	0x14C

Table 5-24. HW_ICOLL_INTERRUPT2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-25. HW_ICOLL_INTERRUPT2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT2_SET(0,0x00000001);
```

5.4.13 Interrupt Collector Interrupt Register 3 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT3	0x150
HW_ICOLL_INTERRUPT3_SET	0x154
HW_ICOLL_INTERRUPT3_CLR	0x158
HW_ICOLL_INTERRUPT3_TOG	0x15C

Table 5-26. HW_ICOLL_INTERRUPT3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-27. HW_ICOLL_INTERRUPT3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT3_SET(0,0x00000001);
```

5.4.14 Interrupt Collector Interrupt Register 4 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT4	0x160
HW_ICOLL_INTERRUPT4_SET	0x164
HW_ICOLL_INTERRUPT4_CLR	0x168
HW_ICOLL_INTERRUPT4_TOG	0x16C

Table 5-28. HW_ICOLL_INTERRUPT4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-29. HW_ICOLL_INTERRUPT4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT4_SET(0,0x00000001);
```

5.4.15 Interrupt Collector Interrupt Register 5 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT5	0x170
HW_ICOLL_INTERRUPT5_SET	0x174
HW_ICOLL_INTERRUPT5_CLR	0x178
HW_ICOLL_INTERRUPT5_TOG	0x17C

Table 5-30. HW_ICOLL_INTERRUPT5

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-31. HW_ICOLL_INTERRUPT5 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT5_SET(0,0x00000001);
```

5.4.16 Interrupt Collector Interrupt Register 6 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT6	0x180
HW_ICOLL_INTERRUPT6_SET	0x184
HW_ICOLL_INTERRUPT6_CLR	0x188
HW_ICOLL_INTERRUPT6_TOG	0x18C

Table 5-32. HW_ICOLL_INTERRUPT6

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-33. HW_ICOLL_INTERRUPT6 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT6_SET(0,0x00000001);
```

5.4.17 Interrupt Collector Interrupt Register 7 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT7	0x190
HW_ICOLL_INTERRUPT7_SET	0x194
HW_ICOLL_INTERRUPT7_CLR	0x198
HW_ICOLL_INTERRUPT7_TOG	0x19C

Table 5-34. HW_ICOLL_INTERRUPT7

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-35. HW_ICOLL_INTERRUPT7 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT7_SET(0,0x00000001);
```

5.4.18 Interrupt Collector Interrupt Register 8 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT8	0x1A0
HW_ICOLL_INTERRUPT8_SET	0x1A4
HW_ICOLL_INTERRUPT8_CLR	0x1A8
HW_ICOLL_INTERRUPT8_TOG	0x1AC

Table 5-36. HW_ICOLL_INTERRUPT8

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-37. HW_ICOLL_INTERRUPT8 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT8_SET(0,0x00000001);
```

5.4.19 Interrupt Collector Interrupt Register 9 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT9	0x1B0
HW_ICOLL_INTERRUPT9_SET	0x1B4
HW_ICOLL_INTERRUPT9_CLR	0x1B8
HW_ICOLL_INTERRUPT9_TOG	0x1BC

Table 5-38. HW_ICOLL_INTERRUPT9

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-39. HW_ICOLL_INTERRUPT9 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT9_SET(0,0x00000001);
```

5.4.20 Interrupt Collector Interrupt Register 10 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT10	0x1C0
HW_ICOLL_INTERRUPT10_SET	0x1C4
HW_ICOLL_INTERRUPT10_CLR	0x1C8
HW_ICOLL_INTERRUPT10_TOG	0x1CC

Table 5-40. HW_ICOLL_INTERRUPT10

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-41. HW_ICOLL_INTERRUPT10 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT10_SET(0,0x00000001);
```

5.4.21 Interrupt Collector Interrupt Register 11 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT11	0x1D0
HW_ICOLL_INTERRUPT11_SET	0x1D4
HW_ICOLL_INTERRUPT11_CLR	0x1D8
HW_ICOLL_INTERRUPT11_TOG	0x1DC

Table 5-42. HW_ICOLL_INTERRUPT11

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-43. HW_ICOLL_INTERRUPT11 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT11_SET(0,0x00000001);
```

5.4.22 Interrupt Collector Interrupt Register 12 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT12	0x1E0
HW_ICOLL_INTERRUPT12_SET	0x1E4
HW_ICOLL_INTERRUPT12_CLR	0x1E8
HW_ICOLL_INTERRUPT12_TOG	0x1EC

Table 5-44. HW_ICOLL_INTERRUPT12

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-45. HW_ICOLL_INTERRUPT12 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT12_SET(0,0x00000001);
```

5.4.23 Interrupt Collector Interrupt Register 13 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT13	0x1F0
HW_ICOLL_INTERRUPT13_SET	0x1F4
HW_ICOLL_INTERRUPT13_CLR	0x1F8
HW_ICOLL_INTERRUPT13_TOG	0x1FC

Table 5-46. HW_ICOLL_INTERRUPT13

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-47. HW_ICOLL_INTERRUPT13 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT13_SET(0,0x00000001);
```

5.4.24 Interrupt Collector Interrupt Register 14 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT14	0x200
HW_ICOLL_INTERRUPT14_SET	0x204
HW_ICOLL_INTERRUPT14_CLR	0x208
HW_ICOLL_INTERRUPT14_TOG	0x20C

Table 5-48. HW_ICOLL_INTERRUPT14

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-49. HW_ICOLL_INTERRUPT14 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT14_SET(0,0x00000001);
```

5.4.25 Interrupt Collector Interrupt Register 15 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT15	0x210
HW_ICOLL_INTERRUPT15_SET	0x214
HW_ICOLL_INTERRUPT15_CLR	0x218
HW_ICOLL_INTERRUPT15_TOG	0x21C

Table 5-50. HW_ICOLL_INTERRUPT15

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-51. HW_ICOLL_INTERRUPT15 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT15_SET(0,0x00000001);
```

5.4.26 Interrupt Collector Interrupt Register 16 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT16	0x220
HW_ICOLL_INTERRUPT16_SET	0x224
HW_ICOLL_INTERRUPT16_CLR	0x228
HW_ICOLL_INTERRUPT16_TOG	0x22C

Table 5-52. HW_ICOLL_INTERRUPT16

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-53. HW_ICOLL_INTERRUPT16 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT16_SET(0,0x00000001);
```

5.4.27 Interrupt Collector Interrupt Register 17 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT17	0x230
HW_ICOLL_INTERRUPT17_SET	0x234
HW_ICOLL_INTERRUPT17_CLR	0x238
HW_ICOLL_INTERRUPT17_TOG	0x23C

Table 5-54. HW_ICOLL_INTERRUPT17

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-55. HW_ICOLL_INTERRUPT17 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT17_SET(0,0x00000001);
```

5.4.28 Interrupt Collector Interrupt Register 18 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT18	0x240
HW_ICOLL_INTERRUPT18_SET	0x244
HW_ICOLL_INTERRUPT18_CLR	0x248
HW_ICOLL_INTERRUPT18_TOG	0x24C

Table 5-56. HW_ICOLL_INTERRUPT18

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-57. HW_ICOLL_INTERRUPT18 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT18_SET(0,0x00000001);
```

5.4.29 Interrupt Collector Interrupt Register 19 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT19	0x250
HW_ICOLL_INTERRUPT19_SET	0x254
HW_ICOLL_INTERRUPT19_CLR	0x258
HW_ICOLL_INTERRUPT19_TOG	0x25C

Table 5-58. HW_ICOLL_INTERRUPT19

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-59. HW_ICOLL_INTERRUPT19 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT19_SET(0,0x00000001);
```

5.4.30 Interrupt Collector Interrupt Register 20 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT20	0x260
HW_ICOLL_INTERRUPT20_SET	0x264
HW_ICOLL_INTERRUPT20_CLR	0x268
HW_ICOLL_INTERRUPT20_TOG	0x26C

Table 5-60. HW_ICOLL_INTERRUPT20

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-61. HW_ICOLL_INTERRUPT20 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT20_SET(0,0x00000001);
```

5.4.31 Interrupt Collector Interrupt Register 21 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT21	0x270
HW_ICOLL_INTERRUPT21_SET	0x274
HW_ICOLL_INTERRUPT21_CLR	0x278
HW_ICOLL_INTERRUPT21_TOG	0x27C

Table 5-62. HW_ICOLL_INTERRUPT21

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-63. HW_ICOLL_INTERRUPT21 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT21_SET(0,0x00000001);
```

5.4.32 Interrupt Collector Interrupt Register 22 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT22	0x280
HW_ICOLL_INTERRUPT22_SET	0x284
HW_ICOLL_INTERRUPT22_CLR	0x288
HW_ICOLL_INTERRUPT22_TOG	0x28C

Table 5-64. HW_ICOLL_INTERRUPT22

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-65. HW_ICOLL_INTERRUPT22 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT22_SET(0,0x00000001);
```

5.4.33 Interrupt Collector Interrupt Register 23 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT23	0x290
HW_ICOLL_INTERRUPT23_SET	0x294
HW_ICOLL_INTERRUPT23_CLR	0x298
HW_ICOLL_INTERRUPT23_TOG	0x29C

Table 5-66. HW_ICOLL_INTERRUPT23

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-67. HW_ICOLL_INTERRUPT23 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT23_SET(0,0x00000001);
```

5.4.34 Interrupt Collector Interrupt Register 24 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT24	0x2A0
HW_ICOLL_INTERRUPT24_SET	0x2A4
HW_ICOLL_INTERRUPT24_CLR	0x2A8
HW_ICOLL_INTERRUPT24_TOG	0x2AC

Table 5-68. HW_ICOLL_INTERRUPT24

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-69. HW_ICOLL_INTERRUPT24 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT24_SET(0,0x00000001);
```

5.4.35 Interrupt Collector Interrupt Register 25 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT25	0x2B0
HW_ICOLL_INTERRUPT25_SET	0x2B4
HW_ICOLL_INTERRUPT25_CLR	0x2B8
HW_ICOLL_INTERRUPT25_TOG	0x2BC

Table 5-70. HW_ICOLL_INTERRUPT25

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-71. HW_ICOLL_INTERRUPT25 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT25_SET(0,0x00000001);
```

5.4.36 Interrupt Collector Interrupt Register 26 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT26	0x2C0
HW_ICOLL_INTERRUPT26_SET	0x2C4
HW_ICOLL_INTERRUPT26_CLR	0x2C8
HW_ICOLL_INTERRUPT26_TOG	0x2CC

Table 5-72. HW_ICOLL_INTERRUPT26

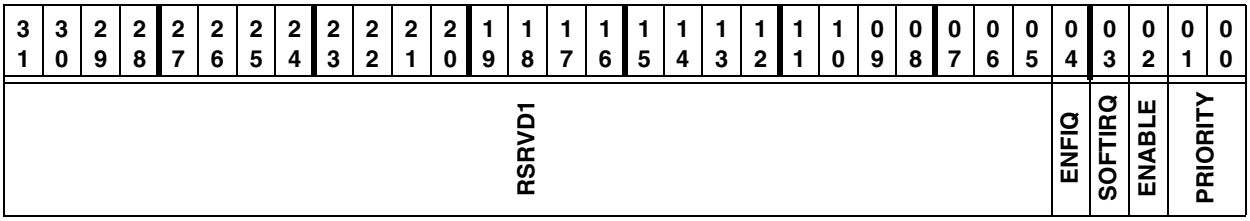


Table 5-73. HW_ICOLL_INTERRUPT26 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT26_SET(0,0x00000001);
```

5.4.37 Interrupt Collector Interrupt Register 27 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT27	0x2D0
HW_ICOLL_INTERRUPT27_SET	0x2D4
HW_ICOLL_INTERRUPT27_CLR	0x2D8
HW_ICOLL_INTERRUPT27_TOG	0x2DC

Table 5-74. HW_ICOLL_INTERRUPT27

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-75. HW_ICOLL_INTERRUPT27 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT27_SET(0,0x00000001);
```

5.4.38 Interrupt Collector Interrupt Register 28 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT28	0x2E0
HW_ICOLL_INTERRUPT28_SET	0x2E4
HW_ICOLL_INTERRUPT28_CLR	0x2E8
HW_ICOLL_INTERRUPT28_TOG	0x2EC

Table 5-76. HW_ICOLL_INTERRUPT28

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-77. HW_ICOLL_INTERRUPT28 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT28_SET(0,0x00000001);
```

5.4.39 Interrupt Collector Interrupt Register 29 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT29	0x2F0
HW_ICOLL_INTERRUPT29_SET	0x2F4
HW_ICOLL_INTERRUPT29_CLR	0x2F8
HW_ICOLL_INTERRUPT29_TOG	0x2FC

Table 5-78. HW_ICOLL_INTERRUPT29

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-79. HW_ICOLL_INTERRUPT29 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT29_SET(0,0x00000001);
```

5.4.40 Interrupt Collector Interrupt Register 30 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT30	0x300
HW_ICOLL_INTERRUPT30_SET	0x304
HW_ICOLL_INTERRUPT30_CLR	0x308
HW_ICOLL_INTERRUPT30_TOG	0x30C

Table 5-80. HW_ICOLL_INTERRUPT30

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-81. HW_ICOLL_INTERRUPT30 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT30_SET(0,0x00000001);
```

5.4.41 Interrupt Collector Interrupt Register 31 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT31	0x310
HW_ICOLL_INTERRUPT31_SET	0x314
HW_ICOLL_INTERRUPT31_CLR	0x318
HW_ICOLL_INTERRUPT31_TOG	0x31C

Table 5-82. HW_ICOLL_INTERRUPT31

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-83. HW_ICOLL_INTERRUPT31 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT31_SET(0,0x00000001);
```

5.4.42 Interrupt Collector Interrupt Register 32 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT32	0x320
HW_ICOLL_INTERRUPT32_SET	0x324
HW_ICOLL_INTERRUPT32_CLR	0x328
HW_ICOLL_INTERRUPT32_TOG	0x32C

Table 5-84. HW_ICOLL_INTERRUPT32

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-85. HW_ICOLL_INTERRUPT32 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT32_SET(0,0x00000001);
```

5.4.43 Interrupt Collector Interrupt Register 33 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT33	0x330
HW_ICOLL_INTERRUPT33_SET	0x334
HW_ICOLL_INTERRUPT33_CLR	0x338
HW_ICOLL_INTERRUPT33_TOG	0x33C

Table 5-86. HW_ICOLL_INTERRUPT33

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-87. HW_ICOLL_INTERRUPT33 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT33_SET(0,0x00000001);
```

5.4.44 Interrupt Collector Interrupt Register 34 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT34	0x340
HW_ICOLL_INTERRUPT34_SET	0x344
HW_ICOLL_INTERRUPT34_CLR	0x348
HW_ICOLL_INTERRUPT34_TOG	0x34C

Table 5-88. HW_ICOLL_INTERRUPT34

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-89. HW_ICOLL_INTERRUPT34 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT34_SET(0,0x00000001);
```

5.4.45 Interrupt Collector Interrupt Register 35 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT35	0x350
HW_ICOLL_INTERRUPT35_SET	0x354
HW_ICOLL_INTERRUPT35_CLR	0x358
HW_ICOLL_INTERRUPT35_TOG	0x35C

Table 5-90. HW_ICOLL_INTERRUPT35

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-91. HW_ICOLL_INTERRUPT35 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT35_SET(0,0x00000001);
```

5.4.46 Interrupt Collector Interrupt Register 36 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT36	0x360
HW_ICOLL_INTERRUPT36_SET	0x364
HW_ICOLL_INTERRUPT36_CLR	0x368
HW_ICOLL_INTERRUPT36_TOG	0x36C

Table 5-92. HW_ICOLL_INTERRUPT36

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-93. HW_ICOLL_INTERRUPT36 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT36_SET(0,0x00000001);
```

5.4.47 Interrupt Collector Interrupt Register 37 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT37	0x370
HW_ICOLL_INTERRUPT37_SET	0x374
HW_ICOLL_INTERRUPT37_CLR	0x378
HW_ICOLL_INTERRUPT37_TOG	0x37C

Table 5-94. HW_ICOLL_INTERRUPT37

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-95. HW_ICOLL_INTERRUPT37 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT37_SET(0,0x00000001);
```

5.4.48 Interrupt Collector Interrupt Register 38 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT38	0x380
HW_ICOLL_INTERRUPT38_SET	0x384
HW_ICOLL_INTERRUPT38_CLR	0x388
HW_ICOLL_INTERRUPT38_TOG	0x38C

Table 5-96. HW_ICOLL_INTERRUPT38

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-97. HW_ICOLL_INTERRUPT38 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT38_SET(0,0x00000001);
```

5.4.49 Interrupt Collector Interrupt Register 39 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT39	0x390
HW_ICOLL_INTERRUPT39_SET	0x394
HW_ICOLL_INTERRUPT39_CLR	0x398
HW_ICOLL_INTERRUPT39_TOG	0x39C

Table 5-98. HW_ICOLL_INTERRUPT39

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-99. HW_ICOLL_INTERRUPT39 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT39_SET(0,0x00000001);
```

5.4.50 Interrupt Collector Interrupt Register 40 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT40	0x3A0
HW_ICOLL_INTERRUPT40_SET	0x3A4
HW_ICOLL_INTERRUPT40_CLR	0x3A8
HW_ICOLL_INTERRUPT40_TOG	0x3AC

Table 5-100. HW_ICOLL_INTERRUPT40

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-101. HW_ICOLL_INTERRUPT40 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT40_SET(0,0x00000001);
```

5.4.51 Interrupt Collector Interrupt Register 41 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT41	0x3B0
HW_ICOLL_INTERRUPT41_SET	0x3B4
HW_ICOLL_INTERRUPT41_CLR	0x3B8
HW_ICOLL_INTERRUPT41_TOG	0x3BC

Table 5-102. HW_ICOLL_INTERRUPT41

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-103. HW_ICOLL_INTERRUPT41 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT41_SET(0,0x00000001);
```

5.4.52 Interrupt Collector Interrupt Register 42 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT42	0x3C0
HW_ICOLL_INTERRUPT42_SET	0x3C4
HW_ICOLL_INTERRUPT42_CLR	0x3C8
HW_ICOLL_INTERRUPT42_TOG	0x3CC

Table 5-104. HW_ICOLL_INTERRUPT42

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-105. HW_ICOLL_INTERRUPT42 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT42_SET(0,0x00000001);
```

5.4.53 Interrupt Collector Interrupt Register 43 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT43	0x3D0
HW_ICOLL_INTERRUPT43_SET	0x3D4
HW_ICOLL_INTERRUPT43_CLR	0x3D8
HW_ICOLL_INTERRUPT43_TOG	0x3DC

Table 5-106. HW_ICOLL_INTERRUPT43

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-107. HW_ICOLL_INTERRUPT43 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT43_SET(0,0x00000001);
```

5.4.54 Interrupt Collector Interrupt Register 44 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT44	0x3E0
HW_ICOLL_INTERRUPT44_SET	0x3E4
HW_ICOLL_INTERRUPT44_CLR	0x3E8
HW_ICOLL_INTERRUPT44_TOG	0x3EC

Table 5-108. HW_ICOLL_INTERRUPT44

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-109. HW_ICOLL_INTERRUPT44 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT44_SET(0,0x00000001);
```

5.4.55 Interrupt Collector Interrupt Register 45 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT45	0x3F0
HW_ICOLL_INTERRUPT45_SET	0x3F4
HW_ICOLL_INTERRUPT45_CLR	0x3F8
HW_ICOLL_INTERRUPT45_TOG	0x3FC

Table 5-110. HW_ICOLL_INTERRUPT45

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-111. HW_ICOLL_INTERRUPT45 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT45_SET(0,0x00000001);
```

5.4.56 Interrupt Collector Interrupt Register 46 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT46	0x400
HW_ICOLL_INTERRUPT46_SET	0x404
HW_ICOLL_INTERRUPT46_CLR	0x408
HW_ICOLL_INTERRUPT46_TOG	0x40C

Table 5-112. HW_ICOLL_INTERRUPT46

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-113. HW_ICOLL_INTERRUPT46 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT46_SET(0,0x00000001);
```

5.4.57 Interrupt Collector Interrupt Register 47 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT47	0x410
HW_ICOLL_INTERRUPT47_SET	0x414
HW_ICOLL_INTERRUPT47_CLR	0x418
HW_ICOLL_INTERRUPT47_TOG	0x41C

Table 5-114. HW_ICOLL_INTERRUPT47

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-115. HW_ICOLL_INTERRUPT47 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT47_SET(0,0x00000001);
```

5.4.58 Interrupt Collector Interrupt Register 48 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT48	0x420
HW_ICOLL_INTERRUPT48_SET	0x424
HW_ICOLL_INTERRUPT48_CLR	0x428
HW_ICOLL_INTERRUPT48_TOG	0x42C

Table 5-116. HW_ICOLL_INTERRUPT48

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-117. HW_ICOLL_INTERRUPT48 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT48_SET(0,0x00000001);
```

5.4.59 Interrupt Collector Interrupt Register 49 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT49	0x430
HW_ICOLL_INTERRUPT49_SET	0x434
HW_ICOLL_INTERRUPT49_CLR	0x438
HW_ICOLL_INTERRUPT49_TOG	0x43C

Table 5-118. HW_ICOLL_INTERRUPT49

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-119. HW_ICOLL_INTERRUPT49 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT49_SET(0,0x00000001);
```

5.4.60 Interrupt Collector Interrupt Register 50 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT50	0x440
HW_ICOLL_INTERRUPT50_SET	0x444
HW_ICOLL_INTERRUPT50_CLR	0x448
HW_ICOLL_INTERRUPT50_TOG	0x44C

Table 5-120. HW_ICOLL_INTERRUPT50

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-121. HW_ICOLL_INTERRUPT50 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT50_SET(0,0x00000001);
```

5.4.61 Interrupt Collector Interrupt Register 51 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT51	0x450
HW_ICOLL_INTERRUPT51_SET	0x454
HW_ICOLL_INTERRUPT51_CLR	0x458
HW_ICOLL_INTERRUPT51_TOG	0x45C

Table 5-122. HW_ICOLL_INTERRUPT51

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-123. HW_ICOLL_INTERRUPT51 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT51_SET(0,0x00000001);
```

5.4.62 Interrupt Collector Interrupt Register 52 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT52	0x460
HW_ICOLL_INTERRUPT52_SET	0x464
HW_ICOLL_INTERRUPT52_CLR	0x468
HW_ICOLL_INTERRUPT52_TOG	0x46C

Table 5-124. HW_ICOLL_INTERRUPT52

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-125. HW_ICOLL_INTERRUPT52 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT52_SET(0,0x00000001);
```

5.4.63 Interrupt Collector Interrupt Register 53 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT53	0x470
HW_ICOLL_INTERRUPT53_SET	0x474
HW_ICOLL_INTERRUPT53_CLR	0x478
HW_ICOLL_INTERRUPT53_TOG	0x47C

Table 5-126. HW_ICOLL_INTERRUPT53

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-127. HW_ICOLL_INTERRUPT53 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT53_SET(0,0x00000001);
```

5.4.64 Interrupt Collector Interrupt Register 54 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT54	0x480
HW_ICOLL_INTERRUPT54_SET	0x484
HW_ICOLL_INTERRUPT54_CLR	0x488
HW_ICOLL_INTERRUPT54_TOG	0x48C

Table 5-128. HW_ICOLL_INTERRUPT54

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-129. HW_ICOLL_INTERRUPT54 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT54_SET(0,0x00000001);
```

5.4.65 Interrupt Collector Interrupt Register 55 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT55	0x490
HW_ICOLL_INTERRUPT55_SET	0x494
HW_ICOLL_INTERRUPT55_CLR	0x498
HW_ICOLL_INTERRUPT55_TOG	0x49C

Table 5-130. HW_ICOLL_INTERRUPT55

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-131. HW_ICOLL_INTERRUPT55 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT55_SET(0,0x00000001);
```

5.4.66 Interrupt Collector Interrupt Register 56 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT56	0x4A0
HW_ICOLL_INTERRUPT56_SET	0x4A4
HW_ICOLL_INTERRUPT56_CLR	0x4A8
HW_ICOLL_INTERRUPT56_TOG	0x4AC

Table 5-132. HW_ICOLL_INTERRUPT56

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-133. HW_ICOLL_INTERRUPT56 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT56_SET(0,0x00000001);
```

5.4.67 Interrupt Collector Interrupt Register 57 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT57	0x4B0
HW_ICOLL_INTERRUPT57_SET	0x4B4
HW_ICOLL_INTERRUPT57_CLR	0x4B8
HW_ICOLL_INTERRUPT57_TOG	0x4BC

Table 5-134. HW_ICOLL_INTERRUPT57

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-135. HW_ICOLL_INTERRUPT57 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT57_SET(0,0x00000001);
```

5.4.68 Interrupt Collector Interrupt Register 58 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT58	0x4C0
HW_ICOLL_INTERRUPT58_SET	0x4C4
HW_ICOLL_INTERRUPT58_CLR	0x4C8
HW_ICOLL_INTERRUPT58_TOG	0x4CC

Table 5-136. HW_ICOLL_INTERRUPT58

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-137. HW_ICOLL_INTERRUPT58 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT58_SET(0,0x00000001);
```

5.4.69 Interrupt Collector Interrupt Register 59 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT59	0x4D0
HW_ICOLL_INTERRUPT59_SET	0x4D4
HW_ICOLL_INTERRUPT59_CLR	0x4D8
HW_ICOLL_INTERRUPT59_TOG	0x4DC

Table 5-138. HW_ICOLL_INTERRUPT59

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-139. HW_ICOLL_INTERRUPT59 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT59_SET(0,0x00000001);
```

5.4.70 Interrupt Collector Interrupt Register 60 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT60	0x4E0
HW_ICOLL_INTERRUPT60_SET	0x4E4
HW_ICOLL_INTERRUPT60_CLR	0x4E8
HW_ICOLL_INTERRUPT60_TOG	0x4EC

Table 5-140. HW_ICOLL_INTERRUPT60

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-141. HW_ICOLL_INTERRUPT60 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT60_SET(0,0x00000001);
```

5.4.71 Interrupt Collector Interrupt Register 61 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT61	0x4F0
HW_ICOLL_INTERRUPT61_SET	0x4F4
HW_ICOLL_INTERRUPT61_CLR	0x4F8
HW_ICOLL_INTERRUPT61_TOG	0x4FC

Table 5-142. HW_ICOLL_INTERRUPT61

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-143. HW_ICOLL_INTERRUPT61 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT61_SET(0,0x00000001);
```

5.4.72 Interrupt Collector Interrupt Register 62 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT62	0x500
HW_ICOLL_INTERRUPT62_SET	0x504
HW_ICOLL_INTERRUPT62_CLR	0x508
HW_ICOLL_INTERRUPT62_TOG	0x50C

Table 5-144. HW_ICOLL_INTERRUPT62

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-145. HW_ICOLL_INTERRUPT62 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT62_SET(0,0x00000001);
```

5.4.73 Interrupt Collector Interrupt Register 63 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT63	0x510
HW_ICOLL_INTERRUPT63_SET	0x514
HW_ICOLL_INTERRUPT63_CLR	0x518
HW_ICOLL_INTERRUPT63_TOG	0x51C

Table 5-146. HW_ICOLL_INTERRUPT63

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-147. HW_ICOLL_INTERRUPT63 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT63_SET(0,0x00000001);
```

5.4.74 Interrupt Collector Interrupt Register 64 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT64	0x520
HW_ICOLL_INTERRUPT64_SET	0x524
HW_ICOLL_INTERRUPT64_CLR	0x528
HW_ICOLL_INTERRUPT64_TOG	0x52C

Table 5-148. HW_ICOLL_INTERRUPT64

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-149. HW_ICOLL_INTERRUPT64 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT64_SET(0,0x00000001);
```

5.4.75 Interrupt Collector Interrupt Register 65 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT65	0x530
HW_ICOLL_INTERRUPT65_SET	0x534
HW_ICOLL_INTERRUPT65_CLR	0x538
HW_ICOLL_INTERRUPT65_TOG	0x53C

Table 5-150. HW_ICOLL_INTERRUPT65

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-151. HW_ICOLL_INTERRUPT65 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT65_SET(0,0x00000001);
```

5.4.76 Interrupt Collector Interrupt Register 66 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT66	0x540
HW_ICOLL_INTERRUPT66_SET	0x544
HW_ICOLL_INTERRUPT66_CLR	0x548
HW_ICOLL_INTERRUPT66_TOG	0x54C

Table 5-152. HW_ICOLL_INTERRUPT66

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-153. HW_ICOLL_INTERRUPT66 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT66_SET(0,0x00000001);
```

5.4.77 Interrupt Collector Interrupt Register 67 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT67	0x550
HW_ICOLL_INTERRUPT67_SET	0x554
HW_ICOLL_INTERRUPT67_CLR	0x558
HW_ICOLL_INTERRUPT67_TOG	0x55C

Table 5-154. HW_ICOLL_INTERRUPT67

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-155. HW_ICOLL_INTERRUPT67 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT67_SET(0,0x00000001);
```

5.4.78 Interrupt Collector Interrupt Register 68 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT68	0x560
HW_ICOLL_INTERRUPT68_SET	0x564
HW_ICOLL_INTERRUPT68_CLR	0x568
HW_ICOLL_INTERRUPT68_TOG	0x56C

Table 5-156. HW_ICOLL_INTERRUPT68

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-157. HW_ICOLL_INTERRUPT68 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT68_SET(0,0x00000001);
```

5.4.79 Interrupt Collector Interrupt Register 69 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT69	0x570
HW_ICOLL_INTERRUPT69_SET	0x574
HW_ICOLL_INTERRUPT69_CLR	0x578
HW_ICOLL_INTERRUPT69_TOG	0x57C

Table 5-158. HW_ICOLL_INTERRUPT69

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-159. HW_ICOLL_INTERRUPT69 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT69_SET(0,0x00000001);
```

5.4.80 Interrupt Collector Interrupt Register 70 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT70	0x580
HW_ICOLL_INTERRUPT70_SET	0x584
HW_ICOLL_INTERRUPT70_CLR	0x588
HW_ICOLL_INTERRUPT70_TOG	0x58C

Table 5-160. HW_ICOLL_INTERRUPT70

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-161. HW_ICOLL_INTERRUPT70 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT70_SET(0,0x00000001);
```

5.4.81 Interrupt Collector Interrupt Register 71 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT71	0x590
HW_ICOLL_INTERRUPT71_SET	0x594
HW_ICOLL_INTERRUPT71_CLR	0x598
HW_ICOLL_INTERRUPT71_TOG	0x59C

Table 5-162. HW_ICOLL_INTERRUPT71

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-163. HW_ICOLL_INTERRUPT71 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT71_SET(0,0x00000001);
```

5.4.82 Interrupt Collector Interrupt Register 72 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT72	0x5A0
HW_ICOLL_INTERRUPT72_SET	0x5A4
HW_ICOLL_INTERRUPT72_CLR	0x5A8
HW_ICOLL_INTERRUPT72_TOG	0x5AC

Table 5-164. HW_ICOLL_INTERRUPT72

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-165. HW_ICOLL_INTERRUPT72 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT72_SET(0,0x00000001);
```

5.4.83 Interrupt Collector Interrupt Register 73 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT73	0x5B0
HW_ICOLL_INTERRUPT73_SET	0x5B4
HW_ICOLL_INTERRUPT73_CLR	0x5B8
HW_ICOLL_INTERRUPT73_TOG	0x5BC

Table 5-166. HW_ICOLL_INTERRUPT73

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-167. HW_ICOLL_INTERRUPT73 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT73_SET(0,0x00000001);
```

5.4.84 Interrupt Collector Interrupt Register 74 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT74	0x5C0
HW_ICOLL_INTERRUPT74_SET	0x5C4
HW_ICOLL_INTERRUPT74_CLR	0x5C8
HW_ICOLL_INTERRUPT74_TOG	0x5CC

Table 5-168. HW_ICOLL_INTERRUPT74

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-169. HW_ICOLL_INTERRUPT74 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT74_SET(0,0x00000001);
```

5.4.85 Interrupt Collector Interrupt Register 75 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT75	0x5D0
HW_ICOLL_INTERRUPT75_SET	0x5D4
HW_ICOLL_INTERRUPT75_CLR	0x5D8
HW_ICOLL_INTERRUPT75_TOG	0x5DC

Table 5-170. HW_ICOLL_INTERRUPT75

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-171. HW_ICOLL_INTERRUPT75 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT75_SET(0,0x00000001);
```

5.4.86 Interrupt Collector Interrupt Register 76 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT76	0x5E0
HW_ICOLL_INTERRUPT76_SET	0x5E4
HW_ICOLL_INTERRUPT76_CLR	0x5E8
HW_ICOLL_INTERRUPT76_TOG	0x5EC

Table 5-172. HW_ICOLL_INTERRUPT76

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-173. HW_ICOLL_INTERRUPT76 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT76_SET(0,0x00000001);
```

5.4.87 Interrupt Collector Interrupt Register 77 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT77	0x5F0
HW_ICOLL_INTERRUPT77_SET	0x5F4
HW_ICOLL_INTERRUPT77_CLR	0x5F8
HW_ICOLL_INTERRUPT77_TOG	0x5FC

Table 5-174. HW_ICOLL_INTERRUPT77

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-175. HW_ICOLL_INTERRUPT77 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT77_SET(0,0x00000001);
```

5.4.88 Interrupt Collector Interrupt Register 78 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT78	0x600
HW_ICOLL_INTERRUPT78_SET	0x604
HW_ICOLL_INTERRUPT78_CLR	0x608
HW_ICOLL_INTERRUPT78_TOG	0x60C

Table 5-176. HW_ICOLL_INTERRUPT78

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-177. HW_ICOLL_INTERRUPT78 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT78_SET(0,0x00000001);
```

5.4.89 Interrupt Collector Interrupt Register 79 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT79	0x610
HW_ICOLL_INTERRUPT79_SET	0x614
HW_ICOLL_INTERRUPT79_CLR	0x618
HW_ICOLL_INTERRUPT79_TOG	0x61C

Table 5-178. HW_ICOLL_INTERRUPT79

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-179. HW_ICOLL_INTERRUPT79 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT79_SET(0,0x00000001);
```

5.4.90 Interrupt Collector Interrupt Register 80 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT80	0x620
HW_ICOLL_INTERRUPT80_SET	0x624
HW_ICOLL_INTERRUPT80_CLR	0x628
HW_ICOLL_INTERRUPT80_TOG	0x62C

Table 5-180. HW_ICOLL_INTERRUPT80

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-181. HW_ICOLL_INTERRUPT80 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT80_SET(0,0x00000001);
```

5.4.91 Interrupt Collector Interrupt Register 81 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT81	0x630
HW_ICOLL_INTERRUPT81_SET	0x634
HW_ICOLL_INTERRUPT81_CLR	0x638
HW_ICOLL_INTERRUPT81_TOG	0x63C

Table 5-182. HW_ICOLL_INTERRUPT81

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-183. HW_ICOLL_INTERRUPT81 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT81_SET(0,0x00000001);
```

5.4.92 Interrupt Collector Interrupt Register 82 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT82	0x640
HW_ICOLL_INTERRUPT82_SET	0x644
HW_ICOLL_INTERRUPT82_CLR	0x648
HW_ICOLL_INTERRUPT82_TOG	0x64C

Table 5-184. HW_ICOLL_INTERRUPT82

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-185. HW_ICOLL_INTERRUPT82 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT82_SET(0,0x00000001);
```

5.4.93 Interrupt Collector Interrupt Register 83 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT83	0x650
HW_ICOLL_INTERRUPT83_SET	0x654
HW_ICOLL_INTERRUPT83_CLR	0x658
HW_ICOLL_INTERRUPT83_TOG	0x65C

Table 5-186. HW_ICOLL_INTERRUPT83

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-187. HW_ICOLL_INTERRUPT83 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT83_SET(0,0x00000001);
```

5.4.94 Interrupt Collector Interrupt Register 84 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT84	0x660
HW_ICOLL_INTERRUPT84_SET	0x664
HW_ICOLL_INTERRUPT84_CLR	0x668
HW_ICOLL_INTERRUPT84_TOG	0x66C

Table 5-188. HW_ICOLL_INTERRUPT84

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-189. HW_ICOLL_INTERRUPT84 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT84_SET(0,0x00000001);
```

5.4.95 Interrupt Collector Interrupt Register 85 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT85	0x670
HW_ICOLL_INTERRUPT85_SET	0x674
HW_ICOLL_INTERRUPT85_CLR	0x678
HW_ICOLL_INTERRUPT85_TOG	0x67C

Table 5-190. HW_ICOLL_INTERRUPT85

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-191. HW_ICOLL_INTERRUPT85 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT85_SET(0,0x00000001);
```

5.4.96 Interrupt Collector Interrupt Register 86 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT86	0x680
HW_ICOLL_INTERRUPT86_SET	0x684
HW_ICOLL_INTERRUPT86_CLR	0x688
HW_ICOLL_INTERRUPT86_TOG	0x68C

Table 5-192. HW_ICOLL_INTERRUPT86

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-193. HW_ICOLL_INTERRUPT86 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT86_SET(0,0x00000001);
```

5.4.97 Interrupt Collector Interrupt Register 87 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT87	0x690
HW_ICOLL_INTERRUPT87_SET	0x694
HW_ICOLL_INTERRUPT87_CLR	0x698
HW_ICOLL_INTERRUPT87_TOG	0x69C

Table 5-194. HW_ICOLL_INTERRUPT87

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-195. HW_ICOLL_INTERRUPT87 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT87_SET(0,0x00000001);
```

5.4.98 Interrupt Collector Interrupt Register 88 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT88	0x6A0
HW_ICOLL_INTERRUPT88_SET	0x6A4
HW_ICOLL_INTERRUPT88_CLR	0x6A8
HW_ICOLL_INTERRUPT88_TOG	0x6AC

Table 5-196. HW_ICOLL_INTERRUPT88

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-197. HW_ICOLL_INTERRUPT88 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT88_SET(0,0x00000001);
```

5.4.99 Interrupt Collector Interrupt Register 89 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT89	0x6B0
HW_ICOLL_INTERRUPT89_SET	0x6B4
HW_ICOLL_INTERRUPT89_CLR	0x6B8
HW_ICOLL_INTERRUPT89_TOG	0x6BC

Table 5-198. HW_ICOLL_INTERRUPT89

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-199. HW_ICOLL_INTERRUPT89 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT89_SET(0,0x00000001);
```

5.4.100 Interrupt Collector Interrupt Register 90 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT90	0x6C0
HW_ICOLL_INTERRUPT90_SET	0x6C4
HW_ICOLL_INTERRUPT90_CLR	0x6C8
HW_ICOLL_INTERRUPT90_TOG	0x6CC

Table 5-200. HW_ICOLL_INTERRUPT90

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-201. HW_ICOLL_INTERRUPT90 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT90_SET(0,0x00000001);
```

5.4.101 Interrupt Collector Interrupt Register 91 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT91	0x6D0
HW_ICOLL_INTERRUPT91_SET	0x6D4
HW_ICOLL_INTERRUPT91_CLR	0x6D8
HW_ICOLL_INTERRUPT91_TOG	0x6DC

Table 5-202. HW_ICOLL_INTERRUPT91

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-203. HW_ICOLL_INTERRUPT91 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT91_SET(0,0x00000001);
```

5.4.102 Interrupt Collector Interrupt Register 92 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT92	0x6E0
HW_ICOLL_INTERRUPT92_SET	0x6E4
HW_ICOLL_INTERRUPT92_CLR	0x6E8
HW_ICOLL_INTERRUPT92_TOG	0x6EC

Table 5-204. HW_ICOLL_INTERRUPT92

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-205. HW_ICOLL_INTERRUPT92 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT92_SET(0,0x00000001);
```

5.4.103 Interrupt Collector Interrupt Register 93 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT93	0x6F0
HW_ICOLL_INTERRUPT93_SET	0x6F4
HW_ICOLL_INTERRUPT93_CLR	0x6F8
HW_ICOLL_INTERRUPT93_TOG	0x6FC

Table 5-206. HW_ICOLL_INTERRUPT93

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-207. HW_ICOLL_INTERRUPT93 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT93_SET(0,0x00000001);
```

5.4.104 Interrupt Collector Interrupt Register 94 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT94	0x700
HW_ICOLL_INTERRUPT94_SET	0x704
HW_ICOLL_INTERRUPT94_CLR	0x708
HW_ICOLL_INTERRUPT94_TOG	0x70C

Table 5-208. HW_ICOLL_INTERRUPT94

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-209. HW_ICOLL_INTERRUPT94 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT94_SET(0,0x00000001);
```

5.4.105 Interrupt Collector Interrupt Register 95 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT95	0x710
HW_ICOLL_INTERRUPT95_SET	0x714
HW_ICOLL_INTERRUPT95_CLR	0x718
HW_ICOLL_INTERRUPT95_TOG	0x71C

Table 5-210. HW_ICOLL_INTERRUPT95

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-211. HW_ICOLL_INTERRUPT95 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT95_SET(0,0x00000001);
```

5.4.106 Interrupt Collector Interrupt Register 96 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT96	0x720
HW_ICOLL_INTERRUPT96_SET	0x724
HW_ICOLL_INTERRUPT96_CLR	0x728
HW_ICOLL_INTERRUPT96_TOG	0x72C

Table 5-212. HW_ICOLL_INTERRUPT96

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-213. HW_ICOLL_INTERRUPT96 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT96_SET(0,0x00000001);
```

5.4.107 Interrupt Collector Interrupt Register 97 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT97	0x730
HW_ICOLL_INTERRUPT97_SET	0x734
HW_ICOLL_INTERRUPT97_CLR	0x738
HW_ICOLL_INTERRUPT97_TOG	0x73C

Table 5-214. HW_ICOLL_INTERRUPT97

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-215. HW_ICOLL_INTERRUPT97 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT97_SET(0,0x00000001);
```

5.4.108 Interrupt Collector Interrupt Register 98 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT98	0x740
HW_ICOLL_INTERRUPT98_SET	0x744
HW_ICOLL_INTERRUPT98_CLR	0x748
HW_ICOLL_INTERRUPT98_TOG	0x74C

Table 5-216. HW_ICOLL_INTERRUPT98

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-217. HW_ICOLL_INTERRUPT98 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT98_SET(0,0x00000001);
```

5.4.109 Interrupt Collector Interrupt Register 99 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT99	0x750
HW_ICOLL_INTERRUPT99_SET	0x754
HW_ICOLL_INTERRUPT99_CLR	0x758
HW_ICOLL_INTERRUPT99_TOG	0x75C

Table 5-218. HW_ICOLL_INTERRUPT99

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-219. HW_ICOLL_INTERRUPT99 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT99_SET(0,0x00000001);
```

5.4.110 Interrupt Collector Interrupt Register 100 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT100	0x760
HW_ICOLL_INTERRUPT100_SET	0x764
HW_ICOLL_INTERRUPT100_CLR	0x768
HW_ICOLL_INTERRUPT100_TOG	0x76C

Table 5-220. HW_ICOLL_INTERRUPT100

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-221. HW_ICOLL_INTERRUPT100 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT100_SET(0,0x00000001);
```

5.4.111 Interrupt Collector Interrupt Register 101 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT101	0x770
HW_ICOLL_INTERRUPT101_SET	0x774
HW_ICOLL_INTERRUPT101_CLR	0x778
HW_ICOLL_INTERRUPT101_TOG	0x77C

Table 5-222. HW_ICOLL_INTERRUPT101

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-223. HW_ICOLL_INTERRUPT101 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT101_SET(0,0x00000001);
```

5.4.112 Interrupt Collector Interrupt Register 102 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT102	0x780
HW_ICOLL_INTERRUPT102_SET	0x784
HW_ICOLL_INTERRUPT102_CLR	0x788
HW_ICOLL_INTERRUPT102_TOG	0x78C

Table 5-224. HW_ICOLL_INTERRUPT102

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-225. HW_ICOLL_INTERRUPT102 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT102_SET(0,0x00000001);
```

5.4.113 Interrupt Collector Interrupt Register 103 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT103	0x790
HW_ICOLL_INTERRUPT103_SET	0x794
HW_ICOLL_INTERRUPT103_CLR	0x798
HW_ICOLL_INTERRUPT103_TOG	0x79C

Table 5-226. HW_ICOLL_INTERRUPT103

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-227. HW_ICOLL_INTERRUPT103 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT103_SET(0,0x00000001);
```

5.4.114 Interrupt Collector Interrupt Register 104 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT104	0x7A0
HW_ICOLL_INTERRUPT104_SET	0x7A4
HW_ICOLL_INTERRUPT104_CLR	0x7A8
HW_ICOLL_INTERRUPT104_TOG	0x7AC

Table 5-228. HW_ICOLL_INTERRUPT104

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-229. HW_ICOLL_INTERRUPT104 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT104_SET(0,0x00000001);
```

5.4.115 Interrupt Collector Interrupt Register 105 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT105	0x7B0
HW_ICOLL_INTERRUPT105_SET	0x7B4
HW_ICOLL_INTERRUPT105_CLR	0x7B8
HW_ICOLL_INTERRUPT105_TOG	0x7BC

Table 5-230. HW_ICOLL_INTERRUPT105

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-231. HW_ICOLL_INTERRUPT105 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT105_SET(0,0x00000001);
```

5.4.116 Interrupt Collector Interrupt Register 106 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT106	0x7C0
HW_ICOLL_INTERRUPT106_SET	0x7C4
HW_ICOLL_INTERRUPT106_CLR	0x7C8
HW_ICOLL_INTERRUPT106_TOG	0x7CC

Table 5-232. HW_ICOLL_INTERRUPT106

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-233. HW_ICOLL_INTERRUPT106 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT106_SET(0,0x00000001);
```

5.4.117 Interrupt Collector Interrupt Register 107 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT107	0x7D0
HW_ICOLL_INTERRUPT107_SET	0x7D4
HW_ICOLL_INTERRUPT107_CLR	0x7D8
HW_ICOLL_INTERRUPT107_TOG	0x7DC

Table 5-234. HW_ICOLL_INTERRUPT107

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-235. HW_ICOLL_INTERRUPT107 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT107_SET(0,0x00000001);
```

5.4.118 Interrupt Collector Interrupt Register 108 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT108	0x7E0
HW_ICOLL_INTERRUPT108_SET	0x7E4
HW_ICOLL_INTERRUPT108_CLR	0x7E8
HW_ICOLL_INTERRUPT108_TOG	0x7EC

Table 5-236. HW_ICOLL_INTERRUPT108

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-237. HW_ICOLL_INTERRUPT108 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT108_SET(0,0x00000001);
```

5.4.119 Interrupt Collector Interrupt Register 109 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT109	0x7F0
HW_ICOLL_INTERRUPT109_SET	0x7F4
HW_ICOLL_INTERRUPT109_CLR	0x7F8
HW_ICOLL_INTERRUPT109_TOG	0x7FC

Table 5-238. HW_ICOLL_INTERRUPT109

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-239. HW_ICOLL_INTERRUPT109 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT109_SET(0,0x00000001);
```

5.4.120 Interrupt Collector Interrupt Register 110 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT110	0x800
HW_ICOLL_INTERRUPT110_SET	0x804
HW_ICOLL_INTERRUPT110_CLR	0x808
HW_ICOLL_INTERRUPT110_TOG	0x80C

Table 5-240. HW_ICOLL_INTERRUPT110

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-241. HW_ICOLL_INTERRUPT110 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT110_SET(0,0x00000001);
```

5.4.121 Interrupt Collector Interrupt Register 111 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT111	0x810
HW_ICOLL_INTERRUPT111_SET	0x814
HW_ICOLL_INTERRUPT111_CLR	0x818
HW_ICOLL_INTERRUPT111_TOG	0x81C

Table 5-242. HW_ICOLL_INTERRUPT111

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-243. HW_ICOLL_INTERRUPT111 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT111_SET(0,0x00000001);
```

5.4.122 Interrupt Collector Interrupt Register 112 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT112	0x820
HW_ICOLL_INTERRUPT112_SET	0x824
HW_ICOLL_INTERRUPT112_CLR	0x828
HW_ICOLL_INTERRUPT112_TOG	0x82C

Table 5-244. HW_ICOLL_INTERRUPT112

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-245. HW_ICOLL_INTERRUPT112 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT112_SET(0,0x00000001);
```

5.4.123 Interrupt Collector Interrupt Register 113 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT113	0x830
HW_ICOLL_INTERRUPT113_SET	0x834
HW_ICOLL_INTERRUPT113_CLR	0x838
HW_ICOLL_INTERRUPT113_TOG	0x83C

Table 5-246. HW_ICOLL_INTERRUPT113

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-247. HW_ICOLL_INTERRUPT113 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT113_SET(0,0x00000001);
```

5.4.124 Interrupt Collector Interrupt Register 114 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT114	0x840
HW_ICOLL_INTERRUPT114_SET	0x844
HW_ICOLL_INTERRUPT114_CLR	0x848
HW_ICOLL_INTERRUPT114_TOG	0x84C

Table 5-248. HW_ICOLL_INTERRUPT114

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-249. HW_ICOLL_INTERRUPT114 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT114_SET(0,0x00000001);
```

5.4.125 Interrupt Collector Interrupt Register 115 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT115	0x850
HW_ICOLL_INTERRUPT115_SET	0x854
HW_ICOLL_INTERRUPT115_CLR	0x858
HW_ICOLL_INTERRUPT115_TOG	0x85C

Table 5-250. HW_ICOLL_INTERRUPT115

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-251. HW_ICOLL_INTERRUPT115 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT115_SET(0,0x00000001);
```

5.4.126 Interrupt Collector Interrupt Register 116 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT116	0x860
HW_ICOLL_INTERRUPT116_SET	0x864
HW_ICOLL_INTERRUPT116_CLR	0x868
HW_ICOLL_INTERRUPT116_TOG	0x86C

Table 5-252. HW_ICOLL_INTERRUPT116

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-253. HW_ICOLL_INTERRUPT116 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT116_SET(0,0x00000001);
```

5.4.127 Interrupt Collector Interrupt Register 117 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT117	0x870
HW_ICOLL_INTERRUPT117_SET	0x874
HW_ICOLL_INTERRUPT117_CLR	0x878
HW_ICOLL_INTERRUPT117_TOG	0x87C

Table 5-254. HW_ICOLL_INTERRUPT117

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-255. HW_ICOLL_INTERRUPT117 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT117_SET(0,0x00000001);
```

5.4.128 Interrupt Collector Interrupt Register 118 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT118	0x880
HW_ICOLL_INTERRUPT118_SET	0x884
HW_ICOLL_INTERRUPT118_CLR	0x888
HW_ICOLL_INTERRUPT118_TOG	0x88C

Table 5-256. HW_ICOLL_INTERRUPT118

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-257. HW_ICOLL_INTERRUPT118 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT118_SET(0,0x00000001);
```

5.4.129 Interrupt Collector Interrupt Register 119 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT119	0x890
HW_ICOLL_INTERRUPT119_SET	0x894
HW_ICOLL_INTERRUPT119_CLR	0x898
HW_ICOLL_INTERRUPT119_TOG	0x89C

Table 5-258. HW_ICOLL_INTERRUPT119

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-259. HW_ICOLL_INTERRUPT119 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT119_SET(0,0x00000001);
```

5.4.130 Interrupt Collector Interrupt Register 120 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT120	0x8A0
HW_ICOLL_INTERRUPT120_SET	0x8A4
HW_ICOLL_INTERRUPT120_CLR	0x8A8
HW_ICOLL_INTERRUPT120_TOG	0x8AC

Table 5-260. HW_ICOLL_INTERRUPT120

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-261. HW_ICOLL_INTERRUPT120 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT120_SET(0,0x00000001);
```

5.4.131 Interrupt Collector Interrupt Register 121 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT121	0x8B0
HW_ICOLL_INTERRUPT121_SET	0x8B4
HW_ICOLL_INTERRUPT121_CLR	0x8B8
HW_ICOLL_INTERRUPT121_TOG	0x8BC

Table 5-262. HW_ICOLL_INTERRUPT121

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-263. HW_ICOLL_INTERRUPT121 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT121_SET(0,0x00000001);
```

5.4.132 Interrupt Collector Interrupt Register 122 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT122	0x8C0
HW_ICOLL_INTERRUPT122_SET	0x8C4
HW_ICOLL_INTERRUPT122_CLR	0x8C8
HW_ICOLL_INTERRUPT122_TOG	0x8CC

Table 5-264. HW_ICOLL_INTERRUPT122

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-265. HW_ICOLL_INTERRUPT122 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT122_SET(0,0x00000001);
```

5.4.133 Interrupt Collector Interrupt Register 123 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT123	0x8D0
HW_ICOLL_INTERRUPT123_SET	0x8D4
HW_ICOLL_INTERRUPT123_CLR	0x8D8
HW_ICOLL_INTERRUPT123_TOG	0x8DC

Table 5-266. HW_ICOLL_INTERRUPT123

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-267. HW_ICOLL_INTERRUPT123 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT123_SET(0,0x00000001);
```

5.4.134 Interrupt Collector Interrupt Register 124 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT124	0x8E0
HW_ICOLL_INTERRUPT124_SET	0x8E4
HW_ICOLL_INTERRUPT124_CLR	0x8E8
HW_ICOLL_INTERRUPT124_TOG	0x8EC

Table 5-268. HW_ICOLL_INTERRUPT124

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-269. HW_ICOLL_INTERRUPT124 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT124_SET(0,0x00000001);
```

5.4.135 Interrupt Collector Interrupt Register 125 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT125	0x8F0
HW_ICOLL_INTERRUPT125_SET	0x8F4
HW_ICOLL_INTERRUPT125_CLR	0x8F8
HW_ICOLL_INTERRUPT125_TOG	0x8FC

Table 5-270. HW_ICOLL_INTERRUPT125

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-271. HW_ICOLL_INTERRUPT125 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT125_SET(0,0x00000001);
```

5.4.136 Interrupt Collector Interrupt Register 126 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT126	0x900
HW_ICOLL_INTERRUPT126_SET	0x904
HW_ICOLL_INTERRUPT126_CLR	0x908
HW_ICOLL_INTERRUPT126_TOG	0x90C

Table 5-272. HW_ICOLL_INTERRUPT126

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-273. HW_ICOLL_INTERRUPT126 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT126_SET(0,0x00000001);
```

5.4.137 Interrupt Collector Interrupt Register 127 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT127	0x910
HW_ICOLL_INTERRUPT127_SET	0x914
HW_ICOLL_INTERRUPT127_CLR	0x918
HW_ICOLL_INTERRUPT127_TOG	0x91C

Table 5-274. HW_ICOLL_INTERRUPT127

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

Table 5-275. HW_ICOLL_INTERRUPT127 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT127_SET(0,0x00000001);
```

5.4.138 Interrupt Collector Debug Register 0 Description

The contents of this register will be defined as the hardware is developed.

HW_ICOLL_DEBUG	0x1120
HW_ICOLL_DEBUG_SET	0x1124
HW_ICOLL_DEBUG_CLR	0x1128
HW_ICOLL_DEBUG_TOG	0x112C

Table 5-276. HW_ICOLL_DEBUG

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
INSERVICE				LEVEL_REQUESTS				REQUESTS_BY_LEVEL				RSRVD2		FIQ	IRQ	RSRVD1						VECTOR_FSM									

Table 5-277. HW_ICOLL_DEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**INSERVICE**	RO	0x0	read-only view of the Inservice bits used for nesting IRQs. LEVEL0 = 0x1 LEVEL0 LEVEL1 = 0x2 LEVEL1 LEVEL2 = 0x4 LEVEL2 LEVEL3 = 0x8 LEVEL3
27:24	**LEVEL_REQUESTS**	RO	0x0	read-only view of the requsts by priority level for the current IRQ. LEVEL0 = 0x1 LEVEL0 LEVEL1 = 0x2 LEVEL1 LEVEL2 = 0x4 LEVEL2 LEVEL3 = 0x8 LEVEL3
23:20	**REQUESTS_BY_LEVEL**	RO	0x0	read-only view of the requsts by priority level for the current IRQ. LEVEL0 = 0x1 LEVEL0 LEVEL1 = 0x2 LEVEL1 LEVEL2 = 0x4 LEVEL2 LEVEL3 = 0x8 LEVEL3
19:18	**RSRVD2**	RO	0x0	Always write zeroes to this bitfield.
17	**FIQ**	RO	0x0	Read-Only View of the FIQ output to the CPU. NO_FIQ_REQUESTED = 0x0 No FIQ Requested FIQ_REQUESTED = 0x1 FIQ Requested
16	**IRQ**	RO	0x0	Read-Only View of the FIQ output to the CPU. NO_IRQ_REQUESTED = 0x0 No IRQ Requested IRQ_REQUESTED = 0x1 IRQ Requested
15:10	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
9:0	**VECTOR_FSM**	RO	0x0	Empty description. FSM_IDLE = 0x000 FSM_IDLE FSM_MULTICYCLE1 = 0x001 FSM_MULTICYCLE1 FSM_MULTICYCLE2 = 0x002 FSM_MULTICYCLE2 FSM_PENDING = 0x004 FSM_PENDING FSM_MULTICYCLE3 = 0x008 FSM_MULTICYCLE3 FSM_MULTICYCLE4 = 0x010 FSM_MULTICYCLE4 FSM_ISR_RUNNING1 = 0x020 FSM_ISR_RUNNING1 FSM_ISR_RUNNING2 = 0x040 FSM_ISR_RUNNING2 FSM_ISR_RUNNING3 = 0x080 FSM_ISR_RUNNING3 FSM_MULTICYCLE5 = 0x100 FSM_MULTICYCLE5 FSM_MULTICYCLE6 = 0x200 FSM_MULTICYCLE6

DESCRIPTION:

This register provides diagnostic visibility into the IRQ request state machine and its various inputs.

EXAMPLE:

```
if (BF_RD(ICOLL_DEBUG, LEVEL_REQUESTS) != HW_ICOLL_DEBUG_LEVEL_REQUESTS__LEVEL3)
Error();
TPRINTF(TP_MED, ("ICOLL INSERVICE = 0x%x
BF_RD(ICOLL_DEBUG, INSERVICE)));
TPRINTF(TP_MED, ("ICOLL STATE = 0x%x
VECTOR_FSM)));
```

5.4.139 Interrupt Collector Debug Read Register 0 Description

This register always returns a known read value for debug purposes.

HW_ICOLL_DBGREAD0	0x1130
HW_ICOLL_DBGREAD0_SET	0x1134
HW_ICOLL_DBGREAD0_CLR	0x1138
HW_ICOLL_DBGREAD0_TOG	0x113C

Table 5-278. HW_ICOLL_DBGREAD0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
VALUE																															

Table 5-279. HW_ICOLL_DBGREAD0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**VALUE**	RO	0xECA94567	Fixed read-only value.

DESCRIPTION:

This register is used to test the read mux paths on the APBH.

EXAMPLE:

```
if (HW_ICOLL_DBGREAD0_RD != 0xECA94567)
Error();
```

5.4.140 Interrupt Collector Debug Read Register 1 Description

This register always returns a known read value for debug purposes.

HW_ICOLL_DBGREAD1	0x1140
HW_ICOLL_DBGREAD1_SET	0x1144
HW_ICOLL_DBGREAD1_CLR	0x1148
HW_ICOLL_DBGREAD1_TOG	0x114C

Table 5-280. HW_ICOLL_DBGREAD1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
VALUE																															

Table 5-281. HW_ICOLL_DBGREAD1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**VALUE**	RO	0x1356DA98	Fixed read-only value.

DESCRIPTION:

This register is used to test the read mux paths on the APBH.

EXAMPLE:

```
if (HW_ICOLL_DBGREAD1_RD != 0x1356DA98)
Error();
```

5.4.141 Interrupt Collector Debug Flag Register Description

The Interrupt Collector debug flag register is used to post diagnostic state into simulation.

HW_ICOLL_DBGFLAG	0x1150
HW_ICOLL_DBGFLAG_SET	0x1154

HW_ICOLL_DBGFLAG_CLR	0x1158
HW_ICOLL_DBGFLAG_TOG	0x115C

Table 5-282. HW_ICOLL_DBGFLAG

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																FLAG															

Table 5-283. HW_ICOLL_DBGFLAG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
15:0	**FLAG**	RW	0x0	This debug facility is probably temporary.

DESCRIPTION:

This register provides a posting register to synchronize C program execution and the internal simulation environment.

EXAMPLE:

```
BF_WR(ICOLL_DBGFLAG, FLAG, 3);
// ... do some diagnostic action
BF_WR(ICOLL_DBGFLAG, FLAG, 4);
// ... do some more diagnostic actions
BF_WR(ICOLL_DBGFLAG, FLAG, 5);
```

5.4.142 Interrupt Collector Debug Read Request Register 0 Description

read-only view into the low 32 bits of the request holding register.

HW_ICOLL_DBGREQUEST0	0x1160
HW_ICOLL_DBGREQUEST0_SET	0x1164
HW_ICOLL_DBGREQUEST0_CLR	0x1168
HW_ICOLL_DBGREQUEST0_TOG	0x116C

Table 5-284. HW_ICOLL_DBGREQUEST0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 5-285. HW_ICOLL_DBGREQUEST0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Low 32 bits of the request holding register.

DESCRIPTION:

This register is used to test interrupt collector state machine and its associated request holding register.

EXAMPLE:

```
if (HW_ICOLL_DBGREQUESTn_RD(0) != 0x00000000)
Error();
```

5.4.143 Interrupt Collector Debug Read Request Register 1 Description

read-only view into bits 32-63 of the request holding register.

HW_ICOLL_DBGREQUEST1	0x1170
HW_ICOLL_DBGREQUEST1_SET	0x1174
HW_ICOLL_DBGREQUEST1_CLR	0x1178
HW_ICOLL_DBGREQUEST1_TOG	0x117C

Table 5-286. HW_ICOLL_DBGREQUEST1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 5-287. HW_ICOLL_DBGREQUEST1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Bits 32-63 of the request holding register.

DESCRIPTION:

This register is used to test interrupt collector state machine and its associated request holding register.

EXAMPLE:

```
if (HW_ICOLL_DBGREQUESTn_RD(n) != 0x00000000)
Error();
```

5.4.144 Interrupt Collector Debug Read Request Register 2 Description

read-only view into bits 64-95 of the request holding register.

HW_ICOLL_DBGREQUEST2	0x1180
HW_ICOLL_DBGREQUEST2_SET	0x1184
HW_ICOLL_DBGREQUEST2_CLR	0x1188
HW_ICOLL_DBGREQUEST2_TOG	0x118C

Table 5-288. HW_ICOLL_DBGREQUEST2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 5-289. HW_ICOLL_DBGREQUEST2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Bits 64-95 of the request holding register.

DESCRIPTION:

This register is used to test interrupt collector state machine and its associated request holding register.

EXAMPLE:

```
if (HW_ICOLL_DBGREQUESTn_RD(n) != 0x00000000)
Error();
```

5.4.145 Interrupt Collector Debug Read Request Register 3 Description

read-only view into bits 96-127 of the request holding register.

HW_ICOLL_DBGREQUEST3	0x1190
HW_ICOLL_DBGREQUEST3_SET	0x1194
HW_ICOLL_DBGREQUEST3_CLR	0x1198
HW_ICOLL_DBGREQUEST3_TOG	0x119C

Table 5-290. HW_ICOLL_DBGREQUEST3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 5-291. HW_ICOLL_DBGREQUEST3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Bits 96-127 of the request holding register.

DESCRIPTION:

This register is used to test interrupt collector state machine and its associated request holding register.

EXAMPLE:

```
if (HW_ICOLL_DBGREQUESTn_RD(n) != 0x00000000)
Error();
```

5.4.146 Interrupt Collector Version Register Description

This register always returns a known read value for debug purposes it indicates the version of the block.

HW_ICOLL_VERSION	0x11E0

Table 5-292. HW_ICOLL_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 5-293. HW_ICOLL_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x03	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x01	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
if (HW_ICOLL_VERSION.B.MAJOR != 3)
Error();
```

ICOLL Block v3.1, Revision 1.50

5.4.147

Chapter 6
Digital Control and On-Chip RAM

This chapter describes the digital control block and the on-chip RAM features of the i.MX23. It includes sections on controlling the SRAM, performance monitors, high-entropy pseudo-random number seed, and free-running microseconds counter. Programmable registers for the block are described in Section 6.4, “Programmable Registers.”

6.1 Overview

The digital control block provides overall control of various items within the top digital block of the chip, including the on-chip RAM controls and HCLK performance counter, as shown in Figure 6-1.

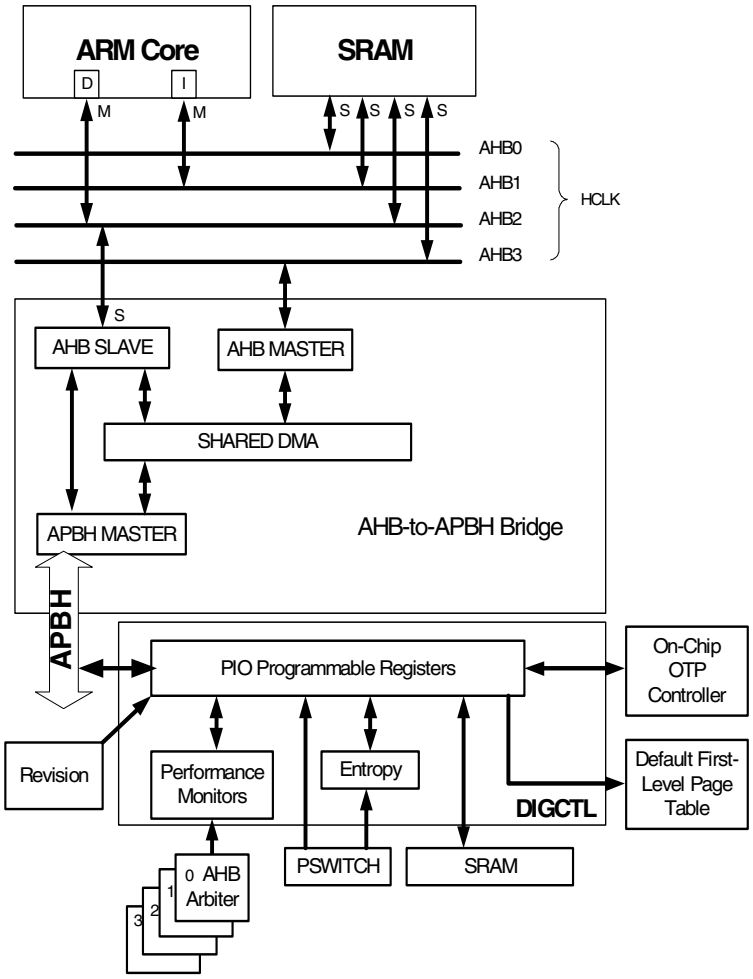


Figure 6-1. Digital Control (DIGCTL) Block Diagram

The on-chip RAM is constructed from an array of six-transistor dynamic RAM bit cells. The repair functions of this SRAM are controlled by registers in the DIGCTL block.

6.2 SRAM Controls

The on-chip RAM is a compiled RAM cell. It is implemented in one segment of 32 Kbytes. The memory is addressed as shown in Figure 6-2.

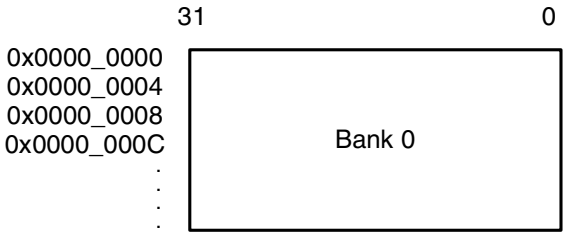


Figure 6-2. On-Chip RAM Partitioning

A 32-bit AHB address is arranged as shown in Table 6-1.

Table 6-1. On-Chip RAM Address Bits

AHB ADDR BITS	USAGE	DESCRIPTION
29:2	Address	Selects one of 32K words in a bank.
-	Bank Address	Tied 2'b00.
1:0	Byte Address	Selects/masks out specific bytes within a word.

Accessing on-chip RAM requires only one initial wait state for arbitration. There is a single cycle access for writes, and two cycle (one wait state) for reads. Other wait states happen whenever there is a read immediately following a write and also when a master is waiting to access a bank because some other master is accessing the same bank on a cycle.

The i.MX23 contains a simple 32-bit word RAM repair scheme. The purpose of this scheme is to address single-bit errors in the on-chip RAM. To enable the repair, HW_DIGCTL_RAMCTRL_RAM_REPAIR_EN must be set. Once this bit is set, the RAM controller replaces any access to the word address specified in HW_DIGCTL_RAMREPAIR_ADDR with a 32-bit redundant hardware memory. The actual repair enable and address must be read from OTP. Because these registers must be loaded by software, ROM boot code reads the OTP to see if the repair enable bit is set. If the repair enable bit is set, the ROM boot code sets HW_DIGCTL_RAMCTRL_RAM_REPAIR_EN and copies the 16-bit word address from OTP to HW_DIGCTL_RAMREPAIR_ADDR.

6.3 Miscellaneous Controls

The digital control block also contains a number of other miscellaneous functions, as detailed in this section.

6.3.1 Performance Monitoring

The digital control block contains several registers for system bus performance monitoring, including HW_DIGCTL_HCLKCOUNT, which counts HCLK rising edges. This register counts at a variable rate as HW_CLKCTRL_HBUS_AUTO_SLOW_ MODE is enabled.

In addition, there exists a performance monitoring register for each AHB layer (L0–L3). The HW_DIGCTL_L(n)_AHB_DATA_STALLED and HW_DIGCTL_L(n)_AHB_ACTIVE_CYCLES registers can be used to measure AHB bus utilization. The Stalled register counts all cycles in which any device has an outstanding and unfulfilled bus operation in flight. The Active Cycles register counts the number of data transfer cycles. Subtract cycles from stalls to determine under utilized bus cycles. These counters can be used to tune the performance of the HCLK frequency for specific activities. In addition, these monitors can be forced to focus on specific masters (which connect to that layer). See the HW_DIGCTL_AHB_STATS_SELECT bit description for details.

6.3.2 High-Entropy PRN Seed

A 32-bit entropy register begins running a pseudo-random number algorithm from the time reset is removed until the PSWITCH is released by the user. This high-entropy value can be used as the seed for other pseudo-random number generators.

6.3.3 Write-Once Register

A 32-bit write-once register holds a runtime-derived locked seed. Once written, it cannot be changed until the next chip wide reset event. The contents of this register are frequently derived from the entropy register.

6.3.4 Microseconds Counter

A 32-bit free-running microseconds counter provides fine-grain real-time control. Its period is determined by dividing the 24.0-MHz crystal oscillator by 24. Thus, its frequency does not change as HCLK, XCLK, and the processor clock frequency are changed.

6.4 Programmable Registers

The following registers provide control of all programmable elements of the digital control block.

6.4.1 DIGCTL Control Register Description

The DIGCTL Control Register provides overall control of various functions throughout the digital portion of the chip.

HW_DIGCTL_CTRL	0x000
HW_DIGCTL_CTRL_SET	0x004
HW_DIGCTL_CTRL_CLR	0x008
HW_DIGCTL_CTRL_TOG	0x00C

Table 6-2. HW_DIGCTL_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD3	XTAL24M_GATE	TRAP_IRQ	RSVD2		CACHE_BIST_TMODE	LCD_BIST_CLKEN	LCD_BIST_START	DCP_BIST_CLKEN	DCP_BIST_START	ARM_BIST_CLKEN	USB_TESTMODE	ANALOG_TESTMODE	DIGITAL_TESTMODE	ARM_BIST_START	UART_LOOPBACK	SAIF_LOOPBACK	SAIF_CLKMUX_SEL		SAIF_CLKMST_SEL	SAIF_ALT_BITCLK_SEL	RSVD1	SY_ENDIAN	SY_SFTRST	SY_CLKGATE	USE_SERIAL_JTAG	TRAP_IN_RANGE	TRAP_ENABLE	DEBUG_DISABLE	USB_CLKGATE	JTAG_SHIELD	LATCH_ENTROPY

Table 6-3. HW_DIGCTL_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSVD3**	RO	0x0	Reserved.
30	**XTAL24M_GATE**	RW	0x0	If set to 1, disable the Digital Control Microseconds counter, STRB1MHZ. If set to 0, enable the Digital Control Microseconds counter..
29	**TRAP_IRQ**	RW	0x0	This bit is set when an AHB access occurs to the range defined by the TRAP_ADDR registers below and the trap function is enabled with the TRAP_ENABLE bit.
28:27	**RSVD2**	RO	0x0	Reserved.
26	**CACHE_BIST_TMODE**	RW	0x0	Set this bit to enable the Cache BIST test mode.
25	**LCD_BIST_CLKEN**	RW	0x0	Set this bit to enable the LCD memory BIST clock.
24	**LCD_BIST_START**	RW	0x0	Set this bit to start the LCD memory BIST.
23	**DCP_BIST_CLKEN**	RW	0x0	Set this bit to enable the DCP memory BIST clock.
22	**DCP_BIST_START**	RW	0x0	Set this bit to start the DCP memory BIST.
21	**ARM_BIST_CLKEN**	RW	0x0	Set this bit to enable the ARM BIST clock.
20	**USB_TESTMODE**	RW	0x0	Set this bit to get into USB test mode.
19	**ANALOG_TESTMODE**	RW	0x0	Set this bit to get into analog test mode.
18	**DIGITAL_TESTMODE**	RW	0x0	Set this bit to get into digital test mode.
17	**ARM_BIST_START**	RW	0x0	Set this bit to start the ARM cache BIST controller.
16	**UART_LOOPBACK**	RW	0x0	Set this bit to loop the two AUARTs back on themselves in a null modem configuration (as well as connect AUART1 to DUART). NORMAL = 0x0 No loopback. LOOPIT = 0x1 Internally connect AUART1 TX to AUART2 RX and DUART RX, also connect AUART2 TX to AUART1 RX (note that DUART TX is unaffected).
15	**SAIF_LOOPBACK**	RW	0x0	Set this bit to loop SAIF1 to SAIF2 and SAIF2 to SAIF1. To use SAIF loopback, configure one SAIF for transmit and the other for receive. Because this bit connects SAIF1 output to SAIF2 input and SAIF2 output to SAIF1 input, it does not matter which of the two ports is configured for TX and the other for RX. Either configuration will produce an internal TX to RX loopback. Note that SAIF_CLKMST_SEL is ignored when loopback is enabled. NORMAL = 0x0 No loopback. LOOPIT = 0x1 Loop SAIF1 and SAIF2 back to each other.

Table 6-3. HW_DIGCTL_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
14:13	**SAIF_CLKMUX_SEL**	RW	0x0	Selects the muxed pins and directions for the SAIF1 and SAIF2 master clock (MCLK), bit clock (BITCLK), and left/right sample clock (LRCLK). MCLK is an optional output and when used, is connected to the SAIF_MCLK_BITCLK muxed pin output. BITCLK can be either an input or output and can be connected to either the SAIF_MCLK_BITCLK muxed pin or the SAIF_ALT_BITCLK muxed pin. LRCLK can also be either an input or output and is connected to the SAIF_LRCLK muxed pin. When either MCLK, MCLK/BITCLK, or MCLK/BITCLK/LRCLK are configured to be outputs, the SAIF_CLKMST_SEL bit is used to determine which of the two SAIFs drives these clocks. Note that only one of the two SAIFs can be clock master at a time (READ_MODE=0 and/or SLAVE_MODE=0). When MCLK is not used and BITCLK/LRCLK are both inputs, one or both SAIFs must be configured as RX clock slaves (READ_MODE=1 and SLAVE_MODE=1). Valid configurations for SAIF_CLKMUX_SEL, as well as SAIF1 and SAIF2 SLAVE_MODE and READ_MODE bits, are: 1) one SAIF port is TX or RX master and controls BITCLK/LRCLK (both to the pins and to the other SAIF), and MCLK can optionally be an output while the other SAIF is an RX slave; 2) both ports are in RX slave mode and are driven by the BITCLK/LRCLK pin inputs, and MCLK can optionally be an output; 3) only one SAIF port is used as a TX or RX master during a given time, and SAIF_CLKMST_SEL is configured to give control of MCLK/BITCLK/LRCLK to the active port; or 4) only one of the two ports is used as an RX slave. See the table earlier in this chapter for a complete list of SAIF_CLKMUX_SEL/SAIF_CLKMST_SEL options, as well as SAIF1 and SAIF2 SLAVE_MODE/READ_MODE configurations. Note that SAIF_CLKMUX_SEL is ignored when SAIF_LOOPBACK=1. Also note that when the SAIF_ALT_BITCLK pinmux is selected to input/output BITCLK, 6-channel mode cannot be used since the SAIF2_SDATA2 pin is used for the alternate BITCLK. MBL_CLK_OUT = 0x0 MCLK output to SAIF_MCLK_BITCLK, BITCLK output to SAIF_ALT_BITCLK, LRCLK output to SAIF_LRCLK. BL_CLK_OUT = 0x1 BITCLK output to SAIF_MCLK_BITCLK, LRCLK output to SAIF_LRCLK. M_CLK_OUT_BL_CLK_IN = 0x2 MCLK output to SAIF_MCLK_BITCLK, BITCLK input to SAIF_ALT_BITCLK, LRCLK input to SAIF_LRCLK. BL_CLK_IN = 0x3 BITCLK input to SAIF_MCLK_BITCLK, LRCLK input to SAIF_LRCLK.

Table 6-3. HW_DIGCTL_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
12	**SAIF_CLKMST_SEL**	RW	0x0	Selects whether SAIF1 or SAIF2 drives MCLK/BITCLK/LRCLK when they are configured as outputs via SAIF_CLKMUX_SEL. Note that the selected SAIF must be configured in clock master mode (READ_MODE=0 and/or SLAVE_MODE=0). This bit must also be configured when SAIF_LOOPBACK=1 to determine the clock master. SAIF1_MST = 0x0 SAIF1 clocks are used to output MCLK/BITCLK/LRCLK. SAIF2_MST = 0x1 SAIF2 clocks are used to output MCLK/BITCLK/LRCLK.
11	**SAIF_ALT_BITCLK_SEL**	RW	0x0	When the master SAIF (as selected by SAIF_CLKMST_SEL) requires all three clocks (MCLK/BITCLK/LRCLK), which is selected by programming SAIF_CLKMUX_SEL = 00 or 10, this bit selects whether SAIF2_SDATA2 or LCD_D16 is used to input/output BITCLK. 0 = SAIF2_SDATA2 pinmux pin selected for BITCLK. 1 = LCD_D16 pin selected for BITCLK. Note that this bit is ignored when SAIF_CLKMUX_SEL=01 or 11. Also note that the corresponding pin selected for BITCLK must have its MUXXEL bit field correctly programmed in the pin control block.
10	**RSVD1**	RO	0x0	Reserved.
9	**SY_ENDIAN**	RW	0x1	Setting this bit to 1 configures the SY to run in big endian mode, clearing this bit to 0 configures the SY to run in little endian mode.
8	**SY_SFTRST**	RW	0x1	Setting this bit to 1 forces a reset to the entire SY. SY_SFTRST has no effect on SY_CLKGATE. Also, the SY_SFTRST bit may be written when SY_CLKGATE=1. This bit must be cleared to 0 for normal SY operation.
7	**SY_CLKGATE**	RW	0x1	This bit gates the clocks to the SY to save power when the clocks are not in use. When set to 1, this bit gates off the clocks to the block. When this bit is cleared to 0, the block receives its clock for normal operation.
6	**USE_SERIAL_JTAG**	RW	0x0	Selects whether the one-wire serial JTAG interface or the alternative six-wire parallel JTAG interface is used. 0 = Parallel six-wire JTAG is enabled and is mapped to a collection of module pins that must be enabled by programming their MUXSEL bits in the pin control block. 1 = Serial JTAG is enabled and uses the dedicated DEBUG pin. The ROM bootcode writes this field prior to enabling JTAG, selecting which type of JTAG pin signaling to use. OLD_JTAG = 0x0 Use six-wire parallel JTAG mode. SERIAL_JTAG = 0x1 Use one-wire serial JTAG mode.

Table 6-3. HW_DIGCTL_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
5	**TRAP_IN_RANGE**	RW	0x0	Determines whether the debug trap function causes a match when the master address is inside (low-address <= current-address <= high-address) the specified range. 0 = The trap occurs when the master address falls outside of the range. 1 = The check is inside the range.
4	**TRAP_ENABLE**	RW	0x0	Enables the AHB arbiter debug trap functions. When a trap occurs and this bit is set, an interrupt is sent to the ARM core.
3	**DEBUG_DISABLE**	RW	0x0	Set this bit to disable the ARM core's debug logic (for power savings). This bit must remain 0 following power-on reset for normal JTAG debugger operation of the ARM core. When set to 1, it gates off the clocks to the ARM core's debug logic. Once this bit is set, the part must undergo a power-on reset to re-enable debug operation. Manually clearing this bit via a write after it has been set produces unknown results.
2	**USB_CLKGATE**	RW	0x1	This bit must be cleared to 0 for normal operation of the USB controller. When set to 1, it gates off the clocks to the USB controller. USB_CLKGATE can be set during suspend to gate the USB clock during suspend. If this is gated, then the USB controller Reset Received bit (HW_USBCTRL_USBSTS_URI) should be not be polled for reset during suspend; use the HW_USBPHY_CTRL_RESUME_IRQ bit instead. RUN = 0x0 Allow USB to operate normally. NO_CLKS = 0x1 Do not clock USB gates in order to minimize power consumption.
1	**JTAG_SHIELD**	RW	0x0	0 = The JTAG debugger is enabled. 1 = The JTAG debugger is disabled. NORMAL = 0x0 JTAG debugger enabled. SHIELDS_UP = 0x1 JTAG debugger disabled.
0	**LATCH_ENTROPY**	RW	0x0	Setting this bit latches the current value of the entropy register into HW_DIGCTL_ENTROPY_VALUE. This can be used get a stable value on players that do not deassert the PSWITCH while powered up.

DESCRIPTION:

This register controls various functions throughout the digital portion of the chip.

EXAMPLE:

```
HW_DIGCTL_CTRL_CLR(BM_DIGCTL_CTRL_USB_CLKGATE); // enable USB clock
```

6.4.2 DIGCTL Status Register Description

The DIGCTL Status Register reports status for the digital control block.

HW_DIGCTL_STATUS	0x010
HW_DIGCTL_STATUS_SET	0x014
HW_DIGCTL_STATUS_CLR	0x018
HW_DIGCTL_STATUS_TOG	0x01C

Table 6-4. HW_DIGCTL_STATUS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
USB_HS_PRESENT	USB_OTG_PRESENT	USB_HOST_PRESENT	USB_DEVICE_PRESENT	RSVD2																	DCP_BIST_FAIL	DCP_BIST_PASS	DCP_BIST_DONE	LCD_BIST_FAIL	LCD_BIST_PASS	LCD_BIST_DONE	JTAG_IN_USE	PACKAGE_TYPE			WRITTEN

Table 6-5. HW_DIGCTL_STATUS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**USB_HS_PRESENT**	RO	0x1	This read-only bit returns a 1 when USB high-speed mode is present.
30	**USB_OTG_PRESENT**	RO	0x1	This read-only bit returns a 1 when USB on-the-go (OTG) functionality is present.
29	**USB_HOST_PRESENT**	RO	0x1	This read-only bit returns a 1 when USB host functionality is present.
28	**USB_DEVICE_PRESENT**	RO	0x1	This read-only bit returns a 1 when USB device functionality is present.
27:11	**RSVD2**	RO	0x0	Reserved.
10	**DCP_BIST_FAIL**	RO	0x0	This read-only bit is a 1 if the DCP memory BIST returns a failure.
9	**DCP_BIST_PASS**	RO	0x0	This read-only bit is a 1 if the DCP memory BIST returns a pass.
8	**DCP_BIST_DONE**	RO	0x0	This read-only bit is a 1 if the DCP memory BIST has completed.
7	**LCD_BIST_FAIL**	RO	0x0	This read-only bit is a 1 if the LCD memory BIST returns a failure.
6	**LCD_BIST_PASS**	RO	0x0	This read-only bit is a 1 if the LCD memory BIST returns a pass.
5	**LCD_BIST_DONE**	RO	0x0	This read-only bit is a 1 if the LCD memory BIST has completed.
4	**JTAG_IN_USE**	RO	0x0	This read-only bit is a 1 if JTAG debugger usage has been detected.
3:1	**PACKAGE_TYPE**	RO	0x0	This read-only bit field returns the pin count and package type. 000=169BGA, 011=128TQFP, all others=Reserved.
0	**WRITTEN**	RO	0x0	Set to 1 by any successful write to the HW_DIGCTL_WRITEONCE register.

DESCRIPTION:

The DIGCTL Status Register provides a read-only view to various input conditions and internal states.

EXAMPLE:

```
if(HW_DIGCTL_STATUS.PACKAGE_TYPE)
{
```

```
// do 100-pin package things
}
```

6.4.3 Free-Running HCLK Counter Register Description

The Free-Running HCLK Counter Register is available for performance metrics.

HW_DIGCTL_HCLKCOUNT	0x020
HW_DIGCTL_HCLKCOUNT_SET	0x024
HW_DIGCTL_HCLKCOUNT_CLR	0x028
HW_DIGCTL_HCLKCOUNT_TOG	0x02C

Table 6-6. HW_DIGCTL_HCLKCOUNT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

Table 6-7. HW_DIGCTL_HCLKCOUNT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COUNT**	RO	0x0	This counter counts up from reset using HCLK. The count is valid for HCLK frequencies greater than 2 MHz.

DESCRIPTION:

This counter increments once per HCLK rising edge.

EXAMPLE:

```
StartTime = HW_DIGCTL_HCLKCOUNT;
// Do something you want timed here
EndTime = HW_DIGCTL_HCLKCOUNT;
Duration = EndTime - StartTime; // make sure to handle rollover in a real application
```

6.4.4 On-Chip RAM Control Register Description

The On-Chip RAM Control Register holds on-chip SRAM control bit fields.

HW_DIGCTL_RAMCTRL	0x030
HW_DIGCTL_RAMCTRL_SET	0x034
HW_DIGCTL_RAMCTRL_CLR	0x038
HW_DIGCTL_RAMCTRL_TOG	0x03C

Table 6-8. HW_DIGCTL_RAMCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1																				SPEED_SELECT				RSVD0							RAM_REPAIR_EN

Table 6-9. HW_DIGCTL_RAMCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:12	**RSVD1**	RO	0x0	Reserved.
11:8	**SPEED_SELECT**	RW	0x0	Speed select for 16Kx32 OCRAM instances. Recommended value is 0x0. To be used for characterization. This value may have to be modified to allow lower voltage operation.
7:1	**RSVD0**	RO	0x0	Reserved.
0	**RAM_REPAIR_EN**	RW	0x0	Enable word repair for OCRAM, using the address specified in HW_DIGCTL_RAMREPAIR.

DESCRIPTION:

This register controls various parts of the on-chip RAM, including the repair state machine that shifts the repair configuration data into the SRAM macro-cell.

EXAMPLE:

```
HW_DIGCTL_RAMCTRL_SET(BM_DIGCTL_RAMCTRL_REPAIR_TRANSMIT); // Start the efuse state machine
```

6.4.5 On-Chip RAM Repair Address Register Description

HW_DIGCTL_RAMREPAIR	0x040
HW_DIGCTL_RAMREPAIR_SET	0x044
HW_DIGCTL_RAMREPAIR_CLR	0x048
HW_DIGCTL_RAMREPAIR_TOG	0x04C

Table 6-10. HW_DIGCTL_RAMREPAIR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1																ADDR															

Table 6-11. HW_DIGCTL_RAMREPAIR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSVD1**	RO	0x0	Reserved.
15:0	**ADDR**	RW	0x0	Word repair address for OCRAM. Must be read from OTP and copied to this register. The repair is enabled when HW_DIGCTL_RAMCTRL_RAM_REPAIR_EN is set.

DESCRIPTION:

The On-Chip RAM Repair Address Register holds repair address for the on-chip SRAM. The value must be read from the OTP and copied here.

EXAMPLE:

```
HW_DIGCTL_RAMREPAIR.ADDR= 0xBADA; // read modify write is ok
```

6.4.6 On-Chip ROM Control Register Description

HW_DIGCTL_ROMCTRL	0x050
HW_DIGCTL_ROMCTRL_SET	0x054
HW_DIGCTL_ROMCTRL_CLR	0x058
HW_DIGCTL_ROMCTRL_TOG	0x05C

Table 6-12. HW_DIGCTL_ROMCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD0																												RD_MARGIN			

Table 6-13. HW_DIGCTL_ROMCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:4	**RSVD0**	RO	0x0	Reserved.
3:0	**RD_MARGIN**	RW	0x2	This field is used for setting the read margin for the on-chip ROM. It programs the sense amp differential setting and allows the trade-off between speed and robustness. This field should not be changed unless instructed by Freescale.

DESCRIPTION:

The On-Chip ROM Control Register provides settings for the OCROM.

EXAMPLE:

```
Empty Example.
```

6.4.7 Software Write-Once Register Description

The Software Write-Once Register hold the value used in software certification management.

HW_DIGCTL_WRITEONCE 0x060

Table 6-14. HW_DIGCTL_WRITEONCE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 6-15. HW_DIGCTL_WRITEONCE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RW	0xA5A5A5A5	This field can be written only one time. The contents are not used by hardware.

DESCRIPTION:

This register is used to hold a portion of a certificate that is not mutable after software initialization.

EXAMPLE:

```
HW_DIGCTL_WRITEONCE.U = my_certificate;
```

6.4.8 Entropy Register Description

HW_DIGCTL_ENTROPY 0x090

Table 6-16. HW_DIGCTL_ENTROPY

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
VALUE																															

Table 6-17. HW_DIGCTL_ENTROPY Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**VALUE**	RO	0x0	This read-only bit field always reads back the results of an entropy calculation. It is used to randomize the seeds for random number generators.

DESCRIPTION:

The Entropy register is a read-only test value register.

EXAMPLE:

```
while(HW_DIGCTL_STATUS.PSWITCH != 0)
{
//wait for pswitch to go away
}
HW_DIGCTL_WRITEONCE.BITS = rand(HW_DIGCTL_ENTROPY.VALUE);
```

6.4.9 Entropy Latched Register Description

HW_DIGCTL_ENTROPY_LATCHED 0x0A0

Table 6-18. HW_DIGCTL_ENTROPY_LATCHED

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
VALUE																															

Table 6-19. HW_DIGCTL_ENTROPY_LATCHED Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**VALUE**	RO	0x0	When the LATCH_ENTROPY bit in the HW_DIGCTL_CTRL register is set to 1, the value of the HW_DIGCTL_ENTROPY register is latched into this register. This can be used to latch a stable random value on players where the PSWITCH is not deasserted after power-up.

DESCRIPTION:

The Entropy Latched Register is a read-only test value register.

EXAMPLE:

```
Empty Example.
```

6.4.10 SJTAG Debug Register Description

The SJTAG Debug Register controls various debug points within the SJTAG block and provides read-only views into the SJTAG state machines.

HW_DIGCTL_SJTAGDBG 0x0B0
HW_DIGCTL_SJTAGDBG_SET 0x0B4
HW_DIGCTL_SJTAGDBG_CLR 0x0B8
HW_DIGCTL_SJTAGDBG_TOG 0x0BC

Table 6-20. HW_DIGCTL_SJTAGDBG

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								SJTAG_STATE								RSVD1					SJTAG_TDO	SJTAG_TDI	SJTAG_MODE	DELAYED_ACTIVE				ACTIVE	SJTAG_PIN_STATE	SJTAG_DEBUG_DATA	SJTAG_DEBUG_OE

Table 6-21. HW_DIGCTL_SJTAGDBG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:27	**RSVD2**	RO	0x0	Reserved.
26:16	**SJTAG_STATE**	RO	0x2	This bitfield shows the state of the sjtag_state flip-flops inside the SJTAG controller. These bits implement the second state machine in the SJTAG block.
15:11	**RSVD1**	RO	0x0	Reserved.
10	**SJTAG_TDO**	RO	0x0	This bit shows the state of the ARM JTAG TDO signal as seen inside the SJTAG controller.
9	**SJTAG_TDI**	RO	0x0	This bit shows the state of the JTAG TDI capture FF inside the SJTAG controller.
8	**SJTAG_MODE**	RO	0x0	This bit shows the state of the JTAG mode capture FF inside the SJTAG controller.
7:4	**DELAYED_ACTIVE**	RO	0x0	This bitfield shows the state of the delay_onewire_active_reg FF inside the SJTAG controller. These bits implement the first state machine in the SJTAG block.
3	**ACTIVE**	RO	0x0	This bit shows the state of the onewire_active_reg FF inside the SJTAG controller.
2	**SJTAG_PIN_STATE**	RO	0x0	This bit reflects the state of the input driver sampling the SJTAG pin. When HW_DIGCTL_CTRL_USE_SERIAL_JTAG is cleared to 0, external source can pull the SJTAG pin high without starting the SJTAG state machines. In this mode, the SJTAG_PIN_STATE bit is used to confirm continuity from the pad to the SJTAG block.
1	**SJTAG_DEBUG_DATA**	RW	0x0	When HW_DIGCTL_CTRL_USE_SERIAL_JTAG is cleared to 0, then the SJTAG pin is placed in a diagnostic mode. In that case, this bit controls the input to the pad data drive signal. If HW_DIGCTL_CTRL_SJTAG_DEBUG_OE is set to 1, then this bit also controls the state of the SJTAG pin itself.
0	**SJTAG_DEBUG_OE**	RW	0x0	When HW_DIGCTL_CTRL_USE_SERIAL_JTAG is cleared to 0, then the SJTAG pin is placed in a diagnostic mode. In that case, this bit controls the input to the pad data output enable signal. Setting this bit to 1 turns on the SJTAG pad and drives it to the state indicated by HW_DIGCTL_CTRL_SJTAG_DEBUG_DATA.

6.4.11 Digital Control Microseconds Counter Register Description

The Digital Control Microseconds Counter Register is a read-only test value register.

HW_DIGCTL_MICROSECONDS	0x0C0
HW_DIGCTL_MICROSECONDS_SET	0x0C4
HW_DIGCTL_MICROSECONDS_CLR	0x0C8
HW_DIGCTL_MICROSECONDS_TOG	0x0CC

Table 6-22. HW_DIGCTL_MICROSECONDS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
VALUE																															

Table 6-23. HW_DIGCTL_MICROSECONDS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**VALUE**	RW	0x0	This register maintains a 32-bit counter that increments at a 1-microsecond rate. The 1-MHz clock driving this counter is derived from the 24.0-MHz crystal oscillator. The count value is not preserved over power-downs. The 32-bit value wraps in less than two hours.

DESCRIPTION:

This fixed-rate timer always increments at 24.0 MHz divided by 24 or 1.0 MHz. It does not generate an interrupt.

EXAMPLE:

```
StartTime = HW_DIGCTL_MICROSECONDS_RD();
EndTime = HW_DIGCTL_MICROSECONDS_RD();
ElapsedTime = StartTime - EndTime; // WARNING, handle rollover in real software
```

6.4.12 Digital Control Debug Read Test Register Description

The Digital Control Debug Read Test Register is a read-only test value register.

HW_DIGCTL_DBGRD 0x0D0

Table 6-24. HW_DIGCTL_DBGRD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COMPLEMENT																															

Table 6-25. HW_DIGCTL_DBGRD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COMPLEMENT**	RO	0x789ABCDE	This read-only bit field always reads back the one's complement of the value in HW_DIGCTL_DBG.

DESCRIPTION:

This register is used for debugging purposes.

EXAMPLE:

```
debug_value = HW_DIGCTL_DBGRD_RD();
```

6.4.13 Digital Control Debug Register Description

The Digital Control Debug Register is a read-only test value register.

HW_DIGCTL_DBG 0x0E0

Table 6-26. HW_DIGCTL_DBG

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
VALUE																															

Table 6-27. HW_DIGCTL_DBG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**VALUE**	RO	0x87654321	This read-only bit field always reads back the fixed value 0x87654321.

DESCRIPTION:

This register is used for debugging purposes.

EXAMPLE:

```
debug_value = HW_DIGCTL_DBG_RD();
```

6.4.14 SRAM BIST Control and Status Register Description

The SRAM BIST Control and Status Register provides overall control of the integrated BIST engine.

HW_DIGCTL_OCRAM_BIST_CSR 0x0F0
HW_DIGCTL_OCRAM_BIST_CSR_SET 0x0F4
HW_DIGCTL_OCRAM_BIST_CSR_CLR 0x0F8
HW_DIGCTL_OCRAM_BIST_CSR_TOG 0x0FC

Table 6-28. HW_DIGCTL_OCRAM_BIST_CSR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1																					BIST_DEBUG_MODE	BIST_DATA_CHANGE	BIST_CLKEN	RSVD0				FAIL	PASS	DONE	START

Table 6-29. HW_DIGCTL_OCRAM_BIST_CSR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:11	**RSVD1**	RO	0x0	Reserved.
10	**BIST_DEBUG_MODE**	RW	0x0	OCRAM BIST debug mode select
9	**BIST_DATA_CHANGE**	RW	0x0	OCRAM BIST data background select.
8	**BIST_CLKEN**	RW	0x0	Enable clock gate for OCRAM BIST.
7:4	**RSVD0**	RO	0x0	Reserved.
3	**FAIL**	RO	0x0	BIST has failed.

Table 6-29. HW_DIGCTL_OCRAM_BIST_CSR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**PASS**	RO	0x0	BIST has passed.
1	**DONE**	RO	0x0	BIST has completed.
0	**START**	RW	0x0	Initiate BIST of internal memory when high.

DESCRIPTION:

This register is used to start off the BIST operation on two RAMS in the DMA block. The status signals are returned after the BIST operation is completed to this register.

EXAMPLE:

```
To start the BIST operation, set HW_DIGCTL_1TRAM_BIST_CSR = 0x00000001.
After the BIST is completed and the test passes, the contents of
HW_DIGCTL_1TRAM_BIST_CSR will be 0x00000007, as the DONE and PASS flags
will be set.
```

6.4.15 SRAM Status Register 0 Description

The SRAM Status Register 0 is a read-only fail data register.

HW_DIGCTL_OCRAM_STATUS0	0x110
HW_DIGCTL_OCRAM_STATUS0_SET	0x114
HW_DIGCTL_OCRAM_STATUS0_CLR	0x118
HW_DIGCTL_OCRAM_STATUS0_TOG	0x11C

Table 6-30. HW_DIGCTL_OCRAM_STATUS0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
FAILDATA00																															

Table 6-31. HW_DIGCTL_OCRAM_STATUS0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**FAILDATA00**	RO	0x0	This read-only bit field contains the fail data for the first fail in block 0.

DESCRIPTION:

This register contains fail data for the first fail in block 0.

EXAMPLE:

```
fail_data = HW_DIGCTL_OCRAM_STATUS0_RD();
```

6.4.16 SRAM Status Register 1 Description

The SRAM Status Register 1 is a read-only fail data register.

HW_DIGCTL_OCRAM_STATUS1	0x120
HW_DIGCTL_OCRAM_STATUS1_SET	0x124
HW_DIGCTL_OCRAM_STATUS1_CLR	0x128
HW_DIGCTL_OCRAM_STATUS1_TOG	0x12C

Table 6-32. HW_DIGCTL_OCRAM_STATUS1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
FAILDATA01																															

Table 6-33. HW_DIGCTL_OCRAM_STATUS1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**FAILDATA01**	RO	0x0	This read-only bit field contains the fail data for the second fail in block 0.

DESCRIPTION:

This register contains fail data for the second fail in block 0.

EXAMPLE:

```
fail_data = HW_DIGCTL_OCRAM_STATUS1_RD();
```

6.4.17 SRAM Status Register 2 Description

SRAM Status Register 2 is a read-only fail data register.

HW_DIGCTL_OCRAM_STATUS2	0x130
HW_DIGCTL_OCRAM_STATUS2_SET	0x134
HW_DIGCTL_OCRAM_STATUS2_CLR	0x138
HW_DIGCTL_OCRAM_STATUS2_TOG	0x13C

Table 6-34. HW_DIGCTL_OCRAM_STATUS2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
FAILDATA10																															

Table 6-35. HW_DIGCTL_OCRAM_STATUS2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**FAILDATA10**	RO	0x0	This read-only bit field contains the fail data for the first fail in block 1.

DESCRIPTION:

This register contains fail data for the first fail in block 1.

EXAMPLE:

```
fail_data = HW_DIGCTL_OCRAM_STATUS2_RD();
```

6.4.18 SRAM Status Register 3 Description

RAM Status Register 3 is a read-only fail data register.

HW_DIGCTL_OCRAM_STATUS3	0x140
HW_DIGCTL_OCRAM_STATUS3_SET	0x144

HW_DIGCTL_OCRAM_STATUS3_CLR 0x148
HW_DIGCTL_OCRAM_STATUS3_TOG 0x14C

Table 6-36. HW_DIGCTL_OCRAM_STATUS3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
FAILDATA11																															

Table 6-37. HW_DIGCTL_OCRAM_STATUS3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**FAILDATA11**	RO	0x0	This read-only bit field contains the fail data for the second fail in block 1.

DESCRIPTION:

This register contains fail data for the second fail in block 1.

EXAMPLE:

```
fail_data = HW_DIGCTL_OCRAM_STATUS3_RD();
```

6.4.19 SRAM Status Register 4 Description

SRAM Status Register 4 is a read-only fail data register.

HW_DIGCTL_OCRAM_STATUS4 0x150
HW_DIGCTL_OCRAM_STATUS4_SET 0x154
HW_DIGCTL_OCRAM_STATUS4_CLR 0x158
HW_DIGCTL_OCRAM_STATUS4_TOG 0x15C

Table 6-38. HW_DIGCTL_OCRAM_STATUS4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
FAILDATA20																															

Table 6-39. HW_DIGCTL_OCRAM_STATUS4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**FAILDATA20**	RO	0x0	This read-only bit field contains the fail data for the first fail in block 2.

DESCRIPTION:

This register contains fail data for the first fail in block 2.

EXAMPLE:

```
fail_data = HW_DIGCTL_OCRAM_STATUS4_RD();
```

6.4.20 SRAM Status Register 5 Description

SRAM Status Register 5 is a read-only fail data register.

HW_DIGCTL_OCRAM_STATUS5	0x160
HW_DIGCTL_OCRAM_STATUS5_SET	0x164
HW_DIGCTL_OCRAM_STATUS5_CLR	0x168
HW_DIGCTL_OCRAM_STATUS5_TOG	0x16C

Table 6-40. HW_DIGCTL_OCRAM_STATUS5

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
FAILDATA21																															

Table 6-41. HW_DIGCTL_OCRAM_STATUS5 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**FAILDATA21**	RO	0x0	This read-only bit field contains the fail data for the second fail in block 2.

DESCRIPTION:

This register contains fail data for the second fail in block 2.

EXAMPLE:

```
fail_data = HW_DIGCTL_OCRAM_STATUS5_RD();
```

6.4.21 SRAM Status Register 6 Description

SRAM Status Register 6 is a read-only fail data register.

HW_DIGCTL_OCRAM_STATUS6	0x170
HW_DIGCTL_OCRAM_STATUS6_SET	0x174
HW_DIGCTL_OCRAM_STATUS6_CLR	0x178
HW_DIGCTL_OCRAM_STATUS6_TOG	0x17C

Table 6-42. HW_DIGCTL_OCRAM_STATUS6

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
FAILDATA30																															

Table 6-43. HW_DIGCTL_OCRAM_STATUS6 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**FAILDATA30**	RO	0x0	This read-only bit field contains the fail data for the first fail in block 3.

DESCRIPTION:

This register contains fail data for the first fail in block 3.

EXAMPLE:

```
fail_data = HW_DIGCTL_OCRAM_STATUS6_RD();
```

6.4.22 SRAM Status Register 7 Description

SRAM Status Register 7 is a read-only fail data register.

HW_DIGCTL_OCRAM_STATUS7	0x180
HW_DIGCTL_OCRAM_STATUS7_SET	0x184
HW_DIGCTL_OCRAM_STATUS7_CLR	0x188
HW_DIGCTL_OCRAM_STATUS7_TOG	0x18C

Table 6-44. HW_DIGCTL_OCRAM_STATUS7

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
FAILDATA31																															

Table 6-45. HW_DIGCTL_OCRAM_STATUS7 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**FAILDATA31**	RO	0x0	This read-only bit field contains the fail data for the second fail in block 3.

DESCRIPTION:

This register contains fail data for the second fail in block 3.

EXAMPLE:

```
fail_data = HW_DIGCTL_OCRAM_STATUS7_RD();
```

6.4.23 SRAM Status Register 8 Description

SRAM Status Register 8 is a read-only fail address register.

HW_DIGCTL_OCRAM_STATUS8	0x190
HW_DIGCTL_OCRAM_STATUS8_SET	0x194
HW_DIGCTL_OCRAM_STATUS8_CLR	0x198
HW_DIGCTL_OCRAM_STATUS8_TOG	0x19C

Table 6-46. HW_DIGCTL_OCRAM_STATUS8

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD3			FAILADDR01													RSVD2			FAILADDR00												

Table 6-47. HW_DIGCTL_OCRAM_STATUS8 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSVD3**	RO	0x0	This field is unused.
28:16	**FAILADDR01**	RO	0x0	This read-only bit field contains the failing address for the second fail in block 0.
15:13	**RSVD2**	RO	0x0	This field is unused.
12:0	**FAILADDR00**	RO	0x0	This read-only bit field contains the failing address for the first fail in block 0.

DESCRIPTION:

This register contains fail data for the first and second failures in block 0.

EXAMPLE:

```
fail_data = HW_DIGCTL_OCRAM_STATUS8_RD();
```

6.4.24 SRAM Status Register 9 Description

SRAM Status Register 9 is a read-only fail address register.

HW_DIGCTL_OCRAM_STATUS9	0x1A0
HW_DIGCTL_OCRAM_STATUS9_SET	0x1A4
HW_DIGCTL_OCRAM_STATUS9_CLR	0x1A8
HW_DIGCTL_OCRAM_STATUS9_TOG	0x1AC

Table 6-48. HW_DIGCTL_OCRAM_STATUS9

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD3			FAILADDR11													RSVD2			FAILADDR10												

Table 6-49. HW_DIGCTL_OCRAM_STATUS9 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSVD3**	RO	0x0	This field is unused.
28:16	**FAILADDR11**	RO	0x0	This read-only bit field contains the failing address for the second fail in block 1.
15:13	**RSVD2**	RO	0x0	This field is unused.
12:0	**FAILADDR10**	RO	0x0	This read-only bit field contains the failing address for the first fail in block 1.

DESCRIPTION:

This register contains fail data for the first second failures in block 1.

EXAMPLE:

```
fail_data = HW_DIGCTL_OCRAM_STATUS9_RD();
```

6.4.25 SRAM Status Register 10 Description

SRAM Status Register 10 is a read-only fail address register.

HW_DIGCTL_OCRAM_STATUS10	0x1B0
HW_DIGCTL_OCRAM_STATUS10_SET	0x1B4
HW_DIGCTL_OCRAM_STATUS10_CLR	0x1B8
HW_DIGCTL_OCRAM_STATUS10_TOG	0x1BC

Table 6-50. HW_DIGCTL_OCRAM_STATUS10

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD3			FAILADDR21													RSVD2			FAILADDR20												

Table 6-51. HW_DIGCTL_OCRAM_STATUS10 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSVD3**	RO	0x0	This field is unused.
28:16	**FAILADDR21**	RO	0x0	This read-only bit field contains the failing address for the second fail in block 2.
15:13	**RSVD2**	RO	0x0	This field is unused.
12:0	**FAILADDR20**	RO	0x0	This read-only bit field contains the failing address for the first fail in block 2.

DESCRIPTION:

This register contains fail data for the first and second failures in block 2.

EXAMPLE:

```
fail_data = HW_DIGCTL_OCRAM_STATUS10_RD();
```

6.4.26 SRAM Status Register 11 Description

SRAM Status Register 11 is a read-only fail address register.

HW_DIGCTL_OCRAM_STATUS11	0x1C0
HW_DIGCTL_OCRAM_STATUS11_SET	0x1C4
HW_DIGCTL_OCRAM_STATUS11_CLR	0x1C8
HW_DIGCTL_OCRAM_STATUS11_TOG	0x1CC

Table 6-52. HW_DIGCTL_OCRAM_STATUS11

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD3				FAILADDR31												RSVD2				FAILADDR30											

Table 6-53. HW_DIGCTL_OCRAM_STATUS11 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSVD3**	RO	0x0	This field is unused.
28:16	**FAILADDR31**	RO	0x0	This read-only bit field contains the failing address for the second fail in block 3.
15:13	**RSVD2**	RO	0x0	This field is unused.
12:0	**FAILADDR30**	RO	0x0	This read-only bit field contains the failing address for the first fail in block 3.

DESCRIPTION:

This register contains fail data for the first and second failures in block 3.

EXAMPLE:

```
fail_data = HW_DIGCTL_OCRAM_STATUS11_RD();
```

6.4.27 SRAM Status Register 12 Description

SRAM Status Register 12 is a read-only fail state register.

HW_DIGCTL_OCRAM_STATUS12	0x1D0
HW_DIGCTL_OCRAM_STATUS12_SET	0x1D4
HW_DIGCTL_OCRAM_STATUS12_CLR	0x1D8
HW_DIGCTL_OCRAM_STATUS12_TOG	0x1DC

Table 6-54. HW_DIGCTL_OCRAM_STATUS12

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD3				FAILSTATE11				RSVD2				FAILSTATE10				RSVD1				FAILSTATE01				RSVD0				FAILSTATE00			

Table 6-55. HW_DIGCTL_OCRAM_STATUS12 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**RSVD3**	RO	0x0	This field is unused.
27:24	**FAILSTATE11**	RO	0x0	This read-only bit field contains the failing state for the second fail in block 1.

Table 6-55. HW_DIGCTL_OCRAM_STATUS12 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
23:20	**RSVD2**	RO	0x0	This field is unused.
19:16	**FAILSTATE10**	RO	0x0	This read-only bit field contains the failing state for the first fail in block 1.
15:12	**RSVD1**	RO	0x0	This field is unused.
11:8	**FAILSTATE01**	RO	0x0	This read-only bit field contains the failing state for the second fail in block 0.
7:4	**RSVD0**	RO	0x0	This field is unused.
3:0	**FAILSTATE00**	RO	0x0	This read-only bit field contains the failing state for the first fail in block 0.

DESCRIPTION:

This register contains fail data for the first and second failures in blocks 0 and 1.

EXAMPLE:

```
fail_data = HW_DIGCTL_OCRAM_STATUS12_RD();
```

6.4.28 SRAM Status Register 13 Description

SRAM Status Register 13 is a read-only fail state register.

HW_DIGCTL_OCRAM_STATUS13	0x1E0
HW_DIGCTL_OCRAM_STATUS13_SET	0x1E4
HW_DIGCTL_OCRAM_STATUS13_CLR	0x1E8
HW_DIGCTL_OCRAM_STATUS13_TOG	0x1EC

Table 6-56. HW_DIGCTL_OCRAM_STATUS13

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD3				FAILSTATE31				RSVD2				FAILSTATE30				RSVD1				FAILSTATE21				RSVD0				FAILSTATE20			

Table 6-57. HW_DIGCTL_OCRAM_STATUS13 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**RSVD3**	RO	0x0	This field is unused.
27:24	**FAILSTATE31**	RO	0x0	This read-only bit field contains the failing state for the second fail in block 3.
23:20	**RSVD2**	RO	0x0	This field is unused.
19:16	**FAILSTATE30**	RO	0x0	This read-only bit field contains the failing state for the first fail in block 3.
15:12	**RSVD1**	RO	0x0	This field is unused.
11:8	**FAILSTATE21**	RO	0x0	This read-only bit field contains the failing state for the second fail in block 2.

Table 6-57. HW_DIGCTL_OCRAM_STATUS13 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
7:4	**RSVD0**	RO	0x0	This field is unused.
3:0	**FAILSTATE20**	RO	0x0	This read-only bit field contains the failing state for the first fail in block 2.

DESCRIPTION:

This register contains fail data for the first and second failures in blocks 2 and 3.

EXAMPLE:

```
fail_data = HW_DIGCTL_OCRAM_STATUS0_RD();
```

6.4.29 Digital Control Scratch Register 0 Description

HW_DIGCTL_SCRATCH0 0x290

Table 6-58. HW_DIGCTL_SCRATCH0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PTR																															

Table 6-59. HW_DIGCTL_SCRATCH0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**PTR**	RW	0x0	

DESCRIPTION:

Scratch Pad Register 0.

EXAMPLE:

```
scratch_pad = (*void)HW_DIGCTL_SCRATCH0.PTR;
```

6.4.30 Digital Control Scratch Register 1 Description

HW_DIGCTL_SCRATCH1 0x2A0

Table 6-60. HW_DIGCTL_SCRATCH1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PTR																															

Table 6-61. HW_DIGCTL_SCRATCH1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**PTR**	RW	0x0	

DESCRIPTION:

Scratch Pad Register 1.

EXAMPLE:

```
scratch_pad = (*void)HW_DIGCTL_SCRATCH1.PTR;
```

6.4.31 Digital Control ARM Cache Register Description

This register provides the ARM cache RAM controls.

HW_DIGCTL_ARMCACHE 0x2B0

Table 6-62. HW_DIGCTL_ARMCACHE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4														VALID_SS		RSVD3		DRTY_SS		RSVD2		CACHE_SS		RSVD1		DTAG_SS		RSVD0		ITAG_SS	

Table 6-63. HW_DIGCTL_ARMCACHE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:18	**RSVD4**	RO	0x0	Reserved.
17:16	**VALID_SS**	RW	0x0	Timing Control for 64x24x1 RAMs (both instruction and data cache_valid arrays).
15:14	**RSVD3**	RO	0x0	Reserved.
13:12	**DRTY_SS**	RW	0x0	Timing Control for 128x8x1 RAM (DDRTY).
11:10	**RSVD2**	RO	0x0	Reserved.
9:8	**CACHE_SS**	RW	0x0	Timing Control for 1024x32x4 RAMs (both instruction and data cache arrays).
7:6	**RSVD1**	RO	0x0	Reserved.
5:4	**DTAG_SS**	RW	0x0	Timing Control for 256x22x4 RAM (DTAG).
3:2	**RSVD0**	RO	0x0	Reserved.
1:0	**ITAG_SS**	RW	0x0	Timing Control for 128x22x4 RAM (ITAG).

DESCRIPTION:

ARM Cache Control Register.

EXAMPLE:

```
cache_timing = HW_DIGCTL_ARMCACHE.CACHE_SS;
```

6.4.32 Debug Trap Range Low Address Description

The Debug Trap Range Low Address Register defines the lower bound for an address range that can be enabled to trigger an interrupt to the ARM core when an AHB cycle occurs within this range.

HW_DIGCTL_DEBUG_TRAP_ADDR_LOW 0x2C0

Table 6-64. HW_DIGCTL_DEBUG_TRAP_ADDR_LOW

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 6-65. HW_DIGCTL_DEBUG_TRAP_ADDR_LOW Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x0	This field contains the 32-bit lower address for the debug trap range.

DESCRIPTION:

This register sets the lower address that defines the debug trap function. When this function is enabled, any active AHB cycle on either Layer 0 or Layer 3 which accesses this range will trigger an interrupt to the ARM core.

6.4.33 Debug Trap Range High Address Description

The Debug Trap Range High Address Register defines the upper bound for an address range that can be enabled to trigger an interrupt to the ARM core when an AHB cycle occurs within this range.

HW_DIGCTL_DEBUG_TRAP_ADDR_HIGH 0x2D0

Table 6-66. HW_DIGCTL_DEBUG_TRAP_ADDR_HIGH

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 6-67. HW_DIGCTL_DEBUG_TRAP_ADDR_HIGH Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x0	This field contains the 32-bit upper address for the debug trap range.

DESCRIPTION:

This register sets the upper address that defines the debug trap function. When this function is enabled, any active AHB cycle on either Layer 0 or Layer 3 which accesses this range will trigger an interrupt to the ARM core.

6.4.34 Freescale Copyright Identifier Register Description

Read-only Freescale Copyright Identifier Register.

HW_DIGCTL_SGTL 0x300

Table 6-68. HW_DIGCTL_SGTL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COPYRIGHT																															

Table 6-69. HW_DIGCTL_SGTL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COPYRIGHT**	RO	0x6d676953	This read-only bit field contains the four bytes of the Freescale Copyright Identification String.

DESCRIPTION:

This register provides read-only access to the zero-terminated twelve-byte Freescale copyright identification string. This register behaves somewhat differently from all other APB registers in that it provides different read-back values at its three successive SCT bus addresses. The following binary values are read back at 0x300, 0x304, and 0x308 respectively:

0x6d676953 m,g,i,S at 0x300

0x6c655461 l,e,T,a at 0x304

0x00AEA92d 0x00, Registered Trademark (Æ), Copyright (©), hyphen (-) at 0x308

The debugger does a string compare on these 12 successive little endian bytes. Any chip that reads back these values is either a Freescale chip or it is a competitors chip that is violating Freescale registered trademarks and or copyrights.

EXAMPLE:

```
printf("%s", (char *)HW_DIGCTL_SGTL_ADDR);
```

6.4.35 Digital Control Chip Revision Register Description

Read-only chip revision register.

HW_DIGCTL_CHIPID 0x310

Table 6-70. HW_DIGCTL_CHIPID

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PRODUCT_CODE																RSVD0								REVISION							

Table 6-71. HW_DIGCTL_CHIPID Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**PRODUCT_CODE**	RO	0x3780	This read-only bit field returns 0x3780, which identifies the generation from which the part is derived.
15:8	**RSVD0**	RO	0x0	Reserved.
7:0	**REVISION**	RO	0x00	This read-only bit field always reads back the mask revision level of the chip. 0x0 = TA1 0x1 = TA2 0x2 = TA3 0x3 = TA4

EXAMPLE:

```
FormatAndPrintChipID(HW_DIGCTL_CHIPID_PRODUCT_CODE,HW_DIGCTL_CHIPID_REVISION );
```

6.4.36 AHB Statistics Control Register Description

The AHB Statistics Control Register selects which AHB masters on each layer of the AHB subsystem are enabled to contribute to the statistics calculations.

HW_DIGCTL_AHB_STATS_SELECT 0x330

Table 6-72. HW_DIGCTL_AHB_STATS_SELECT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD3				L3_MASTER_SELECT				RSVD2				L2_MASTER_SELECT				RSVD1				L1_MASTER_SELECT				RSVD0				L0_MASTER_SELECT			

Table 6-73. HW_DIGCTL_AHB_STATS_SELECT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**RSVD3**	RO	0x0	Reserved.
27:24	**L3_MASTER_SELECT**	RW	0x0	Set various bits of this bit field to one to enable performance monitoring in the AHB Layer 3 arbiter for the corresponding AHB master. APBH = 0x1 Select APBH DMA Master. APBX = 0x2 Select APBX DMA Master. USB = 0x4 Select USB Master.
23:20	**RSVD2**	RO	0x0	Reserved.
19:16	**L2_MASTER_SELECT**	RW	0x0	Set various bits of this bit field to one to enable performance monitoring in the AHB Layer 2 arbiter for the corresponding AHB master. ARM_D = 0x1 Select ARM D Master.
15:12	**RSVD1**	RO	0x0	Reserved.

Table 6-73. HW_DIGCTL_AHB_STATS_SELECT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
11:8	**L1_MASTER_SELECT**	RW	0x0	Set various bits of this bit field to one to enable performance monitoring in the AHB Layer 1 arbiter for the corresponding AHB master. ARM_I = 0x1 Select ARM I Master.
7:4	**RSVD0**	RO	0x0	Reserved.
3:0	**L0_MASTER_SELECT**	RW	0x0	Set various bits of this bit field to one to enable performance monitoring in the AHB Layer 0 arbiter for the corresponding AHB master. ECC8 = 0x1 Select ECC8 Master. CRYPTO = 0x2 Select Crypto Master.

6.4.37 AHB Layer 0 Transfer Count Register Description

The AHB Layer 0 Transfer Count Register counts the number of AHB bus cycles during which a transfer is active on AHB Layer 0.

HW_DIGCTL_L0_AHB_ACTIVE_CYCLES 0x340

Table 6-74. HW_DIGCTL_L0_AHB_ACTIVE_CYCLES

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

Table 6-75. HW_DIGCTL_L0_AHB_ACTIVE_CYCLES Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COUNT**	RW	0x0	This field contains the count of AHB bus cycles during which a master was active on the AHB Layer 0.

DESCRIPTION:

This field counts the number of AHB cycles in which a master was requesting a transfer, and the slave had not responded. This includes cycles in which it was requesting transfers but was not granted them, as well as cycles in which it was granted and driving the bus but the targeted slave was not ready. The master selects in HW_DIGCTL_AHB_STATS_SELECT_L0_MASTER_SELECT are used in the arbiter to mask which master's cycles are actually recorded here.

EXAMPLE:

```
NumberCycles = HW_DIGCTL_L0_AHB_ACTIVE_CYCLES_COUNT_RD();
```

6.4.38 AHB Layer 0 Performance Metric for Stalled Bus Cycles Register Description

Used for AHB bus utilization measurements, the AHB Performance Metric for Stalled Bus Cycles Register counts the number of stalled AHB cycles.

HW_DIGCTL_L0_AHB_DATA_STALLED 0x350

Table 6-76. HW_DIGCTL_L0_AHB_DATA_STALLED

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

Table 6-77. HW_DIGCTL_L0_AHB_DATA_STALLED Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COUNT**	RW	0x0	This field counts the number of AHB cycles in which a master was stalled.

DESCRIPTION:

This counter increments on a data-phase of the AHB in which the HREADY signal is low, indicating a stalled data transfer.

EXAMPLE:

```
NumberStalledCycles = HW_DIGCTL_L0_AHB_DATA_STALLED_COUNT_RD();
```

6.4.39 AHB Layer 0 Performance Metric for Valid Bus Cycles Register Description

Used for AHB bus utilization measurements, the AHB Performance Metric for Valid Bus Cycles Register counts the number of actual AHB cycles in which a data transfer is completed.

HW_DIGCTL_L0_AHB_DATA_CYCLES 0x360

Table 6-78. HW_DIGCTL_L0_AHB_DATA_CYCLES

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

Table 6-79. HW_DIGCTL_L0_AHB_DATA_CYCLES Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COUNT**	RW	0x0	This field contains the count of AHB bus cycles during which data was actually transferred from a master to a slave or from a slave to a master.

DESCRIPTION:

This field counts the number of AHB cycles in which a master completed a data transfer (a data-phase in which HREADY is high).

EXAMPLE:

```
StartTime = HW_DIGCTL_HCLKCOUNT_RD();
while(HW_DIGCTL_L0_AHB_DATA_CYCLES.COUNT less than 1000000)
{
// wait for a specific number of xfers
}
ElapsedTime = HW_DIGCTL_HCLKCOUNT_RD() - StartTime;
```

6.4.40 AHB Layer 1 Transfer Count Register Description

The AHB Layer 1 Transfer Count Register counts the number of AHB bus cycles during which a transfer is active on AHB Layer 1.

HW_DIGCTL_L1_AHB_ACTIVE_CYCLES 0x370

Table 6-80. HW_DIGCTL_L1_AHB_ACTIVE_CYCLES

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

Table 6-81. HW_DIGCTL_L1_AHB_ACTIVE_CYCLES Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COUNT**	RW	0x0	This field contains the count of AHB bus cycles during which a master was active on the AHB Layer 1.

DESCRIPTION:

This field counts the number of AHB cycles in which a master was requesting a transfer, and the slave had not responded. This includes cycles in which it was requesting transfers but was not granted them, as well as cycles in which it was granted and driving the bus but the targeted slave was not ready. The master selects in HW_DIGCTL_AHB_STATS_SELECT_L1_MASTER_SELECT are used in the arbiter to mask which master's cycles are actually recorded here.

EXAMPLE:

```
NumberCycles = HW_DIGCTL_L1_AHB_ACTIVE_CYCLES_COUNT_RD();
```

6.4.41 AHB Layer 1 Performance Metric for Stalled Bus Cycles Register Description

Used for AHB bus utilization measurements, the AHB Performance Metric for Stalled Bus Cycles Register counts the number of stalled AHB cycles.

HW_DIGCTL_L1_AHB_DATA_STALLED 0x380

Table 6-82. HW_DIGCTL_L1_AHB_DATA_STALLED

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

Table 6-83. HW_DIGCTL_L1_AHB_DATA_STALLED Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COUNT**	RW	0x0	This field counts the number of AHB cycles in which a master was stalled.

DESCRIPTION:

This counter increments on a data-phase of the AHB in which the HREADY signal is low, indicating a stalled data transfer.

EXAMPLE:

```
NumberStalledCycles = HW_DIGCTL_L1_AHB_DATA_STALLED_COUNT_RD();
```

6.4.42 AHB Layer 1 Performance Metric for Valid Bus Cycles Register Description

Used for AHB bus utilization measurements, the AHB Performance Metric for Valid Bus Cycles Register counts the number of actual AHB cycles in which a data transfer is completed.

HW_DIGCTL_L1_AHB_DATA_CYCLES 0x390

Table 6-84. HW_DIGCTL_L1_AHB_DATA_CYCLES

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

Table 6-85. HW_DIGCTL_L1_AHB_DATA_CYCLES Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COUNT**	RW	0x0	This field contains the count of AHB bus cycles during which data was actually transferred from a master to a slave or from a slave to a master.

DESCRIPTION:

This field counts the number of AHB cycles in which a master completed a data transfer (a data-phase in which HREADY is high).

EXAMPLE:

```
StartTime = HW_DIGCTL_HCLKCOUNT_RD();
while(HW_DIGCTL_L1_AHB_DATA_CYCLES.COUNT less than 1000000)
{
// wait for a specific number of xfers
}
ElapsedTime = HW_DIGCTL_HCLKCOUNT_RD() - StartTime;
```

6.4.43 AHB Layer 2 Transfer Count Register Description

The AHB Layer 2 Transfer Count Register counts the number of AHB bus cycles during which a transfer is active on AHB Layer 2.

HW_DIGCTL_L2_AHB_ACTIVE_CYCLES 0x3A0

Table 6-86. HW_DIGCTL_L2_AHB_ACTIVE_CYCLES

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

Table 6-87. HW_DIGCTL_L2_AHB_ACTIVE_CYCLES Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COUNT**	RW	0x0	This field contains the count of AHB bus cycles during which a master was active on the AHB Layer 2.

DESCRIPTION:

This field counts the number of AHB cycles in which a master was requesting a transfer, and the slave had not responded. This includes cycles in which it was requesting transfers but was not granted them, as well as cycles in which it was granted and driving the bus but the targeted slave was not ready. The master selects in HW_DIGCTL_AHB_STATS_SELECT_L2_MASTER_SELECT are used in the arbiter to mask which master's cycles are actually recorded here.

EXAMPLE:

```
NumberCycles = HW_DIGCTL_L2_AHB_ACTIVE_CYCLES_COUNT_RD();
```

6.4.44 AHB Layer 2 Performance Metric for Stalled Bus Cycles Register Description

Used for AHB bus utilization measurements, the AHB Performance Metric for Stalled Bus Cycles Register counts the number of stalled AHB cycles.

HW_DIGCTL_L2_AHB_DATA_STALLED 0x3B0

Table 6-88. HW_DIGCTL_L2_AHB_DATA_STALLED

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

Table 6-89. HW_DIGCTL_L2_AHB_DATA_STALLED Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COUNT**	RW	0x0	This field counts the number of AHB cycles in which a master was stalled.

DESCRIPTION:

This counter increments on a data-phase of the AHB in which the HREADY signal is low, indicating a stalled data transfer.

EXAMPLE:

```
NumberStalledCycles = HW_DIGCTL_L2_AHB_DATA_STALLED_COUNT_RD();
```

6.4.45 AHB Layer 2 Performance Metric for Valid Bus Cycles Register Description

Used for AHB bus utilization measurements, the AHB Performance Metric for Valid Bus Cycles Register counts the number of actual AHB cycles in which a data transfer is completed.

HW_DIGCTL_L2_AHB_DATA_CYCLES 0x3C0

Table 6-90. HW_DIGCTL_L2_AHB_DATA_CYCLES

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

Table 6-91. HW_DIGCTL_L2_AHB_DATA_CYCLES Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COUNT**	RW	0x0	This field contains the count of AHB bus cycles during which data was actually transferred from a master to a slave or from a slave to a master.

DESCRIPTION:

This field counts the number of AHB cycles in which a master completed a data transfer (a data-phase in which HREADY is high).

EXAMPLE:

```
StartTime = HW_DIGCTL_HCLKCOUNT_RD();
while(HW_DIGCTL_L2_AHB_DATA_CYCLES.COUNT less than 1000000)
{
// wait for a specific number of xfers
}
ElapsedTime = HW_DIGCTL_HCLKCOUNT_RD() - StartTime;
```

6.4.46 AHB Layer 3 Transfer Count Register Description

The AHB Layer 3 Transfer Count Register counts the number of AHB bus cycles during which a transfer is active on AHB Layer 3.

HW_DIGCTL_L3_AHB_ACTIVE_CYCLES 0x3D0

Table 6-92. HW_DIGCTL_L3_AHB_ACTIVE_CYCLES

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

Table 6-93. HW_DIGCTL_L3_AHB_ACTIVE_CYCLES Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COUNT**	RW	0x0	This field contains the count of AHB bus cycles during which a master was active on the AHB Layer 3.

DESCRIPTION:

This field counts the number of AHB cycles in which a master was requesting a transfer, and the slave had not responded. This includes cycles in which it was requesting transfers but was not granted them, as well as cycles in which it was granted and driving the bus but the targeted slave was not ready. The master selects in HW_DIGCTL_AHB_STATS_SELECT_L3_MASTER_SELECT are used in the arbiter to mask which master's cycles are actually recorded here.

EXAMPLE:

```
NumberCycles = HW_DIGCTL_L3_AHB_ACTIVE_CYCLES_COUNT_RD();
```

6.4.47 AHB Layer 3 Performance Metric for Stalled Bus Cycles Register Description

Used for AHB bus utilization measurements, the AHB Performance Metric for Stalled Bus Cycles Register counts the number of stalled AHB cycles.

HW_DIGCTL_L3_AHB_DATA_STALLED 0x3E0

Table 6-94. HW_DIGCTL_L3_AHB_DATA_STALLED

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

Table 6-95. HW_DIGCTL_L3_AHB_DATA_STALLED Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COUNT**	RW	0x0	This field counts the number of AHB cycles in which a master was stalled.

DESCRIPTION:

This counter increments on a data-phase of the AHB in which the HREADY signal is low, indicating a stalled data transfer.

EXAMPLE:

```
NumberStalledCycles = HW_DIGCTL_L3_AHB_DATA_STALLED_COUNT_RD();
```

6.4.48 AHB Layer 3 Performance Metric for Valid Bus Cycles Register Description

Used for AHB bus utilization measurements, the AHB Performance Metric for Valid Bus Cycles Register counts the number of actual AHB cycles in which a data transfer is completed.

HW_DIGCTL_L3_AHB_DATA_CYCLES 0x3F0

Table 6-96. HW_DIGCTL_L3_AHB_DATA_CYCLES

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

Table 6-97. HW_DIGCTL_L3_AHB_DATA_CYCLES Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COUNT**	RW	0x0	This field contains the count of AHB bus cycles during which data was actually transferred from a master to a slave or from a slave to a master.

DESCRIPTION:

This field counts the number of AHB cycles in which a master completed a data transfer (a data-phase in which HREADY is high).

EXAMPLE:

```
StartTime = HW_DIGCTL_HCLKCOUNT_RD();
while(HW_DIGCTL_L3_AHB_DATA_CYCLES.COUNT less than 1000000)
{
// wait for a specific number of xfers
}
ElapsedTime = HW_DIGCTL_HCLKCOUNT_RD() - StartTime;
```

6.4.49 EMI CLK/CLKN Delay Adjustment Description

Used to adjust delay of the EMI_CLK and EMI_CLKN.

HW_DIGCTL_EMICLK_DELAY 0x500

Table 6-98. HW_DIGCTL_EMICLK_DELAY

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD0																											NUM_TAPS					

Table 6-99. HW_DIGCTL_EMICLK_DELAY Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSVD0**	RO	0x0	Reserved.
4:0	**NUM_TAPS**	RW	0x0	Select one of 32 delay line taps to delay the EMI_CLK and EMI_CLKN pins

DESCRIPTION:

This register setting is useful in cases where additional setup time is required (at the expense of hold) for EMI Address/Command signals in SDRAM and mDDR modes. It can also be used to give additional write data setup time in SDRAM modes.

EXAMPLE:

```
Empty Example.
```

DIGCTL Block digctl, Revision 1.0

Chapter 7
On-Chip OTP (OCOTP) Controller

This chapter describes the on-chip OTP (OCOTP) controller included on the i.MX23. Programmable registers are described in Section 7.4, “Programmable Registers.”

7.1 Overview

The on-chip OTP controller (OCOTP) provides the following functions:

- Full memory-mapped (restricted) read access of 1 Kbit of on-chip OTP ROM.
- Data-register programming interface for the 1 Kbit of OTP.
- Generation of the chip hardware capability bus.
- Chip-level pin access to nonrestricted portions of OTP.

The OCOTP is connected to the APBH system peripheral bus and is accessible via the ARM core Data-AHB layer (Layer 2). Read accesses can be done at maximum HCLK frequency. Programming/writes can be performed at 24 MHz. The system diagram for the OCOTP is shown in Figure 7-1.

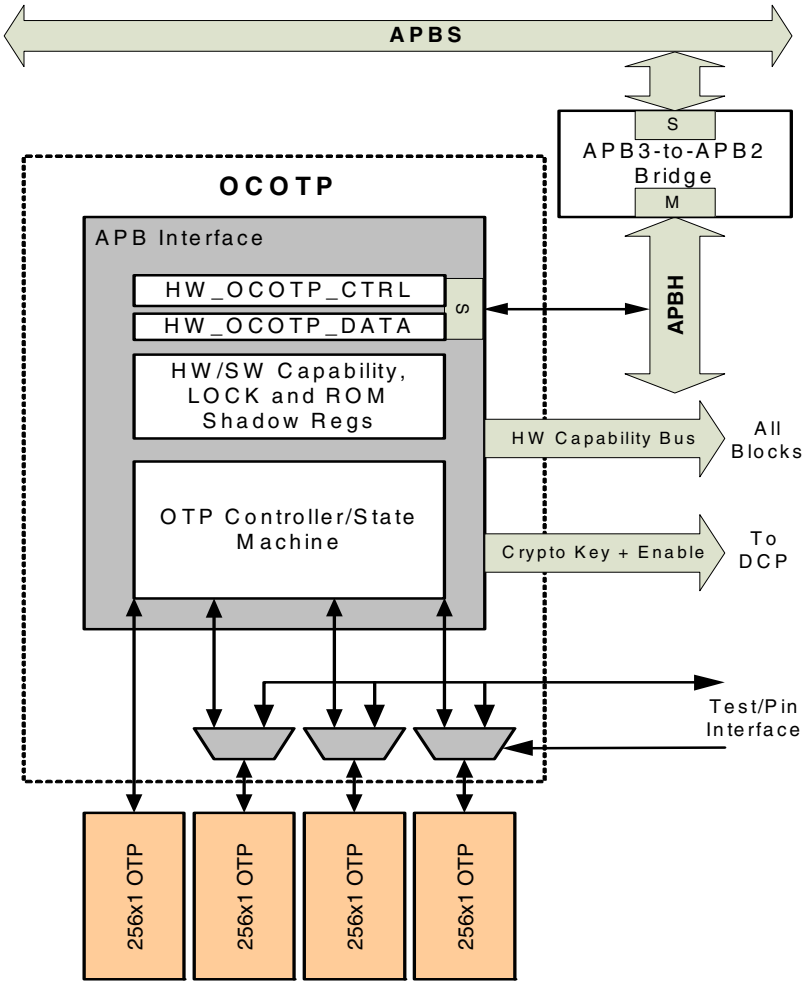


Figure 7-1. On-Chip OTP (OCOTP) Controller Block Diagram

7.2 Operation

The APB interface of the OCOTP provides two functions:

- Programmer-model access to registers (see Section 7.4, “Programmable Registers,” for register details). These operations require a bank opening sequence via HW_OCOTP_CTRL_RD_BANK_OPEN.
- Restricted 32-bit word write/program access to the 1-Kbit OTP

The OTP is divided into 32-bit words (32 in total). All the 32 words are memory-mapped to APBH addresses (for reads only). Writes require the use of HW_OCOTP_CTRL_ADDR. The customer view of the high-level OTP allocation is shown in Figure 7-2.

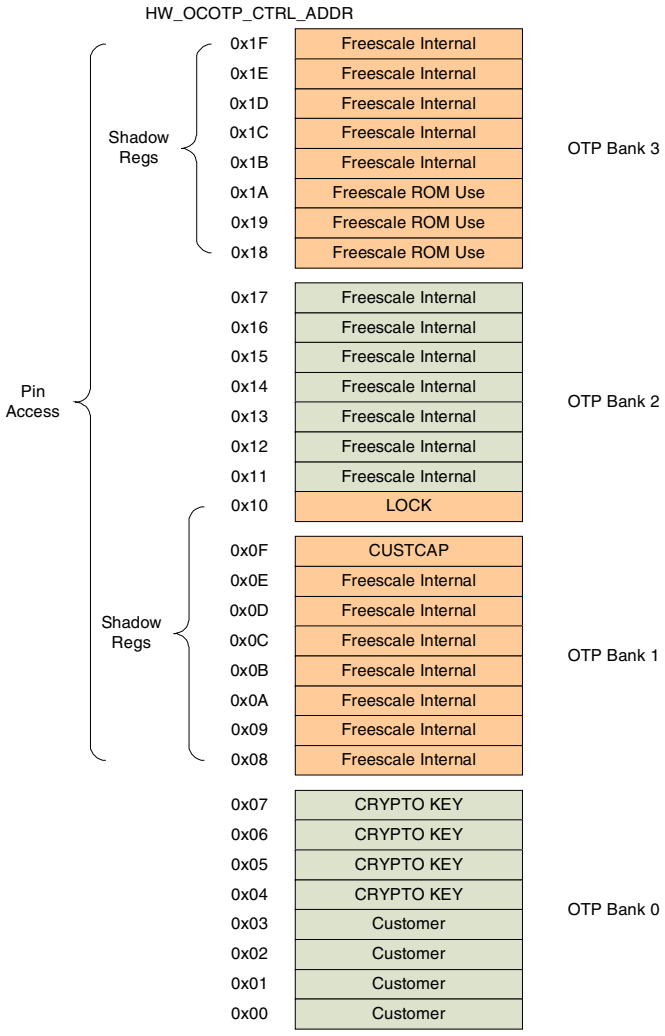


Figure 7-2. OCOTP Allocation

OTP reads and writes can be performed on 32-bit words only. For writes, the 32-bit word reflects the "write-mask", such that bit fields with 0 will not be programmed and bit fields with 1 will be programmed.

For OTP random access, the programming interface consists of:

- HW_OCOTP_DATA—Data register (32- bit) for OTP programming (writes).
- HW_OCOTP_CTRL_ADDR—Address register (5-bit) for OTP programming (writes).
- HW_OCOTP_CTRL_BUSY—Programming/write request/status handshake bit.
- HW_OCOTP_CTRL_ERROR—Read/write access error status.
- HW_OCOTP_CTRL_RD_BANK_OPEN—Status of OTP read availability (reads).

7.2.1 Software Read Sequence

Reading OTP contents is relatively simple, because all OTP words are memory-mapped on the APB space (see Section 7.4, “Programmable Registers,” for details). These registers are read-only, except for the HW/SW capability shadow registers, which are writable until the appropriate LOCK bit in OTP is set.

Due to the fuse-read architecture, the OTP banks must be open before they can be read. This is accomplished as follows (the following does not apply to shadow registers, which can be read at any time).

1. Program the HCLK to a frequency up to the maximum allowable HCLK frequency. Note that this cannot exceed 200 MHz.
2. Check that HW_OCOTP_CTRL_BUSY and HW_OCOTP_CTRL_ERROR are clear.
3. Set HW_OCOTP_CTRL_RD_BANK_OPEN. This will kick the controller to put the fuses into read mode. The controller will set HW_OCOTP_CTRL_BUSY until the OTP contents are readable. Note that if there was a pending write (holding HW_OCOTP_CTRL_BUSY) and HW_OCOTP_CTRL_RD_BANK_OPEN was set, the controller would complete the write and immediately move into read operation (keeping HW_OCOTP_CTRL_BUSY set while the banks are being opened).
4. Poll for HW_OCOTP_CTRL_BUSY clear. When HW_OCOTP_CTRL_BUSY is clear and HW_OCOTP_CTRL_RD_BANK_OPEN is set, read the data from the appropriate memory-mapped address. Note that this is not necessary for registers that are shadowed. Reading before HW_OCOTP_CTRL_BUSY is cleared by the controller, will return 0xBADA_BADA and will result in the setting of HW_OCOTP_CTRL_ERROR. Because opening banks takes approximately 33 HCLK cycles, immediate polling for BUSY is not recommended.
5. Once accesses are complete, clear HW_OCOTP_CTRL_RD_BANK_OPEN. Leaving the banks open will cause current drain.

If data is accessed from a protected region (such as the crypto key, once a read LOCK bit has been set), the controller returns 0xBADA_BADA. In addition HW_OCOTP_CTRL_ERROR is set. It must be cleared by software before any new write access can be issued. Subsequent reads to unrestricted mapped OTP locations will still work successfully assuming that HW_OCOTP_CTRL_RD_BANK_OPEN is set and HW_OCOTP_CTRL_BUSY is clear.

It should be noted that after opening the banks, read latencies to OTP are “instant” (meaning they behave like regular reads from hardware registers), since parallel loading is used.

It should also be noted that setting HW_OCOTP_CTRL_RELOAD_SHADOWS to reload shadow registers does not set HW_OCOTP_CTRL_RD_BANK_OPEN. HW_OCOTP_CTRL_RD_BANK_OPEN can only be set and cleared by software. Forced reloading of shadows is covered in Section 7.2.4, “Shadow Registers and Hardware Capability Bus.”

7.2.2 Software Write Sequence

In order to avoid erroneous code performing erroneous writes to OTP, a special unlocking sequence is required for writes.

1. Program HCLK to 24 MHz. OTP writes do not work at frequencies above 24 MHz.
2. Set the VDDIO voltage to 2.8 V (using HW_POWER_VDDIOCTRL_TRG). The VDDIO voltage is used to program OTP. Incorrect voltage and frequency settings will result in the OTP being programmed with incorrect values.
3. Check that HW_OCOTP_CTRL_BUSY and HW_OCOTP_CTRL_ERROR are clear. Overlapped accesses are not supported by the controller. Any pending write must be completed before a write access can be requested. In addition, the banks cannot be open for reading, so HW_OCOTP_CTRL_RD_BANK_OPEN must also be clear. If the write is done following a previous write, the postamble wait period of 2 µs must be followed after the clearing of HW_OCOTP_CTRL_BUSY (see Section 7.2.3, "Write Postamble,").
4. Write the requested address to HW_OCOTP_CTRL_ADDR and program the unlock code into HW_OCOTP_CTRL_WR_UNLOCK. This must be programmed for each write access. The lock code is documented in the register description. Both the unlock code and address can be written in the same operation.
5. Write the data to HW_OCOTP_DATA. This automatically sets HW_OCOTP_CTRL_BUSY and clears HW_OCOTP_CTRL_WR_UNLOCK. In this case, the data is a programming mask. Bit fields with ones will result in that OTP bit being set. Only the controller can clear HW_OCOTP_CTRL_BUSY. The controller will use the mask to program a 32-bit word in the OTP per the address in ADDR. At the same time that the write is accepted, the controller makes an internal copy of HW_OCOTP_CTRL_ADDR that cannot be updated until the next write sequence is initiated. This copy guarantees that erroneous writes to HW_OCOTP_CTRL_ADDR will not affect an active write operation. It should also be noted that, during the programming, HW_OCOTP_DATA will shift right (with zero fill). This shifting is required to program the OTP serially. During the write operation, HW_OCOTP_DATA cannot be modified.
6. Once complete, the controller clears BUSY. Beyond this, the 2-µs postamble requirement must be met before submitting any further OTP operations (see Section 7.2.3, "Write Postamble,"). A write request to a protected region will result in no OTP access and no setting of HW_OCOTP_CTRL_BUSY. In addition, HW_OCOTP_CTRL_ERROR will be set. It must be cleared by software before any new write access can be issued.

It should be noted that write latencies to OTP are in the order of 10s to 100s of microseconds per word. Write latencies will vary based on the location of the word within the OTP bank. Once a write is initiated, HW_OCOTP_DATA is shifted one bit per every 32 HCLK cycles.

Given:

8 words per OTP bank
32 bits per word
tHCLK is the HCLK clock period
n word locations (where $0 \leq n \leq 7$)

Then, the approximate write latency for a given word is:

*tHCLK * 32 * 32 * n*

In addition to this latency, software must allow for the 2-µs postamble (using HW_DIGCTL_MICROSECONDS), as described in Section 7.2.3, "Write Postamble,"

7.2.3 Write Postamble

Due to internal electrical characteristics of the OTP during writes, all OTP operations following a write must be separated by 2 µs after the clearing of HW_OCOTP_CTRL_BUSY following the write. This guarantees programming voltages on-chip to reach a steady state when exiting a write sequence. This includes reads, shadow reloads, or other writes. A recommended software sequence to meet the postamble requirements is as follows:

1. Issue the write and poll for BUSY (as per Section 7.2.2, "Software Write Sequence,").
2. Once BUSY is clear, use HW_DIGCTL_MICROSECONDS to wait 2 µs.
3. Perform the next OTP operation.

7.2.4 Shadow Registers and Hardware Capability Bus

The on-chip customer hardware capability bus is generated using a direct connection to the HW_OCOTP_CUSTCAP shadow register. The bits are copied from the OTP on reset. They can be modified until HW_OCOTP_LOCK_CUSTCAP_SHADOW is set.

The user can force a reload of the shadow registers (including HW_OCOTP_LOCK) without having to reset the device, which is useful for debugging code. To force a reload:

- Set HW_OCOTP_CTRL_RELOAD_SHADOWS.
- Wait for HW_OCOTP_CTRL_BUSY and HW_OCOTP_CTRL_RELOAD_SHADOWS to be cleared by the controller.
- Attempting to write to the shadow registers while the shadows are being reloaded will result in the setting of HW_OCOTP_CTRL_ERROR. In addition, the register will not take the attempted write (yielding to the reload instead).
- Attempting to write to a shadow register that is locked will result in the setting of HW_OCOTP_CTRL_ERROR.

HW_OCOTP_CTRL_RELOAD_SHADOWS can be set at any time. There is no need to wait for HW_OCOTP_CTRL_BUSY or HW_OCOTP_CTRL_ERROR to be clear.

- In the case of HW_OCOTP_CTRL_BUSY being set due to an active write, the controller will perform the bank opening and shadow reloading immediately after the completion of the write.
- In the case where HW_OCOTP_CTRL_RD_BANK_OPEN is set, the shadow reload will be performed immediately after the banks are closed by software (by clearing HW_OCOTP_CTRL_RD_BANK_OPEN). It should be noted that BUSY will take approximately 33 HCLK cycles to clear, so polling for HW_OCOTP_CTRL_BUSY immediately after clearing HW_OCOTP_CTRL_RD_BANK_OPEN is not recommended.

- In all cases, the controller will clear HW_OCOTP_CTRL_RELOAD_SHADOWS after the successful completion of the operation.

7.3 Behavior During Reset

The OCOTP is always active. The shadow registers described in Section 7.4, "Programmable Registers," automatically load the appropriate OTP contents after reset is deasserted. During this load-time HW_OCOTP_CTRL_BUSY is set. The load time is approximately 32 HCLK cycles after the deassertion of reset. These shadow registers can be reloaded as described in Section 7.2.4, "Shadow Registers and Hardware Capability Bus."

7.4 Programmable Registers

The following registers are available for programmer access and control of the OCOTP.

7.4.1 OTP Controller Control Register Description

The OCOTP Control and Status Register specifies the copy state, as well as the control required for random access of the OTP memory

HW_OCOTP_CTRL	0x000
HW_OCOTP_CTRL_SET	0x004
HW_OCOTP_CTRL_CLR	0x008
HW_OCOTP_CTRL_TOG	0x00C

Table 7-1. HW_OCOTP_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
WR_UNLOCK																RSRVD2		RELOAD_SHADOWS	RD_BANK_OPEN	RSRVD1		ERROR	BUSY	RSRVD0				ADDR			

Table 7-2. HW_OCOTP_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**WR_UNLOCK**	RW	0x0	Write 0x3E77 to enable OTP write accesses. NOTE: This register must be unlocked on a write-by-write basis (a write is initiated when HW_OCOTP_DATA is written), so the UNLOCK bitfield must contain the correct key value during all writes to HW_OCOTP_DATA, otherwise a write shall not be initiated. This field is automatically cleared after a successful write completion (clearing of BUSY). KEY = 0x3E77 Key needed to unlock HW_OCOTP_DATA register.
15:14	**RSRVD2**	RO	0x0	These bits always read back zero.
13	**RELOAD_SHADOWS**	RW	0x0	Set to force re-loading the shadow registers (HW/SW capability and LOCK). This operation will automatically open banks (but will not set RD_BANK_OPEN) and set BUSY. Once the shadow registers have been re-loaded, BUSY and RELOAD_SHADOWS are automatically cleared by the controller. There is no need to set RD_BANK_OPEN to force the reload. If RD_BANK_OPEN is already set, its still possible to set RELOAD_SHADOWS. In this case, the shadow registers will only be updated upon the clearing of RD_BANK_OPEN.
12	**RD_BANK_OPEN**	RW	0x0	Set to open the all the OTP banks for reading. When set, the controller sets BUSY to allow time for the banks to become available (approximatly 32 HCLK cycles later at which time the controller will clear BUSY). Once BUSY is clear, the various OTP words are accessible via their memory mapped address. Note that OTP words which are shadowed, can be read at anytime and will not be affected by RD_BANK_OPEN. This bit must be cleared after reading is complete. Keeping the OTP banks open causes additional current draw. BUSY must be clear before this setting will take affect. If there is a write transaction pending (holding BUSY), then the bank opening sequence will begin automatically upon the previous transaction clears BUSY. Note that if a read is performed from non-shadowed locations without RD_BANK_OPEN, ERROR will be set
11:10	**RSRVD1**	RO	0x0	These bits always read back zero.
9	**ERROR**	RW	0x0	Set by the controller when either an access to a locked region is requested or a read is requested from non-shadowed efuse locations without the banks being open. Must be cleared before any further write access can be performed. This bit can only be set by the controller. This bit is also set if the Pin interface is active and software requests an access to the OTP. In this instance, the ERROR bit cannot be cleared until the Pin interface access has completed. Reset this bit by writing a one to the SCT clear address space and not by a general write.

Table 7-2. HW_OCOTP_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
8	**BUSY**	RO	0x0	OTP controller status bit. When active, no new write access or bank open operations (including RELOAD_SHADOWS) can be performed. Cleared by controller when access complete (for writes), or the banks are open (for reads). After reset (or after setting RELOAD_SHADOWS), this bit is set by the controller until the HW/SW and LOCK registers are successfully copied, after which time it is automatically cleared by the controller.
7:5	**RSRVD0**	RO	0x0	These bits always read back zero.
4:0	**ADDR**	RW	0x0	OTP write access address register. Specifies one of 32 word address locations (0x00 - 0x1F). If a valid write is accepted by the controller (see HW_OCOTP_DATA for details on what constitutes a valid write), the controller makes an internal copy of this value (to avoid the OTP programming being corrupted). This internal copy will not update until the write access is complete.

DESCRIPTION:

The OCOTP Control and Status Register provides the necessary software interface for performing read and write operations to the On-Chip OTP (One-Time Programmable ROM). The control fields such as WR_UNLOCK, ADDR and BUSY/ERROR may be used in conjuction with the HW_OCOTP_DATA register to perform write operations. Read operations are performed via the direct memory mapped registers. In the cases where OTP values are shadowed into local memory storage, the memory mapped location can be read directly. In the cases where the OTP values are not shadowed into local memory, the read-preparation sequence involving RD_BANK_OPEN and BUSY/ERROR fields must be used before performing the read.

EXAMPLE:

```
Empty Example.
```

7.4.2 OTP Controller Write Data Register Description

The OCOTP Data Register is used for OTP Programming

HW_OCOTP_DATA 0x010

Table 7-3. HW_OCOTP_DATA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DATA																															

Table 7-4. HW_OCOTP_DATA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**DATA**	RW	0x0	Used to initiate a write to OTP. Please see the "Software Write Sequence" section for operating details.

DESCRIPTION:

This register is used in conjuction with HW_OCOTP_CTRL to perform one-time writes to the OTP. Please see the "Software Write Sequence" section for operating details.

EXAMPLE:

```
Empty Example.
```

7.4.3 Value of OTP Bank0 Word0 (Customer) Description

OTP banks must be open via HW_OCOTP_CTRL[RD_BANK_OPEN] before reading this register. Reading this register without having HW_OCOTP_CTRL[RD_BANK_OPEN] set and HW_OCOTP_CTRL[BUSY] clear, will result in HW_OCOTP_CTRL[ERROR] being set and 0xBADA_BADA being returned.

HW_OCOTP_CUST0 0x020

Table 7-5. HW_OCOTP_CUST0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-6. HW_OCOTP_CUST0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Reflects value of OTP Bank 0, word 0 (ADDR = 0x00)

DESCRIPTION:

Non-shadowed memory mapped access to OTP Bank 0, word 0 (ADDR = 0x00).

EXAMPLE:

```
Empty Example.
```

7.4.4 Value of OTP Bank0 Word1 (Customer) Description

OTP banks must be open via HW_OCOTP_CTRL[RD_BANK_OPEN] before reading this register. Reading this register without having HW_OCOTP_CTRL[RD_BANK_OPEN] set and HW_OCOTP_CTRL[BUSY] clear, will result in HW_OCOTP_CTRL[ERROR] being set and 0xBADA_BADA being returned.

HW_OCOTP_CUST1 0x030

Table 7-7. HW_OCOTP_CUST1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-8. HW_OCOTP_CUST1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Reflects value of OTP Bank 0, word 1 (ADDR = 0x01)

DESCRIPTION:

Non-shadowed memory mapped access to OTP Bank 0, word 1 (ADDR = 0x01).

EXAMPLE:

```
Empty Example.
```

7.4.5 Value of OTP Bank0 Word2 (Customer) Description

OTP banks must be open via HW_OCOTP_CTRL[RD_BANK_OPEN] before reading this register. Reading this register without having HW_OCOTP_CTRL[RD_BANK_OPEN] set and HW_OCOTP_CTRL[BUSY] clear, will result in HW_OCOTP_CTRL[ERROR] being set and 0xBADA_BADA being returned.

HW_OCOTP_CUST2 0x040

Table 7-9. HW_OCOTP_CUST2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-10. HW_OCOTP_CUST2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Reflects value of OTP Bank 0, word 2 (ADDR = 0x02)

DESCRIPTION:

Non-shadowed memory mapped access to OTP Bank 0, word 2 (ADDR = 0x02).

EXAMPLE:

```
Empty Example.
```

7.4.6 Value of OTP Bank0 Word3 (Customer) Description

OTP banks must be open via HW_OCOTP_CTRL[RD_BANK_OPEN] before reading this register. Reading this register without having HW_OCOTP_CTRL[RD_BANK_OPEN] set and HW_OCOTP_CTRL[BUSY] clear, will result in HW_OCOTP_CTRL[ERROR] being set and 0xBADA_BADA being returned.

HW_OCOTP_CUST3 0x050

Table 7-11. HW_OCOTP_CUST3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-12. HW_OCOTP_CUST3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Reflects value of OTP Bank 0, word 3 (ADDR = 0x03)

DESCRIPTION:

Non-shadowed memory mapped access to OTP Bank 0, word 3 (ADDR = 0x03).

EXAMPLE:

```
Empty Example.
```

7.4.7 Value of OTP Bank0 Word4 (Crypto Key) Description

OTP banks must be open via HW_OCOTP_CTRL[RD_BANK_OPEN] before reading this register. Reading this register without having HW_OCOTP_CTRL[RD_BANK_OPEN] set and HW_OCOTP_CTRL[BUSY] clear, will result in HW_OCOTP_CTRL[ERROR] being set and 0xBADA_BADA being returned.

HW_OCOTP_CRYPTO0 0x060

Table 7-13. HW_OCOTP_CRYPTO0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-14. HW_OCOTP_CRYPTO0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Reflects value of OTP Bank 0, word 4 (ADDR = 0x04). If LOCK[CRYPTOKEY] is set, returns 0xBADA_BADA and sets HW_OCOTP_CTRL[ERROR]

DESCRIPTION:

Non-shadowed memory mapped access to OTP Bank 0, word 4 (ADDR = 0x04).

EXAMPLE:

```
Empty Example.
```

7.4.8 Value of OTP Bank0 Word5 (Crypto Key) Description

OTP banks must be open via HW_OCOTP_CTRL[RD_BANK_OPEN] before reading this register. Reading this register without having HW_OCOTP_CTRL[RD_BANK_OPEN] set and

HW_OCOTP_CTRL[BUSY] clear, will result in HW_OCOTP_CTRL[ERROR] being set and 0xBADA_BADA being returned.

HW_OCOTP_CRYPTO1 0x070

Table 7-15. HW_OCOTP_CRYPTO1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-16. HW_OCOTP_CRYPTO1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Reflects value of OTP Bank 0, word 5 (ADDR = 0x05). If LOCK[CRYPTOKEY] is set, returns 0xBADA_BADA and sets HW_OCOTP_CTRL[ERROR]

DESCRIPTION:

Non-shadowed memory mapped access to OTP Bank 0, word 5 (ADDR = 0x05).

EXAMPLE:

```
Empty Example.
```

7.4.9 Value of OTP Bank0 Word6 (Crypto Key) Description

OTP banks must be open via HW_OCOTP_CTRL[RD_BANK_OPEN] before reading this register. Reading this register without having HW_OCOTP_CTRL[RD_BANK_OPEN] set and HW_OCOTP_CTRL[BUSY] clear, will result in HW_OCOTP_CTRL[ERROR] being set and 0xBADA_BADA being returned.

HW_OCOTP_CRYPTO2 0x080

Table 7-17. HW_OCOTP_CRYPTO2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-18. HW_OCOTP_CRYPTO2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Reflects value of OTP Bank 0, word 6 (ADDR = 0x06). If LOCK[CRYPTOKEY] is set, returns 0xBADA_BADA and sets HW_OCOTP_CTRL[ERROR]

DESCRIPTION:

Non-shadowed memory mapped access to OTP Bank 0, word 6 (ADDR = 0x06).

EXAMPLE:

```
Empty Example.
```

7.4.10 Value of OTP Bank0 Word7 (Crypto Key) Description

OTP banks must be open via HW_OCOTP_CTRL[RD_BANK_OPEN] before reading this register. Reading this register without having HW_OCOTP_CTRL[RD_BANK_OPEN] set and HW_OCOTP_CTRL[BUSY] clear, will result in HW_OCOTP_CTRL[ERROR] being set and 0xBADA_BADA being returned.

HW_OCOTP_CRYPTO3 0x090

Table 7-19. HW_OCOTP_CRYPTO3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-20. HW_OCOTP_CRYPTO3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Reflects value of OTP Bank 0, word 7 (ADDR = 0x07). If LOCK[CRYPTOKEY] is set, returns 0xBADA_BADA and sets HW_OCOTP_CTRL[ERROR]

DESCRIPTION:

Non-shadowed memory mapped access to OTP Bank 0, word 7 (ADDR = 0x07).

EXAMPLE:

```
Empty Example.
```

7.4.11 HW Capability Shadow Register 0 Description

Copied from the OTP automatically after reset. Can be re-loaded by setting HW_OCOTP_CTRL[RELOAD_SHADOWS]

HW_OCOTP_HWCAP0 0x0A0

Table 7-21. HW_OCOTP_HWCAP0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-22. HW_OCOTP_HWCAP0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RW	0x0	Shadow register for HW capability bits 31:0 (copy of OTP bank 1, word 0 (ADDR = 0x08)). These bits become read-only after the HW_OCOTP_LOCK[HWSW_SHADOW] or HW_OCOTP_LOCK[HWSW_SHADOW_ALT] bit is set.

DESCRIPTION:

Shadowed memory mapped access to OTP bank 1, word 0 (ADDR = 0x08).

EXAMPLE:

```
Empty Example.
```

7.4.12 HW Capability Shadow Register 1 Description

Copied from the OTP automatically after reset. Can be re-loaded by setting HW_OCOTP_CTRL[RELOAD_SHADOWS]

HW_OCOTP_HWCAP1 0x0B0

Table 7-23. HW_OCOTP_HWCAP1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-24. HW_OCOTP_HWCAP1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RW	0x0	Shadow register for HW capability bits 63:32 (copy of OTP bank 1, word 1 (ADDR = 0x09)). These bits become read-only after the HW_OCOTP_LOCK[HWSW_SHADOW] or HW_OCOTP_LOCK[HWSW_SHADOW_ALT] bit is set.

DESCRIPTION:

Shadowed memory mapped access to OTP bank 1, word 1 (ADDR = 0x09).

EXAMPLE:

```
Empty Example.
```

7.4.13 HW Capability Shadow Register 2 Description

Copied from the OTP automatically after reset. Can be re-loaded by setting HW_OCOTP_CTRL[RELOAD_SHADOWS]

HW_OCOTP_HWCAP2 0x0C0

Table 7-25. HW_OCOTP_HWCAP2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-26. HW_OCOTP_HWCAP2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RW	0x0	Shadow register for HW capability bits 95:64 (copy of OTP bank 1, word 2 (ADDR = 0x0A)). These bits become read-only after the HW_OCOTP_LOCK[HWSW_SHADOW] or HW_OCOTP_LOCK[HWSW_SHADOW_ALT] bit is set.

DESCRIPTION:

Shadowed memory mapped access to OTP bank 1, word 2 (ADDR = 0x0A).

EXAMPLE:

```
Empty Example.
```

7.4.14 HW Capability Shadow Register 3 Description

Copied from the OTP automatically after reset. Can be re-loaded by setting HW_OCOTP_CTRL[RELOAD_SHADOWS]

HW_OCOTP_HWCAP3 0x0D0

Table 7-27. HW_OCOTP_HWCAP3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-28. HW_OCOTP_HWCAP3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RW	0x0	Shadow register for HW capability bits 127:96 (copy of OTP bank 1, word 3 (ADDR = 0x0B)). These bits become read-only after the HW_OCOTP_LOCK[HWSW_SHADOW] or HW_OCOTP_LOCK[HWSW_SHADOW_ALT] bit is set.

DESCRIPTION:

Shadowed memory mapped access to OTP bank 1, word 3 (ADDR = 0x0B).

EXAMPLE:

```
Empty Example.
```

7.4.15 HW Capability Shadow Register 4 Description

Copied from the OTP automatically after reset. Can be re-loaded by setting HW_OCOTP_CTRL[RELOAD_SHADOWS]

HW_OCOTP_HWCAP4 0x0E0

Table 7-29. HW_OCOTP_HWCAP4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-30. HW_OCOTP_HWCAP4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RW	0x0	Shadow register for HW capability bits 159:128 (copy of OTP bank 1, word 4 (ADDR = 0x0C)). These bits become read-only after the HW_OCOTP_LOCK[HWSW_SHADOW] or HW_OCOTP_LOCK[HWSW_SHADOW_ALT] bit is set.

DESCRIPTION:

Shadowed memory mapped access to OTP bank 1, word 4 (ADDR = 0x0C).

EXAMPLE:

```
Empty Example.
```

7.4.16 HW Capability Shadow Register 5 Description

Copied from the OTP automatically after reset. Can be re-loaded by setting HW_OCOTP_CTRL[RELOAD_SHADOWS]

HW_OCOTP_HWCAP5 0x0F0

Table 7-31. HW_OCOTP_HWCAP5

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-32. HW_OCOTP_HWCAP5 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RW	0x0	Shadow register for HW capability bits 191:160 (copy of OTP bank 1, word 5 (ADDR = 0x0D)). These bits become read-only after the HW_OCOTP_LOCK[HWSW_SHADOW] or HW_OCOTP_LOCK[HWSW_SHADOW_ALT] bit is set.

DESCRIPTION:

Shadowed memory mapped access to OTP bank 1, word 5 (ADDR = 0x0D).

EXAMPLE:

```
Empty Example.
```

7.4.17 SW Capability Shadow Register Description

Copied from the OTP automatically after reset. Can be re-loaded by setting HW_OCOTP_CTRL[RELOAD_SHADOWS]

HW_OCOTP_SWCAP 0x100

Table 7-33. HW_OCOTP_SWCAP

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-34. HW_OCOTP_SWCAP Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RW	0x0	Shadow register for SW capability bits 31:0 (copy of OTP bank 1, word 6 (ADDR = 0x0E)). These bits become read-only after the HW_OCOTP_LOCK[HWSW_SHADOW] or HW_OCOTP_LOCK[HWSW_SHADOW_ALT] bit is set.

DESCRIPTION:

Shadowed memory mapped access to OTP bank 1, word 6 (ADDR = 0x0E).

EXAMPLE:

```
Empty Example.
```

7.4.18 Customer Capability Shadow Register Description

Copied from the OTP automatically after reset. Can be re-loaded by setting HW_OCOTP_CTRL[RELOAD_SHADOWS]

HW_OCOTP_CUSTCAP 0x110

Table 7-35. HW_OCOTP_CUSTCAP

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CUST_DISABLE_WMADRM9	CUST_DISABLE_JANUSDRM10	RSRVD1																									ENABLE_SJTAG_12MA_DRIVE	USE_PARALLEL_JTAG	RTC_XTAL_32768_PRESENT	RTC_XTAL_32000_PRESENT	RSRVD0	

Table 7-36. HW_OCOTP_CUSTCAP Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**CUST_DISABLE_WMADRM9**	RW	0x0	Blow to disable WMA DRM9.
30	**CUST_DISABLE_JANUSDRM10**	RW	0x0	Blow to disable WMA Janus DRM10.
29:5	**RSRVD1**	RW	0x0	Reserved - do not blow these bits.
4	**ENABLE_SJTAG_12MA_DRIVE**	RW	0x0	Blow to force the 1-wire DEBUG (serial JTAG) pin to drive 12mA, the default is 8mA (see ENABLE_PJTAG_12MA_DRIVE in the ROM0 register for 6-wire parallel JTAG). This is a hardware override which will cause the drive select bits in the PINCTRL block to reset to the 12mA drive settings rather than the normal default of 8mA. The user is still free to reprogram these bits to other drive levels.
3	**USE_PARALLEL_JTAG**	RW	0x0	During JTAG boot mode, the ROM reads this bit, then inverts it, and writes the value to the HW_DIGCTL_CTRL_USE_SERIAL_JTAG bit. If this bit is one, indicating parallel JTAG mode is selected, a zero is written to the DIGCTL USE_SERIAL_JTAG bit which places the device into 6-wire JTAG mode, and if this bit is zero, a one instead is written causing the SJTAG block to switch to the 1-wire serial JTAG mode.
2	**RTC_XTAL_32768_PRESENT**	RW	0x0	Blow to indicate the presence of an optional 32.768KHz crystal off-chip.
1	**RTC_XTAL_32000_PRESENT**	RW	0x0	Blow to indicate the presence of an optional 32.000KHz crystal off-chip.
0	**RSRVD0**	RW	0x0	Reserved - do not blow these bits.

DESCRIPTION:

Shadowed memory mapped access to OTP bank 1, word 7 (ADDR = 0x0F).

EXAMPLE:

```
Empty Example.
```

7.4.19 LOCK Shadow Register OTP Bank 2 Word 0 Description

Shadow register for OCOTP Lock Status Value (ADDR = 0x10). Copy of the state of the OTP lock regions. Copied from the OTP upon reset. Can be re-loaded by setting HW_OCOTP_CTRL[RELOAD_SHADOWS]

HW_OCOTP_LOCK 0x120

Table 7-37. HW_OCOTP_LOCK

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ROM7	ROM6	ROM5	ROM4	ROM3	ROM2	ROM1	ROM0	HWSW_SHADOW_ALT	CRYPTODCP_ALT	CRYPTOKEY_ALT	PIN	OPS	UN2	UN1	UN0	UNALLOCATED					ROM_SHADOW	CUSTCAP	HWSW	CUSTCAP_SHADOW	HWSW_SHADOW	CRYPTODCP	CRYPTOKEY	CUST3	CUST2	CUST1	CUST0

Table 7-38. HW_OCOTP_LOCK Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**ROM7**	RO	0x0	Status of ROM use region write lock bits (ADDR = 0x1F). When set, word 0x1F in the ROM region is locked.
30	**ROM6**	RO	0x0	Status of ROM use region write lock bits (ADDR = 0x1E). When set, word 0x1E in the ROM region is locked.
29	**ROM5**	RO	0x0	Status of ROM use region write lock bits (ADDR = 0x1D). When set, word 0x1D in the ROM region is locked.
28	**ROM4**	RO	0x0	Status of ROM use region write lock bits (ADDR = 0x1C). When set, word 0x1C in the ROM region is locked.
27	**ROM3**	RO	0x0	Status of ROM use region write lock bits (ADDR = 0x1B). When set, word 0x1B in the ROM region is locked.
26	**ROM2**	RO	0x0	Status of ROM use region write lock bits (ADDR = 0x1A). When set, word 0x1A in the ROM region is locked.
25	**ROM1**	RO	0x0	Status of ROM use region write lock bits (ADDR = 0x19). When set, word 0x19 in the ROM region is locked.
24	**ROM0**	RO	0x0	Status of ROM use region write lock bits (ADDR = 0x18). When set, word 0x18 in the ROM region is locked.
23	**HWSW_SHADOW_ALT**	RO	0x0	Status of alternate bit for HWSW_SHADOW lock
22	**CRYPTODCP_ALT**	RO	0x0	Status of alternate bit for CRYPTODCP lock
21	**CRYPTOKEY_ALT**	RO	0x0	Status of alternate bit for CRYPTOKEY lock
20	**PIN**	RO	0x0	Status of Pin access lock bit. When set, pin access is disabled.
19	**OPS**	RO	0x0	Status of SGTL-OPS region (ADDR = 0x11-0x14) write lock bit. When set, region is locked.
18	**UN2**	RO	0x0	Status of un-assigned (ADDR = 0x17) write-lock bit. When set, un-asigned word at OTP address 0x17 is locked.
17	**UN1**	RO	0x0	Status of un-assigned (ADDR = 0x16) write-lock bit. When set, un-asigned word at OTP address 0x16 is locked.

Table 7-38. HW_OCOTP_LOCK Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
16	**UN0**	RO	0x0	Status of un-assigned (ADDR = 0x15) write-lock bit. When set, un-asigned word at OTP address 0x15 is locked.
15:11	**UNALLOCATED**	RO	0x0	Value of un-used portion of LOCK word
10	**ROM_SHADOW**	RO	0x0	Status of ROM region shadow register lock. When set, over-ride of ROM-region shadow bits is blocked.
9	**CUSTCAP**	RO	0x0	Status of Customer Capability region (ADDR = 0x0F) write lock bit. When set, region is locked.
8	**HWSW**	RO	0x0	Status of HW/SW region (ADDR = 0x08-0x0E) write lock bit. When set, region is locked.
7	**CUSTCAP_SHADOW**	RO	0x0	Status of Customer Capability shadow register lock. When set, over-ride of customer capabality shadow bits is blocked.
6	**HWSW_SHADOW**	RO	0x0	Status of HW/SW Capability shadow register lock. When set, over-ride of HW/SW capabality shadow bits is blocked.
5	**CRYPTODCP**	RO	0x0	Status of read lock bit for DCP APB crypto access. When set, the DCP will disallow reads of its crypto keys via its APB interface.
4	**CRYPTOKEY**	RO	0x0	Status of crypto key region (ADDR = 0x04-0x07) read/write lock bit. When set, region is locked.
3	**CUST3**	RO	0x0	Status of customer region word (ADDR = 0x03) write lock bit. When set, the region is locked.
2	**CUST2**	RO	0x0	Status of customer region word (ADDR = 0x02) write lock bit. When set, the region is locked.
1	**CUST1**	RO	0x0	Status of customer region word (ADDR = 0x01) write lock bit. When set, the region is locked.
0	**CUST0**	RO	0x0	Status of customer region word (ADDR = 0x00) write lock bit. When set, the region is locked.

DESCRIPTION:

Shadowed memory mapped access to OTP bank 2, word 0 (ADDR = 0x10).

EXAMPLE:

```
Empty Example.
```

7.4.20 Value of OTP Bank2 Word1 (Freescale OPS0) Description

OTP banks must be open via HW_OCOTP_CTRL[RD_BANK_OPEN] before reading this register. Reading this register without having HW_OCOTP_CTRL[RD_BANK_OPEN] set and HW_OCOTP_CTRL[BUSY] clear, will result in HW_OCOTP_CTRL[ERROR] being set and 0xBADA_BADA being returned.

HW_OCOTP_OPS0 0x130

Table 7-39. HW_OCOTP_OPS0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-40. HW_OCOTP_OPS0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Reflects value of OTP Bank 2, word 1 (ADDR = 0x11)

DESCRIPTION:

Shadowed memory mapped access to OTP Bank 2, word 1 (ADDR = 0x11).

EXAMPLE:

```
Empty Example.
```

7.4.21 Value of OTP Bank2 Word2 (Freescale OPS1) Description

OTP banks must be open via HW_OCOTP_CTRL[RD_BANK_OPEN] before reading this register. Reading this register without having HW_OCOTP_CTRL[RD_BANK_OPEN] set and HW_OCOTP_CTRL[BUSY] clear, will result in HW_OCOTP_CTRL[ERROR] being set and 0xBADA_BADA being returned.

HW_OCOTP_OPS1 0x140

Table 7-41. HW_OCOTP_OPS1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-42. HW_OCOTP_OPS1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Reflects value of OTP Bank 2, word 2 (ADDR = 0x12)

DESCRIPTION:

Shadowed memory mapped access to OTP Bank 2, word 2 (ADDR = 0x12).

EXAMPLE:

```
Empty Example.
```

7.4.22 Value of OTP Bank2 Word3 (Freescale OPS2) Description

OTP banks must be open via HW_OCOTP_CTRL[RD_BANK_OPEN] before reading this register. Reading this register without having HW_OCOTP_CTRL[RD_BANK_OPEN] set and HW_OCOTP_CTRL[BUSY] clear, will result in HW_OCOTP_CTRL[ERROR] being set and 0xBADA_BADA being returned.

HW_OCOTP_OPS2 0x150

Table 7-43. HW_OCOTP_OPS2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-44. HW_OCOTP_OPS2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Reflects value of OTP Bank 2, word 3 (ADDR = 0x13)

DESCRIPTION:

Shadowed memory mapped access to OTP Bank 2, word 3 (ADDR = 0x13).

EXAMPLE:

```
Empty Example.
```

7.4.23 Value of OTP Bank2 Word4 (Freescale OPS3) Description

OTP banks must be open via HW_OCOTP_CTRL[RD_BANK_OPEN] before reading this register. Reading this register without having HW_OCOTP_CTRL[RD_BANK_OPEN] set and HW_OCOTP_CTRL[BUSY] clear, will result in HW_OCOTP_CTRL[ERROR] being set and 0xBADA_BADA being returned.

HW_OCOTP_OPS3 0x160

Table 7-45. HW_OCOTP_OPS3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-46. HW_OCOTP_OPS3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Reflects value of OTP Bank 2, word 4 (ADDR = 0x14)

DESCRIPTION:

Shadowed memory mapped access to OTP Bank 2, word 4 (ADDR = 0x14).

EXAMPLE:

```
Empty Example.
```

7.4.24 Value of OTP Bank2 Word5 (Unassigned0) Description

OTP banks must be open via HW_OCOTP_CTRL[RD_BANK_OPEN] before reading this register. Reading this register without having HW_OCOTP_CTRL[RD_BANK_OPEN] set and HW_OCOTP_CTRL[BUSY] clear, will result in HW_OCOTP_CTRL[ERROR] being set and 0xBADA_BADA being returned.

HW_OCOTP_UN0 0x170

Table 7-47. HW_OCOTP_UN0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-48. HW_OCOTP_UN0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Reflects value of OTP Bank 2, word 5 (ADDR = 0x15)

DESCRIPTION:

Non-shadowed memory mapped access to OTP Bank 2, word 5 (ADDR = 0x15).

EXAMPLE:

```
Empty Example.
```

7.4.25 Value of OTP Bank2 Word6 (Unassigned1) Description

OTP banks must be open via HW_OCOTP_CTRL[RD_BANK_OPEN] before reading this register. Reading this register without having HW_OCOTP_CTRL[RD_BANK_OPEN] set and HW_OCOTP_CTRL[BUSY] clear, will result in HW_OCOTP_CTRL[ERROR] being set and 0xBADA_BADA being returned.

HW_OCOTP_UN1 0x180

Table 7-49. HW_OCOTP_UN1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-50. HW_OCOTP_UN1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Reflects value of OTP Bank 2, word 6 (ADDR = 0x16)

DESCRIPTION:

Non-shadowed memory mapped access to OTP Bank 2, word 6 (ADDR = 0x16).

EXAMPLE:

```
Empty Example.
```

7.4.26 Value of OTP Bank2 Word7 (Unassigned2) Description

OTP banks must be open via HW_OCOTP_CTRL[RD_BANK_OPEN] before reading this register. Reading this register without having HW_OCOTP_CTRL[RD_BANK_OPEN] set and

HW_OCOTP_CTRL[BUSY] clear, will result in HW_OCOTP_CTRL[ERROR] being set and 0xBADA_BADA being returned.

HW_OCOTP_UN2 0x190

Table 7-51. HW_OCOTP_UN2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-52. HW_OCOTP_UN2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RO	0x0	Reflects value of OTP Bank 2, word 7 (ADDR = 0x17)

DESCRIPTION:

Non-shadowed memory mapped access to OTP Bank 2, word 7 (ADDR = 0x17).

EXAMPLE:

```
Empty Example.
```

7.4.27 Shadow Register for OTP Bank3 Word0 (ROM Use 0) Description

Copied from the OTP automatically after reset. Can be re-loaded by setting HW_OCOTP_CTRL[RELOAD_SHADOWS].

HW_OCOTP_ROM0 0x1A0

Table 7-53. HW_OCOTP_ROM0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BOOT_MODE								ENABLE_PJTAG_12MA_DRIVE	USE_PARALLEL_JTAG	SD_POWER_GATE_GPIO		SD_POWER_UP_DELAY				SD_BUS_WIDTH				SSP_SCK_INDEX				RSRVD3	DISABLE_SPI_NOR_FAST_ READ	ENABLE_USB_BOOT_SERIAL_NUM	ENABLE_UNENCRYPTED_ BOOT	SD_MBR_BOOT	RSRVD2	RSRVD1	RSRVD0

Table 7-54. HW_OCOTP_ROM0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**BOOT_MODE**	RW	0x0	Encoded boot mode.
23	**ENABLE_PJTAG_12MA_DRIVE**	RW	0x0	Blow to force the 6-wire PJTAG pins to drive 12mA, default is 4mA (note that SJTAG is fixed at 8mA). Blowing this bit causes the ROM to program all six parallel JTAG pins to drive 12mA via the pin control registers.
22	**USE_PARALLEL_JTAG**	RW	0x0	During JTAG boot mode, the ROM reads this bit, then inverts it, and writes the value to the HW_DIGCTL_CTRL_USE_SERIAL_JTAG bit. If this bit is one, indicating parallel JTAG mode is selected, a zero is written to the DIGCTL USE_SERIAL_JTAG bit which places the device into 6-wire JTAG mode, and if this bit is zero, a one instead is written causing the SJTAG block to switch to the 1-wire serial JTAG mode.
21:20	**SD_POWER_GATE_GPIO**	RW	0x0	SD card power gate GPIO pin select: 00 - PWM0, 01 - LCD_DOTCLK, 10 - PWM3, 11 - NO_GATE.
19:14	**SD_POWER_UP_DELAY**	RW	0x0	SD card power up delay required after enabling GPIO power gate: 000000 - 0 ms, 000001 - 10 ms, 000010 - 20 ms, 111111 - 630 ms.
13:12	**SD_BUS_WIDTH**	RW	0x0	SD card bus width: 00 - 4-bit, 01 - 1-bit, 10 - 8-bit, 11 - reserved.
11:8	**SSP_SCK_INDEX**	RW	0x0	Index to the SSP clock speed
7	**RSRVD3**	RW	0x0	Reserved - do not blow this bit.
6	**DISABLE_SPI_NOR_FAST_READ**	RW	0x0	Blow to disable SPI NOR fast reads which are used by default.
5	**ENABLE_USB_BOOT_SERIAL_NUM**	RW	0x0	Blow to enable USB boot serial number.
4	**ENABLE_UNENCRYPTED_BOOT**	RW	0x0	Blow to enable unencrypted boot modes.
3	**SD_MBR_BOOT**	RW	0x0	Blow to enable master boot record (MBR) boot mode for SD boot.
2	**RSRVD2**	RW	0x0	Reserved - do not blow this bit.
1	**RSRVD1**	RW	0x0	Reserved - do not blow this bit.
0	**RSRVD0**	RW	0x0	Reserved - do not blow this bit.

DESCRIPTION:

Shadowed memory mapped access to OTP Bank 3, word 0 (ADDR = 0x18).

EXAMPLE:

```
Empty Example.
```

7.4.28 Shadow Register for OTP Bank3 Word1 (ROM Use 1) Description

Copied from the OTP automatically after reset. Can be re-loaded by setting HW_OCOTP_CTRL[RELOAD_SHADOWS].

HW_OCOTP_ROM1 0x1B0

Table 7-55. HW_OCOTP_ROM1

31:30	29:28	27:26	25	24	23	22	21	20	19	18	17	16	15	14	13:12	11:10	09:08	07:04	03:02	01:00
RSRVD1	USE_ALT_GPMI_RDY3	USE_ALT_GPMI_CE3	USE_ALT_GPMI_RDY2	USE_ALT_GPMI_CE2	ENABLE_NAND3_CE_RDY_PULLUP	ENABLE_NAND2_CE_RDY_PULLUP	ENABLE_NAND1_CE_RDY_PULLUP	ENABLE_NAND0_CE_RDY_PULLUP	UNTOUCH_INTERNAL_SSP_PULLUP	SSP2_EXT_PULLUP	SSP1_EXT_PULLUP	SD_INCREASE_INIT_SEQ_TIME	SD_INIT_SEQ_2_ENABLE	SD_CMD0_DISABLE	SD_INIT_SEQ_1_DISABLE	USE_ALT_SSP1_DATA4-7	BOOT_SEARCH_COUNT	RSRVD0	NUMBER_OF_NANDS

Table 7-56. HW_OCOTP_ROM1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD1**	RW	0x0	Reserved - do not blow this bit.
29:28	**USE_ALT_GPMI_RDY3**	RW	0x0	These bits are used by ROM NAND driver to enable one of 3 alternate pins for GPMI_RDY3. 00-GPMI_RDY3, 01-PWM2 and 10-LCD_DOTCK.
27:26	**USE_ALT_GPMI_CE3**	RW	0x0	These bits are used by ROM NAND driver to enable one of 4 alternate pins for GPMI_CE3. 00-GPMI_D15, 01-LCD_RESET, 10-SSP_DETECT and 11-ROTARYB.
25	**USE_ALT_GPMI_RDY2**	RW	0x0	If the bit is blown then ROM NAND driver will enable alternate pins for GPMI_RDY2.
24	**USE_ALT_GPMI_CE2**	RW	0x0	If the bit is blown then ROM NAND driver will enable alternate pins for GPMI_CE2.
23	**ENABLE_NAND3_CE_RDY_PULLUP**	RW	0x0	If the bit is blown then ROM NAND driver will enable internal pull-up for GPMI_CE3 and GPMI_RDY3 pins.
22	**ENABLE_NAND2_CE_RDY_PULLUP**	RW	0x0	If the bit is blown then ROM NAND driver will enable internal pull-up for GPMI_CE2 and GPMI_RDY2 pins.
21	**ENABLE_NAND1_CE_RDY_PULLUP**	RW	0x0	If the bit is blown then ROM NAND driver will enable internal pull-up for GPMI_CE1 and GPMI_RDY1 pins.
20	**ENABLE_NAND0_CE_RDY_PULLUP**	RW	0x0	If the bit is blown then ROM NAND driver will enable internal pull-up for GPMI_CE0 and GPMI_RDY0 pins.
19	**UNTOUCH_INTERNAL_SSP_PULLUP**	RW	0x0	If this bit is blown then internal pull-ups for SSP are neither enabled nor disabled. This bit is used only if external pull-ups are implemented and ROM1:18 and/or ROM1:17 are blown.
18	**SSP2_EXT_PULLUP**	RW	0x0	Blow to indicate external pull-ups implemented for SSP2.
17	**SSP1_EXT_PULLUP**	RW	0x0	Blow to indicate external pull-ups implemented for SSP1.
16	**SD_INCREASE_INIT_SEQ_TIME**	RW	0x0	Blow to increase the SD card initialization sequence time from 1ms (default) to 4ms.
15	**SD_INIT_SEQ_2_ENABLE**	RW	0x0	Blow to enable the second initialization sequence for SD boot.

Table 7-56. HW_OCOTP_ROM1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
14	**SD_CMD0_DISABLE**	RW	0x0	Cmd0 (reset cmd) is called by default to reset the SD card during startup. Blow this bit to not reset the card during SD boot.
13	**SD_INIT_SEQ_1_DISABLE**	RW	0x0	Blow to disable the first initialization sequence for SD.
12	**USE_ALT_SSP1_DATA4-7**	RW	0x0	This bit is blown to enable alternate pin use for SSP1 data lines 4-7.
11:8	**BOOT_SEARCH_COUNT**	RW	0x0	Number of 64 page blocks that should be read by the boot loader.
7:3	**RSRVD0**	RW	0x0	Reserved - do not blow these bits.
2:0	**NUMBER_OF_NANDS**	RW	0x0	Encoded value indicates number of external NAND devices (0 to 7). Zero indicates ROM will probe for the number of NAND devices connected in the system.

DESCRIPTION:

Shadowed memory mapped access to OTP Bank 3, word 1 (ADDR = 0x19).

EXAMPLE:

```
Empty Example.
```

7.4.29 Shadow Register for OTP Bank3 Word2 (ROM Use 2) Description

Copied from the OTP automatically after reset. Can be re-loaded by setting HW_OCOTP_CTRL[RELOAD_SHADOWS].

HW_OCOTP_ROM2 0x1C0

Table 7-57. HW_OCOTP_ROM2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
USB_VID																USB_PID															

Table 7-58. HW_OCOTP_ROM2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**USB_VID**	RW	0x0	USB Vendor ID.
15:0	**USB_PID**	RW	0x0	USB Product ID

DESCRIPTION:

Shadowed memory mapped access to OTP Bank 3, word 2 (ADDR = 0x1A).

EXAMPLE:

```
Empty Example.
```

7.4.30 Shadow Register for OTP Bank3 Word3 (ROM Use 3) Description

Copied from the OTP automatically after reset. Can be re-loaded by setting HW_OCOTP_CTRL[RELOAD_SHADOWS].

HW_OCOTP_ROM3 0x1D0

Table 7-59. HW_OCOTP_ROM3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																						RSRVD0									

Table 7-60. HW_OCOTP_ROM3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:10	**RSRVD1**	RW	0x0	Reserved - do not blow these bits.
9:0	**RSRVD0**	RW	0x0	SDK reserved bits.

DESCRIPTION:

Shadowed memory mapped access to OTP Bank 3, word 3 (ADDR = 0x1B).

EXAMPLE:

```
Empty Example.
```

7.4.31 Shadow Register for OTP Bank3 Word4 (ROM Use 4) Description

Copied from the OTP automatically after reset. Can be re-loaded by setting HW_OCOTP_CTRL[RELOAD_SHADOWS].

HW_OCOTP_ROM4 0x1E0

Table 7-61. HW_OCOTP_ROM4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-62. HW_OCOTP_ROM4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RW	0x0	Shadow register for ROM-use word4 (Copy of OTP Bank 3, word 4 (ADDR = 0x1C)). These bits become read-only after the HW_OCOTP_LOCK[ROM_SHADOW] bit is set.

DESCRIPTION:

Shadowed memory mapped access to OTP Bank 3, word 4 (ADDR = 0x1C).

EXAMPLE:

```
Empty Example.
```

7.4.32 Shadow Register for OTP Bank3 Word5 (ROM Use 5) Description

Copied from the OTP automatically after reset. Can be re-loaded by setting HW_OCOTP_CTRL[RELOAD_SHADOWS].

HW_OCOTP_ROM5 0x1F0

Table 7-63. HW_OCOTP_ROM5

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-64. HW_OCOTP_ROM5 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RW	0x0	Shadow register for ROM-use word5 (Copy of OTP Bank 3, word 5 (ADDR = 0x1D)). These bits become read-only after the HW_OCOTP_LOCK[ROM_SHADOW] bit is set.

DESCRIPTION:

Shadowed memory mapped access to OTP Bank 3, word 5 (ADDR = 0x1D).

EXAMPLE:

```
Empty Example.
```

7.4.33 Shadow Register for OTP Bank3 Word6 (ROM Use 6) Description

Copied from the OTP automatically after reset. Can be re-loaded by setting HW_OCOTP_CTRL[RELOAD_SHADOWS].

HW_OCOTP_ROM6 0x200

Table 7-65. HW_OCOTP_ROM6

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BITS																															

Table 7-66. HW_OCOTP_ROM6 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**BITS**	RW	0x0	Shadow register for ROM-use word6 (Copy of OTP Bank 3, word 6 (ADDR = 0x1E)). These bits become read-only after the HW_OCOTP_LOCK[ROM_SHADOW] bit is set.

DESCRIPTION:

Shadowed memory mapped access to OTP Bank 3, word 6 (ADDR = 0x1E).

EXAMPLE:

```
Empty Example.
```

7.4.34 Shadow Register for OTP Bank3 Word7 (ROM Use 7) Description

Copied from the OTP automatically after reset. Can be re-loaded by setting HW_OCOTP_CTRL[RELOAD_SHADOWS].

HW_OCOTP_ROM7 0x210

Table 7-67. HW_OCOTP_ROM7

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2																							ENABLE_SSP_12MA_DRIVE	RSRVD1				I2C_USE_400KHZ	ENABLE_ARM_ICACHE	RSRVD0	ENABLE_PIN_BOOT_CHECK

Table 7-68. HW_OCOTP_ROM7 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:9	**RSRVD2**	RW	0x0	Reserved - do not blow these bits.
8	**ENABLE_SSP_12MA_DRIVE**	RW	0x0	Blow to force SSP pins to drive 12mA (default is 4mA)
7:4	**RSRVD1**	RW	0x0	Reserved - do not blow these bits.
3	**I2C_USE_400KHZ**	RW	0x0	Blow to force the I2C to be programmed by the boot loader to run at 400KHz (100KHz is the default)
2	**ENABLE_ARM_ICACHE**	RW	0x0	Blow to enable the ARM 926 ICache during boot
1	**RSRVD0**	RW	0x0	Reserved - do not blow this bit.
0	**ENABLE_PIN_BOOT_CHECK**	RW	0x1	Blow to enable boot loader to first test the LCD_RS pin to determine if pin boot mode is enabled. If this bit is blown, and LCD_RS is pulled high, then boot mode is determined by the state of LCD_D[6:0] pins. If this bit is not blown, skip testing the LCD_RS pin and go directly to determining boot mode by reading the state of LCD_D[6:0].

DESCRIPTION:

Shadowed memory mapped access to OTP Bank 3, word 7 (ADDR = 0x1F).

EXAMPLE:

```
Empty Example.
```

7.4.35 OTP Controller Version Register Description

This register always returns a known read value for debug purposes it indicates the version of the block.

HW_OCOTP_VERSION 0x220

Table 7-69. HW_OCOTP_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 7-70. HW_OCOTP_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x01	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x04	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
Empty Example.
```

OCOTP Block v1.4, Revision 1.21

Chapter 8
USB High-Speed Host/Device Controller

This chapter describes the USB high-speed controller included on the i.MX23. It includes sections on the PIO, DMA, and UTMI interfaces, along with USB controller flowcharts. Programmable USB controller registers are described in Section 8.6, "Programmable Registers." Descriptions for additional programmable registers mentioned in this chapter can be found in Section 4.8, "Programmable Registers," Section 6.4, "Programmable Registers," Section 9.4, "Programmable Registers, and Section 32.11, "Programmable Registers."

8.1 Overview

The i.MX23 includes a Universal Serial Bus (USB) version 2.0 controller capable of operating as either a USB device or a USB host, as shown in Figure 8-1. The USB controller is used to download digital music data or program code into external memory and to upload voice recordings from memory to the PC. Program updates can also be loaded into the flash memory area using the USB interface.

The USB device controller included on the i.MX23 supports five bi-directional endpoints: one control (for the default pipe) and four general purpose endpoints, each capable of operating in either IN, OUT, or both directions simultaneously. A typical portable device application defines 1 bulk-in, 1 bulk-out, and 1 interrupt in pipe.

As a USB host controller, it can enumerate and control USB devices attached to it. Using the USB Host Capability features, the USB controller can negotiate with another USB Host Capable system to be either the host or the device in a peer connection.

The USB controller operates either in full-speed mode or high-speed mode.

Refer to the USB Implementer's Forum website www.usb.org for detailed specifications and information on the USB protocol, timing and electrical characteristics.

The USB 2.0 controller comprises both a programmed I/O (PIO) interface and a DMA interface. Both of these interfaces are designed to meet an ARM Ltd. AMBA Hardware Bus (AHB). The AHB is used by the USB controller as a slave (PIO register accesses) and as a master (DMA memory accesses).

The USB 2.0 PHY is fully integrated on-chip and is described in Chapter 9, "Integrated USB 2.0 PHY," The PHY is controlled over the APBX peripheral bus.

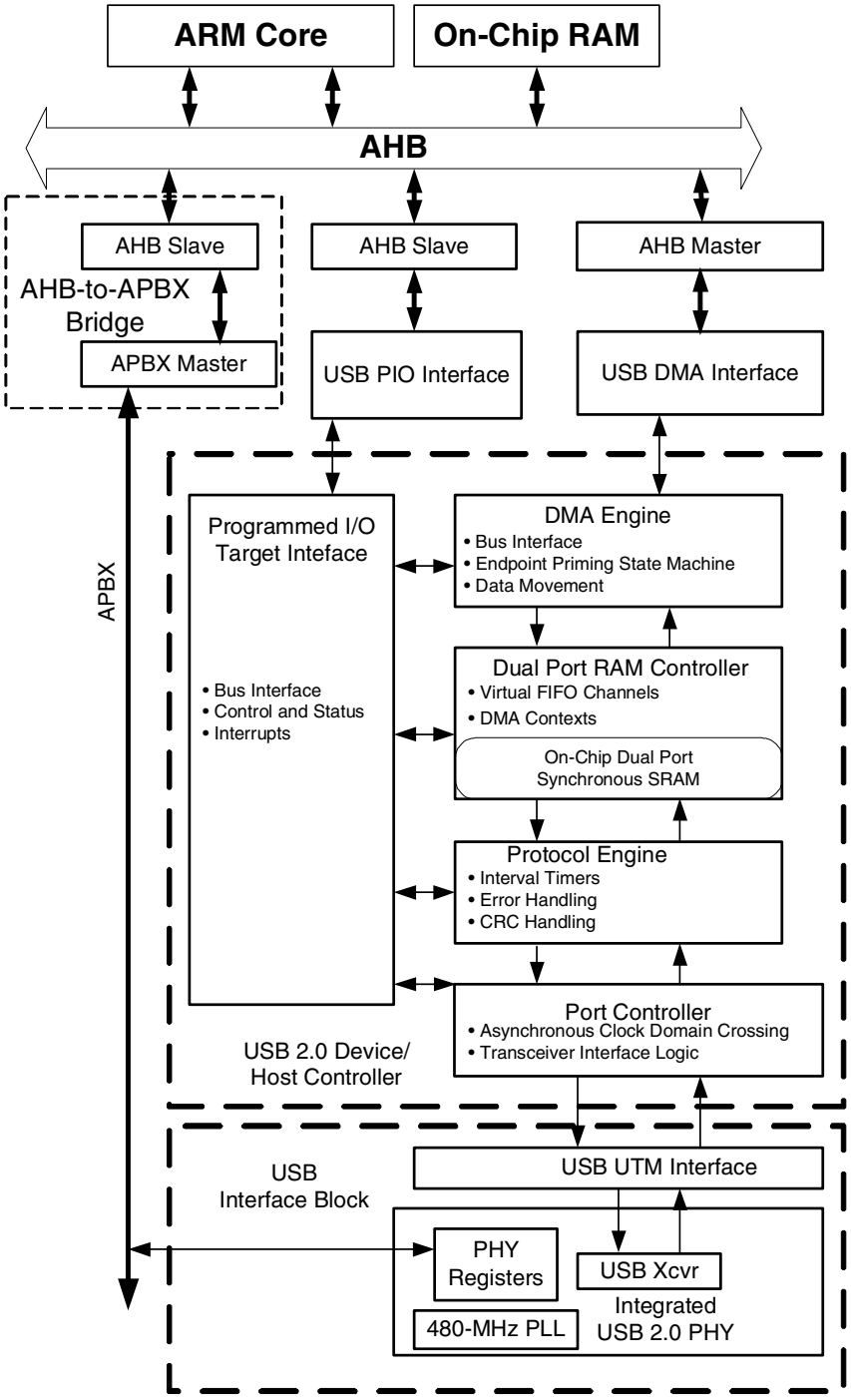


Figure 8-1. USB 2.0 Device Controller Block Diagram

8.2 USB Programmed I/O (PIO) Target Interface

The PIO interface is on an AHB slave of the USB controller. It allows the ARM processor to access the configuration, control, and status registers. There are identification registers for hardware configuration parameters and operational registers for control and status.

8.3 USB DMA Interface

The DMA is a master AHB interface that allows USB data to be transferred to/from the system memory. The data in memory is structured to implement a software framework supported by the controller. For a device controller, this structure is a linked-list interface that consists of queue heads and pointers that are transfer descriptors. The queue head is where transfers are managed. It has status information and location of the data buffers. The hardware controller's PIO registers enable the entire data structure, and once USB data is transferred between the host, the status of the transfer is updated in the queue head, with minimal latency to the system.

For a host controller, there is also a linked-list interface. It consists of a periodic frame list and pointers to transfer descriptors. The period frame list is a schedule of transfers. The frame list points to the data buffers through the transfer descriptors. The hardware controller's PIO registers enable the data structure and manage the transfers within a USB frame. The period frame list works as a sliding window of host transfers over time. As each transfer is completed, the status information is updated in the frame list.

The i.MX23 has the bandwidth to handle the data buffers in DRAM for both high-speed and full-speed USB transmissions. However, the queue heads (dQH) must be placed in on-chip RAM. A design limitation on burst size does not allow the queue heads to be placed in DRAM.

8.4 USB UTM Interface

The USB UTM interface on the i.MX23 implements the specification that allows USB controllers to interface with the USB PHY. Please refer to the *USB 2.0 Transceiver Macrocell Interface (UTMI) Specification, Version 1.05* and *UTMI+ Specification, Version 1.0* for additional details:
http://www.intel.com/technology/usb/spec.htm

8.4.1 Digital/Analog Loopback Test Mode

Since the UTM has to operate at high frequencies (480 MHz), it has a capacity to self-test. A pseudo-random number generator transmits data to the receive path, and data is compared for validity. In the digital loopback, the data transfer only resides in the UTM. It checks for sync, EOP, and bit-stuffing generation and data integrity. The analog loopback is the same as the digital loopback, but involves the analog PHY. This allows for checking of the and full-speed (FS) comparators and transmitters.

8.5 USB Controller Flowcharts

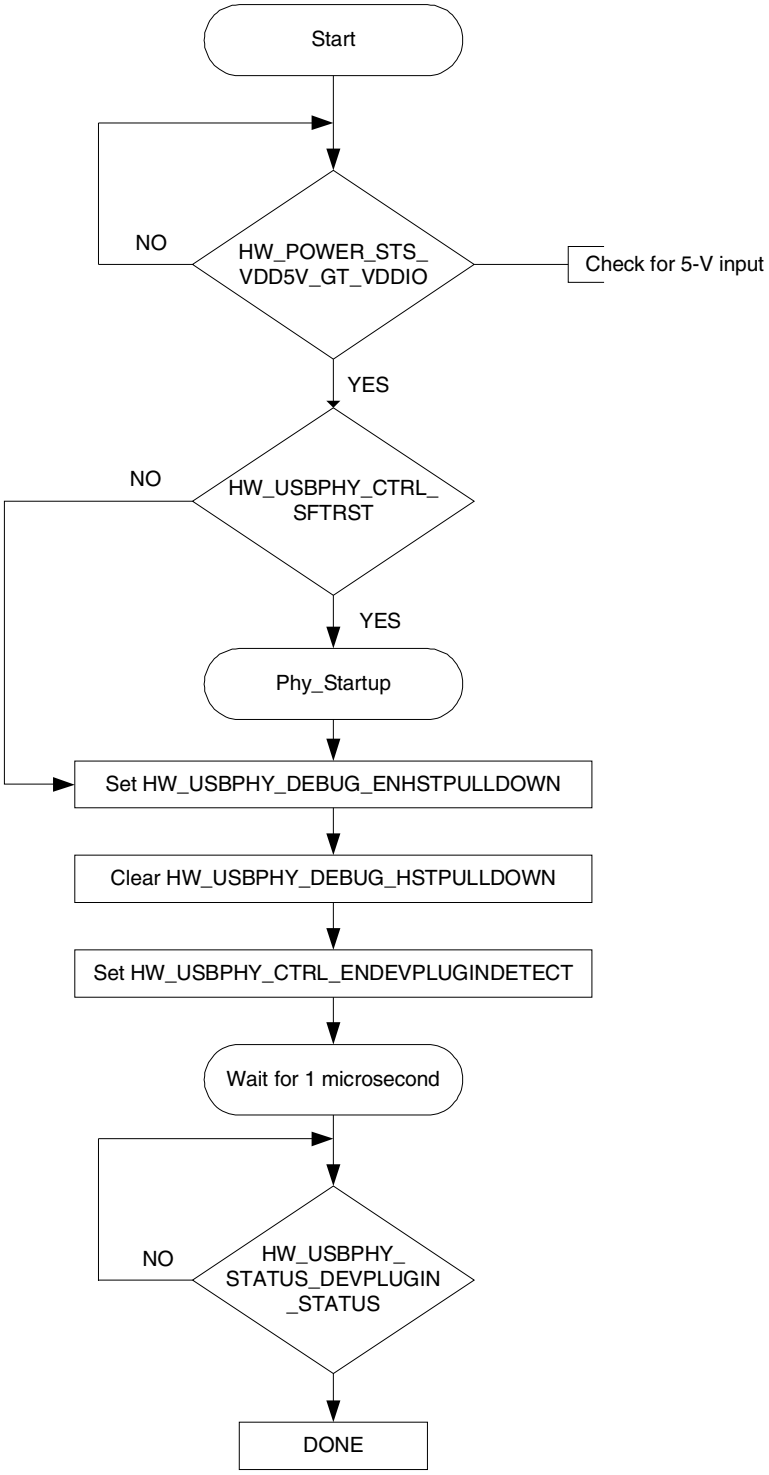


Figure 8-2. USB 2.0 Check_USB_Plugged_In Flowchart

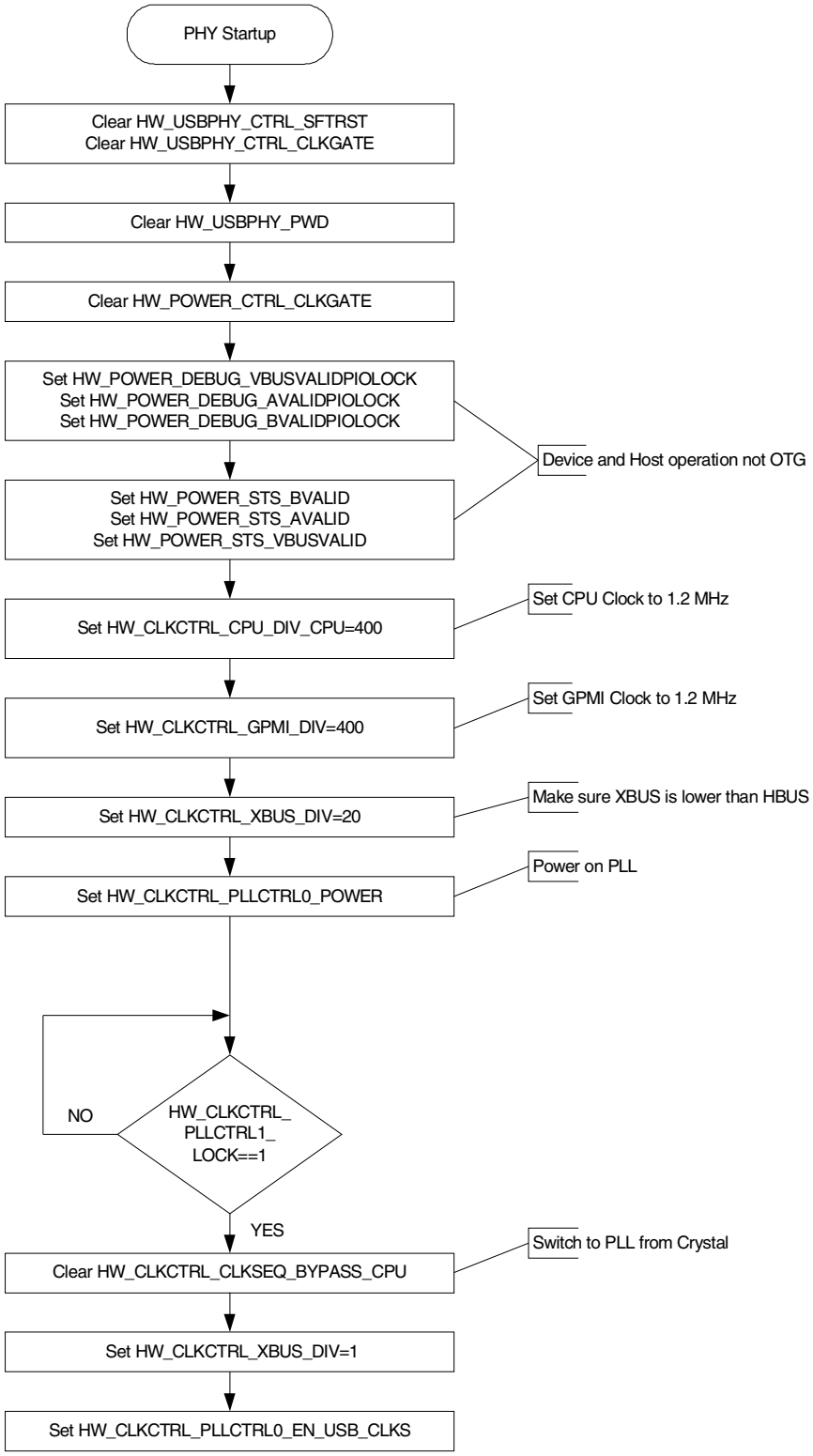


Figure 8-3. USB 2.0 USB PHY Startup Flowchart

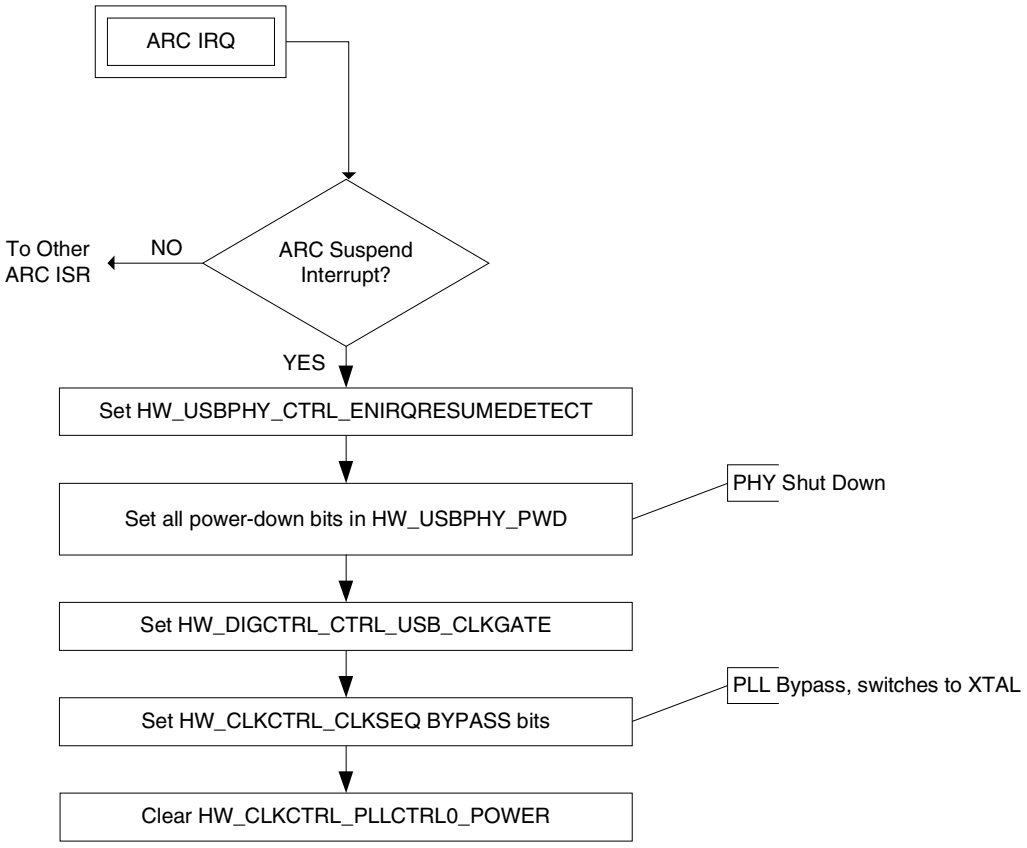


Figure 8-4. USB 2.0 PHY PLL Suspend Flowchart

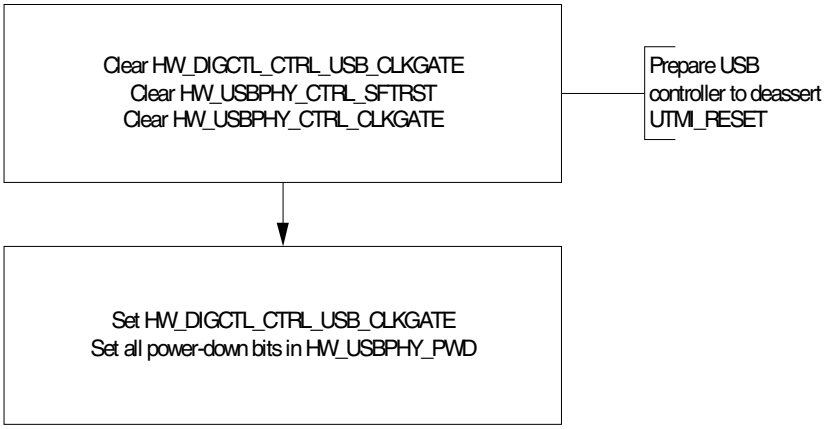


Figure 8-5. UTMI Powerdown

8.5.1 References

- *USB 2.0 Transceiver Macrocell Interface (UTMI) Specification*, Version 1.05, March 2001, Jon Lueker, Steve McGowan (Editor) Ken Oliver, Dean Warren. http://www.intel.com
- *VSI Alliance Virtual Component Interface Standard*, Version 2 (OCB 2 2.0), April 2001, On-Chip Bus Development Working Group. http://www.vsi.org
- *Universal Serial Bus Specification*, Revision 2.0, April 2000, Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips. http://www.usb.org
- *On-The-Go Supplement to the USB 2.0 Specification*, Revision 1.0, Dec 2001, On-The-Go Working Group of the USB-IF. http://www.usb.org
- *Enhanced Host Controller Interface Specification for Universal Serial Bus*, Revision 0.95, November 2000, Intel Corporation. http://www.intel.com
- *Universal Serial Bus Specification*, Revision 1.1, September 1998, Compaq, Intel, Microsoft, NEC. http://www.usb.org
- *AMBA Specification*, Revision 2.0, May 1999, ARM Limited. http://www.arm.com
- *UTMI+ Low Pin Interface (ULPI) Specification*, Revision 1.0 February 2004, ULPI Specification Organization. http://www.ulpi.org

8.6 Programmable Registers

This section includes the programmable registers supported in the USB high-speed Host controller core.

8.6.1 Identification Register Description

The Identification Register provides a simple way to determine if the USB-HS USB 2.0 core is provided in the system. The HW_USBCTRL_ID register identifies the USB-HS USB 2.0 core and its revision. The default value of this register is 0xE241FA05.

HW_USBCTRL_ID 0x000

Table 8-1. HW_USBCTRL_ID

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CIVERSION			VERSION				REVISION				TAG					RSVD1		NID						RSVD0		ID						

Table 8-2. HW_USBCTRL_ID Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**CIVERSION**	RO	0x7	Identifies the Chip Idea product version of the USB-HS USB 2.0 core.
28:25	**VERSION**	RO	0x1	Identifies the version of the USB-HS USB 2.0 core release. (<version>.<revision><tag>)
24:21	**REVISION**	RO	0x2	Identifies the revision of the USB-HS USB 2.0 core release. (<version>.<revision><tag>)
20:16	**TAG**	RO	0x1	Identifies the tag of the USB-HS USB 2.0 core release. (<version>.<revision><tag>)
15:14	**RSVD1**	RO	0x3	Reserved.
13:8	**NID**	RO	0x3a	One's complement version of ID[5:0].
7:6	**RSVD0**	RO	0x0	Reserved.
5:0	**ID**	RO	0x05	Configuration number. This number is set to 0x05 and indicates that the peripheral is the USB-HS USB 2.0 core.

DESCRIPTION:

REV/ID

EXAMPLE:

```
Empty Example.
```

8.6.2 General Hardware Parameters Register Description

The default value of this register is 0x00000015.

HW_USBCTRL_HWGENERAL 0x004

Table 8-3. HW_USBCTRL_HWGENERAL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																					SM		PHYM			PHYW		BWT	CLKC		RT

Table 8-4. HW_USBCTRL_HWGENERAL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:11	**RSVD**	RO	0x0	Reserved.
10:9	**SM**	RO	0x0	PHY Serial Engine Type. Always 0 = Serial engine not present.
8:6	**PHYM**	RO	0x0	PHY Type. Always 1 = UTMI.
5:4	**PHYW**	RO	0x1	Data Interface Width to PHY. Always 1 = 16 bits.
3	**BWT**	RO	0x0	Reserved for internal testing. Always 0.

Table 8-4. HW_USBCTRL_HWGENERAL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2:1	**CLKC**	RO	0x2	USB Controller Clocking Method. Always 2 = Mixed clocked.
0	**RT**	RO	0x1	Reset Type. Always 1 = Synchronous

DESCRIPTION:

General Hardware Parameters

EXAMPLE:

```
Empty Example.
```

8.6.3 Host Hardware Parameters Register Description

The default value of this register is 0x10020001.

HW_USBCTRL_HWHOST 0x008

Table 8-5. HW_USBCTRL_HWHOST

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TTPER								TTASY								RSVD												NPORT			HC

Table 8-6. HW_USBCTRL_HWHOST Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**TTPER**	RO	0x10	Periodic contexts for hub TT.
23:16	**TTASY**	RO	0x02	Asynch contexts for hub TT.
15:4	**RSVD**	RO	0x0	Reserved.
3:1	**NPORT**	RO	0x0	Maximum downstream ports minus 1.
0	**HC**	RO	0x1	Host Capable. Always 0x1.

DESCRIPTION:

Host hardware params as defined in sys-level/core-config

EXAMPLE:

```
Empty Example.
```

8.6.4 Device Hardware Parameters Register Description

The default value of this register is 0x0000000B.

HW_USBCTRL_HWDEVICE 0x00c

Table 8-7. HW_USBCTRL_HWDEVICE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																										DEVEP					DC

Table 8-8. HW_USBCTRL_HWDEVICE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:6	**RSVD**	RO	0x0	Reserved.
5:1	**DEVEP**	RO	0x5	Maximum number of endpoints, which is 5.
0	**DC**	RO	0x1	Device Capable. Always 0x1.

DESCRIPTION:

device hardware params as defined in sys-level/core-config

EXAMPLE:

```
Empty Example.
```

8.6.5 TX Buffer Hardware Parameters Register Description

The default value of this register is 0x40060910.

HW_USBCTRL_HWTXBUF 0x010

Table 8-9. HW_USBCTRL_HWTXBUF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TXLCR	RSVD							TXCHANADD								TXADD								TXBURST							

Table 8-10. HW_USBCTRL_HWTXBUF Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**TXLCR**	RO	0x1	Always 0x1.
30:24	**RSVD**	RO	0x0	Reserved.
23:16	**TXCHANADD**	RO	0x06	Number of address bits for the TX buffer.
15:8	**TXADD**	RO	0x09	Always 0x9.
7:0	**TXBURST**	RO	0x10	Burst size for memory-to-TX-buffer transfers.

DESCRIPTION:

tx hardware buf params

EXAMPLE:

```
Empty Example.
```

8.6.6 RX Buffer Hardware Parameters Register Description

The default value of this register is 0x00000710.

HW_USBCTRL_HWRXBUF 0x014

Table 8-11. HW_USBCTRL_HWRXBUF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																RXADD								RXBURST							

Table 8-12. HW_USBCTRL_HWRXBUF Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSVD**	RO	0x0	Reserved.
15:8	**RXADD**	RO	0x07	Always 0x07.
7:0	**RXBURST**	RO	0x10	Burst size for RX buffer-to-memory transfers.

DESCRIPTION:

rx hardware buf params

EXAMPLE:

```
Empty Example.
```

8.6.7 General-Purpose Timer 0 Load (Non-EHCI-Compliant) Register Description

The host/device controller driver can measure time-related activities using this timer register. This register is not part of the standard EHCI controller. This register contains the timer duration or load value. See the GPTIMER0CTRL (Non-EHCI) for a description of the timer functions.

HW_USBCTRL_GPTIMER0LD 0x080

Table 8-13. HW_USBCTRL_GPTIMER0LD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD0								GPTLD																							

Table 8-14. HW_USBCTRL_GPTIMER0LD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD0**	RO	0x0	Reserved.
23:0	**GPTLD**	RW	0x0	General-Purpose Timer Load Value. This field is the value to be loaded into the GPTCNT countdown timer on a reset action. This value in this register represents the time in microseconds minus 1 for the timer duration. Example: for a one-millisecond timer, load 1000 Note: Max value is 0xFFFFFF or 16.777215 seconds.

DESCRIPTION:

General Purpose Timer #0 Load Register

EXAMPLE:

```
Empty Example.
```

8.6.8 General-Purpose Timer 0 Control (Non-EHCI-Compliant) Register Description

The host/device controller driver can measure time-related activities using this timer register. This register is not part of the standard EHCI controller. This register contains the control for the timer and a data field can be queried to determine the running count value. This timer has granularity on 1 us and can be programmed to a little over 16 seconds. There are two modes supported by this timer, the first is a one-shot and the second is a looped count that is described in the register table below. When the timer counter value transitions to 0, an interrupt can be generated through the use of the timer interrupts in the USBTS and USBINTR registers.

HW_USBCTRL_GPTIMER0CTRL 0x084

Table 8-15. HW_USBCTRL_GPTIMER0CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
GPTRUN	GPTRST	RSVD0					GPTMODE	GPTCNT																							

Table 8-16. HW_USBCTRL_GPTIMER0CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**GPTRUN**	RW	0x0	General-Purpose Timer Run. This bit enables the general-purpose timer to run. Setting or clearing this bit will not have an effect on the GPTCNT except if it stops or starts counting. STOP = 0 Timer stop. RUN = 1 Timer run.
30	**GPTRST**	WO	0x0	General-Purpose Timer Reset. Writing a 1 to this bit will reload the GPTCNT with the value in GPTLD. NOACTION = 0 No action. LOADCOUNTER = 1 Load counter value.
29:25	**RSVD0**	RO	0x0	Reserved.
24	**GPTMODE**	RW	0x0	General-Purpose Timer Mode. This bit selects between a single timer countdown and a looped count down. In one-shot mode, the timer will count down to 0, generate an interrupt, and stop until the counter is reset by software. In repeat mode, the timer will count down to 0, generate an interrupt, and automatically reload the counter to begin again. ONESHOT = 0 One shot. REPEAT = 1 Repeat.
23:0	**GPTCNT**	RO	0x0	General-Purpose Timer Counter. This field is the value of the running timer.

DESCRIPTION:

General Purpose Timer #0 Control Register

EXAMPLE:

```
Empty Example.
```

8.6.9 General-Purpose Timer 1 Load (Non-EHCI-Compliant) Register Description

Same as GPTIMER0LD description.

HW_USBCTRL_GPTIMER1LD 0x088

Table 8-17. HW_USBCTRL_GPTIMER1LD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD0								GPTLD																							

Table 8-18. HW_USBCTRL_GPTIMER1LD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD0**	RO	0x0	Reserved.
23:0	**GPTLD**	RW	0x0	General-Purpose Timer Load Value. This field is the value to be loaded into the GPTCNT countdown timer on a reset action. This value in this register represents the time in microseconds minus 1 for the timer duration. Example: for a one-millisecond timer, load 1000 Note: Max value is 0xFFFFFF or 16.777215 seconds.

DESCRIPTION:

General Purpose Timer #1 Load Register

EXAMPLE:

```
Empty Example.
```

8.6.10 General-Purpose Timer 1 Control (Non-EHCI-Compliant) Register Description

Same as GPTIMER0CTRL description.

HW_USBCTRL_GPTIMER1CTRL 0x08c

Table 8-19. HW_USBCTRL_GPTIMER1CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
GPTRUN	GPTRST	RSVD0					GPTMODE	GPTCNT																							

Table 8-20. HW_USBCTRL_GPTIMER1CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**GPTRUN**	RW	0x0	General-Purpose Timer Run. This bit enables the general-purpose timer to run. Setting or clearing this bit will not have an effect on the GPTCNT except if it stops or starts counting. STOP = 0 Timer stop. RUN = 1 Timer run.
30	**GPTRST**	WO	0x0	General-Purpose Timer Reset. Writing a 1 to this bit will reload the GPTCNT with the value in GPTLD. NOACTION = 0 No action. LOADCOUNTER = 1 Load counter value.
29:25	**RSVD0**	RO	0x0	Reserved.

Table 8-20. HW_USBCTRL_GPTIMER1CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
24	**GPTMODE**	RW	0x0	General-Purpose Timer Mode. This bit selects between a single timer countdown and a looped count down. In one-shot mode, the timer will count down to 0, generate an interrupt, and stop until the counter is reset by software. In repeat mode, the timer will count down to 0, generate an interrupt, and automatically reload the counter to begin again. ONESHOT = 0 One shot. REPEAT = 1 Repeat.
23:0	**GPTCNT**	RO	0x0	General-Purpose Timer Counter. This field is the value of the running timer.

DESCRIPTION:

General Purpose Timer #1 Control Register

EXAMPLE:

```
Empty Example.
```

8.6.11 System Bus Configuration (Non-EHCI-Compliant) Register Description

This register controls the AMBA system bus Master/Slave interfaces.

HW_USBCTRL_SBUSCFG 0x090

Table 8-21. HW_USBCTRL_SBUSCFG

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																													AHBBRST		

Table 8-22. HW_USBCTRL_SBUSCFG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:3	**RSVD**	RO	0x00	Reserved.
2:0	**AHBBRST**	RW	0x0	AMBA AHB BURST. This field selects the following options for the m_hburst signal of the AMBA master interface: U_INCR = 0x0 INCR burst of unspecified length. S_INCR4 = 0x1 INCR4, non-multiple transfers of INCR4 will be decomposed into singles. S_INCR8 = 0x2 INCR8, non-multiple transfers of INCR8 will be decomposed into INCR4 or singles. S_INCR16 = 0x3 INCR16, non-multiple transfers of INCR16 will be decomposed into INCR8, INCR4 or singles. RESERVED = 0x4 This value is reserved and should not be used. U_INCR4 = 0x5 INCR4, non-multiple transfers of INCR4 will be decomposed into smaller unspecified length bursts. U_INCR8 = 0x6 INCR8, non-multiple transfers of INCR8 will be decomposed into smaller unspecified length bursts. U_INCR16 = 0x7 INCR16, non-multiple transfers of INCR16 will be decomposed into smaller unspecified length bursts.

DESCRIPTION:

AHB System Bus Configuration Register

EXAMPLE:

```
Empty Example.
```

8.6.12 Capability Length and HCI Version (EHCI-Compliant) Register Description

This register contains the Capability Length and HCI Version Register.

HW_USBCTRL_CAPLENGTH 0x100

Table 8-23. HW_USBCTRL_CAPLENGTH

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
HCIVERSION																RSVD								CAPLENGTH							

Table 8-24. HW_USBCTRL_CAPLENGTH Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**HCIVERSION**	RO	0x0100	Contains a BCD encoding of the EHCI revision number supported by the host controller. The most significant byte of this register represents a major revision and the least significant byte is the minor revision.
15:8	**RSVD**	RO	0x00	Reserved.
7:0	**CAPLENGTH**	RO	0x40	Offset to add to register base address at beginning of the Operational Register.

DESCRIPTION:

Capability Length and HCI Version register

EXAMPLE:

```
Empty Example.
```

8.6.13 Host Control Structural Parameters (EHCI-Compliant with Extensions) Register Description

Port-steering logic capabilities are described in this register. The default value of this register is 0x00010011.

HW_USBCTRL_HCSPARAMS 0x104

Table 8-25. HW_USBCTRL_HCSPARAMS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2				N_TT				N_PTT				RSVD1			PI	N_CC				N_PCC				RSVD0			PPC	N_PORTS			

Table 8-26. HW_USBCTRL_HCSPARAMS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**RSVD2**	RO	0x0	Reserved.
27:24	**N_TT**	RO	0x0	Number of Transaction Translators (N_TT). Indicates the number of embedded transaction translators associated with the USB2.0 host controller. This in a non-EHCI field to support embedded TT.
23:20	**N_PTT**	RO	0x0	Number of Ports per Transaction Translator (N_PTT). Indicates the number of ports assigned to each transaction translator within the USB2.0 host controller. This in a non-EHCI field to support embedded TT.
19:17	**RSVD1**	RO	0x0	Reserved.
16	**PI**	RO	0x1	Port Indicators (P INDICATOR). Indicates whether the ports support port indicator control. When set to 1, the port status and control registers include a read/writable field for controlling the state of the port indicator.
15:12	**N_CC**	RO	0x0	Number of Companion Controller (N_CC). Indicates the number of companion controllers associated with this USB2.0 host controller. A 0 in this field indicates there are no internal Companion Controllers. Port-ownership hand-off is not supported. A value larger than 0 in this field indicates there are companion USB host controller(s). Port-ownership hand-offs are supported. High- and Full-speed devices are supported on the host controller root ports.
11:8	**N_PCC**	RO	0x0	Number of Ports per Companion Controller. Indicates the number of ports supported per internal Companion Controller. It is used to indicate the port routing configuration to the system software. For example, if N_PORTS has a value of 6 and N_CC has a value of 2 then N_PCC could have a value of 3. The convention is that the first N_PCC ports are assumed to be routed to companion controller 1, the next N_PCC ports to companion controller 2, etc. In the previous example, the N_PCC could have been 4, where the first 4 are routed to companion controller 1 and the last two are routed to companion controller 2. The number in this field must be consistent with N_PORTS and N_CC.
7:5	**RSVD0**	RO	0x0	Reserved.

Table 8-26. HW_USBCTRL_HCSPARAMS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
4	**PPC**	RO	0x1	Port Power Control. Indicates whether the host controller implementation includes port power control. A 1 indicates the ports have port power switches. A 0 indicates the ports do not have port power switches. The value of this field affects the functionality of the Port Power field in each port status and control register.
3:0	**N_PORTS**	RO	0x1	Number of downstream ports. Specifies the number of physical downstream ports implemented on this host controller. The value of this field determines how many port registers are addressable in the Operational Register. Valid values are in the range of 0x11-0xF. A 0 in this field is undefined.

DESCRIPTION:

host controller structural params

EXAMPLE:

```
Empty Example.
```

8.6.14 Host Control Capability Parameters (EHCI-Compliant) Register Description

This register identifies multiple mode control (time-base bit functionality) addressing capability. The default value of this register is 0x00000006.

HW_USBCTRL_HCCPARAMS 0x108

Table 8-27. HW_USBCTRL_HCCPARAMS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2																EECP								IST				RSVD0	ASP	PFL	ADC

Table 8-28. HW_USBCTRL_HCCPARAMS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSVD2**	RO	0x0	Reserved.
15:8	**EECP**	RO	0x0	EHCI Extended Capabilities Pointer. Default = 0. This optional field indicates the existence of a capabilities list. A value of 0x00 indicates no extended capabilities are implemented. A non-zero value in this register indicates the offset in PCI configuration space of the first EHCI extended capability. The pointer value must be 0x40 or greater if implemented to maintain the consistency of the PCI header defined for this class of device.
7:4	**IST**	RO	0x0	Isochronous Scheduling Threshold. Indicates, relative to the current position of the executing host controller, where software can reliably update the isochronous schedule. When bit 7 is 0, the value of the least significant 3 bits indicates the number of micro-frames a host controller can hold a set of isochronous data structures (one or more) before flushing the state. When bit 7 is a 1, then host software assumes the host controller may cache an isochronous data structure for an entire frame.
3	**RSVD0**	RO	0x0	Reserved.
2	**ASP**	RO	0x1	Asynchronous Schedule Park Capability. Default = 1. If this bit is set to a 1, then the host controller supports the park feature for high-speed queue heads in the Asynchronous Schedule. The feature can be disabled or enabled and set to a specific level by using the Asynchronous Schedule Park Mode Enable and Asynchronous Schedule Park Mode Count fields in the USBCMD register.
1	**PFL**	RO	0x1	Programmable Frame List Flag. If this bit is set to 0, then the system software must use a frame list length of 1024 elements with this host controller. The USBCMD register Frame List Size field is a read-only register and must be set to 0. If set to a 1, then the system software can specify and use a smaller frame list and configure the host controller via the USBCMD register Frame List Size field. The frame list must always be aligned on a 4K page boundary. This requirement ensures that the frame list is always physically contiguous.
0	**ADC**	RO	0x0	64-bit Addressing Capability. No 64-bit addressing capability is supported.

DESCRIPTION:

host controller capability params

EXAMPLE:

```
Empty Example.
```

8.6.15 Device Interface Version Number (Non-EHCI-Compliant) Register Description

The device controller interface conforms to the two-byte BCD encoding of the interface version number contained in this register.

HW_USBCTRL_DCIVERSION 0x120

Table 8-29. HW_USBCTRL_DCIVERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																DCIVERSION															

Table 8-30. HW_USBCTRL_DCIVERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSVD**	RO	0x0	Reserved.
15:0	**DCIVERSION**	RW	0x1	Two-byte BCD encoding of the interface version number.

DESCRIPTION:

device interface version

EXAMPLE:

```
Empty Example.
```

8.6.16 Device Control Capability Parameters (Non-EHCI-Compliant) Register Description

These fields describe the overall host/device capability of the controller. The default value of this register is 0x00000185.

HW_USBCTRL_DCCPARAMS 0x124

Table 8-31. HW_USBCTRL_DCCPARAMS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1																							HC	DC	RSVD2		DEN				

Table 8-32. HW_USBCTRL_DCCPARAMS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:9	**RSVD1**	RO	0x0	Reserved.
8	**HC**	RO	0x1	Host Capable. When this bit is 1, this controller is capable of operating as an EHCI-compatible USB 2.0 host controller.
7	**DC**	RO	0x1	Device Capable. When this bit is 1, this controller is capable of operating as a USB 2.0 device.
6:5	**RSVD2**	RO	0x0	Reserved.
4:0	**DEN**	RO	0x5	Device Endpoint Number. This field indicates the number of endpoints built into the device controller, which is 5.

DESCRIPTION:

device controller capability params

EXAMPLE:

```
Empty Example.
```

8.6.17 USB Command Register Description

The serial bus host/device controller executes the command indicated in this register. * Default Value:0x00080B00 (Host mode), 0x00080000 (Device mode)

HW_USBCTRL_USBCMD 0x140

Table 8-33. HW_USBCTRL_USBCMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD3								ITC								FS2	ATDTW	SUTW	RSVD2	ASPE	RSVD1	ASP		LR	IAA	ASE	PSE	FS1	FS0	RST	RS

Table 8-34. HW_USBCTRL_USBCMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD3**	RO	0x0	Reserved.
23:16	**ITC**	RW	0x8	Interrupt Threshold Control. Default 0x08. The system software uses this field to set the maximum rate at which the host/device controller will issue interrupts. ITC contains the maximum interrupt interval measured in micro-frames. Valid values are: IMM = 0x0 Immediate (no threshold). 1_MICROFRAME = 0x1 1_MICROFRAME. 2_MICROFRAME = 0x2 2_MICROFRAME. 4_MICROFRAME = 0x4 4_MICROFRAME. 8_MICROFRAME = 0x8 8_MICROFRAME. 16_MICROFRAME = 0x10 16_MICROFRAME. 32_MICROFRAME = 0x20 32_MICROFRAME. 64_MICROFRAME = 0x40 64_MICROFRAME.

Table 8-34. HW_USBCTRL_USBCMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15	**FS2**	RW	0x0	Bit 2 of Frame List Size field. See definition of bit FS0 for the complete definition.
14	**ATDTW**	RW	0x0	Add dTD TripWire (device mode only). This bit is used as a semaphore to ensure the proper addition of a new dTD to an active (primed) endpoint's linked list. This bit is set and cleared by software. This bit shall also be cleared by hardware when is state machine is hazard region for which adding a dTD to a primed endpoint may go unrecognized.
13	**SUTW**	RW	0x0	Setup TripWire (device mode only). This bit is used as a semaphore to ensure that the setup data payload of 8 bytes is extracted from a QH by the DCD without being corrupted. If the setup lockout mode is off (See USBMODE) then there exists a hazard when new setup data arrives while the DCD is copying the setup data payload from the QH for a previous setup packet. This bit is set and cleared by software and will be cleared by hardware when a hazard exists.
12	**RSVD2**	RO	0x0	Reserved.
11	**ASPE**	RW	0x0	Asynchronous Schedule Park Mode Enable (OPTIONAL). This bit defaults to 0x1. Software uses this bit to enable or disable Park mode. When this bit is 1, Park mode is enabled. When this bit is a 0, Park mode is disabled. This field is set to 1 in host mode; 0 in device mode.
10	**RSVD1**	RO	0x0	Reserved.
9:8	**ASP**	RW	0x0	Asynchronous Schedule Park Mode Count (OPTIONAL). This field defaults to 0x3 and is R/W. It contains a count of the number of successive transactions the host controller is allowed to execute from a high-speed queue head on the Asynchronous schedule before continuing traversal of the Asynchronous schedule. See Section 4.10.3.2 of the EHCI specification for full operational details. Valid values are 0x1-0x3. Software must not write a 0 to this bit as this will result in undefined behavior. This field is set to 0x3 in host mode; 0x0 in device mode.
7	**LR**	RW	0x0	Light Host/Device Controller Reset (OPTIONAL). Not Implemented. This field will always be 0.

Table 8-34. HW_USBCTRL_USBCMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
6	**IAA**	RW	0x0	Interrupt on Async Advance Doorbell. This bit is used as a doorbell by software to tell the host controller to issue an interrupt the next time it advances asynchronous schedule. Software must write a 1 to this bit to ring the doorbell. When the host controller has evicted all appropriate cached schedule states, it sets the Interrupt on Async Advance status bit in the USBSTS register. If the Interrupt on Sync Advance Enable bit in the USBINTR register is 1, then the host controller will assert an interrupt at the next interrupt threshold. The host controller sets this bit to 0 after it has set the Interrupt on Sync Advance status bit in the USBSTS register to 1. Software should not write a 1 to this bit when the asynchronous schedule is inactive. Doing so will yield undefined results. This bit is only used in host mode. Writing a 1 to this bit when device mode is selected will have undefined results.
5	**ASE**	RW	0x0	Asynchronous Schedule Enable. Default 0. This bit controls whether the host controller skips processing the Asynchronous Schedule. 0 = Do not process the Asynchronous Schedule. 1 = Use the ASYNCLISTADDR register to access the Asynchronous Schedule. Only the host controller uses this bit.
4	**PSE**	RW	0x0	Periodic Schedule Enable. Default Ob. This bit controls whether the host controller skips processing the Periodic Schedule. 0 = Do not process the Periodic Schedule 1 = Use the PERIODICLISTBASE register to access the Periodic Schedule. Only the host controller uses this bit.
3	**FS1**	RW	0x0	Bit 1 of Frame List Size field. See definition of bit FS0 for the complete definition.
2	**FS0**	RW	0x0	Bit 0 of Frame List Size field. The Frame List Size field (FS2, FS1, FS0) specifies the size of the frame list that controls which bits in the Frame Index Register should be used for the Frame List Current index. Note that this field is made up from USBCMD bits 15, 3 and 2. Default is 000b. 000b = 1024 ELEMENTS (4096 bytes) Default value. 001b = 512_ELEMENTS (2048 bytes). 010b = 256_ELEMENTS (1024 bytes). 011b = 128_ELEMENTS (512 bytes). 100b = 64_ELEMENTS (256 bytes). 101b = 32_ELEMENTS (128 bytes). 110b = 16_ELEMENTS (64 bytes). 111b = 8_ELEMENTS (32 bytes). Only the host controller uses this field.

Table 8-34. HW_USBCTRL_USBCMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
1	**RST**	RW	0x0	Controller Reset (RESET). Software uses this bit to reset the controller. This bit is set to 0 by the Host/Device Controller when the reset process is complete. Software cannot terminate the reset process early by writing a 0 to this register. Host Controller: When software writes a 1 to this bit, the Host Controller resets its internal pipelines, timers, counters, state machines etc. to their initial value. Any transaction currently in progress on USB is immediately terminated. A USB reset is not driven on downstream ports. Software should not set this bit to a 1 when the HCHalted bit in the USBSTS register is a 0. Attempting to reset an actively running host controller will result in undefined behavior. Device Controller: When software writes a 1 to this bit, the Device Controller resets its internal pipelines, timers, counters, state machines etc. to their initial value. Writing a 1 to this bit when the device is in the attached state is not recommended, since the effect on an attached host is undefined. In order to ensure that the device is not in an attached state before initiating a device controller reset, all primed endpoints should be flushed and the USBCMD Run/Stop bit should be set to 0.
0	**RS**	RW	0x0	Run/Stop (RS). Default 0. 1 = Run. 0 = Stop. Host Controller: When set to a 1, the Host Controller proceeds with the execution of the schedule. The Host Controller continues execution as long as this bit is set to a 1. When this bit is set to 0, the Host Controller completes the current transaction on the USB and then halts. The HC Halted bit in the status register indicates when the Host Controller has finished the transaction and has entered the stopped state. Software should not write a 1 to this field unless the host controller is in the Halted state (i.e., HCHalted in the USBSTS register is a 1). Device Controller: Writing a 1 to this bit will cause the device controller to enable a pullup on D+ and initiate an attach event. This control bit is not directly connected to the pullup enable, as the pullup will become disabled upon transitioning into high-speed mode. Software should use this bit to prevent an attach event before the device controller has been properly initialized. Writing a 0 to this will cause a detach event.

DESCRIPTION:

command

EXAMPLE:

```
Empty Example.
```

8.6.18 USB Status Register Description

This register indicates various states of the Host/Device Controller and any pending interrupts. This register does not indicate status resulting from a transaction on the serial bus. Software clears certain bits in this register by writing a 1 to them. * Default Value:0x00001000 (Host mode), 0x00000000 (Device mode)

HW_USBCTRL_USBSTS 0x144

Table 8-35. HW_USBCTRL_USBSTS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD5						TI1	TI0	RSVD4				UPI	UAI	RSVD3	NAKI	AS	PS	RCL	HCH	RSVD2	ULPII	RSVD1	SLI	SRI	URI	AAI	SEI	FRI	PCI	UEI	UI

Table 8-36. HW_USBCTRL_USBSTS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSVD5**	RO	0x0	Reserved.
25	**TI1**	RW	0x0	General-Purpose Timer Interrupt 1 (GPTINT1). This bit is set when the counter in the GPTIMER1CTRL (Non-EHCI) register transitions to 0. Writing a 1 to this bit will clear it.
24	**TI0**	RW	0x0	General-Purpose Timer Interrupt 0 (GPTINT0). This bit is set when the counter in the GPTIMER0CTRL (Non-EHCI) register transitions to 0. Writing a 1 to this bit will clear it.
23:20	**RSVD4**	RO	0x0	Reserved.
19	**UPI**	RW	0x0	USB Host Periodic Interrupt (USBHSTPERINT). This bit is set by the Host Controller when the cause of an interrupt is a completion of a USB transaction where the Transfer Descriptor (TD) has an interrupt on complete (IOC) bit set and the TD was from the periodic schedule. This bit is also set by the Host Controller when a short packet is detected AND the packet is on the periodic schedule. A short packet is when the actual number of bytes received was less than the expected number of bytes. This bit is not used by the device controller and will always be 0.

Table 8-36. HW_USBCTRL_USBSTS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
18	**UAI**	RW	0x0	USB Host Asynchronous Interrupt (USBHSTASYNCINT). This bit is set by the Host Controller when the cause of an interrupt is a completion of a USB transaction where the Transfer Descriptor (TD) has an interrupt on complete (IOC) bit set AND the TD was from the asynchronous schedule. This bit is also set by the Host when a short packet is detected AND the packet is on the asynchronous schedule. A short packet is when the actual number of bytes received was less than the expected number of bytes. This bit is not used by the device controller and will always be 0.
17	**RSVD3**	RO	0x0	Reserved.
16	**NAKI**	RO	0x0	NAK Interrupt Bit. It is set by hardware when for a particular endpoint both the TX/RX Endpoint NAK bit and the corresponding TX/RX Endpoint NAK Enable bit are set. This bit is automatically cleared by hardware when the all the enabled TX/RX Endpoint NAK bits are cleared.
15	**AS**	RO	0x0	Asynchronous Schedule Status. This bit reports the current real status of the Asynchronous Schedule. When set to 0 the asynchronous schedule status is disabled and if set to 1 the status is enabled. The Host Controller is not required to immediately disable or enable the Asynchronous Schedule when software transitions the Asynchronous Schedule Enable bit in the USBCMD register. When this bit and the Asynchronous Schedule Enable bit are the same value, the Asynchronous Schedule is either enabled (1) or disabled (0). Only used by the host controller.
14	**PS**	RO	0x0	Periodic Schedule Status. 0 = Default. This bit reports the current real status of the Periodic Schedule. When set to 0 the periodic schedule is disabled, and if set to 1 the status is enabled. The Host Controller is not required to immediately disable or enable the Periodic Schedule when software transitions the Periodic Schedule Enable bit in the USBCMD register. When this bit and the Periodic Schedule Enable bit are the same value, the Periodic Schedule is either enabled (1) or disabled (0). Only used by the host controller.
13	**RCL**	RO	0x0	Reclamation. 0 = Default. This is a read-only status bit used to detect an empty asynchronous schedule. Only used by the host controller; 0 in device mode.

Table 8-36. HW_USBCTRL_USBSTS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
12	**HCH**	RW	0x0	HC Halted. 1 = Default. This bit is a 0 whenever the Run/Stop bit is a 1. The Host Controller sets this bit to 1 after it has stopped executing because of the Run/Stop bit being set to 0, either by software or by the Host Controller hardware (e.g. internal error). Only used by the host controller; 0 in device mode.
11	**RSVD2**	RO	0x0	Reserved.
10	**ULPII**	RW	0x0	Not present in this implementation.
9	**RSVD1**	RO	0x0	Reserved.
8	**SLI**	RW	0x0	DC Suspend. 0 = Default. When a device controller enters a suspend state from an active state, this bit will be set to a 1. The device controller clears the bit upon exiting from a suspend state. Only used by the device controller.
7	**SRI**	RW	0x0	SOF Received. 0 = Default. When the device controller detects a Start Of (micro) Frame, this bit will be set to a 1. When a SOF is extremely late, the device controller will automatically set this bit to indicate that an SOF was expected. Therefore, this bit will be set roughly every 1ms in device FS mode and every 125ms in HS mode and will be synchronized to the actual SOF that is received. Since the device controller is initialized to FS before connect, this bit will be set at an interval of 1ms during the prelude to connect and chirp. In host mode, this bit will be set every 125us and can be used by host controller driver as a time base. Software writes a 1 to this bit to clear it. This is a non-EHCI status bit.
6	**URI**	RW	0x0	USB Reset Received. 0 = Default. When the device controller detects a USB Reset and enters the default state, this bit will be set to a 1. Software can write a 1 to this bit to clear the USB Reset Received status bit. Only used by the device controller. NOTE: This bit should not normally be used to detect reset during suspend, as this block will normally be clock-gated during that time. Use HW_USBPHY_CTRL_RESUME_IRQ, instead.
5	**AAI**	RW	0x0	Interrupt on Async Advance. 0 = Default. System software can force the host controller to issue an interrupt the next time the host controller advances the asynchronous schedule by writing a 1 to the Interrupt on Async Advance Doorbell bit in the USBCMD register. This status bit indicates the assertion of that interrupt source. Only used by the host controller.

Table 8-36. HW_USBCTRL_USBSTS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
4	**SEI**	RW	0x0	System Error. This bit is not used in this implementation and will always be set to 0.
3	**FRI**	RW	0x0	Frame List Rollover. The Host Controller sets this bit to a 1 when the Frame List Index rolls over from its maximum value to 0. The exact value at which the rollover occurs depends on the frame list size. For example. If the frame list size (as programmed in the Frame List Size field of the USBCMD register) is 1024, the Frame Index Register rolls over every time FRINDEX [13] toggles. Similarly, if the size is 512, the Host Controller sets this bit to a 1 every time FHINDEX [12] toggles. Only used by the host controller.
2	**PCI**	RW	0x0	Port Change Detect. The Host Controller sets this bit to a 1 when on any port a Connect Status occurs, a Port Enable/Disable Change occurs, or the Force Port Resume bit is set as the result of a J-K transition on the suspended port. The Device Controller sets this bit to a 1 when the port controller enters the full or high-speed operational state. When the port controller exits the full or highspeed operation states due to Reset or Suspend events, the notification mechanisms are the USB Reset Received bit and the DCSuspend bits respectively. This bit is not EHCI compatible.
1	**UEI**	RW	0x0	USB Error Interrupt (USBERRINT). When completion of a USB transaction results in an error condition, this bit is set by the Host/Device Controller. This bit is set along with the USBINT bit, if the TD on which the error interrupt occurred also had its interrupt on complete (IOC) bit set. See Section 4.15.1 in the EHCI specification for a complete list of host error interrupt conditions. The device controller detects resume signaling only.
0	**UI**	RW	0x0	USB Interrupt (USBINT). This bit is set by the Host/Device Controller when the cause of an interrupt is a completion of a USB transaction where the Transfer Descriptor (TD) has an interrupt on complete (IOC) bit set. This bit is also set by the Host/Device Controller when a short packet is detected. A short packet is when the actual number of bytes received was less than the expected number of bytes.

DESCRIPTION:

status

EXAMPLE:

```
Empty Example.
```

8.6.19 USB Interrupt Enable Register Description

The interrupts to software are enabled with this register. An interrupt is generated when a bit is set and the corresponding interrupt is active. The USB Status register (USBSTS) still shows interrupt sources even if they are disabled by the USBINTR register, allowing polling of interrupt events by the software.

HW_USBCTRL_USBINTR 0x148

Table 8-37. HW_USBCTRL_USBINTR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD5						TIE1	TIE0	RSVD4				UPIE	UAIE	RSVD3	NAKE	RSVD2					ULPIE	RSVD1	SLE	SRE	URE	AAE	SEE	FRE	PCE	UEE	UE

Table 8-38. HW_USBCTRL_USBINTR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSVD5**	RO	0x0	Reserved.
25	**TIE1**	RW	0x0	General-Purpose Timer Interrupt Enable 1. When this bit is a 1, and the GPTINT1 bit in the USBSTS register is a 1, the controller will issue an interrupt. The interrupt is acknowledged by software clearing the GPTINT1 bit.
24	**TIE0**	RW	0x0	General-Purpose Timer Interrupt Enable 0. When this bit is a 1, and the GPTINT0 bit in the USBSTS register is a 1, the controller will issue an interrupt. The interrupt is acknowledged by software clearing the GPTINT0 bit.
23:20	**RSVD4**	RO	0x0	Reserved.
19	**UPIE**	RW	0x0	USB Host Periodic Interrupt Enable. When this bit is a 1, and the USBHSTPERINT bit in the USBSTS register is a 1, the host controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the USBHSTPERINT bit.
18	**UAIE**	RW	0x0	USB Host Asynchronous Interrupt Enable. RW 0x0 When this bit is a 1, and the USBHSTASYNCINT bit in the USBSTS register is a 1, the host controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the USBHSTASYNCINT bit.
17	**RSVD3**	RO	0x0	Reserved.
16	**NAKE**	RW	0x0	NAK Interrupt Enable. This bit is set by software if it wants to enable the hardware interrupt for the NAK Interrupt bit. If both this bit and the corresponding NAK Interrupt bit are set, a hardware interrupt is generated.
15:11	**RSVD2**	RO	0x0	Reserved.
10	**ULPIE**	RW	0x0	ULPI Enable. Not used in this implementation.
9	**RSVD1**	RO	0x0	Reserved.

Table 8-38. HW_USBCTRL_USBINTR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
8	**SLE**	RW	0x0	Sleep Enable. When this bit is a 1, and the DCSuspend bit in the USBSTS register transitions, the device controller will issue an interrupt. The interrupt is acknowledged by software writing a 1 to the DCSuspend bit. Only used by the device controller.
7	**SRE**	RW	0x0	SOF Received Enable. When this bit is a 1, and the SOF Received bit in the USBSTS register is a 1, the device controller will issue an interrupt. The interrupt is acknowledged by software clearing the SOF Received bit.
6	**URE**	RW	0x0	USB Reset Enable. When this bit is a 1, and the USB Reset Received bit in the USBSTS register is a 1, the device controller will issue an interrupt. The interrupt is acknowledged by software clearing the USB Reset Received bit. Only used by the device controller.
5	**AAE**	RW	0x0	Interrupt on Async Advance Enable. When this bit is a 1, and the Interrupt on Async Advance bit in the USBSTS register is a 1, the host controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the Interrupt on Async Advance bit. Only used by the host controller.
4	**SEE**	RW	0x0	System Error Enable. When this bit is a 1, and the System Error bit in the USBSTS register is a 1, the host/device controller will issue an interrupt. The interrupt is acknowledged by software clearing the System Error bit.
3	**FRE**	RW	0x0	Frame List Rollover Enable. When this bit is a 1, and the Frame List Rollover bit in the USBSTS register is a 1, the host controller will issue an interrupt. The interrupt is acknowledged by software clearing the Frame List Rollover bit. Only used by the host controller.
2	**PCE**	RW	0x0	Port Change Detect Enable. When this bit is a 1, and the Port Change Detect bit in the USBSTS register is a 1, the host/device controller will issue an interrupt. The interrupt is acknowledged by software clearing the Port Change Detect bit.
1	**UEE**	RW	0x0	USB Error Interrupt Enable. When this bit is a 1, and the USBERRINT bit in the USBSTS register is a 1, the host controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the USBERRINT bit in the USBSTS register.
0	**UE**	RW	0x0	USB Interrupt Enable. When this bit is a 1, and the USBINT bit in the USBSTS register is a 1, the host/device controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the USBINT bit.

DESCRIPTION:

interrupt enables

EXAMPLE:

```
Empty Example.
```

8.6.20 USB Frame Index Register Description

This register is used by the host controller to index the periodic frame list. The register updates every 125 microseconds (once each micro-frame). Bits [N: 3] are used to select a particular entry in the Periodic Frame List during periodic schedule execution. The number of bits used for the index depends on the size of the frame list as set by system software in the Frame List Size field in the USBCMD register. This register must be written as a DWord. Byte writes produce-undefined results. This register cannot be written unless the Host Controller is in the 'Halted' state as indicated by the HCHalted bit. A write to this register while the Run/Stop bit is set to a 1 produces undefined results. Writes to this register also affect the SOF value. In device mode this register is Read-Only and, the device controller updates the FRINDEX [13:3] register from the frame number indicated by the SOF marker. Whenever a SOF is received by the USB bus, FRINDEX [13:3] will be checked against the SOF marker. If FRINDEX [13:3] is different from the SOF marker, FRINDEX [13:3] will be set to the SOF value and FRINDEX [2:0] will be set to 0 (i.e., SOF for 1 ms frame). If FRINDEX [13:3] is equal to the SOF value, FRINDEX [2:0] will be incremented (i.e., SOF for 125 us micro-frame.) * The default value of this register is undefined (free-running counter).

HW_USBCTRL_FRINDEX 0x14c

Table 8-39. HW_USBCTRL_FRINDEX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																		FRINDEX											UINDEX		

Table 8-40. HW_USBCTRL_FRINDEX Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:14	**RSVD**	RO	0x0	Reserved.
13:3	**FRINDEX**	RO	0x0	Frame List Current Index. Read/write in host mode. Read in device mode. The value in this register increments at the end of each time frame (e.g., micro-frame). Bits [N: 3] are used for the Frame List current index. This means that each location of the frame list is accessed 8 times (frames or micro-frames) before moving to the next index. The following illustrates values of N based on the value of the Frame List Size field in the USBCMD register, when used in host mode. In device mode, the value is the current frame number of the last frame transmitted. It is not used as an index. N_12 = 12 FRAME LIST SIZE = 1024, USBCMD = 3'b000. N_11 = 11 FRAME LIST SIZE = 512, USBCMD = 3'b001. N_10 = 10 FRAME LIST SIZE = 256, USBCMD = 3'b010. N_9 = 9 FRAME LIST SIZE = 128, USBCMD = 3'b011. N_8 = 8 FRAME LIST SIZE = 64, USBCMD = 3'b100. N_7 = 7 FRAME LIST SIZE = 32, USBCMD = 3'b101. N_6 = 6 FRAME LIST SIZE = 16, USBCMD = 3'b110. N_5 = 5 FRAME LIST SIZE = 8, USBCMD = 3'b111.
2:0	**UINDEX**	RW	0x0	Current Microframe.

DESCRIPTION:

frame index

EXAMPLE:

```
Empty Example.
```

8.6.21 Frame List Base Address Register (Host Controller mode) Description

In Host Controller mode, this 32-bit register contains the beginning address of the Periodic Frame List in the system memory. HCD loads this register prior to starting the schedule execution by the Host Controller. The memory structure referenced by this physical memory pointer is assumed to be 4-Kbyte aligned. The contents of this register are combined with the Frame Index Register (FRINDEX) to enable the Host Controller to step through the Periodic Frame List in sequence. This is a read/write register. Writes must be DWORD writes.

HW_USBCTRL_PERIODICLISTBASE 0x154

Table 8-41. HW_USBCTRL_PERIODICLISTBASE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PERBASE																				RSVD											

Table 8-42. HW_USBCTRL_PERIODICLISTBASE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:12	**PERBASE**	RW	0x0	Base Address (Low). These bits correspond to memory address signals [31:12], respectively. Only used by the host controller.
11:0	**RSVD**	RO	0x0	Reserved. Must be written as zeros. During runtime, the values of these bits are undefined.

DESCRIPTION:

PERIODIC LIST BASE (host-controller)

EXAMPLE:

```
Empty Example.
```

8.6.22 USB Device Address Register (Device Controller mode) Description

In Device Controller mode, the upper seven bits of this register represent the device address. After any controller reset or a USB reset, the device address is set to the default address (0). The default address will match all incoming addresses. Software shall reprogram the address after receiving a SET_ADDRESS descriptor. The USBADRA is used to accelerate the SET_ADDRESS sequence by allowing the DCD to preset the USBADR register before the status phase of the SET_ADDRESS descriptor. This is a read/write register. Writes must be DWORD writes.

HW_USBCTRL_DEVICEADDR 0x154

Table 8-43. HW_USBCTRL_DEVICEADDR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
USBADR							USBADRA	RSVD																							

Table 8-44. HW_USBCTRL_DEVICEADDR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**USBADR**	RW	0x0	Device Address. These bits correspond to the USB device address.
24	**USBADRA**	RW	0x0	Device Address Advance. Default=0. When this bit is `0', any writes to USBADR are instantaneous. When this bit is written to a 1 at the same time or before USBADR is written, the write to the USBADR field is staged and held in a hidden register. After an IN occurs on endpoint 0 and is ACKed, USBADR will be loaded from the holding register. Hardware will automatically clear this bit on the following conditions: 1. IN is ACKed to endpoint 0. (USBADR is updated from staging register). 2. OUT/SETUP occur to endpoint 0. (USBADR is not updated). 3. Device Reset occurs (USBADR is reset to 0). Note: After the status phase of the SET_ADDRESS descriptor, the DCD has 2 ms to program the USBADR field. This mechanism will ensure this specification is met when the DCD cannot write of the device address within 2ms from the SET_ADDRESS status phase. If the DCD writes the USBADR with USBADRA=1 after the SET_ADDRESS data phase (before the prime of the status phase), the USBADR will be programmed instantly at the correct time and meet the 2ms USB requirement.
23:0	**RSVD**	RO	0x0	Reserved. Must be written as zeros. During runtime, the values of these bits are undefined.

DESCRIPTION:

DEVICE-ADDR

EXAMPLE:

```
Empty Example.
```

8.6.23 Next Asynchronous Address Register (Host Controller mode) Description

In Host Controller mode, this 32-bit register contains the address of the next asynchronous queue head to be executed by the host. Bits [4:0] of this register cannot be modified by the system software and will always return a 0 when read.

HW_USBCTRL_ASYNCLISTADDR 0x158

Table 8-45. HW_USBCTRL_ASYNCLISTADDR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ASYBASE																												RSVD			

Table 8-46. HW_USBCTRL_ASYNCLISTADDR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**ASYBASE**	RW	0x0	Link Pointer Low (LPL). These bits correspond to memory address signals [31:5], respectively. This field may only reference a Queue Head (OH). Only used by the host controller.
4:0	**RSVD**	RO	0x0	Reserved. These bits are reserved and their value has no effect on operation.

DESCRIPTION:

ASYNC-LIST-ADDR (host-controller)

EXAMPLE:

```
Empty Example.
```

8.6.24 Endpoint List Address Register (Device Controller mode) Description

In Device Controller mode, this register contains the address of the top of the endpoint list in system memory. Bits [10:0] of this register cannot be modified by the system software and will always return a 0 when read. The memory structure referenced by this physical memory pointer is assumed 64-byte. This is a read/write register. Writes must be DWORD writes.

HW_USBCTRL_ENDPOINTLISTADDR 0x158

Table 8-47. HW_USBCTRL_ENDPOINTLISTADDR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
EPBASE																					RSVD										

Table 8-48. HW_USBCTRL_ENDPOINTLISTADDR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:11	**EPBASE**	RW	0x0	Endpoint List Pointer (Low). These bits correspond to memory address signals [31:11], respectively. This field will reference a list of up to 32 Queue Heads (QH). (i.e., one queue head per endpoint and direction.)
10:0	**RSVD**	RO	0x0	Reserved. These bits are reserved and their value has no effect on operation.

DESCRIPTION:

EndPoint List Address

EXAMPLE:

```
Empty Example.
```

8.6.25 Embedded TT Asynchronous Buffer Status and Control Register (Host Controller mode) Description

This register contains parameters needed for internal TT operations. This register is not used in the device controller operation. This is a read/write register. Writes must be DWORD writes.

HW_USBCTRL_TTCTRL 0x15c

Table 8-49. HW_USBCTRL_TTCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1	TTHA							RSVD2																							

Table 8-50. HW_USBCTRL_TTCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSVD1**	RO	0x0	Reserved. These bits are reserved and their value has no effect on operation.
30:24	**TTHA**	RW	0x0	Internal TT Hub Address Representation. Default is 0 (Read/Write). This field is used to match against the Hub Address field in QH and siTD to determine if the packet is routed to the internal TT for directly attached FS/LS devices. If the Hub Address in the QH or siTD does not match this address then the packet will be broadcast on the High Speed ports destined for a downstream High Speed hub with the address in the QH/siTD.
23:0	**RSVD2**	RO	0x0	Reserved. These bits are reserved and their value has no effect on operation.

DESCRIPTION:

TT (internal operation) Control

EXAMPLE:

```
Empty Example.
```

8.6.26 Programmable Burst Size Register Description

This register is used to control dynamically change the burst size used during data movement on the initiator (master) interface. This is a read/write register. Writes must be DWORD writes. The default value is 0x00001010.

HW_USBCTRL_BURSTSIZE 0x160

Table 8-51. HW_USBCTRL_BURSTSIZE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																TXPBURST								RXPBURST							

Table 8-52. HW_USBCTRL_BURSTSIZE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSVD**	RO	0x0	Reserved. These bits are reserved and their value has no effect on operation.
15:8	**TXPBURST**	RW	0x10	Programmable TX Burst Length. This register represents the maximum length of a the burst in 32-bit words while moving data from system memory to the USB bus.
7:0	**RXPBURST**	RW	0x10	Programmable RX Burst Length. This register represents the maximum length of a the burst in 32-bit words while moving data from the USB bus to system memory.

DESCRIPTION:

controls (dynamically) burst size for usb->ahb

EXAMPLE:

```
Empty Example.
```

8.6.27 Host Transmit Pre-Buffer Packet Timing Register Description

The fields in this register control performance tuning associated with how the host controller posts data to the TX latency FIFO before moving the data onto the USB bus. The specific areas of performance include the how much data to post into the FIFO and an estimate for how long that operation should take in the target system. Definitions: T0 = Standard packet overhead T1 = Time to send data payload Tff = Time to

fetch packet into TX FIFO up to specified level. Ts = Total Packet Flight Time (send-only) packet Ts = T0 + T1 Tp = Total Packet Time (fetch and send) packet Tp = Tff + T0 + T1 Upon discovery of a transmit (OUT/SETUP) packet in the data structures, host controller checks to ensure Tp remains before the end of the (micro)frame. If so it proceeds to pre-fill the TX FIFO. If at anytime during the pre-fill operation the time remaining the (micro)frame is < Ts then the packet attempt ceases and the packet is tried at a later time. Although this is not an error condition and the host controller will eventually recover, a mark will be made the scheduler health counter to note the occurrence of a "back-off" event. When a back-off event is detected, the partial packet fetched may need to be discarded from the latency buffer to make room for periodic traffic that will begin after the next SOF. Too many back-off events can waste bandwidth and power on the system bus and thus should be minimized (not necessarily eliminated). Back-offs can be minimized with use of the TSCHHEALTH (Tff) described below. This is a read/write register. Writes must be DWORD writes. The default value of this register is 0x00000000.

HW_USBCTRL_TXFILLTUNING 0x164

Table 8-53. HW_USBCTRL_TXFILLTUNING

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2										TXFIFOTHRES						RSVD1			TXSCHEALTH					RSVD0	TXSCHOH						

Table 8-54. HW_USBCTRL_TXFILLTUNING Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:22	**RSVD2**	RO	0x0	Reserved. These bits are reserved and their value has no effect on operation.
21:16	**TXFIFOTHRES**	RW	0x0	FIFO Burst Threshold. This register controls the number of data bursts that are posted to the TX latency FIFO in host mode before the packet begins on to the bus. The minimum value is 2 and this value should be a low as possible to maximize USB performance. A higher value can be used in systems with unpredictable latency and/or insufficient bandwidth where the FIFO may underrun because the data transferred from the latency FIFO to USB occurs before it can be replenished from system memory. This value is ignored if the Stream Disable bit in USBMODE register is set.
15:13	**RSVD1**	RO	0x0	Reserved. These bits are reserved and their value has no effect on operation.

Table 8-54. HW_USBCTRL_TXFILLTUNING Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
12:8	**TXSCHEALTH**	RW	0x0	Scheduler Health Counter. This register increments when the host controller fails to fill the TX latency FIFO to the level programmed by TXFIFOTHRES before running out of time to send the packet before the next Start-Of-Frame. This health counter measures the number of times this occurs to provide feedback to selecting a proper TXSCHOH. Writing to this register will clear the counter and this counter will max. at 31.
7	**RSVD0**	RO	0x0	Reserved. This bit is reserved and its value has no effect on operation.
6:0	**TXSCHOH**	RW	0x0	Scheduler Overhead. This register adds an additional fixed offset to the schedule time estimator described above as Tff. As an approximation, the value chosen for this register should limit the number of back-off events captured in the TXSCHHEALTH to less than 10 per second in a highly utilized bus. Choosing a value that is too high for this register is not desired as it can needlessly reduce USB utilization. The time unit represented in this register is 1.267us when a device is connected in High-Speed Mode for OTG & SPH. The time unit represented in this register is 6.333us when a device is connected in Full Speed Mode for OTG & SPH. The time unit represented in this register is always 1.267 in the MPH product.

DESCRIPTION:

TX Fill Tuning

EXAMPLE:

```
Empty Example.
```

8.6.28 Inter-Chip Control Register Description

This register is present but not used in this implementation.

HW_USBCTRL_IC_USB 0x16c

Table 8-55. HW_USBCTRL_IC_USB

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																												IC_ENABLE	IC_VDD		

Table 8-56. HW_USBCTRL_IC_USB Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:4	**RSVD**	RO	0x0	Reserved.
3	**IC_ENABLE**	RW	0x0	Inter-Chip Transceiver Enable. These bits enables the InterChip transceiver for each port (for the MPH case). To enable the interface, the bits PTS must be set to 0b11 in the PORTSCx. Writing a '1' to each bit selects the IC_USB interface for that port. If the Controller is not MultiPort, IC8 to IC2 will be '0' and Read-Only.
2:0	**IC_VDD**	RW	0x0	Inter-Chip Transceiver Voltage. Selects the voltage being supplied to the peripheral through each port (MPH case). VOLTAGE_NONE = 0x0 . VOLTAGE_1_0 = 0x1 . VOLTAGE_1_2 = 0x2 . VOLTAGE_1_5 = 0x3 . VOLTAGE_1_8 = 0x4 . VOLTAGE_3_0 = 0x5 . RESERVED0 = 0x6 . RESERVED1 = 0x7 .

DESCRIPTION:

This register enables and controls the IC_USB FS/LS transceiver.

EXAMPLE:

```
Empty Example.
```

8.6.29 ULPI Viewport Register Description

This register is present but not used in this implementation.

HW_USBCTRL_ULPI 0x170

Table 8-57. HW_USBCTRL_ULPI

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ULPIWU	ULPIRUN	ULPIRW	RSVD0	ULPISS	ULPIPORT			ULPIADDR								ULPIDATRD								ULPIDATWR							

Table 8-58. HW_USBCTRL_ULPI Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**ULPIWU**	RW	0x0	Not used. Read as 0.
30	**ULPIRUN**	RW	0x0	Not used. Read as 0.
29	**ULPIRW**	RW	0x0	Not used. Read as 0.
28	**RSVD0**	RO	0x0	Not used. Read as 0.
27	**ULPISS**	RO	0x0	Not used. Read as 0.
26:24	**ULPIPORT**	RW	0x0	Not used. Read as 0.
23:16	**ULPIADDR**	RW	0x0	Not used. Read as 0.

Table 8-58. HW_USBCTRL_ULPI Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:8	**ULPIDATRD**	RO	0x0	Not used. Read as 0.
7:0	**ULPIDATWR**	RW	0x0	Not used. Read as 0.

DESCRIPTION:

ULPI control

EXAMPLE:

```
Empty Example.
```

8.6.30 Endpoint NAK Register Description

HW_USBCTRL_ENDPTNAK 0x178

Table 8-59. HW_USBCTRL_ENDPTNAK

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1											EPTN					RSVD0											EPRN				

Table 8-60. HW_USBCTRL_ENDPTNAK Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:21	**RSVD1**	RO	0x0	Reserved.
20:16	**EPTN**	RW	0x0	TX Endpoint NAK. Each TX endpoint has one bit in this field. The bit is set when the device sends a NAK handshake on a received IN token for the corresponding endpoint. EPTN[4] = Endpoint 4 EPTN[3] = Endpoint 3 EPTN[2] = Endpoint 2 EPTN[1] = Endpoint 1 EPTN[0] = Endpoint 0
15:5	**RSVD0**	RO	0x0	Reserved.
4:0	**EPRN**	RW	0x0	RX Endpoint NAK. Each RX endpoint has 1 bit in this field. The bit is set when the device sends a NAK handshake on a received OUT or PING token for the corresponding endpoint. EPRN[4] = Endpoint 4 EPRN[3] = Endpoint 3 EPRN[2] = Endpoint 2 EPRN[1] = Endpoint 1 EPRN[0] = Endpoint 0

DESCRIPTION:

NAK-sent indicator

EXAMPLE:

```
Empty Example.
```

8.6.31 Endpoint NAK Enable Register Description

HW_USBCTRL_ENDPTNAKEN 0x17c

Table 8-61. HW_USBCTRL_ENDPTNAKEN

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1											EPTNE					RSVD0											EPRNE				

Table 8-62. HW_USBCTRL_ENDPTNAKEN Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:21	**RSVD1**	RO	0x0	Reserved.
20:16	**EPTNE**	RW	0x0	TX Endpoint NAK Enable. Each bit is an enable bit for the corresponding TX Endpoint NAK bit. If this bit is set and the corresponding TX Endpoint NAK bit is set, the NAK Interrupt bit is set. EPTNE[4] = Endpoint 4 EPTNE[3] = Endpoint 3 EPTNE[2] = Endpoint 2 EPTNE[1] = Endpoint 1 EPTNE[0] = Endpoint 0
15:5	**RSVD0**	RO	0x0	Reserved.
4:0	**EPRNE**	RW	0x0	RX Endpoint NAK Enable. Each bit is an enable bit for the corresponding RX Endpoint NAK bit. If this bit is set and the corresponding RX Endpoint NAK bit is set, the NAK Interrupt bit is set. EPRNE[4] = Endpoint 4 EPRNE[3] = Endpoint 3 EPRNE[2] = Endpoint 2 EPRNE[1] = Endpoint 1 EPRNE[0] = Endpoint 0

DESCRIPTION:

NAK-sent indicator enable

EXAMPLE:

```
Empty Example.
```

8.6.32 Port Status and Control 1 Register Description

Host Controller: A host controller must implement one to eight port registers. The number of port registers implemented by a particular instantiation of a host controller is documented in the HCSPARAMs register and is fixed at 1 in this implementation. Software uses this information as an input parameter to determine

how many ports need service. This register is only reset when power is initially applied or in response to a controller reset. The initial conditions of a port are: - No device connected - Port disabled If the port has port power control, this state remains until software applies power to the port by setting port power to 1. Device Controller: A device controller must implement only port register 1 and it does not support power control. Port control in device mode is only used for status port reset, suspend, and current connect status. It is also used to initiate test mode or force signaling and allows software to put the PHY into low power suspend mode and disable the PHY clock. * Default Value: 00010000000000000000XX0000000000b (Host mode) 00010000000000000001XX0000000100b (Device mode) X = Unknown

HW_USBCTRL_PORTSC1 0x184

Table 8-63. HW_USBCTRL_PORTSC1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PTS		STS	PTW	PSPD		SRT	PFSC	PHCD	WKOC	WKDS	WKCN	PTC				PIC		PO	PP	LS		HSP	PR	SUSP	FPR	OCC	OCA	PEC	PE	CSC	CCS

Table 8-64. HW_USBCTRL_PORTSC1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**PTS**	RW	0x0	Parallel Transceiver Select. For this implementation, always set to 00b for UTMI. UTMI = 0 UTMI/UTMI+. PHIL = 1 Phillips-Classic. ULPI = 2 ULPI. SERIAL = 3 Serial/1.1FS.
29	**STS**	RW	0x0	Serial Transceiver Select. Always 0.
28	**PTW**	RW	0x1	Parallel Transceiver Width. This bit is always 0, indicating an 8-bit (60-MHz) UTMI interface.
27:26	**PSPD**	RW	0x0	Port Speed. This register field indicates the speed at which the port is operating. For high-speed mode operation in the host controller and high-speed/fullspeed operation in the device controller, the port routing steers data to the protocol engine. This bit is not defined in the EHCI specification. FULL = 0 Full Speed. HIGH = 2 High Speed.
25	**SRT**	RW	0x0	Reserved.
24	**PFSC**	RW	0x0	Port Force Full Speed Connect. Default = 0. Writing this bit to a 1 will force the port to only connect at Full Speed. It disables the chirp sequence that allows the port to identify itself as high-speed. This is useful for testing full-speed configurations with a high-speed host, hub or device. This bit is not defined in the EHCI specification. This bit is for debugging purposes.

Table 8-64. HW_USBCTRL_PORTSC1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
23	**PHCD**	RW	0x0	PHY Low Power Suspend - Clock Disable (PLPSCD). Default = 0. Writing this bit to a 1 will disable the PHY clock. Writing a 0 enables it. Reading this bit will indicate the status of the PHY clock. In Device Mode: The PHY can be put into Low Power Suspend running (USBCMD Run/Stop=0) or the host has signaled suspend (PORTSC SUSPEND=1). Lowpower suspend will be cleared automatically when the host has signaled resume. Before forcing a resume from the device, the device controller driver must clear this bit. In Host Mode: The PHY can be put into Low Power Suspend device has been put into suspend mode or when no downstream device is connected. Low power suspend is completely under the control of software. This bit is not defined in the EHCI specification.
22	**WKOC**	RW	0x0	Wake on Over-current Enable (WKOC_E). Default = 0. Writing this bit to a 1 enables the port to be sensitive to over-current conditions as wake-up events. This field is 0 if Port Power (PP) is 0.
21	**WKDS**	RW	0x0	Wake on Disconnect Enable (WKDSCNNT_E). Default=0. Writing this bit to a 1 enables the port to be sensitive to device disconnects as wake-up events. This field is 0 if Port Power (PP) is 0 or in device mode.
20	**WKCN**	RW	0x0	Wake on Connect Enable (WKCNNT_E). Default=0. Writing this bit to a 1 enables the port to be sensitive to device connects as wake-up events. This field is 0 if Port Power (PP) is 0 or in device mode.
19:16	**PTC**	RW	0x0	Port Test Control. Default = 0000b. Any other value than 0 indicates that the port is operating in test mode. Refer to Chapter 9 of the USB Specification Revision 2.0 for details on each test mode. The TEST_FORCE_ENABLE_FS and TEST_FORCE_ENABLE_LS are extensions to the test mode support specified in the EHCI specification. Writing the PTC field to any of the TEST_FORCE_ENABLE_{HS/FS/LS} values will force the port into the connected and enabled state at the selected speed. Writing the PTC field back to TEST_DISABLE will allow the port state machines to progress normally from that point. Note: Low speed operations are not supported. TEST_DISABLE = 0 Disable. TEST_J_STATE = 1 J-State. TEST_K_STATE = 2 K-State. TEST_J_SE0_NAK = 3 Host:SE0/Dev:NAK. TEST_PACKET = 4 Test-Packet. TEST_FORCE_ENABLE_HS = 5 Force-Enable-HS. TEST_FORCE_ENABLE_FS = 6 Force-Enable-FS. TEST_FORCE_ENABLE_LS = 7 Force-Enable-LS.

Table 8-64. HW_USBCTRL_PORTSC1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:14	**PIC**	RW	0x0	Port Indicator Control. Default = 0. Refer to the USB Specification Revision 2.0 for a description on how these bits are to be used. OFF = 0 OFF. AMBER = 1 Amber. GREEN = 2 Green. UNDEF = 3 undefined.
13	**PO**	RW	0x0	Port Owner. Port owner handoff is not implemented in this design, therefore this bit will always read back as 0. The EHCI definition is include here for reference: Default = 0. This bit unconditionally goes to a 0 when the configured bit in the CONFIGFLAG register makes a 0 to 1 transition. This bit unconditionally goes to 1 whenever the Configured bit is 0. System software uses this field to release ownership of the port to a selected host controller (in the event that the attached device is not a high-speed device). Software writes a 1 to this bit when the attached device is not a high-speed device. A 1 in this bit means that an internal companion controller owns and controls the port.
12	**PP**	RW	0x0	Port Power (PP). This bit represents the current setting of the switch (0=off, 1=on). When power is not available on a port (i.e., PP equals a 0), the port is nonfunctional and will not report attaches, detaches, etc. When an over-current condition is detected on a powered port, the PP bit in each affected port may be transitioned by the host controller driver from a 1 to a 0 (removing power from the port). This feature is implemented in the host/OTG controller (PPC = 1). In a device implementation, port power control is not necessary.
11:10	**LS**	RW	0x0	Line Status. These bits reflect the current logical levels of the D+ (bit 11) and D- (bit 10) signal lines. The bit encodings are listed below. In Host Mode: The use of linestate by the host controller driver is not necessary (unlike EHCI), because the port controller state machine and the port routing manage the connection of LS and FS. In Device Mode: The use of linestate by the device controller driver is not necessary. SE0 = 0 SE0. K_STATE = 1 K. J_STATE = 2 J. UNDEF = 3 Undefined.

Table 8-64. HW_USBCTRL_PORTSC1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
9	**HSP**	RW	0x0	High-Speed Port. Default = 0. When the bit is 1, the host/device connected to the port is in high-speed mode and if set to 0, the host/device connected to the port is not in a high-speed mode. Note: HSP is redundant with PSPD(27:26) but will remain in the design for compatibility. This bit is not defined in the EHCI specification.
8	**PR**	RW	0x0	Port Reset This field is 0 if Port Power (PP) is 0. In Host Mode: (Read/Write). 1 = Port is in Reset. 0 = Port is not in Reset. Default 0. When software writes a 1 to this bit, the bus-reset sequence as defined in the USB Specification Revision 2.0 is started. This bit will automatically change to 0 after the reset sequence is complete. This behavior is different from EHCI where the host controller driver is required to set this bit to a 0 after the reset duration is timed in the driver. In Device Mode: This bit is a Read-Only status bit. Device reset from the USB bus is also indicated in the USBSTS register.

Table 8-64. HW_USBCTRL_PORTSC1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
7	**SUSP**	RW	0x0	Suspend In Host Mode: (Read/Write) 0 = Port not in suspend state. 1 = Port in suspend state. Default = 0. Port Enabled Bit and Suspend bit of this register define the port states as follows: Bits Port State 0x Disable 10 Enable 11 Suspend When in suspend state, the downstream propagation of data is blocked on this port, except for port reset. The blocking occurs at the end of the current transaction if a transaction was in progress when this bit was written to 1. In the suspend state, the port is sensitive to resume detection. Note that the bit status does not change until the port is suspended and that there may be a delay in suspending a port if there is a transaction currently in progress on the USB. The host controller will unconditionally set this bit to 0 when software sets the Force Port Resume bit to 0. The host controller ignores a write of 0 to this bit. If host software sets this bit to a 1 when the port is not enabled (i.e., Port enabled bit is a 0) the results are undefined. This field is 0 if Port Power (PP) is 0 in host mode. In Device Mode: (Read-Only) 1 = Port in suspend state. 0 = Port not in suspend state. Default=0. In device mode, this bit is a Read-Only status bit.

Table 8-64. HW_USBCTRL_PORTSC1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
6	**FPR**	RW	0x0	Force Port Resume. 0 = No resume (K-state) detected/driven on port. 1 = Resume detected/driven on port. Default = 0. In Host Mode: Software sets this bit to 1 to drive resume signaling. The Host Controller sets this bit to 1 if a J-to-K transition is detected while the port is in the Suspend state. When this bit transitions to a 1 because a J-to-K transition is detected, the Port Change Detect bit in the USBSTS register is also set to 1. This bit will automatically change to 0 after the resume sequence is complete. This behavior is different from EHCI where the host controller driver is required to set this bit to a 0 after the resume duration is timed in the driver. Note that when the Host controller owns the port, the resume sequence follows the defined sequence documented in the USB Specification Revision 2.0. The resume signaling (Full-speed 'K') is driven on the port as long as this bit remains a 1. This bit will remain a 1 until the port has switched to the high-speed idle. Writing a 0 has no effect because the port controller will time the resume operation and clear the bit when the port control state switches to HS or FS idle. This field is 0 if Port Power (PP) is 0 in host mode. This bit is not-EHCI compatible. In Device Mode: After the device has been in Suspend State for 5 ms or more, software must set this bit to 1 to drive resume signaling before clearing. The Device Controller will set this bit to 1 if a J-to-K transition is detected while the port is in the Suspend state. The bit will be cleared when the device returns to normal operation. Also, when this bit transitions to a 1 because a J-to-K transition has been detected, the Port Change Detect bit in the USBSTS register is also set to 1.
5	**OCC**	RW	0x0	Over-Current Change. 0 = Default. 1 = This bit gets set to 1 when there is a change to Over-Current Active. Software clears this bit by writing a 1 to this bit position. For host/OTG implementations, the user can provide over-current detection to the vbus_pwr_fault input for this condition. For device-only implementations, this bit shall always be 0.

Table 8-64. HW_USBCTRL_PORTSC1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
4	**OCA**	RW	0x0	Over-Current Active. 0 = This port does not have an over-current condition. 1 = This port currently has an over-current condition. Default = 0. This bit will automatically transition from 1 to 0 when the over current condition is removed. For host/OTG implementations the user can provide over-current detection to the vbus_pwr_fault input for this condition. For device-only implementations this bit shall always be 0.
3	**PEC**	RW	0x0	Port Enable/Disable Change. 0 = No change. 1 = Port enabled/disabled status has changed. Default = 0. In Host Mode: For the root hub, this bit gets set to a 1 only when a port is disabled due to disconnect on the port or due to the appropriate conditions existing at the EOF2 point (See Chapter 11 of the USB Specification). Software clears this by writing a 1 to it. This field is 0 if Port Power (PP) is 0. In Device Mode: The device port is always enabled. (This bit will be 0)
2	**PE**	RW	0x0	Port Enabled/Disabled. 0 = Disable. 1 = Enable. Default = 0. In Host Mode: Ports can only be enabled by the host controller as a part of the reset and enable. Software cannot enable a port by writing a 1 to this field. Ports can be disabled by either a fault condition (disconnect event or other fault condition) or by the host software. Note that the bit status does not change until the port state actually changes. There may be a delay in disabling or enabling a port due to other host controller and bus events. When the port is disabled, (0) downstream propagation of data is blocked except for reset. This field is 0 if Port Power (PP) is 0 in host mode. In Device Mode: The device port is always enabled. (This bit will be 1)

Table 8-64. HW_USBCTRL_PORTSC1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
1	**CSC**	RW	0x0	Connect Status Change. 0 = No change. 1 = Change in Current Connect Status. Default = 0. In Host Mode: Indicates a change has occurred in the port's Current Connect Status. The host/device controller sets this bit for all changes to the port device connect status, even if system software has not cleared an existing connect status change. For example, the insertion status changes twice before system software has cleared the changed condition, hub hardware will be 'setting' an already-set bit (i.e., the bit will remain set). Software clears this bit by writing a 1 to it. This field is 0 if Port Power (PP) is 0 in host mode. In Device Mode: This bit is undefined in device controller mode.
0	**CCS**	RW	0x0	Current Connect Status. In Host Mode: 0 = No device is present. 1 = Device is present on port. Default = 0. This value reflects the current state of the port, and may not correspond directly to the event that caused the Connect Status Change bit (Bit 1) to be set. This field is 0 if Port Power (PP) is 0 in host mode. In Device Mode: 0 = Not Attached. 1 = Attached. Default = 0. A 1 indicates that the device successfully attached and is operating in either high speed or full speed as indicated by the High Speed Port bit in this register. A 0 indicates that the device did not attach successfully or was forcibly disconnected by the software writing a 0 to the Run bit in the USBCMD register. It does not state the device being disconnected or suspended.

DESCRIPTION:

port status and control

EXAMPLE:

```
Empty Example.
```

8.6.33 OTG Status and Control Register Description

Host Controller: A host controller implements one On-The-Go (OTG) Status and Control register corresponding to Port 0 of the host controller. The OTGSC register has four sections: OTG Interrupt Enables (Read/Write) OTG Interrupt Status (Read/Write to Clear) OTG Status Inputs (Read-Only) OTG Controls (Read/Write) The status inputs are debounced using a 1-ms time constant. Values on the status

inputs that do not persist for more than 1 ms will not cause an update of the status input register, or cause an OTG interrupt. See also USBMODE register. The default value of this register is 0x00000120.

HW_USBCTRL_OTGSC 0x1a4

Table 8-65. HW_USBCTRL_OTGSC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2	DPIE	ONEMSE	BSEIE	BSVIE	ASVIE	AVVIE	IDIE	RSVD1	DPIS	ONEMSS	BSEIS	BSVIS	ASVIS	AVVIS	IDIS	RSVD0	DPS	ONEMST	BSE	BSV	ASV	AVV	ID	HABA	HADP	IDPU	DP	OT	HAAR	VC	VD

Table 8-66. HW_USBCTRL_OTGSC Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSVD2**	RO	0x0	Reserved.
30	**DPIE**	RW	0x0	Data Pulse Interrupt Enable
29	**ONEMSE**	RW	0x0	1 Millisecond Timer Interrupt Enable
28	**BSEIE**	RW	0x0	B Session End Interrupt Enable. Setting this bit enables the B session end interrupt.
27	**BSVIE**	RW	0x0	B Session Valid Interrupt Enable. Setting this bit enables the B session valid interrupt.
26	**ASVIE**	RW	0x0	A Session Valid Interrupt Enable. Setting this bit enables the A session valid interrupt.
25	**AVVIE**	RW	0x0	A VBus Valid Interrupt Enable. Setting this bit enables the A VBus valid interrupt.
24	**IDIE**	RW	0x0	USB ID Interrupt Enable. Setting this bit enables the USB ID interrupt.
23	**RSVD1**	RO	0x0	Reserved.
22	**DPIS**	RW	0x0	Data Pulse Interrupt Status. This bit is set when data bus pulsing occurs on DP or DM. Data bus pulsing is only detected when USBMODE.CM = Host (11) and PORTSC(0).PortPower = Off (0). Software must write a 1 to clear this bit.
21	**ONEMSS**	RW	0x0	1 Millisecond Timer Interrupt Status. This bit is set once every millisecond. Software must write a 1 to clear this bit.
20	**BSEIS**	RW	0x0	B Session End Interrupt Status. This bit is set when VBus has fallen below the B session end threshold. Software must write a 1 to clear this bit.
19	**BSVIS**	RW	0x0	B Session Valid Interrupt Status. This bit is set when VBus has either risen above or fallen below the B session valid threshold (0.8 VDC). Software must write a 1 to clear this bit.
18	**ASVIS**	RW	0x0	A Session Valid Interrupt Status. This bit is set when VBus has either risen above or fallen below the A session valid threshold (0.8 VDC). Software must write a 1 to clear this bit.

Table 8-66. HW_USBCTRL_OTGSC Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
17	**AVVIS**	RW	0x0	A VBus Valid Interrupt Status. This bit is set when VBus has either risen above or fallen below the VBus valid threshold (4.4 VDC) on an A device. Software must write a 1 to clear this bit.
16	**IDIS**	RW	0x0	USB ID Interrupt Status. This bit is set when a change on the ID input has been detected. Software must write a 1 to clear this bit.
15	**RSVD0**	RO	0x0	Reserved.
14	**DPS**	RW	0x0	Data Bus Pulsing Status. A 1 indicates data bus pulsing is being detected on the port.
13	**ONEMST**	RW	0x0	1 Millisecond Timer Toggle. This bit toggles once per millisecond.
12	**BSE**	RO	0x0	B Session End. Indicates VBus is below the B session end threshold.
11	**BSV**	RO	0x0	B Session Valid. Indicates VBus is above the B session valid threshold.
10	**ASV**	RO	0x0	A Session Valid. Indicates VBus is above the A session valid threshold.
9	**AVV**	RO	0x0	A VBus Valid. Indicates VBus is above the A VBus valid threshold.
8	**ID**	RO	0x1	USB ID. 0 = A device. 1 = B device.
7	**HABA**	RW	0x0	Hardware Assist B-Disconnect to A-connect. 0 = Disabled. 1 = Enable automatic B-disconnect to A-connect sequence.
6	**HADP**	RW	0x0	Hardware Assist Data-Pulse 1 = Start Data Pulse Sequence.
5	**IDPU**	RW	0x1	ID Pullup. This bit provide control over the ID pullup resister. 0 = off. 1 = on (default). When this bit is 0, the ID input will not be sampled.
4	**DP**	RW	0x0	Data Pulsing. Setting this bit causes the pullup on DP to be asserted for data pulsing during SRP.
3	**OT**	RW	0x0	OTG Termination. This bit must be set when the OTG device is in device mode, this controls the pulldown on DM.
2	**HAAR**	RW	0x0	Hardware Assist Auto-Reset. 0 = Disabled. 1 = Enable automatic reset after connect on host port.
1	**VC**	RW	0x0	VBUS Charge. Setting this bit causes the VBus line to be charged. This is used for VBus pulsing during SRP.
0	**VD**	RW	0x0	VBUS_Discharge. Setting this bit causes VBus to discharge through a resistor.

DESCRIPTION:

OTG status/control

EXAMPLE:

```
Empty Example.
```

8.6.34 USB Device Mode Register Description

Default Value:0x00000000 (implementation OTGmode not selected).

HW_USBCTRL_USBMODE 0x1a8

Table 8-67. HW_USBCTRL_USBMODE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																										VBPS	SDIS	SLOM	ES	CM	

Table 8-68. HW_USBCTRL_USBMODE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:6	**RSVD**	RO	0x0	Reserved.
5	**VBPS**	RW	0x0	Vbus Power Select 0 = Output is 0. 1 = Output is 1. This bit is connected to the vbus_pwr_select output and can be used for any generic control but is named to be used by logic that selects between an on-chip Vbus power source (charge pump) and an off-chip source in systems when both are available.

Table 8-68. HW_USBCTRL_USBMODE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
4	**SDIS**	RW	0x0	Stream Disable Mode. 0 = Inactive (default). 1 = Active. In Device Mode: Setting to a 1 disables double priming on both RX and TX for low bandwidth systems. This mode, when enabled, ensures that the RX and TX buffers are sufficient to contain an entire packet, so that the usual double buffering scheme is disabled to prevent overruns/underruns in bandwidth limited systems. Note: In High Speed Mode, all packets received will be responded to with a NYET handshake when stream disable is active. In Host Mode: Setting to a 1 ensures that overruns/underruns of the latency FIFO are eliminated for low bandwidth systems where the RX and TX buffers are sufficient to contain the entire packet. Enabling stream disable also has the effect of ensuring the TX latency is filled to capacity before the packet is launched onto the USB. Note: Time duration to pre-fill the FIFO becomes significative when stream disable is active. See TXFILLTUNING and TXTTFILLTUNING to characterize the adjustments needed for the scheduler when using this feature. Note: The use of this feature substantially limits of the overall USB performance that can be achieved.
3	**SLOM**	RW	0x0	Setup Lockout Mode. In device mode, this bit controls behavior of the setup lock mechanism. 0 = Setup Lockouts On (default). 1 = Setup Lockouts Off (DCD requires use of Setup Data Buffer Tripwire in USBCMD).

Table 8-68. HW_USBCTRL_USBMODE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**ES**	RW	0x0	Endian Select. This bit can change the byte ordering of the transfer buffers to match the host microprocessor bus architecture. The bit fields in the microprocessor interface and the DMA data structures (including the setup buffer within the device QH) are unaffected by the value of this bit, because they are based upon 32-bit words. 0 = Little Endian (default): First byte referenced in least significant byte of 32-bit word. 1 = Big Endian: First byte referenced in most significant byte of 32-bit word.
1:0	**CM**	RW	0x0	Controller Mode. Controller mode is defaulted to the proper mode for host only and device only implementations. For those designs that contain both host & device capability, the controller will default to an idle state and will need to be initialized to the desired operating mode after reset. For combination host/device controllers, this register can only be written once after reset. If it is necessary to switch modes, software must reset the controller by writing to the RESET bit in the USBCMD register before reprogramming this register. IDLE = 0x0 IDLE. DEVICE = 0x2 DEVICE. HOST = 0x3 HOST.

DESCRIPTION:

device mode

EXAMPLE:

```
Empty Example.
```

8.6.35 Endpoint Setup Status Register Description

HW_USBCTRL_ENDPTSETUPSTAT 0x1ac

Table 8-69. HW_USBCTRL_ENDPTSETUPSTAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																											ENDPTSETUPSTAT				

Table 8-70. HW_USBCTRL_ENDPTSETUPSTAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSVD**	RO	0x0	Reserved.
4:0	**ENDPTSETUPSTAT**	RW	0x0	Setup Endpoint Status. For every setup transaction that is received, a corresponding bit in this register is set to 1. Software must clear or acknowledge the setup transfer by writing a 1 to a respective bit after it has read the setup data from Queue head. The response to a setup packet as in the order of operations and total response time is crucial to limit bus time outs while the setup lock out mechanism is engaged. This register is only used in device mode.

DESCRIPTION:

endpoint setup status

EXAMPLE:

```
Empty Example.
```

8.6.36 Endpoint Prime Register Description

This register is used in device mode only.

HW_USBCTRL_ENDPTPRIME 0x1b0

Table 8-71. HW_USBCTRL_ENDPTPRIME

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1											PETB					RSVD0											PERB				

Table 8-72. HW_USBCTRL_ENDPTPRIME Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:21	**RSVD1**	RO	0x0	Reserved.
20:16	**PETB**	RW	0x0	Prime Endpoint Transmit Buffer. For each endpoint, a corresponding bit is used to request that a buffer be prepared for a transmit operation in order to respond to a USB IN/INTERRUPT transaction. Software should write a 1 to the corresponding bit when posting a new transfer descriptor to an endpoint. Hardware will automatically use this bit to begin parsing for a new transfer descriptor from the queue head and prepare a transmit buffer. Hardware will clear this bit when the associated endpoint(s) is (are) successfully primed. Note: These bits will be momentarily set by hardware during hardware re-priming operations when a dTD is retired, and the dQH is updated. PETB[4] = Endpoint 4. PETB[3] = Endpoint 3. PETB[2] = Endpoint 2. PETB[1] = Endpoint 1. PETB[0] = Endpoint 0.
15:5	**RSVD0**	RO	0x0	Reserved.
4:0	**PERB**	RW	0x0	Prime Endpoint Receive Buffer. For each endpoint, a corresponding bit is used to request a buffer be prepared for a receive operation for when a USB host initiates a USB OUT transaction. Software should write a 1 to the corresponding bit whenever posting a new transfer descriptor to an endpoint. Hardware will automatically use this bit to begin parsing for a new transfer descriptor from the queue head and prepare a receive buffer. Hardware will clear this bit when the associated endpoint(s) is (are) successfully primed. Note: These bits will be momentarily set by hardware during hardware re-priming operations when a dTD is retired, and the dQH is updated. PERB[4] = Endpoint 4. PERB[3] = Endpoint 3. PERB[2] = Endpoint 2. PERB[1] = Endpoint 1. PERB[0] = Endpoint 0.

DESCRIPTION:

endpoint prime request

EXAMPLE:

```
Empty Example.
```

8.6.37 Endpoint Flush Register Description

This register is used in device-mode only.

HW_USBCTRL_ENDPTFLUSH 0x1b4

Table 8-73. HW_USBCTRL_ENDPTFLUSH

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1											FETB					RSVD0											FERB				

Table 8-74. HW_USBCTRL_ENDPTFLUSH Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:21	**RSVD1**	RO	0x0	Reserved.
20:16	**FETB**	RW	0x0	Flush Endpoint Transmit Buffer. Writing a 1 to a bit(s) in this register will cause the associated endpoint(s) to clear any primed buffers. If a packet is in progress for 1 of the associated endpoints, then that transfer will continue until completion. Hardware will clear this register after the endpoint flush operation is successful. FETB[4] = Endpoint 4. FETB[3] = Endpoint 3. FETB[2] = Endpoint 2. FETB[1] = Endpoint 1. FETB[0] = Endpoint 0.
15:5	**RSVD0**	RO	0x0	Reserved.
4:0	**FERB**	RW	0x0	Flush Endpoint Receive Buffer. Writing a 1 to a bit(s) will cause the associated endpoint(s) to clear any primed buffers. If a packet is in progress for one of the associated endpoints, then that transfer will continue until completion. Hardware will clear this register after the endpoint flush operation is successful. FERB[4] = Endpoint 4. FERB[3] = Endpoint 3. FERB[2] = Endpoint 2. FERB[1] = Endpoint 1. FERB[0] = Endpoint 0.

DESCRIPTION:

endpoint flush request

EXAMPLE:

```
Empty Example.
```

8.6.38 Endpoint Status Register Description

This register is used in device mode only.

HW_USBCTRL_ENDPTSTAT 0x1b8

Table 8-75. HW_USBCTRL_ENDPTSTAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1											ETBR					RSVD0											ERBR				

Table 8-76. HW_USBCTRL_ENDPTSTAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:21	**RSVD1**	RO	0x0	Reserved.
20:16	**ETBR**	RO	0x0	Endpoint Transmit Buffer Ready. One bit for each endpoint indicates status of the respective endpoint buffer. This bit is set to a 1 by the hardware as a response to receiving a command from a corresponding bit in the ENDPTPRIME register. There will always be a delay between setting a bit in the ENDPTPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the ENDPTPRIME register. Buffer ready is cleared by USB reset, by the USB DMA system, or through the ENDPTFLUSH register. Note: These bits will be momentarily cleared by hardware during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated. ETBR[4] = Endpoint 4. ETBR[3] = Endpoint 3. ETBR[2] = Endpoint 2. ETBR[1] = Endpoint 1. ETBR[0] = Endpoint 0.
15:5	**RSVD0**	RO	0x0	Reserved.
4:0	**ERBR**	RO	0x0	Endpoint Receive Buffer Ready. One bit for each endpoint indicates status of the respective endpoint buffer. This bit is set to a 1 by the hardware as a response to receiving a command from a corresponding bit in the ENDPTPRIME register. There will always be a delay between setting a bit in the ENDPTPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the ENDPTPRIME register. Buffer ready is cleared by USB reset, by the USB DMA system, or through the ENDPTFLUSH register. Note: These bits will be momentarily cleared by hardware during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated. ERBR[4] = Endpoint 4. ERBR[3] = Endpoint 3. ERBR[2] = Endpoint 2. ERBR[1] = Endpoint 1. ERBR[0] = Endpoint 0.

DESCRIPTION:

endpoint ready

EXAMPLE:

```
Empty Example.
```

8.6.39 Endpoint Complete Register Description

This register is used in device-mode only.

HW_USBCTRL_ENDPTCOMPLETE 0x1bc

Table 8-77. HW_USBCTRL_ENDPTCOMPLETE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1											ETCE					RSVD0												ERCE			

Table 8-78. HW_USBCTRL_ENDPTCOMPLETE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:21	**RSVD1**	RO	0x0	Reserved.
20:16	**ETCE**	RW	0x0	Endpoint Transmit Complete Event. Each bit indicates a transmit event (IN/INTERRUPT) occurred and software should read the corresponding endpoint queue to determine the endpoint status. If the corresponding IOC bit is set in the Transfer Descriptor, then this bit will be set simultaneously with the USBINT. Writing a 1 will clear the corresponding bit in this register. ETCE[4] = Endpoint 4. ETCE[3] = Endpoint 3. ETCE[2] = Endpoint 2. ETCE[1] = Endpoint 1. ETCE[0] = Endpoint 0.
15:5	**RSVD0**	RO	0x0	Reserved.
4:0	**ERCE**	RW	0x0	Endpoint Receive Complete Event. Each bit indicates a received event (OUT/SETUP) occurred and software should read the corresponding endpoint queue to determine the transfer status. If the corresponding IOC bit is set in the Transfer Descriptor, then this bit will be set simultaneously with the USBINT. Writing a 1 will clear the corresponding bit in this register. ERCE[4] = Endpoint 4. ERCE[3] = Endpoint 3. ERCE[2] = Endpoint 2. ERCE[1] = Endpoint 1. ERCE[0] = Endpoint 0.

DESCRIPTION:

endpoint complete

EXAMPLE:

```
Empty Example.
```

8.6.40 Endpoint Control 0 Register Description

Every Device will implement Endpoint0 as a control endpoint. The default value of this register is 0x00800080.

HW_USBCTRL_ENDPTCTRL0 0x1c0

Table 8-79. HW_USBCTRL_ENDPTCTRL0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD6								TXE	RSVD5			TXT		RSVD4	TXS	RSVD3								RXE	RSVD2			RXT		RSVD1	RXS

Table 8-80. HW_USBCTRL_ENDPTCTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD6**	RO	0x0	Reserved.
23	**TXE**	RW	0x1	TX Endpoint Enable. 1 = Enabled. Endpoint0 is always enabled.
22:20	**RSVD5**	RO	0x0	Reserved. Bit reserved and should be read as zeroes.
19:18	**TXT**	RW	0x0	TX Endpoint Transmit Type. Endpoint0 is fixed as a Control endpoint. CONTROL = 0 Control.
17	**RSVD4**	RO	0x0	Reserved.
16	**TXS**	RW	0x0	Endpoint Stall. 0 = Endpoint OK (default). 1 = Endpoint Stalled. Software can write a 1 to this bit to force the endpoint to return a STALL handshake to the Host. It will continue returning STALL until the bit is cleared by software or it will automatically be cleared upon receipt of a new SETUP request. After receiving a SETUP request, this bit will continue to be cleared by hardware until the associated ENDPTSETUPSTAT bit is cleared. Note: There is a slight delay (50 clocks max.) between the ENDPTSETUPSTAT being cleared and hardware continuing to clear this bit. In most systems it is unlikely the DCD software will observe this delay. However, should the DCD observe that the stall bit is not set after writing a 1 to it, then follow this procedure: Continually write this stall bit until it is set OR until a new SETUP has been received by checking the associated ENDPTSETUPSTAT bit.

Table 8-80. HW_USBCTRL_ENDPTCTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:8	**RSVD3**	RO	0x0	Reserved. Bit reserved and should be read as zeroes.
7	**RXE**	RW	0x1	RX Endpoint Enable. 1 = Enabled. Endpoint0 is always enabled.
6:4	**RSVD2**	RO	0x0	Reserved. Bit reserved and should be read as zeroes.
3:2	**RXT**	RW	0x0	RX Endpoint Receive Type. Endpoint0 is fixed as a Control endpoint. CONTROL = 0 Control.
1	**RSVD1**	RO	0x0	Reserved.
0	**RXS**	RW	0x0	RX Endpoint Stall. 0 = Endpoint OK (default). 1 = Endpoint Stalled. Software can write a 1 to this bit to force the endpoint to return a STALL handshake to the Host. It will continue returning STALL until the bit is cleared by software or it will automatically be cleared upon receipt of a new SETUP request. After receiving a SETUP request, this bit will continue to be cleared by hardware until the associated ENDPTSETUPSTAT bit is cleared. Note: There is a slight delay (50 clocks max.) between the ENDPTSETUPSTAT being cleared and hardware continuing to clear this bit. In most systems it is unlikely the DCD software will observe this delay. However, should the DCD observe that the stall bit is not set after writing a 1 to it, then follow this procedure: Continually write this stall bit until it is set OR until a new SETUP has been received by checking the associated ENDPTSETUPSTAT bit.

DESCRIPTION:

endpoint control registers [0-n]

EXAMPLE:

```
Empty Example.
```

8.6.41 Endpoint Control 1 Register Description

Register HW_USBCTRL_ENDPTCTRL1 is the control register for endpoint 1 in a device. CAUTION: If one endpoint direction is enabled and the paired endpoint of opposite direction is disabled then the unused direction type must be changed from the default control-type to any other type (i.e., bulk type). Leaving an unconfigured endpoint control will cause undefined behavior for the data PID tracking on the active endpoint/direction.

HW_USBCTRL_ENDPTCTRL1 0x1c4

Table 8-81. HW_USBCTRL_ENDPTCTRL1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD6								TXE	TXR	TXI	RSVD5	TXT		TXD	TXS	RSVD3								RXE	RXR	RXI	RSVD2	RXT		RXD	RXS

Table 8-82. HW_USBCTRL_ENDPTCTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD6**	RO	0x0	Reserved.
23	**TXE**	RW	0x0	TX Endpoint Enable. 0 = Disabled (default). 1 = Enabled. An endpoint should be enabled only after it has been configured.
22	**TXR**	RW	0x0	TX Data Toggle Reset. Write 1 to reset PID sequence. Whenever a configuration event is received for this endpoint, software must write a 1 to this bit in order to synchronize the data PIDs between the host and device.
21	**TXI**	RW	0x0	TX Data Toggle Inhibit. 0 = PID Sequencing Enabled (default). 1 = PID Sequencing Disabled. This bit is only used for test and should always be written as 0. Writing a 1 to this bit will cause this endpoint to ignore the data toggle sequence and always transmit DATA0 for a data packet.
20	**RSVD5**	RO	0x0	Reserved.
19:18	**TXT**	RW	0x0	TX Endpoint Transmit Type. CONTROL = 0 Control. ISO = 1 Isochronous. BULK = 2 Bulk. INT = 3 Interrupt.
17	**TXD**	RW	0x0	TX Endpoint Data Source. 0 = Dual Port Memory Buffer/DMA Engine (default). Should always be written as 0.

Table 8-82. HW_USBCTRL_ENDPTCTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
16	**TXS**	RW	0x0	Endpoint Stall. 0 = Endpoint OK. 1 = Endpoint Stalled. This bit will be cleared automatically upon receipt of a SETUP request if this Endpoint is configured as a Control Endpoint and this bit will continue to be cleared by hardware until the associated ENDPTSETUPSTAT bit is cleared. Software can write a 1 to this bit to force the endpoint to return a STALL handshake to the Host. This control will continue to STALL until this bit is either cleared by software or automatically cleared as above for control endpoints. Note (control endpoint types only): There is a slight delay (50 clocks max.) between the ENDPTSETUPSTAT being cleared and hardware continuing to clear this bit. In most systems it is unlikely the DCD software will observe this delay. However, Should the DCD observe that the stall bit is not set after writing a 1 to it, then follow this procedure: Continually write this stall bit until it is set OR until a new SETUP has been received by checking the associated ENDPTSETUPSTAT bit.
15:8	**RSVD3**	RO	0x0	Reserved.
7	**RXE**	RW	0x0	RX Endpoint Enable. 0 = Disabled (default). 1 = Enabled. An Endpoint should be enabled only after it has been configured.
6	**RXR**	RW	0x0	Data Toggle Reset. Write 1 to reset PID Sequence. Whenever a configuration event is received for this endpoint, software must write a 1 to this bit in order to synchronize the data PIDs between the host and device.
5	**RXI**	RW	0x0	RX Data Toggle Inhibit. 0 = Disabled (default). 1 = Enabled. This bit is only used for test and should always be written as 0. Writing a 1 to this bit will cause this endpoint to ignore the data toggle sequence and always accept data packet regardless of their data PID.
4	**RSVD2**	RO	0x0	Reserved.
3:2	**RXT**	RW	0x0	RX Endpoint Receive Type. CONTROL = 0 Control. ISO = 1 Isochronous. BULK = 2 Bulk. INT = 3 Interrupt.

Table 8-82. HW_USBCTRL_ENDPTCTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
1	**RXD**	RW	0x0	RX Endpoint Data Sink. 0 = Dual Port Memory Buffer/DMA Engine (default). Should always be written as 0.
0	**RXS**	RW	0x0	RX Endpoint Stall. 0 = Endpoint OK (default). 1 = Endpoint Stalled. This bit will be cleared automatically upon receipt of a SETUP request if this Endpoint is configured as a Control Endpoint and this bit will continue to be cleared by hardware until the associated ENDPTSETUPSTAT bit is cleared. Software can write a 1 to this bit to force the endpoint to return a STALL handshake to the Host. This control will continue to STALL until this bit is either cleared by software or automatically cleared as above for control endpoints. Note (control endpoint types only): There is a slight delay (50 clocks maximum) between the ENDPTSETUPSTAT being cleared and hardware continuing to clear this bit. In most systems, it is unlikely the DCD software will observe this delay. However, should the DCD observe that the stall bit is not set after writing a 1 to it, then follow this procedure: Continually write this stall bit until it is set OR until a new SETUP has been received by checking the associated ENDPTSETUPSTAT bit.

DESCRIPTION:

endpoint control [0-n]

EXAMPLE:

```
Empty Example.
```

8.6.42 Endpoint Control 2 Register Description

Register HW_USBCTRL_ENDPTCTRL2 is the control register for endpoint 2 in a device. See the bit field defintions and descriptions of register HW_USBCTRL_ENDPTCTRL1. CAUTION: If one endpoint direction is enabled and the paired endpoint of opposite direction is disabled then the unused direction type must be changed from the default control-type to any other type (i.e., bulk type). Leaving an unconfigured endpoint control will cause undefined behavior for the data PID tracking on the active endpoint/direction.

HW_USBCTRL_ENDPTCTRL2 0x1c8

Table 8-83. HW_USBCTRL_ENDPTCTRL2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD6								TXE	TXR	TXI	RSVD5	TXT		TXD	TXS	RSVD3								RXE	RXR	RXI	RSVD2	RXT		RXD	RXS

Table 8-84. HW_USBCTRL_ENDPTCTRL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD6**	RO	0x0	
23	**TXE**	RW	0x0	
22	**TXR**	RW	0x0	
21	**TXI**	RW	0x0	
20	**RSVD5**	RO	0x0	
19:18	**TXT**	RW	0x0	CONTROL = 0 Control ISO = 1 Isochronous BULK = 2 Bulk INT = 3 Int
17	**TXD**	RW	0x0	
16	**TXS**	RW	0x0	
15:8	**RSVD3**	RO	0x0	
7	**RXE**	RW	0x0	
6	**RXR**	RW	0x0	
5	**RXI**	RW	0x0	
4	**RSVD2**	RO	0x0	
3:2	**RXT**	RW	0x0	CONTROL = 0 Control ISO = 1 Isochronous BULK = 2 Bulk INT = 3 Int
1	**RXD**	RW	0x0	
0	**RXS**	RW	0x0	

EXAMPLE:

```
Empty Example.
```

8.6.43 Endpoint Control 3 Register Description

Register HW_USBCTRL_ENDPTCTRL3 is the control register for endpoint 3 in a device. See the bit field defintions and descriptions of register HW_USBCTRL_ENDPTCTRL1. CAUTION: If one endpoint direction is enabled and the paired endpoint of opposite direction is disabled then the unused direction type must be changed from the default control-type to any other type (i.e., bulk type). Leaving an unconfigured endpoint control will cause undefined behavior for the data PID tracking on the active endpoint/direction.

HW_USBCTRL_ENDPTCTRL3 0x1cc

Table 8-85. HW_USBCTRL_ENDPTCTRL3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD6								TXE	TXR	TXI	RSVD5	TXT		TXD	TXS	RSVD3								RXE	RXR	RXI	RSVD2	RXT		RXD	RXS

Table 8-86. HW_USBCTRL_ENDPTCTRL3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD6**	RO	0x0	
23	**TXE**	RW	0x0	
22	**TXR**	RW	0x0	
21	**TXI**	RW	0x0	
20	**RSVD5**	RO	0x0	

Table 8-86. HW_USBCTRL_ENDPTCTRL3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:18	**TXT**	RW	0x0	CONTROL = 0 Control ISO = 1 Isochronous BULK = 2 Bulk INT = 3 Int
17	**TXD**	RW	0x0	
16	**TXS**	RW	0x0	
15:8	**RSVD3**	RO	0x0	
7	**RXE**	RW	0x0	
6	**RXR**	RW	0x0	
5	**RXI**	RW	0x0	
4	**RSVD2**	RO	0x0	
3:2	**RXT**	RW	0x0	CONTROL = 0 Control ISO = 1 Isochronous BULK = 2 Bulk INT = 3 Int
1	**RXD**	RW	0x0	
0	**RXS**	RW	0x0	

EXAMPLE:

```
Empty Example.
```

8.6.44 Endpoint Control 4 Register Description

Register HW_USBCTRL_ENDPTCTRL4 is the control register for endpoint 4 in a device. See the bit field defintions and descriptions of register HW_USBCTRL_ENDPTCTRL1. CAUTION: If one endpoint direction is enabled and the paired endpoint of opposite direction is disabled then the unused direction type must be changed from the default control-type to any other type (i.e., bulk type). Leaving an unconfigured endpoint control will cause undefined behavior for the data PID tracking on the active endpoint/direction.

HW_USBCTRL_ENDPTCTRL4 0x1d0

Table 8-87. HW_USBCTRL_ENDPTCTRL4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD6								TXE	TXR	TXI	RSVD5	TXT		TXD	TXS	RSVD3								RXE	RXR	RXI	RSVD2	RXT		RXD	RXS

Table 8-88. HW_USBCTRL_ENDPTCTRL4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD6**	RO	0x0	
23	**TXE**	RW	0x0	
22	**TXR**	RW	0x0	
21	**TXI**	RW	0x0	
20	**RSVD5**	RO	0x0	
19:18	**TXT**	RW	0x0	CONTROL = 0 Control ISO = 1 Isochronous BULK = 2 Bulk INT = 3 Int
17	**TXD**	RW	0x0	
16	**TXS**	RW	0x0	

Table 8-88. HW_USBCTRL_ENDPTCTRL4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:8	**RSVD3**	RO	0x0	
7	**RXE**	RW	0x0	
6	**RXR**	RW	0x0	
5	**RXI**	RW	0x0	
4	**RSVD2**	RO	0x0	
3:2	**RXT**	RW	0x0	CONTROL = 0 Control ISO = 1 Isochronous BULK = 2 Bulk INT = 3 Int
1	**RXD**	RW	0x0	
0	**RXS**	RW	0x0	

EXAMPLE:

```
Empty Example.
```

USBCTRL Block v1.4, Revision 1.11

8.6.45

Chapter 9
Integrated USB 2.0 PHY

This chapter describes the integrated USB 2.0 full-speed and high-speed PHY available on the i.MX23. Programmable registers are described in Section 9.4, "Programmable Registers."

9.1 Overview

The i.MX23 contains an integrated USB 2.0 PHY macrocell capable of connecting to PC host systems at the USB full-speed (FS) rate of 12 Mbits/s or at the USB 2.0 high-speed (HS) rate of 480 Mbits/s. See Figure 9-1 for a block diagram of the PHY. The integrated PHY provides a standard UTM interface. The USB_DP and USB_DN pins connect directly to a USB device connector.

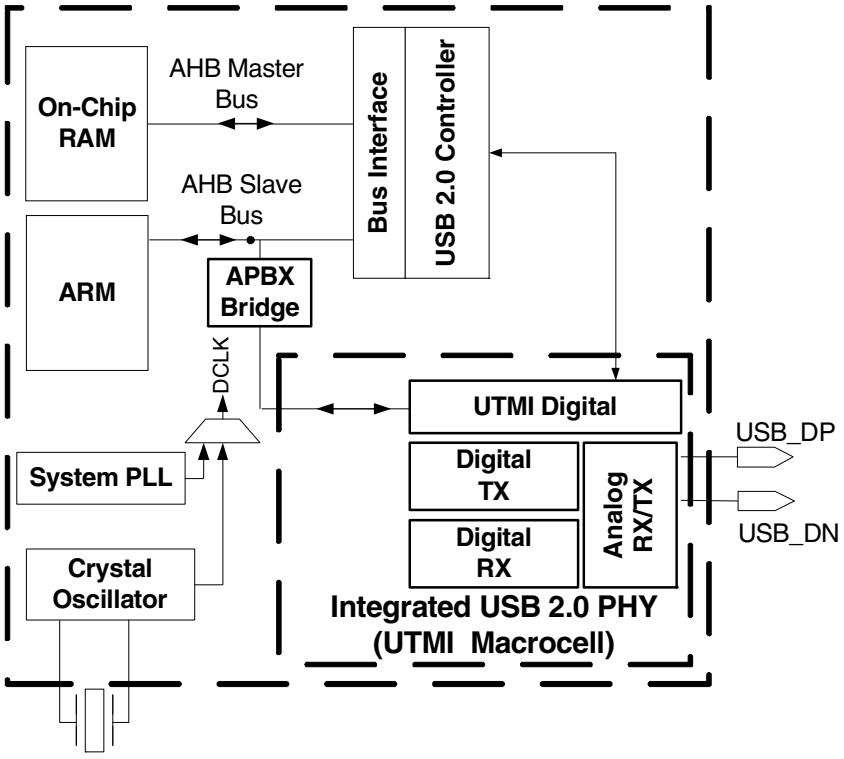


Figure 9-1. USB 2.0 PHY Block Diagram

The following subsections describe the external interfaces, internal interfaces, major blocks, and programmable registers that comprise the integrated USB 2.0 PHY.

9.2 Operation

The UTM provides a 16-bit interface to the USB controller. This interface is clocked at 30 MHz.

- The digital portions of the USBPHY block include the UTMI, digital transmitter, digital receiver, and the programmable registers.
- The analog transceiver section comprises an analog receiver and an analog transmitter, as shown in Figure 9-2.

9.2.1 UTMI

The UTMI block handles the line_state bits, reset buffering, suspend distribution, transceiver speed selection, and transceiver termination selection. The PLL supplies a 120-MHz signal to all of the digital logic. The UTMI block does a final divide-by-four to develop the 30-MHz clock used in the interface.

9.2.2 Digital Transmitter

The digital transmitter receives the 16-bit transmit data from the USB controller and handles the tx_valid, tx_validh and tx_ready handshake. In addition, it contains the transmit serializer that converts the 16-bit parallel words at 30 MHz to a single bitstream at 480 Mbit for high-speed or 12 Mbit for full-speed. It does this while implementing the bit-stuffing algorithm and the NRZI encoder that are used to remove the DC component from the serial bitstream. The output of this encoder is sent to the full-speed (FS) or high-speed (HS) drivers in the analog transceiver section's transmitter block.

9.2.3 Digital Receiver

The digital receiver receives the raw serial bitstream from the full speed (FS) differential transceiver, and a 9X, 480-MHz sampled data from the high speed (HS) differential transceiver. As the phase of the USB host transmitter shifts relative to the local PLL, the receiver section's HS DLL tracks these changes to give a reliable sample of the incoming 480-Mbit/s bitstream. Since this sample point shifts relative to the PLL phase used by the digital logic, a rate-matching elastic buffer is provided to cross this clock domain boundary. Once the bitstream is in the local clock domain, an NRZI decoder and bit unstuffer restore the original payload data bitstream and pass it to a deserializer and holding register. The receive state machine handles the rx_valid, rx_validh, and handshake with the USB controller. The handshake is not interlocked, in that there is no rx_ready signal coming from the controller. The controller must take each 16-bit value as presented by the PHY. The receive state machine provides an rx_active signal to the controller that indicates when it is inside a valid packet (SYNC detected, etc.).

9.2.4 Analog Receiver

The analog receiver comprises five differential receivers, two single-ended receivers, and a 9X, 480-MHz HS data sampling module, as shown in Figure 9-2 and described further in this section.

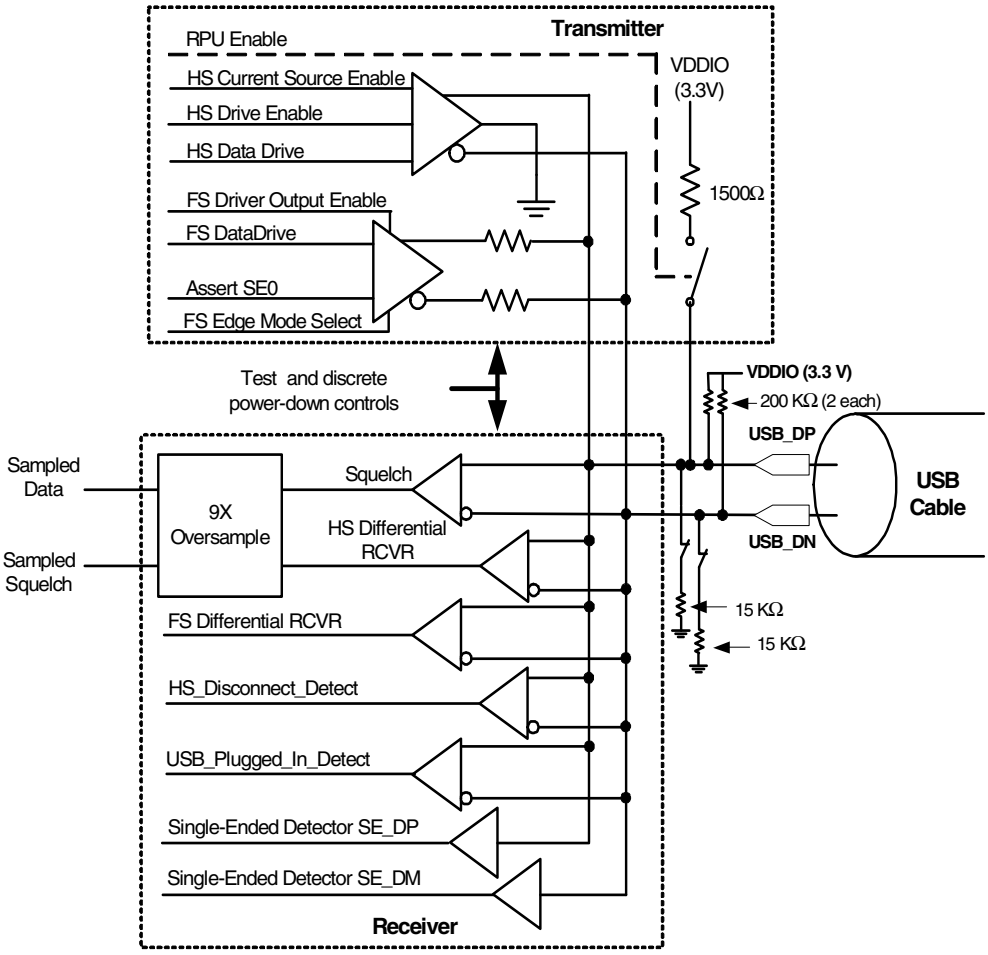


Figure 9-2. USB 2.0 PHY Analog Transceiver Block Diagram

9.2.4.1 HS Differential Receiver

The high-speed differential receiver is both a differential analog receiver and threshold comparator. Its output is a one if the differential signal is greater than a 0-V threshold. Its output is 0, otherwise. Its purpose is to discriminate the ± 400-mV differential voltage resulting from the high-speed drivers current flow into the dual 45Ω terminations found on each leg of the differential pair. The envelope or squelch detector, described below, ensures that the differential signal has sufficient magnitude to be valid. The HS differential receiver tolerates up to 500 mV of common mode offset.

9.2.4.2 Squelch Detector

The squelch detector is a differential analog receiver and threshold comparator. Its output is a 1 if the differential magnitude is less than a nominal 100-mV threshold. Its output is 0, otherwise. Its purpose is to invalidate the HS differential receiver when the incoming signal is simply too low to receive reliably.

9.2.4.3 FS Differential Receiver

The full-speed differential receiver is both a differential analog receiver and threshold comparator. The crossover voltage falls between 1.3 V and 2.0 V. Its output is a 1 when the USB_DP line is above the crossover point and the USB_DN line is below the crossover point.

9.2.4.4 HS Disconnect Detector

This host-side function is not used in i.MX23 applications, but is included to make a complete UTMI macrocell. It is a differential analog receiver and threshold comparator. Its output is a 1 if the differential magnitude is greater than a nominal 575-mV threshold. Its output is 0, otherwise.

9.2.4.5 USB Plugged-In Detector

The USB plugged-in detector looks for both USB_DP and USB_DN to be high. There is a pair of large on-chip pullup resistors (200KΩ) that hold both USB_DP and USB_DN high when the USB cable is not attached. The USB plugged-in detector signals a 0 in this case.

When in device mode, the host/hub interface that is *upstream* from the i.MX23 contains a 15KΩ pull-down resistor that easily overrides the 200KΩ pullup. When plugged in, at least one signal in the pair will be low, which will force the plugged-in detector's output high.

9.2.4.6 Single-Ended USB_DP Receiver

The single-ended USB_DP receiver output is high whenever the USB_DP input is above its nominal 1.8-V threshold.

9.2.4.7 Single-Ended USB_DN Receiver

The single-ended USB_DN receiver output is high whenever the USB_DN input is above its nominal 1.8-V threshold.

9.2.4.8 9X Oversample Module

The 9X oversample module uses nine identically spaced phases of the 480-MHz clock to sample a high speed bit data. The squelch signal is sampled only 1X.

9.2.5 Analog Transmitter

The analog transmitter comprises two differential drivers: one for high-speed signaling and one for full-speed signaling. It also contains the switchable 1.5KΩ pullup resistor. See Figure 9-2.

9.2.5.1 Switchable High-Speed 45Ω Termination Resistors

High-speed current mode differential signaling requires good 90Ω differential termination at each end of the USB cable. This results from switching in 45Ω terminating resistors from each signal line to ground at each end of the cable. Because each signal is parallel terminated with 45Ω at each end, each driver sees a 22.5Ω load. This is much too low of a load impedance for full-speed signaling levels—hence the need for switchable high-speed terminating resistors. Switchable trimming resistors are provided to tune the actual termination resistance of each device, as shown in Figure 9-3. The HW_USBPHY_TX_TXCAL45DP bit field, for example, allows one of 16 trimming resistor values to be placed in parallel with the 45Ω terminator on the USB_DP signal.

9.2.5.2 Full-Speed Differential Driver

The full-speed differential drivers are essentially “open drain” low-impedance pulldown devices that are switched in a differential mode for full-speed signaling, i.e., either one or the other device is turned on to signal the “J” state or the “K” state. These drivers are both turned on, simultaneously, for high-speed signaling. This has the effect of switching in both 45Ω terminating resistors. The tx_fs_hiz signal originates in the digital transmitter section. The hs_term signal that also controls these drivers comes from the UTMI.

9.2.5.3 High-Speed Differential Driver

The high-speed differential driver receives a 17.78-mA current from the constant current source and essentially steers it down either the USB_DP signal or the USB_DN signal or alternatively to ground. This current will produce approximately a 400-mV drop across the 22.5Ω termination seen by the driver when it is steered onto one of the signal lines. The approximately 17.78-mA current source is referenced back to the integrated voltage-band-gap circuit. The Iref, IBias, and V to I circuits are shared with the integrated battery charger.

9.2.5.4 Switchable 1.5KΩ USB_DP Pullup Resistor

The i.MX23 contains a switchable 1.5KΩ pullup resistor on the USB_DP signal. This resistor is switched on to tell the host/hub controller that a full-speed-capable device is on the USB cable, powered on, and ready. This resistor is switched off at power-on reset so the host does not recognize a USB device until processor software enables the announcement of a full-speed device.

9.2.5.5 Switchable 15KΩ USB_DP Pulldown Resistor

The i.MX23 contains a switchable 15KΩ pulldown resistor on both USB_DP and USB_DN signals. This is used in host mode to tell the device controller that a host is present.

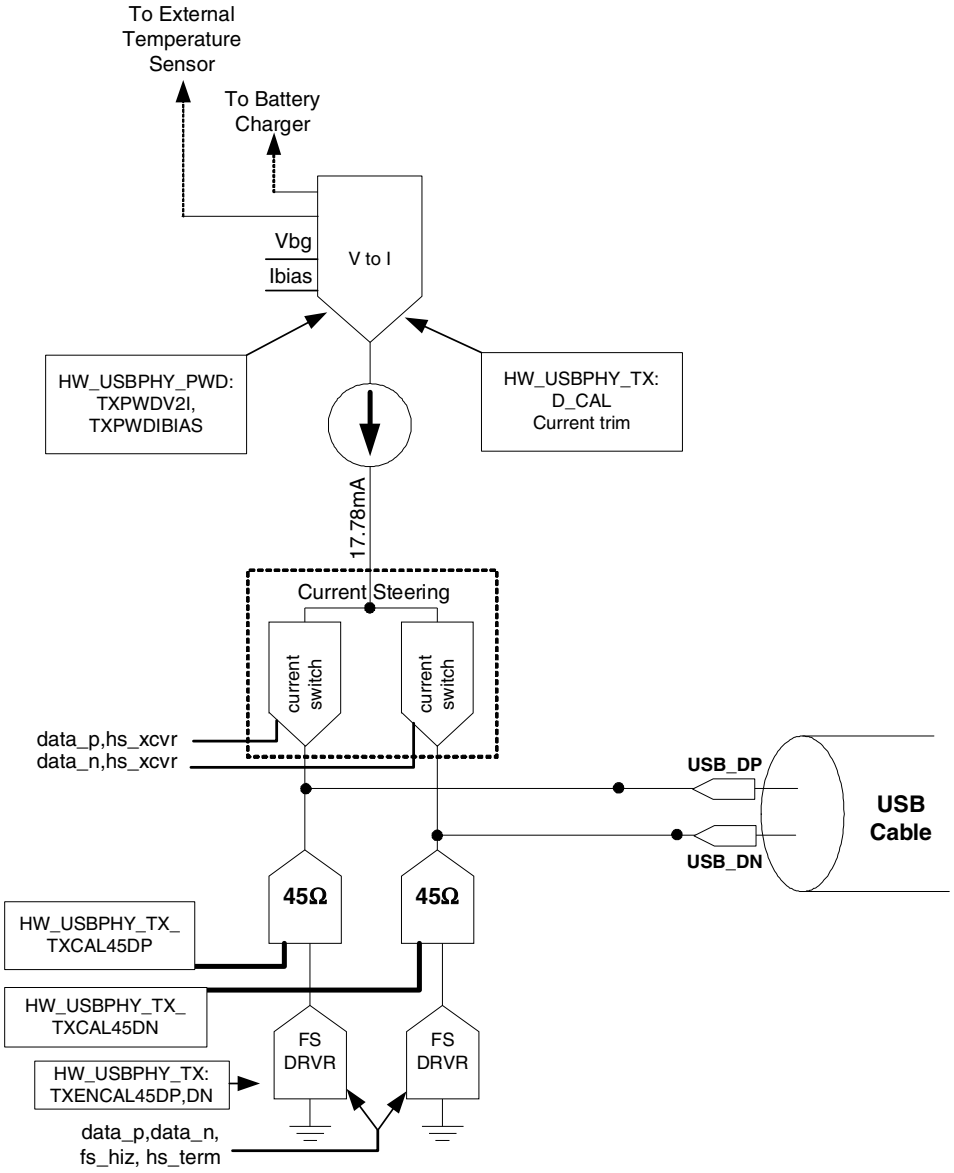


Figure 9-3. USB 2.0 PHY Transmitter Block Diagram

Table 9-1 summarizes the response of the PHY analog transmitter to various states of UTMI input and key transmit/receive state machine states.

Table 9-1. USB PHY Terminator States

UTMI OPMODE	UTM TERM	UTM XCVR	T/R	Function	45 Ω HIZ	1500 Ω HIZ	
00=Normal	0	0	X	HS	0	1	
	1	1	T	FS	0	0	
	1	1	R	FS	1	0	SUSPEND
	1	0	R	CHIRP	1	0	
	1	0	T	CHIRP	1	0	
	0	1	X	DISCONNECT	1	1	
01=NoDrive	0	0	T	HS	1	1	
	0	0	R	HS	1	1	
	1	1	X	FS	1	1	
	1	0	X	CHIRP	1	1	
	0	1	X	DISCONNECT	1	1	POR
10=NoNRZI NoBitStuff	0	0	X	HS	0	1	
	1	1	T	FS	0	0	
	1	1	R	FS	1	0	
	1	0	R	CHIRP	1	0	
	1	0	T	CHIRP	1	0	
	0	1	X	DISCONNECT	1	1	
11= Invalid	0	0	T	HS	1	1	
	0	0	R	HS	1	1	
	1	1	X	FS	1	1	
	1	0	X	CHIRP	1	1	
	0	1	X	DISCONNECT	1	1	

9.2.6 Recommended Register Configuration for USB Certification

The register settings in this section are recommended for passing USB certification.

The following settings lower the J/K levels to certifiable limits:

HW_USBPHY_TX_TXCAL45DP = 0x0
HW_USBPHY_TX_TXCAL45DN = 0x0
HW_USBPHY_TX_D_CAL = 0x7

The following settings help lower jitter in extreme conditions, for example, during heavy SDRAM usage with worst-case bit patterns:

HW_AUDIOOUT_REFCTRL_VDDXTAL_TO_VDDD = 0x1
HW_CLKCTRL_PLLCTRL0_CP_SEL = 0x2

Note that HW_AUDIOOUT_REFCTRL_VDDXTAL_TO_VDDD is controlled by the SFTRST and CLKGATE bits in the AUDIOIN block.

9.3 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, "Correct Way to Soft Reset a Block," for additional information on using the SFTRST and CLKGATE bit fields.

9.4 Programmable Registers

The USB 2.0 integrated PHY contains the following directly programmable registers.

9.4.1 USB PHY Power-Down Register Description

The USB PHY Power-Down Register provides overall control of the PHY power state.

HW_USBPHY_PWD	0x000
HW_USBPHY_PWD_SET	0x004
HW_USBPHY_PWD_CLR	0x008
HW_USBPHY_PWD_TOG	0x00c

Table 9-2. HW_USBPHY_PWD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2											RXPWDRX	RXPWDDIFF	RXPWD1PT1	RXPWDENV	RSVD1				TXPWDV2I	TXPWDIBIAS	TXPWDFS	RSVD0										

Table 9-3. HW_USBPHY_PWD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:21	**RSVD2**	RO	0x0	Reserved.
20	**RXPWDRX**	RW	0x1	0 = Normal operation. 1 = Power-down the entire USB PHY receiver block except for the full-speed differential receiver.
19	**RXPWDDIFF**	RW	0x1	0 = Normal operation. 1 = Power-down the USB high-speed differential receiver.
18	**RXPWD1PT1**	RW	0x1	0 = Normal operation. 1 = Power-down the USB full-speed differential receiver.
17	**RXPWDENV**	RW	0x1	0 = Normal operation. 1 = Power-down the USB high-speed receiver envelope detector (squelch signal).
16:13	**RSVD1**	RO	0x0	Reserved.

Table 9-3. HW_USBPHY_PWD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
12	**TXPWDV2I**	RW	0x1	0 = Normal operation. 1 = Power-down the USB PHY transmit V-to-I converter and the current mirror. Note that these circuits are shared with the battery charge circuit. Setting this to 1 does not power-down these circuits, unless the corresponding bit in the battery charger is also set for power-down.
11	**TXPWDIBIAS**	RW	0x1	0 = Normal operation. 1 = Power-down the USB PHY current bias block for the transmitter. This bit should be set only when the USB is in suspend mode. This effectively powers down the entire USB transmit path. Note that these circuits are shared with the battery charge circuit. Setting this bit to 1 does not power-down these circuits, unless the corresponding bit in the battery charger is also set for power-down.
10	**TXPWDFS**	RW	0x1	0 = Normal operation. 1 = Power-down the USB full-speed drivers. This turns off the current starvation sources and puts the drivers into high-impedance output.
9:0	**RSVD0**	RO	0x0	Reserved.

DESCRIPTION:

This register is used to control the USB PHY power state. See bits description for details.

EXAMPLE:

```
Empty Example.
```

9.4.2 USB PHY Transmitter Control Register Description

The USB PHY Transmitter Control Register handles the transmit controls.

HW_USBPHY_TX	0x010
HW_USBPHY_TX_SET	0x014
HW_USBPHY_TX_CLR	0x018
HW_USBPHY_TX_TOG	0x01c

Table 9-4. HW_USBPHY_TX

31 30	29	28 27 26	25	24	23 22	21	20	19 18 17 16	15 14	13	12	11 10 09 08	07 06 05 04	03 02 01 00
RSVD5	USBPHY_TX_EDGECTRL	USBPHY_TX_SYNC_INVERT	USBPHY_TX_SYNC_MUX	RSVD4	TXENCAL45DP	RSVD3	TXCAL45DP	RSVD2	TXENCAL45DN	RSVD1	TXCAL45DN	RSVD0	D_CAL	

Table 9-5. HW_USBPHY_TX Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSVD5**	RO	0x0	Reserved.
28:26	**USBPHY_TX_EDGECTRL**	RW	0x4	Controls the edge-rate of the current sensing transistors used in HS transmit. NOT FOR CUSTOMER USE.
25	**USBPHY_TX_SYNC_INVERT**	RW	0x0	NOT FOR CUSTOMER USE. While in testmode enables clock jitter analysis by resyncing data to the USB_DP and USB_DM pins. 0 = no Sync, 1 = Sync. When EMI clock is ungated, the USB data is resynced with EMI clock (for EMI jitter analysis), otherwise the USB clock is used for resync.
24	**USBPHY_TX_SYNC_MUX**	RW	0x0	Enables multiplexer to synchronize data from the USB_DP and USB_DM pins 0 = No sync, 1 = Sync. NOT FOR CUSTOMER USE.
23:22	**RSVD4**	RO	0x0	Reserved.
21	**TXENCAL45DP**	RW	0x0	This bit is not used and must remain cleared.
20	**RSVD3**	RO	0x0	Reserved.
19:16	**TXCAL45DP**	RW	0x6	Decode to select a 45-Ohm resistance to the USB_DP output pin. Maximum resistance = 0000. Resistance is centered by design at 0110.
15:14	**RSVD2**	RO	0x0	Reserved.
13	**TXENCAL45DN**	RW	0x0	This bit is not used and must remain cleared.
12	**RSVD1**	RO	0x0	Reserved.
11:8	**TXCAL45DN**	RW	0x6	Decode to select a 45-Ohm resistance to the USB_DN output pin. Maximum resistance = 0000. Resistance is centered by design at 0110.
7:4	**RSVD0**	RO	0x0	Reserved.
3:0	**D_CAL**	RW	0x7	Resistor Trimming Code: 0000 = 0.16% 0111 = Nominal 1111 = +25%

DESCRIPTION:

This register is used to control several items related USB Phy transmitter.

EXAMPLE:

```
Empty Example.
```

9.4.3 USB PHY Receiver Control Register Description

The USB PHY Receiver Control Register handles receive path controls.

HW_USBPHY_RX	0x020
HW_USBPHY_RX_SET	0x024
HW_USBPHY_RX_CLR	0x028
HW_USBPHY_RX_TOG	0x02c

Table 9-6. HW_USBPHY_RX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2									RXDBYPASS	RSVD1															DISCONADJ			RSVD0	ENVADJ		

Table 9-7. HW_USBPHY_RX Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:23	**RSVD2**	RO	0x0	Reserved.
22	**RXDBYPASS**	RW	0x0	0 = Normal operation. 1 = Use the output of the USB_DP single-ended receiver in place of the full-speed differential receiver. This test mode is intended for lab use only.
21:7	**RSVD1**	RO	0x0	Reserved.
6:4	**DISCONADJ**	RW	0x0	The DISCONADJ field adjusts the trip point for the disconnect detector: 0000 = Trip-Level Voltage is 0.57500 V 0001 = Trip-Level Voltage is 0.56875 V 0010 = Trip-Level Voltage is 0.58125 V 0011 = Trip-Level Voltage is 0.58750 V 01XX = Reserved 1XXX = Reserved
3	**RSVD0**	RO	0x0	Reserved.
2:0	**ENVADJ**	RW	0x0	The ENVADJ field adjusts the trip point for the envelope detector. 0000 = Trip-Level Voltage is 0.12500 V 0001 = Trip-Level Voltage is 0.10000 V 0010 = Trip-Level Voltage is 0.13750 V 0011 = Trip-Level Voltage is 0.15000 V 01XX = Reserved 1XXX = Reserved

DESCRIPTION:

This register is used to control several items related USB Phy receiver

EXAMPLE:

```
Empty Example.
```

9.4.4 USB PHY General Control Register Description

The USB PHY General Control Register handles Host controls. This register also includes interrupt enables and connectivity detect enables and results.

HW_USBPHY_CTRL	0x030
HW_USBPHY_CTRL_SET	0x034
HW_USBPHY_CTRL_CLR	0x038
HW_USBPHY_CTRL_TOG	0x03c

Table 9-8. HW_USBPHY_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	UTMI_SUSPENDM	HOST_FORCE_LS_SE0	RSVD3														DATA_ON_LRADC	DEVPLUGIN_IRQ	ENIRQDEVPLUGIN	RESUME_IRQ	ENIRQRESUMEDETECT	RSVD2	ENOTGIDDETECT	RSVD1	DEVPLUGIN_POLARITY	ENDEVPLUGINDETECT	HOSTDISCONDETECT_IRQ	ENIRQHOSTDISCON	ENHOSTDISCONDETECT	RSVD0

Table 9-9. HW_USBPHY_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	Writing a 1 to this bit will soft-reset the HW_USBPHY_PWD, HW_USBPHY_TX, HW_USBPHY_RX, and HW_USBPHY_CTRL registers.
30	**CLKGATE**	RW	0x1	Gate UTMI Clocks. Clear to 0 to run clocks. Set to 1 to gate clocks. Set this to save power while the USB is not actively being used. Configuration state is kept while the clock is gated.
29	**UTMI_SUSPENDM**	RO	0x0	Used by the PHY to indicate a powered-down state. If all the power-down bits in the HW_USBPHY_PWD are enabled, UTMI_SUSPENDM will be 0, otherwise 1. UTMI_SUSPENDM is negative logic, as required by the UTMI specification.
28	**HOST_FORCE_LS_SE0**	RW	0x0	Forces the next FS packet that is transmitted to have a EOP with low-speed timing. This bit is used in host mode for the resume sequence. After the packet is transferred, this bit is cleared. The design can use this function to force the LS SE0 or use the HW_USBPHY_CTRL_UTMI_SUSPENDM to trigger this event when leaving suspend. This bit is used in conjunction with HW_USBPHY_DEBUG_HOST_RESUME_DEBUG.
27:14	**RSVD3**	RO	0x0	Reserved.

Table 9-9. HW_USBPHY_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
13	**DATA_ON_LRADC**	RW	0x0	Enables the LRADC to monitor USB_DP and USB_DM. This is for use in non-USB modes only.
12	**DEVPLUGIN_IRQ**	RW	0x0	Indicates that the device is connected. Reset this bit by writing a 1 to the SCT clear address space and not by a general write.
11	**ENIRQDEVPLUGIN**	RW	0x0	Enables interrupt for the detection of connectivity to the USB line.
10	**RESUME_IRQ**	RW	0x0	Indicates that the host is sending a wake-up after suspend. This bit is also set on a reset during suspend. Use this bit to wake up from suspend for either the resume or the reset case. Reset this bit by writing a 1 to the SCT clear address space and not by a general write.
9	**ENIRQRESUMEDETECT**	RW	0x0	Enables interrupt for detection of a non-J state on the USB line. This should only be enabled after the device has entered suspend mode.
8	**RSVD2**	RO	0x0	Reserved.
7	**ENOTGIDDETECT**	RW	0x0	Enables circuit to detect resistance of MiniAB ID pin.
6	**RSVD1**	RO	0x0	Reserved.
5	**DEVPLUGIN_POLARITY**	RW	0x0	For device mode, if this bit is cleared to 0, then it trips the interrupt if the device is plugged in. If set to 1, then it trips the interrupt if the device is unplugged.
4	**ENDEVPLUGINDETECT**	RW	0x0	For device mode, enables 200-KOhm pullups for detecting connectivity to the host.
3	**HOSTDISCONDETECT_IRQ**	RW	0x0	Indicates that the device has disconnected in high-speed mode. Reset this bit by writing a 1 to the SCT clear address space and not by a general write.
2	**ENIRQHOSTDISCON**	RW	0x0	Enables interrupt for detection of disconnection to Device when in high-speed host mode. This should be enabled after ENDEVPLUGINDETECT is enabled.
1	**ENHOSTDISCONDETECT**	RW	0x0	For host mode, enables high-speed disconnect detector. This signal allows the override of enabling the detection that is normally done in the UTMI controller. The UTMI controller enables this circuit whenever the host sends a start-of-frame packet. Due to a on chip issue (Errata #2791), software must pay attention to when to assert the ENHOSTDISCONDETECT bit in HW_USBPHY_CTRL register: 1. Only set HW_USBPHY_CTRL.ENHOSTDISCONDETECT during high speed host mode. 2. Do not set HW_USBPHY_CTRL.ENHOSTDISCONDETECT during the reset and speed negotiation period. 3. Do not set HW_USBPHY_CTRL.ENHOSTDISCONDETECT during host suspend/resume sequence.
0	**RSVD0**	RO	0x0	Reserved.

DESCRIPTION:

This register is used to control high-level items related USB PHY.

EXAMPLE:

```
Empty Example.
```

9.4.5 USB PHY Status Register Description

The USB PHY Status Register holds results of IRQ and other detects.

HW_USBPHY_STATUS 0x040

Table 9-10. HW_USBPHY_STATUS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4																					RESUME_STATUS	RSVD3	OTGID_STATUS	RSVD2	DEVPLUGIN_STATUS	RSVD1		HOSTDISCONDETECT_STATUS	RSVD0		

Table 9-11. HW_USBPHY_STATUS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:11	**RSVD4**	RO	0x0	Reserved.
10	**RESUME_STATUS**	RO	0x0	Indicates that the host is sending a wake-up after suspend and has triggered an interrupt.
9	**RSVD3**	RO	0x0	Reserved.
8	**OTGID_STATUS**	RW	0x0	Indicates the results of ID pin on MiniAB plug. False (0) is when ID resistance is less than Ra_Plug_ID, indicating host (A) side. True (1) is when ID resistance is greater than Rb_Plug_ID, indicating device (B) side.
7	**RSVD2**	RO	0x0	Reserved.
6	**DEVPLUGIN_STATUS**	RO	0x0	Indicates that the device has been connected on the USB_DP and USB_DM lines.
5:4	**RSVD1**	RO	0x0	Reserved.
3	**HOSTDISCONDETECT_STATUS**	RO	0x0	Indicates that the device has disconnected while in high-speed host mode.
2:0	**RSVD0**	RO	0x0	Reserved.

DESCRIPTION:

This register is a status register and contains IRQ and other status.

EXAMPLE:

```
Empty Example.
```

9.4.6 USB PHY Debug Register Description

This register is used to debug the USB PHY.

HW_USBPHY_DEBUG	0x050
HW_USBPHY_DEBUG_SET	0x054
HW_USBPHY_DEBUG_CLR	0x058
HW_USBPHY_DEBUG_TOG	0x05c

Table 9-12. HW_USBPHY_DEBUG

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD3	CLKGATE	HOST_RESUME_DEBUG	SQUELCHRESETLENGTH				ENSQUELCHRESET	RSVD2			SQUELCHRESETCOUNT					RSVD1			ENTX2RXCOUNT	TX2RXCOUNT			RSVD0		ENHSTPULLDOWN		HSTPULLDOWN		DEBUG_INTERFACE_HOLD	OTGIDPIOLOCK

Table 9-13. HW_USBPHY_DEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSVD3**	RO	0x0	Reserved.
30	**CLKGATE**	RW	0x1	Gate Test Clocks. Clear to 0 for running clocks. Set to 1 to gate clocks. Set this to save power while the USB is not actively being used. Configuration state is kept while the clock is gated.
29	**HOST_RESUME_DEBUG**	RW	0x1	Choose to trigger the host resume SE0 with HOST_FORCE_LS_SE0 = 0 or UTMI_SUSPEND = 1.
28:25	**SQUELCHRESETLENGTH**	RW	0xf	Duration of RESET in terms of the number of 480-MHz cycles.
24	**ENSQUELCHRESET**	RW	0x1	Set bit to allow squelch to reset high-speed receive.
23:21	**RSVD2**	RO	0x0	Reserved.
20:16	**SQUELCHRESETCOUNT**	RW	0x18	Delay in between the detection of squelch to the reset of high-speed RX.
15:13	**RSVD1**	RO	0x0	Reserved.
12	**ENTX2RXCOUNT**	RW	0x0	Set this bit to allow a countdown to transition in between TX and RX.
11:8	**TX2RXCOUNT**	RW	0x0	Delay in between the end of transmit to the beginning of receive. This is a Johnson count value and thus will count to 8.
7:6	**RSVD0**	RO	0x0	Reserved.
5:4	**ENHSTPULLDOWN**	RW	0x0	Set bit 5 to 1 to override the control of the USB_DP 15-KOhm pulldown. Set bit 4 to 1 to override the control of the USB_DM 15-KOhm pulldown. Clear to 0 to disable.

Table 9-13. HW_USBPHY_DEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
3:2	**HSTPULLDOWN**	RW	0x0	Set bit 3 to 1 to pull down 15-KOhm on USB_DP line. Set bit 2 to 1 to pull down 15-KOhm on USB_DM line. Clear to 0 to disable.
1	**DEBUG_INTERFACE_HOLD**	RW	0x0	Use holding registers to assist in timing for external UTMI interface.
0	**OTGIDPIOLOCK**	RW	0x0	Once OTG ID is determined from HW_USBPHY_STATUS_OTGID_STATUS, use this to hold the value. This is to save power for the comparators that are used to determine the ID status.

DESCRIPTION:

Register not intended for customer use.

EXAMPLE:

```
Empty Example.
```

9.4.7 UTMI Debug Status Register 0 Description

The UTMI Debug Status Register 0 holds multiple views for counters and status of state machines. This is used in conjunction with the HW_USBPHY_DEBUG1.DBG_ADDRESS field to choose which function to view. The default is described in the bit fields below and is used to count errors.

HW_USBPHY_DEBUG0_STATUS 0x060

Table 9-14. HW_USBPHY_DEBUG0_STATUS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SQUELCH_COUNT						UTMI_RXERROR_FAIL_COUNT										LOOP_BACK_FAIL_COUNT																

Table 9-15. HW_USBPHY_DEBUG0_STATUS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**SQUELCH_COUNT**	RO	0x0	Running count of the squelch reset instead of normal end for HS RX.

Table 9-15. HW_USBPHY_DEBUG0_STATUS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
25:16	**UTMI_RXERROR_FAIL_COUNT**	RO	0x0	Running count of the UTMI_RXERROR.
15:0	**LOOP_BACK_FAIL_COUNT**	RO	0x0	Running count of the failed pseudo-random generator loopback. Each time entering testmode, counter goes to "900d" and will count up for every detected packet failure in digital/analog loopback tests.

DESCRIPTION:

Register not intended for customer use.

EXAMPLE:

```
Empty Example.
```

9.4.8 UTMI Debug Status Register 1 Description

Chooses the muxing of the debug register to be shown in HW_USBPHY_DEBUG0_STATUS.

HW_USBPHY_DEBUG1	0x070
HW_USBPHY_DEBUG1_SET	0x074
HW_USBPHY_DEBUG1_CLR	0x078
HW_USBPHY_DEBUG1_TOG	0x07c

Table 9-16. HW_USBPHY_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1																	ENTAILADJVD		ENTX2TX	RSVD0								DBG_ADDRESS			

Table 9-17. HW_USBPHY_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:15	**RSVD1**	RO	0x0	Reserved.
14:13	**ENTAILADJVD**	RW	0x0	Delay increment of the rise of squelch: 00 = Delay is nominal 01 = Delay is +20% 10 = Delay is -20% 11 = Delay is -40%
12	**ENTX2TX**	RW	0x1	This bit has no function in this system.
11:4	**RSVD0**	RO	0x0	Reserved.
3:0	**DBG_ADDRESS**	RW	0x0	Chooses the multiplexing of the debug register to be shown in HW_USBPHY_DEBUG0_STATUS.

DESCRIPTION:

Register not intended for customer use.

EXAMPLE:

```
Empty Example.
```

9.4.9 UTMI RTL Version Description

Fields for RTL Version.

HW_USBPHY_VERSION	0x080

Table 9-18. HW_USBPHY_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 9-19. HW_USBPHY_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x4	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x0	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
Empty Example.
```

9.4.10 USB PHY IP Block Register Description

The USB PHY IP Block Register IS FOR INTERNAL USE ONLY. It provides control of miscellaneous control bits found in other non-USB PIO control blocks that affects USB operations.

HW_USBPHY_IP	0x090
HW_USBPHY_IP_SET	0x094
HW_USBPHY_IP_CLR	0x098
HW_USBPHY_IP_TOG	0x09c

Table 9-20. HW_USBPHY_IP

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1							DIV_SEL		LFR_SEL		CP_SEL		TSTI_TX_DP	TSTI_TX_DM	ANALOG_TESTMODE	RSVD0													EN_USB_CLKS	PLL_LOCKED	PLL_POWER

Table 9-21. HW_USBPHY_IP Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSVD1**	RO	0x0	Reserved.
24:23	**DIV_SEL**	RW	0x0	TEST MODE FOR INTERNAL USE ONLY. This field is currently NOT supported. These bits came from the clkctrl PIO control block (clkctrl_pllctrl0_div_sel). DEFAULT = 0x0 PLL frequency is 480 MHz LOWER = 0x1 Lower the PLL fequency from 480MHz to 384MHz LOWEST = 0x2 Lower the PLL fequency from 480MHz to 288MHz UNDEFINED = 0x3 Undefined
22:21	**LFR_SEL**	RW	0x0	TEST MODE FOR INTERNAL USE ONLY. Adjusts loop filter resistor. These bits came from the clkctrl PIO control block (clkctrl_pllctrl0_lfr_sel). DEFAULT = 0x0 Default loop filter resistor TIMES_2 = 0x1 Doubles the loop filter resistor TIMES_05 = 0x2 Halves the loop filter resistor UNDEFINED = 0x3 Undefined
20:19	**CP_SEL**	RW	0x0	TEST MODE FOR INTERNAL USE ONLY. Adjusts charge pump current. These bits came from the clkctrl PIO control block (clkctrl_pllctrl0_cp_sel). DEFAULT = 0x0 Default charge pump current TIMES_2 = 0x1 Doubles charge pump current TIMES_05 = 0x2 Halves the charge pump current UNDEFINED = 0x3 Undefined
18	**TSTI_TX_DP**	RW	0x0	Analog testmode bit. Drives value on the DP pad. Default value is 1'b0. This bit came from the test control module.
17	**TSTI_TX_DM**	RW	0x0	Analog testmode bit. Drives value on the DM pad. Default value is 1'b0. This bit came from the test control module.
16	**ANALOG_TESTMODE**	RW	0x0	Analog testmode bit. Set to 0 for normal operation. Set to 1 for engineering debug of analog PHY block. This bit came from the test control module.
15:3	**RSVD0**	RO	0x0	Reserved.
2	**EN_USB_CLKS**	RW	0x0	If set to 0, 9-phase PLL outputs for USB PHY are powered down. If set to 1, 9-phase PLL outputs for USB PHY are powered up. Additionally, the UTMICLK120_GATE and UTMICLK30_GATE must be deasserted in the UTMI phy to enable USB operation. This bit came from the clkctrl PIO control block (clkctrl_pllctrl0_en_usb_clks).

Table 9-21. HW_USBPHY_IP Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
1	**PLL_LOCKED**	RW	0x0	Software controlled bit to indicate when the USB PLL has locked. Software needs to wait 10 us after enabling the PLL POWER bit (0) before asserting this bit. If set to 0, tells the UTMI module that the USB PLL has not locked. If set to 1, tells the UTMI module that the USB PLL has locked. Software should clear this bit prior to turning off the USB PLL. This bit came from the clkctrl module.
0	**PLL_POWER**	RW	0x0	USB PLL Power On (0 = PLL off; 1 = PLL On). Allow 10 us after turning the PLL on before using the PLL as a clock source. This is the time the PLL takes to lock to 480 MHz. This bit came from the clkctrl PIO control block (clkctrl_pllctrl0_power).

DESCRIPTION:

This register contains control bits in other non AUSTIN USB applications.

EXAMPLE:

```
Empty Example.
```

USBPHY Block v4.0, Revision 1.52

9.4.11

Chapter 10
AHB-to-APBH Bridge with DMA

This chapter describes the AHB-to-APBH bridge on the i.MX23, along with its central DMA function and implementation examples. Programmable registers are described in Section 10.5, "Programmable Registers."

10.1 Overview

The AHB-to-APBH bridge provides the i.MX23 with an inexpensive peripheral attachment bus running on the AHB's HCLK. (The "H" in APBH denotes that the APB*H* is synchronous to HCLK, as compared to APB*X*, which runs on the crystal-derived XCLK.)

As shown in Figure 10-1, the AHB-to-APBH bridge includes the AHB-to-APB PIO bridge for memory-mapped I/O to the APB devices, as well as a central DMA facility for devices on this bus and a vectored interrupt controller for the ARM926 core. Each one of the APB peripherals, including the vectored interrupt controller, are documented in their own chapters elsewhere in this document.

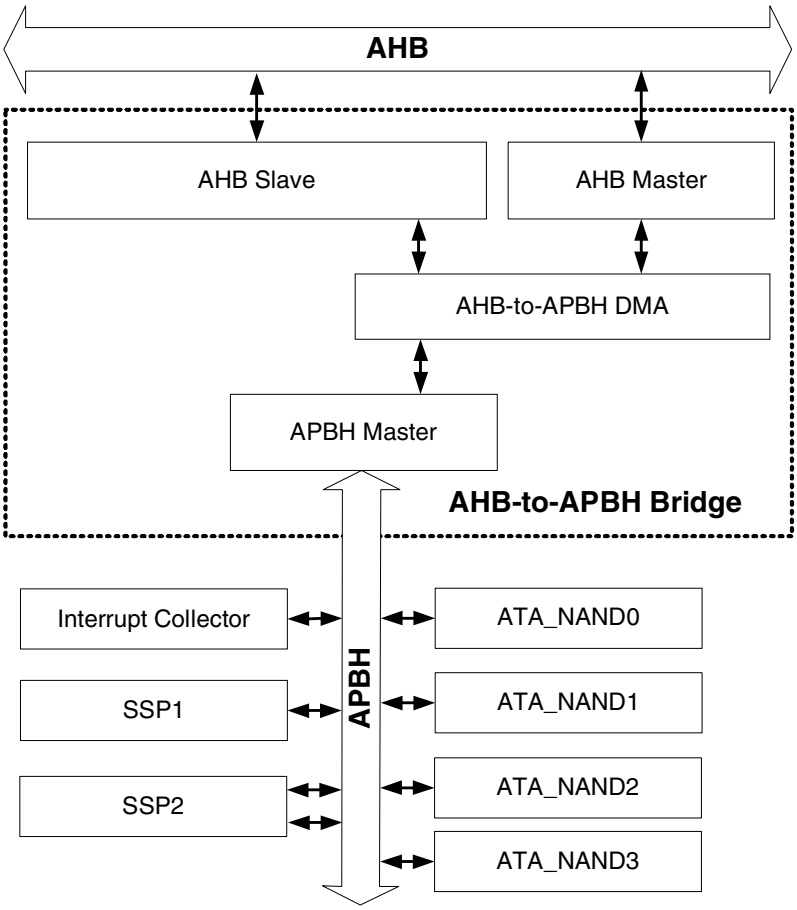


Figure 10-1. AHB-to-APBH Bridge DMA Block Diagram

The DMA controller uses the APBH bus to transfer read and write data to and from each peripheral. There is no separate DMA bus for these devices. Contention between the DMA's use of the APBH bus and the AHB-to-APB bridge functions' use of the APBH is mediated by internal arbitration logic. For contention between these two units, the DMA is favored and the AHB slave will report "not ready" via its HREADY output until the bridge transfer can complete. The arbiter tracks repeated lockouts and inverts the priority, guaranteeing the CPU every fourth transfer on the APB.

10.2 AHBH DMA

The DMA supports eight channels of DMA services, as shown in Table 10-1. The shared DMA resource allows each independent channel to follow a simple chained command list. Command chains are built up using the general structure, as shown in Figure 10-2.

Table 10-1. APBH DMA Channel Assignments

aPBH DMA Channel #	USAGE
0	Reserved
1	SSP1
2	SSP2
3	Reserved
4	NAND_DEVICE0
5	NAND_DEVICE1
6	NAND_DEVICE2
7	NAND_DEVICE3

A single command structure or channel command word specifies a number of operations to be performed by the DMA in support of a given device. Thus, the CPU can set up large units of work, chaining together many DMA channel command words, pass them off to the DMA, and have no further concern for the device until the DMA completion interrupt occurs. The goal here, as with the entire design of the i.MX23, is to have enough intelligence in the DMA and the devices to keep the interrupt frequency from any device below 1-kHz (arrival intervals longer than 1 ms).

Thus, a single command structure can issue 32-bit PIO write operations to key registers in the associated device using the same APB bus and controls that it uses to write DMA data bytes to the device. For example, this allows a chain of operations to be issued to the ATANAND controller to send NAND command bytes, address bytes, and data transfers where the command and address structure is completely under software control, but the administration of that transfer is handled autonomously by the DMA. Each DMA structure can have from 0 to 15 PIO words appended to it. The #PIOWORDs field, if non-zero, instructs the DMA engine to copy these words to the APB, beginning at PADDR = 0x0000 and incrementing its PADDR for each cycle.

The DMA master generates only normal read/write transfers to the APBH. It does *not* generate SCT set, clear, or toggle transfers.

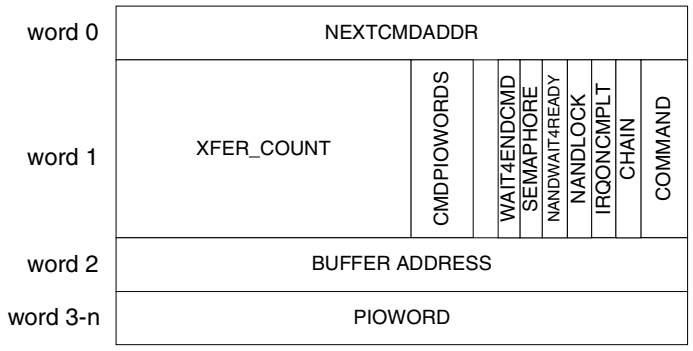


Figure 10-2. AHB-to-APBH Bridge DMA Channel Command Structure

Once any requested PIO words have been transferred to the peripheral, the DMA examines the two-bit command field in the channel command structure. Table 10-2 shows the four commands implemented by the DMA.

Table 10-2. APBH DMA Commands

DMA COMMAND	USAGE
00	NO_DMA_XFER. Perform any requested PIO word transfers, but terminate the command before any DMA transfer.
01	DMA_WRITE. Perform any requested PIO word transfers, then perform a DMA transfer from the peripheral for the specified number of bytes.
10	DMA_READ. Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.
11	DMA_SENSE. Perform any requested PIO word transfers, then perform a conditional branch to the next chained device. Follow the NEXTCMD_ADDR pointer if the peripheral sense is false. Follow the BUFFER_ADDRESS as a chain pointer if the peripheral sense line is true. This command becomes a no-operation for any channel other than a GPMI channel.

DMA_WRITE operations copy data bytes to system memory (on-chip RAM or SDRAM) from the associated peripheral. Each peripheral has a target PADDR value that it expects to receive DMA bytes. This association is synthesized in the DMA. The DMA_WRITE transfer uses the BUFFER_ADDDRESS word in the command structure to point to the beginning byte to write data from the peripheral.

DMA_READ operations copy data bytes to the APB peripheral from system memory. The DMA engine contains a shared byte aligner that aligns bytes from system memory to or from the peripherals. Peripherals always assume little-endian-aligned data arrives or departs on their 32-bit APB. The DMA_READ transfer uses the BUFFER_ADDRESS word in the command structure to point to the DMA data buffer to be read by the DMA_READ command.

The NO_DMA_XFER command is used to write PIO words to a device without performing any DMA data byte transfers. This command is useful in such applications as activating the NAND devices CHECKSTATUS operation. The check status command in the NAND peripheral reads a status byte from the NAND device, performs an XOR and MASK against an expected value supplied as part of the PIO transfer. Once the read check completes (see Section 10.3.1, “NAND Read Status Polling Example,”), the NO_DMA_XFER command completes. The result in the peripheral is that its PSENSE line is driven by the results of the comparison. The sense flip-flop is only updated by CHECKSTATUS for the device that is executed. At some future point, the chain contains a DMA command structure with the fourth and final command value, i.e., the DMA_SENSE command.

As each DMA command completes, it triggers the DMA to load the next DMA command structure in the chain. The normal flow list of DMA commands is found by following the NEXTCMD_ADDR pointer in the DMA command structure. The DMA_SENSE command uses the DMA buffer pointer word of the command structure in a slightly different way. Namely, it points to an alternate DMA command structure chain or list. The DMA_SENSE command examines the sense line of the associated peripheral. If the

sense line is “true,” then the DMA follows the standard list found whose next command is found from the pointer in the NEXTCMD_ADDR word of the command structure. If the sense line is “false,” then the DMA follows the alternate list whose next command is found from the pointer in the DMA Buffer Pointer word of the DMA_SENSE command structure (see Figure 10-3). The sense command ignores the CHAIN bit, so that both pointers must be valid when the DMA comes to sense command.

If the wait-for-end-command bit (WAIT4ENDCMD) is set in a command structure, then the DMA channel waits for the device to signal completion of a command by toggling the ENDCMCD signal before proceeding to load and execute the next command structure. The semaphore is decremented after the end command is seen.

A detailed bit-field view of the DMA command structure is shown in Table 10-3, which shows a field that specifies the number of bytes to be transferred by this DMA command. The transfer-count mechanism is duplicated in the associated peripheral, either as an implied or specified count in the peripheral.

Table 10-3. DMA Channel Command Word in System Memory

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
NEXT_COMMAND_ADDRESS																															
Number DMA Bytes to Transfer																Number PIO Words to Write								WAIT4ENDCMD	DECREMENT SEMAPHORE	NANDwAIT4READY	NANDLOCK	IRQ_COMPLETE	CHAIN	COMMAND			
DMA Buffer or Alternate CCW																															
Zero or More PIO Words to Write to the Associated Peripheral Starting at its Base Address on the APBH Bus																															

Figure 10-3 also shows the CHAIN bit in bit 2 of the second word of the command structure. This bit is set to 1 if the NEXT_COMMAND_ADDRESS contains a pointer to another DMA command structure. If a null pointer (0) is loaded into the NEXT_COMMAND_ADDRESS, it will not be detected by the DMA hardware. Only the CHAIN bit indicates whether a valid list exists beyond the current structure.

If the IRQ_COMPLETE bit is set in the command structure, then the last act of the DMA before loading the next command is to set the interrupt status bit corresponding to the current channel. The sticky interrupt request bit in the DMA CSR remains set until cleared by software. It can be used to interrupt the CPU.

The NAND_LOCK bit is monitored by the DMA channel arbiter. Once a NAND channel ([7:4]) succeeds in the arbiter with its NAND_LOCK bit set, then the arbiter will ignore the other three NAND channels until a command is completed in which the NAND_LOCK is not set. Notice that the semantic here is that the NAND_LOCK state is to limit scheduling of a non-locked DMA. A DMA channel can go

from unlocked to locked in the arbiter at the beginning of a command when the NAND_LOCK bit is set. When the last DMA command of an atomic sequence is completed, the lock should be removed. To accomplish this, the last command does not have the NAND_LOCK bit. It is still locked in the atomic state within the arbiter when the command starts, so that it is the only NAND command that can be executed. At the end, it drops from the atomic state within the arbiter.

The NAND_WAIT4READY bit also has a special use for DMA channels [7:4], i.e., the NAND device channels. The NAND device supplies a sample of the ready line for the NAND device. This ready value is used to hold off of a command with this bit set until the ready line is asserted to 1. Once the arbiter sees a command with a wait-for-ready set, it holds off that channel until ready is asserted.

will continue without waiting for the interrupt.

Receiving an IRQ for HALTONTERMINATE (HOT) is a new feature in the APBH/X DMA descriptor that allows certain peripheral block (e.g. GPMI, SSP, I2C) to signal to the DMA engine that an error has occurred. In prior chips, if a block stalled due to an error, the only practical way to discover this in s/w was via a timer of some sort, or to poll the block. Now, an HOT signal is sent from the peripheral to the DMA engine and causes an IRQ after terminating the DMA descriptor being executed. Note not all peripheral block support this termination feature.

Therefore, it is recommended that software use this signal as follows:

- Always set HALTONTERMINATE to 1 in a DMA descriptor. That way, if a peripheral signals HOT, the transfer will end, leaving the peripheral block and the DMA engine synchronized (but at the end of a command).
- When an IRQ from an APBH/X channel is received, and the IRQ is determined to be due to an error (as opposed to an IRQONCOMPLETE interrupt) the software should:
  - — Reset the channel.
  - — Determine the error from error reporting in the peripheral block, then manage the error in the peripheral that is attached to that channel in whatever appropriate way exists for that device (software recovery, device reset, block reset, etc).

Each channel has an eight-bit counting semaphore that controls whether it is in the run or idle state. When the semaphore is non-zero, the channel is ready to run and process commands and DMA transfers. Whenever a command finishes its DMA transfer, it checks the DECREMENT_SEMAPHORE bit. If set, it decrements the counting semaphore. If the semaphore goes to 0 as a result, then the channel enters the IDLE state and remains there until the semaphore is incremented by software. When the semaphore goes to non-zero and the channel is in its IDLE state, then it uses the value in the HW_APBH_CHn_NXTCMDAR (next command address register) to fetch a pointer to the next command to process. NOTE: This is a double indirect case. This method allows software to append to a running command list under the protection of the counting semaphore.

To start processing the first time, software creates the command list to be processed. It writes the address of the first command into the HW_APBH_CHn_NXTCMDAR register, and then writes a 1 to the counting semaphore in HW_APBH_CHn_SEMA. The DMA channel loads HW_APBH_CHn_CURCMDAR

register and then enters the normal state machine processing for the next command. When software writes a value to the counting semaphore, it is added to the semaphore count by hardware, protecting the case where both hardware and software are trying to change the semaphore on the same clock edge.

Software can examine the value of HW_APBH_CHn_CURCMDAR at any time to determine the location of the command structure currently being processed.

10.3 Implementation Examples

10.3.1 NAND Read Status Polling Example

Figure 10-3 shows a more complicated scenario. This subset of a NAND device workload shows that the first two command structures are used during the data-write phase of an NAND device write operation (CLE and ALE transfers omitted for clarity).

- After writing the data, one must wait until the NAND device status register indicates that the write charge has been transferred. This is built into the workload using a check status command in the NAND in a loop created from the next two DMA command structures.
- The NO_DMA_TRANSFER command is shown here performing the read check, followed by a DMA_SENSE command to branch the DMA command structure list, based on the status of a bit in the external NAND device.

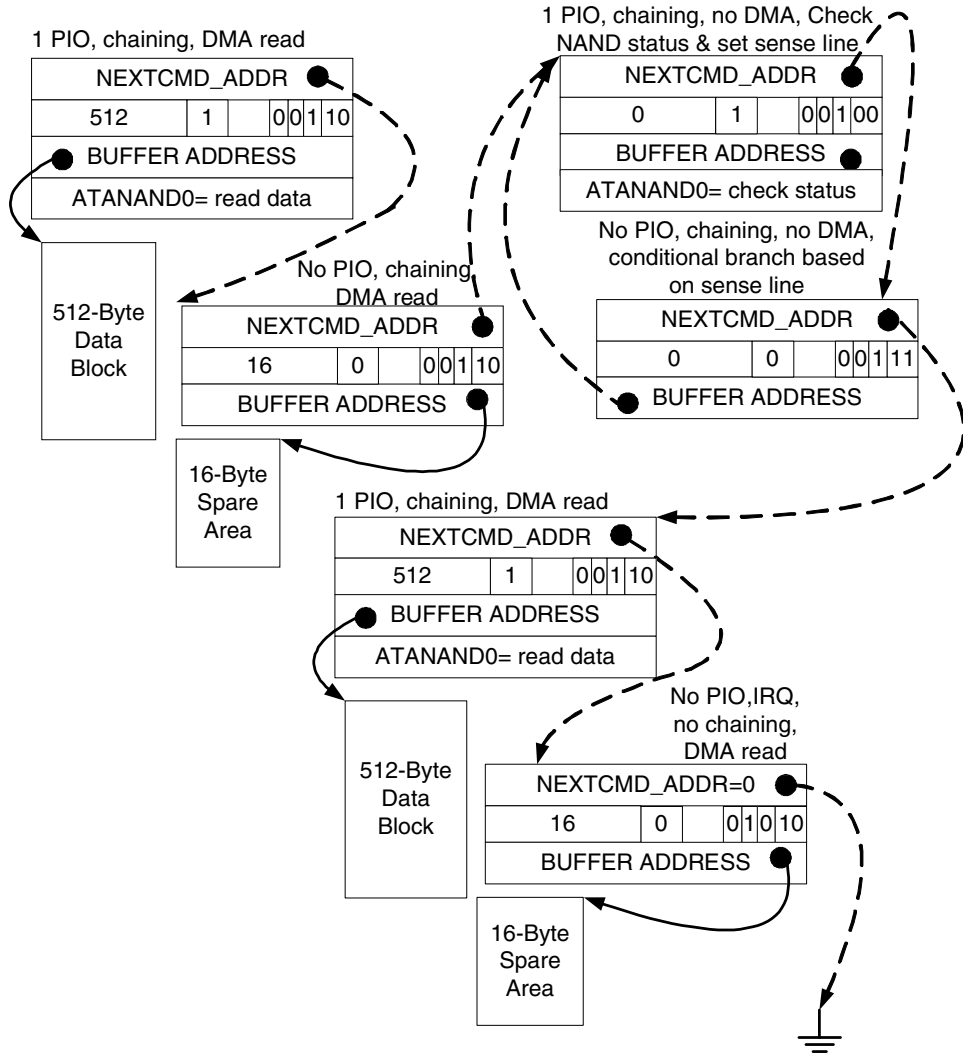


Figure 10-3. AHB-to-APBH Bridge DMA NAND Read Status Polling with DMA Sense Command

The example in Figure 10-3 shows the workload continuing immediately to the next NAND page transfer. However, one could perform a second sense operation to see if an error occurred after the write. One could then point the sense command alternate branch at a NO_DMA_XFER command with the interrupt bit set. If the CHAIN bit is not set on this failure branch, then the CPU is interrupted immediately, and the channel process is also immediately terminated in the presence of a workload-detected NAND error bit.

Note that each word of the three-word DMA command structure corresponds to a PIO register of the DMA that is accessible on the APBH bus. Normally, the DMA copies the next command structure onto these registers for processing at the start of each command by following the value of the pointer previously loaded into the NEXTCMD_ADDR register.

To start DMA processing for the first command, initialize the PIO registers of the desired channel, as follows:

- First, load the next command address register with a pointer to the first command to be loaded.
- Then, write a 1 to the counting semaphore register. This causes the DMA to schedule the targeted channel for DMA command structure load, as if it just finished its previous command.

10.4 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, "Correct Way to Soft Reset a Block," for additional information on using the SFTRST and CLKGATE bit fields.

10.5 Programmable Registers

This section describes the programmable registers of the AHB-to-APBH bridge block.

10.5.1 AHB to APBH Bridge Control and Status Register 0 Description

The APBH CTRL 0 provides overall control of the AHB to APBH bridge and DMA.

HW_APBH_CTRL0	0x000
HW_APBH_CTRL0_SET	0x004
HW_APBH_CTRL0_CLR	0x008
HW_APBH_CTRL0_TOG	0x00C

Table 10-4. HW_APBH_CTRL0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	AHB_BURST8_EN	APB_BURST4_EN	RSVD0				RESET_CHANNEL								CLKGATE_CHANNEL								FREEZE_CHANNEL							

Table 10-5. HW_APBH_CTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	Set this bit to zero to enable normal APBH DMA operation. Set this bit to one (default) to disable clocking with the APBH DMA and hold it in its reset (lowest power) state. This bit can be turned on and then off to reset the APBH DMA block to its default state.
30	**CLKGATE**	RW	0x1	This bit must be set to zero for normal operation. When set to one it gates off the clocks to the block.
29	**AHB_BURST8_EN**	RW	0x1	Set this bit to one (default) to enable AHB 8-beat burst. Set to zero to disable 8-beat burst on AHB interface.
28	**APB_BURST4_EN**	RW	0x0	Set this bit to one (default) to enable apb master do a 4 continous writes when a device request a burst dma. Set to zero will treat a burst dma request as 4 individual request.
27:24	**RSVD0**	RO	0x000000	Reserved, always set to zero.
23:16	**RESET_CHANNEL**	RW	0x0	Setting a bit in this field causes the DMA controller to take the corresponding channel through its reset state. The bit is reset after the channel resources are cleared. SSP1 = 0x02 SSP2 = 0x04 ATA = 0x10 NAND0 = 0x10 NAND1 = 0x20 NAND2 = 0x40 NAND3 = 0x80
15:8	**CLKGATE_CHANNEL**	RW	0x00	These bit must be set to zero for normal operation of each channel. When set to one they gate off the individual clocks to the channels. SSP1 = 0x02 SSP2 = 0x04 ATA = 0x10 NAND0 = 0x10 NAND1 = 0x20 NAND2 = 0x40 NAND3 = 0x80
7:0	**FREEZE_CHANNEL**	RW	0x0	Setting a bit in this field will freeze the DMA channel associated with it. This field is a direct input to the DMA channel arbiter. When frozen, the channel is denied access to the central DMA resources. SSP1 = 0x02 SSP2 = 0x04 ATA = 0x10 NAND0 = 0x10 NAND1 = 0x20 NAND2 = 0x40 NAND3 = 0x80

DESCRIPTION:

This register contains module softreset, clock gating, channel clock gating/freeze bits.

EXAMPLE:

```
Empty Example.
```

10.5.2 AHB to APBH Bridge Control and Status Register 1 Description

The APBH CTRL one provides overall control of the interrupts generated by the AHB to APBH DMA.

HW_APBH_CTRL1	0x010
HW_APBH_CTRL1_SET	0x014
HW_APBH_CTRL1_CLR	0x018
HW_APBH_CTRL1_TOG	0x01C

Table 10-6. HW_APBH_CTRL1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1								CH7_CMDCMPLT_IRQ_EN	CH6_CMDCMPLT_IRQ_EN	CH5_CMDCMPLT_IRQ_EN	CH4_CMDCMPLT_IRQ_EN	CH3_CMDCMPLT_IRQ_EN	CH2_CMDCMPLT_IRQ_EN	CH1_CMDCMPLT_IRQ_EN	CH0_CMDCMPLT_IRQ_EN	RSVD0								CH7_CMDCMPLT_IRQ	CH6_CMDCMPLT_IRQ	CH5_CMDCMPLT_IRQ	CH4_CMDCMPLT_IRQ	CH3_CMDCMPLT_IRQ	CH2_CMDCMPLT_IRQ	CH1_CMDCMPLT_IRQ	CH0_CMDCMPLT_IRQ

Table 10-7. HW_APBH_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD1**	RO	0x00000000	Reserved, always set to zero.
23	**CH7_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBH DMA channel 7.
22	**CH6_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBH DMA channel 6.
21	**CH5_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBH DMA channel 5.
20	**CH4_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBH DMA channel 4.
19	**CH3_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBH DMA channel 3.
18	**CH2_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBH DMA channel 2.
17	**CH1_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBH DMA channel 1.
16	**CH0_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBH DMA channel 0.
15:8	**RSVD0**	RO	0x00000000	Reserved, always set to zero.
7	**CH7_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBH DMA channel 7. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
6	**CH6_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBH DMA channel 6. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
5	**CH5_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBH DMA channel 5. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.

Table 10-7. HW_APBH_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
4	**CH4_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBH DMA channel 4. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
3	**CH3_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBH DMA channel 3. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
2	**CH2_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBH DMA channel 2. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
1	**CH1_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBH DMA channel 1. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
0	**CH0_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBH DMA channel 0. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.

DESCRIPTION:

This register contains the per channel interrupt status bits and the per channel interrupt enable bits. Each channel has a dedicated interrupt vector in the vectored interrupt controller.

EXAMPLE:

```
BF_WR(APBH_CTRL1, CH5_CMDCMPLT_IRQ, 0); // use bitfield write macro
BF_APBH_CTRL1.CH5_CMDCMPLT_IRQ = 0; // or, assign to register struct's bitfield
```

10.5.3 AHB to APBH Bridge Control and Status Register 2 Description

The APBH CTRL 2 provides channel error interrupts generated by the AHB to APBH DMA.

HW_APBH_CTRL2	0x020
HW_APBH_CTRL2_SET	0x024
HW_APBH_CTRL2_CLR	0x028
HW_APBH_CTRL2_TOG	0x02C

Table 10-8. HW_APBH_CTRL2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1								CH7_ERROR_STATUS	CH6_ERROR_STATUS	CH5_ERROR_STATUS	CH4_ERROR_STATUS	CH3_ERROR_STATUS	CH2_ERROR_STATUS	CH1_ERROR_STATUS	CH0_ERROR_STATUS	RSVD0								CH7_ERROR_IRQ	CH6_ERROR_IRQ	CH5_ERROR_IRQ	CH4_ERROR_IRQ	CH3_ERROR_IRQ	CH2_ERROR_IRQ	CH1_ERROR_IRQ	CH0_ERROR_IRQ

Table 10-9. HW_APBH_CTRL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD1**	RO	0x00000000	Reserved, always set to zero.
23	**CH7_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 7. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
22	**CH6_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 6. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
21	**CH5_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 5. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
20	**CH4_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 4. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
19	**CH3_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 3. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
18	**CH2_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 2. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
17	**CH1_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 1. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
16	**CH0_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 0. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
15:8	**RSVD0**	RO	0x00000000	Reserved, always set to zero.

Table 10-9. HW_APBH_CTRL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
7	**CH7_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 7. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
6	**CH6_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 6. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
5	**CH5_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 5. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
4	**CH4_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 4. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
3	**CH3_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 3. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
2	**CH2_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 2. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
1	**CH1_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 1. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
0	**CH0_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 0. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.

DESCRIPTION:

This register contains the per channel interrupt status bits and the per channel interrupt enable bits. Each channel has a dedicated interrupt vector in the vectored interrupt controller.

EXAMPLE:

```
BF_WR(APBH_CTRL1, CH5_CMDCMPLT_IRQ, 0); // use bitfield write macro
BF_APBH_CTRL1.CH5_CMDCMPLT_IRQ = 0; // or, assign to register struct's bitfield
```

10.5.4 AHB to APBH DMA Device Assignment Register Description

This register allows reassignment of the APBH device connected to the DMA Channels.

HW_APBH_DEVSEL 0x030

Table 10-10. HW_APBH_DEVSEL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CH7				CH6				CH5				CH4				CH3				CH2				CH1				CH0			

Table 10-11. HW_APBH_DEVSEL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**CH7**	RO	0x0	Reserved.
27:24	**CH6**	RO	0x0	Reserved.
23:20	**CH5**	RO	0x0	Reserved.
19:16	**CH4**	RO	0x0	Reserved.
15:12	**CH3**	RO	0x0	Reserved.
11:8	**CH2**	RO	0x0	Reserved.
7:4	**CH1**	RO	0x0	Reserved.
3:0	**CH0**	RO	0x0	Reserved.

DESCRIPTION:

This register contains channel mux sel bits. N/A for apbh bridge dma.

EXAMPLE:

```
Empty Example.
```

10.5.5 APBH DMA Channel 0 Current Command Address Register Description

The APBH DMA channel 0 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBH_CH0_CURCMDAR 0x040

Table 10-12. HW_APBH_CH0_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 10-13. HW_APBH_CH0_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for channel 0.

DESCRIPTION:

APBH DMA Channel 0 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty Example.
```

10.5.6 APBH DMA Channel 0 Next Command Address Register Description

The APBH DMA Channel 0 Next Command Address register contains the address of the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to 1 in the DMA command word to process command lists.

HW_APBH_CH0_NXTCMDAR 0x050

Table 10-14. HW_APBH_CH0_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 10-15. HW_APBH_CH0_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for channel 0.

DESCRIPTION:

APBH DMA Channel 0 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 0 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty Example.
```

10.5.7 APBH DMA Channel 0 Command Register Description

The APBH DMA Channel 0 command register specifies the DMA transaction to perform for the current command chain item.

HW_APBH_CH0_CMD 0x060

Table 10-16. HW_APBH_CH0_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1				HALTONTERMINATE	WAIT4ENDCMD	SEMAPHORE	NANDWAIT4READY	NANDLOCK	IRQONCMPLT	CHAIN	COMMAND		

Table 10-17. HW_APBH_CH0_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the DMA device. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the DMA device, starting with the base PIO address of the DMA device register and incrementing from there. Zero means transfer NO command words
11:9	**RSVD1**	RO	0x0	Reserved, always set to zero.
8	**HALTONTERMINATE**	RO	0x0	A value of one indicates that the channel will immeditately terminate the current descriptor and halt the DMA channel if a terminate signal is set. A value of 0 will still cause an immediate terminate of the channel if the terminate signal is set, but the channel will continue as if the count had been exhausted, meaning it will honor IRQONCMPLT, CHAIN, SEMAPHORE, and WAIT4ENDCMD.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBH device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5	**NANDWAIT4READY**	RO	0x0	A value of one indicates that the ATA/NAND DMA channel will will wait until the ATA/NAND device reports 'ready' before executing the command. It is ignored for non-ATA/NAND DMA channels.
4	**NANDLOCK**	RO	0x0	A value of one indicates that the ATA/NAND DMA channel will remain "locked" in the arbiter at the expense of other ATA/NAND DMA channels. It is ignored for non-ATA/NAND DMA channels.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause the interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.

Table 10-17. HW_APBH_CH0_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBH_CH0_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- Write transfers 10- Read transfer 11- SENSE NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes. DMA_SENSE = 0x3 Perform any requested PIO word transfers and then perform a conditional branch to the next chained device. Follow the NEXCMD_ADDR pointer if the perpheral sense is true. Follow the BUFFER_ADDRESS as a chain pointer if the peripheral sense line is false.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty Example.
```

10.5.8 APBH DMA Channel 0 Buffer Address Register Description

The APBH DMA Channel 0 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBH_CH0_BAR 0x070

Table 10-18. HW_APBH_CH0_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 10-19. HW_APBH_CH0_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty Example.
```

10.5.9 APBH DMA Channel 0 Semaphore Register Description

The APBH DMA Channel 0 semaphore register is used to synchronize the CPU instruction stream and the DMA chain processing state.

HW_APBH_CH0_SEMA 0x080

Table 10-20. HW_APBH_CH0_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 10-21. HW_APBH_CH0_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore that is used to synchronize between the program stream and and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore that has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty Example.
```

10.5.10 AHB to APBH DMA Channel 0 Debug Information Description

This register gives debug visibility into the APBH DMA Channel 0 state machine and controls.

HW_APBH_CH0_DEBUG1 0x090

Table 10-22. HW_APBH_CH0_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	SENSE	READY	LOCK	NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1															STATEMACHINE				

Table 10-23. HW_APBH_CH0_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27	**SENSE**	RO	0x0	This bit is reserved for this DMA Channel and always reads 0. For Channels 4-7, this bit reflects the current state of the GPMI Sense Signal sent from the APB GPMI Device
26	**READY**	RO	0x0	This bit is reserved for this DMA Channel and always reads 0. For Channels 4-7, this bit reflects the current state of the GPMI Ready Signal sent from the APB GPMI Device
25	**LOCK**	RO	0x0	This bit is reserved for this Channel and always reads 0. For Channels 4-7, this bit reflects the current state of the DMA Channel Lock for a GPMI Channel.
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.

Table 10-23. HW_APBH_CH0_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 0 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. TERMINATE = 0x14 When a terminate signal is set, the state machine enters this state until the current AHB transfer is completed. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. HALT_AFTER_TERM = 0x1D If HALTONTERMINATE is set and a terminate signal is set, the state machine enters this state and effectively halts. A channel reset is required to exit this state CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBH DMA Channel 0.

EXAMPLE:

```
Empty example.
```

10.5.11 AHB to APBH DMA Channel 0 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 0.

HW_APBH_CH0_DEBUG2 0x0A0

Table 10-24. HW_APBH_CH0_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 10-25. HW_APBH_CH0_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBH DMA Channel 0.

EXAMPLE:

```
Empty example.
```

10.5.12 APBH DMA Channel 1 Current Command Address Register Description

The APBH DMA Channel 1 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBH_CH1_CURCMDAR 0x0B0

Table 10-26. HW_APBH_CH1_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 10-27. HW_APBH_CH1_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for channel 1.

DESCRIPTION:

APBH DMA Channel 1 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
pCurCmd = (hw_apbh_chn_cmd_t *) HW_APBH_CHn_CURCMDAR_RD(1); // read the whole register, since there
is only one field
pCurCmd = (hw_apbh_chn_cmd_t *) BF_RDn(APBH_CHn_CURCMDAR, 1, CMD_ADDR); // or, use multi-register
bitfield read macro
pCurCmd = (hw_apbh_chn_cmd_t *) HW_APBH_CHn_CURCMDAR(1).CMD_ADDR; // or, assign from bitfield of
indexed register's struct
```

10.5.13 APBH DMA Channel 1 Next Command Address Register Description

The APBH DMA Channel 1 Next Command Address register contains the address of the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to 1 in the DMA command word to process command lists.

HW_APBH_CH1_NXTCMDAR 0x0C0

Table 10-28. HW_APBH_CH1_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 10-29. HW_APBH_CH1_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for channel 1.

DESCRIPTION:

APBH DMA Channel 1 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 1 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
HW_APBH_CHn_NXTCMDAR_WR(1, (reg32_t) pCommandTwoStructure); // write the entire register, since
there is only one field
BF_WRn(APBH_CHn_NXTCMDAR, 1, (reg32_t) pCommandTwoStructure); // or, use multi-register bitfield
write macro
HW_APBH_CHn_NXTCMDAR(1).CMD_ADDR = (reg32_t) pCommandTwoStructure; // or, assign to bitfield of
indexed register's struct
```

10.5.14 APBH DMA Channel 1 Command Register Description

The APBH DMA Channel 1 command register specifies the cycle to perform for the current command chain item.

HW_APBH_CH1_CMD 0x0D0

Table 10-30. HW_APBH_CH1_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1			HALTONTERMINATE	WAIT4ENDCMD	SEMAPHORE	NANDWAIT4READY	NANDLOCK	IRQONCMPLT	CHAIN	COMMAND	

Table 10-31. HW_APBH_CH1_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the SSP1 device. A value of 0 indicates a 64 KBytes trasnfer size.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the SSP1, starting with the base PIO address of the SSP1 control register and incrementing from there. Zero means transfer NO command words
11:9	**RSVD1**	RO	0x0	Reserved, always set to zero.
8	**HALTONTERMINATE**	RO	0x0	A value of one indicates that the channel will immeditately terminate the current descriptor and halt the DMA channel if a terminate signal is set. A value of 0 will still cause an immediate terminate of the channel if the terminate signal is set, but the channel will continue as if the count had been exhausted, meaning it will honor IRQONCMPLT, CHAIN, SEMAPHORE, and WAIT4ENDCMD.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBH device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5	**NANDWAIT4READY**	RO	0x0	A value of one indicates that the ATA/NAND DMA channel will will wait until the ATA/NAND device reports 'ready' before execute the command. It is ignored for non-ATA/NAND DMA channels.
4	**NANDLOCK**	RO	0x0	A value of one indicates that the ATA/NAND DMA channel will remain "locked" in the arbiter at the expense of other ATA/NAND DMA channels. It is ignored for non-ATA/NAND DMA channels.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause the interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.

Table 10-31. HW_APBH_CH1_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBH_CH1_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- Write transfers, i.e. data sent from the SSP1 (APB PIO Read) to the system memory (AHB master write). 10- Read transfer 11- SENSE NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes. DMA_SENSE = 0x3 Perform any requested PIO word transfers and then perform a conditional branch to the next chained device. Follow the NEXCMD_ADDR pointer if the perpheral sense is true. Follow the BUFFER_ADDRESS as a chain pointer if the peripheral sense line is false.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty Example.
```

10.5.15 APBH DMA Channel 1 Buffer Address Register Description

The APBH DMA Channel 1 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBH_CH1_BAR 0x0E0

Table 10-32. HW_APBH_CH1_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 10-33. HW_APBH_CH1_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
hw_apbh_chn_bar_t dma_data;
dma_data.ADDRESS = (reg32_t) pDataBuffer;
```

10.5.16 APBH DMA Channel 1 Semaphore Register Description

The APBH DMA Channel 1 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBH_CH1_SEMA 0x0F0

Table 10-34. HW_APBH_CH1_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 10-35. HW_APBH_CH1_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore that is used to synchronize between the program stream and and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore that has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
BF_WR(APBH_CHn_SEMA, 1, INCREMENT_SEMA, 2); // increment semaphore by two
current_sema = BF_RD(APBH_CHn_SEMA, 1, PHORE); // get instantaneous value
```

10.5.17 AHB to APBH DMA Channel 1 Debug Information Description

This register gives debug visibility into the APBH DMA Channel 1 state machine and controls.

HW_APBH_CH1_DEBUG1 0x100

Table 10-36. HW_APBH_CH1_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	SENSE	READY	LOCK	NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 10-37. HW_APBH_CH1_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27	**SENSE**	RO	0x0	This bit is reserved for this DMA Channel and always reads 0. For Channels 4-7, this bit reflects the current state of the GPMI Sense Signal sent from the APB GPMI Device
26	**READY**	RO	0x0	This bit is reserved for this DMA Channel and always reads 0. For Channels 4-7, this bit reflects the current state of the GPMI Ready Signal sent from the APB GPMI Device
25	**LOCK**	RO	0x0	This bit is reserved for this Channel and always reads 0. For Channels 4-7, this bit reflects the current state of the DMA Channel Lock for a GPMI Channel.
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflect the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflect the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflect the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflect the current state of the DMA Channel's Write FIFO Empty signal.

Table 10-37. HW_APBH_CH1_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
20	**WR_FIFO_FULL**	RO	0x0	This bit reflect the current state of the DMA Channel's Write FIFO Full signal.
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 1 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. TERMINATE = 0x14 When a terminate signal is set, the state machine enters this state until the current AHB transfer is completed. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. HALT_AFTER_TERM = 0x1D If HALTONTERMINATE is set and a terminate signal is set, the state machine enters this state and effectively halts. A channel reset is required to exit this state CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBH DMA Channel 1.

EXAMPLE:

```
Empty example.
```

10.5.18 AHB to APBH DMA Channel 1 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 1.

HW_APBH_CH1_DEBUG2 0x110

Table 10-38. HW_APBH_CH1_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 10-39. HW_APBH_CH1_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBH DMA Channel 1.

EXAMPLE:

```
Empty example.
```

10.5.19 APBH DMA Channel 2 Current Command Address Register Description

The APBH DMA Channel 2 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBH_CH2_CURCMDAR 0x120

Table 10-40. HW_APBH_CH2_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 10-41. HW_APBH_CH2_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for channel 2.

DESCRIPTION:

APBH DMA Channel 2 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

10.5.20 APBH DMA Channel 2 Next Command Address Register Description

The APBH DMA Channel 2 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBH_CH2_NXTCMDAR 0x130

Table 10-42. HW_APBH_CH2_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 10-43. HW_APBH_CH2_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for channel 2.

DESCRIPTION:

APBH DMA Channel 2 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 0 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty Example.
```

10.5.21 APBH DMA Channel 2 Command Register Description

The APBH DMA Channel 2 command register specifies the cycle to perform for the current command chain item.

HW_APBH_CH2_CMD 0x140

Table 10-44. HW_APBH_CH2_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1			HALTONTERMINATE	WAIT4ENDCMD	SEMAPHORE	NANDWAIT4READY	NANDLOCK	IRQONCMPLT	CHAIN	COMMAND	

Table 10-45. HW_APBH_CH2_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the SSP2 device. A value of 0 indicates a 64 KBytes transfer size.
15:12	**CMDWORDS**	RO	0x00	This field contains the number of command words to send to the SSP2, starting with the base PIO address of the SSP2 control register and incrementing from there. Zero means transfer NO command words
11:9	**RSVD1**	RO	0x0	Reserved, always set to zero.
8	**HALTONTERMINATE**	RO	0x0	A value of one indicates that the channel will immeditately terminate the current descriptor and halt the DMA channel if a terminate signal is set. A value of 0 will still cause an immediate terminate of the channel if the terminate signal is set, but the channel will continue as if the count had been exhausted, meaning it will honor IRQONCMPLT, CHAIN, SEMAPHORE, and WAIT4ENDCMD.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBH device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5	**NANDWAIT4READY**	RO	0x0	A value of one indicates that the ATA/NAND DMA channel will will wait until the ATA/NAND device reports 'ready' before execute the command. It is ignored for non-ATA/NAND DMA channels.
4	**NANDLOCK**	RO	0x0	A value of one indicates that the ATA/NAND DMA channel will remain "locked" in the arbiter at the expense of other ATA/NAND DMA channels. It is ignored for non-ATA/NAND DMA channels.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause the interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.

Table 10-45. HW_APBH_CH2_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBH_CH2_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- Write transfers, i.e. data sent from the APBH Device (APB PIO Read) to the system memory (AHB master write). 10- Read transfer 11- SENSE NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes. DMA_SENSE = 0x3 Perform any requested PIO word transfers and then perform a conditional branch to the next chained device. Follow the NEXCMD_ADDR pointer if the perpheral sense is true. Follow the BUFFER_ADDRESS as a chain pointer if the peripheral sense line is false.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

10.5.22 APBH DMA Channel 2 Buffer Address Register Description

The APBH DMA Channel 2 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBH_CH2_BAR 0x150

Table 10-46. HW_APBH_CH2_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 10-47. HW_APBH_CH2_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

10.5.23 APBH DMA Channel 2 Semaphore Register Description

The APBH DMA Channel 2 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBH_CH2_SEMA 0x160

Table 10-48. HW_APBH_CH2_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 10-49. HW_APBH_CH2_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore that is used to synchronize between the program stream and and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore that has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

10.5.24 AHB to APBH DMA Channel 2 Debug Information Description

This register gives debug visibility into the APBH DMA Channel 2 state machine and controls.

HW_APBH_CH2_DEBUG1 0x170

Table 10-50. HW_APBH_CH2_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	SENSE	READY	LOCK	NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1															STATEMACHINE				

Table 10-51. HW_APBH_CH2_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27	**SENSE**	RO	0x0	This bit is reserved for this DMA Channel and always reads 0. For Channels 4-7, this bit reflects the current state of the GPMI Sense Signal sent from the APB GPMI Device
26	**READY**	RO	0x0	This bit is reserved for this DMA Channel and always reads 0. For Channels 4-7, this bit reflects the current state of the GPMI Ready Signal sent from the APB GPMI Device
25	**LOCK**	RO	0x0	This bit is reserved for this Channel and always reads 0. For Channels 4-7, this bit reflects the current state of the DMA Channel Lock for a GPMI Channel.
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflect the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflect the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflect the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflect the current state of the DMA Channel's Write FIFO Empty signal.

Table 10-51. HW_APBH_CH2_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
20	**WR_FIFO_FULL**	RO	0x0	This bit reflect the current state of the DMA Channel's Write FIFO Full signal.
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 2 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. TERMINATE = 0x14 When a terminate signal is set, the state machine enters this state until the current AHB transfer is completed. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. HALT_AFTER_TERM = 0x1D If HALTONTERMINATE is set and a terminate signal is set, the state machine enters this state and effectively halts. A channel reset is required to exit this state CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBH DMA Channel 2.

EXAMPLE:

```
Empty example.
```

10.5.25 AHB to APBH DMA Channel 2 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 2.

HW_APBH_CH2_DEBUG2 0x180

Table 10-52. HW_APBH_CH2_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 10-53. HW_APBH_CH2_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBH DMA Channel 2.

EXAMPLE:

```
Empty example.
```

10.5.26 APBH DMA Channel 3 Current Command Address Register Description

The APBH DMA Channel 3 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBH_CH3_CURCMDAR 0x190

Table 10-54. HW_APBH_CH3_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 10-55. HW_APBH_CH3_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for channel 3.

DESCRIPTION:

APBH DMA Channel 3 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

10.5.27 APBH DMA Channel 3 Next Command Address Register Description

The APBH DMA Channel 3 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBH_CH3_NXTCMDAR 0x1A0

Table 10-56. HW_APBH_CH3_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 10-57. HW_APBH_CH3_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for channel 3.

DESCRIPTION:

APBH DMA Channel 3 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 3 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

10.5.28 APBH DMA Channel 3 Command Register Description

The APBH DMA Channel 3 command register specifies the cycle to perform for the current command chain item.

HW_APBH_CH3_CMD 0x1B0

Table 10-58. HW_APBH_CH3_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1			HALTONTERMINATE	WAIT4ENDCMD	SEMAPHORE	NANDWAIT4READY	NANDLOCK	IRQONCMPLT	CHAIN	COMMAND	

Table 10-59. HW_APBH_CH3_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the DMA device egister. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the DMA device, starting with the base PIO address of the DMA device and increment from there. Zero means transfer NO command words
11:9	**RSVD1**	RO	0x0	Reserved, always set to zero.
8	**HALTONTERMINATE**	RO	0x0	A value of one indicates that the channel will immeditately terminate the current descriptor and halt the DMA channel if a terminate signal is set. A value of 0 will still cause an immediate terminate of the channel if the terminate signal is set, but the channel will continue as if the count had been exhausted, meaning it will honor IRQONCMPLT, CHAIN, SEMAPHORE, and WAIT4ENDCMD.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBH device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5	**NANDWAIT4READY**	RO	0x0	A value of one indicates that the ATA/NAND DMA channel will will wait until the ATA/NAND device reports 'ready' before execute the command. It is ignored for non-ATA/NAND DMA channels.
4	**NANDLOCK**	RO	0x0	A value of one indicates that the ATA/NAND DMA channel will remain "locked" in the arbiter at the expense of other ATA/NAND DMA channels. It is ignored for non-ATA/NAND DMA channels.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause the interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.

Table 10-59. HW_APBH_CH3_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBH_CH3_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- Write transfers, i.e. data sent from the APBH device (APB PIO Read) to the system memory (AHB master write). 10- Read transfer 11- SENSE NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes. DMA_SENSE = 0x3 Perform any requested PIO word transfers and then perform a conditional branch to the next chained device. Follow the NEXCMD_ADDR pointer if the perpheral sense is true. Follow the BUFFER_ADDRESS as a chain pointer if the peripheral sense line is false.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

10.5.29 APBH DMA Channel 3 Buffer Address Register Description

The APBH DMA Channel 3 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBH_CH3_BAR 0x1C0

Table 10-60. HW_APBH_CH3_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 10-61. HW_APBH_CH3_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

10.5.30 APBH DMA Channel 3 Semaphore Register Description

The APBH DMA Channel 3 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBH_CH3_SEMA 0x1D0

Table 10-62. HW_APBH_CH3_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 10-63. HW_APBH_CH3_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore that is used to synchronize between the program stream and and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore that has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

10.5.31 AHB to APBH DMA Channel 3 Debug Information Description

This register gives debug visibility into the APBH DMA Channel 3 state machine and controls.

HW_APBH_CH3_DEBUG1 0x1E0

Table 10-64. HW_APBH_CH3_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	SENSE	READY	LOCK	NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1															STATEMACHINE				

Table 10-65. HW_APBH_CH3_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27	**SENSE**	RO	0x0	This bit is reserved for this DMA Channel and always reads 0. For Channels 4-7, this bit reflects the current state of the GPMI Sense Signal sent from the APB GPMI Device
26	**READY**	RO	0x0	This bit is reserved for this DMA Channel and always reads 0. For Channels 4-7, this bit reflects the current state of the GPMI Ready Signal sent from the APB GPMI Device
25	**LOCK**	RO	0x0	This bit is reserved for this Channel and always reads 0. For Channels 4-7, this bit reflects the current state of the DMA Channel Lock for a GPMI Channel.
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflect the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x0	This bit reflect the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflect the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x0	This bit reflect the current state of the DMA Channel's Write FIFO Empty signal.

Table 10-65. HW_APBH_CH3_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
20	**WR_FIFO_FULL**	RO	0x0	This bit reflect the current state of the DMA Channel's Write FIFO Full signal.
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 3 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. TERMINATE = 0x14 When a terminate signal is set, the state machine enters this state until the current AHB transfer is completed. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. HALT_AFTER_TERM = 0x1D If HALTONTERMINATE is set and a terminate signal is set, the state machine enters this state and effectively halts. A channel reset is required to exit this state CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBH DMA Channel 3.

EXAMPLE:

```
Empty example.
```

10.5.32 AHB to APBH DMA Channel 3 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 3.

HW_APBH_CH3_DEBUG2 0x1F0

Table 10-66. HW_APBH_CH3_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 10-67. HW_APBH_CH3_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBH DMA Channel 3.

EXAMPLE:

```
Empty example.
```

10.5.33 APBH DMA Channel 4 Current Command Address Register Description

The APBH DMA Channel 4 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBH_CH4_CURCMDAR 0x200

Table 10-68. HW_APBH_CH4_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 10-69. HW_APBH_CH4_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for channel 4.

DESCRIPTION:

APBH DMA Channel 4 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

10.5.34 APBH DMA Channel 4 Next Command Address Register Description

The APBH DMA Channel 4 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBH_CH4_NXTCMDAR 0x210

Table 10-70. HW_APBH_CH4_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 10-71. HW_APBH_CH4_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for channel 4.

DESCRIPTION:

APBH DMA Channel 4 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 4 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

10.5.35 APBH DMA Channel 4 Command Register Description

The APBH DMA Channel 4 command register specifies the cycle to perform for the current command chain item.

HW_APBH_CH4_CMD 0x220

Table 10-72. HW_APBH_CH4_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1			HALTONTERMINATE	WAIT4ENDCMD	SEMAPHORE	NANDWAIT4READY	NANDLOCK	IRQONCMPLT	CHAIN	COMMAND	

Table 10-73. HW_APBH_CH4_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the GPMI ATANAND_0 device HW_GPMI_DATA register. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the GPMI, starting with the base PIO address of the GPMI (HW_GPMI_CTRL0) and increment from there. Zero means transfer NO command words
11:9	**RSVD1**	RO	0x0	Reserved, always set to zero.
8	**HALTONTERMINATE**	RO	0x0	A value of one indicates that the channel will immeditately terminate the current descriptor and halt the DMA channel if a terminate signal is set. A value of 0 will still cause an immediate terminate of the channel if the terminate signal is set, but the channel will continue as if the count had been exhausted, meaning it will honor IRQONCMPLT, CHAIN, SEMAPHORE, and WAIT4ENDCMD.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBH device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5	**NANDWAIT4READY**	RO	0x0	A value of one indicates that the ATA/NAND DMA channel will will wait until the ATA/NAND device reports 'ready' before execute the command. It is ignored for non-ATA/NAND DMA channels.
4	**NANDLOCK**	RO	0x0	A value of one indicates that the ATA/NAND DMA channel will remain "locked" in the arbiter at the expense of other ATA/NAND DMA channels. It is ignored for non-ATA/NAND DMA channels.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause the interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.

Table 10-73. HW_APBH_CH4_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBH_CH4_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- Write transfers, i.e. data sent from the GPMI (APB PIO Read) to the system memory (AHB master write). 10- Read transfer 11- SENSE NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes. DMA_SENSE = 0x3 Perform any requested PIO word transfers and then perform a conditional branch to the next chained device. Follow the NEXCMD_ADDR pointer if the perpheral sense is true. Follow the BUFFER_ADDRESS as a chain pointer if the peripheral sense line is false.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

10.5.36 APBH DMA Channel 4 Buffer Address Register Description

The APBH DMA Channel 1 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBH_CH4_BAR 0x230

Table 10-74. HW_APBH_CH4_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 10-75. HW_APBH_CH4_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

10.5.37 APBH DMA Channel 4 Semaphore Register Description

The APBH DMA Channel 4 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBH_CH4_SEMA 0x240

Table 10-76. HW_APBH_CH4_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 10-77. HW_APBH_CH4_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore that is used to synchronize between the program stream and and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore that has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

10.5.38 AHB to APBH DMA Channel 4 Debug Information Description

This register gives debug visibility into the APBH DMA Channel 4 state machine and controls.

HW_APBH_CH4_DEBUG1 0x250

Table 10-78. HW_APBH_CH4_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	SENSE	READY	LOCK	NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1															STATEMACHINE				

Table 10-79. HW_APBH_CH4_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27	**SENSE**	RO	0x0	This bit reflects the current state of the GPMI Sense Signal sent from the APB GPMI Device
26	**READY**	RO	0x0	This bit reflects the current state of the GPMI Ready Signal sent from the APB GPMI Device
25	**LOCK**	RO	0x0	This bit reflects the current state of the DMA Channel Lock for a GPMI Channel.
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.

Table 10-79. HW_APBH_CH4_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 4 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. TERMINATE = 0x14 When a terminate signal is set, the state machine enters this state until the current AHB transfer is completed. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. HALT_AFTER_TERM = 0x1D If HALTONTERMINATE is set and a terminate signal is set, the state machine enters this state and effectively halts. A channel reset is required to exit this state CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts. WAIT_READY = 0x1F When the NAND Wait for Ready bit is set, the state machine enters this state until the GPMI device indicates that the external device is ready.

DESCRIPTION:

This register allows debug visibility of the APBH DMA Channel 4.

EXAMPLE:

```
Empty example.
```

10.5.39 AHB to APBH DMA Channel 4 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 4.

HW_APBH_CH4_DEBUG2 0x260

Table 10-80. HW_APBH_CH4_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 10-81. HW_APBH_CH4_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBH DMA Channel 4.

EXAMPLE:

```
Empty example.
```

10.5.40 APBH DMA Channel 5 Current Command Address Register Description

The APBH DMA Channel 5 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBH_CH5_CURCMDAR 0x270

Table 10-82. HW_APBH_CH5_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 10-83. HW_APBH_CH5_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for channel 5.

DESCRIPTION:

APBH DMA Channel 5 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

10.5.41 APBH DMA Channel 5 Next Command Address Register Description

The APBH DMA Channel 5 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBH_CH5_NXTCMDAR 0x280

Table 10-84. HW_APBH_CH5_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 10-85. HW_APBH_CH5_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for channel 5.

DESCRIPTION:

APBH DMA Channel 5 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 5 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

10.5.42 APBH DMA Channel 5 Command Register Description

The APBH DMA Channel 5 command register specifies the cycle to perform for the current command chain item.

HW_APBH_CH5_CMD 0x290

Table 10-86. HW_APBH_CH5_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1			HALTONTERMINATE	WAIT4ENDCMD	SEMAPHORE	NANDWAIT4READY	NANDLOCK	IRQONCMPLT	CHAIN	COMMAND	

Table 10-87. HW_APBH_CH5_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the GPMI ATANAND_1 device HW_GPMI_DATA register. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the GPMI, starting with the base PIO address of the GPMI (HW_GPMI_CTRL0) and increment from there. Zero means transfer NO command words
11:9	**RSVD1**	RO	0x0	Reserved, always set to zero.
8	**HALTONTERMINATE**	RO	0x0	A value of one indicates that the channel will immeditately terminate the current descriptor and halt the DMA channel if a terminate signal is set. A value of 0 will still cause an immediate terminate of the channel if the terminate signal is set, but the channel will continue as if the count had been exhausted, meaning it will honor IRQONCMPLT, CHAIN, SEMAPHORE, and WAIT4ENDCMD.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBH device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5	**NANDWAIT4READY**	RO	0x0	A value of one indicates that the ATA/NAND DMA channel will will wait until the ATA/NAND device reports 'ready' before execute the command. It is ignored for non-ATA/NAND DMA channels.
4	**NANDLOCK**	RO	0x0	A value of one indicates that the ATA/NAND DMA channel will remain "locked" in the arbiter at the expense of other ATA/NAND DMA channels. It is ignored for non-ATA/NAND DMA channels.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause the interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.

Table 10-87. HW_APBH_CH5_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBH_CH5_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- Write transfers, i.e. data sent from the GPMI (APB PIO Read) to the system memory (AHB master write). 10- Read transfer 11- SENSE NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes. DMA_SENSE = 0x3 Perform any requested PIO word transfers and then perform a conditional branch to the next chained device. Follow the NEXCMD_ADDR pointer if the perpheral sense is true. Follow the BUFFER_ADDRESS as a chain pointer if the peripheral sense line is false.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

10.5.43 APBH DMA Channel 5 Buffer Address Register Description

The APBH DMA Channel 5 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBH_CH5_BAR 0x2A0

Table 10-88. HW_APBH_CH5_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 10-89. HW_APBH_CH5_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

10.5.44 APBH DMA Channel 5 Semaphore Register Description

The APBH DMA Channel 5 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBH_CH5_SEMA 0x2B0

Table 10-90. HW_APBH_CH5_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 10-91. HW_APBH_CH5_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore that is used to synchronize between the program stream and and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore that has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

10.5.45 AHB to APBH DMA Channel 5 Debug Information Description

This register gives debug visibility into the APBH DMA Channel 5 state machine and controls.

HW_APBH_CH5_DEBUG1 0x2C0

Table 10-92. HW_APBH_CH5_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	SENSE	READY	LOCK	NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 10-93. HW_APBH_CH5_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27	**SENSE**	RO	0x0	This bit reflects the current state of the GPMI Sense Signal sent from the APB GPMI Device
26	**READY**	RO	0x0	This bit reflects the current state of the GPMI Ready Signal sent from the APB GPMI Device
25	**LOCK**	RO	0x0	This bit reflects the current state of the DMA Channel Lock for a GPMI Channel.
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.

Table 10-93. HW_APBH_CH5_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 5 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. TERMINATE = 0x14 When a terminate signal is set, the state machine enters this state until the current AHB transfer is completed. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. HALT_AFTER_TERM = 0x1D If HALTONTERMINATE is set and a terminate signal is set, the state machine enters this state and effectively halts. A channel reset is required to exit this state CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts. WAIT_READY = 0x1F When the NAND Wait for Ready bit is set, the state machine enters this state until the GPMI device indicates that the external device is ready.

DESCRIPTION:

This register allows debug visibility of the APBH DMA Channel 5.

EXAMPLE:

```
Empty example.
```

10.5.46 AHB to APBH DMA Channel 5 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 5.

HW_APBH_CH5_DEBUG2 0x2D0

Table 10-94. HW_APBH_CH5_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 10-95. HW_APBH_CH5_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBH DMA Channel 5.

EXAMPLE:

```
Empty example.
```

10.5.47 APBH DMA Channel 6 Current Command Address Register Description

The APBH DMA Channel 6 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBH_CH6_CURCMDAR 0x2E0

Table 10-96. HW_APBH_CH6_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 10-97. HW_APBH_CH6_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for channel 6.

DESCRIPTION:

APBH DMA Channel 6 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

10.5.48 APBH DMA Channel 6 Next Command Address Register Description

The APBH DMA Channel 6 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBH_CH6_NXTCMDAR 0x2F0

Table 10-98. HW_APBH_CH6_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 10-99. HW_APBH_CH6_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for channel 6.

DESCRIPTION:

APBH DMA Channel 6 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 6 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

10.5.49 APBH DMA Channel 6 Command Register Description

The APBH DMA Channel 6 command register specifies the cycle to perform for the current command chain item.

HW_APBH_CH6_CMD 0x300

Table 10-100. HW_APBH_CH6_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1			HALTONTERMINATE	WAIT4ENDCMD	SEMAPHORE	NANDWAIT4READY	NANDLOCK	IRQONCMPLT	CHAIN	COMMAND	

Table 10-101. HW_APBH_CH6_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the GPMI ATANAND_2 device HW_GPMI_DATA register. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the GPMI, starting with the base PIO address of the GPMI (HW_GPMI_CTRL0) and increment from there. Zero means transfer NO command words
11:9	**RSVD1**	RO	0x0	Reserved, always set to zero.
8	**HALTONTERMINATE**	RO	0x0	A value of one indicates that the channel will immeditately terminate the current descriptor and halt the DMA channel if a terminate signal is set. A value of 0 will still cause an immediate terminate of the channel if the terminate signal is set, but the channel will continue as if the count had been exhausted, meaning it will honor IRQONCMPLT, CHAIN, SEMAPHORE, and WAIT4ENDCMD.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBH device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5	**NANDWAIT4READY**	RO	0x0	A value of one indicates that the ATA/NAND DMA channel will will wait until the ATA/NAND device reports 'ready' before execute the command. It is ignored for non-ATA/NAND DMA channels.
4	**NANDLOCK**	RO	0x0	A value of one indicates that the ATA/NAND DMA channel will remain "locked" in the arbiter at the expense of other ATA/NAND DMA channels. It is ignored for non-ATA/NAND DMA channels.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause the interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.

Table 10-101. HW_APBH_CH6_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBH_CH6_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- Write transfers, i.e. data sent from the GPMI (APB PIO Read) to the system memory (AHB master write). 10- Read transfer 11- SENSE NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes. DMA_SENSE = 0x3 Perform any requested PIO word transfers and then perform a conditional branch to the next chained device. Follow the NEXCMD_ADDR pointer if the perpheral sense is true. Follow the BUFFER_ADDRESS as a chain pointer if the peripheral sense line is false.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

10.5.50 APBH DMA Channel 6 Buffer Address Register Description

The APBH DMA Channel 6 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBH_CH6_BAR 0x310

Table 10-102. HW_APBH_CH6_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 10-103. HW_APBH_CH6_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

10.5.51 APBH DMA Channel 6 Semaphore Register Description

The APBH DMA Channel 6 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBH_CH6_SEMA 0x320

Table 10-104. HW_APBH_CH6_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 10-105. HW_APBH_CH6_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore that is used to synchronize between the program stream and and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore that has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

10.5.52 AHB to APBH DMA Channel 6 Debug Information Description

This register gives debug visibility into the APBH DMA Channel 6 state machine and controls.

HW_APBH_CH6_DEBUG1 0x330

Table 10-106. HW_APBH_CH6_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	SENSE	READY	LOCK	NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1															STATEMACHINE				

Table 10-107. HW_APBH_CH6_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27	**SENSE**	RO	0x0	This bit reflects the current state of the GPMI Sense Signal sent from the APB GPMI Device
26	**READY**	RO	0x0	This bit reflects the current state of the GPMI Ready Signal sent from the APB GPMI Device
25	**LOCK**	RO	0x0	This bit reflects the current state of the DMA Channel Lock for a GPMI Channel.
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.

Table 10-107. HW_APBH_CH6_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 6 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. TERMINATE = 0x14 When a terminate signal is set, the state machine enters this state until the current AHB transfer is completed. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. HALT_AFTER_TERM = 0x1D If HALTONTERMINATE is set and a terminate signal is set, the state machine enters this state and effectively halts. A channel reset is required to exit this state CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts. WAIT_READY = 0x1F When the NAND Wait for Ready bit is set, the state machine enters this state until the GPMI device indicates that the external device is ready.

DESCRIPTION:

This register allows debug visibility of the APBH DMA Channel 6.

EXAMPLE:

```
Empty example.
```

10.5.53 AHB to APBH DMA Channel 6 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 6.

HW_APBH_CH6_DEBUG2 0x340

Table 10-108. HW_APBH_CH6_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 10-109. HW_APBH_CH6_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBH DMA Channel 6.

EXAMPLE:

```
Empty example.
```

10.5.54 APBH DMA Channel 7 Current Command Address Register Description

The APBH DMA Channel 7 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBH_CH7_CURCMDAR 0x350

Table 10-110. HW_APBH_CH7_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 10-111. HW_APBH_CH7_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for channel 7.

DESCRIPTION:

APBH DMA Channel 7 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

10.5.55 APBH DMA Channel 7 Next Command Address Register Description

The APBH DMA Channel 7 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBH_CH7_NXTCMDAR 0x360

Table 10-112. HW_APBH_CH7_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 10-113. HW_APBH_CH7_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for channel 7.

DESCRIPTION:

APBH DMA Channel 7 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 7 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

10.5.56 APBH DMA Channel 7 Command Register Description

The APBH DMA Channel 7 command register specifies the cycle to perform for the current command chain item.

HW_APBH_CH7_CMD 0x370

Table 10-114. HW_APBH_CH7_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1			HALTONTERMINATE	WAIT4ENDCMD	SEMAPHORE	NANDWAIT4READY	NANDLOCK	IRQONCMPLT	CHAIN	COMMAND	

Table 10-115. HW_APBH_CH7_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the GPMI ATANAND_3 device HW_GPMI_DATA register. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the GPMI, starting with the base PIO address of the GPMI (HW_GPMI_CTRL0) and increment from there. Zero means transfer NO command words
11:9	**RSVD1**	RO	0x0	Reserved, always set to zero.
8	**HALTONTERMINATE**	RO	0x0	A value of one indicates that the channel will immeditately terminate the current descriptor and halt the DMA channel if a terminate signal is set. A value of 0 will still cause an immediate terminate of the channel if the terminate signal is set, but the channel will continue as if the count had been exhausted, meaning it will honor IRQONCMPLT, CHAIN, SEMAPHORE, and WAIT4ENDCMD.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBH device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5	**NANDWAIT4READY**	RO	0x0	A value of one indicates that the ATA/NAND DMA channel will will wait until the ATA/NAND device reports 'ready' before execute the command. It is ignored for non-ATA/NAND DMA channels.
4	**NANDLOCK**	RO	0x0	A value of one indicates that the ATA/NAND DMA channel will remain "locked" in the arbiter at the expense of other ATA/NAND DMA channels. It is ignored for non-ATA/NAND DMA channels.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause the interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.

Table 10-115. HW_APBH_CH7_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBH_CH7_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- Write transfers, i.e. data sent from the GPMI (APB PIO Read) to the system memory (AHB master write). 10- Read transfer 11- SENSE NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes. DMA_SENSE = 0x3 Perform any requested PIO word transfers and then perform a conditional branch to the next chained device. Follow the NEXCMD_ADDR pointer if the perpheral sense is true. Follow the BUFFER_ADDRESS as a chain pointer if the peripheral sense line is false.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

10.5.57 APBH DMA Channel 7 Buffer Address Register Description

The APBH DMA Channel 7 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBH_CH7_BAR 0x380

Table 10-116. HW_APBH_CH7_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 10-117. HW_APBH_CH7_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

10.5.58 APBH DMA Channel 7 Semaphore Register Description

The APBH DMA Channel 7 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBH_CH7_SEMA 0x390

Table 10-118. HW_APBH_CH7_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 10-119. HW_APBH_CH7_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore that is used to synchronize between the program stream and and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore that has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

10.5.59 AHB to APBH DMA Channel 7 Debug Information Description

This register gives debug visibility into the APBH DMA Channel 7 state machine and controls.

HW_APBH_CH7_DEBUG1 0x3A0

Table 10-120. HW_APBH_CH7_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	SENSE	READY	LOCK	NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1															STATEMACHINE				

Table 10-121. HW_APBH_CH7_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27	**SENSE**	RO	0x0	This bit reflects the current state of the GPMI Sense Signal sent from the APB GPMI Device
26	**READY**	RO	0x0	This bit reflects the current state of the GPMI Ready Signal sent from the APB GPMI Device
25	**LOCK**	RO	0x0	This bit reflects the current state of the DMA Channel Lock for a GPMI Channel.
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.

Table 10-121. HW_APBH_CH7_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 7 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. TERMINATE = 0x14 When a terminate signal is set, the state machine enters this state until the current AHB transfer is completed. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. HALT_AFTER_TERM = 0x1D If HALTONTERMINATE is set and a terminate signal is set, the state machine enters this state and effectively halts. A channel reset is required to exit this state CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts. WAIT_READY = 0x1F When the NAND Wait for Ready bit is set, the state machine enters this state until the GPMI device indicates that the external device is ready.

DESCRIPTION:

This register allows debug visibility of the APBH DMA Channel 7.

EXAMPLE:

```
Empty example.
```

10.5.60 AHB to APBH DMA Channel 7 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 7.

HW_APBH_CH7_DEBUG2 0x3B0

Table 10-122. HW_APBH_CH7_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 10-123. HW_APBH_CH7_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBH DMA Channel 7.

EXAMPLE:

```
Empty example.
```

10.5.61 APBH Bridge Version Register Description

This register always returns a known read value for debug purposes it indicates the version of the block.

HW_APBH_VERSION 0x3F0

Table 10-124. HW_APBH_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 10-125. HW_APBH_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x02	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x00	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
if (HW_APBH_VERSION.B.MAJOR != 1)
Error();
```

APBH Block v2.0, Revision 1.57

Chapter 11
AHB-to-APBX Bridge with DMA

This chapter describes the AHB-to-APBX bridge on the i.MX23, along with its central DMA function and implementation examples. Programmable registers are described in Section 11.5, "Programmable Registers."

11.1 Overview

The AHB-to-APBX bridge provides the i.MX23 with an inexpensive peripheral attachment bus running on the AHB's XCLK. (The "X" in APBX denotes that the APB*X* runs on a crystal-derived clock, as compared to APB*H*, which is synchronous to HCLK.)

As shown in Figure 12-1, the AHB-to-APBX bridge includes the AHB-to-APB PIO bridge for memory-mapped I/O to the APB devices, as well a central DMA facility for devices on this bus and a vectored interrupt controller for the ARM926 core. Each one of the APB peripherals are documented in their own chapters elsewhere in this document.

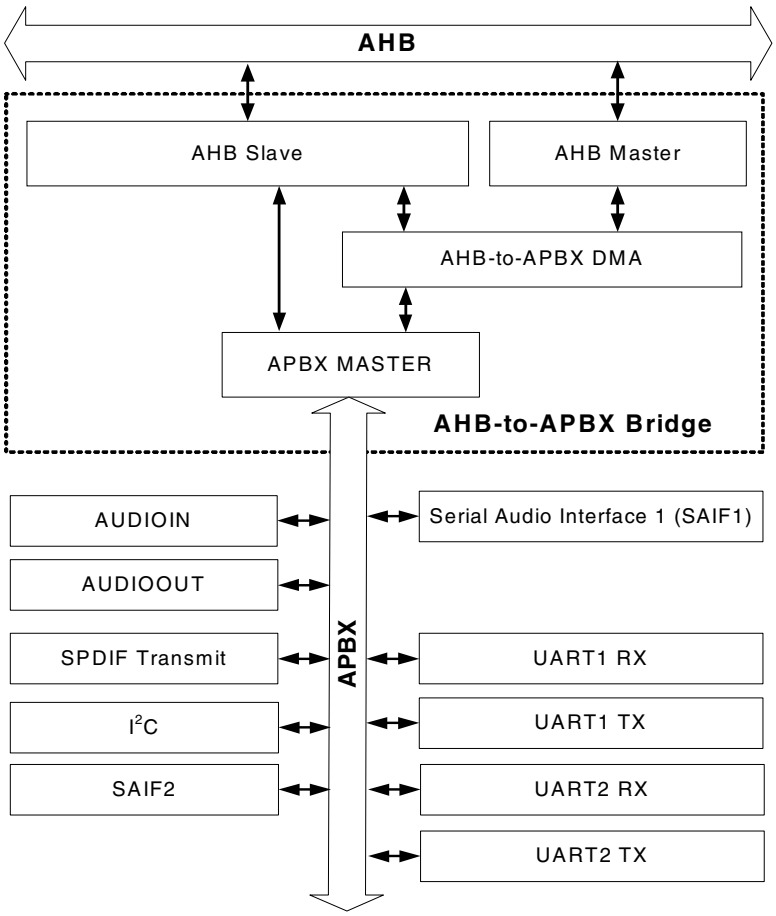


Figure 11-1. AHB-to-APBX Bridge DMA Block Diagram

The DMA controller uses the APBX bus to transfer read and write data to and from each peripheral. There is no separate DMA bus for these devices. Contention between the DMA's use of the APBX bus and AHB-to-APB bridge functions' use of the APBX is mediated by internal arbitration logic. For contention between these two units, the DMA is favored and the AHB slave will report not ready via its HREADY output until the bridge transfer completes. The arbiter tracks repeated lockouts and inverts the priority, so that the CPU is guaranteed every fourth transfer on the APB.

11.2 APBX DMA

The DMA supports sixteen channels of DMA services, as shown in Table 11-1. The shared DMA resource allows each independent channel to follow a simple chained command list. Command chains are built up using the DMA command structure, as shown in Figure 11-2.

Table 11-1. APBX DMA Channel Assignments

APBX DMA Channel #	USAGE
0	Audio ADCs
1	Audio DACs
2	SPDIF TX
3	I^2C
4	SAIF1
5	Reserved
6	UART1 RX, IrDA RX
7	UART1 TX, IrDA TX
8	UART2 RX
9	UART2 TX
10	SAIF2
11	Reserved
12	Reserved
13	Reserved
14	Reserved
15	Reserved

A single command structure or channel command word specifies a number of operations to be performed by the DMA in support of a given device. Thus, the CPU can set up large units of work, chaining together many DMA channel command words, pass them off to the DMA and have no further concern for the device until the DMA completion interrupt occurs. The i.MX23 is designed to have enough intelligence in the DMA and the devices to keep the interrupt frequency from any device below 1 kHz (arrival intervals longer than 1 ms).

Thus, a single command structure can issue 32-bit PIO write operations to key registers in the associated device using the same APB bus and controls it uses to write DMA data bytes to the device. For example, this allows a chain of operations to be issued to the serial audio interface to send command bytes, address bytes, and data transfers, where the command and address structure is completely under software control, but the administration of that transfer is handled autonomously by the DMA.

Each DMA structure can have from 0 to 15 PIO words appended to it. The #PIOWORDs field, if non-zero, instructs the DMA engine to copy these words to the APB beginning at PADDR = 0x0000 and incrementing its PADDR for each cycle. (Note that for APBX DMA Channel 6, which is the UART/IrDA RX channel, the first PIO word in the DMA command is CTRL0. However, for APBX DMA Channel 7, which is the UART/IrDA TX, the first PIO word in a DMA command is CTRL1.)

The HW_APBX_DEVSEL_CHx bit fields allow reassignment of the APBX device connected to DMA channels 2, 6, and 7. The DMA channel can be programmed to enable an alternate channel owner—for example, SAIF2 instead of SPDIF for Channel 2. Note that the CHx bit fields can be set only once after the chip is reset. To have the DMA channel provide DMA for another device, the chip must be reset and the HW_APBX_DEVSEL register must be reprogrammed. Whichever device is selected for the DMA

channel remains the selected device until the chip is reset and the DMA channel selected for another device.

The DMA master generates only normal read/write transfers to the APBX. It does *not* generate set, clear, or toggle SCT transfers.

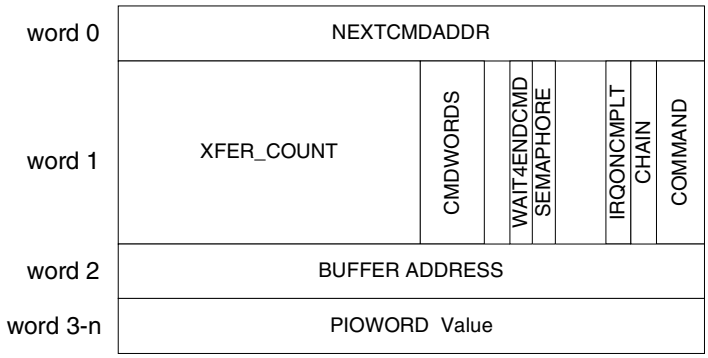


Figure 11-2. AHB-to-APBX Bridge DMA Channel Command Structure

Once any requested PIO words have been transferred to the peripheral, the DMA examines the two-bit command field in the channel command structure. Table 11-2 shows the four commands implemented by the DMA.

Table 11-2. APBX DMA Commands

DMA COMMAND	Usage
00	NO_DMA_XFER. Perform any requested PIO word transfers, but terminate the command before any DMA transfer.
01	DMA_WRITE. Perform any requested PIO word transfers, and then perform a DMA transfer from the peripheral for the specified number of bytes.
10	DMA_READ. Perform any requested PIO word transfers, and then perform a DMA transfer to the peripheral for the specified number of bytes.
11	Reserved

DMA_WRITE operations copy data bytes to system memory (on-chip RAM or SDRAM) from the associated peripheral. Each peripheral has a target PADDR value that it expects to receive DMA bytes. This association is synthesized in the DMA. The DMA_WRITE transfer uses the BUFFER_ADDDRESS word in the command structure to point to the beginning byte to write data from the peripheral.

DMA_READ operations copy data bytes to the APB peripheral from system memory. The DMA engine contains a shared byte aligner that aligns bytes from system memory to or from the peripherals. Peripherals always assume little-endian-aligned data arrives or departs on their 32-bit APB. The DMA_READ transfer uses the BUFFER_ADDRESS word in the command structure to point to the DMA data buffer to be read by the DMA_READ command.

The NO_DMA_XFER command is used to write PIO words to a device without performing any DMA data byte transfers.

As each DMA command completes, it triggers the DMA to load the next DMA command structure in the chain. The normal flow list of DMA commands is found by following the NEXTCMD_ADDR pointer in the DMA command structure. If the wait-for-end-command bit (WAIT4ENDCMD) is set in a command structure, then the DMA channel will wait for the device to signal completion of a command by toggling the apx_endcmcd signal before proceeding to load and execute the next command structure. The semaphore is decremented after the end command is seen.

A detailed bit-field view of the DMA command structure is shown in Table 11-3, which shows a field that specifies the number of bytes to be transferred by this DMA command. The transfer count mechanism is duplicated in the associated peripheral, either as an implied or specified count in the peripheral.

Table 11-3. DMA Channel Command Word in System Memory

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
NEXT_COMMAND_ADDRESS																															
Number DMA Bytes to Transfer																Number PIO Words to Write				Reserved				WAIT4ENDCMD	DECREMENT SEMAPHORE	Reserved			IRQ_COMPLETE	CHAIN	COMMAND		
DMA Buffer or Alternate CCW																															
Zero or More PIO Words to Write to the Associated Peripheral Starting at its Base Address on the APBX Bus																															

Figure 11-3 shows the CHAIN bit in bit 2 of the second word of the command structure. This bit is set to 1 if the NEXT_COMMAND_ADDRESS contains a pointer to another DMA command structure. If a null pointer (0) is loaded into the NEXT_COMMAND_ADDRESS, it will not be detected by the DMA hardware. Only the CHAIN bit indicates whether a valid list exists beyond the current structure.

If the IRQ_COMPLETE bit is set in the command structure, then the last act of the DMA before loading the next command is to set the interrupt status bit corresponding to the current channel. The sticky interrupt request bit in the DMA CSR remains set until cleared by software. It can be used to interrupt the CPU.

Each channel has an eight-bit counting semaphore that controls whether it is in the run or idle state. When the semaphore is non-zero, the channel is ready to run and process commands and DMA transfers. Whenever a command finishes its DMA transfer, it checks the DECREMENT_SEMAPHORE bit. If set, it decrements the counting semaphore. If the semaphore goes to 0 as a result, then the channel enters the IDLE state and remains there until the semaphore is incremented by software. When the semaphore goes to

non-zero and the channel is in its IDLE state, then it uses the value in the HW_APBX_CHn_NXTCMDAR (next command address register) to fetch a pointer to the next command to process. NOTE: this is a double indirect case. This method allows software to append to a running command list under the protection of the counting semaphore.

Receiving an IRQ for HALTONTERMINATE (HOT) is a new feature in the APBH/X DMA descriptor that allows certain peripheral block (e.g. GPMI, SSP, I2C) to signal to the DMA engine that an error has occurred. In prior chips, if a block stalled due to an error, the only practical way to discover this in s/w was via a timer of some sort, or to poll the block. Now, an HOT signal is sent from the peripheral to the DMA engine and causes an IRQ after terminating the DMA descriptor being executed. Note not all peripheral block support this termination feature.

Therefore, it is recommended that s/w use this signal as follows:

- Always set HALTONTERMINATE to 1 in a DMA descriptor. That way, if a peripheral signals HOT, the transfer will end, leaving the peripheral block and the DMA engine synchronized (but at the end of a command).
- When an IRQ from an APBH/X channel is received, and the IRQ is determined to be due to an error (as opposed to an IRQONCOMPLETE interrupt) the software should:

1. reset the channel, and
2. determine the error from error reporting in the peripheral block, then manage the error in the peripheral that is attached to that channel in whatever appropriate way exists for that device (software recovery, device reset, block reset, etc).

To start processing the first time, software creates the command list to be processed. It writes the address of the first command into the HW_APBX_CHn_NXTCMDAR register, and then writes a 1 to the counting semaphore in HW_APBX_CHn_SEMA. The DMA channel loads HW_APBX_CHn_CURCMDAR register and then enters the normal state machine processing for the next command. When software writes a value to the counting semaphore, it is added to the semaphore count by hardware, protecting the case where both hardware and software are trying to change the semaphore on the same clock edge.

Software can examine the value of HW_APBX_CHn_CURCMDAR at any time to determine the location of the command structure that is currently being processed.

11.3 DMA Chain Example

The example in Figure 11-3 shows how to bring the basic items together to make a simple DMA chain to read PCM samples and send them out the Audio Output (DAC) using one DMA channel. This example shows three command structures linked together using their normal command list pointers. The first command writes a single PIO word to the HW_AUDIOOUT_CTRL0 register with a new word count for the DAC. This first command also performs a 512 byte DMA_READ operation to read the data block bytes into the DAC. A second and a third DMA command structure also performs a DMA_READ operation to handle circular buffer style outputs. The completion of each command structure generates an interrupt request. In addition, each command structure decrements the semaphore. If the decompression software

has not provided a buffer in a timely fashion, then the DMA will stall. Without the decrement semaphore interlocking, then the DMA will continue to output a stream of samples. In this mode, it is up to software to use the interrupts to synchronize outputs so that underruns do not occur.

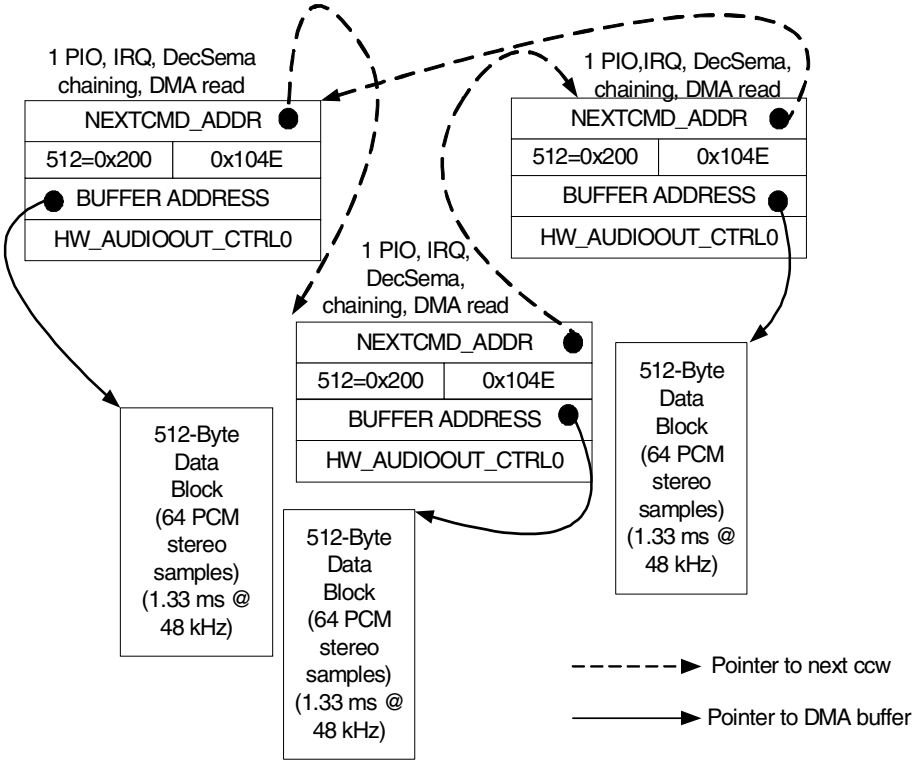


Figure 11-3. AHB-to-APBX Bridge DMA AUDIOOUT (DAC) Example Command Chain

Note that each word of the three-word DMA Command structure corresponds to a PIO register of the DMA that is accessible on the APBX bus. Normally, the DMA copies the next command structure onto these registers for processing at the start of each command by following the value of the pointer previously loaded into the NEXTCMD_ADDR register. In order to start DMA processing, for the first command, one must initialize the PIO registers of the desired channel. FIrst load the next command address register with a pointer to the first command to be loaded. Then write a 1 to the counting semaphore register. This causes the DMA to schedule the targeted channel for DMA command structure load, as if it just finished its previous command.

11.4 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, “Correct Way to Soft Reset a Block,” for additional information on using the SFTRST and CLKGATE bit fields.

11.5 Programmable Registers

This section describes the programmable registers of the AHB-to-APBX bridge block.

11.5.1 AHB to APBX Bridge Control Register 0 Description

The APBX CTRL 0 provides overall control and IRQ status of the AHB to APBX bridge and DMA.

HW_APBX_CTRL0	0x000
HW_APBX_CTRL0_SET	0x004
HW_APBX_CTRL0_CLR	0x008
HW_APBX_CTRL0_TOG	0x00C

Table 11-4. HW_APBX_CTRL0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	RSVD0																													

Table 11-5. HW_APBX_CTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	Set this bit to zero to enable normal APBX DMA operation. Set this bit to one (default) to disable clocking with the APBX DMA and hold it in its reset (lowest power) state. This bit can be turned on and then off to reset the APBX DMA block to its default state.
30	**CLKGATE**	RW	0x1	This bit must be set to zero for normal operation. When set to one it gates off the clocks to the block.
29:0	**RSVD0**	RO	0x000000	Reserved, always set to zero.

DESCRIPTION:

This register contains softreset, clock gating bits.

EXAMPLE:

```
No Example.
```

11.5.2 AHB to APBX Bridge Control Register 1 Description

The APBX CTRL 1 provides channel complete IRQ status of the AHB to APBX bridge and DMA.

HW_APBX_CTRL1	0x010
HW_APBX_CTRL1_SET	0x014
HW_APBX_CTRL1_CLR	0x018
HW_APBX_CTRL1_TOG	0x01C

Table 11-6. HW_APBX_CTRL1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CH15_CMDCMPLT_IRQ_EN	CH14_CMDCMPLT_IRQ_EN	CH13_CMDCMPLT_IRQ_EN	CH12_CMDCMPLT_IRQ_EN	CH11_CMDCMPLT_IRQ_EN	CH10_CMDCMPLT_IRQ_EN	CH9_CMDCMPLT_IRQ_EN	CH8_CMDCMPLT_IRQ_EN	CH7_CMDCMPLT_IRQ_EN	CH6_CMDCMPLT_IRQ_EN	CH5_CMDCMPLT_IRQ_EN	CH4_CMDCMPLT_IRQ_EN	CH3_CMDCMPLT_IRQ_EN	CH2_CMDCMPLT_IRQ_EN	CH1_CMDCMPLT_IRQ_EN	CH0_CMDCMPLT_IRQ_EN	CH15_CMDCMPLT_IRQ	CH14_CMDCMPLT_IRQ	CH13_CMDCMPLT_IRQ	CH12_CMDCMPLT_IRQ	CH11_CMDCMPLT_IRQ	CH10_CMDCMPLT_IRQ	CH9_CMDCMPLT_IRQ	CH8_CMDCMPLT_IRQ	CH7_CMDCMPLT_IRQ	CH6_CMDCMPLT_IRQ	CH5_CMDCMPLT_IRQ	CH4_CMDCMPLT_IRQ	CH3_CMDCMPLT_IRQ	CH2_CMDCMPLT_IRQ	CH1_CMDCMPLT_IRQ	CH0_CMDCMPLT_IRQ

Table 11-7. HW_APBX_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**CH15_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBX DMA Channel 15.
30	**CH14_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBX DMA Channel 14.
29	**CH13_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBX DMA Channel 13.
28	**CH12_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBX DMA Channel 12.
27	**CH11_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBX DMA Channel 11.
26	**CH10_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBX DMA Channel 10.
25	**CH9_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBX DMA Channel 9.
24	**CH8_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBX DMA Channel 8.
23	**CH7_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBX DMA Channel 7.
22	**CH6_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBX DMA Channel 6.
21	**CH5_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBX DMA Channel 5.
20	**CH4_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBX DMA Channel 4.
19	**CH3_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBX DMA Channel 3.
18	**CH2_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBX DMA Channel 2.
17	**CH1_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBX DMA Channel 1.
16	**CH0_CMDCMPLT_IRQ_EN**	RW	0x0	Setting this bit enables the generation of an interrupt request for APBX DMA Channel 0.

Table 11-7. HW_APBX_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15	**CH15_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBX DMA Channel 15. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
14	**CH14_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBX DMA Channel 14. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
13	**CH13_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBX DMA Channel 13. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
12	**CH12_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBX DMA Channel 12. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
11	**CH11_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBX DMA Channel 11. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
10	**CH10_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBX DMA Channel 10. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
9	**CH9_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBX DMA Channel 9. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
8	**CH8_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBX DMA Channel 8. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
7	**CH7_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBX DMA Channel 7. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
6	**CH6_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBX DMA Channel 6. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
5	**CH5_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBX DMA Channel 5. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
4	**CH4_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBX DMA Channel 4. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
3	**CH3_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBX DMA Channel 3. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.

Table 11-7. HW_APBX_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**CH2_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBX DMA Channel 2. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
1	**CH1_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBX DMA Channel 1. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.
0	**CH0_CMDCMPLT_IRQ**	RW	0x0	Interrupt request status bit for APBX DMA Channel 0. This sticky bit is set by DMA hardware and reset by software. It is ANDed with its corresponding enable bit to generate an interrupt.

DESCRIPTION:

This register contains the per channel interrupt status bits. Each channel has a dedicated interrupt vector in the vectored interrupt controller.

EXAMPLE:

```
BF_WR(APBX_CTRL1, CH5_CMDCMPLT_IRQ, 0); // use bitfield write macro
BF_APBX_CTRL1.CH5_CMDCMPLT_IRQ = 0; // or, assign to register struct's bitfield
```

11.5.3 AHB to APBX Bridge Control and Status Register 2 Description

The APBX CTRL 2 provides channel error interrupts generated by the AHB to APBX DMA.

HW_APBX_CTRL2	0x020
HW_APBX_CTRL2_SET	0x024
HW_APBX_CTRL2_CLR	0x028
HW_APBX_CTRL2_TOG	0x02C

Table 11-8. HW_APBX_CTRL2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CH15_ERROR_STATUS	CH14_ERROR_STATUS	CH13_ERROR_STATUS	CH12_ERROR_STATUS	CH11_ERROR_STATUS	CH10_ERROR_STATUS	CH9_ERROR_STATUS	CH8_ERROR_STATUS	CH7_ERROR_STATUS	CH6_ERROR_STATUS	CH5_ERROR_STATUS	CH4_ERROR_STATUS	CH3_ERROR_STATUS	CH2_ERROR_STATUS	CH1_ERROR_STATUS	CH0_ERROR_STATUS	CH15_ERROR_IRQ	CH14_ERROR_IRQ	CH13_ERROR_IRQ	CH12_ERROR_IRQ	CH11_ERROR_IRQ	CH10_ERROR_IRQ	CH9_ERROR_IRQ	CH8_ERROR_IRQ	CH7_ERROR_IRQ	CH6_ERROR_IRQ	CH5_ERROR_IRQ	CH4_ERROR_IRQ	CH3_ERROR_IRQ	CH2_ERROR_IRQ	CH1_ERROR_IRQ	CH0_ERROR_IRQ

Table 11-9. HW_APBX_CTRL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**CH15_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 15. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
30	**CH14_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 14. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
29	**CH13_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 13. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
28	**CH12_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 12. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
27	**CH11_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 11. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
26	**CH10_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 10. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
25	**CH9_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 9. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
24	**CH8_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 8. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.

Table 11-9. HW_APBX_CTRL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
23	**CH7_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 7. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
22	**CH6_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 6. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
21	**CH5_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 5. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
20	**CH4_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 4. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
19	**CH3_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 3. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
18	**CH2_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 2. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
17	**CH1_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 1. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
16	**CH0_ERROR_STATUS**	RO	0x0	Error status bit for APBX DMA Channel 0. Valid when corresponding Error IRQ is set. 1 - AHB bus error 0 - channel early termination. TERMINATION = 0x0 An early termination from the device causes error IRQ. BUS_ERROR = 0x1 An AHB bus error causes error IRQ.
15	**CH15_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 15. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.

Table 11-9. HW_APBX_CTRL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
14	**CH14_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 14. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
13	**CH13_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 13. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
12	**CH12_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 12. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
11	**CH11_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 11. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
10	**CH10_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 10. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
9	**CH9_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 9. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
8	**CH8_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 8. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
7	**CH7_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 7. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
6	**CH6_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 6. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
5	**CH5_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 5. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
4	**CH4_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 4. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
3	**CH3_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 3. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
2	**CH2_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 2. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.

Table 11-9. HW_APBX_CTRL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
1	**CH1_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 1. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.
0	**CH0_ERROR_IRQ**	RW	0x0	Error interrupt status bit for APBX DMA Channel 0. This sticky bit is set by DMA hardware and reset by software. It is ORed with the corresponding cmdcmplt irq to generate an irq to ARM.

DESCRIPTION:

This register contains the per channel bus error interrupt status bits and the per channel completion interrupt enable bits. Each channel has a dedicated interrupt vector in the vectored interrupt controller.

EXAMPLE:

```
Exmpty Example
```

11.5.4 AHB to APBX Bridge Channel Register Description

The APBX CHANNEL CTRL provides reset/freeze control of each DMA channel.

HW_APBX_CHANNEL_CTRL	0x030
HW_APBX_CHANNEL_CTRL_SET	0x034
HW_APBX_CHANNEL_CTRL_CLR	0x038
HW_APBX_CHANNEL_CTRL_TOG	0x03C

Table 11-10. HW_APBX_CHANNEL_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RESET_CHANNEL																FREEZE_CHANNEL															

Table 11-11. HW_APBX_CHANNEL_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RESET_CHANNEL**	RW	0x0	Setting a bit in this field causes the DMA controller to take the corresponding channel through its reset state. The bit is reset after the channel resources are cleared. AUDIOIN = 0x0001 AUDIOOUT = 0x0002 SPDIF_TX = 0x0004 I2C = 0x0008 SAIF1 = 0x0010 UART0_RX = 0x0040 UART0_TX = 0x0080 UART1_RX = 0x0100 UART1_TX = 0x0200 SAIF2 = 0x0400
15:0	**FREEZE_CHANNEL**	RW	0x0	Setting a bit in this field will freeze the DMA channel associated with it. This field is a direct input to the DMA channel arbiter. When frozen, the channel is deined access to the central DMA resources. AUDIOIN = 0x0001 AUDIOOUT = 0x0002 SPDIF_TX = 0x0004 I2C = 0x0008 SAIF1 = 0x0010 UART0_RX = 0x0040 UART0_TX = 0x0080 UART1_RX = 0x0100 UART1_TX = 0x0200 SAIF2 = 0x0400

DESCRIPTION:

This register contains individual channel reset/freeze bits.

EXAMPLE:

```
Empty Example.
```

11.5.5 AHB to APBX DMA Device Assignment Register Description

This register allows reassignment of the APBX device connected to the DMA Channels.

HW_APBX_DEVSEL 0x040

Table 11-12. HW_APBX_DEVSEL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CH15		CH14		CH13		CH12		CH11		CH10		CH9		CH8		CH7		CH6		CH5		CH4		CH3		CH2		CH1		CH0	

Table 11-13. HW_APBX_DEVSEL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**CH15**	RO	0x0	Reserved.
29:28	**CH14**	RO	0x0	Reserved.
27:26	**CH13**	RO	0x0	Reserved.
25:24	**CH12**	RO	0x0	Reserved.
23:22	**CH11**	RO	0x0	Reserved.

Table 11-13. HW_APBX_DEVSEL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
21:20	**CH10**	RO	0x0	Reserved.
19:18	**CH9**	RO	0x0	Reserved.
17:16	**CH8**	RO	0x0	Reserved.
15:14	**CH7**	RW	0x0	Reserved.
13:12	**CH6**	RW	0x0	Reserved.
11:10	**CH5**	RO	0x0	Reserved.
9:8	**CH4**	RO	0x0	Reserved.
7:6	**CH3**	RO	0x0	Reserved.
5:4	**CH2**	RO	0x0	Reserved.
3:2	**CH1**	RO	0x0	Reserved.
1:0	**CH0**	RO	0x0	Reserved.

DESCRIPTION:

This register provides a mechanism for assigning the device which is attached to DMA channels 6 and 7.

EXAMPLE:

```
Empty Example.
```

11.5.6 APBX DMA Channel 0 Current Command Address Register Description

The APBX DMA Channel 0 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBX_CH0_CURCMDAR 0x100

Table 11-14. HW_APBX_CH0_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-15. HW_APBX_CH0_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for channel 0.

DESCRIPTION:

APBX DMA Channel 0 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
pCurCmd = (hw_apbh_chn_cmd_t *) HW_APBX_CHn_CURCMDAR_RD(0); // read the whole register, since there is only one field
pCurCmd = (hw_apbh_chn_cmd_t *) BF_RDn(APBX_CHn_CURCMDAR, 0, CMD_ADDR); // or, use multi-register bitfield read macro
pCurCmd = (hw_apbh_chn_cmd_t *) HW_APBX_CHn_CURCMDAR(0).CMD_ADDR; // or, assign from bitfield of indexed register's struct
```

11.5.7 APBX DMA Channel 0 Next Command Address Register Description

The APBX DMA Channel 0 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBX_CH0_NXTCMDAR 0x110

Table 11-16. HW_APBX_CH0_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-17. HW_APBX_CH0_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for channel 0.

DESCRIPTION:

APBX DMA Channel 0 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 0 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
HW_APBX_CHn_NXTCMDAR_WR(0, (reg32_t) pCommandTwoStructure); // write the entire register, since there is only one field
BF_WRn(APBX_CHn_NXTCMDAR, 0, (reg32_t) pCommandTwoStructure); // or, use multi-register bitfield write macro
HW_APBX_CHn_NXTCMDAR(0).CMD_ADDR = (reg32_t) pCommandTwoStructure; // or, assign to bitfield of indexed register's struct
```

11.5.8 APBX DMA Channel 0 Command Register Description

The APBX DMA Channel 0 command register specifies the DMA transaction to perform for the current command chain item.

HW_APBX_CH0_CMD 0x120

Table 11-18. HW_APBX_CH0_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1				WAIT4ENDCMD	SEMAPHORE	RSVD0		IRQONCMPLT	CHAIN	COMMAND	

Table 11-19. HW_APBX_CH0_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the ADC device HW_AUDIOIN_DATA. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the ADC, starting with the base PIO address of the ADC (HW_AUDIOIN_CTRL) and increment from there. Zero means transfer NO command words
11:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the ABPX device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5:4	**RSVD0**	RO	0x0	Reserved, always set to zero.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause its interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBX_CH0_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- write transfers, i.e. data sent from the APBX device (APB PIO Read) to the system memory (AHB master write). 10- read transfer 11- reserved NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
hw_apbh_chn_cmd_t dma_cmd;
dma_cmd.XFER_COUNT = 512; // transfer 512 bytes
```

```
dma_cmd.COMMAND = BV_APBX_CHn_CMD_COMMAND__DMA_WRITE; // transfer to system memory from peripheral
device
dma_cmd.CHAIN = 1; // chain an additional command structure on to the list
dma_cmd.IRQONCMPLT = 1; // generate an interrupt on completion of this command structure
```

11.5.9 APBX DMA Channel 0 Buffer Address Register Description

The APBX DMA Channel 0 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBX_CH0_BAR 0x130

Table 11-20. HW_APBX_CH0_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 11-21. HW_APBX_CH0_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
hw_apbh_chn_bar_t dma_data;
dma_data.ADDRESS = (reg32_t) pDataBuffer;
```

11.5.10 APBX DMA Channel 0 Semaphore Register Description

The APBX DMA Channel 0 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBX_CH0_SEMA 0x140

Table 11-22. HW_APBX_CH0_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 11-23. HW_APBX_CH0_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore used to synchronize between the program stream and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore which has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
BF_WR(APBX_CHn_SEMA, 0, INCREMENT_SEMA, 2); // increment semaphore by two
current_sema = BF_RD(APBX_CHn_SEMA, 0, PHORE); // get instantaneous value
```

11.5.11 AHB to APBX DMA Channel 0 Debug Information Description

This register gives debug visibility into the APBX DMA Channel 0 state machine and controls.

HW_APBX_CH0_DEBUG1 0x150

Table 11-24. HW_APBX_CH0_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	RSVD2			NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 11-25. HW_APBX_CH0_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27:25	**RSVD2**	RO	0x0	Reserved
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 0 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 0.

EXAMPLE:

```
Empty example.
```

11.5.12 AHB to APBX DMA Channel 0 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 0.

HW_APBX_CH0_DEBUG2 0x160

Table 11-26. HW_APBX_CH0_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 11-27. HW_APBX_CH0_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 0.

EXAMPLE:

```
Empty example.
```

11.5.13 APBX DMA Channel 1 Current Command Address Register Description

The APBX DMA Channel 1 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBX_CH1_CURCMDAR 0x170

Table 11-28. HW_APBX_CH1_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-29. HW_APBX_CH1_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for channel 1.

DESCRIPTION:

APBX DMA Channel 1 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
pCurCmd = (hw_apbh_chn_cmd_t *) HW_APBX_CHn_CURCMDAR_RD(1); // read the whole register, since there is only one field
pCurCmd = (hw_apbh_chn_cmd_t *) BF_RDn(APBX_CHn_CURCMDAR, 1, CMD_ADDR); // or, use multi-register bitfield read macro
pCurCmd = (hw_apbh_chn_cmd_t *) HW_APBX_CHn_CURCMDAR(1).CMD_ADDR; // or, assign from bitfield of indexed register's struct
```

11.5.14 APBX DMA Channel 1 Next Command Address Register Description

The APBX DMA Channel 1 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBX_CH1_NXTCMDAR 0x180

Table 11-30. HW_APBX_CH1_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-31. HW_APBX_CH1_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for channel 1.

DESCRIPTION:

APBX DMA Channel 1 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 1 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
HW_APBX_CHn_NXTCMDAR_WR(1, (reg32_t) pCommandTwoStructure); // write the entire register, since there is only one field
BF_WRn(APBX_CHn_NXTCMDAR, 1, (reg32_t) pCommandTwoStructure); // or, use multi-register bitfield write macro
HW_APBX_CHn_NXTCMDAR(1).CMD_ADDR = (reg32_t) pCommandTwoStructure; // or, assign to bitfield of indexed register's struct
```

11.5.15 APBX DMA Channel 1 Command Register Description

The APBX DMA Channel 1 command register specifies the cycle to perform for the current command chain item.

HW_APBX_CH1_CMD 0x190

Table 11-32. HW_APBX_CH1_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1				WAIT4ENDCMD	SEMAPHORE	RSVD0		IRQONCMPLT	CHAIN	COMMAND	

Table 11-33. HW_APBX_CH1_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the DAC device HW_AUDIOOUT_DATA. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the DAC, starting with the base PIO address of the DAC (HW_AUDIOOUT_CTRL) and increment from there. Zero means transfer NO command words
11:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBX device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5:4	**RSVD0**	RO	0x0	Reserved, always set to zero.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause its interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.

Table 11-33. HW_APBX_CH1_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBX_CH1_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- write transfers, i.e. data sent from the APBX device (APB PIO Read) to the system memory (AHB master write). 10- read transfer 11- reserved NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty Example.
```

11.5.16 APBX DMA Channel 1 Buffer Address Register Description

The APBX DMA Channel 1 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBX_CH1_BAR 0x1A0

Table 11-34. HW_APBX_CH1_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 11-35. HW_APBX_CH1_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
hw_apbh_chn_bar_t dma_data;
dma_data.ADDRESS = (reg32_t) pDataBuffer;
```

11.5.17 APBX DMA Channel 1 Semaphore Register Description

The APBX DMA Channel 1 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBX_CH1_SEMA 0x1B0

Table 11-36. HW_APBX_CH1_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 11-37. HW_APBX_CH1_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore used to synchronize between the program stream and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore which has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
BF_WR(APBX_CHn_SEMA, 1, INCREMENT_SEMA, 2); // increment semaphore by two
current_sema = BF_RD(APBX_CHn_SEMA, 1, PHORE); // get instantaneous value
```

11.5.18 AHB to APBX DMA Channel 1 Debug Information Description

This register gives debug visibility into the APBX DMA Channel 1 state machine and controls.

HW_APBX_CH1_DEBUG1 0x1C0

Table 11-38. HW_APBX_CH1_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	RSVD2			NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 11-39. HW_APBX_CH1_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27:25	**RSVD2**	RO	0x0	Reserved
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.

Table 11-39. HW_APBX_CH1_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 1 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 1.

EXAMPLE:

```
Empty example.
```

11.5.19 AHB to APBX DMA Channel 1 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 1.

HW_APBX_CH1_DEBUG2 0x1D0

Table 11-40. HW_APBX_CH1_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 11-41. HW_APBX_CH1_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 1.

EXAMPLE:

```
Empty example.
```

11.5.20 APBX DMA Channel 2 Current Command Address Register Description

The APBX DMA Channel 2 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBX_CH2_CURCMDAR 0x1E0

Table 11-42. HW_APBX_CH2_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-43. HW_APBX_CH2_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for channel 2.

DESCRIPTION:

APBX DMA Channel 2 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

11.5.21 APBX DMA Channel 2 Next Command Address Register Description

The APBX DMA Channel 2 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBX_CH2_NXTCMDAR 0x1F0

Table 11-44. HW_APBX_CH2_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-45. HW_APBX_CH2_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for channel 2.

DESCRIPTION:

APBX DMA Channel 2 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 0 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty Example.
```

11.5.22 APBX DMA Channel 2 Command Register Description

The APBX DMA Channel 2 command register specifies the cycle to perform for the current command chain item.

HW_APBX_CH2_CMD 0x200

Table 11-46. HW_APBX_CH2_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1				WAIT4ENDCMD	SEMAPHORE	RSVD0		IRQONCMPLT	CHAIN	COMMAND	

Table 11-47. HW_APBX_CH2_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the SPDIF or SAIF1 device. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the SPDIF, starting with the base PIO address of the SPDIF or SAIF1 and incrementing from there. Zero means transfer NO command words
11:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBX device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5:4	**RSVD0**	RO	0x0	Reserved, always set to zero.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause its interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBX_CH2_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- write transfers, i.e. data sent from the APBX device (APB PIO Read) to the system memory (AHB master write). 10- read transfer 11- reserved NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

11.5.23 APBX DMA Channel 2 Buffer Address Register Description

The APBX DMA Channel 2 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBX_CH2_BAR 0x210

Table 11-48. HW_APBX_CH2_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 11-49. HW_APBX_CH2_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

11.5.24 APBX DMA Channel 2 Semaphore Register Description

The APBX DMA Channel 2 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBX_CH2_SEMA 0x220

Table 11-50. HW_APBX_CH2_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 11-51. HW_APBX_CH2_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore used to synchronize between the program stream and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore which has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

11.5.25 AHB to APBX DMA Channel 2 Debug Information Description

This register gives debug visibility into the APBX DMA Channel 2 state machine and controls.

HW_APBX_CH2_DEBUG1 0x230

Table 11-52. HW_APBX_CH2_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	RSVD2			NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 11-53. HW_APBX_CH2_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device

Table 11-53. HW_APBX_CH2_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27:25	**RSVD2**	RO	0x0	Reserved
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 2 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 2.

EXAMPLE:

```
Empty example.
```

11.5.26 AHB to APBX DMA Channel 2 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 2.

HW_APBX_CH2_DEBUG2 0x240

Table 11-54. HW_APBX_CH2_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 11-55. HW_APBX_CH2_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 2.

EXAMPLE:

```
Empty example.
```

11.5.27 APBX DMA Channel 3 Current Command Address Register Description

The APBX DMA Channel 3 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBX_CH3_CURCMDAR 0x250

Table 11-56. HW_APBX_CH3_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-57. HW_APBX_CH3_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for channel 3.

DESCRIPTION:

APBX DMA Channel 3 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

11.5.28 APBX DMA Channel 3 Next Command Address Register Description

The APBX DMA Channel 3 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBX_CH3_NXTCMDAR 0x260

Table 11-58. HW_APBX_CH3_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-59. HW_APBX_CH3_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for channel 3.

DESCRIPTION:

APBX DMA Channel 3 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 3 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

11.5.29 APBX DMA Channel 3 Command Register Description

The APBX DMA Channel 3 command register specifies the cycle to perform for the current command chain item.

HW_APBX_CH3_CMD 0x270

Table 11-60. HW_APBX_CH3_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1			HALTONTERMINATE	WAIT4ENDCMD	SEMAPHORE	RSVD0		IRQONCMPLT	CHAIN	COMMAND	

Table 11-61. HW_APBX_CH3_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the I2C device HW_I2C_DATA. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the I2C, starting with the base PIO address of the I2C (HW_I2C_CTRL0) and increment from there. Zero means transfer NO command words
11:9	**RSVD1**	RO	0x0	Reserved, always set to zero.
8	**HALTONTERMINATE**	RO	0x0	A value of one indicates that the channel will immeditately terminate the current descriptor and halt the DMA channel if a terminate signal is set. A value of 0 will still cause an immediate terminate of the channel if the terminate signal is set, but the channel will continue as if the count had been exhausted, meaning it will honor IRQONCMPLT, CHAIN, SEMAPHORE, and WAIT4ENDCMD.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBX device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5:4	**RSVD0**	RO	0x0	Reserved, always set to zero.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause its interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.

Table 11-61. HW_APBX_CH3_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBX_CH3_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- write transfers, i.e. data sent from the APBX device (APB PIO Read) to the system memory (AHB master write). 10- read transfer 11- reserved NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

11.5.30 APBX DMA Channel 3 Buffer Address Register Description

The APBX DMA Channel 3 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBX_CH3_BAR 0x280

Table 11-62. HW_APBX_CH3_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 11-63. HW_APBX_CH3_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

11.5.31 APBX DMA Channel 3 Semaphore Register Description

The APBX DMA Channel 3 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBX_CH3_SEMA 0x290

Table 11-64. HW_APBX_CH3_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 11-65. HW_APBX_CH3_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore used to synchronize between the program stream and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore which has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

11.5.32 AHB to APBX DMA Channel 3 Debug Information Description

This register gives debug visibility into the APBX DMA Channel 3 state machine and controls.

HW_APBX_CH3_DEBUG1 0x2A0

Table 11-66. HW_APBX_CH3_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	RSVD2			NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 11-67. HW_APBX_CH3_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27:25	**RSVD2**	RO	0x0	Reserved
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.

Table 11-67. HW_APBX_CH3_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 3 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. TERMINATE = 0x14 When a terminate signal is set, the state machine enters this state until the current AHB transfer is completed. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. HALT_AFTER_TERM = 0x1D If HALTONTERMINATE is set and a terminate signal is set, the state machine enters this state and effectively halts. A channel reset is required to exit this state CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 3.

EXAMPLE:

```
Empty example.
```

11.5.33 AHB to APBX DMA Channel 3 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 3.

HW_APBX_CH3_DEBUG2 0x2B0

Table 11-68. HW_APBX_CH3_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 11-69. HW_APBX_CH3_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 3.

EXAMPLE:

```
Empty example.
```

11.5.34 APBX DMA Channel 4 Current Command Address Register Description

The APBX DMA Channel 4 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBX_CH4_CURCMDAR 0x2C0

Table 11-70. HW_APBX_CH4_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-71. HW_APBX_CH4_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for channel 4.

DESCRIPTION:

APBX DMA Channel 4 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

11.5.35 APBX DMA Channel 4 Next Command Address Register Description

The APBX DMA Channel 4 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBX_CH4_NXTCMDAR 0x2D0

Table 11-72. HW_APBX_CH4_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-73. HW_APBX_CH4_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for channel 4.

DESCRIPTION:

APBX DMA Channel 4 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 4 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

11.5.36 APBX DMA Channel 4 Command Register Description

The APBX DMA Channel 4 command register specifies the cycle to perform for the current command chain item.

HW_APBX_CH4_CMD 0x2E0

Table 11-74. HW_APBX_CH4_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1				WAIT4ENDCMD	SEMAPHORE	RSVD0		IRQONCMPLT	CHAIN	COMMAND	

Table 11-75. HW_APBX_CH4_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the SAIF1 device. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the SAIF1. Zero means transfer NO command words
11:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBX device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5:4	**RSVD0**	RO	0x0	Reserved, always set to zero.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause its interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBX_CH4_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- write transfers, i.e. data sent from the APBX device (APB PIO Read) to the system memory (AHB master write). 10- read transfer 11- reserved NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

11.5.37 APBX DMA Channel 4 Buffer Address Register Description

The APBX DMA Channel 4 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBX_CH4_BAR 0x2F0

Table 11-76. HW_APBX_CH4_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 11-77. HW_APBX_CH4_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device associate with this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

11.5.38 APBX DMA Channel 4 Semaphore Register Description

The APBX DMA Channel 4 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBX_CH4_SEMA 0x300

Table 11-78. HW_APBX_CH4_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 11-79. HW_APBX_CH4_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore used to synchronize between the program stream and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore which has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

11.5.39 AHB to APBX DMA Channel 4 Debug Information Description

This register gives debug visibility into the APBX DMA Channel 4 state machine and controls.

HW_APBX_CH4_DEBUG1 0x310

Table 11-80. HW_APBX_CH4_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	RSVD2			NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 11-81. HW_APBX_CH4_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device

Table 11-81. HW_APBX_CH4_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27:25	**RSVD2**	RO	0x0	Reserved
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 4 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 4.

EXAMPLE:

```
Empty example.
```

11.5.40 AHB to APBX DMA Channel 4 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 4.

HW_APBX_CH4_DEBUG2 0x320

Table 11-82. HW_APBX_CH4_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 11-83. HW_APBX_CH4_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 4.

EXAMPLE:

```
Empty example.
```

11.5.41 APBX DMA Channel 5 Current Command Address Register Description

The APBX DMA Channel 5 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBX_CH5_CURCMDAR 0x330

Table 11-84. HW_APBX_CH5_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-85. HW_APBX_CH5_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for channel 5.

DESCRIPTION:

APBX DMA Channel 5 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

11.5.42 APBX DMA Channel 5 Next Command Address Register Description

The APBX DMA Channel 5 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBX_CH5_NXTCMDAR 0x340

Table 11-86. HW_APBX_CH5_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-87. HW_APBX_CH5_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for channel 5.

DESCRIPTION:

APBX DMA Channel 5 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 5 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

11.5.43 APBX DMA Channel 5 Command Register Description

The APBX DMA Channel 5 command register specifies the cycle to perform for the current command chain item.

HW_APBX_CH5_CMD 0x350

Table 11-88. HW_APBX_CH5_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1				WAIT4ENDCMD	SEMAPHORE	RSVD0		IRQONCMPLT	CHAIN	COMMAND	

Table 11-89. HW_APBX_CH5_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the DRI device HW_DRI_DATA register. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the DRI, starting with the base PIO address of the DRI (HW_DRI_CTRL) and increment from there. Zero means transfer NO command words
11:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBX device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5:4	**RSVD0**	RO	0x0	Reserved, always set to zero.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause its interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBX_CH5_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- write transfers, i.e. data sent from the APBX device (APB PIO Read) to the system memory (AHB master write). 10- read transfer 11- reserved NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

11.5.44 APBX DMA Channel 5 Buffer Address Register Description

The APBX DMA Channel 5 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBX_CH5_BAR 0x360

Table 11-90. HW_APBX_CH5_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 11-91. HW_APBX_CH5_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

11.5.45 APBX DMA Channel 5 Semaphore Register Description

The APBX DMA Channel 5 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBX_CH5_SEMA 0x370

Table 11-92. HW_APBX_CH5_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 11-93. HW_APBX_CH5_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore used to synchronize between the program stream and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore which has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

11.5.46 AHB to APBX DMA Channel 5 Debug Information Description

This register gives debug visibility into the APBX DMA Channel 5 state machine and controls.

HW_APBX_CH5_DEBUG1 0x380

Table 11-94. HW_APBX_CH5_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	RSVD2			NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 11-95. HW_APBX_CH5_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27:25	**RSVD2**	RO	0x0	Reserved
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.

Table 11-95. HW_APBX_CH5_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 5 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 5.

EXAMPLE:

```
Empty example.
```

11.5.47 AHB to APBX DMA Channel 5 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 5.

HW_APBX_CH5_DEBUG2 0x390

Table 11-96. HW_APBX_CH5_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 11-97. HW_APBX_CH5_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 5.

EXAMPLE:

```
Empty example.
```

11.5.48 APBX DMA Channel 6 Current Command Address Register Description

The APBX DMA Channel 6 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBX_CH6_CURCMDAR 0x3A0

Table 11-98. HW_APBX_CH6_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-99. HW_APBX_CH6_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for channel 6.

DESCRIPTION:

APBX DMA Channel 6 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

11.5.49 APBX DMA Channel 6 Next Command Address Register Description

The APBX DMA Channel 6 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBX_CH6_NXTCMDAR 0x3B0

Table 11-100. HW_APBX_CH6_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-101. HW_APBX_CH6_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for channel 6.

DESCRIPTION:

APBX DMA Channel 6 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 6 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

11.5.50 APBX DMA Channel 6 Command Register Description

The APBX DMA Channel 6 command register specifies the cycle to perform for the current command chain item.

HW_APBX_CH6_CMD 0x3C0

Table 11-102. HW_APBX_CH6_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1				WAIT4ENDCMD	SEMAPHORE	RSVD0		IRQONCMPLT	CHAIN	COMMAND	

Table 11-103. HW_APBX_CH6_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the UART device HW_UARTAPP_DATA register. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the UART, starting with the base PIO address of the UART (HW_UARTAPP_CTRL0) and increment from there. Zero means transfer NO command words
11:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBX device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5:4	**RSVD0**	RO	0x0	Reserved, always set to zero.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause its interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBX_CH6_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- write transfers, i.e. data sent from the APBX device (APB PIO Read) to the system memory (AHB master write). 10- read transfer 11- reserved NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

11.5.51 APBX DMA Channel 6 Buffer Address Register Description

The APBX DMA Channel 6 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBX_CH6_BAR 0x3D0

Table 11-104. HW_APBX_CH6_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 11-105. HW_APBX_CH6_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

11.5.52 APBX DMA Channel 6 Semaphore Register Description

The APBX DMA Channel 6 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBX_CH6_SEMA 0x3E0

Table 11-106. HW_APBX_CH6_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 11-107. HW_APBX_CH6_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore used to synchronize between the program stream and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore which has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

11.5.53 AHB to APBX DMA Channel 6 Debug Information Description

This register gives debug visibility into the APBX DMA Channel 6 state machine and controls.

HW_APBX_CH6_DEBUG1 0x3F0

Table 11-108. HW_APBX_CH6_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	RSVD2			NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 11-109. HW_APBX_CH6_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device

Table 11-109. HW_APBX_CH6_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27:25	**RSVD2**	RO	0x0	Reserved
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 6 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 6.

EXAMPLE:

```
Empty example.
```

11.5.54 AHB to APBX DMA Channel 6 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 6.

HW_APBX_CH6_DEBUG2 0x400

Table 11-110. HW_APBX_CH6_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 11-111. HW_APBX_CH6_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 6.

EXAMPLE:

```
Empty example.
```

11.5.55 APBX DMA Channel 7 Current Command Address Register Description

The APBX DMA Channel 7 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBX_CH7_CURCMDAR 0x410

Table 11-112. HW_APBX_CH7_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-113. HW_APBX_CH7_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for channel 7.

DESCRIPTION:

APBX DMA Channel 7 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

11.5.56 APBX DMA Channel 7 Next Command Address Register Description

The APBX DMA Channel 7 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBX_CH7_NXTCMDAR 0x420

Table 11-114. HW_APBX_CH7_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-115. HW_APBX_CH7_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for channel 7.

DESCRIPTION:

APBX DMA Channel 7 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 7 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

11.5.57 APBX DMA Channel 7 Command Register Description

The APBX DMA Channel 7 command register specifies the cycle to perform for the current command chain item.

HW_APBX_CH7_CMD 0x430

Table 11-116. HW_APBX_CH7_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1			HALTONTERMINATE	WAIT4ENDCMD	SEMAPHORE	RSVD0		IRQONCMPLT	CHAIN	COMMAND	

Table 11-117. HW_APBX_CH7_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the UART device HW_UARTAPP_DATA register. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the UART, starting with the base PIO address of the UART (HW_UARTAPP_CTRL0) and increment from there. Zero means transfer NO command words
11:9	**RSVD1**	RO	0x0	Reserved, always set to zero.
8	**HALTONTERMINATE**	RO	0x0	A value of one indicates that the channel will immeditately terminate the current descriptor and halt the DMA channel if a terminate signal is set. A value of 0 will still cause an immediate terminate of the channel if the terminate signal is set, but the channel will continue as if the count had been exhausted, meaning it will honor IRQONCMPLT, CHAIN, SEMAPHORE, and WAIT4ENDCMD.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBX device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5:4	**RSVD0**	RO	0x0	Reserved, always set to zero.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause its interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.

Table 11-117. HW_APBX_CH7_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBX_CH7_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- write transfers, i.e. data sent from the APBX device (APB PIO Read) to the system memory (AHB master write). 10- read transfer 11- reserved NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

11.5.58 APBX DMA Channel 7 Buffer Address Register Description

The APBX DMA Channel 7 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBX_CH7_BAR 0x440

Table 11-118. HW_APBX_CH7_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 11-119. HW_APBX_CH7_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

11.5.59 APBX DMA Channel 7 Semaphore Register Description

The APBX DMA Channel 7 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBX_CH7_SEMA 0x450

Table 11-120. HW_APBX_CH7_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 11-121. HW_APBX_CH7_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore used to synchronize between the program stream and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore which has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

11.5.60 AHB to APBX DMA Channel 7 Debug Information Description

This register gives debug visibility into the APBX DMA Channel 7 state machine and controls.

HW_APBX_CH7_DEBUG1 0x460

Table 11-122. HW_APBX_CH7_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	RSVD2			NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 11-123. HW_APBX_CH7_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27:25	**RSVD2**	RO	0x0	Reserved
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.

Table 11-123. HW_APBX_CH7_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 7 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. TERMINATE = 0x14 When a terminate signal is set, the state machine enters this state until the current AHB transfer is completed. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. HALT_AFTER_TERM = 0x1D If HALTONTERMINATE is set and a terminate signal is set, the state machine enters this state and effectively halts. A channel reset is required to exit this state CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 7.

EXAMPLE:

```
Empty example.
```

11.5.61 AHB to APBX DMA Channel 7 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 7.

HW_APBX_CH7_DEBUG2 0x470

Table 11-124. HW_APBX_CH7_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 11-125. HW_APBX_CH7_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 7.

EXAMPLE:

```
Empty example.
```

11.5.62 APBX DMA Channel 8 Current Command Address Register Description

The APBX DMA Channel 8 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBX_CH8_CURCMDAR 0x480

Table 11-126. HW_APBX_CH8_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-127. HW_APBX_CH8_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for Channel 8.

DESCRIPTION:

APBX DMA Channel 8 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

11.5.63 APBX DMA Channel 8 Next Command Address Register Description

The APBX DMA Channel 8 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBX_CH8_NXTCMDAR 0x490

Table 11-128. HW_APBX_CH8_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-129. HW_APBX_CH8_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for Channel 8.

DESCRIPTION:

APBX DMA Channel 8 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 8 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

11.5.64 APBX DMA Channel 8 Command Register Description

The APBX DMA Channel 8 command register specifies the cycle to perform for the current command chain item.

HW_APBX_CH8_CMD 0x4A0

Table 11-130. HW_APBX_CH8_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1			HALTONTERMINATE	WAIT4ENDCMD	SEMAPHORE	RSVD0		IRQONCMPLT	CHAIN	COMMAND	

Table 11-131. HW_APBX_CH8_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the UART device HW_UARTAPP_DATA register. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the UART, starting with the base PIO address of the UART (HW_UARTAPP_CTRL0) and increment from there. Zero means transfer NO command words
11:9	**RSVD1**	RO	0x0	Reserved, always set to zero.
8	**HALTONTERMINATE**	RO	0x0	A value of one indicates that the channel will immeditately terminate the current descriptor and halt the DMA channel if a terminate signal is set. A value of 0 will still cause an immediate terminate of the channel if the terminate signal is set, but the channel will continue as if the count had been exhausted, meaning it will honor IRQONCMPLT, CHAIN, SEMAPHORE, and WAIT4ENDCMD.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBX device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5:4	**RSVD0**	RO	0x0	Reserved, always set to zero.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause its interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBX_CH8_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- write transfers, i.e. data sent from the APBX device (APB PIO Read) to the system memory (AHB master write). 10- read transfer 11- reserved NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included

with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

11.5.65 APBX DMA Channel 8 Buffer Address Register Description

The APBX DMA Channel 8 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBX_CH8_BAR 0x4B0

Table 11-132. HW_APBX_CH8_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 11-133. HW_APBX_CH8_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

11.5.66 APBX DMA Channel 8 Semaphore Register Description

The APBX DMA Channel 8 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBX_CH8_SEMA 0x4C0

Table 11-134. HW_APBX_CH8_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 11-135. HW_APBX_CH8_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore used to synchronize between the program stream and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore which has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

11.5.67 AHB to APBX DMA Channel 8 Debug Information Description

This register gives debug visibility into the APBX DMA Channel 8 state machine and controls.

HW_APBX_CH8_DEBUG1 0x4D0

Table 11-136. HW_APBX_CH8_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	RSVD2			NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 11-137. HW_APBX_CH8_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27:25	**RSVD2**	RO	0x0	Reserved
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.

Table 11-137. HW_APBX_CH8_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 8 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. TERMINATE = 0x14 When a terminate signal is set, the state machine enters this state until the current AHB transfer is completed. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. HALT_AFTER_TERM = 0x1D If HALTONTERMINATE is set and a terminate signal is set, the state machine enters this state and effectively halts. A channel reset is required to exit this state CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 8.

EXAMPLE:

```
Empty example.
```

11.5.68 AHB to APBX DMA Channel 8 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 8.

HW_APBX_CH8_DEBUG2 0x4E0

Table 11-138. HW_APBX_CH8_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 11-139. HW_APBX_CH8_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 8.

EXAMPLE:

```
Empty example.
```

11.5.69 APBX DMA Channel 9 Current Command Address Register Description

The APBX DMA Channel 9 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBX_CH9_CURCMDAR 0x4F0

Table 11-140. HW_APBX_CH9_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-141. HW_APBX_CH9_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for Channel 9.

DESCRIPTION:

APBX DMA Channel 9 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

11.5.70 APBX DMA Channel 9 Next Command Address Register Description

The APBX DMA Channel 9 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBX_CH9_NXTCMDAR 0x500

Table 11-142. HW_APBX_CH9_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-143. HW_APBX_CH9_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for Channel 9.

DESCRIPTION:

APBX DMA Channel 9 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 9 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

11.5.71 APBX DMA Channel 9 Command Register Description

The APBX DMA Channel 9 command register specifies the cycle to perform for the current command chain item.

HW_APBX_CH9_CMD 0x510

Table 11-144. HW_APBX_CH9_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1				WAIT4ENDCMD	SEMAPHORE	RSVD0		IRQONCMPLT	CHAIN	COMMAND	

Table 11-145. HW_APBX_CH9_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the UART device HW_UARTAPP_DATA register. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the UART, starting with the base PIO address of the UART (HW_UARTAPP_CTRL0) and increment from there. Zero means transfer NO command words
11:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBX device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5:4	**RSVD0**	RO	0x0	Reserved, always set to zero.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause its interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBX_CH9_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- write transfers, i.e. data sent from the APBX device (APB PIO Read) to the system memory (AHB master write). 10- read transfer 11- reserved NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

11.5.72 APBX DMA Channel 9 Buffer Address Register Description

The APBX DMA Channel 9 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBX_CH9_BAR 0x520

Table 11-146. HW_APBX_CH9_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 11-147. HW_APBX_CH9_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

11.5.73 APBX DMA Channel 9 Semaphore Register Description

The APBX DMA Channel 9 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBX_CH9_SEMA 0x530

Table 11-148. HW_APBX_CH9_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 11-149. HW_APBX_CH9_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore used to synchronize between the program stream and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore which has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

11.5.74 AHB to APBX DMA Channel 9 Debug Information Description

This register gives debug visibility into the APBX DMA Channel 9 state machine and controls.

HW_APBX_CH9_DEBUG1 0x540

Table 11-150. HW_APBX_CH9_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	RSVD2			NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 11-151. HW_APBX_CH9_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device

Table 11-151. HW_APBX_CH9_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27:25	**RSVD2**	RO	0x0	Reserved
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 9 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 9.

EXAMPLE:

```
Empty example.
```

11.5.75 AHB to APBX DMA Channel 9 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 9.

HW_APBX_CH9_DEBUG2 0x550

Table 11-152. HW_APBX_CH9_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 11-153. HW_APBX_CH9_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 9.

EXAMPLE:

```
Empty example.
```

11.5.76 APBX DMA Channel 10 Current Command Address Register Description

The APBX DMA Channel 10 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBX_CH10_CURCMDAR 0x560

Table 11-154. HW_APBX_CH10_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-155. HW_APBX_CH10_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for Channel 10.

DESCRIPTION:

APBX DMA Channel 10 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

11.5.77 APBX DMA Channel 10 Next Command Address Register Description

The APBX DMA Channel 10 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBX_CH10_NXTCMDAR 0x570

Table 11-156. HW_APBX_CH10_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-157. HW_APBX_CH10_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for Channel 10.

DESCRIPTION:

APBX DMA Channel 10 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 10 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

11.5.78 APBX DMA Channel 10 Command Register Description

The APBX DMA Channel 10 command register specifies the cycle to perform for the current command chain item.

HW_APBX_CH10_CMD 0x580

Table 11-158. HW_APBX_CH10_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1			HALTONTERMINATE	WAIT4ENDCMD	SEMAPHORE	RSVD0		IRQONCMPLT	CHAIN	COMMAND	

Table 11-159. HW_APBX_CH10_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the UART device HW_UARTAPP_DATA register. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the UART, starting with the base PIO address of the UART (HW_UARTAPP_CTRL0) and increment from there. Zero means transfer NO command words
11:9	**RSVD1**	RO	0x0	Reserved, always set to zero.
8	**HALTONTERMINATE**	RO	0x0	A value of one indicates that the channel will immeditately terminate the current descriptor and halt the DMA channel if a terminate signal is set. A value of 0 will still cause an immediate terminate of the channel if the terminate signal is set, but the channel will continue as if the count had been exhausted, meaning it will honor IRQONCMPLT, CHAIN, SEMAPHORE, and WAIT4ENDCMD.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBX device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5:4	**RSVD0**	RO	0x0	Reserved, always set to zero.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause its interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.

Table 11-159. HW_APBX_CH10_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBX_CH10_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- write transfers, i.e. data sent from the APBX device (APB PIO Read) to the system memory (AHB master write). 10- read transfer 11- reserved NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

11.5.79 APBX DMA Channel 10 Buffer Address Register Description

The APBX DMA Channel 10 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBX_CH10_BAR 0x590

Table 11-160. HW_APBX_CH10_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 11-161. HW_APBX_CH10_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

11.5.80 APBX DMA Channel 10 Semaphore Register Description

The APBX DMA Channel 10 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBX_CH10_SEMA 0x5A0

Table 11-162. HW_APBX_CH10_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 11-163. HW_APBX_CH10_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore used to synchronize between the program stream and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore which has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

11.5.81 AHB to APBX DMA Channel 10 Debug Information Description

This register gives debug visibility into the APBX DMA Channel 10 state machine and controls.

HW_APBX_CH10_DEBUG1 0x5B0

Table 11-164. HW_APBX_CH10_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	RSVD2			NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 11-165. HW_APBX_CH10_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27:25	**RSVD2**	RO	0x0	Reserved
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x1	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.

Table 11-165. HW_APBX_CH10_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 10 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. TERMINATE = 0x14 When a terminate signal is set, the state machine enters this state until the current AHB transfer is completed. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. HALT_AFTER_TERM = 0x1D If HALTONTERMINATE is set and a terminate signal is set, the state machine enters this state and effectively halts. A channel reset is required to exit this state CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 10.

EXAMPLE:

```
Empty example.
```

11.5.82 AHB to APBX DMA Channel 10 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 10.

HW_APBX_CH10_DEBUG2 0x5C0

Table 11-166. HW_APBX_CH10_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 11-167. HW_APBX_CH10_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 10.

EXAMPLE:

```
Empty example.
```

11.5.83 APBX DMA Channel 11 Current Command Address Register Description

The APBX DMA Channel 11 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBX_CH11_CURCMDAR 0x5D0

Table 11-168. HW_APBX_CH11_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-169. HW_APBX_CH11_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for Channel 11.

DESCRIPTION:

APBX DMA Channel 11 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

11.5.84 APBX DMA Channel 11 Next Command Address Register Description

The APBX DMA Channel 11 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBX_CH11_NXTCMDAR 0x5E0

Table 11-170. HW_APBX_CH11_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-171. HW_APBX_CH11_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for Channel 11.

DESCRIPTION:

APBX DMA Channel 11 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 11 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

11.5.85 APBX DMA Channel 11 Command Register Description

The APBX DMA Channel 11 command register specifies the cycle to perform for the current command chain item.

HW_APBX_CH11_CMD 0x5F0

Table 11-172. HW_APBX_CH11_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1				WAIT4ENDCMD	SEMAPHORE	RSVD0		IRQONCMPLT	CHAIN	COMMAND	

Table 11-173. HW_APBX_CH11_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the UART device HW_UARTAPP_DATA register. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the UART, starting with the base PIO address of the UART (HW_UARTAPP_CTRL0) and increment from there. Zero means transfer NO command words
11:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBX device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5:4	**RSVD0**	RO	0x0	Reserved, always set to zero.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause its interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBX_CH11_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- write transfers, i.e. data sent from the APBX device (APB PIO Read) to the system memory (AHB master write). 10- read transfer 11- reserved NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

11.5.86 APBX DMA Channel 11 Buffer Address Register Description

The APBX DMA Channel 11 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBX_CH11_BAR 0x600

Table 11-174. HW_APBX_CH11_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 11-175. HW_APBX_CH11_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

11.5.87 APBX DMA Channel 11 Semaphore Register Description

The APBX DMA Channel 11 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBX_CH11_SEMA 0x610

Table 11-176. HW_APBX_CH11_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 11-177. HW_APBX_CH11_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore used to synchronize between the program stream and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore which has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

11.5.88 AHB to APBX DMA Channel 11 Debug Information Description

This register gives debug visibility into the APBX DMA Channel 11 state machine and controls.

HW_APBX_CH11_DEBUG1 0x620

Table 11-178. HW_APBX_CH11_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	RSVD2			NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 11-179. HW_APBX_CH11_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device

Table 11-179. HW_APBX_CH11_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27:25	**RSVD2**	RO	0x0	Reserved
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 11 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 11.

EXAMPLE:

```
Empty example.
```

11.5.89 AHB to APBX DMA Channel 11 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 11.

HW_APBX_CH11_DEBUG2 0x630

Table 11-180. HW_APBX_CH11_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 11-181. HW_APBX_CH11_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 11.

EXAMPLE:

```
Empty example.
```

11.5.90 APBX DMA Channel 12 Current Command Address Register Description

The APBX DMA Channel 12 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBX_CH12_CURCMDAR 0x640

Table 11-182. HW_APBX_CH12_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-183. HW_APBX_CH12_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for Channel 12.

DESCRIPTION:

APBX DMA Channel 12 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

11.5.91 APBX DMA Channel 12 Next Command Address Register Description

The APBX DMA Channel 12 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBX_CH12_NXTCMDAR 0x650

Table 11-184. HW_APBX_CH12_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-185. HW_APBX_CH12_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for Channel 12.

DESCRIPTION:

APBX DMA Channel 12 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 12 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

11.5.92 APBX DMA Channel 12 Command Register Description

The APBX DMA Channel 12 command register specifies the cycle to perform for the current command chain item.

HW_APBX_CH12_CMD 0x660

Table 11-186. HW_APBX_CH12_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1			HALTONTERMINATE	WAIT4ENDCMD	SEMAPHORE	RSVD0		IRQONCMPLT	CHAIN	COMMAND	

Table 11-187. HW_APBX_CH12_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the UART device HW_UARTAPP_DATA register. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the UART, starting with the base PIO address of the UART (HW_UARTAPP_CTRL0) and increment from there. Zero means transfer NO command words
11:9	**RSVD1**	RO	0x0	Reserved, always set to zero.
8	**HALTONTERMINATE**	RO	0x0	A value of one indicates that the channel will immeditately terminate the current descriptor and halt the DMA channel if a terminate signal is set. A value of 0 will still cause an immediate terminate of the channel if the terminate signal is set, but the channel will continue as if the count had been exhausted, meaning it will honor IRQONCMPLT, CHAIN, SEMAPHORE, and WAIT4ENDCMD.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBX device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5:4	**RSVD0**	RO	0x0	Reserved, always set to zero.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause its interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.

Table 11-187. HW_APBX_CH12_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBX_CH12_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- write transfers, i.e. data sent from the APBX device (APB PIO Read) to the system memory (AHB master write). 10- read transfer 11- reserved NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

11.5.93 APBX DMA Channel 12 Buffer Address Register Description

The APBX DMA Channel 12 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBX_CH12_BAR 0x670

Table 11-188. HW_APBX_CH12_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 11-189. HW_APBX_CH12_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

11.5.94 APBX DMA Channel 12 Semaphore Register Description

The APBX DMA Channel 12 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBX_CH12_SEMA 0x680

Table 11-190. HW_APBX_CH12_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 11-191. HW_APBX_CH12_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore used to synchronize between the program stream and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore which has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

11.5.95 AHB to APBX DMA Channel 12 Debug Information Description

This register gives debug visibility into the APBX DMA Channel 12 state machine and controls.

HW_APBX_CH12_DEBUG1 0x690

Table 11-192. HW_APBX_CH12_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	RSVD2			NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 11-193. HW_APBX_CH12_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27:25	**RSVD2**	RO	0x0	Reserved
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.

Table 11-193. HW_APBX_CH12_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 12 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. TERMINATE = 0x14 When a terminate signal is set, the state machine enters this state until the current AHB transfer is completed. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. HALT_AFTER_TERM = 0x1D If HALTONTERMINATE is set and a terminate signal is set, the state machine enters this state and effectively halts. A channel reset is required to exit this state CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 12.

EXAMPLE:

```
Empty example.
```

11.5.96 AHB to APBX DMA Channel 12 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 12.

HW_APBX_CH12_DEBUG2 0x6A0

Table 11-194. HW_APBX_CH12_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 11-195. HW_APBX_CH12_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 12.

EXAMPLE:

```
Empty example.
```

11.5.97 APBX DMA Channel 13 Current Command Address Register Description

The APBX DMA Channel 13 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBX_CH13_CURCMDAR 0x6B0

Table 11-196. HW_APBX_CH13_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-197. HW_APBX_CH13_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for Channel 13.

DESCRIPTION:

APBX DMA Channel 13 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

11.5.98 APBX DMA Channel 13 Next Command Address Register Description

The APBX DMA Channel 13 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBX_CH13_NXTCMDAR 0x6C0

Table 11-198. HW_APBX_CH13_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-199. HW_APBX_CH13_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for Channel 13.

DESCRIPTION:

APBX DMA Channel 13 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 13 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

11.5.99 APBX DMA Channel 13 Command Register Description

The APBX DMA Channel 13 command register specifies the cycle to perform for the current command chain item.

HW_APBX_CH13_CMD 0x6D0

Table 11-200. HW_APBX_CH13_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1				WAIT4ENDCMD	SEMAPHORE	RSVD0		IRQONCMPLT	CHAIN	COMMAND	

Table 11-201. HW_APBX_CH13_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the UART device HW_UARTAPP_DATA register. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the UART, starting with the base PIO address of the UART (HW_UARTAPP_CTRL0) and increment from there. Zero means transfer NO command words
11:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBX device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5:4	**RSVD0**	RO	0x0	Reserved, always set to zero.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause its interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBX_CH13_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- write transfers, i.e. data sent from the APBX device (APB PIO Read) to the system memory (AHB master write). 10- read transfer 11- reserved NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

11.5.100 APBX DMA Channel 13 Buffer Address Register Description

The APBX DMA Channel 13 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBX_CH13_BAR 0x6E0

Table 11-202. HW_APBX_CH13_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 11-203. HW_APBX_CH13_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

11.5.101 APBX DMA Channel 13 Semaphore Register Description

The APBX DMA Channel 13 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBX_CH13_SEMA 0x6F0

Table 11-204. HW_APBX_CH13_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 11-205. HW_APBX_CH13_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore used to synchronize between the program stream and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore which has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

11.5.102 AHB to APBX DMA Channel 13 Debug Information Description

This register gives debug visibility into the APBX DMA Channel 13 state machine and controls.

HW_APBX_CH13_DEBUG1 0x700

Table 11-206. HW_APBX_CH13_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	RSVD2			NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 11-207. HW_APBX_CH13_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device

Table 11-207. HW_APBX_CH13_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27:25	**RSVD2**	RO	0x0	Reserved
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 13 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 13.

EXAMPLE:

```
Empty example.
```

11.5.103 AHB to APBX DMA Channel 13 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 13.

HW_APBX_CH13_DEBUG2 0x710

Table 11-208. HW_APBX_CH13_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 11-209. HW_APBX_CH13_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 13.

EXAMPLE:

```
Empty example.
```

11.5.104 APBX DMA Channel 14 Current Command Address Register Description

The APBX DMA Channel 14 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBX_CH14_CURCMDAR 0x720

Table 11-210. HW_APBX_CH14_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-211. HW_APBX_CH14_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for Channel 14.

DESCRIPTION:

APBX DMA Channel 14 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

11.5.105 APBX DMA Channel 14 Next Command Address Register Description

The APBX DMA Channel 14 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBX_CH14_NXTCMDAR 0x730

Table 11-212. HW_APBX_CH14_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-213. HW_APBX_CH14_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for Channel 14.

DESCRIPTION:

APBX DMA Channel 14 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 14 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

11.5.106 APBX DMA Channel 14 Command Register Description

The APBX DMA Channel 14 command register specifies the cycle to perform for the current command chain item.

HW_APBX_CH14_CMD 0x740

Table 11-214. HW_APBX_CH14_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1			HALTONTERMINATE	WAIT4ENDCMD	SEMAPHORE	RSVD0		IRQONCMPLT	CHAIN	COMMAND	

Table 11-215. HW_APBX_CH14_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the UART device HW_UARTAPP_DATA register. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the UART, starting with the base PIO address of the UART (HW_UARTAPP_CTRL0) and increment from there. Zero means transfer NO command words
11:9	**RSVD1**	RO	0x0	Reserved, always set to zero.
8	**HALTONTERMINATE**	RO	0x0	A value of one indicates that the channel will immeditately terminate the current descriptor and halt the DMA channel if a terminate signal is set. A value of 0 will still cause an immediate terminate of the channel if the terminate signal is set, but the channel will continue as if the count had been exhausted, meaning it will honor IRQONCMPLT, CHAIN, SEMAPHORE, and WAIT4ENDCMD.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBX device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5:4	**RSVD0**	RO	0x0	Reserved, always set to zero.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause its interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.

Table 11-215. HW_APBX_CH14_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBX_CH14_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- write transfers, i.e. data sent from the APBX device (APB PIO Read) to the system memory (AHB master write). 10- read transfer 11- reserved NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

11.5.107 APBX DMA Channel 14 Buffer Address Register Description

The APBX DMA Channel 14 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBX_CH14_BAR 0x750

Table 11-216. HW_APBX_CH14_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 11-217. HW_APBX_CH14_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

11.5.108 APBX DMA Channel 14 Semaphore Register Description

The APBX DMA Channel 14 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBX_CH14_SEMA 0x760

Table 11-218. HW_APBX_CH14_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 11-219. HW_APBX_CH14_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore used to synchronize between the program stream and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore which has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

11.5.109 AHB to APBX DMA Channel 14 Debug Information Description

This register gives debug visibility into the APBX DMA Channel 14 state machine and controls.

HW_APBX_CH14_DEBUG1 0x770

Table 11-220. HW_APBX_CH14_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	RSVD2			NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 11-221. HW_APBX_CH14_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27:25	**RSVD2**	RO	0x0	Reserved
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.

Table 11-221. HW_APBX_CH14_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 14 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. TERMINATE = 0x14 When a terminate signal is set, the state machine enters this state until the current AHB transfer is completed. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. HALT_AFTER_TERM = 0x1D If HALTONTERMINATE is set and a terminate signal is set, the state machine enters this state and effectively halts. A channel reset is required to exit this state CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 14.

EXAMPLE:

```
Empty example.
```

11.5.110 AHB to APBX DMA Channel 14 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 14.

HW_APBX_CH14_DEBUG2 0x780

Table 11-222. HW_APBX_CH14_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 11-223. HW_APBX_CH14_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 14.

EXAMPLE:

```
Empty example.
```

11.5.111 APBX DMA Channel 15 Current Command Address Register Description

The APBX DMA Channel 15 current command address register points to the multiword command that is currently being executed. Commands are threaded on the command address.

HW_APBX_CH15_CURCMDAR 0x790

Table 11-224. HW_APBX_CH15_CURCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-225. HW_APBX_CH15_CURCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RO	0x00000000	Pointer to command structure currently being processed for Channel 15.

DESCRIPTION:

APBX DMA Channel 15 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

11.5.112 APBX DMA Channel 15 Next Command Address Register Description

The APBX DMA Channel 15 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBX_CH15_NXTCMDAR 0x7A0

Table 11-226. HW_APBX_CH15_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-227. HW_APBX_CH15_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for Channel 15.

DESCRIPTION:

APBX DMA Channel 15 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 15 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

11.5.113 APBX DMA Channel 15 Command Register Description

The APBX DMA Channel 15 command register specifies the cycle to perform for the current command chain item.

HW_APBX_CH15_CMD 0x7B0

Table 11-228. HW_APBX_CH15_CMD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XFER_COUNT																CMDWORDS				RSVD1				WAIT4ENDCMD	SEMAPHORE	RSVD0		IRQONCMPLT	CHAIN	COMMAND	

Table 11-229. HW_APBX_CH15_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**XFER_COUNT**	RO	0x0	This field indicates the number of bytes to transfer to or from the appropriate PIO register in the UART device HW_UARTAPP_DATA register. A value of 0 indicates a 64 KBytes transfer.
15:12	**CMDWORDS**	RO	0x00	This field indicates the number of command words to send to the UART, starting with the base PIO address of the UART (HW_UARTAPP_CTRL0) and increment from there. Zero means transfer NO command words
11:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7	**WAIT4ENDCMD**	RO	0x0	A value of one indicates that the channel will wait for the end of command signal to be sent from the APBX device to the DMA before starting the next DMA command.
6	**SEMAPHORE**	RO	0x0	A value of one indicates that the channel will decrement its semaphore at the completion of the current command structure. If the semaphore decrements to zero, then this channel stalls until software increments it again.
5:4	**RSVD0**	RO	0x0	Reserved, always set to zero.
3	**IRQONCMPLT**	RO	0x0	A value of one indicates that the channel will cause its interrupt status bit to be set upon completion of the current command, i.e. after the DMA transfer is complete.
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBX_CH15_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- write transfers, i.e. data sent from the APBX device (APB PIO Read) to the system memory (AHB master write). 10- read transfer 11- reserved NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty example.
```

11.5.114 APBX DMA Channel 15 Buffer Address Register Description

The APBX DMA Channel 15 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBX_CH15_BAR 0x7C0

Table 11-230. HW_APBX_CH15_BAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																															

Table 11-231. HW_APBX_CH15_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

DESCRIPTION:

This register holds a pointer to the data buffer in system memory. After the command values have been read into the DMA controller and the device controlled by this channel, then the DMA transfer will begin, to or from the buffer pointed to by this register.

EXAMPLE:

```
Empty example.
```

11.5.115 APBX DMA Channel 15 Semaphore Register Description

The APBX DMA Channel 15 semaphore register is used to synchronize between the CPU instruction stream and the DMA chain processing state.

HW_APBX_CH15_SEMA 0x7D0

Table 11-232. HW_APBX_CH15_SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								PHORE								RSVD1								INCREMENT_SEMA							

Table 11-233. HW_APBX_CH15_SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**PHORE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT_SEMA**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DMA hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DMA channel decrements the count on the same clock, then the count is incremented by a net one.

DESCRIPTION:

Each DMA channel has an 8 bit counting semaphore used to synchronize between the program stream and the DMA chain processing. DMA processing continues until the DMA attempts to decrement a semaphore which has already reached a value of zero. When the attempt is made, the DMA channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty example.
```

11.5.116 AHB to APBX DMA Channel 15 Debug Information Description

This register gives debug visibility into the APBX DMA Channel 15 state machine and controls.

HW_APBX_CH15_DEBUG1 0x7E0

Table 11-234. HW_APBX_CH15_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REQ	BURST	KICK	END	RSVD2			NEXTCMDADDRVALID	RD_FIFO_EMPTY	RD_FIFO_FULL	WR_FIFO_EMPTY	WR_FIFO_FULL	RSVD1																STATEMACHINE			

Table 11-235. HW_APBX_CH15_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**REQ**	RO	0x0	This bit reflects the current state of the DMA Request Signal from the APB device
30	**BURST**	RO	0x0	This bit reflects the current state of the DMA Burst Signal from the APB device

Table 11-235. HW_APBX_CH15_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
29	**KICK**	RO	0x0	This bit reflects the current state of the DMA Kick Signal sent to the APB Device
28	**END**	RO	0x0	This bit reflects the current state of the DMA End Command Signal sent from the APB Device
27:25	**RSVD2**	RO	0x0	Reserved
24	**NEXTCMDADDRVALID**	RO	0x0	This bit reflects the internal bit which indicates whether the channel's next command address is valid.
23	**RD_FIFO_EMPTY**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Empty signal.
22	**RD_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Read FIFO Full signal.
21	**WR_FIFO_EMPTY**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Empty signal.
20	**WR_FIFO_FULL**	RO	0x0	This bit reflects the current state of the DMA Channel's Write FIFO Full signal.
19:5	**RSVD1**	RO	0x0	Reserved
4:0	**STATEMACHINE**	RO	0x0	PIO Display of the DMA Channel 15 state machine state. IDLE = 0x00 This is the idle state of the DMA state machine. REQ_CMD1 = 0x01 State in which the DMA is waiting to receive the first word of a command. REQ_CMD3 = 0x02 State in which the DMA is waiting to receive the third word of a command. REQ_CMD2 = 0x03 State in which the DMA is waiting to receive the second word of a command. XFER_DECODE = 0x04 The state machine processes the descriptor command field in this state and branches accordingly. REQ_WAIT = 0x05 The state machine waits in this state for the PIO APB cycles to complete. REQ_CMD4 = 0x06 State in which the DMA is waiting to receive the fourth word of a command, or waiting to receive the PIO words when PIO count is greater than 1. PIO_REQ = 0x07 This state determines whether another PIO cycle needs to occur before starting DMA transfers. READ_FLUSH = 0x08 During a read transfers, the state machine enters this state waiting for the last bytes to be pushed out on the APB. READ_WAIT = 0x09 When an AHB read request occurs, the state machine waits in this state for the AHB transfer to complete. WRITE = 0x0C During DMA Write transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. READ_REQ = 0x0D During DMA Read transfers, the state machine waits in this state until the AHB master arbiter accepts the request from this channel. CHECK_CHAIN = 0x0E Upon completion of the DMA transfers, this state checks the value of the Chain bit and branches accordingly. XFER_COMPLETE = 0x0F The state machine goes to this state after the DMA transfers are complete, and determines what step to take next. WAIT_END = 0x15 When the Wait for Command End bit is set, the state machine enters this state until the DMA device indicates that the command is complete. WRITE_WAIT = 0x1C During DMA Write transfers, the state machine waits in this state until the AHB master completes the write to the AHB memory space. CHECK_WAIT = 0x1E If the Chain bit is a 0, the state machine enters this state and effectively halts.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 15.

EXAMPLE:

```
Empty example.
```

11.5.117 AHB to APBX DMA Channel 15 Debug Information Description

This register gives debug visibility for the APB and AHB byte counts for DMA Channel 15.

HW_APBX_CH15_DEBUG2 0x7F0

Table 11-236. HW_APBX_CH15_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
APB_BYTES																AHB_BYTES															

Table 11-237. HW_APBX_CH15_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_BYTES**	RO	0x0	This value reflects the current number of APB bytes remaining to be transfered in the current transfer.
15:0	**AHB_BYTES**	RO	0x0	This value reflects the current number of AHB bytes remaining to be transfered in the current transfer.

DESCRIPTION:

This register allows debug visibility of the APBX DMA Channel 15.

EXAMPLE:

```
Empty example.
```

11.5.118 APBX Bridge Version Register Description

This register always returns a known read value for debug purposes it indicates the version of the block.

HW_APBX_VERSION 0x800

Table 11-238. HW_APBX_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 11-239. HW_APBX_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x02	Fixed read-only value reflecting the MAJOR field of the RTL version.

Table 11-239. HW_APBX_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
23:16	**MINOR**	RO	0x01	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
if (HW_APBX_VERSION.B.MAJOR != 1)
Error();
```

APBX Block v2.1, Revision 1.30

11.5.119

Chapter 12
External Memory Interface (EMI)

This chapter describes the external memory interface (EMI) on the i.MX23. It describes the DRAM controller and EMI power management. Programmable registers for both the DRAM controller are described in Section 12.5, “Programmable Registers.”

12.1 Overview

The i.MX23 supports off-chip DRAM storage via the EMI controller, which is connected to the four internal AHB/AXI busses.

The EMI supports multiple external memory types, including:

- 1.8-V Mobile DDR
- Standard 2.5V DDR1

The DRAM controller supports up to two external chip-select signals for the i.MX23 platform. Programmable registers within the DRAM controller allow great flexibility for device timings, low-power operation, and performance tuning. Note the differences between the two package options:

- The 128-pin LQFP has 1 chip enable. Maximum DRAM supported is 64MB.
- The 169-pin BGA has 2 chip enables. Maximum DRAM supported is 128MB.

The EMI uses two primary clocks: the AHB bus HCLK and the DRAM source clock EMI_CLK. The maximum specified frequencies for these two clocks can be found in Chapter 2, “Characteristics and Specifications. The memory controller operates at frequencies that are asynchronous to the rest of the i.MX23.

The EMI consists of two major components:

- DRAM controller
- Delay compensation circuitry (DCC)

A block diagram of the external memory controller is shown in Figure 12-1.

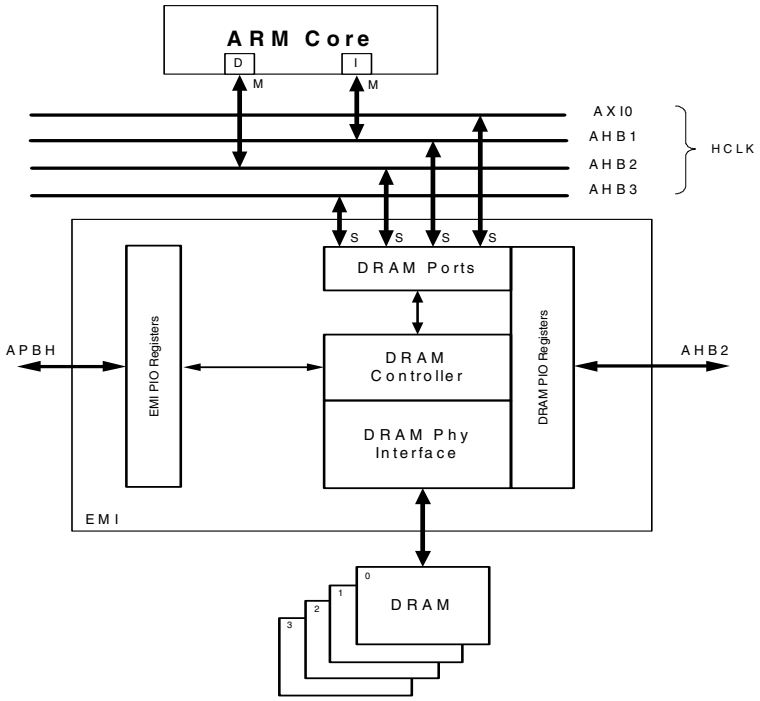


Figure 12-1. External Memory Interface (EMI) Top-Level Block Diagram

12.1.1 AHB Address Ranges

The EMI supports a 512-Mbyte DRAM address space at address 0x40000000. The 512-Mbyte DRAM address space is broken down within the DRAM controller as shown in Figure 12-2.

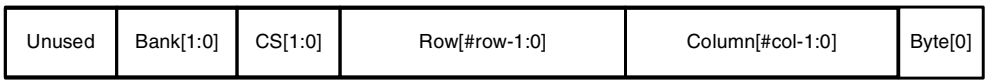


Figure 12-2. DRAM Controller AHB Address Breakdown

Note: This DRAM memory range is not available if the DRAM memory controller is not initialized. A memory access to this range without initializing the DRAM memory controller will result in a system bus hang or a bus error, depending on the state of the TRAP_INIT and TRAP_SR bits in the HW_EMI_CTRL register.

The DRAM controller has programmability to support variously sized DRAM devices. Thus, the number of rows and columns are programmable. In addition, the number of external devices that are in use is programmable, as well. With this organization, the DRAM chips form one large contiguous address space:

$$dram\_memory\_available = 2 * 2^{\#col} * 2^{\#row} * (\#\ dram\_devices) * (\#\ banks\_per\_device)$$

For example, with 10 column bits, 12 row bits, 1 external device and 4 banks per device, the total memory space available would be 32 Mbytes, as follows:

$$2 * 2^{10} * 2^{12} * 1 * 4 = 33{,}554{,}432 \text{ bytes}$$

12.2 DRAM Controller

The DRAM controller handles all of the accesses to the off-chip DRAM devices, including refresh cycles, entry into and exit from low-power modes, and data transfers. This controller supports the following devices:

- 1.8-V mobile DDR
- 2.5-V DDR1

The EMI also supports the connection of simple or multiple external devices with the matrix shown in Table 2-11. See Section 12.6, "EMI Memory Parameters and Register Settings," for configuration examples for DDR and mDDR devices.

The architecture of the DRAM controller is shown in Figure 12-3.

12.2.1 Delay Compensation Circuit (DCC)

The delay compensation circuit (DCC) controls the source-synchronous write and read clocks for data transfer to and from DRAM devices. It is responsible for synchronizing the inbound DRAM data using the DRAM clock (in bypass mode) or the DQS signals. This is done by implementing a series of buffers to delay the clock or DQS signals and then picking the correct tap from the buffer chain to use to sample the data.

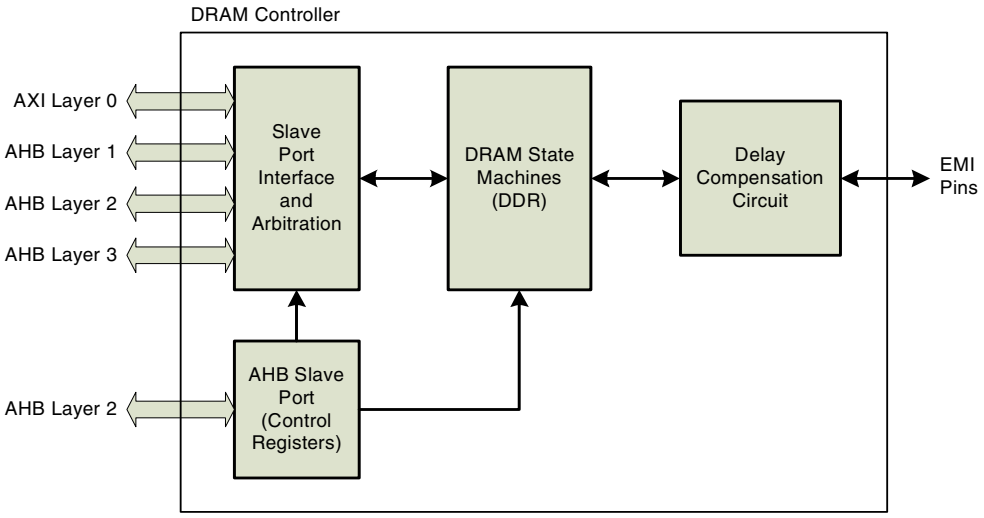


Figure 12-3. DRAM Controller Architecture

12.2.2 Address Mapping

The memory controller automatically maps user addresses to the DRAM memory in a contiguous block. Addressing starts at system address 0x40000000 and extends up to a maximum system address of 0x5FFFFFFF. This allows for a maximum of 512 Mbytes of DRAM storage. This mapping is accomplished by setting certain bit fields in the internal DRAM controller registers.

12.2.2.1 DDR Address Mapping Options

The address structure of DDR devices consists of these fields:

- Datapath
- Column
- Row
- Chip Select
- Bank

The DRAM controller extracts these fields from the lower 30 bits of the system address. The exact bit positions for each field are defined in the programmable registers of the controller. The order of extraction, however, is always fixed as:

Bank-Chip Select-Row-Column-Datapath

The maximum widths of each of these fields are fixed at:

- Bank = 2 bits
- Chip Select = 2 bits
- Row = 13 bits
- Column = 12 bits
- Datapath = 1 bit

The actual width of the column and row fields are programmable using the device address width bit fields (ADDR_PINS and COLUMN_SIZE) in the memory controller.

These maximum values, when combined, define the maximum 512-Mbyte addressable DRAM memory space. Figure 12-4 shows the positioning of the fields within the system address:

Note that practically, the maximum addressable external memory is limited to 128MB for the 169BGA and 64MB for the 128QFP packages. This is because most larger DRAMs require 14 row bits and the EMI controller supports a maximum of 13 row bits.

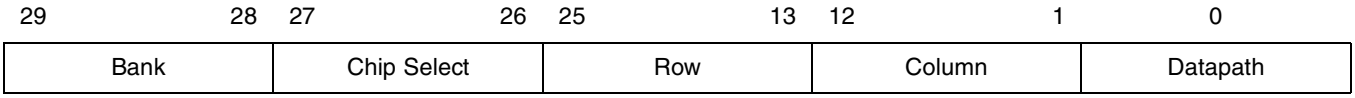


Figure 12-4. Memory Controller Memory Map: Maximum

The ADDR_PINS and COLUMN_SIZE bit fields can each range from their maximum values down to a minimum value defined only by the size of the attached device. This allows the memory controller to function with a wide variety of memory device sizes. The settings for the ADDR_PINS and COLUMN_SIZE bit fields control how the address map is used to decode the user address to the DRAM chip selects and row and column addresses. It is assumed that the values in these bit fields never exceed the maximum values of 13 rows and 12 columns. Using the example shown in Figure 12-4, if the memory

controller is wired to devices with 10 row pins and 11 column bits, the maximum accessible memory space would be reduced. The accessible memory space for this configuration is 64 Mbytes.

The address map for this configuration is shown in Figure 12-5. Note that address bits 26–29 are not used. These bits are ignored when generating the address to the DRAM devices.

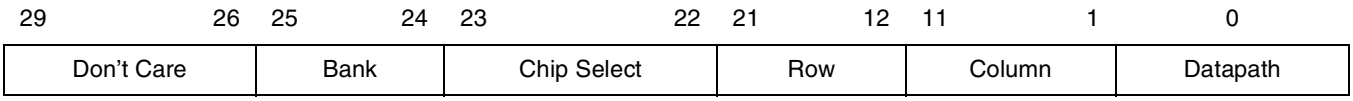


Figure 12-5. Example Memory Map: 10 Row Bits, 11 Column Bits

Note: The Chip Select, Row, Bank, and Column fields are used to address an entire 16-bit memory word. For example, for a read starting at byte address 0x1, the Datapath bit would be a 1 in order to address this byte directly. Reads and writes are 16-bit memory word-aligned if the Datapath bit is 0.

12.2.2.2 Memory Controller Address Control

The available number of accessible memory rows and columns is determined by comparing the maximum values configured with the values programmed into the HW_DRAM_CTL10_ADDR_PINS and HW_DRAM_CTL11_COLUMN_SIZE bit fields. Note that the ADDR_PINS and COLUMN_SIZE bit fields are represented as differences between the maximum configured value and the actual number of pins connected.

The number of connected chip selects and their connection orientation is based on the programming in the HW_DRAM_CTL14_CS_MAP bit field. Because the internal DRAM controller supports up to 4 memory chips, this field is structured to support either one, two, or four memory chips. However, because only 2 memory chip selects are pinned out on the 169BGA package and only 1 memory chip select is pinned out on the 128QFP package, not all CS_MAP configurations can be used.

Below are examples of valid system configurations for the CS_MAP bit field:

- CS_MAP = b0001: One memory device is connected to EMI_CE0n (configuration supported in 128LQFP and 169BGA).
- CS_MAP = b0010: One memory device is connected to EMI_CE1n (configuration supported in 169BGA only).
- CS_MAP = b0011: Two memory devices are connected - one to EMI_CE0n and one to EMI_CE1n. (configuration supported in 169BGA only).

12.2.2.3 Out-of-Range Address Checking

The memory controller is equipped with an out-of-range address checking feature that compares all incoming addresses against the addressable physical memory space. If a transaction is addressed to an out-of-range memory location, then bit 0 of the INT_STATUS bit field is set to 1 to alert the user of this

condition. The memory controller records the address, source ID, length and type of transaction that caused the out-of-range interrupt in the following bit fields:

HW_DRAM_CTL35_OUT_OF_RANGE_ADDR
HW_DRAM_CTL09_OUT_OF_RANGE_SOURCE_ID
HW_DRAM_CTL21_OUT_OF_RANGE_LENGTH
HW_DRAM_CTL09_OUT_OF_RANGE_TYPE

Reading the out-of-range bit fields initiates the memory controller to empty these bit fields and allow them to store out-of-range access information for future errors. The interrupt should be acknowledged by setting bit 0 of the HW_DRAM_CTL16_INT_ACK bit field to 1, which will in turn cause bit 0 of the HW_DRAM_CTL18_INT_STATUS bit field to be cleared to 0.

If a second out-of-range access occurs before the first out-of-range interrupt is acknowledged, then bit 1 of the INT_STATUS bit field is set to 1 to indicate that multiple out-of-range accesses have occurred. If the out-of-range bit fields have been read when the second out-of-range error occurs, then the details for this transaction are stored in the out-of-range bit fields. If they have not been read, then the details of the second error are lost.

Even though the address has been identified as erroneous, the memory controller will still process the read or write transaction. A read transaction will return random data which the user must receive to avoid stalling the memory controller. A write transaction will write the associated data to an unknown location in the memory array, potentially over-writing other stored data. The command cannot be aborted once accepted into the memory controller.

Note that there is no mechanism to indicate an IRQ to the ARM core when this condition occurs. These registers are provided for debugging convenience and can be used with the AHB arbiter debug trap function. To capture an out-of-range error, set an address range with the HW_DIGCTL_DEBUG_TRAP_ADDR_LOW/HIGH registers and enable the trap using HW_DIGCTL_CTRL_TRAP_ENABLE.

12.2.3 Read Data Capture

The read data capture logic is responsible for capturing the DQ outputs from the DRAM devices and passing the data back to the EMI clock domain. The DQS strobes used to capture data are delayed to ensure that the rising and falling edges of the strobes are in the middle of the valid window of data.

DDR (dual data rate) devices send a data strobe (DQS) signal coincident with the read data so that the read data can be reliably captured by the memory controller. The edges of this strobe are aligned with the data output by the DRAM devices. The board traces for the data and the associated data strobe signals should be routed with the same length allowing the rising and falling edges of the data strobe to arrive at the SOC pads.

A delayed version of the data strobe signal must be used to capture the data. The delay added to the data strobe signals should be such that the margin to capture the read data is maximized. Because the frequency of the data strobe signal is matched to the system clock, the delay is a relative number based on the period of the system clock. In the example shown in Figure 12-6, the delay is set to approximately 25% of the system clock. The delay compensation circuit keeps this relative delay constant so that the read data from the DRAM devices can be reliably captured.

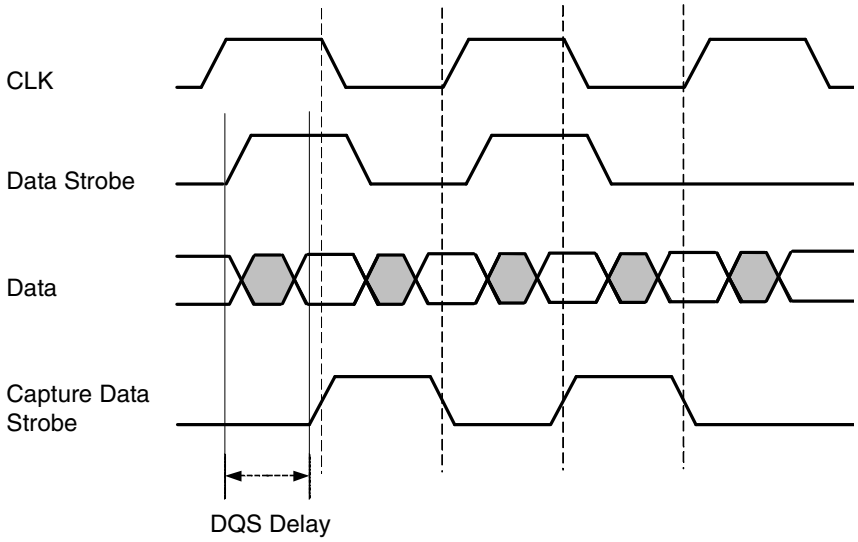


Figure 12-6. DQS Read Timing

12.2.3.1 DQS Gating Control

For the read path, the flight paths must be taken into consideration. There is a certain time lag from when the clock is sent from the memory controller to when the data and DQS signals are received at the memory controller from the memory. Since the DQS from the memory will be sent coincident with the data, and the data must be captured reliably, the DQS signal must be delayed so that it is centered in the data valid window (nominally approximately 1/4 cycle).

The DQS bus is a bidirectional bus that is driven by the memory controller on writes and the memory on reads. When neither device is driving the bus, DQS will remain in a high-impedance state. However, DQS is only relevant to the memory controller during reads in order to capture valid data. For this reason, the DQS signal from memory must be gated so that it is ignored at all other times. Gating of the DQS signal is shown in Figure 12-7.

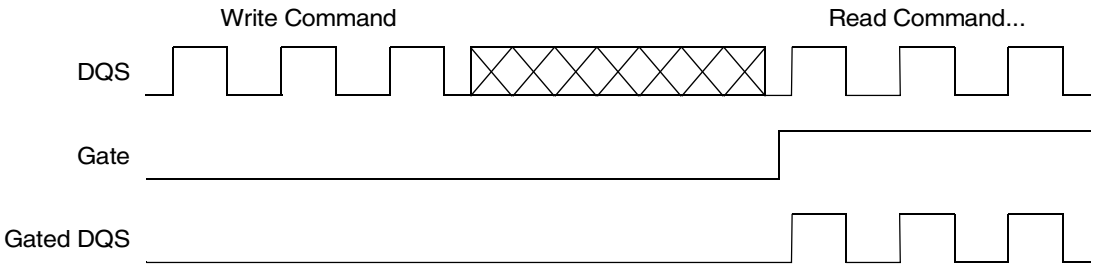


Figure 12-7. DQS Gating

The timing of when to start gating the DQS depends on the design itself, the flight time of the clock to memory, and the flight time of the data/DQS to the memory controller, as follows:

- If the round trip time is between ½ cycle and 1½ cycles, program the *caslat_lin* parameter equal to the *caslat* parameter.
- If the round trip time is less than ½ cycle, program the *caslat_lin* parameter one value less (which translates to ½ cycle) than the *caslat* parameter to open the gate ½ cycle sooner.
- If the round trip time is longer than 1½ cycles, program the *caslat_lin* parameter one value more (which translates to ½ cycle) than the *caslat* parameter to open the gate ½ cycle later.

In addition, the *caslat_lin_gate* parameter controls the opening of the gating signal. Nominally, *caslat_lin_gate* should have the same value as the *caslat_lin* parameter. However, to accommodate the skew of the memory devices, it may be necessary to open the gate a 1/2-cycle sooner or later. Adjusting the value of *caslat_lin_gate* modifies the gate opening by this factor.

There is a requirement that the DQS signals must be known and low when the memory controller is not driving. Because of the large variance in access times for the mobile devices, the gate for the DQS received by the memory controller must be active for longer than the period of time that the memory drives the DQS. Maintaining the DQS bus low when neither the memory controller nor the memory is driving ensures a clean DQS received by the memory controller.

12.2.3.2 mDDR Read Data Timing Registers

When using an mDDR external DRAM device, control of the read data timing is provided through multiple registers, as shown in Figure 12-8. First, the HW_DRAM_CTL04_DLL_BYPASS_MODE selects whether the DCC DLL circuitry is enabled or bypassed. Programming a 1 into this register disables the DLL auto-sync functionality and instead uses a fixed delay-chain select point programmed into the HW_DRAM_CTL19_DLL_DQS_DELAY_BYPASS1 and 0 bit fields. Programming a 0 into the DLL_BYPASS_MODE field enables the DLL auto-sync mode, utilizing the HW_DRAM_CTL18_DLL_DQS_DELAY_BYPASS1 and 0 values to define the percentage of the clock period of delay to add to the DQS inputs before being used as data capture controls.

The BYPASS_MODE or control bit is set based on the desired EMI_CLK frequency. At frequencies above 80 MHz, the BYPASS_MODE should be disabled, allowing the DLL to auto-sync. Frequencies below this point show enable the BYPASS_MODE.

12.2.4 Write Data Timing

DDR DRAM devices require that the DQS data strobe arrive at the DRAM devices within a certain window around the clock. Figure 12-8 describes this relationship. The value for tdqss is specified in fractions of a clock cycle. Most DRAM devices specify this value between ± 0.25 and 0.2 of a clock cycle. This translates to a valid window of between 0.4 and 0.5 of a clock cycle.

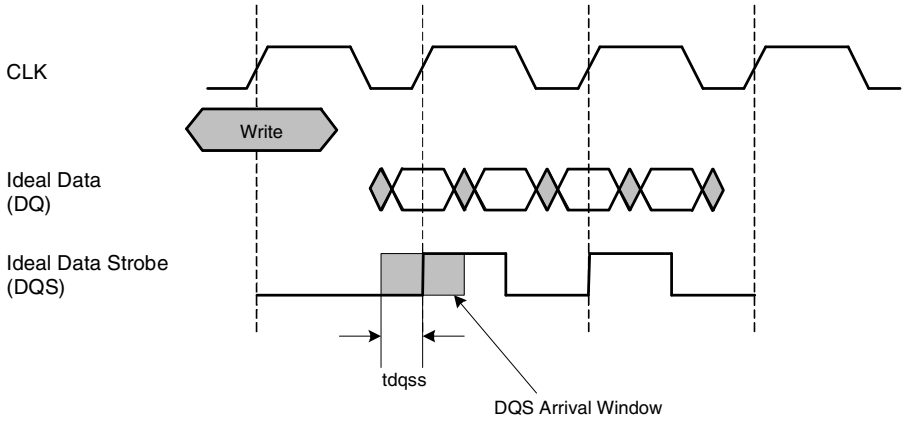


Figure 12-8. DRAM DQS Arrival Time Requirements

The data transfer timing from the memory controller to the DRAM for writes is similar to the read transfer from the DRAM devices to the memory controller. However, there are two differences:

- The DRAM devices expect the data strobe signal to be shifted by the memory controller to allow the DRAM the maximum margin for capturing the data with the data strobe signal.
- The first rising edge of the data strobe signal sent from the memory controller must occur near the rising edge of the clock at the DRAM. This is called the arrival window. DRAM devices typically specify this window as 0.8clk to 1.2clk. Refer to Figure 12-9 for details.

The DCC maintains two delay lines for sending write data and the write data strobe. The first delay line delays the main clock such that the write data strobe transition is as near to the clock edge at the DRAM as possible under typical operating conditions. The second delay line adjusts the clock that is used to output the write data. This clock should be adjusted to maximize the setup and hold requirements around the write strobe.

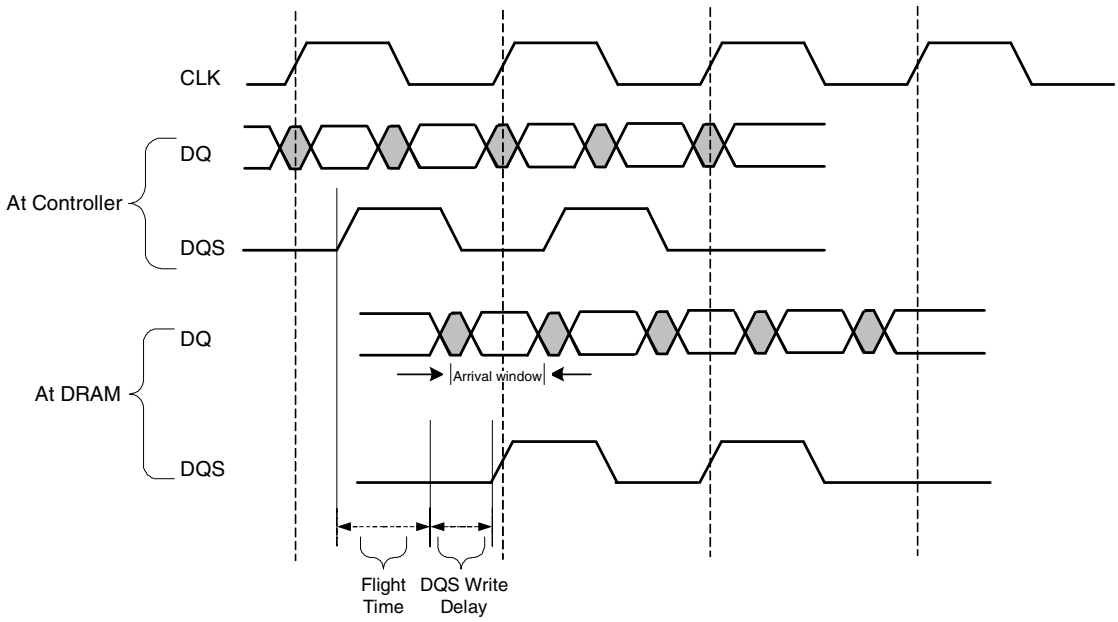


Figure 12-9. DQS Write Timing

Achieving the coincident data strobe signal arrival at a certain point in a clock cycle at the DRAM is a function of the generation of the data strobe signal and the physical delays in transmitting this signal from one point to another. Figure 12-10 illustrates this path in the memory controller.

The write data sent along with the data strobe must be aligned such that the strobe rises and falls within the valid region of the data with maximum setup and hold characteristics. This translates into the write data being clocked 1/4 cycle before the rising edge of the data strobe. This relationship is illustrated in Figure 12-10.

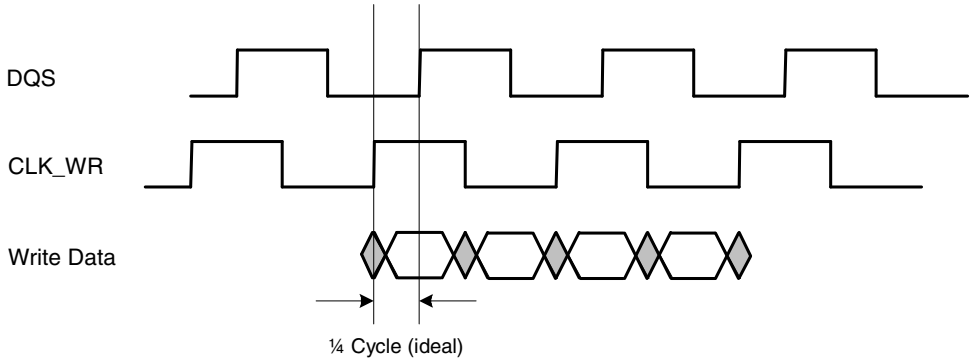


Figure 12-10. Write Data and DQS Relationship

The write data itself originates from a register within the core of the memory controller clocked by the EMI clock. Both the clk_wr and clk_dqs_out signals from the core clock are controlled by the programmable bit fields HW_DRAM_CTL20_WR_DQS_SHIFT and HW_DRAM_CTL19_DQS_OUT_SHIFT,

as shown in Figure 12-11. These bit fields allow these two clocks to be delayed a fixed percentage of the core clock, as illustrated by the example in Figure 12-12.

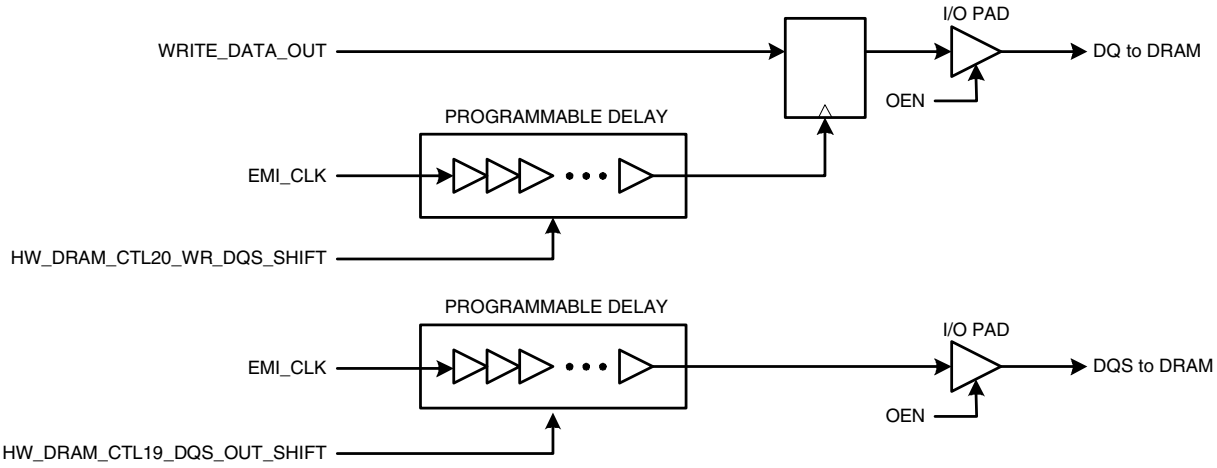


Figure 12-11. Write Data with Programmable Delays

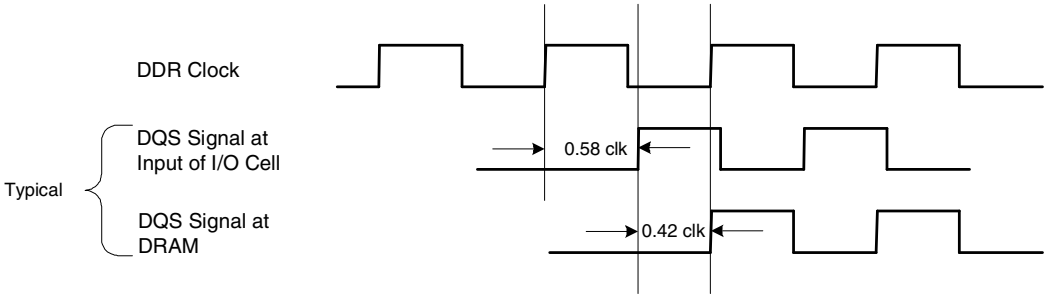


Figure 12-12. WR_DQS_SHIFT Delay Setting Example

12.2.5 DRAM Clock Programmable Delay

The i.MX23 DRAM controller uses an architecture where the address and control signals are launched from the negative edge of the internal EMI clock. The data, DQS, and DM signals are launched from the rising edge of that same clock. At certain higher clock frequencies, this architecture may cause issues with the timing of the signals at the DRAM device relative to the clock itself because the i.MX23 has less flight delay for the clock signals than the address and command signals.

To compensate for this situation, a programmable delay chain is available to delay the output clock to the DRAM device. The delay chain is illustrated in Figure 12-13. This chain consists of 32 delay taps. The delay is voltage-dependent. No other output signals are affected.

The control for this delay is located in the DIGCTL register space in HW_DIGCTL_EMICLK_DELAY_NUM_TAPS. By default, this delay value is 0. In practice, this is not

expected to be used, but it is available as a precaution against high board loads where the address and command signals may not have enough setup time relative to the DRAM clock at the device(s).

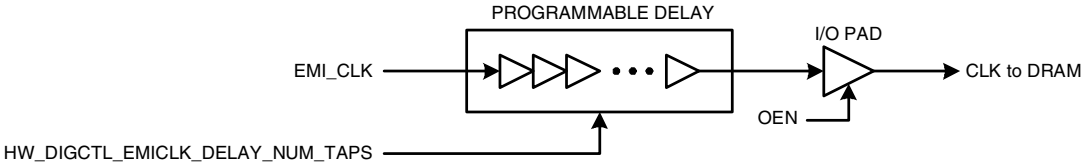


Figure 12-13. DRAM Clock Programmable Delay

12.2.6 Low-Power Operation

In many applications, it is desirable to minimize the power consumption of the memory controller and the memory devices. The memory controller provides various user-configurable low-power options to address power savings. In addition, a partial-array self-refresh option is included for mobile memory devices.

12.2.6.1 Low-Power Modes

There are five low-power modes available in the memory controller. The low-power modes are listed from least to most power saving.

Note: It is not possible to exit one low-power mode and enter another low-power mode simultaneously. The user should plan for a minimum delay between exit and entry between the two low-power modes of 15 cycles in which the memory controller must remain stable.

- **Mode 1: Memory Power-Down**—The memory controller sets the memory devices into power-down, which reduces the overall power consumption of the system, but has the least effect of all the low-power modes. In this mode, the memory controller and memory clocks are fully operational, but the CKE input bit to the memory devices is deasserted. The memory controller continues to monitor memory refresh needs and automatically brings the memory out of power-down to perform these refreshes. When a refresh is required, the CKE input bit to the memory devices is re-enabled. This action brings the memory devices out of power-down. Once the refresh has been completed, the memory devices are returned to power-down by deasserting the CKE input bit.
- **Mode 2: Memory Power-Down with Memory Clock Gating**—The memory controller sets the memory devices into power-down and gates off the clock to the memory devices. Refreshes are handled as in the Memory Power-Down mode (Mode 1), with the exception that gating on the memory clock is removed before asserting the CKE pin. After the refresh has been completed, the memory devices are returned to power-down with the clock gated. Before the memory devices are removed from power-down, the clock is gated on again. Although this mode is supported in both mobile and non-mobile memory devices, clock gating while in power-down is only allowed for mobile memory devices. Therefore, the memory controller will only attempt to gate the clock if it is configured for mobile device operation. For non-mobile memory devices in this low-power

mode, the memory controller operates identically to the Memory Power-Down mode without the clock gating (Mode 1).

- **Mode 3: Memory Self-Refresh**—The memory controller sets the memory devices into self-refresh. In this mode, the memory controller and memory clocks are fully operational and the CKE input bit to the memory devices is deasserted. Since the memory automatically refreshes its contents, the memory controller does not need to send explicit refreshes to the memory.
- **Mode 4: Memory Self-Refresh with Memory Clock Gating**—The memory controller sets the memory devices into self-refresh and gates off the clock to the memory devices. Before the memory devices are removed from self-refresh, the clock is gated on again.
- **Mode 5: Memory Self-Refresh with Memory and Controller Clock Gating**—This is the deepest low-power mode of the memory controller. The memory controller sets the memory devices into self-refresh and gates off the clock to the memory devices. In addition, the clock to the memory controller and the programming bit fields are gated off, except to a small portion of the DLL, which must remain active to maintain the lock. Before the memory devices are removed from self-refresh, the memory controller and memory clocks are gated on.

12.2.6.2 Low-Power Mode Control

The memory controller may enter and exit the various low-power modes in the following ways:

- **Automatic Entry**—When the memory controller is idle, four timing counters begin counting the cycles of inactivity. If any of the counters expires, the memory controller enters the low-power mode associated with that counter.
- **Manual Entry**—The user may initiate any low-power mode by setting the bit of the LOWPOWER_CONTROL bit field associated with the desired mode. The memory controller enters the selected low-power mode when it is has completed its current burst.
- **Hardware Entry**—If the memory pins are being shared between the memory controller and an external source, a handshaking interface is used to control bus activity. The Memory Self-Refresh mode (Mode 3) of the memory controller is used to facilitate the pin sharing.

Automatic and manual entry methods are both controlled by two bit fields: LOWPOWER_CONTROL and LOWPOWER_AUTO_ENABLE located in HW_DRAM_CTL16. The LOWPOWER_CONTROL bit field contains individual enable/disable bits for each low-power mode, and the LOWPOWER_AUTO_ENABLE bit field controls whether each mode is entered automatically or manually.

12.2.6.3 Automatic Entry

Automatic entry occurs if all of the following conditions are true:

- The hardware entry interface is not active or transitioning.
- The mode is programmed for automatic entry by setting the relevant bit in the LOWPOWER_AUTO_ENABLE bit field to 1.
- The particular mode is enabled in the LOWPOWER_CONTROL bit field.
- The memory controller is idle.
- The counter associated with this mode expires.

There are four counters in all to cover the five low-power modes. There are separate counters for each of the three memory self-refresh low-power modes (Modes 3, 4 and 5). Memory Power-Down mode (Mode 1) and Memory Power-Down with Memory Clock Gating mode (Mode 2) share the same counter.

The counters determine the number of idle cycles before entry into the associated low-power mode. All of these counters are re-initialized each time there is a new read or write transaction entering or executing in the memory controller. This ensures that the memory controller does not enter any of the low-power modes when active.

All five low-power modes can be entered through automatic entry and are exited automatically when any of the following conditions occur:

- A new read or write transaction appears at the memory controller interface.
- The memory controller must refresh the memory when in either of the power-down modes (Modes 1 or 2). After completing the memory refresh, the memory controller re-enters power-down.
- The counter for a deeper low-power mode expires. The memory controller must exit the current low-power mode in order to enter the deeper low-power mode. A minimum of 15 cycles occur between exit from one low-power mode before entering into the next low-power mode, even if the counters expire within 15 cycles of each other. Note that the memory controller does not enter a less deep low-power mode, regardless of which counters expire.

12.2.6.4 Manual “On-Demand” Entry

Manual entry occurs if all of the following conditions are true:

- The hardware entry interface is not active or transitioning.
- The mode is programmed for manual entry by clearing the relevant bit in the LOWPOWER_AUTO_ENABLE bit field to 0.
- The particular mode is set to 1 in the LOWPOWER_CONTROL bit field.

For manual entry, the LOWPOWER_CONTROL bit field triggers entry into the low-power modes. The memory controller does not need to be idle when a low-power mode bit is enabled. When a particular mode that is programmed for manual entry is enabled, the memory controller completes the current memory burst access, and then, regardless of the activity inside the memory controller or at the memory interface, it enters the selected low-power mode.

If new transactions enter the memory controller while it is in one of the low-power modes, they accumulate inside the memory controller’s command queue until the queue is full. Exit from a manually-entered low-power mode is also manual. Clearing the LOWPOWER_CONTROL bit field bits to 0 disables the low-power mode of the memory controller, and command processing resumes. In the deepest low-power mode (Mode 5), the clock to the programming registers module is gated off. However, manual low-power mode exit requires the user to clear the LOWPOWER_CONTROL bit field to 0, which is not possible when the clock is off. As a result, the user should not manually activate the deepest low-power mode. If Memory Self-Refresh with Memory and Controller Clock Gating mode (Mode 5) is entered manually, the device cannot be brought out of low-power mode again!

If a different LOWPOWER_CONTROL bit is set to 1 while in one of the low-power modes, or on clearing of the original bit to 0, the memory controller exits the current low-power mode. There will be at least a 15 cycle delay before the memory controller is fully operational or enters the new low-power mode.

NOTE: There is a deadlock possibility that exists when using the manual low-power mode entry. If a read cycle from the ARM core occurs to the DRAM when a manual low-power mode is active, the ARM cycle does not complete. There is no other device within the SOC that can deactivate the low-power mode. Thus, the system will be deadlocked. The same can occur with multiple write cycles that will fill the two-command deep write buffer of the memory controller.

12.2.6.5 Register Programming

The low-power modes of the memory controller are controlled through the LOWPOWER_CONTROL and LOWPOWER_AUTO_ENABLE bit fields in HW_DRAM_CTL16. These five-bit bit fields each contain one bit for controlling each low-power mode. The LOWPOWER_CONTROL bit field enables the associated low-power mode, and the LOWPOWER_AUTO_ENABLE bit field sets the entry method into that mode as manual or automatic. Table 12-1 shows the relationship between the five bits of the lowpower_control and lowpower_auto_enable bit fields and the various low-power modes.

Table 12-1. Low-Power Mode Bit Fields

Low-Power Mode	Enable	Entry
Memory Power-Down (Mode 1)	LOWPOWER_CONTROL [4] =1	LOWPOWER_AUTO_ENABLE [4] • 0 = Manual • 1 = Automatic
Memory Power-Down with Memory Clock Gating (Mode 2)	LOWPOWER_CONTROL [3] =1	LOWPOWER_AUTO_ENABLE [3] • 0 = Manual • 1 = Automatic
Memory Self-Refresh (Mode 3)	LOWPOWER_CONTROL [2] =1	LOWPOWER_AUTO_ENABLE [2] • 0 = Manual • 1 = Automatic
Memory Self-Refresh with Memory Clock Gating (Mode 4)	LOWPOWER_CONTROL [1] =1	LOWPOWER_AUTO_ENABLE [1] • 0 = Manual • 1 = Automatic
Memory Self-Refresh with Memory and Controller Clock Gating (Mode 5)	LOWPOWER_CONTROL [0] =1	LOWPOWER_AUTO_ENABLE [0] • 0 = Manual • 1 = Automatic

When a LOWPOWER_CONTROL bit field bit is set to 1 by the user, the memory controller checks the LOWPOWER_AUTO_ENABLE bit field.

- If the associated bit in the LOWPOWER_AUTO_ENABLE bit field is set to 1, then the memory controller watches the associated counter for expiration, and then enters that low-power mode.Table 12-2 shows the correlation between the low-power modes and the counters that control each mode's automatic entry.

- If the associated bit in the LOWPOWER_AUTO_ENABLE bit field is cleared to 0, then the memory controller completes its current memory burst access and then enters the specified low-power mode.

Table 12-2. Low-Power Mode Counters

Low-Power Mode	Counter
Memory Power-Down (Mode 1)	HW_DRAM_CTL30_LOWPOWER_POWER_DOWN_CNT
Memory Power-Down with Memory Clock Gating (Mode 2)	HW_DRAM_CTL30_ LOWPOWER_POWER_DOWN_CNT
Memory Self-Refresh (Mode 3)	HW_DRAM_CTL31_LOWPOWER_SELF_REFRESH_CNT
Memory Self-Refresh with Memory Clock Gating (Mode 4)	HW_DRAM_CTL29_LOWPOWER_EXTERNAL_CNT
Memory Self-Refresh with Memory and Controller Clock Gating (Mode 5)	HW_DRAM_CTL29_LOWPOWER_INTERNAL_CNT

Note that the values in the LOWPOWER_AUTO_ENABLE bit field are only relevant when the associated LOWPOWER_CONTROL bit is set to 1.

Multiple bits of the LOWPOWER_CONTROL and LOWPOWER_AUTO_ENABLE bit fields can be set to 1 at the same time. When this happens, the memory controller always enters the deepest low-power mode of all the modes that are enabled. If the memory controller is already in one low-power mode when a deeper low-power mode is requested automatically or manually, it must first exit the current low-power mode, and then enter the deeper low-power mode. A minimum 15-cycle delay occurs before the second entry.

The timing for automatic entry into any of the low-power modes is based on the number of idle cycles that have elapsed in the memory controller. There are four counters related to the five low-power modes to determine when any particular low-power mode will be entered if the automatic entry option is chosen. The counters are also shown in Table 12-2. Since the two power-down modes share one counter, if the user wishes to enter Memory Power-Down mode (Mode 1) automatically, then the Memory Power-Down with Memory Clock Gating mode (Mode 2) must not be enabled.

12.2.6.6 Refresh Masking

Regular refresh commands are issued at the same intervals while the memory controller is operating normally, is idle, or is in any of the low-power modes. However, for memory arrays with multiple chip selects, the memory controller supports the ability to mask refreshes while in any of the low-power modes. By clearing bits of the HW_DRAM_CTL14_LOWPOWER_REFRESH_ENABLE bit field to 0, auto-refreshes will be masked for the associated chip selects. It is the user's responsibility to ensure that refreshes are not constantly masked, and that each chip select is refreshed periodically.

12.2.6.7 Mobile DDR Devices

When using a mobile device, the HW_DRAM_CTL05_EN_LOWPOWER_MODE bit field must be set to 1. This enables the memory controller to use the initialization sequence and EMRS addressing appropriate to mobile devices. When the EN_LOWPOWER_MODE bit field is cleared to 0, a standard DDRS-DRAM device may be used.

12.2.6.8 Partial Array Self-Refresh

For mobile devices, the memory controller is capable of supporting refreshes to subsections of the memory array. To facilitate this capability, separate bit fields are provided to supply the EMRS data for each chip select. These are EMRS_DATA_x bit fields, where X represents the chip select.

Having separate control bit fields for the EMRS data allows the individual chips to set their own masked refresh. The WRITE_MODEREG bit field controls the writing of this EMRS data into the registers. When WRITE_MODEREG is set to 1 initially, the EMRS register of chip select 0 will be written. Each subsequent setting of the WRITE_MODEREG bit field to 1 writes the EMRS register of the next chip select (1, 2, then 3).

Note that the memory controller does not check if operations attempt to access addresses outside of the refresh ranges set by the EMRS registers. Any accesses to these addresses may result in corrupt or lost data.

12.2.7 EMI Clock Frequency Change Requirements

Running the EMI block at different operational frequencies involves changing the DRAM controller timing registers and the EMI clock control registers (Chapter 4, “Clock Generation and Control,” for the EMI clock control registers). To change the EMI frequency safely without losing current memory state, the following steps are required:

1. Call code in non-cached, non-buffered OCRAM or ROM (code cannot be executing out of DRAM).
2. Software saves interrupt enable state and disables interrupts.
3. Software flushes instruction and data caches.
4. Software puts DRAM controller in self-refresh mode.
5. Software writes new DRAM controller timing register values (this step is optional, perform if necessary).
6. Software writes new clock frequency.
7. Software polls for EMI clock stability.
8. Software takes DRAM controller out of self-refresh mode.
9. Software restores saved interrupt enable state.
10. Return.

12.3 Power Management

The EMI has multiple levels of power management. Architectural power management is controlled by bits in the programmable registers. The DRAM controller also has automatic engagement of various power-saving modes that are documented in Section 12.2.6, "Low-Power Operation."

The highest level of power-savings in the DRAM controller is achieved by enabling the EMI Clock Gate in the EMI Control register. When enabled, access to all PIO registers except the control register's Soft Reset and Clock Gate bits is disabled, and the DRAM controller is completely shut down. The next step in power savings is the individual DRAM clock gate, which controls the state machines and associated logic. These are also available in the EMI Control Register.

Note: The DRAM control registers have a low-power setting, Level 5, that turns off the DRAM clock inside the controller. It is recommended that Level 5 of the DRAM low-power modes not be used. Instead, use Level 4, which will put the DRAM chip into a self-refresh mode and disable the external clock and CKE. Then, use the EMI Control Register clock gate for the entire EMI or the DRAM-only gate.

The DRAM controller interface, however, has multiple levels of low-power options available. They are, in increasing order of power savings:

1. Memory Power-Down—Controller and EMI_CLK are active, but EMI_CKE is pulled low to the memory devices.
2. Memory Power-Down with Memory Clock Gating—The controller clock remains active, but the EMI_CLK is gated off, and EMI_CKE is pulled low.
3. Memory Self-Refresh—The controller puts the memory devices into self-refresh mode. The controller clock and EMI_CLK remain active, but EMI_CKE is pulled low.
4. Memory Self-Refresh with Memory Clock Gating—The controller puts the memory devices into self-refresh mode. The controller clock remains active, but the EMI_CLK is gated off, and EMI_CKE is pulled low.
5. Memory Self-Refresh with Controller and Memory Clock Gating—The controller puts the memory devices into self-refresh mode. Then, the controller and EMI_CLK are gated off. The only clock that remains active is the clock to the DLL, which must remain active to maintain DLL lock.

The DRAM controller can be programmed to enter these modes automatically, or they can be entered manually via control register accesses. It is expected that Freescale software will set up the DRAM controller to automatically enter modes 1 and 2, but that modes 3–4 would be entered manually after specific requests from the software. Avoid using Level 5.

12.4 AXI/AHB Port Arbitration

The EMI port arbiter supports three operational arbitration modes. The arbiter is provided with PIO control fields, including a two bit HW_EMI_CTRL_ARB_MODE field, which selects one of the three modes. The three arbitration modes are described below.

12.4.1 Legacy Timestamp Mode

When commands are logged into one of the four command queue channels, they are issued a 6-bit sequential count or timestamp. The commands in these four independent channels are then granted access to the downstream controller placement queue in strict timestamp order. The grant decision is simple and is purely combinational in design. Waiting commands can be granted every cycle provided that the placement queue isn't full.

12.4.2 Timestamp/write-priority Hybrid Mode

This new arbitration mode consists of cycling through 2 different priority modes: the legacy timestamp mode (described above) and a new write priority mode. The arbiter first grants a requesting channel based on the timestamp scheme and then goes into the write priority mode. There it loops though all the high priority write channels (3 of the channels can be programmed as high priority write: AXI0, AHB2 and AHB3) a programmed number of iterations granting pending write operations only. It then goes back to timestamp priority mode to grant the operation with the next oldest timestamp. The cycle thus continues alternating between the timestamp and write priority modes. It is important to remember that in the write-priority mode one iteration means to loop through all high priority ports once granting ports with pending commands. And the order in which we consider (or scan through) each port is fixed. The hardware loops through the ports 2, 3 and 0 granting pending commands, in that order. This constitutes one iteration. And this is done repeatedly for the number of iterations programmed. The ordering was set by considering the importance, or priority, of writes of the various masters attached to these busses and therefore is chip specific.

The arbiter receives three 1-bit high priority write masks (HW_EMI_CTRL_HIGH_PRIORITY_WRITE) which select the ports to be given high priority write status. It is also be provided with a 3-bit HP write loop count (HW_EMI_CTRL_PRIORITY_WRITE_ITER), which indicates the maximum number of iterations through the write loop. The write loop will exit when there are no more high priority writes available or when the loop counter reaches the maximum loop count value. The high priority write loop could be skipped if no HP writes are pending. The maximum loop count allowed is 5. These two parameters are PIO programmable.

In practice, software would likely set the ARM data port to have high priority write status and the maximum loop counter would likely be programmed to a value of at least two. This would give highly preferential treatment to ARM data writes and ensure that they get additional commands into the memory controller's placement queue ahead of all other commands. By looping on the writes, we also enable the memory controller to get a stream of write operations that should improve efficiency. Since the arbiter moves back into the timestamp loop periodically, low-priority ports should still have reasonable access to the placement queue. If problems with starvation occur, the maximum loop counter should be programmed to a lower value.

12.4.3 Port Priority Mode

This mode requires the user to program the ports to have a highest to lowest priority. When multiple ports have commands pending, the port with the highest priority will always be granted without regard to timestamp. This mode does not address the problem of possible port starvation.

This mode doesn't inherently guarantee that a read cannot get out of order with a previously issued address-paired write and return stale data. The two previously defined modes guarantee this. However, based on the typical use case, this scenario should only occur on certain port pairs. So if care is taken when programming the port priorities this mode will avoid such errors in this typical case. This mode is very simple, efficient and fully programmable. For this mode it is recommended that the default value be used for the HW_EMI_CTRL_PORT_PRIORITY_ORDER field.

12.5 Programmable Registers

This section describes the programmable registers of the external memory interface (EMI).

12.5.1 EMI Control Register Description

EMI Interface Control Register.

HW_EMI_CTRL	0x000
HW_EMI_CTRL_SET	0x004
HW_EMI_CTRL_CLR	0x008
HW_EMI_CTRL_TOG	0x00C

Table 12-3. HW_EMI_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	RSVD6	TRAP_SR	TRAP_INIT	AXI_DEPTH		DLL_SHIFT_RESET	DLL_RESET	ARB_MODE		RSVD5	PORT_PRIORITY_ORDER					RSVD4	PRIORITY_WRITE_ITER			RSVD3	HIGH_PRIORITY_WRITE			RSVD2	MEM_WIDTH	RSVD1	RESET_OUT	RSVD0			

Table 12-4. HW_EMI_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	Reset EMI register block. 0 = EMI controller is not reset. 1 = EMI controller is reset. Note: This soft reset only affects the EMI registers . There is also a soft-reset for the DRAM controller in the DRAM registers.
30	**RSVD6**	RO	0x0	Reserved
29	**TRAP_SR**	RW	0x0	When set, causes an AHB ERROR response on any access to the DRAM memory space if the DRAM controller is in Self-Refresh mode.
28	**TRAP_INIT**	RW	0x1	When set, causes an AHB ERROR response on any access to the DRAM memory space if the DRAM controller has not been initialized (specifically when START is not set).
27:26	**AXI_DEPTH**	RW	0x3	Specifies the number of commands allowed in the AXI port queue. ONE = 0x0 Allow only one command. TWO = 0x1 Allow two commands. THREE = 0x2 Allow three commands. FOUR = 0x3 Allow four commands.
25	**DLL_SHIFT_RESET**	RW	0x0	When set, forces the DRAM controller DLL startpoint shift logic into a reset state.
24	**DLL_RESET**	RW	0x0	When set, forces the DRAM controller into a reset state.
23:22	**ARB_MODE**	RW	0x0	This field sets the arbitration mode for the DRAM port controller. The supported arbitration schemes are: simple timestamp priority method; enhanced high-priority write and timestamp hibrid method; and port priority method. The programming options are: TIMESTAMP = 0x0 Timestamp Priority (37xx arbitration) WRITE_HYBRID = 0x1 Write Priority Hybrid PORT_PRIORITY = 0x2 Fixed Port Priority
21	**RSVD5**	RO	0x0	Reserved
20:16	**PORT_PRIORITY_ORDER**	RW	0x8	This field specifies the priority order 1-4 (1= highest priority) of the 4 arbitrated ports. The field values define the following order (highest to lowest): (NOTE: Values 0x18-0x1F select PORT1230.) PORT0123 = 0x00 Priority Order: AXI0, AHB1, AHB2, AHB3 PORT0312 = 0x01 Priority Order: AXI0, AHB3, AHB1, AHB2 PORT0231 = 0x02 Priority Order: AXI0, AHB2, AHB3, AHB1 PORT0321 = 0x03 Priority Order: AXI0, AHB3, AHB2, AHB1 PORT0213 = 0x04 Priority Order: AXI0, AHB2, AHB1, AHB3 PORT0132 = 0x05 Priority Order: AXI0, AHB1, AHB3, AHB2 PORT1023 = 0x06 Priority Order: AHB1, AXI0, AHB2, AHB3 PORT1302 = 0x07 Priority Order: AHB1, AHB3, AXI0, AHB2 PORT1230 = 0x08 Priority Order: AHB1, AHB2, AHB3, AXI0 PORT1320 = 0x09 Priority Order: AHB1, AHB3, AHB2, AXI0 PORT1203 = 0x0A Priority Order: AHB1, AHB2, AXI0, AHB3 PORT1032 = 0x0B Priority Order: AHB1, AXI0, AHB3, AHB2 PORT2013 = 0x0C Priority Order: AHB2, AXI0, AHB1, AHB3 PORT2301 = 0x0D Priority Order: AHB2, AHB3, AXI0, AHB1 PORT2130 = 0x0E Priority Order: AHB2, AHB1, AHB3, AXI0 PORT2310 = 0x0F Priority Order: AHB2, AHB3, AHB1, AXI0 PORT2103 = 0x10 Priority Order: AHB2, AHB1, AXI0, AHB3 PORT2031 = 0x11 Priority Order: AHB2, AXI0, AHB3, AHB1 PORT3012 = 0x12 Priority Order: AHB3, AXI0, AHB1, AHB2 PORT3201 = 0x13 Priority Order: AHB3, AHB2, AXI0, AHB1 PORT3120 = 0x14 Priority Order: AHB3, AHB1, AHB2, AXI0 PORT3210 = 0x15 Priority Order: AHB3, AHB2, AHB1, AXI0 PORT3102 = 0x16 Priority Order: AHB3, AHB1, AXI0, AHB2 PORT3021 = 0x17 Priority Order: AHB3, AXI0, AHB2, AHB1

Table 12-4. HW_EMI_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15	**RSVD4**	RO	0x0	Reserved
14:12	**PRIORITY_WRITE_ITER**	RW	0x4	When the hybrid port arbitration scheme is enabled, this field specifies how many times to iterate through the high-priority write phase. This field's range is 1-5 iterations. NOTE: The number of iterations is this field value plus 1.
11	**RSVD3**	RO	0x0	Reserved
10:8	**HIGH_PRIORITY_WRITE**	RW	0x0	Specifies which AHB ports to the EMI have high write priority when the enhanced memory arbitration scheme is enabled. When set bits 12-14 specify high priority for AHB0, AHB2 and AHB3 respectively. The ports are defined as follows: AXI0 = DCP/BCH/PXP port, AHB2 = ARM Data, AHB3 = USB/DMA/ECC8.
7	**RSVD2**	RO	0x0	Reserved
6	**MEM_WIDTH**	RW	0x1	0 = 8-bit memory. 1 = 16-bit memory.
5	**RSVD1**	RO	0x0	Reserved
4	**RESET_OUT**	RW	0x0	0 = Reset output is low. 1 = Reset output is high.
3:0	**RSVD0**	RO	0x0	Reserved

DESCRIPTION:

The EMI Control register is used to control several high-level items related to the EMI controller. This register should be used in conjunction with the DRAM register bits.

EXAMPLE:

```
Empty Example.
```

12.5.2 EMI Version Register Description

This register always returns a known read value for debug purposes. It indicates the version of the block.

HW_EMI_VERSION 0x0F0

Table 12-5. HW_EMI_VERSION

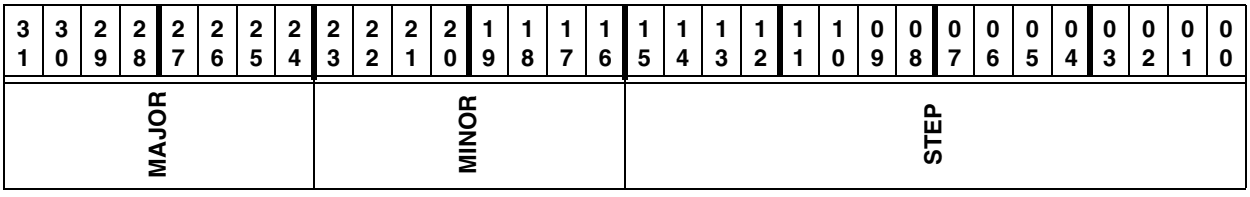


Table 12-6. HW_EMI_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x02	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x01	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
Empty Example.
```

EMI Block v2.1, Revision 1

12.5.3 DRAM Control Register 00 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL00 0x000

Table 12-7. HW_DRAM_CTL00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4							AHB0_W_PRIORITY	RSVD3							AHB0_R_PRIORITY	RSVD2							AHB0_FIFO_TYPE_REG	RSVD1							ADDR_CMP_EN

Table 12-8. HW_DRAM_CTL00 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSVD4**	RO	0x0	Reserved.
24	**AHB0_W_PRIORITY**	RW	0x0	Priority of write commands from port 0. Sets the priority of write commands from AHB port 0 relative to the other AHB Ports. A value of 0 is the highest priority.
23:17	**RSVD3**	RO	0x0	Reserved.
16	**AHB0_R_PRIORITY**	RW	0x0	Priority of read commands from port 0. Sets the priority of read commands from AHB port 0 relative to the other AHB Ports. A value of 0 is the highest priority.

Table 12-8. HW_DRAM_CTL00 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:9	**RSVD2**	RO	0x0	Reserved.
8	**AHB0_FIFO_TYPE_REG**	RW	0x0	Clock domain relativity between port 0 and memory controller core. Sets the relativity of the clock domains between AHB port 0 and the memory controller core clock. 0 = Asynchronous 1 = Synchronous
7:1	**RSVD1**	RO	0x0	Reserved.
0	**ADDR_CMP_EN**	RW	0x0	Enable address collision detection for command queue placement logic. Enables address collision/data coherency detection as a condition when using the placement logic to fill the command queue. 0 = Disabled 1 = Enabled

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.4 DRAM Control Register 01 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL01 0x004

Table 12-9. HW_DRAM_CTL01

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4							AHB2_FIFO_TYPE_REG	RSVD3							AHB1_W_PRIORITY	RSVD2							AHB1_R_PRIORITY	RSVD1							AHB1_FIFO_TYPE_REG

Table 12-10. HW_DRAM_CTL01 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSVD4**	RO	0x0	Reserved.
24	**AHB2_FIFO_TYPE_REG**	RW	0x0	Clock domain relativity between port 2 and memory controller core. Sets the relativity of the clock domains between AHB port 2 and the memory controller core clock. 0 = Asynchronous 1 = Synchronous
23:17	**RSVD3**	RO	0x0	Reserved.
16	**AHB1_W_PRIORITY**	RW	0x0	Priority of write commands from port 1. Sets the priority of write commands from AHB port 1 relative to the other AHB Ports. A value of 0 is the highest priority.
15:9	**RSVD2**	RO	0x0	Reserved.
8	**AHB1_R_PRIORITY**	RW	0x0	Priority of read commands from port 1. Sets the priority of read commands from AHB port 1 relative to the other AHB Ports. A value of 0 is the highest priority.
7:1	**RSVD1**	RO	0x0	Reserved.
0	**AHB1_FIFO_TYPE_REG**	RW	0x0	Clock domain relativity between port 1 and memory controller core. Sets the relativity of the clock domains between AHB port 1 and the memory controller core clock. 0 = Asynchronous 1 = Synchronous

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.5 DRAM Control Register 02 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL02 0x008

Table 12-11. HW_DRAM_CTL02

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4							AHB3_R_PRIORITY	RSVD3							AHB3_FIFO_TYPE_REG	RSVD2							AHB2_W_PRIORITY	RSVD1							AHB2_R_PRIORITY

Table 12-12. HW_DRAM_CTL02 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSVD4**	RO	0x0	Reserved.
24	**AHB3_R_PRIORITY**	RW	0x0	Priority of read commands from port 3. Sets the priority of read commands from AHB port 3 relative to the other AHB Ports. A value of 0 is the highest priority.
23:17	**RSVD3**	RO	0x0	Reserved.
16	**AHB3_FIFO_TYPE_REG**	RW	0x0	Clock domain relativity between port 3 and memory controller core. Sets the relativity of the clock domains between AHB port 3 and the memory controller core clock. 0 = Asynchronous 1 = Synchronous
15:9	**RSVD2**	RO	0x0	Reserved.
8	**AHB2_W_PRIORITY**	RW	0x0	Priority of write commands from port 2. Sets the priority of write commands from AHB port 2 relative to the other AHB Ports. A value of 0 is the highest priority.
7:1	**RSVD1**	RO	0x0	Reserved.
0	**AHB2_R_PRIORITY**	RW	0x0	Priority of read commands from port 2. Sets the priority of read commands from AHB port 2 relative to the other AHB Ports. A value of 0 is the highest priority.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.6 DRAM Control Register 03 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL03 0x00C

Table 12-13. HW_DRAM_CTL03

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4							AUTO_REFRESH_MODE	RSVD3							AREFRESH	RSVD2							AP	RSVD1							AHB3_W_PRIORITY

Table 12-14. HW_DRAM_CTL03 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSVD4**	RO	0x0	Reserved.
24	**AUTO_REFRESH_MODE**	RW	0x0	Controls whether auto-refresh will be at next burst or next command boundary. Sets the mode for when the automatic refresh will occur. If auto_refresh_mode is set and a refresh is required to memory, the memory controller will delay this refresh until the end of the current transaction (if the transaction is fully contained inside a single page), or until the current transaction hits the end of the current page. 0 = Issue refresh on the next DRAM burst boundary, even if the current command is not complete. 1 = Issue refresh on the next command boundary.
23:17	**RSVD3**	RO	0x0	Reserved.
16	**AREFRESH**	WO	0x0	Initiate auto-refresh when specified by AUTO_REFRESH_MODE. Initiates an automatic refresh to the DRAM devices based on the setting of the AUTO_REFRESH_MODE bit field. If there are any open banks when this bit field is set, the memory controller will automatically close these banks before issuing the auto-refresh command. This bit field will always read back 0. 0 = No action 1 = Issue refresh to the DRAM devices
15:9	**RSVD2**	RO	0x0	Reserved.
8	**AP**	RW	0x0	Enable auto pre-charge mode of controller. Enables auto pre-charge mode for DRAM devices. NOTE: This bit field may not be modified after the START bit field has been asserted. 0 = Auto pre-charge mode disabled. Memory banks will stay open until another request requires this bank, the maximum open time (tras_max) has elapsed, or a refresh command closes all the banks. 1 = Auto pre-charge mode enabled. All read and write transactions must be terminated by an auto pre-charge command. If a transaction consists of multiple read or write bursts, only the last command is issued with an auto pre-charge.
7:1	**RSVD1**	RO	0x0	Reserved.
0	**AHB3_W_PRIORITY**	RW	0x0	Priority of write commands from port 3. Sets the priority of write commands from AHB port 3 relative to the other AHB Ports. A value of 0 is the highest priority.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.7 DRAM Control Register 04 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL04 0x010

Table 12-15. HW_DRAM_CTL04

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4							DLL_BYPASS_MODE	RSVD3							DLLLOCKREG	RSVD2							CONCURRENTAP	RSVD1							BANK_SPLIT_EN

Table 12-16. HW_DRAM_CTL04 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSVD4**	RO	0x0	Reserved.
24	**DLL_BYPASS_MODE**	RW	0x0	Enable the DLL bypass feature of the controller. Defines the behavior of the DLL bypass logic and establishes which set of delay parameters will be used. 0 = The values programmed in the DLL_DQS_DELAY_X, DQS_OUT_SHIFT, and WR_DQS_SHIFT are used. These parameters add fractional increments of the clock to the specified lines. 1 = The values programmed into the DLL_DQS_DELAY_BYPASS_X, DQS_OUT_SHIFT_BYPASS, and WR_DQS_SHIFT_BYPASS are used. These parameters specify the actual number of delay elements added to each of the lines. If the total delay time programmed into the delay parameters exceeds the number of delay elements in the delay chain, then the delay will be set to the maximum number of delay elements in the delay chain. 0 = Normal operational atuo-sync mode. 1 = Bypass the auto-sync DLL master delay line.
23:17	**RSVD3**	RO	0x0	Reserved.
16	**DLLLOCKREG**	RO	0x0	Status of DLL lock coming out of master delay. DLL lock/unlock.
15:9	**RSVD2**	RO	0x0	Reserved.
8	**CONCURRENTAP**	RW	0x0	Allow controller to issue commands to other banks while a bank is in auto pre-charge. Enables concurrent auto pre-charge. Some DRAM devices do not allow one bank to be auto pre-charged while another bank is reading or writing. The JEDEC standard allows concurrent auto pre-charge. Set this parameter for the DRAM device being used. 0 = Concurrent auto pre-charge disabled. 1 = Concurrent auto pre-charge enabled.

Table 12-16. HW_DRAM_CTL04 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
7:1	**RSVD1**	RO	0x0	Reserved.
0	**BANK_SPLIT_EN**	RW	0x0	Enable bank splitting for command queue placement logic. Enables bank splitting as a condition when using the placement logic to fill the command queue. 0 = Disabled 1 = Enabled

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.8 DRAM Control Register 05 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL05 0x014

Table 12-17. HW_DRAM_CTL05

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4							INTRPTREADA	RSVD3							INTRPTAPBURST	RSVD2							FAST_WRITE	RSVD1							EN_LOWPOWER_MODE

Table 12-18. HW_DRAM_CTL05 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSVD4**	RO	0x0	Reserved.
24	**INTRPTREADA**	RW	0x0	Allow the controller to interrupt a combined read with auto pre-charge command with another read command. Enables interrupting of a combined read with auto pre-charge command with another read command to the same bank before the first read command is completed. 0 = Disable interrupting the combined read with auto pre-charge command with another read command to the same bank. 1 = Enable interrupting the combined read with auto pre-charge command with another read command to the same bank.

Table 12-18. HW_DRAM_CTL05 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
23:17	**RSVD3**	RO	0x0	Reserved.
16	**INTRPTAPBURST**	RW	0x0	Allow the controller to interrupt an auto pre-charge command with another command. Enables interrupting an auto pre-charge command with another command for a different bank. If enabled, the current operation will be interrupted. However, the bank will be pre-charged as if the current operation were allowed to continue. 0 = Disable interrupting an auto pre-charge operation on a different bank. 1 = Enable interrupting an auto pre-charge operation on a different bank.
15:9	**RSVD2**	RO	0x0	Reserved.
8	**FAST_WRITE**	RW	0x0	Sets when write commands are issued to DRAM devices. Controls when the write commands are issued to the DRAM devices. 0 = The memory controller will issue a write command to the DRAM devices when it has received enough data for one DRAM burst. In this mode, write data can be sent in any cycle relative to the write command. This mode also allows for multi-word write command data to arrive in non-sequential cycles. 1 = The memory controller will issue a write command to the DRAM devices after the first word of the write data is received by the memory controller. The first word can be sent at any time relative to the write command. In this mode, multi-word write command data must be available to the memory controller in sequential cycles.
7:1	**RSVD1**	RO	0x0	Reserved.
0	**EN_LOWPOWER_MODE**	RW	0x0	Enable low-power mode in controller. Enables the low-power mode of the memory controller. 0 = Disabled 1 = Enabled

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.9 DRAM Control Register 06 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL06 0x018

Table 12-19. HW_DRAM_CTL06

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4							POWER_DOWN	RSVD3							PLACEMENT_EN	RSVD2							NO_CMD_INIT	RSVD1							INTRPTWRITEA

Table 12-20. HW_DRAM_CTL06 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSVD4**	RO	0x0	Reserved.
24	**POWER_DOWN**	RW	0x0	Disable clock enable and set DRAMs in power-down state. When this bit field is written with a 1, the memory controller will complete processing of the current burst for the current transaction (if any), issue a pre-charge all command and then disable the clock enable signal to the DRAM devices. Any subsequent commands in the command queue will be suspended until this bit field is written with a 0. 0 = Enable full power state. 1 = Disable the clock enable and power down the memory controller.
23:17	**RSVD3**	RO	0x0	Reserved.
16	**PLACEMENT_EN**	RW	0x0	Enable placement logic for command queue. Enables using the placement logic to fill the command queue. 0 = Placement logic is disabled. The command queue is a straight FIFO. 1 = Placement logic is enabled. The command queue will be filled according to the placement logic factors.
15:9	**RSVD2**	RO	0x0	Reserved.
8	**NO_CMD_INIT**	RW	0x0	Disable DRAM commands until TDLL has expired during initialization. Disables DRAM commands until DLL initialization is complete and tdll has expired. 0 = Issue only REF and PRE commands during DLL initialization of the DRAM devices. 1 = Do not issue any type of command during DLL initialization of the DRAM devices.
7:1	**RSVD1**	RO	0x0	Reserved.
0	**INTRPTWRITEA**	RW	0x0	Allow the controller to interrupt a combined write with auto pre-charge command with another write command. Enables interrupting of a combined write with auto pre-charge command with another read or write command to the same bank before the first write command is completed. 0 = Disable interrupting a combined write with auto pre-charge command with another read or write command to the same bank. 1 = Enable interrupting a combined write with auto pre-charge command with another read or write command to the same bank.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.10 DRAM Control Register 07 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL07 0x01C

Table 12-21. HW_DRAM_CTL07

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4							RW_SAME_EN	RSVD3							REG_DIMM_ENABLE	RSVD2							RD2RD_TURN	RSVD1							PRIORITY_EN

Table 12-22. HW_DRAM_CTL07 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSVD4**	RO	0x0	Reserved.
24	**RW_SAME_EN**	RW	0x0	Enable read/write grouping for command queue placement logic. Enables read/write grouping as a condition when using the placement logic to fill the command queue. 0 = Disabled 1 = Enabled
23:17	**RSVD3**	RO	0x0	Reserved.
16	**REG_DIMM_ENABLE**	RW	0x0	Enable registered DIMM operation of the controller. Enables registered DIMM operations to control the address and command pipeline of the memory controller. 0 = Normal operation 1 = Enable registered DIMM operation
15:9	**RSVD2**	RO	0x0	Reserved.

Table 12-22. HW_DRAM_CTL07 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
8	**RD2RD_TURN**	RW	0x0	Enable insertion of addition turn around clock for back to back reads to different css. Adds an additional clock between back-to-back read operations. The extra clock is required for mobile DDR devices where tac_max > (period/2 + tac_min). Without this additional clock, the first read may drive DQS out at tac_max and the second read may drive DQS out at tac_min, resulting in a contention on the DQS line. 0 = Disabled 1 = Enabled
7:1	**RSVD1**	RO	0x0	Reserved.
0	**PRIORITY_EN**	RW	0x0	Enable priority for command queue placement logic. Enables priority as a condition when using the placement logic to fill the command queue. 0 = Disabled 1 = Enabled

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.11 DRAM Control Register 08 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL08 0x020

Table 12-23. HW_DRAM_CTL08

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4							TRAS_LOCKOUT	RSVD3							START	RSVD2							SREFRESH	RSVD1							SDR_MODE

Table 12-24. HW_DRAM_CTL08 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSVD4**	RO	0x0	Reserved.
24	**TRAS_LOCKOUT**	RW	0x0	Allow the controller to execute auto pre-charge commands before TRAS_MIN expires. Defines the tRAS lockout setting for the DRAM device. tRAS lockout allows the memory controller to execute auto pre-charge commands before the TRAS_MIN parameter has expired. 0 = tRAS lockout not supported by memory device. 1 = tRAS lockout supported by memory device.
23:17	**RSVD3**	RO	0x0	Reserved.
16	**START**	RW	0x0	Initiate command processing in the controller. With this bit field cleared to 0, the memory controller will not issue any commands to the DRAM devices or respond to any signal activity except for reading and writing bit fields. Once this bit field is set to 1, the memory controller will respond to inputs from the ASIC. When set, the memory controller begins its initialization routine. When the interrupt bit in the INT_STATUS bit field associated with completed initialization is set, the user may begin to submit transactions. 0 = Controller is not in active mode. 1 = Initiate active mode for the memory controller.
15:9	**RSVD2**	RO	0x0	Reserved.
8	**SREFRESH**	RW	0x0	Place DRAMs in self-refresh mode. When this bit field is written with a 1, the DRAM device(s) will be placed in self-refresh mode. For this, the current burst for the current transaction (if any) will complete, all banks will be closed, the self-refresh command will be issued to the DRAM, and the clock enable signal will be de-asserted. The system will remain in self-refresh mode until this bit field is written with a 0. The DRAM devices will return to normal operating mode after the self-refresh exit time (txsr) of the device and any DLL initialization time for the DRAM is reached. The memory controller will resume processing of the commands from the interruption point. This bit field will be updated with an assertion of the srefresh_enter pin, regardless of the behavior on the register interface. To disable self-refresh again after a srefresh_enter pin assertion, the user will need to clear the bit field to 0. 0 = Disable self-refresh mode. 1 = Initiate self-refresh of the DRAM devices.
7:1	**RSVD1**	RO	0x0	Reserved.
0	**SDR_MODE**	RW	0x0	Select SDR or DDR mode of the controller. Selects between SDR (single data rate) and DDR (dual data rate) modes. 0 = DDR mode 1 = SDR mode

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.12 DRAM Control Register 09 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL09 0x024

Table 12-25. HW_DRAM_CTL09

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4						OUT_OF_RANGE_TYPE		RSVD3						OUT_OF_RANGE_SOURCE_ID		RSVD2							WRITE_MODEREG	RSVD1							WRITEINTERP

Table 12-26. HW_DRAM_CTL09 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSVD4**	RO	0x0	Reserved.
25:24	**OUT_OF_RANGE_TYPE**	RO	0x0	Type of command that caused an Out-of-Range interrupt. Holds the type of command that caused an out-of-range interrupt request to the memory devices.
23:18	**RSVD3**	RO	0x0	Reserved.
17:16	**OUT_OF_RANGE_SOURCE_ID**	RO	0x0	Source ID of command that caused an Out-of-Range interrupt. Holds the Source ID of the command that caused an out-of-range interrupt request to the memory devices.
15:9	**RSVD2**	RO	0x0	Reserved.
8	**WRITE_MODEREG**	WO	0x0	Write EMRS data to the DRAMs. Supplies the EMRS data for each chip select to allow individual chips to set masked refreshing. When this bit field is written with a 1, the mode bit field(s) [EMRS register] within the DRAM devices will be written. Each subsequent write_modereg setting will write the EMRS register of the next chip select. This bit field will always read back as 0. The mode registers are automatically written at initialization of the memory controller. There is no need to initiate a mode register write after setting the START bit field in the memory controller unless some value in these registers needs to be changed after initialization. Note: This bit field may not be changed when the memory is in power-down mode (when the CKE input is de-asserted).

Table 12-26. HW_DRAM_CTL09 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
7:1	**RSVD1**	RO	0x0	Reserved.
0	**WRITEINTERP**	RW	0x0	Allow controller to interrupt a write bursts to the DRAMs with a read command. Defines whether the memory controller can interrupt a write burst with a read command. Some memory devices do not allow this functionality. 0 = The device does not support read commands interrupting write commands. 1 = The device does support read commands interrupting write commands.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.13 DRAM Control Register 10 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL10 0x028

Table 12-27. HW_DRAM_CTL10

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4					AGE_COUNT			RSVD3					ADDR_PINS			RSVD2						TEMRS		RSVD1							Q_FULLNESS	

Table 12-28. HW_DRAM_CTL10 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:27	**RSVD4**	RO	0x0	Reserved.
26:24	**AGE_COUNT**	RW	0x0	Initial value of master aging-rate counter for command aging. Holds the initial value of the master aging-rate counter. When using the placement logic to fill the command queue, the command aging counters will be decremented one each time the master aging-rate counter counts down age_count cycles.
23:19	**RSVD3**	RO	0x0	Reserved.

Table 12-28. HW_DRAM_CTL10 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
18:16	**ADDR_PINS**	RW	0x0	Difference between number of address pins available and number being used. Defines the difference between the maximum number of address pins configured (13) and the actual number of pins being used. The user address is automatically shifted so that the user address space is mapped contiguously into the memory map based on the value of this bit field.
15:10	**RSVD2**	RO	0x0	Reserved.
9:8	**TEMRS**	RW	0x0	DRAM TEMRS parameter in cycles. Defines the DRAM extended mode parameter set time, in cycles.
7:2	**RSVD1**	RO	0x0	Reserved.
1:0	**Q_FULLNESS**	RW	0x0	Quantity that determines command queue full. Defines quantity of data that will be considered full for the command queue.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.14 DRAM Control Register 11 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL11 0x02C

Table 12-29. HW_DRAM_CTL11

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4					MAX_CS_REG			RSVD3					COMMAND_AGE_COUNT			RSVD2					COLUMN_SIZE			RSVD1					CASLAT		

Table 12-30. HW_DRAM_CTL11 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:27	**RSVD4**	RO	0x0	Reserved.
26:24	**MAX_CS_REG**	RO	0x4	Maximum number of chip selects available. Defines the maximum number of chip selects for the memory controller as the log2 of the number of chip selects.

Table 12-30. HW_DRAM_CTL11 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
23:19	**RSVD3**	RO	0x0	Reserved.
18:16	**COMMAND_AGE_COUNT**	RW	0x0	Initial value of individual command aging counters for command aging. Holds the initial value of the command aging counters associated with each command in the command queue. When using the placement logic to fill the command queue, the command aging counters decrement one each time the master aging-rate counter counts down age_count cycles.
15:11	**RSVD2**	RO	0x0	Reserved.
10:8	**COLUMN_SIZE**	RW	0x0	Difference between number of column pins available and number being used. Shows the difference between the maximum column width available (12) and the actual number of column pins being used. The user address is automatically shifted so that the user address space is mapped contiguously into the memory map based on the value of this bit field.
7:3	**RSVD1**	RO	0x0	Reserved.
2:0	**CASLAT**	RW	0x0	Encoded CAS latency sent to DRAMs during initialization. Sets the CAS (Column Address Strobe) latency encoding that the memory uses. The binary value programmed into this bit field is dependent on the memory device, since the same caslat value may have different meanings to different memories. This will be programmed into the DRAM devices at initialization. The CAS encoding will be specified in the DRAM spec sheet, and should correspond to the CASLAT_LIN bit field.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.15 DRAM Control Register 12 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL12 0x030

Table 12-31. HW_DRAM_CTL12

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD3					TWR_INT			RSVD2					TRRD			OBSOLETE								RSVD1					TCKE		

Table 12-32. HW_DRAM_CTL12 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:27	**RSVD3**	RO	0x0	Reserved.
26:24	**TWR_INT**	RW	0x0	DRAM TWR parameter in cycles. Defines the DRAM write recovery time, in cycles.
23:19	**RSVD2**	RO	0x0	Reserved.
18:16	**TRRD**	RW	0x0	DRAM TRRD parameter in cycles. Defines the DRAM activate to activate delay for different banks, in cycles.
15:8	**OBSOLETE**	RO	0x0	Reserved.
7:3	**RSVD1**	RO	0x0	Reserved.
2:0	**TCKE**	RW	0x0	Minimum CKE pulse width. Defines the minimum CKE pulse width, in cycles.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.16 DRAM Control Register 13 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL13 0x034

Table 12-33. HW_DRAM_CTL13

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4				CASLAT_LIN_GATE				RSVD3				CASLAT_LIN				RSVD2				APREBIT				RSVD1				TWTR			

Table 12-34. HW_DRAM_CTL13 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**RSVD4**	RO	0x0	Reserved.
27:24	**CASLAT_LIN_GATE**	RW	0x0	Adjusts data capture gate open by half cycles. Adjusts the data capture gate open time by 1/2 cycle increments. This bit field is programmed differently than CASLAT_LIN when there are fixed offsets in the flight path between the memories and the memory controller for clock gating. When CASLAT_LIN_GATE is a larger value than CASLAT_LIN, the data capture window will become shorter. A CASLAT_LIN_GATE value smaller than CASLAT_LIN may have no effect on the data capture window, depending on the fixed offsets in the ASIC and the board. 0000 - 0010 = Reserved 0011 = 1.5 cycles 0100 = 2 cycles 0101 = 2.5 cycles 0110 = 3 cycles 0111 = 3.5 cycles 1000 = 4 cycles 1001 = Reserved 1010 = 5 cycles All other settings are Reserved
23:20	**RSVD3**	RO	0x0	Reserved.
19:16	**CASLAT_LIN**	RW	0x0	Sets latency from read command send to data receive from/to controller. Sets the CAS latency linear value in 1/2 cycle increments. This sets an internal adjustment for the delay from when the read command is sent from the memory controller to when data will be received back. The window of time in which the data is captured is a fixed length. The CASLAT_LIN bit field adjusts the start of this data capture window. Note: Not all linear values will be supported for the memory devices being used. Refer to the specification for the memory devices being used. 0000 - 0010 = Reserved 0011 = 1.5 cycles 0100 = 2 cycles 0101 = 2.5 cycles 0110 = 3 cycles 0111 = 3.5 cycles 1000 = 4 cycles 1001 = Reserved 1010 = 5 cycles All other settings are reserved
15:12	**RSVD2**	RO	0x0	Reserved.
11:8	**APREBIT**	RW	0x0	Location of the auto pre-charge bit in the DRAM address. Defines the location of the auto pre-charge bit in the DRAM address in decimal encoding.
7:3	**RSVD1**	RO	0x0	Reserved.
2:0	**TWTR**	RW	0x0	DRAM TWTR parameter in cycles. Sets the number of cycles needed to switch from a write to a read operation, as dictated by the DDR SDRAM specification.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.17 DRAM Control Register 14 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL14 0x038

Table 12-35. HW_DRAM_CTL14

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4				MAX_COL_REG				RSVD3				LOWPOWER_REFRESH_ENABLE				RSVD2				INITAREF				RSVD1				CS_MAP			

Table 12-36. HW_DRAM_CTL14 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**RSVD4**	RO	0x0	Reserved.
27:24	**MAX_COL_REG**	RO	0xd	Maximum width of column address in DRAMs. Defines the maximum width of column address in the DRAM devices. This value can be used to set the COLUMN_SIZE bit field. column_size = max_col_reg - <number of column bits in memory device>.
23:20	**RSVD3**	RO	0x0	Reserved.
19:16	**LOWPOWER_REFRESH_ENABLE**	RW	0x0	Enable refreshes during power down. Enables refreshes during power-down mode. 0 = Disabled 1 = Enabled
15:12	**RSVD2**	RO	0x0	Reserved.
11:8	**INITAREF**	RW	0x0	Number of auto-refresh commands to execute during DRAM initialization. Defines the number of auto-refresh commands needed by the DRAM devices to satisfy the initialization sequence.

Table 12-36. HW_DRAM_CTL14 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
7:4	**RSVD1**	RO	0x0	Reserved.
3:0	**CS_MAP**	RW	0x0	Sets the mask that determines which chip select pins are active. The user address chip select field will be mapped into the active chip selects indicated by this bit field in ascending order from lowest to highest. This allows the memory controller to map the entire contiguous user address into any group of chip selects. Bit 0 of this bit field corresponds to chip select [0]. Note that the number of chip selects, the number of bits set to 1 in this bit field, must be a power of 2 (2 raised to power of 0, 2 raised to power of 1, 2 raised to power of 2, etc.). NOTE: On the 169-pin BGA package, bits [3:2] of CS_MAP should always be 0. On the 128-pin LQFP, bits [3:1] of CS_MAP should always be 0.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.18 DRAM Control Register 15 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL15 0x03C

Table 12-37. HW_DRAM_CTL15

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4				TRP				RSVD3				TDAL				RSVD2				PORT_BUSY				RSVD1				MAX_ROW_REG			

Table 12-38. HW_DRAM_CTL15 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**RSVD4**	RO	0x0	Reserved.
27:24	**TRP**	RW	0x0	DRAM TRP parameter in cycles. Defines the DRAM pre-charge command time, in cycles.
23:20	**RSVD3**	RO	0x0	Reserved.

Table 12-38. HW_DRAM_CTL15 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:16	**TDAL**	RW	0x0	DRAM TDAL parameter in cycles. Defines the auto pre-charge write recovery time when auto pre-charge is enabled (ap is set), in cycles. This is defined internally as tRP (pre-charge time) + auto pre-charge write recovery time. Note that not all memories use this parameter. If tDAL is defined in the memory specification, then program this bit field to the specified value. If the memory does not specify a tDAL time, then program this bit field to tWR + tRP. DO NOT program this bit field with a value of 0x0 or the memory controller will not function properly when auto pre-charge is enabled.
15:12	**RSVD2**	RO	0x0	Reserved.
11:8	**PORT_BUSY**	RO	0x0	Per-port indicator that the controller is processing a command. Indicates that a port is actively processing a command. Each bit controls the corresponding port. 0 = Port is not busy. 1 = Port is busy.
7:4	**RSVD1**	RO	0x0	Reserved.
3:0	**MAX_ROW_REG**	RO	0xd	Maximum width of memory address bus. Defines the maximum width of the memory address bus (number of row bits) for the memory controller. This value can be used to set the ADDR_PINS bit field. ADDR_PINS = MAX_ROW_REG - <number of row bits in memory device>.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.19 DRAM Control Register 16 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL16 0x040

Table 12-39. HW_DRAM_CTL16

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4			TMRD					RSVD3			LOWPOWER_CONTROL					RSVD2			LOWPOWER_AUTO_ENABLE				RSVD1				INT_ACK			

Table 12-40. HW_DRAM_CTL16 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSVD4**	RO	0x0	Reserved.
28:24	**TMRD**	RW	0x00	DRAM TMRD parameter in cycles. Defines the DRAM mode register set command time, in cycles.
23:21	**RSVD3**	RO	0x0	Reserved.
20:16	**LOWPOWER_CONTROL**	RW	0x00	Controls entry into the low-power modes. Enables the individual low-power modes of the device. Bit 0 = Controls memory self-refresh with memory and controller clock gating mode (Mode 5). Reserved and should always be written to a 0. Gate the clock via the CLKCTRL clock-gate for the EMI instead. Bit 1 = Controls memory self-refresh with memory clock gating mode (Mode 4). Bit 2 = Controls memory self-refresh mode (Mode 3). Bit 3 = Controls memory power-down with memory clock gating mode (Mode 2). Bit 4 = Controls memory power-down mode (Mode 1). For all bits: 0 = Disabled. 1 = Enabled.
15:13	**RSVD2**	RO	0x0	Reserved.

Table 12-40. HW_DRAM_CTL16 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
12:8	**LOWPOWER_AUTO_ENABLE**	RW	0x00	Enables automatic entry into the low-power mode on idle. Enables automatic entry into the low-power modes of the memory controller. Bit 0 = Controls memory self-refresh with memory and controller clock gating mode (Mode 5). Reserved and should always be written to a 0. Gate the clock via the CLKCTRL clock-gate for the EMI instead. Bit 1 = Controls memory self-refresh with memory clock gating mode (Mode 4). Reserved and should always be written to a 0. Bit 2 = Controls memory self-refresh mode (Mode 3). Reserved and should always be written to a 0. Bit 3 = Controls memory power-down with memory clock gating mode (Mode 2). Bit 4 = Controls memory power-down mode (Mode 1). For all bits: 0 = Automatic entry into this mode is disabled. The user may enter this mode manually by setting the associated lowpower_control bit. 1 = Automatic entry into this mode is enabled. The mode will be entered automatically when the proper counters expire, and only if the associated lowpower_control bit is set.
7:4	**RSVD1**	RO	0x0	Reserved.
3:0	**INT_ACK**	WO	0x0	Clear mask of the INT_STATUS bit field. Controls the clearing of the INT_STATUS bit field. If any of the INT_ACK bits are set to a 1 the corresponding bit in the INT_STATUS bit field will be cleared to 0. Any INT_ACK bits written with a 0 will not alter the corresponding bit in the INT_STATUS bit field. This bit field will always read back as 0.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.20 DRAM Control Register 17 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL17 0x044

Table 12-41. HW_DRAM_CTL17

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DLL_START_POINT								DLL_LOCK								DLL_INCREMENT								RSVD1				TRC			

Table 12-42. HW_DRAM_CTL17 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**DLL_START_POINT**	RW	0x00	Initial delay count when searching for lock in master DLL. Sets the number of delay elements to place in the master delay line to start searching for lock in master DLL.
23:16	**DLL_LOCK**	RO	0x00	Number of delay elements in master DLL lock. Defines the actual number of delay elements used to capture one full clock cycle. This bit field is automatically updated every time a refresh operation is performed.
15:8	**DLL_INCREMENT**	RW	0x00	Number of elements to add to DLL_START_POINT when searching for lock. Defines the number of delay elements to recursively increment the DLL_START_POINT bit field with when searching for lock.
7:5	**RSVD1**	RO	0x0	Reserved.
4:0	**TRC**	RW	0x00	DRAM TRC parameter in cycles. Defines the DRAM period between active commands for the same bank, in cycles.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.21 DRAM Control Register 18 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL18 0x048

Table 12-43. HW_DRAM_CTL18

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4	DLL_DQS_DELAY_1							RSVD3	DLL_DQS_DELAY_0							RSVD2			INT_STATUS					RSVD1			INT_MASK				

Table 12-44. HW_DRAM_CTL18 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSVD4**	RO	0x0	Reserved.
30:24	**DLL_DQS_DELAY_1**	RW	0x00	Fraction of a cycle to delay the dqs signal from the DRAMs for dll_rd_dqs_slice 1 during reads. Sets the delay for the read_dqs signal from the DDR SDRAM devices for dll_rd_dqs_slice 1. This delay is used center the edges of the read_dqs signal so that the read data will be captured in the middle of the valid window in the I/O logic.Each increment of this bit field adds a delay of 1/128 of the system clock.
23	**RSVD3**	RO	0x0	Reserved.
22:16	**DLL_DQS_DELAY_0**	RW	0x00	Fraction of a cycle to delay the dqs signal from the DRAMs for dll_rd_dqs_slice 0 during reads. Sets the delay for the read_dqs signal from the DDR SDRAM devices for dll_rd_dqs_slice 0. This delay is used center the edges of the read_dqs signal so that the read data will be captured in the middle of the valid window in the I/O logic.Each increment of this bit field adds a delay of 1/128 of the system clock.
15:13	**RSVD2**	RO	0x0	Reserved.
12:8	**INT_STATUS**	RO	0x00	Status of interrupt features in the controller. Shows the status of all possible interrupts generated by the memory controller. The MSB is the result of a logical OR of all the lower bits. The INT_STATUS bits correspond to these interrupts: Bit 0 = A single access outside the defined PHYSICAL memory space detected. Bit 1 = Multiple accesses outside the defined PHYSICAL memory space detected. Bit 2 = DRAM initialization complete. Bit 3 = DLL unlock condition detected. Bit 4 = Logical OR of all lower bits.
7:5	**RSVD1**	RO	0x0	Reserved.
4:0	**INT_MASK**	RW	0x00	Mask for controller_int signals from the INT_STATUS bit field. Active-high mask bits that control the value of the memory controller_int signal on the ASIC interface. This mask is inverted and then logically AND'ed with the outputs of the INT_STATUS bit field.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.22 DRAM Control Register 19 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL19 0x04C

Table 12-45. HW_DRAM_CTL19

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DQS_OUT_SHIFT_BYPASS								RSVD1	DQS_OUT_SHIFT							DLL_DQS_DELAY_BYPASS_1								DLL_DQS_DELAY_BYPASS_0							

Table 12-46. HW_DRAM_CTL19 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**DQS_OUT_SHIFT_BYPASS**	RW	0x00	Sets the delay for the clk_dqs_out signal of the dll_wr_dqs_slice when the DLL is being bypassed. This is used to ensure correct data capture in the I/O logic. The value programmed into this bit field sets the actual number of delay elements in the clk_dqs_out line. If the total delay time programmed exceeds the number of delay elements in the delay chain, then the delay will be set internally to the maximum number of delay elements available.
23	**RSVD1**	RO	0x0	Reserved.
22:16	**DQS_OUT_SHIFT**	RW	0x00	Sets the delay for the clk_dqs_out signal of the dll_wr_dqs_slice to ensure correct data capture in the I/O logic. Each increment of this bit field adds a delay of 1/128 of the system clock.

Table 12-46. HW_DRAM_CTL19 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:8	**DLL_DQS_DELAY_BYPASS_1**	RW	0x00	Sets the delay for the read_dqs signal from the DDR SDRAM devices for dll_rd_dqs_slice 1 for reads when the DLL is being bypassed. This delay is used center the edges of the read_dqs signal so that the read data will be captured in the middle of the valid window in the I/O logic. The value programmed into this bit field sets the actual number of delay elements in the read_dqs line. If the total delay time programmed exceeds the number of delay elements in the delay chain, then the delay will be set internally to the maximum number of delay elements available.
7:0	**DLL_DQS_DELAY_BYPASS_0**	RW	0x00	Sets the delay for the read_dqs signal from the DDR SDRAM devices for dll_rd_dqs_slice 0 for reads when the DLL is being bypassed. This delay is used center the edges of the read_dqs signal so that the read data will be captured in the middle of the valid window in the I/O logic. The value programmed into this bit field sets the actual number of delay elements in the read_dqs line. If the total delay time programmed exceeds the number of delay elements in the delay chain, then the delay will be set internally to the maximum number of delay elements available.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.23 DRAM Control Register 20 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL20 0x050

Table 12-47. HW_DRAM_CTL20

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TRCD_INT								TRAS_MIN								WR_DQS_SHIFT_BYPASS								RSVD1	WR_DQS_SHIFT						

Table 12-48. HW_DRAM_CTL20 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**TRCD_INT**	RW	0x00	DRAM TRCD parameter in cycles. Defines the DRAM RAS to CAS delay, in cycles
23:16	**TRAS_MIN**	RW	0x00	DRAM TRAS_MIN parameter in cycles. Defines the DRAM minimum row activate time, in cycles.
15:8	**WR_DQS_SHIFT_BYPASS**	RW	0x00	Sets the delay for the clk_wr signal when the DLL is being bypassed. This is used to ensure correct data capture in the I/O logic. The value programmed into this bit field sets the actual number of delay elements in the clk_wr line. If the total delay time programmed exceeds the number of delay elements in the delay chain, then the delay will be set internally to the maximum number of delay elements available.
7	**RSVD1**	RO	0x0	Reserved.
6:0	**WR_DQS_SHIFT**	RW	0x00	Sets the delay for the clk_wr signal to ensure correct data capture in the I/O logic. Each increment of this bit field adds a delay of 1/128 of the system clock. The same delay will be added to the clk_dqs_out signal for each slice.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.24 DRAM Control Register 21 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL21 0x054

Table 12-49. HW_DRAM_CTL21

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
OBSOLETE								RSVD1						OUT_OF_RANGE_LENGTH										TRFC							

Table 12-50. HW_DRAM_CTL21 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**OBSOLETE**	RO	0x0	Reserved.
23:18	**RSVD1**	RO	0x0	Reserved.
17:8	**OUT_OF_RANGE_LENGTH**	RO	0x000	Length of command that caused an Out-of-Range interrupt. Holds the length of the command that caused an out-of-range interrupt request to the memory devices.
7:0	**TRFC**	RW	0x00	DRAM TRFC parameter in cycles. Defines the DRAM refresh command time, in cycles.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.25 DRAM Control Register 22 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL22 0x058

Table 12-51. HW_DRAM_CTL22

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2					AHB0_WRCNT											RSVD1					AHB0_RDCNT										

Table 12-52. HW_DRAM_CTL22 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:27	**RSVD2**	RO	0x0	Reserved.
26:16	**AHB0_WRCNT**	RW	0x000	Number of bytes for an INCR WRITE command on port 0. Holds the number of bytes to send to the memory controller core from AHB port 0 for an INCR WRITE AHB command. The AHB logic will subdivide an INCR request into memory controller core commands of the size of this bit field. The logic will continue sending bursts of this size as the previous request has been transmitted by the AHB port. If the INCR command is terminated on an unnatural boundary, the logic will discard the unnecessary words. The value defined in this bit field should be a multiple of the number of bytes in the AHB port width. Clearing this bit field will cause the port to issue commands of 0 length to the controller core, which the core interprets as the pre-configured value of 1024 bytes.
15:11	**RSVD1**	RO	0x0	Reserved.
10:0	**AHB0_RDCNT**	RW	0x000	Number of bytes for an INCR READ command on port 0. Holds the number of bytes to return to AHB port 0 for an INCR READ AHB command. The AHB logic will subdivide an INCR request into memory controller core commands of the size of this bit field. The logic will continue requesting bursts of this size as soon as the previous request has been received by the AHB port. If the INCR command is terminated on an unnatural boundary, the logic will discard the unnecessary words. The value defined in this bit field should be a multiple of the number of bytes in the AHB port width. Clearing this bit field will cause the port to issue commands of 0 length to the controller core, which the core interprets as the pre-configured value of 1024 bytes.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.26 DRAM Control Register 23 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL23 0x05C

Table 12-53. HW_DRAM_CTL23

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2					AHB1_WRCNT											RSVD1					AHB1_RDCNT										

Table 12-54. HW_DRAM_CTL23 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:27	**RSVD2**	RO	0x0	Reserved.
26:16	**AHB1_WRCNT**	RW	0x000	Number of bytes for an INCR WRITE command on port 1. Holds the number of bytes to send to the memory controller core from AHB port 0 for an INCR WRITE AHB command. The AHB logic will subdivide an INCR request into memory controller core commands of the size of this bit field. The logic will continue sending bursts of this size as the previous request has been transmitted by the AHB port. If the INCR command is terminated on an unnatural boundary, the logic will discard the unnecessary words. The value defined in this bit field should be a multiple of the number of bytes in the AHB port width. Clearing this bit field will cause the port to issue commands of 0 length to the controller core, which the core interprets as the pre-configured value of 1024 bytes.
15:11	**RSVD1**	RO	0x0	Reserved.
10:0	**AHB1_RDCNT**	RW	0x000	Number of bytes for an INCR READ command on port 1. Holds the number of bytes to return to AHB port 0 for an INCR READ AHB command. The AHB logic will subdivide an INCR request into memory controller core commands of the size of this bit field. The logic will continue requesting bursts of this size as soon as the previous request has been received by the AHB port. If the INCR command is terminated on an unnatural boundary, the logic will discard the unnecessary words. The value defined in this bit field should be a multiple of the number of bytes in the AHB port width. Clearing this bit field will cause the port to issue commands of 0 length to the controller core, which the core interprets as the pre-configured value of 1024 bytes.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.27 DRAM Control Register 24 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL24 0x060

Table 12-55. HW_DRAM_CTL24

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2					AHB2_WRCNT											RSVD1					AHB2_RDCNT										

Table 12-56. HW_DRAM_CTL24 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:27	**RSVD2**	RO	0x0	Reserved.
26:16	**AHB2_WRCNT**	RW	0x000	Number of bytes for an INCR WRITE command on port 2. Holds the number of bytes to send to the memory controller core from AHB port 2 for an INCR WRITE AHB command. The AHB logic will subdivide an INCR request into memory controller core commands of the size of this bit field. The logic will continue sending bursts of this size as the previous request has been transmitted by the AHB port. If the INCR command is terminated on an unnatural boundary, the logic will discard the unnecessary words. The value defined in this bit field should be a multiple of the number of bytes in the AHB port width. Clearing this bit field will cause the port to issue commands of 0 length to the controller core, which the core interprets as the pre-configured value of 1024 bytes.
15:11	**RSVD1**	RO	0x0	Reserved.
10:0	**AHB2_RDCNT**	RW	0x000	Number of bytes for an INCR READ command on port 2. Holds the number of bytes to return to AHB port 2 for an INCR READ AHB command. The AHB logic will subdivide an INCR request into memory controller core commands of the size of this bit field. The logic will continue requesting bursts of this size as soon as the previous request has been received by the AHB port. If the INCR command is terminated on an unnatural boundary, the logic will discard the unnecessary words. The value defined in this bit field should be a multiple of the number of bytes in the AHB port width. Clearing this bit field will cause the port to issue commands of 0 length to the controller core, which the core interprets as the pre-configured value of 1024 bytes.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.28 DRAM Control Register 25 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL25 0x064

Table 12-57. HW_DRAM_CTL25

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2					AHB3_WRCNT											RSVD1					AHB3_RDCNT										

Table 12-58. HW_DRAM_CTL25 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:27	**RSVD2**	RO	0x0	Reserved.
26:16	**AHB3_WRCNT**	RW	0x000	Number of bytes for an INCR WRITE command on port 3. Holds the number of bytes to send to the memory controller core from AHB port 3 for an INCR WRITE AHB command. The AHB logic will subdivide an INCR request into memory controller core commands of the size of this bit field. The logic will continue sending bursts of this size as the previous request has been transmitted by the AHB port. If the INCR command is terminated on an unnatural boundary, the logic will discard the unnecessary words. The value defined in this bit field should be a multiple of the number of bytes in the AHB port width. Clearing this bit field will cause the port to issue commands of 0 length to the controller core, which the core interprets as the pre-configured value of 1024 bytes.
15:11	**RSVD1**	RO	0x0	Reserved.
10:0	**AHB3_RDCNT**	RW	0x000	Number of bytes for an INCR READ command on port 3. Holds the number of bytes to return to AHB port 3 for an INCR READ AHB command. The AHB logic will subdivide an INCR request into memory controller core commands of the size of this bit field. The logic will continue requesting bursts of this size as soon as the previous request has been received by the AHB port. If the INCR command is terminated on an unnatural boundary, the logic will discard the unnecessary words. The value defined in this bit field should be a multiple of the number of bytes in the AHB port width. Clearing this bit field will cause the port to issue commands of 0 length to the controller core, which the core interprets as the pre-configured value of 1024 bytes.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.29 DRAM Control Register 26 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL26 0x068

Table 12-59. HW_DRAM_CTL26

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
OBSOLETE																RSVD1				TREF											

Table 12-60. HW_DRAM_CTL26 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**OBSOLETE**	RO	0x0	Reserved.
15:12	**RSVD1**	RO	0x0	Reserved.
11:0	**TREF**	RW	0x000	DRAM TREF parameter in cycles. Defines the DRAM cycles between refresh commands.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.30 DRAM Control Register 27 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL27 0x06C

Table 12-61. HW_DRAM_CTL27

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
OBSOLETE																															

Table 12-62. HW_DRAM_CTL27 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**OBSOLETE**	RO	0x0	Reserved.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.31 DRAM Control Register 28 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL28 0x070

Table 12-63. HW_DRAM_CTL28

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
OBSOLETE																															

Table 12-64. HW_DRAM_CTL28 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**OBSOLETE**	RO	0x0	Reserved.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.32 DRAM Control Register 29 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL29 0x074

Table 12-65. HW_DRAM_CTL29

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
LOWPOWER_INTERNAL_CNT																LOWPOWER_EXTERNAL_CNT															

Table 12-66. HW_DRAM_CTL29 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**LOWPOWER_INTERNAL_CNT**	RW	0x0000	Counts idle cycles to self-refresh with memory and controller clk gating. Counts the number of idle cycles before memory self-refresh with memory and controller clock gating low-power mode 5.
15:0	**LOWPOWER_EXTERNAL_CNT**	RW	0x0000	Counts idle cycles to self-refresh with memory clock gating. Counts the number of idle cycles before memory self-refresh with memory clock gating low-power mode 4.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.33 DRAM Control Register 30 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL30 0x078

Table 12-67. HW_DRAM_CTL30

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
LOWPOWER_REFRESH_HOLD																LOWPOWER_POWER_DOWN_CNT															

Table 12-68. HW_DRAM_CTL30 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**LOWPOWER_REFRESH_HOLD**	RW	0x0000	Re-Sync counter for DLL in Clock Gate Mode. Counts the re-synchronization cycles for the DLL in Clock Gate Mode.
15:0	**LOWPOWER_POWER_DOWN_CNT**	RW	0x0000	Counts idle cycles to memory power-down. Counts the number of idle cycles before memory power-down or power-down with memory clock gating low-power mode 1 or 2.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.34 DRAM Control Register 31 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL31 0x07C

Table 12-69. HW_DRAM_CTL31

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TDLL																LOWPOWER_SELF_REFRESH_CNT															

Table 12-70. HW_DRAM_CTL31 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**TDLL**	RW	0x0000	DRAM TDLL parameter in cycles. Defines the DRAM DLL lock time, in cycles.
15:0	**LOWPOWER_SELF_REFRESH_CNT**	RW	0x0000	Counts idle cycles to memory self-refresh. Counts the number of cycles to the next memory self-refresh low-power mode 3.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.35 DRAM Control Register 32 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL32 0x080

Table 12-71. HW_DRAM_CTL32

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TXSNR																TRAS_MAX															

Table 12-72. HW_DRAM_CTL32 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**TXSNR**	RW	0x0000	DRAM TXSNR parameter in cycles. Defines the DRAM tXSNR parameter, in cycles.
15:0	**TRAS_MAX**	RW	0x0000	DRAM TRAS_MAX parameter in cycles. Defines the DRAM maximum row active time, in cycles.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.36 DRAM Control Register 33 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL33 0x084

Table 12-73. HW_DRAM_CTL33

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
VERSION																TXSR															

Table 12-74. HW_DRAM_CTL33 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**VERSION**	RO	0x2041	Controller version number. Holds the version number for this controller.
15:0	**TXSR**	RW	0x0000	DRAM TXSR parameter in cycles. Defines the DRAM self-refresh exit time, in cycles.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.37 DRAM Control Register 34 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL34 0x088

Table 12-75. HW_DRAM_CTL34

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1								TINIT																							

Table 12-76. HW_DRAM_CTL34 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD1**	RO	0x0	Reserved.
23:0	**TINIT**	RW	0x000000	DRAM TINIT parameter in cycles. Defines the DRAM initialization time, in cycles.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.38 DRAM Control Register 35 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL35 0x08C

Table 12-77. HW_DRAM_CTL35

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1																OUT_OF_RANGE_ADDR															

Table 12-78. HW_DRAM_CTL35 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSVD1**	RO	0x0	Reserved.
30:0	**OUT_OF_RANGE_ADDR**	RO	0x00000000	Address of command that caused an Out-of-Range interrupt. Holds the address of the command that caused an out-of-range interrupt request to the memory devices.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.39 DRAM Control Register 36 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL36 0x090

Table 12-79. HW_DRAM_CTL36

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD5							PWRUP_SREFRESH_EXIT	RSVD4							ENABLE_QUICK_SREFRESH	RSVD3							RSVD2	RSVD1							ACTIVE_AGING

Table 12-80. HW_DRAM_CTL36 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSVD5**	RO	0x0	Reserved.
24	**PWRUP_SREFRESH_EXIT**	RW	0x0	Powerup via self-refresh instead of full memory initialization. Allows controller to exit power-down mode by executing a self-refresh instead of the full memory initialization. 0 = Disabled 1 = Enabled
23:17	**RSVD4**	RO	0x0	Reserved.
16	**ENABLE_QUICK_SREFRESH**	RW	0x0	Interrupts memory initialization to enter self-refresh mode. When this bit is set, the memory initialization sequence will be interrupted and self-refresh mode will be entered. 0 = Continue memory initialization. 1 = Interrupt memory initialization and enter self-refresh mode.
15:9	**RSVD3**	RO	0x0	Reserved.
8	**RSVD2**	RW	0x0	Program this field to 0x0.

Table 12-80. HW_DRAM_CTL36 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
7:1	**RSVD1**	RO	0x0	Reserved.
0	**ACTIVE_AGING**	RW	0x0	Enable aging of the active command. Enables aging of the active command as a condition when using the placement logic to fill the command queue. 0 = Disabled 1 = Enabled

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.40 DRAM Control Register 37 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL37 0x094

Table 12-81. HW_DRAM_CTL37

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
OBSOLETE								RSVD2				BUS_SHARE_TIMEOUT								RSVD1											TREF_ENABLE

Table 12-82. HW_DRAM_CTL37 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**OBSOLETE**	RO	0x0	Reserved.
23:18	**RSVD2**	RO	0x0	Reserved.
17:8	**BUS_SHARE_TIMEOUT**	RW	0x000	Wait time from when the memory controller needs the bus to asserting bus_timeout signal. Sets the wait time for the memory controller when the bus is being shared. This value is loaded into the bus share counter when a command is ready to communicate with the memory devices. When the counter expires, the bus_timeout signal will be asserted indicating to the external source that the memory controller is requesting the shared pins.

Table 12-82. HW_DRAM_CTL37 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
7:1	**RSVD1**	RO	0x0	Reserved.
0	**TREF_ENABLE**	RW	0x0	Issue self-refresh commands to the DRAMs every TREF cycles. Enables internal refresh commands. If command refresh mode is configured, then refresh commands will be issued based on the internal tref counter and any refresh commands sent through the command interface. 0 = Internal refresh commands disabled. 1 = Internal refresh commands enabled.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.41 DRAM Control Register 38 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL38 0x098

Table 12-83. HW_DRAM_CTL38

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2			EMRS2_DATA_0													RSVD1			EMRS1_DATA												

Table 12-84. HW_DRAM_CTL38 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSVD2**	RO	0x0	Reserved.
28:16	**EMRS2_DATA_0**	RW	0x0000	EMRS2 data for chip select 0. Holds the EMRS2 data written during DDRII initialization for chip select 0. The contents of this bit field will be programmed into the DRAM at initialization or when the write_modereg bit field is written with a 1. Consult the DRAM specification for the correct settings for this bit field.

Table 12-84. HW_DRAM_CTL38 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:13	**RSVD1**	RO	0x0	Reserved.
12:0	**EMRS1_DATA**	RW	0x0000	EMRS1 data. Holds the EMRS1 data written during DDRII initialization. The contents of this bit field will be programmed into the DRAM at initialization or when the WRITE_MODEREG bit field is written with a 1. Consult the DRAM specification for the correct settings for this bit field.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.42 DRAM Control Register 39 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL39 0x09C

Table 12-85. HW_DRAM_CTL39

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2			EMRS2_DATA_2													RSVD1			EMRS2_DATA_1												

Table 12-86. HW_DRAM_CTL39 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSVD2**	RO	0x0	Reserved.
28:16	**EMRS2_DATA_2**	RW	0x0000	EMRS2 data for chip select 2. Holds the EMRS2 data written during DDRII initialization for chip select 2. The contents of this bit field will be programmed into the DRAM at initialization or when the WRITE_MODEREG bit field is written with a 1. Consult the DRAM specification for the correct settings for this bit field.
15:13	**RSVD1**	RO	0x0	Reserved.
12:0	**EMRS2_DATA_1**	RW	0x0000	EMRS2 data for chip select 1. Holds the EMRS2 data written during DDRII initialization for chip select 1. The contents of this bit field will be programmed into the DRAM at initialization or when the WRITE_MODEREG bit field is written with a 1. Consult the DRAM specification for the correct settings for this bit field.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

12.5.43 DRAM Control Register 40 Description

DRAM control register. See bit fields for detailed descriptions.

HW_DRAM_CTL40 0x0A0

Table 12-87. HW_DRAM_CTL40

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TPDEX																RSVD1			EMRS2_DATA_3												

Table 12-88. HW_DRAM_CTL40 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**TPDEX**	RW	0x0000	DRAM TPDEX parameter in cycles. Defines the DRAM power-down exit command period, in cycles.
15:13	**RSVD1**	RO	0x0	Reserved.
12:0	**EMRS2_DATA_3**	RW	0x0000	EMRS2 data for chip select 3. Holds the EMRS2 data written during DDRII initialization for chip select 3. The contents of this bit field will be programmed into the DRAM at initialization or when the WRITE_MODEREG bit field is written with a 1. Consult the DRAM specification for the correct settings for this bit field.

DESCRIPTION:

DRAM Control registers. Individual fields control various aspects of the DRAM interface. See bit fields for descriptions

EXAMPLE:

```
Empty Example.
```

DRAM Block v2.1, Revision 1.50

12.6 EMI Memory Parameters and Register Settings

12.6.1 Mobile DDR (5 nsec) Parameters

Table 12-89. Frequency Dependent Parameters

Parameter	24 MHz	48 MHz	60 MHz	96 MHz	120 MHz	133 MHz	160 MHz
TCKE	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001
TDAL	0x03	0x03	0x03	0x04	0x04	0x04	0x05
TDLL	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
TEMRS	0x02	0x02	0x02	0x02	0x02	0x02	0x02
TINIT	0x000012C1	0x00002582	0x00002EE5	0x00004B0D	0x00005DCA	0x00006665	0x00007D00
TMRD	0x02	0x02	0x02	0x02	0x02	0x02	0x02
TPDEX	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002
TRAS_MAX	0x0687	0x0D17	0x1060	0x1A3B	0x20CA	0x23CD	0x2BB6
TRAS_MIN	0x01	0x01	0x03	0x04	0x05	0x06	0x07
TRC	0x02	0x03	0x04	0x06	0x07	0x08	0x09
TRCD_INT	0x01	0x01	0x01	0x02	0x02	0x02	0x03
TREF	0x000000B3	0x0000016F	0x000001CC	0x000002E6	0x000003A1	0x000003F7	0x000004DA
TRFC	0x0003	0x0005	0x0006	0x000A	0x000C	0x000D	0x0010
TRP	0x01	0x01	0x01	0x02	0x02	0x02	0x03
TRRD	0x01	0x01	0x01	0x01	0x02	0x02	0x02
TWR_INT	0x02	0x02	0x02	0x02	0x02	0x02	0x02
TWTR	0x03	0x03	0x03	0x03	0x03	0x03	0x03
TXSNR	0x0003	0x0006	0x0008	0x000C	0x000F	0x0010	0x0014
TXSR	0x0004	0x0007	0x0009	0x000D	0x0011	0x0012	0x0016

12.6.1.1 Bypass Cutoff

Suggested bypass cutoff frequency is 60 MHz.

12.6.1.2 Bypass Mode Enabled

Table 12-90. Delays

Parameter	24 MHz	48 MHz	60 MHz
DLL_DQS_DELAY_BYPASS_0	0x24	0x13	0x0E
DLL_DQS_DELAY_BYPASS_1	0x24	0x13	0x0E
DQS_OUT_SHIFT_BYPASS	0x01	0x01	0x01
WR_DQS_SHIFT_BYPASS	0x23	0x12	0x0D

12.6.1.3 Bypass Mode Disabled

Table 12-91. DLL

Parameter	60 MHz	96 MHz	120 MHz	133 MHz
DLL_INCREMENT	TBD	TBD	TBD	TBD
DLL_START_POINT	TBD	TBD	TBD	TBD

Table 12-92. Delays

Parameter	Value
DLL_DQS_DELAY_1	0x20
DLL_DQS_DELAY_0	0x20
DQS_OUT_SHIFT	0x7F
WR_DQS_SHIFT	0x1C

12.6.1.4 Example Register Settings

```
////////////////////////////////////////////////////////////////////////////
//
// Filename: mobile_ddr_mt46h32m16lf_5_regs_3_160MHz.h
//
// Description: Initialization register values for EMI DRAM controller
//
////////////////////////////////////////////////////////////////////////////
//
// WARNING!  THIS FILE IS AUTOMATICALLY GENERATED.
//           DO NOT MODIFY THIS FILE DIRECTLY.
//
// SETTINGS USED TO GENERATE THIS FILE:
//
//            Burst: 4
//              CAS: 3
//             Freq: 160 MHz
//   Auto-Precharge: 0
//       DLL_Bypass: 0
//
////////////////////////////////////////////////////////////////////////////

{
    0x01010001,  /* 0x01 ahb0_w_priority 0x01 ahb0_r_priority 0x00 ahb0_fifo_type_reg
                    0x01 addr_cmp_en */
    0x00010100,  /* 0x00 ahb2_fifo_type_reg 0x01 ahb1_w_priority 0x01 ahb1_r_priority
                    0x00 ahb1_fifo_type_reg */
    0x01000101,  /* 0x01 ahb3_r_priority 0x00 ahb3_fifo_type_reg 0x01 ahb2_w_priority
                    0x01 ahb2_r_priority */
    0x00000001,  /* 0x00 auto_refresh_mode 0x00 arefresh 0x00 ap 0x01 ahb3_w_priority */
   0x00000101,  /* 0x00 dll_bypass_mode 0x00 dlllockreg 0x01 concurrentap 0x01 bank_split_en */
   0x00000001,  /* 0x00 intrptreada 0x00 intrptapburst 0x00 fast_write 0x01 en_lowpower_mode */
    0x00010000,  /* 0x00 power_down 0x01 placement_en 0x00 no_cmd_init 0x00 intrptwritea */
    0x01000101,  /* 0x01 rw_same_en 0x00 reg_dimm_enable 0x01 rd2rd_turn 0x01 priority_en */
    0x01000000,  /* 0x01 tras_lockout 0x00 start 0x00 srefresh 0x00 sdr_mode */
    0x00000001,  /* 0x00 out_of_range_type 0x00 out_of_range_source_id 0x00 write_modereg
                    0x01 writeinterp */
```

```
    0x07000200,  /* 0x07 age_count 0x00 addr_pins 000000_10 temrs 0x00 q_fullness */
    0x00070203,  /* 0x00 max_cs_reg 0x07 command_age_count 0x02 column_size 0x03 caslat */
    0x02020001,  /* 0x02 twr_int 0x02 trrd 0x0001 tcke */
    0x06060a03,  /* 0x06 caslat_lin_gate 0x06 caslat_lin 0x0a aprebit 0x03 twtr */
    0x00000201,  /* 0x00 max_col_reg 0x00 lowpower_refresh_enable 0x02 initaref 0x01 cs_map */
    0x03050000,  /* 0x03 trp 0x05 tdal 0x00 port_busy 0x00 max_row_reg */
    0x02000000,  /* 0x02 tmrd 0x00 lowpower_control 0x00 lowpower_auto_enable 0x00 int_ack */
    0x2d000d09,  /* 0x2d dll_start_point 0x00 dll_lock 0x0d dll_increment 0x09 trc */
    0x20200000,  /* 0x20 dll_dqs_delay_1 0x20 dll_dqs_delay_0 0x00 int_status 0x00 int_mask */
    0x02020f0f,  /* 0x02 dqs_out_shift_bypass 0x02 dqs_out_shift 0x0f dll_dqs_delay_bypass_1
                    0x0f dll_dqs_delay_bypass_0 */
    0x0307121c,  /* 0x03 trcd_int 0x07 tras_min 0x12 wr_dqs_shift_bypass 0x1c wr_dqs_shift */
    0x00000010,  /* 0x000000 out_of_range_length 0x10 trfc */
    0x00080008,  /* 0x0008 ahb0_wrcnt 0x0008 ahb0_rdcnt */
    0x00200020,  /* 0x0020 ahb1_wrcnt 0x0020 ahb1_rdcnt */
    0x00200020,  /* 0x0020 ahb2_wrcnt 0x0020 ahb2_rdcnt */
    0x00200020,  /* 0x0020 ahb3_wrcnt 0x0020 ahb3_rdcnt */
    0x000004da,  /* 0x000004da tref */
    0x00000000,  /* 0x00000000 */
    0x00000000,  /* 0x00000000 */
    0x00000000,  /* 0x0000 lowpower_internal_cnt 0x0000 lowpower_external_cnt */
    0x00000000,  /* 0x0000 lowpower_refresh_hold 0x0000 lowpower_power_down_cnt */
    0x00000000,  /* 0x0000 tdll 0x0000 lowpower_self_refresh_cnt */
    0x00142bb6,  /* 0x0014 txsnr 0x2bb6 tras_max */
    0x00000016,  /* 0x0000 version 0x0016 txsr */
    0x00007d00,  /* 0x00007d00 tinit */
    0x00000000,  /* 0x00000000 out_of_range_addr */
    0x00000101,  /* 0x00 pwrup_srefresh_exit 0x00 enable_quick_srefresh
                    0x01 bus_share_enable 0x01 active_aging */
    0x00040001,  /* 0x000400 bus_share_timeout 0x01 tref_enable */
    0x00400000,  /* 0x0040 emrs2_data_0 0x0000 emrs1_data */
    0x00400040,  /* 0x0040 emrs2_data_2 0x0040 emrs2_data_1 */
    0x00020040,  /* 0x0002 tpdex 0x0040 emrs2_data_3 */
},
```

12.6.2 Mobile DDR (6 nsec)

Table 12-93. Frequency Dependent Parameters

Parameter	24 MHz	48 MHz	60 MHz	96 MHz	120 MHz	133 MHz	167 MHz
TCKE	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002
TDAL	0x03	0x03	0x04	0x04	0x05	0x05	0x05
TDLL	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
TEMRS	0x02	0x02	0x02	0x02	0x02	0x02	0x02
TINIT	0x000012C1	0x00002582	0x00002EE5	0x00004B0D	0x00005DCA	0x00006665	0x000081C7
TMRD	0x02	0x02	0x02	0x02	0x02	0x02	0x02
TPDEX	0x0001	0x0002	0x0002	0x0003	0x0004	0x0004	0x0005
TRAS_MAX	0x0687	0x0D17	0x1060	0x1A3B	0x20CA	0x23CD	0x2D62
TRAS_MIN	0x02	0x03	0x03	0x05	0x06	0x06	0x07
TRC	0x02	0x03	0x04	0x06	0x08	0x08	0x0A

Table 12-93. Frequency Dependent Parameters (continued)

Parameter	24 MHz	48 MHz	60 MHz	96 MHz	120 MHz	133 MHz	167 MHz
TRCD_INT	0x01	0x01	0x02	0x02	0x03	0x03	0x03
TREF	0x000000B3	0x0000016F	0x000001CC	0x000002E6	0x000003A1	0x000003F7	0x00000509
TRFC	0x02	0x04	0x05	0x07	0x09	0x0A	0x0C
TRP	0x01	0x01	0x02	0x02	0x03	0x03	0x03
TRRD	0x01	0x01	0x01	0x02	0x02	0x02	0x02
TWR_INT	0x02	0x02	0x02	0x02	0x02	0x02	0x02
TWTR	0x02	0x02	0x02	0x02	0x02	0x02	0x02
TXSNR	0x0003	0x0006	0x0008	0x000C	0x000F	0x0010	0x0014
TXSR	0x0003	0x0006	0x0008	0x000C	0x000F	0x0010	0x0014

12.6.2.1 Bypass Cutoff

Suggested bypass cutoff frequency is 60 MHz.

12.6.2.2 Bypass Mode Enabled

Table 12-94. Delays

Parameter	24 MHz	48 MHz	60 MHz
DLL_DQS_DELAY_BYPASS_0	0x1A	0x0D	0x0C
DLL_DQS_DELAY_BYPASS_1	0x1A	0x0D	0x0C
DQS_OUT_SHIFT_BYPASS	0x01	0x01	0x01
WR_DQS_SHIFT_BYPASS	0x12	0x12	0x12

12.6.2.3 Bypass Mode Disabled

Table 12-95. DLL

Parameter	60 MHz	96 MHz	120 MHz	133 MHz	167 MHz
DLL_INCREMENT	0x06	0x05	0x05	0x05	0x05
DLL_START_POINT	0x28	0x18	0x14	0x14	0x14

Table 12-96. Delays

PARAMETER	VALUE
DLL_DQS_DELAY_1	0x20
DLL_DQS_DELAY_0	0x20
DQS_OUT_SHIFT	0x7F
WR_DQS_SHIFT	0x20

12.6.2.4 Example Register Settings

```
//////////////////////////////////////////////////////////////////////////////
//
// Filename: mobile_ddr_mt46h16m16lf_6_regs_3_96MHz.h
//
// Description: Initialization register values for EMI DRAM controller
//
//////////////////////////////////////////////////////////////////////////////
//
// WARNING!  THIS FILE IS AUTOMATICALLY GENERATED.
//           DO NOT MODIFY THIS FILE DIRECTLY.
//
// SETTINGS USED TO GENERATE THIS FILE:
//
//            Burst: 4
//              CAS: 3
//             Freq: 96 MHz
//   Auto-Precharge: 0
//       DLL_Bypass: 0
//
//////////////////////////////////////////////////////////////////////////////

{
    0x01010001,  /* 0x01 ahb0_w_priority 0x01 ahb0_r_priority 0x00 ahb0_fifo_type_reg
                    0x01 addr_cmp_en */
    0x00010100,  /* 0x00 ahb2_fifo_type_reg 0x01 ahb1_w_priority 0x01 ahb1_r_priority
                    0x00 ahb1_fifo_type_reg */
    0x01000101,  /* 0x01 ahb3_r_priority 0x00 ahb3_fifo_type_reg 0x01 ahb2_w_priority
                    0x01 ahb2_r_priority */
    0x00000001,  /* 0x00 auto_refresh_mode 0x00 arefresh 0x00 ap 0x01 ahb3_w_priority */
   0x00000101,  /* 0x00 dll_bypass_mode 0x00 dlllockreg 0x01 concurrentap 0x01 bank_split_en */
   0x00000001,  /* 0x00 intrptreada 0x00 intrptapburst 0x00 fast_write 0x01 en_lowpower_mode */
    0x00010000,  /* 0x00 power_down 0x01 placement_en 0x00 no_cmd_init 0x00 intrptwritea */
    0x01000001,  /* 0x01 rw_same_en 0x00 reg_dimm_enable 0x00 rd2rd_turn 0x01 priority_en */
    0x01000000,  /* 0x01 tras_lockout 0x00 start 0x00 srefresh 0x00 sdr_mode */
    0x00000001,  /* 0x00 out_of_range_type 0x00 out_of_range_source_id 0x00 write_modereg
                    0x01 writeinterp */
    0x07000200,  /* 0x07 age_count 0x00 addr_pins 000000_10 temrs 0x00 q_fullness */
    0x00070303,  /* 0x00 max_cs_reg 0x07 command_age_count 0x03 column_size 0x03 caslat */
    0x02020002,  /* 0x02 twr_int 0x02 trrd 0x0002 tcke */
    0x06060a02,  /* 0x06 caslat_lin_gate 0x06 caslat_lin 0x0a aprebit 0x02 twtr */
    0x00000201,  /* 0x00 max_col_reg 0x00 lowpower_refresh_enable 0x02 initaref 0x01 cs_map */
    0x02040000,  /* 0x02 trp 0x04 tdal 0x00 port_busy 0x00 max_row_reg */
    0x02000000,  /* 0x02 tmrd 0x00 lowpower_control 0x00 lowpower_auto_enable 0x00 int_ack */
    0x18000606,  /* 01001110 dll_start_point 0x00 dll_lock 0x06 dll_increment 0x06 trc */
    0x15150000,  /* 0x15 dll_dqs_delay_1 0x15 dll_dqs_delay_0 0x00 int_status 0x00 int_mask */
    0x027f1a1a,  /* 0x02 dqs_out_shift_bypass 0x7f dqs_out_shift 0x1a dll_dqs_delay_bypass_1
                    0x1a dll_dqs_delay_bypass_0 */
    0x02051c12,  /* 0x02 trcd_int 0x05 tras_min 0x1c wr_dqs_shift_bypass 0x12 wr_dqs_shift */
    0x00000007,  /* 0x000000 out_of_range_length 0x07 trfc */
    0x00080008,  /* 0x0008 ahb0_wrcnt 0x0008 ahb0_rdcnt */
    0x00200020,  /* 0x0020 ahb1_wrcnt 0x0020 ahb1_rdcnt */
    0x00200020,  /* 0x0020 ahb2_wrcnt 0x0020 ahb2_rdcnt */
    0x00200020,  /* 0x0020 ahb3_wrcnt 0x0020 ahb3_rdcnt */
    0x000002e6,  /* 0x000002e6 tref */
    0x00000000,  /* 0x00000000 */
    0x00000000,  /* 0x00000000 */
```

```
    0x00000000,  /* 0x0000 lowpower_internal_cnt 0x0000 lowpower_external_cnt */
    0x00000000,  /* 0x0000 lowpower_refresh_hold 0x0000 lowpower_power_down_cnt */
    0x00000000,  /* 0x0000 tdll 0x0000 lowpower_self_refresh_cnt */
    0x000c1a3b,  /* 0x000c txsnr 0x1a3b tras_max */
    0x0000000c,  /* 0x0000 version 0x000c txsr */
    0x00004b0d,  /* 0x00004b0d tinit */
    0x00000000,  /* 0x00000000 out_of_range_addr */
    0x00000101,  /* 0x00 pwrup_srefresh_exit 0x00 enable_quick_srefresh
                    0x01 bus_share_enable 0x01 active_aging */
    0x00040001,  /* 0x000400 bus_share_timeout 0x01 tref_enable */
    0x00400000,  /* 0x0040 emrs2_data_0 0x0000 emrs1_data */
    0x00400040,  /* 0x0040 emrs2_data_2 0x0040 emrs2_data_1 */
    0x00030040,  /* 0x0003 tpdex 0x0040 emrs2_data_3 */
},
```

12.6.3 Mobile DDR (7.5 nsec)

Table 12-97. Frequency Dependent Parameters

Parameter	24 MHz	48 MHz	60 MHz	96 MHz	120 MHz	133 MHz
TCKE	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002
TDAL	0x03	0x04	0x04	0x05	0x05	0x05
TDLL	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
TEMRS	0x02	0x02	0x02	0x02	0x02	0x02
TINIT	0x000012C1	0x00002582	0x00002EE5	0x00004B0D	0x00005DCA	0x00006665
TMRD	0x02	0x02	0x02	0x02	0x02	0x02
TPDEX	0x0001	0x0002	0x0002	0x0003	0x0004	0x0004
TRAS_MAX	0x0687	0x0D17	0x1060	0x1A3B	0x20CA	0x23CD
TRAS_MIN	0x02	0x03	0x03	0x05	0x06	0x06
TRC	0x02	0x04	0x05	0x08	0x0A	0x0A
TRCD_INT	0x01	0x02	0x02	0x03	0x03	0x03
TREF	0x00B3	0x016F	0x01CC	0x02E6	0x03A1	0x03F7
TRFC	0x02	0x04	0x05	0x07	0x09	0x0A
TRP	0x01	0x02	0x02	0x03	0x03	0x03
TRRD	0x01	0x01	0x01	0x02	0x02	0x02
TWR_INT	0x02	0x02	0x02	0x02	0x02	0x02
TWTR	0x02	0x02	0x02	0x02	0x02	0x02
TXSNR	0x0003	0x0006	0x0008	0x000C	0x000F	0x0010
TXSR	0x0003	0x0006	0x0008	0x000C	0x000F	0x0010

12.6.3.1 Bypass Cutoff

Suggested bypass cutoff frequency is 72 MHz.

12.6.3.2 Bypass Mode Enabled

Table 12-98. Delays

Parameter	24 MHz	48 MHz	60 MHz	72 MHz
DLL_DQS_DELAY_BYPASS_0	0x19	0x0D	0x0B	0x0C
DLL_DQS_DELAY_BYPASS_1	0x19	0x0D	0x0B	0x0C
DQS_OUT_SHIFT_BYPASS	0x01	0x01	0x01	0x01
WR_DQS_SHIFT_BYPASS	0x18	0x0D	0x0A	0x0C

12.6.3.3 Bypass Mode Disabled

Table 12-99. DLL

Parameter	60 MHz	96 MHz	120 MHz	133 MHz
DLL_INCREMENT	0x22	0x15	0x11	0x0F
DLL_START_POINT	0x7E	0x25	0x19	0x19

Table 12-100. Delays

Parameter	Value
DLL_DQS_DELAY_1	0x0D
DLL_DQS_DELAY_0	0x0D
DQS_OUT_SHIFT	0x7F
WR_DQS_SHIFT	0x20

12.6.3.4 Example Register Settings

```
////////////////////////////////////////////////////////////////////////////////
//
// Filename: mobile_ddr_mt46h16m16lf_7.5_regs_3_120MHz.h
//
// Description: Initialization register values for EMI DRAM controller
//
////////////////////////////////////////////////////////////////////////////////
//
// WARNING!  THIS FILE IS AUTOMATICALLY GENERATED.
//           DO NOT MODIFY THIS FILE DIRECTLY.
//
// SETTINGS USED TO GENERATE THIS FILE:
//
//            Burst: 4
//              CAS: 3
//             Freq: 120 MHz
//   Auto-Precharge: 0
//       DLL_Bypass: 0
//
////////////////////////////////////////////////////////////////////////////////

{
    0x01010001,  /* 0x01 ahb0_w_priority 0x01 ahb0_r_priority 0x00 ahb0_fifo_type_reg
                    0x01 addr_cmp_en */
```

```
    0x00010100,  /* 0x00 ahb2_fifo_type_reg 0x01 ahb1_w_priority 0x01 ahb1_r_priority
                    0x00 ahb1_fifo_type_reg */
    0x01000101,  /* 0x01 ahb3_r_priority 0x00 ahb3_fifo_type_reg 0x01 ahb2_w_priority
                    0x01 ahb2_r_priority */
    0x00000001,  /* 0x00 auto_refresh_mode 0x00 arefresh 0x00 ap 0x01 ahb3_w_priority */
   0x00000101,  /* 0x00 dll_bypass_mode 0x00 dlllockreg 0x01 concurrentap 0x01 bank_split_en */
   0x00000001,  /* 0x00 intrptreada 0x00 intrptapburst 0x00 fast_write 0x01 en_lowpower_mode */
    0x00010000,  /* 0x00 power_down 0x01 placement_en 0x00 no_cmd_init 0x00 intrptwritea */
    0x01000001,  /* 0x01 rw_same_en 0x00 reg_dimm_enable 0x00 rd2rd_turn 0x01 priority_en */
    0x01000000,  /* 0x01 tras_lockout 0x00 start 0x00 srefresh 0x00 sdr_mode */
    0x00000001,  /* 0x00 out_of_range_type 0x00 out_of_range_source_id 0x00 write_modereg
                    0x01 writeinterp */
    0x07000200,  /* 0x07 age_count 0x00 addr_pins 0x02 temrs 0x00 q_fullness */
    0x00070303,  /* 0x00 max_cs_reg 0x07 command_age_count 0x03 column_size 0x03 caslat */
    0x02020002,  /* 0x02 twr_int 0x02 trrd 0x0002 tcke */
    0x06060a02,  /* 0x06 caslat_lin_gate 0x06 caslat_lin 0x0a aprebit 0x02 twtr */
    0x00000201,  /* 0x00 max_col_reg 0x00 lowpower_refresh_enable 0x02 initaref 0x01 cs_map */
    0x03050000,  /* 0x03 trp 0x05 tdal 0x00 port_busy 0x00 max_row_reg */
    0x02000000,  /* 0x02 tmrd 0x00 lowpower_control 0x00 lowpower_auto_enable 0x00 int_ack */
    0x1900110a,  /* 0x19 dll_start_point 0x00 dll_lock 0x11 dll_increment 0x0a trc */
    0x1e1e0000,  /* 0x1e dll_dqs_delay_1 0x1e dll_dqs_delay_0 0x00 int_status 0x00 int_mask */
    0x01011919,  /* 0x01 dqs_out_shift_bypass 0x01 dqs_out_shift 0x19 dll_dqs_delay_bypass_1
                    0x19 dll_dqs_delay_bypass_0 */
    0x03061820,  /* 0x03 trcd_int 0x06 tras_min 0x18 wr_dqs_shift_bypass 0x20 wr_dqs_shift */
    0x00000009,  /* 0x000000 out_of_range_length 0x09 trfc */
    0x00080008,  /* 0x0008 ahb0_wrcnt 0x0008 ahb0_rdcnt */
    0x00200020,  /* 0x0020 ahb1_wrcnt 0x0020 ahb1_rdcnt */
    0x00200020,  /* 0x0020 ahb2_wrcnt 0x0020 ahb2_rdcnt */
    0x00200020,  /* 0x0020 ahb3_wrcnt 0x0020 ahb3_rdcnt */
    0x000003a1,  /* 0x000003a1 tref */
    0x00000000,  /* 0x00000000 */
    0x00000000,  /* 0x00000000 */
    0x00000000,  /* 0x0000 lowpower_internal_cnt 0x0000 lowpower_external_cnt */
    0x00000000,  /* 0x0000 lowpower_refresh_hold 0x0000 lowpower_power_down_cnt */
    0x00000000,  /* 0x0000 tdll 0x0000 lowpower_self_refresh_cnt */
    0x000f20ca,  /* 0x000f txsnr 0x20ca tras_max */
    0x0000000f,  /* 0x0000 version 0x000f txsr */
    0x00005dca,  /* 0x00005dca tinit */
    0x00000000,  /* 0x00000000 out_of_range_addr */
    0x00000101,  /* 0x00 pwrup_srefresh_exit 0x00 enable_quick_srefresh
                    0x01 bus_share_enable 0x01 active_aging */
    0x00040001,  /* 0x000400 bus_share_timeout 0x01 tref_enable */
    0x00400000,  /* 0x0040 emrs2_data_0 0x0000 emrs1_data */
    0x00400040,  /* 0x0040 emrs2_data_2 0x0040 emrs2_data_1 */
    0x00040040,  /* 0x0004 tpdex 0x0000 emrs2_data_3 */
},
```

12.6.4 DDR

Table 12-101. Frequency Dependent Parameters

Parameter	80 MHz	96 MHz	120 MHz	133 MHz	167 MHz
TCKE	0x0000	0x0000	0x0000	0x0000	0x0000
TDAL	0x04	0x04	0x04	0x04	0x05
TDLL	0x00C8	0x00C8	0x00C8	0x00C8	0x00C8
TEMRS	0x00	0x00	0x00	0x00	0x00
TINIT	0x00003E80	0x00004B0D	0x00005DCA	0x00006665	0x000081C7
TMRD	0x01	0x02	0x02	0x02	0x02
TPDEX	0x0001	0x0001	0x0001	0x0001	0x0001
TRAS_MAX	0x15D6	0x1A3B	0x20CA	0x23CD	0x2D62
TRAS_MIN	0x04	0x05	0x06	0x06	0x07
TRC	0x05	0x06	0x08	0x08	0x0A
TRCD_INT	0x02	0x02	0x02	0x02	0x03
TREF	0x00000269	0x000002E6	0x000003A1	0x000003F7	0x00000509
TRFC	0x0006	0x0007	0x0009	0x000A	0x000C
TRP	0x02	0x02	0x02	0x02	0x03
TRRD	0x01	0x02	0x02	0x02	0x02
TWR_INT	0x02	0x02	0x02	0x02	0x02
TWTR	0x01	0x01	0x01	0x01	0x01
TXSNR	0x0006	0x0008	0x000A	0x000A	0x000D
TXSR	0x00C8	0x00C8	0x00C8	0x00C8	0x00C8

12.6.4.1 Bypass Mode Disabled

NOTE: DDR should always be run with bypass disabled.

Table 12-102. DLL

Parameter	80 MHz	96 MHz	120 MHz	133 MHz	167 MHz
DLL_INCREMENT	0x1C	0x17	0x13	0x11	0x0D
DLL_START_POINT	0x20	0x2F	0x26	0x22	0x18

Table 12-103. Delays

Parameter	80 MHz	96 MHz	120 MHz	133 MHz	160 MHz
DLL_DQS_DELAY_1	0x1F	0x1F	0x1F	0x1F	0x1F
DLL_DQS_DELAY_0	0x1F	0x1F	0x1F	0x1F	0x1F
DQS_OUT_SHIFT	0x7F	0x7F	0x7F	0x7F	0x7F
WR_DQS_SHIFT	0x22	0x22	0x23	0x23	0x24

12.6.4.2 Example Register Settings

```
//////////////////////////////////////////////////////////////////////////////
//
// Filename: ddr_mt46v32m16_6t_regs_2.5_167MHz.h
//
// Description: Initialization register values for EMI DRAM controller
//
//////////////////////////////////////////////////////////////////////////////
//
// WARNING!  THIS FILE IS AUTOMATICALLY GENERATED.
//           DO NOT MODIFY THIS FILE DIRECTLY.
//
// SETTINGS USED TO GENERATE THIS FILE:
//
//            Burst: 4
//              CAS: 2.5
//             Freq: 167MHz
//   Auto-Precharge: 0
//       DLL_Bypass: 0
//
//////////////////////////////////////////////////////////////////////////////

{
    0x01010001,  /* 0x01 ahb0_w_priority 0x01 ahb0_r_priority 0x00 ahb0_fifo_type_reg
                    0x01 addr_cmp_en */
    0x00010100,  /* 0x00 ahb2_fifo_type_reg 0x01 ahb1_w_priority 0x01 ahb1_r_priority
                    0x00 ahb1_fifo_type_reg */
    0x01000101,  /* 0x01 ahb3_r_priority 0x00 ahb3_fifo_type_reg 0x01 ahb2_w_priority
                    0x01 ahb2_r_priority */
    0x00000001,  /* 0x00 auto_refresh_mode 0x00 arefresh 0x00 ap 0x01 ahb3_w_priority */
    0x00000101,  /* 0x00 dll_bypass_mode 0x00 dlllockreg 0x01 concurrentap 0x01 bank_split_en */
    0x00000000,  /* 0x00 intrptreada 0x00 intrptapburst 0x00 fast_write 0x00 en_lowpower_mode */
    0x00010000,  /* 0x00 power_down 0x01 placement_en 0x00 no_cmd_init 0x00 intrptwritea */
    0x01000001,  /* 0x01 rw_same_en 0x00 reg_dimm_enable 0x00 rd2rd_turn 0x01 priority_en */
    0x01000000,  /* 0x01 tras_lockout 0x00 start 0x00 srefresh 0x00 sdr_mode */
    0x00000001,  /* 0x00 out_of_range_type 0x00 out_of_range_source_id 0x00 write_modereg
                    0x01 writeinterp */
    0x07000200,  /* 0x07 age_count 0x00 addr_pins 0x02 temrs 0x00 q_fullness */
    0x00070206,  /* 0x00 max_cs_reg 0x07 command_age_count 0x02 column_size 0x06 caslat */
    0x02020000,  /* 0x02 twr_int 0x02 trrd 0x0000 tcke */
    0x05050a01,  /* 0x05 caslat_lin_gate 0x05 caslat_lin 0x0a aprebit 0x01 twtr */
    0x0000020f,  /* 0x00 max_col_reg 0x00 lowpower_refresh_enable 0x02 initaref 0x01 cs_map */
    0x03050000,  /* 0x03 trp 0x05 tdal 0x00 port_busy 0x00 max_row_reg */
    0x02000000,  /* 0x02 tmrd 0x00 lowpower_control 0x00 lowpower_auto_enable 0x00 int_ack */
    0x18000d0a,  /* 0x18 dll_start_point 0x00 dll_lock 0x0d dll_increment 0x0a trc */
    0x1f1f0000,  /* 0x1f dll_dqs_delay_1 0x1f dll_dqs_delay_0 0x00 int_status 0x00 int_mask */
    0x027f0f0f,  /* 0x02 dqs_out_shift_bypass 0x7f dqs_out_shift 0x0f dll_dqs_delay_bypass_1
                    0x0f dll_dqs_delay_bypass_0 */
    0x03071124,  /* 0x03 trcd_int 0x07 tras_min 0x11 wr_dqs_shift_bypass 0x24 wr_dqs_shift */
    0x0000000c,  /* 0x000000 out_of_range_length 00001100 trfc */
    0x00080008,  /* 0x0008 ahb0_wrcnt 0x0008 ahb0_rdcnt */
    0x00200020,  /* 0x0020 ahb1_wrcnt 0x0020 ahb1_rdcnt */
    0x00200020,  /* 0x0020 ahb2_wrcnt 0x0020 ahb2_rdcnt */
    0x00200020,  /* 0x0020 ahb3_wrcnt 0x0020 ahb3_rdcnt */
    0x00000509,  /* 0x00000509 tref */
    0x00000000,  /* 0x00000000 */
    0x00000000,  /* 0x00000000 */
```

```
    0x00000000,  /* 0x0000 lowpower_internal_cnt 0x0000 lowpower_external_cnt */
    0x00000000,  /* 0x0000 lowpower_refresh_hold 0x0000 lowpower_power_down_cnt */
    0x00c80000,  /* 0x00c8 tdll 0x0000 lowpower_self_refresh_cnt */
    0x000d2d62,  /* 0x000d txsnr 0x2d62 tras_max */
    0x000000c8,  /* 0x0000 version 0x00c8 txsr */
    0x000081c7,  /* 0x000081c7 tinit */
    0x00000000,  /* 0x00000000 out_of_range_addr */
    0x00000101,  /* 0x00 pwrup_srefresh_exit 0x00 enable_quick_srefresh
                    0x01 bus_share_enable 0x01 active_aging */
    0x00040001,  /* 0x000400 bus_share_timeout 0x01 tref_enable */
    0x00000000,  /* 0x0000 emrs2_data_0 0x0000 emrs1_data */
    0x00000000,  /* 0x0000 emrs2_data_2 0x0000 emrs2_data_1 */
    0x00010000,  /* 0x0001 tpdex 0x0000 emrs2_data_3 */
},
```

Chapter 13
General-Purpose Media Interface (GPMI)

This chapter describes the general-purpose media interface (GPMI) on the i.MX23. Programmable registers are described in Section 13.4, “Programmable Registers.”

13.1 Overview

The general-purpose media interface (GPMI) controller is a flexible interface to up to four NAND Flash. The NAND Flash mode has configurable address and command behavior, providing support for future devices not yet specified.

The GPMI resides on the APBH. The GPMI also provides an interface to the ECC8 and BCH modules to allow direct parity processing.

Registers are clocked on the HCLK (CLK_H) domain. The I/O and pin timing are clocked on a dedicated GPMICLK domain. GPMICLK can be set to maximize I/O performance.

Figure 13-1 shows a block diagram of the GPMI controller.

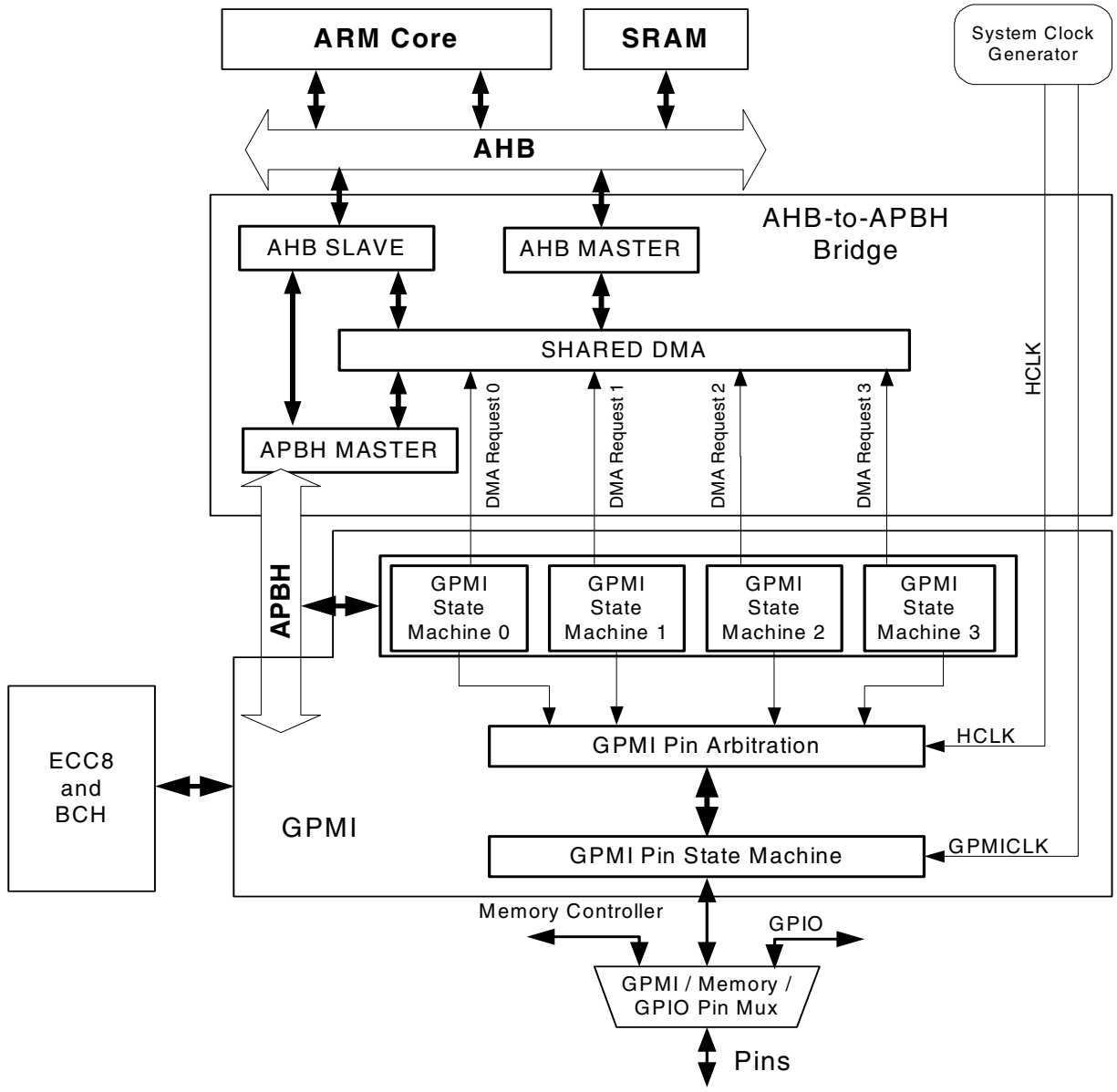


Figure 13-1. General-Purpose Media Interface Controller Block Diagram

13.2 GPMI NAND Flash Mode

The general-purpose media interface has several features to efficiently support NAND Flash:

- Individual chip select and ready/busy pins for four NAND Flash.
- Individual state machine and DMA channel for each chip select.
- Special command modes work with DMA controller to perform all normal NAND Flash functions without CPU intervention.

- Configurable timing based on a dedicated clock allows optimal balance of high NAND Flash performance and low system power.

Since current NAND Flash does not support multiple page read/write commands, the GPMI and DMA have been designed to handle complex multi-page operations without CPU intervention. The DMA uses a linked descriptor function with branching capability to automatically handle all of the operations needed to read/write multiple pages:

- **Data/Register Read/Write**—The GPMI can be programmed to read or write multiple cycles to the NAND Flash address, command or data registers.
- **Wait for NAND Flash Ready**—The GPMI's Wait-for-Ready mode can monitor the ready/busy signal of a single NAND Flash and signal the DMA when the device has become ready. It also has a time-out counter and can indicate to the DMA that a time-out error has occurred. The DMAs can conditionally branch to a different descriptor in the case of an error.
- **Check Status**—The Read-and-Compare mode allows the GPMI to check NAND Flash status against a reference. If an error is found, the GPMI can instruct the DMA to branch to an alternate descriptor, which attempts to fix the problem or asserts a CPU IRQ.

13.2.1 Multiple NAND Flash Support

The GPMI supports up to four NAND Flash chip selects, each with independent ready/busy signals. Since they share a data bus and control lines, the GPMI can only actively communicate with a single NAND Flash at a time. However, all NAND Flashes can concurrently perform internal read, write, or erase operations. With fast NAND Flash and software support for concurrent NAND Flash operations, this architecture allows the total throughput to approach the data bus speed, which can be as high as 80 MB/s (16-bit bus running at 40 MHz).

13.2.2 GPMI NAND Flash Timing and Clocking

The dedicated clock, GPMICLK, is used as a timing reference for NAND Flash I/O. Since various NAND Flashes have different timing requirements, GPMICLK may need to be adjusted for each application. While the actual pin timings are limited by the NAND Flash chips used, the GPMI can support data bus speeds of up to 33 MHz x 16 bits. The actual read/write strobe timing parameters are adjusted as indicated in the register descriptions in Section 13.4, "Programmable Registers." Refer to Chapter 4, "Clock Generation and Control," for more information about setting GPMICLK.

13.2.3 Basic NAND Flash Timing

Figure 13-2 illustrates the operation of the timing parameters in NAND Flash mode.

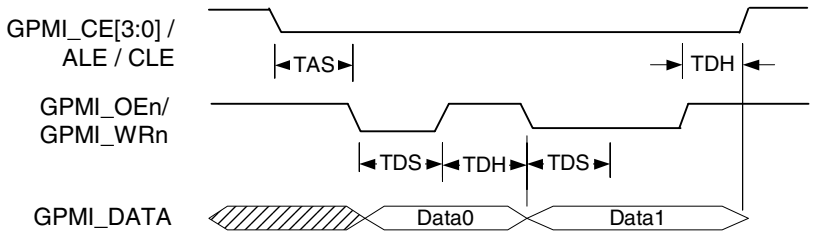


Figure 13-2. BASIC NAND Flash Timing

13.2.4 High-Speed NAND Flash Timing

In high-speed NAND Flashes, the read data may not be valid until after the read strobe (RDN) deasserts. This is the case when the minimum tDS is programmed to achieve higher bandwidth. The GPMI implements a feedback read strobe to sample the read data. The feedback read strobe can be delayed to support fast NAND Flash EDO (Extended Data Out) timing where the read strobe may deassert before the read data is valid, and read data is valid for some time after read strobe. NAND Flash EDO timings is applied typically for read cycle frequency above 33MHz. See Figure 13-3.

The GPMI provides control over the amount of delay applied to the feedback read strobe. This delay depends on the maximum read access time (tREA) of the NAND Flash and the read pulse width (tRP) used to access the NAND Flash. tRP is specified by HW_GPMI_TIMING0_DATA_SETUP register. When (tREA + 4ns) is less than tRP, no delay is required to sample to NAND Flash read data. (The 4ns provides adequate data setup time for the GPMI.) In this case set HW_GPMI_CTRL1_HALF_PERIOD=0; HW_GPMI_CTRL1_RDN_DELAY=0; HW_GPMI_CTRL1_DLL_ENABLE=0.

When (tREA + 4ns) is greater than or equal to tRP, a delay of the feedback read strobe is required to sample to NAND Flash read data. This delay is equal to the difference between these two timings:

DELAY= tREA+4ns - tRP.

Since the GPMI delay chain is limited to 16ns maximum, if DELAY > 16ns then increase tRP by increasing the value of HW_GPMI_TIMING0_DATA_SETUP until DELAY is less than or equal to 16ns.

The GPMI programming for this DELAY depends on the GPMICLK period. The GPMI DLL will not function properly if the GPMICLK period is greater than 32ns: disable the DLL if this is the case. If the GPMICLK period is greater than 16ns (and not greater than 32ns), set the HW_GPMI_CTRL1_HALF_PERIOD=1; This will cause the DLL reference period (RP) to be one-half of the GPMICLK period. If the GPMICLK period is 16ns or less then set the

HW_GPMI_CTRL1_HALF_PERIOD=0; This will cause the DLL reference period (RP) to be equal to the GPMICLK period. DELAY is a multiple (0 to 1.875) of RP.

The HW_GPMI_CTRL1_RDN_DELAY is encoded as a 1-bit integer and 3-bit fraction delay factor. See table below. DELAY is a multiple of the delay factor and the reference period:

Table 13-1.

HW_GPMI_CTRL1_RDN_DELAY	0	1	2	3	4	5	6	7
Delay Factor	0.000	0.125	0.250	0.375	0.500	0.625	0.750	0.875
HW_GPMI_CTRL1_RDN_DELAY	8	9	10	11	12	13	14	15
Delay Factor	1.000	1.125	1.250	1.375	1.500	1.625	1.750	1.875

— DELAY = DelayFactor x RP or

— DELAY = HW_GPMI_CTRL1_RDN_DELAY x 0.125 x RP.

Use this equation to calculate the value for HW_GPMI_CTRL1_RDN_DELAY. Then set HW_GPMI_CTRL1_DLL_ENABLE=1.

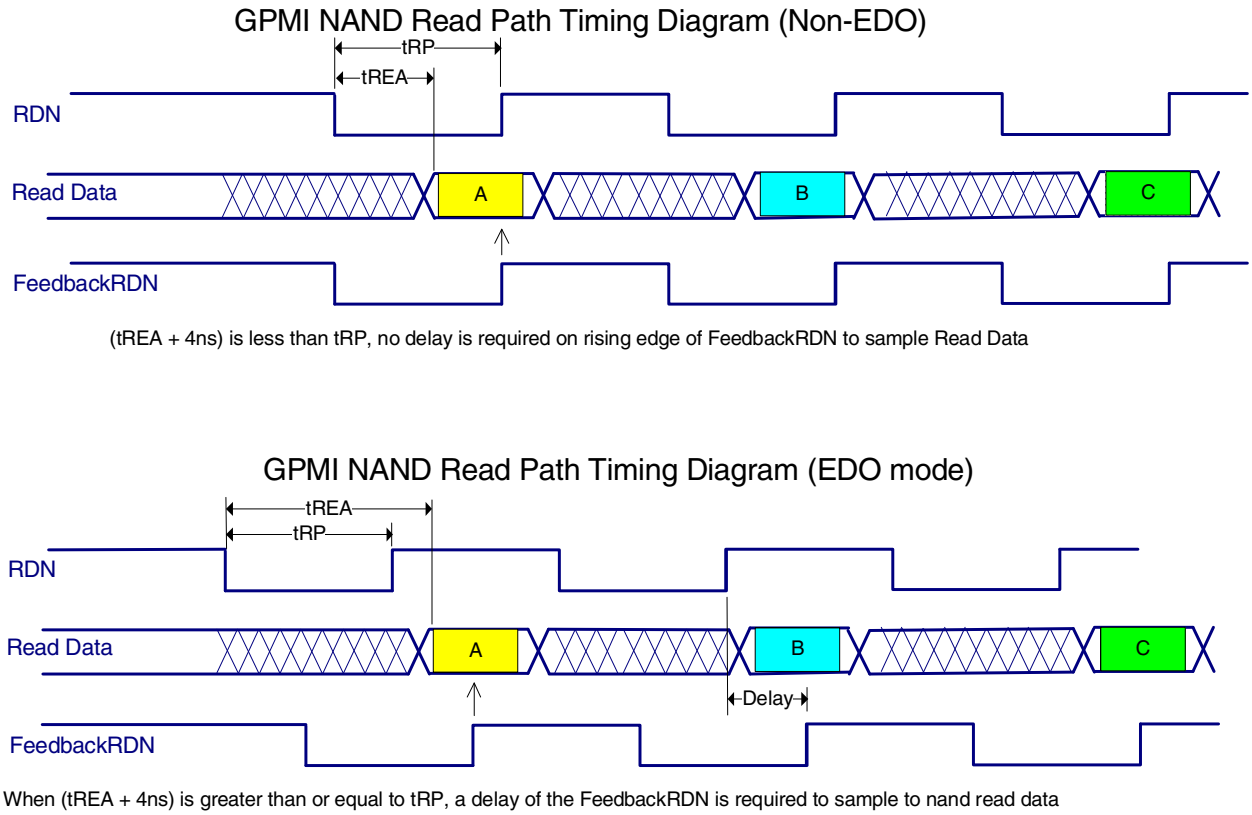


Figure 13-3. NAND Flash Read Path Timing

For example, a NAND Flash with tREAmax=20ns, tRPmin=12ns, and tRCmin=25ns (read cycle time) may be programmed as follows:

- GPMICLK clock frequency: Consider 480/6=80MHz which is 12.5ns clock period. This is too close to the minimum NAND Flash spec if we program the data setup and hold to 1 GPMICLK cycle. Consider 480/7=68.57MHz which is 14.58ns clock period. With data setup and hold set to 1, we have a tRP of 14.58ns and a tRC of 29.16ns (good margins).
- Since (tREA +4ns) is greater than tRP, required DELAY = tREA+4ns - tRP = 20 + 4 - 14.58ns = 9.42ns.
- HALF_PERIOD =0, since GPMICLK period is less than 16ns. So RP=GPMICLK period = 14.58ns.
- DELAY = HW_GPMI_CTRL1_RDN_DELAY x 0.125 x RP.
  9.42ns = HW_GPMI_CTRL1_RDN_DELAY x 0.125 x 14.58ns.
  HW_GPMI_CTRL1_RDN_DELAY = 5 (round off 5.169)

Note: *It is recommended that the drive strength of GPMI_RDn and GPMI_WRn output pins be set to 8 mA. This will reduce the transition time under heavy loads. Low transition times will be important when NAND Flash interface read and write cycle times are below 30 ns. The other GPMI pins may remain at 4 mA, since their frequency is only up to half that of GPMI_RDn and GPMI_WRn.*

13.2.5 NAND Flash Command and Address Timing Example

Figure 13-4 illustrates a command and address being sent to a NAND Flash.

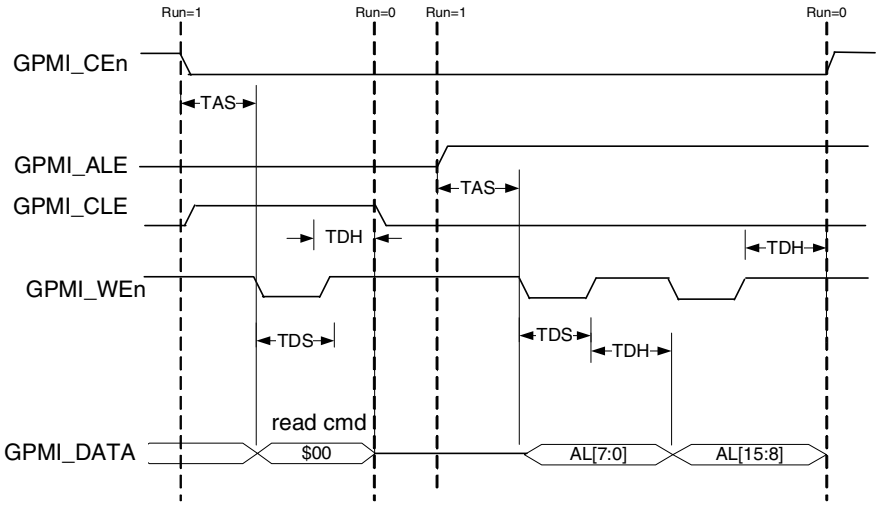


Figure 13-4. NAND Flash Command and Address Timing Example

13.2.6 Hardware BCH/ECC (ECC8) Interface

The GPMI provides an interface to the ECC8 module. This same interface is used by the BCH module when in BCH mode. This reduces the SoC bus traffic and the software involvement. When in BCH or

ECC8 mode, parity information is inserted on-the-fly during writes to 8-bit NAND Flash devices. (Note that ECC8 mode is not avaiable with 16-bit devices.) During NAND Flash reads, parity is checked and ECC processing is performed after each read block.

In ECC8 mode, during NAND Flash writes, each 512-byte block of payload data is sent to the ECC8 module at the same time it is sent to the NAND Flash. The ECC8 module returns the parity information, which is then appended to the block of data written to the NAND Flash. This is repeated for each block of data written to the NAND Flash. During NAND Flash reads, each block of payload data and parity is redirected to the ECC8 module for ECC processing and memory write, instead of DMA to memory. This works the same way for BCH mode.

To program the ECC8 for NAND Flash writes, remove the soft reset and clock gates from HW_ECC8_CTRL_SFTRST and HW_ECC8_CTRL_CLKGATE. The bulk ECC8 programming is actually applied to the GPMI via PIO operations embedded in its DMA command structures. This has a subtle implication when writing to the GPMI ECC8 registers: access to the these registers must be written in progressive register order. Thus, to write to the HW_GPMI_ECCCOUNT register, write first (in order) to registers HW_GPMI_CTRL0, HW_GPMI_COMPARE, and HW_GPMI_ECCCTRL before writing to HW_GPMI_ECCCOUNT. These additional register writes need to be accounted for in the CMDWORDS field of the respective DMA channel command register. See the descriptive text, flowcharts, and code examples in Section 14.2.1, “Reed-Solomon ECC Accelerator,” Section 14.2.2, “Reed-Solomon ECC Encoding for NAND Writes,” and Section 14.2.3, “Reed-Solomon ECC Decoding for NAND Reads” for more information about using GPMI registers to program the ECC8 function.

Note that the HW_GPMI_PAYLOAD and HW_GPMI_AUXILIARY pointers need to be word-aligned for proper ECC8 operation. If those pointers are non-word-aligned, then the ECC8 engine will not operate properly and could possibly corrupt system memory in the adjoining memory regions.

Note that when using DMA to read from the NAND, there are two possible interrupts:

- GPMI DMA completion interrupt
- ECC engine completion interrupt (only used when the data is read using ECC in the DMA descriptor)

When the NAND is read using ECC (as configured in the DMA descriptor for that read), the CPU must wait for **both** interrupts, and the interrupts may occur in either order. That is, the GPMI interrupt may happen before the ECC interrupt, or vice-versa. Software needs to wait until both interrupts have occurred to know that the data has been delivered by the DMA.

When the NAND is read without using ECC (as configured in the DMA descriptor for that read), the CPU only needs to wait for the GPMI interrupt. (Indeed, there will be no ECC interrupt, because the ECC engine is not in use.).

13.3 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, “Correct Way to Soft Reset a Block” for additional information on using the SFTRST and CLKGATE bit fields.

13.4 Programmable Registers

The following registers provide control for programmable elements of the GPMI module.

13.4.1 GPMI Control Register 0 Description

The GPMI control register 0 specifies the GPMI transaction to perform for the current command chain item.

HW_GPMI_CTRL0	0x000
HW_GPMI_CTRL0_SET	0x004
HW_GPMI_CTRL0_CLR	0x008
HW_GPMI_CTRL0_TOG	0x00C

Table 13-2. HW_GPMI_CTRL0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	RUN	RSVD3	TIMEOUT_IRQ_EN	RSVD2	COMMAND_MODE		WORD_LENGTH	LOCK_CS	CS		RSVD1			ADDRESS_INCREMENT	XFER_COUNT															

Table 13-3. HW_GPMI_CTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	Set to zero for normal operation. When this bit is set to one (default), then the entire block is held in its reset state. This will not work if the CLKGATE bit is already set to '1'. CLKGATE must be cleared to '0' before issuing a soft reset. Also the GPMICLK must be running for this to work properly. RUN = 0x0 Allow GPMI to operate normally. RESET = 0x1 Hold GPMI in reset.
30	**CLKGATE**	RW	0x1	Set this bit zero for normal operation. Setting this bit to one (default), gates all of the block level clocks off for miniminizing AC energy consumption. RUN = 0x0 Allow GPMI to operate normally. NO_CLKS = 0x1 Do not clock GPMI gates in order to minimize power consumption.

Table 13-3. HW_GPMI_CTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
29	**RUN**	RW	0x0	The GPMI is busy running a command whenever this bit is set to '1'. The GPMI is idle whenever this bit set to zero. This can be set to one by a CPU write. In addition, the DMA sets this bit each time a DMA command has finished its PIO transfer phase. IDLE = 0x0 The GPMI is idle. BUSY = 0x1 The GPMI is busy running a command.
28	**RSVD3**	RW	0x0	Program this field to 0x0.
27	**TIMEOUT_IRQ_EN**	RW	0x0	Setting this bit to '1' will enable timeout IRQ for WAIT_FOR_READY commands. The Device_Busy_Timeout value is used for this timeout.
26	**RSVD2**	RW	0x0	Program this field to 0x0. DISABLED = 0x0 Use ATA-PIO mode on the external bus. ENABLED = 0x1 Use ATA-Ultra DMA mode on the external bus.
25:24	**COMMAND_MODE**	RW	0x0	00= Write mode. 01= Read Mode. 10= Read and Compare Mode (setting sense flop). 11= Wait for Ready. WRITE = 0x0 Write mode. READ = 0x1 Read mode. READ_AND_COMPARE = 0x2 Read and Compare mode (setting sense flop). WAIT_FOR_READY = 0x3 Wait for Ready mode.
23	**WORD_LENGTH**	RW	0x0	0= 16-bit Data Bus Mode. 1= 8-bit Data Bus mode. This bit should only be changed when RUN==0. 16_BIT = 0x0 16-bit Data Bus Mode. 8_BIT = 0x1 8-bit Data Bus mode.
22	**LOCK_CS**	RW	0x0	For NAND mode: 0= Deassert chip select (CS) after RUN is complete. 1= Continue to assert chip select (CS) after RUN is complete. For Camera Mode: 0= Dont wait for VSYNC rising edge before capturing data. 1= Wait for VSYNC rising edge before capturing data (Camera mode only). DISABLED = 0x0 Deassert chip select (CS) after RUN is complete. ENABLED = 0x1 Continue to assert chip select (CS) after RUN is complete.
21:20	**CS**	RW	0x0	Selects which chip select is active for this command. For WAIT_FOR_READY command, this must be set to b01.
19:17	**RSVD1**	RW	0x0	Program this field to 0x0. NAND_DATA = 0x0 In NAND mode, this address is used to read and write data bytes. NAND_CLE = 0x1 In NAND mode, this address is used to write command bytes. NAND_ALE = 0x2 In NAND mode, this address is used to write address bytes.
16	**ADDRESS_INCREMENT**	RW	0x0	0= Address does not increment. 1= Increment address. In ATA mode, the address will increment with each cycle. In NAND mode, the address will increment once, after the first cycle (going from CLE to ALE). DISABLED = 0x0 Address does not increment. ENABLED = 0x1 Increment address.
15:0	**XFER_COUNT**	RW	0x0	Number of words (8 or 16 bit wide) to transfer for this command. A value of zero will transfer 64K words.

DESCRIPTION:

The GPMI control register 0 specifies the GPMI transaction to perform for the current command chain item.

EXAMPLE:

```
No Example.
```

13.4.2 GPMI Compare Register Description

The GPMI compare register specifies the expect data and the xor mask for comparing to the status values read from the device. This register is used by the Read and Compare command.

HW_GPMI_COMPARE	0x010

Table 13-4. HW_GPMI_COMPARE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MASK																REFERENCE															

Table 13-5. HW_GPMI_COMPARE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**MASK**	RW	0x0000	16-bit mask which is applied after the read data is XORed with the REFERENCE bit field.
15:0	**REFERENCE**	RW	0x0000	16-bit value which is XORed with data read from the NAND device.

DESCRIPTION:

The GPMI compare register specifies the expect data and the xor mask for comparing to the status values read from the device.

This register is used by the Read and Compare command.

EXAMPLE:

```
No Example.
```

13.4.3 GPMI Integrated ECC Control Register Description

The GPMI ECC control register handles configuration of the integrated ECC accelerator.

HW_GPMI_ECCCTRL	0x020
HW_GPMI_ECCCTRL_SET	0x024
HW_GPMI_ECCCTRL_CLR	0x028
HW_GPMI_ECCCTRL_TOG	0x02C

Table 13-6. HW_GPMI_ECCCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
HANDLE																RSVD2	ECC_CMD		ENABLE_ECC	RSVD1				BUFFER_MASK							

Table 13-7. HW_GPMI_ECCCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**HANDLE**	RW	0x0	This is a register available to software to attach an identifier to a transaction in progress. This handle will be available from the ECC register space when the completion interrupt occurs.
15	**RSVD2**	RO	0x0	Always write zeroes to this bit field.
14:13	**ECC_CMD**	RW	0x0	ECC Command information. For Reed-Solomon ECC, this value only controls payload correction, the auxiliary area is always covered by 4-bit mode. Note that when using BCH ECC, bit 1 has no effect, and bit 0 determines Encode (0x1) and Decode (0x0). DECODE_4_BIT = 0x0 Reed-Solomon Decode in 4-bit Mode, BCH Decode ENCODE_4_BIT = 0x1 Reed-Solomon Encode in 4-bit Mode, BCH Encode DECODE_8_BIT = 0x2 Reed-Solomon Decode in 8-bit Mode, BCH Decode ENCODE_8_BIT = 0x3 Reed-Solomon Encode in 8-bit Mode, BCH Encode
12	**ENABLE_ECC**	RW	0x0	Enable ECC processing of GPMI transfers. ENABLE = 0x1 Use integrated ECC for read and write transfers. DISABLE = 0x0 Integrated ECC remains in idle.

Table 13-7. HW_GPMI_ECCCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
11:9	**RSVD1**	RO	0x00	Always write zeroes to this bit field.
8:0	**BUFFER_MASK**	RW	0x000	ECC Command information. Single or multiple buffers may be accessed per command. When multiple buffers are accessed, each buffer to be accessed must be adjacent to the next buffer being accessed. When ECC_CMD indicates 8-bit mode, this means that the BUFFER_MASK bits must not contain any zeros between ones. For example, 0b111100000 is valid (4 contiguous buffers), but 0b100110011 is invalid (because buffers 2, 3, 6, and 7 are skipped over). When ECC_CMD indicates 4-bit mode, bits 4 to 7 are not used and must be set to 0x0 so that Buffer 3 will be adjacent to the Auxiliary buffer. For example, 0b100001110 is valid (4 contiguous buffers), but 0b100000111 is invalid (because Buffer 3 is skipped over). Invalid buffer mask values will cause improper and undefined system behavior. The BCH error correction only allows two configurations of the buffer mask - software may either read just the first block on the flash page or the entire flash page. Write operations must be for the entire flash page. BCH_AUXONLY = 0x100 Set to request transfer from only the auxiliary buffer (block 0 on flash). BCH_PAGE = 0x1FF Set to request transfer to/from the entire page. AUXILIARY = 0x100 Set to request transfer to/from the auxiliary buffer. BUFFER7 = 0x080 Set to request transfer to/from data buffer 7. BUFFER6 = 0x040 Set to request transfer to/from data buffer 6. BUFFER5 = 0x020 Set to request transfer to/from data buffer 5. BUFFER4 = 0x010 Set to request transfer to/from data buffer 4. BUFFER3 = 0x008 Set to request transfer to/from data buffer 3. BUFFER2 = 0x004 Set to request transfer to/from data buffer 2. BUFFER1 = 0x002 Set to request transfer to/from data buffer 1. BUFFER0 = 0x001 Set to request transfer to/from data buffer 0.

DESCRIPTION:

The GPMI ECC control register handles configuration of the integrated ECC accelerator.

EXAMPLE:

```
No Example.
```

13.4.4 GPMI Integrated ECC Transfer Count Register Description

The GPMI ECC Transfer Count Register contains the count of bytes that flow through the ECC subsystem.

HW_GPMI_ECCCOUNT 0x030

Table 13-8. HW_GPMI_ECCCOUNT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2																COUNT															

Table 13-9. HW_GPMI_ECCCOUNT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSVD2**	RO	0x0000	Always write zeroes to this bit field.
15:0	**COUNT**	RW	0x0000	Number of bytes to pass through ECC. This is the GPMI transfer count plus the syndrome count that will be inserted into the stream by the ECC. In DMA2ECC_MODE this count must match the HW_GPMI_CTRL0_XFER_COUNT. A value of zero will transfer 64K words.

DESCRIPTION:

The GPMI ECC Transfer Count Register contains the count of bytes that flow through the ECC subsystem.

EXAMPLE:

```
No Example.
```

13.4.5 GPMI Payload Address Register Description

The GPMI payload address register specifies the location of the data buffers in system memory. This value must be word aligned.

HW_GPMI_PAYLOAD 0x040

Table 13-10. HW_GPMI_PAYLOAD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																														RSVD0	

Table 13-11. HW_GPMI_PAYLOAD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:2	**ADDRESS**	RW	0x00000000	Pointer to an array of one or more 512 byte payload buffers.
1:0	**RSVD0**	RO	0x0	Always write zeroes to this bit field.

DESCRIPTION:

The GPMI payload address register specifies the location of the data buffers in system memory. This value must be word aligned.

EXAMPLE:

```
No Example.
```

13.4.6 GPMI Auxiliary Address Register Description

The GPMI auxiliary address register specifies the location of the auxiliary buffers in system memory. This value must be word aligned.

HW_GPMI_AUXILIARY	0x050

Table 13-12. HW_GPMI_AUXILIARY

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDRESS																														RSVD0	

Table 13-13. HW_GPMI_AUXILIARY Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:2	**ADDRESS**	RW	0x00000000	Pointer to ECC control structure and meta-data storage.
1:0	**RSVD0**	RO	0x0	Always write zeroes to this bit field.

DESCRIPTION:

The GPMI auxiliary address register specifies the location of the auxiliary buffers in system memory. This value must be word aligned.

EXAMPLE:

```
No Example.
```

13.4.7 GPMI Control Register 1 Description

The GPMI control register 1 specifies additional control fields that are not used on a per-transaction basis.

HW_GPMI_CTRL1	0x060
HW_GPMI_CTRL1_SET	0x064
HW_GPMI_CTRL1_CLR	0x068
HW_GPMI_CTRL1_TOG	0x06C

Table 13-14. HW_GPMI_CTRL1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD3								CE3_SEL	CE2_SEL	CE1_SEL	CE0_SEL	GANGED_RDYBUSY	BCH_MODE	DLL_ENABLE	HALF_PERIOD	RDN_DELAY				DMA2ECC_MODE	RSVD2	TIMEOUT_IRQ	BURST_EN	ABORT_WAIT_FOR_READY3	ABORT_WAIT_FOR_READY2	ABORT_WAIT_FOR_READY1	ABORT_WAIT_FOR_READY0	DEV_RESET	ATA_IRQRDY_POLARITY	RSVD1	GPMI_MODE

Table 13-15. HW_GPMI_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD3**	RO	0x0	Always write zeroes to this bit field.
23	**CE3_SEL**	RW	0x0	This field is NOT implemented. Set this bit to 1 to use the alternate Chip Enable for GPMI_CE3n, and deasert GPMI_CE3n. When set to 0, the alternate Chip Enable (GPMI_CE7n) is deasserted and GPMI_CE3n is active during transfers.
22	**CE2_SEL**	RW	0x0	This field is NOT implemented. Set this bit to 1 to use the alternate Chip Enable for GPMI_CE2n, and deasert GPMI_CE2n. When set to 0, the alternate Chip Enable (GPMI_CE6n) is deasserted and GPMI_CE2n is active during transfers.
21	**CE1_SEL**	RW	0x0	This field is NOT implemented. Set this bit to 1 to use the alternate Chip Enable for GPMI_CE1n, and deasert GPMI_CE1n. When set to 0, the alternate Chip Enable (GPMI_CE5n) is deasserted and GPMI_CE1n is active during transfers.
20	**CE0_SEL**	RW	0x0	This field is NOT implemented. Set this bit to 1 to use the alternate Chip Enable for GPMI_CE0n, and deasert GPMI_CE0n. When set to 0, the alternate Chip Enable (GPMI_CE4n) is deasserted and GPMI_CE0n is active during transfers.
19	**GANGED_RDYBUSY**	RW	0x0	Set this bit to 1 will force all Nand RDY_BUSY inputs to be sourced from (tied to) RDY_BUSY0. This will free up all, except one, RDY_BUSY input pins.
18	**BCH_MODE**	RW	0x0	This bit selects which error correction unit will access GPMI. 1 = BCH, 0 = ECC8.
17	**DLL_ENABLE**	RW	0x0	Set this bit to 1 to enable the GPMI DLL. This is required for fast NAND reads (above 30MHz read strobe). After setting this bit, wait 64 GPMI clock cycles for the DLL to lock before performing a NAND read.
16	**HALF_PERIOD**	RW	0x0	Set this bit to 1 if the GPMI clock period is greater than 16ns for proper DLL operation. DLL_ENABLE must be zero while changing this field.

Table 13-15. HW_GPMI_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:12	**RDN_DELAY**	RW	0x0	This variable is a factor in the calculated delay to apply to the internal read strobe for correct read data sampling. The applied delay (AD) is between 0 and 1.875 times the reference period (RP). RP is one half of the GPMI clock period if HALF_PERIOD=1 otherwise it is the full GPMI clock period. The equation is: AD = RDN_DELAY x 0.125 x RP. This value must not exceed 16ns. This variable is used to achieve faster NAND access. For example if the Read Strobe is asserted from time 0 to 13ns but the read access time is 20ns, then choose AD=12ns will cause the data to be sampled at time 25ns (13+12) giving a 5ns data setup time. If RP=13ns then RDN_DELAY = 12/(0.125 x 13ns) = 7.38 (0111b). DLL_ENABLE must be zero while changing this field.
11	**DMA2ECC_MODE**	RW	0x0	This is mainly for testing HWECC without involving the Nand device. Setting this bit will cause DMA write data to redirected to HWECC module (instead of Nand Device) for encoding or decoding.
10	**RSVD2**	RW	0x0	Program this field to 0x0.
9	**TIMEOUT_IRQ**	RW	0x0	This bit is set when a timeout occurs using the Device_Busy_Timeout value. Write 0 to clear.
8	**BURST_EN**	RW	0x0	When set to 1 each DMA request will generate a 4-transfer burst on the APB bus.
7	**ABORT_WAIT_FOR_READY3**	RW	0x0	Abort a wait for ready command on channel 3.
6	**ABORT_WAIT_FOR_READY2**	RW	0x0	Abort a wait for ready command on channel 2.
5	**ABORT_WAIT_FOR_READY1**	RW	0x0	Abort a wait for ready command on NAND channel 1.
4	**ABORT_WAIT_FOR_READY0**	RW	0x0	Abort a wait for ready command on channel 0.
3	**DEV_RESET**	RW	0x0	0= Device Reset pin is held low (asserted). 1= Device Reset pin is held high (de-asserted). ENABLED = 0x0 Device Reset pin is held low (asserted). DISABLED = 0x1 Device Reset pin is held high (de-asserted).
2	**ATA_IRQRDY_POLARITY**	RW	0x1	For NAND MODE: 0= External RDY_BUSY[1] and RDY_BUSY[0] pins are ready when low and busy when high. 1= External RDY_BUSY[1] and RDY_BUSY[0] pins are ready when high and busy when low. Note NAND_RDY_BUSY[3:2] are not affected by this bit. ACTIVELOW = 0x0 NAND_RDY_BUSY[1:0] are active low ready. ACTIVEHIGH = 0x1 NAND_RDY_BUSY[1:0] are active high ready.
1	**RSVD1**	RW	0x0	Program this field to 0x0.
0	**GPMI_MODE**	RW	0x0	0= NAND mode. 1= RESERVED

DESCRIPTION:

The GPMI control register 1 specifies additional control fields that are not used on a per-transaction basis.

EXAMPLE:

```
No Example.
```

13.4.8 GPMI Timing Register 0 Description

The GPMI timing register 0 specifies the timing parameters that are used by the cycle state machine to guarantee the various setup, hold and cycle times for the external media type.

HW_GPMI_TIMING0 0x070

Table 13-16. HW_GPMI_TIMING0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1								ADDRESS_SETUP								DATA_HOLD								DATA_SETUP							

Table 13-17. HW_GPMI_TIMING0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD1**	RO	0x0	Always write zeroes to this bit field.
23:16	**ADDRESS_SETUP**	RW	0x01	Number of GPMICLK cycles that the CE signals are active before a strobe is asserted. A value of zero is interpreted as 0.
15:8	**DATA_HOLD**	RW	0x02	Data bus hold time in GPMICLK cycles. Also the time that the data strobe is de-asserted in a cycle. A value of zero is interpreted as 256.
7:0	**DATA_SETUP**	RW	0x03	Data bus setup time in GPMICLK cycles. Also the time that the data strobe is asserted in a cycle. A value of zero is interpreted as 256.

DESCRIPTION:

The GPMI timing register 0 specifies the timing parameters that are used by the cycle state machine to guarantee the various setup, hold and cycle times for the external media type.

EXAMPLE:

```
No Example.
```

13.4.9 GPMI Timing Register 1 Description

The GPMI timing register 1 specifies the timeouts used when monitoring the NAND READY pin.

HW_GPMI_TIMING1 0x080

Table 13-18. HW_GPMI_TIMING1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DEVICE_BUSY_TIMEOUT																RSVD1															

Table 13-19. HW_GPMI_TIMING1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**DEVICE_BUSY_TIMEOUT**	RW	0x0000	Timeout waiting for NAND Ready/Busy. Used in WAIT_FOR_READY mode. This value is the number of GPMI_CLK cycles multiplied by 4096.
15:0	**RSVD1**	RO	0x0	Always write zeroes to this bit field.

DESCRIPTION:

The GPMI timing register 1 specifies the timeouts used when monitoring the NAND READY pin.

EXAMPLE:

```
No Example.
```

13.4.10 GPMI DMA Data Transfer Register Description

The GPMI DMA data transfer register is used by the DMA to read or write data to or from the NAND control state machine.

HW_GPMI_DATA 0x0A0

Table 13-20. HW_GPMI_DATA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DATA																															

Table 13-21. HW_GPMI_DATA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**DATA**	RW	0x00000	In 16-bit mode, this register can be accessed in two 16-bit operations, one bus cycle per operation. In 8-bit mode, one, two, three or four bytes can can be accessed to send the same number of bus cycles.

DESCRIPTION:

The GPMI DMA data transfer register is used by the DMA to read or write data to or from the NAND control state machine.

EXAMPLE:

```
No Example.
```

13.4.11 GPMI Status Register Description

The GPMI control and status register provides a read back path for various operational states of the GPMI controller.

HW_GPMI_STAT 0x0B0

Table 13-22. HW_GPMI_STAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PRESENT	RSVD2																			RDY_TIMEOUT				RSVD1	INVALID_BUFFER_MASK	FIFO_EMPTY	FIFO_FULL	DEV3_ERROR	DEV2_ERROR	DEV1_ERROR	DEV0_ERROR

Table 13-23. HW_GPMI_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**PRESENT**	RO	0x1	0= GPMI is not present in this product. 1= GPMI is present is in this product. UNAVAILABLE = 0x0 GPMI is not present in this product. AVAILABLE = 0x1 GPMI is present in this product.
30:12	**RSVD2**	RO	0x0	Always write zeroes to this bit field.
11:8	**RDY_TIMEOUT**	RO	0x0	Status of the RDY/BUSY Timeout Flags.
7	**RSVD1**	RO	0x0	Program this field to 0x0.
6	**INVALID_BUFFER_MASK**	RO	0x0	0= ECC Buffer Mask is not invalid. 1= ECC Buffer Mask is invalid.
5	**FIFO_EMPTY**	RO	0x1	0= FIFO is not empty. 1= FIFO is empty. NOT_EMPTY = 0x0 FIFO is not empty. EMPTY = 0x1 FIFO is empty.
4	**FIFO_FULL**	RO	0x0	0= FIFO is not full. 1= FIFO is full. NOT_FULL = 0x0 FIFO is not full. FULL = 0x1 FIFO is full.
3	**DEV3_ERROR**	RO	0x0	0= No error condition present on NAND Device 3. 1= An Error has occurred on NAND Device 3 (Timeout or compare failure, depending on COMMAND_MODE).

Table 13-23. HW_GPMI_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**DEV2_ERROR**	RO	0x0	0= No error condition present on NAND Device 2. 1= An Error has occurred on NAND Device 2 (Timeout or compare failure, depending on COMMAND_MODE).
1	**DEV1_ERROR**	RO	0x0	0= No error condition present on NAND Device 1. 1= An Error has occurred on NAND Device 1 (Timeout or compare failure, depending on COMMAND_MODE).
0	**DEV0_ERROR**	RO	0x0	0= No error condition present on NAND Device 0. 1= An Error has occurred on NAND Device 0 (Timeout or compare failure, depending on COMMAND_MODE).

DESCRIPTION:

The GPMI control and status register provides a read back path for various operational states of the GPMI controller.

EXAMPLE:

```
No Example.
```

13.4.12 GPMI Debug Information Register Description

The GPMI debug information register provides a read back path for diagnostics to determine the current operating state of the GPMI controller.

HW_GPMI_DEBUG 0x0C0

Table 13-24. HW_GPMI_DEBUG

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
READY3	READY2	READY1	READY0	WAIT_FOR_READY_END3	WAIT_FOR_READY_END2	WAIT_FOR_READY_END1	WAIT_FOR_READY_END0	SENSE3	SENSE2	SENSE1	SENSE0	DMAREQ3	DMAREQ2	DMAREQ1	DMAREQ0	CMD_END				UDMA_STATE				BUSY	PIN_STATE			MAIN_STATE			

Table 13-25. HW_GPMI_DEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**READY3**	RO	0x0	Read-only view of Ready Line 3.
30	**READY2**	RO	0x0	Read-only view of Ready Line 2.
29	**READY1**	RO	0x0	Read-only view of Ready Line 1.
28	**READY0**	RO	0x0	Read-only view of Ready Line 0.

Table 13-25. HW_GPMI_DEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
27	**WAIT_FOR_READY_END3**	RO	0x0	Read-only view of WAIT_FOR_READY command end of channel 3. This view sees the toggle state.
26	**WAIT_FOR_READY_END2**	RO	0x0	Read-only view of WAIT_FOR_READY command end of channel 2. This view sees the toggle state.
25	**WAIT_FOR_READY_END1**	RO	0x0	Read-only view of WAIT_FOR_READY command end of channel 1. This view sees the toggle state.
24	**WAIT_FOR_READY_END0**	RO	0x0	Read-only view of WAIT_FOR_READY command end of channel 0. This view sees the toggle state.
23	**SENSE3**	RO	0x0	Read-only view of sense state of channel 3. A value of "1" indicates that a read and compare command failed or a timeout occured.
22	**SENSE2**	RO	0x0	Read-only view of sense state of channel 2. A value of "1" indicates that a read and compare command failed or a timeout occured.
21	**SENSE1**	RO	0x0	Read-only view of sense state of channel 1. A value of "1" indicates that a read and compare command failed or a timeout occured.
20	**SENSE0**	RO	0x0	Read-only view of sense state of channel 0. A value of "1" indicates that a read and compare command failed or a timeout occured.
19	**DMAREQ3**	RO	0x0	Read-only view of DMA request line for channel 3. This view sees the toggle state.
18	**DMAREQ2**	RO	0x0	Read-only view of DMA request line for channel 2. This view sees the toggle state.
17	**DMAREQ1**	RO	0x0	Read-only view of DMA request line for channel 1. This view sees the toggle state.
16	**DMAREQ0**	RO	0x0	Read-only view of DMA request line for channel 0. This view sees the toggle state.
15:12	**CMD_END**	RO	0x0	Read Only view of the Command End toggle to DMA. One per channel
11:8	**UDMA_STATE**	RO	0x0	USM_IDLE = 4'h0, idle USM_DMARQ = 4'h1, DMA req USM_ACK = 4'h2, DMA ACK USM_FIFO_E = 4'h3, Fifo empty USM_WPAUSE = 4'h4, WR DMA Paused by device USM_TSTRB = 4'h5, Toggle HSTROBE USM_CAPTUR = 4'h6, Capture Stage, (data sampled with DSTROBE is valid) USM_DATOUT = 4'h7, Change Burst DATAOUT USM_CRC = 4'h8, Source CRC to Device USM_WAIT_R = 4'h9, Waiting for DDMARDY- USM_END = 4'ha; Negate DMAACK (end of DMA) USM_WAIT_S = 4'hb, Waiting for DSTROBE USM_RPAUSE = 4'hc, Rd DMA Paused by Host USM_RSTOP = 4'hd, Rd DMA Stopped by Host USM_WTERM = 4'he, Wr DMA Termination State USM_RTERM = 4'hf, Rd DMA Termination state
7	**BUSY**	RO	0x0	When asserted the GPMI is busy. Undefined results may occur if any registers are written when BUSY is asserted. DISABLED = 0x0 The GPMI is not busy. ENABLED = 0x1 The GPMI is busy.

Table 13-25. HW_GPMI_DEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
6:4	**PIN_STATE**	RO	0x0	parameter PSM_IDLE = 3'h0, PSM_BYTCNT = 3'h1, PSM_ADDR = 3'h2, PSM_STALL = 3'h3, PSM_STROBE = 3'h4, PSM_ATARDY = 3'h5, PSM_DHOLD = 3'h6, PSM_DONE = 3'h7. PSM_IDLE = 0x0 PSM_BYTCNT = 0x1 PSM_ADDR = 0x2 PSM_STALL = 0x3 PSM_STROBE = 0x4 PSM_ATARDY = 0x5 PSM_DHOLD = 0x6 PSM_DONE = 0x7
3:0	**MAIN_STATE**	RO	0x0	parameter MSM_IDLE = 4'h0, MSM_BYTCNT = 4'h1, MSM_WAITFE = 4'h2, MSM_WAITFR = 4'h3, MSM_DMAREQ = 4'h4, MSM_DMAACK = 4'h5, MSM_WAITFF = 4'h6, MSM_LDFIFO = 4'h7, MSM_LDDMAR = 4'h8, MSM_RDCMP = 4'h9, MSM_DONE = 4'hA. MSM_IDLE = 0x0 MSM_BYTCNT = 0x1 MSM_WAITFE = 0x2 MSM_WAITFR = 0x3 MSM_DMAREQ = 0x4 MSM_DMAACK = 0x5 MSM_WAITFF = 0x6 MSM_LDFIFO = 0x7 MSM_LDDMAR = 0x8 MSM_RDCMP = 0x9 MSM_DONE = 0xA

DESCRIPTION:

The GPMI debug information register provides a read back path for diagnostics to determine the current operating state of the GPMI controller.

EXAMPLE:

```
No Example.
```

13.4.13 GPMI Version Register Description

This register reflects the version number for the GPMI.

HW_GPMI_VERSION 0x0D0

Table 13-26. HW_GPMI_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 13-27. HW_GPMI_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x03	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x00	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register reflects the version number for the GPMI.

EXAMPLE:

```
No Example.
```

13.4.14 GPMI Debug2 Information Register Description

The GPMI Debug2 information register provides a read back path for diagnostics to determine the current operating state of the GPMI controller.

HW_GPMI_DEBUG2 0x0E0

Table 13-28. HW_GPMI_DEBUG2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1																SYND2GPMI_BE				GPMI2SYND_VALID	GPMI2SYND_READY	SYND2GPMI_VALID	SYND2GPMI_READY	VIEW_DELAYED_RDN	UPDATE_WINDOW	RDN_TAP							

Table 13-29. HW_GPMI_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSVD1**	RO	0x0000	Always write zeroes to this bit field.
15:12	**SYND2GPMI_BE**	RO	0xf	Data byte enable Input from ECC8 or BCH.
11	**GPMI2SYND_VALID**	RO	0x0	Data handshake output to ECC8 or BCH.
10	**GPMI2SYND_READY**	RO	0x0	Data handshake output to ECC8 or BCH.
9	**SYND2GPMI_VALID**	RO	0x0	Data handshake Input from ECC8 or BCH.
8	**SYND2GPMI_READY**	RO	0x0	Data handshake Input from ECC8 or BCH.
7	**VIEW_DELAYED_RDN**	RW	0x0	Set to a 1 to select the delayed feedback RDN to drive the GPMI_ADDR[0] (Nand CLE) pin. For debug purposes, this will allow you see if DLL is functioning properly.

Table 13-29. HW_GPMI_DEBUG2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
6	**UPDATE_WINDOW**	RO	0x0	A 1 indicates that the DLL is busy generating the required delay.
5:0	**RDN_TAP**	RO	0x00	This is the DLL tap calculated by the DLL controller. The selects the amount of delay form the DLL chain.

DESCRIPTION:

The GPMI Debug2 information register provides a read back path for diagnostics to determine the current operating state of the GPMI controller.

EXAMPLE:

```
No Example.
```

13.4.15 GPMI Debug3 Information Register Description

The GPMI Debug3 information register provides a read back path for diagnostics to determine the current operating state of the GPMI controller.

HW_GPMI_DEBUG3 0x0F0

Table 13-30. HW_GPMI_DEBUG3

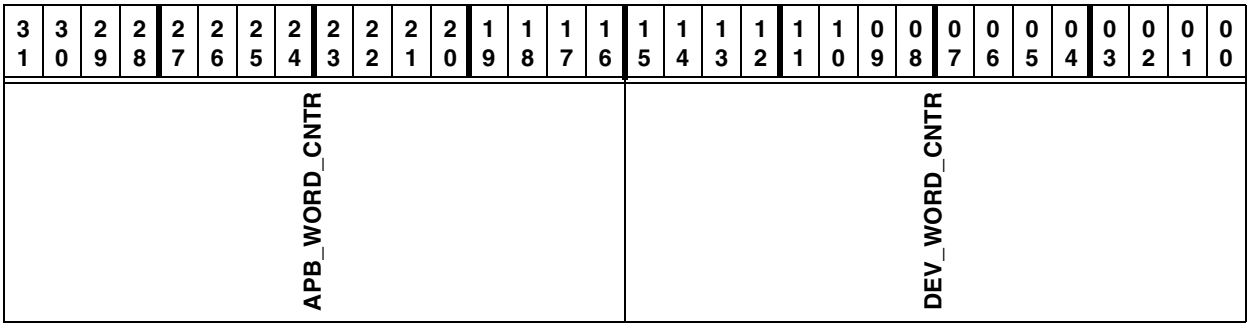


Table 13-31. HW_GPMI_DEBUG3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**APB_WORD_CNTR**	RO	0x0000	Reflects the number of words (16 or 8-bit) remains to be transferred on the APB bus.
15:0	**DEV_WORD_CNTR**	RO	0x0000	Reflects the number of words (16 or 8-bit) remains to be transferred on the Nand bus.

DESCRIPTION:

The GPMI Debug3 information register provides a read back path for diagnostics to determine the current operating state of the GPMI controller.

EXAMPLE:

```
No Example.
```

GPMI Block v3.0, Revision 2.2

13.4.16

Chapter 14
8-Symbol Correcting ECC Accelerator (ECC8)

This chapter describes the DMA-based hardware ECC accelerator (ECC8) available on the i.MX23. It provides detailed descriptions of how to use the Reed-Solomon ECC accelerator. Programmable registers are described in Section 14.4, "Programmable Registers."

14.1 Overview

The hardware ECC accelerator provides a forward error-correction function for improving the reliability of various storage media that may be attached to the i.MX23. For example, modern high-density NAND flash devices presume the existence of forward error-correction algorithms to correct some soft and/or hard bit errors within the device, allowing for higher device yields and, therefore, lower device costs.

The hardware ECC8 accelerator uses the Reed-Solomon block codes, a subset of BCH codes, for multi-symbol error corrections. A *symbol* comprises multiple bits. The ECC8 operates on 9-bit symbols in its computations. A *symbol error* (and correction) means that any one or all bits of the symbol could be in error. Thus, under a best case scenario, a 4-symbol ECC protection can correct up to 36 bits (= 4 * 9) in error. Under the worst case scenario, only 4 bits (= 4 * 1) can be corrected.

In a Reed-Solomon ECC, the fixed *data payload* to be protected is mapped into data symbols to represent a unique polynomial. The polynomial is divided by a known generator polynomial (that is a function of the number of symbols to be corrected), where the residual remainder polynomial becomes the *parity symbols*. An *ECC codeword* is formed by concatenating the data symbols with the parity symbols. This ECC codeword is then written onto the storage media. All arithmetic operations in the Reed-Solomon ECC algorithm operate under Galois fields. The ECC8 supports t=4 symbol correction for 2K page NAND and t=8 symbol correction for 4K page NANDs.

Error correction occurs when the ECC codeword is read back from the storage media through the RS decoder. The RS decoder processes the code word in four phases. All phases may not be necessary, for example when no errors are found or when uncorrectable errors are detected.

The four phases are:

1. **Syndrome Calculation Phase (SC)**—This is the process of reading in all of the symbols of the block and continuously dividing the code word by the generator polynomial that is a function of the number of symbols to be corrected. The remainder of this division is the syndrome polynomial. If the remainder is zero, i.e., the syndrome symbols are all zero, then the RS code word is correct and no bit errors were detected in the read NAND data, and an ECC8 interrupt is generated upon completion of this phase. Otherwise, we proceed to the next phase. This phase takes place

completely on the GPMI clock domain and is fully overlapped with NAND reads from the NAND device. There are approximately two GPMICLK cycles that are not overlapped. The data is passed to the HCLK domain and there are approximately 20 HCLK cycles that are not overlapped on the final block transferred.

2. **Key Equation Solver Phase (KES)**—After the eight (or sixteen) symbol syndromes have been calculated, a set of eight (or sixteen) linear equations with eight (or sixteen) unknowns is formed. The process of solving these equations and selecting from the numerous possible solutions constitutes the KES phase. The hardware block uses the Berlekamp-Massey algorithm to solve the key equations from the syndrome symbols. The resulting λ and Ω polynomials are used in the next phase to determine symbol error locations and the respective correction mask. If the hardware detects an uncorrectable scenario while computing the λ and Ω polynomials, it will terminate and report the appropriate status. This phase takes up to 288 GPMICLK and 20 HCLK cycles, with no planned DMA wait states added.
3. **Chien Search and Forney Evaluator Phase (EVAL)**—This phase takes the λ and Ω polynomials from the KES phase and uses Chien's algorithm for finding the locations of the errors based on the λ polynomial. The method basically involves substituting all 511 nine-bit symbols into the λ polynomial. All non-zero results of these substitutions represent the locations of the various symbol errors. At this point, another calculation involving the λ and Ω polynomials determines the error value or the correction to apply at the symbol in the error locations. This phase consumes approximately 550 GPMICLK cycles and no HCLK cycles, with no planned DMA wait states added.
4. **Error Correction Phase (CORR)—**The CORR phase applies any required read-modify-write cycles to the data payload and/or auxiliary payload to correct any correctable errors. An ECC8 interrupt is generated upon completion of this phase. Firmware examines the error status registers ***and then*** clears the interrupt status bit (in that order).

The ECC8 block was designed to operate in a pipelined fashion to maximize throughput. Aside from the initial latency to fill the pipeline stages, the ECC8 throughput is faster than the fastest GPMI read rate of 2 cycles/byte. Thus, the bottleneck in performing NAND reads and error corrections is the GPMI read rate. Current GPMI read rates are approximately 3 cycles/byte for the current generation of NANDs. The ECC8 block has an AHB master that allows the CPU to focus on signal processing for enhanced functionality and to operate at lower clock frequencies and voltages for improved battery life. The CPU is not directly involved in generating parity symbols, checking for errors, or correcting them.

The hardware ECC8 accelerator is illustrated in Figure 14-1.

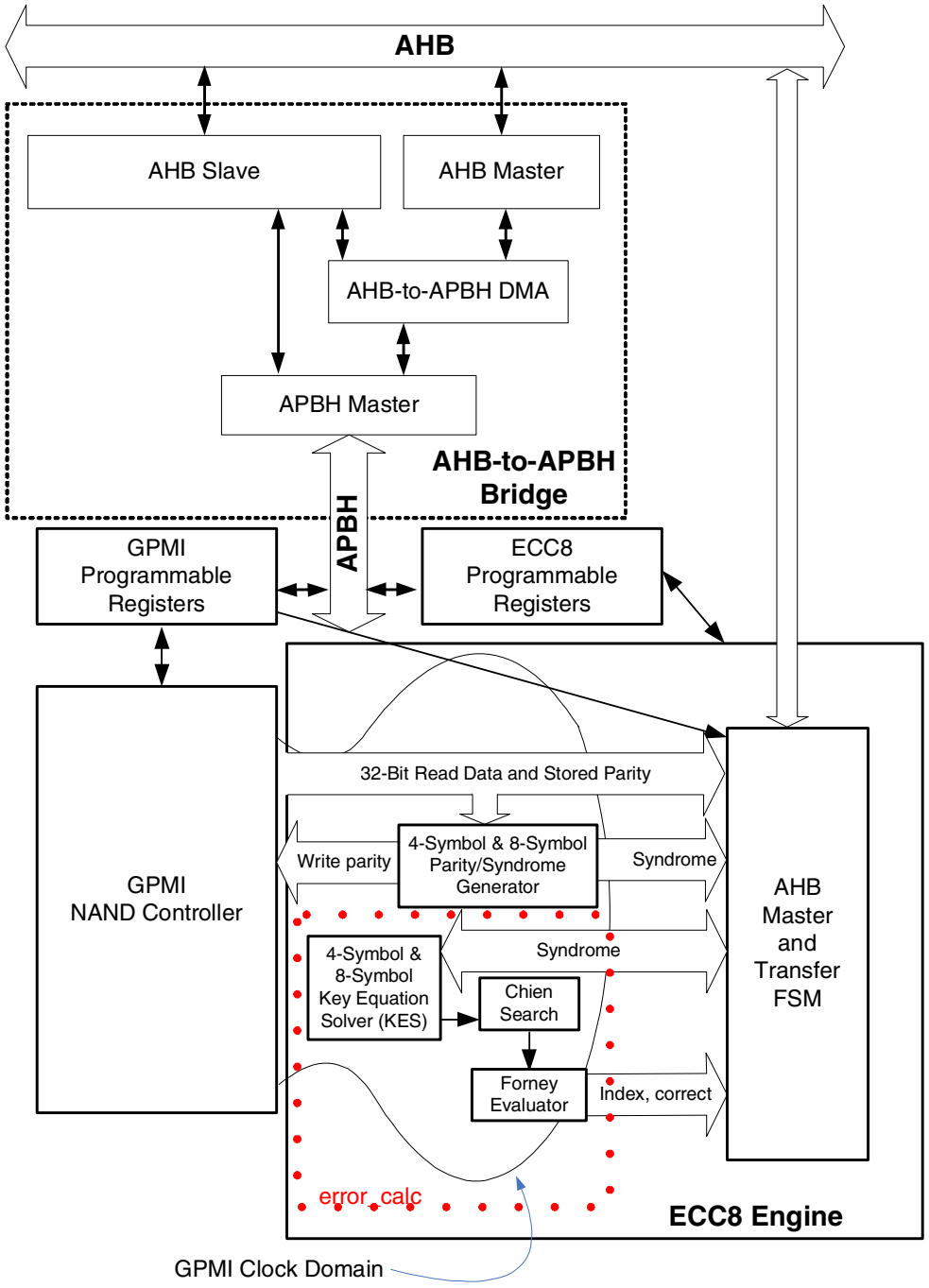


Figure 14-1. Hardware 8-Symbol Correcting ECC Accelerator (ECC8) Block

14.2 Operation

The data flow for NAND reads passes data directly from the GPMI controller to the ECC8 accelerator without first passing through system memory. This is a much higher bandwidth flow than the APBH DMA transfers used on the previous generations of SoCs. In addition, the copying to and from system memory is eliminated.

Because the ECC8 operates on data flowing directly from the GPMI, it handles all error correction operations in an optimized pipelined manner. That is, blocks without errors complete within 20 HCLK cycles of the GPMI completing the read transfer. If errors are present, then the necessary pipeline stages are activated, including error calculation and error correction stages. Unlike the previous generation HWECC, the ECC8 engine directly performs all error corrections in the system memory buffer without CPU intervention, i.e., when the CPU gets an ECC8 interrupt, the error correction process is complete for all blocks of a transaction.

A read transaction for a 4K NAND page can consist of up to 9 block transfers, i.e., eight blocks of 512-byte payload and one block of 65-byte auxiliary data.

For a 2K NAND page, up to 5 block transfers can be specified for a single transaction, i.e., four blocks of 512-byte payload and one block of 19-byte auxiliary data.

For NAND write operations, the GPMI fetches the write data via its DMA interface as usual. However, it forks a copy of the write data to the ECC8 parity/syndrome generator. The ECC8 computes the parity bytes for the transfer on the fly. As soon as the GPMI writes the last data byte to the NAND, it switches its data flow so that the 9 or 18 bytes of Reed-Solomon ECC parity is copied from the ECC8 parity/syndrome generator directly to the NAND. In this case, no extra buffer in system memory is required. The ECC parity generation is fully overlapped with the data write transfer, so that the parity bytes are written immediately after the data is written, with only a few GPMICLK cycles of latency.

The ECC8 engine supports both an 8-symbol correcting mode and a 4-symbol correcting mode. The number of parity bytes required for each mode is different. For example, the 8-symbol correcting mode requires 16 parity symbols to be stored with the data. This corresponds to 18 bytes of parity information. Since a 2K page NAND device has only 64 bytes of spare, it cannot hold the required 4*18 =72 bytes of parity data. Recall that a 2K page holds four 512-byte payload blocks. Thus, the 4-symbol correcting mode must be used for 2K page NAND devices. Fortunately, this is consistent with the bit error densities guaranteed for 2K page NAND devices.

Figure 14-2 and Figure 14-3 show the organization of the 4-symbol correcting mode 2K page NAND storage, both on the NAND and in the system memory footprint.

Figure 14-4 and Figure 14-5 show the NAND image and the system memory footprint used in the 8-symbol correcting mode available for 4K pages only. Notice that the auxiliary data is protected by the 4-symbol error correcting mode, regardless of whether it is stored on a 2K page NAND device or a 4K page NAND device.

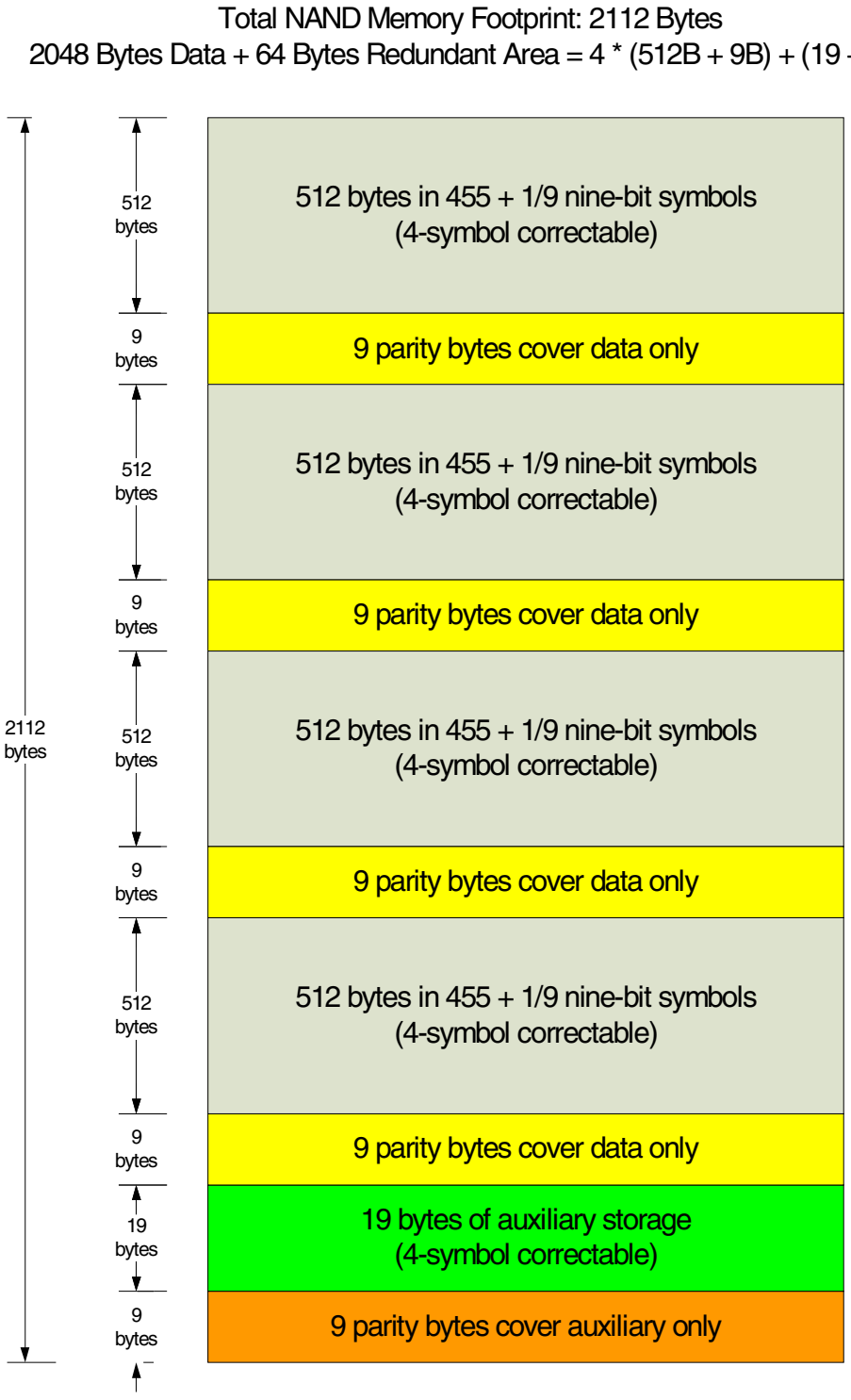


Figure 14-2. ECC-Protected 2K NAND Page Data—NAND Memory Footprint

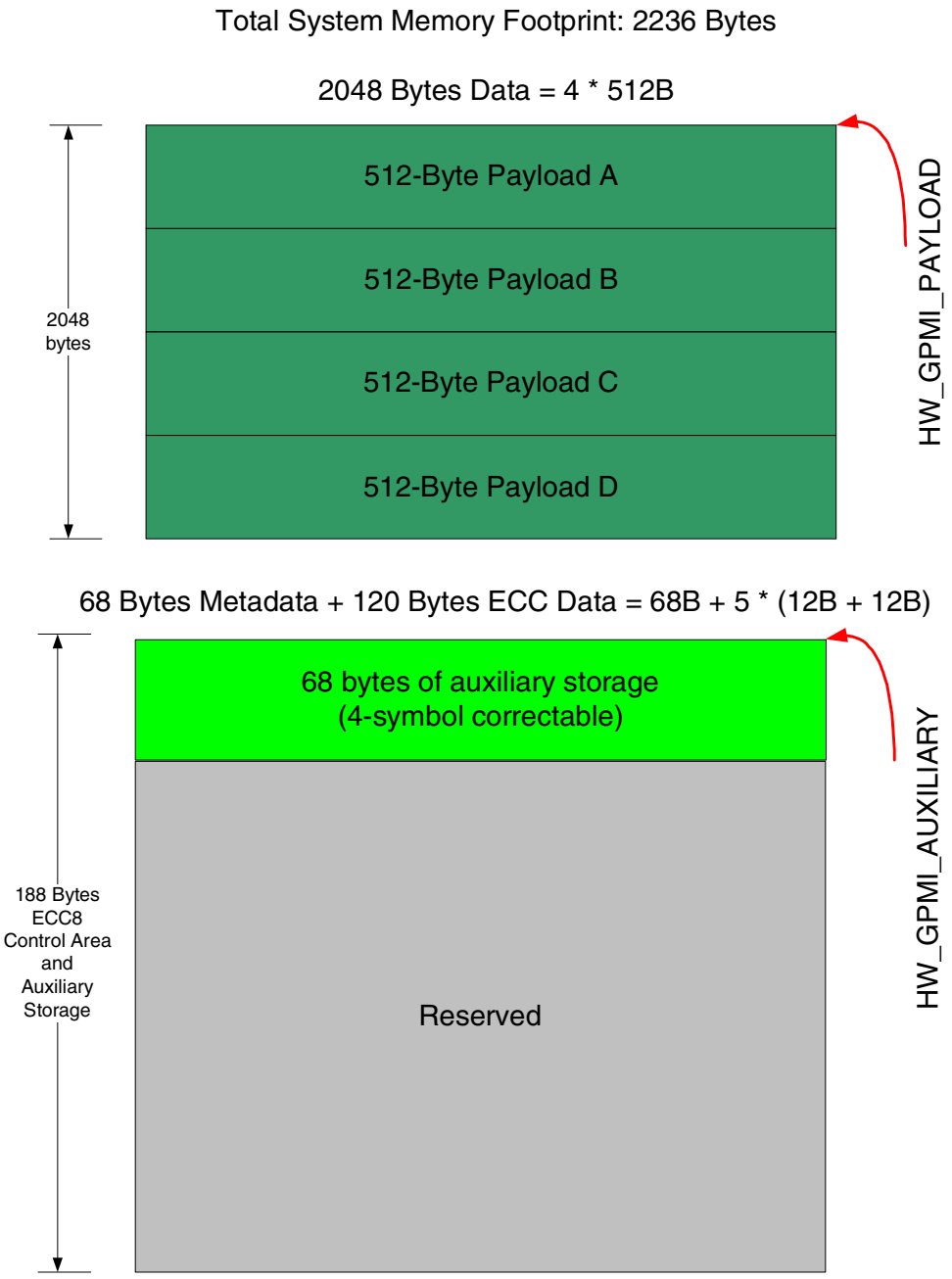


Figure 14-3. ECC-Protected 2K NAND Page Data—System Memory Footprint

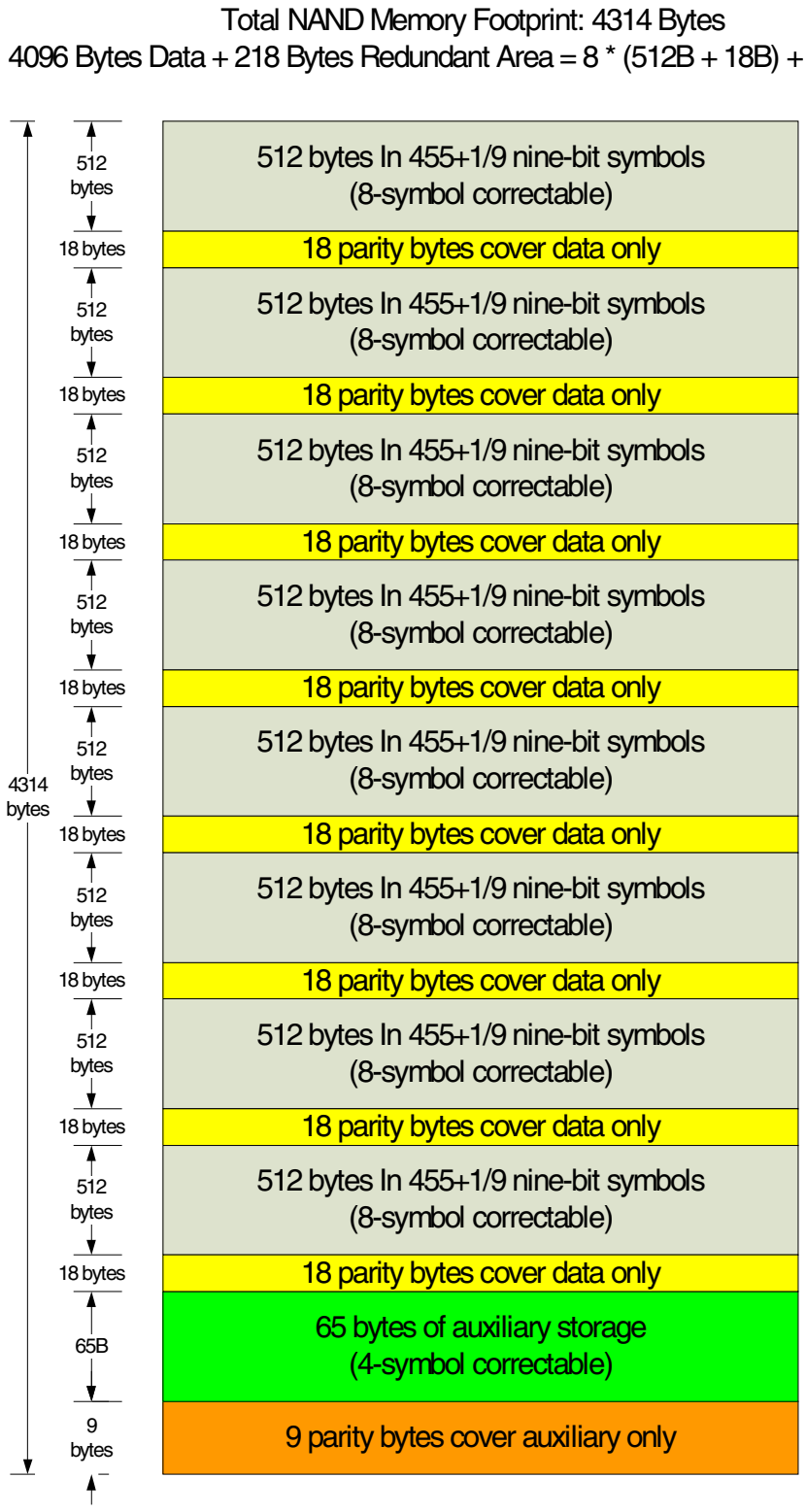


Figure 14-4. ECC-Protected 4K NAND Page Data—NAND Memory Footprint

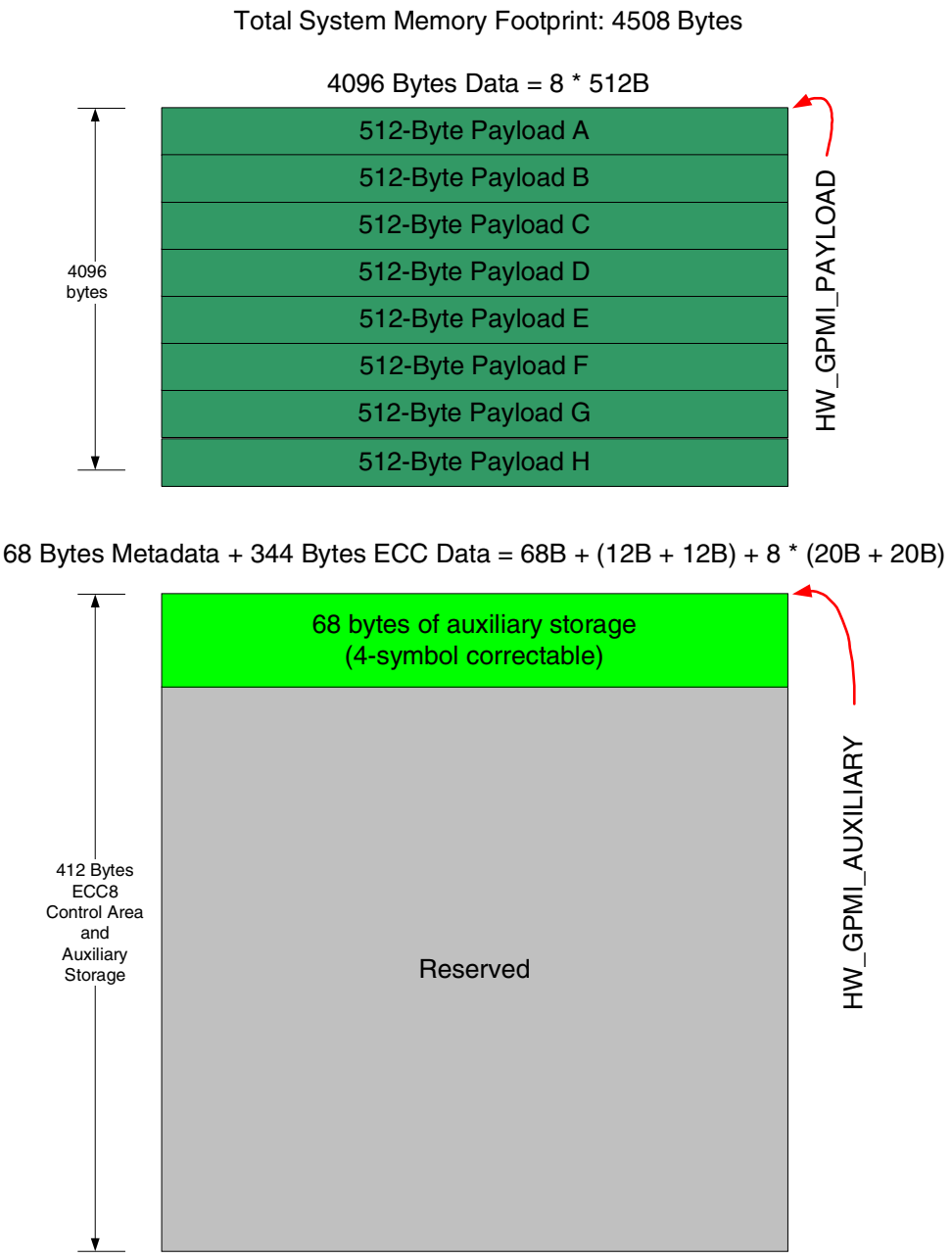


Figure 14-5. ECC-Protected 4K NAND Page Data—System Memory Footprint

14.2.1 Reed-Solomon ECC Accelerator

The Reed-Solomon algorithm used in ECC8 is capable of correcting up to 8 nine-bit symbols in a 512-byte block. Thus, up to 72 bits in error can be corrected in a 512-byte block, provided they are clustered within no more than 8 nine-bit symbols.

- 2K pages have four 512-byte data blocks (+ 9 bytes parity) and one 19-byte auxiliary block (+ 9 bytes parity).

- 4K pages have eight 512-byte data blocks (+ 18 bytes parity) and one 65-byte auxiliary block (+ 9 bytes parity).

To understand how the Reed-Solomon algorithm is implemented on the i.MX23, consider the case where there are eight 512-byte data blocks located in the on-chip RAM that need to be written to a NAND flash device. Further, assume that there is a 65-byte metadata block that needs to be written to the NAND device. Further assume that the NAND is a 4K page device.

Normal DMA channel command word processing in the APBH DMA allows buffers to start on arbitrary byte boundaries within system memory, i.e., buffers are byte-aligned. In operation with the ECC8 engine, the DMA channel command word processing requires the buffers to start on word boundaries within system memory. Specifically, the HW_GPMI_PAYLOAD and HW_GPMI_AUXILIARY pointers need to be word-aligned for proper ECC8 operation. If those pointers are non-word-aligned, then the ECC8 engine will not operate properly and could possibly corrupt system memory in the adjoining memory regions.

Assume that the data is stored in system memory in the layout shown in the memory foot print of Figure 14-5. (Note that the data residing in system memory needs to be word-aligned.) In programming the GPMI to write to the NAND, the DMA must be programmed with two DMA descriptors: one that points to the beginning of the PAYLOAD data area and a second to point to the AUXILIARY metadata block. This programming is set up exactly as for the previous generation's version of the GPMI. Program the GPMI to write these blocks to the NAND, and in addition, program the GPMI to run in ECC8 write mode by setting the following:

```
HW_GPMI_ECCCTRL = BV_FLD(GPMI_ECCCTRL,ECC_CMD,ENCODE_8_BIT)|
                  BV_FLD(GPMI_ECCCTRL,ENABLE_ECC,ENABLE)|
                  BF_GPMI_ECCCTRL_BUFFER_MASK (0x1FF);
HW_GPMI_ECCCOUNT =BF_GPMI_ECCCOUNT_COUNT (8*(512+18) + (65+9))
```

NOTE: The buffer mask value is used to specify which data blocks and/or auxiliary block is involved in a transaction. The buffer mask must be contiguous i.e., the data blocks and/or auxiliary block need to be consecutive. For example, a transaction involving only data blocks 0, 1, and 2 (buffer mask value = 0x007) is legal, while a transaction of data blocks 1, 2, 4, 6, plus the auxiliary block (buffer mask value = 0x155) is illegal. Illegal buffer mask values will cause improper and undefined system behavior.

Set the first DMA command transfer size to (8*512) bytes. Set the second DMA command transfer size to 65 bytes.

In this mode, the GPMI and the ECC8 collaborate to compute the 16-symbol (18-byte) parity values that must be written to the NAND at the end of each of the eight payload data blocks. In addition, the ECC8 calculates the 8-symbol (9-byte) parity value to be appended to the 65-byte metadata block on the NAND device.

Programming the ECC8 module for NAND writes consists of clearing the soft reset and clock gates bits (HW_ECC8_CTRL_SFTRST and HW_ECC8_CTRL_CLKGATE) as well as configuring the interrupt enables. The bulk of the programming is actually applied to the GPMI via PIO operations embedded in its

DMA command structures. This has a subtle implication when writing to the GPMI ECC8 registers: access to the ECC8 registers must be written in progressive register order. Thus, to write to the HW_GPMI_ECCCOUNT register, write first (in order) to registers HW_GPMI_CTRL0, HW_GPMI_COMPARE, and HW_GPMI_ECCCTRL before writing to HW_GPMI_ECCCOUNT. These additional register writes need to be accounted for in the CMDWORDS field of the respective DMA channel command register. See Section 13.4, “Programmable Registers” for the GPMI register descriptions.

When the DMA commands complete, the 4K NAND page will have been written in the format shown in Figure 14-4. Except for diagnostic operations, normal transfers would never read or write the NAND in any mode other than its ECC mode. It is possible to bypass the parity generation and write “RAW” data to the NAND by not turning on the ECC functions.

To summarize the detailed operation, an 18-byte Reed-Solomon parity field is appended in a 4K page at the end of each of the eight 512-byte blocks. Notice that 4K NAND devices have 4096 byte data areas plus 218-byte spare area for each “4KB” NAND flash page. In addition, a 9-byte parity field is written to the end of the 65-byte metadata block.

Assume that the GPMI media interface is used to write the resultant ((8*512)+65) bytes of data from on-chip memory to the NAND flash device. The GPMI and ECC8 then collaborate to generate an additional ((8*18)+9) bytes of parity information.

- Channel commands in APBH DMA Channels 4, 5, 6, or 7 are used to point to the data block in either on-chip or off-chip RAM (as shown in Figure 14-8).

To program the GPMI and ECC8 to read that same 4K page of NAND data back from the NAND to a buffer in the system memory, first reserve a system memory buffer like the one depicted in Figure 14-5. (Note that the reserved system memory buffers need to be word-aligned.) The GPMI DMA engine is not used for the data transfer, see below. Instead, it is used to convey a sequence of commands to the GPMI as DMA PIO operations. Some of the information conveyed to the GPMI is made available to the ECC8 engine to process the NAND read.

In particular, the address of the PAYLOAD BUFFER and the address of the AUXILIARY buffer are written to the GPMI PIO space, not to the ECC8 PIO space. Thus, the normal multi-NAND DMA based device interleaving is preserved, i.e., four NANDs on four separate chip selects can be scheduled for read or write operations using the ECC8. Whichever channel finishes its ready wait first and enters the DMA arbiter with its lock bit set will “own” the GPMI command interface and through it will own the ECC8 resources for the duration of its processing.

So, a nearly standard read DMA descriptor chain is used for the NAND read transfer, including the ready wait commands. The DMA command that kicks off the GPMI has a few extra PIO words attached to preload the HW_GPMI_ECCCTRL, HW_GPMI_ECCCOUNT, HW_GPMI_PAYLOADm and HW_GPMI_AUXILIARY registers.

When the data is read from the NAND by the GPMI, it is passed to the ECC8. Inside the ECC8, the data is copied to the payload buffer or auxiliary buffer using the AHB bus master in the ECC8. The ECC8 needs some work space in system memory to hold intermediate results. These elements are allocated in the auxiliary buffer pointed to by HW_GPMI_AUXILIARY.

Notice that programming the ECC8 for NAND reads consists largely of removing the soft reset and clock gates from HW_ECC8_CTRL and clearing the HW_ECC8_CTRL_COMPLETE_IRQ, since most of the actual programming is accomplished through PIO operations included in GPMI DMA command structures.

Set HW_ECC8_CTRL_COMPLETE_IRQ_EN to one, then start the GPMI's DMA, and let it run. The ECC8 interrupts the CPU after completing the entire transaction. This could be a single 512-byte block if desired or the entire 4K page of payload data and the 65 bytes of metadata. It also could be just the metadata block. Note that the metadata is protected by its own 9-byte parity so that reading metadata is very efficient.

The ECC8 status registers indicate the quality of the data read into each of the nine blocks with a four-bit code.

- A value of 0 means no errors occurred on the block.
- A value of 1–8 means that correctable errors occurred but the data was repaired by the bus master.
- A value of 0xC means that this block was not specified on the read transaction.
- A value of 0xE means that an uncorrectable error occurred on that block.
- A value of 0xF means that this block contains all ones and is therefore considered to be an ERASED block.
- A summary status quickly tells if **any** block in the page had an uncorrectable error.

14.2.2 Reed-Solomon ECC Encoding for NAND Writes

The RS encoder flowchart in Figure 14-6 shows the detailed steps involved in programming and using the ECC8 encoder. This flowchart shows how to use the ECC8 block with the GPMI.

To use the ECC8 encoder with the GPMI's DMA, create a DMA command chain containing ten descriptor structures, as shown in Figure 14-8 and detailed in the DMA structure code example that follows it in Section 14.2.2.1, "DMA Structure Code Example." The ten descriptors perform the following tasks:

1. Disable the ECC8 block (in case it was enabled) and issue NAND write setup command byte (under "CLE") and address bytes (under "ALE").
2. Configure and enable the ECC8 block and write the data payload.
3. Write the auxiliary payload.
4. Disable the ECC8 block and issue NAND write execute command byte (under "CLE").
5. Wait for the NAND device to finish writing the data by watching the ready signal.
6. Check for NAND timeout via "DMA_SENSE". Refer to Section 10.2 for a description of DMA SENSE.

7. Issue NAND status command byte (under “CLE”).
8. Read the status and compare against expected.
9. If status is incorrect/incomplete, branch to error handling descriptor chain.
10. Otherwise, write is complete and emit GPMI interrupt.

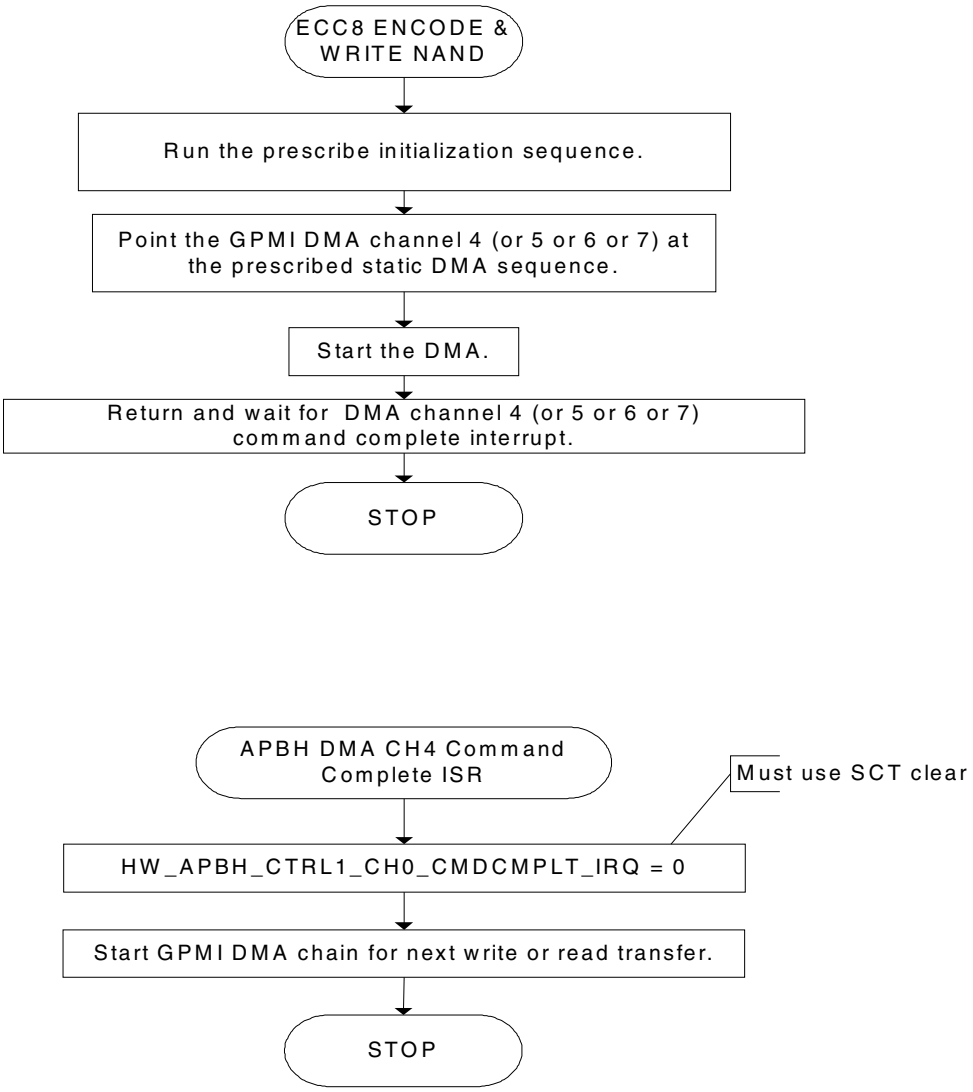


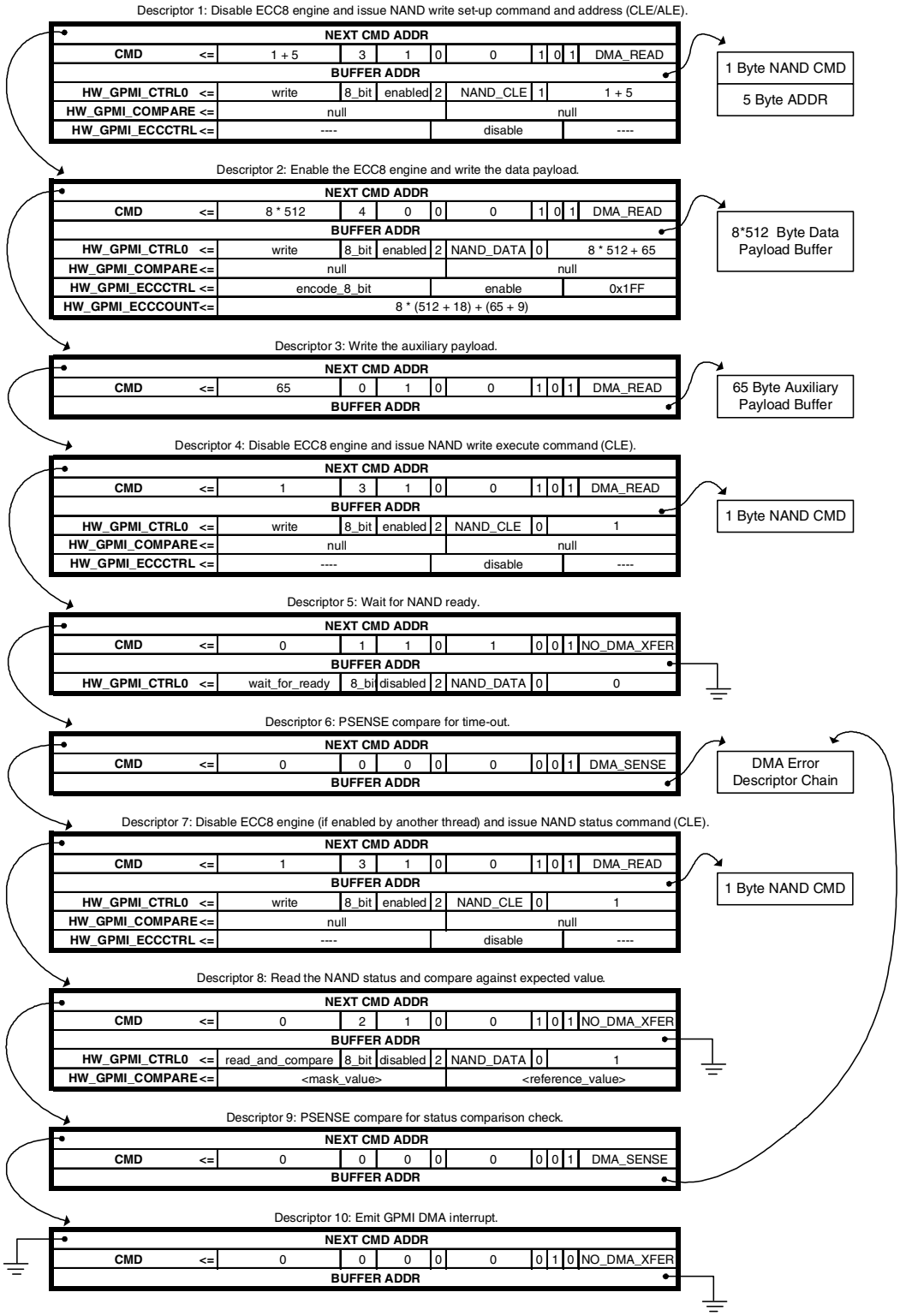
Figure 14-6. ECC8 Reed-Solomon Encode Flowchart

Descriptor Legend

NEXT CMD ADDR										
CMD	<=	xfer_count	cmdwords	wait4endcmd	semaphore	nandwait4ready	nandlock	irqoncmplt	chain	command
BUFFER ADDR										
HW_GPMI_CTRL0	<=	command_mode	word_length	lock_cs	CS	address	address_increment	xfer_count		
HW_GPMI_COMPARE	<=	mask				reference				
HW_GPMI_ECCCTRL	<=	ecc_cmd			enable_ecc			buffer_mask		
HW_GPMI_ECCCOUNT										
HW_GPMI_PAYLOAD										
HW_GPMI_AUXILIARY										

Note: Refer to this legend when examining Figure 14-8 and Figure 14-11.

Figure 14-7. ECC8 DMA Descriptor Legend



Note: To interpret the fields in this diagram, see Figure 14-7 for the descriptor legend.

Figure 14-8. ECC8 Reed-Solomon Encode DMA Descriptor Chain

14.2.2.1 DMA Structure Code Example

The following code sample illustrates the coding for one write transaction involving 4096 bytes of data payload (eight 512-byte blocks) and 65 bytes of auxiliary payload (also referred to as *metadata*) to a 4K NAND page sitting on GPMI CS2.

```
//------------------------------------------------------------------------------
// generic DMA/GPMI/ECC descriptor struct, order sensitive!
//------------------------------------------------------------------------------
typedef struct {
  // DMA related fields
  unsigned int dma_nxtcmdar;
  unsigned int dma_cmd;
  unsigned int dma_bar;

  // GPMI related fields
  unsigned int gpmi_ctrl0;
  unsigned int gpmi_compare;
  unsigned int gpmi_eccctrl;
  unsigned int gpmi_ecccount;
  unsigned int gpmi_data_ptr;
  unsigned int gpmi_aux_ptr;
} GENERIC_DESCRIPTOR;

//------------------------------------------------------------------------------
// allocate 10 descriptors for doing a NAND ECC Write
//------------------------------------------------------------------------------
GENERIC_DESCRIPTOR write[10];

//------------------------------------------------------------------------------
// DMA descriptor pointer to handle error conditions from psense checks
//------------------------------------------------------------------------------
unsigned int * dma_error_handler;

//------------------------------------------------------------------------------
// 8 byte NAND command and address buffer
// any alignment is ok, it is read by the GPMI DMA
//   byte 0 is write setup command
//   bytes 1-5 is the NAND address
//   byte 6 is write execute command
//   byte 7 is status command
//------------------------------------------------------------------------------
unsigned char nand_cmd_addr_buffer[8];

//------------------------------------------------------------------------------
// 4096 byte payload buffer used for reads or writes
// needs to be word aligned
//------------------------------------------------------------------------------
unsigned int write_payload_buffer[(4096/4)];

//------------------------------------------------------------------------------
// 65 byte meta-data to be written to NAND
// needs to be word aligned
//------------------------------------------------------------------------------
unsigned int write_aux_buffer[65];

//------------------------------------------------------------------------------
// Descriptor 1: issue NAND write setup command (CLE/ALE)
//------------------------------------------------------------------------------
write[0].dma_nxtcmdar = &write[1];                              // point to the next descriptor

write[0].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (1 + 5)    | // 1 byte command, 5 byte address
                   BF_APBH_CHn_CMD_CMDWORDS      (3)        | // send 3 words to the GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)        | // wait for command to finish before continuing
                   BF_APBH_CHn_CMD_SEMAPHORE     (0)        |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(0)        |
                   BF_APBH_CHn_CMD_NANDLOCK      (1)        | // prevent other DMA channels from taking over
                   BF_APBH_CHn_CMD_IRQONCMPLT    (0)        |
                   BF_APBH_CHn_CMD_CHAIN         (1)        | // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, DMA_READ);  // read data from DMA, write to NAND

write[0].dma_bar = &nand_cmd_addr_buffer;                       // byte 0 write setup, bytes 1 - 5 NAND address

// 3 words sent to the GPMI
write[0].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, WRITE)     | // write to the NAND
                      BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)     |
                      BV_FLD(GPMI_CTRL0, LOCK_CS,      ENABLED)   |
                      BF_GPMI_CTRL0_CS                 (2)        | // must correspond to NAND CS used
                      BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_CLE)  |
                      BF_GPMI_CTRL0_ADDRESS_INCREMENT  (1)        | // send command and address
                      BF_GPMI_CTRL0_XFER_COUNT         (1 + 5);   // 1 byte command, 5 byte address

write[0].gpmi_compare = NULL;                                    // field not used but necessary to set eccctrl

write[0].gpmi_eccctrl = BV_FLD(GPMI_ECCCTRL, ENABLE_ECC, DISABLE); // disable the ECC block
```

```
//------------------------------------------------------------------------
// Descriptor 2: write the data payload (DATA)
//------------------------------------------------------------------------
write[1].dma_nxtcmdar = &write[2];                          // point to the next descriptor

write[1].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (8*512) | // NOTE: DMA transfer only the data payload
                   BF_APBH_CHn_CMD_CMDWORDS      (4)     | // send 4 words to the GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (0)     | // DON'T wait to end, wait in the next descriptor
                   BF_APBH_CHn_CMD_SEMAPHORE     (0)     |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(0)     |
                   BF_APBH_CHn_CMD_NANDLOCK      (1)     | // maintain resource lock
                   BF_APBH_CHn_CMD_IRQONCMPLT    (0)     |
                   BF_APBH_CHn_CMD_CHAIN         (1)     | // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, DMA_READ);  // read data from DMA, write to NAND

write[1].dma_bar = &write_payload_buffer;                   // pointer for the 4K byte data area

// 4 words sent to the GPMI
write[1].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, WRITE)     |     // write to the NAND
                      BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)     |
                      BV_FLD(GPMI_CTRL0, LOCK_CS,      ENABLED)   |
                      BF_GPMI_CTRL0_CS                 (2)        |     // must correspond to NAND CS used
                      BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_DATA) |
                      BF_GPMI_CTRL0_ADDRESS_INCREMENT  (0)        |
                      BF_GPMI_CTRL0_XFER_COUNT     (8*512+65);  // NOTE: this field contains the total amount
                                                                  // DMA transferred (8 data and 1 aux blocks)
                                                                  // to GPMI!

write[1].gpmi_compare = NULL;                               // field not used but necessary to set eccctrl

write[1].gpmi_eccctrl = BV_FLD(GPMI_ECCCTRL, ECC_CMD,    ENCODE_8_BIT) | // specify t = 8 mode
                        BV_FLD(GPMI_ECCCTRL, ENABLE_ECC, ENABLE)       | // enable ECC module
                        BF_GPMI_ECCCTRL_BUFFER_MASK   (0x1FF);    // write all 8 data blocks and 1 aux block

write[1].gpmi_ecccount = BF_GPMI_ECCCOUNT_COUNT(8*(512+18)+(65+9)); // specify number of bytes written to NAND
                                                                    // NOTE: the extra 8*(18)+9 bytes are
                                                                   // parity bytes generated by the ECC block.

//------------------------------------------------------------------------
// Descriptor 3: write the aux payload (DATA)
//------------------------------------------------------------------------
write[2].dma_nxtcmdar = &write[3];                          // point to the next descriptor

write[2].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (65)     | // NOTE: DMA transfer only the aux block
                   BF_APBH_CHn_CMD_CMDWORDS      (0)      | // no words sent to GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)      | // wait for command to finish before continuing
                   BF_APBH_CHn_CMD_SEMAPHORE     (0)      |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(0)      |
                   BF_APBH_CHn_CMD_NANDLOCK      (1)      | // maintain resource lock
                   BF_APBH_CHn_CMD_IRQONCMPLT    (0)      |
                   BF_APBH_CHn_CMD_CHAIN         (1)      | // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, DMA_READ);  // read data from DMA, write to NAND

write[2].dma_bar = &write_aux_buffer;                       // pointer for the 65 byte meta data area

//------------------------------------------------------------------------
// Descriptor 4: issue NAND write execute command (CLE)
//------------------------------------------------------------------------
write[3].dma_nxtcmdar = &write[4];                          // point to the next descriptor

write[3].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (1)      | // 1 byte command
                   BF_APBH_CHn_CMD_CMDWORDS      (3)      | // send 3 words to the GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)      | // wait for command to finish before continuing
                   BF_APBH_CHn_CMD_SEMAPHORE     (0)      |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(0)      |
                   BF_APBH_CHn_CMD_NANDLOCK      (1)      | // maintain resource lock
                   BF_APBH_CHn_CMD_IRQONCMPLT    (0)      |
                   BF_APBH_CHn_CMD_CHAIN         (1)      | // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, DMA_READ);  // read data from DMA, write to NAND

write[3].dma_bar = &nand_cmd_addr_buffer[6];                // point to byte 6, write execute command

// 3 words sent to the GPMI
write[3].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, WRITE)     | // write to the NAND
                      BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)     |
                      BV_FLD(GPMI_CTRL0, LOCK_CS,      ENABLED)   |
                      BF_GPMI_CTRL0_CS                 (2)        | // must correspond to NAND CS used
                      BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_CLE)  |
                      BF_GPMI_CTRL0_ADDRESS_INCREMENT  (0)        |
                      BF_GPMI_CTRL0_XFER_COUNT         (1);       // 1 byte command

write[3].gpmi_compare = NULL;                               // field not used but necessary to set eccctrl

write[3].gpmi_eccctrl = BV_FLD(GPMI_ECCCTRL, ENABLE_ECC, DISABLE); // disable the ECC block

//------------------------------------------------------------------------
// Descriptor 5: wait for ready (CLE)
//------------------------------------------------------------------------
write[4].dma_nxtcmdar = &write[5];                                  // point to the next descriptor
```

```
write[4].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0)          |    // no dma transfer
                   BF_APBH_CHn_CMD_CMDWORDS      (1)          |    // send 1 word to the GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)          | // wait for command to finish before continuing
                   BF_APBH_CHn_CMD_SEMAPHORE     (0)          |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(1)          |    // wait for nand to be ready
                   BF_APBH_CHn_CMD_NANDLOCK      (0)          |    // relinquish nand lock
                   BF_APBH_CHn_CMD_IRQONCMPLT    (0)          |
                   BF_APBH_CHn_CMD_CHAIN         (1)          |    // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, NO_DMA_XFER);   // no dma transfer

write[4].dma_bar = NULL;                                          // field not used

// 1 word sent to the GPMI
write[4].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, WAIT_FOR_READY) | // wait for NAND ready
                      BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)          |
                      BV_FLD(GPMI_CTRL0, LOCK_CS,      DISABLED)       |
                      BF_GPMI_CTRL0_CS                 (2)             | // must correspond to NAND CS used
                      BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_DATA)      |
                      BF_GPMI_CTRL0_ADDRESS_INCREMENT  (0)             |
                      BF_GPMI_CTRL0_XFER_COUNT         (0);

//--------------------------------------------------------------------------
// Descriptor 6: psense compare (time out check)
//--------------------------------------------------------------------------
write[5].dma_nxtcmdar = &write[6];                              // point to the next descriptor

write[5].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0)          | // no dma transfer
                   BF_APBH_CHn_CMD_CMDWORDS      (0)          | // no words sent to GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (0)          | // do not wait to continue
                   BF_APBH_CHn_CMD_SEMAPHORE     (0)          |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(0)          |
                   BF_APBH_CHn_CMD_NANDLOCK      (0)          |
                   BF_APBH_CHn_CMD_IRQONCMPLT    (0)          |
                   BF_APBH_CHn_CMD_CHAIN         (1)          | // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, DMA_SENSE);  // perform a sense check

write[5].dma_bar = dma_error_handler;                          // if sense check fails, branch to error handler

//--------------------------------------------------------------------------
// Descriptor 7: issue NAND status command (CLE)
//--------------------------------------------------------------------------
write[6].dma_nxtcmdar = &write[7];                                 // point to the next descriptor

write[6].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (1)   |      // 1 byte command
                   BF_APBH_CHn_CMD_CMDWORDS      (3)   |      // send 3 words to the GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)   |      // wait for command to finish before continuing
                   BF_APBH_CHn_CMD_SEMAPHORE     (0)   |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(0)   |
                   BF_APBH_CHn_CMD_NANDLOCK      (1)   |      // prevent other DMA channels from taking over
                   BF_APBH_CHn_CMD_IRQONCMPLT    (0)   |
                   BF_APBH_CHn_CMD_CHAIN         (1)   |      // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, DMA_READ); // read data from DMA, write to NAND

write[6].dma_bar = &nand_cmd_addr_buffer[7];                   // point to byte 7, status command

write[6].gpmi_compare = NULL;                                  // field not used but necessary to set eccctrl

write[6].gpmi_eccctrl = BV_FLD(GPMI_ECCCTRL, ENABLE_ECC, DISABLE);     // disable the ECC block

// 3 words sent to the GPMI
write[6].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, WRITE)     | // write to the NAND
                      BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)     |
                      BV_FLD(GPMI_CTRL0, LOCK_CS,      ENABLED)   |
                      BF_GPMI_CTRL0_CS                 (2)        | // must correspond to NAND CS used
                      BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_CLE)  |
                      BF_GPMI_CTRL0_ADDRESS_INCREMENT  (0)        |
                      BF_GPMI_CTRL0_XFER_COUNT         (1);       // 1 byte command

//--------------------------------------------------------------------------
// Descriptor 8: read status and compare (DATA)
//--------------------------------------------------------------------------
write[7].dma_nxtcmdar = &write[8];                                    // point to the next descriptor

write[7].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0) |        // no dma transfer
                   BF_APBH_CHn_CMD_CMDWORDS      (2) |        // send 2 words to the GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (1) |        // wait for command to finish before continuing
                   BF_APBH_CHn_CMD_SEMAPHORE     (0) |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(0) |
                   BF_APBH_CHn_CMD_NANDLOCK      (1) |        // maintain resource lock
                   BF_APBH_CHn_CMD_IRQONCMPLT    (0) |
                   BF_APBH_CHn_CMD_CHAIN         (1) |        // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, NO_DMA_XFER);            // no dma transfer

write[7].dma_bar = NULL;                                              // field not used

// 2 word sent to the GPMI
write[7].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, READ_AND_COMPARE) |   // read from the NAND and
                                                                             // compare to expect
                      BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)            |
                      BV_FLD(GPMI_CTRL0, LOCK_CS,      DISABLED)         |
```

```
                    BF_GPMI_CTRL0_CS                (2)              | // must correspond to NAND CS used
                    BV_FLD(GPMI_CTRL0, ADDRESS,     NAND_DATA)       |
                    BF_GPMI_CTRL0_ADDRESS_INCREMENT (0)              |
                    BF_GPMI_CTRL0_XFER_COUNT        (1);

write[7].gpmi_compare = <MASK_AND_REFERENCE_VALUE>;             // NOTE: mask and reference values are NAND
                                                                // SPECIFIC to evaluate the NAND status

//-----------------------------------------------------------------------------
// Descriptor 9: psense compare (time out check)
//-----------------------------------------------------------------------------
write[8].dma_nxtcmdar = &write[9];                              // point to the next descriptor

write[8].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0)         | // no dma transfer
                   BF_APBH_CHn_CMD_CMDWORDS      (0)         | // no words sent to GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (0)         | // do not wait to continue
                   BF_APBH_CHn_CMD_SEMAPHORE     (0)         |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(0)         |
                   BF_APBH_CHn_CMD_NANDLOCK      (0)         | // relinquish nand lock
                   BF_APBH_CHn_CMD_IRQONCMPLT    (0)         |
                   BF_APBH_CHn_CMD_CHAIN         (1)         | // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, DMA_SENSE);   // perform a sense check

write[8].dma_bar = dma_error_handler;                          // if sense check fails, branch to error handler

//-----------------------------------------------------------------------------
// Descriptor 10: emit GPMI interrupt
//-----------------------------------------------------------------------------
write[9].dma_nxtcmdar = NULL;                                   // not used since this is last descriptor

write[9].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0)          | // no dma transfer
                   BF_APBH_CHn_CMD_CMDWORDS      (0)          | // no words sent to GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (0)          | // do not wait to continue
                   BF_APBH_CHn_CMD_SEMAPHORE     (0)          |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(0)          |
                   BF_APBH_CHn_CMD_NANDLOCK      (0)          |
                   BF_APBH_CHn_CMD_IRQONCMPLT    (1)          | // emit GPMI interrupt
                   BF_APBH_CHn_CMD_CHAIN         (0)          | // terminate DMA chain processing
                   BV_FLD(APBH_CHn_CMD, COMMAND, NO_DMA_XFER); // no dma transfer
```

14.2.2.2 Using the ECC8 Encoder

To use the ECC8 encoder, first turn off the module-wide soft reset bit in both the GPMI and ECC8 blocks before starting any DMA activity. Note that turning off the soft reset must take place by itself, prior to programming the rest of the control registers. Turn off the ECC8 bus master soft reset bit (bit 29). Turn off the clock gate bits.

Program the remainder of the GPMI, ECC8 and APBH DMA as follows:

```
// bring APBH out of reset
HW_APBH_CTRL0_CLR(BM_APBH_CTRL0_SFRST);
HW_APBH_CTRL0_CLR(BM_APBH_CTRL0_CLKGATE);

// bring ecc8 out of reset
HW_ECC8_CTRL_CLR(BM_ECC8_CTRL_SFTRST);
HW_ECC8_CTRL_CLR(BM_ECC8_CTRL_CLKGATE);
HW_ECC8_CTRL_CLR(BM_ECC8_CTRL_AHBM_SFTRST);

// bring gpmi out of reset
HW_GPMI_CTRL0_CLR(BM_GPMI_CTRL0_SFTRST);
HW_GPMI_CTRL0_CLR(BM_GPMI_CTRL0_CLKGATE);
HW_GPMI_CTRL1_SET(BM_GPMI_CTRL1_DEV_RESET);// deassert in case
                                          //anyone's hooked up to the reset pin

// enable pinctrl
HW_PINCTRL_CTRL_WR(0x00000000);

// enable GPMI through alt pin wiring
HW_PINCTRL_MUXSEL0_CLR(0xff000000);
HW_PINCTRL_MUXSEL0_SET(0xaa000000);

// to use the primary pins do the following
//    HW_PINCTRL_MUXSEL4_CLR(0xff000000);
//    HW_PINCTRL_MUXSEL4_SET(0x55000000);

// enable gpmi pins
```

```
HW_PINCTRL_MUXSEL0_CLR(0x0000ffff);// data bits
HW_PINCTRL_MUXSEL1_CLR(0x000fffff);// control bits
```

Note that for writing NANDs (ECC encoding), only GPMI DMA command complete interrupts are used. The ECC8 engine is used for writing to the NAND but never produces an interrupt. From the sample code in Section 14.2.2.1, “DMA Structure Code Example:”

- DMA descriptor 1 prepares the NAND for data write by using the GPMI to issue a write setup command byte under “CLE”, then sends a 5-byte address under “ALE”. The ECC8 engine is disabled and not used for these commands.
- DMA descriptor 2 enables the ECC8 engine for t=8 encoding to begin the initial writing of the NAND data by specifying where the data payload is coming from in system memory.
- DMA descriptor 3 continues the writing of NAND data by specifying where the auxiliary data is coming from in system memory.
- DMA descriptor 4 issues the write commit command byte under “CLE” to the NAND.
- DMA descriptor 5 waits for the NAND to complete the write commit/transfer by watching the NAND’s ready line status. This descriptor relinquishes the NANDLOCK on the GPMI to enable the other DMA channels to initiate NAND transactions on different NAND CS lines.
- DMA descriptor 7 issues a NAND status command byte under “CLE” to check the status of the NAND device following the page write.
- DMA descriptor 8 reads back the NAND status and compares the status with an expected value. If there are differences, then the DMA processing engine follows an error-handling DMA descriptor path.
- DMA descriptor 9 disables the ECC8 engine and emits a GPMI interrupt to indicate that the NAND write has been completed.

14.2.3 Reed-Solomon ECC Decoding for NAND Reads

When a page is read from NAND flash, RS syndromes will be computed and, if correctable errors are found, they will be corrected on a per block basis within the NAND page. This decoding process is fully overlapped with other NAND data reads and with CPU execution. The RS decoder flowchart in Figure 14-9 shows the steps involved in programming the ECC8 Reed-Solomon decoder. The hardware flow of reading and decoding a 512-byte page encoded for t=8 error correction (18 parity bytes) is shown in Figure 14-10.

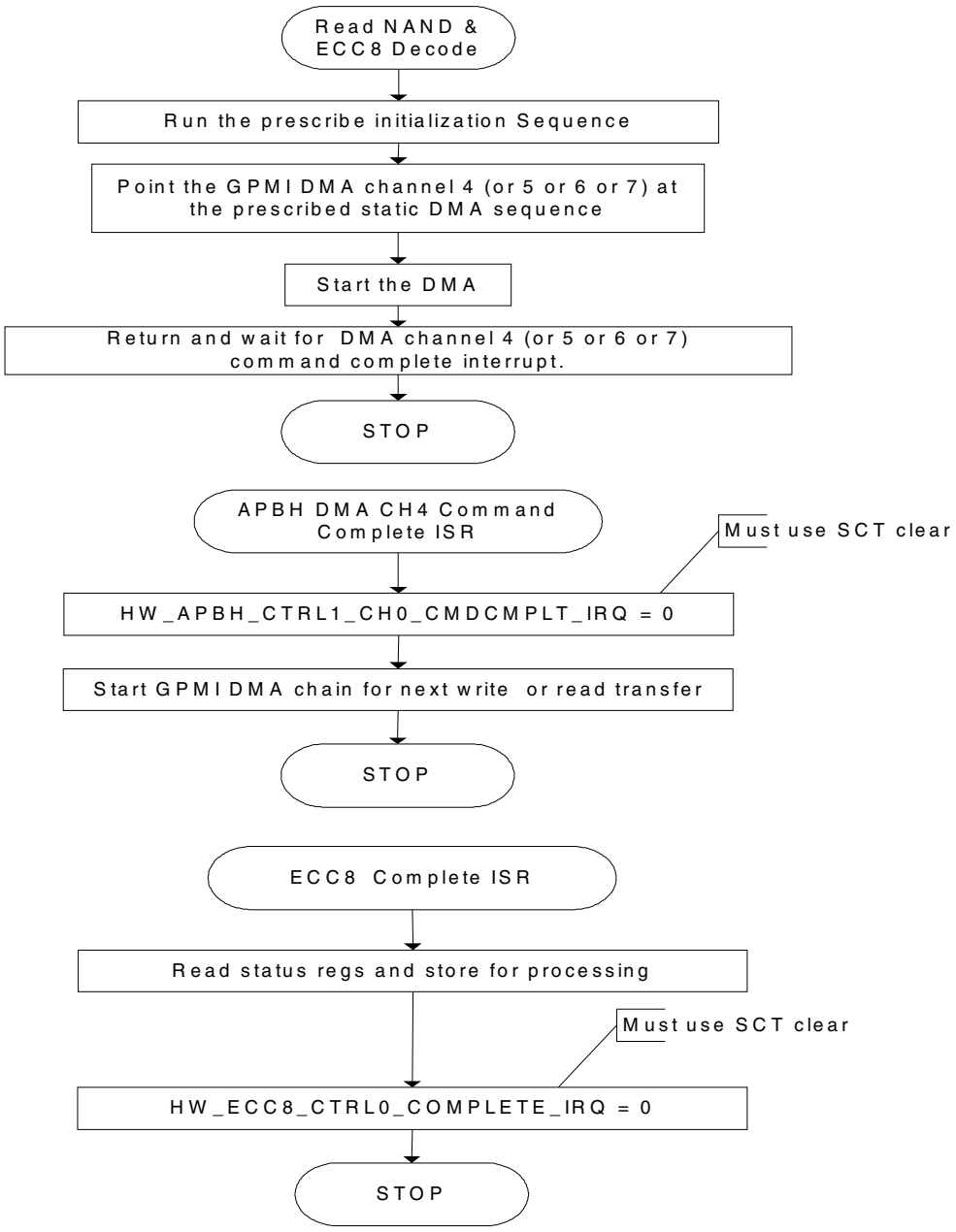
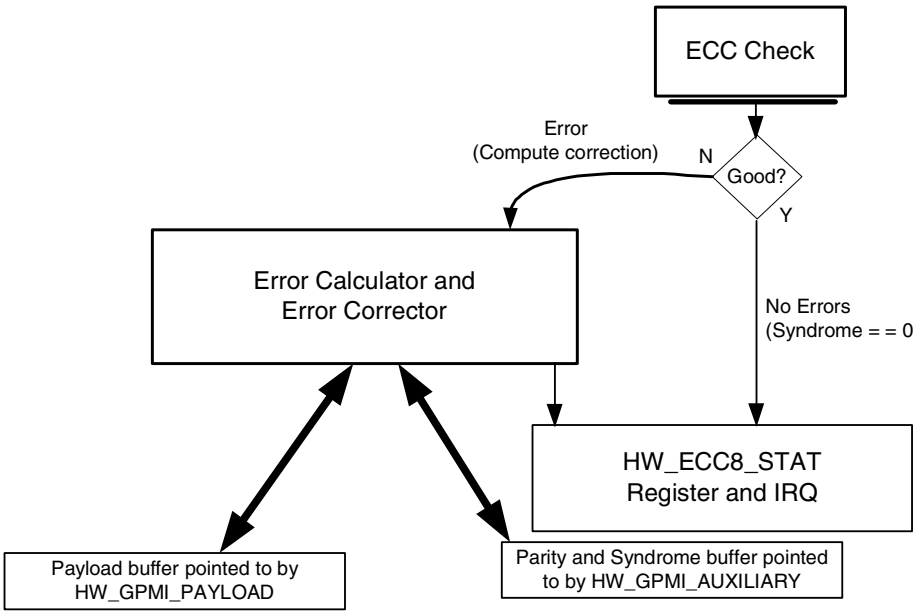


Figure 14-9. ECC8 Reed-Solomon Decode Flowchart



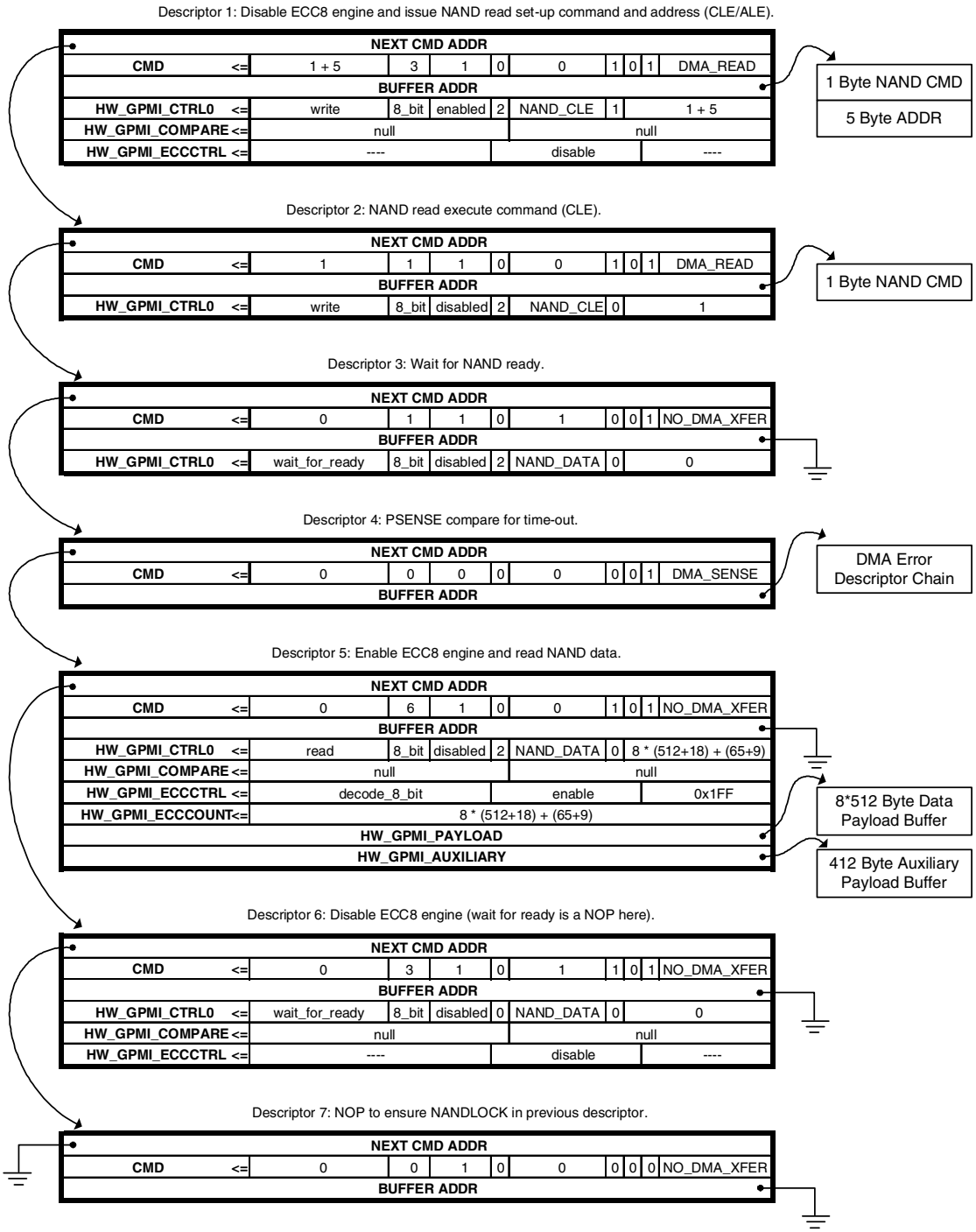
NOTES:
This diagram describes a decoder for reading and correcting a 512-byte data block encoded with t=8 error correction. ECC8 bus master writes the bytes from the NAND to system memory. The error corrector performs read-modify-writes on these system memory buffers as necessary.

Figure 14-10.

Conceptually, an APHB DMA Channel 4, 5, 6, or 7 command chain with seven command structures linked together is used to perform the RS decode operation (as shown in Figure 14-11). NOTE: The GPMI's DMA command structures controls the ECC8 decode operation.

To use the ECC8 decoder with the GPMI's DMA, create a DMA command chain containing seven descriptor structures, as shown in Figure 14-11 and detailed in the DMA structure code example that follows it in Section 14.2.3.1, "DMA Structure Code Example." The seven DMA descriptors perform the following tasks:

1. Issue NAND read setup command byte (under "CLE") and address bytes (under "ALE").
2. Issue NAND read execute command byte (under "CLE").
3. Wait for the NAND device to complete accessing the block data by watching the ready signal.
4. Check for NAND timeout via "DMA_SENSE". Refer to Section 10.2 for a description of DMA SENSE.
5. Configure and enable the ECC8 block and read the NAND block data.
6. Disable the ECC8 block.
7. Descriptor NOP to allow NANDLOCK in the previous descriptor to the thread-safe.



Note: To interpret the fields in this diagram, see Figure 14-7 for the descriptor legend.

Figure 14-11. ECC8 Reed-Solomon Decode DMA Descriptor Chain

14.2.3.1 DMA Structure Code Example

The following sample code illustrates the coding for one read transaction, consisting of a seven DMA command structure chain for reading all 4096 bytes of payload data (eight 512-byte blocks) and 65 bytes of metadata with the associative parity bytes
(8 * (18) + 9) from a 4K NAND page sitting on GPMI CS2.

```
//-----------------------------------------------------------------------------
// generic DMA/GPMI/ECC descriptor struct, order sensitive!
//-----------------------------------------------------------------------------
typedef struct {
  // DMA related fields
  unsigned int dma_nxtcmdar;
  unsigned int dma_cmd;
  unsigned int dma_bar;

  // GPMI related fields
  unsigned int gpmi_ctrl0;
  unsigned int gpmi_compare;
  unsigned int gpmi_eccctrl;
  unsigned int gpmi_ecccount;
  unsigned int gpmi_data_ptr;
  unsigned int gpmi_aux_ptr;
} GENERIC_DESCRIPTOR;

//-----------------------------------------------------------------------------
// allocate 7 descriptors for doing a NAND ECC Read
//-----------------------------------------------------------------------------
GENERIC_DESCRIPTOR read[7];

//-----------------------------------------------------------------------------
// DMA descriptor pointer to handle error conditions from psense checks
//-----------------------------------------------------------------------------
unsigned int * dma_error_handler;

//-----------------------------------------------------------------------------
// 7 byte NAND command and address buffer
// any alignment is ok, it is read by the GPMI DMA
//   byte 0 is read setup command
//   bytes 1-5 is the NAND address
//   byte 6 is read execute command
//-----------------------------------------------------------------------------
unsigned char nand_cmd_addr_buffer[7];

//-----------------------------------------------------------------------------
// 4096 byte payload buffer used for reads or writes
// needs to be word aligned
//-----------------------------------------------------------------------------
unsigned int read_payload_buffer[(4096/4)];

//-----------------------------------------------------------------------------
// 412 byte auxiliary buffer used for reads
// needs to be word aligned
//-----------------------------------------------------------------------------
unsigned int read_aux_buffer[(412/4)];

//-----------------------------------------------------------------------------
// Descriptor 1: issue NAND read setup command (CLE/ALE)
//-----------------------------------------------------------------------------
read[0].dma_nxtcmdar = &read[1];                                  // point to the next descriptor

read[0].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (1 + 5)    |  // 1 byte command, 5 byte address
                  BF_APBH_CHn_CMD_CMDWORDS      (3)        |  // send 3 words to the GPMI
                  BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)        |  // wait for command to finish before continuing
                  BF_APBH_CHn_CMD_SEMAPHORE     (0)        |
                  BF_APBH_CHn_CMD_NANDWAIT4READY(0)        |
                  BF_APBH_CHn_CMD_NANDLOCK      (1)        |  // prevent other DMA channels from taking over
                  BF_APBH_CHn_CMD_IRQONCMPLT    (0)        |
                  BF_APBH_CHn_CMD_CHAIN         (1)        |  // follow chain to next command
                  BV_FLD(APBH_CHn_CMD, COMMAND, DMA_READ);   // read data from DMA, write to NAND

read[0].dma_bar = &nand_cmd_addr_buffer;                          // byte 0 read setup, bytes 1 - 5 NAND address

// 3 words sent to the GPMI
read[0].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, WRITE)     | // write to the NAND
                     BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)     |
                     BV_FLD(GPMI_CTRL0, LOCK_CS,      ENABLED)   |
                     BF_GPMI_CTRL0_CS                 (2)        | // must correspond to NAND CS used
                     BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_CLE)  |
                     BF_GPMI_CTRL0_ADDRESS_INCREMENT  (1)        | // send command and address
                     BF_GPMI_CTRL0_XFER_COUNT         (1 + 5);   // 1 byte command, 5 byte address

read[0].gpmi_compare = NULL;                                       // field not used but necessary to set eccctrl
```

```
read[0].gpmi_eccctrl = BV_FLD(GPMI_ECCCTRL, ENABLE_ECC, DISABLE); // disable the ECC block

//------------------------------------------------------------------------
// Descriptor 2: issue NAND read execute command (CLE)
//------------------------------------------------------------------------
read[1].dma_nxtcmdar = &read[2];                                // point to the next descriptor

read[1].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (1)         |  // 1 byte read command
                  BF_APBH_CHn_CMD_CMDWORDS      (1)         |  // send 1 word to GPMI
                  BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)         |  // wait for command to finish before continuing
                  BF_APBH_CHn_CMD_SEMAPHORE     (0)         |
                  BF_APBH_CHn_CMD_NANDWAIT4READY(0)         |
                  BF_APBH_CHn_CMD_NANDLOCK      (1)         |  // prevent other DMA channels from taking over
                  BF_APBH_CHn_CMD_IRQONCMPLT    (0)         |
                  BF_APBH_CHn_CMD_CHAIN         (1)         |  // follow chain to next command
                  BV_FLD(APBH_CHn_CMD, COMMAND, DMA_READ);    // read data from DMA, write to NAND

read[1].dma_bar = &nand_cmd_addr_buffer[6];                     // point to byte 6, read execute command

// 1 word sent to the GPMI
read[1].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, WRITE)     | // write to the NAND
                     BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)     |
                     BV_FLD(GPMI_CTRL0, LOCK_CS,      DISABLED)  |
                     BF_GPMI_CTRL0_CS                 (2)        | // must correspond to NAND CS used
                     BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_CLE)  |
                     BF_GPMI_CTRL0_ADDRESS_INCREMENT  (0)        |
                     BF_GPMI_CTRL0_XFER_COUNT         (1);       // 1 byte command

//------------------------------------------------------------------------
// Descriptor 3: wait for ready (DATA)
//------------------------------------------------------------------------
read[2].dma_nxtcmdar = &read[3];                                 // point to the next descriptor

read[2].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0)         |    // no dma transfer
                  BF_APBH_CHn_CMD_CMDWORDS      (1)         |    // send 1 word to GPMI
                  BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)         |    // wait for command to finish before continuing
                  BF_APBH_CHn_CMD_SEMAPHORE     (0)         |
                  BF_APBH_CHn_CMD_NANDWAIT4READY(1)         |    // wait for nand to be ready
                  BF_APBH_CHn_CMD_NANDLOCK      (0)         |    // relinquish nand lock
                  BF_APBH_CHn_CMD_IRQONCMPLT    (0)         |
                  BF_APBH_CHn_CMD_CHAIN         (1)         |    // follow chain to next command
                  BV_FLD(APBH_CHn_CMD, COMMAND, NO_DMA_XFER);           // no dma transfer

read[2].dma_bar = NULL;                                          // field not used

// 1 word sent to the GPMI
read[2].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, WAIT_FOR_READY) | // wait for NAND ready
                     BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)          |
                     BV_FLD(GPMI_CTRL0, LOCK_CS,      DISABLED)       |
                     BF_GPMI_CTRL0_CS                 (2)             | // must correspond to NAND CS used
                     BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_DATA)      |
                     BF_GPMI_CTRL0_ADDRESS_INCREMENT  (0)             |
                     BF_GPMI_CTRL0_XFER_COUNT         (0);

//------------------------------------------------------------------------
// Descriptor 4: psense compare (time out check)
//------------------------------------------------------------------------
read[3].dma_nxtcmdar = &read[4];                                 // point to the next descriptor

read[3].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0)         |    // no dma transfer
                  BF_APBH_CHn_CMD_CMDWORDS      (0)         |    // no words sent to GPMI
                  BF_APBH_CHn_CMD_WAIT4ENDCMD   (0)         |    // do not wait to continue
                  BF_APBH_CHn_CMD_SEMAPHORE     (0)         |
                  BF_APBH_CHn_CMD_NANDWAIT4READY(0)         |
                  BF_APBH_CHn_CMD_NANDLOCK      (0)         |
                  BF_APBH_CHn_CMD_IRQONCMPLT    (0)         |
                  BF_APBH_CHn_CMD_CHAIN         (1)         |    // follow chain to next command
                  BV_FLD(APBH_CHn_CMD, COMMAND, DMA_SENSE);       // perform a sense check

read[3].dma_bar = dma_error_handler;                            // if sense check fails, branch to error handler

//------------------------------------------------------------------------
// Descriptor 5: read 4K page plus 65 byte meta-data Nand data
//               and send it to ECC block (DATA)
//------------------------------------------------------------------------
read[4].dma_nxtcmdar = &read[5];                                // point to the next descriptor

read[4].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0)     |    // no dma transfer
                  BF_APBH_CHn_CMD_CMDWORDS      (6)     |    // send 6 words to GPMI
                  BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)     |    // wait for command to finish before continuing
                  BF_APBH_CHn_CMD_SEMAPHORE     (0)     |
                  BF_APBH_CHn_CMD_NANDWAIT4READY(0)     |
                  BF_APBH_CHn_CMD_NANDLOCK      (1)     |    // prevent other DMA channels from taking over
                  BF_APBH_CHn_CMD_IRQONCMPLT    (0)     |    // ECC block generates ecc8 interrupt on completion
                  BF_APBH_CHn_CMD_CHAIN         (1)     |    // follow chain to next command
                  BV_FLD(APBH_CHn_CMD, COMMAND, NO_DMA_XFER); // no DMA transfer, ECC block handles transfer

read[4].dma_bar = NULL;                                          // field not used
```

```
// 6 words sent to the GPMI
read[4].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, READ)             | // read from the NAND
                     BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)            |
                     BV_FLD(GPMI_CTRL0, LOCK_CS,      DISABLED)         |
                     BF_GPMI_CTRL0_CS                 (2)               | // must correspond to NAND CS used
                     BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_DATA)        |
                     BF_GPMI_CTRL0_ADDRESS_INCREMENT  (0)               |
            BF_GPMI_CTRL0_XFER_COUNT (8*(512+18)+(65+9)); // eight 512 byte data blocks (plus parity, t = 8)
                                                          // and one 65 byte aux block (plus parity, t = 4)

read[4].gpmi_compare = NULL;                              // field not used but necessary to set eccctrl

// GPMI ECCCTRL PIO This launches the 4K byte transfer through ECC8's
// bus master. Setting the ECC_ENABLE bit redirects the data flow
// within the GPMI so that read data flows to the ECC8 engine instead
// of flowing to the GPMI's DMA channel.
read[4].gpmi_eccctrl = BV_FLD(GPMI_ECCCTRL, ECC_CMD,    DECODE_8_BIT) |    // specify t = 8 mode
                       BV_FLD(GPMI_ECCCTRL, ENABLE_ECC, ENABLE)       |    // enable ECC module
                       BF_GPMI_ECCCTRL_BUFFER_MASK      (0X1FF);   // read all 8 data blocks and 1 aux block

read[4].gpmi_ecccount = BF_GPMI_ECCCOUNT_COUNT(8*(512+18)+(65+9)); // specify number of bytes read from NAND

read[4].gpmi_data_ptr = &read_payload_buffer;                      // pointer for the 4K byte data area

read[4].gpmi_aux_ptr = &read_aux_buffer;                           // pointer for the 65 byte aux area +
                                                                   // parity and syndrome bytes for both
                                                                   // data and aux blocks.

//--------------------------------------------------------------------------
// Descriptor 6: disable ECC block
//--------------------------------------------------------------------------
read[5].dma_nxtcmdar = &read[6];                       // point to the next descriptor

read[5].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0) | // no dma transfer
                  BF_APBH_CHn_CMD_CMDWORDS      (3) | // send 3 words to GPMI
                  BF_APBH_CHn_CMD_WAIT4ENDCMD   (1) | // wait for command to finish before continuing
                  BF_APBH_CHn_CMD_SEMAPHORE     (0) |
                  BF_APBH_CHn_CMD_NANDWAIT4READY(1) | // wait for nand to be ready
                  BF_APBH_CHn_CMD_NANDLOCK      (1) | // need nand lock to be thread safe while turn-off ECC8
                  BF_APBH_CHn_CMD_IRQONCMPLT    (0) |
                  BF_APBH_CHn_CMD_CHAIN         (1) | // follow chain to next command
                  BV_FLD(APBH_CHn_CMD, COMMAND, NO_DMA_XFER);      // no dma transfer

read[5].dma_bar = NULL;                                            // field not used

// 3 words sent to the GPMI
read[5].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, READ)      |
                     BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)     |
                     BV_FLD(GPMI_CTRL0, LOCK_CS,      DISABLED)  |
                     BF_GPMI_CTRL0_CS                 (2)        | // must correspond to NAND CS used
                     BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_DATA) |
                     BF_GPMI_CTRL0_ADDRESS_INCREMENT  (0)        |
                     BF_GPMI_CTRL0_XFER_COUNT         (0);

read[5].gpmi_compare = NULL;                                 // field not used but necessary to set eccctrl

read[5].gpmi_eccctrl = BV_FLD(GPMI_ECCCTRL, ENABLE_ECC, DISABLE);  // disable the ECC block

//--------------------------------------------------------------------------
// Descriptor 7: deassert nand lock
//--------------------------------------------------------------------------
read[6].dma_nxtcmdar = NULL;                                 // not used since this is last descriptor

read[6].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0)      |   // no dma transfer
                  BF_APBH_CHn_CMD_CMDWORDS      (0)      |   // no words sent to GPMI
                  BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)      |   // wait for command to finish before continuing
                  BF_APBH_CHn_CMD_SEMAPHORE     (0)      |
                  BF_APBH_CHn_CMD_NANDWAIT4READY(0)      |
                  BF_APBH_CHn_CMD_NANDLOCK      (0)      |   // relinquish nand lock
                  BF_APBH_CHn_CMD_IRQONCMPLT    (0)      |   // ECC8 engine generates interrupt
                  BF_APBH_CHn_CMD_CHAIN         (0)      |   // terminate DMA chain processing
                  BV_FLD(APBH_CHn_CMD, COMMAND, NO_DMA_XFER);  // no dma transfer

read[6].dma_bar = NULL;                                      // field not used
```

14.2.3.2 Using the Decoder

As illustrated in Figure 14-11 and the sample code in Section 14.2.3.1, "DMA Structure Code Example:"

- DMA descriptor 1 prepares the NAND for data read by using the GPMI to issue a NAND read setup command byte under "CLE", then sends a 5-byte address under "ALE". The ECC8 engine is not used for these commands.

- DMA descriptor 2 issues a one-byte read execute command to the NAND device that triggers its read access. The NAND then goes not ready.
- DMA descriptor 3 performs a wait for ready operation allowing the DMA chain to remain dormant until the NAND device completes its read access time.
- DMA descriptor 5 handles the reading and error correction of the NAND data. This command's PIOs activate the ECC8 engine to write the read NAND data to system memory and to process it for any errors that need to be corrected. This DMA descriptor contains two PIO values that are system memory addresses pointing to the PAYLOAD data area and to the AUXILIARY data area. These addresses are used by the ECC8 engine's AHB master to move data into system memory and to correct it. While this example is reading an entire 4K page—payload plus metadata—it is equally possible to read just one 512-byte payload block or just the uniquely protected metadata block in a single 7 DMA structure transfer.
- DMA descriptor 6 disables the ECC8 engine with the NANDLOCK asserted. This is necessary to ensure that the GPMI resource is not arbitrated to another DMA channel when multiple DMA channels are active concurrently.
- DMA descriptor 7 deasserts the NANDLOCK to free up the GPMI resource to another channel.

As the ECC8 block receives data from the GPMI:

- The decoder transforms the read NAND data block into an RS code word and computes the codeword syndrome.
- If no errors are present, then the ECC8 block can immediately report back to firmware. This report is passed as the HW_ECC8_CTRL_COMPLETE_IRQ interrupt status bit and the associated status registers in HW_ECC8_STATUS0/1 registers.
- If an error is present, then the ECC8 block corrects the necessary data block or parity block bytes, if possible (not all errors are correctable).

As the RS decoder reads the data block and the 9-byte or 18-byte parity block, it records a special condition, i.e., that all of the bits of a payload data block or metadata block are one, including any associated parity bytes. The "all-ones" case for both parity and data indicates an erased block in the NAND device.

The HW_ECC8_STATUS0 register contains a 4-bit field that indicates the final status of the auxiliary block. HW_ECC8_STATUS1 contains a similar 4-bit field for each of the 512-byte payload data blocks.

- A value of 0x0 indicates no errors found for a block.
- A value of 1 to 8 inclusive indicates that many correctable errors were found and fixed.
- A value of 0xC is reported for any block that was not actually transferred during a specific transaction, i.e., its BLOCK_MASK bit was a zero for the transaction.
- A value of 0xE indicates uncorrectable errors detected on the block.
- A value of 0xF indicates that the block was in the special ALL ONES state and is therefore considered to be an ERASED block.
- All other values are disallowed by the hardware design.

Recall that up to four NAND devices can have DMA chains in-flight at once, i.e. they can all be contending for access to the GPMI data bus. It is impossible to predict which NAND device will enter the ECC8

engine with a transfer first, because each chain includes a wait4ready command structure. As a result, firmware should look at the HW_ECC8_STATUS0_COMPLETED_CE bit field to determine which block is being reported in the status register. There is also a 16 bit HANDLE field in the HW_GPMI_ECCCTRL register that is passed down the pipeline with each transaction. This handle field can be used to speed firmware's detection of which transaction is being reported.

These examples of reading and writing have focused on full page transfers of 4K page NAND devices. Set HW_GPMI_ECCCTRL_ECC_CMD to a value of HW_GPMI_ECCCTRL_ECC_CMD_ENCODE_4_BIT or to a value of HW_GPMI_ECCCTRL_ECC_CMD_DECODE_4_BIT to enable encode and decode of up to full page transfers of 2K page NAND devices.

To reiterate, you can select a single block to transfer within one transaction by setting only one bit in the HW_GPMI_ECCCTRL_BUFFER_MASK bit field. There is a 1:1 correspondence between a bit in this bit field and a 512-byte buffer address offset into the payload area pointed to by the HW_GPMI_PAYLOAD register. The ECC8 and GPMI blocks are designed to be very efficient at reading single 512-byte pages in one transaction. With no errors, the transaction takes less than 20 HCLKs longer than the time to read the raw data from the NAND.

Additionally, you can select multiple contiguous blocks to transfer within one transaction by setting the respective bits in the HW_GPMI_BUFFER_MASK bit fields. The selected bits must represent a contiguous block of data in the NAND.

To summarize, the APBH DMA command chain for a Reed-Solomon decode operation is shown in Figure 14-11. Seven DMA command structures must be present for each NAND read transaction decoded by the ECC8. The seven DMA command structures for multiple NAND read transaction blocks can be chained together to make larger units of work for the ECC8, and each will produce an appropriate error report in the ECC8 PIO space. Multiple NAND devices can have such multiple chains scheduled. The results can come back out of order with respect to the multiple chains.

If uncorrectable errors occur, it is up to software to determine how to deal with the bad block. One strategy might be to reread the data from NAND flash in the hope that enough soft errors will have been removed to make correction possible on a second pass.

14.2.4 Interrupts

There are two interrupt sources used in processing ECC8 protected NAND read and write transfers. Since all ECC8 operations are initiated by GPMI DMA command structures, the DMA completion interrupt for the GPMI is an important ISR. Both of the flow charts of Figure 14-6 and Figure 14-9 show the GPMI DMA complete ISR skeleton. In both reads and writes, the GPMI DMA completion interrupt is used to schedule work ***INTO*** the error correction pipeline. As the front end processing completes, the DMA interrupt is generated and additional work, i.e. DMA chains, are passed to the GPMI DMA to keep it

"*fed*". For write operations, this is the only interrupt that will be generated for processing the NAND write transfer.

For reads, however, two interrupts are needed. Every read is started by a GPMI DMA command chain and the front end queue is fed as described above. The back end of the read pipeline is "drained" by monitoring the ECC8 completion interrupt found in HW_ECC8_CTRL_COMPLETE_IRQ.

When the NAND is read using ECC (as configured in the DMA descriptor for that read), the CPU must wait for **both** the GPMI interrupt and the ECC interrupt, and the interrupts may occur in either order. That is, the GPMI interrupt may happen before the ECC interrupt, or vice-versa. Software needs to wait until both interrupts have occurred to know that the data has been delivered by the DMA.

An ECC8 transaction consists of reading or writing all of the blocks requested in the HW_GPMI_ECCCTRL_BUFFER_MASK bit field. As every read transaction completes, it posts the status of all of the blocks to the HW_ECC8_STATUS0 and HW_ECC8_STATUS1 registers and sets the completion interrupt. The five stages of the ECC8 read pipeline completes, one in the GPMI and four in the ECC8, are independently stalled as they complete and try to deliver to the next stage in the data flow. Several of these stages can be skipped if no-errors are found or once an uncorrectable error is found in a block.

In any case, the final stage will stall if the status register is busy waiting for the CPU to take status register results. The hardware monitors the state of the HW_ECC8_CTRL_COMPLETE_IRQ bit. If it is still set when the last pipeline stage is ready to post data, then the stage will stall. It follows that the next previous stage will stall when it is ready to hand off work to the final stage, and so on up the pipeline.

WARNING: It is important that firmware read the STATUS0/1 results and save them before clearing the interrupt request bit otherwise a transaction and its results could be completely lost.

14.3 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, "Correct Way to Soft Reset a Block," for additional information on using the SFTRST and CLKGATE bit fields.

14.4 Programmable Registers

The following registers are available for programmer access and control of the ECC8 hardware accelerator.

14.4.1 Hardware ECC Accelerator Control Register Description

The Hardware ECC Accelerator Control Register provides overall control of the hardware ECC accelerator.

HW_ECC8_CTRL	0x000
HW_ECC8_CTRL_SET	0x004
HW_ECC8_CTRL_CLR	0x008
HW_ECC8_CTRL_TOG	0x00C

Table 14-1. HW_ECC8_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	AHBM_SFTRST	RSRVD2	THROTTLE				RSRVD1													DEBUG_STALL_IRQ_EN	DEBUG_WRITE_IRQ_EN	COMPLETE_IRQ_EN	RSRVD0				BM_ERROR_IRQ	DEBUG_STALL_IRQ	DEBUG_WRITE_IRQ	COMPLETE_IRQ

Table 14-2. HW_ECC8_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	0 = Normal ECC8 operation. 1 = Disable clocking with the ECC8 and hold it in its reset (lowest power) state (default). This bit can be turned on and then off to reset the ECC8 block to its default state. This bit resets all state machines except for the AHB master state machine. RUN = 0x0 Allow ECC8 to operate normally. RESET = 0x1 Hold ECC8 in reset.
30	**CLKGATE**	RW	0x1	This bit must be cleared to 0 for normal operation. When set to 1, it gates off the clocks to the block. RUN = 0x0 Allow ECC8 to operate normally. NO_CLKS = 0x1 Do not clock ECC8 gates in order to minimize power consumption.
29	**AHBM_SFTRST**	RW	0x1	Resets the AHB state machine. 0 = Normal ECC8 operation. 1 = Disable clocking with the ECC8 and hold it in its reset (lowest power) state (default). This bit can be turned on and then off to reset the ECC8 block to its default state. Do not use this bit for normal device soft-resets unless instructed to do so by Freescale. RUN = 0x0 Allow ECC8 to operate normally. RESET = 0x1 Hold ECC8 in reset.
28	**RSRVD2**	RO	0x0	Reserved.
27:24	**THROTTLE**	RW	0x0	Non-zero values will hold off that number of HCLKs between success burst requests on the AHB.
23:11	**RSRVD1**	RO	0x0	Reserved.
10	**DEBUG_STALL_IRQ_EN**	RW	0x0	1 = Interrupt on debug stall mode is enabled. The IRQ is raised on every block

Table 14-2. HW_ECC8_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
9	**DEBUG_WRITE_IRQ_EN**	RW	0x0	1 = Interrupt on debug write mode is enabled. The IRQ is raised on every transfer. In this mode, no correction occurs.
8	**COMPLETE_IRQ_EN**	RW	0x0	1 = Interrupt on completion of correction is enabled.
7:4	**RSRVD0**	RO	0x0	Reserved.
3	**BM_ERROR_IRQ**	RW	0x0	AHB Bus Interface Error Interrupt Status. Write a 1 to the SCT clear address to clear the interrupt status bit.
2	**DEBUG_STALL_IRQ**	RW	0x0	Debug Stall Interrupt Status. Write a 1 to the SCT clear address to clear the interrupt status bit.
1	**DEBUG_WRITE_IRQ**	RW	0x0	Debug Write Interrupt Status. Write a 1 to the SCT clear address to clear the interrupt status bit.
0	**COMPLETE_IRQ**	RW	0x0	External Interrupt Line Status. Write a 1 to the SCT clear address to clear the interrupt status bit. Note: Subsequent ECC completions will be held off as long as this bit is set. Be sure to read the data from HW_ECC8_STATUS0/1 before clearing this interrupt bit.

14.4.2 Hardware ECC Accelerator Status Register 0 Description

The Hardware ECC Accelerator Status Register 0 provides overall status of the hardware ECC accelerator.

HW_ECC8_STATUS0 0x010

Table 14-3. HW_ECC8_STATUS0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
HANDLE												COMPLETED_CE				RS8ECC_ENC_PRESENT	RS8ECC_DEC_PRESENT	RS4ECC_ENC_PRESENT	RS4ECC_DEC_PRESENT	STATUS_AUX				RSVD1			ALLONES	CORRECTED	UNCORRECTABLE	RSVD0	

Table 14-4. HW_ECC8_STATUS0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:20	**HANDLE**	RO	0x0	Software supplies a 12-bit handle for this transfer as part of the GPMI DMA PIO operation that started the transaction. That handle passes down the pipeline and ends up here at the time the ECC8 interrupt is signaled.
19:16	**COMPLETED_CE**	RO	0x0	Chip enable number corresponding to the NAND device from which this data came.
15	**RS8ECC_ENC_PRESENT**	RO	0x1	Reserved.
14	**RS8ECC_DEC_PRESENT**	RO	0x1	Reserved.
13	**RS4ECC_ENC_PRESENT**	RO	0x1	Reserved.

Table 14-4. HW_ECC8_STATUS0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
12	**RS4ECC_DEC_PRESENT**	RO	0x1	Reserved.
11:8	**STATUS_AUX**	RO	0xc	Count of symbols in error during processing of auxiliary data area. 0xF indicates uncorrectable. NO_ERRORS = 0x0 No errors occurred. ONE_CORRECTABLE = 0x1 One correctable error occurred. TWO_CORRECTABLE = 0x2 Two correctable errors occurred. THREE_CORRECTABLE = 0x3 Three correctable errors occurred. FOUR_CORRECTABLE = 0x4 Four correctable errors occurred. NOT_CHECKED = 0xC This block was not examined by the ECC8. UNCORRECTABLE = 0xE Errors occurred that were uncorrectable. ALL_ONES = 0xF All bits are 1.
7:5	**RSVD1**	RO	0x0	Reserved.
4	**ALLONES**	RO	0x1	1 = All data bits of this transaction are 1.
3	**CORRECTED**	RO	0x0	1 = At least one correctable error encountered during last processing cycle.
2	**UNCORRECTABLE**	RO	0x0	1 = Uncorrectable error encountered during last processing cycle.
1:0	**RSVD0**	RO	0x0	Reserved.

DESCRIPTION:

The Hardware ECC Accelerator Status Register 0 provides visibility into the run-time status of the ECC8. The register also reflects the ECC8 configurations supported in this version of the SoC.

EXAMPLE:

```
Have8BitRSEncode = HW_ECC8_STAT.B.RS8ENC_PRESENT;
```

14.4.3 Hardware ECC Accelerator Status Register 1 Description

The Hardware ECC Accelerator Status Register 1 provides overall status of the hardware ECC accelerator.

HW_ECC8_STATUS1 0x020

Table 14-5. HW_ECC8_STATUS1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
STATUS_PAYLOAD7				STATUS_PAYLOAD6				STATUS_PAYLOAD5				STATUS_PAYLOAD4				STATUS_PAYLOAD3				STATUS_PAYLOAD2				STATUS_PAYLOAD1				STATUS_PAYLOAD0			

Table 14-6. HW_ECC8_STATUS1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**STATUS_PAYLOAD7**	RO	0xc	Count of symbols in error during processing of payload area 7. 0xF indicates uncorrectable. NO_ERRORS = 0x0 No errors occurred. ONE_CORRECTABLE = 0x1 One correctable error occurred. TWO_CORRECTABLE = 0x2 Two correctable errors occurred. THREE_CORRECTABLE = 0x3 Three correctable errors occurred. FOUR_CORRECTABLE = 0x4 Four correctable errors occurred. FIVE_CORRECTABLE = 0x5 Five correctable errors occurred. SIX_CORRECTABLE = 0x6 Six correctable errors occurred. SEVEN_CORRECTABLE = 0x7 Seven correctable errors occurred. EIGHT_CORRECTABLE = 0x8 Eight correctable errors occurred. NOT_CHECKED = 0xC This block was not examined by the ECC8. UNCORRECTABLE = 0xE Errors occurred that were uncorrectable. ALL_ONES = 0xF All bits are 1.
27:24	**STATUS_PAYLOAD6**	RO	0xc	Count of symbols in error during processing of payload area 6. 0xF indicates uncorrectable. NO_ERRORS = 0x0 No errors occurred. ONE_CORRECTABLE = 0x1 One correctable error occurred. TWO_CORRECTABLE = 0x2 Two correctable errors occurred. THREE_CORRECTABLE = 0x3 Three correctable errors occurred. FOUR_CORRECTABLE = 0x4 Four correctable errors occurred. FIVE_CORRECTABLE = 0x5 Five correctable errors occurred. SIX_CORRECTABLE = 0x6 Six correctable errors occurred. SEVEN_CORRECTABLE = 0x7 Seven correctable errors occurred. EIGHT_CORRECTABLE = 0x8 Eight correctable errors occurred. NOT_CHECKED = 0xC This block was not examined by the ECC8. UNCORRECTABLE = 0xE Errors occurred that were uncorrectable. ALL_ONES = 0xF All bits are 1.
23:20	**STATUS_PAYLOAD5**	RO	0xc	Count of symbols in error during processing of payload area 5. 0xF indicates uncorrectable. NO_ERRORS = 0x0 No errors occurred. ONE_CORRECTABLE = 0x1 One correctable error occurred. TWO_CORRECTABLE = 0x2 Two correctable errors occurred. THREE_CORRECTABLE = 0x3 Three correctable errors occurred. FOUR_CORRECTABLE = 0x4 Four correctable errors occurred. FIVE_CORRECTABLE = 0x5 Five correctable errors occurred. SIX_CORRECTABLE = 0x6 Six correctable errors occurred. SEVEN_CORRECTABLE = 0x7 Seven correctable errors occurred. EIGHT_CORRECTABLE = 0x8 Eight correctable errors occurred. NOT_CHECKED = 0xC This block was not examined by the ECC8. UNCORRECTABLE = 0xE Errors occurred that were uncorrectable. ALL_ONES = 0xF All bits are 1.
19:16	**STATUS_PAYLOAD4**	RO	0xc	Count of symbols in error during processing of payload area 4. 0xF indicates uncorrectable. NO_ERRORS = 0x0 No errors occurred. ONE_CORRECTABLE = 0x1 One correctable error occurred. TWO_CORRECTABLE = 0x2 Two correctable errors occurred. THREE_CORRECTABLE = 0x3 Three correctable errors occurred. FOUR_CORRECTABLE = 0x4 Four correctable errors occurred. FIVE_CORRECTABLE = 0x5 Five correctable errors occurred. SIX_CORRECTABLE = 0x6 Six correctable errors occurred. SEVEN_CORRECTABLE = 0x7 Seven correctable errors occurred. EIGHT_CORRECTABLE = 0x8 Eight correctable errors occurred. NOT_CHECKED = 0xC This block was not examined by the ECC8. UNCORRECTABLE = 0xE Errors occurred that were uncorrectable. ALL_ONES = 0xF All bits are 1.
15:12	**STATUS_PAYLOAD3**	RO	0xc	Count of symbols in error during processing of payload area 3. 0xF indicates uncorrectable. NO_ERRORS = 0x0 No errors occurred. ONE_CORRECTABLE = 0x1 One correctable error occurred. TWO_CORRECTABLE = 0x2 Two correctable errors occurred. THREE_CORRECTABLE = 0x3 Three correctable errors occurred. FOUR_CORRECTABLE = 0x4 Four correctable errors occurred. FIVE_CORRECTABLE = 0x5 Five correctable errors occurred. SIX_CORRECTABLE = 0x6 Six correctable errors occurred. SEVEN_CORRECTABLE = 0x7 Seven correctable errors occurred. EIGHT_CORRECTABLE = 0x8 Eight correctable errors occurred. NOT_CHECKED = 0xC This block was not examined by the ECC8. UNCORRECTABLE = 0xE Errors occurred that were uncorrectable. ALL_ONES = 0xF All bits are 1.

Table 14-6. HW_ECC8_STATUS1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
11:8	**STATUS_PAYLOAD2**	RO	0xc	Count of symbols in error during processing of payload area 2. 0xF indicates uncorrectable. NO_ERRORS = 0x0 No errors occurred. ONE_CORRECTABLE = 0x1 One correctable error occurred. TWO_CORRECTABLE = 0x2 Two correctable errors occurred. THREE_CORRECTABLE = 0x3 Three correctable errors occurred. FOUR_CORRECTABLE = 0x4 Four correctable errors occurred. FIVE_CORRECTABLE = 0x5 Five correctable errors occurred. SIX_CORRECTABLE = 0x6 Six correctable errors occurred. SEVEN_CORRECTABLE = 0x7 Seven correctable errors occurred. EIGHT_CORRECTABLE = 0x8 Eight correctable errors occurred. NOT_CHECKED = 0xC This block was not examined by the ECC8. UNCORRECTABLE = 0xE Errors occurred that were uncorrectable. ALL_ONES = 0xF All bits are 1.
7:4	**STATUS_PAYLOAD1**	RO	0xc	Count of symbols in error during processing of payload area 1. 0xF indicates uncorrectable. NO_ERRORS = 0x0 No errors occurred. ONE_CORRECTABLE = 0x1 One correctable error occurred. TWO_CORRECTABLE = 0x2 Two correctable errors occurred. THREE_CORRECTABLE = 0x3 Three correctable errors occurred. FOUR_CORRECTABLE = 0x4 Four correctable errors occurred. FIVE_CORRECTABLE = 0x5 Five correctable errors occurred. SIX_CORRECTABLE = 0x6 Six correctable errors occurred. SEVEN_CORRECTABLE = 0x7 Seven correctable errors occurred. EIGHT_CORRECTABLE = 0x8 Eight correctable errors occurred. NOT_CHECKED = 0xC This block was not examined by the ECC8. UNCORRECTABLE = 0xE Errors occurred that were uncorrectable. ALL_ONES = 0xF All bits are 1.
3:0	**STATUS_PAYLOAD0**	RO	0xc	Count of symbols in error during processing of payload area 0. 0xF indicates uncorrectable. NO_ERRORS = 0x0 No errors occurred. ONE_CORRECTABLE = 0x1 One correctable error occurred. TWO_CORRECTABLE = 0x2 Two correctable errors occurred. THREE_CORRECTABLE = 0x3 Three correctable errors occurred. FOUR_CORRECTABLE = 0x4 Four correctable errors occurred. FIVE_CORRECTABLE = 0x5 Five correctable errors occurred. SIX_CORRECTABLE = 0x6 Six correctable errors occurred. SEVEN_CORRECTABLE = 0x7 Seven correctable errors occurred. EIGHT_CORRECTABLE = 0x8 Eight correctable errors occurred. NOT_CHECKED = 0xC This block was not examined by the ECC8. UNCORRECTABLE = 0xE Errors occurred that were uncorrectable. ALL_ONES = 0xF All bits are 1.

DESCRIPTION:

The Hardware ECC Accelerator Status Register 1 provides visibility into the run-time status of the ECC8. The register also reflects the ECC8 configurations supported in this version of the SoC.

EXAMPLE:

```
if(HW_ECC8_STAT1.B._PAYLOAD0) LifeIsGood();
```

14.4.4 Hardware ECC Accelerator Debug Register 0 Description

The ECC8 internal state machines and signals can be seen in the Hardware ECC Accelerator Debug Register 0.

HW_ECC8_DEBUG0	0x030
HW_ECC8_DEBUG0_SET	0x034
HW_ECC8_DEBUG0_CLR	0x038
HW_ECC8_DEBUG0_TOG	0x03C

Table 14-7. HW_ECC8_DEBUG0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1							KES_DEBUG_SYNDROME_SYMBOL									KES_DEBUG_SHIFT_SYND	KES_DEBUG_PAYLOAD_FLAG	KES_DEBUG_MODE4K	KES_DEBUG_KICK	KES_STANDALONE	KES_DEBUG_STEP	KES_DEBUG_STALL	BM_KES_TEST_BYPASS	RSRVD0		DEBUG_REG_SELECT					

Table 14-8. HW_ECC8_DEBUG0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSRVD1**	RO	0x0	Reserved.
24:16	**KES_DEBUG_SYNDROME_SYMBOL**	RW	0x0	The 9-bit value in this bit field will be shifted into the syndrome register array at the input of the KES engine whenever HW_ECC8_DEBUG0_KES_DEBUG_SHIFT_SYND is toggled. NORMAL = 0x0 Bus master address generator for synd_gen writes operates normally. TEST_MODE = 0x1 Bus master address generator always addresses last four bytes in auxiliary block.
15	**KES_DEBUG_SHIFT_SYND**	RW	0x0	Toggling this bit causes the value in HW_ECC8_DEBUG0_KES_SYNDROME_SYMBOL to be shifted into the syndrome register array at the input to the KES engine. After shifting in 16 symbols, one can kick off both KES and CF cycles by toggling HW_ECC8_DEBUG0_KES_DEBUG_KICK. Be sure to set KES_ECC8_DEBUG0_KES_STANDALONE mode to 1 before kicking.
14	**KES_DEBUG_PAYLOAD_FLAG**	RW	0x0	When running the standalone debug mode on the error calculator, the state of this bit is presented to the KES engine as the input payload flag. DATA = 0x1 Payload is set for 512 byte data block. AUX = 0x1 Payload is set for 65 or 19 byte auxiliary block.
13	**KES_DEBUG_MODE4K**	RW	0x0	When running the standalone debug mode on the error calculator, the state of this bit is presented to the KES engine as the input mode (4K or 2K pages). 4k = 0x1 Mode is set for 4K NAND pages. 2k = 0x1 Mode is set for 2K NAND pages.
12	**KES_DEBUG_KICK**	RW	0x0	Toggling causes KES engine FSM to start as if kicked by the bus master. This allows standalone testing of the KES and Chien Search engines. Be sure to set KES_ECC8_DEBUG0_KES_STANDALONE mode to 1 before kicking.

Table 14-8. HW_ECC8_DEBUG0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
11	**KES_STANDALONE**	RW	0x0	Set to 1 to cause the KES engine to suppress toggling the KES_BM_DONE signal to the bus master and to suppress toggling the CF_BM_DONE signal by the CF engine. NORMAL = 0x0 Bus master address generator for synd_gen writes operates normally. TEST_MODE = 0x1 Bus master address generator always addresses last four bytes in auxiliary block.
10	**KES_DEBUG_STEP**	RW	0x0	Toggling this bit causes the KES FSM to skip past the stall state if it is in DEBUG_STALL mode and it has completed processing a block.
9	**KES_DEBUG_STALL**	RW	0x0	Set to 1 to cause KES FSM to stall after notifying the Chien search engine to start processing its block but before notifying the bus master that the KES computation is complete. This allows a diagnostic to stall the FSM after each block's key equations are solved. This also has the effect of stalling the CSFE search engine so its state can be examined after it finishes processing the KES stalled block. NORMAL = 0x0 KES FSM proceeds to next block supplied by bus master. WAIT = 0x1 KES FSM waits after current equations are solved and the search engine is started.
8	**BM_KES_TEST_BYPASS**	RW	0x0	1 = Point all synd_gen writes to dummy area at the end of the auxiliary block so that diagnostics can preload all payload, parity bytes, and computed syndrome bytes for test the KES engine. NORMAL = 0x0 Bus master address generator for synd_gen writes operates normally. TEST_MODE = 0x1 Bus master address generator always addresses last four bytes in auxiliary block.
7:6	**RSRVD0**	RO	0x0	Reserved.
5:0	**DEBUG_REG_SELECT**	RW	0x0	The value loaded in this bit field is used to select the internal register state view of KES engine or the Chien search engine.

DESCRIPTION:

The Hardware ECC Accelerator Debug Register 0 provides access to various internal state information which might prove useful during hardware debug and validation.

EXAMPLE:

```
Value = HW_ECC8_DEBUG0.U; // diagnostic programs can read and act upon various bit fields.
```

14.4.5 KES Debug Read Register Description

The hardware ECC accelerator key equation solver internal state machines and signals can be seen in the KES Debug Read Register.

HW_ECC8_DBGKESREAD 0x040

Table 14-9. HW_ECC8_DBGKESREAD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
VALUES																															

Table 14-10. HW_ECC8_DBGKESREAD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**VALUES**	RO	0x0	Reserved.

14.4.6 Chien Search Forney Evaluator Debug Read Register Description

The hardware ECC accelerator Chien Search Forney Evaluator (CSFE) internal state machines and signals can be seen in the Chien Search Forney Evaluator Debug Read Register.

HW_ECC8_DBGCSFEREAD 0x050

Table 14-11. HW_ECC8_DBGCSFEREAD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
VALUES																															

Table 14-12. HW_ECC8_DBGCSFEREAD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**VALUES**	RO	0x0	Reserved.

14.4.7 Syndrome Generator Debug Read Register Description

The hardware ECC accelerator syndrome generator internal state machines and signals can be seen in the Syndrome Generator Debug Read Register.

HW_ECC8_DBGSYNDGENREAD 0x060

Table 14-13. HW_ECC8_DBGSYNDGENREAD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
VALUES																															

Table 14-14. HW_ECC8_DBGSYNDGENREAD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**VALUES**	RO	0x0	Reserved.

14.4.8 AHB Master and ECC8 Controller Debug Read Register Description

The hardware ECC accelerator AHB bus master and ECC8 controller internal state machines and signals can be seen in the AHB Master and ECC8 Controller Debug Read Register.

HW_ECC8_DBGAHBMREAD 0x070

Table 14-15. HW_ECC8_DBGAHBMREAD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
VALUES																															

Table 14-16. HW_ECC8_DBGAHBMREAD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**VALUES**	RO	0x0	Reserved.

14.4.9 ECC8 Block Name Register Description

This register presents a read-only view of the block name string ECC8.

HW_ECC8_BLOCKNAME 0x080

Table 14-17. HW_ECC8_BLOCKNAME

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
NAME																															

Table 14-18. HW_ECC8_BLOCKNAME Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**NAME**	RO	0x38434345	Should be the ASCII characters 8, C, C, E.

DESCRIPTION:

This register presents a fixed-pattern, read-only value for test purposes. It can be read as an ASCII string with the zero termination coming from the first byte of the VERSION register.

EXAMPLE:

```
char *cp = ((char *)HW_ECC8_BLOCKNAME_ADDR); reads back ECC8ECC8ECC8ECC8 with zero termination.
```

14.4.10 ECC8 Version Register Description

This register indicates the version of the block for debug purposes.

HW_ECC8_VERSION 0x0a0

Table 14-19. HW_ECC8_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 14-20. HW_ECC8_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x01	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x0	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
if (HW_ECC8_VERSION.B.MAJOR != 1) Error();
```

ECC8 Block v1.1, Revision 2.5

Chapter 15
20-BIT Correcting ECC Accelerator (BCH)

This chapter describes the hardware Bose Ray-Choudhury Hocquenghem (BCH) ECC accelerator available on i.MX23. It provides detailed descriptions of how to use the accelerator and programmable registers (described in Section 15.6, "Programmable Registers").

The BCH block is functionally very similar to the Reed-Solomon based ECC8 hardware available in previous chips. The primary difference from a programming standpoint is that all data transfer to/from memory is handled by the BCH directly instead of using DMA for data write operations. DMA is still used in programming the GPMI control registers.

15.1 Overview

The hardware ECC accelerator provides a forward error-correction function for improving the reliability of various storage media that may be attached to the i.MX23. For example, modern high-density NAND flash devices presume the existence of forward error-correction algorithms to correct some soft and/or hard bit errors within the device, allowing for higher device yields and, therefore, lower NAND device costs.

The Bose, Ray-Chaudhuri, Hocquenghem (BCH) Encoder and Decoder module is capable of correcting from 2 to 20 single bit errors within a block of data no larger than about 900 bytes (512 bytes is typical) in applications such as protecting data and resources stored on modern NAND flash devices. The correction level in the BCH block is programmable to provide flexibility for varying applications and configurations of flash page size. The design can be programmed to encode protection of 2, 4, 8, 10, 12, 14, 16, 18, or 20 bit errors when writing flash and to correct the corresponding number of errors on decode. The correction level when decoding MUST be programmed to the same correction level as was used during the encode phase.

BCH-codes are a type of block-code, which implies that all error-correction is performed over a block of N-symbols. The BCH operation will be performed over GF(2^{13} = 8192), which is the Galois Field consisting of 8191 one-bit symbols. BCH-encoding (or encode for any block-code) can be performed by two algorithms: systematic encoding or multiplicative encoding. Systematic encoding is the process of reading all the symbols which constitute a block, dividing continuously these symbols by the generator polynomial for the GF(8192) and appending the resulting ***t*** parity symbols to the block to create a BCH codeword (where ***t*** is the number of correctable bits).

The BCH encode process creates *t* 13-bit parity symbols for each data block when the data is written to the flash device. The parity symbols are written to the flash device after the corresponding data block,

and together these are collectively called the codeword. The codeword can be used during the decode process to correct errors that occur in either the data or parity blocks.

The BCH decoder processes code words in a 4-step fashion:

- Syndrome Calculation (SC): This is the process of reading in all of the symbols of the codeword and continuously dividing by the generator polynomial for the field. 2*t syndromes must be calculated for each codeword and inspection of the syndromes determines if there are errors: a non-zero set of syndromes indicates one or more errors. This process is implemented in parallel hardware to minimize processing time since it must be done every time the decode is performed.
- Key Equation Solver (KES): The syndromes represent 2t-linear equations with 2t-unknown variables. The process of solving these equations and selecting from the numerous solutions constitutes the KES module. When the KES block completes its operations, it generates an error locator polynomial (sigma) that is used in the proceeding block to determine the locations and values of the errors.
- Chien Search (CS): This block takes input from the KES block and uses the Chien Algorithm for finding the locations of the errors based on the error locator polynomial. The method basically involves substituting all 8191 symbols from the GF(8192) into the locator polynomial. All evaluations that produce a zero solution indicate locations of the various errors. Since each located error corresponds to a single bit, the bit in the original data may be corrected by simply flipping the polarity of the incorrect location.
- Correction: this block has to convert the symbol index and mask information to memory byte indexes and masks.

The BCH block was designed to operate in a pipelined fashion to maximize throughput. Aside from the initial latency to fill the pipeline stages, the BCH throughput is faster than the fastest GPMI read rate of 2 cycles/byte. Thus, the bottleneck in performing NAND reads and error corrections is the GPMI read rate. Current GPMI read rates are approximately 3 cycles/byte for the current generation of NANDs. The CPU is not directly involved in generating parity symbols, checking for errors, or correcting them.

The hardware BCH accelerator is illustrated in Figure 15-1.

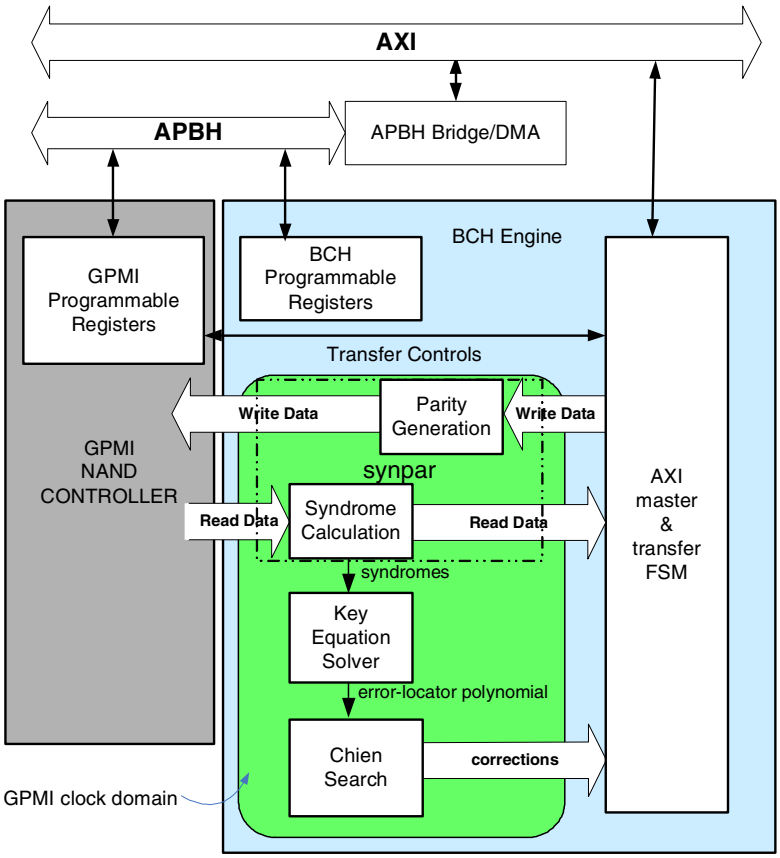


Figure 15-1. Hardware BCH Accelerator

15.2 Operation

Before performing any NAND flash read or write operations, software should first program the BCH's flash layout registers (see Section 15.2.2, "Flash Page Layout") to specify how data is to be formatted on the flash device. The BCH hardware allows full programmability over the flash page layout to enable users flexibility in balancing ECC correction levels and ever-changing flash page sizes.

To initiate a NAND flash write, software will program a GPMI DMA operation. The DMA need only program the GPMI control registers (and handle the requisite flash addressing handshakes) since the BCH will handle all data operations using its AXI bus interface. The BCH will then send the data to the GPMI controller to be written to flash as it computes the parity symbols. At the end of each data block the BCH will insert the parity symbols into the data stream so that the GPMI sees only a continuous stream of data to be written.

NAND flash read operations operate in a similar manner. As the GPMI controller reads the device, all data is sent to the BCH hardware for error detection/correction. The BCH controller writes all incoming read data to system memory and in parallel computes the syndromes used to detect bit errors. If errors are

detected within a block, the BCH hardware activates the error correction logic to determine where bit errors have occurred and ultimately correct them in the data buffer in system memory. After an entire flash page has been read and corrected, the BCH will signal an interrupt to the CPU.

Figure 15-2 indicates how data read from the GPMI is operated on within the BCH hardware. As the BCH receives data from the GPMI (top row) it is written to memory by the BCH's Bus Interface Unit (BIU) (second row). For blocks requiring correction, the KES logic will be activated after the entire block has been received. Once the error locator polynomial has been computed, the corrections are determined by the Chien Search and fed back to the BIU, which performs a read/modify/write operation on the buffer in memory to correct the data.

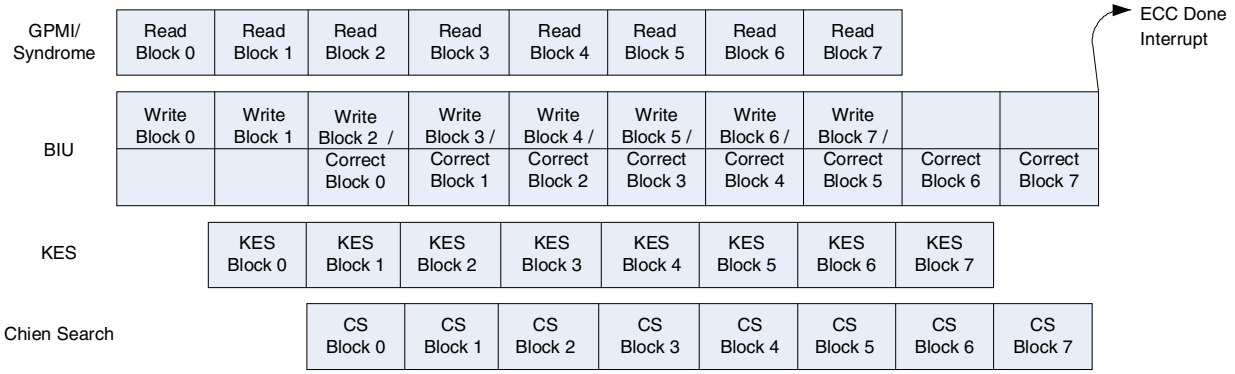


Figure 15-2. Block Pipeline while Reading Flash

15.2.1 BCH Limitations and Assumptions

- The BCH is programmable to support 2, 4, 6, 8, 10, 12, 14, 16, 18, and 20 bit error correction. ECC0 is supported as a passthrough, non-correcting mode.
- Data block sizes must be a multiple of 4 bytes and must be 4-byte aligned in system memory.
- The BCH supports a programmable number of metadata/auxiliary data bytes, from 0 to 255.
- Metadata will be written at the beginning of the flash page to facilitate fast access for filesystem operations.
- Metadata may be treated as an independent block for ECC purposes or combined with the first data block to conserve bits in the flash.
- The BCH does not support a partial page write.
- Flash read operations can read the entire page or the first block on the page.
- The BCH also supports a memory-to-memory mode of operation that does not require the use of DMA or the GPMI.

15.2.2 Flash Page Layout

The BCH supports a fully programmable flash page layout, versus the hardwired modes supported in the former ECC8 engine. The BCH maintains 4 independent layout registers that can describe four completely different NAND devices or layouts. When the BCH initiates an operation, it selects one of the lay-

outs by using the chip select as an index into the HW_BCH_LAYOUTSELECT register the determines which layout should be used for the operation.

Three possible (generic) flash layout schemes are supported, as indicated in Figure 15-3. (In each case, the metadata size may also be programmed to 0 bytes). Metadata may either be combined with the first block of data or the size of the first data block can be programmed to 0 to allow the metadata to be protected by its own ECC parity bits.

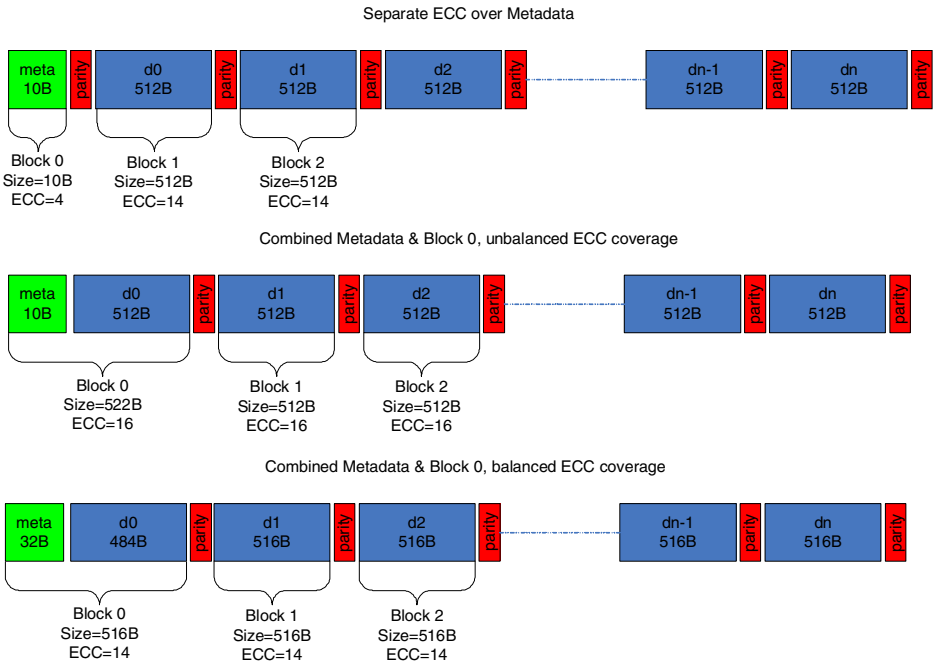


Figure 15-3. FLASH Page Layout Options

Each layout is determined by a pair of registers that define the following parameters:

- DATA0_SIZE: Indicates the number of data bytes in the first block on the page (this should not include parity or metadata bytes). This should be set to 0 when the metadata is to be covered separately with its own ECC. This MUST be a multiple of 4 bytes.
- ECC0: Indicates the ECC level to be used for the first block on the flash (data0+metadata).
- META_SIZE: indicates the number of bytes (from 0-255) that are stored as metadata.
- NBLOCKS: Indicates the number of subsequent “DATAN” blocks on the flash, or the number of blocks following the DATA0 block.
- DATAN_SIZE: Indicates the number of data bytes in all subsequent data blocks. This MUST be a multiple of 4 bytes.
- ECCN: Indicates the ECC level to be used for the subsequent data blocks.
- PAGE_SIZE: Indicates the total number of bytes available per page on the physical flash device. This includes the spare area and is typically 4096+128, 4096+218, or 2048+64 bytes.

15.2.3 Determining the ECC layout for a device

Since the BCH is programmable, a system can trade off ECC levels for flash size and layout configurations. The following examples indicate how to determine a valid layout based on the required storage space and flash size. For all cases, the size of the parity will be 13*ECC level ***bits***-- thus for ECC8, 13 bytes are required (per block).

15.2.3.1 4K+218 flash, 10 bytes metadata, 512 byte data blocks, separate metadata

In this case, we have 8 data blocks each consisting of 512 bytes. Since the flash has 218 “spare” bytes (1744 bits), we first estimate an ECC level for the data blocks by first subtracting the number of metadata bytes from the spare bytes (218-10=208 bytes = 1664 bits) then dividing the number of bits by 8 (number of blocks) and then by 13 (bits per ECC level).

$$(218 - 10) \times 8 = \frac{1664}{13(8)} = 16$$

Thus all the data blocks could be covered by ECC16 if the metadata had no parity. This isn’t acceptable, so assume ECC14 for all the data blocks. We now calculate the number of free bits for the metadata parity as:

$$1664 - (14) \times 13 \times 8 = 208$$

Thus 208 bits remain for metadata parity. Dividing by 13 (bits/ECC) gives 16, thus the metadata can be covered with ECC16. The settings for this device would then be

Table 15-1. Settings for 4K+218 FLASH

Setting	Value
PAGE_SIZE	4096+218=4314=0x10DA
META_SIZE	10=0x0A
DATA0_SIZE	0
ECC0	16=0x10
DATAN_SIZE	512=0x200
ECCN	14=0x0E
NBLOCKS	8

15.2.3.2 4K+128 flash, 10 bytes metadata, 512 byte data blocks, separate metadata

This flash will have 118 bytes available for ECC (after subtracting the metadata size), thus 944 bits. Dividing by 8*13 (number of blocks * bits per ECC level) we get 9.07, thus we can support ECC8 on the data blocks. The number of free spare bits becomes 944-8*8*13=944-832=112, divided by 13 = 8.6, thus the metadata can be also covered by ECC8.

Table 15-2. Settings for 4K+128 FLASH

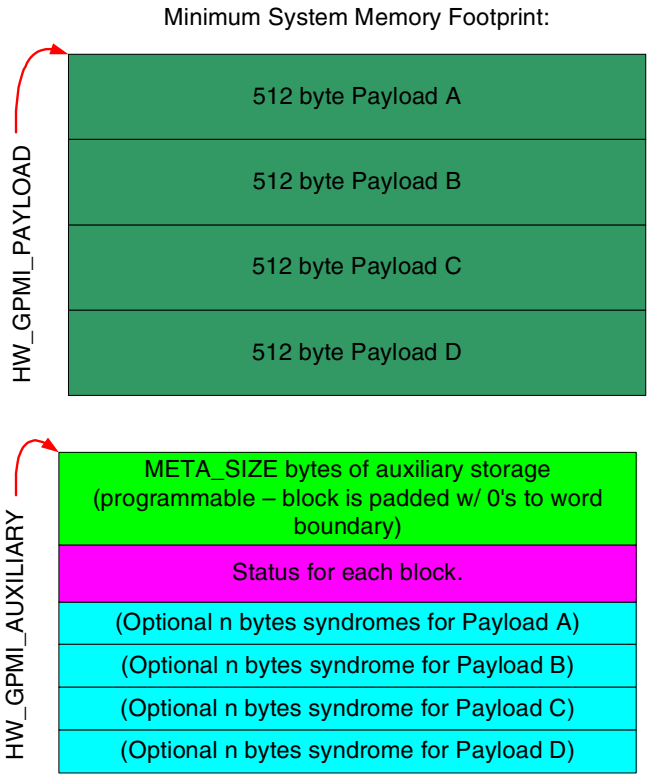
Setting	Value
PAGE_SIZE	4096+128=4224=0x1080
META_SIZE	10=0x0A
DATA0_SIZE	0
ECC0	8
DATAN_SIZE	512=0x200
ECCN	8
NBLOCKS	8

In this case, there will be additional unused spare bits, with the BCH will pad out with zeros.

15.2.4 Data buffers in system memory

While the data on the flash is interleaved with parity symbols, the BCH assumes that the data buffers in memory are contiguous. Metadata read from the flash will be stored to the location pointed to by the HW_GPMI_AUXILIARY register and data will be written to the address specified in the HW_GPMI_PAYLOAD register. Since the number of blocks on a flash page is programmable, the BCH also writes individual block correction status to the auxiliary pointer at the word-aligned address following the end of the metadata. Optionally, the computed syndromes may also be written to the auxiliary area if the DEBUGSYNDROME bit is set in the control register.

As blocks complete processing, the bus master will accumulate the status for each block and write it to the auxiliary data buffer following the metadata. The metadata area will be padded with 0's until the next word boundary and the status for blocks 0-3 will be written to the next word. Status for subsequent blocks will then be written to the buffer. Status for the first block (metadata block) is also stored in the STATUS_BLK0 register in the BCH_STATUS register. The completion codes for the blocks are indicated in the Table 15-3.



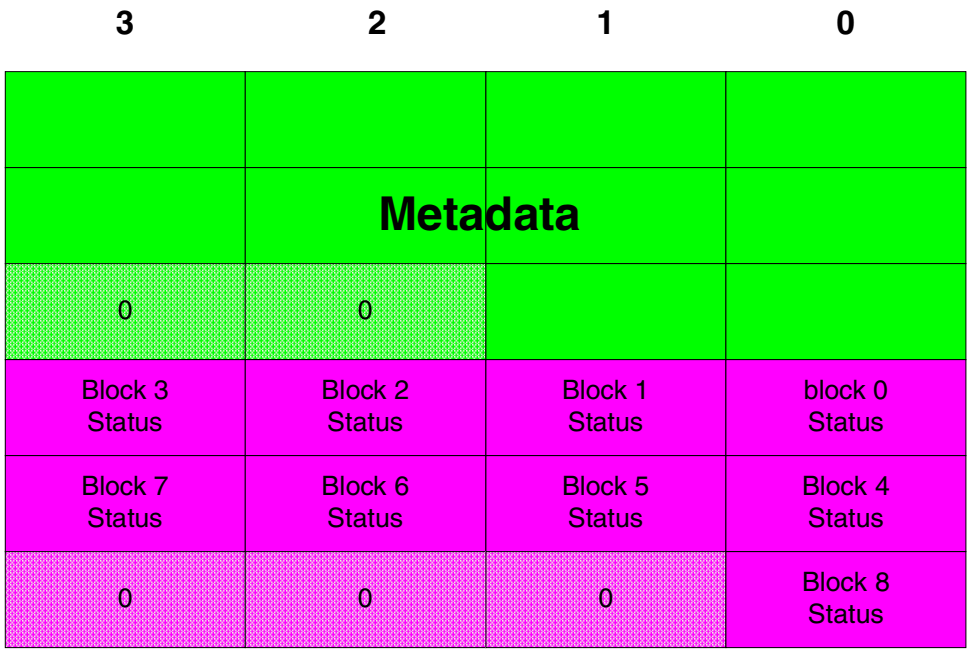
*Computed syndrome area consists of 2*t 13-bit symbols written as 16-bit halfwords.*

Figure 15-4. BCH Data Buffers in Memory

Table 15-3. Status Block Completion Codes

Code	Description
0xFF	Block is erased
0xFE	Block is uncorrectable
0x00	No errors found
0x01-0x14	Number of errors corrected

Figure 15-5 shows the layout of the bytes within the status field.



Status bytes are allocated based on the NBLOCKS programmed into the flash format register. The number of status bytes will be computed by the NBLOCKS+1. The status area will be padded with zeros to the next word boundary.

Syndrome data written for debug purposes will follow the end of the status block.

Figure 15-5. Memory-to-Memory Operations

15.3 Memory to Memory (Loopback) Operation

The BCH supports a memory-to-memory mode of operation where both the encoded and decoded buffers reside in system memory. This can be useful for applications where data must be protected by ECC, but the storage device does not reside on the GPMI bus.

The BCH operation in memory to memory mode is much simpler than in GPMI mode since DMAs are not required to manage the operation. Instead software simply writes the HW_BCH_DATAPTR and HW_BCH_METAPTR with the addresses of the data and metadata (auxiliary) buffers and the HW_BCH_ENCODEPTR with the address of the buffer for encoded data. To initiate the operation, software simply sets the M2M_ENCODE and M2M_ENABLE bits in the control register. The BCH can be programmed to either issue an interrupt at the end of the operation or software may poll the status bits for completion.

Memory to memory decode operations work in a similar manner. The encoded data address is written to the HW_BCH_ENCODEPTR and the data and meta pointers are written to buffers that correspond to the

desired decoded data addresses. To initiate a decode, software must set the M2M_ENCODE bit to 0 while writing the M2M_ENABLE bit.

15.4 Programming the BCH/GPMI Interfaces

Programming the BCH for NAND operations consists largely of disabling the soft reset and clock bits (SFTRST and CLKGATE) from the HW_BCH_CTRL register and then programming the flash layout registers to correspond to the format of the attached NAND device(s). The HW_BCH_LAYOUTSELECT register should also be programmed to map the chip select of each attached device into one of the four layout registers.

The bulk of the programming is actually applied to the GPMI via PIO operations embedded in DMA command structures. The DMA will perform all the requisite handshaking with the GPMI interface to negotiate the address portion of the transfer, then the BCH will handle all the movement of data from memory to the GPMI (writes) or the GPMI to memory (reads). The BCH will direct all data blocks to the buffer pointed to by the PAYLOAD_BUFFER and the metadata will be written to the AUXILIARY_BUFFER. Both of these registers are located in the GPMI PIO data space and are communicated to the BCH hardware at the beginning of the transfer. Thus, the normal multi-NAND DMA based device interleaving is preserved, i.e., four NANDs on four separate chip selects can be scheduled for read or write operations using the BCH. Whichever channel finishes its ready wait first and enters the DMA arbiter with its lock bit set will “own” the GPMI command interface and through it will own the BCH resources for the duration of its processing.

15.4.1 BCH Encoding for NAND Writes

The BCH encoder flowchart in Figure 15-6 shows the detailed steps involved in programming and using the BCH encoder. This flowchart shows how to use the BCH block with the GPMI.

To use the BCH encoder with the GPMI’s DMA, create a DMA command chain containing nine descriptor structures, as shown in Figure 15-8 and detailed in the DMA structure code example that follows it in Section 15.4.1.1, “DMA Structure Code Example.” The nine descriptors perform the following tasks:

1. Disable the BCH block (in case it was enabled) and issue NAND write setup command byte (under “CLE”) and address bytes (under “ALE”).
2. Configure and enable the BCH and GPMI blocks to perform the NAND write.
3. Disable the BCH block and issue NAND write execute command byte (under “CLE”).
4. Wait for the NAND device to finish writing the data by watching the ready signal.
5. Check for NAND timeout via “PSENSE”.
6. Issue NAND status command byte (under “CLE”).
7. Read the status and compare against expected.
8. If status is incorrect/incomplete, branch to error handling descriptor chain.
9. Otherwise, write is complete and emit GPMI interrupt.

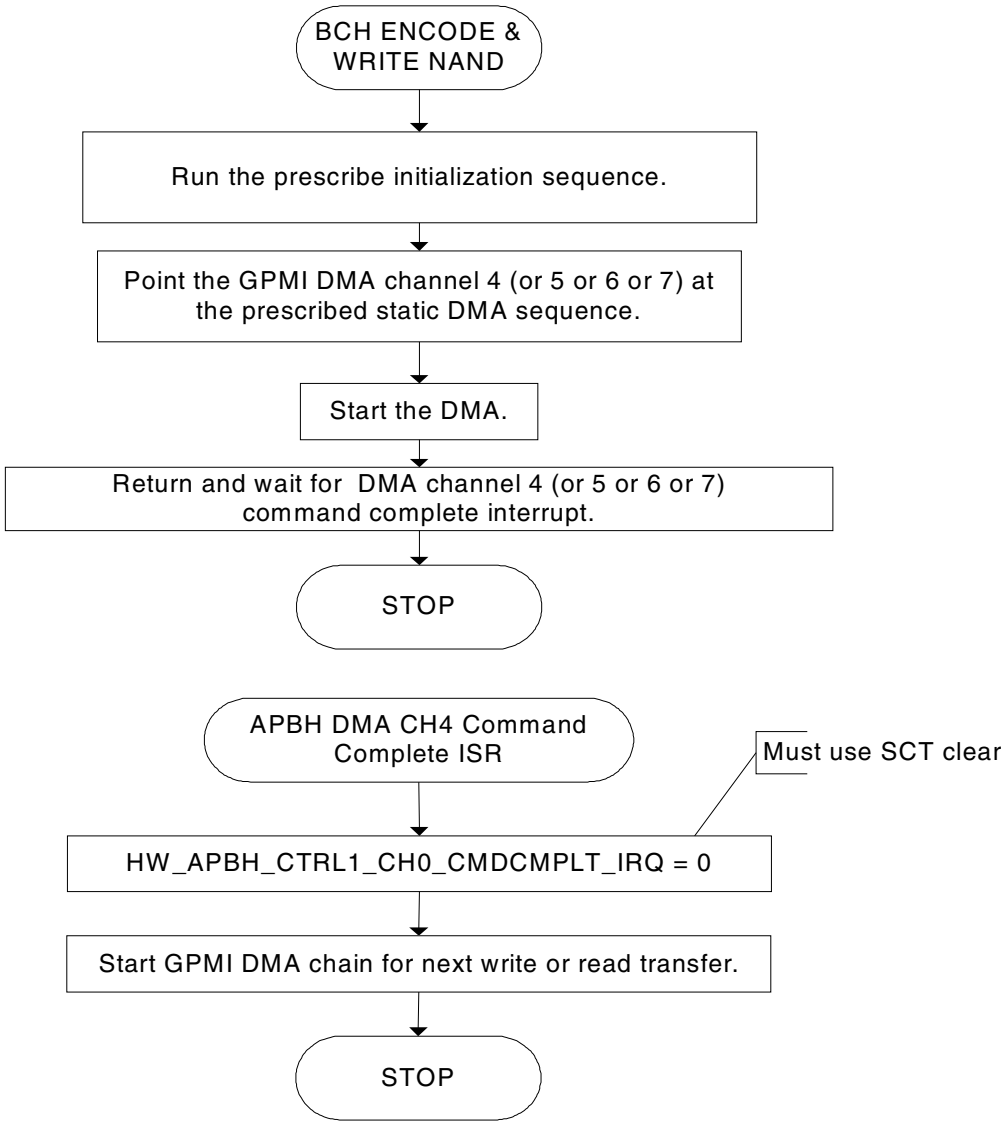


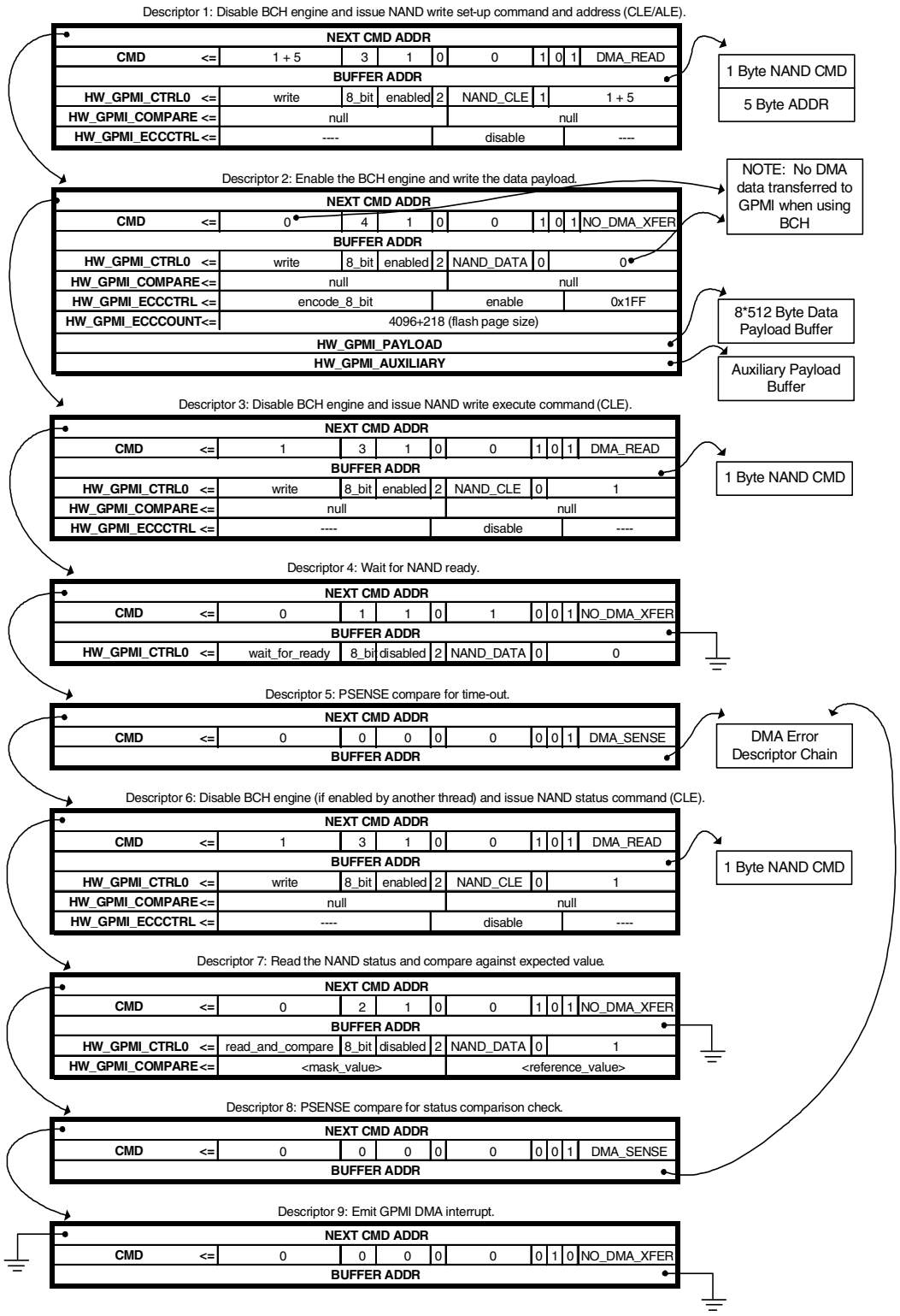
Figure 15-6. BCH Encode Flowchart

Descriptor Legend

NEXT CMD ADDR										
CMD	<=	xfer_count	cmdwords	wait4endcmd	semaphore	nandwait4ready	nandlock	irqoncmplt	chain	command
BUFFER ADDR										
HW_GPMI_CTRL0	<=	command_mode	word_length	lock_cs	CS	address	address_increment	xfer_count		
HW_GPMI_COMPARE	<=	mask				reference				
HW_GPMI_ECCCTRL	<=	ecc_cmd			enable_ecc			buffer_mask		
HW_GPMI_ECCCOUNT										
HW_GPMI_PAYLOAD										
HW_GPMI_AUXILIARY										

Note: Refer to this legend when examining Figure 15-8 and Figure 15-10.

Figure 15-7. BCH DMA Descriptor Legend



Note: To interpret the fields in this diagram, see Figure 15-7 for the descriptor legend.

Figure 15-8. BCH Encode DMA Descriptor Chain

15.4.1.1 DMA Structure Code Example

The following code sample illustrates the coding for one write transaction involving 4096 bytes of data payload (eight 512-byte blocks) and 10 bytes of auxiliary payload (also referred to as *metadata*) to a 4K NAND page sitting on GPMI CS2.

```
//-----------------------------------------------------------------------------
// generic DMA/GPMI/ECC descriptor struct, order sensitive!
//-----------------------------------------------------------------------------
typedef struct {
  // DMA related fields
  unsigned int dma_nxtcmdar;
  unsigned int dma_cmd;
  unsigned int dma_bar;

  // GPMI related fields
  unsigned int gpmi_ctrl0;
  unsigned int gpmi_compare;
  unsigned int gpmi_eccctrl;
  unsigned int gpmi_ecccount;
  unsigned int gpmi_data_ptr;
  unsigned int gpmi_aux_ptr;
} GENERIC_DESCRIPTOR;

//-----------------------------------------------------------------------------
// allocate 9 descriptors for doing a NAND ECC Write
//-----------------------------------------------------------------------------
GENERIC_DESCRIPTOR write[9];

//-----------------------------------------------------------------------------
// DMA descriptor pointer to handle error conditions from psense checks
//-----------------------------------------------------------------------------
unsigned int * dma_error_handler;

//-----------------------------------------------------------------------------
// 8 byte NAND command and address buffer
// any alignment is ok, it is read by the GPMI DMA
//   byte 0 is write setup command
//   bytes 1-5 is the NAND address
//   byte 6 is write execute command
//   byte 7 is status command
//-----------------------------------------------------------------------------
unsigned char nand_cmd_addr_buffer[8];

//-----------------------------------------------------------------------------
// 4096 byte payload buffer used for reads or writes
// needs to be word aligned
//-----------------------------------------------------------------------------
unsigned int write_payload_buffer[(4096/4)];

//-----------------------------------------------------------------------------
// 65 byte meta-data to be written to NAND
// needs to be word aligned
//-----------------------------------------------------------------------------
unsigned int write_aux_buffer[65];

//-----------------------------------------------------------------------------
// Descriptor 1: issue NAND write setup command (CLE/ALE)
//-----------------------------------------------------------------------------
write[0].dma_nxtcmdar = &write[1];                          // point to the next descriptor

write[0].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (1 + 5)|   // 1 byte command, 5 byte address
                   BF_APBH_CHn_CMD_CMDWORDS      (3)    |   // send 3 words to the GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)    |   // wait for command to finish before continuing
                   BF_APBH_CHn_CMD_SEMAPHORE     (0)    |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(0)    |
                   BF_APBH_CHn_CMD_NANDLOCK      (1)    |   // prevent other DMA channels from taking over
                   BF_APBH_CHn_CMD_IRQONCMPLT    (0)    |
                   BF_APBH_CHn_CMD_CHAIN         (1)    |   // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, DMA_READ); // read data from DMA, write to NAND

write[0].dma_bar = &nand_cmd_addr_buffer;                  // byte 0 write setup, bytes 1 - 5 NAND address

// 3 words sent to the GPMI
write[0].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, WRITE)     | // write to the NAND
                      BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)     |
                      BV_FLD(GPMI_CTRL0, LOCK_CS,      ENABLED)   |
                      BF_GPMI_CTRL0_CS                 (2)        | // must correspond to NAND CS used
                      BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_CLE)  |
                      BF_GPMI_CTRL0_ADDRESS_INCREMENT  (1)        | // send command and address
                      BF_GPMI_CTRL0_XFER_COUNT         (1 + 5);   // 1 byte command, 5 byte address

write[0].gpmi_compare = NULL;                               // field not used but necessary to set eccctrl
```

```
write[0].gpmi_eccctrl = BV_FLD(GPMI_ECCCTRL, ENABLE_ECC, DISABLE); // disable the ECC block

//-----------------------------------------------------------------------------
// Descriptor 2: write the data payload (DATA)
//-----------------------------------------------------------------------------
write[1].dma_nxtcmdar = &write[2];                       // point to the next descriptor

write[1].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT      (0)|  // NOTE: No DMA data transfer
                   BF_APBH_CHn_CMD_CMDWORDS        (6)|  // send 6 words to the GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD     (1)|  // Wait to end
                   BF_APBH_CHn_CMD_SEMAPHORE       (0)|
                   BF_APBH_CHn_CMD_NANDWAIT4READY  (0)|
                   BF_APBH_CHn_CMD_NANDLOCK        (1)|  // maintain resource lock
                   BF_APBH_CHn_CMD_IRQONCMPLT      (0)|
                   BF_APBH_CHn_CMD_CHAIN           (1)|  // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, DMA_NO_XFER);  // No data transferred

write[1].dma_bar = &write_payload_buffer;                    // pointer for the 4K byte data area

// 4 words sent to the GPMI
write[1].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, WRITE)     | // write to the NAND
                      BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)     |
                      BV_FLD(GPMI_CTRL0, LOCK_CS,      ENABLED)   |
                      BF_GPMI_CTRL0_CS                 (2)        | // must correspond to NAND CS used
                      BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_DATA)|
                      BF_GPMI_CTRL0_ADDRESS_INCREMENT  (0)        |
                      BF_GPMI_CTRL0_XFER_COUNT         (0);       // NOTE: this field contains
                                                                  //  the total amount
                                                                     // DMA transferred to GPMI via DMA (0)!

write[1].gpmi_compare = NULL;                              // field not used but necessary to set eccctrl

write[1].gpmi_eccctrl = BV_FLD(GPMI_ECCCTRL, ECC_CMD,    ENCODE_8_BIT) | // specify t = 8 mode
                        BV_FLD(GPMI_ECCCTRL, ENABLE_ECC, ENABLE)       | // enable ECC module
                        BF_GPMI_ECCCTRL_BUFFER_MASK   (0x1FF);  // write all 8 data blocks and 1 aux block

write[1].gpmi_ecccount = BF_GPMI_ECCCOUNT_COUNT(4096+218);      // specify number of bytes written to NAND
write[1].gpmi_data_pointer = &write_payload_pointer;            // data buffer address
write[1].gpmi_aux_pointer  = &write_aux_pointer;                // metadata pointer


//-----------------------------------------------------------------------------
// Descriptor 3: issue NAND write execute command (CLE)
//-----------------------------------------------------------------------------
write[2].dma_nxtcmdar = &write[3];                         // point to the next descriptor

write[2].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (1) |     // 1 byte command
                   BF_APBH_CHn_CMD_CMDWORDS      (3) |     // send 3 words to the GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (1) |     // wait for command to finish before continuing
                   BF_APBH_CHn_CMD_SEMAPHORE     (0) |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(0) |
                   BF_APBH_CHn_CMD_NANDLOCK      (1) |     // maintain resource lock
                   BF_APBH_CHn_CMD_IRQONCMPLT    (0) |
                   BF_APBH_CHn_CMD_CHAIN         (1) |     // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, DMA_READ);   // read data from DMA, write to NAND

write[2].dma_bar = &nand_cmd_addr_buffer[6];                  // point to byte 6, write execute command

// 3 words sent to the GPMI
write[2].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, WRITE)     | // write to the NAND
                      BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)     |
                      BV_FLD(GPMI_CTRL0, LOCK_CS,      ENABLED)   |
                      BF_GPMI_CTRL0_CS                 (2)        | // must correspond to NAND CS used
                      BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_CLE)  |
                      BF_GPMI_CTRL0_ADDRESS_INCREMENT  (0)        |
                      BF_GPMI_CTRL0_XFER_COUNT         (1);       // 1 byte command

write[2].gpmi_compare = NULL;                              // field not used but necessary to set eccctrl

write[2].gpmi_eccctrl = BV_FLD(GPMI_ECCCTRL, ENABLE_ECC, DISABLE); // disable the ECC block

//-----------------------------------------------------------------------------
// Descriptor 4: wait for ready (CLE)
//-----------------------------------------------------------------------------
write[3].dma_nxtcmdar = &write[4];                       // point to the next descriptor

write[3].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0) |   // no dma transfer
                   BF_APBH_CHn_CMD_CMDWORDS      (1) |   // send 1 word to the GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (1) |   // wait for command to finish before continuing
                   BF_APBH_CHn_CMD_SEMAPHORE     (0) |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(1) |   // wait for nand to be ready
                   BF_APBH_CHn_CMD_NANDLOCK      (0) |   // relinquish nand lock
                   BF_APBH_CHn_CMD_IRQONCMPLT    (0) |
                   BF_APBH_CHn_CMD_CHAIN         (1) |   // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, NO_DMA_XFER);          // no dma transfer

write[3].dma_bar = NULL;                                                // field not used

// 1 word sent to the GPMI
```

```
write[3].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, WAIT_FOR_READY) | // wait for NAND ready
                      BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)          |
                      BV_FLD(GPMI_CTRL0, LOCK_CS,      DISABLED)       |
                      BF_GPMI_CTRL0_CS                 (2)             | // must correspond to NAND CS used
                      BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_DATA)      |
                      BF_GPMI_CTRL0_ADDRESS_INCREMENT  (0)             |
                      BF_GPMI_CTRL0_XFER_COUNT         (0);

//--------------------------------------------------------------------------
// Descriptor 5: psense compare (time out check)
//--------------------------------------------------------------------------
write[4].dma_nxtcmdar = &write[5];                              // point to the next descriptor

write[4].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0)         | // no dma transfer
                   BF_APBH_CHn_CMD_CMDWORDS      (0)         | // no words sent to GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (0)         | // do not wait to continue
                   BF_APBH_CHn_CMD_SEMAPHORE     (0)         |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(0)         |
                   BF_APBH_CHn_CMD_NANDLOCK      (0)         |
                   BF_APBH_CHn_CMD_IRQONCMPLT    (0)         |
                   BF_APBH_CHn_CMD_CHAIN         (1)         | // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, DMA_SENSE);  // perform a sense check

write[4].dma_bar = dma_error_handler;                     // if sense check fails, branch to error handler

//--------------------------------------------------------------------------
// Descriptor 6: issue NAND status command (CLE)
//--------------------------------------------------------------------------
write[5].dma_nxtcmdar = &write[6];                           // point to the next descriptor

write[5].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (1)      |  // 1 byte command
                   BF_APBH_CHn_CMD_CMDWORDS      (3)      |  // send 3 words to the GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)      |  // wait for command to finish before continuing
                   BF_APBH_CHn_CMD_SEMAPHORE     (0)      |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(0)      |
                   BF_APBH_CHn_CMD_NANDLOCK      (1)      |  // prevent other DMA channels from taking over
                   BF_APBH_CHn_CMD_IRQONCMPLT    (0)      |
                   BF_APBH_CHn_CMD_CHAIN         (1)      |  // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, DMA_READ);  // read data from DMA, write to NAND

write[5].dma_bar = &nand_cmd_addr_buffer[7];                // point to byte 7, status command

write[5].gpmi_compare = NULL;                                 // field not used but necessary to set eccctrl

write[5].gpmi_eccctrl = BV_FLD(GPMI_ECCCTRL, ENABLE_ECC, DISABLE); // disable the ECC block

// 3 words sent to the GPMI
write[5].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, WRITE)     | // write to the NAND
                      BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)     |
                      BV_FLD(GPMI_CTRL0, LOCK_CS,      ENABLED)   |
                      BF_GPMI_CTRL0_CS                 (2)        | // must correspond to NAND CS used
                      BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_CLE)  |
                      BF_GPMI_CTRL0_ADDRESS_INCREMENT  (0)        |
                      BF_GPMI_CTRL0_XFER_COUNT         (1);         // 1 byte command

//--------------------------------------------------------------------------
// Descriptor 7: read status and compare (DATA)
//--------------------------------------------------------------------------
write[6].dma_nxtcmdar = &write[7];                           // point to the next descriptor

write[6].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0)     |   // no dma transfer
                   BF_APBH_CHn_CMD_CMDWORDS      (2)     |   // send 2 words to the GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)     |   // wait for command to finish before continuing
                   BF_APBH_CHn_CMD_SEMAPHORE     (0)     |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(0)     |
                   BF_APBH_CHn_CMD_NANDLOCK      (1)     |   // maintain resource lock
                   BF_APBH_CHn_CMD_IRQONCMPLT    (0)     |
                   BF_APBH_CHn_CMD_CHAIN         (1)     |   // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, NO_DMA_XFER);  // no dma transfer

write[6].dma_bar = NULL;                                      // field not used

// 2 word sent to the GPMI
write[6].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, READ_AND_COMPARE) | // read from the NAND and
//                                                                         // compare to expect
                      BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)     |
                      BV_FLD(GPMI_CTRL0, LOCK_CS,      DISABLED)  |
                      BF_GPMI_CTRL0_CS                 (2)        |    // must correspond to NAND CS used
                      BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_DATA) |
                      BF_GPMI_CTRL0_ADDRESS_INCREMENT  (0)        |
                      BF_GPMI_CTRL0_XFER_COUNT         (1);

write[6].gpmi_compare = <MASK_AND_REFERENCE_VALUE>;           // NOTE: mask and reference values are NAND
                                                              // SPECIFIC to evaluate the NAND status

//--------------------------------------------------------------------------
// Descriptor 8: psense compare (time out check)
//--------------------------------------------------------------------------
write[7].dma_nxtcmdar = &write[8];                           // point to the next descriptor
```

```
write[7].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0)          | // no dma transfer
                   BF_APBH_CHn_CMD_CMDWORDS      (0)          | // no words sent to GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (0)          | // do not wait to continue
                   BF_APBH_CHn_CMD_SEMAPHORE     (0)          |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(0)          |
                   BF_APBH_CHn_CMD_NANDLOCK      (0)          | // relinquish nand lock
                   BF_APBH_CHn_CMD_IRQONCMPLT    (0)          |
                   BF_APBH_CHn_CMD_CHAIN         (1)          | // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, DMA_SENSE);  // perform a sense check

write[7].dma_bar = dma_error_handler;                   // if sense check fails, branch to error handler

//-----------------------------------------------------------------------------
// Descriptor 9: emit GPMI interrupt
//-----------------------------------------------------------------------------
write[8].dma_nxtcmdar = NULL;                                  // not used since this is last descriptor

write[8].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0)           | // no dma transfer
                   BF_APBH_CHn_CMD_CMDWORDS      (0)           | // no words sent to GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (0)           | // do not wait to continue
                   BF_APBH_CHn_CMD_SEMAPHORE     (0)           |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(0)           |
                   BF_APBH_CHn_CMD_NANDLOCK      (0)           |
                   BF_APBH_CHn_CMD_IRQONCMPLT    (1)           | // emit GPMI interrupt
                   BF_APBH_CHn_CMD_CHAIN         (0)           | // terminate DMA chain processing
                   BV_FLD(APBH_CHn_CMD, COMMAND, NO_DMA_XFER);  // no dma transfer
```

15.4.1.2 Using the BCH Encoder

To use the BCH encoder, first turn off the module-wide soft reset bit in both the GPMI and BCH blocks before starting any DMA activity. Note that turning off the soft reset must take place by itself, prior to programming the rest of the control registers. Turn off the BCH bus master soft reset bit (bit 29). Turn off the clock gate bits.

Program the remainder of the GPMI, BCH and APBH DMA as follows:

```
// bring APBH out of reset
HW_APBH_CTRL0_CLR(BM_APBH_CTRL0_SFRST);
HW_APBH_CTRL0_CLR(BM_APBH_CTRL0_CLKGATE);

// bring BCH out of reset
HW_BCH_CTRL_CLR(BM_BCH_CTRL_SFTRST);
HW_BCH_CTRL_CLR(BM_BCH_CTRL_CLKGATE);

// bring gpmi out of reset
HW_GPMI_CTRL0_CLR(BM_GPMI_CTRL0_SFTRST);
HW_GPMI_CTRL0_CLR(BM_GPMI_CTRL0_CLKGATE);
HW_GPMI_CTRL1_SET(BM_GPMI_CTRL1_DEV_RESET | // deassert reset
                  BM_GPMI_CTRL1_BCH_MODE  );// enable BCH mode

// enable pinctrl
HW_PINCTRL_CTRL_WR(0x00000000);

// enable GPMI through alt pin wiring
HW_PINCTRL_MUXSEL0_CLR(0xff000000);
HW_PINCTRL_MUXSEL0_SET(0xaa000000);

// to use the primary pins do the following
//    HW_PINCTRL_MUXSEL4_CLR(0xff000000);
//    HW_PINCTRL_MUXSEL4_SET(0x55000000);

// enable gpmi pins
HW_PINCTRL_MUXSEL0_CLR(0x0000ffff);// data bits
HW_PINCTRL_MUXSEL1_CLR(0x000fffff);// control bits
```

Note that for writing NANDs (ECC encoding), only GPMI DMA command complete interrupts are used. The BCH engine is used for writing to the NAND but may optionally produce an interrupt. From the sample code in Section 15.4.1.1, “DMA Structure Code Example”:

- DMA descriptor 1 prepares the NAND for data write by using the GPMI to issue a write setup command byte under “CLE”, then sends a 5-byte address under “ALE”. The BCH engine is disabled and not used for these commands.
- DMA descriptor 2 enables the BCH engine for encoding to begin the initial writing of the NAND data by specifying where the data and auxiliary payload are coming from in system memory.
- DMA descriptor 3 issues the write commit command byte under “CLE” to the NAND.
- DMA descriptor 4 waits for the NAND to complete the write commit/transfer by watching the NAND’s ready line status. This descriptor relinquishes the NANDLOCK on the GPMI to enable the other DMA channels to initiate NAND transactions on different NAND CS lines.
- DMA descriptor 6 issues a NAND status command byte under “CLE” to check the status of the NAND device following the page write.
- DMA descriptor 7 reads back the NAND status and compares the status with an expected value. If there are differences, then the DMA processing engine follows an error-handling DMA descriptor path.
- DMA descriptor 8 disables the BCH engine and emits a GPMI interrupt to indicate that the NAND write has been completed.

15.4.2 BCH Decoding for NAND Reads

When a page is read from NAND flash, BCH syndromes will be computed and, if correctable errors are found, they will be corrected on a per block basis within the NAND page. This decoding process is fully overlapped with other NAND data reads and with CPU execution. The BCH decoder flowchart in Figure 15-9 shows the steps involved in programming the decoder. The hardware flow of reading and decoding a 4096-byte page is shown in Figure 15-10.

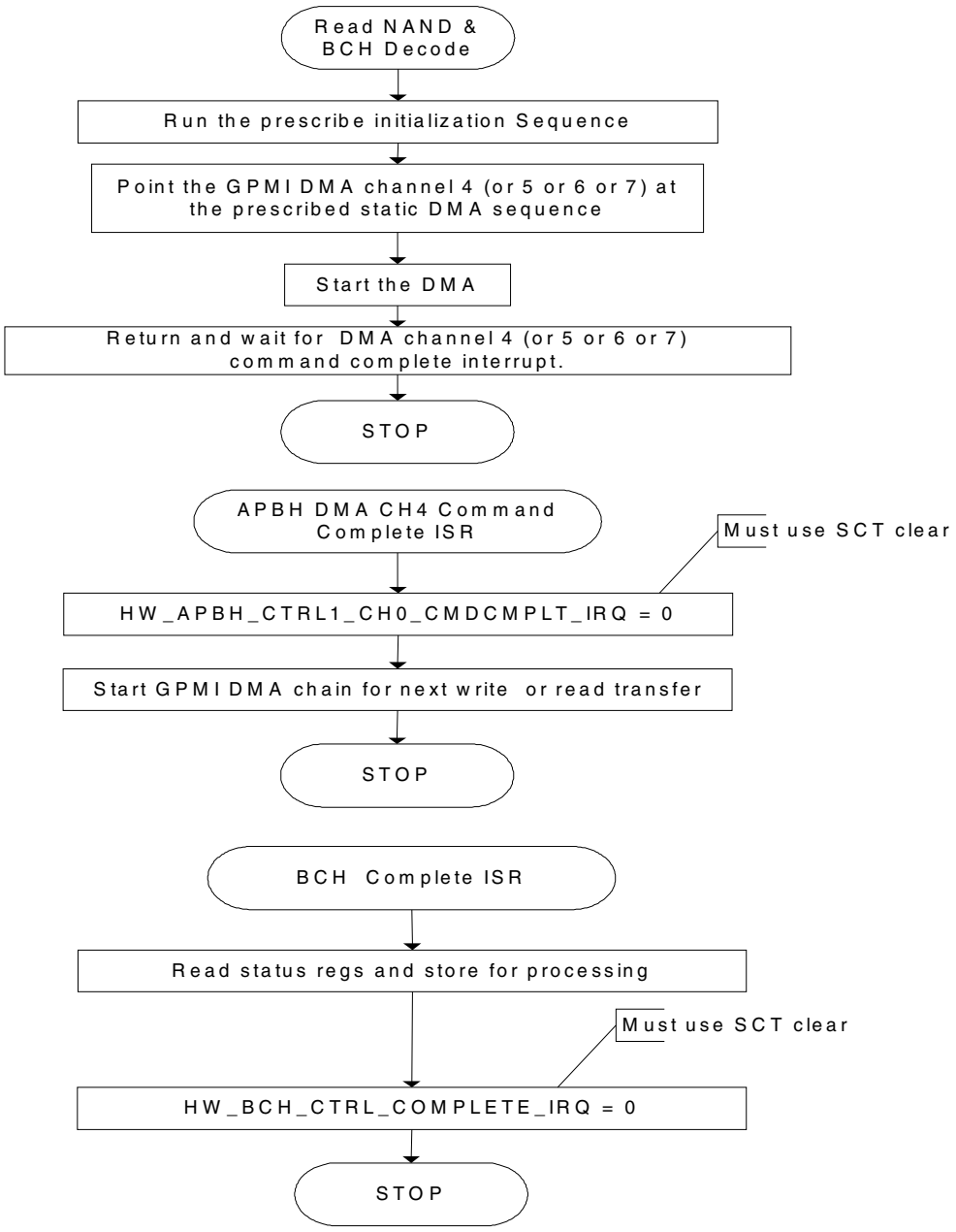


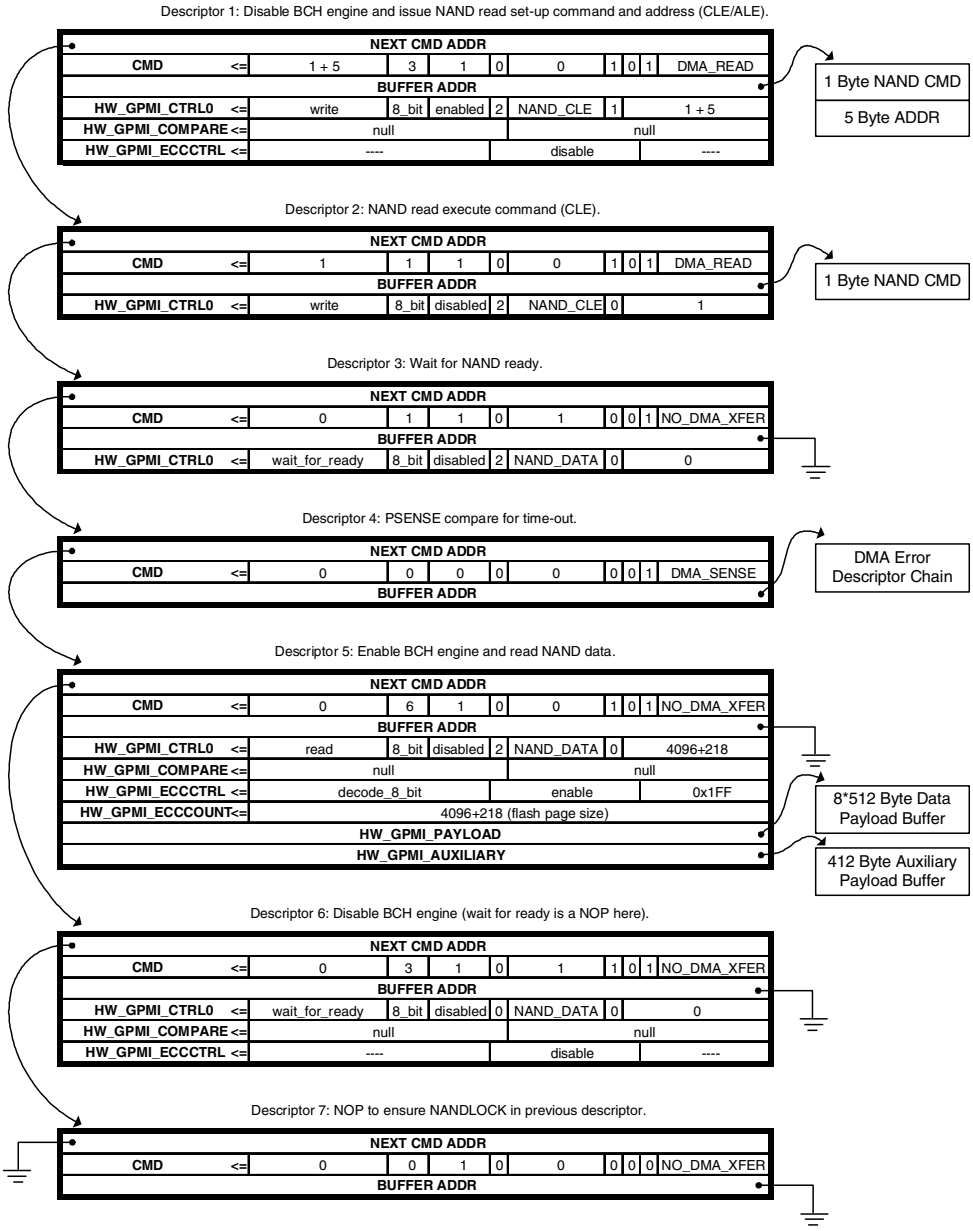
Figure 15-9. BCH Decode Flowchart

Conceptually, an APBH DMA Channel 4, 5, 6, or 7 command chain with seven command structures linked together is used to perform the BCH decode operation (as shown in Figure 15-10). NOTE: The GPMI's DMA command structures controls the BCH decode operation.

To use the BCH decoder with the GPMI's DMA, create a DMA command chain containing seven descriptor structures, as shown in Figure 15-10 and detailed in the DMA structure code example that fol-

lows it in Section 15.4.2.1, “DMA Structure Code Example.” The seven DMA descriptors perform the following tasks:

1. Issue NAND read setup command byte (under “CLE”) and address bytes (under “ALE”).
2. Issue NAND read execute command byte (under “CLE”).
3. Wait for the NAND device to complete accessing the block data by watching the ready signal.
4. Check for NAND timeout via “PSENSE”.
5. Configure and enable the BCH block and read the NAND block data.
6. Disable the BCH block.
7. Descriptor NOP to allow NANDLOCK in the previous descriptor to the thread-safe.



Note: To interpret the fields in this diagram, see Figure 15-7 for the descriptor legend.

Figure 15-10. BCH Decode DMA Descriptor Chain

15.4.2.1 DMA Structure Code Example

The following sample code illustrates the coding for one read transaction, consisting of a seven DMA command structure chain for reading all 4096 bytes of payload data (eight 512-byte blocks) and 65 bytes of metadata with the associative parity bytes
(8 * (18) + 9) from a 4K NAND page sitting on GPMI CS2.

```
//-----------------------------------------------------------------------------
// generic DMA/GPMI/ECC descriptor struct, order sensitive!
//-----------------------------------------------------------------------------
typedef struct {
  // DMA related fields
  unsigned int dma_nxtcmdar;
  unsigned int dma_cmd;
  unsigned int dma_bar;

  // GPMI related fields
  unsigned int gpmi_ctrl0;
  unsigned int gpmi_compare;
  unsigned int gpmi_eccctrl;
  unsigned int gpmi_ecccount;
  unsigned int gpmi_data_ptr;
  unsigned int gpmi_aux_ptr;
} GENERIC_DESCRIPTOR;

//-----------------------------------------------------------------------------
// allocate 7 descriptors for doing a NAND ECC Read
//-----------------------------------------------------------------------------
GENERIC_DESCRIPTOR read[7];

//-----------------------------------------------------------------------------
// DMA descriptor pointer to handle error conditions from psense checks
//-----------------------------------------------------------------------------
unsigned int * dma_error_handler;

//-----------------------------------------------------------------------------
// 7 byte NAND command and address buffer
// any alignment is ok, it is read by the GPMI DMA
//   byte 0 is read setup command
//   bytes 1-5 is the NAND address
//   byte 6 is read execute command
//-----------------------------------------------------------------------------
unsigned char nand_cmd_addr_buffer[7];

//-----------------------------------------------------------------------------
// 4096 byte payload buffer used for reads or writes
// needs to be word aligned
//-----------------------------------------------------------------------------
unsigned int read_payload_buffer[(4096/4)];

//-----------------------------------------------------------------------------
// 412 byte auxiliary buffer used for reads
// needs to be word aligned
//-----------------------------------------------------------------------------
unsigned int read_aux_buffer[(412/4)];

//-----------------------------------------------------------------------------
// Descriptor 1: issue NAND read setup command (CLE/ALE)
//-----------------------------------------------------------------------------
read[0].dma_nxtcmdar = &read[1];                              // point to the next descriptor

read[0].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (1 + 5) |   // 1 byte command, 5 byte address
                  BF_APBH_CHn_CMD_CMDWORDS      (3)     |   // send 3 words to the GPMI
                  BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)     |   // wait for command to finish before continuing
                  BF_APBH_CHn_CMD_SEMAPHORE     (0)     |
                  BF_APBH_CHn_CMD_NANDWAIT4READY(0)     |
                  BF_APBH_CHn_CMD_NANDLOCK      (1)     |   // prevent other DMA channels from taking over
                  BF_APBH_CHn_CMD_IRQONCMPLT    (0)     |
                  BF_APBH_CHn_CMD_CHAIN         (1)     |    // follow chain to next command
                  BV_FLD(APBH_CHn_CMD, COMMAND, DMA_READ);  // read data from DMA, write to NAND

read[0].dma_bar = &nand_cmd_addr_buffer;                     // byte 0 read setup, bytes 1 - 5 NAND address

// 3 words sent to the GPMI
read[0].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, WRITE)     | // write to the NAND
                     BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)     |
                     BV_FLD(GPMI_CTRL0, LOCK_CS,      ENABLED)   |
                     BF_GPMI_CTRL0_CS                 (2)        | // must correspond to NAND CS used
                     BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_CLE)  |
                     BF_GPMI_CTRL0_ADDRESS_INCREMENT  (1)        | // send command and address
                     BF_GPMI_CTRL0_XFER_COUNT         (1 + 5);   // 1 byte command, 5 byte address

read[0].gpmi_compare = NULL;                                  // field not used but necessary to set eccctrl

read[0].gpmi_eccctrl = BV_FLD(GPMI_ECCCTRL, ENABLE_ECC, DISABLE); // disable the ECC block

//-----------------------------------------------------------------------------
// Descriptor 2: issue NAND read execute command (CLE)
//-----------------------------------------------------------------------------
read[1].dma_nxtcmdar = &read[2];                              // point to the next descriptor

read[1].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (1)     |   // 1 byte read command
                  BF_APBH_CHn_CMD_CMDWORDS      (1)     |   // send 1 word to GPMI
                  BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)     |   // wait for command to finish before continuing
                  BF_APBH_CHn_CMD_SEMAPHORE     (0)     |
                  BF_APBH_CHn_CMD_NANDWAIT4READY(0)     |
```

```
                BF_APBH_CHn_CMD_NANDLOCK      (1)     |   // prevent other DMA channels from taking over
                BF_APBH_CHn_CMD_IRQONCMPLT    (0)     |
                BF_APBH_CHn_CMD_CHAIN         (1)     |   // follow chain to next command
                BV_FLD(APBH_CHn_CMD, COMMAND, DMA_READ);     // read data from DMA, write to NAND

read[1].dma_bar = &nand_cmd_addr_buffer[6];                  // point to byte 6, read execute command

// 1 word sent to the GPMI
read[1].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, WRITE)     | // write to the NAND
                     BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)     |
                     BV_FLD(GPMI_CTRL0, LOCK_CS,      DISABLED)  |
                     BF_GPMI_CTRL0_CS                 (2)        | // must correspond to NAND CS used
                     BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_CLE)  |
                     BF_GPMI_CTRL0_ADDRESS_INCREMENT  (0)        |
                     BF_GPMI_CTRL0_XFER_COUNT         (1);       // 1 byte command

//--------------------------------------------------------------------------
// Descriptor 3: wait for ready (DATA)
//--------------------------------------------------------------------------
read[2].dma_nxtcmdar = &read[3];                             // point to the next descriptor

read[2].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0)       | // no dma transfer
                  BF_APBH_CHn_CMD_CMDWORDS      (1)       | // send 1 word to GPMI
                  BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)       | // wait for command to finish before continuing
                  BF_APBH_CHn_CMD_SEMAPHORE     (0)       |
                  BF_APBH_CHn_CMD_NANDWAIT4READY(1)       | // wait for nand to be ready
                  BF_APBH_CHn_CMD_NANDLOCK      (0)       | // relinquish nand lock
                  BF_APBH_CHn_CMD_IRQONCMPLT    (0)       |
                  BF_APBH_CHn_CMD_CHAIN         (1)       | // follow chain to next command
                  BV_FLD(APBH_CHn_CMD, COMMAND, NO_DMA_XFER);   // no dma transfer

read[2].dma_bar = NULL;                                       // field not used

// 1 word sent to the GPMI
read[2].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, WAIT_FOR_READY) | // wait for NAND ready
                     BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)          |
                     BV_FLD(GPMI_CTRL0, LOCK_CS,      DISABLED)       |
                     BF_GPMI_CTRL0_CS                 (2)             | // must correspond to NAND CS used
                     BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_DATA)      |
                     BF_GPMI_CTRL0_ADDRESS_INCREMENT  (0)             |
                     BF_GPMI_CTRL0_XFER_COUNT         (0);

//--------------------------------------------------------------------------
// Descriptor 4: psense compare (time out check)
//--------------------------------------------------------------------------
read[3].dma_nxtcmdar = &read[4];                                 // point to the next descriptor

read[3].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0)         |    // no dma transfer
                  BF_APBH_CHn_CMD_CMDWORDS      (0)         |    // no words sent to GPMI
                  BF_APBH_CHn_CMD_WAIT4ENDCMD   (0)         |    // do not wait to continue
                  BF_APBH_CHn_CMD_SEMAPHORE     (0)         |
                  BF_APBH_CHn_CMD_NANDWAIT4READY(0)         |
                  BF_APBH_CHn_CMD_NANDLOCK      (0)         |
                  BF_APBH_CHn_CMD_IRQONCMPLT    (0)         |
                  BF_APBH_CHn_CMD_CHAIN         (1)         |    // follow chain to next command
                  BV_FLD(APBH_CHn_CMD, COMMAND, DMA_SENSE);      // perform a sense check

read[3].dma_bar = dma_error_handler;                        // if sense check fails, branch to error handler

//--------------------------------------------------------------------------
// Descriptor 5: read 4K page plus 65 byte meta-data Nand data
//               and send it to ECC block (DATA)
//--------------------------------------------------------------------------
read[4].dma_nxtcmdar = &read[5];                           // point to the next descriptor

read[4].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0)     | // no dma transfer
                  BF_APBH_CHn_CMD_CMDWORDS      (6)     | // send 6 words to GPMI
                  BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)     | // wait for command to finish before continuing
                  BF_APBH_CHn_CMD_SEMAPHORE     (0)     |
                  BF_APBH_CHn_CMD_NANDWAIT4READY(0)     |
                  BF_APBH_CHn_CMD_NANDLOCK      (1)     | // prevent other DMA channels from taking over
                  BF_APBH_CHn_CMD_IRQONCMPLT    (0)     | // ECC block generates BCH interrupt on completion
                  BF_APBH_CHn_CMD_CHAIN         (1)     | // follow chain to next command
                  BV_FLD(APBH_CHn_CMD, COMMAND, NO_DMA_XFER);     // no DMA transfer,
                                                                  // ECC block handles transfer

read[4].dma_bar = NULL;                                           // field not used

// 6 words sent to the GPMI
read[4].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, READ)        |  // read from the NAND
                     BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)       |
                     BV_FLD(GPMI_CTRL0, LOCK_CS,      DISABLED)    |
                     BF_GPMI_CTRL0_CS                 (2)          |  // must correspond to NAND CS used
                     BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_DATA)   |
                     BF_GPMI_CTRL0_ADDRESS_INCREMENT  (0)          |
                     BF_GPMI_CTRL0_XFER_COUNT         (4096+218);     // eight 512 byte data blocks
                                                                      // metadata, and parity


read[4].gpmi_compare = NULL;                                  // field not used but necessary to set eccctrl
```

```
// GPMI ECCCTRL PIO This launches the 4K byte transfer through BCH's
// bus master. Setting the ECC_ENABLE bit redirects the data flow
// within the GPMI so that read data flows to the BCH engine instead
// of flowing to the GPMI's DMA channel.
read[4].gpmi_eccctrl = BV_FLD(GPMI_ECCCTRL, ECC_CMD,    DECODE_8_BIT) |    // specify t = 8 mode
                       BV_FLD(GPMI_ECCCTRL, ENABLE_ECC, ENABLE)       |    // enable ECC module
                       BF_GPMI_ECCCTRL_BUFFER_MASK     (0X1FF);  // read all 8 data blocks and 1 aux block

read[4].gpmi_ecccount = BF_GPMI_ECCCOUNT_COUNT(4096+218);        // specify number of bytes read from NAND

read[4].gpmi_data_ptr = &read_payload_buffer;                    // pointer for the 4K byte data area

read[4].gpmi_aux_ptr = &read_aux_buffer;                         // pointer for the 65 byte aux area +
                                                                 // parity and syndrome bytes for both
                                                                 // data and aux blocks.

//------------------------------------------------------------------------
// Descriptor 6: disable ECC block
//------------------------------------------------------------------------
read[5].dma_nxtcmdar = &read[6];                                 // point to the next descriptor

read[5].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0)       |      // no dma transfer
                  BF_APBH_CHn_CMD_CMDWORDS      (3)       |      // send 3 words to GPMI
                  BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)       |      // wait for command to finish before continuing
                  BF_APBH_CHn_CMD_SEMAPHORE     (0)       |
                  BF_APBH_CHn_CMD_NANDWAIT4READY(1)       |      // wait for nand to be ready
                  BF_APBH_CHn_CMD_NANDLOCK      (1)       |      // need nand lock to be
                                                                 // thread safe while turn-off BCH
                  BF_APBH_CHn_CMD_IRQONCMPLT    (0)       |
                  BF_APBH_CHn_CMD_CHAIN         (1)       |      // follow chain to next command
                  BV_FLD(APBH_CHn_CMD, COMMAND, NO_DMA_XFER);      // no dma transfer

read[5].dma_bar = NULL;                                          // field not used

// 3 words sent to the GPMI
read[5].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, READ)      |
                     BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)     |
                     BV_FLD(GPMI_CTRL0, LOCK_CS,      DISABLED)  |
                     BF_GPMI_CTRL0_CS                 (2)        |      // must correspond to NAND CS used
                     BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_DATA) |
                     BF_GPMI_CTRL0_ADDRESS_INCREMENT  (0)        |
                     BF_GPMI_CTRL0_XFER_COUNT         (0);

read[5].gpmi_compare = NULL;                                     // field not used but necessary to set eccctrl

read[5].gpmi_eccctrl = BV_FLD(GPMI_ECCCTRL, ENABLE_ECC, DISABLE);  // disable the ECC block

//------------------------------------------------------------------------
// Descriptor 7: deassert nand lock
//------------------------------------------------------------------------
read[6].dma_nxtcmdar = NULL;                                     // not used since this is last descriptor

read[6].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (0)       |      // no dma transfer
                  BF_APBH_CHn_CMD_CMDWORDS      (0)       |      // no words sent to GPMI
                  BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)       |      // wait for command to finish before continuing
                  BF_APBH_CHn_CMD_SEMAPHORE     (0)       |
                  BF_APBH_CHn_CMD_NANDWAIT4READY(0)       |
                  BF_APBH_CHn_CMD_NANDLOCK      (0)       |      // relinquish nand lock
                  BF_APBH_CHn_CMD_IRQONCMPLT    (0)       |      // BCH engine generates interrupt
                  BF_APBH_CHn_CMD_CHAIN         (0)       |      // terminate DMA chain processing
                  BV_FLD(APBH_CHn_CMD, COMMAND, NO_DMA_XFER);      // no dma transfer

read[6].dma_bar = NULL;                                          // field not used
```

15.4.2.2 Using the Decoder

As illustrated in Figure 15-10 and the sample code in Section 15.4.2.1, "DMA Structure Code Example":

- DMA descriptor 1 prepares the NAND for data read by using the GPMI to issue a NAND read setup command byte under "CLE", then sends a 5-byte address under "ALE". The BCH engine is not used for these commands.
- DMA descriptor 2 issues a one-byte read execute command to the NAND device that triggers its read access. The NAND then goes not ready.
- DMA descriptor 3 performs a wait for ready operation allowing the DMA chain to remain dormant until the NAND device completes its read access time.

- DMA descriptor 5 handles the reading and error correction of the NAND data. This command's PIOs activate the BCH engine to write the read NAND data to system memory and to process it for any errors that need to be corrected. This DMA descriptor contains two PIO values that are system memory addresses pointing to the PAYLOAD data area and to the AUXILIARY data area. These addresses are used by the BCH engine's AHB master to move data into system memory and to correct it. While this example is reading an entire 4K page—payload plus metadata—it is equally possible to read just one 512-byte payload block or just the uniquely protected metadata block in a single 7 DMA structure transfer.
- DMA descriptor 6 disables the BCH engine with the NANDLOCK asserted. This is necessary to ensure that the GPMI resource is not arbitrated to another DMA channel when multiple DMA channels are active concurrently.
- DMA descriptor 7 deasserts the NANDLOCK to free up the GPMI resource to another channel.

As the BCH block receives data from the GPMI:

- The decoder transforms the read NAND data block into an BCH code word and computes the codeword syndrome.
- If no errors are present, then he BCH block can immediately report back to firmware. This report is passed as the HW_BCH_CTRL_COMPLETE_IRQ interrupt status bit and the associated status bits in the HW_BCH_STATUS0 register.
- If an error is present, then the BCH block corrects the necessary data block or parity block bytes, if possible (not all errors are correctable).

As the BCH decoder reads the data and parity blocks, it records a special condition, i.e., that all of the bits of a payload data block or metadata block are one, including any associated parity bytes. The "all-ones" case for both parity and data indicates an erased block in the NAND device. The erased block detection mechanism can be tuned to allow a maximum number of bit errors on an erased page (i.e. zero bits). Write the HW_BCH_MODE_ERASE_THRESHOLD field to a non-zero setting to allow a certain number of zero bits to be read within a block and still consider the block "erased".

The HW_BCH_STATUS0 register contains a 4-bit field that indicates the final status of the auxiliary block. A value of 0x0 indicates no errors found for a block.

- A value of 1 to 20 inclusive indicates that many correctable errors were found and fixed.
- A value of 0xFE indicates uncorrectable errors detected on the block.
- A value of 0xFF indicates that the block was in the special ALL ONES state and is therefore considered to be an ERASED block.
- All other values are disallowed by the hardware design.

Recall that up to four NAND devices can have DMA chains in-flight at once, i.e. they can all be contending for access to the GPMI data bus. It is impossible to predict which NAND device will enter the BCH engine with a transfer first, because each chain includes a wait4ready command structure. As a result, firmware should look at the HW_BCH_STATUS0_COMPLETED_CE bit field to determine which block is being reported in the status register. There is also a 16 bit HANDLE field in the

HW_GPMI_ECCCTRL register that is passed down the pipeline with each transaction. This handle field can be used to speed firmware's detection of which transaction is being reported.

These examples of reading and writing have focused on full page transfers of 4K page NAND devices. Other device configurations can be specified by changing the ECCOUNT field in the GPMI registers and reprogramming the BCH's HW_FLASHnLAYOUTm registers.

The BCH and GPMI blocks are designed to be very efficient at reading single 512-byte pages in one transaction. With no errors, the transaction takes less than 20 HCLKs longer than the time to read the raw data from the NAND.

To summarize, the APBH DMA command chain for a BCH decode operation is shown in Figure 15-10. Seven DMA command structures must be present for each NAND read transaction decoded by the BCH. The seven DMA command structures for multiple NAND read transaction blocks can be chained together to make larger units of work for the BCH, and each will produce an appropriate error report in the BCH PIO space. Multiple NAND devices can have such multiple chains scheduled. The results can come back out of order with respect to the multiple chains.

15.4.3 Interrupts

There are two interrupt sources used in processing BCH protected NAND read and write transfers. Since all BCH operations are initiated by GPMI DMA command structures, the DMA completion interrupt for the GPMI is an important ISR. Both of the flow charts of Section Figure 15-6., "BCH Encode Flow-chart," and Figure 15-9 show the GPMI DMA complete ISR skeleton. In both reads and writes, the GPMI DMA completion interrupt is used to schedule work ***INTO*** the error correction pipeline. As the front end processing completes, the DMA interrupt is generated and additional work, i.e. DMA chains, are passed to the GPMI DMA to keep it "*fed*". For write operations, this is the only interrupt that will be generated for processing the NAND write transfer.

For reads, however, two interrupts are needed. Every read is started by a GPMI DMA command chain and the front end queue is fed as described above. The back end of the read pipeline is "drained" by monitoring the BCH completion interrupt found int HW_BCH_CTRL_COMPLETE_IRQ.

An BCH transaction consists of reading or writing all of the blocks requested in the HW_GPMI_ECCCTRL_BUFFER_MASK bit field. As every read transaction completes, it posts the status of all of the blocks to the HW_BCH_STATUS0 register and sets the completion interrupt. The five stages of the BCH read pipeline completes, one in the GPMI and four in the BCH, are independently stalled as they complete and try to deliver to the next stage in the data flow. Several of these stages can be skipped if no-errors are found or once an uncorrectable error is found in a block.

In any case, the final stage will stall if the status register is busy waiting for the CPU to take status register results. The hardware monitors the state of the HW_BCH_CTRL_COMPLETE_IRQ bit. If it is still

set when the last pipeline stage is ready to post data, then the stage will stall. It follows that the next previous stage will stall when it is ready to hand off work to the final stage, and so on up the pipeline.

WARNING: It is important that firmware read the STATUS0/1 results and save them before clearing the interrupt request bit otherwise a transaction and its results could be completely lost.

15.5 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, "Correct Way to Soft Reset a Block," for additional information on using the SFTRST and CLKGATE bit fields.

15.6 Programmable Registers

The following registers are available for programmer access and control of the BCH hardware accelerator.

15.6.1 Hardware BCH ECC Accelerator Control Register Description

The BCH CTRL provides overall control of the hardware ECC accelerator

HW_BCH_CTRL	0x000
HW_BCH_CTRL_SET	0x004
HW_BCH_CTRL_CLR	0x008
HW_BCH_CTRL_TOG	0x00C

Table 15-4. HW_BCH_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	RSVD5							DEBUGSYNDROME	RSVD4		M2M_LAYOUT		M2M_ENCODE	M2M_ENABLE	RSVD3					DEBUG_STALL_IRQ_EN	RSVD2	COMPLETE_IRQ_EN	RSVD1				BM_ERROR_IRQ	DEBUG_STALL_IRQ	RSVD0	COMPLETE_IRQ

Table 15-5. HW_BCH_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	Set this bit to zero to enable normal BCH operation. Set this bit to one (default) to disable clocking with the BCH and hold it in its reset (lowest power) state. This bit can be turned on and then off to reset the BCH block to its default state. This bit resets all state machines except for the AHB master state machine. IMPORTANT: Due to an on-chip bug, BCH SFTRST use is not recommended. If the BCH is soft-reset after any transfers, then the AXI master will be locked up until a hard reset. Even if the AXI master is idle, the AXI state machine will be locked up. Soft resets after hard-reset should be safe, but once you perform any BCH transfer, then any subsequent attempts to soft reset the BCH will almost always lock up the BCH. The only way to recover from a BCH lock-up is hard-reset. There should not be any reason to soft reset the BCH. The BCH should finish every transfer properly and be ready for the next operation no matter the state of the page (correctable, uncorrectable, erased, etc.). RUN = 0x0 Allow BCH to operate normally. RESET = 0x1 Hold BCH in reset.
30	**CLKGATE**	RW	0x1	This bit must be set to zero for normal operation. When set to one it gates off the clocks to the block. RUN = 0x0 Allow BCH to operate normally. NO_CLKS = 0x1 Do not clock BCH gates in order to minimize power consumption.
29:23	**RSVD5**	RO	0x0	Reserved, always set this bit to zero.
22	**DEBUGSYNDROME**	RW	0x0	(For debug purposes only). Enable write of computed syndromes to memory on BCH decode operations. Computed syndromes will be written to the auxiliary buffer after the status block. Syndromes will be written as padded 16-bit values.
21:20	**RSVD4**	RO	0x0	Reserved, always set these bits to zero.
19:18	**M2M_LAYOUT**	RW	0x0	Selects the flash page format (by indexing into one of the HW_BCH_FLASHnLAYOUT register banks) for memory-to-memory operations.
17	**M2M_ENCODE**	RW	0x0	Selects encode (parity generation) or decode (correction) mode for memory-to-memory operations.
16	**M2M_ENABLE**	RW	0x0	WARNING! WRITING THIS BIT INITIATES A MEMORY-TO-MEMORY OPERATION. The BCH module must be inactive (not processing data from the GPMI) when this bit is set. The M2M_ENCODE and M2M_LAYOUT bits as well as the ENCODEPTR, DATAPTR, and METAPTR registers are used for memory-to-memory operations and must be correctly programmed before writing this bit.
15:11	**RSVD3**	RO	0x0	Reserved, always set these bits to zero.
10	**DEBUG_STALL_IRQ_EN**	RW	0x0	1 = interrupt on debug stall mode is enabled. The irq is raised on every block
9	**RSVD2**	RO	0x0	Reserved, always set these bits to zero.
8	**COMPLETE_IRQ_EN**	RW	0x0	1 = interrupt on completion of correction is enabled.
7:4	**RSVD1**	RO	0x0	Reserved, always set these bits to zero.

Table 15-5. HW_BCH_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
3	**BM_ERROR_IRQ**	RW	0x0	AHB Bus interface Error Interrupt Status. Write a one to the SCT clear address to clear the interrupt status bit.
2	**DEBUG_STALL_IRQ**	RW	0x0	DEBUG STALL Interrupt Status. Write a one to the SCT clear address to clear the interrupt status bit.
1	**RSVD0**	RO	0x0	Reserved, always set these bits to zero.
0	**COMPLETE_IRQ**	RW	0x0	This bit indicates the state of the external interrupt line. Write a one to the SCT clear address to clear the interrupt status bit. NOTE: subsequent ECC completions will be held off as long as this bit is set. Be sure to read the data from HW_BCH_STATUS0 before clearing this interrupt bit.

15.6.2 Hardware BCH ECC Accelerator Status Register 0 Description

The BCH STAT provides overall status of the hardware ECC accelerator

HW_BCH_STATUS0 0x010

Table 15-6. HW_BCH_STATUS0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
HANDLE												COMPLETED_CE				STATUS_BLK0								RSVD1			ALLONES	CORRECTED	UNCORRECTABLE	RSVD0	

Table 15-7. HW_BCH_STATUS0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:20	**HANDLE**	RO	0x0	Software supplies a 12 bit handle for this transfer as part of the GPMI DMA PIO operation that started the transaction. That handle passes down the pipeline and ends up here at the time the BCH interrupt is signaled.
19:16	**COMPLETED_CE**	RO	0x0	This is the chip enable number corresponding to the NAND device from which this data came.
15:8	**STATUS_BLK0**	RO	0x0	Count of symbols in error during processing of first block of flash (metadata block). The number of errors reported will be in the range of 0 to the ECC correction level for block 0. ZERO = 0x00 No errors found on block. ERROR1 = 0x01 One error found on block. ERROR2 = 0x02 One errors found on block. ERROR3 = 0x03 One errors found on block. ERROR4 = 0x04 One errors found on block. UNCORRECTABLE = 0xFE Block exhibited uncorrectable errors. ERASED = 0xFF Page is erased.
7:5	**RSVD1**	RO	0x0	Reserved, always set these bits to zero.
4	**ALLONES**	RO	0x1	1 = All data bits of this transaction are ONE.

Table 15-7. HW_BCH_STATUS0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
3	**CORRECTED**	RO	0x0	1 = At least one correctable error encountered during last processing cycle.
2	**UNCORRECTABLE**	RO	0x0	1 = Uncorrectable error encountered during last processing cycle.
1:0	**RSVD0**	RO	0x0	Reserved, always set these bits to zero.

DESCRIPTION:

The BCH STAT register provides visibility into the run-time status of the BCH and status information when processing is complete.

15.6.3 Hardware BCH ECC Accelerator Mode Register Description

The BCH MODE register provides additional mode controls.

HW_BCH_MODE 0x020

Table 15-8. HW_BCH_MODE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																								ERASE_THRESHOLD							

Table 15-9. HW_BCH_MODE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:8	**RSVD**	RO	0x0	Reserved, always set these bits to zero.
7:0	**ERASE_THRESHOLD**	RW	0x00	This value indicates the maximum number of zero bits on a flash block for it to be considered erased. For SLC NAND devices, this value should be programmed to 0 (meaning that the entire block should consist of bytes of 0xFF. For MLC NAND devices, bit errors may occur on reads (even on blank block), so this threshold can be used to tune the erased block checking algorithm.

DESCRIPTION:

Contains additional global mode controls for the BCH engine.

15.6.4 Hardware BCH ECC Loopback Encode Buffer Register Description

When performing memory to memory operations, indicates the address of the encode buffer. This register should be programmed before writing a 1 to the M2M_ENABLE bit in the CTRL register. This value must be aligned on a 4-byte boundary.

HW_BCH_ENCODEPTR 0x030

Table 15-10. HW_BCH_ENCODEPTR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 15-11. HW_BCH_ENCODEPTR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x00000000	Address pointer to encode buffer. This is the source for decode operations and the destination for encode operations.

DESCRIPTION:

For memory to memory operations, this register is used as the pointer to the encoded data, which is an output when encoding and an input while decoding.

15.6.5 Hardware BCH ECC Loopback Data Buffer Register Description

When performing memory to memory operations, indicates the address of the data buffer.

HW_BCH_DATAPTR 0x040

Table 15-12. HW_BCH_DATAPTR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 15-13. HW_BCH_DATAPTR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x00000000	Address pointer to data buffer. This is the source for encode operations and the destination for decode operations. This register should be programmed before writing a 1 to the M2M_ENABLE bit in the CTRL register. This value must be aligned on a 4-byte boundary.

DESCRIPTION:

For memory to memory operations, this register is used as the pointer to the data to encode or the destination buffer for decode operations.

15.6.6 Hardware BCH ECC Loopback Metadata Buffer Register Description

When performing memory to memory operations, indicates the address of the metadata buffer.

HW_BCH_METAPTR 0x050

Table 15-14. HW_BCH_METAPTR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 15-15. HW_BCH_METAPTR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x00000000	Address pointer to metadata buffer. This is the source for encode metadata read operations and the destination for metadata decode operations. This register should be programmed before writing a 1 to the M2M_ENABLE bit in the CTRL register. This value must be aligned on a 4-byte boundary.

DESCRIPTION:

For memory to memory operations, this register is used as the pointer to the metadata to encode or the extracted metadata for decode operations.

15.6.7 Hardware BCH ECC Accelerator Layout Select Register Description

The BCH LAYOUTSELECT register provides a mapping of chip selects to layout registers.

HW_BCH_LAYOUTSELECT 0x070

Table 15-16. HW_BCH_LAYOUTSELECT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CS15_SELECT		CS14_SELECT		CS13_SELECT		CS12_SELECT		CS11_SELECT		CS10_SELECT		CS9_SELECT		CS8_SELECT		CS7_SELECT		CS6_SELECT		CS5_SELECT		CS4_SELECT		CS3_SELECT		CS2_SELECT		CS1_SELECT		CS0_SELECT	

Table 15-17. HW_BCH_LAYOUTSELECT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**CS15_SELECT**	RW	0x3	Selects which layout is used for chip select 15.
29:28	**CS14_SELECT**	RW	0x2	Selects which layout is used for chip select 14.
27:26	**CS13_SELECT**	RW	0x1	Selects which layout is used for chip select 13.
25:24	**CS12_SELECT**	RW	0x0	Selects which layout is used for chip select 12.
23:22	**CS11_SELECT**	RW	0x3	Selects which layout is used for chip select 11.
21:20	**CS10_SELECT**	RW	0x2	Selects which layout is used for chip select 10.
19:18	**CS9_SELECT**	RW	0x1	Selects which layout is used for chip select 9.

Table 15-17. HW_BCH_LAYOUTSELECT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
17:16	**CS8_SELECT**	RW	0x0	Selects which layout is used for chip select 8.
15:14	**CS7_SELECT**	RW	0x3	Selects which layout is used for chip select 7.
13:12	**CS6_SELECT**	RW	0x2	Selects which layout is used for chip select 6.
11:10	**CS5_SELECT**	RW	0x1	Selects which layout is used for chip select 5.
9:8	**CS4_SELECT**	RW	0x0	Selects which layout is used for chip select 4.
7:6	**CS3_SELECT**	RW	0x3	Selects which layout is used for chip select 3.
5:4	**CS2_SELECT**	RW	0x2	Selects which layout is used for chip select 2.
3:2	**CS1_SELECT**	RW	0x1	Selects which layout is used for chip select 1.
1:0	**CS0_SELECT**	RW	0x0	Selects which layout is used for chip select 0.

DESCRIPTION:

When the BCH engine receives a request to process a data block from the GPMI interface, it will use this register to map the incoming chip select to one of the four possible flash layout registers

15.6.8 Hardware BCH ECC Flash 0 Layout 0 Register Description

The flash format register contains a description of the logical layout of data on the flash device. This register is used in conjuction with the FLASH0LAYOUT1 register to control the format for the devices selecting layout 0 in the LAYOUTSELECT register.

HW_BCH_FLASH0LAYOUT0 0x080

Table 15-18. HW_BCH_FLASH0LAYOUT0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
NBLOCKS								META_SIZE								ECC0				DATA0_SIZE											

Table 15-19. HW_BCH_FLASH0LAYOUT0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**NBLOCKS**	RW	0x07	Number of subsequent blocks on the flash page (excluding the data0 block). A value of 0 indicates that only the DATA0 block is present and a value of 8 indicates that 8 subsequent blocks are present for a total of 9 blocks on the flash (including the DATA0 block). Any values from 0 to 255 are supported by the hardware.
23:16	**META_SIZE**	RW	0x0A	Indicates the size of the metadata (in bytes) to be stored on a flash page. The BCH design support from 0 to 255 bytes for metadata -- if set to zero, no metadata will be stored. Metadata is stored before the associated data in block 0. If the DATA0_SIZE field is programmed to a zero, then metadata effectively be stored with its own parity. When both the metadata and data0 fields are programmed with non-zero values, the first block will contain both portions of data and will be covered by a single parity block.
15:12	**ECC0**	RW	0x8	Indicates the ECC level for the first block on the flash page. The first block covers metadata plus the associated data from the DATA0_SIZE field. NONE = 0x0 No ECC to be performed ECC2 = 0x1 ECC 2 to be performed ECC4 = 0x2 ECC 4 to be performed ECC6 = 0x3 ECC 6 to be performed ECC8 = 0x4 ECC 8 to be performed ECC10 = 0x5 ECC 10 to be performed ECC12 = 0x6 ECC 12 to be performed ECC14 = 0x7 ECC 14 to be performed ECC16 = 0x8 ECC 16 to be performed ECC18 = 0x9 ECC 18 to be performed ECC20 = 0xA ECC 20 to be performed
11:0	**DATA0_SIZE**	RW	0x200	Indicates the size of the data 0 block (in bytes) to be stored on the flash page. The data size MUST be a multiple of four bytes. If set to zero, the first block will only contain metadata.

DESCRIPTION:

Each pair of layout registers describes one of four supported flash configurations. Software should program the LAYOUTSELECT register for each supported GPMI chip select to select from one of the fourlayout values. Each pair of registers contains settings that are used by the BCH block while reading/writing the flash page to control data, metadata, and flash page sizes as well as the ECC correction level. The first block written to flash can be programmed to have different ECC, metadata, and data sizes from subsequent data blocks on the device. In addition, the number of blocks stored on a page of flash is not fixed, but instead is determined by the number of bytes consumed by the initial (block 0) and subsequent data blocks. See the BCH programming reference manual for more information on setting up the flash layout registers.

EXAMPLE:

```
HW_BCH_FLASH0LAYOUT0_WR(0x020C8000);
HW_BCH_FLASH0LAYOUT1_WR(0x04408200);
```

15.6.9 Hardware BCH ECC Flash 0 Layout 1 Register Description

The flash format register contains a description of the logical layout of data on the flash device. This register is used in conjuction with the FLASH0LAYOUT0 register to control the format for the device on chip select 0.

HW_BCH_FLASH0LAYOUT1 0x090

Table 15-20. HW_BCH_FLASH0LAYOUT1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PAGE_SIZE																ECCN				DATAN_SIZE											

Table 15-21. HW_BCH_FLASH0LAYOUT1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**PAGE_SIZE**	RW	0x10DA	Indicates the total size of the flash page (in bytes). This should be set to the page size including spare area. The page size is programmable to accomodate different flash configurations that may be available in the future.
15:12	**ECCN**	RW	0x8	Indicates the ECC level for the subsequent blocks on the flash page (blocks 1-n). Subsequent blocks only contain data (no metadata). NONE = 0x0 No ECC to be performed ECC2 = 0x1 ECC 2 to be performed ECC4 = 0x2 ECC 4 to be performed ECC6 = 0x3 ECC 6 to be performed ECC8 = 0x4 ECC 8 to be performed ECC10 = 0x5 ECC 10 to be performed ECC12 = 0x6 ECC 12 to be performed ECC14 = 0x7 ECC 14 to be performed ECC16 = 0x8 ECC 16 to be performed ECC18 = 0x9 ECC 18 to be performed ECC20 = 0xA ECC 20 to be performed
11:0	**DATAN_SIZE**	RW	0x200	Indicates the size of the subsequent data blocks (in bytes) to be stored on the flash page. The data size MUST be a multiple of four bytes. The size of subsequent data blocks does not have to match the data size for block 0, which is important when metadata is stored separately or for balancing the amount of data stored in each block.

EXAMPLE:

```
HW_BCH_FLASH0LAYOUT0_WR(0x020C8000);
HW_BCH_FLASH0LAYOUT1_WR(0x04408200);
```

15.6.10 Hardware BCH ECC Flash 1 Layout 0 Register Description

The flash format register contains a description of the logical layout of data on the flash device. This register is used in conjuction with the FLASH0LAYOUT1 register to control the format for the devices selecting layout 0 in the LAYOUTSELECT register.

HW_BCH_FLASH1LAYOUT0 0x0a0

Table 15-22. HW_BCH_FLASH1LAYOUT0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
NBLOCKS								META_SIZE								ECC0				DATA0_SIZE											

Table 15-23. HW_BCH_FLASH1LAYOUT0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**NBLOCKS**	RW	0x07	Number of subsequent blocks on the flash page (excluding the data0 block). A value of 0 indicates that only the DATA0 block is present and a value of 8 indicates that 8 subsequent blocks are present for a total of 9 blocks on the flash (including the DATA0 block). Any values from 0 to 255 are supported by the hardware.
23:16	**META_SIZE**	RW	0x0A	Indicates the size of the metadata (in bytes) to be stored on a flash page. The BCH design support from 0 to 255 bytes for metadata -- if set to zero, no metadata will be stored. Metadata is stored before the associated data in block 0. If the DATA0_SIZE field is programmed to a zero, then metadata will effectively be stored with its own parity. When both the metadata and data0 fields are programmed with non-zero values, the first block will contain both portions of data and will be covered by a single parity block.
15:12	**ECC0**	RW	0x8	Indicates the ECC level for the first block on the flash page. The first block covers metadata plus the associated data from the DATA0_SIZE field. NONE = 0x0 No ECC to be performed ECC2 = 0x1 ECC 2 to be performed ECC4 = 0x2 ECC 4 to be performed ECC6 = 0x3 ECC 6 to be performed ECC8 = 0x4 ECC 8 to be performed ECC10 = 0x5 ECC 10 to be performed ECC12 = 0x6 ECC 12 to be performed ECC14 = 0x7 ECC 14 to be performed ECC16 = 0x8 ECC 16 to be performed ECC18 = 0x9 ECC 18 to be performed ECC20 = 0xA ECC 20 to be performed
11:0	**DATA0_SIZE**	RW	0x200	Indicates the size of the data 0 block (in bytes) to be stored on the flash page. The data size MUST be a multiple of four bytes. If set to zero, the first block will only contain metadata.

DESCRIPTION:

Each pair of layout registers describes one of four supported flash configurations. Software should program the LAYOUTSELECT register for each supported GPMI chip select to select from one of the fourlayout values. Each pair of registers contains settings that are used by the BCH block while reading/writing the flash page to control data, metadata, and flash page sizes as well as the ECC correction

level. The first block written to flash can be programmed to have different ECC, metadata, and data sizes from subsequent data blocks on the device. In addition, the number of blocks stored on a page of flash is not fixed, but instead is determined by the number of bytes consumed by the initial (block 0) and subsequent data blocks. See the BCH programming reference manual for more information on setting up the flash layout registers.

EXAMPLE:

```
HW_BCH_FLASH1LAYOUT0_WR(0x020C8000);
HW_BCH_FLASH1LAYOUT1_WR(0x04408200);
```

15.6.11 Hardware BCH ECC Flash 1 Layout 1 Register Description

The flash format register contains a description of the logical layout of data on the flash device. This register is used in conjuction with the FLASH0LAYOUT0 register to control the format for the device on chip select 0.

HW_BCH_FLASH1LAYOUT1 0x0b0

Table 15-24. HW_BCH_FLASH1LAYOUT1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PAGE_SIZE																ECCN				DATAN_SIZE											

Table 15-25. HW_BCH_FLASH1LAYOUT1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**PAGE_SIZE**	RW	0x10DA	Indicates the total size of the flash page (in bytes). This should be set to the page size including spare area. The page size is programmable to accomodate different flash configurations that may be available in the future.

Table 15-25. HW_BCH_FLASH1LAYOUT1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:12	**ECCN**	RW	0x8	Indicates the ECC level for the subsequent blocks on the flash page (blocks 1-n). Subsequent blocks only contain data (no metadata). NONE = 0x0 No ECC to be performed ECC2 = 0x1 ECC 2 to be performed ECC4 = 0x2 ECC 4 to be performed ECC6 = 0x3 ECC 6 to be performed ECC8 = 0x4 ECC 8 to be performed ECC10 = 0x5 ECC 10 to be performed ECC12 = 0x6 ECC 12 to be performed ECC14 = 0x7 ECC 14 to be performed ECC16 = 0x8 ECC 16 to be performed ECC18 = 0x9 ECC 18 to be performed ECC20 = 0xA ECC 20 to be performed
11:0	**DATAN_SIZE**	RW	0x200	Indicates the size of the subsequent data blocks (in bytes) to be stored on the flash page. The data size MUST be a multiple of four bytes. The size of subsequent data blocks does not have to match the data size for block 0, which is important when metadata is stored separately or for balancing the amount of data stored in each block.

EXAMPLE:

```
HW_BCH_FLASH1LAYOUT0_WR(0x020C8000);
HW_BCH_FLASH1LAYOUT1_WR(0x04408200);
```

15.6.12 Hardware BCH ECC Flash 2 Layout 0 Register Description

The flash format register contains a description of the logical layout of data on the flash device. This register is used in conjuction with the FLASH0LAYOUT1 register to control the format for the devices selecting layout 0 in the LAYOUTSELECT register.

HW_BCH_FLASH2LAYOUT0 0x0c0

Table 15-26. HW_BCH_FLASH2LAYOUT0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
NBLOCKS								META_SIZE								ECC0				DATA0_SIZE											

Table 15-27. HW_BCH_FLASH2LAYOUT0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**NBLOCKS**	RW	0x07	Number of subsequent blocks on the flash page (excluding the data0 block). A value of 0 indicates that only the DATA0 block is present and a value of 8 indicates that 8 subsequent blocks are present for a total of 9 blocks on the flash (including the DATA0 block). Any values from 0 to 255 are supported by the hardware.
23:16	**META_SIZE**	RW	0x0A	Indicates the size of the metadata (in bytes) to be stored on a flash page. The BCH design support from 0 to 255 bytes for metadata -- if set to zero, no metadata will be stored. Metadata is stored before the associated data in block 0. If the DATA0_SIZE field is programmed to a zero, then metadata effectively be stored with its own parity. When both the metadata and data0 fields are programmed with non-zero values, the first block will contain both portions of data and will be covered by a single parity block.
15:12	**ECC0**	RW	0x8	Indicates the ECC level for the first block on the flash page. The first block covers metadata plus the associated data from the DATA0_SIZE field. NONE = 0x0 No ECC to be performed ECC2 = 0x1 ECC 2 to be performed ECC4 = 0x2 ECC 4 to be performed ECC6 = 0x3 ECC 6 to be performed ECC8 = 0x4 ECC 8 to be performed ECC10 = 0x5 ECC 10 to be performed ECC12 = 0x6 ECC 12 to be performed ECC14 = 0x7 ECC 14 to be performed ECC16 = 0x8 ECC 16 to be performed ECC18 = 0x9 ECC 18 to be performed ECC20 = 0xA ECC 20 to be performed
11:0	**DATA0_SIZE**	RW	0x200	Indicates the size of the data 0 block (in bytes) to be stored on the flash page. The data size MUST be a multiple of four bytes. If set to zero, the first block will only contain metadata.

DESCRIPTION:

Each pair of layout registers describes one of four supported flash configurations. Software should program the LAYOUTSELECT register for each supported GPMI chip select to select from one of the fourlayout values. Each pair of registers contains settings that are used by the BCH block while reading/writing the flash page to control data, metadata, and flash page sizes as well as the ECC correction level. The first block written to flash can be programmed to have different ECC, metadata, and data sizes from subsequent data blocks on the device. In addition, the number of blocks stored on a page of flash is not fixed, but instead is determined by the number of bytes consumed by the initial (block 0) and subsequent data blocks. See the BCH programming reference manual for more information on setting up the flash layout registers.

EXAMPLE:

```
HW_BCH_FLASH2LAYOUT0_WR(0x020C8000);
HW_BCH_FLASH2LAYOUT1_WR(0x04408200);
```

15.6.13 Hardware BCH ECC Flash 2 Layout 1 Register Description

The flash format register contains a description of the logical layout of data on the flash device. This register is used in conjuction with the FLASH0LAYOUT0 register to control the format for the device on chip select 0.

HW_BCH_FLASH2LAYOUT1 0x0d0

Table 15-28. HW_BCH_FLASH2LAYOUT1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PAGE_SIZE																ECCN				DATAN_SIZE											

Table 15-29. HW_BCH_FLASH2LAYOUT1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**PAGE_SIZE**	RW	0x10DA	Indicates the total size of the flash page (in bytes). This should be set to the page size including spare area. The page size is programmable to accomodate different flash configurations that may be available in the future.
15:12	**ECCN**	RW	0x8	Indicates the ECC level for the subsequent blocks on the flash page (blocks 1-n). Subsequent blocks only contain data (no metadata). NONE = 0x0 No ECC to be performed ECC2 = 0x1 ECC 2 to be performed ECC4 = 0x2 ECC 4 to be performed ECC6 = 0x3 ECC 6 to be performed ECC8 = 0x4 ECC 8 to be performed ECC10 = 0x5 ECC 10 to be performed ECC12 = 0x6 ECC 12 to be performed ECC14 = 0x7 ECC 14 to be performed ECC16 = 0x8 ECC 16 to be performed ECC18 = 0x9 ECC 18 to be performed ECC20 = 0xA ECC 20 to be performed
11:0	**DATAN_SIZE**	RW	0x200	Indicates the size of the subsequent data blocks (in bytes) to be stored on the flash page. The data size MUST be a multiple of four bytes. The size of subsequent data blocks does not have to match the data size for block 0, which is important when metadata is stored separately or for balancing the amount of data stored in each block.

EXAMPLE:

```
HW_BCH_FLASH2LAYOUT0_WR(0x020C8000);
HW_BCH_FLASH2LAYOUT1_WR(0x04408200);
```

15.6.14 Hardware BCH ECC Flash 3 Layout 0 Register Description

The flash format register contains a description of the logical layout of data on the flash device. This register is used in conjuction with the FLASH0LAYOUT1 register to control the format for the devices selecting layout 0 in the LAYOUTSELECT register.

HW_BCH_FLASH3LAYOUT0 0x0e0

Table 15-30. HW_BCH_FLASH3LAYOUT0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
NBLOCKS								META_SIZE								ECC0				DATA0_SIZE											

Table 15-31. HW_BCH_FLASH3LAYOUT0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**NBLOCKS**	RW	0x07	Number of subsequent blocks on the flash page (excluding the data0 block). A value of 0 indicates that only the DATA0 block is present and a value of 8 indicates that 8 subsequent blocks are present for a total of 9 blocks on the flash (including the DATA0 block). Any values from 0 to 255 are supported by the hardware.
23:16	**META_SIZE**	RW	0x0A	Indicates the size of the metadata (in bytes) to be stored on a flash page. The BCH design support from 0 to 255 bytes for metadata -- if set to zero, no metadata will be stored. Metadata is stored before the associated data in block 0. If the DATA0_SIZE field is programmed to a zero, then metadata effectively be stored with its own parity. When both the metadata and data0 fields are programmed with non-zero values, the first block will contain both portions of data and will be covered by a single parity block.
15:12	**ECC0**	RW	0x8	Indicates the ECC level for the first block on the flash page. The first block covers metadata plus the associated data from the DATA0_SIZE field. NONE = 0x0 No ECC to be performed ECC2 = 0x1 ECC 2 to be performed ECC4 = 0x2 ECC 4 to be performed ECC6 = 0x3 ECC 6 to be performed ECC8 = 0x4 ECC 8 to be performed ECC10 = 0x5 ECC 10 to be performed ECC12 = 0x6 ECC 12 to be performed ECC14 = 0x7 ECC 14 to be performed ECC16 = 0x8 ECC 16 to be performed ECC18 = 0x9 ECC 18 to be performed ECC20 = 0xA ECC 20 to be performed
11:0	**DATA0_SIZE**	RW	0x200	Indicates the size of the data 0 block (in bytes) to be stored on the flash page. The data size MUST be a multiple of four bytes. If set to zero, the first block will only contain metadata.

DESCRIPTION:

Each pair of layout registers describes one of four supported flash configurations. Software should program the LAYOUTSELECT register for each supported GPMI chip select to select from one of the fourlayout values. Each pair of registers contains settings that are used by the BCH block while reading/writing the flash page to control data, metadata, and flash page sizes as well as the ECC correction

level. The first block written to flash can be programmed to have different ECC, metadata, and data sizes from subsequent data blocks on the device. In addition, the number of blocks stored on a page of flash is not fixed, but instead is determined by the number of bytes consumed by the initial (block 0) and subsequent data blocks. See the BCH programming reference manual for more information on setting up the flash layout registers.

EXAMPLE:

```
HW_BCH_FLASH3LAYOUT0_WR(0x020C8000);
HW_BCH_FLASH3LAYOUT1_WR(0x04408200);
```

15.6.15 Hardware BCH ECC Flash 3 Layout 1 Register Description

The flash format register contains a description of the logical layout of data on the flash device. This register is used in conjuction with the FLASH0LAYOUT0 register to control the format for the device on chip select 0.

HW_BCH_FLASH3LAYOUT1 0x0f0

Table 15-32. HW_BCH_FLASH3LAYOUT1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PAGE_SIZE																ECCN				DATAN_SIZE											

Table 15-33. HW_BCH_FLASH3LAYOUT1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**PAGE_SIZE**	RW	0x10DA	Indicates the total size of the flash page (in bytes). This should be set to the page size including spare area. The page size is programmable to accomodate different flash configurations that may be available in the future.

Table 15-33. HW_BCH_FLASH3LAYOUT1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:12	**ECCN**	RW	0x8	Indicates the ECC level for the subsequent blocks on the flash page (blocks 1-n). Subsequent blocks only contain data (no metadata). NONE = 0x0 No ECC to be performed ECC2 = 0x1 ECC 2 to be performed ECC4 = 0x2 ECC 4 to be performed ECC6 = 0x3 ECC 6 to be performed ECC8 = 0x4 ECC 8 to be performed ECC10 = 0x5 ECC 10 to be performed ECC12 = 0x6 ECC 12 to be performed ECC14 = 0x7 ECC 14 to be performed ECC16 = 0x8 ECC 16 to be performed ECC18 = 0x9 ECC 18 to be performed ECC20 = 0xA ECC 20 to be performed
11:0	**DATAN_SIZE**	RW	0x200	Indicates the size of the subsequent data blocks (in bytes) to be stored on the flash page. The data size MUST be a multiple of four bytes. The size of subsequent data blocks does not have to match the data size for block 0, which is important when metadata is stored separately or for balancing the amount of data stored in each block.

EXAMPLE:

```
HW_BCH_FLASH3LAYOUT0_WR(0x020C8000);
HW_BCH_FLASH3LAYOUT1_WR(0x04408200);
```

15.6.16 Hardware BCH ECC Debug Register0 Description

The hardware BCH accelerator internal state machines and signals can be seen in the ECC debug register.

HW_BCH_DEBUG0	0x100
HW_BCH_DEBUG0_SET	0x104
HW_BCH_DEBUG0_CLR	0x108
HW_BCH_DEBUG0_TOG	0x10C

Table 15-34. HW_BCH_DEBUG0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1					ROM_BIST_ENABLE	ROM_BIST_COMPLETE	KES_DEBUG_SYNDROME_SYMBOL									KES_DEBUG_SHIFT_SYND	KES_DEBUG_PAYLOAD_FLAG	KES_DEBUG_MODE4K	KES_DEBUG_KICK	KES_STANDALONE	KES_DEBUG_STEP	KES_DEBUG_STALL	BM_KES_TEST_BYPASS	RSVD0				DEBUG_REG_SELECT			

Table 15-35. HW_BCH_DEBUG0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:27	**RSVD1**	RO	0x0	Reserved, always set these bits to zero.
26	**ROM_BIST_ENABLE**	RW	0x0	Software may initiate a ROM BIST operation by toggling this bit from a zero to a one. When the operation is complete, the ROM_BIST_COMPLETE bit will be set and the ROM's CRC value will be available in the DEBUG data register.
25	**ROM_BIST_COMPLETE**	RW	0x0	This bit will be set after a BIST operation completes, at which time the ROM CRC is available in the DBGKESREAD register. The CRC value will be cleared after the BIST_ENABLE bit is cleared.
24:16	**KES_DEBUG_SYNDROME_SYMBOL**	RW	0x0	The 9 bit value in this bit field will be shifted into the syndrome register array at the input of the KES engine whenever HW_BCH_DEBUG0_KES_DEBUG_SHIFT_SYND is toggled. NORMAL = 0x0 Bus master address generator for synd_gen writes operates normally. TEST_MODE = 0x1 Bus master address generator always addresses last four bytes in Auxilliary block.
15	**KES_DEBUG_SHIFT_SYND**	RW	0x0	Toggling this bit causes the value in HW_BCH_DEBUG0_KES_SYNDROME_SYMBOL to be shift into the syndrome register array at the input to the KES engine. After shifting in 16 symbols, one can kick off both KES and CF cycles by toggling HW_BCH_DEBUG0_KES_DEBUG_KICK. Be sure to set KES_BCH_DEBUG0_KES_STANDALONE mode to 1 before kicking.
14	**KES_DEBUG_PAYLOAD_FLAG**	RW	0x0	When running the stand alone debug mode on the error calculator, the state of this bit is presented to the KES engine as the input payload flag. DATA = 0x1 Payload is set for 512 byte data block. AUX = 0x1 Payload is set for 65 or 19 byte auxilliary block.
13	**KES_DEBUG_MODE4K**	RW	0x0	When running the stand alone debug mode on the error calculator, the state of this bit is presented to the KES engine as the input mode (4K or 2K pages). 4k = 0x1 Mode is set for 4K NAND pages. 2k = 0x1 Mode is set for 2K NAND pages.
12	**KES_DEBUG_KICK**	RW	0x0	Toggling causes KES engine FSM to start as if kicked by the Bus Master. This allows stand alone testing of the KES and Chien Search engines. Be sure to set KES_BCH_DEBUG0_KES_STANDALONE mode to 1 before kicking.
11	**KES_STANDALONE**	RW	0x0	Set to one to cause the KES engine to suppress toggling the KES_BM_DONE signal to the bus master and to suppress toggling the CF_BM_DONE signal by the CF engine. NORMAL = 0x0 Bus master address generator for synd_gen writes operates normally. TEST_MODE = 0x1 Bus master address generator always addresses last four bytes in Auxilliary block.
10	**KES_DEBUG_STEP**	RW	0x0	Toggling this bit causes the KES FSM to skip past the stall state if it is in DEBUG_STALL mode and it has completed processing a block.

Table 15-35. HW_BCH_DEBUG0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
9	**KES_DEBUG_STALL**	RW	0x0	Set to one to cause KES FSM to stall after notifying Chien search engine to start processing its block but before notifying the bus master that the KES computation is complete. This allows a diagnostic to stall the FSM after each block's key equations are solved. This also has the effect of stalling the CSFE search engine so it's state can be examined after it finishes processing the KES stalled block. NORMAL = 0x0 KES FSM proceeds to next block supplied by bus master. WAIT = 0x1 KES FSM waits after current equations are solved and the search engine is started.
8	**BM_KES_TEST_BYPASS**	RW	0x0	1 = Point all synd_gen writes to dummy area at the end of the AUXILIARY block. With this data diagnostics can preload all payload, parity bytes and computed syndrome bytes for testing the KES engine. NORMAL = 0x0 Bus master address generator for synd_gen writes operates normally. TEST_MODE = 0x1 Bus master address generator always addresses last four bytes in Auxilliary block.
7:6	**RSVD0**	RO	0x0	Reserved, always set these bits to zero.
5:0	**DEBUG_REG_SELECT**	RW	0x0	The value loaded in this bit field is used to select the internal register state view of KES engine or the Chien search engine.

DESCRIPTION:

The HW_BCH_DEBUG0 register provides access to various internal state information which might prove useful during hardware debug and validation.

EXAMPLE:

```
// perform BIST operation
HW_BCH_DEBUG0_SET(BM_BCH_DEBUG0_ROM_BIST_ENABLE); // enable BIST operation
// poll until BIST_DONE
while( (HW_BCH_DEBUG0_RD() & BM_BCH_DEBUG0_ROM_BIST_COMPLETE) == 0 );
i=HW_BCH_DBGKESREAD();
if(HW_BCH_DBGKESREAD_RD() != 0x7AA3792F) {
// BIST FAILED
err++;
}
HW_BCH_DEBUG0_CLR(BM_BCH_DEBUG0_ROM_BIST_ENABLE | BM_BCH_DEBUG0_ROM_BIST_COMPLETE); // clear bist
status
```

15.6.17 Hardware BCH ECC KES Debug Read Register Description

The hardware ECC 8 accelerator key equation solver internal state machines and signals can be seen in the ECC debug registers.

HW_BCH_DBGKESREAD 0x110

Table 15-36. HW_BCH_DBGKESREAD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
VALUES																															

Table 15-37. HW_BCH_DBGKESREAD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**VALUES**	RO	0x0	This register will return the ROM BIST CRC value after a BIST test.

15.6.18 Hardware BCH ECC Chien Search Debug Read Register Description

The hardware ECC 8 accelerator Chien Search internal state machines and signals can be seen in the ECC debug registers.

HW_BCH_DBGCSFEREAD 0x120

Table 15-38. HW_BCH_DBGCSFEREAD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
VALUES																															

Table 15-39. HW_BCH_DBGCSFEREAD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**VALUES**	RO	0x0	Reserved

15.6.19 Hardware BCH ECC Syndrome Generator Debug Read Register Description

The hardware ECC 8 accelerator syndrome generator internal state machines and signals can be seen in the ECC debug registers.

HW_BCH_DBGSYNDGENREAD 0x130

Table 15-40. HW_BCH_DBGSYNDGENREAD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
VALUES																															

Table 15-41. HW_BCH_DBGSYNDGENREAD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**VALUES**	RO	0x0	Reserved

15.6.20 Hardware BCH ECC AXI Master Debug Read Register Description

The hardware BCH ECC 8 AXI bus master internal state machines and signals can be seen in the ECC debug registers.

HW_BCH_DBGAHBMREAD 0x140

Table 15-42. HW_BCH_DBGAHBMREAD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
VALUES																															

Table 15-43. HW_BCH_DBGAHBMREAD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**VALUES**	RO	0x0	Reserved

15.6.21 Hardware BCH ECC Block Name Register Description

Read only view of the block name string BCH.

HW_BCH_BLOCKNAME 0x150

Table 15-44. HW_BCH_BLOCKNAME

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
NAME																															

Table 15-45. HW_BCH_BLOCKNAME Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**NAME**	RO	0x20484342	Should be the ascii characters "BCH " (0x20, H, C, B).

DESCRIPTION:

Fixed pattern read only value for test puposes.

Can be read as an ASCII string with the zero termination coming from the first byte of the BLOCKVERSION register.

EXAMPLE:

```
char *cp = ((char *)HW_BCH_BLOCKNAME_ADDR); reads back "BCH ".
```

15.6.22 Hardware BCH ECC Version Register Description

This register always returns a known read value for debug purposes. The read value indicates the version of the block.

HW_BCH_VERSION 0x160

Table 15-46. HW_BCH_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 15-47. HW_BCH_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x01	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x0	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
if (HW_BCH_VERSION.B.MAJOR != 1) Error();
```

BCH Block v1.0, Revision 2.5

Chapter 16
Data Co-Processor (DCP)

This chapter describes the data co-processor (DCP) block included on the i.MX23 and how to use it. It includes sections on memory copy functionality, Advanced Encryption Standard (AES), hashing, color-space conversion, and arbitration. Sections on programming channel operations and the color-space converter are also provided, along with example code. Programmable registers are described in Section 16.3, "Programmable Registers."

16.1 Overview

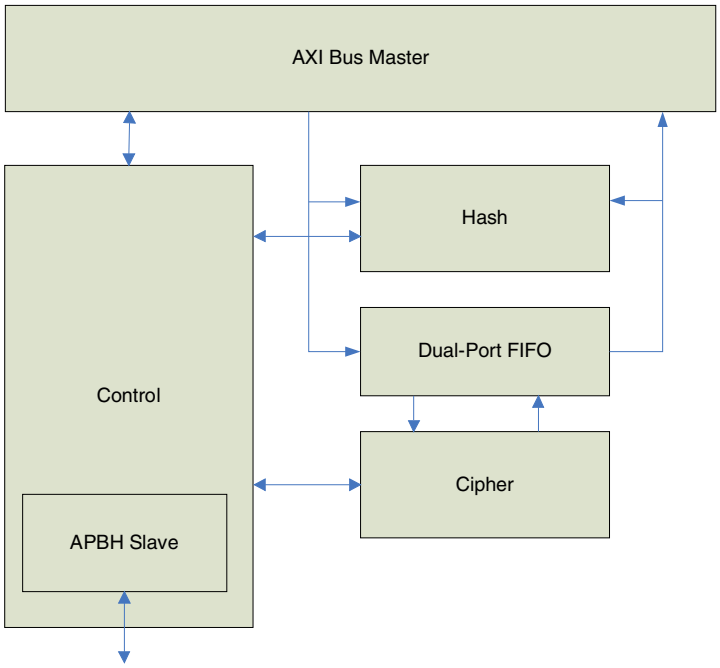


Figure 16-1. Data Co-Processor (DCP) Block Diagram

The DCP module provides support for general encryption and hashing functions typically used for security functions. Because its basic job is moving data from memory to memory, it also incorporates memory-copy (memcopy) function for both debugging and as a more efficient method of copying data between memory blocks than the DMA-based approach. The memcopy function also has a "blit" mode of operation where it can transfer a rectangular block of data to a video frame buffer.

The DCP has been designed to support a wide variety of encryption and hashing algorithms, and the control structures have been designed to allow flexibility in adding additional algorithms and modes in the future. It supports up to 16 encryption algorithms (for example DES, TDEA, AES-128, etc.) with 16 different modes of operation (ECB, CBC, etc.) as well as 16 hashing algorithms (for example MD5, SHA-1, SHA-2, etc.). While the DCP module has been designed to support numerous algorithms, only a subset may be implemented in any given implementation (see the Capability Register).

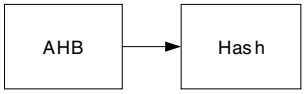
The DCP module processes data based on chained command structures written to system memory by software (in a manner similar to the DMA engine). The control block maintains registers to support four independent and concurrent chains, allowing software to virtualize access to the DCP block. Each command in a chain represents a work unit that the module will process to completion. At the end of each work unit, the control logic arbitrates among chains with outstanding commands and processes a command from that chain. Arbitration among the channels is round-robin, allowing all active channels equal access to the data engine. Each channel also supports a “high-priority” mode that allows it to have priority over the remaining channels. If multiple channels are selected as high-priority, the channel arbiter selects among the high-priority channels in round-robin fashion.

The data flow through the DCP module can be configured in one of five fashions, depending on the functionality activated by the control packet:

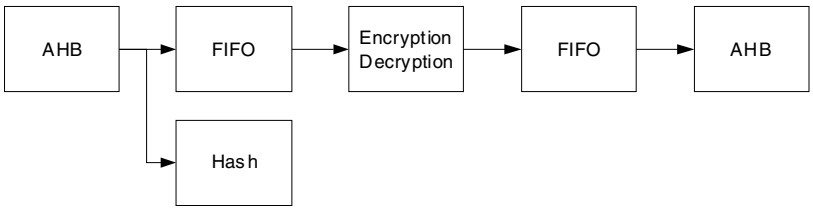
- **Memcopy/Blit Mode—**Data is moved unchanged from one memory buffer to another.

- **Encryption Only—**Data from source buffer is encrypted/decrypted into the destination buffer

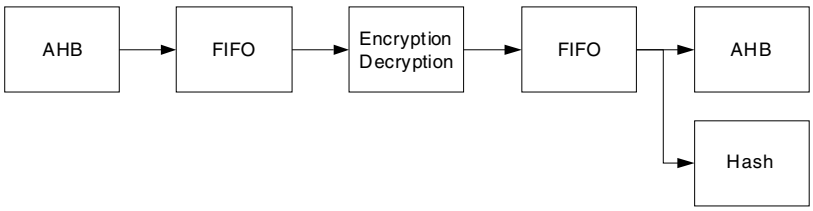
- **Hashing Only—**Data from source buffer is read, and a hash is generated.



- **Encryption and Input Hashing—**Data from source buffer is encrypted/decrypted into destination, and source buffer is hashed.



- **Encryption and Output Hashing—**Data from source buffer is encrypted/decrypted into destination, and output data is hashed.



16.1.1 DCP Limitations for Software

While the DCP module has been designed to be as flexible as possible, there are a few limitations to which software must adhere:

- Buffer sizes for all operations MUST be aligned to the natural size of the transfer algorithm used. Memcopy operations can transfer any number of bytes (one-byte granularity) and AES operations must be multiples of 16 bytes (four-word granularity). For all operations, if the byte count is not a word granularity, the hardware rounds up to the next word. Hashing is supported at a byte granularity.
- The DCP module supports buffer operations to any byte alignment, but performance will be improved if buffers are aligned to a four-byte boundary, since fetch/store operations can be performed without having to do byte operations to accommodate the misaligned addresses.
- Hash operations are limited to a 512-Mbyte buffer size. The hardware only implements a 32-bit hash length counter instead of the 64-bit counter supported by the SHA-1 algorithm (counter counts bits, not bytes, therefore a total of 512 Mbytes).
- For chained hashing operations (operations involving multiple descriptors), every descriptor except the last must have a byte count that is a 16-word multiple (granularity of the hash algorithm).
- Key values cannot be written while the AES block is active. This limitation exists because the key RAM is in use while AES is operational. Any writes from the APB cannot be held in wait states; therefore, the RAM must be accessible during key writes.
- The byte-swap controls can only be used with modulo-4 length buffers. For non-modulo-4 lengths, the final partial word will contain incorrect data. Any address alignment can be used with byte swapping, however.

- The word-swap controls are only useful with cipher operations, because the logic forms the 128-bit cipher data from four words from system memory. The word-swap controls are ignored for memcopy or hashing operations.

16.2 Operation

The top-level DCP module contains the AXI master, APB slave bus interface units, the main control block and FIFO, and any encryption or hashing blocks included with the design.

The controller manages the fetching of work blocks, the fetching/storing of context information when switching between chain pointers, and the managing of the data flow through the FIFO, SHA, and AES blocks. Data entering the block from the AXI master is placed in the FIFO for consumption by the cipher block. After the cipher module has finished its operation, data is placed back into the FIFO and stored back to memory via the AXI master. When hashing is enabled, the SHA block takes its inputs from the bus side of the FIFO to allow it to operate without waiting for the cipher block to complete. The APB slave provides all register controls and interfaces mainly with the control block.

16.2.1 Memory Copy, Blit, and Fill Functionality

In its most basic operation, the DCP supports moving unmodified data from one place in system memory to another. This functionality is referred to as “memcopy”, because it operates only on memory and it copies data from one place to another. Typical uses for memcopy might be for fast virtual memory page moves or repositioning data blocks in memory. Memcopy buffers can be aligned to any memory address and can be of any length (byte granularity). For best performance, buffers should be word-aligned, although the DCP includes enhancements to improve performance for unaligned cases.

The DCP also has the ability to do basic “blit” operations that are typical in graphics operations. To specify a blit, the control packet must have the ENABLE_BLIT bit set in the packet control register. Blit source buffers must be contiguous. The output destination buffer for a blit operation is defined as a “M runs of N bytes” that define a rectangular region in a frame buffer. For blit operations, each line of the blit may consist of any number of bytes. After performing a “run”, the DCP updates the destination pointer such that the next destination address falls on the pixel below the start of the previous run operation. This is done by incrementing the starting pointer by the frame buffer width, which is specified in the Control1 field.

In addition to being able to copy data within memory, the DCP also provides a “fill” operation, where source data comes not from another memory location, but from an internal register (the source buffer address in the control packet). This is done whenever the CONSTANT_FILL flag is set in the packet control register. This feature may be used with memcopy to prefill memory with a specified value or during a blit operation to fill a rectangular region with a constant color.

16.2.2 Advanced Encryption Standard (AES)

The AES block implements a 128-bit key/data encryption/decryption block as defined by the National Institute of Standards and Technology (NIST) as US FIPS PUB 197, dated November 2001 (see references for specifications and toolkits)[1].

There are three variations of AES, each corresponding to the key size used: AES-128, AES-192 or AES-256. AES always operates on 128 bits of data at a time. Only the AES-128 algorithm is implemented at this time.

16.2.2.1 Key Storage

The DCP implements four SRAM-based keys that may be used by software to securely store keys on a semi-permanent basis. The keys may be written via the PIO interface by specifying a key index to specify which key to load and a subword pointer that indicates which word to write within the key. After a subword is written, the logic automatically increments the subword pointer so that software can program the higher-order words of the key without rewriting the key index. Keys written into the key storage are not readable.

To use a key in the key storage, the cipher descriptor packet should select the key by setting the KEY_SELECT field in the Control1 descriptor field without setting the OTP_KEY or PAYLOAD_KEY fields in the Control0 register.

16.2.2.2 OTP Key

After a system reset, the OTP controller reads the e-fuse devices and provides OTP key information over a parallel 128-bit interface. The key transfer interface runs on HCLK and provides the key over the serialized otp_data signal. The otp_crypto_key_smpl signal indicates when the key value is valid and causes the control logic to capture the key into the key RAM.

To use the OTP key, the descriptor packet should set the OTP_KEY field in the Control1 register.

16.2.2.3 Encryption Modes

The most basic form of encryption is the Electronic Code Book (ECB) mode. In this mode, the encryption output is a function only of the key and the plaintext, thus it can be visualized as a giant lookup table. While this provides a great deal of security, there are a few limitations. For instance, if the same plaintext appears more than once in a block of data, the same ciphertext will also appear. This can be very evident if the plaintext contains large blocks of constant data (0s for example) and can be used to formulate attacks against a key.

1. *The AES core used in the design was derived from the AES design available from OpenCores.org under a modified BSD license as described here: http://www.opencores.org/projects.cgi/web/aes_core/overview*
The license for this code is documented on page <Link>56 for compliance.

In order to make ciphers stronger, several modes of operation can be implemented around the basic ECB cipher to provide additional security. One such mode is CBC mode (Cipher Block Chaining), which takes the previous encrypted data and logically XORs it with the next incoming plaintext before performing the encryption. During decryption, the process is reversed and the previous encrypted data is XORed with the decrypted ECB data to provide the plaintext again.

The AES engine supports handling the various modes of operation. The core AES block supports ECB mode, and other algorithms are handled in the wrapper around the encryption blocks. The DCP module supports Cipher Block Chaining (CBC), which chains the data blocks by XORing the previously encrypted data with the plaintext before encryption. Cipher block chaining encryption is illustrated in Figure 16-2.

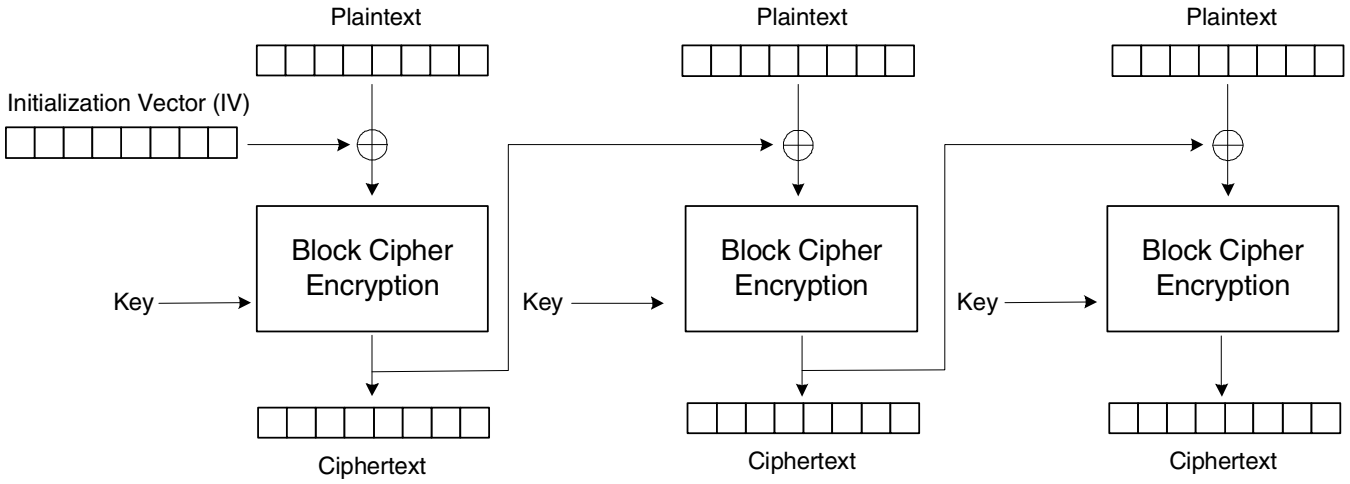


Figure 16-2. Cipher Block Chaining (CBC) Mode Encryption

Decryption (shown in Figure 16-3) works in a similar manner, where the cipher text is first decrypted and then XORed with the previous ciphertext. For the first encryption/decryption operation, an initialization vector (IV) is used to seed the operation. The IV must be the same for both the encryption and decryption steps; otherwise, decrypted data will not match the original plaintext.

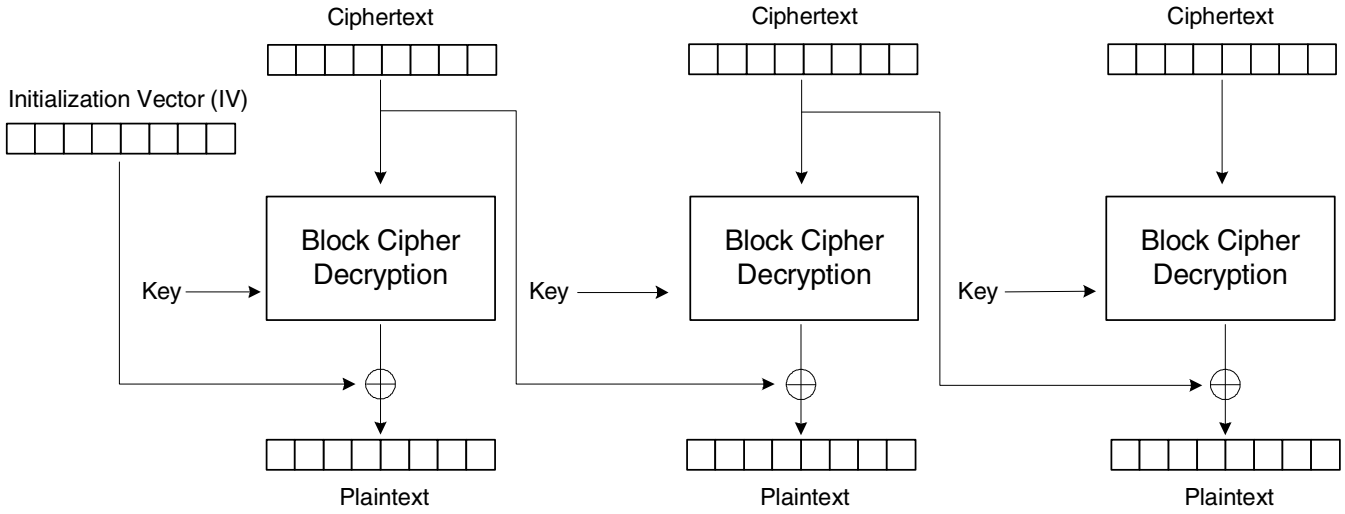


Figure 16-3. Cipher Block Chaining (CBC) Mode Decryption

16.2.3 Hashing

The hashing module implements the SHA-1 hashing algorithm and a modified CRC-32 checksum algorithm. These algorithms produce a signature for a block of data that can be used to determine whether the data is intact.

The CRC-32 algorithm implements a 32-bit CRC algorithm similar to the one used by Ethernet and many other protocols. The CRC differs from the Unix cksum() function in three ways:

- The CRC is initialized as 0xFFFFFFFF instead of 0x00000000.
- Logic pads zeros to a 32-bit boundary for trailing bytes.
- Logic does not post-pend the file length.

The SHA-1 block implements a 160-bit hashing algorithm that operates on 512-bit (64-byte) blocks as defined by US FIPS PUB 180-1 in 1995. The purpose of the hashing module is to generate a unique signature for a block of data that can be used to validate the integrity of the data by comparing the resulting "digest" with the original digest.

Results from hash operations are written to the beginning of the payload for the descriptor. The DCP also has the ability to check the resulting hash against a value in the payload and issue an interrupt if a mismatch occurs.

16.2.4 Managing DCP Channel Arbitration and Performance

The DCP can have four channels compete for DCP resources to complete their operations. Depending on the situation, critical operations may need to be prioritized above less important operations to ensure smooth system operation. To help software achieve this goal, the DCP implements a programmable arbi-

ter for internal DCP operations and provides “recovery” timers on each channel to throttle channel activity.

16.2.4.1 DCP Arbitration

The DCP implements a multi-tiered arbitration policy that allows software a lot of flexibility in scheduling DCP operations. Figure 16-4 illustrates the arbitration levels and where each channel fits into the arbitration scheme.

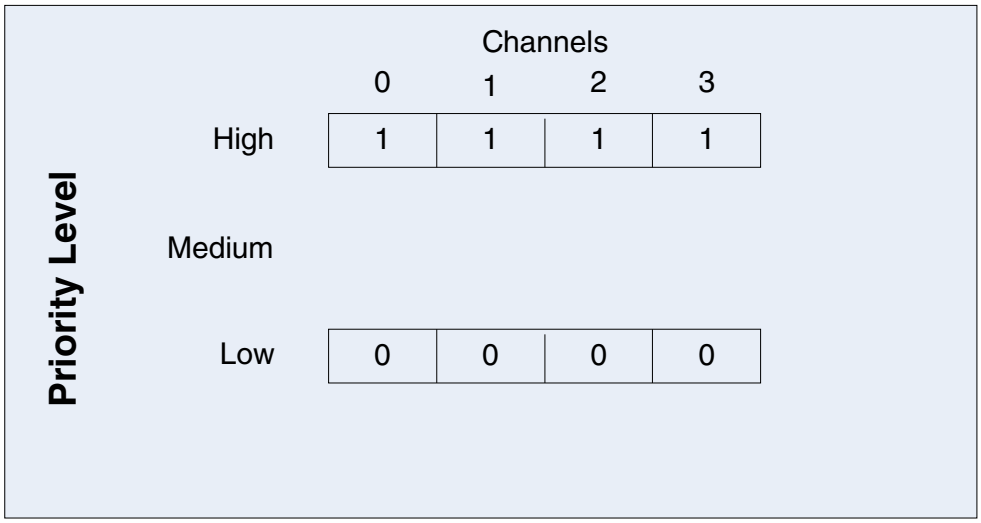


Figure 16-4. DCP Arbitration

Each channel can be programmed as being in either the high-priority or low-priority arbitration pool, depending on the setting in the HIGH_PRIORITY_CHANNEL field of the HW_DCP_CHANNELCTRL register. When the corresponding bit is programmed as a 1, the channel arbitrates in the high-priority pool; otherwise it arbitrates in the low-priority pool. Once a channel has been selected, it completes one packet and then the arbiter re-arbitrates. The channel that just completed participates in the new arbitration round.

Each arbitration pool is arbitrated independently in a round-robin fashion. This ensures that the arbiter is perfectly fair in its distribution of resources. For each arbitration cycle, any pending requests in the high-priority pool are serviced first, followed by requests in the medium and low pools.

16.2.4.2 Channel Recovery Timers

Each channel also contains a channel recovery timer in its HW_DCP_CHnOPTS register. The purpose of the recovery timer is to keep the channel inactive for a period of time after it completes an operation. This capability could be used for a high-priority channel to ensure that at least some lower-priority requests get serviced between packets or to simply allow more timeslices for other channels to perform operations. The value programmed into the recovery timers register delays the channel from operations until 16 times

the programmed value. Any non-zero value should prevent the channel from participating in the next arbitration cycle.

16.2.5 Programming Channel Operations

The control logic block maintains the channel pointers and manages arbitration and context switching between the different channels. It also manages the fetching of work packets and data fetch/store operations from the AXI master interface and coordinates the actions of the hashing and encryption blocks.

The control logic maintains four channels that allow software to effectively create four independent work sets for the DCP module. Software can construct chained control packets in memory that describe encryption/hashing/memcopy operations to the hardware unit. The address for this first control packet can be written to one of the four virtual channels and enabled. When one or more of the channels is enabled, the controller fetches the control packet pointed to by that channel and initiate data fetches from the source buffer. Data is then processed as described in the control packet and stored back to system memory.

Once the processing for a control packet is complete, the controller writes completion status information back to the control packet, and optionally stores relevant state information to the context buffer. If the control packet specifies a subsequent control packet, the channel's pointer is updated to the address for the next packet and an optional interrupt can be issued to the processor.

At this point, the DCP module arbitrates among the virtual channels for the next operation and processes its control packet. If a subsequent operation is continued from a previous operation, the controller automatically loads the context from the previous session into the working registers before resuming operation for that channel.

16.2.5.1 Virtual Channels

The DCP module processes data via work packets stored in memory. Each of the channels contains a pointer to the current work packet and enough control logic to determine whether the channel is active and to provide status to the processor. Each channel also provides a recovery timer to help throttle processing by a particular channel. After processing a packet, the channel enables its 16-bit recovery timer (if the recovery time is set to a non-zero value). The channel will not become active again until its recovery timer has expired. The recovery timer timebase is 16 HCLK cycles, so the timer acts as a 20-bit timer with the bottom four bits implicitly tied to 0. This provides an effective range of zero to 2^{20-1} clocks or 0 ns to 7.8 ms at 133 MHz.

A channel is activated any time its semaphore is non-zero and its recovery timer is cleared. The semaphore can be incremented by software to indicate that the chain pointer has been loaded with a valid pointer. As the hardware completes the work packets, it decrements the semaphore if the Decrement Semaphore flag in the Control 0 field set. Work packets may be chained together using the CHAIN or CHAIN_CONTIGUOUS bits in the Control0 field, which causes the channel to automatically update the

work packet pointer to the value in the NEXT_COMMAND_ADDRESS field at the end of the current work packet.

All channels have the same capabilities, but channel 0 is special in that it has a private interrupt line (dcp_vmi_irq). This allows software to use it for VMI (virtual memory) page-copy operations and have a dedicated interrupt vector to reduce latency. All other channels (and the color-space converter) share the other interrupt (dcp_irq).

16.2.5.2 Context Switching

The control logic maintains four virtual channels that allow the DCP block to time-multiplex encryption, hashing, and memcopy operations, it must also retain state information when changing channels so that when a channel is resumed, it can resume the operation from where it left off. This process is called context-switching.

To minimize the number of registers used in the design, the controller saves context information from each channel into a private context area in system memory. When initializing the DCP module, software must allocate memory for the context buffer and write the address into the Context Buffer Pointer register. As the DCP module processes packets, it saves the context information for each channel to the buffer after completing each control packet. When the channel is subsequently activated, the DCP module's internal registers are then reloaded with the proper context before resuming the operation.

Each channel reserves one-fourth the context buffer area for its context storage. The context buffer consumes 160 bytes of system memory and is formatted as shown in Table 16-1.

Table 16-1. DCP Context Buffer Layout

Range	Channel	Data	Size
0x00–0x0C	3	Cipher Context	16 bytes
0x10–0x24		Hash Context	24 bytes
0x28–0x34	2	Cipher Context	16 bytes
0x38–0x4C		Hash Context	24 bytes
0x50–0x5C	1	Cipher Context	16 bytes
0x60–0x74		Hash Context	24 bytes
0x78–0x84	0	Cipher Context	16 bytes
0x88–0x9C		Hash Context	24 bytes

The control logic writes to the context buffer only if the function is being used. This effectively means that the cipher context is stored for CBC encryption/decryption operations only, and the hash context is written only if SHA-1 is utilized. If neither of these modes are used for a given channel, the memory for the context buffer need not be allocated by software.

Since channel 0 is likely to be used for VMI in an SDRAM-based system-to-page data from the SDRAM to on-chip SRAM, the buffer allocation table has been organized so that the *highest* numbered channel uses the *lowest* area in the context buffer. For this reason, software should allocate the higher numbered channels for encryption/hashing operations and the lower numbered channels for memcopy operations to reduce the size of the context buffer.

If only a single channel is used for CBC mode operations or hashing operations, the controller provides a bit in the control register to disable context switching. In this scenario, context switching is not required, because other channels will not corrupt the state of the hashing or cipher modes. Thus, when the channel resumes, a context load is not required.

Additionally, the control logic monitors the use of CBC/hashing, so that a context reload is not done if the previous channel resumes an operation without an intermediate operation from another channel.

16.2.5.3 Working with Semaphores

Each channel has a semaphore register to indicate that control packets are ready to be processed. Several techniques can be used when programming the semaphores to control the execution of packets within a channel. The channel will continue to execute packets as long as the semaphore is non-zero, a chain bit is set in the descriptor, and no error has occurred.

- Software can write the number of pending packets into the semaphore register with the Decrement Semaphore bit in each control packet set. In this scenario, the channel simply decrements the semaphore for each packet set and terminates at the end of the packet chain. The benefit of this method is that software can easily determine how many packets have executed by reading the semaphore status register.
- Software can create a packet chain with the Decrement Semaphore bit set only in the last packet. In this case, software would write a 1 to the semaphore register to start the chains and the DCP will terminate after executing the last packet.
- Software can create a packet chain with the Decrement Semaphore bit set for each packet and write either a 1 or a number less than the number of packets to the semaphore register. This can be useful when debugging, because it allows the channel to execute only a portion of the packets and software can inspect intermediate values before restarting the channel again.

If an error occurs, the channel issues an interrupt and clears the semaphore register. The channel does not perform any further operations until the error bits in the status register have been cleared. Software can manually clear a non-zero semaphore by writing 0xFF to the CLR (clear) address of the semaphore register.

16.2.5.4 Work Packet Structure

Work packets for the DCP module are created in memory by the processor. Each work packet includes all information required for the hardware to process the data. The general structure of the packet is shown in Figure 16-5.

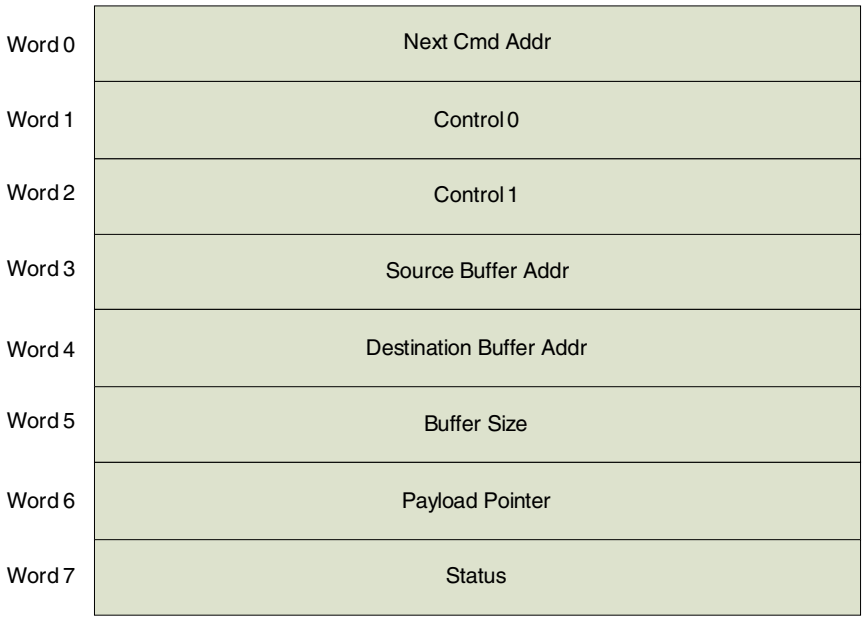


Figure 16-5. DCP Work Packet Structure

For some operations, additional information is required to process the data in the packet. This additional information is provided in the variable-sized payload buffer, which can be found at the address specified in the payload pointer field. When encryption is specified, the payload may include the encryption key (if the key selected resides in the packet), an initialization vector (for modes of operation such as CBC), and expected hash values when data hashing is indicated. The hardware can automatically compare the expected hash with the actual hash and interrupt the processor only on a mismatch. The payload area is used by software to store the calculated hash value at the end of hashing operations (when the HASH_TERM control bit is set).

16.2.5.4.1 Next Command Address Field

The NEXT_COMMAND_ADDRESS field (as shown in Table 16-2) is used to point to the next work packet in the chain. This field is loaded into the channel's command pointer after the current packet has completed processing if the CHAIN bit in the CONTROL0 field (see Table 16-3) is set. The Next Command Address field should be programmed to a non-zero value when the CHAIN bit is set; otherwise, the channel will flag an invalid setup error.

Table 16-2. DCP Next Command Address Field

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
NEXT_COMMAND_ADDRESS																															

16.2.5.4.2 Control0 Field

The main functions of the DCP module are enabled with the ENABLE_MEMCOPY, ENABLE_BLIT, ENABLE_CIPHER, and ENABLE_HASH bits from the Control0 field in the work packet. The combi-

nations of these bits determine the function performed by the DCP. Table 16-4 summarizes the function performed for each combination.

Table 16-3. DCP Control0 Field

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TAG								OUTPUT_WORDSWAP	OUTPUT_BYTESWAP	INPUT_WORDSWAP	INPUT_BYTESWAP	KEY_WORDWRAP	KEY_BYTEWRAP	TEST_SEMA_IRQ	CONSTANT_FILL	HASH_OUTPUT	HASH_CHECK	HASH_TERM	HASH_INIT	PAYLOAD_KEY	OTP_KEY	CIPHER_INIT	CIPHER_ENCRYPT	ENABLE_BLIT	ENABLE_HASH	ENABLE_CIPHER	ENABLE_MEMCOPY	CHAIN_CONTINUOUS	CHAIN	DECR_SEMAPHORE	INTERRUPT_ENABLE

Table 16-4. DCP Function Enable Bits

Hash	Cipher	Blit	Memcopy	Description
0	0	0	X	Simple memcopy operation.
0	0	1	0	Blit operation.
0	1	0	0	Simple encrypt/decrypt operation.
1	0	0	0	Simple hash. Only reads performed.
1	0	0	1	Memcopy and hash operation.
1	1	0	0	Hash with encryption/decryption.
All Others				Invalid setup.

The CHAIN bit should be set if the NEXT_COMMAND_ADDRESS field has a valid pointer to the next work packet. When set, this bit causes the channel to update its pointer to the next packet. The CHAIN_CONTINUOUS bit is similar, but it indicates that the next packet follows immediately after the current packet. This allows software to generate templates of operations without regard to the actual physical addresses used, which makes programming easier, especially in a virtual memory environment.

If the INTERRUPT_ENABLE bit is set, the channel generates an interrupt to the processor after completing the work for this packet. When the interrupt is issued, the packet will have been completely processed, including the update of the status/payload fields in the work packet.

Each channel contains an eight-bit counting semaphore that controls whether it is in the run or idle state. When the semaphore in non-zero, the channel is ready to run and process commands. Whenever a command finishes its operation, it checks the DECR_SEMAPHORE bit. If set, it decrements the counting semaphore. If the semaphore goes to 0 as a result, then the channel enters the idle state and remains there until the semaphore is incremented by software. When the semaphore goes to non-zero and the channel is in its idle state, it then uses the value in the NEXT_COMMAND_ADDRESS field to begin processing.

If the ENABLE_CIPHER bit is set, the data is encrypted or decrypted based on the CIPHER_ENCRYPT bit using the key/encryption settings from the Control1 field. The OTP_KEY and PAYLOAD_KEY indicate the source of the keys for the operation. If the OTP_KEY value is set, the KEY_SELECT field from the Control1 register indicates which OTP key is to be used. If the PAYLOAD_KEY bit is set, the first entry in the payload is the key to be used for the operation. If neither bit is set, the KEY_SELECT field indicates an index in the key RAM that contains the key to be used. (For cases when both the OTP_KEY and PAYLOAD_KEY bits are set, the PAYLOAD_KEY takes precedence.)

The HASH_ENABLE enables hashing of the data with the HASH_OUTPUT bit controlling whether the input data (HASH_OUTPUT=0) or output data (HASH_OUTPUT=1) is hashed. In addition, the hardware can be programmed to automatically check the hashed data if the HASH_CHECK bit is set. If the hash does not match, an interrupt is generated and the channel terminates operation on the current chain. The hashing algorithms also require two additional fields in order to operate properly. The HASH_INIT bit should be set for the first block in a hashing pass to properly initialize the hashing function. The HASH_TERM bit must be set on the final block of a hash to notify the hardware that the hashing operation is complete so that it can properly pad the tail of the buffer according to the hashing algorithm. This flag also triggers the write-back of the hash output to the work packet's payload area.

The CONSTANT_FILL flag may be used for both memcopy and blit operations. When this is set, the source address field is used instead as a constant that is written to all locations in the output buffer.

The WORDSWAP and BYTESWAP bits enable muxes around the FIFO to swap data to handle little-endian and big-endian storage of data in system memory. When these bits are cleared, data is assumed to be in little-endian format. When all bits are set, data is assumed to be in big-endian format.

The TAG field is used to identify the work packet and the associated completion status. As each packet is completed, the channel provides the status and tag information for the last packet processed.

16.2.5.4.3 Control1 Field

The Control1 field (shown in Table 16-5) provides additional values used when hashing or encrypting/decrypting data. For blit operations, this field indicates the number of bytes in a frame buffer line, which is used to calculate the address for successive lines in each blit operation.

Table 16-5. DCP Control1 Field

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CIPHER_CONFIG												HASH_ SELECT				KEY_SELECT								CIPHER_ MODE				CIPHER_ SELECT			
																FRAMEBUFFER_LENGTH (in bytes, blit mode)															

The CIPHER_SELECT field selects from one of sixteen possible encryption algorithms (0=AES128). The CIPHER_MODE selects the mode of operation for that cipher (0=ECB, 1=CBC).

The KEY_SELECT field allows key selection for one of several sources. Keys can be included in the packet payload or may come from OTP or write-only registers.

The HASH_SELECT field selects a hashing algorithm for the operation (0=SHA-1, 1=CRC32).

The CIPHER_CONFIG field provides optional configuration information for the selected cipher. An example would be a key length for the RC4 algorithm.

16.2.5.4.4 Source Buffer

The SOURCE_BUFFER pointer (shown in Table 16-6) specifies the location of the source buffer in memory. The buffer may reside at any byte alignment and should refer to an on-chip SRAM of off-chip SDRAM location. For optimal performance, buffers should be word aligned. When the CONSTANT_FILL flag is set in the Control0 field, the value in this field is used as the data written to all destination buffer locations.

Table 16-6. DCP Source Buffer Field

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SOURCE_BUFFER																															
CONSTANT (CONSTANT_FILL mode)																															

16.2.5.4.5 Destination Buffer

The DESTINATION_BUFFER (shown in Table 16-7) specifies the location of the destination buffer in on-chip SRAM or off-chip SDRAM memory and may be set to any byte alignment. For in-place encryption, the destination buffer and source buffer should be the same value. For optimal performance, the buffer location should be word-aligned.

Table 16-7. DCP Destination Buffer Field

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DESTINATION_BUFFER																															

16.2.5.4.6 Buffer Size Field

The BUFFER_SIZE field (shown in Table 16-8) indicates the size of the buffers for memcopy, encryption, and hashing modes. For memcopy and hashing operations, the value may be any number of bytes (byte granularity), and for encryption modes, the length must be a multiple of the selected encryption algorithm's natural data size (16 bytes for AES).

Table 16-8. DCP Buffer Size Field

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
NUMBER_LINES (blit mode)																BLIT_WIDTH (in bytes, blit mode)															
BUFFER_SIZE (in bytes, memcopy, encryption, hashing modes)																															

For blit operations, the buffer size field is split into two portions, a BLIT_WIDTH value, which specifies the number of bytes in each “line” of the blit operation, and a NUMBER_LINES value, which specifies how many vertical rows of pixels are in the image buffer.

16.2.5.4.7 Payload Pointer

Some operations require additional control values that are stored in a Payload Buffer (shown in Table 16-9), which is pointed to by this field. After the DCP reads the control packet, it examines the Control0 register and determines whether any payload information is required. If so, the DCP loads the payload from the address specified in this field. (See Section 16.2.5.4.9, “Payload” for more details.)

Table 16-9. DCP Payload Buffer Pointer

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PAYLOAD_POINTER																															

The payload area is also written to by the DCP at the completion of a hash operation (when the HASH_TERM) bit is set. Software must allocate the appropriate amount of storage (20 bytes for SHA-1 and 4 bytes for CRC32) in the payload or risk having the DCP write to an unallocated address.

16.2.5.4.8 Status

After the DCP engine has completed processing of a descriptor, it writes the packet status (shown in Table 16-10) back to the descriptor In the Status field. The packet status is the value of the channel status register at the time the packet completed processing.

Table 16-10. DCP Status Field

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TAG								ERROR_CODE																		ERROR_DST	ERROR_SRC	ERROR_PACKET	ERROR_SETUP	HASH_MISMATCH	COMPLETE

For various error conditions, the DCP encodes additional error information in the ERROR_CODE field. See Section 16.3, “Programmable Registers.” for more details on assignments of error codes. For any completion codes besides the COMPLETE flag, the channel suspends processing of the command chain until the error status values are cleared in the channel status register.

The format for this field is the same as for the channel status register, with the exception that the COMPLETE bit is written to the payload status but is not present in the channel status register.

16.2.5.4.9 Payload

The payload is a variable-length field that is used to provide key, initialization values, and expected results from hashing operations. The payload may consist of the data fields listed in Table 16-11.

Table 16-11. DCP Payload Field

Field Name	Size	Description	Condition
Cipher Key	16 bytes	AES key	Cipher enable with PAYLOAD_KEY
Cipher IV	16 bytes	CBC initialization vector	Cipher enable with CBC mode.
Hash Check	20 bytes	Hash completion value	Hash enabled with HASH_TERM fields set.

If fields are not used, they do not appear in the payload and the other payload values move upwards in the payload area. For instance, if only hashing is used, then the HASH_CHECK value would appear at offset 0 in the payload area. Table 16-12 should be used by software to determine the amount of payload to allocate and initialize.

Table 16-12. DCP Payload Allocation by Software

Control Bits Present			Payload Size
Hash_Check	Cipher_Init	Payload_Key	
0	0	0	0 words / 0 bytes
0	0	1	4 words / 16 bytes
0	1	0	4 words / 16 bytes
0	1	1	8 words / 32 bytes
1	0	0	5 words / 20 bytes
1	0	1	9 words / 36 bytes
1	1	0	9 words / 36 bytes
1	1	1	13 words / 52 bytes

For hashing operations, the DCP module writes the final hash value back to the start of the payload area for descriptors with the HASH_TERM bit set in the control packet. It is important that software allocate the required payload space, even though it is not required to set up the payload for control purposes.

16.2.6 Programming Other DCP Functions

16.2.6.1 Basic Memory Copy Programming Example

To perform a basic memcopy operation, only a single descriptor is required. The DCP simply copies data from the source to the destination buffer. This process is illustrated in Figure 16-6.

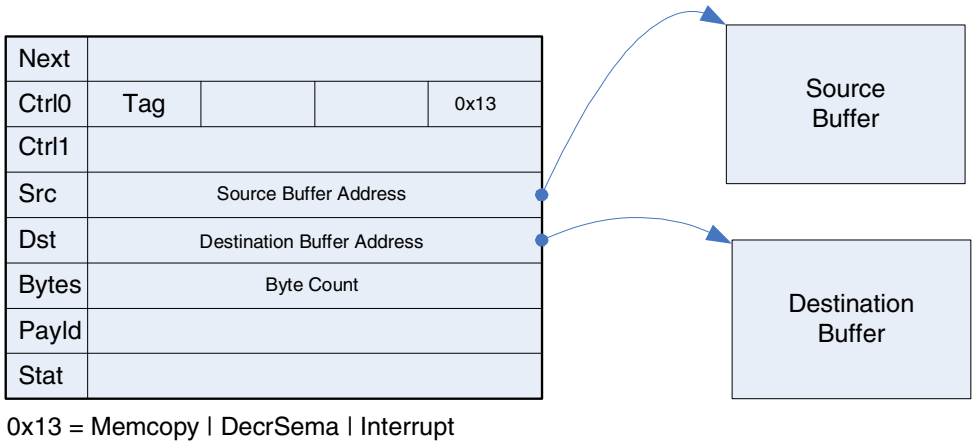


Figure 16-6. Basic Memory Copy Operation

```
<code>

typedef struct _dcp_descriptor
{
    u32              *next;
    hw_dcp_packet1_t  ctrl0;
    hw_dcp_packet2_t  ctrl1;
    u32              *src,
                     *dst,
                      buf_size,
                     *payload,
                      stat;
}  DCP_DESCRIPTOR;

  DCP_DESCRIPTOR dcp1;
  u32 *srcbuffer, *dstbuffer;

  // set up control packet
  dcp1.next = 0;                    // single packet in chain
  dcp1.ctrl0.U = 0;                 // clear ctrl0 field
  dcp1.ctrl0.B.ENABLE_MEMCOPY = 1; // enable memcopy
  dcp1.ctrl0.B.DECR_SEMAPHORE = 1; // decrement semaphore
  dcp1.ctrl0.B.INTERRUPT = 1;       // interrupt
  dcp1.ctrl1.U = 0;                 // clear ctrl1
  dcp1.src = srcbuffer;             // source buffer
  dcp1.dst = dstbuffer;             // destination buffer
  dcp1.buf_size = 512;              // 512 bytes
  dcp1.payload = NULL;              // not required
  dcp1.status = 0;                  // clear status

  // Enable channel 0
  HW_DCP_CHnCMDPTR_WR(0, dcp1);     // write packet address to pointer register
  HW_DCP_CHnSEMA_WR(0, 1);          // increment semaphore by 1

  // now wait for interrupt or poll
  // polling code
  while ( (HW_DCP_STAT_RD() & 0x01) == 0x00 );

  // now check/clear channel status
  if ( (HW_DCP_CHnSTAT_RD(0) & 0xFF) != 0 ) {
    // an error occurred
    HW_DCP_CHnSTAT_CLR(0, 0xff);
  }
  // clear interrupt register
  HW_DCP_STAT_CLR(1);

</code>
```

16.2.6.2 Basic Hash Operation Programming Example

To perform a basic hash operation, only a single descriptor is required. The DCP simply reads data from the source buffer and computes the hash value on the contents. This process is illustrated in Figure 16-7.

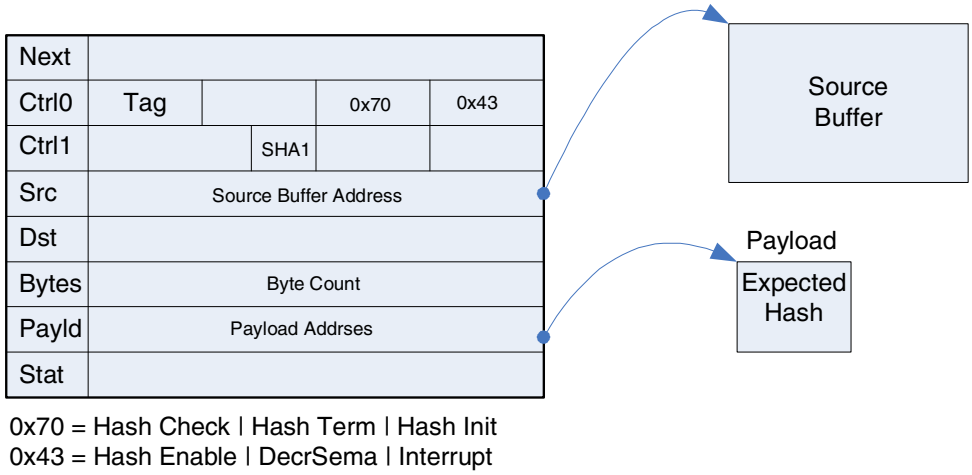


Figure 16-7. Basic Hash Operation

```
<code>

typedef struct _dcp_descriptor
{
    u32              *next;
    hw_dcp_packet1_t  ctrl0;
    hw_dcp_packet2_t  ctrl1;
    u32              *src,
                     *dst,
                      buf_size,
                     *payload,
                      stat;
}  DCP_DESCRIPTOR;

  DCP_DESCRIPTOR dcp1;
  u32 *srcbuffer;
  u32 payload[5];

  // set up expected hash check value
  payload[0]=0x01234567;
  payload[1]=0x89ABCDEF;
  payload[2]=0x00112233;
  payload[3]=0x44556677;
  payload[4]=0x8899aabb;

  // set up control packet
  dcp1.next = 0;                  // single packet in chain
  dcp1.ctrl0.U = 0;               // clear ctrl0 field
  dcp1.ctrl0.B.HASH_CHECK = 1;    // check hash when complete
  dcp1.ctrl0.B.HASH_INIT = 1;     // initialize hash with this block
  dcp1.ctrl0.B.HASH_TERM = 1;     // terminate hash with this block
  dcp1.ctrl0.B.ENABLE_HASH = 1;   // enable hash
  dcp1.ctrl0.B.DECR_SEMAPHORE = 1; // decrement semaphore
  dcp1.ctrl0.B.INTERRUPT = 1;     // interrupt
  dcp1.ctrl1.U = 0;               // clear ctrl1
  dcp1.src = srcbuffer;           // source buffer
  dcp1.dst = 0;                   // no destination buffer
  dcp1.buf_size = 512;            // 512 bytes
  dcp1.payload = payload;         // holds expected hash value
  dcp1.status = 0;                // clear status

  // Enable channel 0
  HW_DCP_CHnCMDPTR_WR(0, dcp1);   // write packet address to pointer register
  HW_DCP_CHnSEMA_WR(0, 1);        // increment semaphore by 1

  // now wait for interrupt or poll
  // polling code
  while ( (HW_DCP_STAT_RD() & 0x01) == 0x00 );

  // now check/clear channel status
  if ( (HW_DCP_CHnSTAT_RD(0) & 0xFF) != 0 ) {
    // an error occurred
    HW_DCP_CHnSTAT_CLR(0, 0xff);
  }
```

```
  // clear interrupt register
  HW_DCP_STAT_CLR(1);

</code>
```

16.2.6.3 Basic Cipher Operation Programming Example

To perform a basic cipher operation, only a single descriptor is required. The DCP reads data from the source buffer, encrypts it, and writes it to the destination buffer. For this example, the key is provided in the payload and the algorithm uses the AES CBC mode of operation. This requires a payload with both the key and CBC initialization value. This process is illustrated in Figure 16-8.

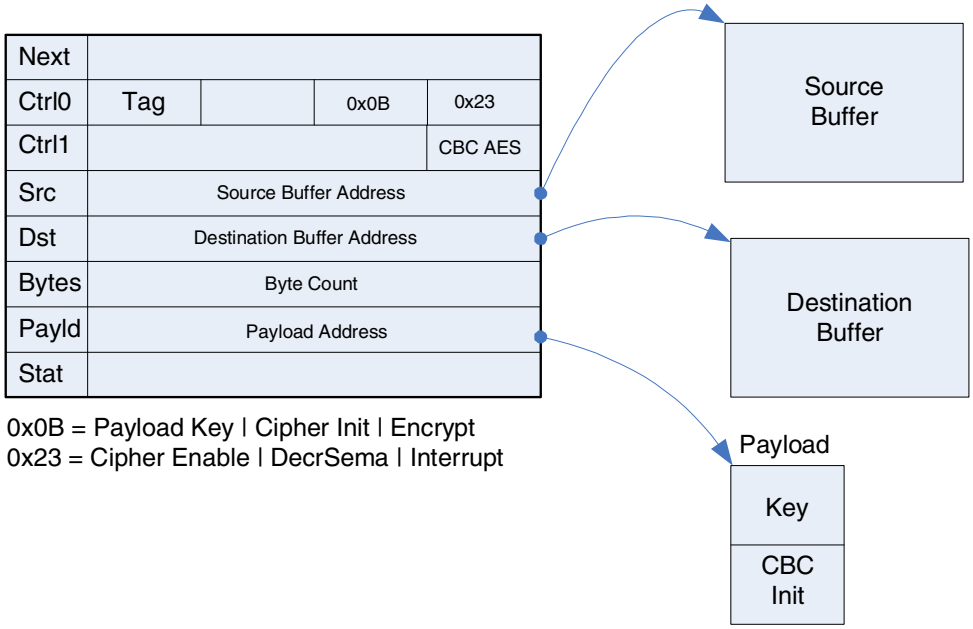


Figure 16-8. Basic Cipher Operation

```
<code>

typedef struct _dcp_descriptor
{
    u32              *next;
    hw_dcp_packet1_t  ctrl0;
    hw_dcp_packet2_t  ctrl1;
    u32              *src,
                     *dst,
                      buf_size,
                     *payload,
                      stat;
}  DCP_DESCRIPTOR;

  DCP_DESCRIPTOR dcp1;
  u32 *srcbuffer;
  u32 *dstbuffer;
  u32 payload[8];

  // set up key/CBC init in the payload
  payload[0]=0x01234567;           // key
  payload[1]=0x89ABCDEF;
  payload[2]=0xfedcba98;
  payload[3]=0x76543210;
  payload[4]=0x00112233;           // CBC initialization
  payload[5]=0x44556677;
  payload[6]=0x8899aabb;
  payload[7]=0xccddeeff;

  // set up control packet
```

```
  dcp1.next = 0;                   // single packet in chain
  dcp1.ctrl0.U = 0;                // clear ctrl0 field
  dcp1.ctrl0.B.PAYLOAD_KEY = 1;    // key is located in payload
  dcp1.ctrl0.B.CIPHER_ENCRYPT = 1; // encryption operation
  dcp1.ctrl0.B.CIPHER_INIT = 1;    // init CBC for this block (from payload)
  dcp1.ctrl0.B.ENABLE_CIPHER = 1;  // enable cipher
  dcp1.ctrl0.B.DECR_SEMAPHORE = 1; // decrement semaphore
  dcp1.ctrl0.B.INTERRUPT = 1;      // interrupt
  dcp1.ctrl1.U = 0;                // clear ctrl1
  dcp1.ctrl1.B.CIPHER_MODE = 1;    // select CBC mode of operation
  dcp1.src = srcbuffer;            // source buffer
  dcp1.dst = dstbuffer;            // destination buffer
  dcp1.buf_size = 512;             // 512 bytes
  dcp1.payload = payload;          // holds key/CBC init
  dcp1.status = 0;                 // clear status

  // Enable channel 0
  HW_DCP_CHnCMDPTR_WR(0, dcp1);    // write packet address to pointer register
  HW_DCP_CHnSEMA_WR(0, 1);         // increment semaphore by 1

  // now wait for interrupt or poll
  // polling code
  while ( (HW_DCP_STAT_RD() & 0x01) == 0x00 );

  // now check/clear channel status
  if ( (HW_DCP_CHnSTAT_RD(0) & 0xFF) != 0 ) {
    // an error occurred
    HW_DCP_CHnSTAT_CLR(0, 0xff);
  }
  // clear interrupt register
  HW_DCP_STAT_CLR(1);

</code>
```

16.2.6.4 Multi-Buffer Scatter/Gather Cipher and Hash Operation Programming Example

For this example, three separate buffers are encrypted and hashed with the results being directed to a unified buffer (gather operation). Three descriptors are used for the operation because there are three separate source buffer pointers. The DCP reads data from the source buffer and computes a hash on the unencrypted data. It then encrypts the data and writes it to the destination buffer. For this example, the key is located in the key RAM, and the algorithm uses the AES CBC mode of operation with a SHA-1 hash. The payload for the first operation contains the CBC initialization value, and the payload for the last packet contains the expected hash value. The middle packet requires no payload. This process is illustrated in Figure 16-9.

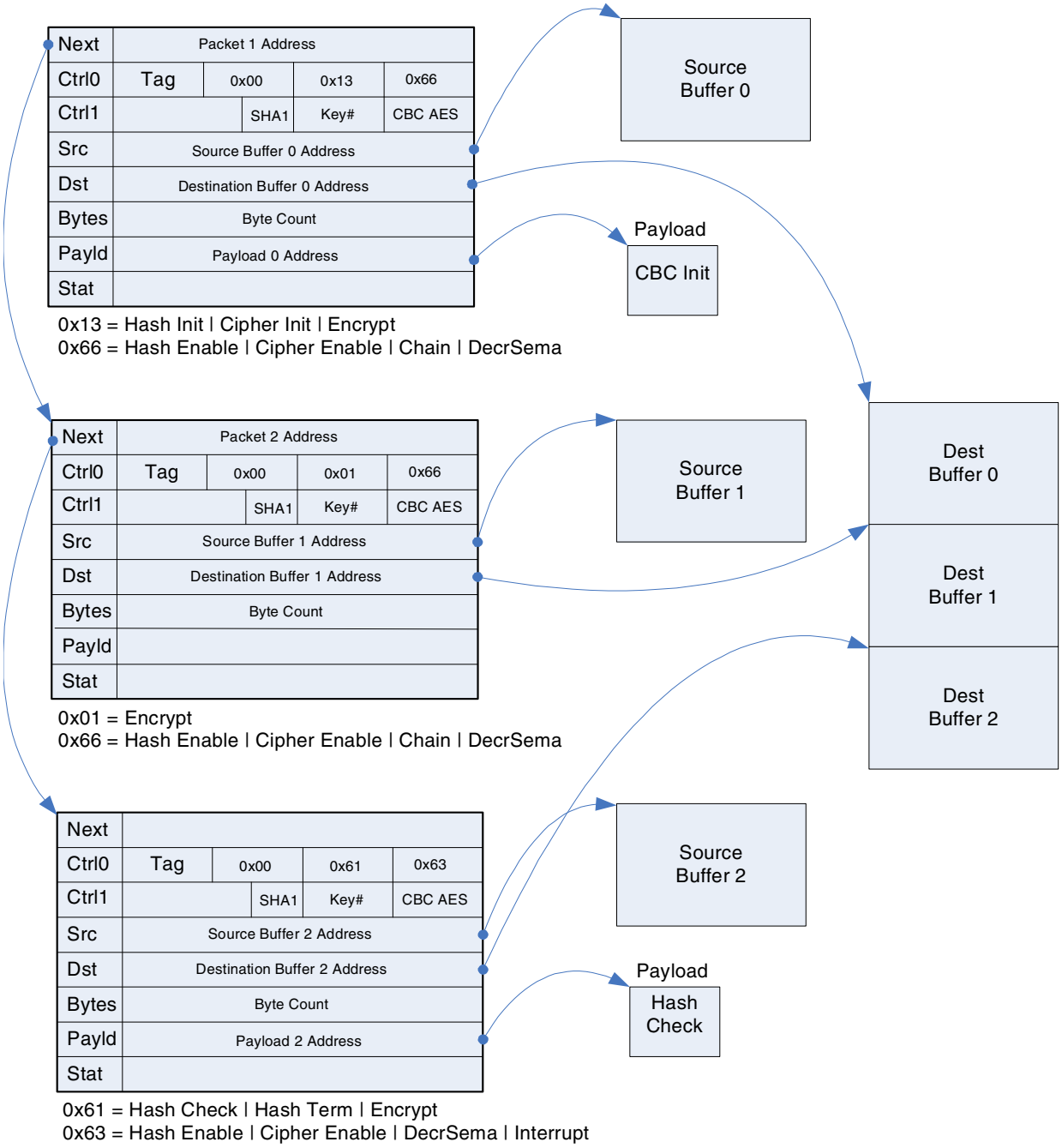


Figure 16-9. Multi-Buffer Scatter/Gather Cipher and Hash Operation

```
<code>

typedef struct _dcp_descriptor
{
    u32              *next;
    hw_dcp_packet1_t  ctrl0;
    hw_dcp_packet2_t  ctrl1;
    u32              *src,
                     *dst,
                      buf_size,
                     *payload,
                      stat;
}  DCP_DESCRIPTOR;
```

```
DCP_DESCRIPTOR dcp1, dcp2, dcp3;
u32 *srcbuffer0, *srcbuffer2, *srcbuffer3;
u32 *dstbuffer;
u32 payload0[4], payload2[5];

// set up CBC init in the payload
payload0[0]=0x01234567;           // key
payload0[1]=0x89ABCDEF;
payload0[2]=0xfedcba98;
payload0[3]=0x76543210;

payload2[0]=0x00112233;           // CBC initialization
payload2[1]=0x44556677;
payload2[2]=0x8899aabb;
payload2[3]=0xccddeeff;
payload2[3]=0xaabbccdd;

// set up control packet
dcp1.next = dcp2;                // point to packet 2
dcp1.ctrl0.U = 0;                // clear ctrl0 field
dcp1.ctrl0.B.CIPHER_ENCRYPT = 1; // encryption operation
dcp1.ctrl0.B.CIPHER_INIT = 1;    // init CBC for this block (from payload)
dcp1.ctrl0.B.HASH_INIT = 1;      // init hash this block
dcp1.ctrl0.B.ENABLE_CIPHER = 1;  // enable cipher
dcp1.ctrl0.B.ENABLE_HASH = 1;    // enable cipher
dcp1.ctrl0.B.DECR_SEMAPHORE = 1; // decrement semaphore
dcp1.ctrl0.B.CHAIN = 1;          // chain to next packet
dcp1.ctrl1.U = 0;                // clear ctrl1
dcp1.ctrl1.B.CIPHER_MODE = BV_DCP_PACKET2_CIPHER_MODE__CBC; //select CBC mode of operation
dcp1.ctrl1.B.HASH_SELECT = BV_DCP_PACKET2_HASH_SELECT__SHA1;    // select SHA1 hash
dcp1.ctrl1.B.KEY_SELECT  = 2;    // select key #2
dcp1.src = srcbuffer0;            // source buffer
dcp1.dst = dstbuffer;            // destination buffer
dcp1.buf_size = 512;             // 512 bytes
dcp1.payload = payload0;          // holds key/CBC init
dcp1.status = 0;                 // clear status

// set up control packet
dcp2.next = dcp3;                // point to packet 2
dcp2.ctrl0.U = 0;                // clear ctrl0 field
dcp2.ctrl0.B.CIPHER_ENCRYPT = 1; // encryption operation
dcp2.ctrl0.B.ENABLE_CIPHER = 1;  // enable cipher
dcp2.ctrl0.B.ENABLE_HASH = 1;    // enable cipher
dcp2.ctrl0.B.DECR_SEMAPHORE = 1; // decrement semaphore
dcp2.ctrl0.B.CHAIN = 1;          // chain to next packet
dcp2.ctrl1.U = 0;                // clear ctrl1
dcp2.ctrl1.B.CIPHER_MODE = BV_DCP_PACKET2_CIPHER_MODE__CBC; //select CBC mode of operation
dcp2.ctrl1.B.HASH_SELECT = BV_DCP_PACKET2_HASH_SELECT__SHA1;    // select SHA1 hash
dcp2.ctrl1.B.KEY_SELECT  = 2;    // select key #2
dcp2.src = srcbuffer1;           // source buffer
dcp2.dst = dstbuffer+512;        // destination buffer
dcp2.buf_size = 512;             // 512 bytes
dcp2.payload = NULL;             // no payload required
dcp2.status = 0;                 // clear status

// set up control packet
dcp3.next = dcp3;                // point to packet 2
dcp3.ctrl0.U = 0;                // clear ctrl0 field
dcp3.ctrl0.B.HASH_TERM      = 1; // terminate hash block
dcp3.ctrl0.B.HASH_CHECK     = 1; // check hash against payload value
dcp3.ctrl0.B.CIPHER_ENCRYPT = 1; // encryption operation
dcp3.ctrl0.B.ENABLE_CIPHER = 1;  // enable cipher
dcp3.ctrl0.B.ENABLE_HASH = 1;    // enable cipher
dcp3.ctrl0.B.DECR_SEMAPHORE = 1; // decrement semaphore
dcp3.ctrl0.B.INTERRUPT = 1;      // interrupt on completion
dcp3.ctrl1.U = 0;                // clear ctrl1
dcp3.ctrl1.B.CIPHER_MODE = BV_DCP_PACKET2_CIPHER_MODE__CBC; //select CBC mode of operation
dcp3.ctrl1.B.HASH_SELECT = BV_DCP_PACKET2_HASH_SELECT__SHA1;    // select SHA1 hash
dcp3.ctrl1.B.KEY_SELECT  = 2;    // select key #2
dcp3.src = srcbuffer1;           // source buffer
dcp3.dst = dstbuffer+1024;       // destination buffer
dcp3.buf_size = 512;             // 512 bytes
dcp3.payload = payload2;         // payload is hash check value
dcp3.status = 0;                 // clear status

// Enable channel 0
HW_DCP_CHnCMDPTR_WR(0, dcp1);    // write packet address to pointer register
HW_DCP_CHnSEMA_WR(0, 3);         // increment semaphore by 3 (for 3 packets)
```

```
  // now wait for interrupt or poll
  // polling code
  while ( (HW_DCP_STAT_RD() & 0x01) == 0x00 );

  // now check/clear channel status
  if ( (HW_DCP_CHnSTAT_RD(0) & 0xFF) != 0 ) {
    // an error occurred
    HW_DCP_CHnSTAT_CLR(0, 0xff);
  }
  // clear interrupt register
  HW_DCP_STAT_CLR(1);

</code>
```

16.3 Programmable Registers

The following registers are available for programmer access and control of the DCP.

16.3.1 DCP Control Register 0 Description

The CTRL register contains controls for the DCP module.

HW_DCP_CTRL	0x000
HW_DCP_CTRL_SET	0x004
HW_DCP_CTRL_CLR	0x008
HW_DCP_CTRL_TOG	0x00C

Table 16-13. HW_DCP_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	PRESENT_CRYPTO	RSVD3	RSVD2				GATHER_RESIDUAL_WRITES	ENABLE_CONTEXT_CACHING	ENABLE_CONTEXT_SWITCHING	RSVD1												RSVD0	CHANNEL_INTERRUPT_ENABLE							

Table 16-14. HW_DCP_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	Set this bit to zero to enable normal DCP operation. Set this bit to one (default) to disable clocking with the DCP and hold it in its reset (lowest power) state. This bit can be turned on and then off to reset the DCP block to its default state.
30	**CLKGATE**	RW	0x1	This bit must be set to zero for normal operation. When set to one it gates off the clocks to the block.

Table 16-14. HW_DCP_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
29	**PRESENT_CRYPTO**	RO	0x1	Indicates whether the crypto (Cipher/Hash) functions are present. Present = 0x1 Absent = 0x0
28	**RSVD3**	RO	0x000000	Reserved
27:24	**RSVD2**	RO	0x000000	Reserved, always set to zero.
23	**GATHER_RESIDUAL_WRITES**	RW	0x1	Software should set this bit to enable ragged writes to unaligned buffers to be gathered between multiple write operations. This improves performance by removing several byte operations between write bursts. Trailing byte writes are held in a residual write data buffer and combined with a subsequent write to the buffer to form a word write.
22	**ENABLE_CONTEXT_CACHING**	RW	0x0	Software should set this bit to enable caching of contexts between operations. If only a single channel is used for encryption/hashing, enabling caching causes the context to not be reloaded if the channel was the last to be used.
21	**ENABLE_CONTEXT_SWITCHING**	RW	0x0	Enable automatic context switching for the channels. Software should set this bit if more than one channel is doing hashing or cipher operations that require context to be saved (for instance, when CBC mode is enabled). By disabling context switching, software can save the 160 bytes used for the context buffer.
20:9	**RSVD1**	RO	0x000000	Reserved, always set to zero.
8	**RSVD0**	RO	0x000000	Reserved
7:0	**CHANNEL_INTERRUPT_ENABLE**	RW	0x0	Per-channel interrupt enable bit. When set, the channel's interrupt will get routed to the interrupt controller. Channel 0 is routed to the dcp_vmi_irq signal and the other channels are combined (along with the CRC interrupt) into the dcp_irq signal. CH0 = 0x01 CH1 = 0x02 CH2 = 0x04 CH3 = 0x08

DESCRIPTION:

The Control register contains the primary controls for the DCP block. The present bits indicate which of the sub-features of the block are present in the hardware. The context control bits control how the DCP utilizes it's context buffer and the gather residual writes bit controls how the master handles writing misaligned data to the bus. Each channel and the color-space converter contains an independent interrupt enable.

EXAMPLE:

```
HW_DCP_CTRL_SET(BM_DCP_CTRL_SFTRST);
HW_DCP_CTRL_CLR(BM_DCP_CTRL_SFTRST | BM_DCP_CTRL_CLKGATE);
```

16.3.2 DCP Status Register Description

The DCP Interrupt Status register provides channel interrupt status information.

HW_DCP_STAT	0x010
HW_DCP_STAT_SET	0x014

HW_DCP_STAT_CLR	0x018
HW_DCP_STAT_TOG	0x01C

Table 16-15. HW_DCP_STAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD3			OTP_KEY_READY	CUR_CHANNEL				READY_CHANNELS								RSVD2							RSVD1	RSVD0				IRQ			

Table 16-16. HW_DCP_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSVD3**	RO	0x000000	Reserved, always set to zero.
28	**OTP_KEY_READY**	RO	0x1	When set, indicates that the OTP key has been shifted from the fuse block and is ready for use.
27:24	**CUR_CHANNEL**	RO	0x0	Current (active) channel (encoded). None = 0x0 CH0 = 0x1 CH1 = 0x2 CH2 = 0x3 CH3 = 0x4 CSC = 0x8
23:16	**READY_CHANNELS**	RO	0x0	Indicates which channels are ready to proceed with a transfer (active channel also included). Each bit is a one-hot indicating the request status for the associated channel. CH0 = 0x01 CH1 = 0x02 CH2 = 0x04 CH3 = 0x08
15:9	**RSVD2**	RO	0x000000	Reserved, always set to zero.
8	**RSVD1**	RO	0x000000	Reserved
7:4	**RSVD0**	RO	0x000000	Reserved, always set to zero.
3:0	**IRQ**	RW	0x0	Indicates which channels have pending interrupt requests. Channel 0's interrupt is routed through the dcp_vmi_irq and the other interrupt bits are routed through the dcp_irq.

DESCRIPTION:

This register provides status feedback indicating the channel currently performing an operation and which channels have pending operations.

EXAMPLE:

```
HW_DCP_STAT_CLR(BM_DCP_STAT_CSCIRQ); // clear CSC interrupt
```

16.3.3 DCP Channel Control Register Description

The DCP Channel Control register provides controls for channel arbitration and channel enables.

HW_DCP_CHANNELCTRL	0x020
HW_DCP_CHANNELCTRL_SET	0x024
HW_DCP_CHANNELCTRL_CLR	0x028
HW_DCP_CHANNELCTRL_TOG	0x02C

Table 16-17. HW_DCP_CHANNELCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD													RSVD1		CH0_IRQ_MERGED	HIGH_PRIORITY_CHANNEL								ENABLE_CHANNEL							

Table 16-18. HW_DCP_CHANNELCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:19	**RSVD**	RO	0x0000	Reserved, always set to zero.
18:17	**RSVD1**	RO	0x000000	Reserved
16	**CH0_IRQ_MERGED**	RW	0x0	Indicates that the interrupt for channel 0 should be merged with the other interrupts on the shared dcp_irq interrupt. When set to 0, channel 0's interrupt will be routed to the separate dcp_vmi_irq. When set to 1, the interrupt will be routed to the shared DCP interrupt.
15:8	**HIGH_PRIORITY_CHANNEL**	RW	0x0	Setting a bit in this field causes the corresponding channel to have high-priority arbitration. High priority channels will be arbitrated round-robin and will take precedence over other channels that are not marked as high priority. CH0 = 0x01 CH1 = 0x02 CH2 = 0x04 CH3 = 0x08
7:0	**ENABLE_CHANNEL**	RW	0x0	Setting a bit in this field will enabled the DMA channel associated with it. This field is a direct input to the DMA channel arbiter. When not enabled, the channel is denied access to the central DMA resources. CH0 = 0x01 CH1 = 0x02 CH2 = 0x04 CH3 = 0x08

DESCRIPTION:

This register provides status feedback indicating the channel currently performing an operation and which channels have pending operations.

EXAMPLE:

```
BW_DCP_CHANNELCTRL_ENABLE_CHANNEL(BV_DCP_CHANNELCTRL_ENABLE_CHANNEL__CH0); // enable channel 0
```

16.3.4 DCP Capability 0 Register Description

This register contains additional information about the DCP module implementation parameters.

HW_DCP_CAPABILITY0 0x030

Table 16-19. HW_DCP_CAPABILITY0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DISABLE_DECRYPT	ENABLE_TZONE	RSVD																		NUM_CHANNELS				NUM_KEYS							

Table 16-20. HW_DCP_CAPABILITY0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**DISABLE_DECRYPT**	RW	0x0	Write to a 1 to disable decryption. This bit can only be written by secure software and the value can only be cleared by a reset.
30	**ENABLE_TZONE**	RW	0x0	Write to a 1 enable trustzone support. Channel operations initiated by secure operations will be protected from non-secure operations and the secure channel interrupts will be routed to the secure DCP interrupt. This bit can only be written by secure software and the value can only be cleared by a reset.
29:12	**RSVD**	RO	0x000000	Reserved, always set to zero.
11:8	**NUM_CHANNELS**	RO	0x4	Encoded value indicating the number of channels implemented in the design.
7:0	**NUM_KEYS**	RO	0x4	Encoded value indicating the number of key storage locations implemented in the design.

DESCRIPTION:

This register provides capability information for the DCP block. It indicates the number of channels implemented as well as the number of key storage locations available for software use.

EXAMPLE:

```
Empty Example
```

16.3.5 DCP Capability 1 Register Description

This register contains information about the algorithms available on the implementation.

HW_DCP_CAPABILITY1 0x040

Table 16-21. HW_DCP_CAPABILITY1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
HASH_ALGORITHMS																CIPHER_ALGORITHMS															

Table 16-22. HW_DCP_CAPABILITY1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**HASH_ALGORITHMS**	RO	0x1	One-hot field indicating which hashing algorithms are available. SHA1 = 0x0001 CRC32 = 0x0002
15:0	**CIPHER_ALGORITHMS**	RO	0x1	One-hot field indicating which cipher algorithms are available. AES128 = 0x0001

DESCRIPTION:

This register provides capability information for the DCP block. It contains two fields indicating which encryption and hashing algorithms are present in the design. Each bit set indicates that support for the associated function is present.

EXAMPLE:

```
Empty Example.
```

16.3.6 DCP Context Buffer Pointer Description

This register contains a pointer to the memory region to be used for DCP context swap operations.

HW_DCP_CONTEXT 0x050

Table 16-23. HW_DCP_CONTEXT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 16-24. HW_DCP_CONTEXT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x0	Context pointer address. Address should be located in system RAM and should be word-aligned for optimal performance.

DESCRIPTION:

This register contains a pointer to the start of the context pointer memory in on-chip SRAM or off-chip SDRAM. This buffer will be used to store state information when the DCP module changes from one channel to another.

EXAMPLE:

```
Empty Example.
```

16.3.7 DCP Key Index Description

This register contains a pointer to the key location to be written.

HW_DCP_KEY 0x060

Table 16-25. HW_DCP_KEY

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																								RSVD_INDEX		INDEX		RSVD_SUBWORD		SUBWORD	

Table 16-26. HW_DCP_KEY Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:8	**RSVD**	RO	0x000000	Reserved, always set to zero.
7:6	**RSVD_INDEX**	RO	0x000000	Reserved, always set to zero.
5:4	**INDEX**	RW	0x0	Key index pointer. Valid indices are 0-[number_keys].
3:2	**RSVD_SUBWORD**	RO	0x0	Reserved, always set to zero.
1:0	**SUBWORD**	RW	0x0	Key subword pointer. Valid indices are 0-3. After each write to the key data register, this field will increment.

DESCRIPTION:

The DCP module maintains a set of write-only keys that may be used by software. To write a key, software must first write the desired key index/subword to this register and then write the key values to the key registers (below). After each write to the key data register, the SUBWORD field will increment to allow software to write the subsequent key to be written without having to rewrite the key index.

EXAMPLE:

```
// write key 0 to 0x00112233_44556677_8899aabb_ccddeeff
HW_DCP_KEY_WR(BF_DCP_KEY_INDEX(0) | BF_DCP_KEY_SUBWORD(0)); // set key index to key 0, subword 0
HW_DCP_KEYDATA_WR(0xccddeeff); // write key values (subword 0)
HW_DCP_KEYDATA_WR(0x8899aabb); // write key values (subword 1)
HW_DCP_KEYDATA_WR(0x44556677); // write key values (subword 2)
HW_DCP_KEYDATA_WR(0x00112233); // write key values (subword 3)
```

16.3.8 DCP Key Data Description

This register provides write access to the key/key subword specified by the Key Index Register.

HW_DCP_KEYDATA 0x070

Table 16-27. HW_DCP_KEYDATA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DATA																															

Table 16-28. HW_DCP_KEYDATA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**DATA**	RW	0x0	Word 0 data for key. This is the least-significant word.

DESCRIPTION:

Writing this location updates the selected subword for the key located at the index specified by the Key Index Register. A write also triggers the SUBWORD field of the KEY register to increment to the next higher word in the key.

EXAMPLE:

```
// write key 0 to 0x00112233_44556677_8899aabb_ccddeeff
HW_DCP_KEY_WR(BF_DCP_KEY_INDEX(0) | BF_DCP_KEY_SUBWORD(0)); // set key index to key 0, subword 0
HW_DCP_KEYDATA_WR(0xccddeeff); // write key values (subword 0)
HW_DCP_KEYDATA_WR(0x8899aabb); // write key values (subword 1)
HW_DCP_KEYDATA_WR(0x44556677); // write key values (subword 2)
HW_DCP_KEYDATA_WR(0x00112233); // write key values (subword 3)
```

16.3.9 DCP Work Packet 0 Status Register Description

This register displays the values for the current work packet offset 0x00 (Next Command) field.

HW_DCP_PACKET0 0x080

Table 16-29. HW_DCP_PACKET0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 16-30. HW_DCP_PACKET0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RO	0x0	Next Pointer Register,

DESCRIPTION:

The Work Packet Status Registers show the contents of the currently executing packet. When the channels are inactive (or the CSC is active), the packet status register return 0. The register bits are fully documented here to document the packet structure in memory.

EXAMPLE:

```
Empty Example.
```

16.3.10 DCP Work Packet 1 Status Register Description

This register displays the values for the current work packet offset 0x04 (control) field.

HW_DCP_PACKET1 0x090

Table 16-31. HW_DCP_PACKET1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TAG								OUTPUT_WORDSWAP	OUTPUT_BYTESWAP	INPUT_WORDSWAP	INPUT_BYTESWAP	KEY_WORDSWAP	KEY_BYTESWAP	TEST_SEMA_IRQ	CONSTANT_FILL	HASH_OUTPUT	CHECK_HASH	HASH_TERM	HASH_INIT	PAYLOAD_KEY	OTP_KEY	CIPHER_INIT	CIPHER_ENCRYPT	ENABLE_BLIT	ENABLE_HASH	ENABLE_CIPHER	ENABLE_MEMCOPY	CHAIN_CONTIGUOUS	CHAIN	DECR_SEMAPHORE	INTERRUPT

Table 16-32. HW_DCP_PACKET1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**TAG**	RO	0x0	Packet Tag
23	**OUTPUT_WORDSWAP**	RO	0x0	Reflects whether the DCP engine will wordswap output data (big-endian data).
22	**OUTPUT_BYTESWAP**	RO	0x0	Reflects whether the DCP engine will byteswap output data (big-endian data).
21	**INPUT_WORDSWAP**	RO	0x0	Reflects whether the DCP engine will wordswap input data (big-endian data).
20	**INPUT_BYTESWAP**	RO	0x0	Reflects whether the DCP engine will byteswap input data (big-endian data).
19	**KEY_WORDSWAP**	RO	0x0	Reflects whether the DCP engine will swap key words (big-endian key).
18	**KEY_BYTESWAP**	RO	0x0	Reflects whether the DCP engine will swap key bytes (big-endian key).
17	**TEST_SEMA_IRQ**	RO	0x0	This bit is used to test the channel semaphore transition to 0. FOR TEST USE ONLY!
16	**CONSTANT_FILL**	RO	0x0	When this bit is set (MEMCOPY and BLIT modes only), the DCP will simply fill the destination buffer with the value found in the Source Address field.
15	**HASH_OUTPUT**	RO	0x0	When hashing is enabled, this bit controls whether the input or output data is hashed. INPUT = 0x00 OUTPUT = 0x01
14	**CHECK_HASH**	RO	0x0	Reflects whether the calculated hash value should be compared against the hash provided in the payload.
13	**HASH_TERM**	RO	0x0	Reflects whether the current hashing block is the final block in the hashing operation, so the hash padding should be applied by hardware.

Table 16-32. HW_DCP_PACKET1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
12	**HASH_INIT**	RO	0x0	Reflects whether the current hashing block is the initial block in the hashing operation, so the hash registers should be initialized before the operation.
11	**PAYLOAD_KEY**	RO	0x0	When set, indicates the payload contains the key. This bit takes precedence over the OTP_KEY control
10	**OTP_KEY**	RO	0x0	Reflects whether a hardware-based key should be used. The KEY_SELECT field from the Control1 field is used to select from multiple hardware keys. The PAYLOAD_KEY bit takes precedence over the OTP_KEY bit.
9	**CIPHER_INIT**	RO	0x0	Reflects whether the cipher block should load the initialization vector from the payload for this operation.
8	**CIPHER_ENCRYPT**	RO	0x0	When the cipher block is enabled, this bit indicates whether the operation is encryption or decryption. ENCRYPT = 0x01 DECRYPT = 0x00
7	**ENABLE_BLIT**	RO	0x0	Reflects whether the DCP should perform a blit operation. Source data is always continuous and the destination buffer is written in run/stride format. When set, the BUFFER_SIZE field is treated as two 16-bit values for the X-Y extents of the blit operation.
6	**ENABLE_HASH**	RO	0x0	Reflects whether the selected hashing function should be enabled for this operation.
5	**ENABLE_CIPHER**	RO	0x0	Reflects whether the selected cipher function should be enabled for this operation.
4	**ENABLE_MEMCOPY**	RO	0x0	Reflects whether the selected hashing function should be enabled for this operation.
3	**CHAIN_CONTIGUOUS**	RO	0x000000	Reflects whether the next packet's address is located following this packet's payload.
2	**CHAIN**	RO	0x0	Reflects whether the next command pointer register should be loaded into the channel's current descriptor pointer.
1	**DECR_SEMAPHORE**	RO	0x0	Reflects whether the channel's semaphore should be decremented at the end of the current operation. When the semaphore reaches a value of zero, no more operations will be issued from the channel.
0	**INTERRUPT**	RO	0x0	Reflects whether the channel should issue an interrupt upon completion of the packet.

DESCRIPTION:

This register shows the contents of the Control0 register from the packet being processed.

EXAMPLE:

```
Empty Example.
```

16.3.11 DCP Work Packet 2 Status Register Description

This register displays the values for the current work packet offset 0x08 (Control1) field.

HW_DCP_PACKET2 0x0A0

Table 16-33. HW_DCP_PACKET2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CIPHER_CFG								RSVD				HASH_SELECT				KEY_SELECT								CIPHER_MODE				CIPHER_SELECT			

Table 16-34. HW_DCP_PACKET2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**CIPHER_CFG**	RO	0x0	Cipher configuration bits. Optional configuration bits required for ciphers
23:20	**RSVD**	RO	0x0	Reserved, always set to zero.
19:16	**HASH_SELECT**	RO	0x0	Hash Selection Field SHA1 = 0x00 CRC32 = 0x01
15:8	**KEY_SELECT**	RO	0x0	Key Selection Field. The value here reflects the key index for the cipher operation.
7:4	**CIPHER_MODE**	RO	0x0	Cipher Mode Selection Field. Reflects the mode of operation for cipher operations. ECB = 0x00 CBC = 0x01
3:0	**CIPHER_SELECT**	RO	0x0	Cipher Selection Field AES128 = 0x00

DESCRIPTION:

This register shows the contents of the Control0 register from the packet being processed.

EXAMPLE:

```
Empty Example.
```

16.3.12 DCP Work Packet 3 Status Register Description

This register displays the values for the current work packet offset 0x0C (Source Address) field.

HW_DCP_PACKET3 0x0B0

Table 16-35. HW_DCP_PACKET3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 16-36. HW_DCP_PACKET3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RO	0x0	Source Buffer Address Pointer. This value is the working value and will update as the operation proceeds.

DESCRIPTION:

This register shows the contents of the Source Address register from the packet being processed. When the CONSTANT_FILL bit in the Control 0 field is set, this field contains the data written to the destination buffer.

EXAMPLE:

```
Empty Example.
```

16.3.13 DCP Work Packet 4 Status Register Description

This register displays the values for the current work packet offset 0x10 (Destination Address) field.

HW_DCP_PACKET4 0x0C0

Table 16-37. HW_DCP_PACKET4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 16-38. HW_DCP_PACKET4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RO	0x0	Destination Buffer Address Pointer. This value is the working value and will update as the operation proceeds.

DESCRIPTION:

This register shows the contents of the Destination Address register from the packet being processed.

EXAMPLE:

```
Empty Example.
```

16.3.14 DCP Work Packet 5 Status Register Description

This register displays the values for the current work packet offset 0x14 (Buffer Size) field.

HW_DCP_PACKET5 0x0D0

Table 16-39. HW_DCP_PACKET5

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

Table 16-40. HW_DCP_PACKET5 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COUNT**	RO	0x0	Byte Count register. This value is the working value and will update as the operation proceeds.

DESCRIPTION:

This register shows the contents of the bytecount register from the packet being processed. The field can be considered either a byte count

or a buffer size. The logic treats this as a decrmenting count of bytes from the buffer size programmed into the field. As the transaction

proceeds, the logic will decrement the bytecount as data is written to the destination buffer. For blit operations, the top 16-bits of this field

represents the number of lines (y size) in the blit and the lower 16-bits represent the number of bytes in a line (x size).

EXAMPLE:

```
Empty Example.
```

16.3.15 DCP Work Packet 6 Status Register Description

This register displays the values for the current work packet offset 0x1C (Payload Pointer) field.

HW_DCP_PACKET6 0x0E0

Table 16-41. HW_DCP_PACKET6

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 16-42. HW_DCP_PACKET6 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RO	0x0	This regiser reflects the payload pointer for the current control packet.

DESCRIPTION:

This register shows the contents of the payload pointer fieldr from the packet being processed.

EXAMPLE:

```
Empty Example.
```

16.3.16 DCP Channel 0 Command Pointer Address Register Description

The DCP channel 0 current command address register points to the multiword descriptor that is to be executed (or currently being executed). The channel may be activated by writing the command pointer address to a valid descriptor in memory and then updating the semaphore to a non-zero value. After the

engine completes processing of a descriptor, the "next_ptr" field from the descriptor is moved into this register to enable processing of the next descriptor. All channels with a non-zero semaphore value will arbitrate for access to the engine for the subsequent operation.

HW_DCP_CH0CMDPTR 0x100

Table 16-43. HW_DCP_CH0CMDPTR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 16-44. HW_DCP_CH0CMDPTR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x00000000	Pointer to descriptor structure to be processed for channel 0.

DESCRIPTION:

DCP Channel 0 is controlled by a variable sized command structure. This register points to the command structure to be executed.

EXAMPLE:

```
HW_DCP_CHnCMDPTR_WR(0, v); // Write channel 0 command pointer
pCurptr = (hw_DCP_chncmdptr_t *) HW_DCP_CHnCMDPTR_RD(0); // Read current command pointer
```

16.3.17 DCP Channel 0 Semaphore Register Description

The DCP Channel 0 semaphore register is used to synchronize the CPU instruction stream and the DMA chain processing state. After a command chain has been generated in memory, software should write the address of the first command descriptor to the CMDPTR register and then write a non-zero value to the semaphore register to indicate that the channel is active. Each command packet has a chaining bit which indicates that another descriptor should be loaded into the channel upon completion of the current descriptor. If the chaining bit is not set, the next address will not be loaded into the CMDPTR register. Each packet also contains a "decrement semaphore" bit, which indicates that the counting semaphore should be decremented after the operation. A channel is considered active when the semaphore is a non-zero value. When programming a series operations, software must properly program the semaphore values in conjuction with the "decrement_semaphore" bits in the control packets to ensure that the proper number of descriptors are activated. A semaphore may be cleared by software by writing 0xFF to the HW_DCP_CHnSEMA_CLR register. The logic will also clear the semaphore if an error has occurred.

HW_DCP_CH0SEMA 0x110

Table 16-45. HW_DCP_CH0SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								VALUE								RSVD1								INCREMENT							

Table 16-46. HW_DCP_CH0SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**VALUE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DCP hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DCP channel decrements the count on the same clock, then the count is incremented by a net one. The semaphore may be cleared by writing 0xFF to the HW_DCP_CHnSEMA_CLR register.

DESCRIPTION:

Each DCP channel has an 8 bit counting semaphore that is used to synchronize between the program stream and and the DCP chain processing. After processing each control packet, the DCP decrements the semaphore if it is non-zero. The channel will continue processing packets as long as the semaphore contains a non-zero value and the CHAIN or CHAIN_CONTIGOUS control bits in the Control0 field are set.

EXAMPLE:

```
Empty Example.
```

16.3.18 DCP Channel 0 Status Register Description

The DCP Channel 0 Interrupt Status register contains the interrupt status bit and the tag of the last completed operation from the command chain. If an error occurs during processing, the ERROR bit is set and an interrupt is generated.

HW_DCP_CH0STAT	0x120
HW_DCP_CH0STAT_SET	0x124
HW_DCP_CH0STAT_CLR	0x128
HW_DCP_CH0STAT_TOG	0x12C

Table 16-47. HW_DCP_CH0STAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TAG								ERROR_CODE								RSVD3								RSVD2	ERROR_DST	ERROR_SRC	ERROR_PACKET	ERROR_SETUP	HASH_MISMATCH	RSVD1	

Table 16-48. HW_DCP_CH0STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**TAG**	RO	0x00	Indicates the tag from the last completed packet in the command structure
23:16	**ERROR_CODE**	RW	0x0	Indicates additional error codes for some error conditions. NEXT_CHAIN_IS_0 = 0x01 Error signalled because the next pointer is 0x00000000 NO_CHAIN = 0x02 Error signalled because the semaphore is nonzero and neither chain bit is set CONTEXT_ERROR = 0x03 Error signalled because an error was reported reading/writing the context buffer PAYLOAD_ERROR = 0x04 Error signalled because an error was reported reading/writing the payload INVALID_MODE = 0x05 Error signalled because the control packet specifies an invalid mode select (for instance, blit + hash)
15:7	**RSVD3**	RO	0x0000	Reserved, always set to zero.
6	**RSVD2**	RW	0x0	Program this field to 0x0.
5	**ERROR_DST**	RW	0x0	This bit indicates a bus error occurred when storing to the destination buffer. When an error is detected, the channel's processing will stop until the error handled by software.
4	**ERROR_SRC**	RW	0x0	This bit indicates a bus error occurred when reading from the source buffer. When an error is detected, the channel's processing will stop until the error handled by software.
3	**ERROR_PACKET**	RW	0x0	This bit indicates that a a bus error occurred when reading the packet or payload or when writing status back to the packet payload. When an error is detected, the channel's processing will stop until the error is handled by software.
2	**ERROR_SETUP**	RW	0x0	This bit indicates that the hardware has detected an invalid programming configuration such as a buffer length that is not a multiple of the natural data size for the operation. When an error is detected, the channel's processing will stop until the error is handled by software.

Table 16-48. HW_DCP_CH0STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
1	**HASH_MISMATCH**	RW	0x0	The bit indicates that a hashing check operation mismatched for control packets that enable the HASH_CHECK bit. When an error is detected, the channel's processing will stop until the error is handled by software.
0	**RSVD1**	RO	0x0	This bit will always read 0 in the status register, but will be set to 1 in the packet status field after processing of the packet has completed. This was done so that software can verify that each packet completed properly in a chain of commands for cases when an interrupt is issued only for the last item in a packet. The completion bit for the channel is effectively the channel interrupt status bit.

DESCRIPTION:

The interrupt status register is updated at the end of each work packet. If the interrupt bit is set in the command packet's command field, an interrupt will be generated once the packet has completed. In addition, the tag value from the command is stored in the TAG field so that software can identify which command structure was the last to complete. If an error occurs, the ERROR bit is set and processing of the command chain is halted.

EXAMPLE:

```
Empty Example.
```

16.3.19 DCP Channel 0 Options Register Description

The DCP Channel 0 Options Status register contains optional control information that may be used to further tune the behavior of the channel.

HW_DCP_CH0OPTS	0x130
HW_DCP_CH0OPTS_SET	0x134
HW_DCP_CH0OPTS_CLR	0x138
HW_DCP_CH0OPTS_TOG	0x13C

Table 16-49. HW_DCP_CH0OPTS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																RECOVERY_TIMER															

Table 16-50. HW_DCP_CH0OPTS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSVD**	RO	0x0	Reserved, always set to zero.
15:0	**RECOVERY_TIMER**	RW	0x0	This field indicates the recovery time for the channel. After each operation, the recover timer for the channel is initiallized with this value and then decremented until the timer reaches zero. The channel will not initiate another operation for the next packet in the chain until the recovery time has been satisfied. The timebase for the recovery timer is 16 HCLK clock cycles, providing a range of 0ns to 8.3ms at 133 MHz operation.

DESCRIPTION:

The options register can be used to control optional features of the channels.

EXAMPLE:

```
Empty Example.
```

16.3.20 DCP Channel 1 Command Pointer Address Register Description

The DCP channel 1 current command address register points to the multiword descriptor that is to be executed (or currently being executed). The channel may be activated by writing the command pointer address to a valid descriptor in memory and then updating the semaphore to a non-zero value. After the engine completes processing of a descriptor, the "next_ptr" field from the descriptor is moved into this register to enable processing of the next descriptor. All channels with a non-zero semaphore value will arbitrate for access to the engine for the subsequent operation.

HW_DCP_CH1CMDPTR 0x140

Table 16-51. HW_DCP_CH1CMDPTR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 16-52. HW_DCP_CH1CMDPTR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x00000000	Pointer to descriptor structure to be processed for channel 1.

DESCRIPTION:

DCP Channel 1 is controlled by a variable sized command structure. This register points to the command structure to be executed.

EXAMPLE:

```
HW_DCP_CHn_CMDPTR_WR(1, v); // Write channel 1 command pointer
pCurptr = (hw_DCP_chn_cmdptr_t *) HW_DCP_CHn_CMDPTR_RD(1); // Read current command pointer
```

16.3.21 DCP Channel 1 Semaphore Register Description

The DCP Channel 1 semaphore register is used to synchronize the CPU instruction stream and the DMA chain processing state. After a command chain has been generated in memory, software should write the address of the first command descriptor to the CMDPTR register and then write a non-zero value to the semaphore register to indicate that the channel is active. Each command packet has a chaining bit which indicates that another descriptor should be loaded into the channel upon completion of the current descriptor. If the chaining bit is not set, the next address will not be loaded into the CMDPTR register. Each packet also contains a "decrement semaphore" bit, which indicates that the counting semaphore should be decremented after the operation. A channel is considered active when the semaphore is a non-zero value. When programming a series operations, software must properly program the semaphore values in conjuction with the "decrement_semaphore" bits in the control packets to ensure that the proper number of descriptors are activated. A semaphore may be cleared by software by writing 0xFF to the HW_DCP_CHnSEMA_CLR register. The logic will also clear the semaphore if an error has occurred.

HW_DCP_CH1SEMA 0x150

Table 16-53. HW_DCP_CH1SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								VALUE								RSVD1								INCREMENT							

Table 16-54. HW_DCP_CH1SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**VALUE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DCP hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DCP channel decrements the count on the same clock, then the count is incremented by a net one. The semaphore may be cleared by writing 0xFF to the HW_DCP_CHnSEMA_CLR register.

DESCRIPTION:

Each DCP channel has an 8 bit counting semaphore that is used to synchronize between the program stream and and the DCP chain processing. DCP processing continues until the engine attempts to decrement a semaphore that has already reached a value of zero. When the attempt is made, the DCP channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty Example.
```

16.3.22 DCP Channel 1 Status Register Description

The DCP Channel 1 Interrupt Status register contains the interrupt status bit and the tag of the last completed operation from the command chain. If an error occurs during processing, the ERROR bit is set and an interrupt is generated.

HW_DCP_CH1STAT	0x160
HW_DCP_CH1STAT_SET	0x164
HW_DCP_CH1STAT_CLR	0x168
HW_DCP_CH1STAT_TOG	0x16C

Table 16-55. HW_DCP_CH1STAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TAG								ERROR_CODE								RSVD3									RSVD2	ERROR_DST	ERROR_SRC	ERROR_PACKET	ERROR_SETUP	HASH_MISMATCH	RSVD1

Table 16-56. HW_DCP_CH1STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**TAG**	RO	0x00	Indicates the tag from the last completed packet in the command structure
23:16	**ERROR_CODE**	RW	0x0	Indicates additional error codes for some error conditions. NEXT_CHAIN_IS_0 = 0x01 Error signalled because the next pointer is 0x00000000 NO_CHAIN = 0x02 Error signalled because the semaphore is nonzero and neither chain bit is set CONTEXT_ERROR = 0x03 Error signalled because an error was reported reading/writing the context buffer PAYLOAD_ERROR = 0x04 Error signalled because an error was reported reading/writing the payload INVALID_MODE = 0x05 Error signalled because the control packet specifies an invalid mode select (for instance, blit + hash)
15:7	**RSVD3**	RO	0x0000	Reserved, always set to zero.
6	**RSVD2**	RW	0x0	Program this field to 0x0.
5	**ERROR_DST**	RW	0x0	This bit indicates a bus error occurred when storing to the destination buffer. When an error is detected, the channel's processing will stop until the error handled by software.
4	**ERROR_SRC**	RW	0x0	This bit indicates a bus error occurred when reading from the source buffer. When an error is detected, the channel's processing will stop until the error handled by software.

Table 16-56. HW_DCP_CH1STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
3	**ERROR_PACKET**	RW	0x0	This bit indicates that a bus error occurs when reading the packet or payload or when writing status back to the packet paylaod. When an error is detected, the channel's processing will stop until the error is handled by software.
2	**ERROR_SETUP**	RW	0x0	This bit indicates that the hardware detected an invalid programming configuration such as a buffer length that is not a multiple of the natural data size for the operation. When an error is detected, the channel's processing will stop until the error is handled by software.
1	**HASH_MISMATCH**	RW	0x0	The bit indicates that a hashing check operation mismatched for control packets that enable the HASH_CHECK bit. When an error is detected, the channel's processing will stop until the error is handled by software.
0	**RSVD1**	RO	0x0	This bit will always read 0 in the status register, but will be set to 1 in the packet status field after processing of the packet has completed. This was done so that software can verify that each packet completed properly in a chain of commands for cases when an interrupt is issued only for the last item in a packet. The completion bit for the channel is effectively the channel interrupt status bit.

DESCRIPTION:

The interrupt status register is updated at the end of each work packet. If the interrupt bit is set in the command packet's command field, an interrupt will be generated once the packet has completed. In addition, the tag value from the command is stored in the TAG field so that software can identify which command structure was the last to complete. If an error occurs, the ERROR bit is set and processing of the command chain is halted.

EXAMPLE:

```
Empty Example.
```

16.3.23 DCP Channel 1 Options Register Description

The DCP Channel 1 Options Status register contains optional control information that may be used to further tune the behavior of the channel.

HW_DCP_CH1OPTS	0x170
HW_DCP_CH1OPTS_SET	0x174
HW_DCP_CH1OPTS_CLR	0x178
HW_DCP_CH1OPTS_TOG	0x17C

Table 16-57. HW_DCP_CH1OPTS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																RECOVERY_TIMER															

Table 16-58. HW_DCP_CH1OPTS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSVD**	RO	0x0	Reserved, always set to zero.
15:0	**RECOVERY_TIMER**	RW	0x0	This field indicates the recovery time for the channel. After each operation, the recover timer for the channel is initiallized with this value and then decremented until the timer reaches zero. The channel will not initiate operation on the next packet in the chain until the recovery time has been satisfied. The timebase for the recovery timer is 16 HCLK clock cycles, providing a range of 0ns to 8.3ms at 133 MHz operation.

DESCRIPTION:

The options register can be used to control optional features of the channels.

EXAMPLE:

```
Empty Example.
```

16.3.24 DCP Channel 2 Command Pointer Address Register Description

The DCP channel 2 current command address register points to the multiword descriptor that is to be executed (or currently being executed). The channel may be activated by writing the command pointer address to a valid descriptor in memory and then updating the semaphore to a non-zero value. After the engine completes processing of a descriptor, the "next_ptr" field from the descriptor is moved into this register to enable processing of the next descriptor. All channels with a non-zero semaphore value will arbitrate for access to the engine for the subsequent operation.

HW_DCP_CH2CMDPTR 0x180

Table 16-59. HW_DCP_CH2CMDPTR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 16-60. HW_DCP_CH2CMDPTR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x00000000	Pointer to descriptor structure to be processed for channel 2.

DESCRIPTION:

DCP Channel 2 is controlled by a variable sized command structure. This register points to the command structure to be executed.

EXAMPLE:

```
HW_DCP_CHn_CMDPTR_WR(2, v); // Write channel 2 command pointer
pCurptr = (hw_DCP_chn_cmdptr_t *) HW_DCP_CHn_CMDPTR_RD(2); // Read current command pointer
```

16.3.25 DCP Channel 2 Semaphore Register Description

The DCP Channel 2 semaphore register is used to synchronize the CPU instruction stream and the DMA chain processing state. After a command chain has been generated in memory, software should write the address of the first command descriptor to the CMDPTR register and then write a non-zero value to the semaphore register to indicate that the channel is active. Each command packet has a chaining bit which indicates that another descriptor should be loaded into the channel upon completion of the current descriptor. If the chaining bit is not set, the next address will not be loaded into the CMDPTR register. Each packet also contains a "decrement semaphore" bit, which indicates that the counting semaphore should be decremented after the operation. A channel is considered active when the semaphore is a non-zero value. When programming a series operations, software must properly program the semaphore values in conjuction with the "decrement_semaphore" bits in the control packets to ensure that the proper number of descriptors are activated. A semaphore may be cleared by software by writing 0xFF to the HW_DCP_CHnSEMA_CLR register. The logic will also clear the semaphore if an error has occurred.

HW_DCP_CH2SEMA 0x190

Table 16-61. HW_DCP_CH2SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								VALUE								RSVD1								INCREMENT							

Table 16-62. HW_DCP_CH2SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**VALUE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.

Table 16-62. HW_DCP_CH2SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DCP hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DCP channel decrements the count on the same clock, then the count is incremented by a net one. The semaphore may be cleared by writing 0xFF to the HW_DCP_CHnSEMA_CLR register.

DESCRIPTION:

Each DCP channel has an 8 bit counting semaphore that is used to synchronize between the program stream and and the DCP chain processing. DCP processing continues until the engine attempts to decrement a semaphore that has already reached a value of zero. When the attempt is made, the DCP channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty Example.
```

16.3.26 DCP Channel 2 Status Register Description

The DCP Channel 2 Interrupt Status register contains the interrupt status bit and the tag of the last completed operation from the command chain. If an error occurs during processing, the ERROR bit is set and an interrupt is generated.

HW_DCP_CH2STAT	0x1A0
HW_DCP_CH2STAT_SET	0x1A4
HW_DCP_CH2STAT_CLR	0x1A8
HW_DCP_CH2STAT_TOG	0x1AC

Table 16-63. HW_DCP_CH2STAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TAG								ERROR_CODE								RSVD3								RSVD2	ERROR_DST	ERROR_SRC	ERROR_PACKET	ERROR_SETUP	HASH_MISMATCH	RSVD1	

Table 16-64. HW_DCP_CH2STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**TAG**	RO	0x00	Indicates the tag from the last completed packet in the command structure
23:16	**ERROR_CODE**	RW	0x0	Indicates additional error codes for some error conditions. NEXT_CHAIN_IS_0 = 0x01 Error signalled because the next pointer is 0x00000000 NO_CHAIN = 0x02 Error signalled because the semaphore is nonzero and neither chain bit is set CONTEXT_ERROR = 0x03 Error signalled because an error was reported reading/writing the context buffer PAYLOAD_ERROR = 0x04 Error signalled because an error was reported reading/writing the payload INVALID_MODE = 0x05 Error signalled because the control packet specifies an invalid mode select (for instance, blit + hash)
15:7	**RSVD3**	RO	0x0000	Reserved, always set to zero.
6	**RSVD2**	RW	0x0	Program this field to 0x0.
5	**ERROR_DST**	RW	0x0	This bit indicates a bus error occurred when storing to the destination buffer. When an error is detected, the channel's processing will stop until the error handled by software.
4	**ERROR_SRC**	RW	0x0	This bit indicates a bus error occurred when reading from the source buffer. When an error is detected, the channel's processing will stop until the error handled by software.
3	**ERROR_PACKET**	RW	0x0	This bit indicates that a bus error occurred when reading the packet or payload or when writing status back to the packet paylaod. When an error is detected, the channel's processing will stop until the error is handled by software.
2	**ERROR_SETUP**	RW	0x0	This bit indicates that the hardware detected an invalid programming configuration such as a buffer length that is not a multiple of the natural data size for the operation. When an error is detected, the channel's processing will stop until the error is handled by software.
1	**HASH_MISMATCH**	RW	0x0	The bit indicates that a hashing check operation mismatched for control packets that enable the HASH_CHECK bit. When an error is detected, the channel's processing will stop until the error is handled by software.
0	**RSVD1**	RO	0x0	This bit will always read 0 in the status register, but will be set to 1 in the packet status field after processing of the packet has completed. This was done so that software can verify that each packet completed properly in a chain of commands for cases when an interrupt is issued only for the last item in a packet. The completion bit for the channel is effectively the channel interrupt status bit.

DESCRIPTION:

The interrupt status register is updated at the end of each work packet. If the interrupt bit is set in the command packet's command field, an interrupt will be generated once the packet has completed. In addition, the tag value from the command is stored in the TAG field so that software can identify which

command structure was the last to complete. If an error occurs, the ERROR bit is set and processing of the command chain is halted.

EXAMPLE:

```
Empty Example.
```

16.3.27 DCP Channel 2 Options Register Description

The DCP Channel 2 Options Status register contains optional control information that may be used to further tune the behavior of the channel.

HW_DCP_CH2OPTS	0x1B0
HW_DCP_CH2OPTS_SET	0x1B4
HW_DCP_CH2OPTS_CLR	0x1B8
HW_DCP_CH2OPTS_TOG	0x1BC

Table 16-65. HW_DCP_CH2OPTS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																RECOVERY_TIMER															

Table 16-66. HW_DCP_CH2OPTS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSVD**	RO	0x0	Reserved, always set to zero.
15:0	**RECOVERY_TIMER**	RW	0x0	This field indicates the recovery time for the channel. After each operation, the recover timer for the channel is initiallized with this value and then decremented until the timer reaches zero. The channel will not initiate operation on the next packet in the chain until the recovery time has been satisfied. The timebase for the recovery timer is 16 HCLK clock cycles, providing a range of 0ns to 8.3ms at 133 MHz operation.

DESCRIPTION:

The options register can be used to control optional features of the channels.

EXAMPLE:

```
Empty Example.
```

16.3.28 DCP Channel 3 Command Pointer Address Register Description

The DCP channel 3 current command address register points to the multiword descriptor that is to be executed (or currently being executed). The channel may be activated by writing the command pointer address to a valid descriptor in memory and then updating the semaphore to a non-zero value. After the engine completes processing of a descriptor, the "next_ptr" field from the descriptor is moved into this register to enable processing of the next descriptor. All channels with a non-zero semaphore value will arbitrate for access to the engine for the subsequent operation.

HW_DCP_CH3CMDPTR 0x1C0

Table 16-67. HW_DCP_CH3CMDPTR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 16-68. HW_DCP_CH3CMDPTR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x00000000	Pointer to descriptor structure to be processed for channel 3.

DESCRIPTION:

DCP Channel 3 is controlled by a variable sized command structure. This register points to the command structure to be executed.

EXAMPLE:

```
HW_DCP_CHn_CMDPTR_WR(3, v); // Write channel 3 command pointer
pCurptr = (hw_DCP_chn_cmdptr_t *) HW_DCP_CHn_CMDPTR_RD(3); // Read current command pointer
```

16.3.29 DCP Channel 3 Semaphore Register Description

The DCP Channel 3 semaphore register is used to synchronize the CPU instruction stream and the DMA chain processing state. After a command chain has been generated in memory, software should write the address of the first command descriptor to the CMDPTR register and then write a non-zero value to the semaphore register to indicate that the channel is active. Each command packet has a chaining bit which indicates that another descriptor should be loaded into the channel upon completion of the current descriptor. If the chaining bit is not set, the next address will not be loaded into the CMDPTR register. Each packet also contains a "decrement semaphore" bit, which indicates that the counting semaphore should be decremented after the operation. A channel is considered active when the semaphore is a non-zero value. When programming a series operations, software must properly program the semaphore values in conjuction with the "decrement_semaphore" bits in the control packets to ensure that the proper number of descriptors are activated. A semaphore may be cleared by software by writing 0xFF to the HW_DCP_CHnSEMA_CLR register. The logic will also clear the semaphore if an error has occurred.

HW_DCP_CH3SEMA 0x1D0

Table 16-69. HW_DCP_CH3SEMA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2								VALUE								RSVD1								INCREMENT							

Table 16-70. HW_DCP_CH3SEMA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD2**	RO	0x0	Reserved, always set to zero.
23:16	**VALUE**	RO	0x0	This read-only field shows the current (instantaneous) value of the semaphore counter.
15:8	**RSVD1**	RO	0x0	Reserved, always set to zero.
7:0	**INCREMENT**	RW	0x00	The value written to this field is added to the semaphore count in an atomic way such that simultaneous software adds and DCP hardware substracts happening on the same clock are protected. This bit field reads back a value of 0x00. Writing a value of 0x02 increments the semaphore count by two, unless the DCP channel decrements the count on the same clock, then the count is incremented by a net one. The semaphore may be cleared by writing 0xFF to the HW_DCP_CHnSEMA_CLR register.

DESCRIPTION:

Each DCP channel has an 8 bit counting semaphore that is used to synchronize between the program stream and and the DCP chain processing. DCP processing continues until the engine attempts to decrement a semaphore that has already reached a value of zero. When the attempt is made, the DCP channel is stalled until software increments the semaphore count.

EXAMPLE:

```
Empty Example.
```

16.3.30 DCP Channel 3 Status Register Description

The DCP Channel 3 Interrupt Status register contains the interrupt status bit and the tag of the last completed operation from the command chain. If an error occurs during processing, the ERROR bit is set and an interrupt is generated.

HW_DCP_CH3STAT	0x1E0
HW_DCP_CH3STAT_SET	0x1E4
HW_DCP_CH3STAT_CLR	0x1E8
HW_DCP_CH3STAT_TOG	0x1EC

Table 16-71. HW_DCP_CH3STAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TAG								ERROR_CODE								RSVD3								RSVD2	ERROR_DST	ERROR_SRC	ERROR_PACKET	ERROR_SETUP	HASH_MISMATCH	RSVD1	

Table 16-72. HW_DCP_CH3STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**TAG**	RO	0x00	Indicates the tag from the last completed packet in the command structure
23:16	**ERROR_CODE**	RW	0x0	Indicates additional error codes for some error conditions. NEXT_CHAIN_IS_0 = 0x01 Error signalled because the next pointer is 0x00000000 NO_CHAIN = 0x02 Error signalled because the semaphore is nonzero and neither chain bit is set CONTEXT_ERROR = 0x03 Error signalled because an error was reported reading/writing the context buffer PAYLOAD_ERROR = 0x04 Error signalled because an error was reported reading/writing the payload INVALID_MODE = 0x05 Error signalled because the control packet specifies an invalid mode select (for instance, blit + hash)
15:7	**RSVD3**	RO	0x0000	Reserved, always set to zero.
6	**RSVD2**	RW	0x0	Program this field to 0x0.
5	**ERROR_DST**	RW	0x0	This bit indicates a bus error occurred when storing to the destination buffer. When an error is detected, the channel's processing will stop until the error handled by software.
4	**ERROR_SRC**	RW	0x0	This bit indicates a bus error occurred when reading from the source buffer. When an error is detected, the channel's processing will stop until the error handled by software.
3	**ERROR_PACKET**	RW	0x0	This bit indicates that a bus error occurred when reading the packet or payload or when writing status back to the packet paylaod. When an error is detected, the channel's processing will stop until the error is handled by software.
2	**ERROR_SETUP**	RW	0x0	This bit indicates that the hardware detected an invalid programming configuration such as a buffer length that is not a multiple of the natural data size for the operation. When an error is detected, the channel's processing will stop until the error is handled by software.

Table 16-72. HW_DCP_CH3STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
1	**HASH_MISMATCH**	RW	0x0	The bit indicates that a hashing check operation mismatched for control packets that enable the HASH_CHECK bit. When an error is detected, the channel's processing will stop until the error is handled by software.
0	**RSVD1**	RO	0x0	This bit will always read 0 in the status register, but will be set to 1 in the packet status field after processing of the packet has completed. This was done so that software can verify that each packet completed properly in a chain of commands for cases when an interrupt is issued only for the last item in a packet. The completion bit for the channel is effectively the channel interrupt status bit.

DESCRIPTION:

The interrupt status register is updated at the end of each work packet. If the interrupt bit is set in the command packet's command field, an interrupt will be generated once the packet has completed. In addition, the tag value from the command is stored in the TAG field so that software can identify which command structure was the last to complete. If an error occurs, the ERROR bit is set and processing of the command chain is halted.

EXAMPLE:

```
Empty Example.
```

16.3.31 DCP Channel 3 Options Register Description

The DCP Channel 3 Options Status register contains optional control information that may be used to further tune the behavior of the channel.

HW_DCP_CH3OPTS	0x1F0
HW_DCP_CH3OPTS_SET	0x1F4
HW_DCP_CH3OPTS_CLR	0x1F8
HW_DCP_CH3OPTS_TOG	0x1FC

Table 16-73. HW_DCP_CH3OPTS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																RECOVERY_TIMER															

Table 16-74. HW_DCP_CH3OPTS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSVD**	RO	0x0	Reserved, always set to zero.
15:0	**RECOVERY_TIMER**	RW	0x0	This field indicates the recovery time for the channel. After each operation, the recover timer for the channel is initiallized with this value and then decremented until the timer reaches zero. The channel will not initiate operation on the next packet in the chain until the recovery time has been satisfied. The timebase for the recovery timer is 16 HCLK clock cycles, providing a range of 0ns to 8.3ms at 133 MHz operation.

DESCRIPTION:

The options register can be used to control optional features of the channels.

EXAMPLE:

```
Empty Example.
```

16.3.32 DCP Debug Select Register Description

This register selects a debug register to view.

HW_DCP_DBGSELECT 0x400

Table 16-75. HW_DCP_DBGSELECT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																								INDEX							

Table 16-76. HW_DCP_DBGSELECT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:8	**RSVD**	RO	0x00	Reserved, always set to zero.
7:0	**INDEX**	RW	0x0	Selects a value to read via the debug data register. CONTROL = 0x01 OTPKEY0 = 0x10 OTPKEY1 = 0x11 OTPKEY2 = 0x12 OTPKEY3 = 0x13

DESCRIPTION:

This register selects debug information to return in the debug data register.

EXAMPLE:

```
Empty Example.
```

16.3.33 DCP Debug Data Register Description

Reading this register returns the debug data value from the selected index.

HW_DCP_DBGDATA 0x410

Table 16-77. HW_DCP_DBGDATA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DATA																															

Table 16-78. HW_DCP_DBGDATA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**DATA**	RO	0x0	Debug Data

DESCRIPTION:

This register returns the debug data from the selected debug index source.

EXAMPLE:

```
Empty Example.
```

16.3.34 DCP Version Register Description

Read-only register indicating implemented version of the DCP.

HW_DCP_VERSION 0x430

Table 16-79. HW_DCP_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 16-80. HW_DCP_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x2	Fixed read-onlyl value reflecting the MAJOR version of the design implementation.
23:16	**MINOR**	RO	0x0	Fixed read-onlyl value reflecting the MINOR version of the design implementation.
15:0	**STEP**	RO	0x0	Fixed read-onlyl value reflecting the stepping of version of the design implementation.

DESCRIPTION:

This register returns the debug data from the selected debug index source.

EXAMPLE:

```
Empty Example.
```

DCP Block v2.0, Revision 1.57

The license for the AEC code is documented here for compliance:

Copyright (C) 2000-2003, ASICS World Services, LTD., AUTHORS
All rights reserved. Redistribution and use in source, netlist, binary and silicon forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of ASICS World Services, the Authors and/or the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

Chapter 17
Pixel Pipeline (PXP)

This chapter describes the pixel pipeline (PXP) included on the i.MX23 SoC and how to operate it. Programmable registers are described in Section 17.4, “Programmable Registers.”

17.1 Overview

The pixel pipeline is used to perform alpha blending of graphic or video buffers with graphics data before sending to an LCD display or TV encoder. The PXP provides a performance-optimized engine that can meet the needs of both SDRAM and SDRAM-less systems. The PXP also supports image rotation for hand-held devices that require both portrait and landscape image support.

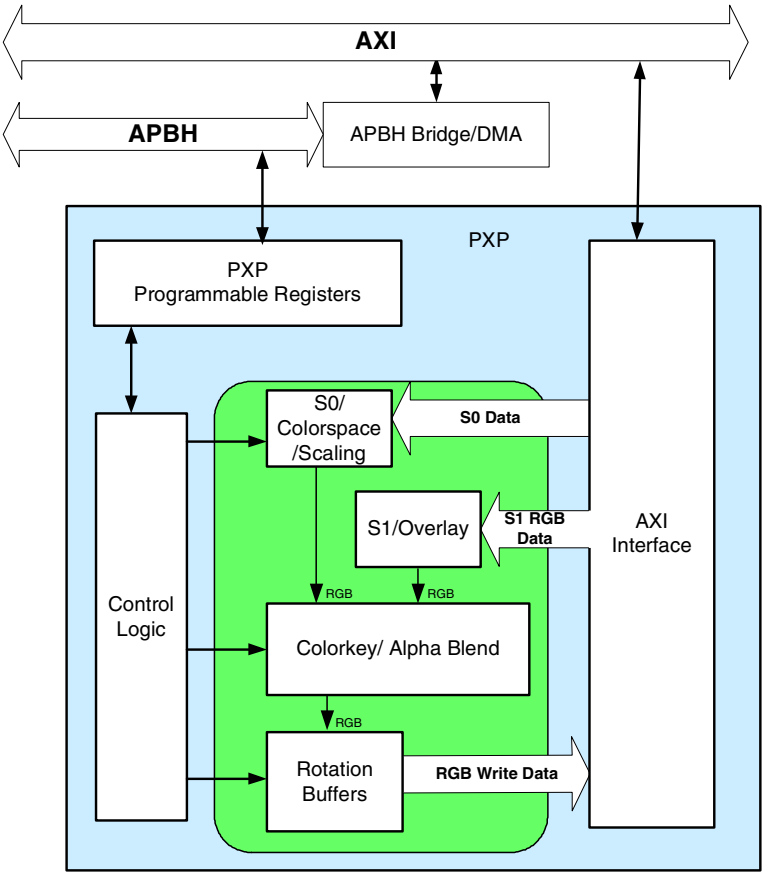


Figure 17-1. Pixel Pipeline (PXP) Block Diagram

The PXP is organized as having a background image (S0) and one or more overlay images that can be blended with the background. Each overlay image must be a multiple of eight pixels in both height and width and the offset of the overlay into the background image must be a multiple of eight pixels. As the PXP processes data, it reads each 8x8 block from the background image and finds the highest priority (lowest numbered) overlay that is co-located at that block coordinate. The PXP then fetches the overlay and performs the alpha blending and color key operations on the two blocks. The resulting 8x8 pixel block is then written to the corresponding block in the output buffer.

For the S0 plane, the PXP supports RGB images (unscaled) or colorspace conversion (YUV->RGB) and scaling of YUV images. The S1 plane consists of up to eight overlay regions consisting of 16 or 32-bit RGB data. The S0 and S1 planes may then be combined by alpha blending, color key substitution, or raster operations (ROPs) to form the output image. Finally the resulting image may be clock-wise rotated in 90 degree increments or flipped horizontally or vertically. The PXP also supports letterboxing and interlacing of progressive content (by writing alternate lines to different frame buffers).

The flow of data through the PXP is shown in Figure 17-2.

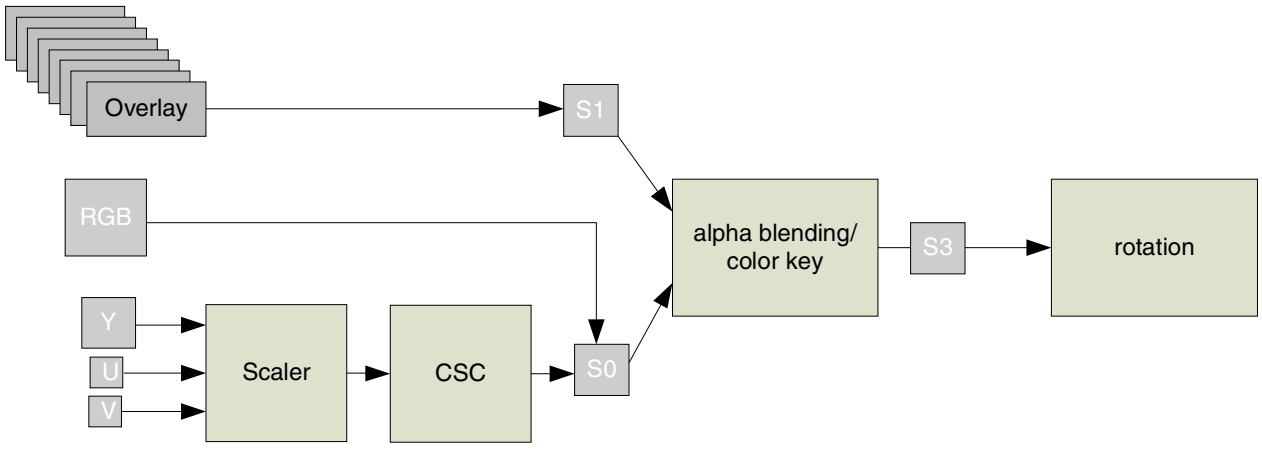


Figure 17-2. Pixel Pipeline (PXP) Data Flow

17.1.1 Image Support

The PXP's S0 buffer supports the following image formats:

- 24-bit unpacked RGB (32bpp)
- 16-bit RGB in either 555 or 565 format
- 3-plane YUV/YCbCr in 4:2:0 or 4:2:2 format

The PXP's S1 buffer supports the following image formats:

- 32-bit RGB (with or without alpha)
- 16-bit RGB in either 555, 565, or 1555 (alpha)

The PXP's output buffer supports

- 32-bit RGB (with alpha)
- 24-bit packed RGB (24bpp)
- 16-bit RGB in either 565, 555, or 1555 format

Internally, all image data is handled as 32bpp data for all steps after the colorspace conversion. Input RGB images are always converted to the equivalent 32bpp format before processing.

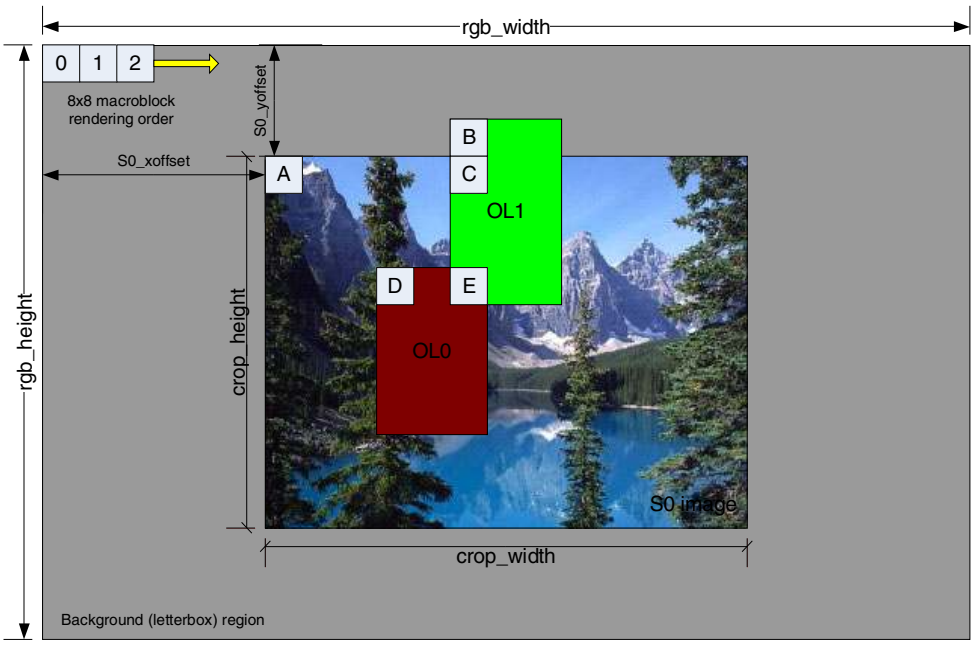
17.1.2 PXP Limitations/Issues

- The PXP's scalar uses a bilinear scaling algorithm and can scale YUV images from 0.5x to 4096x in 12-bit fractional steps.
- The default YUV coefficient register value is incorrect. The C2/C3 field values are reversed.
- When using the NEXT register, the interrupt enable setting should remain the same for all frames. If not, the PXP will change the interrupt enable register value and possibly cause the loss of an interrupt.
- The PXP cannot rotate/flip video in the interlaced modes.
- When performing input interlacing, the input image and overlays must be multiples of 8x16 pixels. Overlays must also reside on 8x16 boundaries.
- The PXP will support images up to 1024 pixels in either the X or Y coordinates.
- The PXP does not support inplace processing when rotation is enabled.

17.2 Operation

The PXP operates by rendering the output frame buffer in 8x8 pixel macroblocks in display order (left to right, then top to bottom). At each output macroblock location the PXP determines whether the S0 buffer is visible based on the cropping register and S0 offset parameters. If the S0 plane is visible, the PXP will fetch and process the required data from the S0 image, otherwise the S0's contribution to the output macroblock will be the S0BACKGROUND register value. This value is effectively the color of the letterboxed region or background color.

The PXP will also determine if an overlay is present for that macroblock location, and if so, instruct the S1 buffer to fetch the required data. If multiple overlays cover the macroblock, the PXP will select only the lowest numbered overlay and direct the S1 buffer to load the data for this overlay. For areas with no overlays the S1 buffer contributes nothing to the rendered image. Figure 17-3 shows the order in which the output blocks are generated (blocks 0, 1, 2) and indicates how various blocks are rendered (blocks A-E).



0,1,2: Pixel blocks rendered with the background color. The numbering and arrow indicate the order of macroblock rendering.
A: S0 Image rendered
B: Overlay 1 blended with background
C: Overlay 1 blended with S0 image
D: Overlay 0 blended with S0 image
E: Overlay 0 blended with S0 image (OL0 takes precedence over OL1)

Figure 17-3. Pixel Pipeline (PXP) Macro Blocks

It is important to understand how the PXP renders each output macroblock to properly understand how it accomplishes cropping, letterboxing, and overlay blending. The following sections will provide more details on these operations.

The PXP also has the ability to rotate/flip images for cases when the pixel scan order is not in the traditional left-to-right/top-to-bottom raster scan (landscape raster). This can occur when a handheld device with a traditional landscape scan is rotated into a portrait orientation (in which the scan order is now bottom-to-top/left-to-right or vice versa) or when a cell phone oriented display (portrait raster) is rotated into a landscape orientation for viewing videos. In these cases, the PXP still renders the image in scan-order format (as sent to the device), but it will traverse the input images based on the transformations required.

The following sections detail each of the PXP's functional capabilities.

17.2.1 Pixel Handling

All pixels are internally represented as 24-bit RGB values with an 8-bit alpha value at all stages in the PXP after the colorspace converter. Input pixels are converted into this format using the following rules:

- 32-bit ARGB8888 pixels are read directly with no conversion for both the S0 and overlay images.
- 32-bit RGB888 pixels are assumed to have an alpha value of 0xFF (full opaque).
- 16-bit RGB565 and RGB555 values are expanded into the corresponding 24-bit colorspace and assigned an alpha value of 0xFF (opaque). The expansion process replicates the upper pixel bits

into the lower pixel bits (for instance a 16-bit RGB565 triplet of 0x1F/0x20/0x07 would be expanded to 0xFF/0x82/0x39).

- 16-bit RGB1555 values are expanded into the corresponding 24-bit colorspace and assigned an alpha value of either 0x00 or 0xFF, based on the 1-bit alpha value in the pixel. The ALPHA_MULTIPLY function is useful in this scenario to allow scaling of the opaque pixels to a semi-transparent value.

Output pixels will retain the effective alpha value of the overlay or can be set to a programmed alpha value using the ALPHA field of the S0PARAM register. 16-bit pixels values are formed from the most significant bits of the 24-bit pixel values.

17.2.2 S0 Cropping/Masking

The PXP's cropping operation should be viewed as a mask on the output image through which the background S0 plane can be viewed. Using this definition clarifies a subtlety on the usage of cropping an image when the image is scaled. When scaling is not used, the input and output image sizes are the same, thus the operation is analogous to cropping the input source image.

The background output image can be cropped to a width and height independent of the image size at a given offset into the image (all sizes are in terms of 8 pixel units) using the values in the S0CROP register. The XBASE and YBASE provide the coordinates of the first block to be displayed from the source image and the WIDTH and HEIGHT parameters specify an effective size of the resulting image in the output buffer.

Cropping must be enabled by setting the CROP bit in the CTRL register to a 1. When not set, the visible portions of the S0 image will be rendered based on the WIDTH and HEIGHT specified in the S0SIZE field. Figure 17-4 indicates how the various cropping parameters relate to the source and RGB images (non-scaled case).

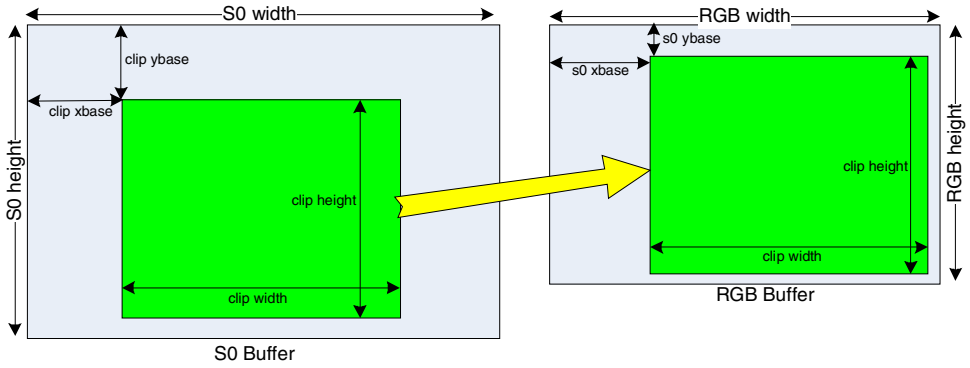


Figure 17-4. Pixel Pipeline (PXP) Cropping

It is important to note that when scaling an image, software **must** specify a valid cropping region since the PXP will default to using the source image size. When downscaling, this is not an issue, but with

upscaling the resulting image will be a scaled up version of the source, but cropped to the same size as the source image as shown in Figure 17-5.

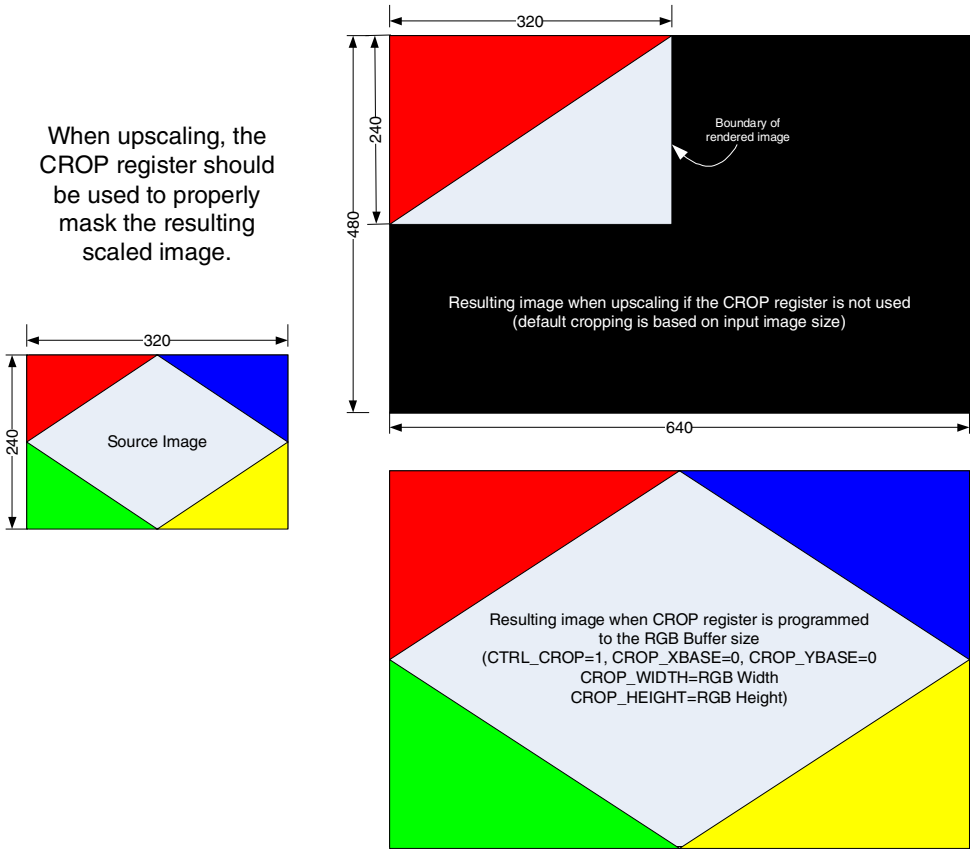


Figure 17-5. Pixel Pipeline (PXP) Scaling and Cropping Example

The cropping extents should fall completely within the S0 buffer to avoid displaying incorrect data. The PXP hardware does not check for these conditions and will render the image as shown in the following two diagrams. (Note that the cropping width and height can be viewed as applying to the input buffer only because it is not scaled. In actuality, it is applied to the output buffer).

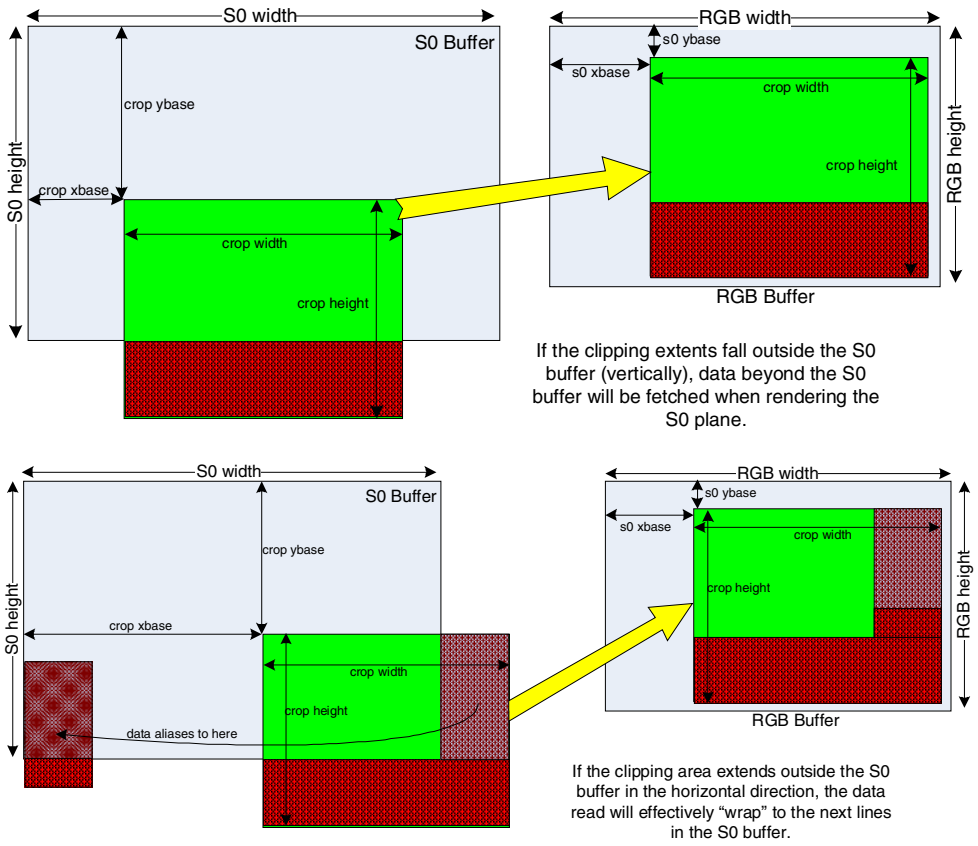


Figure 17-6. Invalid PXP Cropping Examples

17.2.3 Scaling

The PXP can scale YUV images from 1/2x to about 4096x (although the upper range is technically unlimited) using a bilinear scaling algorithm. The hardware is capable of scaling with 12-bit fractional resolution, or in 1/4096th pixel increments with independent scaling ratios for the X and Y direction. The scaler also implements an initial offset, which can be useful when scaling by powers of 2 in order to ensure that the resulting pixels are averages of the source pixels instead of producing a decimated or replicated image.

The scaling parameter is specified to the hardware in terms of the inverse of the scaling ratio desired. This can also be viewed as the step size between computed sample values. For instance, when scaling by 2x the inverse is 1/2, thus the scaler will increment by 1/2 pixel steps across the input image and compute the bilinear average for each sample point.

The scaling values are represented by 12-bit fractional values in the scaling register and hardware. The scaling ratios are computed as the input size divided by the output size. The resulting decimal value must then be converted into a 12-bit fixed point value by multiplying by 2^{12} or 4096 to produce the value programmed into the scaling registers.

To scale an image from 400x300 to 320x200, the horizontal XSCALE factor is computed as

$$\text{XSCALE} = \frac{\text{Input Size}}{\text{Output Size}} \times 4096 = \frac{400}{320} \times 4096 = 1.25 \times 4096 = 0\text{x}1400$$

The vertical YSCALE can be similarly computed as

$$\text{YSCALE} = \frac{\text{Input Size}}{\text{Output Size}} \times 4096 = \frac{300}{200} \times 4096 = 1.5 \times 4096 = 0\text{x}1800$$

The scaler will use the CROP_XBASE and CROP_YBASE values as an offset into the source S0 image for the origin of the input image to be scaled. The CROP_WIDTH and CROP_HEIGHT parameters will be used to determine the extent of the ***scaled*** image in the output buffer. It is tempting to view the cropping width and height as being applied to the input buffer, but this is incorrect -- the PXP uses these values as a mask on the output buffer to determine which regions of the output buffer require data from the scaled input image.

To enable scaling, the HW_PXP_CTRL_SCALE bit must be set and the desired scaling ratios written into the HW_PXP_S0SCALE registers. Initial offsets should be programmed into the HW_PXP_S0OFFSET register.

17.2.3.1 Scaling Operation

The scaling engine operates on YUV (or YCbCr) 422 or 420 formatted pixels. The scaled output image is presented to the CSC module as YUV444 pixels with a single byte for each the Y, U, and V channels. The scaler can reduce an input image by a maximum factor of 2. In this case, the output image will be ½ the dimension of the input image in each of the X and Y axis. There are no limits, essentially, on increasing the source image size. The theoretical maximum increase is 4096 since a 12 bit fractional step function is used when scaling an input image. Scaling in either axis, X or Y is independent, so a source image can appear stretched in either direction.

All source images using YUV/YCbCr pixels must pass through the scale engine. The PXP alpha blend module and Overlay pixel streams are in the RGB888 format, so S0 pixel buffers must be converted to the RGB888 format for alpha blending, the final transform just before rotation. Even when the S0 image is passed without scaling, or 1:1 scaling is selected, the scaling engine is required to convert the incoming pixels from 422/420 format to YUV444. This is the format required by the CSC engine to convert to the RGB color space, The scaling engine works with the CSC module to translate YUV/YCbCr pixel formats to RGB888 for output frame buffer compositing using the alpha blender.

NOTE

RGB S0 source images cannot use the scaling engine. Scaling is not supported on RGB pixel formats and the scale and CSC modules are bypassed in this case.

17.2.3.1.1 Bilinear Image Scaling Filter

The PXP implements a bilinear scaling filter to resize an input image to a different resolution for display output. The bilinear filter is a weighted average of the four nearest pixels that can be sourced to approximate the pixel in the output frame buffer.

To compute the output pixel value at position as indicated by P, consider the diagram in Figure 17-7.

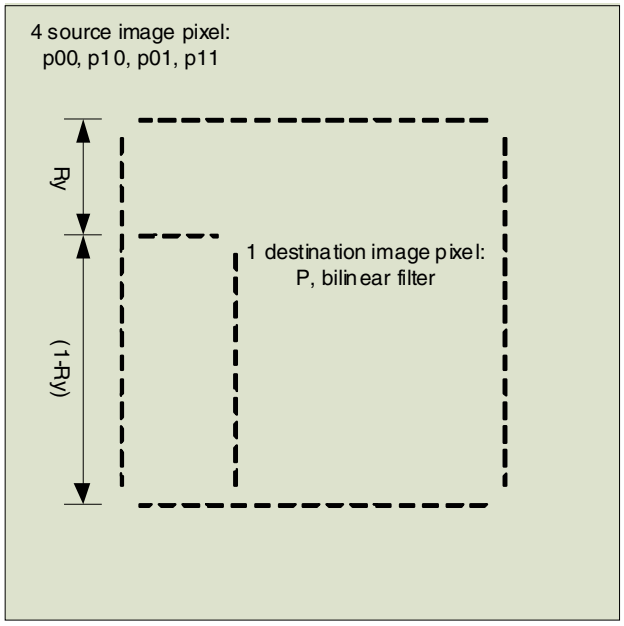


Figure 17-7. Computing Pixel Value in Output Frame Buffer

A step function is used to indicate the position of the pixel "P" in the output frame. This position may not coincide with a single pixel position in the input frame buffer. In this case, the four closest pixels in the input frame are used to approximate the value of the pixel in the output frame.

The PXP scaler first computes a linear filter in the X axis to create the two intermediate pixel values Px0 and Px1. The step function's X fractional component is used to provide the weighting factor for blending p00 with p10 to provide Px0. Likewise, Px1 is also derived from a linear filter using p01 and p11.

The equations for Px0 and Px1 are as follows:

$$Px0 = p00*(1-Rx) + p10*Rx \qquad \textit{Eqn. 17-1}$$

$$Px1 = p01*(1-Rx) + p11*Rx \qquad \textit{Eqn. 17-2}$$

The PXP scaler uses the intermediate X pixels Px0 and Px1 and implements a bilinear filter on these two pixel values to produce the final pixel value at position P. The remainder of the step function for the Y axis is used to compute the weighted average pixel result. The equation for the final filtered pixel follows:

$$P = Px0*(1-Ry) + Px1*Ry \qquad \textit{Eqn. 17-3}$$

17.2.3.1.2 YUV 4:2:2 Image Scaling

Figure 17-8 illustrates the positioning of YUV samples for the 4:2:2 formats. There are twice as many Y luma samples as U and V chroma samples horizontally.

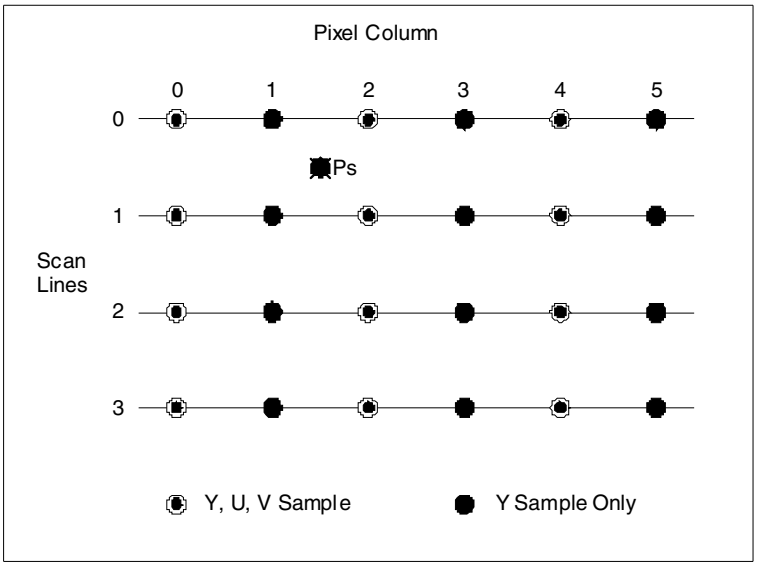


Figure 17-8. YUV Samples for 4:2:2 Formats

Consider the scaled output pixel Ps (pixel scaled) which has an accumulated step function of X=1.5 and Y=0.5. The remainder for the step function is Rx = 0.5 and Ry = 0.5. Or, the sub pixel position of output pixel Ps is half way between line 0 and 1 and half way between column 1 and 2.

The Y output component of Ps is simply the bilinear function of the four nearest Y samples from the input image. Specifically, the Y values at [1,0], [2,0], [1,1], and [2,1] are used to compute the Y for Ps.

For the U and V components of Ps, there are no samples present in the column position 1. The bilinear filter uses chroma components located at [0,0], [2,0], [0,1] and [2,1]. Since the chroma components are not sub sampled vertically, the remainder used to combine pixels vertically is Ry=0.5 (the same as for Y). However, horizontally, the scaling engine shifts the remainder by a factor of 2. So an X axis step function value of X=1.5 has a remainder Rx=0.75. Source chroma values are not replicated, they are completely interpolated using the four nearest chroma samples to approximate U and V at Ps.

17.2.3.1.3 YUV 4:2:0 Image Scaling

Figure 17-9 illustrates the positioning of YUV samples for the 4:2:0 formats. Chroma is sub sampled both horizontally and vertically. In this format, the chroma frame buffers contain ¼ the data that the luma frame buffers store.

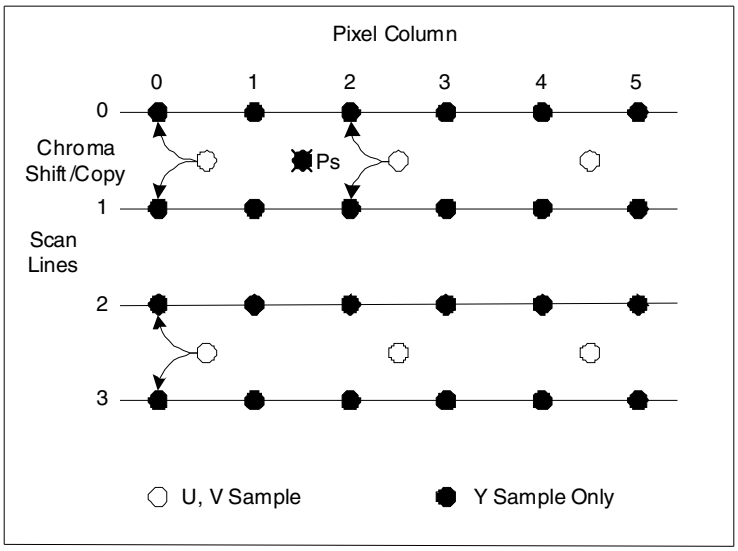


Figure 17-9. YUV Samples for 4:2:0 Formats

The Y output component for all scaled pixels in 4:2:0 formats are the same as for the 4:2:2 pixel formats.

The U and V output components have two considerations when computing the output pixel Ps.

1. All chroma samples from the input source image are shifted left and up by ½ a sample position of the input pixel matrix.
2. Odd scan lines are replicated using the previous even chroma scan line values. So, output image chroma values that map between even to odd scan lines are **replicated** in the vertical axis. In contrast, output image chroma values between odd to even scan lines are **interpolated** vertically.

The chroma values are interpolated horizontally as in the 4:2:2 pixel format.

As an example, consider the interpolated pixel Ps in the 4:2:0 diagram above. For the Y component, the interpolated output luma is a function of the Y values in the source frame buffer at position [1,0], [2,0], [1,1], [2,1].

For the U and V interpolated samples, the chroma values on scan line position 0.5 are shifted so that they coincide with the even luma sample points. They are also replicated so that a single chroma scan line is used twice. The chroma scan line at 0.5 is replicated to represent the 4:2:2 sample points for scan line 0 and 1. The chroma scan line at 2.5 is replicated to represent the 4:2:2 sample points for scan line 2 and 3. This pattern of chroma replication occurs for the entire source frame buffer during the scaling operation.

Figure 17-10 has two examples for the computation of the scaled chroma output pixel. For chroma at output position PsA (vertical position 0.5), interpolation occurs in the X axis using chroma values at col-

umn 0 and column 2. However, since line 0 and line 1 have equal chroma values due to chroma line replication, scaling in the Y axis results in replication of chroma values.

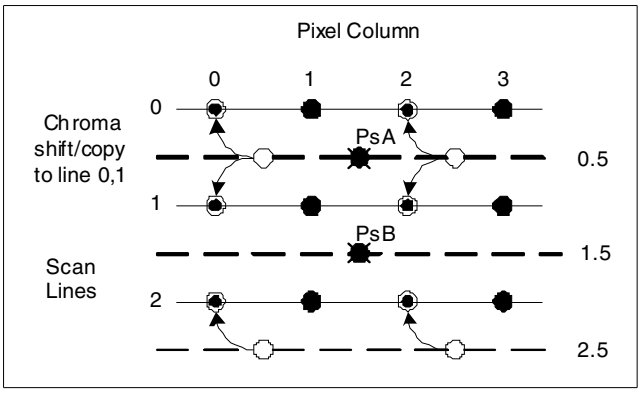


Figure 17-10. Examples for Scaled Chroma Output Pixel

For chroma at output position PsB (vertical position 1.5), interpolation occurs in both the X and Y axis. The Y axis is an interpolation since the chroma values for scan line 1 and 2 and not replicated respectfully.

In summary, any output image pixels that map to an odd scan line above and an even scan line below are interpolated vertically. Output image pixels that map to an even scan line above and an odd scan line below are replicated vertically.

17.2.3.1.4 Out-of-Range Image Access

An important note with respect to the scaling engine is that there are no provisions for accessing data that is out of range with respect to the source image. Under some circumstances (even typical use scenarios), it would appear that the data should exist. The important note is that the resolution of the Y luma plane is not the same as the resolution of the U and V chroma planes. Also, all pixels are interpolated horizontally with respect to their nearest neighbors.

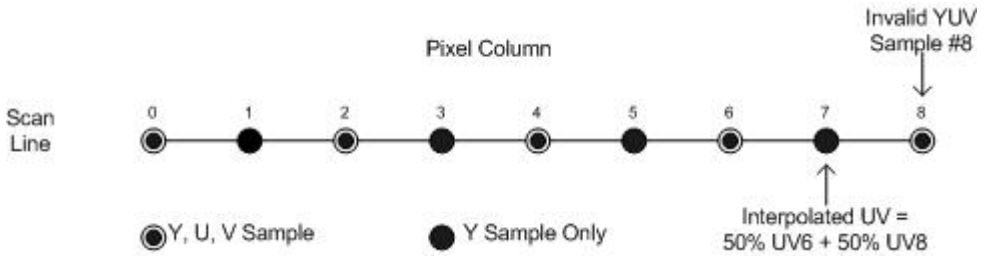


Figure 17-11. Scan Line Sample Positions

An easy example of this would be sourcing an 8x8 YUV 422 input image and displaying it as an 8x8 output image. In this scenario, an obvious but incorrect scaling ratio would be 1:1. Horizontally, there are 8 Y values and only 4 chroma values. In a given scan line, there are Y samples at positions 0 – 7 as indicated in Figure 17-11. Also, the 4 UV samples are at positions 0, 2, 4, and 6. A UV sample does not exist at position 7. The scaling engine will interpolate the UV sample at position 7 from the UV sample at position 6 and 8. From Figure 17-11, it can be observed that position 8 is not a valid sample for the displayed scan line. The step function would be initialized to 0 to interpolate the pixel at position 0. The step function would increment 7 times thereafter to produce 8 output pixels as follows:

Step 0: 100% Y0, 100% U0V0

Step 1: 100% Y1, 50% U0V0, 50% U2V2

Step 2: 100% Y2, 100% U2V2

Step 3: 100% Y3, 50% U2V2, 50% U4V4

Step 4: 100% Y4, 100% U4V4

Step 5: 100% Y5, 50% U4V4, 50% U6V6

Step 6: 100% Y6, 100% U6V6

Step 7: 100% Y7, 50% U6V6, **50% U8V8**

Since the resolution of the UV plane is half of the Y plane, there are valid UV samples at pixel position 0, 2, 4, and 6. At pixel position 7, the final pixel is interpolated with a UV value that apparently does not exist in position 8 for this provided S0 image. An artifact could be visible when programming the scale engine to scale 1:1 for this simple example. The UV data at position 8 is likely the first UV sample on the next line. Interpolating the pixel at (x,y) = 7,0 using the UV sample at 0,1 likely would produce an artifact on the right side of the “scaled” image. “Scaled” is quoted in this case, since the scale factor is 1:1 which is used to convert the YUV422 input image to YUV444 in preparation for color space conversion.

There are several methods to compensate for this anomaly. Also, similar cases of scaling factors that are not equal to 1:1 could produce a similar scenario on the bottom horizontal line of pixels. The user should take care to understand the nature of the source image data and the scaling algorithm implemented in the PXP to account for potential out of range image access.

17.2.4 Colorspace Conversion

The CSC module receives scaled YUV/YCbCr444 pixels from the scale engine and converts the pixels to the RGB888 color space. These pixels are loaded into the pixel FIFO for processing by the alpha blend module.

The following equations are used to perform YUV/YCbCr -> RGB conversion. The constants will be stored in the PXP control registers as two’s compliment values to allow flexibility in the implementation and to allow for differences in the video encode and decode operations. In addition, this provides a software mechanism to manipulate brightness or contrast.

R = C0(Y+Yoffset) + C1(V+UVoffset)

G = C0(Y+Yoffset) + C3(U+UVoffset) + C2(V+UVoffset)

B = C0(Y+Yoffset) + C4(U+UVoffset)

Note: In the equations above, U and V are synonymous with Cb and Cr in regards to the color space format of the source frame buffer. Since UV values have been converted into an unsigned integer representation before entering the scaler, the Coffset for both UV and CbCr modes should be 0x180 (-0x80 or -128).

Saturation of each color channel is checked and corrected for excursions outside the nominal YUV/YCbCr color spaces. Overflow for the three channels are saturated at 0x255 and underflow is saturated at 0x00.

Table 17-1 indicates the expected coefficients for YUV and YCbCr modes of operation:

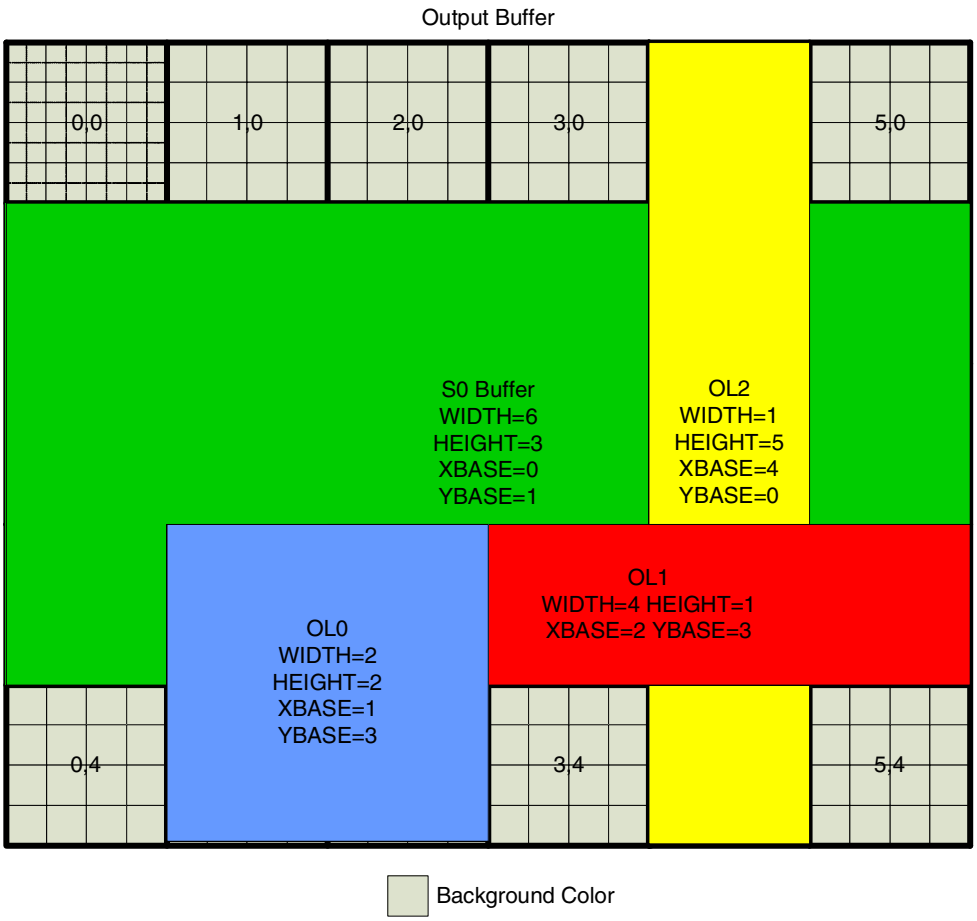
Table 17-1. Coefficients for YUV and YCbCr Operation

Coefficient	YUV	YCbCr
Yoffset	0x000	0x1F0 (-16)
UVoffset	0x180 (-128)	0x180 (-128)
C0	0x100 (1.00)	0x12A (1.164)
C1	0x123 (1.140)	0x198 (1.596)
C2	0x76B (-0.581)	0x730 (-0.813)
C3	0x79B (-0.394)	0x79C (-0.392)
C4	0x208 (2.032)	0x204 (2.017)

By default, the PXP colorspace coefficients are set to support the conversion of YUV data to RGB data. If YCbCr input is present, software must change the coefficient registers appropriately (see the register definitions for values). Software must also set the YCBCR_MODE bit in the COEFF0 register to ensure proper conversion of YUV versus YCBCR data.

17.2.5 Overlays

The PXP supports up to eight overlays that can be used to merge graphic data with video (or other graphic data). Each overlay consists of a rectangular area that is a multiple of eight pixels in both the vertical and horizontal directions. Overlays must also be located on 8x8 boundaries within the output image. As the PXP processes each 8x8 macroblock, it determines if any of the enabled overlays cover the block and then merges the overlay data with the background image as specified in the overlay's control registers. If multiple overlays overlap for a given 8x8 block, the PXP will select the lowest numbered one for the blending operation. If the desired affect is to blend the overlays together, this can be accomplished as a multi-step process using the IN_PLACE functionality (see Section 17.2.10, "In-place Rendering")



The S0 buffer and each overlay can be placed within the output buffer using their XBASE and YBASE registers and the dimensions of each region are set using their WIDTH and HEIGHT parameters. Overlay 0 has the highest priority (effectively it is the highest in the stacking order) and the S0 buffer and background color have the lowest priority.

Overlays can be blended with the background or S0 planes, but not with each other. Effectively only a single overlay is active for each 8x8 pixel block.

Figure 17-12. Pixel Pipeline Overlay Support

Each overlay can perform one of three classes of operations between the overlay and the underlying background (S0) image: alpha blending, color keying, or raster operations.

An overlay can be enabled by writing the address of the overlay image to the OLn register, the overlay's size and location information into the OLnSIZE register, and then setting the OLnPARAM_ENABLE bit. The OLnPARAM registers also contain further controls to select the modes of operation (below).

17.2.6 Alpha Blending

The alpha value for an individual pixel represents a mathematical weighting factor applied to the S1 pixel. An alpha value of 0x00 corresponds to a transparent pixel and a value of 0xFF corresponds to an opaque pixel.

The effective alpha value for an overlay pixel is determined by the ALPHA bit-field and the two ALPHA control bits in the OLnPARAM register. If the ALPHA_CTRL field is set to ALPHA_OVERRIDE, the alpha value for the pixel is taken from the ALPHA bit-field. This can be useful for applying a constant alpha to an entire image or for image formats that don't include an alpha value. If ALPHA_MULTIPLY is selected, the pixel's alpha value will be multiplied by the ALPHA value in order to allow scaling of the pixel's alpha or to provide better control for pixel formats such as RGB1555, which only contains a single bit of alpha.

For each color channel, the equation used to blend two source pixels is defined below:

$$E\alpha = \text{Embedded alpha associated with S1 pixel}$$

$$\alpha = G\alpha \times E\alpha + 0x80$$

$$G\alpha = \text{PIO programmed global alpha (8-bit value)}$$

The result for the red channel as an example

$$\dot{Y}r[7:0] = (\alpha \times S1.r) + ((1 - \alpha) \times S0.r)$$

When alpha is 0xff, the S1 pixel will not be blended with S0, but S1 will be passed as the output pixel and will not be blended with S0. In this case, S0 will be discarded. Likewise, if alpha is 0x00 for a given pixel, S0 will be loaded as the output pixel.

Alpha values in the overlays are loaded from the source image for all pixel formats. For formats that do not support an alpha value, the pixel is assigned an alpha value of 0xFF (opaque). This can be modified by the overlay processing by setting either the ALPHA_MULTIPLY or ALPHA_OVERRIDE bit in the associated OLnPARAM register.

17.2.7 Color Key

Pixels may be made transparent to the corresponding overlay by using the S0 colorkey registers. If an S0 pixel matches the range specified by the S0COLORKEYLOW and S0COLORKEYHIGH registers, the pixel from the associated overlay will be displayed. If no overlay is present for that block, a black pixel will be generated since the default overlay pixel is 0x00000000 (transparent black pixel).

The most common use for this is when a bitmap does not support an alpha-field or for applications such as "green screen" where an image is substituted for a solid background color as shown in Figure 17-13.

Figure 17-13. Pixel Pipeline (PXP) Colorkey Example

The green portion of the overlay image can be colorkeyed to display the contents of the S0 buffer for locations that match the color range. For this example, the color range is

OL COLORKEY: 00<R<80 70<G<FF 00<B<80

Conversely, background colorkeying could also have been used if the images had been swapped.

If colorkeying is enabled for an overlay, any pixels matching the colorkey parameters will be handled as colorkeyed pixels. Non-matching pixels will be alpha blended or handled by ROP operations as normal.

17.2.8 Raster Operations (ROPs)

In addition to alpha blending and color keying, the PXP's alpha blender also supports a set of raster operations that may be performed between the active overlay and the background image. The operations are done on a per-pixel basis and are performed using the 24-bit overlay and background image values. The following table lists the supported ROP operations

Table 17-2. Supported ROP Operations

Mnemonic	Value	Operation
MASKOL	0x0	OL & S0
MASKNOTOL	0x1	~OL & S0
MASKOLNOT	0x2	OLL & ~S0
MERGEOL	0x3	OL | S0
MERGEOLNET	0x4	~OL | S0
MERGEOLNOT	0x5	OL | ~S0

Table 17-2. Supported ROP Operations (continued)

Mnemonic	Value	Operation
NOTCOPYOL	0x6	~OL
NOT	0x7	~S0
NOTMASKOL	0x8	~(OL & S0) (nand)
NOTMERGEOL	0x9	~(OL | S0) (nor)
XOROL	0xA	OL ^ S0 (xor)
NOTXOROL	0xB	~(OL ^ S0) (xnor)

These operations are specified in the overlay's PARAM register and must be enabled by setting the ALPHA_CTRL field to ROPs.

17.2.9 Rotation

Rotation is an inherently inefficient operation, especially for a graphics device operating in a raster-scan fashion since the resulting memory fetches would be non-contiguous. The PXP solves this problem by operating on 8x8 pixel blocks. This allows the PXP to rotate a subportion of the image, where it can fetch 8 lines of pixels, process them, and then write 8 lines of pixels regardless of the rotation orientation.

Rotation is mainly useful for reorganizing the frame buffer for handheld LCD displays for cases when the user rotates the device from a portrait to landscape orientation. Consider the following scenario:

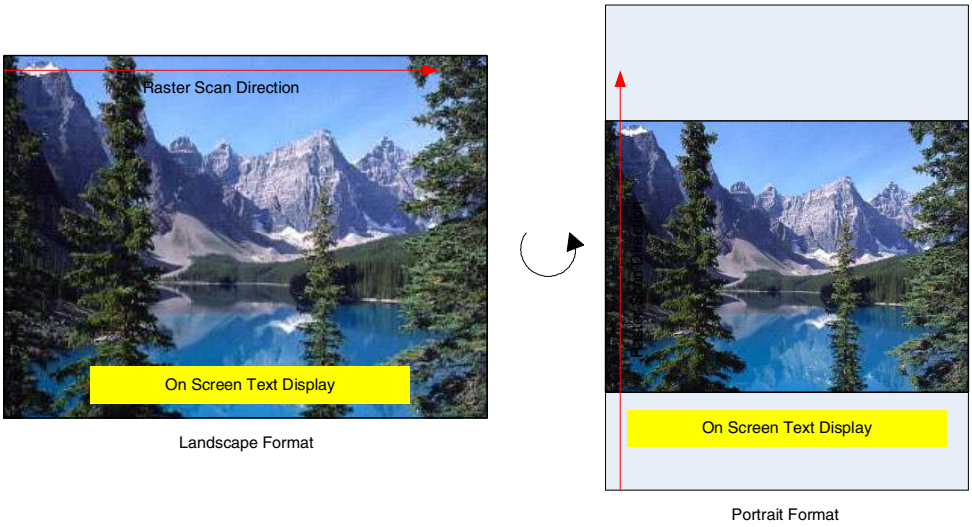


Figure 17-14. Pixel Pipeline (PXP) Rotation Example 1

While this looks like a trivial operation, consider what the frame buffer must look like in memory before being sent to the LCD in raster-scan format

Portrait Format Frame Buffer

Figure 17-15. Pixel Pipeline (PXP) Rotation Example 2

Not only must the image be rotated, but any on-screen graphics must also be rendered in a different orientation. By building rotation into the rendering process, the PXP allows software to construct the image in the traditional portrait format and simply rotate the image/overlays for the LCD interface during composition.

The rotation operations are defined as rotations in a clockwise direction and the flip operations will flip the pixels in the specified direction.

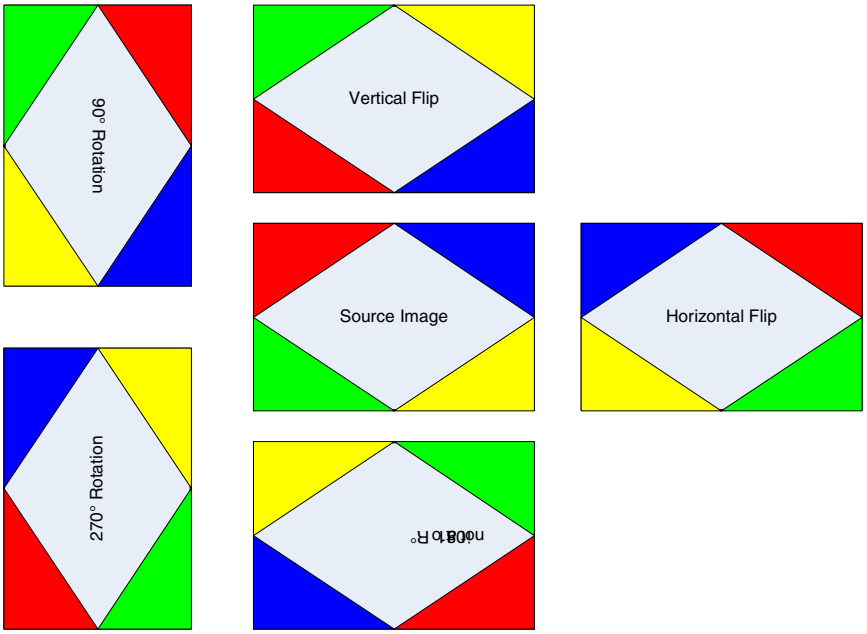


Figure 17-16. Pixel Pipeline (PXP) Rotation and Flip Definition

The PXP supports rotation in 90 degree increments as well as horizontal and vertical flip operations. These can be done in any combination (for example, 90 degree rotation with both vertical and horizontal flip). When a flip operation is specified in combination with a rotation operation, the PXP will render the output such that the effect of the flip operation(s) occur BEFORE the rotation operation.

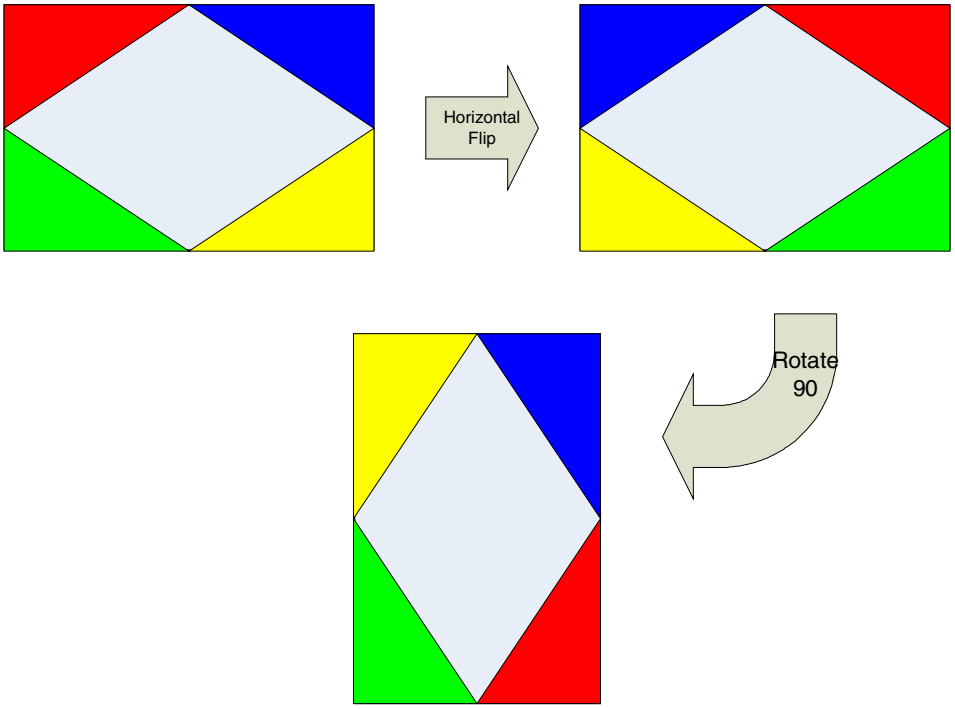


Figure 17-17. Pixel Pipeline (PXP) Rotation Plus Flip Definition

Rotations and flip operations are enabled by setting the VFLIP, HFLIP, and ROTATE fields of the HW_PXP_CTRL register.

17.2.10 In-place Rendering

The PXP also has the ability to process an image and write the resulting buffer back to the original S0 buffer. This is referred to as "in place" rendering. This scenario may be useful when software wishes to alpha blend multiple images where the overlays effectively overlap each other.

When the IN_PLACE control bit is set to 1, the control logic will optimize the PXP's operations to only process the blocks that match an overlay region since all other pixels will be unmodified. This considerably reduces processing time as well as memory bandwidth used.

In place rendering is enabled by setting the IN_PLACE bit in the HW_PXP_CTRL registers. Note the following restrictions when rendering in place:

- The source buffer is used as the destination buffer (RGBBUF is not used),
- Only RGB S0 images are supported (not YUV)

- The output RGB format must be programmed to the same value as the input RGB format.

17.2.11 Interlaced Video Support

The PXP has some minimal ability to generate interlaced video content from a progressive source. There are two available options, based on the bandwidth requirements and how software is managing video frames. The PXP can either interlace on the input side (by reading every other line of input data) or on the output side (by writing the individual lines of video into two separate fields). Generally, output interleaving should be used since it is the most flexible mode (it allows scaling and full overlay support) and it only requires a single pass of the PXP to generate two separate output fields. Input interleaving can be beneficial in cases where the PXP is running at 60fps, since it requires fewer fetches to produce the output data.

The PXP will perform input interlacing when the INTERLACED_INPUT field is programmed to either FIELD0 or FIELD1 (to select the desired field). When performing output interlacing, the PXP will write field0 data to the RGBBUF pointer and the field1 data to the RGBBUF2 pointer. The OUTPUT_INTERLACING field of the HW_PXP_CTRL register controls which of these fields (or both) are generated.

17.2.12 Queueing Frame Operations

The PXP supports a primitive ability to queue up one operation while the current operation is running. This is enabled through the use of the HW_PXP_NEXT register. When this register is written, it enables the PXP to reload its current register contents with the data found at the location pointed to by this address (when it completes processing of the current frame. This feature may be useful in helping to reduce the interrupt latency in servicing the PXP.

If the PXP is idle when the HW_PXP_NEXT register is written, the PXP treats this as an indication that it should immediately load the values at the pointer and begin processing the frame. This ability should allow software to use the same routines when programming the PXP (so that the first frame doesn't differ from subsequent frames).

When loading values from the NEXT register, nearly all registers in the PXP are reloaded, including the interrupt enable bit in the control register. It is recommended that the interrupt enable value not be changed when using queued operations to ensure that interrupts are not spuriously lost or generated. The following table indicates the registers that are affected and the offset into the block address in memory.

Table 17-3. Registers and Offsets

Offset	Register	OFFSET	REGISTER
0x00	CTRL	0x60	OL2
0x04	RGBBUF	0x64	OL2SIZE
0x08	RGBBUF2	0x68	OL2PARAM
0x0C	RGBSIZE	0x6C	OL2PARAM2

Table 17-3. Registers and Offsets (continued)

Offset	Register	OFFSET	REGISTER
0x10	S0BUF	0x70	OL3
0x14	S0UBUF	0x74	OL3SIZE
0x18	S0VBUF	0x78	OL3PARAM
0x1C	S0PARAM	0x7C	OL3PARAM2
0x20	S0BACKGROUND	0x70	OL4
0x24	S0CROP	0x84	OL4SIZE
0x28	S0SCALE	0x88	OL4PARAM
0x2C	S0OFFSET	0x8C	OL4PARAM2
0x30	S0COLORKEYLOW	0x90	OL5
0x34	S0COLORKEYHIGH	0x94	OL5SIZE
0x38	OLCOLORKEYLOW	0x98	OL5PARAM
0x3C	OLCOLORKEYHIGH	0x9A	OL5PARAM2
0x40	OL0	0xA0	OL6
0x44	OL0SIZE	0xA4	OL6SIZE
0x48	OL0PARAM	0xA8	OL6PARAM
0x4C	OL0PARAM2	0xAC	OL6PARAM2
0x50	OL1	0xB0	OL7
0x54	OL1SIZE	0xB4	OL7SIZE
0x58	OL1PARAM	0xB8	OL7PARAM
0x5C	OL1PARAM2	0xBC	OL7PARAM2

17.3 Examples

This section includes several examples of programming the PXP to render an output image. The image could be either a still image or one frame of a sequence of video images. For each case, the input and output images will be shown along with a table of PXP register settings. In all examples, pointers to the data structures with image data will be referred to in the following notation: **imagename_type*, where imagename indicates which image is being used and type indicates either luma (y), chroma (u, v) or RGB data (rgb). All register names are assumed to have the **HW_PXP_** register prefix. The registers can be written in any order except the HW_PXP_CTRL register, which must be written last since it enables the PXP's operation.

17.3.1 Basic QVGA Example

This example shows how to perform basic colorspace conversion of a 3-plane YUV image into an RGB image suitable for an LCD device.

Table 17-4. Register Use for Conversion

Register	Value	Description
RGBBUF	*example1_rgb	Pointer to the output buffer.
RGBSIZE	0xFF1400F0	ALPHA=0xFF WIDTH=0x140=320 HEIGHT=0x0F0=240
S0BUF	*morraine_y	Pointer to input Y buffer
S0UBUF	*morraine_u	Pointer to input U buffer
SOVBUF	*morraine_v	Pointer to input V buffer
S0PARAM	0x0000281E	WIDTH=0x28=40 (40*8=320 pixels) HEIGHT=0x1E=30 (30*8=240 pixels)
S0BACKGROUND	0x00000000	Black background region
S0CROP	0x00000000	No Cropping
S0CSCCOEFF0 S0CSCCOEFF1 S0CSCCOEFF2	0x04030000 0x01230208 0x076b079b	YUV->RGB Coefficient Values
OL0PARAM	0x00000000	Overlay 0 disabled
OL1PARAM	0x00000000	Overlay 1 disabled
OL2PARAM	0x00000000	Overlay 2 disabled
OL3PARAM	0x00000000	Overlay 3 disabled
OL4PARAM	0x00000000	Overlay 4 disabled
OL5PARAM	0x00000000	Overlay 5 disabled
OL6PARAM	0x00000000	Overlay 6 disabled
OL7PARAM	0x00000000	Overlay 7 disabled
CTRL	0x00009003	S0_FORMAT=9 (YUV420) IRQ_ENABLE=1 ENABLE=1

The resulting image is simply the RGB equivalent of the YUV image:

Figure 17-18. Example: RGB Equivalent of YUV image

17.3.2 Basic QVGA with Overlays

This example is similar to the last, but adds two overlay images, one for a logo and the other as a time counter/control bar. The two overlay images are shown in Figure 17-19. (Note that the black background is actually transparent in the real image).

Figure 17-19. Example: QVGA with Overlays

Changes from the previous example are in bold.

Table 17-5. Register Use for Conversion

Register	Value	Description
RGBBUF	*example1_rgb	Pointer to the output buffer.
RGBSIZE	0xFF1400F0	ALPHA=0xFF WIDTH=0x140=320 HEIGHT=0x0F0=240
S0BUF	*morraine_y	Pointer to input Y buffer
S0UBUF	*morraine_u	Pointer to input U buffer
SOVBUF	*morraine_v	Pointer to input V buffer
S0PARAM	0x0000281E	WIDTH=0x28=40 (40*8=320 pixels) HEIGHT=0x1E=30 (30*8= 240 pixels)
S0BACKGROUND	0x00000000	Black background region
S0CROP	0x00000000	No Cropping
S0CSCCOEFF0 S0CSCCOEFF1 S0CSCCOEFF2	0x04030000 0x01230208 0x076b079b	YUV->RGB Coefficient Values
OL0	***overlay1_rgb**	**Pointer to control graphic**
OL0SIZE	**0x00000A02**	**WIDTH=0x0A=80 pixels HEIGHT=0x02=16pixels**
OL0PARAM	**0x0000FF01**	**ALPHA=0xFF FORMAT=0x0 (RGB8888) ALPHA_CTRL=0 (embedded alpha) ENABLE=1**
OL1	***logo_rgb**	**Pointer to logo graphic**
OL1SIZE	**0x0A181D06**	**XBASE=0x0A=80pixels YBASE=0x18=192pixels WIDTH=0x1D=232pixels HEIGHT=0x06=48pixels**
OL1PARAM	**0x0000FF01**	**ALPHA=0xFF FORMAT=0x0 (RGB8888) ALPHA_CTRL=0 (embedded alpha) ENABLE=1**
OL2PARAM	0x00000000	Overlay 2 disabled
OL3PARAM	0x00000000	Overlay 3 disabled
OL4PARAM	0x00000000	Overlay 4 disabled

Table 17-5. Register Use for Conversion (continued)

Register	Value	Description
OL5PARAM	0x00000000	Overlay 5 disabled
OL6PARAM	0x00000000	Overlay 6 disabled
OL7PARAM	0x00000000	Overlay 7 disabled
CTRL	0x00009003	S0_FORMAT=9 (YUV420) IRQ_ENABLE=1 ENABLE=1

The resulting image is shown below. Note the presence of the overlays in the upper left and lower right corners of the image.

Figure 17-20. Example: QVGA with Overlays

17.3.3 Cropped QVGA Example

This example displays the same image as the first example, but does so on a portrait-oriented display (240x320) without the overlays. Changes from the first example are shown in bold.

Table 17-6. Register Use for Conversion

Register	Value	Description
RGBBUF	*example1_rgb	Pointer to the output buffer.
RGBSIZE	0xFF**0F0140**	ALPHA=0xFF **WIDTH=0x0F0=240 pixels** **HEIGHT=0x140=320 pixels**
S0BUF	*morraine_y	Pointer to input Y buffer
S0UBUF	*morraine_u	Pointer to input U buffer
SOVBUF	*morraine_v	Pointer to input V buffer
S0PARAM	0x00**05**281E	**YBASE=0x05=40pixels** WIDTH=0x28=40 (40*8=320 pixels) HEIGHT=0x1E=30 (30*8= 240 pixels)
S0BACKGROUND	0x00000000	Black background region

Table 17-6. Register Use for Conversion (continued)

Register	Value	Description
S0CROP	0x05001E1E	XBASE=0x05=40 pixels YBASE=00=0pixels WIDTH=0x1E=240pixels HEIGHT=0x1E=240 pixels
S0CSCCOEFF0 S0CSCCOEFF1 S0CSCCOEFF2	0x04030000 0x01230208 0x076b079b	YUV->RGB Coefficient Values
OL0PARAM	0x00000000	Overlay 0 disabled
OL1PARAM	0x00000000	Overlay 1 disabled
OL2PARAM	0x00000000	Overlay 2 disabled
OL3PARAM	0x00000000	Overlay 3 disabled
OL4PARAM	0x00000000	Overlay 4 disabled
OL5PARAM	0x00000000	Overlay 5 disabled
OL6PARAM	0x00000000	Overlay 6 disabled
OL7PARAM	0x00000000	Overlay 7 disabled
CTRL	0x000**8**9003	**CROP=1** S0_FORMAT=9 (YUV420) IRQ_ENABLE=1 ENABLE=1

In this case, we have now changed the RGB size to reflect the portrait nature of the display. The S0PARAM_YBASE has been changed to 0x05 (40 pixels) to place the S0 plane down 40 pixels from the top of the screen. The cropping register is now also used to control the cropping extents. The CROP_XBASE is set to 0x05 (40 pixels) to move the origin of the S0 buffer to the (40,0) location within the buffer. The CROP_WIDTH/CROP_HEIGHT are also programmed to ensure that the resulting image in the output buffer is cropped to 240x240 pixels. Since the image no longer covers the entire output buffer, the S0BACKGROUND register is used to letterbox the image in black. The resulting image is shown below.

Figure 17-21. Example: Cropped QVGA

17.3.4 Upscale QVGA to VGA with Overlays

In this example, the image will be upscaled from QVGA to VGA resolution and displayed with the two overlays from the second example. Changes from the second example are shown in bold.

Table 17-7. Register Use for Conversion

Register	Value	Description
RGBBUF	*example1_rgb	Pointer to the output buffer.
RGBSIZE	0xFF2801E0	ALPHA=0xFF **WIDTH=0x280=640** **HEIGHT=0x1E0=480**
S0BUF	*morraine_y	Pointer to input Y buffer
S0UBUF	*morraine_u	Pointer to input U buffer
SOVBUF	*morraine_v	Pointer to input V buffer
S0PARAM	0x0000281E	WIDTH=0x28=40 (40*8=320 pixels) HEIGHT=0x1E=30 (30*8= 240 pixels)
S0BACKGROUND	0x00000000	Black background region
S0CROP	0x0000503C	WIDTH=0x50=640pixels HEIGHT=0x3C=320pixels
S0SCALE	0x08000800	XSCALE=0x0800=2x scale YSCALE=0x0800=2x scale

Table 17-7. Register Use for Conversion (continued)

Register	Value	Description
S0CSCCOEFF0 S0CSCCOEFF1 S0CSCCOEFF2	0x04030000 0x01230208 0x076b079b	YUV->RGB Coefficient Values
OL0	*overlay1_rgb	Pointer to control graphic
OL0SIZE	0x**23**000A02	**XBASE=0x23=280pixels** WIDTH=0x0A=80 pixels HEIGHT=0x02=16pixels
OL0PARAM	0x0000FF01	ALPHA=0xFF FORMAT=0x0 (RGB8888) ALPHA_CTRL=0 (embedded alpha) ENABLE=1
OL1	*logo_rgb	Pointer to logo graphic
OL1SIZE	0x**1936**1D06	**XBASE=0x19=200pixels** **YBASE=0x36=432pixels** WIDTH=0x1D=232pixels HEIGHT=0x06=48pixels
OL1PARAM	0x0000FF01	ALPHA=0xFF FORMAT=0x0 (RGB8888) ALPHA_CTRL=0 (embedded alpha) ENABLE=1
OL2PARAM	0x00000000	Overlay 2 disabled
OL3PARAM	0x00000000	Overlay 3 disabled
OL4PARAM	0x00000000	Overlay 4 disabled
OL5PARAM	0x00000000	Overlay 5 disabled
OL6PARAM	0x00000000	Overlay 6 disabled
OL7PARAM	0x00000000	Overlay 7 disabled
CTRL	0x000**c**9003	**SCALE=1** **CROP=1** S0_FORMAT=9 (YUV420) IRQ_ENABLE=1 ENABLE=1

The resulting image is shown in the figure below. Note that the overlays have moved in this image and that the overall image size is now larger than before.

Figure 17-22. Example: Upscale QVGA to VGA with Overlays

17.3.5 Downscale VGA to WQVGA (480x272) to fill screen

In this example, a VGA image will be downscaled to fix the extents of a 480x272 WQVGA display. This means that the aspect ratio of the resulting image will not match that of the source image, thus the scaling factors in the horizontal and vertical directions will differ from each other.

Table 17-8. Register Use for Conversion

Register	Value	Description
RGBBUF	*example_rgb	Pointer to the output buffer.
RGBSIZE	0xFFf1E0110	ALPHA=0xFF WIDTH=0x1E0=480 HEIGHT=0x110=272
S0BUF	*garden_y	Pointer to input Y buffer
S0UBUF	*garden_u	Pointer to input U buffer
SOVBUF	*garden_v	Pointer to input V buffer
S0PARAM	0x0000503C	WIDTH=0x50=80=640 pixels HEIGHT=0x3C=60=480 pixels
S0BACKGROUND	0x00000000	Black background region
S0CROP	0x00003C22	WIDTH=0x3C=480 pixels HEIGHT=0x22=272 pixels
S0SCALE	0x1C3C1555	YSCALE=0x1C3C=1/1.765x XSCALE=0x1555=1/1.333x
S0CSCCOEFF0 S0CSCCOEFF1 S0CSCCOEFF2	0x04030000 0x01230208 0x076b079b	YUV->RGB Coefficient Values
OL0PARAM	0x00000000	Overlay 0 disabled
OL1PARAM	0x00000000	Overlay 1 disabled

Table 17-8. Register Use for Conversion (continued)

Register	Value	Description
OL2PARAM	0x00000000	Overlay 2 disabled
OL3PARAM	0x00000000	Overlay 3 disabled
OL4PARAM	0x00000000	Overlay 4 disabled
OL5PARAM	0x00000000	Overlay 5 disabled
OL6PARAM	0x00000000	Overlay 6 disabled
OL7PARAM	0x00000000	Overlay 7 disabled
CTRL	0x000C9003	SCALE=1 CROP=1 S0_FORMAT=9 (YUV420) IRQ_ENABLE=1 ENABLE=1

Note that the scaling factors are computed as (source/dest)*4096, thus in the horizontal direction 640/480*4096=5461=0x1555. In the vertical direction, the scaling factor is computed as 480/272*4096=7228=0x1C3C. The original source image and resulting scaled images are shown below:

Original Image (640x480)

Scaled Image (480x272)

Figure 17-23. Example: Downscale VGA to WQVGA (480x272) to fill screen

17.3.6 Downscale VGA to QVGA with Overlapping Overlays

The final example will perform a 1/2x scaling of a VGA image to QVGA to maintain the aspect ratio. It will also add four overlays to present the image as if it were a photo album application.

Table 17-9. Register Use for Conversion

Register	Value	Description
RGBBUF	*example_rgb	Pointer to the output buffer.
RGBSIZE	0xFF1400F0	ALPHA=0xFF WIDTH=0x1E0=320 HEIGHT=0x110=240
S0BUF	*garden_y	Pointer to input Y buffer
S0UBUF	garden_u	Pointer to input U buffer
SOVBUF	garden_v	Pointer to input V buffer

Table 17-9. Register Use for Conversion (continued)

Register	Value	Description
S0PARAM	0x0000503C	WIDTH=0x50=80=640 pixels HEIGHT=0x3C=60=480 pixels
S0BACKGROUND	0x00000040	Dark Blue background region
S0CROP	0x0000281E	WIDTH=0x28=320 pixels HEIGHT=0x1E=240 pixels
S0SCALE	0x20002000	YSCALE=0x2000=1/2x XSCALE=0x1555=1/2x
S0OFFSET	0x08000800	XOFFSET=0x0800 (1/2 pixel) YOFFSET=0x0800 (1/2 pixel)
S0CSCCOEFF0 S0CSCCOEFF1 S0CSCCOEFF2	0x04030000 0x01230208 0x076b079b	YUV->RGB Coefficient Values
OL0	*prev_rgb	Pointer to “previous” graphic
OL0SIZE	0x0B1B0402	XBASE=0x0B=88pixels YBASE=0x1B=216pixels WIDTH=0x04=32 pixels HEIGHT=0x02=16pixels
OL0PARAM	0x0000FF01	ALPHA=0xFF FORMAT=0x0 (RGB8888) ALPHA_CTRL=0 (embedded alpha) ENABLE=1
OL1	*next_rgb	Pointer to “next” graphic
OL1SIZE	0x2D1B0402	XBASE=0x2D=360pixels YBASE=0x1B=216pixels WIDTH=0x04=32 pixels HEIGHT=0x02=16pixels
OL1PARAM	0x0000FF01	ALPHA=0xFF FORMAT=0x0 (RGB8888) ALPHA_CTRL=0 (embedded alpha) ENABLE=1
OL2	*text_overlay	Pointer to text graphic
OL2SIZE	0x00000A1E	XBASE=0x00=0pixels YBASE=0x00=0pixels WIDTH=0x0A=80pixels HEIGHT=0x1E=240pixels
OL2PARAM	0x0000FF01	ALPHA=0xFF FORMAT=0x0 (RGB8888) ALPHA_CTRL=0 (embedded alpha) ENABLE=1
OL3	*border_rgb	Pointer to rectangular border graphic
OL3SIZE	0x0A00281E	XBASE=0x0A=80pixels YBASE=0x00=0pixels WIDTH=0x28=320pixels HEIGHT=0x1E=240pixels
OL3PARAM	0x0000FF01	ALPHA=0xFF FORMAT=0x0 (RGB8888) ALPHA_CTRL=0 (embedded alpha) ENABLE=1
OL4PARAM	0x00000000	Overlay 4 disabled

Table 17-9. Register Use for Conversion (continued)

Register	Value	Description
OL5PARAM	0x00000000	Overlay 5 disabled
OL6PARAM	0x00000000	Overlay 6 disabled
OL7PARAM	0x00000000	Overlay 7 disabled
CTRL	0x000C9003	SCALE=1 CROP=1 S0_FORMAT=9 (YUV420) IRQ_ENABLE=1 ENABLE=1

The resulting image is shown below. Note that the “text” is rendered in a transparent overlay (overlay #2)on the right side of the screen. The background color (#000040) is dark blue and shows through the overlay as the background color. Overlay #3 applies a thin white alpha-blended border around the image to frame it. Overlays #0 and #1 generate the “Next>” and “<Prev” images alpha blended onto the image. Because these overlays are higher priority (lower numbered) than the border, they are used at these locations instead of overlay #3.

Figure 17-24. Example: Downscale VGA to QVGA with Overlapping Overlays

17.4 Programmable Registers

This section includes the programmable registers supported.

17.4.1 PXP Control Register 0 Description

The CTRL register contains controls for the PXP module.

HW_PXP_CTRL 0x000

HW_PXP_CTRL_SET	0x004
HW_PXP_CTRL_CLR	0x008
HW_PXP_CTRL_TOG	0x00C

Table 17-10. HW_PXP_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	RSVD4		INTERLACED_OUTPUT		INTERLACED_INPUT		RSVD3	ALPHA_OUTPUT	IN_PLACE	DELTA	CROP	SCALE	UPSAMPLE	SUBSAMPLE	S0_FORMAT				VFLIP	HFLIP	ROTATE		OUTPUT_RGB_FORMAT				RSVD2	RSVD1	IRQ_ENABLE	ENABLE

Table 17-11. HW_PXP_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	Set this bit to zero to enable normal PXP operation. Set this bit to one (default) to disable clocking with the PXP and hold it in its reset (lowest power) state. This bit can be turned on and then off to reset the PXP block to its default state.
30	**CLKGATE**	RW	0x1	This bit must be set to zero for normal operation. When set to one it gates off the clocks to the block.
29:28	**RSVD4**	RO	0x0	Reserved, always set to zero.
27:26	**INTERLACED_OUTPUT**	RW	0x0	Determines how the PXP writes it's output RGB data. Output interlacing should not be used in conjunction with input interlacing. Splitting frames into fields is most efficient using output interlacing. PROGRESSIVE = 0x0 All data written in progressive format to the RGBBUF Pointer. FIELD0 = 0x1 Interlaced output: only data for field 0 is written to the RGBBUF Pointer. FIELD1 = 0x2 Interlaced output: only data for field 1 is written to the RGBBUF2 Pointer. INTERLACED = 0x3 Interlaced output: data for field 0 is written to RGBBUF and data for field 1 is written to RGBBUF2.
25:24	**INTERLACED_INPUT**	RW	0x0	When set, causes the fetch side of the PXP to fetch every other line from the source buffers. This effectively produces one field of interlaced output data. Scaling should NOT be enabled for interlaced operation and only overlays with boundaries on 8x16 multiples are supported. PROGRESSIVE = 0x0 All data will be read and processed in progressive format. FIELD0 = 0x2 Interlaced, Field 0: only data for field 0 (even lines) is read/processed. FIELD1 = 0x3 Interlaced, Field 1: only data for field 1 (odd lines) is read/processed.
23	**RSVD3**	RO	0x0	Reserved, always set to zero.
22	**ALPHA_OUTPUT**	RW	0x0	Indicates, if 0, that the output buffer pixels should retain their alpha value from the computed alpha for that pixel. If 1, then the ALPHA field from the output buffer parameters register will be used.

Table 17-11. HW_PXP_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
21	**IN_PLACE**	RW	0x0	When set, this enables the PXP to perform an alpha blend operation on an existing buffer (output buffer is set to S0 buffer). In this case, the PXP will perform the alpha blending of the overlays into the source buffer. Since only pixels containing an overlay are processed, the PXP does this very efficiently.
20	**DELTA**	RO	0x0	Reserved for future use.
19	**CROP**	RW	0x0	Indicates that the S0 plane should use the cropping register to provide the extents for the output S0 buffer cropping. If not set, the input video cropping extents will be inferred from the S0 WIDTH and HEIGHT fields. When scaling, the CROP bit and controls should be used to specify the scaled image size in the RGB buffer.
18	**SCALE**	RW	0x0	Indicates that the output image should be scaled (only YUV/YCbCr images may be scaled -- RGB scaling is not supported). The XSCALE and YSCALE registers should be programmed accordingly. In addition, the CROP bit and the S0CROP registers should be programmed to ensure that the scaled image is properly cropped in the output buffer. When this bit is zero, the contents of the scaling registers are ignored.
17	**UPSAMPLE**	RO	0x0	Reserved for future use.
16	**SUBSAMPLE**	RO	0x0	Reserved for future use.
15:12	**S0_FORMAT**	RW	0x0	Source 0 buffer format. To select between YUV and YCbCr formats, see bit 31 of the CSCCOEFF0 register. RGB888 = 0x1 32-bit pixels (unpacked 24-bit format) RGB565 = 0x4 16-bit pixels RGB555 = 0x5 16-bit pixels YUV422 = 0x8 16-bit pixels YUV420 = 0x9 16-bit pixels
11	**VFLIP**	RW	0x0	Indicates that the output buffer should be flipped vertically (effect applied before rotation).
10	**HFLIP**	RW	0x0	Indicates that the output buffer should be flipped horizontally (effect applied before rotation).
9:8	**ROTATE**	RW	0x0	Indicates the clockwise rotation to be applied at the output buffer. The rotation effect is defined as occurring after the FLIP_X and FLIP_Y permutation. ROT_0 = 0x0 ROT_90 = 0x1 ROT_180 = 0x2 ROT_270 = 0x3
7:4	**OUTPUT_RGB_FORMAT**	RW	0x0	Target RGB framebuffer format. ARGB8888 = 0x0 32-bit pixels RGB888 = 0x1 32-bit pixels (unpacked 24-bit format) RGB888P = 0x2 24-bit pixels (packed 24-bit format) ARGB1555 = 0x3 16-bit pixels RGB565 = 0x4 16-bit pixels RGB555 = 0x5 16-bit pixels
3	**RSVD2**	RO	0x0	Reserved, always set to zero.
2	**RSVD1**	RW	0x0	Program this field to 0x0.

Table 17-11. HW_PXP_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
1	**IRQ_ENABLE**	RW	0x0	Interrupt enable. NOTE: When using the HW_PXP_NEXT functionality to reprogram the PXP, the new value of this bit will be used and may therefore enable or disable an interrupt unintentionally.
0	**ENABLE**	RW	0x0	Enables PXP operation with specified parameters. The ENABLE bit will remain set while the PXP is active and will be cleared once the current operation completes. Software should use the IRQ bit in the HW_PXP_STAT when polling for PXP completion.

DESCRIPTION:

The Control register contains the primary controls for the PXP block. The present bits indicate which of the sub-features of the block are present in the hardware.

EXAMPLE:

```
HW_PXP_CTRL_SET(BM_PXP_CTRL_SFTRST);
HW_PXP_CTRL_CLR(BM_PXP_CTRL_SFTRST | BM_PXP_CTRL_CLKGATE);
```

17.4.2 PXP Status Register Description

The PXP Interrupt Status register provides interrupt status information.

HW_PXP_STAT	0x010
HW_PXP_STAT_SET	0x014
HW_PXP_STAT_CLR	0x018
HW_PXP_STAT_TOG	0x01C

Table 17-12. HW_PXP_STAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BLOCKX								BLOCKY								RSVD2								AXI_ERROR_ID				RSVD1	AXI_READ_ERROR	AXI_WRITE_ERROR	IRQ

Table 17-13. HW_PXP_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**BLOCKX**	RO	0x00	Indicates the X coordinate of the block currently being rendered.
23:16	**BLOCKY**	RO	0x00	Indicates the X coordinate of the block currently being rendered.
15:8	**RSVD2**	RO	0x000000	Reserved, always set to zero.
7:4	**AXI_ERROR_ID**	RO	0x0	Indicates the AXI ID of the failing bus operation.
3	**RSVD1**	RO	0x0	Reserved, always set to zero.

Table 17-13. HW_PXP_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**AXI_READ_ERROR**	RW	0x0	Indicates PXP encountered an AXI read error and processing has been terminated.
1	**AXI_WRITE_ERROR**	RW	0x0	Indicates PXP encountered an AXI write error and processing has been terminated.
0	**IRQ**	RW	0x0	Indicates current PXP interrupt status. The IRQ is routed through the pxp_irq when the IRQ_ENABLE bit in the control register is set.

DESCRIPTION:

This register provides PXP interrupt status and the current X/Y block coordinate that is being processed.

EXAMPLE:

```
HW_PXP_STAT_CLR(BM_PXP_STAT_IRQ); // clear CSC interrupt
```

17.4.3 RGB Output Frame Buffer Pointer Description

RGB Output Framebuffer Pointer. This register points to the beginning of the RGB output frame buffer. This pointer is used for progressive format and field 0 when generating interlaced output.

HW_PXP_RGBBUF 0x020

Table 17-14. HW_PXP_RGBBUF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 17-15. HW_PXP_RGBBUF Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x0	Current address pointer for the output frame buffer. The address MUST be word-aligned for proper PXP operation.

DESCRIPTION:

This register is used by the logic to point to the current output location for the RGB frame buffer.

EXAMPLE:

```
HW_PXP_RGBBUF_WR( buffer );
```

17.4.4 RGB Output Frame Buffer Pointer #2 Description

RGB Output Framebuffer Pointer #2. This register points to the beginning of the RGB output frame buffer for field 1 when generating interlaced output.

HW_PXP_RGBBUF2 0x030

Table 17-16. HW_PXP_RGBBUF2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 17-17. HW_PXP_RGBBUF2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x0	Current address pointer for the output frame buffer. The address MUST be word-aligned for proper PXP operation.

DESCRIPTION:

This register is used by the logic to point to the current output location for the field 1 RGB frame buffer.

EXAMPLE:

```
HW_PXP_RGBBUF_WR( field0 ); // buffer for interlaced field 0
HW_PXP_RGBBUF2_WR( field1 ); // buffer for interlaced field 1
```

17.4.5 PXP Output Buffer Size Description

This register contains framebuffer size information for the output RGB buffer (independent of the rotation). When rotating the framebuffer, the PXP will automatically modify the output WIDTH/HEIGHT to accomodate the rotated size.

HW_PXP_RGBSIZE 0x040

Table 17-18. HW_PXP_RGBSIZE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ALPHA								WIDTH												HEIGHT											

Table 17-19. HW_PXP_RGBSIZE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**ALPHA**	RW	0x00	When generating an output RGB buffer with an alpha component, the value in this field will be used.
23:12	**WIDTH**	RW	0x0	Indicates number of horizontal PIXELS in the image (non-rotated). The image size IS required to be a multiple of 8 pixels. The PXP will handle clipping the pixel output at this boundary.
11:0	**HEIGHT**	RW	0x0	Indicates the number of vertical PIXELS in the image (non-rotated). The image size IS required to be a multiple of 8 pixels. The PXP will handle clipping the pixel output at this boundary.

DESCRIPTION:

This register sets the size of the output frame buffer. The frame buffer need not be a multiple of 8x8 pixels. Partial 8x8 blocks will be written for blocks that extend beyond the extents of the frame buffer.

EXAMPLE:

```
HW_PXP_RGBSIZE.U.WIDTH=320; // set width
HW_PXP_RGBSIZE.U.HEIGHT=240; // set height
HW_PX_RGBSIZE_WR( BF_PXP_RGBSIZE_WIDTH(320) | BF_PXP_RGBSIZE_HEIGHT(240) );
```

17.4.6 PXP Source 0 (video) Input Buffer Pointer Description

S0 Input Buffer Pointer. This should be programmed to the starting address of the RGB data or Y (luma) data for the S0 plane.

HW_PXP_S0BUF 0x050

Table 17-20. HW_PXP_S0BUF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 17-21. HW_PXP_S0BUF Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x0	Address pointer for the S0 RGB or Y (luma) input buffer. The address MUST be word-aligned for proper PXP operation.

DESCRIPTION:

This register contains the pointer to the Luma/RGB buffer.

EXAMPLE:

```
HW_PXP_S0BUF_WR(image_rgb); // RGB image
HW_PXP_S0BUF_WR(image_y); // Y (luma) image data
HW_PXP_S0UBUF_WR(image_u); // U (Cb) image data
HW_PXP_S0VBUF_WR(image_v); // V (Cr) image data
```

17.4.7 Source 0 U/Cb Input Buffer Pointer Description

S0 Chroma (U/Cb) Input Buffer Pointer. This register points to the beginning of the Source 0 U/Cb input buffer.

HW_PXP_S0UBUF 0x060

Table 17-22. HW_PXP_S0UBUF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 17-23. HW_PXP_S0UBUF Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x0	Address pointer for the S0 (video) U/Cb Chroma input buffer. The address MUST be word-aligned for proper PXP operation.

DESCRIPTION:

This register contains the pointer to the Chroma U/Cb buffer when performing colorspace conversion. This register is unused when processing RGB data.

EXAMPLE:

```
HW_PXP_S0BUF_WR(image_y); // Y (luma) image data
HW_PXP_S0UBUF_WR(image_u); // U (Cb) image data
HW_PXP_S0VBUF_WR(image_v); // V (Cr) image data
```

17.4.8 Source 0 V/Cr Input Buffer Pointer Description

S0 Chroma (V/Cr) Input Buffer Pointer. This register points to the beginning of the Source 0 V/Cr input buffer.

HW_PXP_S0VBUF 0x070

Table 17-24. HW_PXP_S0VBUF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 17-25. HW_PXP_S0VBUF Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x0	Address pointer for the S0 (video) V/Cr Chroma input buffer. The address MUST be word-aligned for proper PXP operation.

DESCRIPTION:

This register contains the pointer to the Chroma V/Cr buffer when performing colorspace conversion. This register is unused when processing RGB data.

EXAMPLE:

```
HW_PXP_S0BUF_WR(image_y); // Y (luma) image data
HW_PXP_S0UBUF_WR(image_u); // U (Cb) image data
HW_PXP_S0VBUF_WR(image_v); // V (Cr) image data
```

17.4.9 PXP Source 0 (video) Buffer Parameters Description

This register contains buffer information for the S0 input RGB/YUV buffer.

HW_PXP_S0PARAM 0x080

Table 17-26. HW_PXP_S0PARAM

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XBASE								YBASE								WIDTH								HEIGHT							

Table 17-27. HW_PXP_S0PARAM Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**XBASE**	RW	0x00	This field indicates the horizontal offset location (in 8x8 block) of the S0 buffer within the output frame buffer.
23:16	**YBASE**	RW	0x00	This field indicates the vertical offset location (in 8x8 block) of the S0 buffer within the output frame buffer.
15:8	**WIDTH**	RW	0x0	Indicates number of horizontal 8x8 blocks in the image (non-rotated).
7:0	**HEIGHT**	RW	0x0	Indicates the number of vertical 8x8 blocks in the image (non-rotated).

DESCRIPTION:

The S0 Parameter register contains the size of the S0 input buffer (WIDTH, HEIGHT) as well as provides an offset for the display of this buffer within the output frame buffer (XBASE,YBASE). All four values are in terms of 8x8 pixel blocks.

EXAMPLE:

```
HW_PXP_S0PARAM_WR(0x0101281E); // S0 buffer will appear at offset (8,8) in the RGB buffer.
// the size is 0x28 (40*8=320 pixels) by 0x1E (30*8=240 pixels)
```

17.4.10 Source 0 Background Color Description

S0 Background Pixel Color. This register provides a pixel value used when processing blocks outside of the region specified by the S0SIZE register. This value can effectively be used to set the color of the letterboxing region around a video image.

HW_PXP_S0BACKGROUND 0x090

Table 17-28. HW_PXP_S0BACKGROUND

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COLOR																															

Table 17-29. HW_PXP_S0BACKGROUND Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COLOR**	RW	0x0	Background color (in 32bpp format) for any pixels not in the S0 buffer range specified in the S0SIZE register.

DESCRIPTION:

This register contains a pixel value to be used for any S0 blocks that fall outside the S0 extents. This is effectively a background or letterbox color.

EXAMPLE:

```
HW_PXP_S0BACKGROUND_WR(0x00000000); // letterbox is black
HW_PXP_S0BACKGROUND_WR(0x00800000); // letterbox is dark red
HW_PXP_S0BACKGROUND_WR(0x00008000); // letterbox is dark green
HW_PXP_S0BACKGROUND_WR(0x00000080); // letterbox is dark blue
```

17.4.11 Source 0 Cropping Register Description

This register contains controls for image/video cropping. XBASE and YBASE select the origin of the S0 buffer for PXP operations. The WIDTH and HEIGHT determine the visible size of the selected region in the output frame buffer. Software should program the input framebuffer cropped width/height values into these fields. Cropping is applied in the output buffer, therefore after any scaling operations. Scaled regions may need to be cropped to avoid artifacts at the edge of a scaled region.

HW_PXP_S0CROP 0x0A0

Table 17-30. HW_PXP_S0CROP

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XBASE								YBASE								WIDTH								HEIGHT							

Table 17-31. HW_PXP_S0CROP Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**XBASE**	RW	0x00	This field indicates the horizontal offset (in terms of 8-pixel blocks) into the S0 buffer which is considered the origin of the image. This allows selection of a subset of a source image for processing.
23:16	**YBASE**	RW	0x00	This field indicates the vertical offset (in terms of 8-pixel blocks) into the S0 buffer which is considered the origin of the image. This allows selection of a subset of a source image for processing.

Table 17-31. HW_PXP_S0CROP Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:8	**WIDTH**	RW	0x00	Ouput buffer cropped video width (in terms of 8 pixel blocks). This field should be programmed to the desired cropped width of the S0 plane in the output buffer. When scaling is not used, this value is effectively the width of the input buffer that should appear in the output buffer. For scaling operations, it's important that this field be programmed to the width of the scaled size of the S0 output image.
7:0	**HEIGHT**	RW	0x00	Input buffer cropped video height (in terms of 8 pixel blocks). This field should be programmed to the desired cropped height of the S0 plan in the output buffer. When scaling is not used, this value is effectively the height of the input buffer that should appear in the output buffer. For scaling operations, it's important that this field be programmed to the height of the scaled size of the S0 output image.

DESCRIPTION:

The cropping register can be used to specify cropping extents for S0 plane in the output buffer. It is only used if the CROP bit is set in the PXP_CTRL register is set. When this bit is not set, no cropping of the input image will be performed and the PXP will default to using the S0 WIDTH and HEIGHT parameters. Cropping should always be used when scaling images since the PXP cannot determine the scaled image size.

EXAMPLE:

```
HW_PXP_S0CROP_WR(0x02021810); // S0 origin is at (16,16) -- 0x0202
// output width is 192 (0x18->24*8=192 pixels)
// output height is 128 (0x10->16*8=128 pixels)
```

17.4.12 Source 0 Scale Factor Register Description

S0 Scale Factor. This register provides the scale factor for the S0 (video) buffer.

HW_PXP_S0SCALE 0x0B0

Table 17-32. HW_PXP_S0SCALE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2		YSCALE														RSVD1		XSCALE													

Table 17-33. HW_PXP_S0SCALE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSVD2**	RO	0x00	Reserved, always set to zero.
29:16	**YSCALE**	RW	0x1000	This is a two bit integer and 12 bit fractional representation (##.####_####_####) of the Y scaling factor for the S0 source buffer.

Table 17-33. HW_PXP_S0SCALE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:14	**RSVD1**	RO	0x00	Reserved, always set to zero.
13:0	**XSCALE**	RW	0x1000	This is a two bit integer and 12 bit fractional representation (##.####_####_####) of the X scaling factor for the S0 source buffer.

DESCRIPTION:

The maximum down scaling factor is 1/2 such that the output image in either axis is half the size of the source. The maximum up scaling factor is 2^12 for either axis. The reciprocal of the scale factor should be loaded into this register. To reduce the S0 buffer by a factor of two in the output frame buffer, a value of 10.0000_0000_0000 should be loaded into this register. The scale up by a factor of 4, the value of 1/4, or 00.0100_0000_0000, should be loaded into this register. To scale up by 8/5, the value of 00.1010_0000_0000 should be loaded.

EXAMPLE:

```
HW_PXP_S0SCALE_WR(0x10001000); // 1:1 scaling (0x1.000)
HW_PXP_S0SCALE_WR(0x08000800); // 2x scaling (0x0.800)
HW_PXP_S0SCALE_WR(0x20002000); // 1/2x scaling (0x2.000)
```

17.4.13 Source 0 Scale Offset Register Description

S0 Scale Offset. This register provides the initial scale offset for the S0 (video) buffer.

HW_PXP_S0OFFSET 0x0C0

Table 17-34. HW_PXP_S0OFFSET

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2				YOFFSET												RSVD1				XOFFSET											

Table 17-35. HW_PXP_S0OFFSET Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**RSVD2**	RO	0x00	Reserved, always set to zero.
27:16	**YOFFSET**	RW	0x000	This is a 12 bit fractional representation (0.####_####_####) of the Y scaling offset. This represents a fixed block offset which gets added to the scaled block address to determine source data for the scaling engine.
15:12	**RSVD1**	RO	0x00	Reserved, always set to zero.
11:0	**XOFFSET**	RW	0x000	This is a 12 bit fractional representation (0.####_####_####) of the X scaling offset. This represents a fixed block offset which gets added to the scaled block address to determine source data for the scaling engine.

DESCRIPTION:

The X and Y offset provides the ability to access the source image with a per pixel or per sub-pixel granularity. To shift the source input image by a single pixel, for example, a value of 0x200 would be loaded into this offset field. Since a pixel block is 8x8 pixels, 0x200 (or 1/8), will provide a fixed offset of 1 pixel for the entire PXP operation. The fixed offset values can also be used for sub-pixel adjustments in the bilinear scaling filter. For example, when scaling an image down by a factor of 2, an initial offset of 0x0 would result in sub-sampling every other pixel. If a fixed offset of 0x100 (1/16) is programmed, all pixels are used in scaling the final output pixel value.

EXAMPLE:

```
Empty Example.
HW_PXP_S0SCALE_WR(0x20002000); // 1/2x scaling (0x2.000)
HW_PXP_S0OFFSET_WR(0x01000100); // half-pixel offset in both X and Y to ensure averaging versus
pixel replication
```

17.4.14 Color Space Conversion Coefficient Register 0 Description

This register contains color space conversion coefficients in two's compliment notation.

HW_PXP_CSCCOEFF0 0x0D0

Table 17-36. HW_PXP_CSCCOEFF0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
YCBCR_MODE	RSVD1																														
												C0																			
																UV_OFFSET															
																								Y_OFFSET							

Table 17-37. HW_PXP_CSCCOEFF0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**YCBCR_MODE**	RW	0x0	Set to 1 when performing YCbCr conversion to RGB. Set to 0 when converting YUV to RGB data. This bit changes the behavior of the scaler when performing U/V scaling.
30:29	**RSVD1**	RO	0x00	Reserved, always set to zero.
28:18	**C0**	RW	0x100	Two's compliment Y multiplier coefficient. YUV=0x100 (1.000) YCbCr=0x12A (1.164)
17:9	**UV_OFFSET**	RW	0x180	Two's compliment phase offset implicit for UV data. YUV=YCbCr=0x180 (typically -128 or 0x180 to indicate a -0.5 to 0.5 range)
8:0	**Y_OFFSET**	RW	0x000	Two's compliment amplitude offset implicit in the Y data. For YUV, this is typically 0 and for YCbCr, this is typically -16 (0x1F0)

DESCRIPTION:

The Coeffient 0 register contains coeffients used in the color space conversion algorithm. The Y and UV offsets are added to the source buffer to normalize them before the conversion. C0 is the coeffient that is used to multiply the luma component of the data for all three RGB components.

EXAMPLE:

```
// The equations used for Colorspace conversion are:
// R = C0*(Y+YOFFSET) + C1(V+UV_OFFSET)
// G = C0*(Y+YOFFSET) + C3(U+UV_OFFSET) + C2(V+UV_OFFSET)
// R = C0*(Y+YOFFSET) + C4(U+UV_OFFSET)
HW_PXP_CSCCOEFF0_WR(0x04030000); // YUV coefficients: C0, Yoffset, UVoffset
HW_PXP_CSCCOEFF1_WR(0x01230208); // YUV coefficients: C1, C4
HW_PXP_CSCCOEFF2_WR(0x076B079b); // YUV coefficients: C2, C3
```

17.4.15 Color Space Conversion Coefficient Register 1 Description

This register contains color space conversion coefficients in two's compliment notation.

HW_PXP_CSCCOEFF1 0x0E0

Table 17-38. HW_PXP_CSCCOEFF1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1					C1											RSVD0					C4										

Table 17-39. HW_PXP_CSCCOEFF1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:27	**RSVD1**	RO	0x00	Reserved, always set to zero.
26:16	**C1**	RW	0x123	Two's compliment Red V/Cr multiplier coefficient. YUV=0x123 (1.140) YCbCr=0x198 (1.596)
15:11	**RSVD0**	RO	0x00	Reserved, always set to zero.
10:0	**C4**	RW	0x208	Two's compliment Blue U/Cb multiplier coefficient. YUV=0x208 (2.032) YCbCr=0x204 (2.017)

DESCRIPTION:

The Coeffient 1 register contains coeffients used in the color space conversion algorithm.

C1 is the coeffient that is used to multiply the chroma (Cr/V) component of the data for the red component.

C4 is the coeffient that is used to multiply the chroma (Cb/U) component of the data for the blue component.

Both values should be coded as a two's compliment fixed point number with 8 bits right of the decimal.

EXAMPLE:

```
HW_PXP_CSCCOEFF0_WR(0x04030000); // YUV coefficients: C0, Yoffset, UVoffset
HW_PXP_CSCCOEFF1_WR(0x01230208); // YUV coefficients: C1, C4
HW_PXP_CSCCOEFF2_WR(0x076B079b); // YUV coefficients: C2, C3
```

17.4.16 Color Space Conversion Coefficient Register 2 Description

This register contains color space conversion coefficients in two's compliment notation.

HW_PXP_CSCCOEFF2	0x0F0

Table 17-40. HW_PXP_CSCCOEFF2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1					C2											RSVD0					C3										

Table 17-41. HW_PXP_CSCCOEFF2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:27	**RSVD1**	RO	0x00	Reserved, always set to zero.
26:16	**C2**	RW	0x79b	Two's compliment Green V/Cr multiplier coefficient. YUV=0x76B (-0.581) YCbCr=0x730 (-0.813)
15:11	**RSVD0**	RO	0x00	Reserved, always set to zero.
10:0	**C3**	RW	0x76c	Two's compliment Green U/Cb multiplier coefficient. YUV=0x79C (-0.394) YCbCr=0x79C (-0.392)

DESCRIPTION:

The Coeffient 2 register contains coeffients used in the color space conversion algorithm.

C2 is the coeffient that is used to multiply the chroma (Cr/V) component of the data for the green component.

C3 is the coeffient that is used to multiply the chroma (Cb/U) component of the data for the green component.

Both values should be coded as a two's compliment fixed point number with 8 bits right of the decimal.

EXAMPLE:

```
// NOTE: The default values for the CSCCOEFF2 register are incorrect. C2 should be 0x76B and C3
should be 0x79C for proper operation.
HW_PXP_CSCCOEFF0_WR(0x04030000); // YUV coefficients: C0, Yoffset, UVoffset
HW_PXP_CSCCOEFF1_WR(0x01230208); // YUV coefficients: C1, C4
HW_PXP_CSCCOEFF2_WR(0x076B079b); // YUV coefficients: C2, C3
```

17.4.17 PXP Next Frame Pointer Description

This register contains a pointer to a data structure used to reload the PXP registers at the end of the current frame.

HW_PXP_NEXT	0x100
HW_PXP_NEXT_SET	0x104
HW_PXP_NEXT_CLR	0x108
HW_PXP_NEXT_TOG	0x10C

Table 17-42. HW_PXP_NEXT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
POINTER																														RSVD	ENABLED

Table 17-43. HW_PXP_NEXT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:2	**POINTER**	RW	0x0	A pointer to a data structure containing register values to be used when processing the next frame. The pointer must be 32-bit aligned and should reside in on-chip or off-chip memory.
1	**RSVD**	RO	0x0	Reserved, always set to zero.
0	**ENABLED**	RO	0x0	Indicates that the "next frame" functionality has been enabled. This bit reflects the status of the hardware semaphore indicating that a reload operation is pending at the end of the current frame.

DESCRIPTION:

To enable this functionality, software must write this register while the PXP is processing the current data frame (if the PXP is currently idle, this will also initiate an immediate load of registers from the pointer).

The process of writing this register (WRITE operation) will set a semaphore in hardware to notify the control logic that a register reload operation must be

performed when the current frame processing is complete. At the end of a frame, the PXP will fetch the register settings

from this location, signal an interrupt to software, then proceed with rendering the next frame of data.

Software may cancel the reload operation by issuing a CLEAR operation to this register. SET and TOGGLE operations should not be used when addressing this register.

All registers will be reloaded with the exception of the following: STAT, CSCCOEFFn, NEXT, VERSION. All other registers will be loaded in the order they appear in the register map.

EXAMPLE:

```
// create register command structure in memory
u32* pxp_commands0[48], pxp_commands1;
u32 rc;
// initialize control structure for frame 0
pxp_commands0[0] = ...; // CTRL
pxp_commands0[1] = ...; // RGB Buffer
...
pxp_commands0[47] = ..; // Overlay7 param2
// initialize control structure for frame 1
pxp_commands1[0] = ...; // CTRL
pxp_commands1[1] = ...; // RGB Buffer
...
pxp_commands1[47] = ..; // Overlay7 param2
// poll until a command isn't queued
while (rc=HW_PXP_NEXT_RD() & BM_PXP_NEXT_ENABLED );
HW_PXP_NEXT_WR(pxp_commands0); // enable PXP operation 0 via command pointer
```

```
// poll until first command clears
while (rc=HW_PXP_NEXT_RD() & BM_PXP_NEXT_ENABLED );
HW_PXP_NEXT_WR(pxp_commands1); // enable PXP operation 1 via command pointer
```

17.4.18 PXP S0 Color Key Low Description

This register contains the color key low value for the S0 buffer.

HW_PXP_S0COLORKEYLOW 0x180

Table 17-44. HW_PXP_S0COLORKEYLOW

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1								PIXEL																							

Table 17-45. HW_PXP_S0COLORKEYLOW Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD1**	RO	0x00	Reserved, always set to zero.
23:0	**PIXEL**	RW	0xFFFFFF	Low range of RGB color key applied to S0 buffer. To disable S0 colorkeying, set the low colorkey to 0xFFFFFF and the high colorkey to 0x000000.

DESCRIPTION:

When processing an image, the if the PXP finds a pixel in the background image with a color that falls in the range from the S0COLORKEYLOW to S0COLORKEYHIGH

range, it will substitute the color found in the matching overlay. If no overlay is present or if the overlay also matches its colorkey range, the s0background color is used.

EXAMPLE:

```
// colorkey values between
HW_PXP_S0COLORKEYLOW_WR (0x008000); // medium green and
HW_PXP_S0COLORKEYHIGH_WR(0x00FF00); // light green
```

17.4.19 PXP S0 Color Key High Description

This register contains the color key high value for the S0 buffer.

HW_PXP_S0COLORKEYHIGH 0x190

Table 17-46. HW_PXP_S0COLORKEYHIGH

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1								PIXEL																							

Table 17-47. HW_PXP_S0COLORKEYHIGH Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD1**	RO	0x00	Reserved, always set to zero.
23:0	**PIXEL**	RW	0x0	High range of RGB color key applied to S0 buffer. To disable S0 colorkeying, set the low colorkey to 0xFFFFFF and the high colorkey to 0x000000.

DESCRIPTION:

When processing an image, the if the PXP finds a pixel in the background image with a color that falls in the range from the S0COLORKEYLOW to S0COLORKEYHIGH

range, it will substitute the color found in the matching overlay. If no overlay is present or if the overlay also matches its colorkey range, the s0background color is used.

EXAMPLE:

```
// colorkey values between
HW_PXP_S0COLORKEYLOW_WR (0x008000); // medium green and
HW_PXP_S0COLORKEYHIGH_WR(0x00FF00); // light green
```

17.4.20 PXP Overlay Color Key Low Description

This register contains the color key low value for the OL buffer.

HW_PXP_OLCOLORKEYLOW 0x1A0

Table 17-48. HW_PXP_OLCOLORKEYLOW

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1								PIXEL																							

Table 17-49. HW_PXP_OLCOLORKEYLOW Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD1**	RO	0x00	Reserved, always set to zero.
23:0	**PIXEL**	RW	0xFFFFFF	Low range of RGB color key applied to OL buffer. Each overlay has an independent colorkey enable.

DESCRIPTION:

When processing an image, the if the PXP finds a pixel in the current overlay image with a color that falls in the range from the OLCOLORKEYLOW to OLCOLORKEYHIGH

range, it will use the S0 pixel value for that location. If no S0 image is present or if the S0 image also matches its colorkey range, the s0background color is used.

Colorkey operations are higher priority than alpha or ROP operations.

EXAMPLE:

```
// colorkey values between
HW_PXP_OLCOLORKEYLOW_WR (0x000000); // black and
```

```
HW_PXP_OLCOLORKEYHIGH_WR(0x800000); // medium red
```

17.4.21 PXP Overlay Color Key High Description

This register contains the color key high value for the OL buffer.

HW_PXP_OLCOLORKEYHIGH 0x1B0

Table 17-50. HW_PXP_OLCOLORKEYHIGH

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1								PIXEL																							

Table 17-51. HW_PXP_OLCOLORKEYHIGH Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD1**	RO	0x00	Reserved, always set to zero.
23:0	**PIXEL**	RW	0x0	High range of RGB color key applied to OL buffer. Each overlay has an independent colorkey enable.

DESCRIPTION:

When processing an image, the if the PXP finds a pixel in the current overlay image with a color that falls in the range from the OLCOLORKEYLOW to OLCOLORKEYHIGH

range, it will use the S0 pixel value for that location. If no S0 image is present or if the S0 image also matches its colorkey range, the s0background color is used.

Colorkey operations are higher priority than alpha or ROP operations.

EXAMPLE:

```
// colorkey values between
HW_PXP_OLCOLORKEYLOW_WR (0x000000); // black and
HW_PXP_OLCOLORKEYHIGH_WR(0x800000); // medium red
```

17.4.22 PXP Debug Control Register Description

This register controls the debug features of the PXP.

HW_PXP_DEBUGCTRL 0x1D0

Table 17-52. HW_PXP_DEBUGCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2																							RSVD1	SELECT							

Table 17-53. HW_PXP_DEBUGCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:9	**RSVD2**	RO	0x0	Reserved, always set to zero.
8	**RSVD1**	RW	0x0	Program this field to 0x0.
7:0	**SELECT**	RW	0x00	Index into one of the PXP debug registers. The data for the selected register will be returned NONE = 0x0 None CTRL = 0x1 Control Debug S0REGS = 0x2 S0 Debug S0BAX = 0x3 S0 BA X Scale S0BAY = 0x4 S0 BA Y Scale PXBUF = 0x5 PXBUF Debug ROTATION = 0x6 Rotation Debug ROTBUF0 = 0x7 Rotation Buffer 0 ROTBUF1 = 0x8 Rotation Buffer 1

DESCRIPTION:

This register controls the PXP Debug features. This register is not intended for customer use.

EXAMPLE:

NA

17.4.23 PXP Debug Register Description

This register returns selected debug register values.

HW_PXP_DEBUG 0x1E0

Table 17-54. HW_PXP_DEBUG

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DATA																															

Table 17-55. HW_PXP_DEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**DATA**	RO	0x000000	Debug data

DESCRIPTION:

Debug register. Select the appropriate register in debug control and the values are returned here.

EXAMPLE:

NA

17.4.24 PXP Version Register Description

This register always returns a known read value for debug purposes it indicates the version of the block.

HW_PXP_VERSION 0x1F0

Table 17-56. HW_PXP_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 17-57. HW_PXP_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x02	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x0	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
if (HW_PXP_VERSION.B.MAJOR != 2) Error();
```

17.4.25 PXP Overlay 0 Buffer Pointer Description

Overlay 0 Buffer Address Pointer. This register points to the beginning of the RGB Overlay 0 input buffer.

HW_PXP_OL0 0x200

Table 17-58. HW_PXP_OL0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 17-59. HW_PXP_OL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x0	Address pointer for the overlay 0 buffer. The address MUST be word-aligned for proper PXP operation.

DESCRIPTION:

This register points to the is used by the logic to point to the current output location for the RGB frame buffer.

EXAMPLE:

```
u32* overlay_ptr;
HW_PXP_OLn_WR(0,overlay_ptr);
```

17.4.26 PXP Overlay 0 Size Description

This register contains buffer size/location information for the Overlay 0 input buffer.

HW_PXP_OL0SIZE 0x210

Table 17-60. HW_PXP_OL0SIZE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XBASE								YBASE								WIDTH								HEIGHT							

Table 17-61. HW_PXP_OL0SIZE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**XBASE**	RW	0x00	This field indicates the X-coordinate (in blocks) of the top-left 8x8 block in the overlay within the output frame buffer.
23:16	**YBASE**	RW	0x00	This field indicates the Y-coordinate (in blocks) of the top-left 8x8 block in the overlay within the output frame buffer.
15:8	**WIDTH**	RW	0x0	Indicates number of horizontal 8x8 blocks in the image (non-rotated).
7:0	**HEIGHT**	RW	0x0	Indicates the number of vertical 8x8 blocks in the image (non-rotated).

DESCRIPTION:

This register contains information about Overlay 0 indicating the size of the overlay (in 8x8 blocks) and the overlay's location within the output frame buffer (in 8x8 blocks).

EXAMPLE:

```
HW_PXP_OLnSIZE_WR(0,0x10000401); // 32x8 overlay at offset +128+0
```

17.4.27 PXP Overlay 0 Parameters Description

This register contains buffer parameters for the Overlay 0 input buffer.

HW_PXP_OL0PARAM 0x220

Table 17-62. HW_PXP_OL0PARAM

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1												ROP				ALPHA								FORMAT				ENABLE_COLORKEY	ALPHA_CNTL		ENABLE

Table 17-63. HW_PXP_OL0PARAM Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:20	**RSVD1**	RO	0x0000	Reserved, always set to zero.
19:16	**ROP**	RW	0x0	Indicates a raster operation to perform when enabled. Raster operations are enabled through the ALPHA_CNTL field. MASKOL = 0x0 OL AND S0 MASKNOTOL = 0x1 nOL AND S0 MASKOLNOT = 0x2 OL AND nS0 MERGEOL = 0x3 OL OR S0 MERGENOTOL = 0x4 nOL OR S0 MERGEOLNOT = 0x5 OL OR nS0 NOTCOPYOL = 0x6 nOL NOT = 0x7 nS0 NOTMASKOL = 0x8 OL NAND S0 NOTMERGEOL = 0x9 OL NOR S0 XOROL = 0xA OL XOR S0 NOTXOROL = 0xB OL XNOR S0
15:8	**ALPHA**	RW	0x0	Alpha modifier used when the ALPHA_MULTIPLY or ALPHA_OVERRIDE bits are set. The output alpha value will either be replaced (ALPHA_OVERRIDE) or scaled (ALPHA_MULTIPLY) when enabled in the ALPHA_CNTL field.
7:4	**FORMAT**	RW	0x0	Indicates the input buffer format for overlay 0. ARGB8888 = 0x0 32-bit pixels with alpha RGB888 = 0x1 32-bit pixels without alpha (unpacked 24-bit format) ARGB1555 = 0x3 16-bit pixels with alpha RGB565 = 0x4 16-bit pixels without alpha RGB555 = 0x5 16-bit pixels without alpha
3	**ENABLE_COLORKEY**	RW	0x0	Indicates that colorkey functionality is enabled for this overlay. Pixels found in the overlay colorkey range will be displayed as transparent (the S0 pixel will be used).
2:1	**ALPHA_CNTL**	RW	0x0	Determines how the alpha value is constructed for this overlay. Indicates that the value in the ALPHA field should be used instead of the alpha values present in the input pixels. Embedded = 0x0 Indicates that the OL pixel alpha value will be used to blend the OL with S0. The ALPHA field is ignored. Override = 0x1 Indicates that the value in the ALPHA field should be used instead of the alpha values present in the input pixels. Multiply = 0x2 Indicates that the value in the ALPHA field should be used to scale all pixel alpha values. Each pixel alpha is multiplied by the value in the ALPHA field. ROPs = 0x3 Enable ROPs. The ROP field indicates an operation to be performed on the overlay and S0 pixels.
0	**ENABLE**	RW	0x0	Indicates that the overlay is active for this operation.

DESCRIPTION:

The S1 Overlay 0 Parameter register provides additional controls for Overlay 0.

EXAMPLE:

```
u32 olparam;
olparam = BF_PXP_OLnPARAM_ENABLE (1);
olparam |= BF_PXP_OLnPARAM_ALPHA_CNTL(BV_PXP_OLnPARAM_ALPHA_CNTL__ROPs);
olparam |= BF_PXP_OLnPARAM_FORMAT (BV_PXP_OLnPARAM_FORMAT__ARGB8888);
olparam |= BF_PXP_OLnPARAM_ROP (BV_PXP_OLnPARAM_ROP__XOROL);
HW_PXP_OLnPARAM_WR(0,olparam); // enable overlay to perform XOR ROP using RGB8888 overlay
```

17.4.28 PXP Overlay 0 Parameters 2 Description

This register contains buffer parameters for the Overlay 0 input buffer.

HW_PXP_OL0PARAM2 0x230

Table 17-64. HW_PXP_OL0PARAM2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																															

Table 17-65. HW_PXP_OL0PARAM2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**RSVD**	RO	0x00000000	Reserved, always set to zero.

DESCRIPTION:

The Overlay 0 Parameter 2 register is reserved for future use.

EXAMPLE:

```
Empty Example.
```

17.4.29 PXP Overlay 1 Buffer Pointer Description

Overlay 1 Buffer Address Pointer. This register points to the beginning of the RGB Overlay 1 input buffer.

HW_PXP_OL1 0x240

Table 17-66. HW_PXP_OL1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 17-67. HW_PXP_OL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x0	Address pointer for the overlay 1 buffer. The address MUST be word-aligned for proper PXP operation.

DESCRIPTION:

This register points to the is used by the logic to point to the current output location for the RGB frame buffer.

EXAMPLE:

```
u32* overlay_ptr;
HW_PXP_OLn_WR(1,overlay_ptr);
```

17.4.30 PXP Overlay 1 Size Description

This register contains buffer size/location information for the Overlay 1 input buffer.

HW_PXP_OL1SIZE 0x250

Table 17-68. HW_PXP_OL1SIZE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XBASE								YBASE								WIDTH								HEIGHT							

Table 17-69. HW_PXP_OL1SIZE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**XBASE**	RW	0x00	This field indicates the X-coordinate (in blocks) of the top-left 8x8 block in the overlay within the output frame buffer.
23:16	**YBASE**	RW	0x00	This field indicates the Y-coordinate (in blocks) of the top-left 8x8 block in the overlay within the output frame buffer.
15:8	**WIDTH**	RW	0x0	Indicates number of horizontal 8x8 blocks in the image (non-rotated).
7:0	**HEIGHT**	RW	0x0	Indicates the number of vertical 8x8 blocks in the image (non-rotated).

DESCRIPTION:

This register contains information about Overlay 1 indicating the size of the overlay (in 8x8 blocks) and the overlay's location within the output frame buffer (in 8x8 blocks).

EXAMPLE:

```
HW_PXP_OLnSIZE_WR(1,0x10000401); // 32x8 overlay at offset +128+0
```

17.4.31 PXP Overlay 1 Parameters Description

This register contains buffer parameters for the Overlay 1 input buffer.

HW_PXP_OL1PARAM 0x260

Table 17-70. HW_PXP_OL1PARAM

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1												ROP				ALPHA								FORMAT				ENABLE_COLORKEY	ALPHA_CNTL		ENABLE

Table 17-71. HW_PXP_OL1PARAM Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:20	**RSVD1**	RO	0x0000	Reserved, always set to zero.
19:16	**ROP**	RW	0x0	Indicates a raster operation to perform when enabled. Raster operations are enabled through the ALPHA_CNTL field. MASKOL = 0x0 OL AND S0 MASKNOTOL = 0x1 nOL AND S0 MASKOLNOT = 0x2 OL AND nS0 MERGEOL = 0x3 OL OR S0 MERGENOTOL = 0x4 nOL OR S0 MERGEOLNOT = 0x5 OL OR nS0 NOTCOPYOL = 0x6 nOL NOT = 0x7 nS0 NOTMASKOL = 0x8 OL NAND S0 NOTMERGEOL = 0x9 OL NOR S0 XOROL = 0xA OL XOR S0 NOTXOROL = 0xB OL XNOR S0
15:8	**ALPHA**	RW	0x0	Alpha modifier used when the ALPHA_MULTIPLY or ALPHA_OVERRIDE bits are set. The output alpha value will either be replaced (ALPHA_OVERRIDE) or scaled (ALPHA_MULTIPLY) when enabled in the ALPHA_CNTL field.
7:4	**FORMAT**	RW	0x0	Indicates the input buffer format for overlay 0. ARGB8888 = 0x0 32-bit pixels with alpha RGB888 = 0x1 32-bit pixels without alpha (unpacked 24-bit format) ARGB1555 = 0x3 16-bit pixels with alpha RGB565 = 0x4 16-bit pixels without alpha RGB555 = 0x5 16-bit pixels without alpha
3	**ENABLE_COLORKEY**	RW	0x0	Indicates that colorkey functionality is enabled for this overlay. Pixels found in the overlay colorkey range will be displayed as transparent (the S0 pixel will be used).
2:1	**ALPHA_CNTL**	RW	0x0	Determines how the alpha value is constructed for this overlay. Indicates that the value in the ALPHA field should be used instead of the alpha values present in the input pixels. Embedded = 0x0 Indicates that the OL pixel alpha value will be used to blend the OL with S0. The ALPHA field is ignored. Override = 0x1 Indicates that the value in the ALPHA field should be used instead of the alpha values present in the input pixels. Multiply = 0x2 Indicates that the value in the ALPHA field should be used to scale all pixel alpha values. Each pixel alpha is multiplied by the value in the ALPHA field. ROPs = 0x3 Enable ROPs. The ROP field indicates an operation to be performed on the overlay and S0 pixels.
0	**ENABLE**	RW	0x0	Indicates that the overlay is active for this operation.

DESCRIPTION:

The S1 Overlay 1 Parameter register provides additional controls for Overlay 1.

EXAMPLE:

```
u32 olparam;
olparam = BF_PXP_OLnPARAM_ENABLE (1);
olparam |= BF_PXP_OLnPARAM_ALPHA_CNTL(BV_PXP_OLnPARAM_ALPHA_CNTL__ROPs);
olparam |= BF_PXP_OLnPARAM_FORMAT (BV_PXP_OLnPARAM_FORMAT__ARGB8888);
olparam |= BF_PXP_OLnPARAM_ROP (BV_PXP_OLnPARAM_ROP__XOROL);
HW_PXP_OLnPARAM_WR(1,olparam); // enable overlay to perform XOR ROP using RGB8888 overlay
```

17.4.32 PXP Overlay 1 Parameters 2 Description

This register contains buffer parameters for the Overlay 1 input buffer.

HW_PXP_OL1PARAM2 0x270

Table 17-72. HW_PXP_OL1PARAM2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																															

Table 17-73. HW_PXP_OL1PARAM2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**RSVD**	RO	0x00000000	Reserved, always set to zero.

DESCRIPTION:

The Overlay 1 Parameter 2 register is reserved for future use.

EXAMPLE:

```
Empty Example.
```

17.4.33 PXP Overlay 2 Buffer Pointer Description

Overlay 2 Buffer Address Pointer. This register points to the beginning of the RGB Overlay 2 input buffer.

HW_PXP_OL2 0x280

Table 17-74. HW_PXP_OL2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 17-75. HW_PXP_OL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x0	Address pointer for the overlay 2 buffer. The address MUST be word-aligned for proper PXP operation.

DESCRIPTION:

This register points to the is used by the logic to point to the current output location for the RGB frame buffer.

EXAMPLE:

```
u32* overlay_ptr;
HW_PXP_OLn_WR(2,overlay_ptr);
```

17.4.34 PXP Overlay 2 Size Description

This register contains buffer size/location information for the Overlay 2 input buffer.

HW_PXP_OL2SIZE 0x290

Table 17-76. HW_PXP_OL2SIZE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XBASE								YBASE								WIDTH								HEIGHT							

Table 17-77. HW_PXP_OL2SIZE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**XBASE**	RW	0x00	This field indicates the X-coordinate (in blocks) of the top-left 8x8 block in the overlay within the output frame buffer.
23:16	**YBASE**	RW	0x00	This field indicates the Y-coordinate (in blocks) of the top-left 8x8 block in the overlay within the output frame buffer.
15:8	**WIDTH**	RW	0x0	Indicates number of horizontal 8x8 blocks in the image (non-rotated).
7:0	**HEIGHT**	RW	0x0	Indicates the number of vertical 8x8 blocks in the image (non-rotated).

DESCRIPTION:

This register contains information about Overlay 2 indicating the size of the overlay (in 8x8 blocks) and the overlay's location within the output frame buffer (in 8x8 blocks).

EXAMPLE:

```
HW_PXP_OLnSIZE_WR(2,0x10000401); // 32x8 overlay at offset +128+0
```

17.4.35 PXP Overlay 2 Parameters Description

This register contains buffer parameters for the Overlay 2 input buffer.

HW_PXP_OL2PARAM 0x2a0

Table 17-78. HW_PXP_OL2PARAM

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1												ROP				ALPHA								FORMAT				ENABLE_COLORKEY	ALPHA_CNTL		ENABLE

Table 17-79. HW_PXP_OL2PARAM Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:20	**RSVD1**	RO	0x0000	Reserved, always set to zero.
19:16	**ROP**	RW	0x0	Indicates a raster operation to perform when enabled. Raster operations are enabled through the ALPHA_CNTL field. MASKOL = 0x0 OL AND S0 MASKNOTOL = 0x1 nOL AND S0 MASKOLNOT = 0x2 OL AND nS0 MERGEOL = 0x3 OL OR S0 MERGENOTOL = 0x4 nOL OR S0 MERGEOLNOT = 0x5 OL OR nS0 NOTCOPYOL = 0x6 nOL NOT = 0x7 nS0 NOTMASKOL = 0x8 OL NAND S0 NOTMERGEOL = 0x9 OL NOR S0 XOROL = 0xA OL XOR S0 NOTXOROL = 0xB OL XNOR S0
15:8	**ALPHA**	RW	0x0	Alpha modifier used when the ALPHA_MULTIPLY or ALPHA_OVERRIDE bits are set. The output alpha value will either be replaced (ALPHA_OVERRIDE) or scaled (ALPHA_MULTIPLY) when enabled in the ALPHA_CNTL field.
7:4	**FORMAT**	RW	0x0	Indicates the input buffer format for overlay 0. ARGB8888 = 0x0 32-bit pixels with alpha RGB888 = 0x1 32-bit pixels without alpha (unpacked 24-bit format) ARGB1555 = 0x3 16-bit pixels with alpha RGB565 = 0x4 16-bit pixels without alpha RGB555 = 0x5 16-bit pixels without alpha
3	**ENABLE_COLORKEY**	RW	0x0	Indicates that colorkey functionality is enabled for this overlay. Pixels found in the overlay colorkey range will be displayed as transparent (the S0 pixel will be used).
2:1	**ALPHA_CNTL**	RW	0x0	Determines how the alpha value is constructed for this overlay. Indicates that the value in the ALPHA field should be used instead of the alpha values present in the input pixels. Embedded = 0x0 Indicates that the OL pixel alpha value will be used to blend the OL with S0. The ALPHA field is ignored. Override = 0x1 Indicates that the value in the ALPHA field should be used instead of the alpha values present in the input pixels. Multiply = 0x2 Indicates that the value in the ALPHA field should be used to scale all pixel alpha values. Each pixel alpha is multiplied by the value in the ALPHA field. ROPs = 0x3 Enable ROPs. The ROP field indicates an operation to be performed on the overlay and S0 pixels.
0	**ENABLE**	RW	0x0	Indicates that the overlay is active for this operation.

DESCRIPTION:

The S1 Overlay 2 Parameter register provides additional controls for Overlay 2.

EXAMPLE:

```
u32 olparam;
olparam = BF_PXP_OLnPARAM_ENABLE (1);
olparam |= BF_PXP_OLnPARAM_ALPHA_CNTL(BV_PXP_OLnPARAM_ALPHA_CNTL__ROPs);
olparam |= BF_PXP_OLnPARAM_FORMAT (BV_PXP_OLnPARAM_FORMAT__ARGB8888);
olparam |= BF_PXP_OLnPARAM_ROP (BV_PXP_OLnPARAM_ROP__XOROL);
HW_PXP_OLnPARAM_WR(2,olparam); // enable overlay to perform XOR ROP using RGB8888 overlay
```

17.4.36 PXP Overlay 2 Parameters 2 Description

This register contains buffer parameters for the Overlay 2 input buffer.

HW_PXP_OL2PARAM2 0x2b0

Table 17-80. HW_PXP_OL2PARAM2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																															

Table 17-81. HW_PXP_OL2PARAM2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**RSVD**	RO	0x00000000	Reserved, always set to zero.

DESCRIPTION:

The Overlay 2 Parameter 2 register is reserved for future use.

EXAMPLE:

```
Empty Example.
```

17.4.37 PXP Overlay 3 Buffer Pointer Description

Overlay 3 Buffer Address Pointer. This register points to the beginning of the RGB Overlay 3 input buffer.

HW_PXP_OL3 0x2c0

Table 17-82. HW_PXP_OL3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 17-83. HW_PXP_OL3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x0	Address pointer for the overlay 3 buffer. The address MUST be word-aligned for proper PXP operation.

DESCRIPTION:

This register points to the is used by the logic to point to the current output location for the RGB frame buffer.

EXAMPLE:

```
u32* overlay_ptr;
HW_PXP_OLn_WR(3,overlay_ptr);
```

17.4.38 PXP Overlay 3 Size Description

This register contains buffer size/location information for the Overlay 3 input buffer.

HW_PXP_OL3SIZE 0x2d0

Table 17-84. HW_PXP_OL3SIZE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XBASE								YBASE								WIDTH								HEIGHT							

Table 17-85. HW_PXP_OL3SIZE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**XBASE**	RW	0x00	This field indicates the X-coordinate (in blocks) of the top-left 8x8 block in the overlay within the output frame buffer.
23:16	**YBASE**	RW	0x00	This field indicates the Y-coordinate (in blocks) of the top-left 8x8 block in the overlay within the output frame buffer.
15:8	**WIDTH**	RW	0x0	Indicates number of horizontal 8x8 blocks in the image (non-rotated).
7:0	**HEIGHT**	RW	0x0	Indicates the number of vertical 8x8 blocks in the image (non-rotated).

DESCRIPTION:

This register contains information about Overlay 3 indicating the size of the overlay (in 8x8 blocks) and the overlay's location within the output frame buffer (in 8x8 blocks).

EXAMPLE:

```
HW_PXP_OLnSIZE_WR(3,0x10000401); // 32x8 overlay at offset +128+0
```

17.4.39 PXP Overlay 3 Parameters Description

This register contains buffer parameters for the Overlay 3 input buffer.

HW_PXP_OL3PARAM 0x2e0

Table 17-86. HW_PXP_OL3PARAM

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1												ROP				ALPHA								FORMAT				ENABLE_COLORKEY	ALPHA_CNTL		ENABLE

Table 17-87. HW_PXP_OL3PARAM Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:20	**RSVD1**	RO	0x0000	Reserved, always set to zero.
19:16	**ROP**	RW	0x0	Indicates a raster operation to perform when enabled. Raster operations are enabled through the ALPHA_CNTL field. MASKOL = 0x0 OL AND S0 MASKNOTOL = 0x1 nOL AND S0 MASKOLNOT = 0x2 OL AND nS0 MERGEOL = 0x3 OL OR S0 MERGENOTOL = 0x4 nOL OR S0 MERGEOLNOT = 0x5 OL OR nS0 NOTCOPYOL = 0x6 nOL NOT = 0x7 nS0 NOTMASKOL = 0x8 OL NAND S0 NOTMERGEOL = 0x9 OL NOR S0 XOROL = 0xA OL XOR S0 NOTXOROL = 0xB OL XNOR S0
15:8	**ALPHA**	RW	0x0	Alpha modifier used when the ALPHA_MULTIPLY or ALPHA_OVERRIDE bits are set. The output alpha value will either be replaced (ALPHA_OVERRIDE) or scaled (ALPHA_MULTIPLY) when enabled in the ALPHA_CNTL field.
7:4	**FORMAT**	RW	0x0	Indicates the input buffer format for overlay 0. ARGB8888 = 0x0 32-bit pixels with alpha RGB888 = 0x1 32-bit pixels without alpha (unpacked 24-bit format) ARGB1555 = 0x3 16-bit pixels with alpha RGB565 = 0x4 16-bit pixels without alpha RGB555 = 0x5 16-bit pixels without alpha
3	**ENABLE_COLORKEY**	RW	0x0	Indicates that colorkey functionality is enabled for this overlay. Pixels found in the overlay colorkey range will be displayed as transparent (the S0 pixel will be used).
2:1	**ALPHA_CNTL**	RW	0x0	Determines how the alpha value is constructed for this overlay. Indicates that the value in the ALPHA field should be used instead of the alpha values present in the input pixels. Embedded = 0x0 Indicates that the OL pixel alpha value will be used to blend the OL with S0. The ALPHA field is ignored. Override = 0x1 Indicates that the value in the ALPHA field should be used instead of the alpha values present in the input pixels. Multiply = 0x2 Indicates that the value in the ALPHA field should be used to scale all pixel alpha values. Each pixel alpha is multiplied by the value in the ALPHA field. ROPs = 0x3 Enable ROPs. The ROP field indicates an operation to be performed on the overlay and S0 pixels.
0	**ENABLE**	RW	0x0	Indicates that the overlay is active for this operation.

DESCRIPTION:

The S1 Overlay 3 Parameter register provides additional controls for Overlay 3.

EXAMPLE:

```
u32 olparam;
olparam = BF_PXP_OLnPARAM_ENABLE (1);
olparam |= BF_PXP_OLnPARAM_ALPHA_CNTL(BV_PXP_OLnPARAM_ALPHA_CNTL__ROPs);
olparam |= BF_PXP_OLnPARAM_FORMAT (BV_PXP_OLnPARAM_FORMAT__ARGB8888);
olparam |= BF_PXP_OLnPARAM_ROP (BV_PXP_OLnPARAM_ROP__XOROL);
HW_PXP_OLnPARAM_WR(3,olparam); // enable overlay to perform XOR ROP using RGB8888 overlay
```

17.4.40 PXP Overlay 3 Parameters 2 Description

This register contains buffer parameters for the Overlay 3 input buffer.

HW_PXP_OL3PARAM2 0x2f0

Table 17-88. HW_PXP_OL3PARAM2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																															

Table 17-89. HW_PXP_OL3PARAM2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**RSVD**	RO	0x00000000	Reserved, always set to zero.

DESCRIPTION:

The Overlay 3 Parameter 2 register is reserved for future use.

EXAMPLE:

```
Empty Example.
```

17.4.41 PXP Overlay 4 Buffer Pointer Description

Overlay 4 Buffer Address Pointer. This register points to the beginning of the RGB Overlay 4 input buffer.

HW_PXP_OL4 0x300

Table 17-90. HW_PXP_OL4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 17-91. HW_PXP_OL4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x0	Address pointer for the overlay 4 buffer. The address MUST be word-aligned for proper PXP operation.

DESCRIPTION:

This register points to the is used by the logic to point to the current output location for the RGB frame buffer.

EXAMPLE:

```
u32* overlay_ptr;
HW_PXP_OLn_WR(4,overlay_ptr);
```

17.4.42 PXP Overlay 4 Size Description

This register contains buffer size/location information for the Overlay 4 input buffer.

HW_PXP_OL4SIZE 0x310

Table 17-92. HW_PXP_OL4SIZE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XBASE								YBASE								WIDTH								HEIGHT							

Table 17-93. HW_PXP_OL4SIZE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**XBASE**	RW	0x00	This field indicates the X-coordinate (in blocks) of the top-left 8x8 block in the overlay within the output frame buffer.
23:16	**YBASE**	RW	0x00	This field indicates the Y-coordinate (in blocks) of the top-left 8x8 block in the overlay within the output frame buffer.
15:8	**WIDTH**	RW	0x0	Indicates number of horizontal 8x8 blocks in the image (non-rotated).
7:0	**HEIGHT**	RW	0x0	Indicates the number of vertical 8x8 blocks in the image (non-rotated).

DESCRIPTION:

This register contains information about Overlay 4 indicating the size of the overlay (in 8x8 blocks) and the overlay's location within the output frame buffer (in 8x8 blocks).

EXAMPLE:

```
HW_PXP_OLnSIZE_WR(4,0x10000401); // 32x8 overlay at offset +128+0
```

17.4.43 PXP Overlay 4 Parameters Description

This register contains buffer parameters for the Overlay 4 input buffer.

HW_PXP_OL4PARAM 0x320

Table 17-94. HW_PXP_OL4PARAM

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1												ROP				ALPHA								FORMAT				ENABLE_COLORKEY	ALPHA_CNTL		ENABLE

Table 17-95. HW_PXP_OL4PARAM Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:20	**RSVD1**	RO	0x0000	Reserved, always set to zero.
19:16	**ROP**	RW	0x0	Indicates a raster operation to perform when enabled. Raster operations are enabled through the ALPHA_CNTL field. MASKOL = 0x0 OL AND S0 MASKNOTOL = 0x1 nOL AND S0 MASKOLNOT = 0x2 OL AND nS0 MERGEOL = 0x3 OL OR S0 MERGENOTOL = 0x4 nOL OR S0 MERGEOLNOT = 0x5 OL OR nS0 NOTCOPYOL = 0x6 nOL NOT = 0x7 nS0 NOTMASKOL = 0x8 OL NAND S0 NOTMERGEOL = 0x9 OL NOR S0 XOROL = 0xA OL XOR S0 NOTXOROL = 0xB OL XNOR S0
15:8	**ALPHA**	RW	0x0	Alpha modifier used when the ALPHA_MULTIPLY or ALPHA_OVERRIDE bits are set. The output alpha value will either be replaced (ALPHA_OVERRIDE) or scaled (ALPHA_MULTIPLY) when enabled in the ALPHA_CNTL field.
7:4	**FORMAT**	RW	0x0	Indicates the input buffer format for overlay 0. ARGB8888 = 0x0 32-bit pixels with alpha RGB888 = 0x1 32-bit pixels without alpha (unpacked 24-bit format) ARGB1555 = 0x3 16-bit pixels with alpha RGB565 = 0x4 16-bit pixels without alpha RGB555 = 0x5 16-bit pixels without alpha
3	**ENABLE_COLORKEY**	RW	0x0	Indicates that colorkey functionality is enabled for this overlay. Pixels found in the overlay colorkey range will be displayed as transparent (the S0 pixel will be used).
2:1	**ALPHA_CNTL**	RW	0x0	Determines how the alpha value is constructed for this overlay. Indicates that the value in the ALPHA field should be used instead of the alpha values present in the input pixels. Embedded = 0x0 Indicates that the OL pixel alpha value will be used to blend the OL with S0. The ALPHA field is ignored. Override = 0x1 Indicates that the value in the ALPHA field should be used instead of the alpha values present in the input pixels. Multiply = 0x2 Indicates that the value in the ALPHA field should be used to scale all pixel alpha values. Each pixel alpha is multiplied by the value in the ALPHA field. ROPs = 0x3 Enable ROPs. The ROP field indicates an operation to be performed on the overlay and S0 pixels.
0	**ENABLE**	RW	0x0	Indicates that the overlay is active for this operation.

DESCRIPTION:

The S1 Overlay 4 Parameter register provides additional controls for Overlay 4.

EXAMPLE:

```
u32 olparam;
olparam = BF_PXP_OLnPARAM_ENABLE (1);
olparam |= BF_PXP_OLnPARAM_ALPHA_CNTL(BV_PXP_OLnPARAM_ALPHA_CNTL__ROPs);
olparam |= BF_PXP_OLnPARAM_FORMAT (BV_PXP_OLnPARAM_FORMAT__ARGB8888);
olparam |= BF_PXP_OLnPARAM_ROP (BV_PXP_OLnPARAM_ROP__XOROL);
HW_PXP_OLnPARAM_WR(4,olparam); // enable overlay to perform XOR ROP using RGB8888 overlay
```

17.4.44 PXP Overlay 4 Parameters 2 Description

This register contains buffer parameters for the Overlay 4 input buffer.

HW_PXP_OL4PARAM2 0x330

Table 17-96. HW_PXP_OL4PARAM2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																															

Table 17-97. HW_PXP_OL4PARAM2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**RSVD**	RO	0x00000000	Reserved, always set to zero.

DESCRIPTION:

The Overlay 4 Parameter 2 register is reserved for future use.

EXAMPLE:

```
Empty Example.
```

17.4.45 PXP Overlay 5 Buffer Pointer Description

Overlay 5 Buffer Address Pointer. This register points to the beginning of the RGB Overlay 5 input buffer.

HW_PXP_OL5 0x340

Table 17-98. HW_PXP_OL5

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 17-99. HW_PXP_OL5 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x0	Address pointer for the overlay 5 buffer. The address MUST be word-aligned for proper PXP operation.

DESCRIPTION:

This register points to the is used by the logic to point to the current output location for the RGB frame buffer.

EXAMPLE:

```
u32* overlay_ptr;
HW_PXP_OLn_WR(5,overlay_ptr);
```

17.4.46 PXP Overlay 5 Size Description

This register contains buffer size/location information for the Overlay 5 input buffer.

HW_PXP_OL5SIZE 0x350

Table 17-100. HW_PXP_OL5SIZE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XBASE								YBASE								WIDTH								HEIGHT							

Table 17-101. HW_PXP_OL5SIZE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**XBASE**	RW	0x00	This field indicates the X-coordinate (in blocks) of the top-left 8x8 block in the overlay within the output frame buffer.
23:16	**YBASE**	RW	0x00	This field indicates the Y-coordinate (in blocks) of the top-left 8x8 block in the overlay within the output frame buffer.
15:8	**WIDTH**	RW	0x0	Indicates number of horizontal 8x8 blocks in the image (non-rotated).
7:0	**HEIGHT**	RW	0x0	Indicates the number of vertical 8x8 blocks in the image (non-rotated).

DESCRIPTION:

This register contains information about Overlay 5 indicating the size of the overlay (in 8x8 blocks) and the overlay's location within the output frame buffer (in 8x8 blocks).

EXAMPLE:

```
HW_PXP_OLnSIZE_WR(5,0x10000401); // 32x8 overlay at offset +128+0
```

17.4.47 PXP Overlay 5 Parameters Description

This register contains buffer parameters for the Overlay 5 input buffer.

HW_PXP_OL5PARAM 0x360

Table 17-102. HW_PXP_OL5PARAM

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1												ROP				ALPHA								FORMAT				ENABLE_COLORKEY	ALPHA_CNTL		ENABLE

Table 17-103. HW_PXP_OL5PARAM Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:20	**RSVD1**	RO	0x0000	Reserved, always set to zero.
19:16	**ROP**	RW	0x0	Indicates a raster operation to perform when enabled. Raster operations are enabled through the ALPHA_CNTL field. MASKOL = 0x0 OL AND S0 MASKNOTOL = 0x1 nOL AND S0 MASKOLNOT = 0x2 OL AND nS0 MERGEOL = 0x3 OL OR S0 MERGENOTOL = 0x4 nOL OR S0 MERGEOLNOT = 0x5 OL OR nS0 NOTCOPYOL = 0x6 nOL NOT = 0x7 nS0 NOTMASKOL = 0x8 OL NAND S0 NOTMERGEOL = 0x9 OL NOR S0 XOROL = 0xA OL XOR S0 NOTXOROL = 0xB OL XNOR S0
15:8	**ALPHA**	RW	0x0	Alpha modifier used when the ALPHA_MULTIPLY or ALPHA_OVERRIDE bits are set. The output alpha value will either be replaced (ALPHA_OVERRIDE) or scaled (ALPHA_MULTIPLY) when enabled in the ALPHA_CNTL field.
7:4	**FORMAT**	RW	0x0	Indicates the input buffer format for overlay 0. ARGB8888 = 0x0 32-bit pixels with alpha RGB888 = 0x1 32-bit pixels without alpha (unpacked 24-bit format) ARGB1555 = 0x3 16-bit pixels with alpha RGB565 = 0x4 16-bit pixels without alpha RGB555 = 0x5 16-bit pixels without alpha
3	**ENABLE_COLORKEY**	RW	0x0	Indicates that colorkey functionality is enabled for this overlay. Pixels found in the overlay colorkey range will be displayed as transparent (the S0 pixel will be used).
2:1	**ALPHA_CNTL**	RW	0x0	Determines how the alpha value is constructed for this overlay. Indicates that the value in the ALPHA field should be used instead of the alpha values present in the input pixels. Embedded = 0x0 Indicates that the OL pixel alpha value will be used to blend the OL with S0. The ALPHA field is ignored. Override = 0x1 Indicates that the value in the ALPHA field should be used instead of the alpha values present in the input pixels. Multiply = 0x2 Indicates that the value in the ALPHA field should be used to scale all pixel alpha values. Each pixel alpha is multiplied by the value in the ALPHA field. ROPs = 0x3 Enable ROPs. The ROP field indicates an operation to be performed on the overlay and S0 pixels.
0	**ENABLE**	RW	0x0	Indicates that the overlay is active for this operation.

DESCRIPTION:

The S1 Overlay 5 Parameter register provides additional controls for Overlay 5.

EXAMPLE:

```
u32 olparam;
olparam = BF_PXP_OLnPARAM_ENABLE (1);
olparam |= BF_PXP_OLnPARAM_ALPHA_CNTL(BV_PXP_OLnPARAM_ALPHA_CNTL__ROPs);
olparam |= BF_PXP_OLnPARAM_FORMAT (BV_PXP_OLnPARAM_FORMAT__ARGB8888);
olparam |= BF_PXP_OLnPARAM_ROP (BV_PXP_OLnPARAM_ROP__XOROL);
HW_PXP_OLnPARAM_WR(5,olparam); // enable overlay to perform XOR ROP using RGB8888 overlay
```

17.4.48 PXP Overlay 5 Parameters 2 Description

This register contains buffer parameters for the Overlay 5 input buffer.

HW_PXP_OL5PARAM2 0x370

Table 17-104. HW_PXP_OL5PARAM2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																															

Table 17-105. HW_PXP_OL5PARAM2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**RSVD**	RO	0x00000000	Reserved, always set to zero.

DESCRIPTION:

The Overlay 5 Parameter 2 register is reserved for future use.

EXAMPLE:

```
Empty Example.
```

17.4.49 PXP Overlay 6 Buffer Pointer Description

Overlay 6 Buffer Address Pointer. This register points to the beginning of the RGB Overlay 6 input buffer.

HW_PXP_OL6 0x380

Table 17-106. HW_PXP_OL6

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 17-107. HW_PXP_OL6 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x0	Address pointer for the overlay 6 buffer. The address MUST be word-aligned for proper PXP operation.

DESCRIPTION:

This register points to the is used by the logic to point to the current output location for the RGB frame buffer.

EXAMPLE:

```
u32* overlay_ptr;
HW_PXP_OLn_WR(6,overlay_ptr);
```

17.4.50 PXP Overlay 6 Size Description

This register contains buffer size/location information for the Overlay 6 input buffer.

HW_PXP_OL6SIZE 0x390

Table 17-108. HW_PXP_OL6SIZE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XBASE								YBASE								WIDTH								HEIGHT							

Table 17-109. HW_PXP_OL6SIZE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**XBASE**	RW	0x00	This field indicates the X-coordinate (in blocks) of the top-left 8x8 block in the overlay within the output frame buffer.
23:16	**YBASE**	RW	0x00	This field indicates the Y-coordinate (in blocks) of the top-left 8x8 block in the overlay within the output frame buffer.
15:8	**WIDTH**	RW	0x0	Indicates number of horizontal 8x8 blocks in the image (non-rotated).
7:0	**HEIGHT**	RW	0x0	Indicates the number of vertical 8x8 blocks in the image (non-rotated).

DESCRIPTION:

This register contains information about Overlay 6 indicating the size of the overlay (in 8x8 blocks) and the overlay's location within the output frame buffer (in 8x8 blocks).

EXAMPLE:

```
HW_PXP_OLnSIZE_WR(6,0x10000401); // 32x8 overlay at offset +128+0
```

17.4.51 PXP Overlay 6 Parameters Description

This register contains buffer parameters for the Overlay 6 input buffer.

HW_PXP_OL6PARAM 0x3a0

Table 17-110. HW_PXP_OL6PARAM

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1												ROP				ALPHA								FORMAT				ENABLE_COLORKEY	ALPHA_CNTL		ENABLE

Table 17-111. HW_PXP_OL6PARAM Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:20	**RSVD1**	RO	0x0000	Reserved, always set to zero.
19:16	**ROP**	RW	0x0	Indicates a raster operation to perform when enabled. Raster operations are enabled through the ALPHA_CNTL field. MASKOL = 0x0 OL AND S0 MASKNOTOL = 0x1 nOL AND S0 MASKOLNOT = 0x2 OL AND nS0 MERGEOL = 0x3 OL OR S0 MERGENOTOL = 0x4 nOL OR S0 MERGEOLNOT = 0x5 OL OR nS0 NOTCOPYOL = 0x6 nOL NOT = 0x7 nS0 NOTMASKOL = 0x8 OL NAND S0 NOTMERGEOL = 0x9 OL NOR S0 XOROL = 0xA OL XOR S0 NOTXOROL = 0xB OL XNOR S0
15:8	**ALPHA**	RW	0x0	Alpha modifier used when the ALPHA_MULTIPLY or ALPHA_OVERRIDE bits are set. The output alpha value will either be replaced (ALPHA_OVERRIDE) or scaled (ALPHA_MULTIPLY) when enabled in the ALPHA_CNTL field.
7:4	**FORMAT**	RW	0x0	Indicates the input buffer format for overlay 0. ARGB8888 = 0x0 32-bit pixels with alpha RGB888 = 0x1 32-bit pixels without alpha (unpacked 24-bit format) ARGB1555 = 0x3 16-bit pixels with alpha RGB565 = 0x4 16-bit pixels without alpha RGB555 = 0x5 16-bit pixels without alpha
3	**ENABLE_COLORKEY**	RW	0x0	Indicates that colorkey functionality is enabled for this overlay. Pixels found in the overlay colorkey range will be displayed as transparent (the S0 pixel will be used).
2:1	**ALPHA_CNTL**	RW	0x0	Determines how the alpha value is constructed for this overlay. Indicates that the value in the ALPHA field should be used instead of the alpha values present in the input pixels. Embedded = 0x0 Indicates that the OL pixel alpha value will be used to blend the OL with S0. The ALPHA field is ignored. Override = 0x1 Indicates that the value in the ALPHA field should be used instead of the alpha values present in the input pixels. Multiply = 0x2 Indicates that the value in the ALPHA field should be used to scale all pixel alpha values. Each pixel alpha is multiplied by the value in the ALPHA field. ROPs = 0x3 Enable ROPs. The ROP field indicates an operation to be performed on the overlay and S0 pixels.
0	**ENABLE**	RW	0x0	Indicates that the overlay is active for this operation.

DESCRIPTION:

The S1 Overlay 6 Parameter register provides additional controls for Overlay 6.

EXAMPLE:

```
u32 olparam;
olparam = BF_PXP_OLnPARAM_ENABLE (1);
olparam |= BF_PXP_OLnPARAM_ALPHA_CNTL(BV_PXP_OLnPARAM_ALPHA_CNTL__ROPs);
olparam |= BF_PXP_OLnPARAM_FORMAT (BV_PXP_OLnPARAM_FORMAT__ARGB8888);
olparam |= BF_PXP_OLnPARAM_ROP (BV_PXP_OLnPARAM_ROP__XOROL);
HW_PXP_OLnPARAM_WR(6,olparam); // enable overlay to perform XOR ROP using RGB8888 overlay
```

17.4.52 PXP Overlay 6 Parameters 2 Description

This register contains buffer parameters for the Overlay 6 input buffer.

HW_PXP_OL6PARAM2 0x3b0

Table 17-112. HW_PXP_OL6PARAM2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																															

Table 17-113. HW_PXP_OL6PARAM2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**RSVD**	RO	0x00000000	Reserved, always set to zero.

DESCRIPTION:

The Overlay 6 Parameter 2 register is reserved for future use.

EXAMPLE:

```
Empty Example.
```

17.4.53 PXP Overlay 7 Buffer Pointer Description

Overlay 7 Buffer Address Pointer. This register points to the beginning of the RGB Overlay 7 input buffer.

HW_PXP_OL7 0x3c0

Table 17-114. HW_PXP_OL7

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 17-115. HW_PXP_OL7 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x0	Address pointer for the overlay 7 buffer. The address MUST be word-aligned for proper PXP operation.

DESCRIPTION:

This register points to the is used by the logic to point to the current output location for the RGB frame buffer.

EXAMPLE:

```
u32* overlay_ptr;
HW_PXP_OLn_WR(7,overlay_ptr);
```

17.4.54 PXP Overlay 7 Size Description

This register contains buffer size/location information for the Overlay 7 input buffer.

HW_PXP_OL7SIZE 0x3d0

Table 17-116. HW_PXP_OL7SIZE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
XBASE								YBASE								WIDTH								HEIGHT							

Table 17-117. HW_PXP_OL7SIZE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**XBASE**	RW	0x00	This field indicates the X-coordinate (in blocks) of the top-left 8x8 block in the overlay within the output frame buffer.
23:16	**YBASE**	RW	0x00	This field indicates the Y-coordinate (in blocks) of the top-left 8x8 block in the overlay within the output frame buffer.
15:8	**WIDTH**	RW	0x0	Indicates number of horizontal 8x8 blocks in the image (non-rotated).
7:0	**HEIGHT**	RW	0x0	Indicates the number of vertical 8x8 blocks in the image (non-rotated).

DESCRIPTION:

This register contains information about Overlay 7 indicating the size of the overlay (in 8x8 blocks) and the overlay's location within the output frame buffer (in 8x8 blocks).

EXAMPLE:

```
HW_PXP_OLnSIZE_WR(7,0x10000401); // 32x8 overlay at offset +128+0
```

17.4.55 PXP Overlay 7 Parameters Description

This register contains buffer parameters for the Overlay 7 input buffer.

HW_PXP_OL7PARAM 0x3e0

Table 17-118. HW_PXP_OL7PARAM

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1												ROP				ALPHA								FORMAT				ENABLE_COLORKEY	ALPHA_CNTL		ENABLE

Table 17-119. HW_PXP_OL7PARAM Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:20	**RSVD1**	RO	0x0000	Reserved, always set to zero.
19:16	**ROP**	RW	0x0	Indicates a raster operation to perform when enabled. Raster operations are enabled through the ALPHA_CNTL field. MASKOL = 0x0 OL AND S0 MASKNOTOL = 0x1 nOL AND S0 MASKOLNOT = 0x2 OL AND nS0 MERGEOL = 0x3 OL OR S0 MERGENOTOL = 0x4 nOL OR S0 MERGEOLNOT = 0x5 OL OR nS0 NOTCOPYOL = 0x6 nOL NOT = 0x7 nS0 NOTMASKOL = 0x8 OL NAND S0 NOTMERGEOL = 0x9 OL NOR S0 XOROL = 0xA OL XOR S0 NOTXOROL = 0xB OL XNOR S0
15:8	**ALPHA**	RW	0x0	Alpha modifier used when the ALPHA_MULTIPLY or ALPHA_OVERRIDE bits are set. The output alpha value will either be replaced (ALPHA_OVERRIDE) or scaled (ALPHA_MULTIPLY) when enabled in the ALPHA_CNTL field.
7:4	**FORMAT**	RW	0x0	Indicates the input buffer format for overlay 0. ARGB8888 = 0x0 32-bit pixels with alpha RGB888 = 0x1 32-bit pixels without alpha (unpacked 24-bit format) ARGB1555 = 0x3 16-bit pixels with alpha RGB565 = 0x4 16-bit pixels without alpha RGB555 = 0x5 16-bit pixels without alpha
3	**ENABLE_COLORKEY**	RW	0x0	Indicates that colorkey functionality is enabled for this overlay. Pixels found in the overlay colorkey range will be displayed as transparent (the S0 pixel will be used).
2:1	**ALPHA_CNTL**	RW	0x0	Determines how the alpha value is constructed for this overlay. Indicates that the value in the ALPHA field should be used instead of the alpha values present in the input pixels. Embedded = 0x0 Indicates that the OL pixel alpha value will be used to blend the OL with S0. The ALPHA field is ignored. Override = 0x1 Indicates that the value in the ALPHA field should be used instead of the alpha values present in the input pixels. Multiply = 0x2 Indicates that the value in the ALPHA field should be used to scale all pixel alpha values. Each pixel alpha is multiplied by the value in the ALPHA field. ROPs = 0x3 Enable ROPs. The ROP field indicates an operation to be performed on the overlay and S0 pixels.
0	**ENABLE**	RW	0x0	Indicates that the overlay is active for this operation.

DESCRIPTION:

The S1 Overlay 7 Parameter register provides additional controls for Overlay 7.

EXAMPLE:

```
u32 olparam;
olparam = BF_PXP_OLnPARAM_ENABLE (1);
olparam |= BF_PXP_OLnPARAM_ALPHA_CNTL(BV_PXP_OLnPARAM_ALPHA_CNTL__ROPs);
olparam |= BF_PXP_OLnPARAM_FORMAT (BV_PXP_OLnPARAM_FORMAT__ARGB8888);
olparam |= BF_PXP_OLnPARAM_ROP (BV_PXP_OLnPARAM_ROP__XOROL);
HW_PXP_OLnPARAM_WR(7,olparam); // enable overlay to perform XOR ROP using RGB8888 overlay
```

17.4.56 PXP Overlay 7 Parameters 2 Description

This register contains buffer parameters for the Overlay 7 input buffer.

HW_PXP_OL7PARAM2 0x3f0

Table 17-120. HW_PXP_OL7PARAM2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD																															

Table 17-121. HW_PXP_OL7PARAM2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**RSVD**	RO	0x00000000	Reserved, always set to zero.

DESCRIPTION:

The Overlay 7 Parameter 2 register is reserved for future use.

EXAMPLE:

```
Empty Example.
```

PXP Block v2.0, Revision 1.57

Chapter 18
LCD Interface (LCDIF)

This chapter describes the LCD interface included on the i.MX23 and includes operation examples. Programmable registers are described in Section 18.4, “Programmable Registers.”

18.1 Overview

Many products based on the i.MX23 include an LCD panel with an integrated controller/driver. These smart LCDs are available in a range of sizes and capabilities, from simple text-only displays to QVGA, 16/18/24 bpp color TFT panels. Traditionally, many of these display controllers have had an asynchronous parallel System (or MCU) interface for command and data transfer to the frame buffer. There are other popular displays that support moving pictures and require the RGB interface mode (called DOTCLK interface in this document) or the VSYNC mode for high-speed data transfers. In addition to these displays, it is also common to provide support for digital video encoders that accept ITU-R BT.656 format 4:2:2 YCbCr digital component video and convert it to analog TV signals. The LCDIF block on i.MX23 supports all of these different interfaces by providing fully programmable functionality and sharing register space, FIFOs, and ALU resources at the same time.

The high-level block diagram of the LCD interface provided on the i.MX23 is shown in Figure 18-1.

The block has several major features:

- Bus master and PIO operating modes for LCD writes requiring minimal CPU overhead.
- 8/16/18/24 bit LCD data bus support available depending on the package size.
- Programmable timing and parameters for system, VSYNC and DOTCLK LCD interfaces to support a wide variety of displays.
- ITU-R BT.656 mode (called Digital Video Interface or DVI mode here) including progressive-to-interlace feature and RGB to YCbCr 4:2:2 color space conversion to support 525/60 (NTSC) and 625/50 (PAL) operation.
- Hardware-based pin sharing with NAND data pins in system and VSYNC modes.

18.2 Operation

The general description provided in Section 18.2.1, “Bus Interface Mechanisms,” through Section 18.2.4, “Initializing the LCDIF,” is applicable to all the interface modes, because they all share the same pipeline until the TXFIFO. The differences for each mode are then described in separate sections, as follows:

- Section 18.2.5, “System Interface,” "System Interface"
- Section 18.2.6, “VSYNC Interface,” "VSYNC Interface"

- Section 18.2.7, “DOTCLK Interface,”"DOTCLK Interface"
- Section 18.2.8, “ITU-R BT.656 Digital Video Interface (DVI),” "ITU-R BT.656 Digital Video Interface (DVI)"

LCDIF pin usage by interface mode is described in Section 18.2.9, “LCDIF Pin Usage by Interface Mode.”

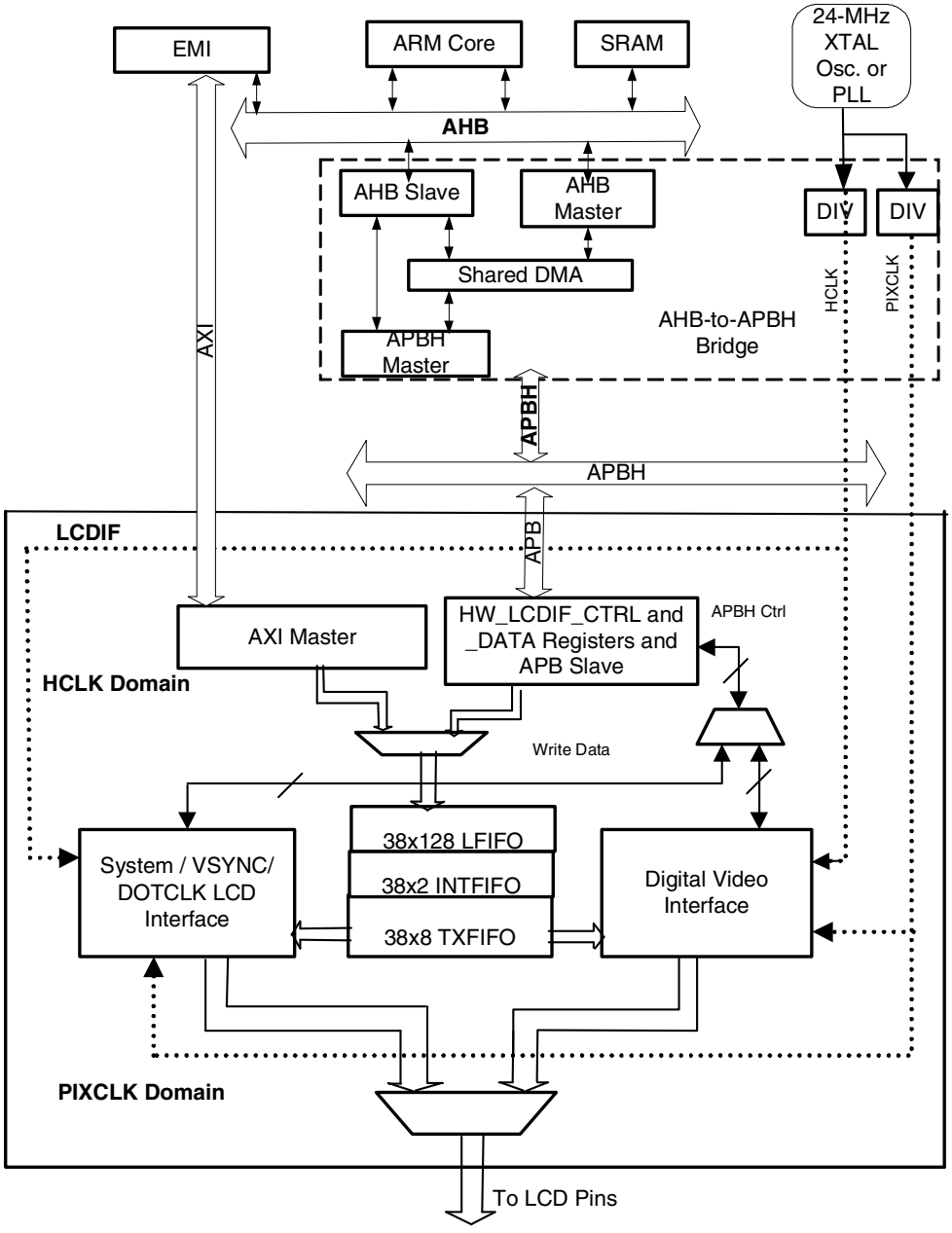


Figure 18-1. LCDIF Top Level Diagram

18.2.1 Bus Interface Mechanisms

The LCDIF block has memory-mapped control, data and status registers. It provides two efficient methods of transferring data, namely APB PIO and AXI bus master, to an external LCD controller or a digital video encoder. In either mode of operation, the bus interface portion of the block works off the HCLK domain, while the actual interface to external display/encoder works off the PIXCLK domain. There are two main FIFOs in the block datapath. The latency FIFO (LFIFO) is a 128 words deep synchronous FIFO that offers buffering against system bandwidth and latency. The other FIFO, called TXFIFO, is an 8 words deep asynchronous FIFO that assists data crossing between the two clock domains. In between these two FIFOs is a small 2-word deep INTFIFO that is used for unpacking the data and color space conversion. The following sub-sections describe the two system bus interface mechanisms.

18.2.1.1 PIO Operation

The CPU can directly send commands or data to the LCD panel by setting up the block in non-bus-master mode (HW_LCDIF_CTRL_LCDIF_MASTER = 0) and writing directly to the HW_LCDIF_DATA register. The FIFO status bits in the HW_LCDIF_STAT register indicate the full and empty states of both the LFIFO and the TXFIFO. When the LFIFO is not full, the data register can be safely written with a word, halfword, or byte as required; doing otherwise will result in incorrect operation. Software can also wait for the DMA_REQ bit in HW_LCDIF_STAT register to toggle before writing new data into the HW_LCDIF_DATA register. Since PIO mode is a very low speed mode of operation, it should be used only for system mode, not in DOTCLK and DVI modes.

18.2.1.2 Bus Master Operation

In this mode, the LCDIF block acts as a master on the AXI bus shared by other blocks like DCP, PXP and BCH. This is a high performance mode that can be used for sending large frames of data quickly and efficiently. In this mode, LCDIF issues bursts of 16-word fetches (or 15-word in packed 24-bit mode) from the memory. In Bus Master operation, the HW_LCDIF_CTRL_LCDIF_MASTER bit should be set to 1. The HW_LCDIF_CUR_BUF_ADDR and HW_LCDIF_NEXT_BUF_ADDR registers should be programmed to point to the base address of the frame buffers that needs to be transferred out.

In system and VSYNC modes, once the data in the frame buffer pointed to by the HW_LCDIF_CUR_BUF_ADDR is transferred out, the LCDIF stops transmitting and turns off the RUN bit in HW_LCDIF_CTRL. Hence, HW_LCDIF_CUR_BUF_ADDR has to be setup and kicked off again for transmitting the next frame; In this mode, the HW_LCDIF_NEXT_BUF_ADDR register is not used at all.

In the DOTCLK and DVI modes, before the RUN bit is set, software should start off with programming both the HW_LCDIF_CUR_BUF_ADDR and HW_LCDIF_NEXT_BUF_ADDR registers, and then, it should update only the HW_LCDIF_NEXT_BUF_ADDR register at the end of every frame. The LCDIF will automatically copy the value in the HW_LCDIF_NEXT_BUF register to the

HW_LCDIF_CUR_BUF register before issuing the cur_frame_done interrupt and it will start fetching the next frame from the new address. In other words, when the ISR for the cur_frame_done interrupt is entered, HW_LCDIF_CUR_BUF and HW_LCDIF_NEXT_BUF registers will have the same value until a new value is programmed in the HW_LCDIF_NEXT_BUF register. Thus, software has about one frame worth of time to update HW_LCDIF_NEXT_BUF_ADDR before it actually gets used. If for some reason, the HW_LCDIF_NEXT_BUF_ADDR register was not updated within a frame, the LCDIF will continue transmitting the last frame until a new value is programmed into that register.

The LCDIF also provides the capability of interlacing a progressive frame by fetching odd lines in the first field and then fetching even lines in the second field. This feature is likely to be used in the DVI mode for transmitting to an internal/external TV encoder and can be turned on by setting the INTERLACE_FIELDS bit in the HW_LCDIF_CTRL1 register.

18.2.2 Write Datapath

All frame buffers must be arranged in the raster format since external displays and digital video encoders require data in the raster format. The LCDIF receives little-endian data from the CPU or directly fetches it from memory. This raw input data can be swizzled according to the INPUT_DATA_SWIZZLE field in the HW_LCDIF_CTRL1 register before any other operation is performed on the incoming data. The following four combinations are supported:

- 00 (0): No swizzle (little-endian)
- 01 (1): Swap bytes 0 and 3, swap bytes 1 and 2 (big-endian)
- 10 (2): Swap half-words
- 11 (3): Swap bytes within each half-word

The WORD_LENGTH field of the HW_LCDIF_CTRL register indicates the input data/pixel format. HW_LCDIF_TRANSFER_COUNT register denotes how much data is contained in each frame. The H_COUNT field of this register indicates the number of active pixels per line and V_COUNT indicates the total active number of lines per frame. A special bit field in the CTRL1 register, called the BYTE_PACKING_FORMAT, can be used to specify which bytes within the 32-bit word are going to be valid. For example, if the entire 32-bit word is valid, BYTE_PACKING_FORMAT should be set to 0xF, if only lower 3 bytes of each word in the frame buffer are valid, then BYTE_PACKING_FORMAT should be set to 0x7.

The LCD_DATABUS_WIDTH field in HW_LCDIF_CTRL register indicates the width of the bus going to the external display controller. If the LCD_DATABUS_WIDTH is not the same as WORD_LENGTH, LCDIF will do minor RGB-to-RGB color space conversion (CSC). For example, if the input frame has more bits per pixel than the display, e.g. 16 bpp input frame going to 24 bpp LCD, the LCDIF will pad the MSBs of each color component to the LSBs of the same color component for each pixel. If the input frame has fewer bits per pixel than the display, e.g. a 24 bpp input frame going to a 16 bpp LCD, the LCDIF will drop the LSBs of each color component to go to the lower resolution. If software wants to

make sure that CSC does not take place, it must ensure that WORD_LENGTH and LCD_DATABUS_WIDTH have the same value.

The LCDIF also supports RGB to YCbCr 4:2:2 color space conversion. This is useful in the DVI mode since the TV encoder requires input in YCbCr 4:2:2 format. The HW_LCDIF_CSC registers have complete programmability over the CSC coefficients and offsets. The values must be written into these registers in the signed two's complement format.

The following list shows how the different input/output combinations can be obtained:

- WORD_LENGTH=1 indicates that the input is 8-bit data. This is most likely going to be used for sending commands in System interface, or maybe a grayscale image. Any combination of BYTE_PACKING_FORMAT [3:0] is permissible (except 0).

  Limitations:

  — H_COUNT must be a multiple of the sum of BYTE_PACKING_FORMAT [3], BYTE_PACKING_FORMAT [2], BYTE_PACKING_FORMAT [1] and BYTE_PACKING_FORMAT [0].

  — LCD_DATABUS_WIDTH must be 1, indicating an 8-bit data bus.
- WORD_LENGTH=0 implies the input frame buffer is RGB 16 bits per pixel. DATA_FORMAT_16_BIT field indicates if the pixels are in RGB 555 or RGB 565 format.

  Limitations:

  — BYTE_PACKING_FORMAT [3:0] should be 0x3 or 0xC if there is only one pixel per word.

  — If there are two pixels per word, BYTE_PACKING_FORMAT [3:0] should be 0xF and H_COUNT will be restricted to be a multiple of 2 pixels.
- WORD_LENGTH=2 indicates that input frame buffer is RGB 18 bits per pixel, i.e. RGB 666. The valid RGB values can be left-aligned or right-aligned within a 32-bit word. The alignment of the valid 18 bits within a word is indicated by the DATA_FORMAT_18_BIT bit. There is no limitation on H_COUNT.

  Limitations:

  — BYTE_PACKING_FORMAT must be 0xF.

  — Packed pixels are not supported in this case.
- WORD_LENGTH=3 indicates that the input frame-buffer is RGB 24 bits per pixel (RGB 888). If BYTE_PACKING_FORMAT [3:0] is 0x7, it indicates that there is only one pixel per 32-bit word and there is no restriction on H_COUNT.

  Limitations:

  — If BYTE_PACKING_FORMAT [3:0] is 0xF, it indicates that the pixels are packed, i.e. there are 4 pixels in 3 words or 12 bytes. In that case, H_COUNT must be a multiple of 4 pixels.
- YCBCR422_INPUT=1 implies that the input frame is in YCbCr 4:2:2 format. BYTE_PACKING_FORMAT must be 0xF.

  Limitations:

  — LCD_DATABUS_WIDTH must be 8-bit

  — H_COUNT must be a multiple of 2 pixels.

After the RGB-to-RGB or RGB-to-YCbCr 4:2:2 color space conversions, there is one more opportunity to swizzle the data before sending it out to the display or the encoder. This can be done with the CSC_DATA_SWIZZLE field in the HW_LCDIF_CTRL1 register, and it provides the same options as the INPUT_DATA_SWIZZLE register.

Finally, there is an option to shift the output data before sending it out to the display. This is done based on the SHIFT_DIR and SHIFT_NUM_BITS fields in HW_LCDIF_CTRL1 register.

Figure 18-2 shows the general operations that occur in the write data path.

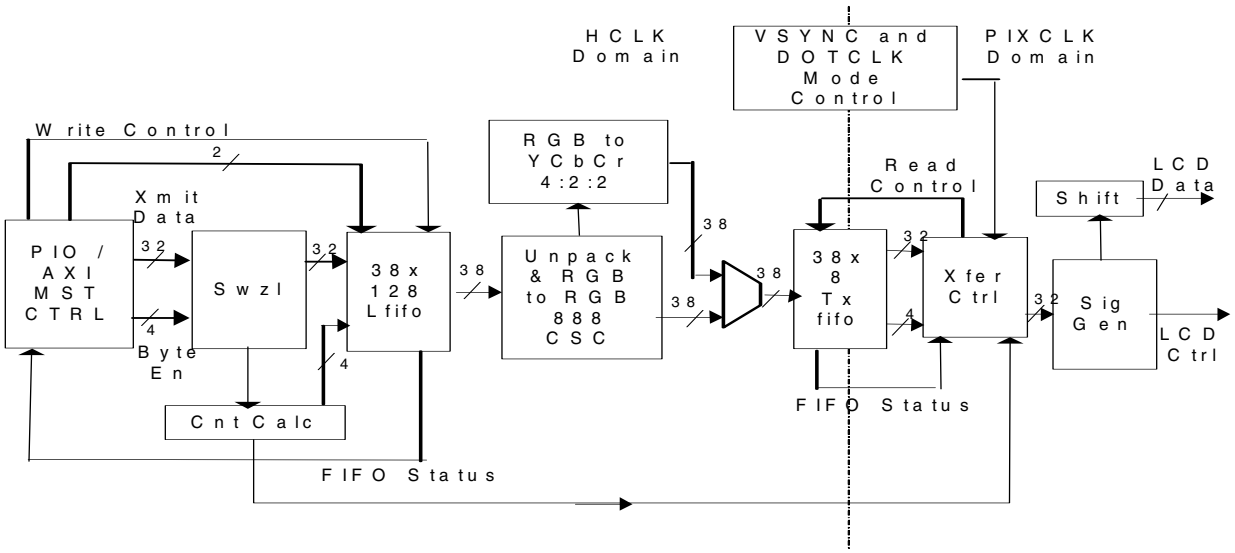


Figure 18-2. LCDIF Write DataPath

The examples in Figure 18-3–Figure 18-6 illustrate some different combinations of register programming for write mode. Assume that the data written into the HW_LCDIF_DATA register is of the format {A7–A0, B7–B0, C7–C0, D7–D0} in 8-bit mode and {A15–A0, B15–B0} in 16-bit mode.

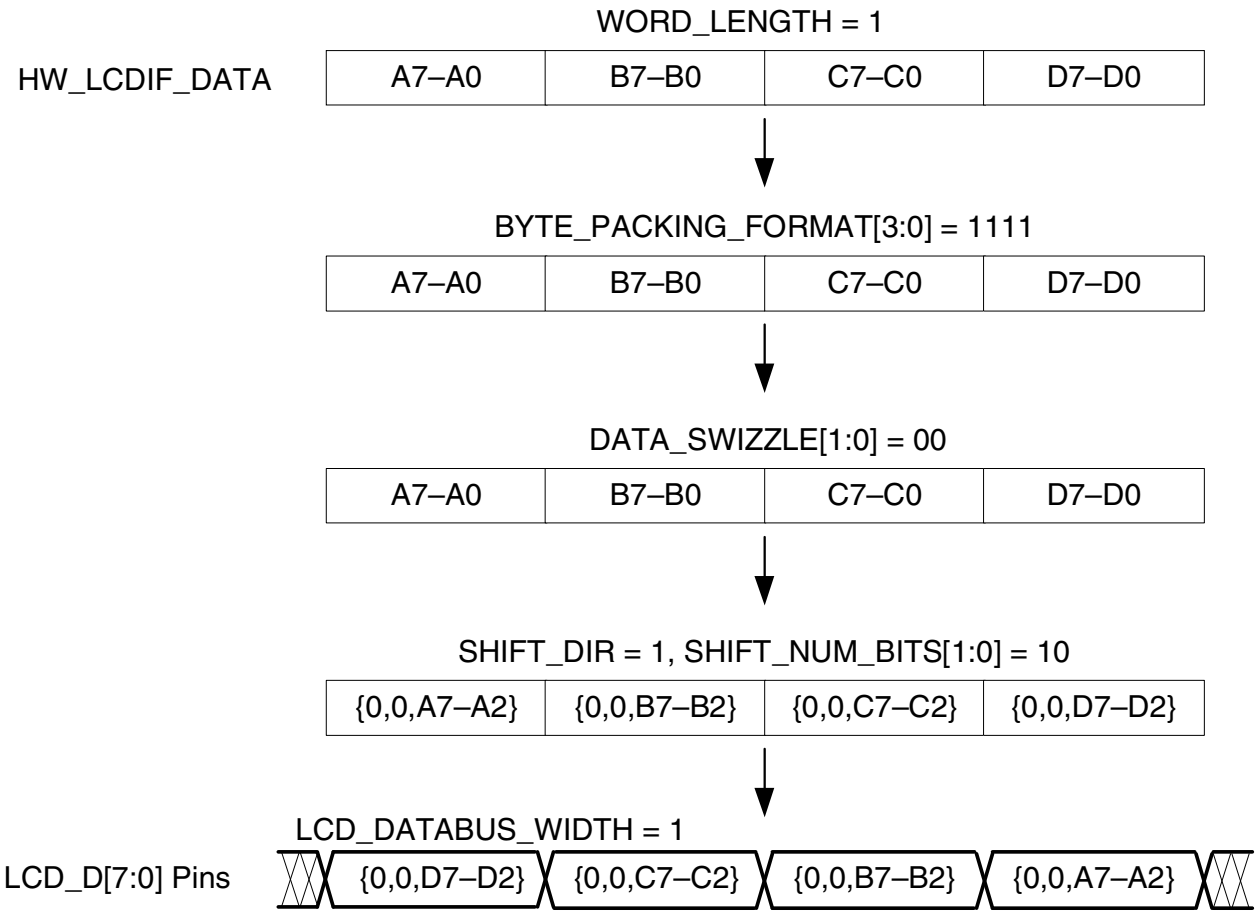


Figure 18-3. 8-Bit LCDIF Register Programming—Example A

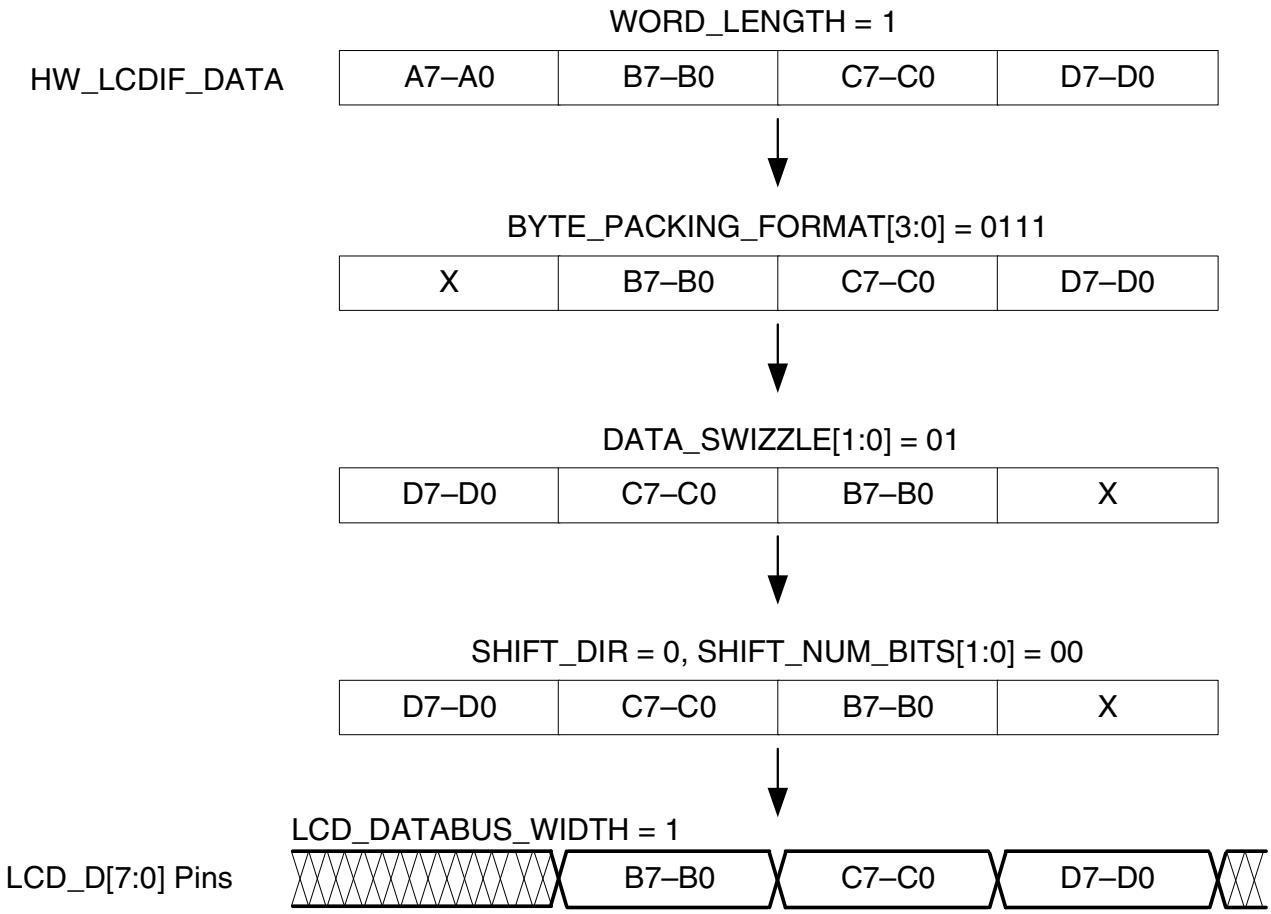


Figure 18-4. 8-Bit LCDIF Register Programming—Example B

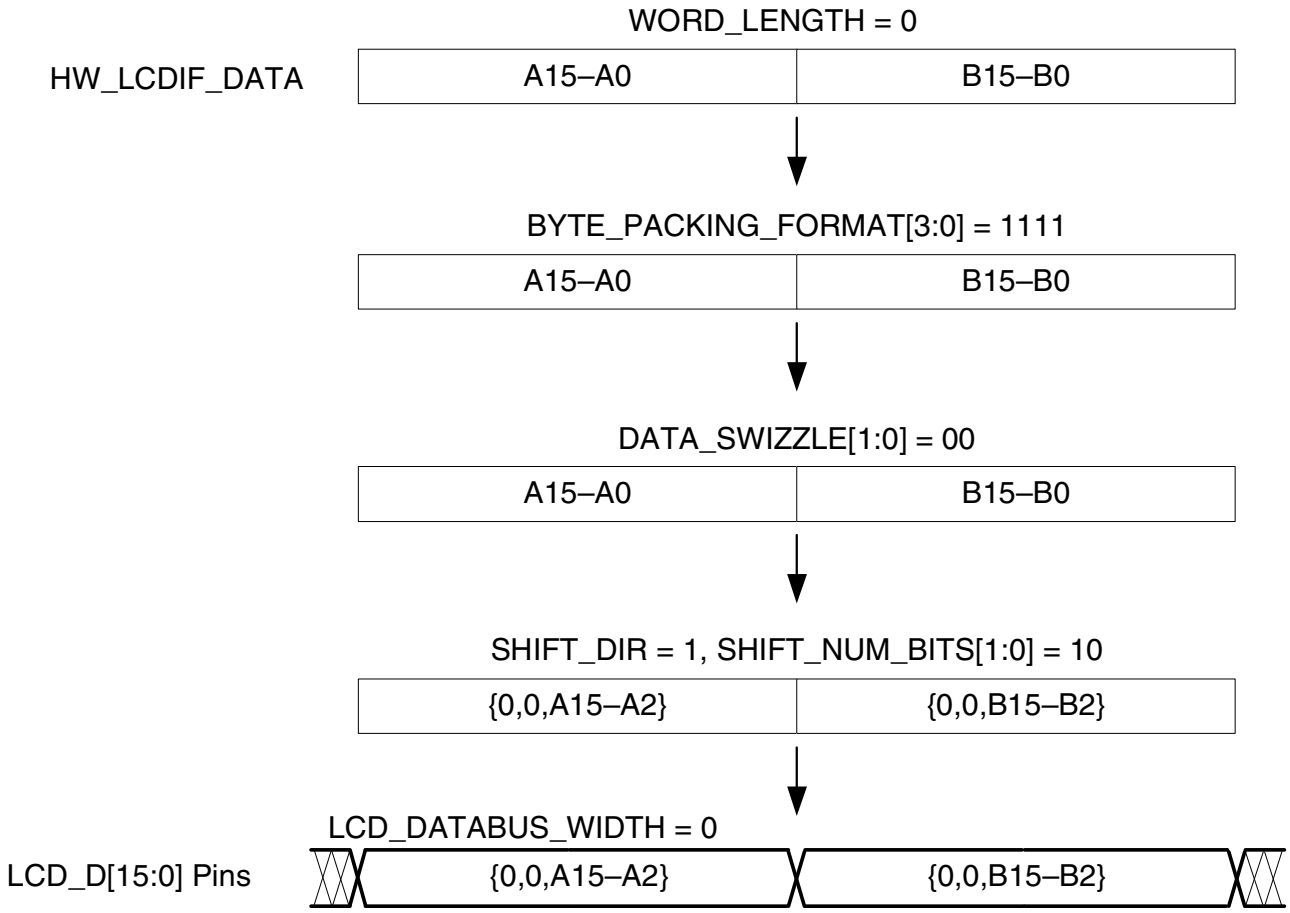


Figure 18-5. 16-Bit LCDIF Register Programming—Example A

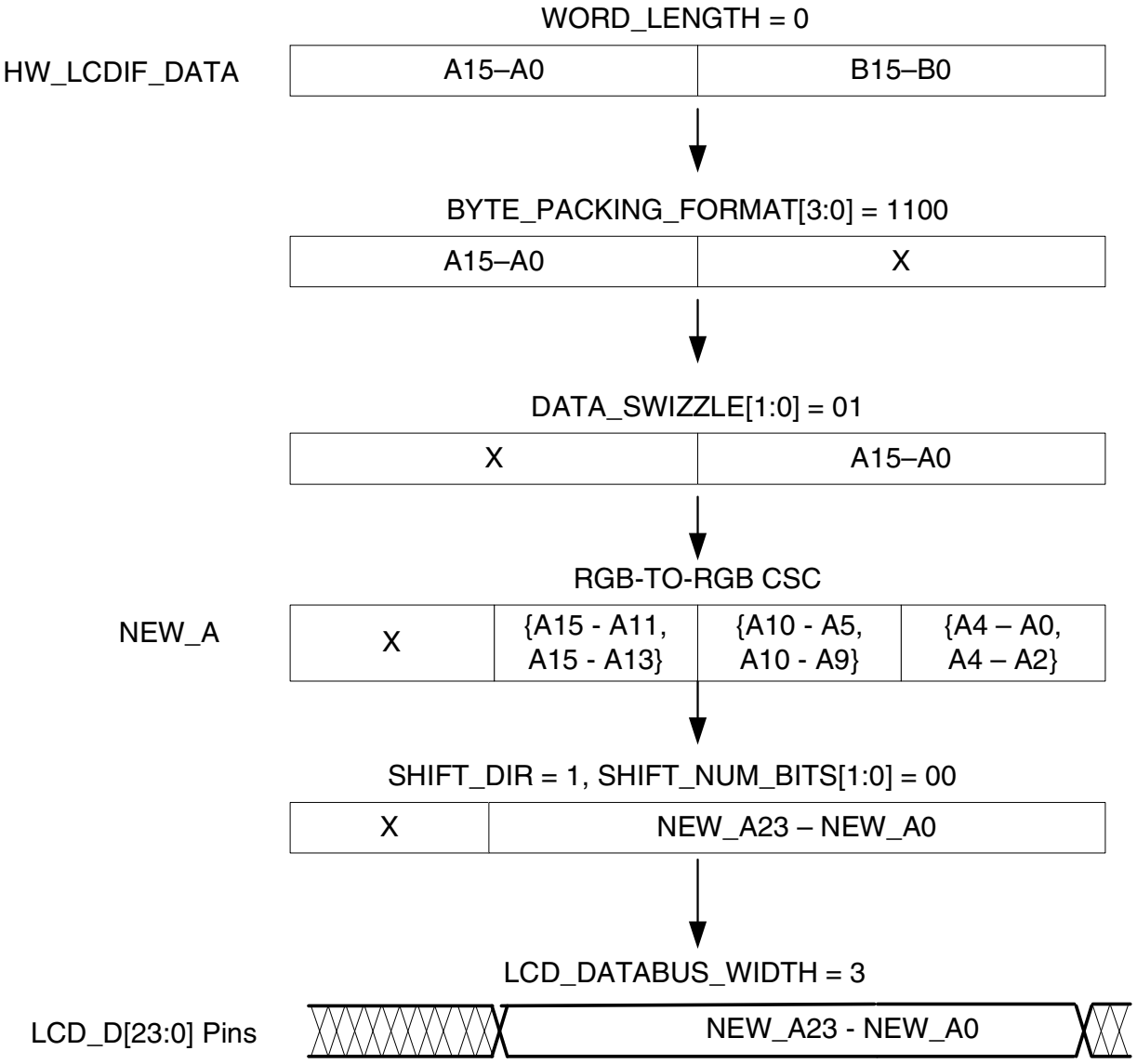


Figure 18-6. 16-Bit LCDIF Register Programming—Example B

18.2.3 LCDIF Interrupts

The LCDIF supports a number of interrupts to aid controlling and status reporting of the block. All the interrupts have individual mask bits for enabling or disabling each of them. They all get funneled through a single interrupt line connected to the interrupt collector (ICOLL). The following list describes the different interrupts supported by the LCDIF:

- Underflow interrupt is asserted when the clock domain crossing FIFO (TXFIFO) becomes empty but the block is in active display portion at that time. Software should take corrective action to make sure that this does not happen. This interrupt is of value only in DOTCLK and DVI modes.
- Overflow interrupt will be asserted in PIO mode if software writes to LFIFO while it is full.

- VSYNC edge interrupt will be asserted every time a leading VSYNC edge occurs.
- Cur_frame_done interrupt occurs at the end of every frame in all modes except DVI. In DVI mode, if IRQ_ON_ALTERNATE_FIELDS bit is set, it will occur at the end of every frame, otherwise it will occur at the end of every field. In the DOTCLK and DVI modes, the LCDIF will automatically copy the value in the HW_LCDIF_NEXT_BUF register to the HW_LCDIF_CUR_BUF register before issuing the cur_frame_done interrupt.
- Bus master error interrupt occurs when the LCDIF receives an AXI bus error from the slave memory it is trying to access. Software should debug the source of this error before proceeding further. The address corresponding to the error response is reflected in the BM_ERROR_STAT register.

18.2.4 Initializing the LCDIF

The following initialization steps are common to all LCDIF modes of operation before entering any particular mode.

Initialization Steps:

1. To select the pins and their directions for interfacing to the LCD panel, set the appropriate bits in the HW_PINCTRL_MUXSELx registers in the PINCTRL block.
2. Start the PIXCLK and set the appropriate frequency by programming the registers in CLKCTRL.
3. Bring the LCDIF out of soft reset and clock gate.
4. Reset the LCD controller by setting LCDIF_CTRL1_RESET bit appropriately, being careful to observe the reset requirements of the controller.

Select the bus interface mechanism with the LCDIF_MASTER bit in HW_LCDIF_CTRL register. If bus master mode is enabled (LCDIF_MASTER=1), set the HW_LCDIF_CUR_BUF and HW_LCDIF_NEXT_BUF registers by following the description in Section 18.2.1, “Bus Interface Mechanisms.

5. Set the INPUT_DATA_SWIZZLE according to the endianness of the LCD controller. Also set the DATA_SHIFT_DIR and SHIFT_NUM_BITS if it is required to shift the data left or right before it is output.
6. Set the WORD_LENGTH field appropriately, indicating the resolution of the pixels in the frame buffer: 0 = 16-bit pixel, 1 = 8-bit pixel, 2 = 18-bit pixel, 3 = 24-bit pixel. Also select the correct 16/18/24 bit data format with the corresponding fields in HW_LCDIF_CTRL register.
7. Set the BYTE_PACKING_FORMAT field in HW_LCDIF_CTRL1 according to the input frame.
8. Set the LCD_DATABUS_WIDTH appropriately: 0 = 16-bit output, 1 = 8-bit output, 2 = 18-bit output, 3 = 24-bit output.
9. Enable the necessary IRQs.

The following sections are dedicated to detailing the different external interfaces supported by LCDIF.

18.2.5 System Interface

The System interface is used to transfer data and commands between the internal buffer of an LCD controller/display and the ARM processor at relatively lower speeds. The LCDIF supports both the 6800 and the 8080 MCU protocols. If DOTCLK_MODE, DVI_MODE and VSYNC_MODE bits in HW_LCDIF_CTRL registers are 0, it implies that the block is in system interface mode. The LCDIF system mode has four basic timing parameters: Setup and Hold for the Command/Data register selection (TCS, TCH) and Setup and Hold for the Data bus (TDS, TDH). These parameters are expressed in PIXCLK cycles. The LCD_WR signal is used as the write strobe while LCD_RS (Register Select) signal is typically used to switch between command and data modes.

Figure 18-7 shows the timing-related information in the write mode of both 6800 and 8080 protocols.

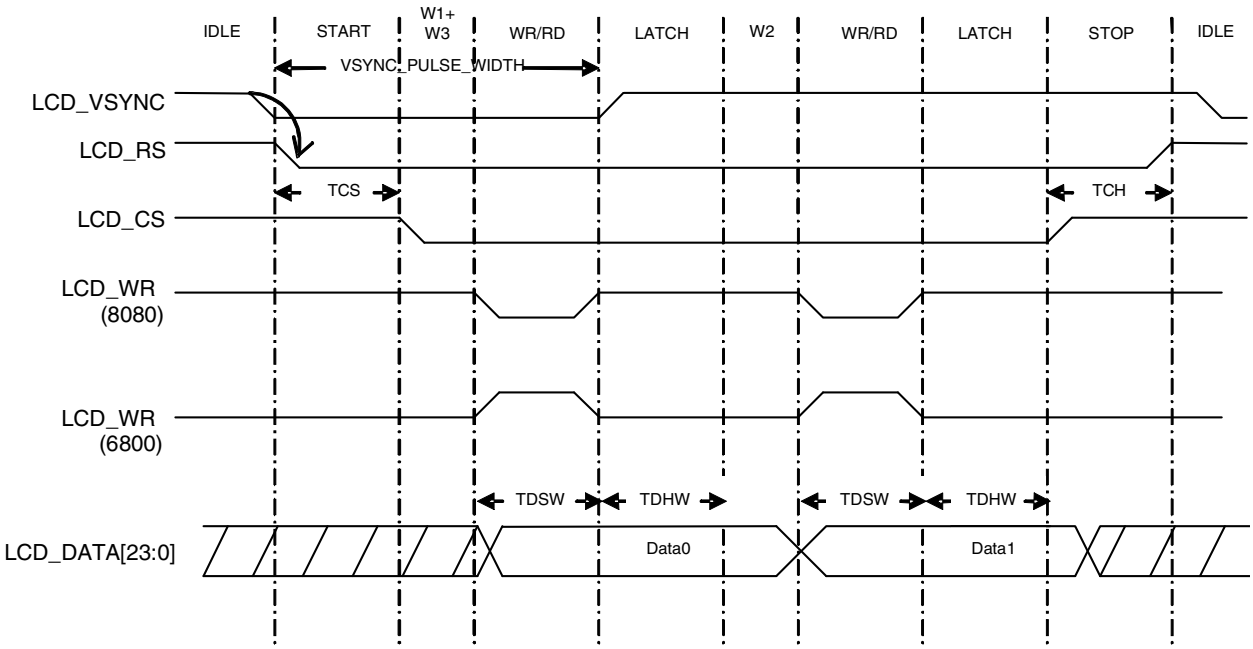


Figure 18-7. LCD Interface Signals in System Write Mode

In the 6800 mode, the LCD_WR pin acts as an active high data strobe (ENABLE) and in the 8080 mode, it acts as an active low data strobe (WRn). The LCDIF has flexible pin and strobe timings which enable it to optimally support a wide range of LCDs. The minimum cycle time is two PIXCLK cycles (TDS=TDH=1). For example, this results in a maximum LCD data rate of 12MB/s when PIXCLK is 24 MHz. TDS and TDH are 8-bit values, so the minimum LCDIF period is 510 PIXCLK cycles (47 kHz with a 24-MHz PIXCLK). The timings are not automatically adjusted if the PIXCLK frequency changes, so it may be necessary to adjust the timings if PIXCLK changes.

In the system interface mode, the HW_LCDIF_CTRL_ BYPASS_COUNT bit must be 0. The RUN bit is cleared automatically once the LCDIF has transmitted all the data as per the HW_LCDIF_TRANSFER_COUNT register and has completed the transfer to the panel. The current transfer can be cancelled/aborted if the RUN bit is manually set to 0.

18.2.5.1 Code Example to initialize LCDIF in System mode

```
// Note: Common initialization steps in Section 18.2.4, "Initializing the LCDIF," must also be
// executed along with the following code
BF_CS1(LCDIF_CTRL, DATA_SELECT, 1); // 0 if sending command, 1 if sending data. Note that the idle
state for LCD_RS signal is high, regardless of the programming of the DATA_SELECT register.
BF_CS1 (LCDIF_CTRL, MODE86, 8080_MODE);
BF_CS1 (LCDIF_CTRL, BYPASS_COUNT, 0); //Must be 0 in system mode
BF_CS1 (LCDIF_CTRL1, BUSY_ENABLE, 1);//Only if LCD controller implements a busy line
BF_CS4 (LCDIF_TIMING, CMD_HOLD, 2, CMD_SETUP, 2, DATA_HOLD, 2, DATA_SETUP, 2);  //Values based on
PIXCLK frequency and timing requirements of controller. Note that these register must be non-zero
for correct operation.
BF_CS2 (LCDIF_TRANSFER_COUNT, H_COUNT, 320, V_COUNT, 240); //For a 320 RGB x 240 display
BF_CS1 (LCDIF_CTRL, RUN, 1);
```

The LCDIF is now ready to receive data through PIO writes to the HW_LCDIF_DATA register or fetch data directly from memory as a bus master. Also, note that, while in PIO mode, the software will need to poll the FIFO STATUS bits to ensure that it does not overflow the LCDIF data buffers. When LCDIF is done transmitting H_COUNT x V_COUNT pixels, it will stop, turn off the RUN bit and assert the cur_frame_done interrupt.

18.2.6 VSYNC Interface

The VSYNC interface uses the same protocol as the System interface, with an additional signal VSYNC at the frame rate of the display, as shown in Figure 18-7. It is used in the moving picture display mode where data has to be written to the internal LCD buffer at a speed higher than the display rate and displayed in synchronization with the VSYNC signal. This mode is selected by setting the VSYNC_MODE bit in HW_LCDIF_CTRL register. The VSYNC signal is programmable for period, polarity and direction. Many other programmable parameters are shared with the System interface. The VSYNC_OEB bit in HW_LCDIF_VDCTRL0 register indicates whether the display controller will send the VSYNC signal, or whether it should be generated by the LCDIF. The timing of the VSYNC signal is based on the PIXCLK (make sure VSYNC_PULSE_WIDTH_UNIT = VSYNC_PERIOD_UNIT = 0 and VSYNC_ONLY = 1) and it is determined by the VSYNC_PERIOD, VSYNC_PULSE_WIDTH and VSYNC_POL fields in HW_LCDIF_VDCTRL0-4 registers. The SYNC_SIGNALS_ON bit in HW_LCDIF_VDCTRL4 register must be set if the target requires the VSYNC signal to be generated by the LCDIF. If the WAIT_FOR_VSYNC_EDGE bit in HW_LCDIF_CTRL register is set, it indicates that the hardware should wait until it sees the leading VSYNC edge before starting the data transfer. The VERTICAL_WAIT_CNT indicates the number of PIXCLKs from the leading VSYNC edge after which data transfer will be started on the interface.

In the VSYNC interface mode, the HW_LCDIF_CTRL_ BYPASS_COUNT bit must be 0. The RUN bit is cleared automatically once the LCDIF has received/transmitted all the data as per the HW_LCDIF_TRANSFER_COUNT register and has completed the transfer to the panel. The current transfer can be cancelled/aborted if the RUN bit is manually set to 0.

18.2.6.1 Code Example to initialize LCDIF in VSYNC mode

```
// Note: Common initialization steps in Section 18.2.4, "Initializing the LCDIF," must also be
// executed along with the following code
BF_CS1 (LCDIF_CTRL, DATA_SELECT, 1); // 0 if sending command, 1 if sending data. Note that //the
idle state for LCD_RS signal is high, regardless of the programming of the DATA_SELECT //register.
BF_CS1 (LCDIF_CTRL, MODE86, 8080_MODE);
BF_CS1 (LCDIF_CTRL, BYPASS_COUNT, 0); //Must be 0 in system mode
BF_CS1 (LCDIF_CTRL1, BUSY_ENABLE, 0);
BF_CS4 (LCDIF_TIMING, CMD_HOLD, 2, CMD_SETUP, 2, DATA_HOLD, 2, DATA_SETUP, 2);  //Values //based on
PIXCLK frequency and timing requirements of controller. Note that these register //must be non-zero
for the system and VSYNC modes.
BF_CS2 (LCDIF_TRANSFER_COUNT, H_COUNT, 320, V_COUNT, 240);//For a 320 RGB x 240 display
//The following section indicates setting up the VSYNC signal timing when VSYNC is an output
BF_CS1 (LCDIF_VDCTRL0, VSYNC_OEB, 0); //Making VSYNC signal an output
BF_CS1  (LCDIF_VDCTRL4,  VSYNC_ONLY,  1);  //Only  need  to  generate  VSYNC  signal,  not  HSYNC/DOT-
CLK/ENABLE
BF_CS1  (VDCTRL0,  VSYNC_POL,  0);  //Setting  the  polarity  of  VSYNC  signal  to  be  low  during
//VSYNC_PULSE_WIDTH time
BF_CS2 (LCDIF_VDCTRL0, VSYNC_PERIOD_UNIT, 0, VSYNC_PULSE_WIDTH_UNIT, 0);
BF_CS2  (LCDIF_VDCTRL1,  VSYNC_PERIOD,  400000,  VSYNC_PULSE_WIDTH,  100);//Frame  display  rate  in
//terms of number of PIXCLKs.
BF_CS2 (LCDIF_VDCTRL2, HSYNC_PULSE_WIDTH, 0, HSYNC_PERIOD, 0);
BF_CS1 (LCDIF_VDCTRL3, VERTICAL_WAIT_CNT, 50);
BF_CS1 (LCDIF_VDCTRL4, SYNC_SIGNALS_ON, 1);
BF_CS2 (LCDIF_CTRL, VSYNC_MODE, 1, WAIT_FOR_VSYNC_EDGE, 1); //set WAIT_FOR_VSYNC_EDGE if //software
wishes to transfer the next frame after the VSYNC edge occurs.
BF_CS1 (LCDIF_CTRL, RUN, 1);
```

The LCDIF is now ready to receive data through PIO writes to the HW_LCDIF_DATA register or fetch data directly from memory as a bus master. When LCDIF is done transmitting H_COUNT x V_COUNT pixels, it will stop, turn off the RUN bit and assert the cur_frame_done interrupt.

18.2.7 DOTCLK Interface

The DOTCLK interface is another mode used in moving picture displays that constantly require display refreshes. It includes the VSYNC, HSYNC, DOTCLK and (optional) ENABLE signals. The interface is popularly called the RGB interface if the ENABLE signal is present.

Figure 18-8 shows the DOTCLK protocol with its programmable parameters.

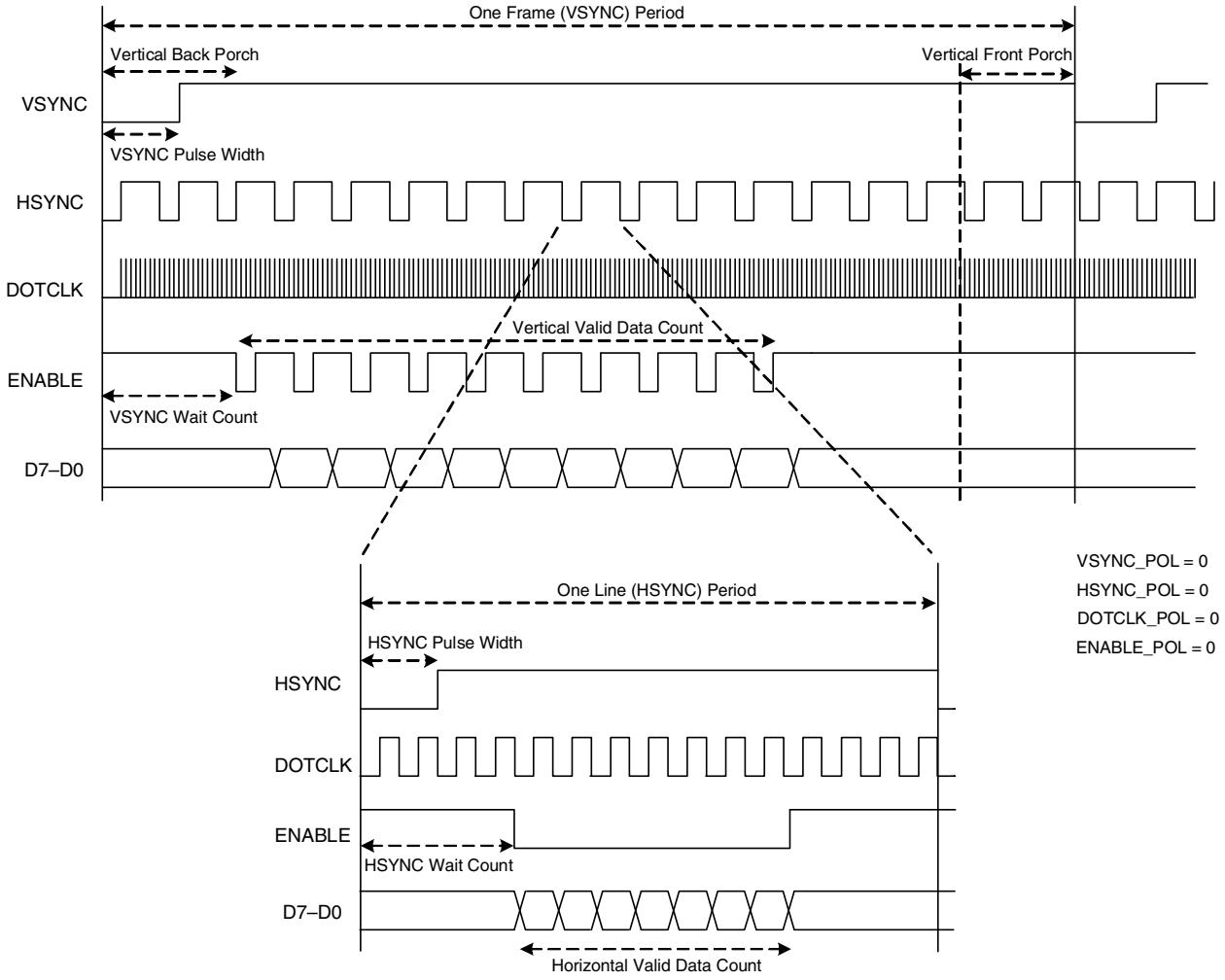


Figure 18-8. LCD Interface Signals in DOTCLK Mode

The DOTCLK mode writes data at high speed to the LCD, and the display operation is synchronized with the VSYNC, HSYNC, ENABLE and DOTCLK signals. The polarities, periods and pulse-widths of the sync signals are programmable using the HW_LCDIF_VDCTRL0–4 registers. The units for the VSYNC signal must be number of horizontal lines and can be set using the VSYNC_PULSE_WIDTH_UNIT and VSYNC_PERIOD_UNIT bit fields. The VERTICAL_WAIT_CNT is by default given the same unit as the VSYNC_PERIOD. The PIXCLK is controlled using the HW_CLKCTRL_PIX, HW_CLKCTRL_FRAC, and HW_CLKCTRL_CLKSEQ registers in the CLCKTRL block.

In DOTCLK mode, HW_LCDIF_CTRL_BYPASS_COUNT bit must be set to 1. To end the current transfer, the software should make the DOTCLK_MODE bit 0, so that all data that is currently in the LCDIF FIFOs is transmitted. Once that transfer is complete, the block will automatically clear the RUN bit and issue the cur_frame_done interrupt.

18.2.7.1 Code Example

The following code shows an example for programming a 320x240 display. Note that setting up the display must be done through the System mode or via SPI.

```
// Note: Common initialization steps in Section 18.2.4, "Initializing the LCDIF," must also be
// executed along with the following code
BF_CS1 (LCDIF_CTRL, DOTCLK_MODE, 1);
BF_CS1 (LCDIF_CTRL, BYPASS_COUNT, 1); //Always for DOTCLK mode
BF_CS1 (LCDIF_VDCTRL0, VSYNC_OEB, 0); //Vsync is always an output in the DOTCLK mode
BF_CS4 (LCDIF_VDCTRL0, VSYNC_POL, 0, HSYNC_POL, 0, DOTCLK_POL, 0, ENABLE_POL, 0);
BF_CS1 (LCDIF_VDCTRL0, ENABLE_PRESENT, 1);
BF_CS2 (LCDIF_VDCTRL0, VSYNC_PERIOD_UNIT, 1, VSYNC_PULSE_WIDTH_UNIT, 1);
BF_CS1 (LCDIF_VDCTRL0, VSYNC_PULSE_WIDTH, 2);
BF_CS1 (LCDIF_VDCTRL1, VSYNC_PERIOD, 280);
BF_CS2 (LCDIF_VDCTRL2, HSYNC_PULSE_WIDTH, 10, HSYNC_PERIOD, 360); //Assuming LCD_DATABUS_WIDTH
                                                                  //is 24bit
BF_CS2 (LCDIF_VDCTRL3, VSYNC_ONLY, 0);
BF_CS2 (LCDIF_VDCTRL3, HORIZONTAL_WAIT_CNT, 20, VERTICAL_WAIT_CNT, 20);
BF_CS1 (LCDIF_VDCTRL4, DOTCLK_H_VALID_DATA_CNT, 320);//Note that DOTCLK_V_VALID_DATA_CNT is
                                 //implicitly assumed to be HW_LCDIF_TRANSFER_COUNT_V_COUNT
BF_CS1 (LCDIF_VDCTRL4, SYNC_SIGNALS_ON, 1);
BF_CS1 (LCDIF_CTRL, RUN, 1);
```

18.2.8 ITU-R BT.656 Digital Video Interface (DVI)

ITU-R BT.656 Digital Video Interface shown in Figure 18-9 transmits 4:2:2 YCbCr digital component video to the digital video encoder block (TVENC) that can translate it into 525/60 or 625/50 analog TV signal. Unique timing codes (timing reference signals) are embedded within the video stream to indicate the different timing events that would have been otherwise indicated by VSYNC, HSYNC and BLANK signals. The hardware supports 8-bit data transfers; the pins are shared with the lower 8 bits of LCD data bus. The LCD_RS pin is shared with the clock signal of the interface (called CCIRCLK here for uniqueness). CCIRCLK also can be obtained on the LCD_DOTCK pin. The DVI mode shares the write FIFO with the LCD interface and the associated pipeline. The programmable parameters in registers HW_LCDIF_DVICTRL0-3 allow setting the total number of horizontal lines per frame, vertical and horizontal blanking interval, odd and even field start and end positions, etc. In short, these parameters are provided to ensure that the hardware has enough flexibility to generate the right 525/60 or 625/50 data streams. Most of the initialization steps in Section 18.2.4, "Initializing the LCDIF," such as data shifting, swizzle, etc., are applicable to DVI mode also. The register descriptions in Section 18.4, "Programmable Registers," include example code for programming the DVICTRL0-4 registers.

In DVI mode, HW_LCDIF_CTRL_BYPASS_COUNT bit must be set to 1. To end the current transfer, the software should make the DVI_MODE bit the value 0, so that all data that is currently in the LCDIF LFIFO and TXFIFO is transmitted. Once that transfer is complete, the block will automatically clear the RUN bit and assert the cur_frame_done interrupt.

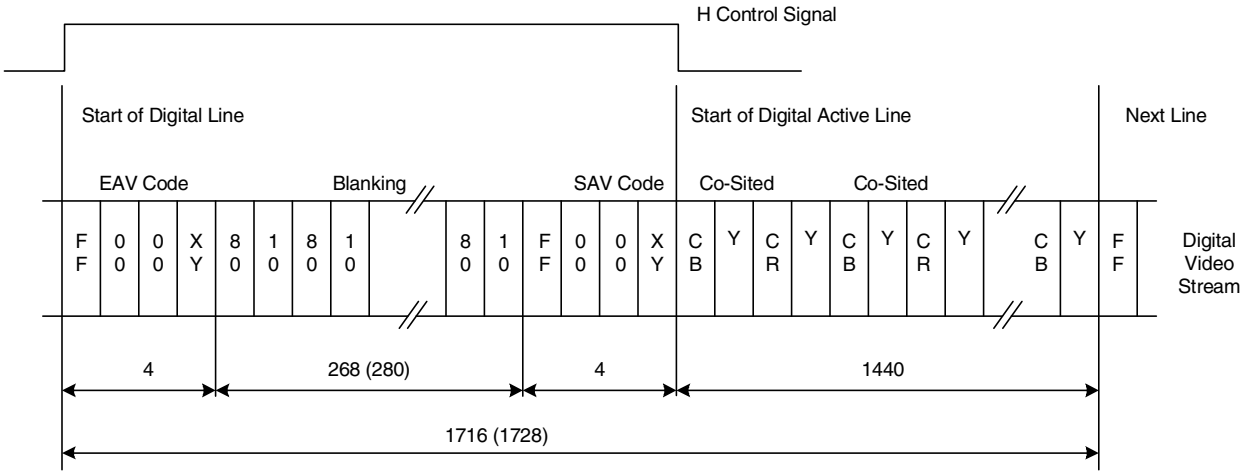


Figure 18-9. LCDIF Interface Signals in ITU-R BT.656 Digital Video Interface Mode

18.2.9 LCDIF Pin Usage by Interface Mode

The following table shows the pin usage for all of the supported modes when the HW_PINCTRL_MUXSEL2 and HW_PINCTRL_MUXSEL3 registers are programmed for LCD functionality. See Chapter 37, "Pin Control and GPIO," for a more complete description of pin multiplexing options and how to program each pin individually.

Notes:

1. When LCD_DATABUS_WIDTH is more than 8 bits, the R component is on the MSB bits of the LCD data pins, the G component is in the middle and the B component is on the LSB bits. For example, if LCD_DATABUS_WIDTH = 16 bits, LCD_D15 - LCD_D11 = R[4:0], LCD_D10 -LCD_D05 = G[5:0], LCD_D04 - LCD_D00 = B[4:0].

2. The VSYNC signal has been mapped onto two pins, LCD_BUSY and LCD_VSYNC. The PINCTRL block can be programmed to select either of those pins to function as VSYNC.

3. There is an option to internally mux the HSYNC, DOTCLK and ENABLE signals in the DOTCLK mode onto the LCD_D14-LCD_12 pins by setting the MUX_SYNC_SIGNALS bit in the VDCTRL0 register for backward compatibility with previous generation SoCs.

Table 18-1. Pin Usage in System Mode and VSYNC Mode

PIN	SYS - 8	SYS - 16	SYS -18	SYS - 24	VSYNC - 8	VSYNC - 16	VSYNC - 18	VSYNC - 24
LCD_RS	LCD_ RS	LCD_ RS	LCD_ RS	LCD_RS	LCD_RS	LCD_ RS	LCD_RS	LCD_RS
LCD_CS	LCD_CS	LCD_CS	LCD_ CS	LCD_CS	LCD_CS	LCD_CS	LCD_ CS	LCD_CS
LCD_WR	LCD_WR	LCD_ WR	LCD_WR	LCD_WR	LCD_WR	LCD_ WR	LCD_WR	LCD_WR
LCD_VSYNC*	X	X	X	X	LCD_VSYNC	LCD_VSYNC	LCD_ VSYNC	LCD_VSYNC
LCD_HSYNC	X	X	X	X	X	X	X	X

Table 18-1. Pin Usage in System Mode and VSYNC Mode (continued)

LCD_DOTCLK	X	X	X	X	X	X	X	X
LCD_ENABLE	X	X	X	X	X	X	X	X
LCD_D23	X	X	X	LCD_D23	X	X	X	LCD_D23
LCD_D22	X	X	X	LCD_D22	X	X	X	LCD_D22
LCD_D21	X	X	X	LCD_D21	X	X	X	LCD_D21
LCD_D20	X	X	X	LCD_D20	X	X	X	LCD_D20
LCD_D19	X	X	X	LCD_D19	X	X	X	LCD_D19
LCD_D18	X	X	X	LCD_D18	X	X	X	LCD_D18
LCD_D17	X	X	LCD_D17	LCD_D17	X	X	LCD_D17	LCD_D17
LCD_D16	X	X	LCD_D16	LCD_D16	X	X	LCD_D16	LCD_D16
LCD_D15*	X	LCD_D15	LCD_D15	LCD_D15	VSYNC (optional)	LCD_D15	LCD_D15	LCD_D15
LCD_D14 *	X	LCD_D14	LCD_D14	LCD_D14	X	LCD_D14	LCD_D14	LCD_D14
LCD_D13*	X	LCD_D13	LCD_D13	LCD_D13	X	LCD_D13	LCD_D13	LCD_D13
LCD_D12*	X	LCD_D12	LCD_D12	LCD_D12	X	LCD_D12	LCD_D12	LCD_D12
LCD_D11	X	LCD_D11	LCD_D11	LCD_D11	X	LCD_D11	LCD_D11	LCD_D11
LCD_D10	X	LCD_D10	LCD_D10	LCD_D10	X	LCD_D10	LCD_D10	LCD_D10
LCD_D09	X	LCD_D09	LCD_D09	LCD_D09	X	LCD_D09	LCD_D09	LCD_D09
LCD_D08	X	LCD_D08	LCD_D08	LCD_D08	X	LCD_D08	LCD_D07	LCD_D08
LCD_D07	LCD_D07	LCD_D07	LCD_D07	LCD_D07	LCD_D07	LCD_D07	LCD_D07	LCD_D07
LCD_D06	LCD_D06	LCD_D06	LCD_D06	LCD_D06	LCD_D06	LCD_D06	LCD_D06	LCD_D06
LCD_D05	LCD_D05	LCD_D05	LCD_D05	LCD_D05	LCD_D05	LCD_D05	LCD_D05	LCD_D05
LCD_D04	LCD_D04	LCD_D04	LCD_D04	LCD_D04	LCD_D04	LCD_D04	LCD_D04	LCD_D04
LCD_D03	LCD_D03	LCD_D03	LCD_D03	LCD_D03	LCD_D03	LCD_D03	LCD_D03	LCD_D03
LCD_D02	LCD_D02	LCD_D02	LCD_D02	LCD_D02	LCD_D02	LCD_D02	LCD_D02	LCD_D02
LCD_D01	LCD_D01	LCD_D01	LCD_D01	LCD_D01	LCD_D01	LCD_D01	LCD_D01	LCD_D01
LCD_D00	LCD_D00	LCD_D00	LCD_D00	LCD_D00	LCD_D00	LCD_D00	LCD_D00	LCD_D00
LCD_RESET	LCD_RESET	LCD_RESET	LCD_RESET	LCD_RESET	LCD_RESET	LCD_RESET	LCD_RESET	LCD_RESET
LCD_BUSY/ LCD_VSYNC	LCD_BUSY	LCD_BUSY	LCD_BUSY	LCD_BUSY	LCD_BUSY (OR optional LCD_VSYNC)	LCD_BUSY (OR optional LCD_VSYNC)	LCD_BUSY (OR optional LCD_VSYNC)	LCD_BUSY (OR optional LCD_VSYNC)

Table 18-2. Pin Usage in DOTCLK Mode and DVI Mode

PIN	DOTCLK - 8	DOTCLK - 16	DOTCLK - 18	DOTCLK - 24	DVI - 8
LCD_RS	X	X	X	X	CCIR_CLK
LCD_CS	X	X	X	X	X
LCD_WR	X	X	X	X	X

Table 18-2. Pin Usage in DOTCLK Mode and DVI Mode (continued)

LCD_VSYNC (Two options)	LCD_VSYNC	LCD_VSYNC	LCD_VSYNC	LCD_VSYNC	X
LCD_HSYNC	LCD_HSYNC	LCD_HSYNC	LCD_HSYNC	LCD_HSYNC	X
LCD_DOTCLK	LCD_DOTCLK	LCD_DOTCLK	LCD_DOTCLK	LCD_DOTCLK	X
LCD_ENABLE	LCD_ENABLE	LCD_ENABLE	LCD_ENABLE	LCD_ENABLE	X
LCD_D23	X	X	X	LCD_D23	X
LCD_D22	X	X	X	LCD_D22	X
LCD_D21	X	X	X	LCD_D21	X
LCD_D20	X	X	X	LCD_D20	X
LCD_D19	X	X	X	LCD_D19	X
LCD_D18	X	X	X	LCD_D18	X
LCD_D17	X	X	LCD_D17	LCD_D17	X
LCD_D16	X	X	LCD_D16	LCD_D16	X
LCD_D15*	X	LCD_D15	LCD_D15	LCD_D15	X
LCD_D14*	X	LCD_D14	LCD_D14	LCD_D14	X
LCD_D13*	X	LCD_D13	LCD_D13	LCD_D13	X
LCD_D12*	X	LCD_D12	LCD_D12	LCD_D12	X
LCD_D11	X	LCD_D11	LCD_D11	LCD_D11	X
LCD_D10	X	LCD_D10	LCD_D10	LCD_D10	X
LCD_D09	X	LCD_D09	LCD_D09	LCD_D09	X
LCD_D08	X	LCD_D08	LCD_D08	LCD_D08	X
LCD_D07	LCD_D07	LCD_D07	LCD_D07	LCD_D07	LCD_D07
LCD_D06	LCD_D06	LCD_D06	LCD_D06	LCD_D06	LCD_D06
LCD_D05	LCD_D05	LCD_D05	LCD_D05	LCD_D05	LCD_D05
LCD_D04	LCD_D04	LCD_D04	LCD_D04	LCD_D04	LCD_D04
LCD_D03	LCD_D03	LCD_D03	LCD_D03	LCD_D03	LCD_D03
LCD_D02	LCD_D02	LCD_D02	LCD_D02	LCD_D02	LCD_D02
LCD_D01	LCD_D01	LCD_D01	LCD_D01	LCD_D01	LCD_D01
LCD_D00	LCD_D00	LCD_D00	LCD_D00	LCD_D00	LCD_D00
LCD_RESET	LCD_RESET	LCD_RESET	LCD_RESET	LCD_RESET	X
LCD_BUSY / LCD_VSYNC	LCD_BUSY (OR optional LCD_VSYNC)	LCD_BUSY (OR optional LCD_VSYNC)	LCD_BUSY (OR optional LCD_VSYNC)	LCD_BUSY (OR optional LCD_VSYNC)	X

18.3 Behavior During Reset

Note that HCLK and PIXCLK must be running before making any changes to SFTRST or CLKGATE bits. A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when

setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, "Correct Way to Soft Reset a Block," for additional information on using the SFTRST and CLKGATE bit fields.

18.4 Programmable Registers

The LCD interface block contains the following directly programmable registers. The starting address in LCDIF's memory map is 0x80030000 and all the register addresses mentioned below are offset from this address. For example, address of HW_LCDIF_CTRL is 0x80030000 and HW_LCDIF_CTRL1 is 0x80030010.

18.4.1 LCDIF General Control Register Description

The LCD Interface Control Register provides overall control of the LCDIF block.

HW_LCDIF_CTRL	0x000
HW_LCDIF_CTRL_SET	0x004
HW_LCDIF_CTRL_CLR	0x008
HW_LCDIF_CTRL_TOG	0x00C

Table 18-3. HW_LCDIF_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	YCBCR422_INPUT	RSVD3	WAIT_FOR_VSYNC_EDGE	DATA_SHIFT_DIR	SHIFT_NUM_BITS					DVI_MODE	BYPASS_COUNT	VSYNC_MODE	DOTCLK_MODE	DATA_SELECT	INPUT_DATA_SWIZZLE		CSC_DATA_SWIZZLE		LCD_DATABUS_WIDTH		WORD_LENGTH		RGB_TO_YCBCR422_CSC	RSVD2	LCDIF_MASTER	RSVD1	DATA_FORMAT_16_BIT	DATA_FORMAT_18_BIT	DATA_FORMAT_24_BIT	RUN

Table 18-4. HW_LCDIF_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	This bit must be set to zero to enable normal operation of the LCDIF. When set to one, it forces a block level reset.
30	**CLKGATE**	RW	0x1	This bit must be set to zero for normal operation. When set to one it gates off the clocks to the block.

Table 18-4. HW_LCDIF_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
29	**YCBCR422_INPUT**	RW	0x0	Zero implies input data is in RGB color space. One implies input data is in YCbCr 4:2:2 format, such that YCbYCr are packed in a 32-bit word. It also means that there are 2 pixels in 4 bytes. If this bit is set, software should program the H_COUNT field in the TRANSFER_COUNT register to the total number of pixels that will have to be fetched by the LCDIF block per line and the BYTE_PACKING_FORMAT should be 0xF. The WORD_LENGTH does not matter in this case.
28	**RSVD3**	RO	0x0	Reserved bits. Write as 0.
27	**WAIT_FOR_VSYNC_EDGE**	RW	0x0	Setting this bit to 1 will make the hardware wait for the triggering VSYNC edge before starting write transfers to the LCD. Used only in the VSYNC mode of operation.
26	**DATA_SHIFT_DIR**	RW	0x0	Use this bit to determine the direction of shift of transmit data. In the DVI mode, it works only on the active data, not on the timing codes and ancillary data. TXDATA_SHIFT_LEFT = 0x0 Data to be transmitted is shifted LEFT by SHIFT_NUM_BITS bits. TXDATA_SHIFT_RIGHT = 0x1 Data to be transmitted is shifted RIGHT by SHIFT_NUM_BITS bits.
25:21	**SHIFT_NUM_BITS**	RW	0x0	The data to be transmitted is shifted left or right by this number of bits.
20	**DVI_MODE**	RW	0x0	Set this bit to 1 to get into theITU-R BT.656 digital video interface mode. Toggle this bit from 1 to 0 to make the hardware go out of DVI mode after completing all data transfer and deassert the RUN bit.
19	**BYPASS_COUNT**	RW	0x0	When this bit is 0, it means that LCDIF will stop the block operation and turn off the RUN bit after the amount of data indicated by the HW_LCDIF_TRANSFER_COUNT register has been transferred out. When this bit is set to 1, the block will continue normal operation indefinitely until it is told to stop. This bit must be 0 in system and VSYNC modes, and must be 1 in DOTCLK and DVI modes of operation.
18	**VSYNC_MODE**	RW	0x0	Setting this bit to 1 will make the LCDIF hardware go into VSYNC mode. WAIT_FOR_VSYNC_EDGE can be used only if this bit is set. If VSYNC signal is required to be an output from the block, SYNC_SIGNALS_ON bit in HW_LCDIF_VDCTRL4 register must be set.
17	**DOTCLK_MODE**	RW	0x0	Set this bit to 1 to make the hardware go into the DOTCLK mode, i.e. VSYNC/HSYNC/DOTCLK/ENABLE interface mode. ENABLE is optional, selected by the ENABLE_PRESENT bit. Toggle this bit from 1 to 0 to make the hardware go out of DOTCLK mode after completing all data transfer and deasserting the RUN bit.
16	**DATA_SELECT**	RW	0x0	Command Mode polarity bit. This bit should only be changed when RUN is 0. CMD_MODE = 0x0 Command Mode. DCn signal is Low. DATA_MODE = 0x1 Data Mode. DCn signal is High.

Table 18-4. HW_LCDIF_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:14	**INPUT_DATA_SWIZZLE**	RW	0x0	This field specifies how to swap the bytes either in the HW_LCDIF_DATA register or those fetched by the AXI master part of LCDIF.The swizzle function is independent of the WORD_LENGTH bit. See the explanation of the HW_LCDIF_DATA below for names and definitions of data register fields. The supported swizzle configurations are: NO_SWAP = 0x0 No byte swapping.(Little endian) LITTLE_ENDIAN = 0x0 Little Endian byte ordering (same as NO_SWAP). BIG_ENDIAN_SWAP = 0x1 Big Endian swap (swap bytes 0,3 and 1,2). SWAP_ALL_BYTES = 0x1 Swizzle all bytes, swap bytes 0,3 and 1,2 (aka Big Endian). HWD_SWAP = 0x2 Swap half-words. HWD_BYTE_SWAP = 0x3 Swap bytes within each half-word.
13:12	**CSC_DATA_SWIZZLE**	RW	0x0	This field specifies how to swap the bytes after the data has been converted into an internal representation of 24 bits per pixel and before it is transmitted over the LCD interface bus. The data is always transmitted with the least significant byte/hword (half word) first after the swizzle takes place. So, INPUT_DATA_SWIZZLE takes place first on the incoming data, and then CSC_DATA_SWIZZLE is applied. The swizzle function is independent of the WORD_LENGTH or the LCD_DATABUS_WIDTH fields. If RGB_TO_YCRCB422_CSC bit is set, the swizzle occurs on the Y, Cb, Cr values. The supported swizzle configurations are: NO_SWAP = 0x0 No byte swapping.(Little endian) LITTLE_ENDIAN = 0x0 Little Endian byte ordering (same as NO_SWAP). BIG_ENDIAN_SWAP = 0x1 Big Endian swap (swap bytes 0,3 and 1,2). SWAP_ALL_BYTES = 0x1 Swizzle all bytes, swap bytes 0,3 and 1,2 (aka Big Endian). HWD_SWAP = 0x2 Swap half-words. HWD_BYTE_SWAP = 0x3 Swap bytes within each half-word.
11:10	**LCD_DATABUS_WIDTH**	RW	0x0	LCD Data bus transfer width. 16_BIT = 0x0 16-bit data bus mode. 8_BIT = 0x1 8-bit data bus mode. 18_BIT = 0x2 18-bit data bus mode. 24_BIT = 0x3 24-bit data bus mode.
9:8	**WORD_LENGTH**	RW	0x0	Input data format. 16_BIT = 0x0 Input data is 16 bits per pixel. Valid BYTE_PACKING_FORMAT settings are 0x3, 0xC and 0xF. H_COUNT must be a multiple of 2 pixels if BYTE_PACKING_FORMAT = 0xF. 8_BIT = 0x1 Input data is 8 bits wide. Any setting in BYTE_PACKING_FORMAT is valid as long as it is non-zero. H_COUNT must be a multiple of sum of the bits of BYTE_PACKING_FORMAT[3:0]. 18_BIT = 0x2 Input data is 18 bits per pixel. Valid BYTE_PACKING_FORMAT setting is 0xF. There are no restrictions on H_COUNT. 24_BIT = 0x3 Input data is 24 bits per pixel. Valid BYTE_PACKING_FORMAT settings are 0x7, 0xE and 0xF. If BYTE_PACKING_FORMAT = 0xF, H_COUNT must be a multiple of 4 pixels, otherwise there are no restrictions.
7	**RGB_TO_YCBCR422_CSC**	RW	0x0	Set this bit to 1 to enable conversion from RGB to YCbCr colorspace. See the HW_LCDIF_CSC_ registers for further details.
6	**RSVD2**	RW	0x0	Program this field to 0x0.

Table 18-4. HW_LCDIF_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
5	**LCDIF_MASTER**	RW	0x0	Set this bit to make the LCDIF act as a bus master. If this bit is reset, the LCDIF will act in PIO mode.
4	**RSVD1**	RW	0x0	Program this field to 0x0.
3	**DATA_FORMAT_16_BIT**	RW	0x0	When this bit is 1 and WORD_LENGTH = 0, it implies that the the 16-bit pixel is in ARGB555 format. In other words, the pixel is right-aligned within 16 bits. When DATA_FORMAT_16_BIT = 0 and WORD_LENGTH = 0, it implies that the 16-bit pixel is in RGB565 format, and the pixel completely fits in 16 bits. When WORD_LENGTH != 0, this bit is a dont care.
2	**DATA_FORMAT_18_BIT**	RW	0x0	Used only when WORD_LENGTH = 2, i.e. 18-bit. Valid BYTE_PACKING_FORMAT setting is 0xF. LOWER_18_BITS_VALID = 0x0 Data input to the block is in 18 bpp format, such that lower 18 bits contain RGB 666 and upper 14 bits do not contain any useful data. In other words, each pixel is right-aligned within 32 bits. UPPER_18_BITS_VALID = 0x1 Data input to the block is in 18 bpp format, such that upper 18 bits contain RGB 666 and lower 14 bits do not contain any useful data. In other words, each pixel is left-aligned within 32 bits.
1	**DATA_FORMAT_24_BIT**	RW	0x0	Used only when WORD_LENGTH = 3, i.e. 24-bit. Note that this applies to both packed and unpacked 24-bit data. Valid BYTE_PACKING_FORMAT settings are 0x7, 0xE and 0xF. ALL_24_BITS_VALID = 0x0 Data input to the block is in 24 bpp format, such that all RGB 888 data is contained in 24 bits. The alignment within 32 bits is determined by BYTE_PACKING_FORMAT field. DROP_UPPER_2_BITS_PER_BYTE = 0x1 Data input to the block is actually RGB 18 bpp, but there is 1 color per byte, hence the upper 2 bits in each byte do not contain any useful data, and should be dropped. Thus, each color component is right aligned, but alignment of each pixel within 32 bits is determined by the BYTE_PACKING_FORMAT field.
0	**RUN**	RW	0x0	When this bit is set by software, the LCDIF will start fetching data in either the PIO mode or the bus master mode and sending it across the interface. This bit must remain set for all the time the block is in operation.

DESCRIPTION:

The LCDIF Control Register provides a variety of control functions to the programmer. These functions allow the interface to be very flexible to work with a variety of LCD controllers, and to minimize overhead and increase performance of LCD programming. The register has been organized such that switching between the different LCD modes can be done with minimum PIO writes.

EXAMPLE:

```
Empty Example.
```

18.4.2 LCDIF General Control1 Register Description

The LCDIF Control Register provides overall control of the LCDIF block.

HW_LCDIF_CTRL1	0x010
HW_LCDIF_CTRL1_SET	0x014
HW_LCDIF_CTRL1_CLR	0x018
HW_LCDIF_CTRL1_TOG	0x01C

Table 18-5. HW_LCDIF_CTRL1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD5					BM_ERROR_IRQ_EN	BM_ERROR_IRQ	RECOVER_ON_UNDERFLOW	INTERLACE_FIELDS	START_INTERLACE_FROM_SECOND_FIELD	FIFO_CLEAR	IRQ_ON_ALTERNATE_FIELDS	BYTE_PACKING_FORMAT				OVERFLOW_IRQ_EN	UNDERFLOW_IRQ_EN	CUR_FRAME_DONE_IRQ_EN	VSYNC_EDGE_IRQ_EN	OVERFLOW_IRQ	UNDERFLOW_IRQ	CUR_FRAME_DONE_IRQ	VSYNC_EDGE_IRQ	RSVD4	RSVD3	RSVD2	RSVD1	LCD_CS_CTRL	BUSY_ENABLE	MODE86	RESET

Table 18-6. HW_LCDIF_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:27	**RSVD5**	RO	0x0	Reserved bits. Write as 0.
26	**BM_ERROR_IRQ_EN**	RW	0x0	This bit is set to enable bus master error interrupt in the LCDIF master mode.
25	**BM_ERROR_IRQ**	RW	0x0	This bit is set to indicate that an interrupt is requested by the LCDIF block. This bit is cleared by software by writing a one to its SCT clear address. This bit will be set when the LCDIF is in master mode and an error response was returned by the slave. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.
24	**RECOVER_ON_UNDERFLOW**	RW	0x0	Set this bit to enable the LCDIF block to recover in the next field/frame if there was an underflow in the current field/frame.
23	**INTERLACE_FIELDS**	RW	0x0	Set this bit if it is required that the LCDIF block fetches odd lines in one field and even lines in the other field. It will work only in LCDIF_MASTER is set to 1.
22	**START_INTERLACE_FROM_SECOND_FIELD**	RW	0x0	The default is to grab the odd lines first and then the even lines. Set this bit if it is required to grab the even lines first and then the odd lines. (Line numbers start from 1, so odd lines are 1,3,5,etc. and even lines are 2,4,6, etc.)
21	**FIFO_CLEAR**	RW	0x0	Set this bit to clear all the data in the latency FIFO (LFIFO), TXFIFO and the RXFIFO.
20	**IRQ_ON_ALTERNATE_FIELDS**	RW	0x0	If this bit is set, the LCDIF block will assert the cur_frame_done interrupt only on alternate fields, otherwise it will issue the interrupt on both odd and even field. This bit is mostly relevant if INTERLACE_FIELDS is set. This feature is only available in DOTCLK and DVI modes.

Table 18-6. HW_LCDIF_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:16	**BYTE_PACKING_FORMAT**	RW	0xf	This bitfield is used to show which data bytes in the 32-bit word written in the HW_LCDIF_DATA register or fetched from memory are valid and should be transmitted. Default value 0xf indicates that all bytes are valid. For 8-bit transfers, any combination in this bitfield will mean valid data is present in the corresponding bytes. In the 16-bit mode, a 16-bit half-word is valid only if the value in this field is 0x3, 0xC or 0xF. In the 18-bit mode, BYTE_PACKING_FORMAT must be 0xF. If the frame buffer data is unpacked 24-bit, set the bit field value to 0x7 (X-R-G-B where X value is invalid and should not be transmitted). If it is packed 24-bit (i.e. 4 pixels in 3 words), set BYTE_PACKING_FORMAT to 0xF. When input data is is in YCbCr 4:2:2 format (YCBCR422_INPUT = 1), the BYTE_PACKING_FORMAT should be 0xF.(Note - YCBCR422_INPUT = 1 implies 2 pixels per 32 bits). BYTE_PACKING_FORMAT = 0 means that none of the bytes are valid and should not be used in any mode.
15	**OVERFLOW_IRQ_EN**	RW	0x0	This bit is set to enable an overflow interrupt in the TXFIFO in the write mode.
14	**UNDERFLOW_IRQ_EN**	RW	0x0	This bit is set to enable an underflow interrupt in the TXFIFO in the write mode.
13	**CUR_FRAME_DONE_IRQ_EN**	RW	0x0	This bit is set to 1 enable an interrupt every time the hardware enters in the vertical blanking state.
12	**VSYNC_EDGE_IRQ_EN**	RW	0x0	This bit is set to enable an interrupt every time the hardware encounters the leading VSYNC edge in the VSYNC and DOTCLK modes, or the beginning of every field in DVI mode.
11	**OVERFLOW_IRQ**	RW	0x0	This bit is set to indicate that an interrupt is requested by the LCDIF block. This bit is cleared by software by writing a one to its SCT clear address. A latency FIFO (LFIFO) overflow in the write mode (system/VSYNC/DOTCLK/DVI mode) was detected, data samples have been lost. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.
10	**UNDERFLOW_IRQ**	RW	0x0	This bit is set to indicate that an interrupt is requested by the LCDIF block. This bit is cleared by software by writing a one to its SCT clear address. A TXFIFO underflow in the write mode (system/VSYNC/DOTCLK/DVI mode) was detected. Could produce an error in the DOTCLK / DVI modes. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.

Table 18-6. HW_LCDIF_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
9	**CUR_FRAME_DONE_IRQ**	RW	0x0	This bit is set to indicate that an interrupt is requested by the LCDIF block. This bit is cleared by software by writing a one to its SCT clear address. It indicates that the hardware has completed transmitting the current frame and is in the vertical blanking period in the DOTCLK/DVI modes. In the VSYNC and system modes, this IRQ is asserted at the end of the data transfer indicated by HW_LCDIF_TRANSFER_COUNT register. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.
8	**VSYNC_EDGE_IRQ**	RW	0x0	This bit is set to indicate that an interrupt is requested by the LCDIF block. This bit is cleared by software by writing a one to its SCT clear address. It is set whenever the leading VSYNC edge is detected in the VSYNC and DOTCLK modes. In the DVI mode, it is asserted every time the block enters a new field. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.
7	**RSVD4**	RO	0x0	Reserved bits. Write as 0.
6	**RSVD3**	RW	0x0	Program this field to 0x0.
5	**RSVD2**	RW	0x0	Program this field to 0x0.
4	**RSVD1**	RW	0x0	Program this field to 0x0. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.
3	**LCD_CS_CTRL**	RW	0x0	If this bit is set to 0 in the VSYNC mode, the LCD_CS pin will toggle according to the system interface protocol. If it set to 0 in the DOTCLK mode, the LCD_CS pin will be driven high throughout the operation. In both the VSYNC and DOTCLK modes, if this bit is set to 1, the LCD_CS pin will be driven low throughout the operation. In the system interface mode, this bit must be set to the default value of 0.
2	**BUSY_ENABLE**	RW	0x0	This bit enables the use of the interface's busy signal input. This should be enabled for LCD controllers that implement a busy line (to stall the LCDIF from sending more data until ready). Otherwise this bit should be cleared. BUSY_DISABLED = 0x0 The busy signal from the LCD controller will be ignored. BUSY_ENABLED = 0x1 Enable the use of the busy signal from the LCD controller.
1	**MODE86**	RW	0x0	This bit is used to select between the 8080 and 6800 series of microprocessor modes. This bit should only be changed when RUN is 0. 8080_MODE = 0x0 Pins LCD_WR_RWn and LCD_RD_E function as active low WR and active low RD signals respectively. 6800_MODE = 0x1 Pins LCD_WR_RWn and LCD_RD_E function as Read/Writeb and active high Enable signals respectively.
0	**RESET**	RW	0x0	Reset bit for the external LCD controller. This bit can be changed at any time. It CANNOT be reset by SFTRST. LCDRESET_LOW = 0x0 LCD_RESET output signal is low. LCDRESET_HIGH = 0x1 LCD_RESET output signal is high.

DESCRIPTION:

The LCDIF Control1 Register provides additional programming to the LCDIF. It implements some bits which are unlikely to change often in a particular application. It also carries interrupt-related bits which are common across more than one mode of operation.

EXAMPLE:

```
Empty Example.
```

18.4.3 LCDIF Horizontal and Vertical Valid Data Count Register Description

This register tells the LCDIF how much data will be sent for this frame, or transaction. The total number of words is a product of the V_COUNT and H_COUNT fields. The word size is specified by the WORD_LENGTH field.

HW_LCDIF_TRANSFER_COUNT 0x020

Table 18-7. HW_LCDIF_TRANSFER_COUNT

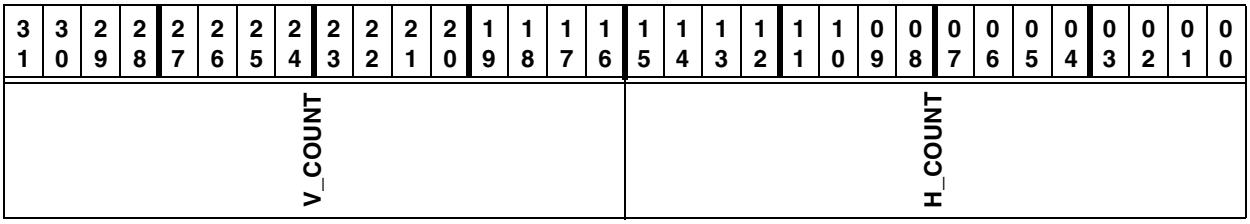


Table 18-8. HW_LCDIF_TRANSFER_COUNT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**V_COUNT**	RW	0x0001	Number of horizontal lines per frame which contain valid data. In DOTCLK mode, V_COUNT should be the same as the number of active horizontal lines in a progressive frame. In DVI mode, V_COUNT should be the number of active horizontal lines per frame, and not per field.
15:0	**H_COUNT**	RW	0x0000	Total valid data (pixels) in each horizontal line. The data size is given by the WORD_LENGTH. When input data is is in YCbCr 4:2:2 format (YCBCR422_INPUT is 1), H_COUNT should be the number of 32-bit words that should be fetched by the block and the BYTE_PACKING_FORMAT should be 0xF. In 24-bit packed format (WORD_LENGTH=0x3, BYTE_PACKING_FORMAT=0xF), the H_COUNT must be a multiple of 4 pixels. In 16-bit packed format (WORD_LENGTH=0x0, BYTE_PACKING_FORMAT=0xF), the H_COUNT must be a multiple of 2 pixels.

DESCRIPTION:

This register gives the dimensions of the input frame. For normal operation, but V_COUNT and H_COUNT should be non-zero.

EXAMPLE:

```
Empty Example.
```

18.4.4 LCD Interface Current Buffer Address Register Description

This register indicates the address of the current frame that is being transmitted by LCDIF.

HW_LCDIF_CUR_BUF 0x030

Table 18-9. HW_LCDIF_CUR_BUF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 18-10. HW_LCDIF_CUR_BUF Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x00	

DESCRIPTION:

When the LCDIF is behaving as a master, this address points to the address of the current frame of data being sent out via the LCDIF. When the current frame is done, the LCDIF block will assert the cur_frame_done interrupt for software to take action. The block will also copy the HW_LCDIF_NEXT_BUF_ADDR into this bitfield so that the software can program the next frame address into the HW_LCDIF_NEXT_BUF_ADDR bitfield.

EXAMPLE:

```
Empty Example.
```

18.4.5 LCD Interface Next Buffer Address Register Description

This register indicates the address of next frame that will be transmitted by LCDIF.

HW_LCDIF_NEXT_BUF 0x040

Table 18-11. HW_LCDIF_NEXT_BUF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 18-12. HW_LCDIF_NEXT_BUF Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x00	

DESCRIPTION:

When the LCDIF is behaving as a master, this address points to the address of the next frame of data that will be sent out via the LCDIF. It is upto the software to make sure that this register is programmed before the end of the current frame, otherwise it might result in old data going out the LCDIF.

EXAMPLE:

```
Empty Example.
```

18.4.6 LCD Interface Timing Register Description

The LCD interface timing register controls the various setup and hold times enforced by the LCD interface in the 6800/8080 system and VSYNC modes of operation.

HW_LCDIF_TIMING 0x060

Table 18-13. HW_LCDIF_TIMING

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_HOLD								CMD_SETUP								DATA_HOLD								DATA_SETUP							

Table 18-14. HW_LCDIF_TIMING Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**CMD_HOLD**	RW	0x00	Number of PIXCLK cycles that the DCn signal is active after CEn is deasserted.
23:16	**CMD_SETUP**	RW	0x00	Number of PIXCLK cycles that the the DCn signal is active before CEn is asserted.
15:8	**DATA_HOLD**	RW	0x00	Data bus hold time in PIXCLK cycles. Also the time that the data strobe is de-asserted in a cycle
7:0	**DATA_SETUP**	RW	0x00	Data bus setup time in PIXCLK cycles. Also the time that the data strobe is asserted in a cycle.

DESCRIPTION:

The values used in this register are dependent on the particular LCD controller used, consult the users manual for the particular controller for required timings. Each field of the register must be non-zero, therefore the minimum value is: 0x01010101. NOTE: the timings are not automatically adjusted if the PIXCLK frequency changes--it may be necessary to adjust the timings if PIXCLK changes. NOTE: Each field in this register must be non-zero for the system and VSYNC modes to function. The settings in this register do not affect the DOTCLK and DVI modes.

EXAMPLE:

```
Empty Example.
```

18.4.7 LCDIF VSYNC Mode and Dotclk Mode Control Register0 Description

This register is used to control the VSYNC and DOTCLK modes of the LCDIF so as to work with different types of LCDs like moving picture displays and delta pixel displays.

HW_LCDIF_VDCTRL0	0x070
HW_LCDIF_VDCTRL0_SET	0x074
HW_LCDIF_VDCTRL0_CLR	0x078
HW_LCDIF_VDCTRL0_TOG	0x07C

Table 18-15. HW_LCDIF_VDCTRL0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2		VSYNC_OEB	ENABLE_PRESENT	VSYNC_POL	HSYNC_POL	DOTCLK_POL	ENABLE_POL	RSRVD1		VSYNC_PERIOD_UNIT	VSYNC_PULSE_WIDTH_UNIT	HALF_LINE	HALF_LINE_MODE	VSYNC_PULSE_WIDTH																	

Table 18-16. HW_LCDIF_VDCTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD2**	RO	0x0	Reserved bits. Write as 0.
29	**VSYNC_OEB**	RW	0x0	0 means the VSYNC signal is an output, 1 means it is an input. Should be set to 0 in the DOTCLK mode. VSYNC_OUTPUT = 0x0 The VSYNC pin is in the output mode and the VSYNC signal has to be generated by the LCDIF block. VSYNC_INPUT = 0x1 The VSYNC pin is in the input mode and the LCD controller sends the VSYNC signal to the block.
28	**ENABLE_PRESENT**	RW	0x0	Setting this bit to 1 will make the hardware generate the ENABLE signal in the DOTCLK mode, thereby making it the true RGB interface along with the remaining three signals VSYNC, HSYNC and DOTCLK.
27	**VSYNC_POL**	RW	0x0	Default 0 active low during VSYNC_PULSE_WIDTH time and will be high during the rest of the VSYNC period. Set it to 1 to invert the polarity.
26	**HSYNC_POL**	RW	0x0	Default 0 active low during HSYNC_PULSE_WIDTH time and will be high during the rest of the HSYNC period. Set it to 1 to invert the polarity.
25	**DOTCLK_POL**	RW	0x0	Default is data launched at negative edge of DOTCLK and captured at positive edge. Set it to 1 to invert the polarity. Set it to 0 in DVI mode.
24	**ENABLE_POL**	RW	0x0	Default 0 active low during valid data transfer on each horizontal line.
23:22	**RSRVD1**	RO	0x0	Reserved bits. Write as 0.

Table 18-16. HW_LCDIF_VDCTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
21	**VSYNC_PERIOD_UNIT**	RW	0x0	Default 0 for counting VSYNC_PERIOD in terms of PIXCLKs. Set it to 1 to count in terms of complete horizontal lines. PIXCLKs should be used in the VSYNC mode, while horizontal line should be used in the DOTCLK mode.
20	**VSYNC_PULSE_WIDTH_UNIT**	RW	0x0	Default 0 for counting VSYNC_PULSE_WIDTH in terms of PIXCLKs. Set it to 1 to count in terms of complete horizontal lines.
19	**HALF_LINE**	RW	0x0	Setting this bit to 1 will make the total VSYNC period equal to the VSYNC_PERIOD field plus half the HORIZONTAL_PERIOD field (i.e. VSYNC_PERIOD field plus half horizontal line), otherwise it is just VSYNC_PERIOD. Should be only used in the DOTCLK mode, not in the VSYNC interface mode.
18	**HALF_LINE_MODE**	RW	0x0	When this bit is 0, the first field (VSYNC period) will end in half a horizontal line and the second field will begin with half a horizontal line. When this bit is 1, all fields will end with half a horizontal line, and none will begin with half a horizontal line.
17:0	**VSYNC_PULSE_WIDTH**	RW	0x00000	Number of units for which VSYNC signal is active. For the DOTCLK mode, the unit is determined by the VSYNC_PULSE_WIDTH_UNIT. If the VSYNC_PULSE_WIDTH_UNIT is 0 for DOTCLK mode, VSYNC_PULSE_WIDTH must be less than HSYNC_PERIOD. For the VSYNC interface mode, it should be in terms of number of PIXCLKs only.

DESCRIPTION:

This register gives general programmability to the VSYNC signal including polarity, direction, pulse width, etc.

EXAMPLE:

```
Empty Example.
```

18.4.8 LCDIF VSYNC Mode and Dotclk Mode Control Register1 Description

This register is used to control the VSYNC signal in the VSYNC and DOTCLK modes of the block.

HW_LCDIF_VDCTRL1 0x080

Table 18-17. HW_LCDIF_VDCTRL1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
VSYNC_PERIOD																															

Table 18-18. HW_LCDIF_VDCTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**VSYNC_PERIOD**	RW	0x000000	Total number of units between two positive or two negative edges of the VSYNC signal. If HALF_LINE is set, it is implicitly calculated to be VSYNC_PERIOD plus half HSYNC_PERIOD.

DESCRIPTION:

This register determines the period and duty cycle of the VSYNC signal when it is generated in the block.

EXAMPLE:

```
Empty Example.
```

18.4.9 LCDIF VSYNC Mode and Dotclk Mode Control Register2 Description

This register is used to control the HSYNC signal in the DOTCLK mode of the block.

HW_LCDIF_VDCTRL2 0x090

Table 18-19. HW_LCDIF_VDCTRL2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
HSYNC_PULSE_WIDTH								RSRVD0								HSYNC_PERIOD															

Table 18-20. HW_LCDIF_VDCTRL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**HSYNC_PULSE_WIDTH**	RW	0x00	Number of PIXCLKs for which HSYNC signal is active.
23:18	**RSRVD0**	RO	0x0	Reserved bits, write as 0.
17:0	**HSYNC_PERIOD**	RW	0x0000	Total number of PIXCLKs between two positive or two negative edges of the HSYNC signal.

DESCRIPTION:

This register determines the period and duty cycle of the HSYNC signal when it is generated in the block.

EXAMPLE:

```
Empty Example.
```

18.4.10 LCDIF VSYNC Mode and Dotclk Mode Control Register3 Description

This register is used to determine the vertical and horizontal wait counts.

HW_LCDIF_VDCTRL3 0x0a0

Table 18-21. HW_LCDIF_VDCTRL3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD0		MUX_SYNC_SIGNALS	VSYNC_ONLY	HORIZONTAL_WAIT_CNT												VERTICAL_WAIT_CNT																

Table 18-22. HW_LCDIF_VDCTRL3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD0**	RO	0x0	Reserved bits, write as 0.
29	**MUX_SYNC_SIGNALS**	RW	0x0	When this bit is set, the LCDIF block will internally mux HSYNC with LCD_D14, DOTCLK with LCD_D13 and ENABLE with LCD_D12, otherwise these signals will go out on separate pins. This feature can be used to maintain backward compatability with older generation SoCs.
28	**VSYNC_ONLY**	RW	0x0	This bit must be set to 1 in the VSYNC mode of operation, and 0 in the DOTCLK mode of operation.
27:16	**HORIZONTAL_WAIT_CNT**	RW	0x000	In the DOTCLK mode, wait for this number of clocks from falling edge (or rising if HSYNC_POL is 1) of HSYNC signal to account for horizontal back porch plus the number of DOTCLKs before the moving picture information begins.
15:0	**VERTICAL_WAIT_CNT**	RW	0x000	In the VSYNC interface mode, wait for this number of PIXCLKs from the falling VSYNC edge (or rising if VSYNC_POL is 1) before starting LCD transactions and is applicable only if WAIT_FOR_VSYNC_EDGE is set. Minimum is CMD_SETUP+5. In the DOTCLK mode, it accounts for the veritcal back porch lines plus the number of horizontal lines before the moving picture begins. The unit for this parameter is inherently the same as the VSYNC_PERIOD_UNIT.

DESCRIPTION:

This register determines the back porches of HSYNC and VSYNC signals when they are generated by the block.

EXAMPLE:

```
Empty Example.
```

18.4.11 LCDIF VSYNC Mode and Dotclk Mode Control Register4 Description

This register is used to control the DOTCLK mode of the block.

HW_LCDIF_VDCTRL4 0x0b0

Table 18-23. HW_LCDIF_VDCTRL4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD0													SYNC_SIGNALS_ON	DOTCLK_H_VALID_DATA_CNT																			

Table 18-24. HW_LCDIF_VDCTRL4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:19	**RSRVD0**	RO	0x0000	Reserved bits, write as 0.
18	**SYNC_SIGNALS_ON**	RW	0x0	Set this field to 1 if the LCD controller requires that the VSYNC or VSYNC/HSYNC/DOTCLK control signals should be active atleast one frame before the data transfers actually start and remain active atleast one frame after the data transfers end. The hardware does not count the number of frames automatically. Rather, the VSYNC edge interrupt can be monitored by software to count the number of frames that have occured after this bit is set and then the RUN bit can be set to start the data transactions. This bit must always be set in the DOTCLK mode of operation, and it must be set in the VSYNC mode of operation when VSYNC signal is an output.
17:0	**DOTCLK_H_VALID_DATA_CNT**	RW	0x00000	Total number of PIXCLKs on each horizontal line that carry valid data in DOTCLK mode.

DESCRIPTION:

This register determines the active data in each horizontal line in the DOTCLK mode. Note that the total number of active horizontal lines in the DOTCLK mode is the same as the V_COUNT bitfield in the HW_LCDIF_TRANSFER_COUNT register.

EXAMPLE:

```
Empty Example.
```

18.4.12 Digital Video Interface Control0 Register Description

The Digital Video interface Control0 register provides the overall control of the Digital Video interface.

HW_LCDIF_DVICTRL0 0x0c0

Table 18-25. HW_LCDIF_DVICTRL0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1	H_ACTIVE_CNT											H_BLANKING_CNT										V_LINES_CNT									

Table 18-26. HW_LCDIF_DVICTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSVD1**	RW	0x0	Program this field to 0x0.
30:20	**H_ACTIVE_CNT**	RW	0x000	Number of active video samples to be transmitted. (Mostly will be 1440 for both PAL and NTSC). Must always be a multiple of 4.
19:10	**H_BLANKING_CNT**	RW	0x000	Number of blanking samples to be inserted between EAV and SAV during horizontal blanking interval.
9:0	**V_LINES_CNT**	RW	0x000	Total number of vertical lines per frame (generally 525 or 625)

DESCRIPTION:

This register gives information about the horizontal active, horizontal blanking and total number of lines in the ITU-R BT.656 interface.

EXAMPLE:

```
//525/60 video system
HW_LCDIF_DVICTRL0_H_ACTIVE_CNT_WR(0x5A0);//1440
HW_LCDIF_DVICTRL0_H_BLANKING_CNT_WR(0x106);//262
HW_LCDIF_DVICTRL0_V_LINES_CNT_WR(0x20D);//525
//625/50 video system
HW_LCDIF_DVICTRL0_H_ACTIVE_CNT_WR(0x5A0);//1440
HW_LCDIF_DVICTRL0_H_BLANKING_CNT_WR(0x112);//274
HW_LCDIF_DVICTRL0_V_LINES_CNT_WR(0x271);//625
```

18.4.13 Digital Video Interface Control1 Register Description

The Digital Video interface Control1 register provides the overall control of the Digital Video interface.

HW_LCDIF_DVICTRL1 0x0d0

Table 18-27. HW_LCDIF_DVICTRL1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD0		F1_START_LINE										F1_END_LINE										F2_START_LINE									

Table 18-28. HW_LCDIF_DVICTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD0**	RO	0x0	Reserved bits, write as 0.
29:20	**F1_START_LINE**	RW	0x000	Vertical line number from which Field 1 begins.
19:10	**F1_END_LINE**	RW	0x000	Vertical line number at which Field1 ends.
9:0	**F2_START_LINE**	RW	0x000	Vertical line number from which Field 2 begins.

DESCRIPTION:

This register contains information about the Field1 start and end, and the Field2 start in the ITU-R BT.656 interface.

EXAMPLE:

```
//525/60 video system
HW_LCDIF_DVICTRL1_F1_START_LINE_WR(0x4);//4
HW_LCDIF_DVICTRL1_F1_END_LINE_WR(0x109);//265
HW_LCDIF_DVICTRL1_F2_START_LINE_WR(0x10A);//266
//625/50 video system
HW_LCDIF_DVICTRL1_F1_START_LINE_WR(0x1);//1
HW_LCDIF_DVICTRL1_F1_END_LINE_WR(0x138);//312
HW_LCDIF_DVICTRL1_F2_START_LINE_WR(0x139);//313
```

18.4.14 Digital Video Interface Control2 Register Description

The Digital Video interface Control2 register provides the overall control of the Digital Video interface.

HW_LCDIF_DVICTRL2 0x0e0

Table 18-29. HW_LCDIF_DVICTRL2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD0		F2_END_LINE										V1_BLANK_START_LINE										V1_BLANK_END_LINE									

Table 18-30. HW_LCDIF_DVICTRL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD0**	RO	0x0	Reserved bits, write as 0.
29:20	**F2_END_LINE**	RW	0x000	Vertical line number at which Field 2 ends.
19:10	**V1_BLANK_START_LINE**	RW	0x000	Vertical line number towards the end of Field1 where first Vertical Blanking interval starts.
9:0	**V1_BLANK_END_LINE**	RW	0x000	Vertical line number in the beginning part of Field2 where first Vertical Blanking interval ends.

DESCRIPTION:

This register contains information about the Field2 end, and the Vertical Blanking1 interval in the ITU-R BT.656 interface.

EXAMPLE:

```
//525/60 video system
HW_LCDIF_DVICTRL2_F2_END_LINE_WR(0x3);//3
HW_LCDIF_DVICTRL2_V1_BLANK_START_LINE_WR(0x108);//264
HW_LCDIF_DVICTRL2_V1_BLANK_END_LINE_WR(0x11A);//282
//625/50 video system
HW_LCDIF_DVICTRL2_F2_END_LINE_WR(0x271);//625
HW_LCDIF_DVICTRL2_V1_BLANK_START_LINE_WR(0x137);//311
HW_LCDIF_DVICTRL2_V1_BLANK_END_LINE_WR(0x14F);//335
```

18.4.15 Digital Video Interface Control3 Register Description

The Digital Video interface Control3 register provides the overall control of the Digital Video interface.

HW_LCDIF_DVICTRL3 0x0f0

Table 18-31. HW_LCDIF_DVICTRL3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1						V2_BLANK_START_LINE										RSRVD0						V2_BLANK_END_LINE									

Table 18-32. HW_LCDIF_DVICTRL3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSRVD1**	RO	0x0	Reserved bits, write as 0.
25:16	**V2_BLANK_START_LINE**	RW	0x000	Vertical line number towards the end of Field2 where second Vertical Blanking interval starts.
15:10	**RSRVD0**	RO	0x00	Reserved bits, write as 0.
9:0	**V2_BLANK_END_LINE**	RW	0x000	Vertical line number in the beginning part of Field1 where second Vertical Blanking interval ends.

DESCRIPTION:

This register contains information about the Vertical Blanking2 interval in the ITU-R BT.656 interface.

EXAMPLE:

```
//525/60 video system
HW_LCDIF_DVICTRL3_V2_BLANK_START_LINE_WR(0x1);//1
HW_LCDIF_DVICTRL3_V2_BLANK_END_LINE_WR(0x13);//19
//625/50 video system
HW_LCDIF_DVICTRL3_V2_BLANK_START_LINE_WR(0x270);//624
HW_LCDIF_DVICTRL3_V2_BLANK_END_LINE_WR(0x16);//22
```

18.4.16 Digital Video Interface Control4 Register Description

The Digital Video interface Control4 register provides the overall control of the Digital Video interface.

HW_LCDIF_DVICTRL4 0x100

Table 18-33. HW_LCDIF_DVICTRL4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Y_FILL_VALUE								CB_FILL_VALUE								CR_FILL_VALUE								H_FILL_CNT							

Table 18-34. HW_LCDIF_DVICTRL4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**Y_FILL_VALUE**	RW	0x00	Value of Y component of filler data
23:16	**CB_FILL_VALUE**	RW	0x00	Value of CB component of filler data
15:8	**CR_FILL_VALUE**	RW	0x00	Value of CR component of filler data.
7:0	**H_FILL_CNT**	RW	0x00	Number of active video samples that have to be filled with the filler data in the front and back portions of the active horizontal interval. Must be a multiple of 4. This field will have to be programmed if the input frame has less than 720 pixels per line.

DESCRIPTION:

This register is used to add side borders to the output if the input frame width is less than 720 pixels.

EXAMPLE:

```
//If input frame has only 640 pixels per line, but output is supposed to have 720 pixels per line.
HW_LCDIF_DVICTRL4_H_FILL_CNT_WR(0x50);//80
HW_LCDIF_DVICTRL4_Y_FILL_VALUE_WR(0x10);//16
HW_LCDIF_DVICTRL4_CB_FILL_VALUE_WR(0x80);//128
HW_LCDIF_DVICTRL4_CR_FILL_VALUE_WR(0x80);//128
```

18.4.17 RGB to YCbCr 4:2:2 CSC Coefficient0 Register Description

HW_LCDIF_CSC_COEFF0 register provides overall control over color space conversion from RGB to 4:2:2 YCbCr. The equations for the conversion are given by: Y = C0*R + C1*G + C2*B + Y_offset Cb= C3*R + C4*G + C5*B + CbCr_offset Cr= C6*R + C7*G + C8*B + CbCr_offset

HW_LCDIF_CSC_COEFF0 0x110

Table 18-35. HW_LCDIF_CSC_COEFF0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1						C0										RSRVD0														CSC_SUBSAMPLE_FILTER	

Table 18-36. HW_LCDIF_CSC_COEFF0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSRVD1**	RO	0x00	Reserved bits, write as 0.
25:16	**C0**	RW	0x000	Two's complement red multiplier coefficient for Y
15:2	**RSRVD0**	RO	0x000	Reserved bits, write as 0.
1:0	**CSC_SUBSAMPLE_FILTER**	RW	0x0	This register describes the filtering and subsampling scheme to be performed on the chroma components in order to convert from YCbCr 4:4:4 to YCbCr 4:2:2 space. Note that the following descriptions apply individually to Cb and Cr. SAMPLE_AND_HOLD = 0x0 No filtering, simply keep every chroma value for samples numbered 2n and discard chroma values associated with all samples numbered 2n+1. RSRVD = 0x1 Reserved INTERSTITIAL = 0x2 Chroma samples numbered 2n and 2n+1 are averaged (weights 1/2, 1/2) and that chroma value replaces the two chroma values at 2n and 2n+1. This chroma now exists horizontally halfway between the two luma samples. COSITED = 0x3 Chroma samples numbered 2n-1, 2n, and 2n+1 are averaged (weights 1/4,1/2,1/4) and that chroma value exists at the same site as the luma sample numbered 2n and the chroma samples at 2n+1 are discarded.

DESCRIPTION:

This register carries programming information about RGB to YCbCr 4:2:2 CSC.

EXAMPLE:

```
HW_LCDIF_CSC_COEFF0_C0_WR(0x41);//0.257x256=65
HW_LCDIF_CSC_COEFF0_CSC_SUBSAMPLE_FILTER_WR(0x3);
```

18.4.18 RGB to YCbCr 4:2:2 CSC Coefficient1 Register Description

HW_LCDIF_CSC_COEFF1 register provides overall control over color space conversion from RGB to 4:2:2 YCbCr. The equations for the conversion are given by: Y = C0*R + C1*G + C2*B + Y_offset Cb= C3*R + C4*G + C5*B + CbCr_offset Cr= C6*R + C7*G + C8*B + CbCr_offset

HW_LCDIF_CSC_COEFF1 0x120

Table 18-37. HW_LCDIF_CSC_COEFF1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1						C2										RSRVD0				C1												

Table 18-38. HW_LCDIF_CSC_COEFF1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSRVD1**	RO	0x00	Reserved bits, write as 0.
25:16	**C2**	RW	0x000	Two's complement blue multiplier coefficient for Y
15:10	**RSRVD0**	RO	0x00	Reserved bits, write as 0.
9:0	**C1**	RW	0x000	Two's complement green multiplier coefficient for Y

DESCRIPTION:

This register carries programming information about RGB to YCbCr 4:2:2 CSC.

EXAMPLE:

```
HW_LCDIF_CSC_COEFF1_C1_WR(0x81);//0.504x256=129
HW_LCDIF_CSC_COEFF1_C2_WR(0x19);//0.098x256=25
```

18.4.19 RGB to YCbCr 4:2:2 CSC Coefficent2 Register Description

HW_LCDIF_CSC_COEFF2 register provides overall control over color space conversion from RGB to 4:2:2 YCbCr. The equations for the conversion are given by: Y = C0*R + C1*G + C2*B + Y_offset Cb= C3*R + C4*G + C5*B + CbCr_offset Cr= C6*R + C7*G + C8*B + CbCr_offset

HW_LCDIF_CSC_COEFF2 0x130

Table 18-39. HW_LCDIF_CSC_COEFF2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1						C4										RSRVD0				C3												

Table 18-40. HW_LCDIF_CSC_COEFF2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSRVD1**	RO	0x00	Reserved bits, write as 0.
25:16	**C4**	RW	0x000	Two's complement green multiplier coefficient for Cb
15:10	**RSRVD0**	RO	0x00	Reserved bits, write as 0.
9:0	**C3**	RW	0x000	Two's complement red multiplier coefficient for Cb

DESCRIPTION:

This register carries programming information about RGB to YCbCr 4:2:2 CSC.

EXAMPLE:

```
HW_LCDIF_CSC_COEFF2_C3_WR(0x3DB);//-0.148x256=-37
HW_LCDIF_CSC_COEFF2_C4_WR(0x3B6);//-0.291x256=-74
```

18.4.20 RGB to YCbCr 4:2:2 CSC Coefficient3 Register Description

HW_LCDIF_CSC_COEFF3 register provides overall control over color space conversion from RGB to 4:2:2 YCbCr. The equations for the conversion are given by: Y = C0*R + C1*G + C2*B + Y_offset Cb= C3*R + C4*G + C5*B + CbCr_offset Cr= C6*R + C7*G + C8*B + CbCr_offset

HW_LCDIF_CSC_COEFF3 0x140

Table 18-41. HW_LCDIF_CSC_COEFF3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1						C6										RSRVD0						C5									

Table 18-42. HW_LCDIF_CSC_COEFF3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSRVD1**	RO	0x00	Reserved bits, write as 0.
25:16	**C6**	RW	0x000	Two's complement red multiplier coefficient for Cr
15:10	**RSRVD0**	RO	0x00	Reserved bits, write as 0.
9:0	**C5**	RW	0x000	Two's complement blue multiplier coefficient for Cb

DESCRIPTION:

This register carries programming information about RGB to YCbCr 4:2:2 CSC.

EXAMPLE:

```
HW_LCDIF_CSC_COEFF3_C5_WR(0x70);//0.439x256=112
HW_LCDIF_CSC_COEFF3_C6_WR(0x70);//0.439x256=112
```

18.4.21 RGB to YCbCr 4:2:2 CSC Coefficient4 Register Description

HW_LCDIF_CSC_COEFF4 register provides overall control over color space conversion from RGB to 4:2:2 YCbCr. The equations for the conversion are given by: Y = C0*R + C1*G + C2*B + Y_offset Cb= C3*R + C4*G + C5*B + CbCr_offset Cr= C6*R + C7*G + C8*B + CbCr_offset

HW_LCDIF_CSC_COEFF4 0x150

Table 18-43. HW_LCDIF_CSC_COEFF4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1						C8										RSRVD0				C7											

Table 18-44. HW_LCDIF_CSC_COEFF4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSRVD1**	RO	0x00	Reserved bits, write as 0.
25:16	**C8**	RW	0x000	Two's complement blue multiplier coefficient for Cr
15:10	**RSRVD0**	RO	0x00	Reserved bits, write as 0.
9:0	**C7**	RW	0x000	Two's complement green multiplier coefficient for Cr

DESCRIPTION:

This register carries programming information about RGB to YCbCr 4:2:2 CSC.

EXAMPLE:

```
HW_LCDIF_CSC_COEFF4_C7_WR(0x3A2);//-0.368x256=-94
HW_LCDIF_CSC_COEFF4_C8_WR(0x3EE);//-0.071x256=-18
```

18.4.22 RGB to YCbCr 4:2:2 CSC Offset Register Description

HW_LCDIF_CSC_ register provides overall control over color space conversion from RGB to 4:2:2 YCbCr. The equations for the conversion are given by: Y = C0*R + C1*G + C2*B + Y_offset Cb= C3*R + C4*G + C5*B + CbCr_offset Cr= C6*R + C7*G + C8*B + CbCr_offset

HW_LCDIF_CSC_OFFSET 0x160

Table 18-45. HW_LCDIF_CSC_OFFSET

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1							CBCR_OFFSET									RSRVD0				Y_OFFSET											

Table 18-46. HW_LCDIF_CSC_OFFSET Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSRVD1**	RO	0x00	Reserved bits, write as 0.
24:16	**CBCR_OFFSET**	RW	0x080	Two's complement offset for the Cb and Cr components
15:9	**RSRVD0**	RO	0x00	Reserved bits, write as 0.
8:0	**Y_OFFSET**	RW	0x010	Two's complement offset for the Y component

DESCRIPTION:

This register carries programming information about RGB to YCbCr 4:2:2 CSC.

EXAMPLE:

```
Empty Example.
```

18.4.23 RGB to YCbCr 4:2:2 CSC Limit Register Description

HW_LCDIF_CSC_CTRL0 register provides overall control over color space conversion from RGB to 4:2:2 YCbCr. The equations for the conversion are given by: Y = C0*R + C1*G + C2*B + Y_offset Cb= C3*R + C4*G + C5*B + CbCr_offset Cr= C6*R + C7*G + C8*B + CbCr_offset

HW_LCDIF_CSC_LIMIT 0x170

Table 18-47. HW_LCDIF_CSC_LIMIT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CBCR_MIN								CBCR_MAX								Y_MIN								Y_MAX							

Table 18-48. HW_LCDIF_CSC_LIMIT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**CBCR_MIN**	RW	0x00	Lower limit of Cb and Cr after RGB to 4:2:2 YCbCr conversion
23:16	**CBCR_MAX**	RW	0xff	Upper limit of Cb and Cr after RGB to 4:2:2 YCbCr conversion
15:8	**Y_MIN**	RW	0x00	Lower limit of Y after RGB to 4:2:2 YCbCr conversion
7:0	**Y_MAX**	RW	0xff	Upper limit of Y after RGB to 4:2:2 YCbCr conversion

DESCRIPTION:

This register carries programming information about RGB to YCbCr 4:2:2 CSC. Note that the values in this register are unsigned.

EXAMPLE:

```
HW_LCDIF_CSC_LIMIT_CBCR_MIN_WR(0x10);//16
HW_LCDIF_CSC_LIMIT_CBCR_MAX_WR(0xF0);//240
HW_LCDIF_CSC_LIMIT_Y_MIN_WR(0x10);//16
HW_LCDIF_CSC_LIMIT_Y_MAX_WR(0xEB);//235
```

18.4.24 LCD Interface Data Register Description

The data sent to an external LCD controller is written to this register. Data can be written to this register (from the processor's perspective) as bytes half-words (16 bits) or words (32 bits) as appropriate.

HW_LCDIF_DATA 0x1b0

Table 18-49. HW_LCDIF_DATA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DATA_THREE								DATA_TWO								DATA_ONE								DATA_ZERO							

Table 18-50. HW_LCDIF_DATA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**DATA_THREE**	RW	0x00	Byte 3 (most significant byte) of data written to LCDIF by the CPU.
23:16	**DATA_TWO**	RW	0x00	Byte 2 of data written to LCDIF by the CPU.
15:8	**DATA_ONE**	RW	0x00	Byte 1 of data written to LCDIF by the CPU.
7:0	**DATA_ZERO**	RW	0x00	Byte 0 (least significant byte) of data written to LCDIF by the CPU.

DESCRIPTION:

This register holds the 32-bit word written by the CPU into LCDIF. This data then gets sent out by the block across the interface. This register plays no role in the bus master mode of operation.

EXAMPLE:

```
Empty Example.
```

18.4.25 Bus Master Error Status Register Description

This register reflects the virtual address at which the AXI master received an error response from the slave.

HW_LCDIF_BM_ERROR_STAT 0x1c0

Table 18-51. HW_LCDIF_BM_ERROR_STAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 18-52. HW_LCDIF_BM_ERROR_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RW	0x00000000	Virtual address at which bus master error occurred.

DESCRIPTION:

When the BM_ERROR_IRQ is asserted, the address of the bus error is updated in the register.

EXAMPLE:

```
Empty Example.
```

18.4.26 LCD Interface Status Register Description

The LCD interface status register can be used to check the current status of the LCDIF block.

HW_LCDIF_STAT 0x1d0

Table 18-53. HW_LCDIF_STAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PRESENT	DMA_REQ	LFIFO_FULL	LFIFO_EMPTY	TXFIFO_FULL	TXFIFO_EMPTY	BUSY	DVI_CURRENT_FIELD	RSRVD0																							

Table 18-54. HW_LCDIF_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**PRESENT**	RO	0x1	0: LCDIF not present on this product 1: LCDIF is present.
30	**DMA_REQ**	RO	0x0	Reflects the current state of the DMA Request line for the LCDIF. The DMA Request line toggles for each new request.
29	**LFIFO_FULL**	RO	0x0	Read only view of the signal that indicates that LCD read datapath FIFO is full, will be generally used in the write mode of the LCD interface.
28	**LFIFO_EMPTY**	RO	0x1	Read only view of the signal that indicates that LCD read dapatath FIFO is empty, will be generally used in the read mode of the LCD interface.
27	**TXFIFO_FULL**	RO	0x0	Read only view of the signal that indicates that LCD write datapath FIFO is full, will be generally used in the write mode of the LCD interface.
26	**TXFIFO_EMPTY**	RO	0x0	Read only view of the signal that indicates that LCD write dapatath FIFO is empty, will be generally used in the read mode of the LCD interface.
25	**BUSY**	RO	0x0	Read only view of the input busy signal from the external LCD controller.
24	**DVI_CURRENT_FIELD**	RO	0x0	Read only view of the current field being transmitted. DVI_CURRENT_FIELD = 0 means field 1. DVI_CURRENT_FIELD = 1 means field 2.
23:0	**RSRVD0**	RO	0x0	Reserved bits. Write as 0.

DESCRIPTION:

The LCD interface status register that contains read only views of some parameters or current state of the block.

EXAMPLE:

```
Empty Example.
```

18.4.27 LCD Interface Version Register Description

The LCD interface version register can be used to read the version of the LCDIF IP being used in this SoC.

HW_LCDIF_VERSION 0x1e0

Table 18-55. HW_LCDIF_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 18-56. HW_LCDIF_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x3	Fixed read-only value reflecting the MAJOR field of RTL version.
23:16	**MINOR**	RO	0x0	Fixed read-only value reflecting the MINOR field of RTL version.
15:0	**STEP**	RO	0x0	Fixed read-only value reflecting the stepping of RTL version.

DESCRIPTION:

The LCD interface debug register is for diagnostic use only.

EXAMPLE:

```
Empty Example.
```

18.4.28 LCD Interface Debug0 Register Description

The LCD interface debug0 register provides a diagnostic view of the state machine and other useful internal signals.

HW_LCDIF_DEBUG0 0x1f0

Table 18-57. HW_LCDIF_DEBUG0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
STREAMING_END_DETECTED	WAIT_FOR_VSYNC_EDGE_OUT	SYNC_SIGNALS_ON_REG	RSVD8	ENABLE	HSYNC	VSYNC	CUR_FRAME_TX	EMPTY_WORD	CUR_STATE							RSVD7	RSVD6	RSVD5	RSVD4	RSVD3	RSVD2	RSVD1									

Table 18-58. HW_LCDIF_DEBUG0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**STREAMING_END_DETECTED**	RO	0x0	Read only view of the DOTCLK_MODE or DVI_MODE bit going from 1 to 0.
30	**WAIT_FOR_VSYNC_EDGE_OUT**	RO	0x0	Read only view of WAIT_FOR_VSYNC_EDGE bit in the VSYNC mode after it comes out of the TXFIFO.
29	**SYNC_SIGNALS_ON_REG**	RO	0x0	Read only view of internal sync_signals_on_reg signal.
28	**RSVD8**	RO	0x0	Program this field to 0x0.
27	**ENABLE**	RO	0x1	Read only view of ENABLE signal.
26	**HSYNC**	RO	0x1	Read only view of HSYNC signal.
25	**VSYNC**	RO	0x1	Read only view of VSYNC signal.
24	**CUR_FRAME_TX**	RO	0x0	This bit is 1 for the time the current frame is being transmitted in the VSYNC mode. Useful for VSYNC mode debug.
23	**EMPTY_WORD**	RO	0x1	Indicates that the current word is empty.
22:16	**CUR_STATE**	RO	0x01	Read only view of the current state machine state in the current mode of operation.
15	**RSVD7**	RO	0x0	Program this field to 0x0.
14	**RSVD6**	RO	0x0	Program this field to 0x0.
13	**RSVD5**	RO	0x0	Program this field to 0x0.
12	**RSVD4**	RO	0x0	Program this field to 0x0.
11	**RSVD3**	RO	0x0	Program this field to 0x0.
10	**RSVD2**	RO	0x1	Program this field to 0x1.
9:0	**RSVD1**	RO	0x0	Reserved bits. Write as 0.

DESCRIPTION:

The LCD interface debug register is for diagnostic use only.

EXAMPLE:

```
Empty Example.
```

18.4.29 LCD Interface Debug1 Register Description

The LCD interface debug1 register provides a diagnostic view of the state machine and other useful internal signals.

HW_LCDIF_DEBUG1 0x200

Table 18-59. HW_LCDIF_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
H_DATA_COUNT																V_DATA_COUNT															

Table 18-60. HW_LCDIF_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**H_DATA_COUNT**	RO	0x0000	Read only view of the current state of the horizontal data counter.
15:0	**V_DATA_COUNT**	RO	0x0000	Read only view of the current state of the vertical data counter.

DESCRIPTION:

The LCD interface debug register is for diagnostic use only.

EXAMPLE:

```
Empty Example.
```

LCDIF Block v3.0, Revision 1.32

Chapter 19
TV-Out NTSC/PAL Encoder

19.1 Implementation

The TV encoder (TVENC) is a digital video encoder that takes its input from LCDIF in the ITU-R BT.656 format and sends its output to the Video DAC (VDAC). The VDAC in turn gets connected to the video jack of the device and the output can be displayed on standard definition television. The DVE block from Sarnoff is the main functional block of the TV encoder. The DVE block generates the output data that is sent to an asynchronous FIFO and then sent to the pix_int block. Finally, pix_int sends the video data to the VDAC. Figure 19-1 contains a brief diagram showing this data flow.

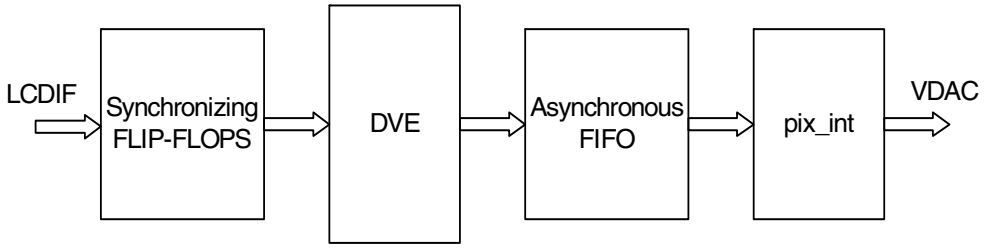


Figure 19-1. TV Encoder Block Diagram

The data that comes from LCDIF runs on pix_clk, which is asynchronous to the tv27m_clk that runs the TV encoder. The pix_int receives input from the asynchronous FIFO using vdac_clk, which is the clock that runs Video DAC.

Tha major component of the TVENC block is the Digital Video Encoder (DVE), which is used under license from the Sarnoff Corporation. The DVE design specification from Sarnoff is the main documentation about how the encoder works and it has all the information available about the IP from the vendor. (Please refer to Appendix A at the end of this document for DVE specification.) One thing to note about this documentation is that the register map of TVENC used on i.MX23 has been considerably changed from the original Sarnoff register map. The register settings have been re-arranged and more control bits have been added to the wrapper. The user should refer to the register definitions in Section 19.4, "Programmable Registers" of this chapter for the correct address and bit positions of the programmable registers.

The PIX_INT block interpolates the Composite video samples to a higher rate to reduce the requirements of off-chip video filtering. The Sarnoff IP block outputs the composite video pixel stream at 27M samples/sec. The PIX_INT interpolates this to 108 MS/sec. There are no official specifications for the spectral characteristics of the composite video signal. The PIX_INT block and the interpolation done in the

Sarnoff block combined have passband droop better than 0.4 dB from 0 to 5 MHz and stopband attenuation better than 40 dB. The off-chip filter required is an RC filter with a cut-off frequency of 10 MHz.

19.2 Unsupported DVE features

The DVE from Sarnoff supports component, S-video, and composite video outputs. However, only the composite video mode is supported on i.MX23. Accordingly, logic functions and registers that are only intended for S-video and component video functionalities have been removed. This is a brief summary of features that have been changed in the original DVE engine:

- Component and S-video support removed.
- rgbmatrix module and its corresponding control registers removed.
- Only one DAC support is available, instead of four.
- Only supported input data format is single 8-bit port clocked at 27 MHz in CbYCrY order.
- Register map has been significantly modified, so please refer to the register map at the end of this chapter.

19.3 Programming Example

The following are example register settings that can be used to set up the LCDIF, CLKCTRL, TVENC and VDAC blocks for NTSC and PAL systems. Please note that these are just for reference, and some parameters might have to be tweaked by the programmer if needed.

```
//LCDIF PIXCLK setup
HW_CLKCTRL_FRAC.B.PIXFRAC = 32;//270 MHz ref pix
HW_CLKCTRL_FRAC_CLR(BM_CLKCTRL_FRAC_CLKGATEPIX);
// now set PIX clk to use ref_pix
HW_CLKCTRL_CLKSEQ_CLR(BM_CLKCTRL_CLKSEQ_BYPASS_PIX);
HW_CLKCTRL_PIX.B.DIV_FRAC_EN = 0;
HW_CLKCTRL_PIX.B.DIV = 10; //27  MHz
HW_CLKCTRL_PIX_CLR(BM_CLKCTRL_PIX_CLKGATE);

//Setup LCDIF registers (HW_LCDIF_*) according to NTSC or PAL settings.
#if NTSC_FORMAT
        // NTSC is 525/60
        HW_LCDIF_DVICTRL0.B.H_ACTIVE_CNT = 1440;
        HW_LCDIF_DVICTRL0.B.H_BLANKING_CNT = 262;
        HW_LCDIF_DVICTRL0.B.V_LINES_CNT = 525;

        HW_LCDIF_DVICTRL1.B.F1_START_LINE = 4;
        HW_LCDIF_DVICTRL1.B.F1_END_LINE = 265;
        HW_LCDIF_DVICTRL1.B.F2_START_LINE = 266;

        HW_LCDIF_DVICTRL2.B.F2_END_LINE = 3;
        HW_LCDIF_DVICTRL2.B.V1_BLANK_START_LINE = 263;
        HW_LCDIF_DVICTRL2.B.V1_BLANK_END_LINE = 285;

        HW_LCDIF_DVICTRL3.B.V2_BLANK_START_LINE = 1;
        HW_LCDIF_DVICTRL3.B.V2_BLANK_END_LINE = 22;
#else
       // PAL is 625/50
```

```
        HW_LCDIF_DVICTRL0.B.H_ACTIVE_CNT = 1440;
        HW_LCDIF_DVICTRL0.B.H_BLANKING_CNT = 274;
        HW_LCDIF_DVICTRL0.B.V_LINES_CNT = 625;

        HW_LCDIF_DVICTRL1.B.F1_START_LINE = 1;
        HW_LCDIF_DVICTRL1.B.F1_END_LINE = 312;
        HW_LCDIF_DVICTRL1.B.F2_START_LINE = 313;

        HW_LCDIF_DVICTRL2.B.F2_END_LINE = 625;
        HW_LCDIF_DVICTRL2.B.V1_BLANK_START_LINE = 311;
        HW_LCDIF_DVICTRL2.B.V1_BLANK_END_LINE = 335;
        HW_LCDIF_DVICTRL3.B.V2_BLANK_START_LINE = 624;
        HW_LCDIF_DVICTRL3.B.V2_BLANK_END_LINE = 22;
#endif

    BW_LCDIF_CSC_COEFF0_C0(0x41);//0.257X256 = 65
    BW_LCDIF_CSC_COEFF0_CSC_SUBSAMPLE_FILTER(2);//co-sited

    BW_LCDIF_CSC_COEFF1_C1(0x81);//0.504x256 = 129
    BW_LCDIF_CSC_COEFF1_C2(0x19);//0.098x256 = 25

    BW_LCDIF_CSC_COEFF2_C3(0x3DB);//-0.148x256 = -37
    BW_LCDIF_CSC_COEFF2_C4(0x3B6);//-0.291x256 = -74

    BW_LCDIF_CSC_COEFF3_C5(0x70);//0.439x256 = 112
    BW_LCDIF_CSC_COEFF3_C6(0x70);//0.439x256 = 112

    BW_LCDIF_CSC_COEFF4_C7(0x3A2);//-0.368x256 = -94
    BW_LCDIF_CSC_COEFF4_C8(0x3EE);//-0.071x256 = -18

    BW_LCDIF_CSC_OFFSET_CBCR_OFFSET(128);
    BW_LCDIF_CSC_OFFSET_Y_OFFSET(16);

    //limiting values to be applied in both YCBCR input and CSC
    BW_LCDIF_CSC_LIMIT_CBCR_MIN (16);
    BW_LCDIF_CSC_LIMIT_CBCR_MAX (240);
    BW_LCDIF_CSC_LIMIT_Y_MIN (16);
    BW_LCDIF_CSC_LIMIT_Y_MAX (235);

//Setup TVENC block for NTSC/PAL mode of operation. The registers which are not mentioned
here maintain their reset values.
#if NTSC_FORMAT//NTSC system

    HW_CLKCTRL_PLLCTRL0_WR (0x00050000);
    HW_CLKCTRL_FRAC1_WR (0x00000000);
    HW_CLKCTRL_TV_WR (0x00000000);
    HW_TVENC_CTRL_WR (0x00000000);
    HW_TVENC_CONFIG_WR (0x0C000240);
    HW_TVENC_FILTCTRL_WR (0x00080000);
    HW_TVENC_SYNCOFFSET_WR (0x00000359);
    HW_TVENC_VDACTEST_WR (0x00001000);
    HW_TVENC_MACROVISION0_WR( 0x00000000);
    HW_TVENC_MACROVISION1_WR ( 0x00000000);
    HW_TVENC_MACROVISION2_WR( 0x00000000);
    HW_TVENC_MACROVISION3_WR( 0x00000000);
    HW_TVENC_MACROVISION4_WR( 0x00000000);
```

```
#else//PAL system

    HW_CLKCTRL_PLLCTRL0_WR (0x00050000);
    HW_CLKCTRL_FRAC1_WR (0x00000000);
    HW_CLKCTRL_TV_WR (0x00000000);
    HW_TVENC_CTRL_WR (0x00000000);
    HW_TVENC_CONFIG_WR (0x0C000241);
    HW_TVENC_FILTCTRL_WR (0x00080000);
    HW_TVENC_SYNCOFFSET_WR (0x00000359);
    HW_TVENC_VDACTEST_WR (0x00001000);
    HW_TVENC_MACROVISION0_WR( 0x00000000);
    HW_TVENC_MACROVISION1_WR ( 0x00000000);
    HW_TVENC_MACROVISION2_WR( 0x00000000);
    HW_TVENC_MACROVISION3_WR( 0x00000000);
    HW_TVENC_MACROVISION4_WR( 0x00000000);
    HW_TVENC_COLORSUB0_WR (0x2A098ACB);
    HW_TVENC_COLORBURST_WR (0xD62A0000);

#endif

//VDAC setup
HW_TVENC_DACCTRL_WR(0x000c113b);

//TVENC clock settings
HW_CLKCTRL_TV.B.CLK_TV108M_GATE = 0;
HW_CLKCTRL_TV.B.CLK_TV_GATE = 0;

//Start the LCDIF
HW_LCDIF_CTRL.B.RUN = 1;
```

19.4 Programmable Registers

The following registers are available for programmer access and control of the TV encoder.

19.4.1 TV Encoder Control Register Description

This is the control register of the TV Encoder Block

HW_TVENC_CTRL	0x000
HW_TVENC_CTRL_SET	0x004
HW_TVENC_CTRL_CLR	0x008
HW_TVENC_CTRL_TOG	0x00C

Table 19-1. HW_TVENC_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	TVENC_MACROVISION_PRESENT	TVENC_COMPOSITE_PRESENT	TVENC_SVIDEO_PRESENT	TVENC_COMPONENT_PRESENT	RSRVD1																				DAC_FIFO_NO_WRITE	DAC_FIFO_NO_READ	DAC_DATA_FIFO_RST	RSRVD2			DAC_MUX_MODE

Table 19-2. HW_TVENC_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	This bit must be set to zero for normal operation. When set to one, it forces a block wide reset.
30	**CLKGATE**	RW	0x1	This bit must be set to zero for normal operation. When set to one, it gates off the clocks to the block.
29	**TVENC_MACROVISION_PRESENT**	RO	0x1	0= TV Encoder is not present in this product.
28	**TVENC_COMPOSITE_PRESENT**	RO	0x1	0= TV Encoder is not present in this product.
27	**TVENC_SVIDEO_PRESENT**	RO	0x0	0= TV Encoder is not present in this product.
26	**TVENC_COMPONENT_PRESENT**	RO	0x0	0= TV Encoder is not present in this product.
25:6	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
5	**DAC_FIFO_NO_WRITE**	RW	0x0	Setting this bit prohibits writes the DAC data fifo. This may be useful for draining the FIFO for diagnostic testing.
4	**DAC_FIFO_NO_READ**	RW	0x0	Setting this bit prohibits reads from the DAC data fifo. This may be useful for filling the FIFO at the beginning of an experiment in test mode.
3	**DAC_DATA_FIFO_RST**	RW	0x0	Set this bit to one to clear the DAC data FIFO and reset the FIFO pointers. Software should then set this bit back to zero before data can be written to the FIFO.
2:1	**RSRVD2**	RO	0x0	Always write zero to this bit field.
0	**DAC_MUX_MODE**	RW	0x0	Specifies the meaning of the output stream to each DAC. 0: Default. Composite mode where the composite signal is on DAC-A. 1: DAC test mode. This outputs the data written to the HW_TVENC_VDACTEST register directly to DAC-A. Set to 1 to enable DAC Test Mode.

DESCRIPTION:

Control register of TV Encoder

EXAMPLE:

```
Empty example.
```

19.4.2 TV Encoder Configuration Register Description

This is configuration register of the TV Encoder Block

HW_TVENC_CONFIG	0x010
HW_TVENC_CONFIG_SET	0x014
HW_TVENC_CONFIG_CLR	0x018
HW_TVENC_CONFIG_TOG	0x01C

Table 19-3. HW_TVENC_CONFIG

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD5				DEFAULT_PICFORM	YDEL_ADJ			RSRVD4	RSRVD3	ADD_YPBPR_PED	PAL_SHAPE	NO_PED	COLOR_BAR_EN	YGAIN_SEL		CGAIN		CLK_PHS		RSRVD2	FSYNC_ENBL	FSYNC_PHS	HSYNC_PHS	VSYNC_PHS	SYNC_MODE			RSRVD1	ENCD_MODE		

Table 19-4. HW_TVENC_CONFIG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**RSRVD5**	RO	0x0	Always write zeroes to this bit field.
27	**DEFAULT_PICFORM**	RW	0x1	Permits use of a set of default parameters, tailored to the mode defined by T_ENCD_MODE, to be used in place of the values in the LINEx registers.
26:24	**YDEL_ADJ**	RW	0x4	Delays luma versus chroma for composite output. Luma lags chroma by YDEL_ADJ-4 cycles of 27MHz clock. For example, if YDEL_ADJ=0, the luma leads by 4 cycles. And if YDEL_ADJ=7, the luma lags by 3 cycles.
23	**RSRVD4**	RO	0x0	Always write zeroes to this bit field.
22	**RSRVD3**	RO	0x0	Enables Svideo output on DAC-B and DAC-D, not available on HuaShan.
21	**ADD_YPBPR_PED**	RW	0x0	Permits the insertion of a black pedestal when sync is inserted on one or more component signals.
20	**PAL_SHAPE**	RW	0x0	Set to impose a 250nS edge shape as required by PAL, otherwise the steeper edges as specified by NTSC are used.
19	**NO_PED**	RW	0x0	Can be set to prevent insertion of a black pedestal as required by NTSC-J.
18	**COLOR_BAR_EN**	RW	0x0	Enable insertion of internally generated color bars.
17:16	**YGAIN_SEL**	RW	0x0	Controls the luma gain: 00 : NTSC 01 : PAL 1x : no gain

Table 19-4. HW_TVENC_CONFIG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:14	**CGAIN**	RW	0x0	Controls chroma gain for composite: 00 : NTSC Gain 01 : PAL Gain 1x : no gain
13:12	**CLK_PHS**	RW	0x0	Sync gen sub-block control. Phase adjustment of pixel clock at line-end. In 8-bit input mode, these distinguish Cb, Y1, Cr, Y2. Use to adjust the phase at the beginning of the line. The value of 00 has been used in all cases simulated to date.
11	**RSRVD2**	RO	0x0	Always write zeroes to this bit field.
10	**FSYNC_ENBL**	RW	0x0	External Sync sub-block control. Relates the internal field polarity to those of the input and/or output signals in interlaced modes. Set to 0.
9	**FSYNC_PHS**	RW	0x0	External Sync sub-block control. Relates the internal field polarity to those of the input and/or output signals in interlaced modes. 0 for PAL, 1 for NTSC
8	**HSYNC_PHS**	RW	0x0	External Sync sub-block control. If this bit is 0, in external-sync ("slave") mode, the rising edge of an external hsync input is used to derive the horiztonal sync for the TVENC block. Otherwise, the falling edge is used.
7	**VSYNC_PHS**	RW	0x0	External Sync sub-block control. If this bit is 0, in external-sync ("slave") mode, the rising edge of an external vsync input is used to derive the vertical sync for the TVENC block. Otherwise, the falling edge is used.
6:4	**SYNC_MODE**	RW	0x0	External Sync sub-block control. Defines the manner in which the input is synchronized to the display: 000: Ext slave: 8-bit Y/C in, SYNC in 001: Ext slave: 16-bit Y/C in, SYNC in 010: Master: 8-bit Y/C in, SYNC in 011: Master: 16-bit Y/C in, SYNC in 1xx: D1 mode: 8-bit Y/C in, SYNC out
3	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
2:0	**ENCD_MODE**	RW	0x0	External Sync sub-block control. Defines the video mode: 000: NTSC-M Mode 001: PAL-B Mode 010: PAL-M Mode 011: PAL-N Mode 100: PAL-CN Mode 101: NTSC with 700:300 scaling on "G" 110: PAL-60 Mode 111: NTSC Progressive

DESCRIPTION:

Configuration register of TV Encoder. Configures the output mode and related parameters.

EXAMPLE:

```
Empty example.
```

19.4.3 TV Encoder Filter Control Register Description

This is filter control register of the TV Encoder Block

HW_TVENC_FILTCTRL	0x020
HW_TVENC_FILTCTRL_SET	0x024
HW_TVENC_FILTCTRL_CLR	0x028
HW_TVENC_FILTCTRL_TOG	0x02C

Table 19-5. HW_TVENC_FILTCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1												YSHARP_BW	YD_OFFSETSEL	SEL_YLPF	SEL_CLPF	SEL_YSHARP	YLPF_COEFSEL	COEFSEL_CLPF	YS_GAINSGN	YS_GAINSEL		RSRVD2	RSRVD3	RSRVD4							

Table 19-6. HW_TVENC_FILTCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:20	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
19	**YSHARP_BW**	RW	0x0	Controls the luma sharpness filter bandwidth inside the luma filter sub-block.
18	**YD_OFFSETSEL**	RW	0x0	LU sub-block vector controls. Control the luma offset: 0 : do not substract 1 : subtract 16 from luma.
17	**SEL_YLPF**	RW	0x0	LU sub-block vector controls. Enables the luma low pass filter.
16	**SEL_CLPF**	RW	0x0	LU sub-block vector controls. Enables the chroma low pass filter.
15	**SEL_YSHARP**	RW	0x0	LU sub-block vector controls. Enables the luma sharpness filter.
14	**YLPF_COEFSEL**	RW	0x0	LU sub-block vector controls. Controls the luma low pass filter bandwidth: 0 : 5.5 MHz 1 : 4.2 MHz
13	**COEFSEL_CLPF**	RW	0x0	LU sub-block vector controls. Controls the chroma low pass filter bandwidth: 0 : 1.3 MHz 1 : 0.6 MHz
12	**YS_GAINSGN**	RW	0x0	LU sub-block vector controls. Controls the sign of the sharpness modification: 0 : positive 1 : negative
11:10	**YS_GAINSEL**	RW	0x0	LU sub-block vector controls. Controls the degree of luma sharpness enhancement by luma sharpness filter: 00 : 3dB 01 : 6dB 10 : 9dB 11 : 12dB

Table 19-6. HW_TVENC_FILTCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
9	**RSRVD2**	RO	0x0	Always write zeroes to this bit field.
8	**RSRVD3**	RO	0x0	Always write zeroes to this bit field.
7:0	**RSRVD4**	RO	0x0	Always write zeroes to this bit field.

DESCRIPTION:

Filter register of TV Encoder. Settings for input filter and RGB conversion blocks.

EXAMPLE:

```
Empty example.
```

19.4.4 TV Sync Offset Register Description

This is the Sync Offset register of the TV Encoder Block

HW_TVENC_SYNCOFFSET	0x030
HW_TVENC_SYNCOFFSET_SET	0x034
HW_TVENC_SYNCOFFSET_CLR	0x038
HW_TVENC_SYNCOFFSET_TOG	0x03C

Table 19-7. HW_TVENC_SYNCOFFSET

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1	HSO											VSO										HLC									

Table 19-8. HW_TVENC_SYNCOFFSET Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
30:20	**HSO**	RW	0x3ff	Defines the horizontal linestart in counts of CLK27 following the external sync edge (or EAV code in the case of D1). Works in conjuction with the CTRL0_HSYNC_PHS bit.
19:10	**VSO**	RW	0x3ff	Similar to HSO bit above in that it defines the vertical fieldstart in units of lines following the designated edge of the external vertical sync (or SFB rising edge in D1).
9:0	**HLC**	RW	0x359	Pixel count (pixels per line minus 1). It should be set to 857 for NTSC (interlaced and progressive) and to 863 for PAL.

DESCRIPTION:

Sync Offset register of TV Encoder. This register sets vertical and horizontal sync offset values and the pixels/line count.

EXAMPLE:

```
Empty example.
```

19.4.5 TV Encoder Horizontal Timing Sync Register 0 Description

This is the Horizontal Timing Sync Register 0 of the TV Encoder Block

HW_TVENC_HTIMINGSYNC0	0x040
HW_TVENC_HTIMINGSYNC0_SET	0x044
HW_TVENC_HTIMINGSYNC0_CLR	0x048
HW_TVENC_HTIMINGSYNC0_TOG	0x04C

Table 19-9. HW_TVENC_HTIMINGSYNC0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2						SYNC_END										RSRVD1						SYNC_STRT									

Table 19-10. HW_TVENC_HTIMINGSYNC0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSRVD2**	RO	0x0	Always write zeroes to this bit field.
25:16	**SYNC_END**	RW	0x04d	Normal end of sync pulse in first half line of video line (pixel count - 1).
15:10	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
9:0	**SYNC_STRT**	RW	0x00e	Start of sync pulse in line or half-line(pixel count - 1).

DESCRIPTION:

Horizontal Timing Sync Register 0 of TV Encoder. The line registers only needs to be programmed if the DEFAULT_PICFORM bit is cleared.

EXAMPLE:

```
Empty example.
```

19.4.6 TV Encoder Horizontal Timing Sync Register 1 Description

This is the Horizontal Timing Sync Register 1 of the TV Encoder Block

HW_TVENC_HTIMINGSYNC1	0x050
HW_TVENC_HTIMINGSYNC1_SET	0x054
HW_TVENC_HTIMINGSYNC1_CLR	0x058
HW_TVENC_HTIMINGSYNC1_TOG	0x05C

Table 19-11. HW_TVENC_HTIMINGSYNC1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2						SYNC_EQEND										RSRVD1						SYNC_SREND									

Table 19-12. HW_TVENC_HTIMINGSYNC1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSRVD2**	RO	0x0	Always write zeroes to this bit field.
25:16	**SYNC_EQEND**	RW	0x02d	End of sync pulse in each half line in equalization regions of vertical blanking (pixel count - 1).
15:10	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
9:0	**SYNC_SREND**	RW	0x17b	End of sync pulse in each half line in serration region of vertical blanking (pixel count - 1).

DESCRIPTION:

Horizontal Timing Sync Register 1 of TV Encoder. This register only needs to be programmed if the DEFAULT_PICFORM bit is cleared.

EXAMPLE:

```
Empty example.
```

19.4.7 TV Encoder Horizontal Timing Active Register Description

This is the Horizontal Timing Active Register of the TV Encoder Block

HW_TVENC_HTIMINGACTIVE	0x060
HW_TVENC_HTIMINGACTIVE_SET	0x064
HW_TVENC_HTIMINGACTIVE_CLR	0x068
HW_TVENC_HTIMINGACTIVE_TOG	0x06C

Table 19-13. HW_TVENC_HTIMINGACTIVE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2						ACTV_END										RSRVD1						ACTV_STRT									

Table 19-14. HW_TVENC_HTIMINGACTIVE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSRVD2**	RO	0x0	Always write zeroes to this bit field.
25:16	**ACTV_END**	RW	0x3a4	Horizontal end of active video (pixel count - 1).
15:10	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
9:0	**ACTV_STRT**	RW	0x089	Horizontal start of active video (pixel count - 1).

DESCRIPTION:

Horizontal Timing Active Register of TV Encoder. Specifies active line timing. This register only needs to be programmed if the DEFAULT_PICFORM bit is cleared.

EXAMPLE:

```
Empty example.
```

19.4.8 TV Encoder Horizontal Timing Color Burst Register 0 Description

This is the Horizontal Timing Color Burst Register 0 of the TV Encoder Block

HW_TVENC_HTIMINGBURST0	0x070
HW_TVENC_HTIMINGBURST0_SET	0x074
HW_TVENC_HTIMINGBURST0_CLR	0x078
HW_TVENC_HTIMINGBURST0_TOG	0x07C

Table 19-15. HW_TVENC_HTIMINGBURST0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2						WBRST_STRT										RSRVD1						NBRST_STRT									

Table 19-16. HW_TVENC_HTIMINGBURST0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSRVD2**	RO	0x0	Always write zeroes to this bit field.
25:16	**WBRST_STRT**	RW	0x051	Start of wide color burst for Macrovision (pixel count - 1).
15:10	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
9:0	**NBRST_STRT**	RW	0x059	Start of normal color burst (pixel count - 1).

DESCRIPTION:

Horizontal Timing Color Burst Register 0 of TV Encoder. This register only needs to be programmed if the DEFAULT_PICFORM bit is cleared.

EXAMPLE:

```
Empty example.
```

19.4.9 TV Encoder Horizontal Timing Color Burst Register 1 Description

This is the Horizontal Timing Color Burst Register 1 of the TV Encoder Block

HW_TVENC_HTIMINGBURST1	0x080
HW_TVENC_HTIMINGBURST1_SET	0x084
HW_TVENC_HTIMINGBURST1_CLR	0x088
HW_TVENC_HTIMINGBURST1_TOG	0x08C

Table 19-17. HW_TVENC_HTIMINGBURST1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																						BRST_END									

Table 19-18. HW_TVENC_HTIMINGBURST1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:10	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
9:0	**BRST_END**	RW	0x07b	End of normal or wide color burst (pixel count - 1).

DESCRIPTION:

Horizontal Timing Color Burst Register 1 of TV Encoder. This register only needs to be programmed if the DEFAULT_PICFORM bit is cleared.

EXAMPLE:

```
Empty example.
```

19.4.10 TV Encoder Vertical Timing Register 0 Description

This is the Vertical Timing Register 0 of the TV Encoder Block

HW_TVENC_VTIMING0	0x090
HW_TVENC_VTIMING0_SET	0x094
HW_TVENC_VTIMING0_CLR	0x098
HW_TVENC_VTIMING0_TOG	0x09C

Table 19-19. HW_TVENC_VTIMING0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD3						VSTRT_PREEQ										RSRVD2		VSTRT_ACTV						RSRVD1		VSTRT_SUBPH					

Table 19-20. HW_TVENC_VTIMING0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSRVD3**	RO	0x0	Always write zeroes to this bit field.
25:16	**VSTRT_PREEQ**	RW	0x20c	Last half line of active video followed by pre-equalization.
15:14	**RSRVD2**	RO	0x0	Always write zeroes to this bit field.
13:8	**VSTRT_ACTV**	RW	0x29	Last half line of sub-phase followed by active video.
7:6	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
5:0	**VSTRT_SUBPH**	RW	0x11	Last half line of post equalization followed by sub-phase -- vertical blanking lines after post equalization.

DESCRIPTION:

Vertical Timing Register 0 of TV Encoder. Specifies frame vertical timing parameters. This register only needs to be programmed if the DEFAULT_PICFORM bit is cleared.

EXAMPLE:

```
Empty example.
```

19.4.11 TV Encoder Vertical Timing Register 1 Description

This is the Vertical Timing Register 1 of the TV Encoder Block

HW_TVENC_VTIMING1	0x0a0
HW_TVENC_VTIMING1_SET	0x0a4
HW_TVENC_VTIMING1_CLR	0x0a8
HW_TVENC_VTIMING1_TOG	0x0aC

Table 19-21. HW_TVENC_VTIMING1

31 30	29 28 27 26 25 24	23 22	21 20 19 18 17 16	15 14 13 12 11 10	09 08 07 06 05 04	03 02 01 00
RSRVD3	VSTRT_POSTEQ	RSRVD2	VSTRT_SERRA	RSRVD1	LAST_FLD_LN	

Table 19-22. HW_TVENC_VTIMING1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD3**	RO	0x0	Always write zeroes to this bit field.
29:24	**VSTRT_POSTEQ**	RW	0x0b	Last half line of serration followed by post-equalization.
23:22	**RSRVD2**	RO	0x0	Always write zeroes to this bit field.
21:16	**VSTRT_SERRA**	RW	0x05	Last half line of pre-equalization followed by serration.
15:10	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
9:0	**LAST_FLD_LN**	RW	0x20c	Last half line of field. Usually the same as VSTRT_PREEQ.

DESCRIPTION:

Vertical Timing Register 1 of TV Encoder. Specifies frame vertical timing parameters. This register only needs to be programmed if the DEFAULT_PICFORM bit is cleared.

EXAMPLE:

```
Empty example.
```

19.4.12 TV Encoder Miscellaneous Line Control Register Description

This is the Miscellaneous Register of the TV Encoder Block

HW_TVENC_MISC	0x0b0
HW_TVENC_MISC_SET	0x0b4
HW_TVENC_MISC_CLR	0x0b8
HW_TVENC_MISC_TOG	0x0bC

Table 19-23. HW_TVENC_MISC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD3							LPF_RST_OFF									RSRVD2				NTSC_LN_CNT	PAL_FSC_PHASE_ALT	FSC_PHASE_RST		BRUCHB		AGC_LVL_CTRL		RSRVD1	CS_INVERT_CTRL		Y_BLANK_CTRL	

Table 19-24. HW_TVENC_MISC Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSRVD3**	RO	0x0	Always write zeroes to this bit field.
24:16	**LPF_RST_OFF**	RW	0x110	Programs the time to generate a pulse (in External Sync unit) so as to preload a pipeline in the Y delay module in the Filter unit. The value was emperically found to be 272 for NTSC and D1 mode, 284 for PAL in D1 mode and 136 for progressive in external sync mode.
15:12	**RSRVD2**	RO	0x0	Always write zeroes to this bit field.
11	**NTSC_LN_CNT**	RW	0x1	Aligns even/odd field identification with internal field count in Sync Gen unit. 0 : for PAL-B. 1 : for NTSC.
10	**PAL_FSC_PHASE_ALT**	RW	0x0	Enables PAL manner of phase alternation by field of color subcarrier in Sync Gen unit.
9:8	**FSC_PHASE_RST**	RW	0x1	Controls timing of color subcarrier phase reset in Sync Gen unit. 00 : for progressive. 01 : for NTSC. 10 : for PAL-M, PAL-N and PAL-60. 11 : for other PAL cases.

Table 19-24. HW_TVENC_MISC Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
7:6	**BRUCHB**	RW	0x1	Controls Bruch blanking in Sync Gen unit. 00 : for progressive. 01 : for 525 line cases with NTSC color. 10 : for PAL-M and PAL-60. 11 : for other PAL cases.
5:4	**AGC_LVL_CTRL**	RW	0x1	Controls AGC levels in Macrovision sub-block. 00 : for "mixed NTSC". i.e., 714:286 on composite and 700:300 on component. 01 : for NTSC. 10 : for PAL. 11 : for progressive.
3	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
2	**CS_INVERT_CTRL**	RW	0x0	Disables illegal color stripe inversion modes in Macrovision sub-block.
1:0	**Y_BLANK_CTRL**	RW	0x1	Controls the blanking level in the luma processing sub-block. 00 : 700:300 blanking for progressive. 01 : 714:286 blanking on both composite and component. 10 : 714:286 blanking for somposite and 700:300 blanking for component. 11 : 700:300 blanking for PAL systems.

DESCRIPTION:

Miscellaneous Control Register of TV Encoder. This register contains miscellaneous line control parameters. This register only needs to be programmed if the DEFAULT_PICFORM bit is cleared.

EXAMPLE:

```
Empty example.
```

19.4.13 TV Encoder Color Subcarrier Register 0 Description

This is Color Subcarrier register 0 of the TV Encoder Block

HW_TVENC_COLORSUB0	0x0c0
HW_TVENC_COLORSUB0_SET	0x0c4
HW_TVENC_COLORSUB0_CLR	0x0c8
HW_TVENC_COLORSUB0_TOG	0x0cC

Table 19-25. HW_TVENC_COLORSUB0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PHASE_INC																															

Table 19-26. HW_TVENC_COLORSUB0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**PHASE_INC**	RW	0x21F07C1F	Defines the frequency for the color subcarrier. The value should be set to 0x21f07c1f for NTSC or 0x2a098acb for PAL-B. This relevant to composite output only. The units are such that a phase increment of 360 degrees is entered as 0xffffffff.

DESCRIPTION:

Color Subcarrier register 0 of TV Encoder

EXAMPLE:

```
Empty example.
```

19.4.14 TV Encoder Color Subcarrier Register 1 Description

This is Color Subcarrier register 1 of the TV Encoder Block

HW_TVENC_COLORSUB1	0x0d0
HW_TVENC_COLORSUB1_SET	0x0d4
HW_TVENC_COLORSUB1_CLR	0x0d8
HW_TVENC_COLORSUB1_TOG	0x0dC

Table 19-27. HW_TVENC_COLORSUB1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PHASE_OFFSET																															

Table 19-28. HW_TVENC_COLORSUB1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**PHASE_OFFSET**	RW	0x0	Phase offset for color subcarrier. This is added to the phase as otherwise generated. This should typically be set to zero (default).

DESCRIPTION:

Color Subcarrier register 1 of TV Encoder

EXAMPLE:

```
Empty example.
```

19.4.15 TV Encoder Copy Protect Register Description

This is the Copy Protect register of the TV Encoder Block

HW_TVENC_COPYPROTECT	0x0e0
HW_TVENC_COPYPROTECT_SET	0x0e4
HW_TVENC_COPYPROTECT_CLR	0x0e8
HW_TVENC_COPYPROTECT_TOG	0x0eC

Table 19-29. HW_TVENC_COPYPROTECT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																WSS_ENBL	CGMS_ENBL	WSS_CGMS_DATA													

Table 19-30. HW_TVENC_COPYPROTECT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
15	**WSS_ENBL**	RW	0x0	Enable WSS insertion for 625 line modes.
14	**CGMS_ENBL**	RW	0x0	Enable CGMS insertion for 525 line modes (both interlaced and progressive).
13:0	**WSS_CGMS_DATA**	RW	0x0	Payload data for insertion in either WSS or CGMS mode.

DESCRIPTION:

Copy Protect register of TV Encoder

EXAMPLE:

```
Empty example.
```

19.4.16 TV Encoder Closed Caption Register Description

This is the Closed Caption register of the TV Encoder Block

HW_TVENC_CLOSEDCAPTION	0x0f0
HW_TVENC_CLOSEDCAPTION_SET	0x0f4
HW_TVENC_CLOSEDCAPTION_CLR	0x0f8
HW_TVENC_CLOSEDCAPTION_TOG	0x0fC

Table 19-31. HW_TVENC_CLOSEDCAPTION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1												CC_ENBL		CC_FILL		CC_DATA															

Table 19-32. HW_TVENC_CLOSEDCAPTION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:20	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
19:18	**CC_ENBL**	RW	0x0	Closed Caption Enable. Set mutually exclusive????? bit 1: enable insertion in even (bottom) field bit 0: enable insertion in odd (top) field }
17:16	**CC_FILL**	RW	0x0	Holds bits used to determine whether the two bytes in CC_DATA are to be inserted in the odd (upper) field or the even (lower) field.
15:0	**CC_DATA**	RW	0x0	Data to be inserted.

DESCRIPTION:

Closed Caption register of TV Encoder

EXAMPLE:

```
Empty example.
```

19.4.17 TV Encoder Color Burst Register Description

This is Color Burst Register of the TV Encoder Block

HW_TVENC_COLORBURST	0x140
HW_TVENC_COLORBURST_SET	0x144
HW_TVENC_COLORBURST_CLR	0x148
HW_TVENC_COLORBURST_TOG	0x14C

Table 19-33. HW_TVENC_COLORBURST

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
NBA								PBA								RSRVD1				RSRVD2											

Table 19-34. HW_TVENC_COLORBURST Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**NBA**	RW	0xc8	Parameter to define color burst. Should be set to 0xc8 for NTSC and 0xd6 for PAL-B
23:16	**PBA**	RW	0x0	Parameter to define color burst. Should be set to 0x00 for NTSC and 0x2a for PAL-B
15:12	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
11:0	**RSRVD2**	RO	0x0	Always write zeros to this bit field.

DESCRIPTION:

Color Burst Register of TV Encoder

EXAMPLE:

```
Empty example.
```

19.4.18 TV Encoder DAC Control Register Description

This register controls the Video DACs associated with the TVENC.

HW_TVENC_DACCTRL	0x1a0
HW_TVENC_DACCTRL_SET	0x1a4
HW_TVENC_DACCTRL_CLR	0x1a8
HW_TVENC_DACCTRL_TOG	0x1aC

Table 19-35. HW_TVENC_DACCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TEST3	RSRVD1	RSRVD2	JACK1_DIS_DET_EN	TEST2	RSRVD3	RSRVD4	JACK1_DET_EN	TEST1	DISABLE_GND_DETECT	JACK_DIS_ADJ		GAINDN	GAINUP	INVERT_CLK	SELECT_CLK	BYPASS_ACT_CASCODE	RSRVD5	RSRVD6	PWRUP1	WELL_TOVDD	RSRVD7	RSRVD8	DUMP_TOVDD1	LOWER_SIGNAL	RVAL			NO_INTERNAL_TERM	HALF_CURRENT	CASC_ADJ	

Table 19-36. HW_TVENC_DACCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**TEST3**	RW	0x0	Test bit.
30	**RSRVD1**	RO	0x0	Always write zero to this bit.
29	**RSRVD2**	RO	0x0	Always write zero to this bit.
28	**JACK1_DIS_DET_EN**	RW	0x0	Enables the jack1 disconnect detector. This should only be activated when pwrup1 is high (the video dac is active).
27	**TEST2**	RW	0x0	Test bit.
26	**RSRVD3**	RO	0x0	Always write zero to this bit.
25	**RSRVD4**	RO	0x0	Always write zero to this bit.
24	**JACK1_DET_EN**	RW	0x0	Enables the jack1 connect detector. This places a weak pullup to VDDA on the video1 output pad. When a 75ohm termination load is applied the pad is pulled to ground enabling the detect circuitry. This signal should only be active when PWRUP1 is low (the video dac is turned off).
23	**TEST1**	RW	0x0	Test bit.
22	**DISABLE_GND_DETECT**	RW	0x0	If the video jack detects a grounded output (headphone plugged into video/audio jack) then the IRQ handler should set this bit to turn off the power of the ground detector until the jack is unplugged and replugged (to check the impedance again).
21:20	**JACK_DIS_ADJ**	RW	0x0	These bits adjust the trip voltage for the disconnect detector. 00=1.35 (default), 01=1.50, 10=1.65, 11=1.80.

Table 19-36. HW_TVENC_DACCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19	**GAINDN**	RW	0x0	Set to gain the digital code down. This mode will usually be used with the gainup bit to turn the DAC conversion gain down 20%. When both gaindn and gainup are set the max digital code (in non-Macrovision mode) will be 1003 and will give a DAC output voltage of 1.02V.
18	**GAINUP**	RW	0x0	Set to gain the digital code up by 25%. This mode (along with GAINDN) can be used in non-macrovision mode to reduce power consumption by 20% and use more dynamic range in the DAC. In non-Macrovision mode the max digital code (pregain) is 802 and is 1003 after the gain.
17	**INVERT_CLK**	RW	0x0	Set to invert the dac clk (change the clock edge that the DAC updates on).
16	**SELECT_CLK**	RW	0x0	Default value of 0 selects a 27MHz sample rate for the video DACs. Set to one to use an internal 4X upsampler to run the video DAC at 108MHz. This should reduce out of band energy and reduce needed external filtering.
15	**BYPASS_ACT_CASCODE**	RW	0x0	This bit bypasses/disables the active cascode amp that is used to improve the low frequency PSRR of the video DACs. It defaults low where the active cascode is enabled.
14	**RSRVD5**	RO	0x0	Always write zero to this bit.
13	**RSRVD6**	RO	0x0	Always write zero to this bit.
12	**PWRUP1**	RW	0x0	Power up video dac channel 1.
11	**WELL_TOVDD**	RW	0x0	Set to change the nwell connection for the current steering PFET switched from VDDA to VDDD for all 3 video DACs. It trades off slightly better glitch energy for reduced VDDD PSRR.
10	**RSRVD7**	RO	0x0	Always write zero to this bit.
9	**RSRVD8**	RO	0x0	Always write zero to this bit.
8	**DUMP_TOVDD1**	RW	0x0	Enables low power feature for video DAC1 to dump the unused DAC current to the VDDD rail instead of ground. Each channel has a separate control because VDDD may not be able to handle the power from all three DACs.
7	**LOWER_SIGNAL**	RW	0x0	Decreases video DAC output swings by 33% to save corresponding power. Default is low- disabled.
6:4	**RVAL**	RW	0x0	Used to adjust the on-chip resistances that set the termination resistor and the peak output current. 000 = +20%, 001 = +12.4%, 010=+5.8%, 011=nom, 100=-4.4%, 101=-10%, 110=-14.2%, 111=-17.3%. This effects all video DAC channels.
3	**NO_INTERNAL_TERM**	RW	0x0	This bit disables the internal termination mode for all three video DACs. With this bit cleared each video dac (that is powered up) has an internal 75ohm termation. When the bit is set the 75ohm termination is removed.

Table 19-36. HW_TVENC_DACCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**HALF_CURRENT**	RW	0x0	When cleared the nominal peak output current for each DAC is 34.67mA. When set the nominal peak output current is 17.3mA. The half current bit should be set whenever NO_INTERNAL_TERM is set AND there is only a signal 75ohm external load.
1:0	**CASC_ADJ**	RW	0x0	This field adjusts the cascode voltage for the current sources (all three video DACs). 00=nom, 01=-50mV, 10=+50mV, 11=+100mV. The higher settings might be useful when headroom is low.

DESCRIPTION:

This register contains control bits for programming the DAC.

EXAMPLE:

```
Empty example.
```

19.4.19 TV Encoder DAC Status Register Description

This register contains status bits for DAC connect/disconnect interrupts.

HW_TVENC_DACSTATUS	0x1b0
HW_TVENC_DACSTATUS_SET	0x1b4
HW_TVENC_DACSTATUS_CLR	0x1b8
HW_TVENC_DACSTATUS_TOG	0x1bC

Table 19-37. HW_TVENC_DACSTATUS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																			RSRVD2	RSRVD3	JACK1_DET_STATUS	RSRVD4	RSRVD5	JACK1_GROUNDED	RSRVD6	RSRVD7	JACK1_DIS_DET_IRQ	RSRVD8	RSRVD9	JACK1_DET_IRQ	ENIRQ_JACK

Table 19-38. HW_TVENC_DACSTATUS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:13	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
12	**RSRVD2**	RO	0x0	Always write zero to this bit.
11	**RSRVD3**	RO	0x0	Always write zero to this bit.
10	**JACK1_DET_STATUS**	RO	0x0	This status bit is high if the output is pulled to ground with an impedance less than 20ohm (not a video load).
9	**RSRVD4**	RO	0x0	Always write zero to this bit.
8	**RSRVD5**	RO	0x0	Always write zero to this bit.
7	**JACK1_GROUNDED**	RO	0x0	This status bit is high if the output is pulled to ground with an impedance less than 20ohm (not a video load).

Table 19-38. HW_TVENC_DACSTATUS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
6	**RSRVD6**	RO	0x0	Always write zero to this bit.
5	**RSRVD7**	RO	0x0	Always write zero to this bit.
4	**JACK1_DIS_DET_IRQ**	RW	0x0	Jack1 disconnect detect interrupt status bit. Reset this bit by writing a one to the SCT clear address space and not by a general write.
3	**RSRVD8**	RO	0x0	Always write zero to this bit.
2	**RSRVD9**	RO	0x0	Always write zero to this bit.
1	**JACK1_DET_IRQ**	RW	0x0	Jack1 detect interrupt status bit. Reset this bit by writing a one to the SCT clear address space and not by a general write. This IRQ goes high if the video output is pulled to ground. It will not trigger if the video output is pulled to HP_VGND in capless mode (VAG voltage).
0	**ENIRQ_JACK**	RW	0x0	Setting this bit enables all DAC jack detect interrupt sources.

DESCRIPTION:

This register contains DAC interrupt control and status bits.

EXAMPLE:

```
Empty example.
```

19.4.20 TV Encoder vDAC Test Register Description

This register used to write values directly to the VDACs.

HW_TVENC_VDACTEST	0x1c0
HW_TVENC_VDACTEST_SET	0x1c4
HW_TVENC_VDACTEST_CLR	0x1c8
HW_TVENC_VDACTEST_TOG	0x1cC

Table 19-39. HW_TVENC_VDACTEST

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																		ENABLE_PIX_INT_GAIN	BYPASS_PIX_INT	BYPASS_PIX_INT_DROOP	TEST_FIFO_FULL	DATA									

Table 19-40. HW_TVENC_VDACTEST Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:14	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
13	**ENABLE_PIX_INT_GAIN**	RW	0x0	Enabling this bit will attenuate the output of the pix_int block to ensure that the frequency response is never greater than 0.0dB, however, the DC gain will then be less than 1. When cleared, the DC response is set to 0 dB, and the maximum gain in the passband is about 0.45dB.
12	**BYPASS_PIX_INT**	RW	0x0	When this bit is set, the pixel interpolator is bypassed. Setting this bit will require a higher order analog reconstruction filter outside the chip.
11	**BYPASS_PIX_INT_DROOP**	RW	0x0	This bypasses the droop compensation portion of the pixel interpolator. If set, the pixel interpolator will have approximately 1dB of droop across the pass band.
10	**TEST_FIFO_FULL**	RO	0x0	When this bit is asserted then the DAC test fifo is full and the HW_TVENC_DACTEST register should not be written. This bit only has meaning when the DAC_MUX_MODE field equals 11b.
9:0	**DATA**	RW	0x000	Digital data sample going to DAC 0.

DESCRIPTION:

This register contains 10 bit data samples going to the DAC and one status bit.

EXAMPLE:

```
Empty example.
```

19.4.21 TV Encoder Version Register Description

This register always returns a known read value for debug purposes it indicates the version of the block.

HW_TVENC_VERSION 0x1d0

Table 19-41. HW_TVENC_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 19-42. HW_TVENC_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x01	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x00	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
if (HW_TVENC_VERSION.B.MAJOR != 1) Error();
```

TVENC Block v1.1, Revision 1.00

Chapter 20
Video DAC

20.1 Overview

The i.MX23 provides an on-chip video DAC to drive an external video device.

20.2 Details of Operations

The video DAC is designed to directly drive a 75 Ω composite video load. Composite signals have a maximum signal bandwidth of 5.1 MHz. The video DAC uses ~10X oversampling (108 MHz sample rate) to allow for a very simple anti-alias filter at the output. In some applications, it may be acceptable to use a single capacitor at the output for the anti-alias filter. If this does not provide adequate signal quality, a 2-pole LC filter can be used.

The video DAC includes a number of power saving features. First, it runs off the 1.62-2.0 V VDDA power rail instead of a more common 3.3 V power rail. Second, in non-Macrovision mode, the power can be reduced by ~25% by setting tvenc_dacctrl_gaindn AND dacctrl_gainup. These bits reduce the peak output signal from 1.3 V (only needed for Macrovision) to 1.02 V.

A third power saving feature for the video DAC is the ability to run in single or double termination mode. In single termination mode the power consumption of the video DAC is reduced by 50%. In a typical video driver, the cable is terminated with 75 Ω on the driver side and receiver side of the cable. This minimizes signal reflections and provides the best signal quality. However, the composite video signals have relatively low bandwidth, and the typical video cable is relatively short. Therefore, the signal reflections caused by a single terminated load (only at the receiver end) should not cause a noticeable reduction in signal quality. The video DAC has an integrated/switchable 75 Ω termination resistor. This allows the DAC to run in either low power (single termination mode) or high quality (double termination mode). It is expected that the low power mode of operation will provide video quality that is acceptable for nearly all users. But this can be adjusted in software as desired. To run in the low power, single termination mode, set bits tvenc_dacctrl_no_internal_term AND tvenc_dacctrl_half_current.

A fourth low power feature for the video DAC is jack detect. Jack detect enables the video DACs to be turned on only when a video load is present and can allow the video DACs to turn off as soon as the video load is removed. Some video players will choose to use a combined composite & headphone out jack. When these jacks are used, a normal headphone (non-video plug) will cause the video terminal to be

shorted to ground. The jack detector includes a short detect to distinguish this grounded load from a 75 Ω video DAC load. The list below describes the expected usage of the video jack detect.

1. Enable jack detect and wait for IRQ
2. When IRQ indicates a jack detect event, if the player supports headphone and composite combined jack, then go to step 3, else jump to step 4.
3. Check status of jack_grounded. If jack_grounded is high (indicates headphone [not video] is plugged in) then set disable_gnd_detect to cut the power consumption and wait in this state until jack_detect goes low (do not disable jack detect). You will probably need to disable the interrupt and just poll the status of jack_detect. When jack_detect goes low (indicating an unplug) then unset the disable_gnd_detect and return to step 1.
4. A video jack has been detected. Unset the jack_detect_en, power-up the video DAC, and set jack_disconn_det_en. Wait in this state until an IRQ for a jack_disconn is detected. It may be desirable to add a special "double check" feature here to make sure it detects twice before proceeding. When jack_disconn is detected, power off the video DAC and return to step1.

Chapter 21
Synchronous Serial Ports (SSP)

This chapter describes the two identical synchronous serial ports (SSP) included on the i.MX23. It includes sections on external pins, bit rate generation, frame formats, Winbond SPI mode, Motorola SPI mode, Texas Instruments Synchronous Serial Interface (SSI) mode, and SD/SDIO/MMC mode. Programmable registers are described in Section 21.10, "Programmable Registers.

21.1 Overview

The synchronous serial port is a flexible interface for inter-IC and removable media control and communication. The SSP supports master operation of SPI, Texas Instruments SSI and 1-bit, 4-bit, and 8-bit SD/SDIO/MMC. The SPI mode has enhancements to support 1-bit legacy MMC cards. SPI master dual (2-bit) and quad (4-bit) mode reads are also supported. The SSP also supports slave operation for the SPI and SSI modes. The SSP has a dedicated DMA channel in the bridge and can also be controlled directly by the CPU through PIO registers. Figure 21-1 illustrates one of the two SSP ports included on the i.MX23. The only interaction between SSP1 and SSP2 is that they share the same input, SSPCLK.

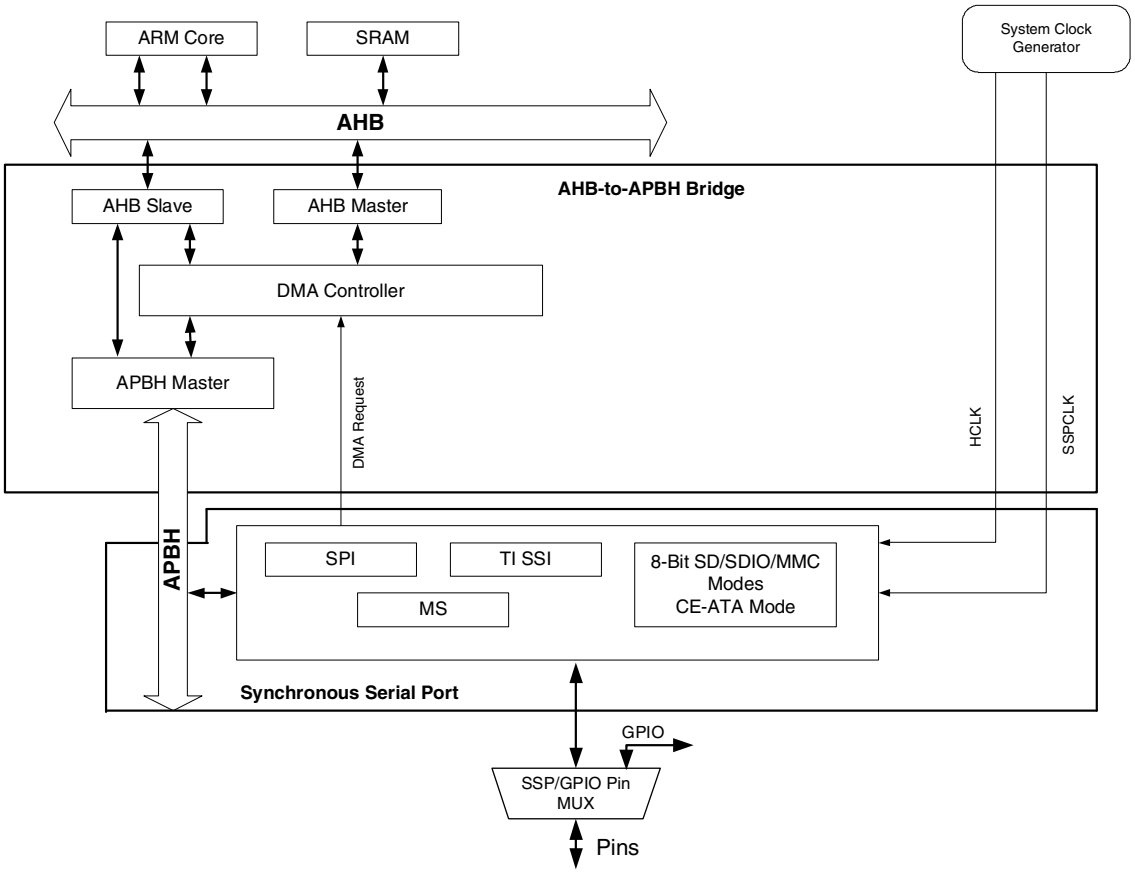


Figure 21-1. Synchronous Serial Port Block Diagram

21.2 External Pins

Table 21-1 lists the SSP pin placements for all supported modes.

PIN NAME	MOTOROLA SPI MODE	WINBOND SPI MODE	TI SSI MODE	SD/SDIO/ MMC MODES
SSP_SCK	SCK	CLK	CLK	CLK
SSP_CMD	MOSI	DI (IO0)	MOSI	CMD
SSP_DATA0	MISO	DO (IO1)	MISO	DATA0
SSP_DATA1		WPn (IO2)		DATA1/IRQ
SSP_DATA2		HOLDn (IO3)		DATA2
SSP_DATA3	SSn0	SSn0	SSn	DATA3
SSP_DATA4	SSn1	SSn1		DATA4
SSP_DATA5	SSn2	SSn2		DATA5
SSP_DATA6				DATA6
SSP_DATA7				DATA7
SSP_DETECT				CARD_DETECT

Table 21-1. SSP Pin Matrix

The pin control interface on the i.MX23 provides all digital pins with selectable output drive strengths. In addition, all SSP data pins have selectable 47-KΩ pullup resistors, and SSP command pins have 10-KΩ pullups. Configuring the SSP_CMD pad to connect to the internal 10-KΩ pullup is recommended for SD/SDIO/MMC modes during the Card_ID phase. After the Card_ID phase, the 10-KΩ pullup should be disabled, and the weaker external 47-KΩ pullup takes over. The SSP_DATA pads also can be configured to connect to an internal 47-KΩ pullup, which is required for SD/SDIO/MMC modes.

21.3 Bit Rate Generation

The serial bit rate is derived by dividing down the internal clock SSPCLK. The clock is first divided by an even prescale value, CLOCK_DIVIDE, from 2 to 254, which is programmed in HW_SSP_TIMING. The clock is further divided by a value from 1 to 256, which is 1 + CLOCK_RATE, where CLOCK_RATE is the value programmed in HW_SSP_TIMING.

The frequency of the output signal bit clock SSP_SCK is defined as follows:

$$SSP\_SCK = \frac{SSPCLK}{CLOCK\_DIVIDE * (1+CLOCK\_RATE)}$$

For example, if SSPCLK is 3.6864 MHz, and CLOCK_DIVIDE = 2, then SSP_SCK has a frequency range from 7.2 kHz to 1.8432 MHz. See Chapter 4, “Clock Generation and Control,” for more clock details.

21.4 Frame Format for SPI and SSI

Each data frame is between 4 and 16 bits long, depending on the size of data programmed, and is transmitted starting with the MSB. Two basic frame types can be selected:

- Motorola SPI
- Texas Instruments Synchronous Serial Interface (SSI)

For both formats, the serial clock (SSP_SCK) is held inactive while the SSP is idle and transitions at the programmed frequency only during active transmissions or reception of data. The idle state of SSP_SCK is used to provide a receive time-out indication, which occurs when the FIFO still contains data after a time-out period.

For Motorola SPI frame format, the serial frame (SSn) pin is active low and is asserted (pulled down) during the entire transmission of the frame.

For Texas Instruments synchronous serial interface (SSI) frame format, the SSn pin is pulsed for one serial clock period starting at its rising edge, prior to the transmission of each frame. For this frame format, both the SSP and the off-chip slave device drive their output on data on the rising edge of SSP_SCK, and latch data from the other device on the falling edge.

The SSP master supports up to three combinations of SPI and SSI slave devices connected. Three SSn pins are provided but only one can be active at a time.

21.5 Motorola SPI Mode

The SPI mode is used for general inter-component communication and legacy 1-bit MMC cards.

21.5.1 SPI DMA Mode

The SPI bus is inherently a full-duplex bidirectional interface. However, as most applications only require half-duplex data transmission, the i.MX23 has a single DMA channel for the SSP. It can be configured for either transmit or receive. In DMA receive mode, the SPI continuously repeats the word written to its data register. In DMA transmit mode, the SPI ignores the incoming data.

21.5.2 Motorola SPI Frame Format

The Motorola SPI interface is a four-wire interface where the SSn signal behaves as a slave select. The main feature of the Motorola SPI format is that the inactive state and phase of SSP_SCK signal are programmable through the polarity and phase bits within the HW_SSP_CTRL1.

21.5.2.1 Clock Polarity

- When the clock polarity control bit is low, it produces a steady-state low value on the SSP_SCK pin.
- When the clock polarity control bit is high, a steady-state high value is placed on the SSP_SCK pin when data is not being transferred.

21.5.2.2 Clock Phase

The phase control bit selects the clock edge that captures data and allows it to change state. It has the most impact on the first bit transmitted, by either allowing or not allowing a clock transition before the first data-capture edge.

- When the phase control bit is low, data is captured on the first clock-edge transition.
- When the clock phase control bit is high, data is captured on the second clock-edge transition.

21.5.3 Motorola SPI Format with Polarity=0, Phase=0

SIngle and continuous transmission signal sequences for Motorola SPI format with POLARITY=0, PHASE=0 are shown in Figure 21-2 and Figure 21-3.

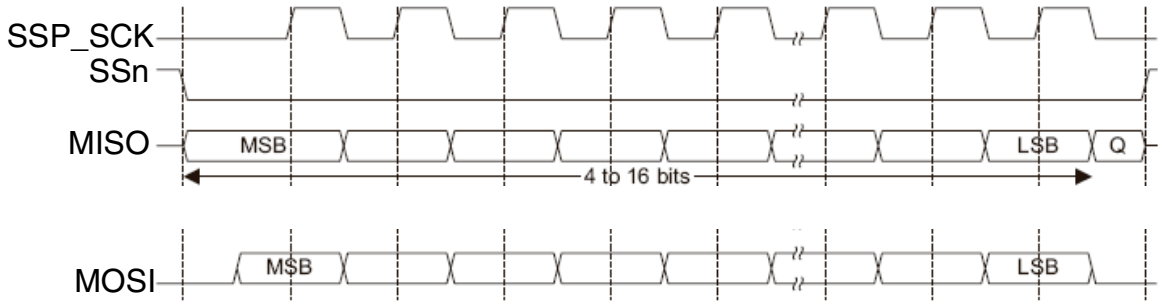


Figure 21-2. Motorola SPI Frame Format (Single Transfer) with POLARITY=0 and PHASE=0

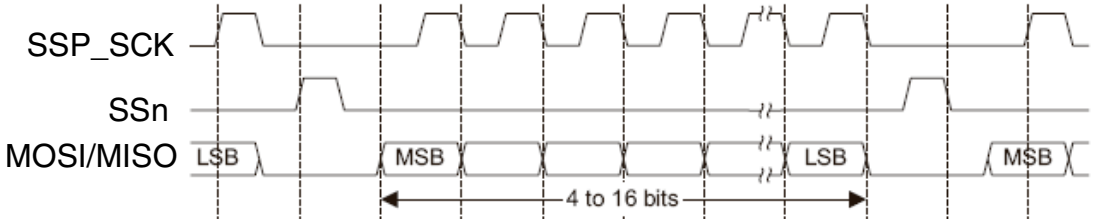


Figure 21-3. Motorola SPI Frame Format with POLARITY=0 and PHASE=0

In this configuration, during idle periods:

- The SSP_SCK signal is forced low.
- SSn is forced high.
- The Transmit data line MOSI is arbitrarily forced low.
- When the SSP is configured as a master, SSP_SCK is an output.
- When the SSP is configured as a slave, SSP_SCK is an input.

If the SSP is enabled and there is valid data within the FIFO, the start of the transmission is signified by the SSn master signal being low. This causes slave data to be enabled onto the MISO input line of the master, and the enables the master MOSI output pad.

One-half SSP_SCK period later, valid master data is transferred to the MOSI pin. Now that both the master and slave data have been set, the SSP_SCK master clock pin goes high after one further half SSP_SCK period.

The data is now captured on the rising and propagated on the falling edges of the SSP_SCK signal.

In the case of a single word transmission, after all bits of the data word have been transferred, the SSn line is returned to its idle high state one SSP_SCK period after the last bit has been captured.

However, in the case of continuous back-to-back transmissions, the SSn signal must be pulsed high between each data word transfer. This is because the slave select pin freezes the data in its serial peripheral register and does not allow it to be altered if the PHASE bit is logic 0. Therefore, the master device must raise the SSn pin of the slave device between each data transfer to enable the serial peripheral data write. On completion of the continuous transfer, the SSn pin is returned to its idle state one SSP_SCK period after the last bit has been captured.

21.5.4 Motorola SPI Format with Polarity=0, Phase=1

The transfer signal sequence for Motorola SPI format with POLARITY=0 and PHASE=1 is shown in Figure 21-4, which covers both single and continuous transfers.

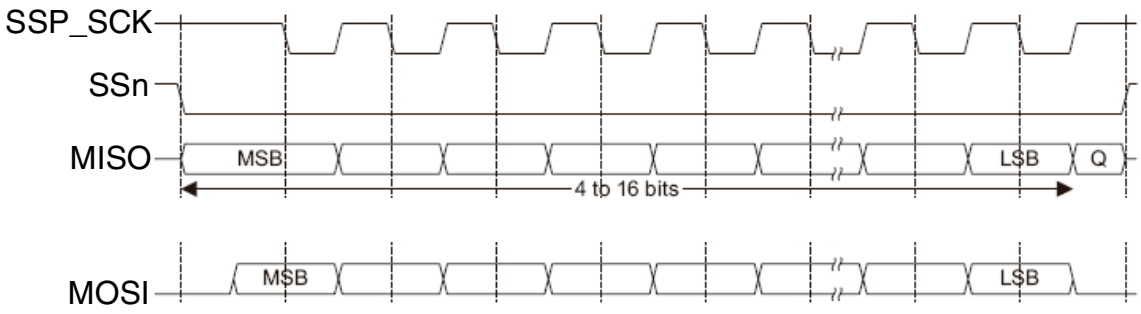


Figure 21-4. Motorola SPI Frame Format (Single Transfer) with POLARITY=1 and PHASE=0

In this configuration, during idle periods:

- The SSP_SCK signal is forced low.
- SSn is forced high.
- The Transmit data line MOSI is arbitrarily forced low.
- When the SSP is configured as a master, the SSP_SCK pad is an output.
- When the SSP is configured as a slave, the SSP_SCK is an input.

If the SSP is enabled and there is valid data within the FIFO, the start of the transmission is signified by the SSn master signal being low. After a further one-half SSP_SCK period, both master and slave valid data are enabled with a rising-edge transition.

Data is then captured on the falling edges and propagated on the rising edges of the SSP_SCK signal.

In the case of a single word transfer, after all bits have been transferred, the SSn line is returned to its idle high state one SSP_SCK period after the last bit has been captured.

For continuous back-to-back transfers, SSPFSOUT (the SSn pin in master mode) is held low between successive data words and termination is the same as that of a single word transfer.

21.5.5 Motorola SPI Format with Polarity=1, Phase=0

Single and continuous transmission signal sequences for Motorola SPI format with POLARITY=1 and PHASE=0 are shown in Figure 21-5 and Figure 21-6.

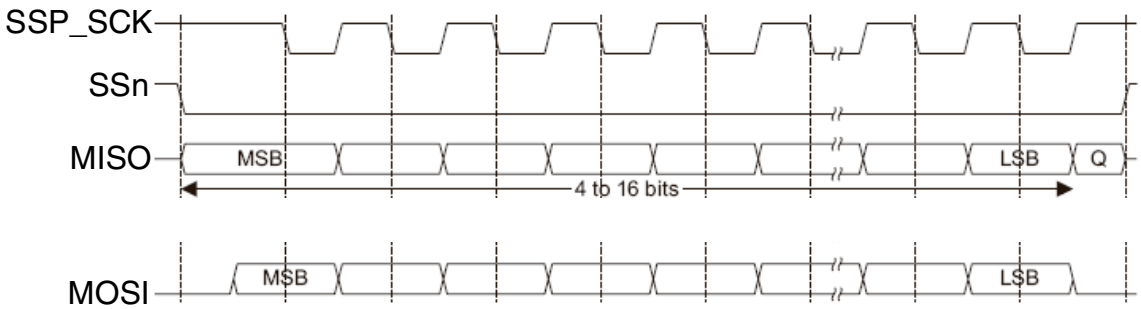


Figure 21-5. Motorola SPI Frame Format (Single Transfer) with POLARITY=1 and PHASE=0

Note: In Figure 21-5, Q is an undefined signal.

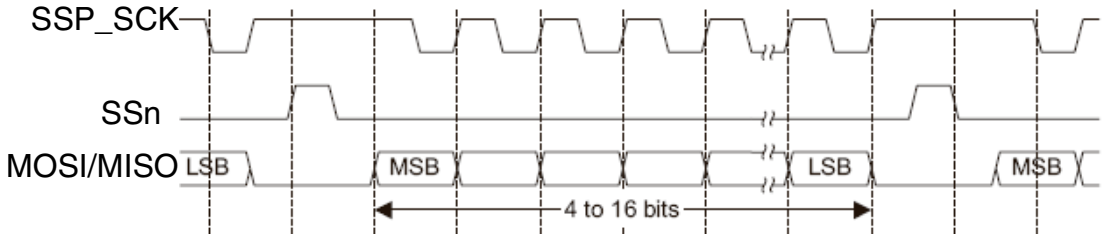


Figure 21-6. Motorola SPI Frame Format (Continuous Transfer) with POLARITY=1 and PHASE=0

In this configuration, during idle periods:

- The SSP_SCK signal is forced high.
- SSn is forced high.
- The Transmit data line MOSI is arbitrarily forced low.
- When the SSP is configured as a master, the SSP_SCK pad is an output.
- When the SSP is configured as a slave, the SSP_SCK is an input.

If the SSP is enabled and there is valid data within the FIFO, the start of transmission is signified by the SSn master signal being driven low, which causes slave data to be immediately transferred onto the MISO line of the master, and enabling the master MOSI output pad.

One half-period later, valid master data is transferred to the MOSI line. Now that both master and slave data have been set, the SSP_SCK master clock pin becomes low after one further half SSP_SCK period.

This means that data is captured on the falling edges and propagated on the rising edges of the SSP_SCK signal.

In the case of a single word transmission, after all bits of the data word are transferred, the SSn line is returned to its idle high state one SSP_SCK period after the last bit has been captured.

However, in the case of continuous back-to-back transmissions, the SSn signal must be pulsed high between each data word transfer. This is because the slave select pin freezes the data in its serial peripheral register and does not allow it to be altered if the PHASE bit is logic 0. Therefore, the master device must raise
SSPSFSSIN (the SSn pin in slave mode) of the slave device between each data transfer to enable the serial peripheral data write. On completion of the continuous transfer, the SSn pin is returned to its idle state one SSP_SCK period after the last bit has been captured.

21.5.6 Motorola SPI Format with Polarity=1, Phase=1

The transfer signal sequence for Motorola SPI format with POLARITY=1 and PHASE=1 is shown in Figure 21-7, which covers both single and continuous transfers.

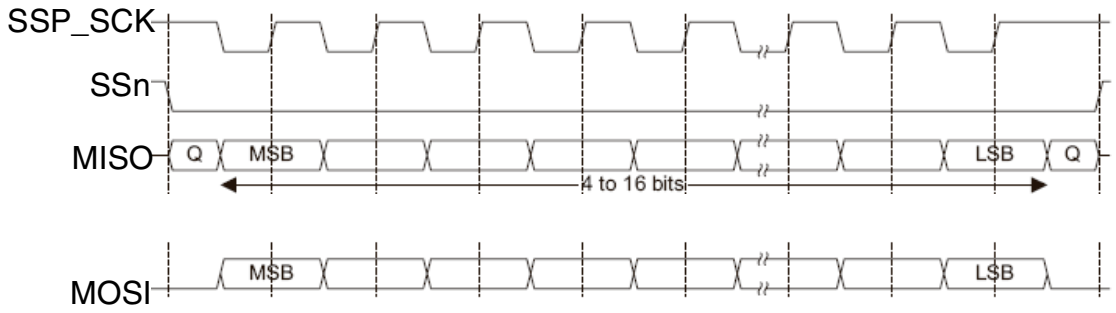


Figure 21-7. Motorola SPI Frame Format with POLARITY=1 and PHASE=1

Note: In Figure 21-7, Q is an undefined signal.

In this configuration, during idle periods:

- The SSP_SCK signal is forced high.
- SSn is forced high.
- The Transmit data line MOSI is arbitrarily forced low.
- When the SSP is configured as a master, the SSP_SCK pad is an output.
- When the SSP is configured as a slave, the SSP_SCK is an input.

If the SSP is enabled and there is valid data within the FIFO, the start of transmission is signified by the SSn master signal being driven low, and MOSI output is enabled. After a further one-half SSP_SCK period, both master and slave are enabled onto their respective transmission lines. At the same time, the SSP_SCK is enabled with a falling edge transition. Data is then captured on the rising edge and propagated on the falling edges of the SSP_SCK signal.

After all bits have been transferred, in the case of a single word transmission, the SSn line is returned to its idle high state one SSP_SCK period after the last bit has been captured.

For continuous back-to-back transmissions, the SSn pin remains in its active low state, until the final bit of the last word has been captured, and then returns to its idle state as described above.

For continuous back-to-back transfers, the SSn pin is held low between successive data words and termination is the same as that of the single word transfer.

21.6 Winbond SPI Mode

The Winbond SPI mode is similar to Motorola's SPI mode when POLARITY = PHASE, where data is sampled on the rising edge. In addition to serial 1-bit reads and writes, 2-bit (dual) and 4-bit (quad) reads are supported. Only 8-bit word length is supported for dual and quad read modes. See Figure 21-9. The numbers in the IO signal waveform correspond to the bit position in the byte being transferred.

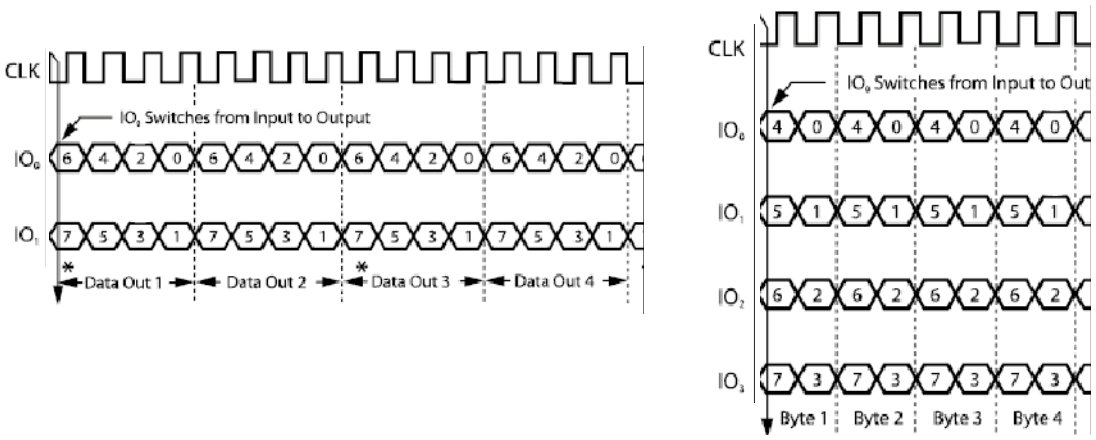


Figure 21-8. Fast Read Dual and Quad Output Diagram

21.7 Texas Instruments Synchronous Serial Interface (SSI) Mode

Figure 21-9 shows the Texas Instruments synchronous serial frame format for a single transmitted frame.

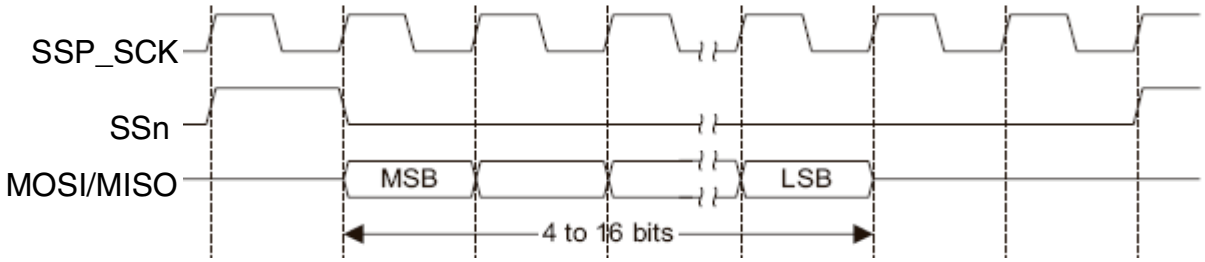


Figure 21-9. Texas Instruments Synchronous Serial Frame Format (Single Transfer)

In this mode, SSP_SCK and SSn are forced low, and the transmit data line MOSI is three-stated whenever the SSP is idle. Once the bottom entry of the FIFO contains data, SSn is pulsed high for one SSP_SCK period. The value to be transmitted is also transferred from the FIFO to the serial shift register of the transmit logic. On the next rising edge of SSP_SCK, the MSB of the 4-to-16-bit data frame is shifted out on the SSPRXD pin by the off-chip serial slave device.

Both the SSP and the off-chip serial slave device then clock each data bit into their serial shifter on the falling edge of each SSP_SCK. The received data is transferred from the serial shifter to the FIFO on the first rising edge of SSP_SCK after the LSB has been latched.

Figure 21-10 shows the Texas Instrument synchronous serial frame format when back-to-back frames are transmitted.

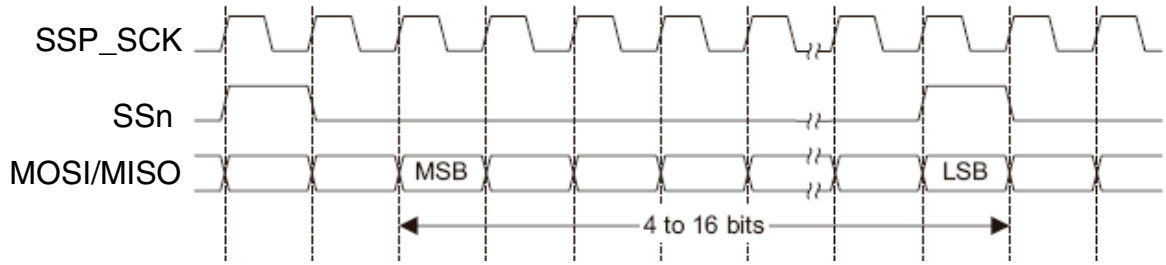


Figure 21-10. Texas Instruments Synchronous Serial Frame Format (Continuous Transfer)

21.8 SD/SDIO/MMC Mode

This mode is used to provide high performance with SD, SDIO, MMC, and high-speed (4-bit and 8-bit) MMC cards.

SD/MMC mode supports simultaneous command and data transfers. Commands are sent to the card and responses are returned to the host on the CMD line. Register data, such as card information, is sent as a

command response and is therefore on the CMD line. Block data read from or written to the card's flash is transferred on the DAT line(s). The SSP also supports the SDIO IRQ.

The SSP's SD/MMC controller can automatically perform a single block read/write or card register operation with a single PIO setup and RUN. For example, the SD/MMC controller can perform these steps with a single write to the PIO registers:

- Send command to the card.
- Receive response from the card.
- Check response for errors (and assert a CPU IRQ if there is an error).
- Wait for the DAT line(s) to be ready to transfer data (while counting for time-out)
- Transfer multiple blocks of data to/from the card.
- Check the CRC or CRC status of received/sent data (and assert IRQ if there is an error).

The SD/MMC controller is generally used with the DMA. Each DMA descriptor is set up the SD/MMC controller to perform a single complex operation as exemplified above. Multiple DMA descriptors can be chained to perform multiple card block transfers without CPU intervention. A single DMA descriptor can also perform multiple card block transfers.

21.8.1 SD/MMC Command/Response Transfer

SD/MMC commands are written to the HW_SSP_CMDn registers and sent on the CMD line. Command tokens consist of a start bit (0), a source bit (1), the actual command, which is padded to 38 bits, a 7-bit CRC and a stop bit (1). The command token format is shown in Table 21-2.

Table 21-2. SD/MMC Command/Response Transfer

Line	Start	Source	Data	CRC	End
CMD	0	1 (Host)	38-bit Command	CRC7	1

SD/MMC cards transmit command words with the most significant bit first. After the card receives the command, it checks for CRC errors or invalid commands. If an error occurs, the card withholds the usual response to the command.

After transmitting the end bit, the SSP releases the CMD line to the high-impedance state. A pullup resistor on the CMD node keeps it at the 1 state until the response packet is received. The slave waits to issue the reply until the SCK line is clocking again.

After the SSP sends an SD/MMC command, it optionally starts looking for a response from the card. It waits for the CMD line to go low, indicating the start of the response token. Once the SSP has received the Start and Source bits, it begins shifting the response content into the receive shift register. The SSP calculates the CRC7 of the incoming data.

If the card fails to start sending an expected response packet within 64 SCK cycles, then an error has occurred; the command may be invalid or have a bad CRC. After the SSP detects a time-out, it stops any

DMA request activity and sets the RESP_TIMEOUT flag. If RESP_TIMEOUT_IRQ_EN is set, then a CPU IRQ is asserted.

The SSP calculates the CRC of the received response and compare it to the CRC received from the card. If they do not match, then the SSP sets the RESP_ERR status flag. If RESP_ERR_IRQ_EN is set, then a CPU IRQ is asserted on a command response CRC mismatch.

The SSP can also compare the 32-bit card status word, known as response R1, against a reference to check for errors. If CHECK_RESP in HW_SSP_CTRL0 is set, then the SSP XORs the response with the XOR field in the HW_SSP_COMPREF register. It then masks the results with the MASK field in the HW_SSP_COMPMASK register. If there are any differences between the masked response and the reference, then an error has occurred. The CPU asserts the RESP_ERR status flag. If RESP_ERR_IRQ_EN is set, then the RESP_ERR_IRQ is asserted. In the ISR, the CPU can read the status word to see which error flags are set.

The regular and long response tokens are shown in Table 21-3 and Table 21-4:

Table 21-3. SD/MMC Command Regular Response Token

Line	Start	Source	Data	CRC	End
CMD	0	0 (Card)	38-bit Response	CRC7	1

Table 21-4. SD/MMC Command Regular Long Response Token

Line	Start	Source	Data	CRC	End
CMD	0	0 (Card)	117-bit response	CRC16	1

21.8.2 SD/MMC Data Block Transfer

Block data is transferred on the DATA0 pin. In 1-bit I/O mode, the block data is formatted as shown in Table 21-5. Block data transfers typically have 512 bytes of payload, plus a 16-bit CRC, a Start bit, and an End bit. The block size is programmable with the XFER_COUNT field in the HW_SSP_CTRL0 register. In SD/MMC mode, WORD_LENGTH in the HW_SSP_CTRL1 register field should always be set to 8 bits. Data is always sent Most Significant Bit of the Least Significant Byte first.

The SSP is designed to support block transfer modes only. Streaming modes may not be supported. Figure 21-11 shows a flowchart of SD/MMC block read and write transfers.

In block write mode, the card holds the DATA0 line low while it is busy. SSP must wait for the DATA0 line to be high for one clock cycle before starting to write a block.

In block read mode, the card begins sending the data when it is ready. The first bit transmitted by the card is a Start bit 0. Prior to the 0 Start Bit, the DATA0 bus is high. After the start bit is received, data is shifted in. The SSP bus width is set using the BUS_WIDTH bit in the HW_SSP_CTRL0 register.

In 1-bit bus mode, the block data is formatted as shown in Table 21-5.

Table 21-5. SD/MMC Data Block Transfer 1-Bit Bus Mode

Line	Start	Data	Data	Data	CRC	End
DATA0	0	Data Bit 7 Byte 0	...	Data Bit 0 Byte 511	CRC16	1

In 4-bit I/O mode, the block data is formatted as shown in Table 21-6.

Table 21-6. SD/MMC Data Block Transfer 4-Bit Bus Mode

Line	Start	Data	Data	Data	CRC	End
DATA3	0	Data Bit 7 Byte 0	...	Data Bit 3 Byte 511	CRC16	1
DATA2	0	Data Bit 6 Byte 0	...	Data Bit 2 Byte 511	CRC16	1
DATA1	0	Data Bit 5 Byte 0	...	Data Bit 1 Byte 511	CRC16	1
DATA0	0	Data Bit 4 Byte 0	...	Data Bit 0 Byte 511	CRC16	1

In 8-bit bus mode, the block data is formatted as shown in Table 21-7.

Table 21-7. SD/MMC Data Block Transfer 8-Bit Bus Mode

Line	Start	Data	Data	Data	CRC	End
DATA7	0	Data Bit 7 Byte 0	...	Data Bit 7 Byte 511	CRC16	1
DATA6	0	Data Bit 6 Byte 0	...	Data Bit 6 Byte 511	CRC16	1
DATA5	0	Data Bit 5 Byte 0	...	Data Bit 5 Byte 511	CRC16	1
DATA4	0	Data Bit 4 Byte 0	...	Data Bit 4 Byte 511	CRC16	1
DATA3	0	Data Bit 3 Byte 0	...	Data Bit 3 Byte 511	CRC16	1
DATA2	0	Data Bit 2Byte 0	...	Data Bit 2 Byte 511	CRC16	1
DATA1	0	Data Bit 1 Byte 0	...	Data Bit 1 Byte 511	CRC16	1
DATA0	0	Data Bit 0 Byte 0	...	Data Bit 0 Byte 511	CRC16	1

21.8.2.1 SD/MMC Multiple Block Transfers

The SSP supports SD/MMC multiple block transfers. The CPU or DMA will configure the SD/MMC controller to issue a Multi-Block Read or Write command. If DMA is used, then the first descriptor issues the multi-block read/write command and receives/sends the first block (512 bytes) of data. Subsequent DMA descriptors only receive/send blocks of data and do not issue new SD/MMC commands. If the card is configured for an open-ended multi-block transfer, then the last DMA descriptor needs to issue a STOP command to the card. Multiple blocks can also be transferred with a single DMA descriptor.

After each block of data has been transferred, the SSP sends/receives the CRC and checks the CRC or the CRC token. If the CRC is okay, then the SSP signals the DMA that it is done.

The SSP supports transferring multiple SD/MMC blocks per DMA descriptor. The SSP state machine needs to know the number of blocks and the size of a block. When HW_SSP_CMD0_BLOCK_COUNT is non-zero, the actual block size is:

0x1 shiftleft BLOCK_SIZE

For example, setting a value of 9 will result in a block size of 512 bytes. When BLOCK_COUNT is 0, BLOCK_SIZE is ignored and the bit field HW_SSP_CTRL0_XFER_COUNT represents the single block size or the number of byte to transfer. This must satisfy the equation:

HW_SSP_CTRL0_XFER_COUNT = (0x1 shiftleft BLOCK_SIZE) x (BLOCK_COUNT+1)

for BLOCK_COUNT greater than 0.

21.8.2.2 SD/MMC Block Transfer CRC Protection

All block data transferred over the data bus is protected by CRC16. For reads, the SSP calculates the CRC of incoming data and compares it to the CRC16 reference that is provided by the card at the end of the block. If a CRC mismatch occurs, then the block asserts the DATA_CRC_ERR status flag. If DATA_CRC_IRQ_EN is set, then a CPU IRQ is asserted.

For block write operations, the card determines if a CRC error has occurred. After the SSP has sent a block of data, it transmits the reference CRC16. The card compares that to its calculated CRC16. The card then sends a CRC status token on the DATA bus. It sends a positive status (‘010’) if the transfer was good, and a negative status (‘101’) if the CRC16 did not match. If the SSP receives a CRC bad token, it sets the DATA_CRC_ERROR in the HW_SSP_STATUS register, and then it indicates it to the CPU if DATA_CRC_IRQ_EN is set.

21.8.3 SDIO Interrupts

The SSP supports SDIO interrupts. When the SSP is in SD/MMC mode and the SDIO_IRQ bit in the HW_SSP_CTRL0 register is set, the SSP looks for interrupts on DATA1 during the valid IRQ periods. The valid IRQ periods are defined in the SDIO specification. If the card asserts an interrupt and SDIO_IRQ_EN is set, then the SSP sets the SDIO_IRQ status bit and asserts a CPU IRQ. Other than detecting when card IRQs are valid, the SDIO IRQ function operates independently from the rest of the SSP. After the CPU receives an IRQ, it should monitor the SSP and DMA status to determine when it should send commands to the SDIO card to handle the interrupt.

21.8.4 SD/MMC Mode Error Handling

There are several errors that can occur during SD/MMC operation. These errors can be caused by normal unexpected events, such as having a card removed or unusual events such as a card failure. The detected error cases are listed below. Please note that in all cases below, a CPU IRQ is only asserted if DATA_CRC_IRQ_EN is set in HW_SSP_CTRL1 register.

- **Data Receive CRC Error**—Detected by the SSP after a block receive. If this occurs, the SSP will not indicate to the DMA that the transfer is complete. It will set the DATA_CRC_ERR status flag and assert a CPU IRQ. The ISR should reset the SSP DMA channel and instruct the DMA to re-try the block read operation.

- **Data Transmit CRC Error**—Transmit CRC error token is received from the SD/MMC card on the DAT line after a block transmit. If this occurs, the SSP will not indicate to the DMA that the transfer is complete. It will set the DATA_CRC_ERR status flag and assert a CPU IRQ. The ISR should reset the SSP DMA channel and instruct the DMA to re-try the block write operation.

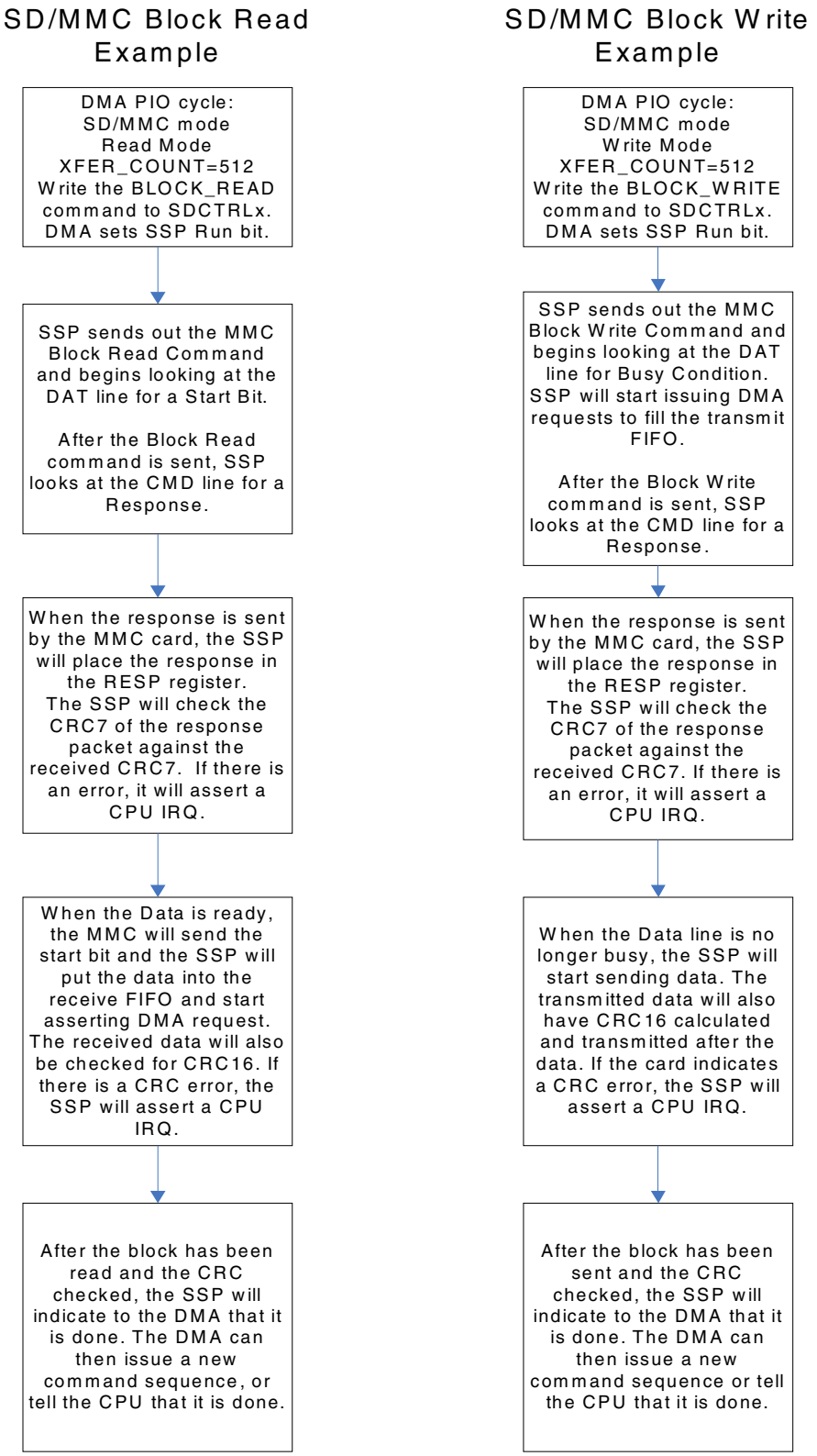


Figure 21-11. SD/MMC Block Transfer Flowchart

- **Data Time-Out Error**—The SSP TIMEOUT counter is used to detect a time-out condition during data write or read operations. The time-out counts any time that the SSP is waiting on a busy DAT bus. For read operations, the DAT line(s) indicate busy before the card sends the start bit. For write operations, the DAT line(s) may indicate busy after the block has been sent to the card. If the time-out counter expires before the DAT line(s) become ready, the SSP stops any DMA requests, sets the DATA_TIMEOUT status flag, and asserts a CPU IRQ. The ISR should check the status register to see that a data time-out has occurred. It can then reset the DMA channel and SSP to re-try the operation.
- **DMA Overflow/Underflow**—The SSP should stop SCK if the FIFO is full or the FIFO is empty during data transfer. So, a DMA underflow or overflow should not occur. However, if it does due to some unforeseen problem, the FIFO_OVRFLW or FIFO_UNDRFLW status bit is set in the SSP Status Register and asserts a CPU IRQ.
- **Command Response Error**—The SD/MMC card returns an R1 status response after most commands. The SSP can compare the R1 response against a mask/reference pair. If any of the enabled bits are set, then an error has occurred. The SSP stops requesting any DMAs, sets the RESP_ERR status flag, and asserts a CPU IRQ. The CPU can read the SSP Status Register to see the RESP_ERR flag and read the HW_SSP_SDRESP0 register to get the actual response from the SD/MMC card. That response contains the specific error information. Once the error is understood, the CPU can reset the DMA channel and SSP and re-try the operation or take some other action to recover or inform the user of a non-recoverable error.
- **Command Response Time-Out**—If an expected response is not received within 64 SCK cycles, then the command response has timed out. If this occurs, the SSP stops any DMA requests, stops transferring data to the card, sets the RESP_TIME-OUT status flag, and asserts the RESP_TIME-OUT_IRQ. The ISR should read the status register to find that a command response time-out has occurred. It can then decide to reset the DMA channel and SSP and re-try the operation.

21.8.5 SD/MMC Clock Control

- When SD/MMC block is idle, the serial clock (SCK) toggling will be based on the value of CONT_CLKING_EN and SLOW_CLKING_EN. See HW_SSP_CMD0 register description.
- SCK runs any time that RUN is set and a data or command is active or pending. If a command has been sent and a response is expected, then SCK continues to run until the response is received. If a data operation is active or if the DAT line is busy, then SCK runs.
- If CONT_CLKING_EN=0, SCK stops running if received command response status R1 indicates an error.
- If CONT_CLKING_EN=0, SCK stops running if a data operation has timed out or a CRC error has occurred.
- If CONT_CLKING_EN=0, SCK stops running after all pending commands and data operations have completed. SCK restarts when a new command or data operation has been requested.

21.9 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, "Correct Way to Soft Reset a Block," for additional information on using the SFTRST and CLKGATE bit fields.

21.10 Programmable Registers

The following registers provide control for programmable elements of the SSP1 port. An identical set of registers exists for the SSP2 port, beginning at 0x8003_4000. Note that the register descriptions for SSP2 are not duplicated in this section.

21.10.1 SSP Control Register 0 Description

SSP Control Register 0

HW_SSP_CTRL0	0x000
HW_SSP_CTRL0_SET	0x004
HW_SSP_CTRL0_CLR	0x008
HW_SSP_CTRL0_TOG	0x00C

Table 21-8. HW_SSP_CTRL0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	RUN	SDIO_IRQ_CHECK	LOCK_CS	IGNORE_CRC	READ	DATA_XFER	BUS_WIDTH		WAIT_FOR_IRQ	WAIT_FOR_CMD	LONG_RESP	CHECK_RESP	GET_RESP	ENABLE	XFER_COUNT																	

Table 21-9. HW_SSP_CTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	SSP Reset - 0: SSP is not Reset. 1: SSP Held in Reset. After Reset, all registers are returned to their reset state. This will not work if the CLKGATE bit is already set to '1'. CLKGATE must be cleared to '0' before issuing a soft reset. Also the SSPCLK must be running for this to work properly.
30	**CLKGATE**	RW	0x1	Gate SSP Clocks - 0: SSP Clocks not gated. 1: SSP Clocks are gated. Set this to save power while the SSP is not actively being used. Configuration state is kept while the clock is gated.

Table 21-9. HW_SSP_CTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
29	**RUN**	RW	0x0	SSP Run. 0: SSP is not running. 1: SSP is running. Automatically set during DMA operation.
28	**SDIO_IRQ_CHECK**	RW	0x0	In SD/MMC mode: SDIO IRQ: 1= Enable checking for SDIO Card IRQ. In CE_ATA mode: 1= Wait for CCS (Command Completion Signal). 0= ignore CCS. In CE_ATA mode this bit must not be set to 1 if LOCK_CS (Disable CCS) is set to 1; i.e. both bit cannot be set simultaneously.
27	**LOCK_CS**	RW	0x0	In SPI mode: This affects the SSn output. When set to 1, SSn will be asserted throughout the current command. SSn will remain asserted after the command if IGNORE_CRC = 0. SSn will deassert at the end of the next command that has IGNORE_CRC = 1. In SD/MMC mode: 0= Look for a CRC status token from the card on DATA0 after a block write. 1= Ignore the CRC status response on DATA0 after a write operation. Note that the SD/MMC function should be used when performing MMC BUSTEST_W operation. In CE_ATA mode: 1= Disable/Cancel CCS (Command Completion Signal) from card. In CE_ATA mode, if this bit is set then the DATA_XFER bit must also be set.
26	**IGNORE_CRC**	RW	0x0	Ignore CRC - In SD/MMC mode: Ignores the Response CRC when set to 1. In SPI/SSI modes: When set to 1, deassert the chip select (SSn) pin after the command is executed.
25	**READ**	RW	0x0	Read Mode - When this and DATA_XFER are set, the SSP will read data from the device. If this is not set, then the SSP will write data to the device.
24	**DATA_XFER**	RW	0x0	Data Transfer Mode - When set, transfer XFER_COUNT bytes of data. When not set, the SSP will not transfer any data (command or Wait for IRQ only).
23:22	**BUS_WIDTH**	RW	0x0	Data Bus Width - SD/MMC mode supports all widths. SSI mode supports only 1-bit bus width. In SPI mode 1, 2, and 4-bit bus widths are supported. In SPI mode the Data Bus Width field is redefined: 0 means 1-bit; 1 means 2-bit; 2 means 4-bit. ONE_BIT = 0x0 SD/MMC data bus is 1-bit wide FOUR_BIT = 0x1 SD/MMC data bus is 4-bits wide EIGHT_BIT = 0x2 SD/MMC data bus is 8-bits wide
21	**WAIT_FOR_IRQ**	RW	0x0	In SD/MMC mode this signal means wait for MMC ready before sending command. (MMC is busy when databit0 is low.) In SPI/SSI mode this bit and WAIT_FOR_CMD bit select which SSn output to assert: (WAIT_FOR_IRQ,WAIT_FOR_CMD) = b00: SSn0 will assert during access (SSn2, SSn1 inactive); b01: SSn1 will assert during access (SSn2, SSn0 inactive); b10: SSn2 will assert during access (SSn1, SSn0 inactive).

Table 21-9. HW_SSP_CTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
20	**WAIT_FOR_CMD**	RW	0x0	Wait for Data Done - SD/MMC - 0: Send commands immediately after they are written. 1: Wait to send command until after the CRC-checking phase of a data transfer has completed successfully. This delays sending a command until a block of data is transferred. This can be used to send a STOP command during an SD/MMC multi-block read. In SD/MMC mode this signal means wait for MMC ready before sending command. (MMC is busy when databit0 is low.) In SPI/SSI mode this bit and WAIT_FOR_IRQ bit select which SSn output to assert: (WAIT_FOR_IRQ,WAIT_FOR_CMD) = b00: SSn0 will assert during access (SSn2, SSn1 inactive); b01: SSn1 will assert during access (SSn2, SSn0 inactive); b10: SSn2 will assert during access (SSn1, SSn0 inactive).
19	**LONG_RESP**	RW	0x0	Get Long Response - SD/MMC - 0: The card response will be short. 1: The card will provide a 136-bit response. Only valid if GET_RESP is set. A long response cannot be checked using CHECK_RESP.
18	**CHECK_RESP**	RW	0x0	Check Response - SD/MMC. If this bit is set, the SSP will XOR the result with the REFERENCE field and then mask the incoming status word with the MASK field in the COMPARE register. If there is a mismatch, then the SSP will set the RESP_ERR status bit, and, if enabled, the RESP_ERR_IRQ. This should not be used with LONG_RESP.
17	**GET_RESP**	RW	0x0	Get Response - SD/MMC - 0: Do not wait for a response from the card. 1: This command should receive a response from the card.
16	**ENABLE**	RW	0x0	Command Transmit Enable - SD/MMC - 0: Commands are not enabled. 1: Data in Command registers will be sent. This is normally enabled in SD/MMC Mode.
15:0	**XFER_COUNT**	RW	0x1	Number of words to transfer, as referenced in WORD_LENGTH in HW_SSP_CTRL1. The run bit and DMA request will clear after this many words have been transferred. In SD/MMC Mode, this should be a multiple of the block size.

DESCRIPTION:

This is the most frequently changed fields.

EXAMPLE:

```
No Example.
```

21.10.2 SD/MMC Command Register 0 Description

SD/MMC Command Index and control register

HW_SSP_CMD0	0x010
HW_SSP_CMD0_SET	0x014

HW_SSP_CMD0_CLR 0x018
HW_SSP_CMD0_TOG 0x01C

Table 21-10. HW_SSP_CMD0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD0									SLOW_CLKING_EN	CONT_CLKING_EN	APPEND_8CYC	BLOCK_SIZE				BLOCK_COUNT								CMD							

Table 21-11. HW_SSP_CMD0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:23	**RSVD0**	RO	0x0	Reserved
22	**SLOW_CLKING_EN**	RW	0x0	Enable Continuous clocking on SCK to occur at a frequency eight times slower that when actively transferring command and data. This field is ignored when CONT_CLKING is zero.
21	**CONT_CLKING_EN**	RW	0x0	Set this bit to enable Continous clocking of SCK when no SD/MMC/CE_ATA command/reaponse or data transfer is active. This is used in SD/MMC/CE_ATA mode. When set to zero, SCK is idle when no transfer is taking place.
20	**APPEND_8CYC**	RW	0x0	Append 8 SCK cycles. This is used in SD/MMC/CE_ATA mode. When set to one, the SCK will toggle for 8 more cycles before going idle. When set to zero SCK will toggle up to 4 cycles before going idle. This should be set to one at the end of a single or multiple block transfer.
19:16	**BLOCK_SIZE**	RW	0x0	SD/MMC block size encode. When BLOCK_COUNT is nonzero, the actual block size is (1 shiftleft BLOCK_SIZE). For example setting a value of 9 will result in a block size of 512 bytes. When BLOCK_COUNT is zero, BLOCK_SIZE is ignored and HW_SSP_CTRL0_XFER_COUNT represents the single block size or the number of byte to transfer. This must satisfy equation HW_SSP_CTRL0_XFER_COUNT = (1 shiftleft BLOCK_SIZE) x (BLOCK_COUNT+1) for BLOCK_COUNT greater than zero.

Table 21-11. HW_SSP_CMD0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:8	**BLOCK_COUNT**	RW	0x0	SD/MMC block count. This value one less than the number of blocks to transfer. For example setting a value of one will transfer two blocks. This must satisfy equation HW_SSP_CTRL0_XFER_COUNT = (1 shiftleft BLOCK_SIZE) x (BLOCK_COUNT+1) for BLOCK_COUNT greater than zero. This is also SPI/SSI control word[15:8] for RX transfers.

Table 21-11. HW_SSP_CMD0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
7:0	**CMD**	RW	0x0	SD/MMC Command Index (uses 5:0) to be sent to card. This is also SPI/SSI control word[7:0] for RX transfers. MMC_GO_IDLE_STATE = 0x00 MMC_SEND_OP_COND = 0x01 MMC_ALL_SEND_CID = 0x02 MMC_SET_RELATIVE_ADDR = 0x03 MMC_SET_DSR = 0x04 MMC_RESERVED_5 = 0x05 MMC_SWITCH = 0x06 MMC_SELECT_DESELECT_CARD = 0x07 MMC_SEND_EXT_CSD = 0x08 MMC_SEND_CSD = 0x09 MMC_SEND_CID = 0x0A MMC_READ_DAT_UNTIL_STOP = 0x0B MMC_STOP_TRANSMISSION = 0x0C MMC_SEND_STATUS = 0x0D MMC_BUSTEST_R = 0x0E MMC_GO_INACTIVE_STATE = 0x0F MMC_SET_BLOCKLEN = 0x10 MMC_READ_SINGLE_BLOCK = 0x11 MMC_READ_MULTIPLE_BLOCK = 0x12 MMC_BUSTEST_W = 0x13 MMC_WRITE_DAT_UNTIL_STOP = 0x14 MMC_SET_BLOCK_COUNT = 0x17 MMC_WRITE_BLOCK = 0x18 MMC_WRITE_MULTIPLE_BLOCK = 0x19 MMC_PROGRAM_CID = 0x1A MMC_PROGRAM_CSD = 0x1B MMC_SET_WRITE_PROT = 0x1C MMC_CLR_WRITE_PROT = 0x1D MMC_SEND_WRITE_PROT = 0x1E MMC_ERASE_GROUP_START = 0x23 MMC_ERASE_GROUP_END = 0x24 MMC_ERASE = 0x26 MMC_FAST_IO = 0x27 MMC_GO_IRQ_STATE = 0x28 MMC_LOCK_UNLOCK = 0x2A MMC_APP_CMD = 0x37 MMC_GEN_CMD = 0x38 SD_GO_IDLE_STATE = 0x00 SD_ALL_SEND_CID = 0x02 SD_SEND_RELATIVE_ADDR = 0x03 SD_SET_DSR = 0x04 SD_IO_SEND_OP_COND = 0x05 SD_SELECT_DESELECT_CARD = 0x07 SD_SEND_CSD = 0x09 SD_SEND_CID = 0x0A SD_STOP_TRANSMISSION = 0x0C SD_SEND_STATUS = 0x0D SD_GO_INACTIVE_STATE = 0x0F SD_SET_BLOCKLEN = 0x10 SD_READ_SINGLE_BLOCK = 0x11 SD_READ_MULTIPLE_BLOCK = 0x12 SD_WRITE_BLOCK = 0x18 SD_WRITE_MULTIPLE_BLOCK = 0x19 SD_PROGRAM_CSD = 0x1B SD_SET_WRITE_PROT = 0x1C SD_CLR_WRITE_PROT = 0x1D SD_SEND_WRITE_PROT = 0x1E SD_ERASE_WR_BLK_START = 0x20 SD_ERASE_WR_BLK_END = 0x21 SD_ERASE_GROUP_START = 0x23 SD_ERASE_GROUP_END = 0x24 SD_ERASE = 0x26 SD_LOCK_UNLOCK = 0x2A SD_IO_RW_DIRECT = 0x34 SD_IO_RW_EXTENDED = 0x35 SD_APP_CMD = 0x37 SD_GEN_CMD = 0x38

DESCRIPTION:

This is the command code for the SD/MMC devices. See device specification for details.

EXAMPLE:

```
No Example.
```

21.10.3 SD/MMC Command Register 1 Description

SD/MMC Command Argument Register

HW_SSP_CMD1 0x020

Table 21-12. HW_SSP_CMD1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ARG																															

Table 21-13. HW_SSP_CMD1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ARG**	RW	0x0	SD/MMC Command Argument. See SSP_CTRL0_DATA_XFER bit description.

DESCRIPTION:

This is the SD/MMC argument. See SD/MMC specification for details.

EXAMPLE:

```
No Example.
```

21.10.4 SD/MMC Compare Reference Description

MMC/SD Status response reference.

HW_SSP_COMPREF 0x030

Table 21-14. HW_SSP_COMPREF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REFERENCE																															

Table 21-15. HW_SSP_COMPREF Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**REFERENCE**	RW	0x0	SD/MMC Compare mode reference. If CHECK_RESP is set, the response will be XOR'd with this value. The results will be masked by the MASK bitfield. If there are any differences, then the SSP will indicate an error state to the DMA.

DESCRIPTION:

SD/MMC status can be compared with a reference value.

EXAMPLE:

```
No Example.
```

21.10.5 SD/MMC compare mask Description

MMC/SD Status response mask .

HW_SSP_COMPMASK 0x040

Table 21-16. HW_SSP_COMPMASK

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MASK																															

Table 21-17. HW_SSP_COMPMASK Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**MASK**	RW	0x0	SD/MMC Compare mode Mask. If CHECK_RESP is set, the response is compared to REFERENCE, and the results are masked by this bitfield.

DESCRIPTION:

A mask allows the comparison of one or more bit fields in the reference value.

EXAMPLE:

```
No Example.
```

21.10.6 SSP Timing Register Description

SSP Timing Config Register

HW_SSP_TIMING 0x050

Table 21-18. HW_SSP_TIMING

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TIMEOUT																CLOCK_DIVIDE								CLOCK_RATE							

Table 21-19. HW_SSP_TIMING Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**TIMEOUT**	RW	0x0	Timeout counter. This specifies the number of SCK cycles multiplied by 4096, to wait before asserting the DATA TIMEOUT IRQ. It is used during timeout is used for data transfer/write operations in SD/MMC mode.
15:8	**CLOCK_DIVIDE**	RW	0x0	Clock Pre-Divider. CLOCK_DIVIDE must be an even value from 2 to 254.
7:0	**CLOCK_RATE**	RW	0x0	Serial Clock Rate. The value CLOCK_RATE is used to generate the transmit and receive bit rate of the SSP. The bit rate is SSPCLK / (CLOCK_DIVIDE x (1+ CLOCK_RATE)). CLOCK_RATE is a value from 0 to 255.

DESCRIPTION:

The Timeout field and the clock dividers are contained in this register.

EXAMPLE:

```
No Example.
```

21.10.7 SSP Control Register 1 Description

Control Register 1.

HW_SSP_CTRL1	0x060
HW_SSP_CTRL1_SET	0x064
HW_SSP_CTRL1_CLR	0x068
HW_SSP_CTRL1_TOG	0x06C

Table 21-20. HW_SSP_CTRL1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SDIO_IRQ	SDIO_IRQ_EN	RESP_ERR_IRQ	RESP_ERR_IRQ_EN	RESP_TIMEOUT_IRQ	RESP_TIMEOUT_IRQ_EN	DATA_TIMEOUT_IRQ	DATA_TIMEOUT_IRQ_EN	DATA_CRC_IRQ	DATA_CRC_IRQ_EN	FIFO_UNDERRUN_IRQ	FIFO_UNDERRUN_EN	RSVD3	RSVD2	RECV_TIMEOUT_IRQ	RECV_TIMEOUT_IRQ_EN	FIFO_OVERRUN_IRQ	FIFO_OVERRUN_IRQ_EN	DMA_ENABLE	RSVD1	SLAVE_OUT_DISABLE	PHASE	POLARITY	SLAVE_MODE	WORD_LENGTH				SSP_MODE			

Table 21-21. HW_SSP_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SDIO_IRQ**	RW	0x0	If this is set, an SDIO card interrupt has occurred and an IRQ, if enabled, has been sent to the ICOLL. Write a one to the SCT Clear address to reset this interrupt request status bit.
30	**SDIO_IRQ_EN**	RW	0x0	SDIO Card Interrupt IRQ Enable. 0: SDIO card IRQs masked. 1: SDIO card IRQs will be sent to the ICOLL.

Table 21-21. HW_SSP_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
29	**RESP_ERR_IRQ**	RW	0x0	When the CHECK_RESP bit in CTRL0 is set, if an unexpected response (or response CRC) is received from the card, this bit will be set. Write a one to the SCT Clear address to reset this interrupt request status bit.
28	**RESP_ERR_IRQ_EN**	RW	0x0	SD/MMC Card Error IRQ Enable. 0: Card Error IRQ is Masked. 1: Card Error IRQ is enabled. When set to 1, if an SD/MMC card indicates a card error (bit is set in both the SD/MMC Error Mask and R1 Card Status response), then a CPU IRQ will be asserted.
27	**RESP_TIMEOUT_IRQ**	RW	0x0	If this is set, a command response timeout has occurred, and an IRQ, if enabled, has been sent to the IRQ Collector. This is used for SD/MMC response timeout. Write a one to the SCT Clear address to reset this interrupt request status bit.
26	**RESP_TIMEOUT_IRQ_EN**	RW	0x0	SD/MMC Card Command Respone Timeout Error IRQ Enable. 0: Response Timeout IRQ is Masked. 1: Response Timeout IRQ is enabled. When set to 1, if an SD/MMC card does not respond to a command within 64 cycles then this CPU IRQ will be asserted.
25	**DATA_TIMEOUT_IRQ**	RW	0x0	Data Transmit/Receive Timeout Error IRQ. If the timeout counter expires before the DAT bus is ready for write or sends read data, then a data timeout has occurred. Only Valid For SD/MMC Mode. Write a one to the SCT Clear address to reset this interrupt request status bit.
24	**DATA_TIMEOUT_IRQ_EN**	RW	0x0	Data Transmit/Receive Timeout Error IRQ Enable. If the timeout counter expires before the DAT bus is ready for write or sends read data, then a data timeout has occurred. Only Valid For SD/MMC Mode.
23	**DATA_CRC_IRQ**	RW	0x0	Data Transmit/Receive CRC Error IRQ. Only valid for SD/MMC Mode. Write a one to the SCT Clear address to reset this interrupt request status bit.
22	**DATA_CRC_IRQ_EN**	RW	0x0	Data Transmit/Receive CRC Error IRQ Enable. Only valid for SD/MMC Mode.
21	**FIFO_UNDERRUN_IRQ**	RW	0x0	FIFO Underrun Interrupt. If the FIFO is read when it is empty this bit will be set. Write a one to the SCT Clear address to reset this interrupt request status bit.
20	**FIFO_UNDERRUN_EN**	RW	0x0	FIFO Underrun IRQ Enable. If set and the FIFO_UNDERRUN_IRQ bit is asserted, an IRQ will be generated.
19	**RSVD3**	RW	0x0	Program this field to 0x0.
18	**RSVD2**	RW	0x0	Program this field to 0x0.
17	**RECV_TIMEOUT_IRQ**	RW	0x0	Data Timeout Interrupt. If enabled and the FIFO is not empty, an IRQ will be generated if 128 HCLK Cycles Pass before the DATA register is read. This is supported for SPI modes only (Modes 0,1,2). Write a one to the SCT Clear address to reset this interrupt request status bit.
16	**RECV_TIMEOUT_IRQ_EN**	RW	0x0	Receive Timeout. If set and the FIFO is not empty, an IRQ will be generated if 128 HCLK Cycles Pass before the DATA register is read.

Table 21-21. HW_SSP_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15	**FIFO_OVERRUN_IRQ**	RW	0x0	FIFO Overrun Interrupt. Indicated that the FIFO has been written to while full. Write a one to the SCT Clear address to reset this interrupt request status bit.
14	**FIFO_OVERRUN_IRQ_EN**	RW	0x0	FIFO Overrun Interrupt Enable. If set, an IRQ will be generated if the FIFO is written to while full.
13	**DMA_ENABLE**	RW	0x0	DMA Enable. This signal enables DMA request and DMA Command End signals to be asserted.
12	**RSVD1**	RW	0x0	Program this field to 0x0.
11	**SLAVE_OUT_DISABLE**	RW	0x0	Slave Output Disable. 0: SSP can drive MISO in Slave Mode. 1: SSP does not drive MISO in slave mode.
10	**PHASE**	RW	0x0	Serial Clock Phase. For SPI mode only.
9	**POLARITY**	RW	0x0	Serial Clock Polarity. In SD/MMC mode, 0: Command and TX data change after rising edge of SCK, 1: Command and TX data change after falling edge of SCK. In SPI mode, 0: Steady-state '0' on SCK when data is not being transferred. 1: Steady-state '1' on SCK when data is not being transferred.
8	**SLAVE_MODE**	RW	0x0	Slave Mode. 0: SSP is in Master Mode. 1: SSP is in Slave Mode. Set to zero for SD/MMC mode.
7:4	**WORD_LENGTH**	RW	0x8	Word Length in bits per word. 0x0 to 0x2 are Reserved and Undefined. 0x3 is 4-bits per word...0xF is 16-bits per word. Always use 8 bits per word in SD/MMC Mode. RESERVED0 = 0x0 0x0 is Reserved and Undefined RESERVED1 = 0x1 0x1 is Reserved and Undefined RESERVED2 = 0x2 0x2 is Reserved and Undefined FOUR_BITS = 0x3 use 4-bits per word EIGHT_BITS = 0x7 use 8-bits per word SIXTEEN_BITS = 0xF use 16-bits per word
3:0	**SSP_MODE**	RW	0x0	Operating Mode. 0x0 = Motorola and Winbond SPI Mode, 0x1 = TI syncronous serial mode, 0x3 = SD/MMC Card, All other values are undefined. Before changing ssp_mode, a softreset must be issued to clear the FIFO. SPI = 0x0 Motorola and Winbond SPI mode SSI = 0x1 Texas Instruments SSI mode SD_MMC = 0x3 SD/MMC mode

DESCRIPTION:

This contains interrupt status and enable fields.

EXAMPLE:

```
No Example.
```

21.10.8 SSP Data Register Description

The HW_SSP_DATA register allows user access to the SSP FIFO.

HW_SSP_DATA 0x070

Table 21-22. HW_SSP_DATA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DATA																															

Table 21-23. HW_SSP_DATA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**DATA**	RW	0x0	Data Register. Holds one, two, three, or four words, depending on the WORD_LENGTH. If WORD_LENGTH is not 8, 16, or 32, the words are padded to those lengths. Data is right justified. When the run bit is set reads will cause the FIFO read pointer to increment and writes will cause the FIFO write pointer to increment.

DESCRIPTION:

This register is the gateway for data transfer to and from the attached device.

EXAMPLE:

```
No Example.
```

21.10.9 SD/MMC Card Response Register 0 Description

SD/SDIO/MMC Card Response Register 0.

HW_SSP_SDRESP0 0x080

Table 21-24. HW_SSP_SDRESP0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RESP0																															

Table 21-25. HW_SSP_SDRESP0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**RESP0**	RO	0x0	SD/MMC Response Status Bits[31:0]. See SSP_CTRL0_DATA_XFER bit description.

DESCRIPTION:

SD/SDIO/MMC Card Response Register 0.

EXAMPLE:

```
No Example.
```

21.10.10 SD/MMC Card Response Register 1 Description

SD/SDIO/MMC Card Response Register 1.

HW_SSP_SDRESP1 0x090

Table 21-26. HW_SSP_SDRESP1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RESP1																															

Table 21-27. HW_SSP_SDRESP1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**RESP1**	RO	0x0	SD/MMC Long Response [63:32]

DESCRIPTION:

SD/SDIO/MMC/CE_ATA Card Response Register 1.

EXAMPLE:

```
No Example.
```

21.10.11 SD/MMC Card Response Register 2 Description

SD/SDIO/MMC/CE_ATA Card Response Register 2.

HW_SSP_SDRESP2 0x0A0

Table 21-28. HW_SSP_SDRESP2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RESP2																															

Table 21-29. HW_SSP_SDRESP2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**RESP2**	RO	0x0	SD/MMC Long Response [95:64]

DESCRIPTION:

SD/SDIO/MMC/CE_ATA Card Response Register 2.

EXAMPLE:

```
No Example.
```

21.10.12 SD/MMC Card Response Register 3 Description

SD/SDIO/MMC/CE_ATA Card Response Register 3.

HW_SSP_SDRESP3 0x0B0

Table 21-30. HW_SSP_SDRESP3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RESP3																															

Table 21-31. HW_SSP_SDRESP3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**RESP3**	RO	0x0	SD/MMC Long Response [127:96]

DESCRIPTION:

SD/SDIO/MMC/CE_ATA Card Response Register 3.

EXAMPLE:

```
No Example.
```

21.10.13 SSP Status Register Description

SSP Read Only Status Registers.

HW_SSP_STATUS 0x0C0

Table 21-32. HW_SSP_STATUS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PRESENT	RSVD5	SD_PRESENT	CARD_DETECT	RSVD4						DMASENSE	DMATERM	DMAREQ	DMAEND	SDIO_IRQ	RESP_CRC_ERR	RESP_ERR	RESP_TIMEOUT	DATA_CRC_ERR	TIMEOUT	RECV_TIMEOUT_STAT	RSVD3	FIFO_OVRFLW	FIFO_FULL	RSVD2		FIFO_EMPTY	FIFO_UNDRFLW	CMD_BUSY	DATA_BUSY	RSVD1	BUSY

Table 21-33. HW_SSP_STATUS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**PRESENT**	RO	0x1	SSP Present Bit. 0: SSP is not present in this product. 1: SSP is present.
30	**RSVD5**	RO	0x1	Program this field to 0x1.
29	**SD_PRESENT**	RO	0x1	SD/MMC Controller Present bit. 0: SD/MMC controller is not present in this product. 1: SD/MMC controller is present.
28	**CARD_DETECT**	RO	0x0	Reflects the state of the SSP_DETECT input pin.
27:22	**RSVD4**	RO	0x0	Reserved
21	**DMASENSE**	RO	0x0	Reflects the state of the ssp_dmasense output port. It indicates a DMA error (Timeout or CRC) when asserted high at the end of a DMA command.

Table 21-33. HW_SSP_STATUS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
20	**DMATERM**	RO	0x0	Reflects the state of the ssp_dmaterm output port. This is a toggle signal.
19	**DMAREQ**	RO	0x0	Reflects the state of the ssp_dmareq output port. This is a toggle signal.
18	**DMAEND**	RO	0x0	Reflects the state of the ssp_dmaend output port. This is a toggle signal.
17	**SDIO_IRQ**	RO	0x0	SDIO IRQ has been detected.
16	**RESP_CRC_ERR**	RO	0x0	SD/MMC Response failed CRC check. This bit is cleared with soft reset or the rising edge of HW_SSP_CTRL0_RUN.
15	**RESP_ERR**	RO	0x0	SD/MMC Card Responded to Command with an Error Condition. This bit is cleared with soft reset or the rising edge of HW_SSP_CTRL0_RUN.
14	**RESP_TIMEOUT**	RO	0x0	SD/MMC Card Expected Command Response not received within 64 CLK cycles. This indicates a card error, bad command, or command that failed CRC check. This bit is cleared with soft reset or the rising edge of HW_SSP_CTRL0_RUN.
13	**DATA_CRC_ERR**	RO	0x0	Data CRC Error. This bit is cleared with soft reset or the rising edge of HW_SSP_CTRL0_RUN.
12	**TIMEOUT**	RO	0x0	SD/MMC - timeout counter expired before data bus was ready. This bit is cleared with soft reset or the rising edge of HW_SSP_CTRL0_RUN.
11	**RECV_TIMEOUT_STAT**	RO	0x0	Raw Receive Timeout Status. Indicates that no read has occurred to non-empty receive data FIFO for 128 cycles
10	**RSVD3**	RO	0x0	Program this field to 0x0.
9	**FIFO_OVRFLW**	RO	0x0	FIFO Overflow Interrupt.
8	**FIFO_FULL**	RO	0x0	FIFO FULL
7:6	**RSVD2**	RO	0x0	Reserved
5	**FIFO_EMPTY**	RO	0x1	FIFO Empty.
4	**FIFO_UNDRFLW**	RO	0x0	FIFO Underflow has occurred.
3	**CMD_BUSY**	RO	0x0	SD/MMC command controller is busy sending a command or receiving a response
2	**DATA_BUSY**	RO	0x0	SD/MMC command controller is busy transferring data.
1	**RSVD1**	RO	0x0	Reserved
0	**BUSY**	RO	0x0	SSP State Machines are Busy.

DESCRIPTION:

Various SSP status fields are provided in this register.

EXAMPLE:

```
No Example.
```

21.10.14 SSP Debug Register Description

SSP Read Only Debug Registers.

HW_SSP_DEBUG 0x100

Table 21-34. HW_SSP_DEBUG

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DATACRC_ERR				DATA_STALL	DAT_SM			RSVD1				CMD_OE	DMA_SM			MMC_SM				CMD_SM		SSP_CMD	SSP_RESP	SSP_RXD							

Table 21-35. HW_SSP_DEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**DATACRC_ERR**	RO	0x0	Data CRC error
27	**DATA_STALL**	RO	0x0	MMC mode: FIFO transfer not ready
26:24	**DAT_SM**	RO	0x0	MMC dataxfer state machine DSM_IDLE = 0x0 DSM_WORD = 0x2 DSM_CRC1 = 0x3 DSM_CRC2 = 0x4 DSM_END = 0x5
23:20	**RSVD1**	RO	0x0	Program this field to 0x0. MSTK_IDLE = 0x0 MSTK_CKON = 0x1 MSTK_BS1 = 0x2 MSTK_TPC = 0x3 MSTK_BS2 = 0x4 MSTK_HDSHK = 0x5 MSTK_BS3 = 0x6 MSTK_RW = 0x7 MSTK_CRC1 = 0x8 MSTK_CRC2 = 0x9 MSTK_BS0 = 0xA MSTK_END1 = 0xB MSTK_END2W = 0xC MSTK_END2R = 0xD MSTK_DONE = 0xE
19	**CMD_OE**	RO	0x0	Enable for SSP_CMD
18:16	**DMA_SM**	RO	0x0	DMA state machine DMA_IDLE = 0x0 DMA_DMAREQ = 0x1 DMA_DMAACK = 0x2 DMA_STALL = 0x3 DMA_BUSY = 0x4 DMA_DONE = 0x5 DMA_COUNT = 0x6
15:12	**MMC_SM**	RO	0x0	MMC_state machine MMC_IDLE = 0x0 MMC_CMD = 0x1 MMC_TRC = 0x2 MMC_RESP = 0x3 MMC_RPRX = 0x4 MMC_TX = 0x5 MMC_CTOK = 0x6 MMC_RX = 0x7 MMC_CCS = 0x8 MMC_PUP = 0x9 MMC_WAIT = 0xA
11:10	**CMD_SM**	RO	0x0	MMC command_state machine CSM_IDLE = 0x0 CSM_INDEX = 0x1 CSM_ARG = 0x2 CSM_CRC = 0x3
9	**SSP_CMD**	RO	0x0	SSP_CMD

Table 21-35. HW_SSP_DEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
8	**SSP_RESP**	RO	0x0	SSP_RESP
7:0	**SSP_RXD**	RO	0x0	SSP_RXD.

DESCRIPTION:

Debug Register provide access to State machines.

EXAMPLE:

```
No Example.
```

21.10.15 SSP Version Register Description

This register reflects the version number for the SSP.

HW_SSP_VERSION 0x110

Table 21-36. HW_SSP_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 21-37. HW_SSP_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x03	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x00	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register reflects the version number for the SSP.

EXAMPLE:

```
No Example.
```

SSP Block v3.0, Revision 2.0

Chapter 22
Timers and Rotary Decoder

This chapter describes the timers and rotary decoder included on the i.MX23. Programmable registers are described in Section 22.4, “Programmable Registers.”

22.1 Overview

The i.MX23 implements four timers and a rotary decoder, as shown in Figure 22-1. The timers and decoder can take their inputs from any of the pins defined for PWM, rotary encoders, or certain divisions from the 32-kHz clock input. Thus, the PWM pins can be inputs or outputs, depending on the application.

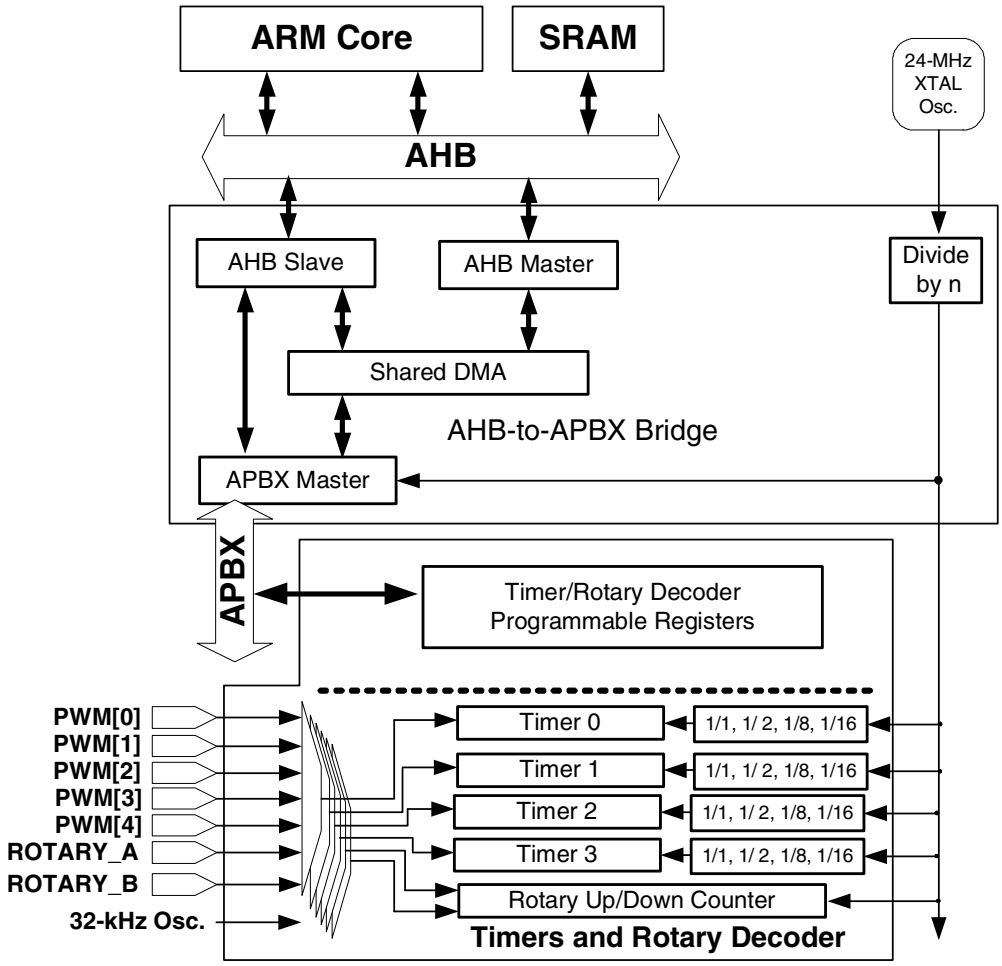


Figure 22-1. Timers and Rotary Decoder Block Diagram

The timer/rotary decoder block is a programmed I/O interface connected to the APBX bus. Recall that the APBX typically runs at a divided clock rate from the 24-MHz crystal clock. Each timer and rotary channel can sample at a rate that is further subdivided from the APBX clock. Each timer can select a different pre-scaler value.

22.2 Timers

Each of the four timers consists of a 16-bit fixed count value and a 16-bit free-running count value. In most cases, the free-running count decrements to 0. When it decrements to 0, it sets an interrupt status bit associated with the counter.

- If the RELOAD bit is set to 1, then the fixed count is automatically copied to the free-running counter and the count continues.
- If the RELOAD bit is not set, the timer stalls when it reaches 0.

Figure 22-2 shows a detailed view of either Timer 0, Timer 1, or Timer 2. Timer 3 has additional functionality, which is shown in Figure 22-3.

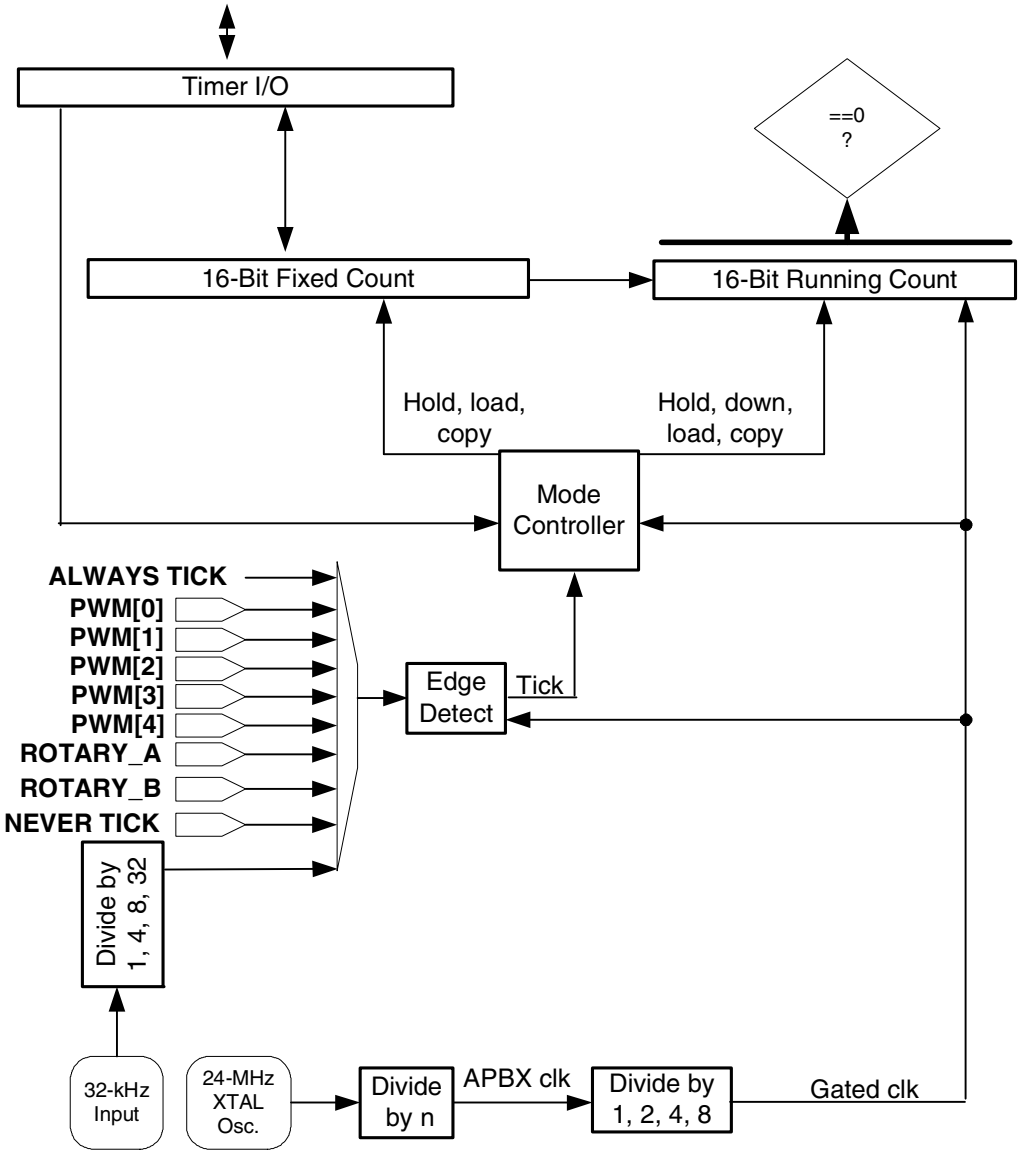


Figure 22-2. Timer 0, Timer 1, or Timer 2 Detail

Each timer has an UPDATE bit that controls whether the free-running-counter is loaded at the same time the fixed-count register is written from the CPU. The output of each timer's source select has a polarity control that allows the timer to operate on either edge.

Table 22-1 lists the timer state machine transitions.

Table 22-1. Timer State Machine Transitions

UPDATE	RELOAD	RUNNING
0	0	PIO writes to the fixed-count bit field have no effect on the running count.
0	1	The value written to the fixed count is used to reload the running count the next time it reaches 0. When the fixed count has been written with a value of 0 and the running count reaches 0, it continuously copies the fixed count value to the running count. Thus, writing a non-zero value to the fixed count register kicks off a continuous count and update operation.
1	0	The value written to the fixed count bit field is copied, immediately, to the running count, restarting any existing running count operation. When the new running count reaches 0, it freezes.
1	1	The value written to the fixed count bit field is copied, immediately, to the running count, restarting any existing running count operation. When the new running count reaches 0, it is reloaded from the value in the fixed count bit field, thus running continuously using the newly supplied fixed count.

When generating a periodic timer interrupt using the RELOAD bit, the user must compute the proper fixed-count value (count_value) based on clock speeds and clock divider settings. Note that, in this case, the actual value written to the FIXED_COUNT register field should be count_value – 1. For one-shot interrupts (RELOAD bit not set), the value written should be count_value. Any timer interrupt can be programmed as an FIQ or as a regular IRQ. See Chapter 5, “Interrupt Collector,” for programming details.

For proper detection of the input source signal, it should be much slower than the pre-scaled APBX clock (no greater than one-third the frequency of the pre-scaled APBX clock).

Selecting the ALWAYS tick causes the timer to decrement continuously at the rate established by the pre-scaled APBX clock. The NEVER TICK selection causes the timer to stall. Setting the fixed-count to 0xFFFF and setting the RELOAD bit causes the timer to operate in a continuous-count 65536 count mode.

The state of the 16-bit free-running count can be read by the CPU for each timer.

22.2.1 Using External Signals as Inputs

External signals can be used as inputs to the block. They can be used as either the test signal or sampling input signals (duty cycle or normal timer mode). This can be accomplished by using the rotary input pins or any unused PWM pins. If PWM pins are being used for this purpose, conflicts with the PWM or other blocks that could drive the pins as outputs must be avoided. In this case, the PWM pins being used should be programmed as GPIO inputs. (See Chapter 37, “Pin Control and GPIO,” for details.) Then, the external signal can be wired to the pin, and the PWM number selected in the appropriate TIMROT registers.

22.2.2 Timer 3 and Duty Cycle Mode

Timer 3 can operate in the same modes as Timer 0, Timer 1, and Timer 2. However, it has an additional duty cycle measurement mode. Figure 22-3 shows a detailed view of Timer 3.

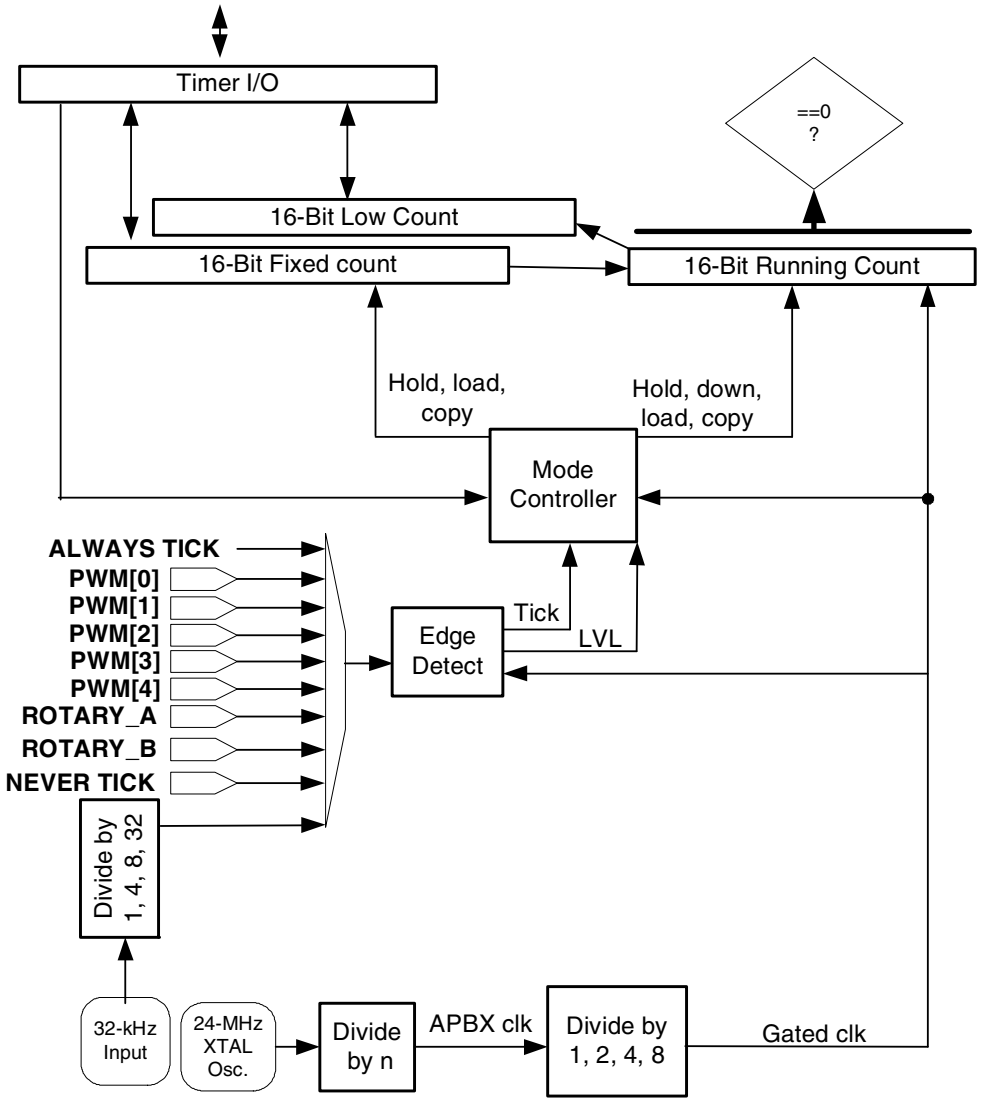


Figure 22-3. Timer 3 Detail

In the duty cycle mode, Timer 3 samples the free-running counter at the rising and falling edges of the input test signal, resetting the free-running counter on the same clock that is sampled.

- On the rising edge of the test signal, the free-running count is copied to the LOW_RUNNING_COUNT bit field of the HW_TIMROT_TIMCOUNT3 register.
- On the falling edge of the source clock, the free-running count is copied to the HIGH_FIXED_COUNT bit field (as shown in Figure 22-4).

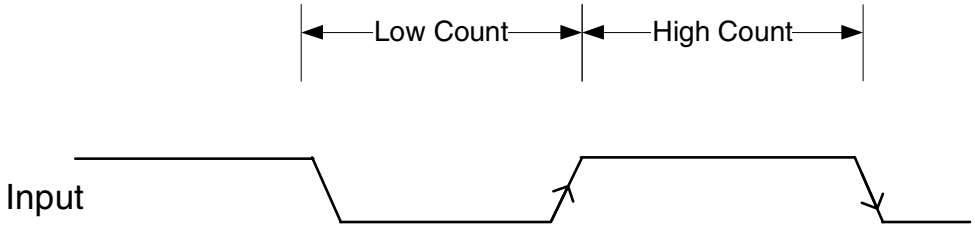


Figure 22-4. Pulse-Width Measurement Mode

- Once duty cycle mode is programmed and the input signal is stable, software should poll the DUTY_VALID bit in the HW_TIMROT_TIMCTRL3 register.
- This bit is automatically set and cleared by the hardware. When this bit is set, count values in the HW_TIMROT_TIMCOUNT3 register are stable and ready to be read.

Refer to the Timer 3 control and status register, HW_TIMROT_TIMCTRL3, where the DUTY_CYCLE bit controls whether HW_TIMROT_TIMCOUNT3 register's LOW_RUNNING_COUNT bit field reads back the running count or the low count of a duty cycle measurement. The DUTY_CYCLE bit also controls whether the HIGH_FIXED_COUNT bit field reads back the fixed-count value used in normal timer operations or the duty cycle high-time measurement.

It should be noted that for duty cycle mode to function properly, the timer "tick" source selected (SELECT field of the HW_TIMROT_TIMCTRL3 register) should be an appropriate frequency to sample the test signal. The NEVER_TICK value should never be used in this mode, as it will yield incorrect count results.

22.2.3 Testing Timer 3 Duty Cycle Modes

To test the duty cycle modes of Timer 3, select PWM1 as the input. PWM1 can generate waveforms of arbitrary duty cycle suitable for testing the duty cycle measurement capability.

22.3 Rotary Decoder

The rotary decoder uses two input selectors and edge detectors, as shown in Figure 22-5. It includes a debounce circuit for each input, as shown in Figure 22-6. This figure shows the debounce circuit for input A, though the circuit is identical for input B.

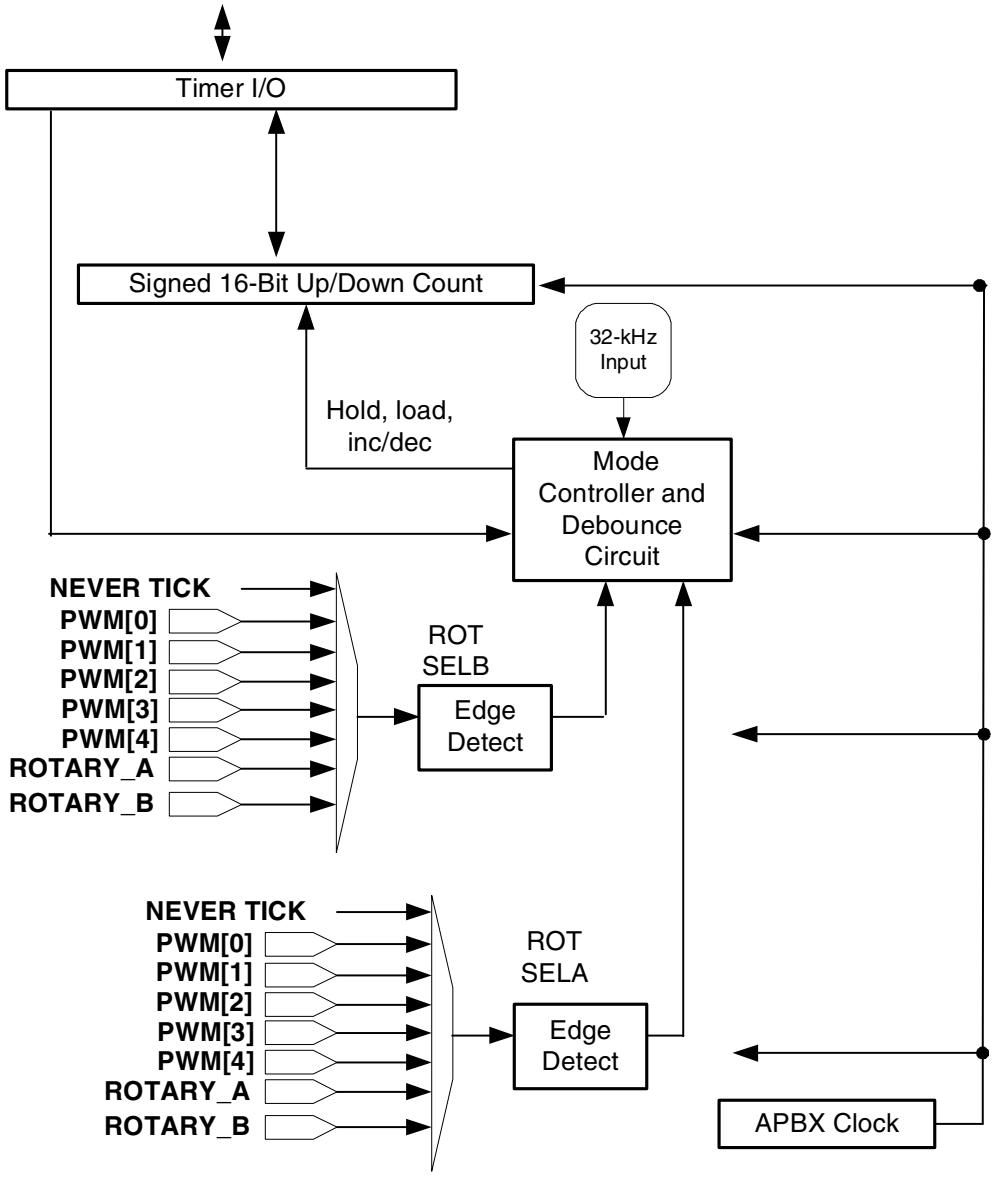


Figure 22-5. Detail of Rotary Decoder

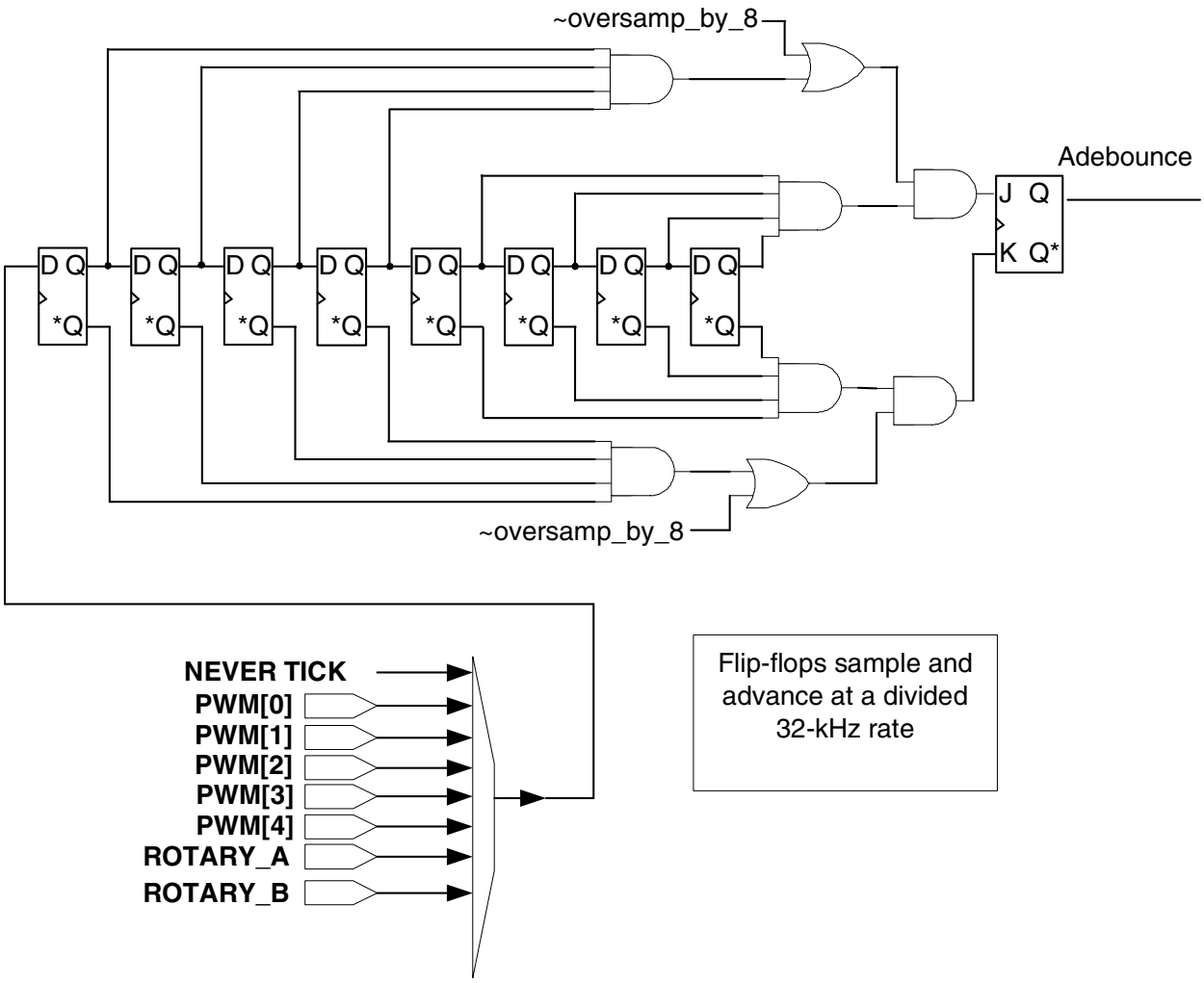


Figure 22-6. Rotary Decoding Mode—Debouncing Rotary A and B Inputs

A state machine following rotary decoder transition is provided to detect the direction of rotation and the time at which to increment or decrement the 16-bit signed counter in HW_TIMROT_ROTCOUNT. The updown counter can be treated as either a relative count or an absolute count, depending on the state of the HW_TIMROT_ROTCTRL_RELATIVE bit. When set to the relative mode, each read of the counter has the side effect of resetting it. The edge detectors respond to both edges of each input to determine the self-timed transition inputs to the state machine (see Figure 22-7).

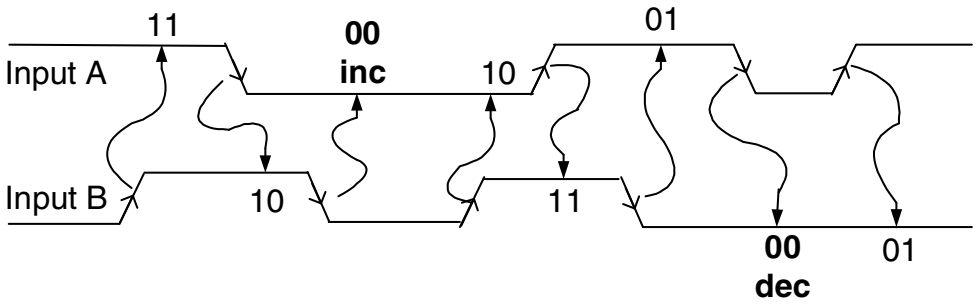


Figure 22-7. Rotary Decoding Mode—Input Transitions

Figure 22-7 shows that each detected edge causes a transition in the decoder state machine. Not all transitions are legal (see Table 22-2). For example, there is no legal way to transition directly from state 11 to 00 using normal inputs. In the cases where this occurs, the state machine goes to an alternate set of states and follows the input sequence until a valid sequence leading to state 00 is detected. No increment or decrement action is taken from the alternate state sequence.

Table 22-2. Rotary Decoder State Machine Transitions

CURRENT STATE	"INPUT" BA=00	"INPUT" BA=01	"INPUT" BA=10	"INPUT" BA=11
00	00	01	10	error
01	00, dec	01	error	11
10	00, inc	error	10	11
11	error	01	10	11

22.3.1 Testing the Rotary Decoder

To test the rotary decoder, select PWM1 and PWM2 as inputs to ROTARYA and ROTARYB. Since PWM1 and PWM2 can be started with known phase offsets and duty cycles, a continuous increment or decrement stream can be generated. Since PWM1 and PWM2 can be used as GPIO devices, the final part of the test is to generate and test a sequence of clockwise and counter-clockwise rotations to cover the entire state machine transitions, including the error conditions.

22.3.2 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, "Correct Way to Soft Reset a Block," for additional information on using the SFTRST and CLKGATE bit fields.

22.4 Programmable Registers

The following registers describe the programming interface for the timers and the rotary decoder.

22.4.1 Rotary Decoder Control Register Description

The Rotary Decoder Control Register specifies the reset state and the source selection for the rotary decoder. In addition, it specifies the polarity of any external input source that is used. This register also contains some general block controls including soft reset, clock gate, and present bits.

HW_TIMROT_ROTCTRL	0x000
HW_TIMROT_ROTCTRL_SET	0x004
HW_TIMROT_ROTCTRL_CLR	0x008
HW_TIMROT_ROTCTRL_TOG	0x00C

Table 22-3. HW_TIMROT_ROTCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	ROTARY_PRESENT	TIM3_PRESENT	TIM2_PRESENT	TIM1_PRESENT	TIM0_PRESENT	STATE			DIVIDER						RSRVD3			RELATIVE	OVERSAMPLE		POLARITY_B	POLARITY_A	RSRVD2				RSRVD1			SELECT_A

Table 22-4. HW_TIMROT_ROTCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	This bit must be set to zero to enable operation of any timer or the rotary decoder. When set to one, it forces a block-level reset and gates off the clocks to the block.
30	**CLKGATE**	RW	0x1	This bit must be set to zero for normal operation. When set to one, it gates off the clocks to the block.
29	**ROTARY_PRESENT**	RO	0x1	0= Rotary decoder is not present in this product. 1= Rotary decoder is present is in this product.
28	**TIM3_PRESENT**	RO	0x1	0= Timer 3 is not present in this product. 1= TIMER3 is present is in this product.
27	**TIM2_PRESENT**	RO	0x1	0= Timer 2 is not present in this product. 1= TIMER2 is present is in this product.
26	**TIM1_PRESENT**	RO	0x1	0= Timer 1 is not present in this product. 1= TIMER1 is present is in this product.
25	**TIM0_PRESENT**	RO	0x1	0= Timer 0 is not present in this product. 1= TIMER0 is present is in this product.
24:22	**STATE**	RO	0x0	Read-only view of the rotary decoder transition detecting state machine.
21:16	**DIVIDER**	RW	0x0	This bit field determines the divisor used to divide the 32-kHz on chip clock rate for oversampling (debouncing) the rotary A and B inputs. Note that the divider value is actually the (value of this field+1).
15:13	**RSRVD3**	RO	0x0	Always write zeroes to this bit field.
12	**RELATIVE**	RW	0x0	Set this bit to one to cause the rotary decoders updown counter to be reset to zero whenever it is read.

Table 22-4. HW_TIMROT_ROTCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
11:10	**OVERSAMPLE**	RW	0x0	This bit field determines the oversample rate to use in debouncing Rotary A and B inputs. 8X = 0x0 8x Oversample: 8 successive ones or zeroes to transition. 4X = 0x1 4x Oversample: 4 successive ones or zeroes to transition. 2X = 0x2 2x Oversample: 2 successive ones or zeroes to transition. 1X = 0x3 1x Oversample: Transition on each first input change.
9	**POLARITY_B**	RW	0x0	Set this bit to one to invert the input to the edge detector.
8	**POLARITY_A**	RW	0x0	Set this bit to one to invert the input to the edge detector.
7	**RSRVD2**	RO	0x0	Always write zeroes to this bit field.
6:4	**SELECT_B**	RW	0x0	Selects the source for the timer "tick" that increments the free-running counter that measures the A2B and B2A overlap counts. NEVER_TICK = 0x0 SelectB: Never tick. PWM0 = 0x1 SelectB: Input from PWM0. PWM1 = 0x2 SelectB: Input from PWM1. PWM2 = 0x3 SelectB: Input from PWM2. PWM3 = 0x4 SelectB: Input from PWM3. PWM4 = 0x5 SelectB: Input from PWM4. ROTARYA = 0x6 SelectB: Input from Rotary A. ROTARYB = 0x7 SelectB: Input from Rotary B.
3	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
2:0	**SELECT_A**	RW	0x0	Selects the source for the timer "tick" that increments the free-running counter that measures the A2B and B2A overlap counts. NEVER_TICK = 0x0 SelectA: Never tick. PWM0 = 0x1 SelectA: Input from PWM0. PWM1 = 0x2 SelectA: Input from PWM1. PWM2 = 0x3 SelectA: Input from PWM2. PWM3 = 0x4 SelectA: Input from PWM3. PWM4 = 0x5 SelectA: Input from PWM4. ROTARYA = 0x6 SelectA: Input from Rotary A. ROTARYB = 0x7 SelectA: Input from Rotary B.

DESCRIPTION:

This register contains control parameters to specify the rotary decoder setup. It also contains some general block controls including soft reset, clock gate, and present bits.

EXAMPLE:

```
HW_TIMROT_ROTCTRL_WR(0x00000076); // Set up rotary control fields.
```

22.4.2 Rotary Decoder Up/Down Counter Register Description

The Rotary Decoder Up/Down Counter Register contains the timer counter value that counts up or down as the rotary encoder is rotated.

HW_TIMROT_ROTCOUNT 0x010

Table 22-5. HW_TIMROT_ROTCOUNT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																UPDOWN															

Table 22-6. HW_TIMROT_ROTCOUNT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
15:0	**UPDOWN**	RO	0x00	At each edge of the Rotary A input, the Rotary B value is sampled, similarly at each edge of the Rotary B input, the Rotary A input is sampled. These values drive a rotary decoder state machine that determines when this counter is incremented or decremetned. When set in the RELATIVE mode, reads from this register clear this register as a side effect. Counter values in this register are signed 16-bit values.

DESCRIPTION:

This register contains the read-only current count for the rotary decoder.

EXAMPLE:

```
count = HW_TIMROT_ROTCTRL_RD(); // Read count
```

22.4.3 Timer 0 Control and Status Register Description

The Timer 0 Control and Status Register specifies timer control parameters, as well as interrupt status and the enable for Timer 0.

HW_TIMROT_TIMCTRL0	0x020
HW_TIMROT_TIMCTRL0_SET	0x024
HW_TIMROT_TIMCTRL0_CLR	0x028
HW_TIMROT_TIMCTRL0_TOG	0x02C

Table 22-7. HW_TIMROT_TIMCTRL0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2																IRQ	IRQ_EN	RSRVD1						POLARITY	UPDATE	RELOAD	PRESCALE		SELECT		

Table 22-8. HW_TIMROT_TIMCTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSRVD2**	RO	0x0	Always write zeroes to this bit field.
15	**IRQ**	RW	0x0	This bit is set to one when Timer 0 decrements to zero. Write a zero to clear it or use Clear SCT mode.
14	**IRQ_EN**	RW	0x0	Set this bit to one to enable the generation of a CPU interrupt when the count reaches zero in normal counter mode.
13:9	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
8	**POLARITY**	RW	0x0	Set this bit to one to invert the input to the edge detector. 0: Positive edge detection. 1: Invert to negative edge detection.
7	**UPDATE**	RW	0x0	Set this bit to one to cause the running count to be written from the CPU at the same time a new fixed count register value is written.
6	**RELOAD**	RW	0x0	Set this bit to one to cause the timer to reload its current count from its fixed count value whenever the current count decrements to zero. When set to zero, the timer enters a mode that freezes at a count of zero. When the fixed count is zero, setting this bit to one causes a continuous reload of the fixed count register so that writting a non-zero value will start the timer.
5:4	**PRESCALE**	RW	0x0	Selects the divisor used for clock generation. The APBX clock is divided by the following amount. Note the APBX clock itself is initially divided down from the 24.0-MHz crystal clock frequency. DIV_BY_1 = 0x0 PreScale: Divide the APBX clock by 1. DIV_BY_2 = 0x1 PreScale: Divide the APBX clock by 2. DIV_BY_4 = 0x2 PreScale: Divide the APBX clock by 4. DIV_BY_8 = 0x3 PreScale: Divide the APBX clock by 8.
3:0	**SELECT**	RW	0x0	Selects the source for the timer "tick" that decrements the free running counter. Note: programming an undefined value will result in "always tick" behavior. NEVER_TICK = 0x0 Never tick. PWM0 = 0x1 Input from PWM0. PWM1 = 0x2 Input from PWM1. PWM2 = 0x3 Input from PWM2. PWM3 = 0x4 Input from PWM3. PWM4 = 0x5 Input from PWM4. ROTARYA = 0x6 Input from Rotary A. ROTARYB = 0x7 Input from Rotary B. 32KHZ_XTAL = 0x8 Input from 32-kHz crystal. 8KHZ_XTAL = 0x9 Input from 8-kHz (divided from 32-kHz crystal). 4KHZ_XTAL = 0xA Input from 4-kHz (divided from 32-kHz crystal). 1KHZ_XTAL = 0xB Input from 1-kHz (divided from 32-kHz crystal). TICK_ALWAYS = 0xC Always tick.

DESCRIPTION:

This control register specifies control parameters, as well as interrupt status and the enable for Timer 0.

EXAMPLE:

```
HW_TIMROT_TIMCTRLn_WR(0, 0x00000008); // Set up control fields for timer0
```

22.4.4 Timer 0 Count Register Description

The Timer 0 Count Register contains the timer counter values for Timer 0.

HW_TIMROT_TIMCOUNT0 0x030

Table 22-9. HW_TIMROT_TIMCOUNT0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RUNNING_COUNT																FIXED_COUNT															

Table 22-10. HW_TIMROT_TIMCOUNT0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RUNNING_COUNT**	RO	0x00	This bit field shows the current state of the running count as it decrements.
15:0	**FIXED_COUNT**	RW	0x00	Software loads the fixed count bit field with the value to down count. If the reload bit is set to one, then the new value will be loaded into the running count the next time it reaches zero. If the update bit is set to one, then the new value is also copied into the running count, immediately. If both the reload and update bits are set to zero, then the new value is never picked up by the running count.

DESCRIPTION:

This timer count register contains the programable and readback counter values for Timer 0.

EXAMPLE:

```
HW_TIMROT_TIMCOUNTn_WR(0, 0x0000f0dd); // Set up timer count
```

22.4.5 Timer 1 Control and Status Register Description

The Timer 1 Control and Status Register specifies timer control parameters, as well as interrupt status and the enable for Timer 1.

HW_TIMROT_TIMCTRL1	0x040
HW_TIMROT_TIMCTRL1_SET	0x044
HW_TIMROT_TIMCTRL1_CLR	0x048
HW_TIMROT_TIMCTRL1_TOG	0x04C

Table 22-11. HW_TIMROT_TIMCTRL1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2																IRQ	IRQ_EN	RSRVD1					POLARITY	UPDATE	RELOAD	PRESCALE		SELECT			

Table 22-12. HW_TIMROT_TIMCTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSRVD2**	RO	0x0	Always write zeroes to this bit field.
15	**IRQ**	RW	0x0	This bit is set to one when Timer 1 decrements to zero. Write a zero to clear it or use Clear SCT mode.
14	**IRQ_EN**	RW	0x0	Set this bit to one to enable the generation of a CPU interrupt when the count reaches zero in normal counter mode.
13:9	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
8	**POLARITY**	RW	0x0	Set this bit to one to invert the input to the edge detector. 0: Positive edge detection. 1: Invert to negative edge detection.
7	**UPDATE**	RW	0x0	Set this bit to one to cause the running count to be written from the CPU at the same time a new fixed count register value is written.
6	**RELOAD**	RW	0x0	Set this bit to one to cause the timer to reload its current count from its fixed count value whenever the current count decrements to zero. When set to zero, the timer enters a mode that freezes at a count of zero. When the fixed count is zero, setting this bit to one causes a continuous reload of the fixed count register so that writting a non-zero value will start the timer.
5:4	**PRESCALE**	RW	0x0	Selects the divisor used for clock generation. The APBX clock is divided by the following amount. Note the APBX clock itself is initially divided down from the 24.0-MHz crystal clock frequency. DIV_BY_1 = 0x0 PreScale: Divide the APBX clock by 1. DIV_BY_2 = 0x1 PreScale: Divide the APBX clock by 2. DIV_BY_4 = 0x2 PreScale: Divide the APBX clock by 4. DIV_BY_8 = 0x3 PreScale: Divide the APBX clock by 8.
3:0	**SELECT**	RW	0x0	Selects the source for the timer "tick" that decrements the free running counter. Note: programming an undefined value will result in "always tick" behavior. NEVER_TICK = 0x0 Never tick. PWM0 = 0x1 Input from PWM0. PWM1 = 0x2 Input from PWM1. PWM2 = 0x3 Input from PWM2. PWM3 = 0x4 Input from PWM3. PWM4 = 0x5 Input from PWM4. ROTARYA = 0x6 Input from Rotary A. ROTARYB = 0x7 Input from Rotary B. 32KHZ_XTAL = 0x8 Input from 32-kHz crystal. 8KHZ_XTAL = 0x9 Input from 8 kHz (divided from 32-kHz crystal). 4KHZ_XTAL = 0xA Input from 4 kHz (divided from 32-kHz crystal). 1KHZ_XTAL = 0xB Input from 1 kHz (divided from 32-kHz crystal). TICK_ALWAYS = 0xC Always tick.

DESCRIPTION:

This control register specifies control parameters, as well as interrupt status and the enable for Timer 1.

EXAMPLE:

```
HW_TIMROT_TIMCTRLn_WR(1, 0x00000008); // Set up control fields for timer1
```

22.4.6 Timer 1 Count Register Description

The Timer 1 Count Register contains the timer counter values for Timer 1.

HW_TIMROT_TIMCOUNT1	0x050

Table 22-13. HW_TIMROT_TIMCOUNT1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RUNNING_COUNT																FIXED_COUNT															

Table 22-14. HW_TIMROT_TIMCOUNT1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RUNNING_COUNT**	RO	0x00	This bit field shows the current state of the running count as it decrements.
15:0	**FIXED_COUNT**	RW	0x00	Software loads the fixed count bit field with the value to down count. If the reload bit is set to one, then the new value will be loaded into the running count the next time it reaches zero. If the update bit is set to one, then the new value is also copied into the running count, immediately. If both the reload and update bits are set to zero, then the new value is never picked up by the running count.

DESCRIPTION:

This timer count register contains the programable and readback counter values for Timer 1.

EXAMPLE:

```
HW_TIMROT_TIMCOUNTn_WR(1, 0x0000f0dd); // Set up timer count
```

22.4.7 Timer 2 Control and Status Register Description

The Timer 2 Control and Status Register specifies timer control parameters, as well as interrupt status and the enable for Timer 2.

HW_TIMROT_TIMCTRL2	0x060
HW_TIMROT_TIMCTRL2_SET	0x064

HW_TIMROT_TIMCTRL2_CLR 0x068
HW_TIMROT_TIMCTRL2_TOG 0x06C

Table 22-15. HW_TIMROT_TIMCTRL2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2																IRQ	IRQ_EN	RSRVD1					POLARITY	UPDATE	RELOAD	PRESCALE		SELECT			

Table 22-16. HW_TIMROT_TIMCTRL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSRVD2**	RO	0x0	Always write zeroes to this bit field.
15	**IRQ**	RW	0x0	This bit is set to one when Timer 2 decrements to zero. Write a zero to clear it or use Clear SCT mode.
14	**IRQ_EN**	RW	0x0	Set this bit to one to enable the generation of a CPU interrupt when the count reaches zero in normal counter mode.
13:9	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
8	**POLARITY**	RW	0x0	Set this bit to one to invert the input to the edge detector. 0: Positive edge detection. 1: Invert to negative edge detection.
7	**UPDATE**	RW	0x0	Set this bit to one to cause the running count to be written from the CPU at the same time a new fixed count register value is written.
6	**RELOAD**	RW	0x0	Set this bit to one to cause the timer to reload its current count from its fixed count value whenever the current count decrements to zero. When set to zero, the timer enters a mode that freezes at a count of zero. When the fixed count is zero, setting this bit to one causes a continuous reload of the fixed count register so that writting a non-zero value will start the timer.

Table 22-16. HW_TIMROT_TIMCTRL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
5:4	**PRESCALE**	RW	0x0	Selects the divisor used for clock generation. The APBX clock is divided by the following amount. Note the APBX clock itself is initially divided down from the 24.0-MHz crystal clock frequency. DIV_BY_1 = 0x0 PreScale: Divide the APBX clock by 1. DIV_BY_2 = 0x1 PreScale: Divide the APBX clock by 2. DIV_BY_4 = 0x2 PreScale: Divide the APBX clock by 4. DIV_BY_8 = 0x3 PreScale: Divide the APBX clock by 8.
3:0	**SELECT**	RW	0x0	Selects the source for the timer "tick" that decrements the free running counter. Note: programming an undefined value will result in "always tick" behavior. NEVER_TICK = 0x0 Never tick. PWM0 = 0x1 Input from PWM0. PWM1 = 0x2 Input from PWM1. PWM2 = 0x3 Input from PWM2. PWM3 = 0x4 Input from PWM3. PWM4 = 0x5 Input from PWM4. ROTARYA = 0x6 Input from Rotary A. ROTARYB = 0x7 Input from Rotary B. 32KHZ_XTAL = 0x8 Input from 32-kHz crystal. 8KHZ_XTAL = 0x9 Input from 8 kHz (divided from 32-kHz crystal). 4KHZ_XTAL = 0xA Input from 4 kHz (divided from 32-kHz crystal). 1KHZ_XTAL = 0xB Input from 1 kHz (divided from 32-kHz crystal). TICK_ALWAYS = 0xC Always tick.

DESCRIPTION:

This control register specifies control parameters as well as interrupt status and the enable for Timer 2.

EXAMPLE:

```
HW_TIMROT_TIMCTRLn_WR(2, 0x00000008); // Set up control fields for timer2
```

22.4.8 Timer 2 Count Register Description

The Timer 2 Count Register contains the timer counter values for Timer 2.

HW_TIMROT_TIMCOUNT2 0x070

Table 22-17. HW_TIMROT_TIMCOUNT2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RUNNING_COUNT																FIXED_COUNT															

Table 22-18. HW_TIMROT_TIMCOUNT2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RUNNING_COUNT**	RO	0x00	This bit field shows the current state of the running count as it decrements.
15:0	**FIXED_COUNT**	RW	0x00	Software loads the fixed count bit field with the value to down count. If the reload bit is set to one, then the new value will be loaded into the running count the next time it reaches zero. If the update bit is set to one, then the new value is also copied into the running count, immediately. If both the reload and update bits are set to zero, then the new value is never picked up by the running count.

DESCRIPTION:

This timer count register contains the programable and readback counter values for Timer 2.

EXAMPLE:

```
HW_TIMROT_TIMCOUNTn_WR(2, 0x0000f0dd); // Set up timer count
```

22.4.9 Timer 3 Control and Status Register Description

The Timer 3 Control and Status Register specifies timer control parameters, as well as interrupt status and the enable for Timer 3.

HW_TIMROT_TIMCTRL3	0x080
HW_TIMROT_TIMCTRL3_SET	0x084
HW_TIMROT_TIMCTRL3_CLR	0x088
HW_TIMROT_TIMCTRL3_TOG	0x08C

Table 22-19. HW_TIMROT_TIMCTRL3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2												TEST_SIGNAL				IRQ	IRQ_EN	RSRVD1			DUTY_VALID	DUTY_CYCLE	POLARITY	UPDATE	RELOAD	PRESCALE		SELECT			

Table 22-20. HW_TIMROT_TIMCTRL3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:20	**RSRVD2**	RO	0x0	Always write zeroes to this bit field.
19:16	**TEST_SIGNAL**	RW	0x0	Selects the source of the signal to be measured in duty cycle mode. NEVER_TICK = 0x0 Never tick. Freeze the count. PWM0 = 0x1 Input from PWM0. PWM1 = 0x2 Input from PWM1. PWM2 = 0x3 Input from PWM2. PWM3 = 0x4 Input from PWM3. PWM4 = 0x5 Input from PWM4. ROTARYA = 0x6 Input from Rotary A. ROTARYB = 0x7 Input from Rotary B. 32KHZ_XTAL = 0x8 Input from 32-kHz crystal. 8KHZ_XTAL = 0x9 Input from 8 kHz (divided from 32-kHz crystal). 4KHZ_XTAL = 0xA Input from 4 kHz (divided from 32-kHz crystal). 1KHZ_XTAL = 0xB Input from 1 kHz (divided from 32-kHz crystal). TICK_ALWAYS = 0xC Always tick.
15	**IRQ**	RW	0x0	This bit is set to one when Timer 3 decrements to zero. Write a zero to clear it or use Clear SCT mode.
14	**IRQ_EN**	RW	0x0	Set this bit to one to enable the generation of a CPU interrupt when the count reaches zero in normal counter mode.
13:11	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
10	**DUTY_VALID**	RO	0x0	This bit is set and cleared by the hardware. It is set only when in duty cycle measuring mode and the HW_TIMROT_TIMCOUNT3 has valid duty cycle data to be read. This register will be cleared if not in duty cycle mode or on writes to this register. In the case that it is written while in duty cycle mode, this bit will clear but will again be set at the appropriate time for reading the count register.
9	**DUTY_CYCLE**	RW	0x0	Set this bit to one to cause the timer to operate in duty cycle measuring mode.
8	**POLARITY**	RW	0x0	Set this bit to one to invert the input to the edge detector. 0: Positive edge detection. 1: Invert to negative edge detection.
7	**UPDATE**	RW	0x0	Set this bit to one to cause the running count to be written from the CPU at the same time a new fixed count register value is written.
6	**RELOAD**	RW	0x0	Set this bit to one to cause the timer to reload its current count from its fixed count value whenever the current count decrements to zero. When set to zero, the timer enters a mode that freezes at a count of zero. When the fixed count is zero, setting this bit to one causes a continuous reload of the fixed count register so that writting a non-zero value will start the timer.

Table 22-20. HW_TIMROT_TIMCTRL3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
5:4	**PRESCALE**	RW	0x0	Selects the divisor used for clock generation. The APBX clock is divided by the following amount. Note the APBX clock itself is initially divided down from the 24.0-MHz crystal clock frequency. DIV_BY_1 = 0x0 PreScale: Divide the APBX clock by 1. DIV_BY_2 = 0x1 PreScale: Divide the APBX clock by 2. DIV_BY_4 = 0x2 PreScale: Divide the APBX clock by 4. DIV_BY_8 = 0x3 PreScale: Divide the APBX clock by 8.
3:0	**SELECT**	RW	0x0	Selects the source for the timer "tick" that decrements the free running counter. Note: programming an undefined value will result in "always tick" behavior. In duty cycle mode it increments the counter used to calculate the high and low cycle counts. NEVER_TICK = 0x0 Never tick. Freeze the count. PWM0 = 0x1 Input from PWM0. PWM1 = 0x2 Input from PWM1. PWM2 = 0x3 Input from PWM2. PWM3 = 0x4 Input from PWM3. PWM4 = 0x5 Input from PWM4. ROTARYA = 0x6 Input from Rotary A. ROTARYB = 0x7 Input from Rotary B. 32KHZ_XTAL = 0x8 Input from 32-kHz crystal. 8KHZ_XTAL = 0x9 Input from 8 kHz (divided from 32-kHz crystal). 4KHZ_XTAL = 0xA Input from 4 kHz (divided from 32-kHz crystal). 1KHZ_XTAL = 0xB Input from 1 kHz (divided from 32-kHz crystal). TICK_ALWAYS = 0xC Always tick.

DESCRIPTION:

This control register specifies control parameters, as well as interrupt status and the enable for Timer 3.

EXAMPLE:

```
HW_TIMROT_TIMCTRLn_WR(3, 0x00000008); // Set up control fields for timer3
```

22.4.10 Timer 3 Count Register Description

The Timer 3 Count Register contains the timer counter values for Timer 3. NOTE: This timer can be put in a special duty cycle mode that will measure the duty cycle of an input test signal.

HW_TIMROT_TIMCOUNT3 0x090

Table 22-21. HW_TIMROT_TIMCOUNT3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
LOW_RUNNING_COUNT																HIGH_FIXED_COUNT															

Table 22-22. HW_TIMROT_TIMCOUNT3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**LOW_RUNNING_COUNT**	RO	0x00	In duty cycle mode, this bit field is loaded from the running counter when it has just finished measuring the low portion of the duty cycle. In normal timer mode, it shows the running count as a read-only value.
15:0	**HIGH_FIXED_COUNT**	RW	0x00	Software loads the fixed count bit field with the value to down count. If the reload bit is set to one, then the new value will be loaded into the running count the next time it reaches zero. If the update bit is set to one, then the new value is also copied into the running count, immediately. If both the reload and update bits are set to zero, then the new value is never picked up by the running count. In duty cycle mode, this bit field is loaded from the running counter when it has finished measuring the high portion of the duty cycle.

DESCRIPTION:

This timer count register contains the programable and readback counter values for Timer 3. The definitions of the fields change depending whether the timer is in normal or duty cycle mode.

EXAMPLE:

```
HW_TIMROT_TIMCOUNTn_WR(3, 0x0000f0dd); // Set up timer count
```

22.4.11 TIMROT Version Register Description

This register always returns a known read value for debug purposes it indicates the version of the block.

HW_TIMROT_VERSION 0x0a0

Table 22-23. HW_TIMROT_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 22-24. HW_TIMROT_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x01	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x01	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
if (HW_TIMROT_VERSION.B.MAJOR != 1) Error();
```

TIMROT Block v1.1, Revision 1.38

Chapter 23
Real-Time Clock, Alarm, Watchdog, Persistent Bits

This chapter describes the real-time clock, alarm clock, watchdog reset, persistent bits, and millisecond counter included on the i.MX23. Programmable registers are described in Section 23.8, "Programmable Registers."

23.1 Overview

The real-time clock (RTC) and alarm share a one-second pulse time domain. The watchdog reset and millisecond counter run on a one-millisecond time domain. The RTC, alarm, and persistent bits use persistent storage and reside in a special power domain (crystal domain) that remains powered up even when the rest of the chip is in its powered-down state. Figure 23-1 illustrates this block.

NOTE: The term *power-down*, as used here, refers to a state in which the DC-DC converter and various parts of the crystal power domain are still powered up, but the rest of the chip is powered down. If the battery is removed, then the persistent bits, the alarm value, and the second counter value will be lost. The *crystal power domain* powers both the 32-kHz and 24-MHz crystals. Note that the 32 kHz crystal is not available in the 128-pin LQFP package.

Upon battery insertion, the crystals (32-kHz and 24-MHz) are in a quiescent state. The activation of these crystals is under software control through the *RTC persistent bits*, as described later in this chapter.

- The XTAL32KHZ_PWRUP bit in the Persistent Register 0 controls the activity of the 32-kHz crystal at all times (chip power on or off).
- The XTAL24MHZ_PWRUP bit in the Persistent Register 0 controls the behavior of the 24-MHz crystal during the power-off state. (The 24-MHz crystal is always on when the chip is powered up.)

The one-second time base is derived either from the 24.0-MHz crystal oscillator or a 32-kHz crystal oscillator (which can be either exactly 32.0 kHz or 32.768 kHz), as controlled by bits in Persistent Register 0. The time base thus generated is used to increment the value of the persistent seconds count register. Like the values of the other persistent registers, the value of the persistent seconds count register is not lost across a power-down state and will continue to count seconds during that time.

Contrary to the one-second time base, no record or count is made of the one-millisecond time base in the crystal power domain. The one-millisecond time base is always derived from the 24.0-MHz crystal oscillator, but is not available when the chip is powered down.

The real-time clock seconds counter, alarm functions, and persistent bit storage are kept in the (always on) crystal power domain. Shadow versions of these values are maintained in the CPU's power and

APBX clock domain when the chip is in the power-up state. When the chip transitions from power-off to power-on, the master values are copied to shadow registers by the copy controller. Whenever software writes to a shadow register, then the copy controller copies the new value into the master register in the crystal oscillator power domain.

Some of the persistent bits are used to control features that can continue to operate after power-down, such as the second counter and the alarm function and bits in the HW_RTC_PERSISTENT0 register. The bits in registers HW_RTC_PERSISTENT1 through HW_RTC_PERSISTENT5 are available to store application state information over power-downs and are completely software-defined.

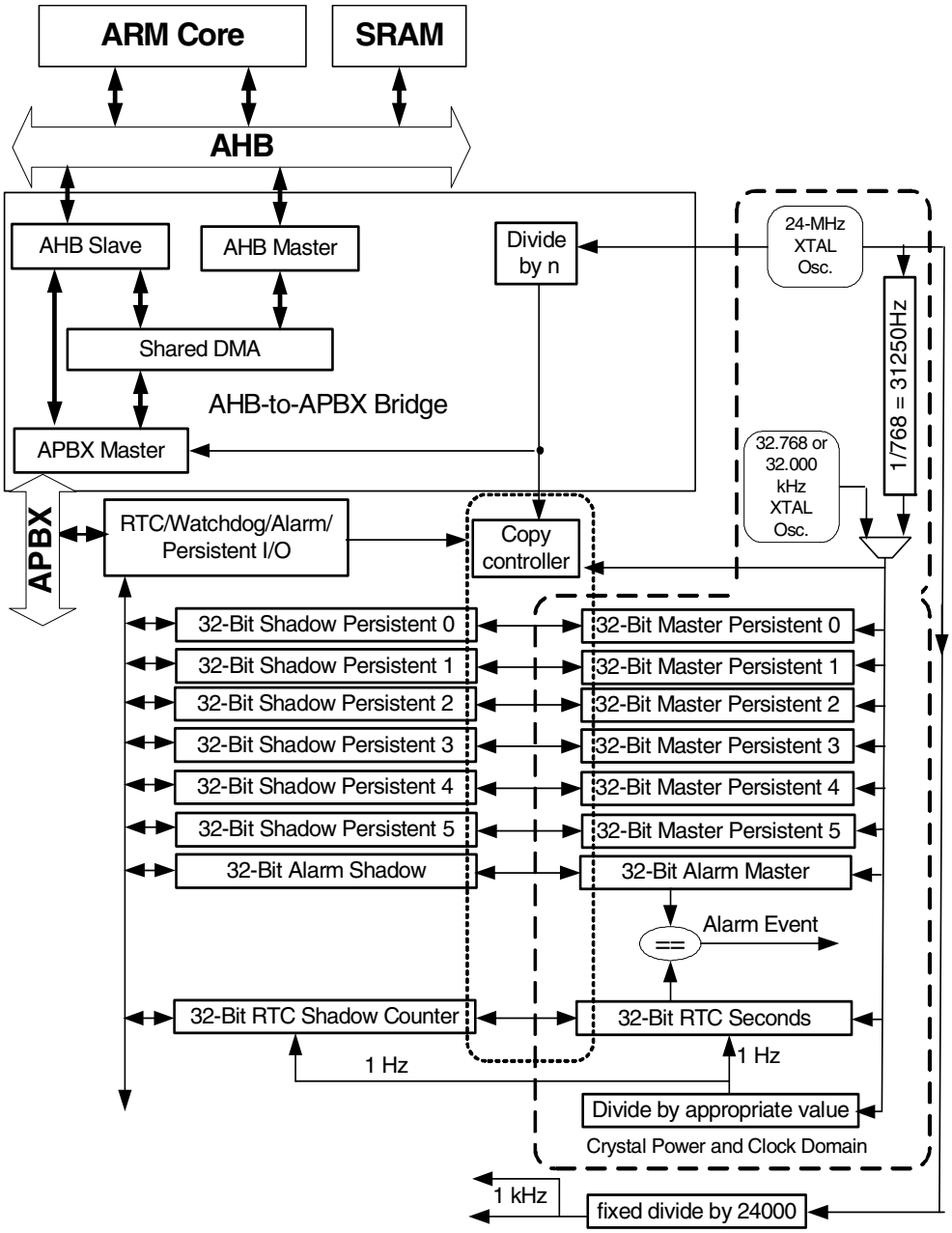


Figure 23-1. RTC, Watchdog, Alarm, and Persistent Bits Block Diagram

Immediately after reset, it will take approximately three milliseconds for the copy controller to complete the copy process from the analog domain to the digital domain. Software cannot rely on the contents of the seconds counter, alarm, or persistent bits until this copy is complete. Therefore, software must wait until all bits of interest in the HW_RTC_STAT_STALE_REGS field have been reset to 0 by the copy controller before reading the initial state of these values (see Figure 23-2). The order in which registers are updated is Persistent 0, 1, 2, 3, 4, 5, Alarm, Seconds. (This list is in bitfield order, from LSB to MSB,

as they would appear in the STALE_REGS and NEW_REGS bitfields of the HW_RTC_STAT register. For example, the Seconds register corresponds to STALE_REGS or NEW_REGS containing 0x80.)

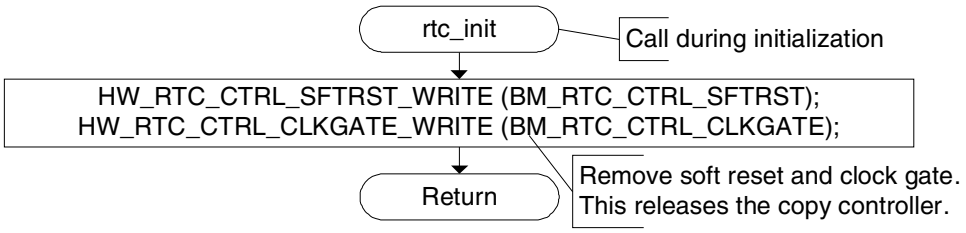


Figure 23-2. RTC Initialization Sequence

23.2 Programming and Enabling the RTC Clock

The RTC functions implemented in the crystal power domain are referred to as RTC analog functions. The clock frequency and clock source for the RTC analog functions are programmable. There are three possible clock options.

- If the HW_RTC_PERSISTENT0_CLOCKSOURCE bit is set to 0, these functions operate on a clock domain derived from the 24.0-MHz crystal oscillator divided by 768 to yield 31.250 kHz.
- If the HW_RTC_PERSISTENT0_CLOCKSOURCE bit is 1 and the HW_RTC_PERSISTENT0_XTAL32_FREQ bit is 0, then the optional external driving crystal clock will be used and its frequency should be 32.768 kHz.
- However, if the HW_RTC_PERSISTENT0_CLOCKSOURCE is 1 and the HW_RTC_PERSISTENT0_XTAL32_FREQ bit is also 1, then the external crystal generated clock should be 32.000 kHz for correct operation.

Thus, the HW_RTC_PERSISTENT0_XTAL32_FREQ bit gives the systems designer some flexibility as to which external crystal to use on the board.

Switching between these two clock domains is handled by a glitch-free clock mux and can be done on the fly. The 1-Hz time base is derived by dividing either 32.768 kHz by 32768 or by dividing 31.250 kHz by 31,250, or by dividing 32.000 kHz by 32000.

By reading and examining the HW_RTC_STAT_XTAL32000_PRESENT and HW_RTC_STAT_XTAL32768_PRESENT bits, software can discover if there is an optional crystal clock present and the frequency at which it runs (32.768 kHz or 32.000 kHz). Only one of these fuse bits will be asserted if there is such a crystal attached. If there is no crystal present, both bits will be deasserted.

23.3 RTC Persistent Register Copy Control

The copying of a persistent shadow register (digital) to persistent master storage (analog) occurs automatically. This automatic write-back that occurs for each register as the copy controller services writes to the shadow registers can lead to some very long timing loops if efficient write procedures are not used. Writing all eight shadow registers can take several milliseconds to complete. Do not attempt to write to more than one shadow register immediately before power down. Whenever possible, software should ensure

that the HW_RTC_STAT_NEW_REGS field is 0 before powering down the chip or setting the HW_RTC_CTRL_SFTRST bit. Otherwise, some of the write data could be lost because the shadow registers could be powered down/reset before the new values can be copied to persistent master storage.

Registers are copied between the digital and analog sides one by one and in 32-bit words. There are no hardwired uses for any of the bits of Persistent registers 1 through 5. And thus, the bits in these registers can be defined and set by software. Persistent Register 0 is reserved for hardware programming and configuration.

Before a new value is written to a shadow register by the CPU, software must first confirm that the corresponding bit of HW_RTC_STAT_NEW_REGS is a 0. This ensures that a value previously written to the register has been completely handled by the copy state machine. Failure to obey this constraint could cause a newer updated value to be lost.

NOTE: The HW_RTC_CTRL_SUPPRESS_COPY2ANALOG diagnostic bit is never set while any copy or update operation is underway. Doing so will result in undefined operation of the copy controller.

Figure 23-3 shows the copy and test procedure for a single register. However, software can write to any register whose HW_RTC_STAT_NEW_REGS bit is 0 even if the copy controller is currently busy copying a different register. For example, if the copy controller is busy copying Persistent Register 1 from a previous write, software can simultaneously write to Persistent Register 0 during this copy. After the copy controller has finished copying register 1, it will then, in turn, begin the copy process for the new value of Register 0. Thus, registers can thus be written in any order. Again, the main important rule that must be followed is that the HW_RTC_STAT_NEW_REGS bit for a particular register must be 0 before it can be written and is independent of the state of the HW_RTC_STAT_NEW_REGS bits for the other registers.

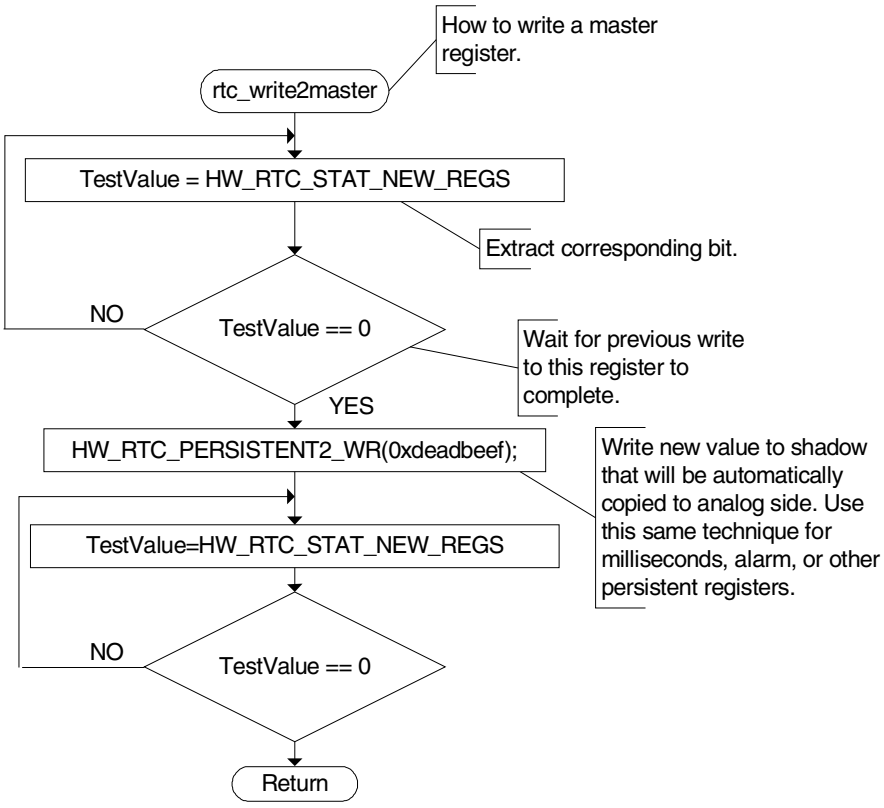


Figure 23-3. RTC Writing to a Master Register from CPU

The digital shadow registers will be updated (copied from analog to digital) with values from their analog persistent counterparts under two conditions: first, whenever the chip is powered on, and secondly, whenever the HW_RTC_CTRL_FORCE_UPDATE bit is set by software. Then, an update occurs, and all persistent registers are updated. Persistent registers cannot be updated singly.

23.4 Real-Time Clock Function

The real-time clock is a CPU-accessible, continuously-running 32-bit counter that increments every second and that can be derived from either the 24-MHz clock or the 32-kHz clock, as determined by writable bit values in the RTC Persistent Register 0.

A 32-bit second counter has enough resolution to count up to 136 years with one-second increments. The RTC can continue to count time as long as a voltage is applied to the BATT pin, irrespective of whether the rest of the chip is powered up. The normal digital reset has no effect on the master RTC registers located in the crystal power and clock domain. A special first-power-on reset establishes the default value of the master RTC registers when a voltage is first applied to the BATT pin (battery insertion).

For consistency across applications, it is recommended that the seconds timer should be referenced to January 1, 1980 at a 32-bit value of 0 (same epoch reference as PC) in applications that use it as a time-of-day clock. If the real-time clock function is not present on a specific chip, as indicated in the con-

trol and status register (HW_RTC_STAT_RTC_PRESENT), then no real-time epoch is maintained over power-down cycles.

23.4.1 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, "Correct Way to Soft Reset a Block," for additional information on using the SFTRST and CLKGATE bit fields.

23.5 Millisecond Resolution Timing Function

A millisecond counter facility is provided based on a 1-kHz signal derived from the 24-MHz clock. The count value is neither maintained nor incremented during power-down cycles. At each power-up, this register is set to its reset state. On each tick of the 1-kHz source, the milliseconds counter increments. With a 32-bit counter, a kernel can run up to 4,294,967,294 milliseconds or 49.7 days before it must deal with a counter wrap. The programmer can change the resolution of the millisecond counter to be 1, 2, 4, 8, or 16 milliseconds. This is programmed through bits in Persistent Register 0.

WARNING: When the 32.768-kHz or 32.000-kHz crystal oscillator is selected as the source for the seconds counter, an anomaly is created between the time intervals of the millisecond counter and the seconds counter. That is, the manufacturing tolerance of the two crystals are such that 1000 millisecond counter increments are not exactly one second as measured by the real-time clock seconds counter.

23.6 Alarm Clock Function

The alarm clock function allows an application to specify a future instant at which the chip should be awakened, i.e., if powered down, it can be powered up. The alarm clock setting is a CPU-accessible, 32-bit value that is continuously matched against the 32-bit real-time clock seconds counter. When the two values are equal, an alarm event is triggered. Persistent bits indicate whether an alarm event should power up the chip from its powered-down state. In addition to or instead of powering up the chip, the alarm event can also cause a CPU interrupt. Although these two functions can be enabled at the same time, one should remember that the CPU will only be interrupted if the chip is powered up at the time of the alarm event.

NOTE: If the alarm is set to power up the chip in the event of an alarm and such an event occurs, then the only record of the cause of the wake-up is located in the analog side. At power-up, the analog side registers are copied to the digital shadow registers and the ALARM_WAKE bit in the Persistent register 0 is visible in the digital shadow register. If an alarm wake event occurs while the chip is powered up, the ALARM_WAKE bit will not be set in the persistent register because the chip was not woken up.

The alarm must be "present" on an actual chip to perform this function (see the HW_RTC_STAT_ALARM_PRESENT bit description).

23.7 Watchdog Reset Function

The watchdog reset is a CPU-configurable device. It is programmed by software to generate a chip-wide reset after HW_RTC_WATCHDOG milliseconds. The watchdog generates this reset if software does not rewrite this register before this time elapses.

The watchdog timer decrements the register value once for every tick of the 1-kHz clock supplied from the RTC analog section (see Figure 23-1). The reset generated by the watchdog timer has no effect on the values retained in the master registers of the real-time clock seconds counter, alarm, or persistent registers (analog persistent storage).

The watchdog timer is initially disabled and set to count 4,294,967,295 milliseconds before generating a watchdog reset.

The watchdog timer does not run when the chip is in its powered-down state. Therefore, there is no master/shadow register pairing for the watchdog timer, and it must be reprogrammed after cycling power or resetting the block.

The watchdog timer must be “present” on an actual chip to perform this function (see the HW_RTC_STAT_WATCHDOG_PRESENT bit description).

23.8 Programmable Registers

This section describes the programmable registers of the real-time clock, including the watchdog register, alarm register, laser fuse registers, and persistent registers.

23.8.1 Real-Time Clock Control Register Description

HW_RTC_CTRL is the control register for the RTC.

HW_RTC_CTRL	0x000
HW_RTC_CTRL_SET	0x004
HW_RTC_CTRL_CLR	0x008
HW_RTC_CTRL_TOG	0x00C

Table 23-1. HW_RTC_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	RSVD0																							SUPPRESS_COPY2ANALOG	FORCE_UPDATE	WATCHDOGEN	ONEMSEC_IRQ	ALARM_IRQ	ONEMSEC_IRQ_EN	ALARM_IRQ_EN

Table 23-2. HW_RTC_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	1= Hold real time clock digital side in soft reset state. This bit has no effect on the RTC analog.
30	**CLKGATE**	RW	0x1	This bit must be set to zero for normal operation. When set to one it gates off the clocks to the block. This bit has no effect on RTC analog.
29:7	**RSVD0**	RO	0x0	Reserved, write only zeroes.
6	**SUPPRESS_COPY2ANALOG**	RW	0x0	This bit is used for diagnostic purposes. 1= suppress the automatic copy that normally occurs to the analog side, whenever a shadow register is written. 0= Normal operation. Use SCT writes to set, clear, or toggle.
5	**FORCE_UPDATE**	RW	0x1	This bit is how the software requests the update of the shadow registers from values in RTC analog. When software sets this bit, all eight of the shadow registers are updated from the corresponding values in the persistent registers in RTC analog. The state of this update operation is reflected on the STALEREGS bits on the STAT register, which are set to all ones upon an update request and are cleared by hardware as the update proceeds. Hardware clears this bit immediately after detecting it has been set. Software does not need to clear. Software must NOT look to the state of this bit to determine the status of the update operation, it must look to the STALEREG bits in the STAT register to determine when any given register has been updated and/or when the update operation is complete. Notice that the default value of this bit is 1, so that that a reset (either chip-wide or soft) always results in an update.
4	**WATCHDOGEN**	RW	0x0	1= Enable Watchdog Timer to force chip wide resets. Use SCT writes to set, clear, or toggle.
3	**ONEMSEC_IRQ**	RW	0x0	1= one millisecond interrupt request status. Use SCT writes to clear this interrupt status bit.
2	**ALARM_IRQ**	RW	0x0	1= Alarm Interrupt Status. Use SCT writes to clear this interrupt status bit.

Table 23-2. HW_RTC_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
1	**ONEMSEC_IRQ_EN**	RW	0x0	1= Enable one millisecond interrupt. Use SCT writes to set, clear, or toggle.
0	**ALARM_IRQ_EN**	RW	0x0	1= Enable Alarm Interrupt. Use SCT writes to set, clear, or toggle.

DESCRIPTION:

The contents of this register control the operation of the RTC portions implemented as an APBX peripheral running in the APBX clock domain. These functions only operate when the chip is in its full power up state.

EXAMPLE:

```
HW_RTC_CTRL_CLR(BM_RTC_CTRL_SFTRST); // remove the soft reset condition
HW_RTC_CTRL_CLR(BM_RTC_CTRL_CLKGATE); // enable clocks within the RTC
HW_RTC_CTRL_CLR(BM_RTC_CTRL_ALARM_IRQ); // reset the alarm interrupt by clearing its status bit
```

23.8.2 Real-Time Clock Status Register Description

HW_RTC_STAT is the status register for the RTC.

HW_RTC_STAT	0x010
HW_RTC_STAT_SET	0x014
HW_RTC_STAT_CLR	0x018
HW_RTC_STAT_TOG	0x01C

Table 23-3. HW_RTC_STAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RTC_PRESENT	ALARM_PRESENT	WATCHDOG_PRESENT	XTAL32000_PRESENT	XTAL32768_PRESENT	RSVD1			STALE_REGS								NEW_REGS								RSVD0							

Table 23-4. HW_RTC_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RTC_PRESENT**	RO	0x1	This read-only bit reads back a one if the RTC is present in the device.
30	**ALARM_PRESENT**	RO	0x1	This read-only bit reads back a one if the Alarm function is present in the device.
29	**WATCHDOG_PRESENT**	RO	0x1	This read-only bit reads back a one if the Watchdog Timer function is present in the device.
28	**XTAL32000_PRESENT**	RO	0x0	This read-only bit reads back a one if the 32.000-kHz crystal oscillator function is present in the device.

Table 23-4. HW_RTC_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
27	**XTAL32768_PRESENT**	RO	0x0	This read-only bit reads back a one if the 32.768-kHz crystal oscillator function is present in the device.
26:24	**RSVD1**	RO	0x0	Reserved, write only zeroes.
23:16	**STALE_REGS**	RO	0xFF	These read-only bits are set to one whenever the corresponding shadow register contents are older than the analog side contents. These bits are set by reset and cleared by the copy controller. They are also set by writing a one to the FORCE_UPDATE bit.
15:8	**NEW_REGS**	RO	0x00	These read-only bits are set to one whenever the corresponding shadow register contents are newer than the analog side contents. These bits are set by writing to the corresponding register and cleared by the copy controller.
7:0	**RSVD0**	RO	0x0	Reserved, write only zeroes.

DESCRIPTION:

This register reflects the current state of the RTC in terms of its enabled capabilities and the state of the persistent registers.

EXAMPLE:

```
while(HW_RTC_STAT.STALE_REGS !=0)
{
printf(" something is stale in one of the digital side registers

// the copy controller will copy analog registers to digtial registers as required,
// turning off staleregs bits as it goes about its business.
}
if(HW_RTC_STAT.WATCHDOG_PRESENT != 0) // then you can use the watchdog timer on this chip
```

23.8.3 Real-Time Clock Milliseconds Counter Description

The millisecond count register provides a reliable elapsed time reference to the kernel with millisecond resolution.

HW_RTC_MILLISECONDS	0x020
HW_RTC_MILLISECONDS_SET	0x024
HW_RTC_MILLISECONDS_CLR	0x028
HW_RTC_MILLISECONDS_TOG	0x02C

Table 23-5. HW_RTC_MILLISECONDS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

Table 23-6. HW_RTC_MILLISECONDS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COUNT**	RW	0x00000000	32 bit milliseconds counter.

DESCRIPTION:

HW_RTC_MILLISECONDS provides access to the 32-bit millisecond counter. This counter is not a shadow register, i.e., the contents of this register are not preserved over power-down states. This counter increments once per millisecond based on a pulse from RTC analog which is derived from the 24.0-MHz crystal clock. This 1-kHz source hence does not vary as the APBX clock frequency is changed.

The millisecond counter wraps at 4,294,967,294 milliseconds or 49.7 days.

EXAMPLE:

```
HW_RTC_MILLISECONDS_WR(0); // write an initial starting value to the milliseconds counter
Count = HW_RTC_MILLISECONDS_RD(); // read the current value of the milliseconds counter.
```

23.8.4 Real-Time Clock Seconds Counter Description

The real-time clock seconds counter is used to maintain real time for applications, even across certain chip power-down states.

HW_RTC_SECONDS	0x030
HW_RTC_SECONDS_SET	0x034
HW_RTC_SECONDS_CLR	0x038
HW_RTC_SECONDS_TOG	0x03C

Table 23-7. HW_RTC_SECONDS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

Table 23-8. HW_RTC_SECONDS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COUNT**	RW	0x00000000	Increments once per second.

DESCRIPTION:

HW_RTC_SECONDS provides access to the 32-bit real-time seconds counter.

Both the shadow register on the digital side and the analog side register update every second. When the chip enters the power-down state, the shadow register is powered down and loses its state value. When the chip powers up, the analog side register contents are automatically copied to the shadow register. The reset value of 0x0 for the digital side register is only visible until the copy controller overwrites with the value from the analog side. The analog side register resets to zero only upon power-on reset (POR), i.e., when the battery is first inserted or whenever a battery-less part is plugged into USB power or into a wall transformer.

Note that if a low frequency (32.000khz or 32.768khz) crystal is available in the system, then the seconds count and the milliseconds count will be derived from different clocks. Namely, the msec count is derived from the 24MHz crystal and the seconds count from the low-frequency crystal. This limits the precision of the relation between these two clocks.

EXAMPLE:

```
HW_RTC_SECONDS_WR(0); // write an initial value to the digital side. This value will
// be automatically copied to the analog side
rt_clock = HW_RTC_SECONDS_RD(); // read the 32 seconds counter value
```

23.8.5 Real-Time Clock Alarm Register Description

The 32-bit alarm value is matched against the 32-bit seconds counter to detect an alarm condition.

HW_RTC_ALARM	0x040
HW_RTC_ALARM_SET	0x044
HW_RTC_ALARM_CLR	0x048
HW_RTC_ALARM_TOG	0x04C

Table 23-9. HW_RTC_ALARM

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
VALUE																															

Table 23-10. HW_RTC_ALARM Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**VALUE**	RW	0x00000000	Seconds match-value used to trigger assertion of the RTC alarm.

DESCRIPTION:

The 32-bit alarm value can be used to awaken the chip from a power-down state or simply to cause an interrupt at a specific time.

When the chip enters the power-down state, the shadow register is powered down and loses its state value. When the chip powers up, the analog side register contents are automatically copied to the shadow register. The reset value of 0x0 for the digital side register is only visible until the copy controller overwrites with the value from the analog side. The analog side register resets to zero only upon power-on reset (POR), i.e., when the battery is first inserted or whenever a battery-less part is plugged into a power USB or into a wall transformer.

EXAMPLE:

```
HW_RTC_ALARM_WR(60); // generate rtc alarm after 60 seconds
```

23.8.6 Watchdog Timer Register Description

The 32-bit watch dog timer can be used to reset the chip if enabled and not adequately serviced.

HW_RTC_WATCHDOG	0x050
HW_RTC_WATCHDOG_SET	0x054
HW_RTC_WATCHDOG_CLR	0x058
HW_RTC_WATCHDOG_TOG	0x05C

Table 23-11. HW_RTC_WATCHDOG

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

Table 23-12. HW_RTC_WATCHDOG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**COUNT**	RW	0xffffffff	If the watchdog timer decrements to zero and the watchdog timer reset is enabled, then the chip will be reset. The watchdog timer decrements once per millisecond, when enabled.

DESCRIPTION:

The 32-bit watchdog timer will reset the chip upon decrementing to zero, if this function is enabled and present on the chip. The 1-kHz source is based on a msec pulse from RTC analog which is derived from the 24-MHz crystal oscillator and hence does not vary when the APBX clock is changed.

EXAMPLE:

```
HW_RTC_WATCHDOG_WR(10000); // reload the watchdog and keep it from resetting the chip
```

23.8.7 Persistent State Register 0 Description

The 32-bit persistent registers are used to retain certain control states during chip wide power-down states.

HW_RTC_PERSISTENT0	0x060
HW_RTC_PERSISTENT0_SET	0x064
HW_RTC_PERSISTENT0_CLR	0x068
HW_RTC_PERSISTENT0_TOG	0x06C

Table 23-13. HW_RTC_PERSISTENT0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SPARE_ANALOG														AUTO_RESTART	DISABLE_PSWITCH	LOWERBIAS		DISABLE_XTALOK		MSEC_RES				ALARM_WAKE	XTAL32_FREQ	XTAL32KHZ_PWRUP	XTAL24MHZ_PWRUP	LCK_SECS	ALARM_EN	ALARM_WAKE_EN	CLOCKSOURCE

Table 23-14. HW_RTC_PERSISTENT0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:18	**SPARE_ANALOG**	RW	0x0	This field is broken into the following fields: ADJ_POSLIMITBUCK - bits [31:28]: This field can be used to allow dcdc startup with lower Liion battery voltages. With the default value of 0x0, the dcdc converter will not startup below 2.83V, and increasing this value will lower the Liion startup voltage. It is not recommended to set a value higher than 0x6. The value of this field effects the minimum startup voltage as follows: 0x0=2.83V, 0x1=2.78V, 0x2=2.73V, 0x3=2.68V, 0x4=2.62V, 0x5=2.57V, 0x6=2.52V, 0x7-0xF=2.48V. SPARE_ANALOG - bits [27:20]: Reserved. RELEASE_GND - bit [19]: Set to one to remove the pulldown resistors on the headphone outputs. This bit should be cleared after powering down the headphone, and before the chip is powered down for optimum anti-pop performance. ENABLE_LRADC_PWRUP - bit [18]: Set to one to enable a low voltage on LRADC0 to powerup the chip. This is to support powerup via open drain pulldowns connected to LRADC0. Ths requires external circuitry and this bit should not be set unless the correct circuitry is present. Freescale will provide information on the external circuitry, if this functionality is required.
17	**AUTO_RESTART**	RW	0x0	Set to one to enable the chip to automatically power up approximately 180 ms after powering down.
16	**DISABLE_PSWITCH**	RW	0x0	Disables the pswitch pin startup functionality unless the voltage on the pswitch pin goes above the VDDXTAL pin voltage by a threshold voltage. Typically, this voltage is created by pulling pswitch up with a current limiting resistor to a higher voltage, such as the Liion battery voltage.
15:14	**LOWERBIAS**	RW	0x0	Reduce bias current of 24mhz crystal. b00: nominal, b01: -25%, b10: -25%, b11: -50%,
13	**DISABLE_XTALOK**	RW	0x0	Set to one to disable the circuit that resets the chip if 24-MHz frequency falls below 2 MHz. The circuit defaults to enabled and will power down the device if the 24-MHz stop oscillating for any reason.
12:8	**MSEC_RES**	RW	0x1	This bit field encodes the value of the millisecond count resolution in a one-hot format. Resolutions supported are: 1, 2, 4, 8, and 16 msec. The bitfield directly gives the resolution without need for decode.
7	**ALARM_WAKE**	RW	0x0	Set when the chip is powered up by an alarm event from rtc_ana. Can then be cleared by software as desired.
6	**XTAL32_FREQ**	RW	0x0	If CLOCKSOURCE (bit 0 of this register) is one, then this bit gives the exact frequency of the 32kHz crystal. If this bit is zero, the frequency is 32.768kHZ, if it is one, the frequency is 32.000kHz. If CLOCKSOURCE is zero, the value of this bit is immaterial.

Table 23-14. HW_RTC_PERSISTENT0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
5	**XTAL32KHZ_PWRUP**	RW	0x0	Set to one to power up the 32kHz crystal oscillator. Set to zero to disable the oscillator. This bit controls the oscillator at all times (chip powered on or not).
4	**XTAL24MHZ_PWRUP**	RW	0x0	Set to one to keep the 24.0-MHz crystal oscillator powered up while the chip is powered down. Set to zero to disable during chip power down. Note: The oscillator is always on while the chip is powered on.
3	**LCK_SECS**	RW	0x0	Set to one to lock down the seconds count. Once this bit is written with a 1, the user will not be able to either write to the seconds register or to change this bit back to a zero -- except by removing the battery.
2	**ALARM_EN**	RW	0x0	Set this bit to one to enable the detection of an alarm event. This bit must be turned on before an alarm event can awaken a powered-down device, or before it can generate an alarm interrupt to a powered-up CPU. When the alarm is not present in the device (as indicated by the fuse bits) the copy of shadow0 to persistent0 will not allow bit[2] to be written and persistent bit[2] will always read back 0, regardless of the values in the shadow register.
1	**ALARM_WAKE_EN**	RW	0x0	This bit is set to one to upon the arrival of an alarm event that powers up the chip. ALARM_EN must be set to one to enable the detection of an alarm event. This bit is reset by writing a zero directly to the shadow register, which causes the copy controller to move it across to the analog domain.
0	**CLOCKSOURCE**	RW	0x0	Set to one to select the 32-kHz crystal oscillator as the source for the 32-kHz clock domain used by the RTC analog domain circuits. Set to zero to select the 24-MHz crystal oscillator as the source for generating the 32-kHz clock domain used by the RTC analog domain circuits.

DESCRIPTION:

The general persistent bits are available for software use. The register initalizes to a known reset pattern. The copy controller overwrites the digital reset values very soon after power on, but not in zero time.

EXAMPLE:

```
HW_RTC_PERSISTENT0_SET(BM_RTC_PERSISTENT0_ALARM_WAKE_EN); // wake up the chip if the alarm event occurs
HW_RTC_PERSISTENT0_SET(BM_RTC_PERSISTENT0_CLOCKSOURCE); // select the 32KHz oscillator as the source for the RTC analog clock
```

23.8.8 Persistent State Register 1 Description

The 32-bit persistent registers are used to retain certain control states during chip wide power-down states.

HW_RTC_PERSISTENT1	0x070
HW_RTC_PERSISTENT1_SET	0x074
HW_RTC_PERSISTENT1_CLR	0x078
HW_RTC_PERSISTENT1_TOG	0x07C

Table 23-15. HW_RTC_PERSISTENT1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
GENERAL																															

Table 23-16. HW_RTC_PERSISTENT1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**GENERAL**	RW	0x0000	Firmware use, defined as follows: ENUMERATE_500MA_TWICE = 0x1000 Enumerate at 500mA twice before dropping back to 100mA. USB_BOOT_PLAYER_MODE = 0x0800 Boot to player when connected to USB. SKIP_CHECKDISK = 0x0400 Run Checkdisk flag. USB_LOW_POWER_MODE = 0x0200 USB Hi/Lo Current select. OTG_HNP_BIT = 0x0100 HNP has been required if set to one. OTG_ATL_ROLE_BIT = 0x0080 USB role.

DESCRIPTION:

The register initalizes to a known reset pattern. The copy controller overwrites this digital reset value very soon after power on but not in zero time.

EXAMPLE:

```
HW_RTC_PERSISTENT1_WR(0x12345678); // this write will ultimately push data to the analog side via the copy controller
```

23.8.9 Persistent State Register 2 Description

The 32-bit persistent registers are used to retain certain control states during chip-wide power-down states.

HW_RTC_PERSISTENT2	0x080
HW_RTC_PERSISTENT2_SET	0x084
HW_RTC_PERSISTENT2_CLR	0x088
HW_RTC_PERSISTENT2_TOG	0x08C

Table 23-17. HW_RTC_PERSISTENT2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
GENERAL																															

Table 23-18. HW_RTC_PERSISTENT2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**GENERAL**	RW	0x00000000	FIRMWARE/SOFTWARE

DESCRIPTION:

The register initalizes to a known reset pattern. The copy controller overwrites this digital reset value very soon after power on but not in zero time.

EXAMPLE:

```
HW_RTC_PERSISTENT2_WR(0x12345678); // this write will ultimately push data to the analog side via
the copy controller
```

23.8.10 Persistent State Register 3 Description

The 32-bit persistent registers are used to retain certain control states during chip-wide power-down states.

HW_RTC_PERSISTENT3	0x090
HW_RTC_PERSISTENT3_SET	0x094
HW_RTC_PERSISTENT3_CLR	0x098
HW_RTC_PERSISTENT3_TOG	0x09C

Table 23-19. HW_RTC_PERSISTENT3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
GENERAL																															

Table 23-20. HW_RTC_PERSISTENT3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**GENERAL**	RW	0x00000000	FIRMWARE/SOFTWARE

DESCRIPTION:

The register initalizes to a known reset pattern. The copy controller overwrites this digital reset value very soon after power on but not in zero time.

EXAMPLE:

```
HW_RTC_PERSISTENT3_WR(0x12345678); // this write will ultimately push data to the analog side via
the copy controller
```

23.8.11 Persistent State Register 4 Description

The 32-bit persistent registers are used to retain certain control states during chip-wide power-down states.

HW_RTC_PERSISTENT4	0x0A0
HW_RTC_PERSISTENT4_SET	0x0A4
HW_RTC_PERSISTENT4_CLR	0x0A8
HW_RTC_PERSISTENT4_TOG	0x0AC

Table 23-21. HW_RTC_PERSISTENT4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
GENERAL																															

Table 23-22. HW_RTC_PERSISTENT4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**GENERAL**	RW	0x00000000	FIRMWARE/SOFTWARE

DESCRIPTION:

The register initalizes to a known reset pattern. The copy controller overwrites this digital reset value very soon after power on but not in zero time.

EXAMPLE:

```
HW_RTC_PERSISTENT4_WR(0x12345678); // this write will ultimately push data to the analog side via
the copy controller
```

23.8.12 Persistent State Register 5 Description

The 32-bit persistent registers are used to retain certain control states during chip-wide power-down states.

HW_RTC_PERSISTENT5	0x0B0
HW_RTC_PERSISTENT5_SET	0x0B4
HW_RTC_PERSISTENT5_CLR	0x0B8
HW_RTC_PERSISTENT5_TOG	0x0BC

Table 23-23. HW_RTC_PERSISTENT5

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
GENERAL																															

Table 23-24. HW_RTC_PERSISTENT5 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**GENERAL**	RW	0x00000000	FIRMWARE/SOFTWARE

DESCRIPTION:

The register initalizes to a known reset pattern. The copy controller overwrites this digital reset value very soon after power on but not in zero time.

EXAMPLE:

```
HW_RTC_PERSISTENT5_WR(0x12345678); // this write will ultimately push data to the analog side via
the copy controller
```

23.8.13 Real-Time Clock Debug Register Description

This 32-bit register provides debug read access to various internal states for diagnostic purposes.

HW_RTC_DEBUG	0x0C0
HW_RTC_DEBUG_SET	0x0C4
HW_RTC_DEBUG_CLR	0x0C8
HW_RTC_DEBUG_TOG	0x0CC

Table 23-25. HW_RTC_DEBUG

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD0																														WATCHDOG_RESET_MASK	WATCHDOG_RESET

Table 23-26. HW_RTC_DEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:2	**RSVD0**	RO	0x0	Debug read-only view of various state machine bits.
1	**WATCHDOG_RESET_MASK**	RW	0x0	When set, mask the reset generation by the watchdog timer for testing purposes.
0	**WATCHDOG_RESET**	RO	0x0	Reflects the state of the watchdog reset. Used for testing purposes so the watchdog can be tested without resetting part. When set, Watchdog reset is asserted.

DESCRIPTION:

Read-only view into the internals of the digital side of the RTC for diagnostic purposes.

EXAMPLE:

```
DebugValue = HW_RTC_DEBUG_RD(); // read debug register value
```

23.8.14 Real-Time Clock Version Register Description

Version register.

HW_RTC_VERSION 0x0D0

Table 23-27. HW_RTC_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 23-28. HW_RTC_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x02	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x0	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0	Fixed read-only value reflecting the stepping of the RTL version.

EXAMPLE:

```
VersionValue = HW_RTC_VERSION_RD(); // read debug register value
```

RTC Block v2.0, Revision 1.75

Chapter 24
Pulse-Width Modulator (PWM) Controller

This chapter describes the pulse-width modulator (PWM) controller included on the i.MX23 and how to use it. Programmable registers are described in Section 24.4, "Programmable Registers."

24.1 Overview

The i.MX23 contains five PWM output controllers that can be used in place of GPIO pins. Applications include LED and backlight brightness control. Independent output control of each phase allows 0, 1, or high-impedance to be independently selected for the active and inactive phases. Individual outputs can be run in lock step with guaranteed non-overlapping portions for differential drive applications.

Figure 24-1 shows the block diagram of the PWM controller. The controller does not use DMA. Initial values of Period, Active, and Inactive widths are set for each desired channel. The outputs are selected by phase and then the desired PWM channels are simultaneously enabled. This effectively launches the PWM outputs to autonomously drive their loads without further intervention.

24.2 Operation

Each PWM channel has two control registers that are used to specify the channel output: HW_PWM_ACTIVEn and HW_PWM_PERIODn.

When programming a channel, it is important to remember that there is an order dependence for register writes.

- The HW_PWM_ACTIVEn register must be written first, followed by HW_PWM_PERIODn.
- If the order is reversed, the parameters written to the HW_PWM_ACTIVEn register will not take effect in the hardware.

The hardware waits for a HW_PWM_PERIODn register write to update the hardware with the values in both registers. This register write order dependence allows smooth on-the-fly reprogramming of the channel. Also, when the user reprograms the channel in this manner, the new register values will not take effect until the beginning of a new output period. This eliminates the potential for output glitches that could occur if the registers were updated while the channel was enabled and in the middle of a cycle.

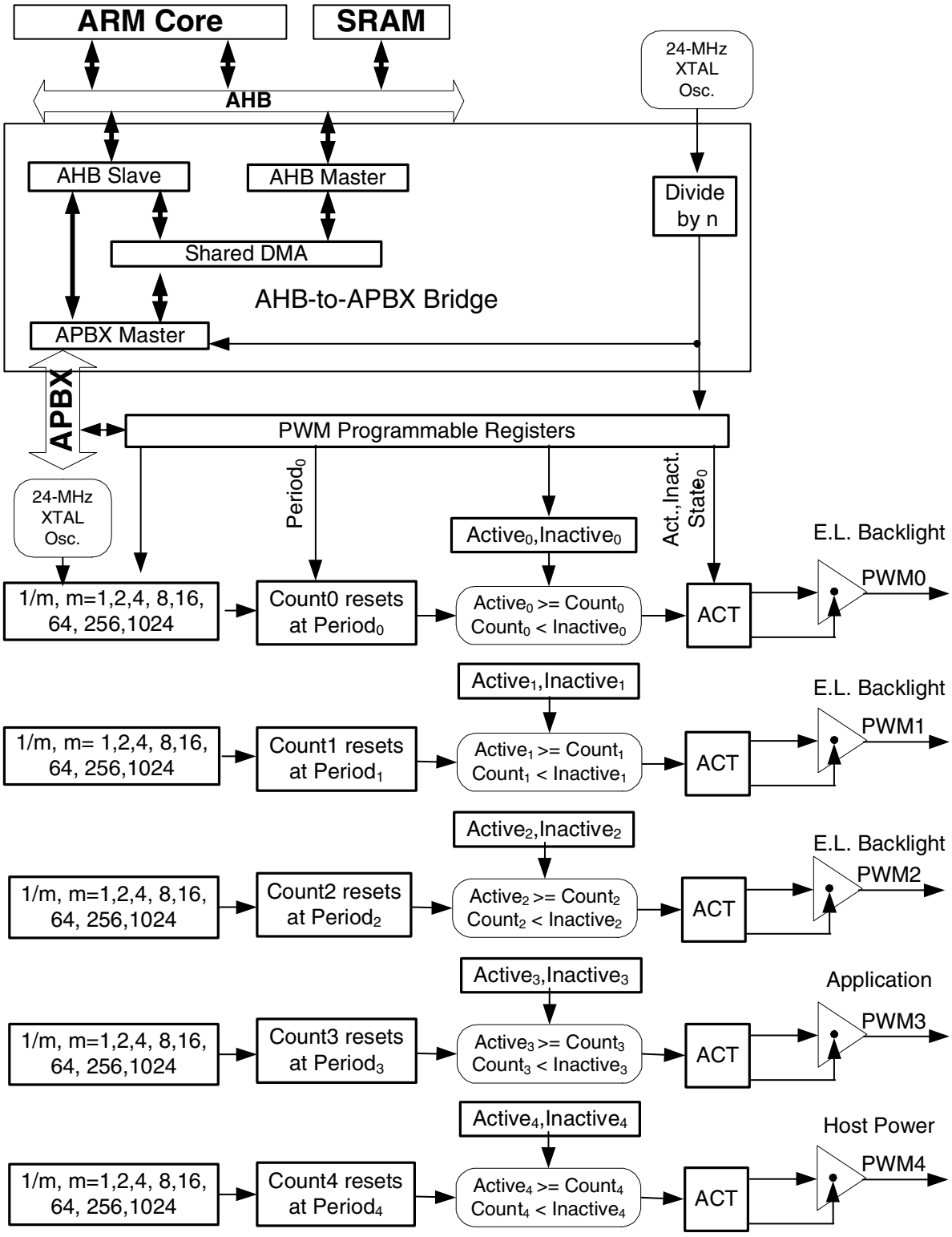


Figure 24-1. Pulse-Width Modulation Controller (PWM) Block Diagram

Each channel has a dedicated internal 16-bit counter that increments once for each divided clock period presented from the clock divider:

- The internal counter resets when it reaches the value stored in the channel control registers, e.g., HW_PWM_PERIOD0_PERIOD.
- The Active flip-flop is set to 1 when the internal counter reaches the value stored in HW_PWM_ACTIVE0_ACTIVE.
- It remains high until the internal counter exceeds the value stored in HW_PWM_ACTIVE0_INACTIVE.

These two values define the starting and ending points for the logically *"active"* portion of the waveform. As shown in Figure 24-2, the actual state on the output for each phase, e.g., active or inactive, is completely controlled by the active and inactive state values in the channel control registers.

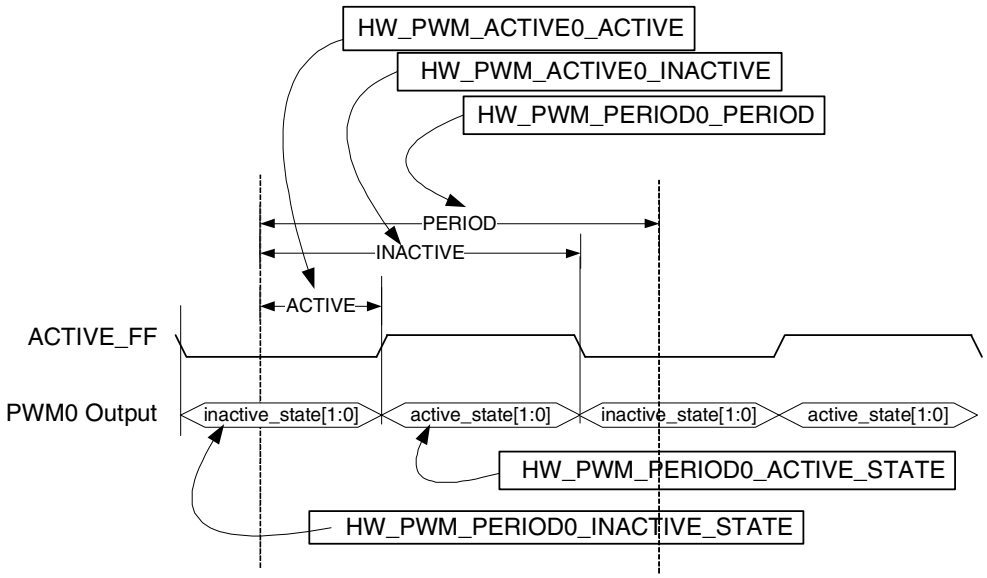


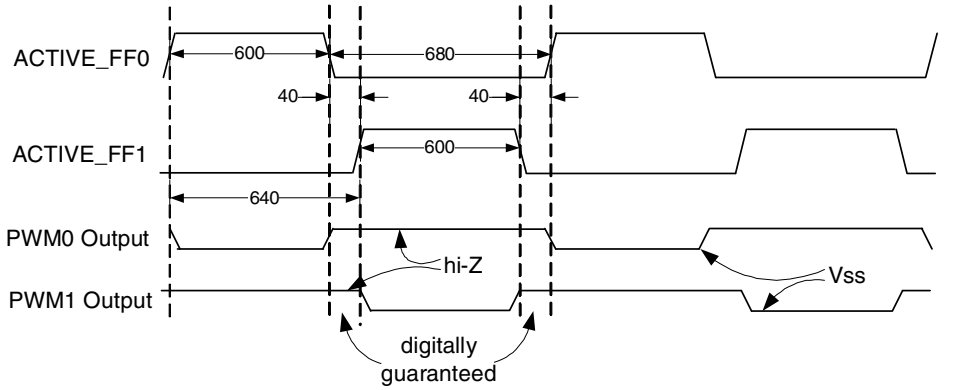
Figure 24-2. PWM Output Example

The actual values obtainable on the output are shown in Figure 24-2. Notice that one possible state is to turn off the output driver to provide a high-Z output. This is useful for external circuits that drive E.L. backlights and for direct drive of LEDs.

By setting up two channels in lock step and by setting their low and high states to opposite values, one can generate a differential signal pair that alternates between pulling to Vss and floating to high-Z. By creating an appropriate offset in the settings of the two channels with the same period and the same enables, one can generate differential drive pulses with digitally guaranteed non-overlapping intervals suitable for controlling high-voltage switches.

In Figure 24-3, a differential pair is established using Channel 0 and Channel 1. The period is set for 1280 divided clocks for both channels. All active phases are set for 600 divided clocks. There is a 40 divided clock guaranteed off-time between each active phase. Since this is based on a crystal oscillator, it is a

very stable non-overlapping period. The total period is also a very stable crystal-oscillator-based time interval. In this example, the active phases are pulled to Vss (ground), while the inactive phases are allowed to float to a high-Z state.



HW_PWM_PERIOD0_PERIOD = 1280	HW_PWM_PERIOD1= 1280
HW_PWM_ACTIVE0_ACTIVE = 0	HW_PWM_ACTIVE1 = 640
HW_PWM_ACTIVE0_INACTIVE = 600	HW_PWM_INACTIVE1 = 1240
HW_PWM_PERIOD0_ACTIVE_STATE = 10	HW_PWM_ACTIVE1_STATE = 10
HW_PWM0BR_INACTIVE_STATE = 00	HW_PWM_INACTIVE1_STATE = 00

Figure 24-3. PWM Differential Output Pair Example

Figure 24-4 shows the generation of the PWM Channel 3 output. This channel controls the output pin when PWM control is selected in PINCTRL block and HW_PWM_CTRL_PWM3_ENABLE is set to 1. The output pin can be set to a 0, a 1, or left to float in the high-impedance state. These choices can be made independently for either the active or inactive phase of the output.

24.2.1 Multi-Chip Attachment Mode

The multi-chip attachment mode (MATT) allows a 24-MHz or 32KHz crystal clock that is an input to the i.MX23 to be routed to the PWM output pins. In this case, the normal PWM programming parameters (e.g., PERIOD, ACTIVE, etc.) are ignored. This mode allows for supplying and controlling the crystal clock for external application interfaces, as shown in Figure 24-4.

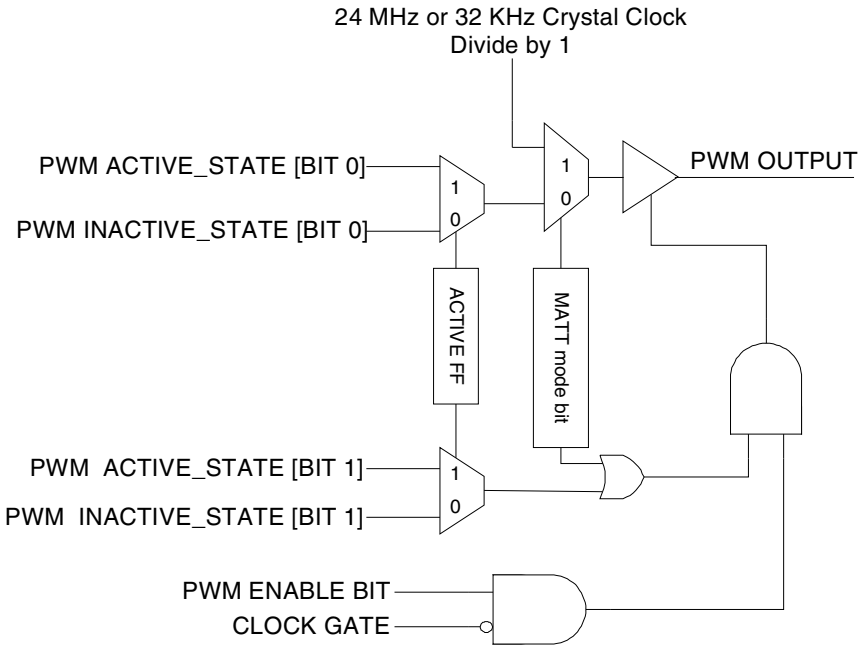


Figure 24-4. PWM Output Driver

24.2.2 Channel 2 Analog Enable Function

The output generation for channel 2 is slightly different than shown in Figure 24-4. In this channel, there is an additional enable that is controlled from the analog LRADC block. This signal is synchronized in the XTAL domain and ANDed with the PWM ENABLE bit. So, in this case, either enable source can disable the output. Also, this analog enable control signal can be disabled through the PWM2_ANA_CTRL_ENABLE bit in the HW_PWM_CTRL register. When disabled, Channel 2 behaves identically to the other channels.

24.2.3 Channel Output Cutoff Using Module Clock Gate

Whenever the clkgate is enabled (gated) the output from all PWM channels is hi-Z. If the clock gate is asserted while PWM is enabled and generating an output signal the output is immediately disabled. This will not affect the current state, programming or enables of the pwm channels themselves. When the clkgate is de-asserted, the PWM outputs will resume according to their programmed parameters and current states. Therefore glitches on the enabled channel outputs will likely occur when the clkgate state is changed. All 5 channels will function identically in this regard.

24.3 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, "Correct Way to Soft Reset a Block," for additional information on using the SFTRST and CLKGATE bit fields.

24.4 Programmable Registers

The following registers are available for CPU programmer access and control of the PWM controller.

24.4.1 PWM Control and Status Register Description

The PWM Control and Status Register specifies the reset state, availability, and the enables for the five PWM units.

HW_PWM_CTRL	0x000
HW_PWM_CTRL_SET	0x004
HW_PWM_CTRL_CLR	0x008
HW_PWM_CTRL_TOG	0x00C

Table 24-1. HW_PWM_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	PWM4_PRESENT	PWM3_PRESENT	PWM2_PRESENT	PWM1_PRESENT	PWM0_PRESENT	RSRVD1																		OUTPUT_CUTOFF_EN	PWM2_ANA_CTRL_ENABLE	PWM4_ENABLE	PWM3_ENABLE	PWM2_ENABLE	PWM1_ENABLE	PWM0_ENABLE

Table 24-2. HW_PWM_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	This bit must be cleared to 0 for normal operation. When set to 1, it forces a block-wide reset.
30	**CLKGATE**	RW	0x1	This bit must be cleared to 0 for normal operation. When set to 1, it gates off the clocks to the block.
29	**PWM4_PRESENT**	RO	0x1	0 = PWM4 is not present in this product.
28	**PWM3_PRESENT**	RO	0x1	0 = PWM3 is not present in this product.
27	**PWM2_PRESENT**	RO	0x1	0 = PWM2 is not present in this product.
26	**PWM1_PRESENT**	RO	0x1	0 = PWM1 is not present in this product.
25	**PWM0_PRESENT**	RO	0x1	0 = PWM0 is not present in this product.
24:7	**RSRVD1**	RO	0x0	Reserved.

Table 24-2. HW_PWM_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
6	**OUTPUT_CUTOFF_EN**	RW	0x0	When asserted this bit enables the block to automatically Hi-Z state the outputs whenever the clkgate is asserted. The default is enabled.
5	**PWM2_ANA_CTRL_ENABLE**	RW	0x0	Setting this bit to 1 enables PWM Channel 2 output to be enabled by analog circuitry outside this block much the way the PWM2_ENABLE bit controls it. Note that the PWM2_ENABLE bit must still be set and the channel must still be programmed normally for the PWM output to cycle. The PWM2_ENABLE bit acts like a master switch in this context.
4	**PWM4_ENABLE**	RW	0x0	Enables PWM channel 4 to begin cycling when set to 1. To enable PWM4 onto the output pin, the pin control registers must programmed accordingly.
3	**PWM3_ENABLE**	RW	0x0	Enables PWM channel 3 to begin cycling when set to 1. To enable PWM3 onto the output pin, the pin control registers must programmed accordingly.
2	**PWM2_ENABLE**	RW	0x0	Enables PWM channel 2 to begin cycling when set to 1. To enable PWM2 onto the output pin, the pin control registers must programmed accordingly.
1	**PWM1_ENABLE**	RW	0x0	Enables PWM channel 1 to begin cycling when set to 1. To enable PWM1 onto the output pin, the pin control registers must programmed accordingly.
0	**PWM0_ENABLE**	RW	0x0	Enables PWM channel 0 to begin cycling when set to 1. To enable PWM0 onto the output pin, the pin control registers must programmed accordingly.

DESCRIPTION:

This register specifies the reset state, availability, and the enables for the five PWM units.

EXAMPLE:

```
HW_PWM_CTRL_WR(0x0000001f); // Enable all channels
```

24.4.2 PWM Channel 0 Active Register Description

The PWM Channel 0 Active Register specifies the active time and inactive time for Channel 0.

HW_PWM_ACTIVE0	0x010
HW_PWM_ACTIVE0_SET	0x014
HW_PWM_ACTIVE0_CLR	0x018
HW_PWM_ACTIVE0_TOG	0x01C

Table 24-3. HW_PWM_ACTIVE0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
INACTIVE																ACTIVE															

Table 24-4. HW_PWM_ACTIVE0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**INACTIVE**	RW	0x0	Number of divided XTAL clock cycles to count from the beginning of the period before changing the output from the active state to the inactive state. The internal count of the channel is compared for greater than this value to change to the inactive state.
15:0	**ACTIVE**	RW	0x0	Number of divided XTAL clock cycles to count from the beginning of the period before changing the output to the active state. The internal count of the channel is compared for greater than this value to change to the active state.

DESCRIPTION:

This register contains the active time and inactive time programming parameters for Channel 0.

EXAMPLE:

```
HW_PWM_ACTIVEn_WR(0, 0x000000ff); // Set active and inactive counts
```

24.4.3 PWM Channel 0 Period Register Description

The PWM Channel 0 Period Register specifies the multi-chip attachment mode, clock divider value, active high, low values and period.

HW_PWM_PERIOD0	0x020
HW_PWM_PERIOD0_SET	0x024
HW_PWM_PERIOD0_CLR	0x028
HW_PWM_PERIOD0_TOG	0x02C

Table 24-5. HW_PWM_PERIOD0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2							MATT_SEL	MATT	CDIV			INACTIVE_STATE		ACTIVE_STATE		PERIOD																

Table 24-6. HW_PWM_PERIOD0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSRVD2**	RO	0x0	Reserved.
24	**MATT_SEL**	RW	0x0	Multichip Attachment Mode clock select. When the MATT bit is asserted this bit selects which clock to output. 0: 32 kHz, 1: 24 MHz.
23	**MATT**	RW	0x0	Multichip Attachment Mode. This bit overrides the normal signal generation parameters and enables either the 24 MHz or 32 kHz crystal clock on the PWM0 output pin for inter-chip signaling.

Table 24-6. HW_PWM_PERIOD0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
22:20	**CDIV**	RW	0x0	Clock divider ratio to apply to the crystal clock frequency (24.0MHz) that times the PWM output signal. DIV_1 = 0x0 Divide by 1. DIV_2 = 0x1 Divide by 2. DIV_4 = 0x2 Divide by 4. DIV_8 = 0x3 Divide by 8. DIV_16 = 0x4 Divide by 16. DIV_64 = 0x5 Divide by 64. DIV_256 = 0x6 Divide by 256. DIV_1024 = 0x7 Divide by 1024.
19:18	**INACTIVE_STATE**	RW	0x0	The logical inactive state that is mapped to the PWM output signal. Note that the undefined state of 0x1 is mapped to hi-Z. HI_Z = 0x0 Inactive state sets PWM output to high-impedance. 0 = 0x2 Inactive state sets PWM output to 0 (low). 1 = 0x3 Inactive state sets PWM output to 1 (high).
17:16	**ACTIVE_STATE**	RW	0x0	The logical active state is mapped to the PWM output signal. Note that the undefined state of 0x1 is mapped to high-impedance. HI_Z = 0x0 Active state sets PWM output to high-impedance. 0 = 0x2 Active state sets PWM output to 0 (low). 1 = 0x3 Active state sets PWM output to 1 (high).
15:0	**PERIOD**	RW	0x0	Number of divided XTAL clock cycles in the entire period of the PWM waveform, minus 1. For example, to obtain 6 clock cycles in the actual period, set this field to 5.

DESCRIPTION:

This register contains the programming paramters for multi-chip attachment mode, clock divider value, active high, low values and period for channel 0.

EXAMPLE:

```
HW_PWM_PERIODn_WR(0, 0x00000b1f); // Set up period and active/inactive output states
```

24.4.4 PWM Channel 1 Active Register Description

The PWM Channel 1 Active Register specifies the active time and inactive time for Channel 1.

HW_PWM_ACTIVE1	0x030
HW_PWM_ACTIVE1_SET	0x034
HW_PWM_ACTIVE1_CLR	0x038
HW_PWM_ACTIVE1_TOG	0x03C

Table 24-7. HW_PWM_ACTIVE1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
INACTIVE																ACTIVE															

Table 24-8. HW_PWM_ACTIVE1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**INACTIVE**	RW	0x0	Number of divided XTAL clock cycles to count from the beginning of the period before changing the output from the active state to the inactive state. The internal count of the channel is compared for greater than this value to change to the inactive state.
15:0	**ACTIVE**	RW	0x0	Number of divided XTAL clock cycles to count from the beginning of the period before changing the output to the active state. The internal count of the channel is compared for greater than this value to change to the active state.

DESCRIPTION:

The PWM Active register specifies the active time and inactive time for channel 1.

EXAMPLE:

```
HW_PWM_ACTIVEn_WR(1, 0x000000ff); // Set active and inactive counts
```

24.4.5 PWM Channel 1 Period Register Description

The PWM Channel 1 Period Register specifies the multi-chip attachment mode, clock divider value, active high, low values and period.

HW_PWM_PERIOD1	0x040
HW_PWM_PERIOD1_SET	0x044
HW_PWM_PERIOD1_CLR	0x048
HW_PWM_PERIOD1_TOG	0x04C

Table 24-9. HW_PWM_PERIOD1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2							MATT_SEL	MATT	CDIV			INACTIVE_STATE		ACTIVE_STATE		PERIOD																

Table 24-10. HW_PWM_PERIOD1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSRVD2**	RO	0x0	Reserved.
24	**MATT_SEL**	RW	0x0	Multichip Attachment Mode clock select. When the MATT bit is asserted this bit selects which clock to output. 0: 32 kHz, 1: 24 MHz.
23	**MATT**	RW	0x0	Multichip Attachment Mode. This bit overrides the normal signal generation parameters and enables either the 24 MHz or 32 kHz crystal clock on the PWM1 output pin for inter-chip signaling.

Table 24-10. HW_PWM_PERIOD1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
22:20	**CDIV**	RW	0x0	Clock divider ratio to apply to the crystal clock frequency (24.0 MHz) that times the PWM output signal. DIV_1 = 0x0 Divide by 1. DIV_2 = 0x1 Divide by 2. DIV_4 = 0x2 Divide by 4. DIV_8 = 0x3 Divide by 8. DIV_16 = 0x4 Divide by 16. DIV_64 = 0x5 Divide by 64. DIV_256 = 0x6 Divide by 256. DIV_1024 = 0x7 Divide by 1024.
19:18	**INACTIVE_STATE**	RW	0x0	The logical inactive state that is mapped to the PWM output signal. Note that the undefined state of 0x1 is mapped to high-impedance. HI_Z = 0x0 Inactive state sets PWM output to high-impedance. 0 = 0x2 Inactive state sets PWM output to 0 (low). 1 = 0x3 Inactive state sets PWM output to 1 (high).
17:16	**ACTIVE_STATE**	RW	0x0	The logical active state is mapped to the PWM output signal. Note that the undefined state of 0x1 is mapped to high-impedance. HI_Z = 0x0 Active state sets PWM output to high-impedance. 0 = 0x2 Active state sets PWM output to 0 (low). 1 = 0x3 Active state sets PWM output to 1 (high).
15:0	**PERIOD**	RW	0x0	Number of divided XTAL clock cycles in the entire period of the PWM waveform, minus 1. For example, to obtain 6 clock cycles in the actual period, set this field to 5.

DESCRIPTION:

This register contains the programming paramters for multi-chip attachment mode, clock divider value, active high, low values and period for channel 1.

EXAMPLE:

```
HW_PWM_PERIODn_WR(1, 0x00000b1f); // Set up period and active/inactive output states
```

24.4.6 PWM Channel 2 Active Register Description

The PWM Channel 2 Active Register specifies the active time and inactive time for Channel 2.

HW_PWM_ACTIVE2	0x050
HW_PWM_ACTIVE2_SET	0x054
HW_PWM_ACTIVE2_CLR	0x058
HW_PWM_ACTIVE2_TOG	0x05C

Table 24-11. HW_PWM_ACTIVE2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
INACTIVE																ACTIVE															

Table 24-12. HW_PWM_ACTIVE2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**INACTIVE**	RW	0x0	Number of divided XTAL clock cycles to count from the beginning of the period before changing the output from the active state to the inactive state. The internal count of the channel is compared for greater than this value to change to the inactive state.
15:0	**ACTIVE**	RW	0x0	Number of divided XTAL clock cycles to count from the beginning of the period before changing the output to the active state. The internal count of the channel is compared for greater than this value to change to the active state.

DESCRIPTION:

This register contains the active time and inactive time programming parameters for Channel 2.

EXAMPLE:

```
HW_PWM_ACTIVEn_WR(2, 0x000000ff); // Set active and inactive counts
```

24.4.7 PWM Channel 2 Period Register Description

The PWM Channel 2 Period Register specifies the multi-chip attachment mode, clock divider value, active high, low values and period.

HW_PWM_PERIOD2	0x060
HW_PWM_PERIOD2_SET	0x064
HW_PWM_PERIOD2_CLR	0x068
HW_PWM_PERIOD2_TOG	0x06C

Table 24-13. HW_PWM_PERIOD2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2							MATT_SEL	MATT	CDIV			INACTIVE_STATE		ACTIVE_STATE		PERIOD																

Table 24-14. HW_PWM_PERIOD2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSRVD2**	RO	0x0	Reserved.
24	**MATT_SEL**	RW	0x0	Multichip Attachment Mode clock select. When the MATT bit is asserted this bit selects which clock to output. 0: 32 kHz, 1: 24 MHz.
23	**MATT**	RW	0x0	Multichip Attachment Mode. This bit overrides the normal signal generation parameters and enables either the 24 MHz or 32 kHz crystal clock on the PWM2 output pin for inter-chip signaling.

Table 24-14. HW_PWM_PERIOD2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
22:20	**CDIV**	RW	0x0	Clock divider ratio to apply to the crystal clock frequency (24.0 MHz) that times the PWM output signal. DIV_1 = 0x0 Divide by 1. DIV_2 = 0x1 Divide by 2. DIV_4 = 0x2 Divide by 4. DIV_8 = 0x3 Divide by 8. DIV_16 = 0x4 Divide by 16. DIV_64 = 0x5 Divide by 64. DIV_256 = 0x6 Divide by 256. DIV_1024 = 0x7 Divide by 1024.
19:18	**INACTIVE_STATE**	RW	0x0	The logical inactive state that is mapped to the PWM output signal. Note that the undefined state of 0x1 is mapped to high-impedance. HI_Z = 0x0 Inactive state sets PWM output to high-impedance. 0 = 0x2 Inactive state sets PWM output to 0 (low). 1 = 0x3 Inactive state sets PWM output to 1 (high).
17:16	**ACTIVE_STATE**	RW	0x0	The logical active state is mapped to the PWM output signal. Note that the undefined state of 0x1 is mapped to high-impedance. HI_Z = 0x0 Active state sets PWM output to high-impedance. 0 = 0x2 Active state sets PWM output to 0 (low). 1 = 0x3 Active state sets PWM output to 1 (high).
15:0	**PERIOD**	RW	0x0	Number of divided XTAL clock cycles in the entire period of the PWM waveform, minus 1. For example, to obtain 6 clock cycles in the actual period, set this field to 5.

DESCRIPTION:

This register contains the programming paramters for multi-chip attachment mode, clock divider value, active high, low values and period for channel 2.

EXAMPLE:

```
HW_PWM_PERIODn_WR(2, 0x00000b1f); // Set up period and active/inactive output states
```

24.4.8 PWM Channel 3 Active Register Description

The PWM Channel 3 Active Register specifies the active time and inactive time for Channel 3.

HW_PWM_ACTIVE3	0x070
HW_PWM_ACTIVE3_SET	0x074
HW_PWM_ACTIVE3_CLR	0x078
HW_PWM_ACTIVE3_TOG	0x07C

Table 24-15. HW_PWM_ACTIVE3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
INACTIVE																ACTIVE															

Table 24-16. HW_PWM_ACTIVE3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**INACTIVE**	RW	0x0	Number of divided XTAL clock cycles to count from the beginning of the period before changing the output from the active state to the inactive state. The internal count of the channel is compared for greater than this value to change to the inactive state.
15:0	**ACTIVE**	RW	0x0	Number of divided XTAL clock cycles to count from the beginning of the period before changing the output to the active state. The internal count of the channel is compared for greater than this value to change to the active state.

DESCRIPTION:

This register contains the active time and inactive time programming parameters for Channel 3.

EXAMPLE:

```
HW_PWM_ACTIVEn_WR(3, 0x000000ff); // Set active and inactive counts
```

24.4.9 PWM Channel 3 Period Register Description

The PWM Channel 3 Period Register specifies the multi-chip attachment mode, clock divider value, active high, low values and period.

HW_PWM_PERIOD3	0x080
HW_PWM_PERIOD3_SET	0x084
HW_PWM_PERIOD3_CLR	0x088
HW_PWM_PERIOD3_TOG	0x08C

Table 24-17. HW_PWM_PERIOD3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2							MATT_SEL	MATT	CDIV			INACTIVE_STATE		ACTIVE_STATE		PERIOD																

Table 24-18. HW_PWM_PERIOD3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSRVD2**	RO	0x0	Reserved.
24	**MATT_SEL**	RW	0x0	Multichip Attachment Mode clock select. When the MATT bit is asserted this bit selects which clock to output. 0: 32 kHz, 1: 24 MHz.
23	**MATT**	RW	0x0	Multichip Attachment mode. This bit overrides the normal signal generation parameters and enables either the 24 MHz or 32 kHz crystal clock on the PWM3 output pin for inter-chip signaling.

Table 24-18. HW_PWM_PERIOD3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
22:20	**CDIV**	RW	0x0	Clock divider ratio to apply to the crystal clock frequency (24.0 MHz) that times the PWM output signal. DIV_1 = 0x0 Divide by 1. DIV_2 = 0x1 Divide by 2. DIV_4 = 0x2 Divide by 4. DIV_8 = 0x3 Divide by 8. DIV_16 = 0x4 Divide by 16. DIV_64 = 0x5 Divide by 64. DIV_256 = 0x6 Divide by 256. DIV_1024 = 0x7 Divide by 1024.
19:18	**INACTIVE_STATE**	RW	0x0	The logical inactive state that is mapped to the PWM output signal. Note that the undefined state of 0x1 is mapped to high-impedance. HI_Z = 0x0 Inactive state sets PWM output to high-impedance. 0 = 0x2 Inactive state sets PWM output to 0 (low). 1 = 0x3 Inactive state sets PWM output to 1 (high).
17:16	**ACTIVE_STATE**	RW	0x0	The logical active state is mapped to the PWM output signal. Note that the undefined state of 0x1 is mapped to high-impedance. HI_Z = 0x0 Active state sets PWM output to high-impedance. 0 = 0x2 Active state sets PWM output to 0 (low). 1 = 0x3 Active state sets PWM output to 1 (high).
15:0	**PERIOD**	RW	0x0	Number of divided XTAL clock cycles in the entire period of the PWM waveform, minus 1. For example, to obtain 6 clock cycles in the actual period, set this field to 5.

DESCRIPTION:

This register contains the programming paramters for multi-chip attachment mode, clock divider value, active high, low values and period for channel 3.

EXAMPLE:

```
HW_PWM_PERIODn_WR(3, 0x00000b1f); // Set up period and active/inactive output states
```

24.4.10 PWM Channel 4 Active Register Description

The PWM Channel 4 Active Register specifies the active time and inactive time for Channel 4.

HW_PWM_ACTIVE4	0x090
HW_PWM_ACTIVE4_SET	0x094
HW_PWM_ACTIVE4_CLR	0x098
HW_PWM_ACTIVE4_TOG	0x09C

Table 24-19. HW_PWM_ACTIVE4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
INACTIVE																ACTIVE															

Table 24-20. HW_PWM_ACTIVE4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**INACTIVE**	RW	0x0	Number of divided XTAL clock cycles to count from the beginning of the period before changing the output from the active state to the inactive state. The internal count of the channel is compared for greater than this value to change to the inactive state.
15:0	**ACTIVE**	RW	0x0	Number of divided XTAL clock cycles to count from the beginning of the period before changing the output to the active state. The internal count of the channel is compared for greater than this value to change to the active state.

DESCRIPTION:

This register contains the active time and inactive time programming parameters for Channel 4.

EXAMPLE:

```
HW_PWM_ACTIVEn_WR(4, 0x000000ff); // Set active and inactive counts
```

24.4.11 PWM Channel 4 Period Register Description

The PWM Channel 4 Period Register specifies the multi-chip attachment mode, clock divider value, active high, low values and period.

HW_PWM_PERIOD4	0x0A0
HW_PWM_PERIOD4_SET	0x0A4
HW_PWM_PERIOD4_CLR	0x0A8
HW_PWM_PERIOD4_TOG	0x0AC

Table 24-21. HW_PWM_PERIOD4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2							MATT_SEL	MATT	CDIV			INACTIVE_STATE		ACTIVE_STATE		PERIOD																

Table 24-22. HW_PWM_PERIOD4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSRVD2**	RO	0x0	Reserved.
24	**MATT_SEL**	RW	0x0	Multichip Attachment Mode clock select. When the MATT bit is asserted this bit selects which clock to output. 0: 32 kHz, 1: 24 MHz.
23	**MATT**	RW	0x0	Multichip Attachment Mode. This bit overrides the normal signal generation parameters and enables either the 24 MHz or 32 kHz crystal clock on the PWM4 output pin for inter-chip signaling.

Table 24-22. HW_PWM_PERIOD4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
22:20	**CDIV**	RW	0x0	Clock divider ratio to apply to the crystal clock frequency (24.0 MHz) that times the PWM output signal. DIV_1 = 0x0 Divide by 1. DIV_2 = 0x1 Divide by 2. DIV_4 = 0x2 Divide by 4. DIV_8 = 0x3 Divide by 8. DIV_16 = 0x4 Divide by 16. DIV_64 = 0x5 Divide by 64. DIV_256 = 0x6 Divide by 256. DIV_1024 = 0x7 Divide by 1024.
19:18	**INACTIVE_STATE**	RW	0x0	The logical inactive state that is mapped to the PWM output signal. Note that the undefined state of 0x1 is mapped to high-impedance. HI_Z = 0x0 Inactive state sets PWM output to high-impedance. 0 = 0x2 Inactive state sets PWM output to 0 (low). 1 = 0x3 Inactive state sets PWM output to 1 (high).
17:16	**ACTIVE_STATE**	RW	0x0	The logical active state is mapped to the PWM output signal. Note that the undefined state of 0x1 is mapped to high-impedance. HI_Z = 0x0 Active state sets PWM output to high-impedance. 0 = 0x2 Active state sets PWM output to 0 (low). 1 = 0x3 Active state sets PWM output to 1 (high).
15:0	**PERIOD**	RW	0x0	Number of divided XTAL clock cycles in the entire period of the PWM waveform, minus 1. For example, to obtain 6 clock cycles in the actual period, set this field to 5.

DESCRIPTION:

This register contains the programming paramters for multi-chip attachment mode, clock divider value, active high, low values and period for channel 4.

EXAMPLE:

```
HW_PWM_PERIODn_WR(4, 0x00000b1f); // Set up period and active/inactive output states
```

24.4.12 PWM Version Register Description

This register indicates the version of the block for debug purposes.

HW_PWM_VERSION 0x0b0

Table 24-23. HW_PWM_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 24-24. HW_PWM_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x01	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x03	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
if (HW_PWM_VERSION.B.MAJOR != 1) Error();
```

PWM Block v1.3, Revision 1.23

Chapter 25
I^{2}C Interface

This chapter describes the I^{2}C interface implemented on the i.MX23. It includes sections on the external pins, interrupt sources, I^{2}C bus protocol, and programming examples. Programmable registers are included in Section 25.4, "Programmable Registers."

25.1 Overview

The I^{2}C is a standard two-wire serial interface used to connect the chip with peripherals or host controllers. This interface provides a standard speed (up to 100 kbps), and a fast speed (up to 400 kbps) I^{2}C connection to multiple devices with the chip acting in I^{2}C master. Typical applications for the I^{2}C bus include: EEPROM, LED/LCD, FM tuner, cell phone baseband chip connection, etc.

As implemented on the i.MX23, the I^{2}C block includes the following functions:

- The I^{2}C block can be configured as a master device. In master mode, it generates the clock (I2C_SCL) and initiates transactions on the data line (I2C_SDA).
- The I^{2}C block packs/unpacks data into 8-, 16-, 24-, or 32-bit words for DMA transactions. Data on the I^{2}C bus is always byte-oriented. Short transmission (up to three bytes plus address) can be easily triggered using only PIO operations, i.e., no DMA setup required.
- The I^{2}C block has programmable device addresses for master transactions.
- Master transactions are composed of one or more DMA commands chained together. The first byte conveys the slave address and read/write bit for the first command. If the entire transaction is an I^{2}C write command, then it can be sent by a single DMA command. If the command is an I^{2}C read transaction, then at least two DMA commands are required to handle it.

Figure 25-1 shows a block diagram of the I²C interface implemented on the i.MX23.

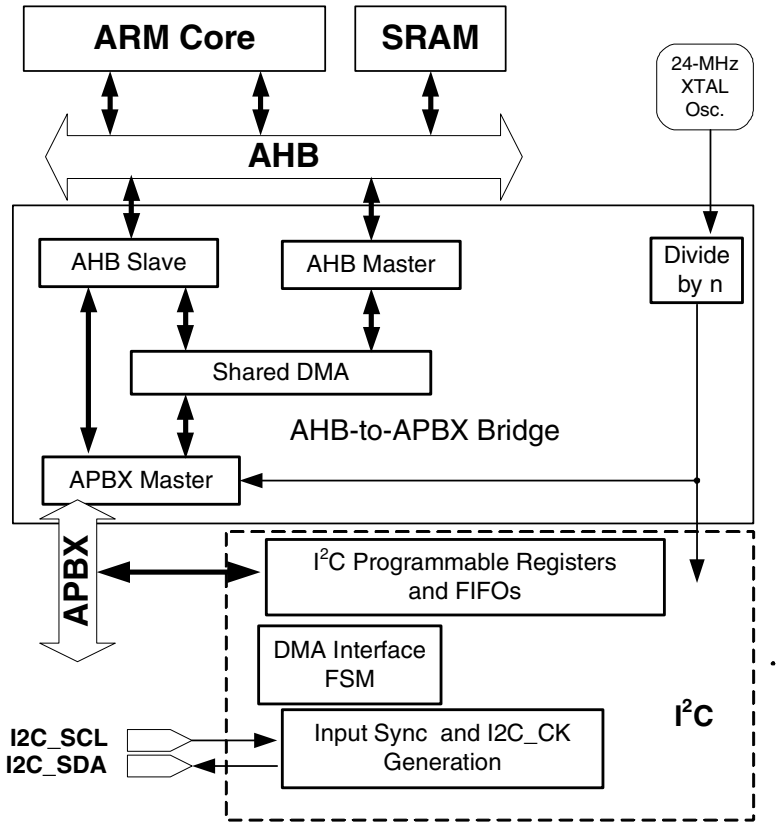


Figure 25-1. I²C Interface Block Diagram

25.2 Operation

The I²C Interface on the i.MX23 includes the following external pins:

- **I2C_SDAQ: I²C Serial Data**—This pin carries all address and data bits.
- **I2C_SCL: I²C Serial Clock**—This pin carries the clock used to time the address and data.

Pullup resistors are required on both of the I²C lines as all of the I²C drivers are open drain (pulldown only). Typically, external 2kΩ resistors are used to pull the signals up to VddIO for normal and fast speeds.

25.2.1 I²C Interrupt Sources

The I²C port can be used in either interrupt-driven or polled modes. An interrupt can be generated by the completion of a DMA command in the APBX DMA. DMA interrupts are the reporting mechanism for I²C transactions that terminate normally. Abnormal terminations or partial completions are signaled by interrupts generated within the I²C controller.

If I^2C interrupts are enabled, a level-sensitive interrupt will be signaled to the processor upon one of the events listed in Table 25-1.

Table 25-1. I^2C Interrupt Condition in HW_I2C_CTRL1

SOURCE	BIT NAME	DESCRIPTION
Slave Address	SLAVE_IRQ	This interrupt is generated when an address match occurs. It indicates that the CPU should read the captured RW bit from the I^2C address byte to determine the type of DMA to use for the data transfer phase.
Slave Stop	SLAVE_STOP_IRQ	This interrupt is generated when a stop condition is detected after a slave address has been matched.
Oversize Xfer	OVERSIZE_XFER_TERM_IRQ	The DMA and I^2C controller are initialized for an expected transfer size. If the data phase is not terminated within this transfer size then oversize transfer processing goes into effect and the CPU is alerted via this interrupt.
Early Termination	EARLY_TERM_IRQ	The DMA and I^2C controller are initialized for an expected transfer size. If the data phase is terminated before this transfer size then early termination processing goes into effect and the CPU is alerted via this interrupt.
Master Loss	MASTER_LOSS_IRQ	A master begins transmission on an idle I^2C bus and monitors the data line. If it ever attempts to send a one on the line and notes that a zero has been sent instead, then it notes that it has lost mastership of the I^2C bus. It terminates its transfer and reports the condition to the CPU via this interrupt. This detection only happens on master transmit operations.
No Slave Ack	NO_SLAVE_ACK_IRQ	When a start condition is transmitted in master mode, the next byte contains an address for a targeted slave. If the targeted slave does not acknowledge the address byte, then this interrupt is set, no further I^2C protocol is processed, and the I^2C bus returns to the idle state.
Data Engine Complete	DATA_ENGINE_CMPLT_IRQ	This bit is set whenever the DMA interface state machine completes a transaction and resets its run bit. This is useful for PIO mode transmit transactions that are not mediated by the DMA and therefore cannot use the DMA command completion interrupt. This bit is still set for master completions when the DMA is used, but can be ignored in that case.
Bus Free	BUS_FREE_IRQ	When bus mastership is lost during the I^2C arbitration phase, the bus becomes busy running services for another master. This interrupt is set whenever a stop command is detected so the master transaction can attempt a retry.

The interrupt lines are tied directly to the bits of Control Register 1. Clearing these bits through software removes the interrupt request.

25.2.2 I^2C Bus Protocol

The I^2C interface operates as shown in Figure 25-2 and Figure 25-3.

- A START condition is defined as a high-to-low transition on the data line while the I2C_SCL line is held high.
- After this has been transmitted by the master, the bus is considered busy.
- The next byte of data transmitted after the start condition contains the address of the slave in the first seven bits, and the eighth bit tells whether the Master is receiving data from the slave or transmitting data to the slave.

- When an address is sent, each device in the system compares the first seven bits after a start condition with its address.
- If they match, the device considers itself addressed by the master.

Data transfer with acknowledge is obligatory.

- The transmitter must release the I2C_SDA line during the acknowledge pulse.
- The receiver must then pull the data line low, so that it remains stable low during the high period of the acknowledge clock pulse.
- A receiver that has been addressed is obliged to generate an acknowledge after each byte of data has been received.
- A slave device can terminate a transfer by withholding its acknowledgement.

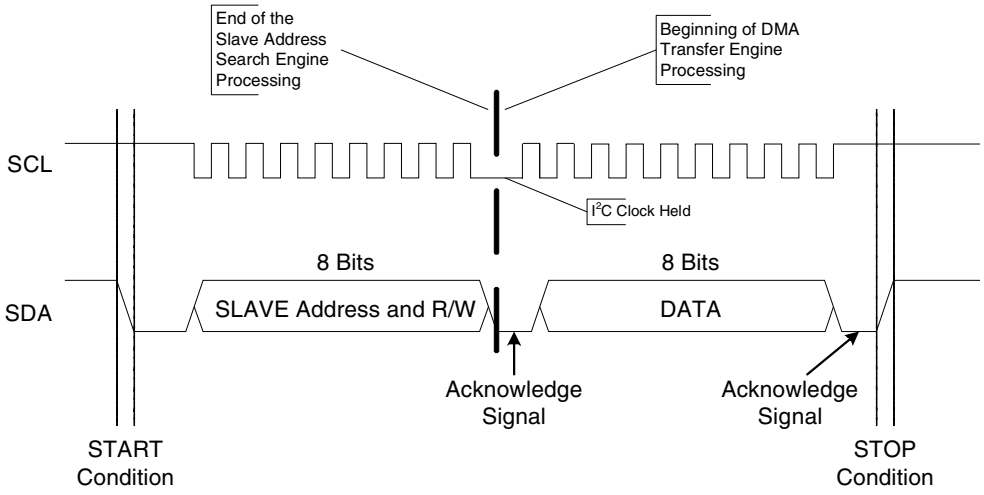


Figure 25-2. I²C Data and Clock Timing

The clock is generated by the master, according to parameters set in the HW_I2C_TIMINGn register. This register also provides programmable timing for capturing received data, as well as for changing the transmitted data bit.

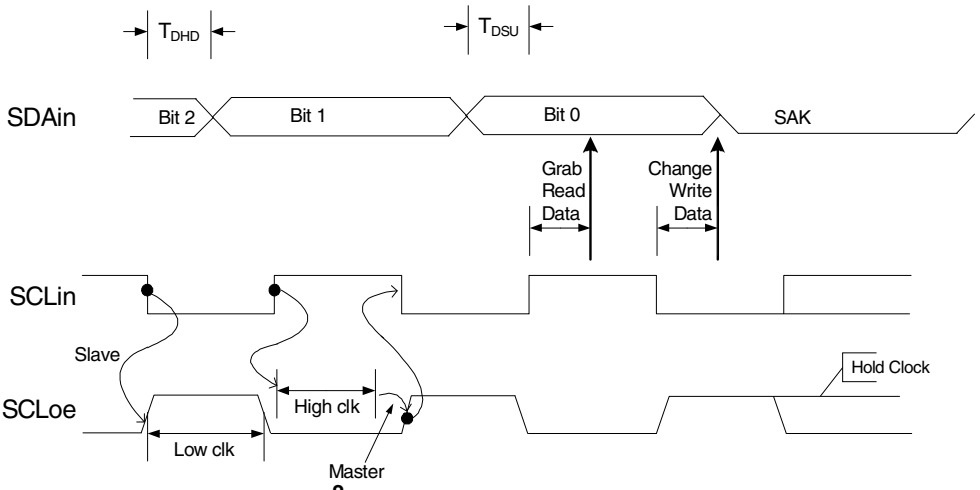


Figure 25-3. I²C Data and Clock Timing Generation

25.2.2.1 Simple Device Transactions

The simplest transfer of interest on an I²C bus is writing a single data byte from a master to a slave, for example, writing a single byte to an FM tuner. In this transaction, a start condition is transmitted, followed by the device address byte, followed by a single byte of write data. This sequence always ends with a stop condition.

Table 25-2. I²C Transfer When the Interface is Transmitting as a Master

ST	SAD+W	SAK	DATA	SAK	SP

Table 25-3 defines the symbols used in describing I²C transactions. For example, in the single byte write operation, ST is a start condition, and SP is a stop condition. The data transfer occurs between these two bus events. It starts with a slave address plus write byte (SAD+W) addressing the targeted slave. A slave-generated acknowledge bit (SAK) tells the master that a slave has recognized the address and will accept the transfer. The master sends the data byte (DATA), and the slave acknowledges it with an SAK.

Table 25-3. I²C Address Definitions

BIT	DESCRIPTION
ST	Start Condition
SR	Repeated Start Condition
SAD	Slave Address
SAK	Slave Acknowledge
SUB	Sub-Address, e.g., for EEPROMs
DATA	Data
SP	Stop Condition
MAK	Master Acknowledge
NMAK	No Master Acknowledge

To receive one data byte from a slave device such as an FM tuner, the following bus transaction takes place.

Table 25-4. I²C Transfer "FM Tuner" Read of One Byte

ST	SAD+R	SAK	DATA	NMAK	SP

In this transaction:

- The master first generates a start condition, ST.
- It then sends the seven-bit slave address for the FM tuner plus a read bit (SAD+R).
- The slave in the FM tuner responds with a slave acknowledge bit (SAK).
- The master then generates I²C clocks for a data byte to be transferred (DATA).
- The slave provides data to the I²C data bus during the DATA byte transfer.
- Next, the master generates a master non-acknowledge to the slave (NMAK), indicating the end of the data transfer to the slave. The slave will then release the data line.
- Finally, the master generates a stop condition (SP), terminating the transaction and freeing the I²C bus for other masters to use.

The following example shows a multiple byte read from an FM tuner or other slave device:

Table 25-5. I²C Transfer "FM Tuner" Read of Three Bytes

ST	SAD+R	SAK	DATA	MAK	DATA	MAK	DATA	NMAK	SP

25.2.2.2 Typical EEPROM Transactions

I²C EEPROMs typically have a specific transaction sequence for reading and writing data bytes to and from the EEPROM array. Table 25-6 through Table 25-9 show the first two bytes of data as a sub-address for purposes of illustration. The sub-address is used to address the memory space inside the device. Table 25-3 defines each element of the transactions shown. When writing a single byte of data to the EEPROM, one must first transfer two bytes of sub-address as follows:

Table 25-6. I²C Transfer When Master is Writing One Byte of Data to a Slave

ST	SAD+W	SAK	SUB	SAK	SUB	SAK	DATA	SAK	SP

The sub-address only needs to be specified once for a multibyte transfer, as shown here. Note that the sub-address must be sent for each start condition that initiates a transaction.

Table 25-7. I²C Transfer When Master is Writing Multiple Bytes to a Slave

ST	SAD+W	SAK	SUB	SAK	SUB	SAK	DATA	SAK	DATA	SAK	SP

One must also provide the sub-address before reading bytes from the EEPROM. The sub-address is transmitted from the master to the slave before it can receive data bytes. The two transfers are joined into a single bus transaction though the use of a repeated start condition (SR). Normally, a stop condition precedes a start condition. However, when a start condition is preceded by another start condition, it is

known as a repeated start (SR). Note that the two-byte sub address is transferred using an SAD+W address, while the data is received using a SAD+R address.

Table 25-8. I²C Transfer When Master is Receiving One Byte of Data from a Slave

ST	SAD+W	SAK	SUB	SAK	SUB	SAK	SR	SAD+R	SAK	DATA	NMAK	SP

Table 25-9. I²C Transfer When Master is Receiving Multiple Bytes of Data from a Slave

ST	SAD+W	SAK	SUB	SAK	SUB	SAK	SR	SAD+R	SAK	DATA	MAK	DATA	MAK	DATA	NMAK	SP

25.2.2.3 Master Mode Protocol

In master mode, the I²C interface generates the clock and initiates all transfers.

25.2.2.4 Clock Generation

The I²C clock is generated from the APBX clock, as described in the register description.

- If another device pulls the clock low before the I²C block has counted the high period, then the I²C block immediately pulls the clock low as well and starts counting its low period.
- Once the low period has been counted, the I²C block releases the clock line high, but must then check to see if another device stills holds the line low, in which case it enters a high wait state.

In this way, the I2C_SCL clock is generated, with its low period determined by the device with the longest clock low period and its high period determined by the one with the shortest clock high period.

25.2.2.5 Master Mode Operation

The finite state machine for master mode operation is shown in Figure 25-4 through Figure 25-7. Figure 25-4 shows the generation of the optional start condition. Figure 25-5 shows the receive states, Figure 25-6 shows the transmit states. Figure 25-7 shows the generation of the optional stop state.

Table 25-10 through Table 25-13 show examples of Master Mode I²C transactions. Table 25-3 defines each sub-address shown. The following read-after-write transactions are performed using the restart technique.

Table 25-10. I²C Transfer When the Interface as Master is Transmitting One Byte of Data

ST	SAD+W	SAK	SUB	SAK	SUB	SAK	DATA	SAK	SP

Table 25-11. I²C Transfer When the Interface as Master is Receiving >1 Byte of Data from Slave

ST	SAD+R	SAK	DATA	MAK	DATA	MAK	DATA	NMAK	SP

Table 25-12. I²C Transfer when Master is Receiving 1 Byte of Data from Slave Internal Subaddress

ST	SAD+W	SAK	SUB	SAK	SUB	SAK	SR	SAD+R	SAK	DATA	NMAK	SP

Table 25-13. I²C Transfer When Master is Receiving >1 byte of Data from Slave Internal Subaddress

ST	SAD+W	SAK	SUB	SAK	SUB	SAK	SR	SAD+R	SAK	DATA	MAK	DATA	MAK	DATA	NMAK	SP

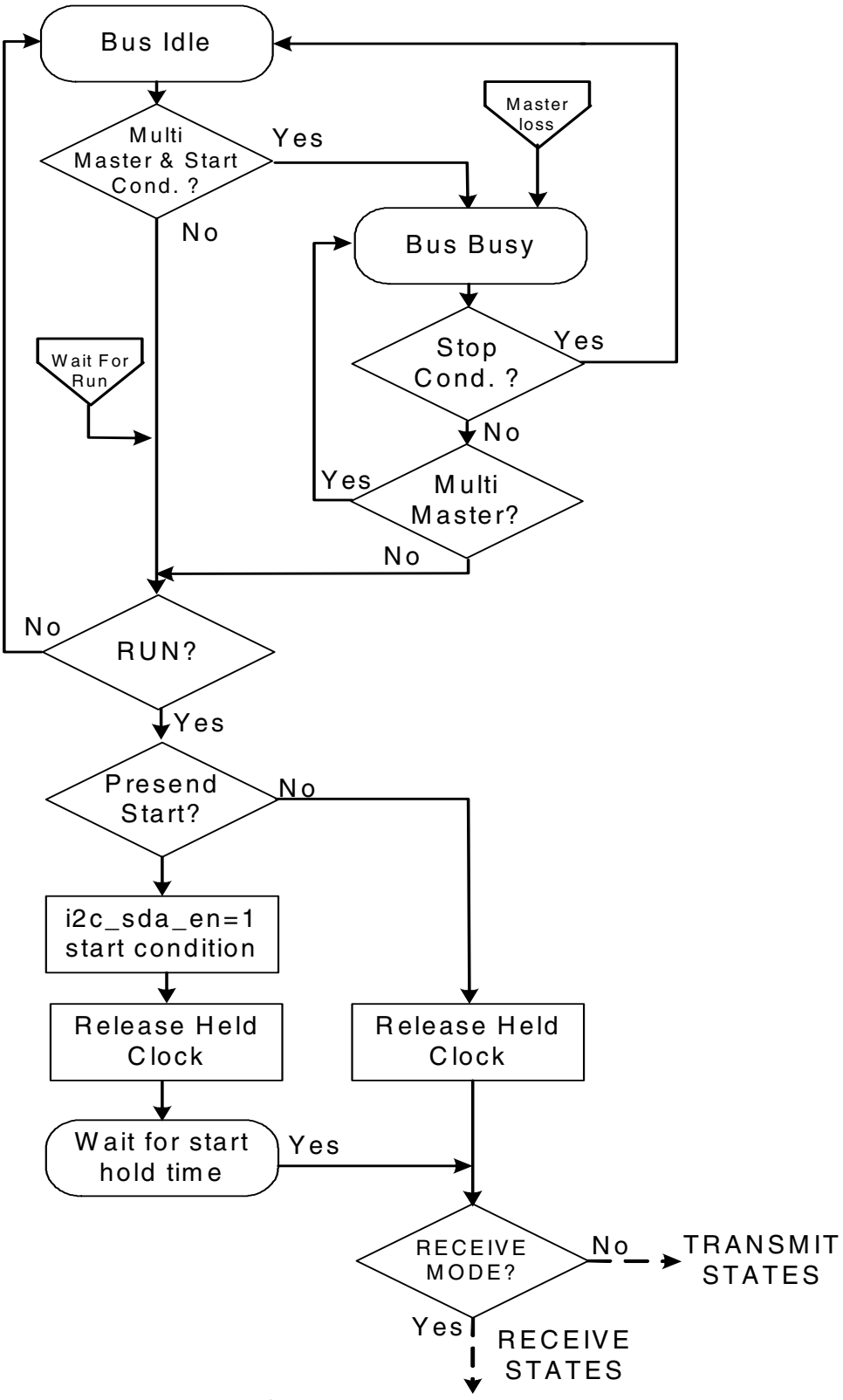


Figure 25-4. I²C Master Mode Flow Chart—Initial States

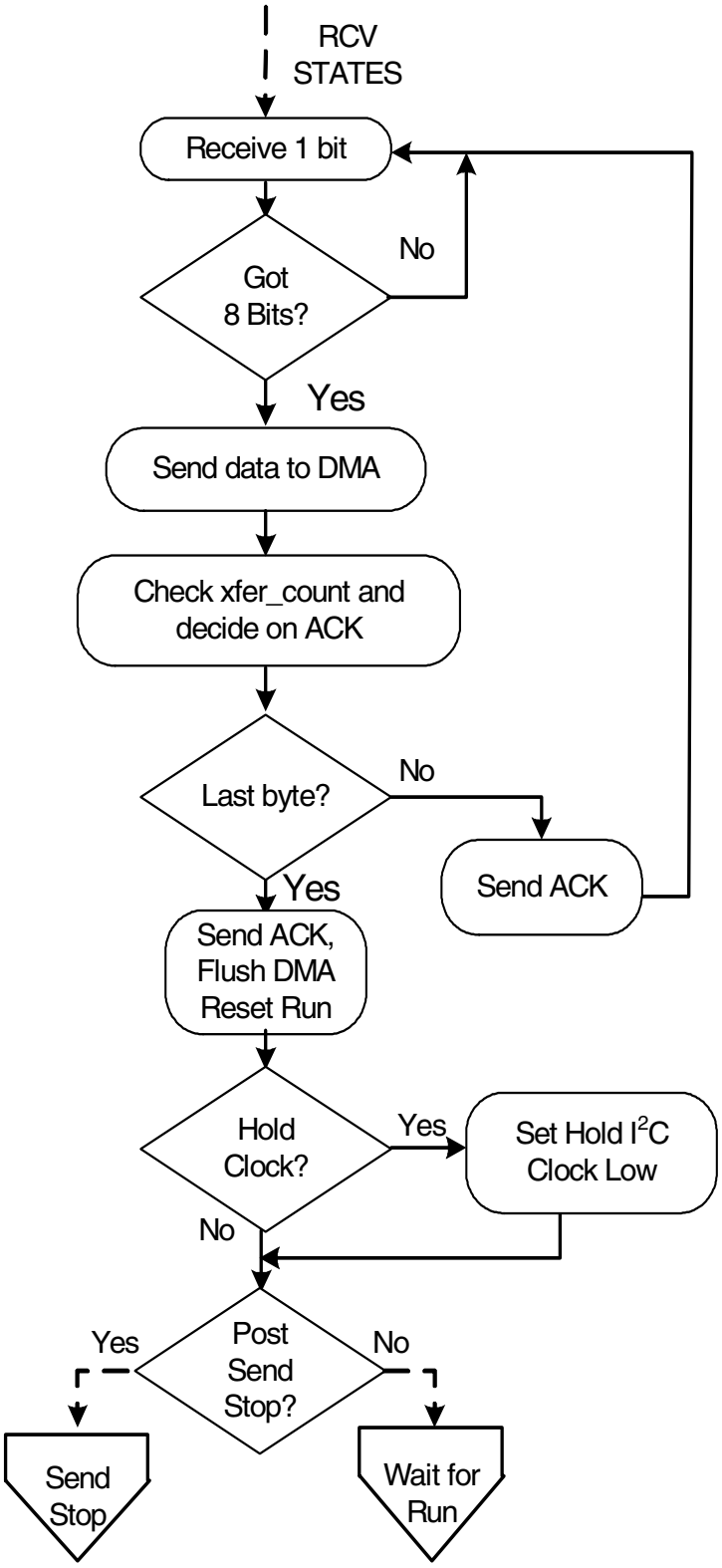


Figure 25-5. I²C Master Mode Flow Chart—Receive States

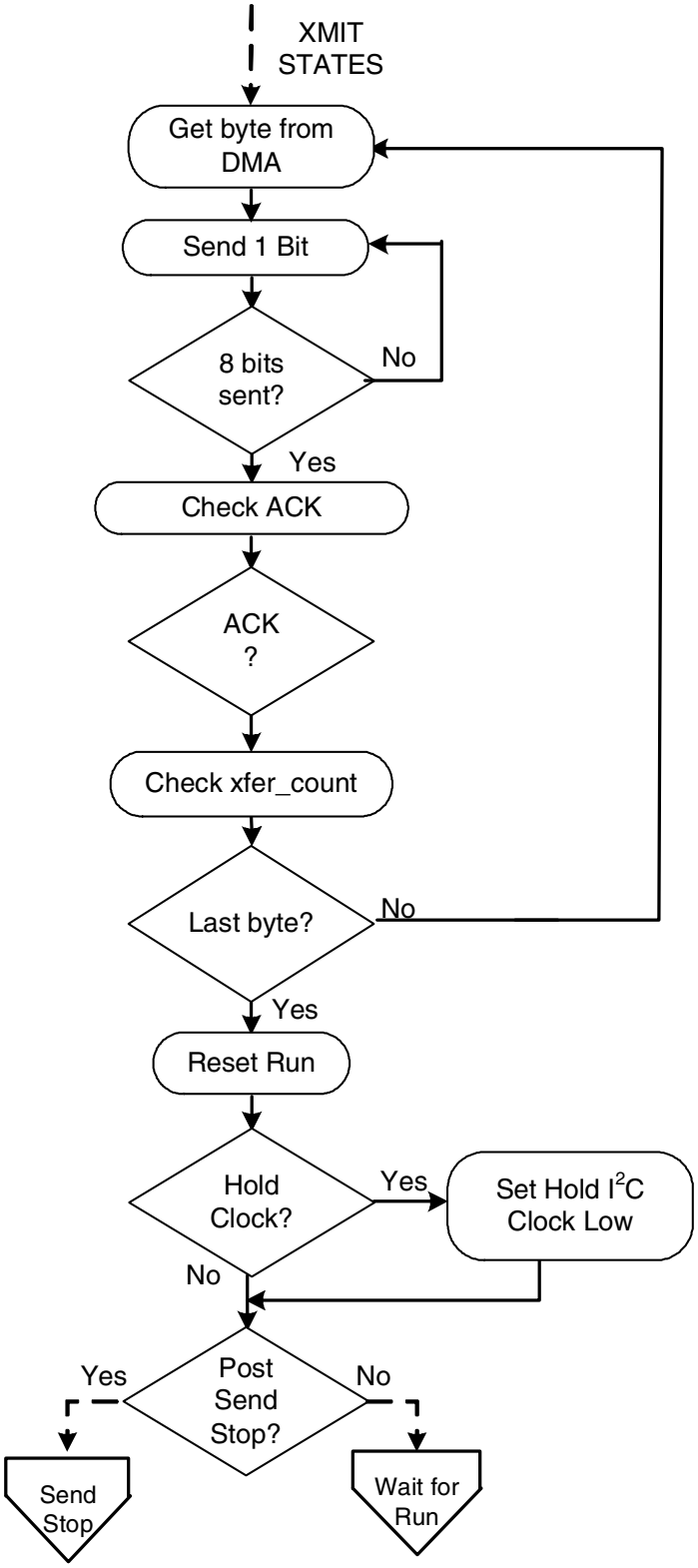


Figure 25-6. I²C Master Mode Flow Chart—Transmit States

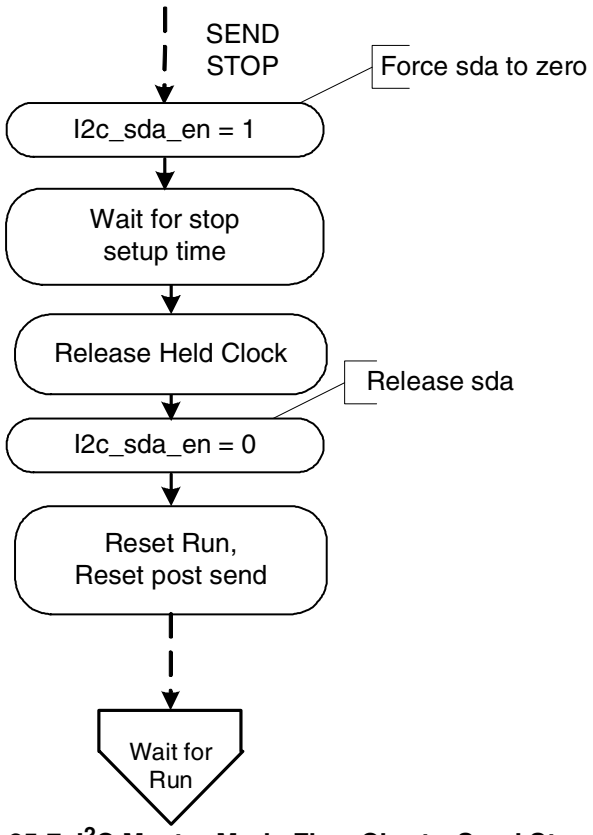


Figure 25-7. I²C Master Mode Flow Chart—Send Stop States

25.2.3 Programming Examples

25.2.3.1 Five Byte Master Write Using DMA

The example in Figure 25-8 shows sending five bytes from an i.MX23 operating as an I²C master to another device acting as an I²C slave.

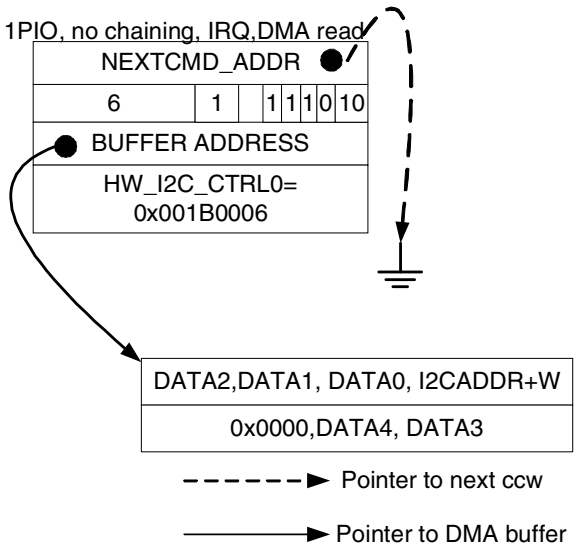


Figure 25-8. I²C Writing Five Bytes

The DMA command is initialized to send six bytes to the I²C controller and one word of PIO information to the HW_I2C_CTRL0 register.

Table 25-14. I²C Transfer When the Master Transmits 5 Bytes of Data to the Slave

ST	SAD+W	SAK	DATA0	SAK	DATA1	SAK	DATA2	SAK	DATA3	SAK	DATA4	SAK	SP

The following C code is used to send a five-byte transmission:

```
// SEND: start, 0x56, 0x01,0x02,0x03,0x04,0x05,stop
//--------------------------------------------------------
#define I2C_CHANNEL_NUM 3
// dma buffer of 6 bytes (i2c address + 5 data bytes)
static reg32_t I2C_DATA_BUFFER[2]=
{
   0x03020156,  //slave address 56+W
   0x00000504   // last two data bytes
};
// DMA command chain
const static reg32_t  I2C_DMA_CMD[4] =
{
    (reg32_t)  I2C_DMA_CMD2,
    (BF_APBX_CHn_CMD_XFER_COUNT(6)
     BF_APBX_CHn_CMD_SEMAPHORE(1)
     BF_APBX_CHn_CMD_CMDWORDS(1)
     BF_APBX_CHn_CMD_WAIT4ENDCMD(1)
     BF_APBX_CHn_CMD_CHAIN(0)
     BV_FLD(APBX_CHn_CMD, COMMAND, DMA_READ)),
    (reg32_t) &eeprom_command_buffer[0],
  BF_I2C_CTRL0_POST_SEND_STOP(BV_I2C_CTRL0_POST_SEND_STOP__SEND_STOP)
  BF_I2C_CTRL0_PRE_SEND_START(BV_I2C_CTRL0_PRE_SEND_START__SEND_START)
  BF_I2C_CTRL0_MASTER_MODE(BV_I2C_CTRL0_MASTER_MODE__MASTER)
  BF_I2C_CTRL0_DIRECTION(BV_I2C_CTRL0_DIRECTION__TRANSMIT)
  BF_I2C_CTRL0_XFER_COUNT(6)
};



void SendFiveBytes(){
 // Reset the APBX dma channels associated with I2C.
 reset_mask = BF_APBX_CTRL0_RESET_CHANNEL((1 << I2C_CHANNEL_NUM));
 HW_APBX_CTRL0_SET(reset_mask);
```

```
 // Poll for reset to clear the channel.
 for (retries = 0; retries < RESET_TIMEOUT; retries++)
      if ((reset_mask & HW_APBX_CTRL0_RD()) == 0)
            break;
 if( retries == RESET_TIMEOUT) exit(1);

 // Setup dma channel configuration.
 BF_WRn(APBX_CHn_NXTCMDAR,I2C_CHANNEL_NUM,
        CMD_ADDR,(reg32_t) I2C_DMA_CMD);
 BF_WR(APBX_CTRL1, CH3_CMDCMPLT_IRQ, 0); // clear interrupt

 // Start the dma channel by incrementing semaphore.
 BF_WRn(APBX_CHn_SEMA, I2C_CHANNEL_NUM, INCREMENT_SEMA, 1);

 // Poll for the semaphore to decrement to zero on the DMA channel.
 for (retries = 0; retries < SEMAPHORE_TIMEOUT; retries++)
      if (0 == BF_RDn(APBX_CHn_SEMA, I2C_CHANNEL_NUM, PHORE))
            break;

 // a frame with one byte of address and five bytes of data was just sent
}
```

25.2.3.2 Reading 256 Bytes from an EEPROM

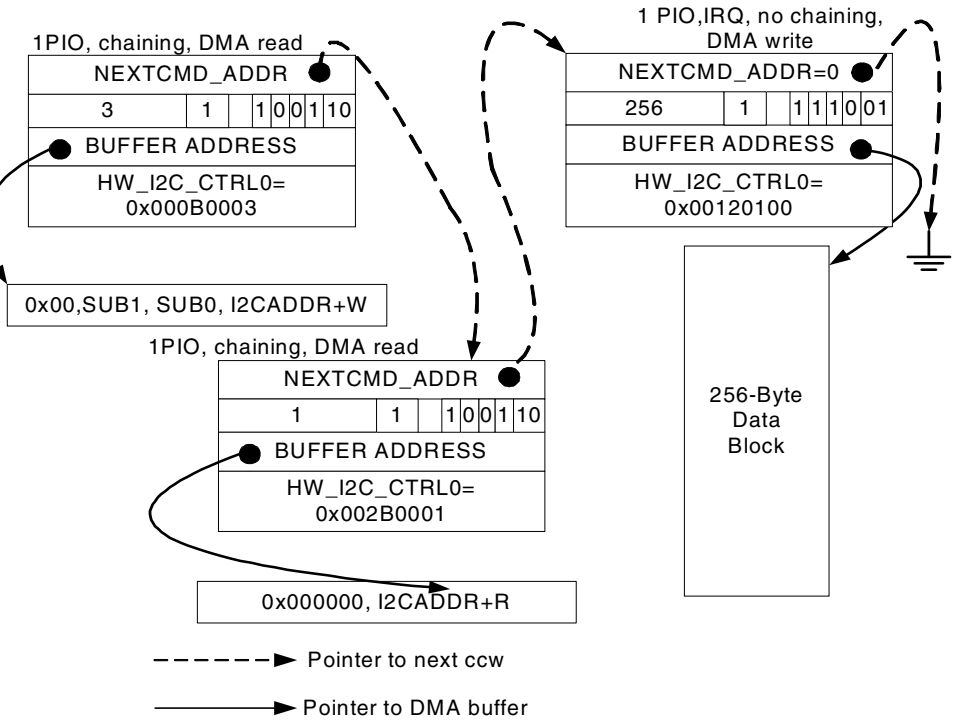


Figure 25-9. I²C Reading 256 Bytes from an EEPROM

```
//-----------------------------------------------------------------------
// dma buffers to hold i2c command string for slave addres+W plus sub0,
// sub 1 and the second command, a slave address+R
// eePROM write address == 0xA0,  read address == 0xA1
//-----------------------------------------------------------------------
unsigned char eeprom_command_buffer[4] = {0xA0,0x34,0x12,0xA1};

//-----------------------------------------------------------------------
// I2C DMA chain
//-----------------------------------------------------------------------
const static reg32_t  I2C_DMA_CMD3[4] =
{
    0x0,
    (BF_APBX_CHn_CMD_XFER_COUNT(256)
     BF_APBX_CHn_CMD_WAIT4ENDCMD(1)
```

```
     BF_APBX_CHn_CMD_SEMAPHORE(1)     |
     BF_APBX_CHn_CMD_CMDWORDS(1)      |
     BF_APBX_CHn_CMD_CHAIN(0)         |             // last command
     BV_FLD(APBX_CHn_CMD, COMMAND, DMA_WRITE)),
    (reg32_t) &eeprom_command_buffer[3],
    BF_I2C_CTRL0_POST_SEND_STOP(BV_I2C_CTRL0_POST_SEND_STOP__SEND_STOP)|
    BF_I2C_CTRL0_MASTER_MODE(BV_I2C_CTRL0_MASTER_MODE__MASTER)         |
    BF_I2C_CTRL0_DIRECTION(BV_I2C_CTRL0_DIRECTION__RECEIVE)            |
    BF_I2C_CTRL0_XFER_COUNT(256)
};
const static reg32_t  I2C_DMA_CMD2[4] =
{
    (reg32_t)  I2C_DMA_CMD3,
    (BF_APBX_CHn_CMD_XFER_COUNT(1)  |
     BF_APBX_CHn_CMD_CMDWORDS(1)    |
     BF_APBX_CHn_CMD_WAIT4ENDCMD(1)|
     BF_APBX_CHn_CMD_CHAIN(1)       |
     BV_FLD(APBX_CHn_CMD, COMMAND, DMA_READ)),
    (reg32_t) &eeprom_command_buffer[3],
    BF_I2C_CTRL0_RETAIN_CLOCK(BV_I2C_CTRL0_RETAIN_CLOCK__HOLD_LOW)|

   BF_I2C_CTRL0_PRE_SEND_START(BV_I2C_CTRL0_PRE_SEND_START__SEND_START)|
   BF_I2C_CTRL0_MASTER_MODE(BV_I2C_CTRL0_MASTER_MODE__MASTER)          |
   BF_I2C_CTRL0_DIRECTION(BV_I2C_CTRL0_DIRECTION__TRANSMIT)            |
   BF_I2C_CTRL0_XFER_COUNT(1)
};
const static reg32_t  I2C_DMA_CMD1[4] =
{
    (reg32_t)  I2C_DMA_CMD2,
    (BF_APBX_CHn_CMD_XFER_COUNT(3)  |
     BF_APBX_CHn_CMD_CMDWORDS(1)    |
     BF_APBX_CHn_CMD_WAIT4ENDCMD(1)|
     BF_APBX_CHn_CMD_CHAIN(1)       |
     BV_FLD(APBX_CHn_CMD, COMMAND, DMA_READ)),
    (reg32_t) &eeprom_command_buffer[0],
  BF_I2C_CTRL0_PRE_SEND_START(BV_I2C_CTRL0_PRE_SEND_START__SEND_START)  |
  BF_I2C_CTRL0_MASTER_MODE(BV_I2C_CTRL0_MASTER_MODE__MASTER)           |
  BF_I2C_CTRL0_DIRECTION(BV_I2C_CTRL0_DIRECTION__TRANSMIT)             |
  BF_I2C_CTRL0_XFER_COUNT(3)
};


////////////////////////////////////////////////////
//Read256BytesFromEEPROM returns 1 for errors and 0 for OK
////////////////////////////////////////////////////
int Read256BytesFromEEPROM(unsigned short usAddress){
 // insert eePROM addres param into dma command buffer
  I2C_CMD_BUFFER[1] = (unsigned char) (usAddress      &0x00ff);
  I2C_CMD_BUFFER[2] = (unsigned char) ((usAddress>>8) &0x00ff);

 // Reset the APBX dma channels associated with I2C.
 reset_mask = BF_APBX_CTRL0_RESET_CHANNEL((1 << I2C_CHANNEL_NUM));
 HW_APBX_CTRL0_SET(reset_mask);

 // Poll for reset to clear the channel.
 for (retries = 0; retries < RESET_TIMEOUT; retries++)
      if ((reset_mask & HW_APBX_CTRL0_RD()) == 0)
              break;
 if (retries == RESET_TIMEOUT)exit(1);
 // Setup dma channel configuration.
 BF_WRn(APBX_CHn_NXTCMDAR,I2C_CHANNEL_NUM,
        CMD_ADDR,(reg32_t) I2C_DMA_CMD_SUBADDR);
 BF_WR(APBX_CTRL1, CH3_CMDCMPLT_IRQ, 0);

 // Start the dma channel by incrementing semaphore.
 BF_WRn(APBX_CHn_SEMA, I2C_CHANNEL_NUM, INCREMENT_SEMA, 1);

 // Poll for the semaphore to decrement to zero on the DMA channel.
 for (retries = 0; retries < SEMAPHORE_TIMEOUT; retries++)}
      if (0 == BF_RDn(APBX_CHn_SEMA, I2C_CHANNEL_NUM, PHORE))
              break;
      if (1 == HW_I2C_CTRL1.MASTER_LOSS_IRQ)        return 1; // error
      if (1 == HW_I2C_CTRL1.NO_SLAVE_ACK_IRQ)       return 1; // error
      if (1 == HW_I2C_CTRL1.EARLY_TERM_IRQ)         return 1; // error

 }
 ferreters == SEMAPHORE_TIMEOUT) exit(2);
 // the 256 bytes were read from the eePROM so return with no Error
```

```
    return 0;
}
```

25.3 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, "Correct Way to Soft Reset a Block," for additional information on using the SFTRST and CLKGATE bit fields.

25.3.1 Pinmux Selection During Reset

For proper I^2C operation, the appropriate pinmux(s) must be selected before taking the block out of reset. Failure to select the I^2C pinmux selections before taking the block out of reset will cause the I^2C clock to operate incorrectly and will require another I^2C hardware reset.

25.3.1.1 Correct and Incorrect Reset Examples

Incorrect:

Clear I^2C SFTRST/CLKGATE
... Setup ...
I^2C PinMux Selections
*** I^2C will not operate.*

Correct:

I^2C PinMux Selections
Clear I^2C SFTRST/CLKGATE
... Setup ...
*** I^2C operates correctly.*

25.4 Programmable Registers

The following registers describe the programming interface for the master I^2C controller.

25.4.1 I2C Control Register 0 Description

The I2C Control Register specifies the reset state and the command and transfer size information for the I2C controller.

HW_I2C_CTRL0	0x000
HW_I2C_CTRL0_SET	0x004
HW_I2C_CTRL0_CLR	0x008
HW_I2C_CTRL0_TOG	0x00C

Table 25-15. HW_I2C_CTRL0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	RUN	RSVD2	PRE_ACK	ACKNOWLEDGE	SEND_NAK_ON_LAST	PIO_MODE	RSVD1	CLOCK_HELD	RETAIN_CLOCK	POST_SEND_STOP	PRE_SEND_START	SLAVE_ADDRESS_ENABLE	MASTER_MODE	DIRECTION	XFER_COUNT															

Table 25-16. HW_I2C_CTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	Set to zero for normal operation. When this bit is set to one (default), then the entire block is held in its reset state. RUN = 0x0 Allow I2C to operate normally. RESET = 0x1 Hold I2C in reset.
30	**CLKGATE**	RW	0x1	This bit must be set to zero for normal operation. When set to one it gates off the clocks to the block. RUN = 0x0 Allow I2C to operate normally. NO_CLKS = 0x1 Do not clock I2C gates in order to minimize power consumption.
29	**RUN**	RW	0x0	Set this bit to one to enable the I2C Controller operation. This bit is automatically set by DMA commands that write to CTRL1 after the last PIO write of the DMA command. For Soft DMA operation, software can set this bit to enable the controller. HALT = 0x0 No I2C command in progress. RUN = 0x1 Process a master I2C command.
28	**RSVD2**	RO	0x0	Always set this bit field to zero.
27	**PRE_ACK**	RW	0x0	Reserved for Freescale use.
26	**ACKNOWLEDGE**	RW	0x0	Set this bit to one to cause a pending acknowledge bit (prior to DMA transfer) to be acknowledged. Set it to zero to NAK the pending acknowledge bit. This bit is set to one by the slave search engine if the criteria is met for acknowledging a slave address. Software can reset the bit to slave-not-acknowledge the address. This bit defines the state of the i2c_data line during the address acknowledge bit time. The slave search engine holds the clock at this point for a software decision. This bit has no effect when the presend start option is selected. SNAK = 0x0 slave not acknowledge when the held clock is released. ACK = 0x1 slave acknowledge when the held clock is released.
25	**SEND_NAK_ON_LAST**	RW	0x0	Set this bit to one to cause the DMA transfer engine to send a NAK on the last byte. ACK_IT = 0x0 Send an ACK on the last byte received. NAK_IT = 0x1 Send a NAK on the last byte received.

Table 25-16. HW_I2C_CTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
24	**PIO_MODE**	RW	0x0	Set this bit to one to enable PIO mode of operation for the I2C master. One can preload up to four bytes into HW_I2C_DATA register before setting the RUN bit. The state machine will not attempt to use the DMA for master transmit operation. The normal start and stop conditions can be sent and the clock can be held at the end of the transfer, if desired. NOTE: all receive operations must use the DMA mode, not the PIO mode.
23	**RSVD1**	RW	0x0	Program this field to 0x0.
22	**CLOCK_HELD**	RW	0x0	This bit is set to one by the I2C controller state machines. It holds the I2C clock line low until cleared. It must be cleared by firmware, either by CPU instructions or DMA PIO transactions. It is set high when a slave address is matched by the slave controller. It is also set high at the end of a master or slave transaction that had the RETAIN_CLOCK bit set high. Software should not set this bit to one. RELEASE = 0x0 Release the clock line. HELD_LOW = 0x1 The clock line is currently being held low.
21	**RETAIN_CLOCK**	RW	0x0	Set this bit to one to retain the clock at the end of this transaction. This has the effect of holding the clock low until the start of the next transaction. RELEASE = 0x0 Release the clock line after this data transfer. HOLD_LOW = 0x1 Hold the clock line low after this data transfer.
20	**POST_SEND_STOP**	RW	0x0	Set this bit to one to send a stop condition after transferring the data associated with this transaction. This bit is automatically cleared by the hardware after the operation has been performed. NO_STOP = 0x0 Do not send a stop condition before this transaction. SEND_STOP = 0x1 Send a stop condition before this transaction.
19	**PRE_SEND_START**	RW	0x0	Set this bit to one to send a start condition before transferring the data associated with this transaction. This bit is automatically cleared by the hardware after the operation has been performed. NO_START = 0x0 Do not send a start condition before this transaction. SEND_START = 0x1 Send a start condition before this transaction.
18	**SLAVE_ADDRESS_ENABLE**	RW	0x0	Set this bit to one to enable the slave address decoder. When an address match occurs, the I2C bus clock is frozen, by setting HW_I2C_CTRL0_CLOCK_HELD, and an interrupt is generated. DISABLED = 0x0 Disable the slave address decoder. ENABLED = 0x1 Enable the slave address decoder.
17	**MASTER_MODE**	RW	0x0	Set this bit to one to select master mode. MASTER_DISABLED = 0x0 Master mode disabled. MASTER_ENABLED = 0x1 Operate in master mode.
16	**DIRECTION**	RW	0x0	Set this bit to one to select an I2C transmit operation. XMIT = write. Set this bit to zero to select an I2C receive operation. RECEIVE = 0x0 I2C receive operation. TRANSMIT = 0x1 I2C transmit operation.
15:0	**XFER_COUNT**	RW	0x0000	Number of bytes to transfer. This field decrements as bytes are transferred.

DESCRIPTION:

This register is either written by the DMA or the CPU depending on the state of an I2C transaction.

EXAMPLE:

```
// turn off soft reset and clock gating
HW_I2C_CTRL0_CLR(BM_I2C_CTRL0_SFTRST | BM_I2C_CTRL0_CLKGATE);
```

25.4.2 I2C Timing Register 0 Description

The timing for various phases of I2C Controller Commands are further defined by fields in the I2C Timing Register 0.

HW_I2C_TIMING0	0x010
HW_I2C_TIMING0_SET	0x014
HW_I2C_TIMING0_CLR	0x018
HW_I2C_TIMING0_TOG	0x01C

Table 25-17. HW_I2C_TIMING0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2						HIGH_COUNT										RSVD1						RCV_COUNT									

Table 25-18. HW_I2C_TIMING0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSVD2**	RO	0x0	Always set this bit field to zero.
25:16	**HIGH_COUNT**	RW	0x78	Load this bit field with the APBX clock count for the high period of the I2C clock.
15:10	**RSVD1**	RO	0x0	Always set this bit field to zero.
9:0	**RCV_COUNT**	RW	0x30	Load this bit field with the APBX clock count for capturing read data after the I2C clock goes high.

DESCRIPTION:

This register is primarily used for clock and timing generation.

EXAMPLE:

```
HW_I2C_TIMING0_WR(0x00780030); // high time = 120 clocks, read bit at 48 for 95KHz at 24MHz
HW_I2C_TIMING0_WR(0x000F0007); // high time = 15 clocks, read bit at 7 for 400KHz at 24MHz
```

25.4.3 I2C Timing Register 1 Description

The timing for various phases of I2C Controller Commands are further defined by fields in the I2C Timing Register 1.

HW_I2C_TIMING1	0x020
HW_I2C_TIMING1_SET	0x024
HW_I2C_TIMING1_CLR	0x028
HW_I2C_TIMING1_TOG	0x02C

Table 25-19. HW_I2C_TIMING1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2						LOW_COUNT										RSVD1						XMIT_COUNT									

Table 25-20. HW_I2C_TIMING1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSVD2**	RO	0x0	Always set this bit field to zero.
25:16	**LOW_COUNT**	RW	0x80	Load this bit field with the APBX clock count for the low period of the I2C clock.
15:10	**RSVD1**	RO	0x0	Always set this bit field to zero.
9:0	**XMIT_COUNT**	RW	0x30	Load this bit field with the APBX clock count for changing transmitted data after the I2C clock goes low. Set this value to produce valid i2c setup and hold times at the desired bit rate for the current APBX clock rate.

DESCRIPTION:

This register is primarily used for clock and timing generation.

EXAMPLE:

```
HW_I2C_TIMING1_WR(0x00800030); // low time at 128, write bit at 48 for 95 kHz at 24 MHz
HW_I2C_TIMING1_WR(0x001F000F); // low time at 31, write bit at 15 for 400 kHz at 24 MHz
```

25.4.4 I2C Timing Register 2 Description

The timing for various phases of I2C Controller Commands are further defined by fields in the I2C Timing Register 2.

HW_I2C_TIMING2	0x030
HW_I2C_TIMING2_SET	0x034
HW_I2C_TIMING2_CLR	0x038
HW_I2C_TIMING2_TOG	0x03C

Table 25-21. HW_I2C_TIMING2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2						BUS_FREE										RSVD1						LEADIN_COUNT									

Table 25-22. HW_I2C_TIMING2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSVD2**	RO	0x0	Always set this bit field to zero.
25:16	**BUS_FREE**	RW	0x30	Load this bit field with the APBX clock count for delaying the transition to the bus idle state after entering stop state in the clock generator.
15:10	**RSVD1**	RO	0x0	Always set this bit field to zero.
9:0	**LEADIN_COUNT**	RW	0x30	Load this bit field with the APBX clock count for delaying the rising edge of i2c_sck after the kick.

DESCRIPTION:

This register is primarily used for clock and timing generation.

EXAMPLE:

```
HW_I2C_TIMING2_WR(0x0015000d); // bus free count of 21 lead in count of 13
```

25.4.5 I2C Control Register 1 Description

The I2C Controller Command is further defined by fields in this control extension register. The I2C Control Register 1 is where the I2C slave address is specified. Fast or normal mode is selected here.

HW_I2C_CTRL1	0x040
HW_I2C_CTRL1_SET	0x044
HW_I2C_CTRL1_CLR	0x048
HW_I2C_CTRL1_TOG	0x04C

Table 25-23. HW_I2C_CTRL1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2			CLR_GOT_A_NAK	ACK_MODE	FORCE_DATA_IDLE	FORCE_CLK_IDLE	BCAST_SLAVE_EN	RSVD1								BUS_FREE_IRQ_EN	DATA_ENGINE_CMPLT_IRQ_EN	NO_SLAVE_ACK_IRQ_EN	OVERSIZE_XFER_TERM_IRQ_EN	EARLY_TERM_IRQ_EN	MASTER_LOSS_IRQ_EN	SLAVE_STOP_IRQ_EN	SLAVE_IRQ_EN	BUS_FREE_IRQ	DATA_ENGINE_CMPLT_IRQ	NO_SLAVE_ACK_IRQ	OVERSIZE_XFER_TERM_IRQ	EARLY_TERM_IRQ	MASTER_LOSS_IRQ	SLAVE_STOP_IRQ	SLAVE_IRQ

Table 25-24. HW_I2C_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSVD2**	RO	0x0	Always set this bit field to zero.
28	**CLR_GOT_A_NAK**	RW	0x0	Setting this bit will clear the got_a_nak. DO_NOTHING = 0x0 CLEAR = 0x1 Clear got_a_nak.

Table 25-24. HW_I2C_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
27	**ACK_MODE**	RW	0x0	This setting affects the behavior of the ACK pulse when RETAIN_CLOCK=1. ACK_AFTER_HOLD_LOW = 0x0 ACK will occur after clock is held low at start of next access. ACK_BEFORE_HOLD_LOW = 0x1 ACK will occur at end of access before clock is held low.
26	**FORCE_DATA_IDLE**	RW	0x0	Writing a one to this bit will force the data state machine to return to its idle state and stay there.
25	**FORCE_CLK_IDLE**	RW	0x0	Writing a one to this bit will force the clock generator state machine to return to its idle state and stay there.
24	**BCAST_SLAVE_EN**	RW	0x0	Set this bit to one to enable the slave address search machine to look for both a match to the programmed slave address as well as a match to the broadcast address of all zeroes. NO_BCAST = 0x0 Do not watch for broadcast address while matching programmed slave address. WATCH_BCAST = 0x1 Watch for the all zeroes broadcast address while matching programmed slave address.
23:16	**RSVD1**	RW	0x86	Program this field to 0x86.
15	**BUS_FREE_IRQ_EN**	RW	0x0	Set this bit to one to enable bus free interrupt requests to be routed to the interrupt collector. Set to zero to disable interrupts from the I2C controller. DISABLED = 0x0 No Interrupt Request Pending. ENABLED = 0x1 Interrupt Request Pending.
14	**DATA_ENGINE_CMPLT_IRQ_EN**	RW	0x0	Set this bit to one to enable data engine complete interrupt requests to be routed to the interrupt collector. Set to zero to disable interrupts from the I2C controller. DISABLED = 0x0 No Interrupt Request Pending. ENABLED = 0x1 Interrupt Request Pending.
13	**NO_SLAVE_ACK_IRQ_EN**	RW	0x0	Set this bit to one to enable interrupt requests to be routed to the interrupt collector. Set to zero to disable interrupts from the I2C controller. DISABLED = 0x0 No Interrupt Request Pending. ENABLED = 0x1 Interrupt Request Pending.
12	**OVERSIZE_XFER_TERM_IRQ_EN**	RW	0x0	Set this bit to one to enable interrupt requests to be routed to the interrupt collector. Set to zero to disable interrupts from the I2C controller. DISABLED = 0x0 No Interrupt Request Pending. ENABLED = 0x1 Interrupt Request Pending.
11	**EARLY_TERM_IRQ_EN**	RW	0x0	Set this bit to one to enable interrupt requests to be routed to the interrupt collector. Set to zero to disable interrupts from the I2C controller. DISABLED = 0x0 No Interrupt Request Pending. ENABLED = 0x1 Interrupt Request Pending.
10	**MASTER_LOSS_IRQ_EN**	RW	0x0	Set this bit to one to enable interrupt requests to be routed to the interrupt collector. Set to zero to disable interrupts from the I2C controller. DISABLED = 0x0 No Interrupt Request Pending. ENABLED = 0x1 Interrupt Request Pending.
9	**SLAVE_STOP_IRQ_EN**	RW	0x0	Set this bit to one to enable interrupt requests to be routed to the interrupt collector. Set to zero to disable interrupts from the I2C controller. DISABLED = 0x0 No Interrupt Request Pending. ENABLED = 0x1 Interrupt Request Pending.

Table 25-24. HW_I2C_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
8	**SLAVE_IRQ_EN**	RW	0x0	Set this bit to one to enable interrupt requests to be routed to the interrupt collector. Set to zero to disable interrupts from the I2C controller. The corresponding HW_I2C_CTRL1_SLAVE_IRQ interrupt bit is set by the slave search engine to indicate that it has stopped searching due to an address match or error. DISABLED = 0x0 No Interrupt Request Pending. ENABLED = 0x1 Interrupt Request Pending.
7	**BUS_FREE_IRQ**	RW	0x0	This bit is set to indicate that an interrupt is requested by the I2C controller because the bus has become free. This bit is cleared by software by writing a one to its SCT clear address. This interrupt indicates that the I2C bus, which was busy, has just become free. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.
6	**DATA_ENGINE_CMPLT_IRQ**	RW	0x0	This bit is set to indicate that an interrupt is requested by the I2C controller because the data engine transfer has completed. This bit is cleared by software by writing a one to its SCT clear address. This interrupt indicates that the data engine has completed a DMA transfer in either master or slave mode. This notification is useful for pio mode master write (transmit) or slave read (transmit) operations, i.e., data engine transmit operations. PIO receive operations are not supported. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.
5	**NO_SLAVE_ACK_IRQ**	RW	0x0	This bit is set to indicate that an interrupt is requested by the I2C controller because the slave addressed by a master transfer did not respond with an acknowledge. This bit is cleared by software by writing a one to its SCT clear address. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.
4	**OVERSIZE_XFER_TERM_IRQ**	RW	0x0	This bit is set to indicate that an interrupt is requested by the I2C controller. This bit is cleared by software by writing a one to its SCT clear address. This interrupt indicates that a master DMA transfer did not complete by the end of the transfer size. This is indicated by the slave acknowledging the last byte of a write transfer instead of NAKing it. The master should then send additional bytes of data if desired. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.
3	**EARLY_TERM_IRQ**	RW	0x0	This bit is set to indicate that an interrupt is requested by the I2C controller. This bit is cleared by software by writing a one to its SCT clear address. This interrupt indicates that a master write transfrom from the SoC to a slave device was NAKed by the slave before the transfer was completed. In slave mode, it indicates that the master NAKed a byte transmitted by the slave causing early termination of the expected transfer. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.

Table 25-24. HW_I2C_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**MASTER_LOSS_IRQ**	RW	0x0	This bit is set to indicate that an interrupt is requested by the I2C controller. This bit is cleared by software by writing a one to its SCT clear address. This interrupt indicates that a master read or write transaction lost an arbitration with another master. Master loss is indicated by the master attempting to transmit a one to the bus at the same time as another master writes a zero. The wired and bus produces a zero on the bus which is detected by the lossing master. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.
1	**SLAVE_STOP_IRQ**	RW	0x0	This bit is set to indicate that an I2C Stop Condition was received by the slave address search engine after it had found a start command addressed to its slave address. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.
0	**SLAVE_IRQ**	RW	0x0	This bit is set to indicate that an interrupt is requested by the I2C controller. This bit is cleared by software by writing a one to its SCT clear address. This bit is set by the slave search engine to indicate that it has stopped searching due to an address match or error. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.

DESCRIPTION:

This control register is primarily used for interrupt management. It also controls the special slave address matching mode. In addition, it controls the protocol speed, i.e fast or 400KHz versus normal or 100KHz operation.

EXAMPLE:

```
HW_I2C_CTRL1_CLR(BM_I2C_CTRL1_SLAVE_IRQ); // clear the slave interrupt
```

25.4.6 I2C Status Register Description

The I2C Controller reports status information in the I2C Status Register.

HW_I2C_STAT 0x050

Table 25-25. HW_I2C_STAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MASTER_PRESENT	RSVD2	ANY_ENABLED_IRQ	GOT_A_NAK	RSVD1				RCVD_SLAVE_ADDR								SLAVE_ADDR_EQ_ZERO	SLAVE_FOUND	SLAVE_SEARCHING	DATA_ENGINE_DMA_WAIT	BUS_BUSY	CLK_GEN_BUSY	DATA_ENGINE_BUSY	SLAVE_BUSY	BUS_FREE_IRQ_SUMMARY	DATA_ENGINE_CMPLT_IRQ_SUMMARY	NO_SLAVE_ACK_IRQ_SUMMARY	OVERSIZE_XFER_TERM_IRQ_SUMMARY	EARLY_TERM_IRQ_SUMMARY	MASTER_LOSS_IRQ_SUMMARY	SLAVE_STOP_IRQ_SUMMARY	SLAVE_IRQ_SUMMARY

Table 25-26. HW_I2C_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**MASTER_PRESENT**	RO	0x1	This read-only bit indicates that the I2C master function is present when it reads back a one. This I2C function is not available on a device that returns a zero for this bit field. UNAVAILABLE = 0x0 I2C is not present in this product. AVAILABLE = 0x1 I2C is present in this product.
30	**RSVD2**	RO	0x1	Program this field to 0x1.
29	**ANY_ENABLED_IRQ**	RO	0x0	This read-only bit indicates that the I2C controller has at least one enable interrupt requesting service. It is the logic OR of all of the IRQ summary bits. NO_REQUESTS = 0x0 No enabled interrupts are requesting service. AT_LEAST_ONE_REQUEST = 0x1 At least one of the summary interrupt bits is set.
28	**GOT_A_NAK**	RO	0x0	Read-only view of the got-a-nak signal. NO_NAK = 0x0 I2C master has not detected a NAK. DETECTED_NAK = 0x1 I2C master has detected a NAK.
27:24	**RSVD1**	RO	0x0	Always set this bit field to zero.
23:16	**RCVD_SLAVE_ADDR**	RO	0x00	This read-only byte indicates that the state of the slave I2C address byte received, including the read/write bit received from an address byte that matched our slave address.
15	**SLAVE_ADDR_EQ_ZERO**	RO	0x0	This read-only bit indicates that the I2C slave function was searching for a transaction that matches the current slave address. When set to one, it indicates that an address match was found for the exact adderss 0x00. ZERO_NOT_MATCHED = 0x0 I2C slave search did not match a zero. WAS_ZERO = 0x1 I2C has found an address match against address 0x00.

Table 25-26. HW_I2C_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
14	**SLAVE_FOUND**	RO	0x0	This read-only bit indicates that the I2C slave function was searching for a transaction that matches the current slave address. When set to one, it indicates that an address match was found and the I2C clock is frozen by the slave search. This bit is cleared by starting the appropriate slave DMA transfer or restarting a slave search. IDLE = 0x0 I2C slave search is idle. WAITING = 0x1 I2C has found an address match and is holding the I2C clock line low.
13	**SLAVE_SEARCHING**	RO	0x0	This read-only bit indicates that the I2C slave function is searching for a transaction that matches the current slave address. IDLE = 0x0 I2C slave search is idle. ACTIVE = 0x1 I2C is actively searching for an address match.
12	**DATA_ENGINE_DMA_WAIT**	RO	0x0	This read-only bit is set to one when the data engine is waitng for data from a DMA device. This bit can be used to transmit short I2C transactions without using a DMA channel. This generally works for up to three data bytes transmitted with one address byte. CONTINUE = 0x0 I2C master is not waiting on data from the DMA. WAITING = 0x1 I2C master is waiting on data from the DMA.
11	**BUS_BUSY**	RO	0x0	This read-only bit indicates that the I2C bus is busy with a transaction. It is set by a start condition and reset by a detected stop condition. IDLE = 0x0 I2C bus is idle, i.e. reset state or at least one stop condition detected. BUSY = 0x1 I2C bus is busy, i.e. at least one start condition has been detected.
10	**CLK_GEN_BUSY**	RO	0x0	This read-only bit indicates that the I2C clock generator is busy with a transaction. IDLE = 0x0 I2C clock generator is idle. BUSY = 0x1 I2C clock generator is busy performing a command.
9	**DATA_ENGINE_BUSY**	RO	0x0	This read-only bit indicates that the I2C data transfer engine is busy with a data transmit or recieve opertion. In addition, it can be busy, as a master, sending a start or stop condition. IDLE = 0x0 I2C Data Engine is idle. BUSY = 0x1 I2C is Data Engine busy performing a data transfer.
8	**SLAVE_BUSY**	RO	0x0	This read-only bit indicates that the I2C slave address search engine is busy with a transaction. This bit will go high when an address search is started and will remain high until the slave search engine returns to its idle state. IDLE = 0x0 I2C slave search engine is idle. BUSY = 0x1 I2C slave search engine is busy searching for an address match.
7	**BUS_FREE_IRQ_SUMMARY**	RO	0x0	This bit is set to indicate that an interrupt is requested by the I2C controller. It is a logical AND of the corresponding interrupt status bit and interrupt enable bit. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.
6	**DATA_ENGINE_CMPLT_IRQ_SUMMARY**	RO	0x0	This bit is set to indicate that an interrupt is requested by the I2C controller. It is a logical AND of the corresponding interrupt status bit and interrupt enable bit. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.

Table 25-26. HW_I2C_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
5	**NO_SLAVE_ACK_IRQ_SUMMARY**	RO	0x0	This bit is set to indicate that an interrupt is requested by the I2C controller. It is a logical AND of the corresponding interrupt status bit and interrupt enable bit. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.
4	**OVERSIZE_XFER_TERM_IRQ_SUMMARY**	RO	0x0	This bit is set to indicate that an interrupt is requested by the I2C controller. It is a logical AND of the corresponding interrupt status bit and interrupt enable bit. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.
3	**EARLY_TERM_IRQ_SUMMARY**	RO	0x0	This bit is set to indicate that an interrupt is requested by the I2C controller. It is a logical AND of the corresponding interrupt status bit and interrupt enable bit. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.
2	**MASTER_LOSS_IRQ_SUMMARY**	RO	0x0	This bit is set to indicate that an interrupt is requested by the I2C controller. It is a logical AND of the corresponding interrupt status bit and interrupt enable bit. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.
1	**SLAVE_STOP_IRQ_SUMMARY**	RO	0x0	This bit is set to indicate that an interrupt is requested by the I2C controller. It is a logical AND of the corresponding interrupt status bit and interrupt enable bit. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.
0	**SLAVE_IRQ_SUMMARY**	RO	0x0	This bit is set to indicate that an interrupt is requested by the I2C controller. It is a logical AND of the corresponding interrupt status bit and interrupt enable bit. NO_REQUEST = 0x0 No Interrupt Request Pending. REQUEST = 0x1 Interrupt Request Pending.

DESCRIPTION:

The status register provides read-only access to the function presence bits, as well as the busy indicators for the state machines.

EXAMPLE:

```
while(HW_I2C_STAT.SLAVE_BUSY != BV_I2C_STAT_SLAVE_BUSY__IDLE_VAL);// then wait till it finishes
```

25.4.7 I2C Controller DMA Read and Write Data Register Description

The I2C Controller DMA Read and Write Data Register is the target for both source and destination DMA transfers. This register is backed by an eight-deep FIFO.

HW_I2C_DATA 0x060

Table 25-27. HW_I2C_DATA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DATA																															

Table 25-28. HW_I2C_DATA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**DATA**	RW	0x00000000	The source DMA channel writes to this address. The Destination DMA channel reads from this address.

DESCRIPTION:

DMA reads and writes are directed to this register.

EXAMPLE:

```
The DMA data register is used by the DMA to read or write data from the I2C controller, as mediated
by the I2C controller`s DMA request signal.
```

25.4.8 I2C Device Debug Register 0 Description

The I2C Device Debug Register 0 provides a diagnostic view into the internal state machine and states of the I2C device.

HW_I2C_DEBUG0	0x070
HW_I2C_DEBUG0_SET	0x074
HW_I2C_DEBUG0_CLR	0x078
HW_I2C_DEBUG0_TOG	0x07C

Table 25-29. HW_I2C_DEBUG0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DMAREQ	DMAENDCMD	DMAKICK	DMATERMINATE	TBD		DMA_STATE										START_TOGGLE	STOP_TOGGLE	GRAB_TOGGLE	CHANGE_TOGGLE	TESTMODE	SLAVE_HOLD_CLK	SLAVE_STATE									

Table 25-30. HW_I2C_DEBUG0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**DMAREQ**	RO	0x0	Read-only view of the toggle state of the DMA request signal.
30	**DMAENDCMD**	RO	0x0	Read-only view of the toggle state of the DMA End Command signal.
29	**DMAKICK**	RO	0x0	Read-only view of the toggle state of the DMA Kick signal.

Table 25-30. HW_I2C_DEBUG0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
28	**DMATERMINATE**	RO	0x0	Read-only view of the toggle state of the DMA Terminate signal.
27:26	**TBD**	RW	0x0	Reserved
25:16	**DMA_STATE**	RO	0x010	Current state of the DMA state machine.
15	**START_TOGGLE**	RO	0x0	Read-only view of the start detector. Toggles once for each detected start condition.
14	**STOP_TOGGLE**	RO	0x0	Read-only view of the stop detector. Toggles once for each detected stop condition.
13	**GRAB_TOGGLE**	RO	0x0	Read-only view of the grab receive data timing point. Toggles once for each read timing point, as delayed from rising clock.
12	**CHANGE_TOGGLE**	RO	0x0	Read-only view of the change xmit data timing point. Toggles once for each change xmit data timing point, as delayed from falling clock.
11	**TESTMODE**	RW	0x0	To be completed by designer.
10	**SLAVE_HOLD_CLK**	RO	0x0	Current State of the Slave Address Search FSM clock hold register.
9:0	**SLAVE_STATE**	RO	0x0000	Current State of the Slave Address Search FSM.

DESCRIPTION:

This register provides access to various internal states and controls that are used in diagnostic modes of operation.

EXAMPLE:

```
while(HW_I2C_DEBUG0.DMAREQ == old_dma_req_value); // wait for next dma request toggle
old_dma_req_value = HW_I2C_DEBUG0.DMAREQ; // remember the new state of the dma request toggle
```

25.4.9 I2C Device Debug Register 1 Description

The I2C Device Debug Register 1 provides a diagnostic view of the external bus and provides OE control for the clock and data.

HW_I2C_DEBUG1	0x080
HW_I2C_DEBUG1_SET	0x084
HW_I2C_DEBUG1_CLR	0x088
HW_I2C_DEBUG1_TOG	0x08C

Table 25-31. HW_I2C_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
I2C_CLK_IN	I2C_DATA_IN	RSVD4		DMA_BYTE_ENABLES				CLK_GEN_STATE								RSVD2				LST_MODE			LOCAL_SLAVE_TEST	RSVD1			FORCE_CLK_ON	FORCE_ARB_LOSS	FORCE_RCV_ACK	FORCE_I2C_DATA_OE	FORCE_I2C_CLK_OE

Table 25-32. HW_I2C_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**I2C_CLK_IN**	RO	0x1	A copy of the pad input signal for the I2C clock pad.
30	**I2C_DATA_IN**	RO	0x1	A copy of the pad input signal for the I2C clock pad.
29:28	**RSVD4**	RO	0x0	Always set this bit field to zero.
27:24	**DMA_BYTE_ENABLES**	RO	0x0	A read-only view of the byte enables for HW_I2C_DATA register writes. These bits are used in the I2C DMA state machine to track the number of bytes written by the DMA. Individual bits are cleared as they are consummed.
23:16	**CLK_GEN_STATE**	RO	0x0	A read-only view of the byte enables for HW_I2C_DATA register writes. These bits are used in the I2C DMA state machine to track the number of bytes written by the DMA. Individual bits are cleared as they are consummed.
15:11	**RSVD2**	RO	0x0	Always set this bit field to zero.
10:9	**LST_MODE**	RW	0x0	When in local slave test mode, this bit field defines the type of address generated for the slave. BCAST = 0x0 Broadcast, i.e. i2c address 0x00. MY_WRITE = 0x1 Send to my slave address with a RW bit equal 0. MY_READ = 0x2 Send to my slave address with a RW bit equal 1. NOT_ME = 0x3 Send to an address that is not mine, i.e. bit four is complemented.
8	**LOCAL_SLAVE_TEST**	RW	0x0	Writting a one to this bit places the slave in local test mode. one of three slave address can be sent in either read or write mode.
7:5	**RSVD1**	RO	0x0	Always set this bit field to zero.
4	**FORCE_CLK_ON**	RW	0x0	Writing a one to this bit will force the clock generator to send a continuous stream of clocks on the I2C bus.
3	**FORCE_ARB_LOSS**	RW	0x0	Writing a one to this bit will force the appearance of an arbitration loss on the next one a master attempts to transmit.
2	**FORCE_RCV_ACK**	RW	0x0	Writing a one to this bit will force the appearance of a receive acknowledge to the byte level state machine at bit 9 of the transfer.
1	**FORCE_I2C_DATA_OE**	RW	0x0	Writting a one to this bit will force an output enable at the pad. The pad data line is tied to zero. Thus the I2C data line will either be hi-z or zero.
0	**FORCE_I2C_CLK_OE**	RW	0x0	Writing a one to this bit will force an output enable at the pad. The pad data line is tied to zero. Thus the I2C clock line will either be hi-z or zero.

DESCRIPTION:

This register provides access to the I2C clock and data pad cell state that are used in diagnostic modes of operation.

EXAMPLE:

```
while(HW_I2C_DEBUG1.I2C_CLK_IN == 0); // wait for I2C clock line to go high
```

25.4.10 I2C Version Register Description

This register always returns a known read value for debug purposes it indicates the version of the block.

HW_I2C_VERSION 0x090

Table 25-33. HW_I2C_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 25-34. HW_I2C_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x01	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x02	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
if (HW_I2C_VERSION.B.MAJOR != 1) Error();
```

I2C Block v1.3, Revision 1.54

Chapter 26
Application UART

This chapter describes the Application UART included on the i.MX23, how to operate it, and how to disable the FIFOs. Programmable registers are described in Section 26.4, "Programmable Registers."

26.1 Overview

The Application UART:

- Performs serial-to-parallel conversion on data received from a peripheral device.
- Performs parallel-to-serial conversion on data transmitted to the peripheral device.
- Operates up to 3.25 Mb/s.
- IrDA is not supported.

The Application UART has certain package dependencies:

- Application UART2 is only available in the 169-pin BGA package.
- Application UART1 flow control is not avaiable in the 128-pin LQFP package.

The CPU or DMA controller reads and writes data and control/status information through the APBX interface. The transmit and receive paths are buffered with internal FIFO memories, enabling up to 16-bytes to be stored independently in both transmit and receive modes.

The Application UART includes a programmable baud rate generator that generates a transmit and receive internal clock from the 24-MHz UART internal reference clock input UARTCLK. XCLK is not tied to the UARTCLK in the i.MX23.

It offers similar functionality to the industry-standard 16C550 UART device and supports baud rates of up to 3.25 Mbits/s (in high-speed configuration with a minimum XCLK frequency of 1.5 MHz). Figure 26-1 shows a block diagram of the Application UART. The Application UART operation and baud rate values are controlled by the line control register (HW_UARTAPP_LINECTRL). The HW_UARTAPP_LINECTRL register controls both receive and transmit operations. However, when HW_UARTAPP_CTRL2_USE_LCR2 is set, then HW_UARTAPP_LINECTRL controls receive operations and HW_UARTAPP_LINECTRL2 controls transmit operations.

The Application UART can generate a single combined interrupt, so that the output is asserted if any of the individual interrupts are asserted and unmasked. Interrupt sources include the receive (including timeout), transmit, modem status, and error conditions.

Two DMA channels are supported, one for transmit and one for receive.

If a timeout condition occurs in the middle of a receive DMA block transfer, then the UART ends the DMA transfer and signals the end of the DMA block transfer. A receive DMA can be setup to get the status of the previous receive DMA block transfer. The status indicates the amount of valid data bytes in the previous receive DMA block transfer.

If a framing, parity, or break error occurs during reception, the appropriate error bit is set and stored in the FIFO. If an overrun condition occurs, the overrun register bit is set immediately and FIFO data is prevented from being overwritten. You can program the FIFOs to be one-byte deep, providing a conventional double-buffered UART interface.

The modem status input signal Clear To Send (CTS) and output modem control line Request To Send (RTS) are supported. A programmable hardware flow control feature uses the nUARTCTS input and the nUARTRTS output to automatically control the serial data flow.

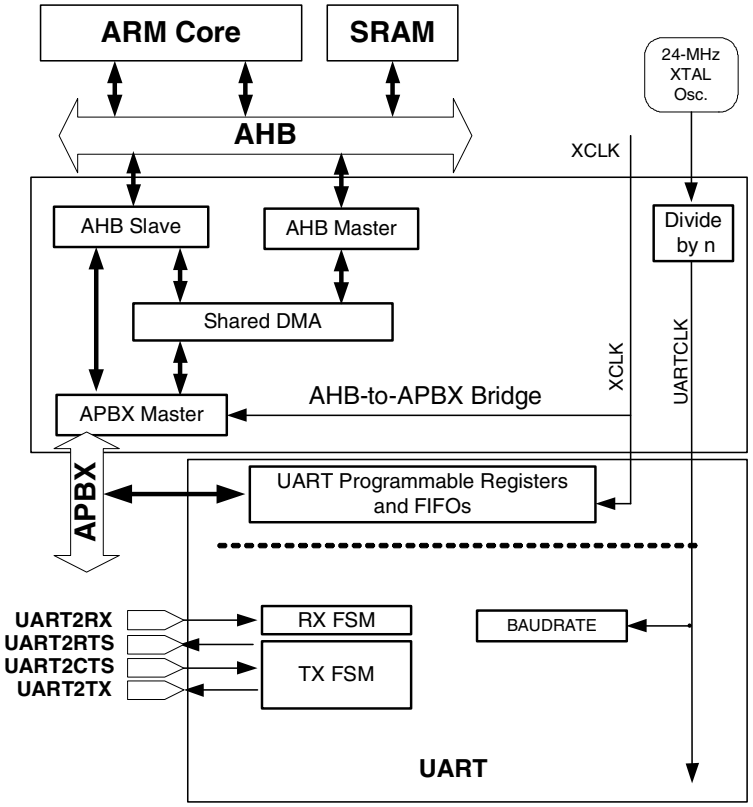


Figure 26-1. Application UART Block Diagram

26.2 Operation

Control data is written to the Application UART line control register. This register defines:

- Transmission parameters
- Word length
- Buffer mode

- Number of transmitted stop bits
- Parity mode
- Break generation
- Baud rate divisor

If USE_LCR2 is set, the Application UART Line Control Register applies to the receive operation, and similar control data written to the Application UART Line Control 2 Register applies to the transmit operation.

26.2.1 Fractional Baud Rate Divider

The baud rate divisor is calculated from the frequency of UARTCLK and the desired baud rate by using the following formula:

```
divisor = (UARTCLK * 32) / baud rate, rounded to the nearest integer
```

The divisor must be between 0x000000EC and 0x003FFFC0, inclusive. Program the lowest 6 bits of the divisor into BAUD_DIVFRAC, and the next 16 bits of the divisor into BAUD_DIVINT.

26.2.2 UART Character Frame

Figure 26-2 illustrates the UART character frame.

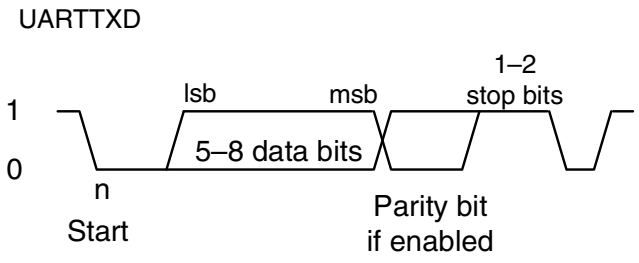


Figure 26-2. Application UART Character Frame

26.2.3 DMA Operation

The Application UART can generate a DMA request signal for interfacing with a Direct Memory Access (DMA) controller. Two DMA channels are supported, one for transmit and one for receive. Each channel has an associated 16-bit transfer counter for the number of bytes to transfer. Each DMA request is associated with one to four data bytes. For APBX DMA Channel 6, which is the UART RX channel, the first PIO word in the DMA command is CTRL0. However, for APBX DMA Channel 7, which is the UART TX, the first PIO word in a DMA command is CTRL1.

At the end of a receive DMA block transfer, the status register indicates any error conditions. If a timeout condition occurs in the middle of a receive DMA block transfer, then the UART sends dummy data to the DMA controller until the transfer counter is decremented to zero. A receive DMA can be setup to get the

status of the previous receive DMA block transfer. The status indicates the amount of valid data bytes in the previous receive DMA block transfer.

26.2.4 Data Transmission or Reception

Data received or transmitted is stored in two 16-byte FIFOs, although the receive FIFO has an extra four bits per character for status information.

For transmission, data is written into the transmit FIFO. If the Application UART is enabled, it causes a data frame to start transmitting with the parameters indicated in UARTLCR_H or UARTLCR2_H (if USE_LCR2 is set). Data continues to be transmitted until there is no data left in the transmit FIFO.

The BUSY signal goes HIGH as soon as data is written to the transmit FIFO (that is, the FIFO is non-empty) and remains asserted HIGH while data is being transmitted. BUSY is negated only when the transmit FIFO is empty, and the last character has been transmitted from the shift register, including the stop bits. BUSY can be asserted HIGH, even though the Application UART might no longer be enabled.

For each sample of data, three readings are taken and the majority value is kept. In the following paragraphs, the middle sampling point is defined, and one sample is taken on either side of it.

- When the receiver is idle (UARTRXD continuously 1, in the marking state) and a LOW is detected on the data input (a start bit has been received), the receive counter, with the clock enabled by BaudClk, begins running and data is sampled on the first cycle of that counter in normal UART mode to allow for the shorter logic 0 pulses (half way through a bit period).
- The start bit is valid if UARTRXD is still LOW on the first cycle of BaudClk, otherwise a false start bit is detected and it is ignored. If the start bit was valid, successive data bits are sampled on every second cycle of BaudClk (that is, one bit period later) according to the programmed length of the data characters. The parity bit is then checked if parity mode was enabled.
- Lastly, a valid stop bit is confirmed if UARTRXD is HIGH, otherwise a framing error has occurred. When a full word is received, the data is stored in the receive FIFO, with any error bits associated with that word (see Table 26-1).

26.2.5 Error Bits

Three error bits are stored in bits [10:8] of the receive FIFO and are associated with a particular character. An additional error indicating an overrun error is stored in bit 11 of the receive FIFO.

26.2.6 Overrun Bit

The overrun bit is not associated with the character in the receive FIFO. The overrun error is set when the FIFO is full and the next character is completely received in the shift register. The data in the shift register is overwritten, but it is not written into the FIFO. When an empty location is available in the receive FIFO and another character is received, the state of the overrun bit is copied into the receive FIFO along with

the received character. The overrun state is then cleared. Table 26-1 shows the bit functions of the receive FIFO.

Table 26-1. Receive FIFO Bit Functions

FIFO BIT	FUNCTION
11	Overrun indicator
10	Break error
9	Parity error
8	Framing error
7:0	Received data

26.2.7 Disabling the FIFOs

FIFOs can be disabled. In this case, the transmit and receive sides of the Application UART have one-byte holding registers (the bottom entry of the FIFOs). The overrun bit is set when a word has been received and the previous one was not yet read.

In this implementation, the FIFOs are not physically disabled, but the flags are manipulated to give the illusion of a one-byte register.

26.3 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, “Correct Way to Soft Reset a Block.” for additional information on using the SFTRST and CLKGATE bit fields.

26.4 Programmable Registers

This section describes the Application UART’s programable registers.

26.4.1 UART Receive DMA Control Register Description

The UART Receive DMA Control Register contains the dynamic information associated with the receive command.

HW_UARTAPP_CTRL0	0x000
HW_UARTAPP_CTRL0_SET	0x004
HW_UARTAPP_CTRL0_CLR	0x008
HW_UARTAPP_CTRL0_TOG	0x00C

Table 26-2. HW_UARTAPP_CTRL0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	RUN	RX_SOURCE	RXTO_ENABLE	RXTIMEOUT											XFER_COUNT															

Table 26-3. HW_UARTAPP_CTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	Set to zero for normal operation. When this bit is set to one (default), then the entire block is held in its reset state.
30	**CLKGATE**	RW	0x1	Set this bit zero for normal operation. Setting this bit to one (default), gates all of the block level clocks off for miniminizing AC energy consumption.
29	**RUN**	RW	0x0	Tell the UART to execute the RX DMA Command. The UART will clear this bit at the end of receive execution.
28	**RX_SOURCE**	RW	0x0	Source of Receive Data. If this bit is set to 1, the status register will be the source of the DMA, otherwise RX data will be the source.
27	**RXTO_ENABLE**	RW	0x0	RXTIMEOUT Enable: If this bit is set to 0, the RX timeout will not affect receive DMA operation. If this bit is set to 1, a receive timeout will cause the receive DMA logic to terminate.
26:16	**RXTIMEOUT**	RW	0x03	Receive Timeout Counter Value: number of 8-bit-time to wait before asserting timeout on the RX input. If the RXFIFO is not empty and the RX input is idle, then the watchdog counter will decrement each bit-time. Note 7-bit-time is added to the programmed value, so a value of zero will set the counter to 7-bit-time, a value of 0x1 gives 15-bit-time and so on. Also note that the counter is reloaded at the end of each frame, so if the frame is 10 bits long and the timeout counter value is zero, then timeout will occur (when FIFO is not empty) even if the RX input is not idle. The default value is 0x3 (31 bit-time).
15:0	**XFER_COUNT**	RW	0x00	Number of bytes to receive. This must be a multiple of 4.

DESCRIPTION:

This register contains the main DMA controls for Receiving data.

EXAMPLE:

```
No Example.
```

26.4.2 UART Transmit DMA Control Register Description

The UART Transmit DMA Control Register contains the dynamic information associated with the transmit command.

HW_UARTAPP_CTRL1	0x010
HW_UARTAPP_CTRL1_SET	0x014
HW_UARTAPP_CTRL1_CLR	0x018
HW_UARTAPP_CTRL1_TOG	0x01C

Table 26-4. HW_UARTAPP_CTRL1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD2			RUN	RSVD1												XFER_COUNT															

Table 26-5. HW_UARTAPP_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSVD2**	RO	0x0	Reserved, read as zero, do not modify.
28	**RUN**	RW	0x0	Tell the UART to execute the TX DMA Command. The UART will clear this bit at the end of transmit execution.
27:16	**RSVD1**	RO	0x0	Reserved, read as zero, do not modify.
15:0	**XFER_COUNT**	RW	0x00	Number of bytes to transmit.

DESCRIPTION:

This register contains the main DMA controls for Transmitting data.

EXAMPLE:

```
No Example.
```

26.4.3 UART Control Register Description

The UART Control Register contains configuation, including interrupt FIFO level select and the DMA control.

HW_UARTAPP_CTRL2	0x020
HW_UARTAPP_CTRL2_SET	0x024
HW_UARTAPP_CTRL2_CLR	0x028
HW_UARTAPP_CTRL2_TOG	0x02C

Table 26-6. HW_UARTAPP_CTRL2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
INVERT_RTS	INVERT_CTS	INVERT_TX	INVERT_RX	RTS_SEMAPHORE	DMAONERR	TXDMAE	RXDMAE	RSVD5	RXIFLSEL			RSVD4	TXIFLSEL			CTSEN	RTSEN	OUT2	OUT1	RTS	DTR	RXE	TXE	LBE	USE_LCR2	RSVD3			RSVD2	RSVD1	UARTEN

Table 26-7. HW_UARTAPP_CTRL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**INVERT_RTS**	RW	0x0	Invert RTS signal. If this bit is set to 1, the RTS output is inverted before transmitted.
30	**INVERT_CTS**	RW	0x0	Invert CTS signal. If this bit is set to 1, the CTS input is inverted before sampled.
29	**INVERT_TX**	RW	0x0	Invert TX signal. If this bit is set to 1, the TX output is inverted before transmitted.
28	**INVERT_RX**	RW	0x0	Invert RX signal. If this bit is set to 1, the RX input is inverted before sampled.
27	**RTS_SEMAPHORE**	RW	0x0	If this bit is set to 1, RTS is deasserted when the semaphore threshold is less than 2.
26	**DMAONERR**	RW	0x0	DMA On Error. If this bit is set to 1, receive dma will terminate on error. (Cmd_end signal may not be asserted when this occurs.)
25	**TXDMAE**	RW	0x0	Transmit DMA Enable. Data Register can be loaded with up to 4 bytes per write. TXFIFO must be enabled in TXDMA mode.
24	**RXDMAE**	RW	0x0	Receive DMA Enable. Data Register can be contain up to 4 bytes per read. RXFIFO must be enabled in RXDMA mode.
23	**RSVD5**	RO	0x0	Reserved, do not modify, read as zero.
22:20	**RXIFLSEL**	RW	0x2	Receive Interrupt FIFO Level Select. The trigger points for the receive interrupt are as follows: NOT_EMPTY = 0x0 Trigger on FIFO not empty, i.e., at least 1 of 16 entries. ONE_QUARTER = 0x1 Trigger on FIFO full to at least 4 of 16 entries. ONE_HALF = 0x2 Trigger on FIFO full to at least 8 of 16 entries. THREE_QUARTERS = 0x3 Trigger on FIFO full to at least 12 of 16 entries. SEVEN_EIGHTHS = 0x4 Trigger on FIFO full to at least 14 of 16 entries. INVALID5 = 0x5 Reserved. INVALID6 = 0x6 Reserved. INVALID7 = 0x7 Reserved.
19	**RSVD4**	RO	0x0	Reserved, do not modify, read as zero.
18:16	**TXIFLSEL**	RW	0x2	Transmit Interrupt FIFO Level Select. The trigger points for the transmit interrupt are as follows: EMPTY = 0x0 Trigger on FIFO empty, i.e., no entries. ONE_QUARTER = 0x1 Trigger on FIFO less than 4 of 16 entries. ONE_HALF = 0x2 Trigger on FIFO less than 8 of 16 entries. THREE_QUARTERS = 0x3 Trigger on FIFO less than 12 of 16 entries. SEVEN_EIGHTHS = 0x4 Trigger on FIFO less than 14 of 16 entries. INVALID5 = 0x5 Reserved. INVALID6 = 0x6 Reserved. INVALID7 = 0x7 Reserved.

Table 26-7. HW_UARTAPP_CTRL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15	**CTSEN**	RW	0x0	CTS Hardware Flow Control Enable. If this bit is set to 1, CTS hardware flow control is enabled. Data is only transmitted when the nUARTCTS signal is asserted.
14	**RTSEN**	RW	0x0	RTS Hardware Flow Control Enable. If this bit is set to 1, RTS hardware flow control is enabled. Data is only requested when there is space in the receive FIFO for it to be received. The FIFO space is controlled by RXIFLSEL value.
13	**OUT2**	RW	0x0	This bit is the complement of the UART Out2 (nUARTOut2) modem status output. This bit is not supported.
12	**OUT1**	RW	0x0	This bit is the complement of the UART Out1 (nUARTOut1) modem status output. This bit is not supported.
11	**RTS**	RW	0x0	Request To Send. Software can manually control the nUARTRTS pin via this bit when RTSEN = 0. This bit is the complement of the UART request to send (nUARTRTS) modem status output. That is, when the bit is programmed to a 1, the output is 0.
10	**DTR**	RW	0x0	Data Transmit Ready. This bit is the complement of the UART data transmit ready (nUARTDTR) modem status output. This bit is not supported.
9	**RXE**	RW	0x1	Receive Enable. If this bit is set to 1, the receive section of the UART is enabled. Data reception occurs for the UART signals. When the UART is disabled in the middle of reception, it completes the current character before stopping.
8	**TXE**	RW	0x1	Transmit Enable. If this bit is set to 1, the transmit section of the UART is enabled. Data transmission occurs for the UART signals. When the UART is disabled in the middle of transmission, it completes the current character before stopping.
7	**LBE**	RW	0x0	Loop Back Enable. This feature reduces the amount of external coupling required during system test. If this bit is set to 1, the UARTTXD path is fed through to the UARTRXD path. When this bit is set, the modem outputs are also fed through to the modem inputs.
6	**USE_LCR2**	RW	0x0	=If this bit is set to 1, the Line Control 2 Register values are used.
5:3	**RSVD3**	RO	0x0	Reserved, do not modify, read as zero.
2	**RSVD2**	RW	0x0	Program this field to 0x0.
1	**RSVD1**	RW	0x0	Program this field to 0x0.
0	**UARTEN**	RW	0x0	UART Enable. If this bit is set to 1, the UART is enabled. Data transmission and reception occurs for the UART signals. When the UART is disabled in the middle of transmission or reception, it completes the current character before stopping.

DESCRIPTION:

Use this register to define the FIFO level at which the UARTTXINTR and UARTRXINTR are triggered. The interrupts are generated based on a transition through a level rather than being based on the level. That

is, the design is such that the interrupts are generated when the fill level progresses through the trigger level. The bits are reset so that the trigger level is when the FIFOs are at the half-way mark.

EXAMPLE:

```
No Example.
```

26.4.4 UART Line Control Register Description

The UART Line Control Register contains integer and fractional part of the baud rate divisor value. It also contains the line control bits.

HW_UARTAPP_LINECTRL	0x030
HW_UARTAPP_LINECTRL_SET	0x034
HW_UARTAPP_LINECTRL_CLR	0x038
HW_UARTAPP_LINECTRL_TOG	0x03C

Table 26-8. HW_UARTAPP_LINECTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BAUD_DIVINT																RSVD		BAUD_DIVFRAC						SPS	WLEN			FEN	STP2	EPS	PEN	BRK

Table 26-9. HW_UARTAPP_LINECTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**BAUD_DIVINT**	RW	0x0	Baud Rate Integer [15:0]. The integer baud rate divisor.
15:14	**RSVD**	RO	0x0	Reserved, do not modify, read as zero.
13:8	**BAUD_DIVFRAC**	RW	0x0	Baud Rate Fraction [5:0]. The fractional baud rate divisor.
7	**SPS**	RW	0x0	Stick Parity Select. When bits 1, 2, and 7 of this register are set, the parity bit is transmitted and checked as a 0. When bits 1 and 7 are set, and bit 2 is 0, the parity bit is transmitted and checked as a 1. When this bit is cleared stick parity is disabled.
6:5	**WLEN**	RW	0x0	Word length [1:0]. The select bits indicate the number of data bits transmitted or received in a frame as follows: 11 = 8 bits, 10 = 7 bits, 01 = 6 bits, 00 = 5 bits.
4	**FEN**	RW	0x0	Enable FIFOs. If this bit is set to 1, transmit and receive FIFO buffers are enabled (FIFO mode). When cleared to 0, the FIFOs are disabled (character mode); that is, the FIFOs become 1-byte-deep holding registers.
3	**STP2**	RW	0x0	Two Stop Bits Select. If this bit is set to 1, two stop bits are transmitted at the end of the frame. The receive logic does not check for two stop bits being received.

Table 26-9. HW_UARTAPP_LINECTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**EPS**	RW	0x0	Even Parity Select. If this bit is set to 1, even parity generation and checking is performed during transmission and reception, which checks for an even number of 1s in data and parity bits. When cleared to 0, then odd parity is performed which checks for an odd number of 1s. This bit has no effect when parity is disabled by Parity Enable (PEN, bit 1) being cleared to 0.
1	**PEN**	RW	0x0	Parity Enable. If this bit is set to 1, parity checking and generation is enabled, else parity is disabled and no parity bit added to the data frame.
0	**BRK**	RW	0x0	Send Break. If this bit is set to 1, a low-level is continually output on the UARTTXD output, after completing transmission of the current character. For the proper execution of the break command, the software must set this bit for at least two complete frames. For normal use, this bit must be cleared to 0.

DESCRIPTION:

The UART Line Control Register contains integer and fractional part of the baud rate divisor value. It also contains the line control bits.

EXAMPLE:

```
No Example.
```

26.4.5 UART Line Control 2 Register Description

The UART Line Control 2 Register contains integer and fractional part of the baud rate divisor value. It also contains the line control bits.

HW_UARTAPP_LINECTRL2	0x040
HW_UARTAPP_LINECTRL2_SET	0x044
HW_UARTAPP_LINECTRL2_CLR	0x048
HW_UARTAPP_LINECTRL2_TOG	0x04C

Table 26-10. HW_UARTAPP_LINECTRL2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BAUD_DIVINT																RSVD		BAUD_DIVFRAC						SPS	WLEN			FEN	STP2	EPS	PEN	RSVD1

Table 26-11. HW_UARTAPP_LINECTRL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**BAUD_DIVINT**	RW	0x0	Baud Rate Integer [15:0]. The integer baud rate divisor.
15:14	**RSVD**	RO	0x0	Reserved, do not modify, read as zero.
13:8	**BAUD_DIVFRAC**	RW	0x0	Baud Rate Fraction [5:0]. The fractional baud rate divisor.
7	**SPS**	RW	0x0	Stick Parity Select. When bits 1, 2, and 7 of this register are set, the parity bit is transmitted and checked as a 0. When bits 1 and 7 are set, and bit 2 is 0, the parity bit is transmitted and checked as a 1. When this bit is cleared stick parity is disabled.
6:5	**WLEN**	RW	0x0	Word length [1:0]. The select bits indicate the number of data bits transmitted or received in a frame as follows: 11 = 8 bits, 10 = 7 bits, 01 = 6 bits, 00 = 5 bits.
4	**FEN**	RW	0x0	Enable FIFOs. If this bit is set to 1, transmit and receive FIFO buffers are enabled (FIFO mode). When cleared to 0, the FIFOs are disabled (character mode); that is, the FIFOs become 1-byte-deep holding registers.
3	**STP2**	RW	0x0	Two Stop Bits Select. If this bit is set to 1, two stop bits are transmitted at the end of the frame. The receive logic does not check for two stop bits being received.
2	**EPS**	RW	0x0	Even Parity Select. If this bit is set to 1, even parity generation and checking is performed during transmission and reception, which checks for an even number of 1s in data and parity bits. When cleared to 0, then odd parity is performed which checks for an odd number of 1s. This bit has no effect when parity is disabled by Parity Enable (PEN, bit 1) being cleared to 0.
1	**PEN**	RW	0x0	Parity Enable. If this bit is set to 1, parity checking and generation is enabled, else parity is disabled and no parity bit added to the data frame.
0	**RSVD1**	RO	0x0	Reserved, do not modify, read as zero.

DESCRIPTION:

The UART Line Control 2 Register contains integer and fractional part of the baud rate divisor value. It also contains the line control bits.

EXAMPLE:

```
No Example.
```

26.4.6 UART Interrupt Register Description

The UART Interrupt Register contains the interrupt enables and the interrupt status. The interrupt status bits report the unmasked state of the interrupts. To clear a particular interrupt status bit, write the bit-clear address with the particular bit set to 1. The enable bits control the UART interrupt output: a 1 will enable a particular interrupt to assert the UART interrupt output, while a 0 will disable the particular interrupt from affecting the interrupt output. All the bits, except for the modem status interrupt bits, are cleared to 0 when reset. The modem status interrupt bits are undefined after reset.

HW_UARTAPP_INTR 0x050

HW_UARTAPP_INTR_SET	0x054
HW_UARTAPP_INTR_CLR	0x058
HW_UARTAPP_INTR_TOG	0x05C

Table 26-12. HW_UARTAPP_INTR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1					OEIEN	BEIEN	PEIEN	FEIEN	RTIEN	TXIEN	RXIEN	DSRMIEN	DCDMIEN	CTSMIEN	RIMIEN	RSVD2					OEIS	BEIS	PEIS	FEIS	RTIS	TXIS	RXIS	DSRMIS	DCDMIS	CTSMIS	RIMIS

Table 26-13. HW_UARTAPP_INTR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:27	**RSVD1**	RO	0x0	Reserved, read as zero, do not modify.
26	**OEIEN**	RW	0x0	Overrun Error Interrupt Enable.
25	**BEIEN**	RW	0x0	Break Error Interrupt Enable.
24	**PEIEN**	RW	0x0	Parity Error Interrupt Enable.
23	**FEIEN**	RW	0x0	Framing Error Interrupt Enable.
22	**RTIEN**	RW	0x0	Receive Timeout Interrupt Enable.
21	**TXIEN**	RW	0x0	Transmit Interrupt Enable.
20	**RXIEN**	RW	0x0	Receive Interrupt Enable.
19	**DSRMIEN**	RW	0x0	nUARTDSR Modem Interrupt Enable. This bit is not supported.
18	**DCDMIEN**	RW	0x0	nUARTDCD Modem Interrupt Enable. This bit is not supported.
17	**CTSMIEN**	RW	0x0	nUARTCTS Modem Interrupt Enable.
16	**RIMIEN**	RW	0x0	nUARTRI Modem Interrupt Enable. This bit is not supported.
15:11	**RSVD2**	RO	0x0	Reserved, read as zero, do not modify.
10	**OEIS**	RW	0x0	Overrun Error Interrupt Status. To clear this bit, write the bit-clear address with the particular bit set to 1.
9	**BEIS**	RW	0x0	Break Error Interrupt Status. To clear this bit, write the bit-clear address with the particular bit set to 1.
8	**PEIS**	RW	0x0	Parity Error Interrupt Status. To clear this bit, write the bit-clear address with the particular bit set to 1.
7	**FEIS**	RW	0x0	Framing Error Interrupt Status. To clear this bit, write the bit-clear address with the particular bit set to 1.
6	**RTIS**	RW	0x0	Receive Timeout Interrupt Status. To clear this bit, write the bit-clear address with the particular bit set to 1.
5	**TXIS**	RW	0x0	Transmit Interrupt Status. To clear this bit, write the bit-clear address with the particular bit set to 1.
4	**RXIS**	RW	0x0	Receive Interrupt Status. To clear this bit, write the bit-clear address with the particular bit set to 1.
3	**DSRMIS**	RW	0x0	nUARTDSR Modem Interrupt Status. This bit is not supported.
2	**DCDMIS**	RW	0x0	nUARTDCD Modem Interrupt Status. This bit is not supported.

Table 26-13. HW_UARTAPP_INTR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
1	**CTSMIS**	RW	0x0	nUARTCTS Modem Interrupt Status. To clear this bit, write the bit-clear address with the particular bit set to 1.
0	**RIMIS**	RW	0x0	nUARTRI Modem Interrupt Status. This bit is not supported.

DESCRIPTION:

The UART Interrupt Register contains the interrupt enables and the interrupt status. The interrupt status bits report the unmasked state of the interrupts. To clear a particular interrupt status bit, write the bit-clear address with the particular bit set to 1. The enable bits control the UART interrupt output: a 1 will enable a particular interrupt to assert the UART interrupt output, while a 0 will disable the particular interrupt from affecting the interrupt output. All the bits, except for the modem status interrupt bits, are cleared to 0 when reset. The modem status interrupt bits are undefined after reset.

EXAMPLE:

```
No Example.
```

26.4.7 UART Data Register Description

The UART Data Register is the receive and transmit data register. Receive (read) and transmit (write) up to four data characters per APB cycle.

HW_UARTAPP_DATA 0x060

Table 26-14. HW_UARTAPP_DATA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DATA																															

Table 26-15. HW_UARTAPP_DATA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**DATA**	RW	0x0	In DMA mode, up to 4 Received/Transmit characters can be accessed at a time. In PIO mode, only one character can be accessed at a time. The status register contains the receive data flags and valid bits.

DESCRIPTION:

For words to be transmitted: 1) If the FIFOs are enabled, data written to this location is pushed onto the transmit FIFO; 2) If the FIFOs are not enabled, data is stored in the transmitter holding register (the bottom word of the transmit FIFO). The write operation initiates transmission from the PrimeCell UART. The data is prefixed with a start bit, appended with the appropriate parity bit (if parity is enabled), and a stop bit. The resultant word is then transmitted. Note: With the use of APB byte-enables you can write 1, 2, or 4 valid bytes sumultaneously to the TXFIFO. The invalid bytes will also take up space in the TXFIFO. So every write cycle will consume 4 bytes in the TXFIFO. If TXFIFO is disabled, you must only write the LSByte of the DATA register.

For received words: 1) If the FIFOs are enabled, the data byte and the 4-bit status (break, frame, parity, and overrun) is pushed onto the 12-bit wide receive FIFO; 2) if the FIFOs are not enabled, the data byte and status are stored in the receiving holding register (the bottom word of the receive FIFO). The received data bytes (up to 4) are read by performing reads from the 32-bit DATA register. The status information can be read by a read of the UART Status register.

The Overrun Error bit is set to 1 if data is received and the receive FIFO is already full. This is cleared to 0 once there is an empty space in the FIFO and a new character can be written to it. The Break Error bit is set to 1 if a break condition was detected, indicating that the received data input was held LOW for longer than a full-word transmission time (defined as start, data, parity and stop bits). In FIFO mode, this error is associated with the character at the top of the FIFO. When a break occurs, only one 0 character is loaded into the FIFO. The next character is only enabled after the receive data input goes to a 1 (marking state), and the next valid start bit is received. When the Parity Error bit is set to 1, it indicates that the parity of the received data character does not match the parity selected as defined by bits 2 and 7 of the LCR_H register. In FIFO mode, this error is associated with the character at the top of the FIFO. When the Framing Error bit is set to 1, it indicates that the received character did not have a valid stop bit (a valid stop bit is 1). In FIFO mode, this error is associated with the character at the top of the FIFO.

EXAMPLE:

```
No Example.
```

26.4.8 UART Status Register Description

The UART Status Register contains the various flags and receive status. If the status is read from this register, then the status information for break, framing and parity corresponds to the data character read from the UART Data Register prior to reading the UART Status Register. The status information for overrun is set immediately when an overrun condition occurs.

HW_UARTAPP_STAT 0x070

Table 26-16. HW_UARTAPP_STAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PRESENT	HISPEED	BUSY	CTS	TXFE	RXFF	TXFF	RXFE	RXBYTE_INVALID				OERR	BERR	PERR	FERR	RXCOUNT															

Table 26-17. HW_UARTAPP_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**PRESENT**	RO	0x1	This read-only bit indicates that the Application UART function is present when it reads back a one. This Application UART function is not available on a device that returns a zero for this bit field. UNAVAILABLE = 0x0 UARTAPP is not present in this product. AVAILABLE = 0x1 UARTAPP is present in this product.
30	**HISPEED**	RO	0x1	This read-only bit indicates that the high-speed function is present when it reads back a one. This high speed function is not available on a device that returns a zero for this bit field. UNAVAILABLE = 0x0 HISPEED is not present in this product. AVAILABLE = 0x1 HISPEED is present in this product.
29	**BUSY**	RO	0x0	UART Busy.
28	**CTS**	RO	0x0	Clear To Send.
27	**TXFE**	RO	0x1	Transmit FIFO Empty. The meaning of this bit depends on the state of the FEN bit in the UART Line Control Register. If the FIFO is disabled, this bit is set when the transmit holding register is empty. If the FIFO is enabled, the TXFE bit is set when the transmit FIFO is empty.
26	**RXFF**	RO	0x0	Receive FIFO Full.
25	**TXFF**	RO	0x0	Transmit FIFO Full.
24	**RXFE**	RO	0x1	Receive FIFO Empty.
23:20	**RXBYTE_INVALID**	RW	0xf	The invalid state of the last read of Receive Data. Each bit corresponds to one byte of the RX data. (1 = invalid.)
19	**OERR**	RO	0x0	Overrun Error. This bit is set to 1 if data is received and the FIFO is already full. This bit is cleared to 0 by any write to the Status Register. The FIFO contents remain valid since no further data is written when the FIFO is full; only the contents of the shift register are overwritten. The CPU must now read the data in order to empty the FIFO.
18	**BERR**	RW	0x0	Break Error. For PIO mode, this is for the last character read from the data register. For DMA mode, it will be set to 1 if any received character for a particular RXDMA command had a Break Error. To clear this bit, write a zero to it. Note that clearing this bit does not affect the interrupt status, which must be cleared by writing the interrupt register.
17	**PERR**	RW	0x0	Parity Error. For PIO mode, this is for the last character read from the data register. For DMA mode, it will be set to 1 if any received character for a particular RXDMA command had a Parity Error. To clear this bit, write a zero to it. Note that clearing this bit does not affect the interrupt status, which must be cleared by writing the interrupt register.

Table 26-17. HW_UARTAPP_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
16	**FERR**	RW	0x0	Framing Error. For PIO mode, this is for the last character read from the data register. For DMA mode, it will be set to 1 if any received character for a particular RXDMA command had a Framing Error. To clear this bit, write a zero to it. Note that clearing this bit does not affect the interrupt status, which must be cleared by writing the interrupt register.
15:0	**RXCOUNT**	RO	0x0	Number of bytes received during a Receive DMA command.

DESCRIPTION:

Various UART status information is available in this register.

EXAMPLE:

```
No Example.
```

26.4.9 UART Debug Register Description

The UART Debug Register contains the state of the DMA signals.

HW_UARTAPP_DEBUG 0x080

Table 26-18. HW_UARTAPP_DEBUG

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RXIBAUD_DIV																RXFBAUD_DIV						RSVD1				TXDMARUN	RXDMARUN	TXCMDEND	RXCMDEND	TXDMARQ	RXDMARQ

Table 26-19. HW_UARTAPP_DEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RXIBAUD_DIV**	RO	0x0	RX Integer Baud Divisor.
15:10	**RXFBAUD_DIV**	RO	0x0	RX Fractional Baud Divisor.
9:6	**RSVD1**	RO	0x0	Reserved, read as zero, do not modify.
5	**TXDMARUN**	RO	0x0	DMA Command Run Status: This bit reflects the state of the toggle signal for TXDMARUN.
4	**RXDMARUN**	RO	0x0	DMA Command Run Status: This bit reflects the state of the toggle signal for RXDMARUN.
3	**TXCMDEND**	RO	0x0	DMA Command End Status: This bit reflects the state of the toggle signal for UART_TXCMDEND.
2	**RXCMDEND**	RO	0x0	DMA Command End Status: This bit reflects the state of the toggle signal for UART_RXCMDEND.

Table 26-19. HW_UARTAPP_DEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
1	**TXDMARQ**	RO	0x0	DMA Request Status: This bit reflects the state of the toggle signal for UART_TXDMAREQ. Note that TX burst request is not supported.
0	**RXDMARQ**	RO	0x0	DMA Request Status: This bit reflects the state of the toggle signal for UART_RXDMAREQ. Note that RX burst request is not supported.

DESCRIPTION:

The UART Debug Register contains the state of the DMA signals.

EXAMPLE:

```
No Example.
```

26.4.10 UART Version Register Description

The UART version register can be used to read the version of the UARTAPP IP being used in this chip.

HW_UARTAPP_VERSION 0x090

Table 26-20. HW_UARTAPP_VERSION

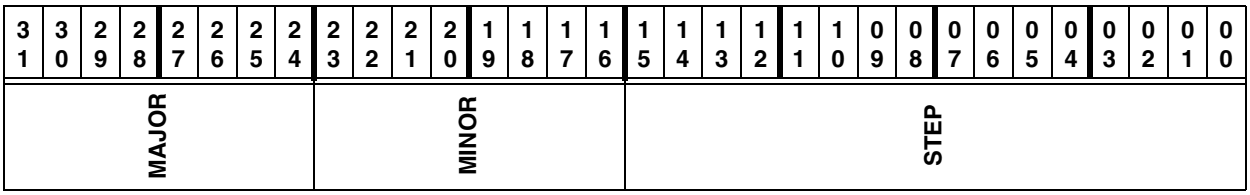


Table 26-21. HW_UARTAPP_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x3	Fixed read-only value reflecting the MAJOR field of RTL version.
23:16	**MINOR**	RO	0x0	Fixed read-only value reflecting the MINOR field of RTL version.
15:0	**STEP**	RO	0x0	Fixed read-only value reflecting the stepping of RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
No Example.
```

26.4.11 UART AutoBaud Register Description

The UART AutoBaud register provides the reference characters and control info for the automatic baudrate detection logic.

HW_UARTAPP_AUTOBAUD 0x0A0

Table 26-22. HW_UARTAPP_AUTOBAUD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REFCHAR1								REFCHAR0								RSVD1											UPDATE_TX	TWO_REF_CHARS	START_WITH_RUNBIT	START_BAUD_DETECT	BAUD_DETECT_ENABLE

Table 26-23. HW_UARTAPP_AUTOBAUD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**REFCHAR1**	RW	0x0	Second reference character used in baud rate detection. During autobaud detection of the second reference character received is available for reading, and is placed in the RXFIFO if TWO_REF_CHARS is a 1. This is ignored when TWO_REF_CHARS is 0.
23:16	**REFCHAR0**	RW	0x0	First reference character used in baud rate detection. During autobaud detection of the first character received is not available for reading. The UART receiver does not get this character and the RXFIFO is not updated.
15:5	**RSVD1**	RO	0x0	Reserved, read as zero, do not modify.
4	**UPDATE_TX**	RW	0x0	Set this bit to 1 will cause the TX baud rate divisor to be updated when the RX baud rate divisor is updated by the autobaud detection logic. This should not be set if it is possible that transmit function may be busy.
3	**TWO_REF_CHARS**	RW	0x0	Set this bit to 1 when using 2 reference characters for baud detection, and set to 0 for 1 reference character.
2	**START_WITH_RUNBIT**	RW	0x0	Set this bit to 1 will cause the assertion of HW_UARTAPP_CTRL0_RUN to start the autobaud detection logic. Set this bit to 0 will cause the assertion of START_BAUD_DETECT to start the autobaud detection logic.
1	**START_BAUD_DETECT**	RW	0x0	Set to 1 to start automatic baudrate detection. This bit is ignored when START_WITH_RUNBIT is set to 1. Each time a 1 is written to START_BAUD_DETECT it toggles the read value if START_WITH_RUNBIT is zero.
0	**BAUD_DETECT_ENABLE**	RW	0x0	Enable automatic baudrate detection.

DESCRIPTION:

This automatic baudrate detection logic used one or two reference character to dectect the baud rate of the received data.

EXAMPLE:

```
No Example.
```

UARTAPP Block v3.0, Revision 1.42

Chapter 27
Debug UART

This chapter describes the debug UART included on the i.MX23, how to operate it, and how to disable the FIFOs. Programmable registers are described in Section 27.3, "Programmable Registers."

27.1 Overview

The debug UART performs:

- Serial-to-parallel conversion on data received from a peripheral device
- Parallel-to-serial conversion on data transmitted to the peripheral device

The CPU reads and writes data and control/status information through the APBX interface. The transmit and receive paths are buffered with internal FIFO memories, enabling up to 16 bytes to be stored independently in both transmit and receive modes.

The debug UART includes a programmable baud rate generator that creates a transmit and receive internal clock from the 24-MHz UART internal reference clock input UARTCLK. XCLK is not tied to the UARTCLK in the i.MX23.

It offers similar functionality to the industry-standard 16C550 UART device and supports baud rates of up to 115 Kb/s. Figure 27-2 shows a block diagram of the debug UART. The debug UART operation and baud rate values are controlled by the line control register (HW_UARTDBGLCR_H, HW_UARTDBGIBRD, and HW_UARTDBGFBRD).

The debug UART can generate a single combined interrupt, so output is asserted if any individual interrupt is asserted and unmasked. Interrupt sources include the receive (including time-out), transmit, modem status, and error conditions.

If a framing, parity, or break error occurs during reception, the appropriate error bit is set, and is stored in the FIFO. If an overrun condition occurs, the overrun register bit is set immediately, and FIFO data is prevented from being overwritten. You can program the FIFOs to be one-byte deep, providing a conventional double-buffered UART interface.

Note: Names for the external pins used by the debug UART begin with "UART1". Pin names beginning with "UART2" are used by the Application UART.

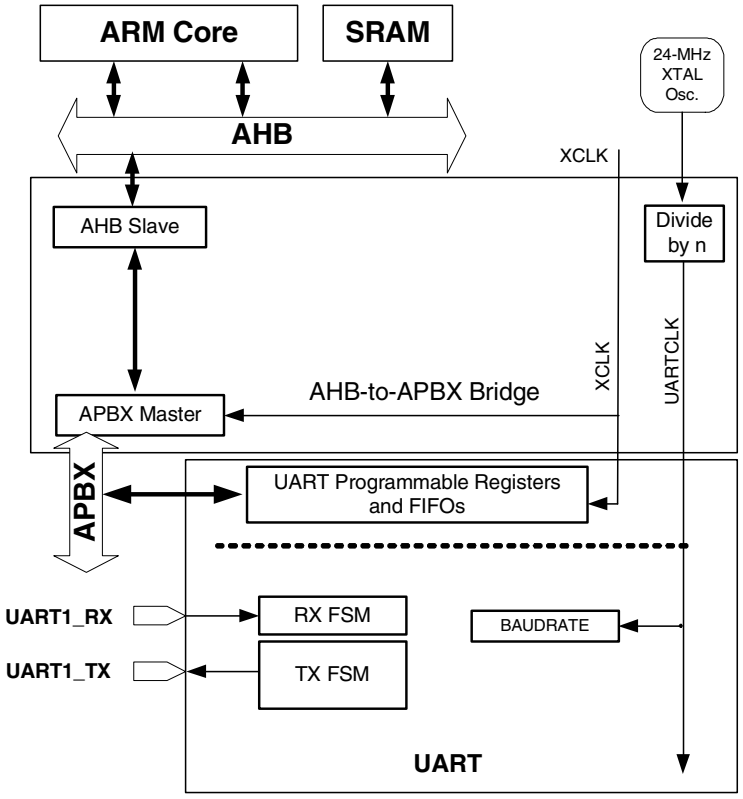


Figure 27-1. Debug UART Block Diagram

27.2 Operation

Control data is written to the debug UART line control register. This register defines:

- Transmission parameters
- Word length
- Buffer mode
- Number of transmitted stop bits
- Parity mode
- Break generation
- Baud rate divisor

27.2.1 Fractional Baud Rate Divider

The baud rate divisor is calculated from the frequency of UARTCLK and the desired baud rate by using the following formula:

```
divisor = (UARTCLK * 4) / baud rate, rounded to the nearest integer
```

The divisor must be between 0x00000040 and 0x003FFFC0, inclusive. Program the lowest 6 bits of the divisor into BAUD_DIVFRAC, and the next 16 bits of the divisor into BAUD_DIVINT.

In the debug UART, HW_UARTDBGLCR_H, HW_UARTDBGIBRD and HW_UARTDBGFBRD form a single 30-bit wide register (UARTLCR) that is updated on a single write strobe generated by an HW_UARTDBGLCR_H write. So, in order to internally update the contents of HW_UARTDBGIBRD or HW_UARTDBGFBRD, a write to HW_UARTDBGLCR_H must always be performed at the end.

27.2.2 UART Character Frame

Figure 27-2 illustrates the UART character frame.

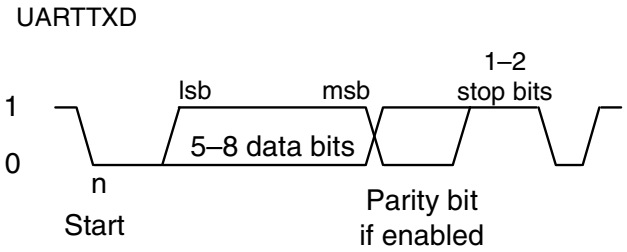


Figure 27-2. Debug UART Character Frame

27.2.3 Data Transmission or Reception

Data received or transmitted is stored in two 16-byte FIFOs, though the receive FIFO has an extra four bits per character for status information.

For transmission, data is written into the transmit FIFO. If the debug UART is enabled, it causes a data frame to start transmitting with the parameters indicated in UARTLCR_H. Data continues to be transmitted until there is no data left in the transmit FIFO.

The BUSY signal goes HIGH as soon as data is written to the transmit FIFO (that is, the FIFO is non-empty) and remains asserted HIGH while data is being transmitted. BUSY is negated only when the transmit FIFO is empty and the last character has been transmitted from the shift register, including the stop bits. BUSY can be asserted HIGH even though the debug UART might no longer be enabled.

For each sample of data, three readings are taken and the majority value is kept. In the following paragraphs, the middle sampling point is defined and one sample is taken either side of it.

- When the receiver is idle (UARTRXD continuously 1, in the marking state) and a LOW is detected on the data input (a start bit has been received), the receive counter, with the clock enabled by Baud16, begins running and data is sampled on the eighth cycle of that counter in normal UART mode to allow for the shorter logic 0 pulses (half way through a bit period).
- The start bit is valid if UARTRXD is still LOW on the eighth cycle of Baud16, otherwise a false start bit is detected and it is ignored. If the start bit was valid, successive data bits are sampled on every 16th cycle of Baud16 (that is, one bit period later) according to the programmed length of the data characters. The parity bit is then checked, if parity mode was enabled.

- Lastly, a valid stop bit is confirmed if UARTRXD is HIGH, otherwise a framing error has occurred. When a full word is received, the data is stored in the receive FIFO, with any error bits associated with that word (see Table 27-1).

27.2.4 Error Bits

Three error bits are stored in bits [10:8] of the receive FIFO and are associated with a particular character. An additional error indicating an overrun error is stored in bit 11 of the receive FIFO.

27.2.5 Overrun Bit

The overrun bit is not associated with the character in the receive FIFO. The overrun error is set when the FIFO is full, and the next character is completely received in the shift register. The data in the shift register is overwritten, but it is not written into the FIFO. When an empty location is available in the receive FIFO, and another character is received, the state of the overrun bit is copied into the receive FIFO along with the received character. The overrun state is then cleared. Table 27-1 shows the bit functions of the receive FIFO.

Table 27-1. Receive FIFO Bit Functions

FIFO BIT	FUNCTION
11	Overrun indicator
10	Break error
9	Parity error
8	Framing error
7:0	Received data

27.2.6 Disabling the FIFOs

FIFOs can be disabled. In this case, the transmit and receive sides of the UART have one-byte holding registers (the bottom entry of the FIFOs). The overrun bit is set when a word has been received and the previous one was not yet read.

In this implementation, the FIFOs are not physically disabled, but the flags are manipulated to give the illusion of a one-byte register.

27.3 Programmable Registers

This section describes the debug UART's programable registers.

27.3.1 UART Data Register Description

Debug Uart Data Register. For words to be transmitted: 1) If the FIFOs are enabled, data written to this location is pushed onto the transmit FIFO 2) If the FIFOs are not enabled, data is stored in the transmitter holding register (the bottom word of the transmit FIFO). The write operation initiates transmission from the PrimeCell UART. The data is prefixed with a start bit, appended with the appropriate parity bit (if parity is enabled), and a stop bit. The resultant word is then transmitted. For received words: 1) If the FIFOs are enabled, the data byte and the 4-bit status (break, frame, parity, and overrun) is pushed onto the 12-bit wide receive FIFO 2) If the FIFOs are not enabled, the data byte and status are stored in the receiving holding register (the bottom word of the receive FIFO). The received data byte is read by performing reads from the DR register along with the corresponding status information. The status information can also be read by a read of the RSR_ECR register.

HW_UARTDBGDR 0x000

Table 27-2. HW_UARTDBGDR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
UNAVAILABLE																RESERVED				OE	BE	PE	FE	DATA							

Table 27-3. HW_UARTDBGDR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**UNAVAILABLE**	RO	0x0	The UART IP only implements 16 and 8-bit registers, so the top 2 or 3 bytes of every 32-bit register are always unavailable.
15:12	**RESERVED**	RO	0x0	Reserved.
11	**OE**	RO	0x0	Overrun Error. This bit is set to 1 if data is received and the receive FIFO is already full. This is cleared to 0 once there is an empty space in the FIFO and a new character can be written to it.
10	**BE**	RO	0x0	Break Error. This bit is set to 1 if a break condition was detected, indicating that the received data input was held LOW for longer than a full-word transmission time (defined as start, data, parity and stop bits). In FIFO mode, this error is associated with the character at the top of the FIFO. When a break occurs, only one 0 character is loaded into the FIFO. The next character is only enabled after the receive data input goes to a 1 (marking state), and the next valid start bit is received.
9	**PE**	RO	0x0	Parity Error. When this bit is set to 1, it indicates that the parity of the received data character does not match the parity selected as defined by bits 2 and 7 of the LCR_H register. In FIFO mode, this error is associated with the character at the top of the FIFO.

Table 27-3. HW_UARTDBGDR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
8	**FE**	RO	0x0	Framing Error. When this bit is set to 1, it indicates that the received character did not have a valid stop bit (a valid stop bit is 1). In FIFO mode, this error is associated with the character at the top of the FIFO.
7:0	**DATA**	RW	0x0	Receive (read) data character. Transmit (write) data character.

DESCRIPTION:

Debug Uart Data Register. For words to be transmitted: 1) If the FIFOs are enabled, data written to this location is pushed onto the transmit FIFO 2) If the FIFOs are not enabled, data is stored in the transmitter holding register (the bottom word of the transmit FIFO). The write operation initiates transmission from the PrimeCell UART. The data is prefixed with a start bit, appended with the appropriate parity bit (if parity is enabled), and a stop bit. The resultant word is then transmitted. For received words: 1) If the FIFOs are enabled, the data byte and the 4-bit status (break, frame, parity, and overrun) is pushed onto the 12-bit wide receive FIFO 2) If the FIFOs are not enabled, the data byte and status are stored in the receiving holding register (the bottom word of the receive FIFO). The received data byte is read by performing reads from the DR register along with the corresponding status information. The status information can also be read by a read of the RSR_ECR register.

EXAMPLE:

```
No Example.
```

27.3.2 UART Receive Status Register (Read) / Error Clear Register (Write) Description

The RSR_ECR register is the receive status register/error clear register. Receive status can also be read from RSR_ECR. If the status is read from this register, then the status information for break, framing and parity corresponds to the data character read from DR prior to reading RSR_ECR. The status information for overrun is set immediately when an overrun condition occurs. A write to RSR_ECR clears the framing, parity, break, and overrun errors.

HW_UARTDBGRSR_ECR 0x004

Table 27-4. HW_UARTDBGRSR_ECR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
UNAVAILABLE																								EC				OE	BE	PE	FE

Table 27-5. HW_UARTDBGRSR_ECR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:8	**UNAVAILABLE**	RO	0x0	The UART IP only implements 16 and 8-bit registers, so the top 2 or 3 bytes of every 32-bit register are always unavailable.
7:4	**EC**	RW	0x0	Error Clear. Any write to this bitfield clears the framing, parity, break, and overrun errors. The value is unpredictable when read.
3	**OE**	RW	0x0	Overrun Error. This bit is set to 1 if data is received and the FIFO is already full. This bit is cleared to 0 by any write to RSR_ECR. The FIFO contents remain valid since no further data is written when the FIFO is full, only the contents of the shift register are overwritten. The CPU must now read the data in order to empty the FIFO.
2	**BE**	RW	0x0	Break Error.
1	**PE**	RW	0x0	Parity Error.
0	**FE**	RW	0x0	Framing Error.

DESCRIPTION:

The RSR_ECR register is the receive status register/error clear register. Receive status can also be read from RSR_ECR. If the status is read from this register, then the status information for break, framing and parity corresponds to the data character read from DR prior to reading RSR_ECR. The status information for overrun is set immediately when an overrun condition occurs. A write to RSR_ECR clears the framing, parity, break, and overrun errors.

EXAMPLE:

```
No Example.
```

27.3.3 UART Flag Register Description

The FR register is the flag register.

HW_UARTDBGFR 0x018

Table 27-6. HW_UARTDBGFR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
UNAVAILABLE																RESERVED							RI	TXFE	RXFF	TXFF	RXFE	BUSY	DCD	DSR	CTS

Table 27-7. HW_UARTDBGFR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**UNAVAILABLE**	RO	0x0	The UART IP only implements 16 and 8-bit registers, so the top 2 or 3 bytes of every 32-bit register are always unavailable.
15:9	**RESERVED**	RO	0x0	Reserved, do not modify, read as zero.
8	**RI**	RO	0x0	Ring Indicator. This bit is the complement of the UART ring indicator (nUARTRI) modem status input. That is, the bit is 1 when the modem status input is 0.
7	**TXFE**	RO	0x1	Transmit FIFO Empty. The meaning of this bit depends on the state of the FEN bit in the LCR_H register. If the FIFO is disabled, this bit is set when the transmit holding register is empty. If the FIFO is enabled, the TXFE bit is set when the transmit FIFO is empty.
6	**RXFF**	RO	0x0	Receive FIFO Full.
5	**TXFF**	RO	0x0	Transmit FIFO Full.
4	**RXFE**	RO	0x1	Receive FIFO Empty.
3	**BUSY**	RO	0x0	UART Busy.
2	**DCD**	RO	0x0	Data Carrier Detect.
1	**DSR**	RO	0x0	Data Set Ready.
0	**CTS**	RO	0x0	Clear To Send.

DESCRIPTION:

The UART IP only implements 16 and 8-bit registers, so the top 2 or 3 bytes of every 32-bit register are always unavailable.

EXAMPLE:

```
No Example.
```

27.3.4 UART IrDA Low-Power Counter Register Description

The ILPR register is the IrDA Low-Power Counter Register. This is an 8-bit read/write register which stores a low-power counter divisor value used to divde down the UARTCLK to generate the IrLPBaud16 signal.

HW_UARTDBGILPR 0x020

Table 27-8. HW_UARTDBGILPR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
UNAVAILABLE																								ILPDVSR							

Table 27-9. HW_UARTDBGILPR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:8	**UNAVAILABLE**	RO	0x0	The UART IP only implements 16 and 8-bit registers, so the top 2 or 3 bytes of every 32-bit register are always unavailable.
7:0	**ILPDVSR**	RW	0x0	IrDA Low Power Divisor [7:0]. 8-bit low-power divisor value.

DESCRIPTION:

The ILPR register is the IrDA Low-Power Counter Register. This is an 8-bit read/write register which stores a low-power counter divisor value used to divde down the UARTCLK to generate the IrLPBaud16 signal.

EXAMPLE:

```
No Example.
```

27.3.5 UART Integer Baud Rate Divisor Register Description

The IBRD register is the integer part of the baud rate divisor value.

HW_UARTDBGIBRD 0x024

Table 27-10. HW_UARTDBGIBRD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
UNAVAILABLE																BAUD_DIVINT															

Table 27-11. HW_UARTDBGIBRD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**UNAVAILABLE**	RO	0x0	The UART IP only implements 16 and 8-bit registers, so the top 2 or 3 bytes of every 32-bit register are always unavailable.
15:0	**BAUD_DIVINT**	RW	0x0	Baud Rate Integer [15:0]. The integer baud rate divisor.

DESCRIPTION:

The IBRD register is the integer part of the baud rate divisor value.

EXAMPLE:

```
No Example.
```

27.3.6 UART Fractional Baud Rate Divisor Register Description

The FBRD register is the fractional part of the baud rate divisor value.

HW_UARTDBGFBRD 0x028

Table 27-12. HW_UARTDBGFBRD

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
UNAVAILABLE																								RESERVED				BAUD_DIVFRAC			

Table 27-13. HW_UARTDBGFBRD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:8	**UNAVAILABLE**	RO	0x0	The UART IP only implements 16 and 8-bit registers, so the top 2 or 3 bytes of every 32-bit register are always unavailable.
7:6	**RESERVED**	RO	0x0	Not documented.
5:0	**BAUD_DIVFRAC**	RW	0x0	Baud Rate Fraction [5:0]. The fractional baud rate divisor.

DESCRIPTION:

The FBRD register is the fractional part of the baud rate divisor value.

EXAMPLE:

```
No Example.
```

27.3.7 UART Line Control Register, HIGH Byte Description

The LCR_H is the Line Control Register.

HW_UARTDBGLCR_H 0x02C

Table 27-14. HW_UARTDBGLCR_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
UNAVAILABLE																RESERVED								SPS	WLEN		FEN	STP2	EPS	PEN	BRK

Table 27-15. HW_UARTDBGLCR_H Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**UNAVAILABLE**	RO	0x0	The UART IP only implements 16 and 8-bit registers, so the top 2 or 3 bytes of every 32-bit register are always unavailable.
15:8	**RESERVED**	RO	0x0	Reserved, do not modify, read as zero.
7	**SPS**	RW	0x0	Stick Parity Select. When bits 1, 2, and 7 of the LCR_H register are set, the parity bit is transmitted and checked as a 0. When bits 1 and 7 are set, and bit 2 is 0, the parity bit is transmitted and checked as a 1. When this bit is cleared stick parity is disabled.
6:5	**WLEN**	RW	0x0	Word length [1:0]. The select bits indicate the number of data bits transmitted or received in a frame as follows: 11 = 8 bits, 10 = 7 bits, 01 = 6 bits, 00 = 5 bits.
4	**FEN**	RW	0x0	Enable FIFOs. If this bit is set to 1, transmit and receive FIFO buffers are enabled (FIFO mode). When cleared to 0, the FIFOs are disabled (character mode); that is, the FIFOs become 1-byte-deep holding registers.
3	**STP2**	RW	0x0	Two Stop Bits Select. If this bit is set to 1, two stop bits are transmitted at the end of the frame. The receive logic does not check for two stop bits being received.
2	**EPS**	RW	0x0	Even Parity Select. If this bit is set to 1, even parity generation and checking is performed during transmission and reception, which checks for an even number of 1s in data and parity bits. When cleared to 0, then odd parity is performed which checks for an odd number of 1s. This bit has no effect when parity is disabled by Parity Enable (PEN, bit 1) being cleared to 0.
1	**PEN**	RW	0x0	Parity Enable. If this bit is set to 1, parity checking and generation is enabled, else parity is disabled and no parity bit added to the data frame.
0	**BRK**	RW	0x0	Send Break. If this bit is set to 1, a low-level is continually output on the UARTTXD output, after completing transmission of the current character. For the proper execution of the break command, the software must set this bit for at least two complete frames. For normal use, this bit must be cleared to 0.

DESCRIPTION:

The LCR_H is the Line Control Register.

EXAMPLE:

```
No Example.
```

27.3.8 UART Control Register Description

The CR is the Control Register.

HW_UARTDBGCR 0x030

Table 27-16. HW_UARTDBGCR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
UNAVAILABLE																CTSEN	RTSEN	OUT2	OUT1	RTS	DTR	RXE	TXE	LBE	RESERVED				SIRLP	SIREN	UARTEN

Table 27-17. HW_UARTDBGCR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**UNAVAILABLE**	RO	0x0	The UART IP only implements 16 and 8-bit registers, so the top 2 or 3 bytes of every 32-bit register are always unavailable.
15	**CTSEN**	RW	0x0	CTS Hardware Flow Control Enable.
14	**RTSEN**	RW	0x0	RTS Hardware Flow Control Enable.
13	**OUT2**	RW	0x0	This bit is the complement of the UART Out2 (nUARTOut2) modem status output. Not Implemented.
12	**OUT1**	RW	0x0	This bit is the complement of the UART Out1 (nUARTOut1) modem status output. Not Implemented.
11	**RTS**	RW	0x0	Request To Send.
10	**DTR**	RW	0x0	Data Transmit Ready. Not Implemented.
9	**RXE**	RW	0x1	Receive Enable. If this bit is set to 1, the receive section of the UART is enabled. Data reception occurs for the UART signals. When the UART is disabled in the middle of reception, it completes the current character before stopping.
8	**TXE**	RW	0x1	Transmit Enable. If this bit is set to 1, the transmit section of the UART is enabled. Data transmission occurs for the UART signals. When the UART is disabled in the middle of transmission, it completes the current character before stopping.
7	**LBE**	RW	0x0	Loop Back Enable. This feature reduces the amount of external coupling required during system test. If this bit is set to 1, the UARTTXD path is fed through to the UARTRXD path. When this bit is set, the modem outputs are also fed through to the modem inputs.
6:3	**RESERVED**	RO	0x0	Reserved, do not modify, read as zero.
2	**SIRLP**	RW	0x0	IrDA SIR low-power mode. Not Supported.
1	**SIREN**	RW	0x0	SIR Enable. Not Supported.
0	**UARTEN**	RW	0x0	UART Enable. If this bit is set to 1, the UART is enabled. Data transmission and reception occurs for the UART signals. When the UART is disabled in the middle of transmission or reception, it completes the current character before stopping.

DESCRIPTION:

The UART IP only implements 16 and 8-bit registers, so the top 2 or 3 bytes of every 32-bit register are always unavailable.

EXAMPLE:

```
No Example.
```

27.3.9 UART Interrupt FIFO Level Select Register Description

The IFLS register is the Interrupt FIFO Level Select Register. You can use the IFLS register to define the FIFO level at which the UARTTXINTR and UARTRXINTR are triggered. The interrupts are generated based on a transition through a level rather than being based on the level. That is, the design is such that the interrupts are generated when the fill level progresses through the trigger level. The bits are reset so that the trigger level is when the FIFOs are at the half-way mark.

HW_UARTDBGIFLS 0x034

Table 27-18. HW_UARTDBGIFLS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
UNAVAILABLE																RESERVED										RXIFLSEL			TXIFLSEL		

Table 27-19. HW_UARTDBGIFLS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**UNAVAILABLE**	RO	0x0	The UART IP only implements 16 and 8-bit registers, so the top 2 or 3 bytes of every 32-bit register are always unavailable.
15:6	**RESERVED**	RO	0x0	Reserved, do not modify, read as zero.
5:3	**RXIFLSEL**	RW	0x2	Receive Interrupt FIFO Level Select. The trigger points for the receive interrupt are as follows: NOT_EMPTY = 0x0 Trigger on FIFO not empty, i.e. at least 1 of 8 entries. ONE_QUARTER = 0x1 Trigger on FIFO full to at least 2 of 8 entries. ONE_HALF = 0x2 Trigger on FIFO full to at least 4 of 8 entries. THREE_QUARTERS = 0x3 Trigger on FIFO full to at least 6 of 8 entries. SEVEN_EIGHTHS = 0x4 Trigger on FIFO full to at least 7 of 8 entries. INVALID5 = 0x5 Reserved. INVALID6 = 0x6 Reserved. INVALID7 = 0x7 Reserved.
2:0	**TXIFLSEL**	RW	0x2	Transmit Interrupt FIFO Level Select. The trigger points for the transmit interrupt are as follows: EMPTY = 0x0 Trigger on FIFO empty, i.e. no entries. ONE_QUARTER = 0x1 Trigger on FIFO less than 2 of 8 entries. ONE_HALF = 0x2 Trigger on FIFO less than 4 of 8 entries. THREE_QUARTERS = 0x3 Trigger on FIFO less than 6 of 8 entries. SEVEN_EIGHTHS = 0x4 Trigger on FIFO less than 7 of 8 entries. INVALID5 = 0x5 Reserved. INVALID6 = 0x6 Reserved. INVALID7 = 0x7 Reserved.

DESCRIPTION:

The IFLS register is the Interrupt FIFO Level Select Register. You can use the IFLS register to define the FIFO level at which the UARTTXINTR and UARTRXINTR are triggered. The interrupts are generated

based on a transition through a level rather than being based on the level. That is, the design is such that the interrupts are generated when the fill level progresses through the trigger level. The bits are reset so that the trigger level is when the FIFOs are at the half-way mark.

EXAMPLE:

```
No Example.
```

27.3.10 UART Interrupt Mask Set/Clear Register Description

The IMSC register is the Interrupt Mask Set/Clear Register. On a read, this register gives the current value of the mask on the relevant interrupt. On a write of 1 to the particular bit, it sets the corresponding mask of that interrupt. A write of 0 clears the corresponding mask.

HW_UARTDBGIMSC 0x038

Table 27-20. HW_UARTDBGIMSC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
UNAVAILABLE																RESERVED					OEIM	BEIM	PEIM	FEIM	RTIM	TXIM	RXIM	DSRMIM	DCDMIM	CTSMIM	RIMIM

Table 27-21. HW_UARTDBGIMSC Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**UNAVAILABLE**	RO	0x0	The UART IP only implements 16 and 8-bit registers, so the top 2 or 3 bytes of every 32-bit register are always unavailable.
15:11	**RESERVED**	RO	0x0	Reserved, do not modify, read as zero.
10	**OEIM**	RW	0x0	Overrun Error Interrupt Mask. On a read, the current mask for the OEIM interrupt is returned. On a write of 1, the mask of the OEIM interrupt is set. A write of 0 clears the mask.
9	**BEIM**	RW	0x0	Break Error Interrupt Mask.
8	**PEIM**	RW	0x0	Parity Error Interrupt Mask.
7	**FEIM**	RW	0x0	Framing Error Interrupt Mask.
6	**RTIM**	RW	0x0	Receive Timeout Interrupt Mask.
5	**TXIM**	RW	0x0	Transmit Interrupt Mask.
4	**RXIM**	RW	0x0	Receive Interrupt Mask.
3	**DSRMIM**	RW	0x0	nUARTDSR Modem Interrupt Mask.
2	**DCDMIM**	RW	0x0	nUARTDCD Modem Interrupt Mask.
1	**CTSMIM**	RW	0x0	nUARTCTS Modem Interrupt Mask.
0	**RIMIM**	RW	0x0	nUARTRI Modem Interrupt Mask.

DESCRIPTION:

The IMSC register is the Interrupt Mask Set/Clear Register. On a read, this register gives the current value of the mask on the relevant interrupt. On a write of 1 to the particular bit, it sets the corresponding mask of that interrupt. A write of 0 clears the corresponding mask.

EXAMPLE:

```
No Example.
```

27.3.11 UART Raw Interrupt Status Register Description

The RIS register is the Raw Interrupt Status Register. It is a read-only register. On a read this register gives the current raw status value of the corresponding interrupt. A write has no effect. All the bits, except for the modem status interrupt bits (bits 3 to 0), are cleared to 0 when reset. The modem status interrupt bits are undefined after reset.

HW_UARTDBGRIS 0x03C

Table 27-22. HW_UARTDBGRIS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
UNAVAILABLE																RESERVED					OERIS	BERIS	PERIS	FERIS	RTRIS	TXRIS	RXRIS	DSRRMIS	DCDRMIS	CTSRMIS	RIRMIS

Table 27-23. HW_UARTDBGRIS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**UNAVAILABLE**	RO	0x0	The UART IP only implements 16 and 8-bit registers, so the top 2 or 3 bytes of every 32-bit register are always unavailable.
15:11	**RESERVED**	RO	0x0	Reserved, read as zero, do not modify.
10	**OERIS**	RO	0x0	Overrun Error Interrupt Status.
9	**BERIS**	RO	0x0	Break Error Interrupt Status.
8	**PERIS**	RO	0x0	Parity Error Interrupt Status.
7	**FERIS**	RO	0x0	Framing Error Interrupt Status.
6	**RTRIS**	RO	0x0	Receive Timeout Interrupt Status.
5	**TXRIS**	RO	0x0	Transmit Interrupt Status.
4	**RXRIS**	RO	0x0	Receive Interrupt Status.
3	**DSRRMIS**	RO	0x0	nUARTDSR Modem Interrupt Status.
2	**DCDRMIS**	RO	0x0	nUARTDCD Modem Interrupt Status.
1	**CTSRMIS**	RO	0x0	nUARTCTS Modem Interrupt Status.
0	**RIRMIS**	RO	0x0	nUARTRI Modem Interrupt Status.

DESCRIPTION:

The RIS register is the Raw Interrupt Status Register. It is a read-only register. On a read this register gives the current raw status value of the corresponding interrupt. A write has no effect. All the bits, except for

the modem status interrupt bits (bits 3 to 0), are cleared to 0 when reset. The modem status interrupt bits are undefined after reset.

EXAMPLE:

```
No Example.
```

27.3.12 UART Masked Interrupt Status Register Description

The MIS register is the Masked Interrupt Status Register. It is a read-only register. On a read this register gives the current masked status value of the corresponding interrupt. A write has no effect. All the bits except for the modem status interrupt bits (bits 3 to 0) are cleared to 0 when reset. The modem status interrupt bits are undefined after reset.

HW_UARTDBGMIS 0x040

Table 27-24. HW_UARTDBGMIS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
UNAVAILABLE																RESERVED					OEMIS	BEMIS	PEMIS	FEMIS	RTMIS	TXMIS	RXMIS	DSRMMIS	DCDMMIS	CTSMMIS	RIMMIS

Table 27-25. HW_UARTDBGMIS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**UNAVAILABLE**	RO	0x0	The UART IP only implements 16 and 8-bit registers, so the top 2 or 3 bytes of every 32-bit register are always unavailable.
15:11	**RESERVED**	RO	0x0	Reserved, read as zero, do not modify.
10	**OEMIS**	RO	0x0	Overrun Error Masked Interrupt Status.
9	**BEMIS**	RO	0x0	Break Error Masked Interrupt Status.
8	**PEMIS**	RO	0x0	Parity Error Masked Interrupt Status.
7	**FEMIS**	RO	0x0	Framing Error Masked Interrupt Status.
6	**RTMIS**	RO	0x0	Receive Timeout Masked Interrupt Status.
5	**TXMIS**	RO	0x0	Transmit Masked Interrupt Status.
4	**RXMIS**	RO	0x0	Receive Masked Interrupt Status.
3	**DSRMMIS**	RO	0x0	nUARTDSR Modem Masked Interrupt Status.
2	**DCDMMIS**	RO	0x0	nUARTDCD Modem Masked Interrupt Status.
1	**CTSMMIS**	RO	0x0	nUARTCTS Modem Masked Interrupt Status.
0	**RIMMIS**	RO	0x0	nUARTRI Modem Masked Interrupt Status.

DESCRIPTION:

The MIS register is the Masked Interrupt Status Register. It is a read-only register. On a read this register gives the current masked status value of the corresponding interrupt. A write has no effect. All the bits except for the modem status interrupt bits (bits 3 to 0) are cleared to 0 when reset. The modem status interrupt bits are undefined after reset.

EXAMPLE:

```
No Example.
```

27.3.13 UART Interrupt Clear Register Description

The ICR register is the Interrupt Clear Register and is write-only. On a write of 1, the corresponding interrupt is cleared. A write of 0 has no effect.

HW_UARTDBGICR 0x044

Table 27-26. HW_UARTDBGICR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
UNAVAILABLE																RESERVED					OEIC	BEIC	PEIC	FEIC	RTIC	TXIC	RXIC	DSRMIC	DCDMIC	CTSMIC	RIMIC

Table 27-27. HW_UARTDBGICR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**UNAVAILABLE**	RO	0x0	The UART IP only implements 16 and 8-bit registers, so the top 2 or 3 bytes of every 32-bit register are always unavailable.
15:11	**RESERVED**	RO	0x0	Reserved, read as zero, do not modify.
10	**OEIC**	WO	0x0	Overrun Error Interrupt Clear.
9	**BEIC**	WO	0x0	Break Error Interrupt Clear.
8	**PEIC**	WO	0x0	Parity Error Interrupt Clear.
7	**FEIC**	WO	0x0	Framing Error Interrupt Clear.
6	**RTIC**	WO	0x0	Receive Timeout Interrupt Clear.
5	**TXIC**	WO	0x0	Transmit Interrupt Clear.
4	**RXIC**	WO	0x0	Receive Interrupt Clear.
3	**DSRMIC**	WO	0x0	nUARTDSR Modem Interrupt Clear.
2	**DCDMIC**	WO	0x0	nUARTDCD Modem Interrupt Clear.
1	**CTSMIC**	WO	0x0	nUARTCTS Modem Interrupt Clear.
0	**RIMIC**	WO	0x0	nUARTRI Modem Interrupt Clear.

DESCRIPTION:

The ICR register is the Interrupt Clear Register and is write-only. On a write of 1, the corresponding interrupt is cleared. A write of 0 has no effect.

EXAMPLE:

```
No Example.
```

27.3.14 UART DMA Control Register Description

This register is reserved.

HW_UARTDBGDMACR 0x048

Table 27-28. HW_UARTDBGDMACR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
UNAVAILABLE																RESERVED													DMAONERR	TXDMAE	RXDMAE

Table 27-29. HW_UARTDBGDMACR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**UNAVAILABLE**	RO	0x0	The UART IP only implements 16 and 8-bit registers, so the top 2 or 3 bytes of every 32-bit register are always unavailable.
15:3	**RESERVED**	RO	0x0	Reserved.
2	**DMAONERR**	RW	0x0	Reserved.
1	**TXDMAE**	RW	0x0	Reserved.
0	**RXDMAE**	RW	0x0	Reserved.

DESCRIPTION:

This register is reserved.

EXAMPLE:

```
No Example.
```

UARTDBG Block v2.0, Revision 1.21

Chapter 28
AUDIOIN/ADC

This chapter describes the AUDIOIN/ADC module implemented on the i.MX23, including DMA, sample rate conversion, and internal operation. Programmable registers are described in Section 28.4, "Programmable Registers."

28.1 Overview

The i.MX23 features an audio record path that consists of a sigma-delta analog-to-digital converter (ADC), followed by the AUDIOIN digital multi-stage Finite Impulse Response (FIR) filter.

The microphone or line input is oversampled by the ADC, and the 1-bit digital stream is input to a cascaded-integrator comb filter, where the signal is parallelized, sent through a high-pass filter to remove DC offset, and the sample rate is converted to the AUDIOIN's internal rate. Next, the signal is filtered using a three-stage FIR filter. The resultant parallel PCM samples are then transferred to a buffer in memory using the APBX bridge DMA, where it can be read by system software.

The analog audio source can be selected from one of three possible inputs:

- Mono microphone input, with settings for 0dB, 20dB, 30dB and 40dB gain.
- Stereo line inputs, with 0dB to 22.5dB gain, in 1.5dB steps.
- Looped back from the stereo headphone amplifier

The AUDIOIN module implements the following functions:

- Serial to parallel bit-stream integrator/averager
- Sample rate converting (SRC) cascaded-integrator comb (CIC) filter
- High-pass filter (HPF)
- Three-stage downsampling FIR filter: 7-tap (8:4), 11-tap (4:2), 33-tap (2:1) supporting conversion from quarter, half, full, double, and quad sample rates that are multiples of the standard 32 kHz, 44.1 kHz, and 48 kHz rates
- 16- or 32-bit PCM sample widths
- APBX bridge DMA interface
- Independent control of each channel's volume (including mute)
- DAC-to-ADC internal loopback for product development
- Control bit fields used for analog ADC settings

Figure 28-1 shows a high-level block diagram of the AUDIOIN module. See Figure 1-4 for a diagram of the audio path and control options.

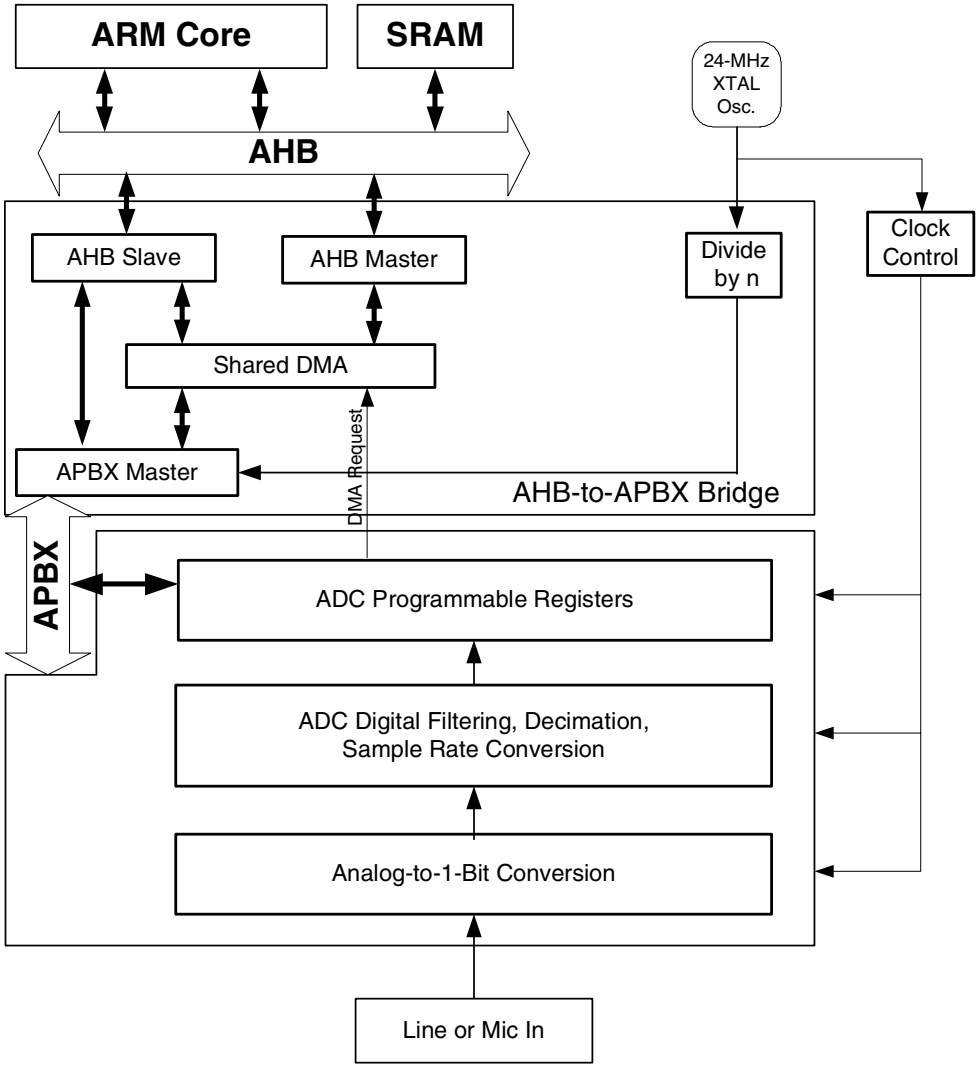


Figure 28-1. AUDIOIN/ADC Block Diagram

28.2 Operation

The first step in receiving audio to the AUDIOIN module requires the analog-to-digital converter (ADC). The i.MX23 includes a high-performance analog stereo sigma-delta ADC. It converts analog audio to two (left and right channel) single-bit digital streams that are input to the AUDIOIN module, along with a clock that runs at the sigma-delta oversampling clock rate. The AUDIOIN module includes hardware for oversampling, decimation, and arbitrary sample rate conversion. The 1-bit stream is input to a cascaded-integrator comb filter where serial-to-parallel data conversion, as well as sample rate conversion, takes place, along with a high-pass filter to eliminate DC offset. Serial audio is first input to an averager that initially converts samples to 8-bit values. The CIC then interpolates/decimates as well as sign-extends the parallel data, converting the samples from the programmed standard external sample

rate to the AUDIOIN module's internal rate. The resultant 24-bit PCM samples are then stored to the module's RAM.

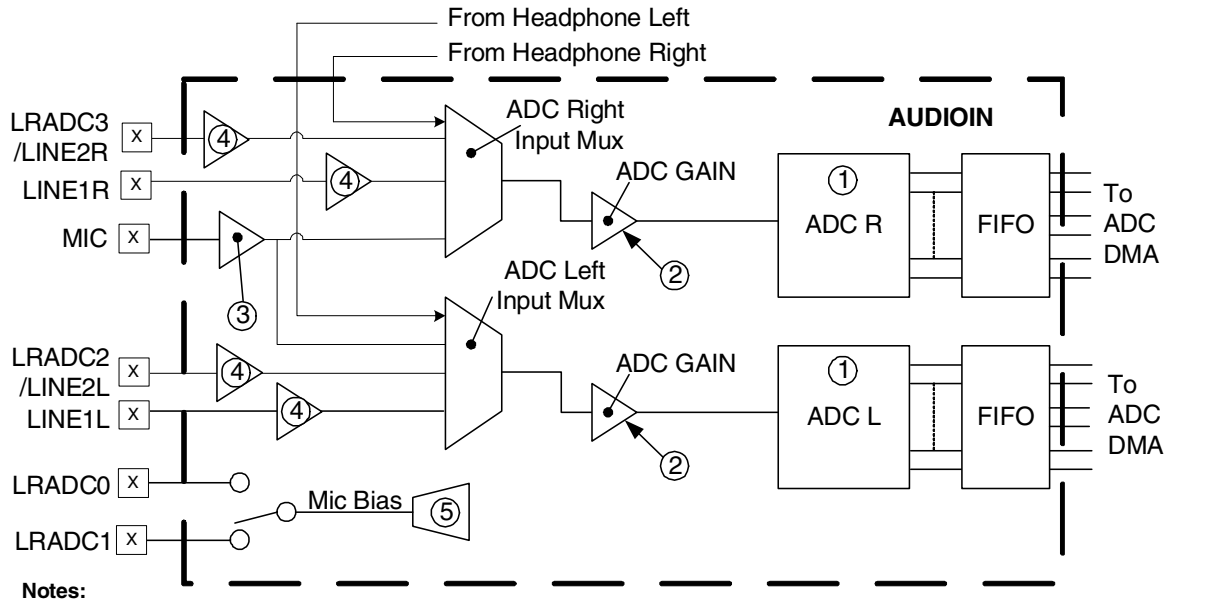
These 24-bit samples are then filtered using a three-stage FIR filter, consisting of 7, 11, and 33 taps, respectively. The AUDIOIN contains a sequencer, multiply-accumulate hardware, and a set of filter coefficients that performs successive iterations on the data stored in RAM. Intermediate data that is calculated along the taps/stages of the FIR are also stored in the AUDIOIN's RAM. The resultant filtered PCM data is then stored in a FIFO that can either be directly accessed by the host CPU or read by the i.MX23's AHB-APBX bridge DMA engine to store the data in on- or off-chip memory to allow access to system software.

In most cases, access to the AUDIOIN's data is made by the AHB-APBX bridge DMA. DMA channel 0 is dedicated to the AUDIOIN module. The DMA moves data from the AUDIOIN's memory-mapped data register to a RAM buffer every time a request is made. The buffer may be in on- or off-chip RAM. It is also possible for the CPU to manually move data from the AUDIOIN data register (HW_AUDIOIN_DATA) while monitoring either the FIFO or DMA request status bits in the AUDIOIN debug register (HW_AUDIOIN_ADCDEBUG).

Also present on the i.MX23 is an audio playback path called AUDIOOUT/DAC. Although each functions independently of one another, both the AUDIOIN and AUDIOOUT blocks share their FIR filter (sequencer/RAM/coefficients) and DMA controller. This combined module is titled the "digital filter" or DIGFILT. The register descriptions that follow both refer to each path independently (AUDIOIN and AUDIOOUT) as well as a whole (DIGFILT), due to the fact that clocks and resets affect either the shared resources or the design as a whole.

In order to configure the AUDIOIN/ADC for operation, the user must first clear the clock gate (CLKGATE) and soft reset (SFTRST) bits within the AUDIOIN control register (HW_AUDIOIN_CTRL). The run bit should remain off (zero), while all other control bits are initialized. It is important to note that there are also a number of control bits within the AUDIOOUT's address space that control functions within the analog ADC. The user must clear the clock gate and soft reset of the AUDIOOUT block in order to program these bits. Next, the bridge DMA controller channel 0 should be programmed and enabled to collect input audio samples to one or more RAM buffers. Finally, the run bit should be set to start AUDIOIN/ADC operation.

Each 32-bit register within the AUDIOIN's address space is aliased to four adjacent words. The first word is used for normal read-write access while the subsequent three words are contained within the register's set-clear-toggle (SCT) address space. Only bits that are written to with a one in this space are affected. For example, writing a one to bit using the register's set address sets that particular bit, while maintaining the state of all other bits. This convention allows easy bit manipulation without requiring the standard read-modify-write procedure. Bits that are written with a one to the register's clear address clear the bit, while the toggle address causes bits to invert their current state.



Notes:

1. HW_AUDIOIN_ADCVOLUME: Digital volume control. -0.5 dB to -100 dB in 0.5 dB steps.
2. HW_AUDIOIN_ADCVOL: Analog volume control that controls the ADC gain block. 0 dB to 22.5 dB gain in 1.5 dB steps.
3. HW_AUDIOIN_MICLINE_MICGAIN: Analog volume control that controls the microphone amplifier. 0, 20, 30, 40 dB gain.
4. HW_AUDIOIN_MICLINE_DIVIDE_LINE1/2.
5. HW_AUDIOOUT_MICLINE_MIC_BIAS, HW_AUDIOOUT_MICLINE_MIC_RESISTOR, HW_AUDIOOUT_MICLINE_MIC_SELECT.

Figure 28-2. AUDIOIn/ADC Block Diagram

28.2.1 AUDIOIN DMA

The DMA is typically controlled by a linked list of descriptors. The descriptors are usually circularly linked, causing the DMA to cycle through the set of DMA buffers. The DMA can be programmed to assert an IRQ when some or all of the buffers have been filled.

For example, AUDIOIN DMA descriptor 0 may program the DMA to fill a buffer, set the done IRQ, and fetch descriptor 1. Descriptor 1 programs the DMA to fill the next buffer. The DMA continues to operate normally while the IRQ is asserted. The CPU needs to respond to the IRQ before the DMA has filled all of the buffers. The DMA ISR clears the IRQ flag and informs the operating system that the buffers are filled.

In general, software copies data out of the buffers or adjusts the descriptors to point to other empty buffers. Software should also take advantage of the DMA's counting semaphore feature to synchronize the addition of new descriptors to the chain.

The DMA can put the AUDIOIN's PCM data into any memory-mapped location. For 32-bit PCM data, the left-channel sample is stored first in the lowest address, followed by the corresponding right-channel sample in the next word address (+4 bytes). For 16-bit mode, sample pairs are stored in each word. Right samples are stored in the upper half-word while left samples are stored in the lower half-word. Because the AUDIOIN always operates on stereo data, the PCM buffer should always have an integer number of

words. The audio data values are in two's complement format, where full-scale values range from 0x7FFFFFFF to 0x80000000 for 32-bit data or 0x7FFF to 0x8000 for 16-bit data.

In addition to the DMA IRQ used to indicate a filled AUDIOIN buffer, the module also has an overflow and underflow IRQ. Underflows should never occur, because (by design) the DMA should never attempt to read more data than is present within the AUDIOIN's FIFO. However, if the AUDIOIN ever attempts to write data into a full FIFO, an overflow occurs. This causes the overflow flag to be set in the AUDIOIN control register (HW_AUDIOIN_CTRL). If the overflow/underflow IRQ enable bit is set, then this condition also asserts an interrupt. The interrupt is cleared by writing a one to the overflow flag in the HW_AUDIOIN_CTRL's SCT clear address space. An AUDIOIN underflow is typically caused by the DMA running out of new buffers, or if the AHB or APBX is stalled or are otherwise unable to meet the bandwidth requirements at the current operating frequency. If the counting semaphore reaches 0, the DMA stops processing new descriptors and stops moving data from the AUDIOIN's data register (HW_AUDIOIN_DATA).

28.2.2 ADC Sample Rate Converter and Internal Operation

Table 28-1 contains the required value of the HW_AUDIOIN_ADCSRR register for various common sample rates. To make small sample rate adjustments (for example to track Fs fluctuations during a mix with an FM output to the DAC), the user may change the last few LSBs of the SRC_FRAC bit field to speed or slow the rate of sample consumption until equilibrium between the ADC's sample rate and the rate of another audio stream is met. Note that, unlike the DAC, only small deviations to SRC_FRAC can be made. The only valid values for BASEMULT, SRC_HOLD, and SRC_INT are listed in Table 28-1.

Table 28-1. Bit Field Values for Standard Sample Rates

SAMPLE RATE	HW_AUDIOIN_ADCSRR			
$Fsample_{ADC}$	BASEMULT	SRC_HOLD	SRC_INT	SRC_FRAC
192,000 Hz	0x4	0x0	0x0F	0x13FF
176,400 Hz	0x4	0x0	0x11	0x0037
128,000 Hz	0x4	0x0	0x17	0x0E00
96,000 Hz	0x2	0x0	0x0F	0x13FF
88,200 Hz	0x2	0x0	0x11	0x0037
64,000 Hz	0x2	0x0	0x17	0x0E00
48,000 Hz	0x1	0x0	0x0F	0x13FF
44,100 Hz	0x1	0x0	0x11	0x0037
32,000 Hz	0x1	0x0	0x17	0x0E00
24,000 Hz	0x1	0x1	0x0F	0x13FF
22,050 Hz	0x1	0x1	0x11	0x0037
16,000 Hz	0x1	0x1	0x17	0x0E00
12,000 Hz	0x1	0x3	0x0F	0x13FF
11,025 Hz	0x1	0x3	0x11	0x0037
8,000 Hz	0x1	0x3	0x17	0x0E00

Note: Sample rates greater than 48 kHz can only be used when the AUDIOOUT is disabled, and 44.1 kHz is the maximum sample rate at which both the AUDIOIN and AUDIOOUT can operate simultaneously.

For any of the desired sample rates, the internal sample-rate conversion factor is calculated according to the following formula:

$$\mathbf{SRConv_{ADC} = 65536 * [(Fanalog_{ADC})/ (8 * Fsample_{ADC})]}$$

The 1-bit sigma delta A/D converter is always sampled on a submultiple of the 24.0-MHz crystal oscillator frequency, as specified in the HW_AUDIOIN_ANACLKCTRL_ADCDIV register (see Figure 28-3). This divider generates sample strobes at $Fanalog_{ADC}$ where the divisors available come from the set {4,6,8,12,16,24}. It is recommended that ADCDIV always be set to 000 so that a 6.0-MHz 1-bit A/D sample rate is used. The sample strobe is used to integrate the 1-bit A/D values. As shown in Figure 28-3, these integrated values are filtered and then delivered to the ADC DMA to write into on-chip RAM.

Notice that the integrators run continuously while the filters produce samples at the decimated rate. Depending on the decimation or over-sample ratio of the CIC filter engine, the integrators will produce samples of various precisions and scale factors. The filtered values written to the ADC FIFO are signed 16-bit or 24-bit numbers with the conversion data LSB-justified, i.e., downscaled in the lower end of the word.

The scale factor column of the 48-kHz family of sample rates satisfies the property:

$$\mathbf{24.576\ MHz = Q * Fsample_{ADC}\ where\ Q\ comes\ from\ the\ set\ of\ integers}$$

These sample rates include 48 kHz, 32 kHz, 24 kHz, 16 kHz, 12 kHz, and 8 kHz.

There are also the members of the 44.1-kHz family, whose members satisfy the property:

$$\mathbf{16.9344\ MHz = Q * Fsample_{ADC}\ where\ Q\ comes\ from\ the\ set\ of\ integers}$$

These sample rates include 44.1 kHz, 22.05 kHz, and 11.025 kHz.

Since 24.576 kHz and 16.9344 MHz are relatively prime to 24.0 MHz, members of the 48-kHz family and 44.1-kHz family are related to the 24.0-MHz source clock by the relationship:

$$\mathbf{24.0\ MHz = P * Fsample_{ADC},\ where\ P\ is\ a\ rational\ number}$$

The A/D block includes a variable rate or rational decimator as shown in Figure 28-3 to accommodate these sample rates. Rational numbers in the ADC are approximated with a scaled fixed-point 24-bit value. In this case, the decimal point falls between bit 15 and bit 16. Therefore, the lower two bytes hold the fractional part, while the upper byte holds the whole number portion of the scaled fixed point. The position register uses this scaled fixed-point representation to hold the number of 1-bit samples to be

dropped (decimated) to find the next sample at which to produce a filtered multibit sigma delta A/D value to send to the DMA. Whenever the whole number part (bits 23:16) is zero, then a sample is produced.

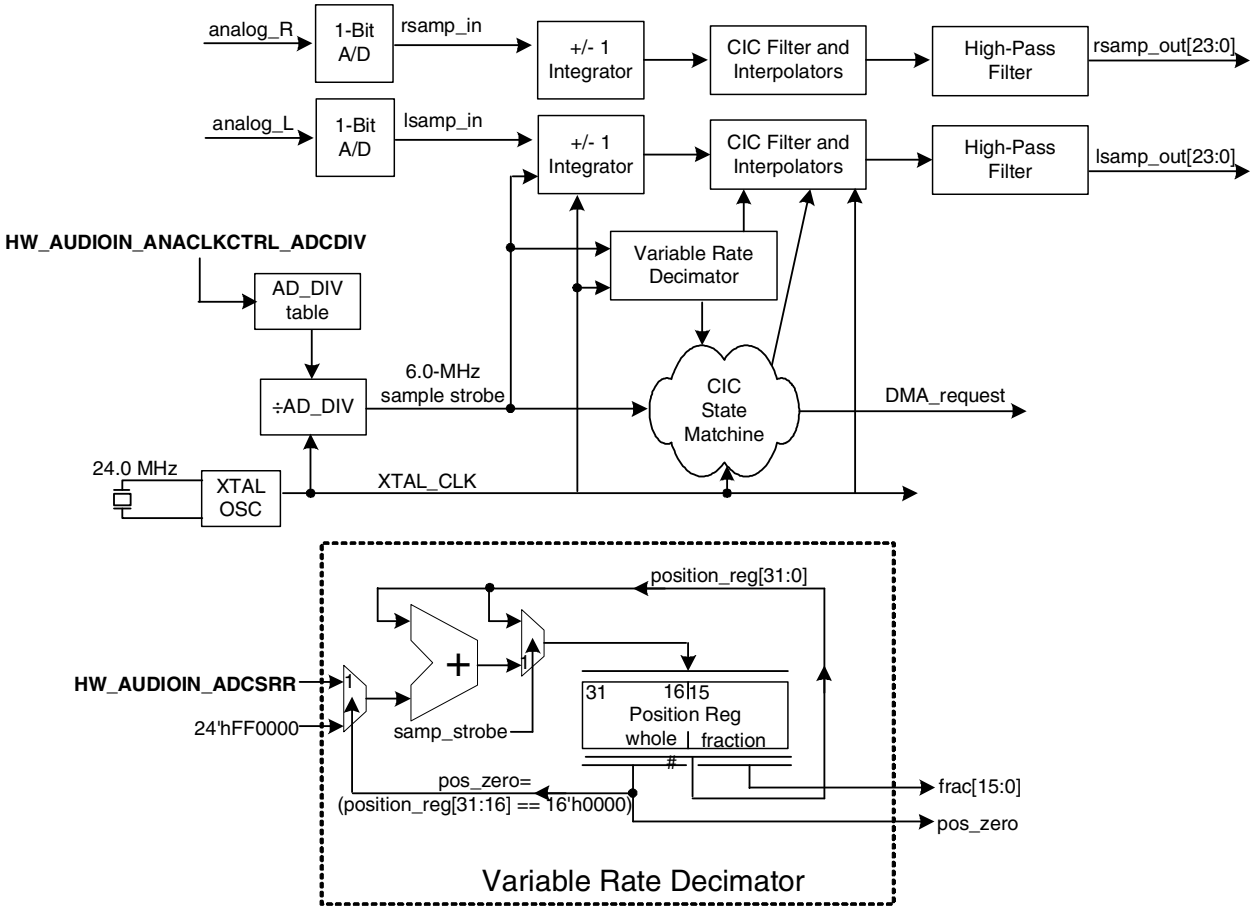


Figure 28-3. Variable-Rate A/D Converter

The range of values of the samples stored into the on-chip RAM is proportional to the square of the over-sample rate (OSR) used in the capture process. The larger the OSR, the longer period the integrators run in the ADC. As a result, the range of values seen for the same signal wave form captured at the same sample rate but with two different OSR will be different.

For example:

- An 8-kHz microphone captured at F_{ADC} = 6.0 MHz will be 36 times smaller than the values resulting from capturing the same source signal at F_{ADC} = 1.0 MHz.
- The peak range of values seen in a capture of a signal at 44.1 kHz with $Fanalog_{ADC}$ = 6.0 MHz is ±3200.
- The oversample ratio in this case is OSR= 136.054.
- Calculate a magnitude constant, K_{filter} for ADC's filter from this as $K_{filter} = OSR^2/Peak\ Value = (136.054)^2/3200 = 5.7846$.
- For any OSR in any sample rate, the peak value can be approximated by $Value_{peak} = OSR^2 / K_{filter}$

In signal processing, one frequently normalizes the range of values to ±1.0, as seen in a fixed-point scaled integer[1]. For a 24-bit DSP, the fixed point is placed between bit 23 and the sign bit (bit 24) (bit 1 = 2^0). So the desired maximum excursion is then ± 2^{23} or ±8388608.

One can calculate a normalization constant to multiply all incoming samples for each sampling condition from the following equation (note that OSR is fixed at 6 MHz for the i.MX23):

$$\mathbf{ScaleFactor = 2^{23} * K_{filter} / OSR^2}$$

If the incoming sample stream is multiplied, sample by sample, by ScaleFactor, then normalized ±1.0 samples result. All data output from the DIGFILT ADC are scaled according to this equation.

28.2.3 Line-In

The line inputs should be AC-coupled with 1 µF, X7R capacitors to isolate the source DC bias from the ADC's internal bias.

28.2.4 Microphone

The external microphone needs a bias voltage to enable it to operate. This bias voltage can be generated externally using discrete components as shown in Figure 28-4. Or, if either the LRADC0 or LRADC1 pin is available, it can be used to supply a bias voltage from an on-chip generator, as shown in Figure 28-5. To enable the generation of the microphone bias voltage on pin LRADC0 or LRADC1, the two MIC_RESISTOR bits in the HW_AUDIOIN_MICLINE register need to be written with required values for desired internal resistor selection. To select either pin LRADC1 or LRADC0 as the microphone bias source, write the MIC_SELECT bit in the HW_AUDIOIN_MICLINE register as follows: 0 for pin LRADC0, 1 for pin LRADC1.

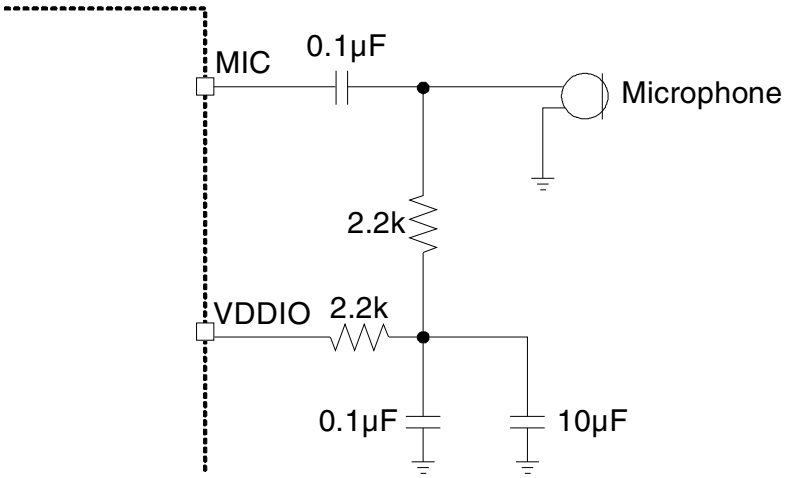


Figure 28-4. External Microphone Bias Generation

1.A normalized two's complement 24-bit number cannot actually express a value of +1.0 without overflowing.

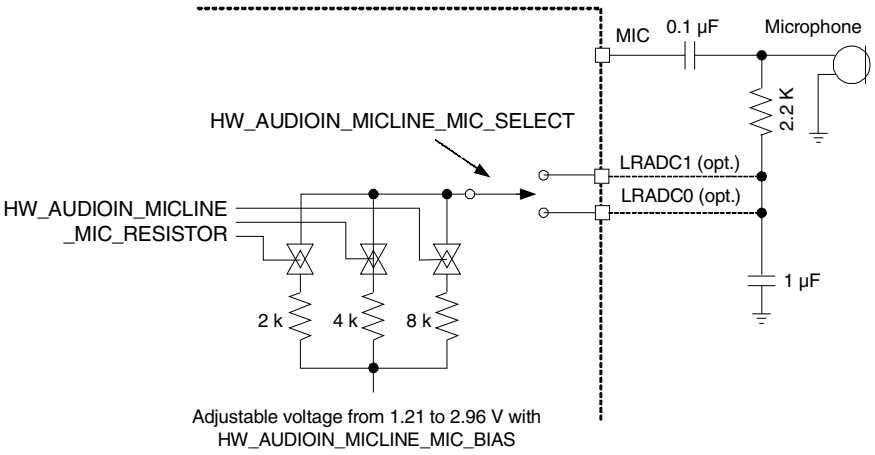


Figure 28-5. Internal Microphone Bias Generation

28.3 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, "Correct Way to Soft Reset a Block," for additional information on using the SFTRST and CLKGATE bit fields.

Note that the SFTRST and CLKGATE bits of both AUDIOOUT and AUDIOIN must be cleared before doing any operation with this block.

28.4 Programmable Registers

The following registers provide control for programmable elements of the AUDIOIN/ACD block.

28.4.1 AUDIOIN Control Register Description

The AUDIOIN Control Register provides overall control of the digital portion of the analog-to-digital converter.

HW_AUDIOIN_CTRL	0x000
HW_AUDIOIN_CTRL_SET	0x004
HW_AUDIOIN_CTRL_CLR	0x008
HW_AUDIOIN_CTRL_TOG	0x00C

Table 28-2. HW_AUDIOIN_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	RSRVD3									DMAWAIT_COUNT					RSRVD1					LR_SWAP	EDGE_SYNC	INVERT_1BIT	OFFSET_ENABLE	HPF_ENABLE	WORD_LENGTH	LOOPBACK	FIFO_UNDERFLOW_IRQ	FIFO_OVERFLOW_IRQ	FIFO_ERROR_IRQ_EN	RUN

Table 28-3. HW_AUDIOIN_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	AUDIOIN Module Soft Reset. Setting this bit to one forces a reset to portions of DIGFILT that control audio input and then gates the clocks off because the CLKGATE bit's reset state is to disable clocks. This bit must be cleared to zero for normal operation. Note that the CLKGATE bit does not affect SFTRST, because it must remain writeable during clock gating.
30	**CLKGATE**	RW	0x1	AUDIOIN Clock Gate Enable. When this bit is set to 1, it gates off the clocks to the portions of the DIGFILT block that control only input audio functions. It does not affect portions of the block that control AUDIOOUT. Clear the bit to zero for normal AUDIOIN operation. Note that when this bit is set, it remains writeable during clock gating so that it may be disabled by the user.
29:21	**RSRVD3**	RO	0x0	Reserved
20:16	**DMAWAIT_COUNT**	RW	0x0	DMA Request Delay Count. This bit field specifies the number of APBX clock cycles (0 to 31) to delay before each DMA request. This field acts as a throttle on the bandwidth consumed by the DIGFILT block. This field can be loaded by the DMA.
15:11	**RSRVD1**	RO	0x0	Reserved
10	**LR_SWAP**	RW	0x0	Left/Right Input Channel Swap Enable. Setting this bit to one swaps the left and right serial audio inputs from the ADC before being parallelized and having the sample rate converted by the AUDIOIN's CIC block.
9	**EDGE_SYNC**	RW	0x0	Serial Input Clock Edge Sync Select. This bit selects the edge of the ADC's serial input clock upon which the CIC-filter synchronizes for data receive. 0=Rising edge. 1=Falling edge
8	**INVERT_1BIT**	RW	0x0	Invert Serial Audio Input Enable. When set, this bit inverts the 1-bit serial input of both left and right channels from the ADC's sigma-delta modulator. 0=Normal operation. 1=Invert L/R serial audio input to the CIC block.

Table 28-3. HW_AUDIOIN_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
7	**OFFSET_ENABLE**	RW	0x1	ADC Analog High-Pass Filter Offset Calculation Enable. When this bit is set, the ADC's high pass filter actively adjusts the serial audio input, removing DC offset present within the signal. Active DC offset only takes place when the HPF_ENABLE bit is set. Once DC offset has been achieved, this bit can be cleared to maintain a constant level of offset. After clearing this bit, the HPF_ENABLE bit should remain set to maintain a constant DC offset.
6	**HPF_ENABLE**	RW	0x1	ADC High-Pass Filter Enable. When this bit is set, the ADC's analog high pass filter is enabled. Once enabled, the OFFSET_ENABLE bit can be set to cause the filter to begin removing DC offset from the incoming serial analog data. Once DC offset has been removed, the OFFSET_ENABLE bit should be cleared while the HPF_ENABLE bit remains set.
5	**WORD_LENGTH**	RW	0x0	PCM Audio Bit Size Select. This bit selects the size of the parallel PCM data collected by the AUDIOIN's input FIFO. 0=32-bit PCM samples. 1=16 bit samples. Note that the PCM audio data output from the FIR filter stages is 24 bits. For 16-bit operation, the resultant data is normalized by dropping the least significant 8 bits. For 32-bit mode, the two's complement PCM data is sign extended to 32 bits.
4	**LOOPBACK**	RW	0x0	AUDIOOUT-to-AUDIOIN Loopback Enable. Setting this bit to one connects the AUDIOOUT's digital serial data from the SDM module to the AUDIOIN's serial digital input to the CIC module, bypassing the analog DAC and ADC. This test mode provides a digital-only loopback which ties the output filter chain back to the input filter chain. This bit should be cleared to zero for normal operation.
3	**FIFO_UNDERFLOW_IRQ**	RW	0x0	FIFO Underflow Interrupt Status Bit. This bit is set by hardware if the AUDIOIN's FIFO underflows any time during operation. It is reset by software by writing a one to the SCT clear address space. An interrupt is issued to the host processor if this bit is set and FIFO_ERROR_IRQ_EN=1. Note that underflows should not occur by design because requests to the DMA are not made unless there is data present within the FIFO, and would indicate a serious DMA error.
2	**FIFO_OVERFLOW_IRQ**	RW	0x0	FIFO Overflow Interrupt Status Bit. This bit is set by hardware if the AUDIOIN's FIFO overflows due to a DMA request that is not serviced in time. It is reset by software writing a one to the SCT clear address space. An interrupt is issued to the host processor if this bit is set and FIFO_ERROR_IRQ_EN=1.

Table 28-3. HW_AUDIOIN_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
1	**FIFO_ERROR_IRQ_EN**	RW	0x0	FIFO Error Interrupt Enable. Set this bit to one to enable an AUDIOIN interrupt request to the host processor when either the FIFO overflow or underflow status bits are set. Note that this bit does not affect the state of the underflow/overflow status bits, but rather their ability to signal an interrupt to the CPU.
0	**RUN**	RW	0x0	AUDIOIN Enable. Setting this bit to one causes the AUDIOIN to begin converting data. Once 8 words of audio input samples are collected in its FIFO, it makes a DMA service request. Clearing this bit to zero stops data conversion and also causes the CLKGATE bit to be set.

DESCRIPTION:

The AUDIOIN Control Register contains bit fields used to control and monitor AUDIOIN operation including: reset, clocks, DMA transfers, analog ADC signal interface, high-pass filter operation, PCM data size, test, and interrupt control.

EXAMPLE:

```
HW_AUDIOIN_CTRL.RUN = 1; // start AUDIOIN conversion
```

28.4.2 AUDIOIN Status Register Description

The AUDIOIN Status Register is used to determine if the digital-to-analog converter is operational.

HW_AUDIOIN_STAT	0x010
HW_AUDIOIN_STAT_SET	0x014
HW_AUDIOIN_STAT_CLR	0x018
HW_AUDIOIN_STAT_TOG	0x01C

Table 28-4. HW_AUDIOIN_STAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADC_PRESENT	RSRVD3																														

Table 28-5. HW_AUDIOIN_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**ADC_PRESENT**	RO	0x1	AUDIOIN Functionality Present. This status bit is set to one in products that include the AUDIOIN/ADC. If this bit is zero, the AUDIOIN/ADC is permanently disabled and cannot be operated by the user.
30:0	**RSRVD3**	RO	0x0	Reserved

DESCRIPTION:

The AUDIOIN Status Register provides an indication of the presence of the ADC functionality.

EXAMPLE:

```
unsigned TestValue= HW_AUDIOIN_STAT.ADC_PRESENT;
```

28.4.3 AUDIOIN Sample Rate Register Description

The AUDIOIN Sample Rate Register is used to specify the sample rate from which the incoming serial audio data is converted as it is received by the CIC module from the analog ADC.

HW_AUDIOIN_ADCSRR	0x020
HW_AUDIOIN_ADCSRR_SET	0x024
HW_AUDIOIN_ADCSRR_CLR	0x028
HW_AUDIOIN_ADCSRR_TOG	0x02C

Table 28-6. HW_AUDIOIN_ADCSRR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
OSR	BASEMULT			RSRVD2	SRC_HOLD			RSRVD1			SRC_INT					RSRVD0			SRC_FRAC												

Table 28-7. HW_AUDIOIN_ADCSRR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**OSR**	RO	0x0	AUDIOIN Oversample Rate. Note that the oversample rate is fixed at 6 MHz. OSR6 = 0x0 AUDIOIN oversample rate at 6 MHz. OSR12 = 0x1 AUDIOIN oversample rate at 12 MHz.
30:28	**BASEMULT**	RW	0x1	Base Sample Rate Multiplier. This bit field is used to configure the ADC's sample rate to one of three ranges: single, double, or quad. This multiply factor is used to achieve sample rates greater than the standard rates of 32/44.1/48 kHz. A value of 0x1 should be used when selecting half and quarter sample rates. Note that sample rates greater than 48 kHz may only be used when the AUDIOOUT is disabled, and 44.1 kHz is the maximum sample rate at which both the AUDIOIN and AUDIOOUT can operate simultaneously. SINGLE_RATE = 0x1 Single-rate multiplier (for 48/44.1/32 kHz as well as half and quarter rates). DOUBLE_RATE = 0x2 Double-rate multiplier (for 96/88.2/64 kHz). QUAD_RATE = 0x4 Quad-rate multiplier (for 192/176.4/128 kHz).
27	**RSRVD2**	RO	0x0	Reserved. Always write a zero to this bit field.

Table 28-7. HW_AUDIOIN_ADCSRR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
26:24	**SRC_HOLD**	RW	0x0	Sample Rate Conversion Hold Factor. This bit is used to hold a sample of a variable number of clock cycles in order to generate half and quarter sample rates when dividing down the AUDIOIN's internal rate using the equation: output_sample_rate = (6x10^6 * BASEMULT) / (SRC_INT.SRC_FRAC * 8 * (SRC_HOLD + 1)). Refer to the sample rate table earlier in this chapter that provides a list of bit field values required to achieve all common sample rates.
23:21	**RSRVD1**	RO	0x0	Reserved. Always write zeros to this bit field.
20:16	**SRC_INT**	RW	0x11	Sample Rate Conversion Integer Factor. This bit field is the integer portion of a divide term used to sample-rate-convert the AUDIOIN's internal rate using the equation; output_sample_rate = (6x10^6 * BASEMULT) / (SRC_INT.SRC_FRAC * 8 * (SRC_HOLD + 1)). Refer to the sample rate table earlier in this chapter that provides a list of bit field values required to achieve all common sample rates.
15:13	**RSRVD0**	RO	0x0	Reserved. Always write zeros to this bit field.
12:0	**SRC_FRAC**	RW	0x37	Sample Rate Conversion Fraction Factor. This bit field is the fractional portion of a divide term used to sample-rate-convert the AUDIOIN's internal rate using the equation; output_sample_rate = (6x10^6 * BASEMULT) / (SRC_INT.SRC_FRAC * 8 * (SRC_HOLD + 1)). Refer to the sample rate table earlier in this chapter that provides a list of bit field values required to achieve all common sample rates.

DESCRIPTION:

The AUDIOIN Sample Rate Register contains bit fields used to specify the rate at which the ADC samples incoming analog audio.

EXAMPLE:

```
// Program the DAC to output a sample rate of 48 kHz:
HW_AUDIOIN_ADCSRR.BASEMULT = 0x1; // quad-rate
HW_AUDIOIN_ADCSRR.SRC_HOLD = 0x0; // 0 for full- double- quad-rates
HW_AUDIOIN_ADCSRR.SRC_INT = 0xF; // 15 for the integer portion
HW_AUDIOIN_ADCSRR.SRC_FRAC = 0x13FF; // the fractional portion
```

28.4.4 AUDIOIN Volume Register Description

The AUDIOIN Volume Register is used to adjust the signal level of the recorded audio input from the ADC.

HW_AUDIOIN_ADCVOLUME	0x030
HW_AUDIOIN_ADCVOLUME_SET	0x034
HW_AUDIOIN_ADCVOLUME_CLR	0x038
HW_AUDIOIN_ADCVOLUME_TOG	0x03C

Table 28-8. HW_AUDIOIN_ADCVOLUME

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD5			VOLUME_UPDATE_LEFT	RSRVD4		EN_ZCD	RSRVD3	VOLUME_LEFT								RSRVD2			VOLUME_UPDATE_RIGHT	RSRVD1				VOLUME_RIGHT							

Table 28-9. HW_AUDIOIN_ADCVOLUME Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSRVD5**	RO	0x00	Reserved
28	**VOLUME_UPDATE_LEFT**	RO	0x0	Left Channel Volume Update Pending. This bit is set to one by the hardware when an AUDIOIN volume update is pending, i.e., waiting on a zero crossing on the left channel. The bit is set following a write to the VOLUME_LEFT bit field and is cleared when the attenuation value is applied to the PCM input stream (at a zero-crossing). This status bit is not used when EN_ZCD=0.
27:26	**RSRVD4**	RO	0x00	Reserved
25	**EN_ZCD**	RW	0x0	Enable Zero Cross Detect. This bit enables/disables use of the zero cross detect circuit in the ADC (rather than enabling the circuit itself). When enabled, changes to the volume bit fields are not applied until it is detected that the input signal's sign bit toggles (crosses zero amplitude). When disabled, changes to the volume bit fields take effect immediately when written.
24	**RSRVD3**	RO	0x0	Reserved
23:16	**VOLUME_LEFT**	RW	0xfe	Left Channel Volume Setting. This bit field is used to establish the incoming audio signal strength during record. Volume ranges from -0.5 dB (0xFE) to -100 dB (0x37). Each increment of this bit field causes a half-dB increase in volume. Note that values 0x00-0x37 all produce the same attenuation level of -100 dB. Also note that a setting of 0xFF is reserved.
15:13	**RSRVD2**	RO	0x00	Reserved
12	**VOLUME_UPDATE_RIGHT**	RO	0x0	Right Channel Volume Update Pending. This bit is set to one by the hardware when an AUDIOIN volume update is pending, i.e., waiting on a zero crossing on the right channel. The bit is set following a write to the VOLUME_RIGHT bit field and is cleared when the attenuation value is applied to the PCM input stream (at a zero-crossing). This status bit is not used when EN_ZCD=0.

Table 28-9. HW_AUDIOIN_ADCVOLUME Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
11:8	**RSRVD1**	RO	0x0	Reserved
7:0	**VOLUME_RIGHT**	RW	0xfe	Right Channel Volume Setting. This bit field is used to establish the incoming audio signal strength during record. Volume ranges from -0.5 dB (0xFF) to -100 dB (0x37). Each increment of this bit field causes a half-dB increase in volume. Note that values 0x00-0x37 all produce the same attenuation level of -100 dB. Also note that a setting of 0xFF is reserved.

DESCRIPTION:

The AUDIOIN Volume Register allows independent volume control of the left and right channels. Input audio can be attenuated in 0.5-dB steps, from full scale down to a minimum of -100 dB. This register is also used to enable/control volume updates such that they are only applied when PCM values cross zero to prevent unwanted audio artifacts.

EXAMPLE:

```
HW_AUDIOIN_ADCVOLUME.U = 0x00ff00ff; maximum volume for left and right channels.
```

28.4.5 AUDIOIN Debug Register Description

The AUDIOIN Debug Register is used for testing and debugging the AUDIOIN block.

HW_AUDIOIN_ADCDEBUG	0x040
HW_AUDIOIN_ADCDEBUG_SET	0x044
HW_AUDIOIN_ADCDEBUG_CLR	0x048
HW_AUDIOIN_ADCDEBUG_TOG	0x04C

Table 28-10. HW_AUDIOIN_ADCDEBUG

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ENABLE_ADCDMA	RSRVD1																											ADC_DMA_REQ_HAND_SHAKE_CLK_CROSS	SET_INTERRUPT3_HAND_SHAKE	DMA_PREQ	FIFO_STATUS

Table 28-11. HW_AUDIOIN_ADCDEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**ENABLE_ADCDMA**	RW	0x0	AUDIOIN Digital Path Test Enable. This bit is used solely for development and debug and is not functional on production parts. When enabled, it causes the AUDIOIN's serial audio data input to bypass the CIC block, to be assembled into 32-bit words and transferred out to memory using the AUDIOOUT's DMA Channel 1. Unlike loopback, this test mode provides a means of verifying the digital portion of the AUDIOIN/ADC logic without causing the audio data to pass through the AUDIOOUT's FIR filter stages.
30:4	**RSRVD1**	RO	0x00	Reserved
3	**ADC_DMA_REQ_HAND_SHAKE_CLK_CROSS**	RO	0x0	DMA Request Sync Status. This bit reflects the current state of the second flop on the chain of three flip-flops used to synchronize the AUDIOIN's DMA request signal from the module's internal 24-MHz clock to the APBX's memory clock domain. This bit is only intended for test.
2	**SET_INTERRUPT3_HAND_SHAKE**	RO	0x0	Interrupt[3] Status. This bit reflects the current state of the APBX interface state machine's internal interrupt[3] signal used to prioritize channels 0 and 1 DMA requests from the DIGFILT. This bit is only intended for test.
1	**DMA_PREQ**	RO	0x0	DMA Request Status. This bit reflects the current state of the AUDIOIN's DMA request signal. DMA requests are issued any time the request signal toggles. This bit can be polled by software, in order to manually move samples from the AUDIOIN's FIFO to a memory buffer when the AUDIOIN's DMA channel is not used.
0	**FIFO_STATUS**	RO	0x0	FIFO Status. This bit is set when the AUDIOIN's FIFO contains any amount of valid data and is cleared when the FIFO is empty.

DESCRIPTION:

The AUDIOIN Debug Register provides read-only access of various internal AUDIOIN module signals to assist in debug and validation, as well as control of ADCDMA test mode.

EXAMPLE:

```
unsigned tempStatus = HW_AUDIOIN_ADCDEBUG.FIFO_STATUS;
```

28.4.6 ADC Mux Volume and Select Control Register Description

This register controls operation of the analog ADC input mux.

HW_AUDIOIN_ADCVOL	0x050
HW_AUDIOIN_ADCVOL_SET	0x054
HW_AUDIOIN_ADCVOL_CLR	0x058
HW_AUDIOIN_ADCVOL_TOG	0x05C

Table 28-12. HW_AUDIOIN_ADCVOL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD4			VOLUME_UPDATE_PENDING	RSRVD3		EN_ADC_ZCD	MUTE	RSRVD2										SELECT_LEFT		GAIN_LEFT				RSRVD1		SELECT_RIGHT		GAIN_RIGHT			

Table 28-13. HW_AUDIOIN_ADCVOL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSRVD4**	RO	0x0	Reserved
28	**VOLUME_UPDATE_PENDING**	RO	0x0	Volume Update Pending. This bit is set to one by the hardware when either an ADC left or right volume update is pending, i.e., waiting on a zero crossing. The bit is set following a write to either the GAIN_LEFT or GAIN_RIGHT bit fields and is cleared when both gain values are applied to the input coincident with zero-crossings in both the right and left channels. This status bit is not used when EN_ADC_ZCD=0.
27:26	**RSRVD3**	RO	0x0	Reserved
25	**EN_ADC_ZCD**	RW	0x0	Enable Zero Cross Detect for ADC Amplifier.
24	**MUTE**	RW	0x1	ADC Mute. When set, this bit mutes both the left and right channel analog inputs. 1=Mute. 0=Unmute.
23:14	**RSRVD2**	RO	0x000	Reserved
13:12	**SELECT_LEFT**	RW	0x0	ADC Left Channel Input Source Select. This bit field is used to select the analog input source of the ADC's left channel. 00=Microphone. 01=Line1. 10=Headphone. 11=Line2 (169-BGA only). Line2 left input is LRADC2.
11:8	**GAIN_LEFT**	RW	0x0	Left Channel ADC Gain. This bit selects the level of gain applied to the left channel analog input. Each increment of this field represents a 1.5dB gain. Programming a value of 0x0, applies a 0dB gain, 0x1 applies a 1.5dB gain, and so on up to a maximum gain of 22.5dB when a value of 0xF is used.
7:6	**RSRVD1**	RO	0x0	Reserved
5:4	**SELECT_RIGHT**	RW	0x0	ADC Right Channel Input Source Select. This bit field is used to select the analog input source of the ADC's right channel. 00=Microphone. 01=Line1. 10=Headphone. 11=Line2 (169-BGA only).
3:0	**GAIN_RIGHT**	RW	0x0	Right Channel ADC Gain. This bit selects the level of gain applied to the right channel analog input. Each increment of this field represents a 1.5-dB gain. Programming a value of 0x0, applies a 0-dB gain, 0x1 applies a 1.5-dB gain, and so on up to a maximum gain of 22.5 dB when a value of 0xF is used.

DESCRIPTION:

This register supplies the volume, mute, and input select controls for the analog ADC mux/gain amplifier.

EXAMPLE:

```
HW_AUDIOIN_ADCVOL.MUTE = 0;
```

28.4.7 Microphone and Line Control Register Description

This register provides the microphone and line control bits.

HW_AUDIOIN_MICLINE	0x060
HW_AUDIOIN_MICLINE_SET	0x064
HW_AUDIOIN_MICLINE_CLR	0x068
HW_AUDIOIN_MICLINE_TOG	0x06C

Table 28-14. HW_AUDIOIN_MICLINE

31 30	29	28	27 26 25	24	23 22	21 20	19	18 17 16	15 14 13 12	11 10 09 08	07 06	05 04	03 02	01 00
RSRVD6	DIVIDE_LINE1	DIVIDE_LINE2	RSRVD5	MIC_SELECT	RSRVD4	MIC_RESISTOR	RSRVD3	MIC_BIAS	RSRVD2			MIC_CHOPCLK	RSRVD1	MIC_GAIN

Table 28-15. HW_AUDIOIN_MICLINE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD6**	RO	0x0	Reserved
29	**DIVIDE_LINE1**	RW	0x0	Attenuate Line1 Input. When used in conjunction with a 10K series external resistor on the Line1 pin, this bit causes the Line1 input signal to be attenuated by 9.5dB (+/-1.5 dB) t o allow a 2-Vrms input signal. This bit affects the left and right channels of both the ADC and headphone.
28	**DIVIDE_LINE2**	RW	0x0	Attenuate Line2 Input. When used in conjunction with a 10K series external resistor on the line2 pin, this bit causes the Line2 input signal to be attenuated by 9.5 dB (+/-1.5 dB) to allow a 2-Vrms input signal. This bit affects the left and right channels of the ADC.
27:25	**RSRVD5**	RO	0x0	Reserved
24	**MIC_SELECT**	RW	0x0	Microphone Bias Pin Select. When MIC_RESISTOR is enabled (non-zero), this bit is used to select the pin source for the Micbias input voltage reference. 0=LRADC0. 1=LRADC1. Note that the LRADC pin that is selected for Micbias cannot also be used as an LRADC input.
23:22	**RSRVD4**	RO	0x0	Reserved

Table 28-15. HW_AUDIOIN_MICLINE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
21:20	**MIC_RESISTOR**	RW	0x0	Microphone Bias Resistor Select. Note that the analog ADC block must be powered on before turning on the Micbias circuit (ADC bit within the HW_AUDIOOUT_PWRDN register must be cleared to zero) 00=Off. 01=2 KOhm. 10=4 KOhm. 11=8 KOhm.
19	**RSRVD3**	RO	0x0	Reserved
18:16	**MIC_BIAS**	RW	0x0	Microphone Bias Voltage Select. 0=1.21 V, 1=1.46 V, up to 7=2.96 V (0.25-V increments)
15:6	**RSRVD2**	RO	0x00	Reserved
5:4	**MIC_CHOPCLK**	RW	0x0	Enable chopping in the microphone amplifier: 00=Disabled. 01=192 kHz. 10=96 kHz. 11=48 kHz.
3:2	**RSRVD1**	RO	0x00	Reserved
1:0	**MIC_GAIN**	RW	0x0	Microphone Gain. 00=0 dB, 01=20 dB, 10=30 dB, 11=40 dB.

DESCRIPTION:

This register provides the microphone and line control bits.

EXAMPLE:

```
HW_AUDIOIN_MICLINE.MIC_GAIN = 0x2; // 30 dB
```

28.4.8 Analog Clock Control Register Description

This register provides analog clock control.

HW_AUDIOIN_ANACLKCTRL	0x070
HW_AUDIOIN_ANACLKCTRL_SET	0x074
HW_AUDIOIN_ANACLKCTRL_CLR	0x078
HW_AUDIOIN_ANACLKCTRL_TOG	0x07C

Table 28-16. HW_AUDIOIN_ANACLKCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CLKGATE	RSRVD4																				DITHER_OFF	SLOW_DITHER	INVERT_ADCCLK	RSRVD3		ADCCLK_SHIFT		RSRVD2		ADCDIV	

Table 28-17. HW_AUDIOIN_ANACLKCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**CLKGATE**	RW	0x1	Analog Clock Gate. Set this bit to gate the clocks for the ADC converter and associated digital filter.
30:11	**RSRVD4**	RO	0x0	Reserved
10	**DITHER_OFF**	RW	0x1	ADC Dither Disable. When this bit is set, dither is disabled within the ADC.

Table 28-17. HW_AUDIOIN_ANACLKCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
9	**SLOW_DITHER**	RW	0x0	Slow Dither. When dither is enabled (DITHER_OFF=0), and this bit is set, ADC input signal dithering is slowed to half its normal rate.
8	**INVERT_ADCCLK**	RW	0x0	ADC clock invert. Set this bit to invert the ADC_CLK for the ADC sigma-delta converter and associated digital filters.
7:6	**RSRVD3**	RO	0x0	Reserved
5:4	**ADCCLK_SHIFT**	RW	0x0	ADC Analog Clock Phase Shift. This bit field is used to shift the phase of the adc clock realtive to the switching of the dcdc converter. This bit field should only be changed per Freescale. 00=no shift. 01=20ns shift 10=40ns shift 11=60ns shift.
3	**RSRVD2**	RO	0x0	Reserved
2:0	**ADCDIV**	RW	0x0	ADC Analog Clock Divider. This bit field is used to select the oversampling clock rate used by the ADC. This bit field should only be changed per Freescale. 000=6 MHz. 001=4 MHz. 010/100=3 MHz. 011/101=2 MHz. 110=1.5 MHz. 111=1 MHz.

DESCRIPTION:

This register provides analog clock control.

EXAMPLE:

```
HW_AUDIOIN_ANACLKCTRL.ADCDIV = 0x2; // 3 MHz
```

28.4.9 AUDIOIN Read Data Register Description

The AUDIOIN Read Data Register provides access to incoming PCM audio samples.

HW_AUDIOIN_DATA	0x080
HW_AUDIOIN_DATA_SET	0x084
HW_AUDIOIN_DATA_CLR	0x088
HW_AUDIOIN_DATA_TOG	0x08C

Table 28-18. HW_AUDIOIN_DATA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
HIGH																LOW															

Table 28-19. HW_AUDIOIN_DATA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**HIGH**	RO	0x0000	Right Sample or Sample High Half-Word. For 16-bit sample mode, this field contains the right channel sample. For 32-bit sample mode, this field contains the most significant 16 bits of the 32-bit sample (either left or right).
15:0	**LOW**	RO	0x0000	Left Sample or Sample Low Half-Word. For 16-bit sample mode, this field contains the left channel sample. For 32-bit per sample mode, this field contains the least significant 16 bits of the 32-bit sample (either left or right).

DESCRIPTION:

The AUDIOIN Read Data Register provides 32-bit data transfers for the DMA, or, alternatively, can be directly read by the CPU. Each data value read from the register is transferred from the AUDIOIN's FIFO that contains the resultant audio data that has passed through it's digital FIR filter stages. These 32-bit values contain either one 32-bit sample or two 16-bit samples, depending on how the data size is programmed. Note that the PCM audio data output from the FIR filter stages is 24-bit. For 16-bit operation, the resultant data is normalized by dropping the least significant 8 bits. For 32-bit mode, the two's complement PCM data is sign extended to 32 bits.

EXAMPLE:

```
unsigned long TestValue= HW_AUDIOIN_DATA.U; // read a 32 bit value from the read data register in
CPU diagnostic (non-DMA) mode
```

AUDIOIN Block v1.3, Revision 1.52

Chapter 29
AUDIOOUT/DAC

This chapter describes the AUDIOOUT/DAC module implemented on the i.MX23, including DMA, sample rate conversion, internal operation, reference control settings, and headphone amplifier operation. Programmable registers are described in Section 29.4, "Programmable Registers."

29.1 Overview

The i.MX23 features an audio playback path that consists of the AUDIOOUT digital multi-stage FIR filter, followed by a sigma-delta digital-to-analog converter (DAC). PCM audio samples are transferred from a buffer in memory using the APBX bridge DMA to the AUDIOOUT's FIFO. Sample pairs are processed by a three-stage finite impulse response filter. The resultant PCM samples are input to the sigma-delta modulation (SDM) block, where they are serialized, sample rate converted to the desired output rate, and output to the analog DAC.

The analog audio destination can be selected from one of two possible outputs:

- Stereo Headphone Amplifier Output
- Speaker Amplifier Output

The AUDIOOUT module provides the following functions:

- 1-bit sigma-delta DAC
- Three stage upsampling FIR filter: 33-tap (1:2), 11-tap (2:4), 7-tap (4:8), supporting conversion from quarter, half, full, double and quad sample rates that are multiples of the standard 32K, 44.1K, and 48K Hz rates
- Parallel-to-serial bit-stream decimator
- Sample rate converter (SRC)
- 16- or 32-bit PCM sample widths
- APBX bridge DMA interface
- Independent control of each channel's volume (including mute)
- Freescale 3D virtualization
- ADC-to-DAC internal loopback for product development
- Control bit fields used for analog DAC settings

Figure 29-1 shows a high-level diagram of the AUDIOOUT module. See Figure 1-4 for a diagram of the audio path and control options.

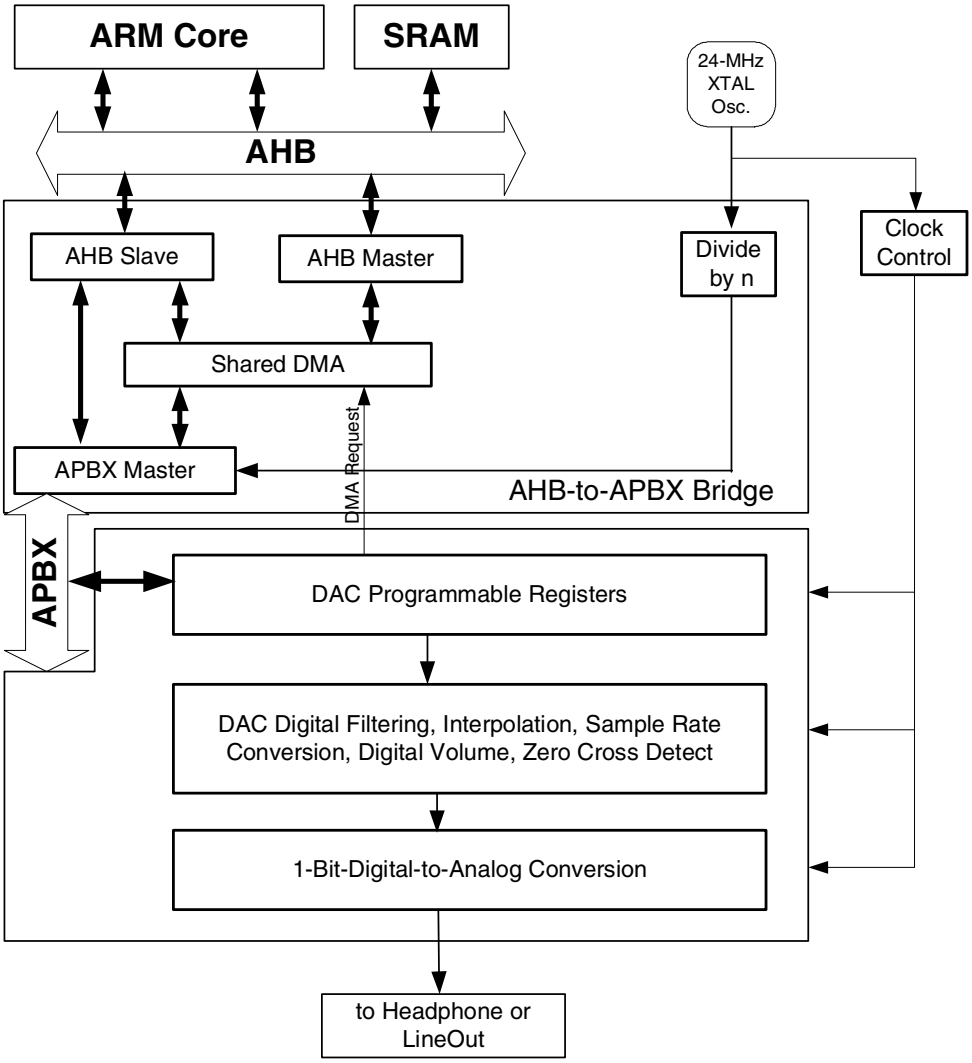


Figure 29-1. Functional AUDIOOUT/DAC Block Diagram

29.2 Operation

Audio data conversion begins by either using the i.MX23's AHB-APBX bridge DMA engine to write two's complement PCM data to the AUDIOOUT's input FIFO, or by writing the data directly to the AUDIOOUT's data register via the host CPU. The data is then normalized to 24-bit samples and then filtered using a 3-stage FIR filter, consisting of 33, 11, and 7 taps, respectively. The AUDIOOUT contains a sequencer, multiply-accumulate hardware, and a set of filter coefficients that performs successive iterations on the data stored in RAM. Intermediate data calculated along the taps/stages of the FIR are also stored in the AUDIOOUT's RAM. The AUDIOOUT module includes hardware for interpolation, sample

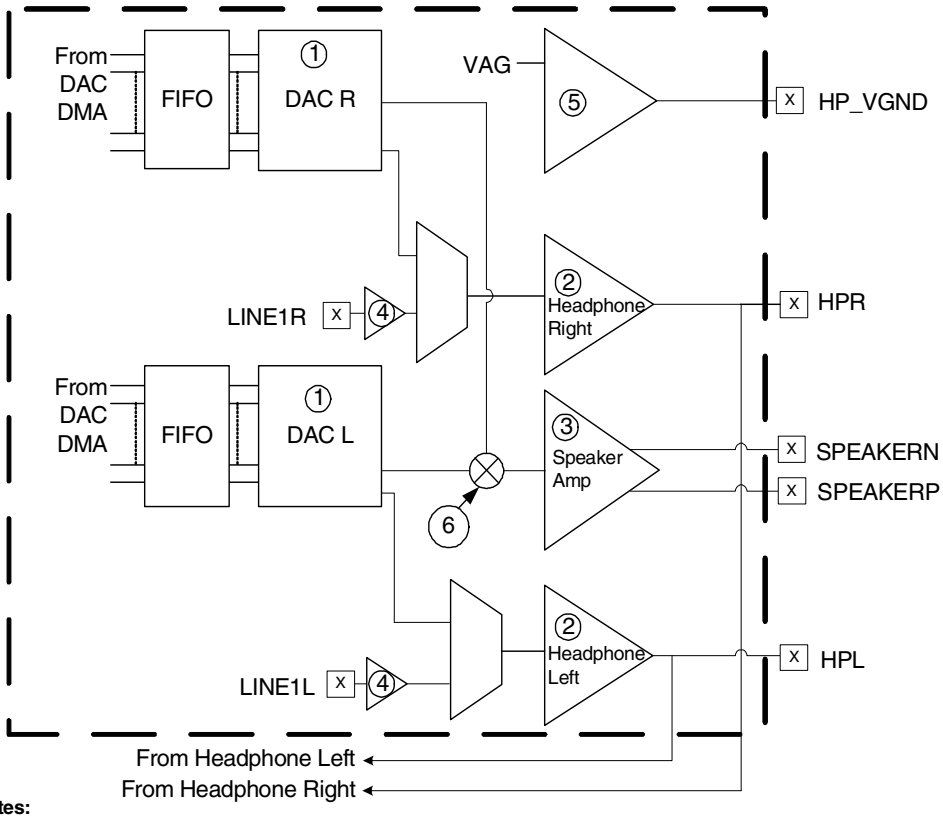
and hold, and sigma-delta modulation that is applied to the filtered parallel PCM data. The resultant over-sampled 1-bit serial stream is then output to the high-performance analog stereo sigma-delta DAC.

In most cases, the AUDIOOUT's PCM data is transferred by the AHB-APBX bridge DMA. DMA channel 1 is dedicated to the AUDIOOUT module. The DMA moves data to the AUDIOOUT's memory-mapped data register from a RAM buffer every time a request is made. The buffer may be in on-chip or off-chip RAM. It is also possible for the CPU to manually move data to the AUDIOOUT data register (HW_AUDIOOUT_DATA) while monitoring either the FIFO or DMA request status bits in the AUDIOOUT debug register (HW_AUDIOOUT_DACDEBUG).

Also present on the i.MX23 is an audio record path called AUDIOIN/ADC. Although each functions independently of one another, both the AUDIOOUT and AUDIOIN blocks share their FIR filter (sequencer/RAM/coefficients) and DMA controller. This combined module is titled the "digital filter" or DIGFILT. The register descriptions that follow refer both to each path independently (AUDIOOUT and AUDIOIN) as well as a whole (DIGFILT), due to the fact that clocks and resets affect either the shared resources or the design as a whole.

In order to configure the AUDIOOUT/DAC for operation, the user must first clear the clock gate (CLKGATE) and soft reset (SFTRST) bits within the AUDIOOUT control register (HW_AUDIOOUT_CTRL). The run bit should remain off (zero), while all other control bits are initialized. It is important to note that there are also a number of control bits within the AUDIOOUT's address space that control functions within the analog DAC. Next, the bridge DMA controller channel 1 should be programmed and enabled to supply output audio samples from one or more RAM buffers. Finally, the run bit should be set to start AUDIOOUT/DAC operation. After the 8-word AUDIOOUT FIFO is filled, conversion begins.

Each 32-bit register within the AUDIOOUT's address space is aliased to four adjacent words. The first word is used for normal read-write access, while the subsequent three words are contained within the register's set-clear-toggle (SCT) address space. Only bits that are written to with a one in this space are affected. For example, writing a one to a bit using the register's set address, sets that particular bit while maintaining the state of all other bits. This convention allows easy bit manipulation without requiring the standard read-modify-write procedure. Bits that are written with a one to the register's clear address clear the bit, while the toggle address causes bits to invert their current state.



Notes:

1. HW_AUDIOOUT_DACVOLUME: Digital volume control. -0.5 dB to -100 dB in 0.5 dB steps.
2. HW_AUDIOOUT_HPVOL: Analog volume control. 6 dB to -57.5 dB in 0.5 dB steps.
3. HW_AUDIOOUT_SPEAKERCTRL: Analog control for speaker amplifier, fixed gain of 9.5 dB from each DAC, 15.5dB total.
4. HW_AUDIOIN_MICLINE_DIVIDE_LINE1/2.
5. HW_AUDIOOUT_PWDN: Enable capless headphone common amplifier.
6. The speaker amplifier mixes the left and right DAC outputs and provides a fixed gain of 6x (or 15.5dB).

Figure 29-2. AUDIOOUT/DAC Block Diagram

29.2.1 AUDIOOUT DMA

The DMA is typically controlled by a linked list of descriptors. Generally, the descriptors are circularly linked to cause the DMA to cycle through the set of DMA buffers. The DMA can be programmed to assert an IRQ when some or all of the buffers have been transmitted. For example, AUDIOOUT DMA descriptor 0 may program the DMA to transmit a buffer, set the done IRQ, and fetch descriptor 1. Descriptor 1 programs the DMA to transmit the next buffer. The DMA continues to operate normally while the IRQ is asserted. The CPU needs to respond to the IRQ before the DMA has transmitted all of the buffers with new data. The DMA ISR clears the IRQ flag and prepares buffers and/or descriptors with new data. In general, software copies new data into the buffers or adjust the descriptors to point to existing buffers. Software should also take advantage of the DMA's counting semaphore feature to synchronize the addition of new descriptors to the chain.

The DMA can take PCM data from any memory-mapped location. For 32-bit PCM data, the left channel sample is stored first in the lowest address, followed by the corresponding right channel sample in the

next word address (+4 bytes). For 16-bit mode, sample pairs are stored in each word. Right samples are stored in the upper half word, while left samples are stored in the lower half word. Because the AUDIOOUT always operates on stereo data, the PCM buffer should always have an integer number of words. It is not possible to play mono data unless the mono samples are each repeated twice in memory, once for the left channel and once for the right channel. The audio data values are in two's complement format, where full scale values range from 0x7FFFFFFF to 0x80000000 for 32-bit data or 0x7FFF to 0x8000 for 16-bit data.

In addition to the DMA IRQ used for AUDIOOUT buffer refill, the AUDIOOUT also has an underflow and overflow IRQ. Overflows should never occur, because (by design) the DMA should never attempt to write more data than there is space available within the AUDIOOUT's FIFO. However, if the DMA does not keep up with requests and the FIFO is emptied by the AUDIOOUT's filter stages, an underflow occurs. This causes the underflow flag to be set in the AUDIOOUT control register (HW_AUDIOOUT_CTRL). If the overflow/underflow IRQ enable bit is set, then this condition also asserts an interrupt. The interrupt is cleared by writing a one to the underflow flag in the HW_AUDIOOUT_CTRL's SCT clear address space. An AUDIOOUT underflow is typically caused by the DMA running out of new buffers, or if the AHB or APBX is stalled or is otherwise unable to meet the bandwidth requirements at the current operating frequency. If the counting semaphore reaches 0, the DMA stops processing new descriptors and stops moving data to the AUDIOOUT's data register (HW_AUDIOOUT_DATA).

In some cases, it may be desirable to synchronize the DAC clock speed with some other reference. Examples include a system playing from a network stream or digital FM receiver. In these cases, the AUDIOOUT sample rate register can be adjusted to speed up or slow down the data rate. Software needs to periodically monitor the buffer positions to make corrections as necessary.

29.2.2 DAC Sample Rate Converter and Internal Operation

Table 29-1 contains the required value of the HW_AUDIOOUT_DACSRR register for various common sample rates. Although these are the standard rates, any sample rate from 8K to 192 kHz can be programmed.

Table 29-1. Bit Field Values for Standard Sample Rates

Sample Rate	HW_audioout_dacsRr			
$Fsample_{DAC}$	BASEMULT	SRC_HOLD	SRC_INT	SRC_FRAC
192,000 Hz	0x4	0x0	0x0F	0x13FF
176,400 Hz	0x4	0x0	0x11	0x0037
128,000 Hz	0x4	0x0	0x17	0x0E00
96,000 Hz	0x2	0x0	0x0F	0x13FF
88,200 Hz	0x2	0x0	0x11	0x0037
64,000 Hz	0x2	0x0	0x17	0x0E00
48,000 Hz	0x1	0x0	0x0F	0x13FF
44,100 Hz	0x1	0x0	0x11	0x0037

Table 29-1. Bit Field Values for Standard Sample Rates (continued)

Sample Rate	HW_audioout_dacsRr			
$Fsample_{DAC}$	BASEMULT	SRC_HOLD	SRC_INT	SRC_FRAC
32,000 Hz	0x1	0x0	0x17	0x0E00
24,000 Hz	0x1	0x1	0x0F	0x13FF
22,050 Hz	0x1	0x1	0x11	0x0037
16,000 Hz	0x1	0x1	0x17	0x0E00
12,000 Hz	0x1	0x3	0x0F	0x13FF
11,025 Hz	0x1	0x3	0x11	0x0037
8,000 Hz	0x1	0x3	0x17	0x0E00

NOTE: Sample-rates greater than 48 kHz can only be used when the AUDIOIN is disabled, and 44.1 kHz is the maximum sample rate at which both the AUDIOOUT and AUDIOIN can operate simultaneously.

For any of the desired sample rates, a fractional sample-rate conversion factor is calculated within the DIGFILT according to the following equation

$$SRConv_{DAC} = 65536 * [(Fanalog_{DAC})/ (8 * Hold_{DAC} * Fsample_{DAC})]$$

If computed with the above explicit operator precedence, the resulting sample-rate conversion factor ($SRConv_{DAC}$) will be a 24-bit scaled fixed-point representation of the desired decimation factor.

The 1-bit sigma delta D/A converter is always sampled on a submultiple of the 24.0-MHz crystal oscillator frequency, as specified in the HW_AUDIOOUT_ANACLKCTRL_DACDIV register (see Figure 29-3). This divider generates sample strobes at $Fanalog_{DAC}$ where the divisors available come from the set {4,6,8,12,16,24}. With HW_AUDIOOUT_ANACLKCTRL_DACDIV cleared to 0, to divide by 4, $Fanalog_{DAC}$ becomes 6.0 MHz for a 24.0-MHz crystal. The sample strobe derived from this divider is used to interpolate the 1-bit D/A values. The 1-bit sigma delta modulator is effectively running at $Fanalog_{DAC}$. As shown in Figure 29-3, the 16-bit or 32-bit D/A values are extracted from on-chip RAM via the DMA. They are filtered to band-limit the audio stream. This filter runs on xtal_clk, but filters samples at the source sample rate, which is slower than the output D/A sample rate. In the process, this filter performs a fixed 1:8 interpolation or up-sample input stream. A single 24-bit sample at the output of the fixed filter is further interpolated up to the 1-bit D/A rate. The variable rate sample, hold and interpolate block performs this function.

It stalls the filter pipeline and DMA source, using the handshake lines that connect with the previous filter stage to supply samples at the correct over-sample ratio. The 1-bit DAC runs at the fixed sample rate of $Fanalog_{DAC}$ while the DMA fetches samples in burst at irregular intervals to maintain the required input to the modulator.

In this case, the 1-bit D/A is running at 6 MHz. The sample hold and interpolate block accepts a new sample from the filter at a (44.1 kHz * 8) = 352.8 kHz. It passes interpolated samples to the modulator at a 6.0-MHz rate. The sample, hold and interpolate block passes a source sample from the fixed 1:8 interpo-

lation filter to the sigma delta modulator corresponding to every 8.503 $Fanalog_{DAC}$ samples. Recall that this is a variable rate interpolation stage that changes for every Over Sample Rate (OSR) setting in use.

If the desired sample rate $Fsample_{DAC}$= 44.1 kHz, for example, the sample hold and interpolate block will accept samples from fixed interpolation filter at 352.8 kHz, i.e., 8x the desired sample rate. There is a handshake pair (request/ack) between the variable rate sample hold and interpolate block and the fixed interpolating filter block. This handshake is used to pace the samples from the FIFO to 44.1 kHz.

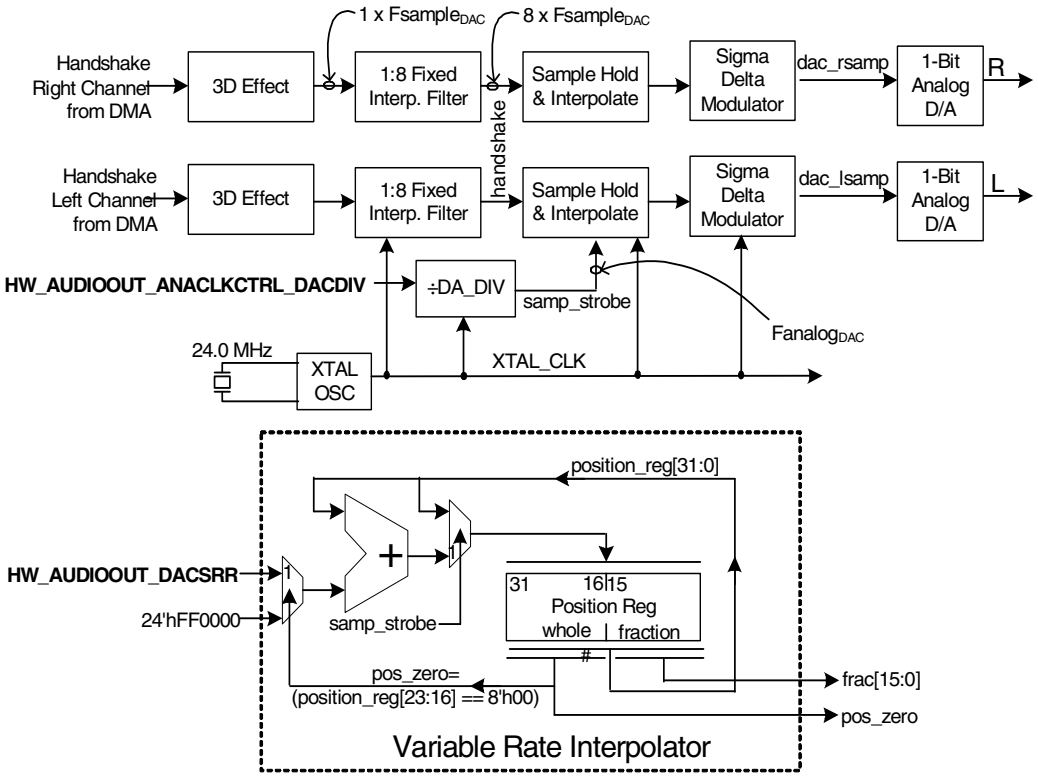


Figure 29-3. Stereo Sigma Delta D/A Converter

There are also members of the 48-kHz family whose members satisfy the property:

24.576 MHz = Q * $Fsample_{DAC}$

These sample rates include 48 kHz, 32 kHz, 24 kHz, 16 kHz, and 12 kHz.

There are also the members of the 44.1-kHz family whose members satisfy the property:

16.9344 MHz = Q * $Fsample_{ADC}$

where Q comes from the set of integers. These sample rates include 44.1 kHz, 22.05 kHz, and 11.025 kHz.

The D/A converter block includes a variable rate or rational interpolator to accommodate these sample rates, as shown in Figure 29-3. Rational numbers in the DAC are approximated with a scaled fixed-point 24-bit value. In this case, the decimal point falls between bit 15 and bit 16. Therefore, the lower two bytes

hold the fractional part, while the upper byte holds the whole number portion of the scaled fixed point. The position register uses this scaled fixed-point representation for the number of 1-bit samples to be interpolated (generated) to find the next sample to be sent to the sigma delta modulator. Whenever the whole number part (bits 23:16) is zero, then the next DMA sample is consumed. For playback at 44.1 kHz, set the interpolator to generate 67.027 new interpolated samples between every DMA sample.

29.2.3 Reference Control Settings

The reference control register allows adjustment of reference voltages and currents. VBG is the internal bandgap voltage (no external pin). This is a stable voltage reference used to establish *all* other voltage and current references for the chip, including VAG, vrefp, the Li-Ion charge termination voltage, etc. All the voltage and current references on the chip are proportional to VBG.

VAG is the analog ground voltage. It sets the DC bias on the input and output of all of the audio pins. This is typically set near VDDA/2. VAG also affects the peak output swing of the DAC. As VAG is lowered, the output swing of the DAC scales with it. However, at low VDDA levels, the analog performance can be improved by setting VAG somewhat below VDDA/2. The DAC is particularly susceptible to power supply noise if VDDA-VAG is not large enough.

29.2.4 Headphone

The i.MX23 supports a conventional stereo headphone drive, as shown in Figure 29-4. Headphone leads with an impedance of 16 Ω or greater are recommended.

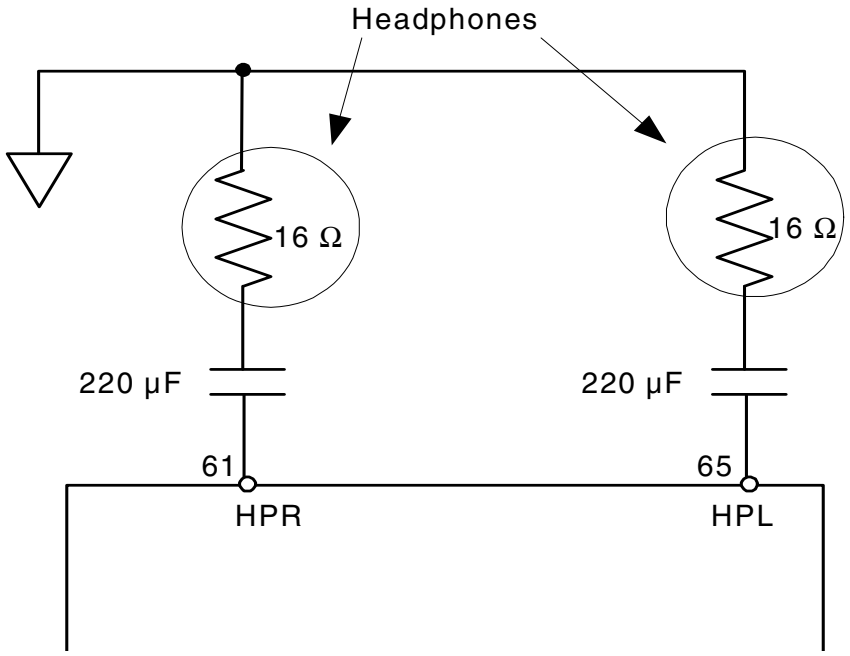


Figure 29-4. Conventional Stereo Headphone Application Circuit

In addition, as shown in Figure 29-5–Figure 29-7, the chip can generate an optional headphone common node circuit for the headphones that eliminates the need for the large and expensive DC blocking capacitors. It also improves the anti-pop performance. These benefits are obtained at a slight increase in power consumption, i.e., at 30 mV rms output, the resultant increase in power consumption is approximately 2.7 mW.

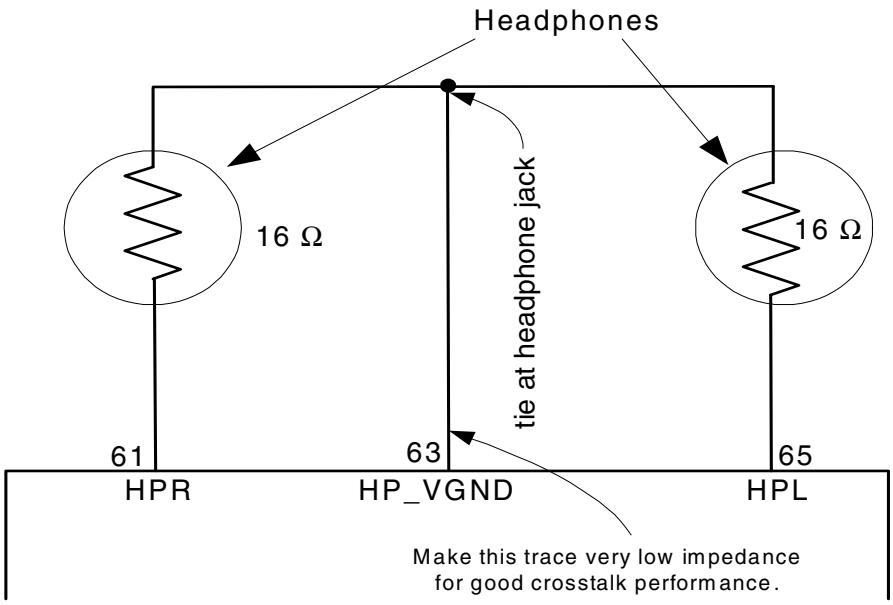


Figure 29-5. Stereo Headphone Application Circuit with Common Node

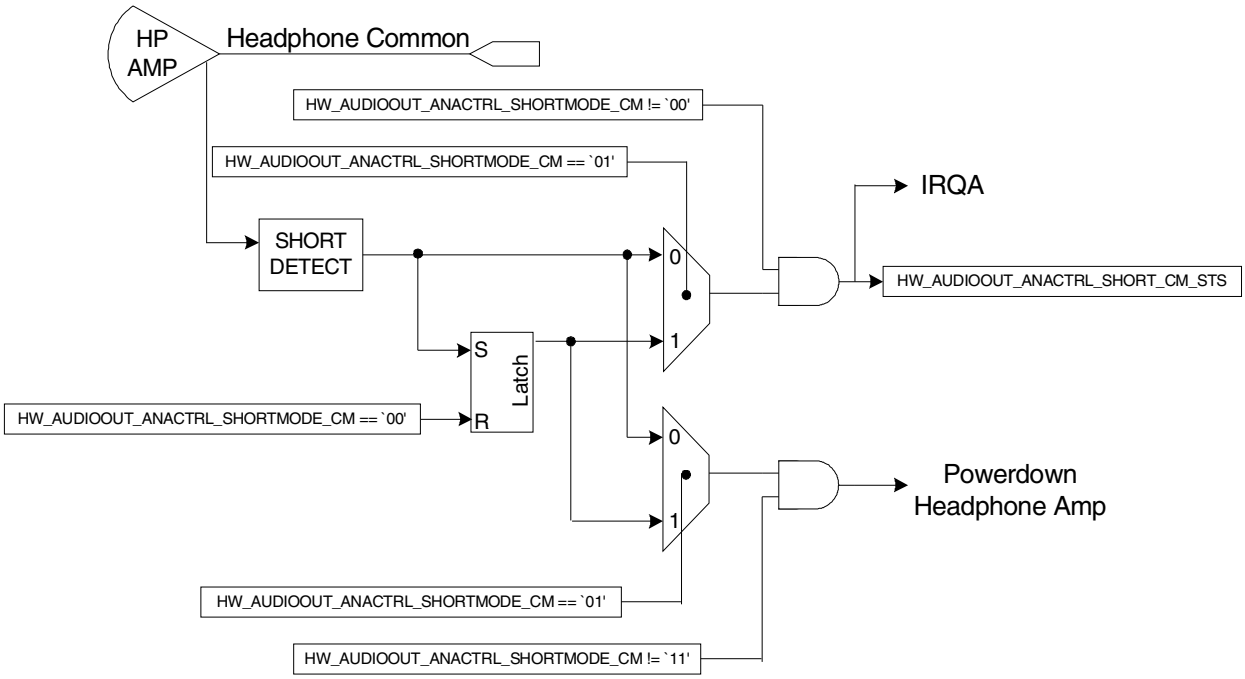


Figure 29-6. Stereo Headphone Common Short Detection and Powerdown Circuit

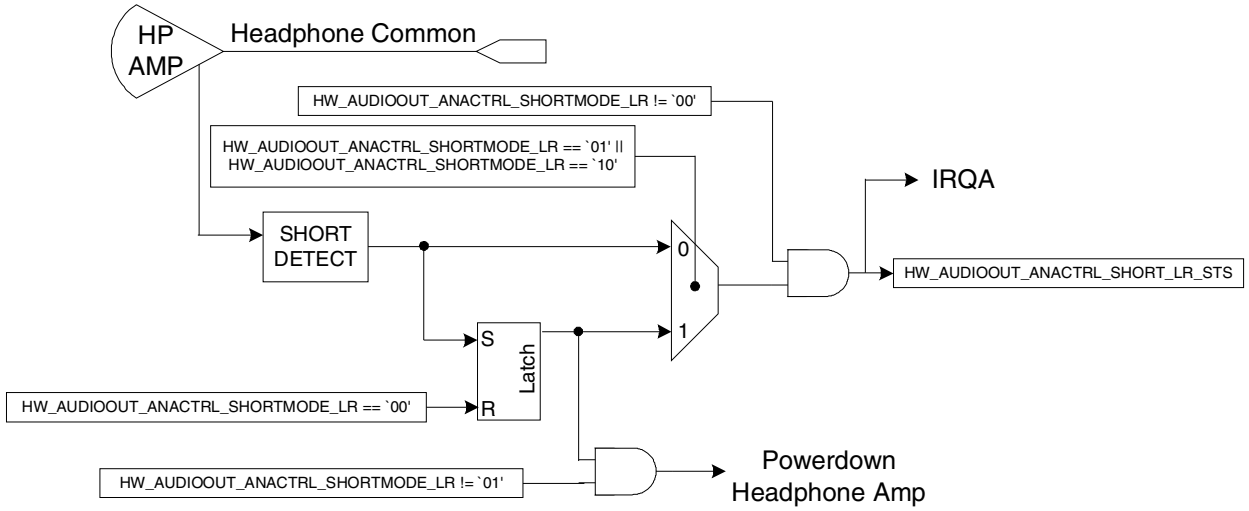


Figure 29-7. Stereo Headphone L/R Short Detection and Powerdown Circuit

29.2.4.1 Board Components

The headphone amplifier output requires a few external resistors and capacitors. There is also a 100Ω resistor to the headphone ground, which has multiple functions:

- In cap mode operation (a DC blocking cap is present), the 100Ω resistor improves startup pop suppression.

- In capless mode operation (no DC blocking cap), the resistor avoids 60-Hz hum into a powered set of speakers plugged into the headphone jack when the player is turned off.
- In capless and cap mode players, the 100Ω resistor minimizes the power-off signal feedthrough from line-in to headphone out.
- In powerdown mode, there is a 100kΩ resistor between line-in and headphone out. The 100Ω load resistor keeps the power-down feedthrough level at –60 dB. When the part is powered up, there is no signal path between line-in and headphone out, thus no bleedthrough.

29.2.4.2 Capless Mode Operation

The headphone amplifier is designed to work with or without a DC blocking cap. In capless mode, an amplifier is used to bias the headphone ground at the analog ground level (VAG), which is typically near VDDA/2. This avoids any DC signal across the output load.

Capless operation provides slight improvement to PSRR and low frequency performance. It slightly degrades SNR and THD (approximately 1 dB). The biggest advantage is the savings in cost and area by eliminating the large DC blocking caps. The biggest disadvantage is extra power consumption.

The capless mode of operation doubles the power consumed in the headphone amps. At normal listening levels, this has a fairly small effect on battery life. But with a full scale sine wave, it can significantly reduce the battery life.

With a sinusoidal signal, the headphone power consumption per channel with a 16Ω load is roughly:

Power = 1mW + Vpeak * VDDA / (16ohm * PI) (per channel)

This number is doubled in capless mode.

However, normal music files have a much higher peak-to-average ratio than a sinusoid. A PAR of 10 is typical in a music file, compared to a PAR of 1.414 for a sinusoid. So headphone power for a normal music file is:

Power = 1mW + Vpeak * VDDA * 2.8mW (per channel)

Again, this number is doubled in capless mode.

29.2.5 Speaker Amplifier

29.2.5.1 Overview

The i.MX23 provides an on-chip amplifier that can be used to drive an external speaker (see Figure 29-8). The external speaker must be bridge tied. Single ended (half-bridge) speakers are not supported. If ferrite beads are used they should have very low DC resistance and high saturation current to avoid causing power loss or distortion in the speaker.

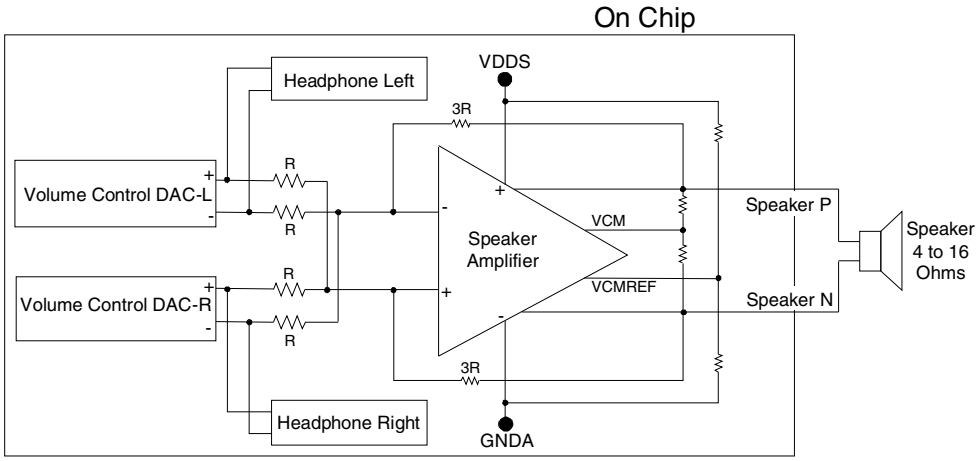


Figure 29-8. Speaker Amplifier with External Speaker

29.2.5.2 Details of Operations

The speaker driver is designed to handle a wide range of speaker impedances but it is assumed that the speaker will be in the range of 4 Ω to 16 Ω. The speaker driver can also tolerate multiple speakers in parallel. If the total speaker load is less than 4 Ω it will be necessary to limit the signal such that the peak current does not exceed 1.1 A. Care should be taken to minimize board trace impedances for the speaker outputs and for the power and ground. It is also necessary to minimize the audio band (20-20KHz) impedance of the speaker power supply to avoid reducing the maximum output power of the speaker.

Due to the differential output, the speaker can be powered up or down nearly instantaneously without any pop problems. This makes it easy to save power by shutting off the speaker amplifier when the speaker is idle.

The speaker amplifier does not have output short circuit protection. However, battery brown-out or VDD4P2 LDO current limit can act as a short circuit protection if those supplies are used to power the speaker amplifier.

The power supply for the speaker amplifier can be set to any level between 2.7 V and 4.2 V. The speaker amplifier power supply can be tied directly to a Li-Ion battery to avoid losses in DC-DC voltage conversion. There are no constraints on the speaker (VDDS) supply vs. the VDDIO supply. The speaker amplifier supply can be higher, lower, or tied to the VDDIO supply externally. When plugged into an external power source, that source must be able to provide the peak currents of the speaker amplifier or the speaker amplifier current draw could cause the power supply to brownout. If necessary, the peak currents can be reduced by lowering the maximum speaker volume. The on-chip 4.2 V regulator (used when an external supply is provided) can only supply a total of 780 mA peak current. If it is desired to use the maximum output power of the speaker amplifier when plugged into an external power source, it will be necessary to provide an external power supply for the speaker amplifier (2.7 - 4.2 V that doesn't pass through the on-chip 4.2 V regulator).

Volume control for the speaker is provided through the digital DAC volume. It is presumed that the headphone outputs will be disabled or muted while the speaker is driving. If they are not disabled, the digital DAC volume control will affect the headphone volume as well as the speaker. Additionally, the analog mix in the speaker amplifier will cause some reduced channel separation in the headphone. When the speaker is turned off, there is NO degradation of headphone performance. If the speaker amplifier is connected to the battery, it should be recognized that the maximum output swing without clipping will drop as the battery voltage drops. If desired, it is possible to monitor the battery level and reduce the maximum volume with the battery voltage level (to limit the amount of clipping).

The speaker amplifier mixes the right and left DAC outputs and provides a fixed gain of 6X (15.5dB). This high gain allows for the voltage shift from the low VDDA power rail to the speaker amplifier power rail and also allows the speaker outputs to be overdriven to provide more power output. The high peak-to-average ratio of most voice or music signals usually allows for 3-6dB of overdrive without substantial audio quality degradation (the clips happen rarely enough).

Maximum output power levels can reach 1.9 Watts with a 4 Ω load. The worst case on-chip power dissipation is expected to be 0.9 Watts (when using a 4.2 V rail and a 4 Ω load). The package thermal impedance is ~50 °C/Watt. So at the maximum output level the chip junction temperature can increase by up to 45 °C. This alone should not be a problem. But if the battery charger is also activated (another high dissipation circuit), use the on-chip temperature sensor to ensure that the temperature stays within an acceptable range.

29.3 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, “Correct Way to Soft Reset a Block,” for additional information on using the SFTRST and CLKGATE bit fields.

Note that the SFTRST and CLKGATE bits of both AUDIOOUT and AUDIOIN must be cleared before doing any operation with this block.

29.4 Programmable Registers

The following registers provide control for programmable elements of the AUDIOOUT/DAC block.

29.4.1 AUDIOOUT Control Register Description

The AUDIOOUT Control Register provides overall control of the digital portion of the digital-to-analog converter.

HW_AUDIOOUT_CTRL	0x000
HW_AUDIOOUT_CTRL_SET	0x004
HW_AUDIOOUT_CTRL_CLR	0x008

HW_AUDIOOUT_CTRL_TOG 0x00C

Table 29-2. HW_AUDIOOUT_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	RSRVD4									DMAWAIT_COUNT					RSRVD3	LR_SWAP	EDGE_SYNC	INVERT_1BIT	RSRVD2		SS3D_EFFECT		RSRVD1	WORD_LENGTH	DAC_ZERO_ENABLE	LOOPBACK	FIFO_UNDERFLOW_IRQ	FIFO_OVERFLOW_IRQ	FIFO_ERROR_IRQ_EN	RUN

Table 29-3. HW_AUDIOOUT_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	AUDIOOUT Module Soft Reset. Setting this bit to one forces a reset to portions of DIGFILT that control audio output and then gates the clocks off since the CLKGATE bit's reset state is to disable clocks. This bit must be cleared to zero for normal operation. Note that the CLKGATE bit does not affect SFTRST since it must remain writeable during clock gating. A note is included in the bit field descriptions below for those bits that are not affected by the AUDIOOUT's soft reset bit. These bits either control AUDIOIN or both AUDIOIN and AUDIOOUT functions.
30	**CLKGATE**	RW	0x1	AUDIOOUT Clock Gate Enable. When this bit is set to 1, it gates off the clocks to the portions of the DIGFILT block that control only output audio functions. It does not affect portions of the block that control AUDIOIN. Clear this bit to zero for normal AUDIOOUT operation. Note that when this bit is set, it remains writeable during clock gating so that it may be disabled by the user.
29:21	**RSRVD4**	RO	0x0	Reserved. Always write zeroes to this bit field.
20:16	**DMAWAIT_COUNT**	RW	0x0	DMA Request Delay Count. This bit field specifies the number of APBX clock cycles (0 to 31) to delay before each DMA request. This field acts as a throttle on the bandwidth consumed by the DIGFILT block. This field can be loaded by the DMA.
15	**RSRVD3**	RO	0x0	Reserved. Always write zeroes to this bit field.
14	**LR_SWAP**	RW	0x0	Left/Right Output Channel Swap Enable. Setting this bit to one swaps the left and right serial audio outputs from the SDM block to the analog DAC .
13	**EDGE_SYNC**	RW	0x0	Serial Output Clock Edge Sync Select. This bit selects the edge of the DAC's serial output clock upon which the SDM synchronizes for data transmit. 0=Rising edge. 1=Falling edge.

Table 29-3. HW_AUDIOOUT_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
12	**INVERT_1BIT**	RW	0x0	Invert Serial Audio Output Enable. When set, this bit inverts the 1-bit serial output of both the left and right channels to the DAC's sigma-delta modulator. 0=Normal operation. 1=Invert L/R serial audio output from the SDM block.
11:10	**RSRVD2**	RO	0x0	Reserved. Always write zeroes to this bit field.
9:8	**SS3D_EFFECT**	RW	0x0	Virtual 3D Effect Enable. This bit field provides a virtual 3D effect for a two channel system by subtracting a portion of the opposite channel's content for each channel. Three reduction ratios are available (dB value represents amount of opposite channel content subtracted). 00=Off 01=Low (3 dB) 10=Medium (4.5 dB) 11=High (6 dB)
7	**RSRVD1**	RO	0x0	Reserved. Always write zeroes to this bit field.
6	**WORD_LENGTH**	RW	0x0	PCM Audio Bit Size Select. This bit selects the size of the parallel PCM data that is input to the AUDIOOUT's FIFO. 0=32-bit PCM samples. 1=16-bit samples. Note that the PCM audio data input to the FIR filter stages is 24 bit. For 16-bit operation, the data is first sign-extended to 24 bits. For 32-bit operation, the data is first normalized by dropping the least significant 8 bits.

Table 29-3. HW_AUDIOOUT_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
5	**DAC_ZERO_ENABLE**	RW	0x0	Never set DAC_ZERO_ENABLE! Make sure it is always cleared. (Since this bit is clear on reset, you'll be fine if you never touch it at all.) If you want to silence the audio output, but not go into a lower power mode, set HW_AUDIOOUT_DACVOLUME.MUTE_LEFT and HW_AUDIOOUT_DACVOLUME.MUTE_RIGHT simultaneously. This will immediately mute the digital audio signal sent to the DAC. There are two important things to note: 1. These bits ignore zero-crossing detection. They immediately mute the digital audio signal, so this can cause a "pop." The "pop" can be avoided by ramping the volume. 2. These bits cannot be toggled when clock gated. If you want to minimize power consumption, mute the amplifiers first, and then power down the DAC. Reverse these operations to come back out of the low power state. The amplifiers will hold the output signal to VAG so everything will be quiet, and the DAC can power back up very quickly when you have to start playing sound again. To enter the low-power state: 1. Set HW_AUDIOOUT_HPVOL.MUTE and HW_AUDIOOUT_LINEOUTCTRL.MUTE to mute the Headphone and Line Out amplifiers, respectively. 2. Set HW_AUDIOOUT_PWRDN.DAC to power down the DAC. To exit the low-power state: 1. Clear HW_AUDIOOUT_PWRDN.DAC to power up the DAC. 2. Clear HW_AUDIOOUT_HPVOL.MUTE and HW_AUDIOOUT_LINEOUTCTRL.MUTE to un-mute the Headphone and Line Out amplifiers, respectively.
4	**LOOPBACK**	RW	0x0	AUDIOIN-to-AUDIOOUT Loopback Enable. Setting this bit to one routes the audio data received by the AUDIOIN's FIFO to the AUDIOOUT's FIFO. This test mode provides a loopback that does not use the DIGFILT's DMA memory inteface. This bit should be cleared to zero for normal operation.
3	**FIFO_UNDERFLOW_IRQ**	RW	0x0	FIFO Underflow Interrupt Status Bit. This bit is set by hardware if the AUDIOOUT's FIFO underflows any time during operation due to a DMA request that is not serviced in time. It is reset by software by writing a one to the corresponding bit in the HW_AUDIOOUT_CTRL_CLR register. An interrupt is issued to the host processor if this bit is set and FIFO_ERROR_IRQ_EN=1.

Table 29-3. HW_AUDIOOUT_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**FIFO_OVERFLOW_IRQ**	RW	0x0	FIFO Overflow Interrupt Status Bit. This bit is set by hardware if the AUDIOOUT's FIFO overflows. It is reset by software by writing a one to the corresponding bit in the HW_AUDIOOUT_CTRL_CLR register. An interrupt is issued to the host processor if this bit is set, and FIFO_ERROR_IRQ_EN=1. Note that overflows should not occur by design since requests to the DMA are not made unless there is adequate space present within the FIFO. Therefore, this condition would indicate a serious DMA error.
1	**FIFO_ERROR_IRQ_EN**	RW	0x0	FIFO Error Interrupt Enable. Set this bit to one to enable an AUDIOOUT interrupt request to the host processor when either the FIFO overflow or underflow status bits are set. Note that this bit does not affect the state of the underflow/overflow status bits, but rather their ability to signal an interrupt to the CPU.
0	**RUN**	RW	0x0	AUDIOOUT Enable. Setting this bit to one causes AUDIOOUT operation to start, beginning with a DMA request to fill its 8-word FIFO. Clearing this bit to zero stops data conversion and also causes the CLKGATE bit to be set.

DESCRIPTION:

The AUDIOOUT Control Register contains bit fields used to control and monitor AUDIOOUT operation including: reset, clocks, DMA transfers, data to the analog DAC module, PCM data size, test, and interrupt control.

EXAMPLE:

```
HW_AUDIOOUT_CTRL.RUN = 1; // start DAC conversion
```

29.4.2 AUDIOOUT Status Register Description

The AUDIOOUT Status Register is used to determine if the digital-to-analog converter is operational.

HW_AUDIOOUT_STAT	0x010
HW_AUDIOOUT_STAT_SET	0x014
HW_AUDIOOUT_STAT_CLR	0x018
HW_AUDIOOUT_STAT_TOG	0x01C

Table 29-4. HW_AUDIOOUT_STAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DAC_PRESENT	RSRVD1																														

Table 29-5. HW_AUDIOOUT_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**DAC_PRESENT**	RO	0x1	AUDIOOUT Functionality Present. This status bit is set to one in products that include the AUDIOOUT/DAC. If this bit is zero, the AUDIOOUT/DAC is permanently disabled and cannot be operated by the user.
30:0	**RSRVD1**	RO	0x0	Reserved. Always write zeroes to this bit field.

DESCRIPTION:

The AUDIOOUT Status Register provides an indication of the presence of the DAC functionality.

EXAMPLE:

```
unsigned statusValue = HW_AUDIOOUT_STAT.DAC_PRESENT;
```

29.4.3 AUDIOOUT Sample Rate Register Description

The AUDIOOUT Sample Rate Register is used to specify the sample rate that the parallel PCM audio data is converted to within the SDM module before being output to the analog DAC.

HW_AUDIOOUT_DACSRR	0x020
HW_AUDIOOUT_DACSRR_SET	0x024
HW_AUDIOOUT_DACSRR_CLR	0x028
HW_AUDIOOUT_DACSRR_TOG	0x02C

Table 29-6. HW_AUDIOOUT_DACSRR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
OSR	BASEMULT			RSRVD2	SRC_HOLD			RSRVD1			SRC_INT					RSRVD0			SRC_FRAC												

Table 29-7. HW_AUDIOOUT_DACSRR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**OSR**	RO	0x0	AUDIOOUT Oversample Rate. Note that the oversample rate is fixed at 6 MHz. OSR6 = 0x0 AUDIOOUT oversample rate at 6 MHz. OSR12 = 0x1 AUDIOOUT oversample rate at 12 MHz.
30:28	**BASEMULT**	RW	0x1	Base Sample Rate Multiplier. This bit field is used to configure the DAC's sample rate to one of three ranges: single, double, or quad. This multiply factor is used to achieve sample rates greater than the standard rates of 32/44.1/48 kHz. A value of 0x1 should be used when selecting half and quarter sample rates. Note that sample rates greater than 48 kHz may only be used when the AUDIOIN is disabled, and 44.1 kHz is the maximum sample rate at which both the AUDIOIN and AUDIOOUT can operate simultaneously. SINGLE_RATE = 0x1 Single rate multiplier (for 48/44.1/32 kHz as well as half and quarter rates). DOUBLE_RATE = 0x2 Double rate multiplier (for 96/88.2/64 kHz). QUAD_RATE = 0x4 Quad rate multiplier (for 192/176.4/128 kHz).
27	**RSRVD2**	RO	0x0	Reserved. Always write a zero to this bit field.
26:24	**SRC_HOLD**	RW	0x0	Sample Rate Conversion Hold Factor. This bit is used to hold a sample of a variable number of clock cycles in order to generate half and quarter sample rates when dividing down the AUDIOOUT's internal rate using the equation: output_sample_rate = (6x10^6 * BASEMULT) / (SRC_INT.SRC_FRAC * 8 * (SRC_HOLD + 1)). Refer to the sample rate table earlier in this chapter that provides a list of bit field values required to achieve all common sample rates.
23:21	**RSRVD1**	RO	0x0	Reserved. Always write zeros to this bit field.
20:16	**SRC_INT**	RW	0x11	Sample Rate Conversion Integer Factor. This bit field is the integer portion of a divide term used to sample rate convert the AUDIOOUT's internal rate using the equation: output_sample_rate = (6x10^6 * BASEMULT) / (SRC_INT.SRC_FRAC * 8 * (SRC_HOLD + 1)). Refer to the sample rate table earlier in this chapter that provides a list of bit field values required to achieve all common sample rates.
15:13	**RSRVD0**	RO	0x0	Reserved. Always write zeros to this bit field.
12:0	**SRC_FRAC**	RW	0x37	Sample Rate Conversion Fraction Factor. This bit field is the fractional portion of a divide term used to sample rate convert the AUDIOOUT's internal rate using the equation: output_sample_rate = (6x10^6 * BASEMULT) / (SRC_INT.SRC_FRAC * 8 * (SRC_HOLD + 1)). Refer to the sample rate table earlier in this chapter that provides a list of bit field values required to achieve all common sample rates.

DESCRIPTION:

The AUDIOOUT Sample Rate Register provides a number of bit fields to direct the SDM module's hardware to sample-rate-convert the audio stream output to one of a number of common sample rates. Note that these values can also be dynamically altered in small amounts in order to track variations in the outgoing audio stream from sources such as FM digital radio.

EXAMPLE:

```
// Program the DAC to output a sample rate of 48 kHz:
HW_AUDIOOUT_DACSRR.BASEMULT = 0x1; // quad-rate
HW_AUDIOOUT_DACSRR.SRC_HOLD = 0x0; // 0 for full- double- quad-rates
HW_AUDIOOUT_DACSRR.SRC_INT = 0xF; // 15 for the integer portion
HW_AUDIOOUT_DACSRR.SRC_FRAC = 0x13FF; // the fractional portion
```

29.4.4 AUDIOOUT Volume Register Description

The AUDIOOUT Volume Register is used to adjust the signal level of the playback audio output to the DAC.

HW_AUDIOOUT_DACVOLUME	0x030
HW_AUDIOOUT_DACVOLUME_SET	0x034
HW_AUDIOOUT_DACVOLUME_CLR	0x038
HW_AUDIOOUT_DACVOLUME_TOG	0x03C

Table 29-8. HW_AUDIOOUT_DACVOLUME

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD4			VOLUME_UPDATE_LEFT	RSRVD3		EN_ZCD	MUTE_LEFT	VOLUME_LEFT								RSRVD2			VOLUME_UPDATE_RIGHT	RSRVD1			MUTE_RIGHT	VOLUME_RIGHT							

Table 29-9. HW_AUDIOOUT_DACVOLUME Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSRVD4**	RO	0x00	Reserved. Always write zeroes to this bit field.
28	**VOLUME_UPDATE_LEFT**	RO	0x0	Left Channel Volume Update Pending. This bit is set to one by the hardware when an AUDIOOUT volume update is pending, i.e., waiting on a zero crossing on the left channel. The bit is set following a write to the VOLUME_LEFT bit field and is cleared when the attenuation value is applied to the PCM output stream (at a zero-crossing). This status bit is not used when EN_ZCD=0.
27:26	**RSRVD3**	RO	0x00	Reserved. Always write zeroes to this bit field.
25	**EN_ZCD**	RW	0x0	Enable Zero Cross Detect. This bit enables/disables use of the zero cross detect circuit in the DAC (rather than enabling the circuit itself). When enabled, changes to the volume bit fields are not applied until it is detected that the output signal's sign bit toggles (crosses zero amplitude). When disabled, changes to the volume bit fields take effect immediately when written.

Table 29-9. HW_AUDIOOUT_DACVOLUME Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
24	**MUTE_LEFT**	RW	0x1	Mute Left Channel. 0=unmute, 1=mute. Note that mute is always applied immediately when written (unlike volume when EN_ZCD=1). The channel volume should always be ramped down to the minimum level (-100dB) before setting the mute bit.
23:16	**VOLUME_LEFT**	RW	0xfe	Left Channel Volume Setting. Establishes the outgoing audio signal strength during playback. Volume ranges from -0.5 dB (0xFE) to -100 dB (0x37). Each increment of this bit field causes a half-dB increase in volume. The values 0x00-0x37 all produce the same attenuation level of -100 dB. Note: Do not change the DAC volume into or from the 0xFF state or else it will create a large pop. Remaining in the 0xFF state is acceptable if all of the volume control is done in the headphone.
15:13	**RSRVD2**	RO	0x00	Reserved. Always write zeroes to this bit field.
12	**VOLUME_UPDATE_RIGHT**	RO	0x0	Right Channel Volume Update Pending. This bit is set to one by the hardware when an AUDIOOUT volume update is pending, i.e., waiting on a zero crossing on the right channel. The bit is set following a write to the VOLUME_RIGHT bit field and is cleared when the attenuation value is applied to the PCM output stream (at a zero-crossing). This status bit is not used when EN_ZCD=0.
11:9	**RSRVD1**	RO	0x00	Reserved. Always write zeroes to this bit field.
8	**MUTE_RIGHT**	RW	0x1	Mute Right Channel. 0=Unmute, 1=Mute. Note that mute is always applied immediately when written (unlike volume when EN_ZCD=1), therefore the user should always ramp down the channel's volume to the minimum level (-100 dB) before setting the mute bit.
7:0	**VOLUME_RIGHT**	RW	0xfe	Right Channel Volume Setting. Establishes the outgoing audio signal strength during playback. Volume ranges from -0.5 dB (0xFE) to -100 dB (0x37). Each increment of this bit field causes a half-dB increase in volume. The values 0x00-0x37 all produce the same attenuation level of -100 dB. Note: Do not change the DAC volume into or from the 0xFF state or else it will create a large pop. Remaining in the 0xFF state is acceptable if all of the volume control is done in the headphone.

DESCRIPTION:

The AUDIOOUT Volume Register allows independent volume and mute control of the left and right channels. Output audio can be attenuated in 0.5-dB steps, from full-scale down to a minimum of -100 dB. This register is also used to enable/control volume updates such that they are only applied when PCM values cross zero to prevent unwanted audio artifacts.

EXAMPLE:

```
HW_AUDIOOUT_DACVOLUME.U = 0x01ff01ff; // mute both left and right channels
HW_AUDIOOUT_DACVOLUME.U = 0x00ff00ff; // maximum volume for left and right channels.
```

29.4.5 AUDIOOUT Debug Register Description

The AUDIOOUT Debug Register is used for test and debug of the AUDIOOUT block.

HW_AUDIOOUT_DACDEBUG	0x040
HW_AUDIOOUT_DACDEBUG_SET	0x044
HW_AUDIOOUT_DACDEBUG_CLR	0x048
HW_AUDIOOUT_DACDEBUG_TOG	0x04C

Table 29-10. HW_AUDIOOUT_DACDEBUG

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ENABLE_DACDMA	RSRVD2																			RAM_SS				RSRVD1		SET_INTERRUPT1_CLK_CROSS	SET_INTERRUPT0_CLK_CROSS	SET_INTERRUPT1_HAND_SHAKE	SET_INTERRUPT0_HAND_SHAKE	DMA_PREQ	FIFO_STATUS

Table 29-11. HW_AUDIOOUT_DACDEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**ENABLE_DACDMA**	RW	0x0	AUDIOOUT Digital Path Test Enable. This bit is used solely for development and debugging, and is not functional on production parts. When enabled, it causes the AUDIOIOUT's serial audio data output to bypass the DAC analog block, to be assembled into 32-bit words and transferred out to memory using the AUDIOIN's DMA Channel 0. Unlike loopback, this test mode provides a means of verifying the digital portion of the AUDIOOOUT logic without causing the audio data to pass through the AUDIOIN's FIR filter stages.
30:12	**RSRVD2**	RO	0x00	Reserved. Always write zeroes to this bit field.
11:8	**RAM_SS**	RW	0x0	DIGFILT RAM Speed Select. Sense AMP Timing Control for DIGFILT 208x24 RAM. Do not alter unless instructed by Freescale.
7:6	**RSRVD1**	RO	0x0	Reserved. Always write zeroes to this bit field.
5	**SET_INTERRUPT1_CLK_CROSS**	RO	0x0	Interrupt[1] Sync Status. This bit reflects the current state of the second flop on the chain of three flip-flops used to synchronize the AUDIOOUT's interrupt[1] signal used to prioritize channels 0 and 1 DMA requests from the DIGFILT. This signal is synchronized from the module's internal 24-MHz clock to the APBX's memory clock domain. This bit is intended for test only.

Table 29-11. HW_AUDIOOUT_DACDEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
4	**SET_INTERRUPT0_CLK_CROSS**	RO	0x0	Interrupt[0] Sync Status. This bit reflects the current state of the second flop on the chain of three flip-flops used to synchronize the AUDIOOUT's interrupt[0] signal used to prioritize channel 0 and 1 DMA requests from the DIGFILT. This signal is synchronized from the module's internal 24-MHz clock to the APBX's memory clock domain. This bit is intended for test only.
3	**SET_INTERRUPT1_HAND_SHAKE**	RO	0x0	Interrupt[1] Status. This bit reflects the current state of the APBX interface state machine's internal interrupt[1] signal used to prioritize channel 0 and 1 DMA requests from the DIGFILT. This bit is intended for test only.
2	**SET_INTERRUPT0_HAND_SHAKE**	RO	0x0	Interrupt[0] Status. This bit reflects the current state of the APBX interface state machine's internal interrupt[0] signal used to prioritize channel 0 and 1 DMA requests from the DIGFILT. This bit is intended for test only.
1	**DMA_PREQ**	RO	0x0	DMA Request Status. This bit reflects the current state of the AUDIOOUT's DMA request signal. DMA requests are issued any time the request signal toggles. This bit can be polled by software, in order to manually move samples to the AUDIOOUT's FIFO from a memory buffer when the AUDIOOUT's DMA channel is not used.
0	**FIFO_STATUS**	RO	0x1	FIFO Status. This bit is set by hardware when the AUDIOOUT's FIFO contains any empty entries and is cleared when the FIFO is full.

DESCRIPTION:

The AUDIOOUT Debug Register provides read-only access of various internal AUDIOOUT module signals to assist in debug and validation, as well as control of DACDMA test mode.

EXAMPLE:

```
unsigned tempStatus = HW_AUDIOOUT_DACDEBUG.FIFO_STATUS;
```

29.4.6 Headphone Volume and Select Control Register Description

The Headphone Volume and Select Control Register provides volume, mute, and input select controls for the headphone.

HW_AUDIOOUT_HPVOL	0x050
HW_AUDIOOUT_HPVOL_SET	0x054
HW_AUDIOOUT_HPVOL_CLR	0x058
HW_AUDIOOUT_HPVOL_TOG	0x05C

Table 29-12. HW_AUDIOOUT_HPVOL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD5			VOLUME_UPDATE_PENDING	RSRVD4		EN_MSTR_ZCD	MUTE	RSRVD3							SELECT	RSRVD2	VOL_LEFT							RSRVD1	VOL_RIGHT						

Table 29-13. HW_AUDIOOUT_HPVOL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSRVD5**	RO	0x0	Reserved. Always write zeroes to this bit field.
28	**VOLUME_UPDATE_PENDING**	RO	0x0	Volume Update Pending. This bit is set to one by the hardware when either a headphone left or right volume update is pending, i.e., waiting on a zero crossing. The bit is set following a write to either the VOL_LEFT or VOL_RIGHT bit fields and is cleared when both attenuation values are applied to the output coincident with zero-crossings in both the right and left channels. This status bit is not used when EN_MSTR_ZCD=0.
27:26	**RSRVD4**	RO	0x0	Reserved. Always write zeroes to this bit field.
25	**EN_MSTR_ZCD**	RW	0x0	Enable Zero Cross Detect for Headphone Amplifier.
24	**MUTE**	RW	0x1	Headphone Mute. It is reset by a power-on reset only.
23:17	**RSRVD3**	RO	0x00	Reserved. Always write zeroes to this bit field.
16	**SELECT**	RW	0x0	Input Signal Select. 0=DAC, 1=Line1. It is reset by a power-on reset only.
15	**RSRVD2**	RO	0x0	Reserved. Always write zeroes to this bit field.
14:8	**VOL_LEFT**	RW	0x0C	Left Headphone Volume Control. 0.5 dB volume steps. Volume range depends on HW_AUDIOOUT_HPVOL_SELECT setting. In DAC mode 0x00=+6dB and 0x7F=-57.5dB. In Line1 mode 0x00=+12dB and 0x7F=-51.5dB. Reset state is DAC mode 0x0C=0dB. It is reset by a power-on reset only.
7	**RSRVD1**	RO	0x0	Reserved. Always write zeroes to this bit field.
6:0	**VOL_RIGHT**	RW	0x0C	Right Headphone Volume Control. 0.5 dB volume steps. Volume range depends on HW_AUDIOOUT_HPVOL_SELECT setting. In DAC mode 0x00=+6dB and 0x7F=-57.5dB. In Line1 mode 0x00=+12dB and 0x7F=-51.5dB. Reset state is DAC mode 0x0C=0dB. It is reset by a power-on reset only.

DESCRIPTION:

This register provides volume, mute, and input select controls for the headphone.

EXAMPLE:

```
HW_AUDIOOUT_HPVOL.MUTE = 0;
```

29.4.7 Reserved Register Description

This register is reserved.

HW_AUDIOOUT_RESERVED	0x060
HW_AUDIOOUT_RESERVED_SET	0x064
HW_AUDIOOUT_RESERVED_CLR	0x068
HW_AUDIOOUT_RESERVED_TOG	0x06C

Table 29-14. HW_AUDIOOUT_RESERVED

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																															

Table 29-15. HW_AUDIOOUT_RESERVED Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**RSRVD1**	RO	0x00000000	Reserved. Always write zeroes to this bit field.

DESCRIPTION:

This register is reserved.

EXAMPLE:

```
EMPTY
```

29.4.8 Audio Power-Down Control Register Description

The Audio Power-Down Control Register provides all power-down control bits.

HW_AUDIOOUT_PWRDN	0x070
HW_AUDIOOUT_PWRDN_SET	0x074
HW_AUDIOOUT_PWRDN_CLR	0x078
HW_AUDIOOUT_PWRDN_TOG	0x07C

Table 29-16. HW_AUDIOOUT_PWRDN

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD7							SPEAKER	RSRVD6			SELFBIAS	RSRVD5			RIGHT_ADC	RSRVD4			DAC	RSRVD3			ADC	RSRVD2			CAPLESS	RSRVD1			HEADPHONE

Table 29-17. HW_AUDIOOUT_PWRDN Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSRVD7**	RO	0x00	Reserved. Always write zeroes to this bit field.
24	**SPEAKER**	RW	0x1	Speaker Power-Down. It is reset by a power-on reset only.
23:21	**RSRVD6**	RO	0x00	Reserved. Always write zeroes to this bit field.
20	**SELFBIAS**	RW	0x0	This bit currently does not control any logic, altering it's value has no effect.
19:17	**RSRVD5**	RO	0x00	Reserved. Always write zeroes to this bit field.
16	**RIGHT_ADC**	RW	0x0	Right ADC Power Down. When enabled, powers down the ADC's right channel while allowing the left to function normally (mono). Note that while this bit is located in the AUDIOOUT address space, it is an ADC function controlled by the AUDIOIN's SFTRST and CLKGATE bits.
15:13	**RSRVD4**	RO	0x00	Reserved. Always write zeroes to this bit field.
12	**DAC**	RW	0x1	Power Down DAC Analog Circuitry. It is reset by a power-on reset only.
11:9	**RSRVD3**	RO	0x00	Reserved. Always write zeroes to this bit field.
8	**ADC**	RW	0x1	Power Down ADC and Input Mux Circuitry. 1=power down, 0=power up. Before clearing this bit you must set rtc_persistent0_release_gnd. Note that while this bit is located in the AUDIOOUT address space, it is an ADC function controlled by the AUDIOIN's SFTRST and CLKGATE bits.
7:5	**RSRVD2**	RO	0x00	Reserved. Always write zeroes to this bit field.
4	**CAPLESS**	RW	0x1	Power Down Headphone Common Amplifier Used in Capless Headphone. It is reset by a power-on reset only.
3:1	**RSRVD1**	RO	0x00	Reserved. Always write zeroes to this bit field.
0	**HEADPHONE**	RW	0x1	Master (Headphone) Power Down. 1=power down, 0=power up. Before clearing this bit you must set rtc_persistent0_release_gnd bit otherwise there will be large current draw in the vag buffer amp.¬† Before setting the persistent bit during powerup to avoid antipop you should set audioout_anactrl_hp_hold_gnd this will hold the headphone outputs at ground. It is reset by a power-on reset only.

DESCRIPTION:

The Audio Power-Down Register provides control to power-down sections of the audio analog circuit.

EXAMPLE:

```
HW_AUDIOOUT_PWRDN.DAC = 0;
```

29.4.9 AUDIOOUT Reference Control Register Description

This register provides the voltage and current reference control bits.

HW_AUDIOOUT_REFCTRL	0x080
HW_AUDIOOUT_REFCTRL_SET	0x084
HW_AUDIOOUT_REFCTRL_CLR	0x088
HW_AUDIOOUT_REFCTRL_TOG	0x08C

Table 29-18. HW_AUDIOOUT_REFCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD4					FASTSETTLING	RAISE_REF	XTAL_BGR_BIAS	RSRVD3	VBG_ADJ			LOW_PWR	LW_REF	BIAS_CTRL		RSRVD2	VDDXTAL_TO_VDDD	ADJ_ADC	ADJ_VAG	ADC_REFVAL				VAG_VAL				RSRVD1	DAC_ADJ		

Table 29-19. HW_AUDIOOUT_REFCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:27	**RSRVD4**	RO	0x00	Reserved. Always write zeroes to this bit field.
26	**FASTSETTLING**	RW	0x0	Increases the output current for vag buffer by 2X to 20uA to improve the startup settling time. Startup settling time is roughly vag_cap*vag_value/current. Note that while this bit is located in the AUDIOOUT address space, since it controls both DAC and ADC functions, it is not reset by the AUDIOOUT's SFTRST bit. It is reset by a power-on reset only, and CLKGATE has no effect on reads/writes.
25	**RAISE_REF**	RW	0x0	This must be low when VDDA is less than ~1.7V. When it is low the VAG and ADC reference voltages all drop by 1.4V/1.6V=0.875. Note that while this bit is located in the AUDIOOUT address space, since it controls both DAC and ADC functions, it is not reset by the AUDIOOUT's SFTRST bit. It is reset by a power-on reset only, and CLKGATE has no effect on reads/writes.
24	**XTAL_BGR_BIAS**	RW	0x0	Switch the XTAL bias from self-bias to bandgap-based bias current. Also switches the source of the XTAL supply in series AA or LiIon to core supply to save power. Note that while this bit is located in the AUDIOOUT address space, since it controls both DAC and ADC functions, it is not reset by the AUDIOOUT's SFTRST bit. It is reset by a power-on reset only, and CLKGATE has no effect on reads/writes.
23	**RSRVD3**	RO	0x0	Reserved. Always write zeroes to this bit field.
22:20	**VBG_ADJ**	RW	0x0	Small adjustment for VBG value. Will affect ALL reference voltages. Expected to be used to tweak final Li-Ion charge voltage. 000=Nominal. 001=+0.3%. 010=+0.6%. 011=0.85%. 100=-0.3%. 101=-0.6%. 110=-0.9%. 111=-1.2%. Note that while this bit is located in the AUDIOOUT address space, since it controls both DAC and ADC functions, it is not reset by the AUDIOOUT's SFTRST bit. It is reset by a power-on reset only, and CLKGATE has no effect on reads/writes.

Table 29-19. HW_AUDIOOUT_REFCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19	**LOW_PWR**	RW	0x0	Lowers power (~100 uA) in the bandgap amplifier. This mode is useful in USB suspend or standby when bandgap accuracy is not critical. Note that while this bit is located in the AUDIOOUT address space, since it controls both DAC and ADC functions, it is not reset by the AUDIOOUT's SFTRST bit. It is reset by a power-on reset only, and CLKGATE has no effect on reads/writes.
18	**LW_REF**	RW	0x0	This bit currently does not control any logic, altering it's value has no effect. It is reset by power-on reset, and CLKGATE has no effect on reads/writes.
17:16	**BIAS_CTRL**	RW	0x0	Bias current control for all analog blocks: 00=Nominal. 01=-20%. 10=-10%. 11=+10%. Note that while this bit is located in the AUDIOOUT address space, since it controls both DAC and ADC functions, it is not reset by the AUDIOOUT's SFTRST bit. It is reset by a power-on reset only, and CLKGATE has no effect on reads/writes. These bits should only be used for test/debug, and not in an application.
15	**RSRVD2**	RO	0x0	Reserved. Always write zeroes to this bit field.
14	**VDDXTAL_TO_VDDD**	RW	0x0	Shorts the supply of the XTAL oscillator to VDDD. This bit may be used to reduce power consumption, but should only be used on the advice of Freescale. Note that while this bit is located in the AUDIOOUT address space, it is an ADC function controlled by the AUDIOIN's SFTRST and CLKGATE bits.
13	**ADJ_ADC**	RW	0x0	ADC Reference Voltage Adjust. When set the bias current in the ADC drop by 20% to save power. Note that while this bit is located in the AUDIOOUT address space, it is an ADC function controlled by the AUDIOIN's SFTRST and CLKGATE bits.
12	**ADJ_VAG**	RW	0x0	When cleared, VAG is VDD/2 (resistor divider). When set, VAG is controlled by VAGVAL 7:4. Note that while this bit is located in the AUDIOOUT address space, since it controls both DAC and ADC functions, it is not reset by the AUDIOOUT SFTRST bit. It is reset by a power-on reset only, and CLKGATE has no effect on reads/writes.
11:8	**ADC_REFVAL**	RW	0x0	ADC Reference Value. These bits set the reference voltage for the ADC as a ratio of the VAG voltage. 0x0=1.75*VAG, 0x1=1.85*VAG, 0x2=1.97*VAG. Codes 0x3-0xF are invalid. Note that while this bit is located in the AUDIOOUT address space, it is an ADC function controlled by the AUDIOIN's SFTRST and CLKGATE bits.

Table 29-19. HW_AUDIOOUT_REFCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
7:4	**VAG_VAL**	RW	0x0	VAG Reference Value (when ADJVAG set): MSB is NOT used for voltage control. 0x7=1.0 V, 0x0=0.825 V, 25-mV steps, also affected by RAISE_REF. If RAISE_REF=0, 0x7=0.875, 0x0=0.722 V. If the MSB is high the DAC reference buffer is bypassed which can improve SNR performance. See section on selecting the VAG level earlier in this chapter. Note that while this bit is located in the AUDIOOUT address space, since it controls both DAC and ADC functions, it is not reset by the AUDIOOUT's SFTRST bit. It is reset by a power-on reset only, and CLKGATE has no effect on reads/writes.
3	**RSRVD1**	RO	0x0	Reserved. Always write zeroes to this bit field.
2:0	**DAC_ADJ**	RW	0x0	Adjusts the reference current (signal swing) in the DAC : 000=Nominal. 001=+0.25 dB. 010=+0.5 dB. 011=+0.75 dB. 100==-0.25 dB. 101=-0.5 dB. 110=-0.75 dB. 111=-6.0 dB. It is reset by a power-on reset only.

DESCRIPTION:

The AUDIOOUT Reference Control Register provides control over the voltage and power for the audio analog circuits.

EXAMPLE:

```
HW_AUDIOOUT_REFCTRL.ADJ_VAG = 1;
```

29.4.10 Miscellaneous Audio Controls Register Description

This register provides miscellenous audio control bits.

HW_AUDIOOUT_ANACTRL	0x090
HW_AUDIOOUT_ANACTRL_SET	0x094
HW_AUDIOOUT_ANACTRL_CLR	0x098
HW_AUDIOOUT_ANACTRL_TOG	0x09C

Table 29-20. HW_AUDIOOUT_ANACTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD8			SHORT_CM_STS	RSRVD7			SHORT_LR_STS	RSRVD6		SHORTMODE_CM		RSRVD5	SHORTMODE_LR		RSRVD4		SHORT_LVLADJL			RSRVD3	SHORT_LVLADJR			RSRVD2		HP_HOLD_GND	HP_CLASSAB	RSRVD1			

Table 29-21. HW_AUDIOOUT_ANACTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSRVD8**	RO	0x00	Reserved. Always write zeroes to this bit field. It is reset by a power-on reset only.
28	**SHORT_CM_STS**	RW	0x0	Status of common mode amplifier short detection: 0=No short. To clear this interrupt and then rearm it: (1) Set HW_AUDIOOUT_ANACTRL_SHORTMODE_CM to 00. (2) Clear this bit. (3) Set HW_AUDIOOUT_ANACTRL_SHORTMODE_CM to 01. There are two sets of edge-triggered latches in this path. All three steps must be executed to rearm the short detect.
27:25	**RSRVD7**	RO	0x0	Reserved. Always write zeroes to this bit field.
24	**SHORT_LR_STS**	RW	0x0	Status of headphone amplifier short detection: 0=No short. To clear this interrupt and then rearm it: (1) Set HW_AUDIOOUT_ANACTRL_SHORTMODE_LR to 00. (2) Clear this bit. (3) Set HW_AUDIOOUT_ANACTRL_SHORTMODE_LR to 01. There are two sets of edge-triggered latches in this path. All three steps must be executed to rearm the short detect.
23:22	**RSRVD6**	RO	0x0	Reserved. Always write zeroes to this bit field.
21:20	**SHORTMODE_CM**	RW	0x0	Headphone Common Mode Amplifier Short Control Mode. 00=Reset analog latch, HW power down on unlatched short signal. 01=Latch short signal. HW power down on latched signal. 10=Do not use. 11=Do not use.
19	**RSRVD5**	RO	0x00	Reserved. Always write zeroes to this bit field.
18:17	**SHORTMODE_LR**	RW	0x0	Headphone Left and Right Channel Short Control mode. 00=Reset analog latch, HW power-down disabled. 01=Latch short signal, HW power-down enabled. 10=Do not use. 11=Do not use.
16:15	**RSRVD4**	RO	0x0	Reserved. Always write zeroes to this bit field.
14:12	**SHORT_LVLADJL**	RW	0x0	Adjust the left headphone current short detect trip point: 000=Nominal. 001=-25%. 010=-50%. 011=-75%. 100=+25%. 101=+50%. 110=+75%. 111=+100%. It is reset by a power-on reset only.
11	**RSRVD3**	RO	0x0	Reserved. Always write zeroes to this bit field.
10:8	**SHORT_LVLADJR**	RW	0x0	Adjust the right headphone current short detect trip point: 000=Nominal. 001=-25%. 010=-50%. 011=-75%. 100=+25%. 101=+50%. 110=+75%. 111=+100%. It is reset by a power-on reset only.
7:6	**RSRVD2**	RO	0x0	Reserved. Always write zeroes to this bit field.
5	**HP_HOLD_GND**	RW	0x0	Hold Headphone Output to Ground (used for power-up/power-down procedures). It is reset by a power-on reset only.
4	**HP_CLASSAB**	RW	0x0	Default is 0 (ClassA mode). ClassA mode can be useful for power-up/power-down and short handling. This bit should be set (ClassAB mode) before starting audio signal. It is reset by a power-on reset only.
3:0	**RSRVD1**	RO	0x0	Reserved. Always write zeroes to this bit field.

DESCRIPTION:

This register provides miscellaneous audio control bits.

EXAMPLE:

```
HW_AUDIOOUT_ANACTRL.EN_ZCD = 1; // Enable zero cross detect.
```

29.4.11 Miscellaneous Test Audio Controls Register Description

This register provides miscellaneous audio test bits.

HW_AUDIOOUT_TEST	0x0a0
HW_AUDIOOUT_TEST_SET	0x0a4
HW_AUDIOOUT_TEST_CLR	0x0a8
HW_AUDIOOUT_TEST_TOG	0x0aC

Table 29-22. HW_AUDIOOUT_TEST

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD4	HP_ANTIPOP			RSRVD3	TM_ADCIN_TOHP	TM_LOOP	TM_HPCOMMON	HP_I1_ADJ		HP_IALL_ADJ		RSRVD2						VAG_CLASSA	VAG_DOUBLE_I	RSRVD1								ADCTODAC_LOOP	DAC_CLASSA	DAC_DOUBLE_I	DAC_DIS_RTZ

Table 29-23. HW_AUDIOOUT_TEST Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSRVD4**	RO	0x0	Reserved. Always write zeroes to this bit field.
30:28	**HP_ANTIPOP**	RW	0x0	Reserved
27	**RSRVD3**	RO	0x0	Reserved. Always write zeroes to this bit field.
26	**TM_ADCIN_TOHP**	RW	0x0	Testmode to pipe ADC Mux Out (ADC In) to Headphone Output pins. No longer have ADC filter pins, this allows visibility to ADC Mux amp performance. To use this mode, the headphone load and the headphone board compensation must be removed (the ADC amp cannot drive it). Note that while this bit is located in the AUDIOOUT address space, it is an ADC function controlled by the AUDIOIN's SFTRST and CLKGATE bits.
25	**TM_LOOP**	RW	0x0	Testmode to connect headphone out left to the microphone input to the ADC, and speakerp to the ADC. This is used for analog loopback DAC-speaker-Mix-ADC Mode. There should be no load on the microphone input pin during this mode.
24	**TM_HPCOMMON**	RW	0x0	Uses headphone common VAG, instead of vaggate in ADC Mux. This is used for analog loopback DAC-HP-ADC mode to include common amp in path.
23:22	**HP_I1_ADJ**	RW	0x0	Adjusts bias current in first stage of headphone amplifier : 00=Nominal. 01=-50%. 10=+100%. 11=+50%. It is reset by a power-on reset only.
21:20	**HP_IALL_ADJ**	RW	0x0	Adjusts all bias current in headphone amplifier : 00=nom, 01=-50%, 10=+50%, 11=-40%. It is reset by a power-on reset only.
19:14	**RSRVD2**	RO	0x0	Reserved. Always write zeroes to this bit field.

Table 29-23. HW_AUDIOOUT_TEST Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
13	**VAG_CLASSA**	RW	0x0	Set to one to disable ClassAB mode in VAG Amp. Will increase current by ~200 uA. Note that while this bit is located in the AUDIOOUT address space, since it controls both DAC and ADC functions, it is not reset by the AUDIOOUT's SFTRST bit. It is reset by a power-on reset only, and CLKGATE has no effect on reads/writes.
12	**VAG_DOUBLE_I**	RW	0x0	Set to one to double ClassA current in VAG amplifier (+240uA). Note that while this bit is located in the AUDIOOUT address space, since it controls both DAC and ADC functions, it is not reset by the AUDIOOUT's SFTRST bit. It is reset by a power-on reset only, and CLKGATE has no effect on reads/writes.
11:4	**RSRVD1**	RO	0x00	Reserved. Always write zeroes to this bit field.
3	**ADCTODAC_LOOP**	RW	0x0	Set to one to loop the 1-bit SDM ADC data in to the 1-bit SDM DAC data out for test.
2	**DAC_CLASSA**	RW	0x0	Set to one to disable ClassAB mode in DAC. Will increase power by ~600 uA.
1	**DAC_DOUBLE_I**	RW	0x0	Set to one to double ClassA current in DAC amplifier (+360 uA in each DAC).
0	**DAC_DIS_RTZ**	RW	0x0	Set to one to disable DAC RTZ mode. Test-only bit that disables the return-to-zero function. This bit should remain cleared.

DESCRIPTION:

This register provides miscellaneous audio test bits..

EXAMPLE:

```
HW_AUDIOOUT_TEST.TM_HPCOMMON = 1; // Use headphone common VAG.
```

29.4.12 BIST Control and Status Register Description

The BIST Control and Status Register provides overall control of the integrated BIST engine.

HW_AUDIOOUT_BISTCTRL	0x0b0
HW_AUDIOOUT_BISTCTRL_SET	0x0b4
HW_AUDIOOUT_BISTCTRL_CLR	0x0b8
HW_AUDIOOUT_BISTCTRL_TOG	0x0bC

Table 29-24. HW_AUDIOOUT_BISTCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD0																												FAIL	PASS	DONE	START

Table 29-25. HW_AUDIOOUT_BISTCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:4	**RSVD0**	RO	0x0	Reserved
3	**FAIL**	RO	0x0	BIST has failed.
2	**PASS**	RO	0x0	BIST has passed
1	**DONE**	RO	0x0	BIST has completed.
0	**START**	RW	0x0	Initiate BIST of internal memory.

DESCRIPTION:

The BISTCTRL Register provides overall control of the integrated BIST engine.

EXAMPLE:

```
HW_AUDIOUT_BISTCTRL.U = 0x00000000;
```

29.4.13 Hardware BIST Status 0 Register Description

This register provides visibility into memory failures detected by the BIST engine.

HW_AUDIOOUT_BISTSTAT0	0x0c0
HW_AUDIOOUT_BISTSTAT0_SET	0x0c4
HW_AUDIOOUT_BISTSTAT0_CLR	0x0c8
HW_AUDIOOUT_BISTSTAT0_TOG	0x0cC

Table 29-26. HW_AUDIOOUT_BISTSTAT0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD0								DATA																							

Table 29-27. HW_AUDIOOUT_BISTSTAT0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSVD0**	RO	0x0	Reserved
23:0	**DATA**	RO	0x0	Failing data at the failing address.

DESCRIPTION:

The AUDIOOUT BISTSTAT0 provides visibility into memory failures detected by the BIST engine.

EXAMPLE:

```
HW_AUDIOUT_BISTSTAT0.U = 0x00000000;
```

29.4.14 Hardware AUDIOUT BIST Status 1 Register Description

The AUDIOOUT BISTATTS1 provides visibility into memory failures detected by the BIST engine.

HW_AUDIOOUT_BISTSTAT1	0x0d0
HW_AUDIOOUT_BISTSTAT1_SET	0x0d4
HW_AUDIOOUT_BISTSTAT1_CLR	0x0d8
HW_AUDIOOUT_BISTSTAT1_TOG	0x0dC

Table 29-28. HW_AUDIOOUT_BISTSTAT1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD1				STATE				RSVD0																ADDR							

Table 29-29. HW_AUDIOOUT_BISTSTAT1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSVD1**	RO	0x0	Reserved
28:24	**STATE**	RO	0x0	Fail state of the BIST engine.
23:8	**RSVD0**	RO	0x0	Failing data at the failing address.
7:0	**ADDR**	RO	0x0	Failing data at the failing address.

DESCRIPTION:

The AUDIOOUT BISTATTS1 provides visibility into memory failures detected by the BIST engine.

EXAMPLE:

```
HW_AUDIOUT_BISTATTS1.U = 0x00000000;
```

29.4.15 Analog Clock Control Register Description

This register provides analog clock control.

HW_AUDIOOUT_ANACLKCTRL	0x0e0
HW_AUDIOOUT_ANACLKCTRL_SET	0x0e4
HW_AUDIOOUT_ANACLKCTRL_CLR	0x0e8
HW_AUDIOOUT_ANACLKCTRL_TOG	0x0eC

Table 29-30. HW_AUDIOOUT_ANACLKCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CLKGATE																											INVERT_DACCLK	RSRVD2			DACDIV	
												RSRVD3																			

Table 29-31. HW_AUDIOOUT_ANACLKCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**CLKGATE**	RW	0x1	Analog clock Gate. Set this bit to gate the clocks for the DAC converter and associated digital filter. It is reset by a power-on reset only.
30:5	**RSRVD3**	RO	0x0	Reserved

Table 29-31. HW_AUDIOOUT_ANACLKCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
4	**INVERT_DACCLK**	RW	0x0	DAC Clock Invert. Set this bit to invert the DAC_CLK for the DAC sigma-delta converter and associated digital filters.
3	**RSRVD2**	RO	0x0	Reserved
2:0	**DACDIV**	RW	0x0	DAC Analog Clock Divider. This bit field is used to select the oversampling clock rate used by the ADC. This bit field should only be changed per Freescale. 000=6 MHz. 001=4 MHz. 010/100=3 MHz. 011/101=2 MHz. 110=1.5 MHz. 111=1 MHz.

DESCRIPTION:

This register provides analog clock control.

EXAMPLE:

```
HW_AUDIOOUT_ANACLKCTRL.INVERT_DACCLK = 1; // Invert DAC clock.
```

29.4.16 AUDIOOUT Write Data Register Description

The AUDIOOUT Write Data Register provides a means to output PCM audio samples.

HW_AUDIOOUT_DATA	0x0f0
HW_AUDIOOUT_DATA_SET	0x0f4
HW_AUDIOOUT_DATA_CLR	0x0f8
HW_AUDIOOUT_DATA_TOG	0x0fC

Table 29-32. HW_AUDIOOUT_DATA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
HIGH																LOW															

Table 29-33. HW_AUDIOOUT_DATA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**HIGH**	RW	0x0000	Right Sample or Sample High Half-Word. For 16-bit sample mode, this field contains the right channel sample. For 32-bit sample mode, this field contains the most significant 16 bits of the 32-bit sample (either left or right).
15:0	**LOW**	RW	0x0000	Left Sample or Sample Low Half-Word. For 16-bit sample mode, this field contains the left channel sample. For 32-bit per sample mode, this field contains the least significant 16 bits of the 32-bit sample (either left or right).

DESCRIPTION:

The AUDIOOUT Write Data Register provides 32-bit data transfers for the DMA or alternatively can be directly written by the CPU. Each data value written to the register is placed in the AUDIOOUT's FIFO,

which in turn is used by the digital FIR filter stages. These 32-bit values contain either one 32-bit sample or two 16-bit samples, depending on how the data size is programmed. Note that the PCM audio data input to the FIR filter stages is 24 bit. For 16-bit operation, the input data is sign extended to 24 bits. For 32-bit mode, it is normalized by dropping the least significant 8 bits.

EXAMPLE:

```
HW_AUDIOOUT_DATA.U = 0x12345678; // write 0x1234 to the right channel and 0x5678 to the left channel
in 16 bit per sample mode
HW_AUDIOOUT_DATA.U = 0x12345678; // write 0x12345678 to either the left or right channel in 32 bit
per sample mode.
```

29.4.17 AUDIOOUT Speaker Control Register Description

The AUDIOOUT Speaker Control Register contains bit-fields to control the Speaker analog block.

HW_AUDIOOUT_SPEAKERCTRL	0x100
HW_AUDIOOUT_SPEAKERCTRL_SET	0x104
HW_AUDIOOUT_SPEAKERCTRL_CLR	0x108
HW_AUDIOOUT_SPEAKERCTRL_TOG	0x10C

Table 29-34. HW_AUDIOOUT_SPEAKERCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2							MUTE	I1_ADJ		IALL_ADJ		RSRVD1				POSDRIVER		NEGDRIVER		RSRVD0											

Table 29-35. HW_AUDIOOUT_SPEAKERCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSRVD2**	RO	0x00	Reserved
24	**MUTE**	RW	0x1	This bit mutes the speaker outputs, SPEAKERP and SPEAKERN. The speaker antipop startup/shutdown sequence should be followed before toggling this bit.
23:22	**I1_ADJ**	RW	0x0	Adjusts bias current in first stage of speaker amplifier : 00=Nominal. 01=-50%. 10=+100%. 11=+50%. It is reset by a power-on reset only.
21:20	**IALL_ADJ**	RW	0x0	Adjusts all bias current in speaker amplifier : 00=Nominal, 01=-50%, 10=+50%, 11=-40%. It is reset by a power-on reset only.
19:16	**RSRVD1**	RO	0x0000	Reserved
15:14	**POSDRIVER**	RW	0x0	SPEAKERP pin control: 00=Normal speaker mode, 01=Drive SPEAKERP low, 10=Drive SPEAKERP high, 11=Tri-state SPEAKERP. It is reset by a power-on reset only.
13:12	**NEGDRIVER**	RW	0x0	SPEAKERN pin control: 00=Normal speaker mode, 01=Drive SPEAKERN low, 10=Drive SPEAKERN high, 11=Tri-state SPEAKERN. It is reset by a power-on reset only.
11:0	**RSRVD0**	RO	0x000	Reserved

DESCRIPTION:

The AUDIOOUT Speaker Control Register controls the analog speaker module. It provides programmability for the amplifier bia current and a mute.

EXAMPLE:

```
HW_AUDIOOUT_MUTE= 0x1; // mute line out channel
```

29.4.18 AUDIOOUT Version Register Description

The AUDIOOUT Version Register is read-only and is used for debug to determine the implementation version number for the block.

HW_AUDIOOUT_VERSION 0x200

Table 29-36. HW_AUDIOOUT_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 29-37. HW_AUDIOOUT_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x01	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x01	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
if (HW_AUDIOOUT_VERSION.B.MAJOR != 1)
Error();
```

AUDIOOUT Block v1.3, Revision 1.65

Chapter 30
SPDIF Transmitter

This chapter describes the SPDIF transmitter provided on the i.MX23. It includes sections on interrupts, clocking, DMA operation, and PIO debug mode. Programmable registers are described in Section 30.3, "Programmable Registers."

30.1 Overview

The Sony-Philips Digital Interface Format (SPDIF) transmitter module transmits data according to the SPDIF digital audio interface standard (IEC-60958). Figure 30-1 shows a block diagram of the SPDIF transmitter module.

Data samples are transmitted as blocks of 192 frames, each frame consisting of two 32-bit sub-frames.

A 32-bit sub-frame is composed of a 4-bit preamble, a 24-bit data payload (e.g., a left- or right-channel PCM sample), and a 4-bit status field. The status fields are encoded according to the IEC-60958 consumer specification, reflecting the contents of the HW_SPDIF_FRAMECTRL and HW_SPDIF_CTRL registers. See the IEC-60958 specification for proper programming of these fields.

The sub-frame is transmitted serially, LSB-first, using a biphase-mark channel-coding scheme. This encoding allows an SPDIF receiver to recover the embedded clock signal. NOTE: Sub-frame information can be changed "on-the-fly" but is not reflected in the serial stream until the current frame is transmitted. This ensures consistency of the frame and the generated parity appended to that frame.

30.2 Operation

The SPDIF transmitter operates at one of three register-selectable base sample rates: 32 kHz, 44.1 kHz, or 48 kHz. Double-rate output (64 kHz, 88.2 kHz, and 96 kHz) can also be selected using HW_SPDIF_SRR_BASEMULT. The data-clock required to transmit an SPDIF frame at these sample-rates is generated using a fractional clock-divider. This divider uses both edges of a 120-MHz clock, which is derived from a divide-by-4 of the PLL 480-MHz clock. This divider is located in the CLKCTRL module where all system clocks are generated; the resultant clock (pcm_spdif_clk) is output to the SPDIF module to be used for data transmission.

Programming the HW_SPDIF_SRR register automatically generates the right frequency of pcm_spdif_clk from the divider in the clock controller block. There are no separate registers to control these dividers. Make sure that the CLKGATE bit in HW_CLKCTRL_SPDIF is cleared before starting the transmission.

The SPDIF module receives data by one of two methods:

- Software-directed PIO writes to the HW_SPDIF_DATA register
- Appropriate programming of the DMA-engine. (See Chapter 11, "AHB-to-APBX Bridge with DMA," for a detailed description of the DMA module and how to perform DMA data transfers to/from modules and memory.)

Once provided by the DMA, the received data is placed in a 2x24 word FIFO for each channel, left and right. At initialization, the FIFO is filled before SPDIF data transfer occurs. After this, data is requested whenever this FIFO has an empty entry or at a nominal rate corresponding to the programmed sample-rate in HW_SPDIF_SRR.

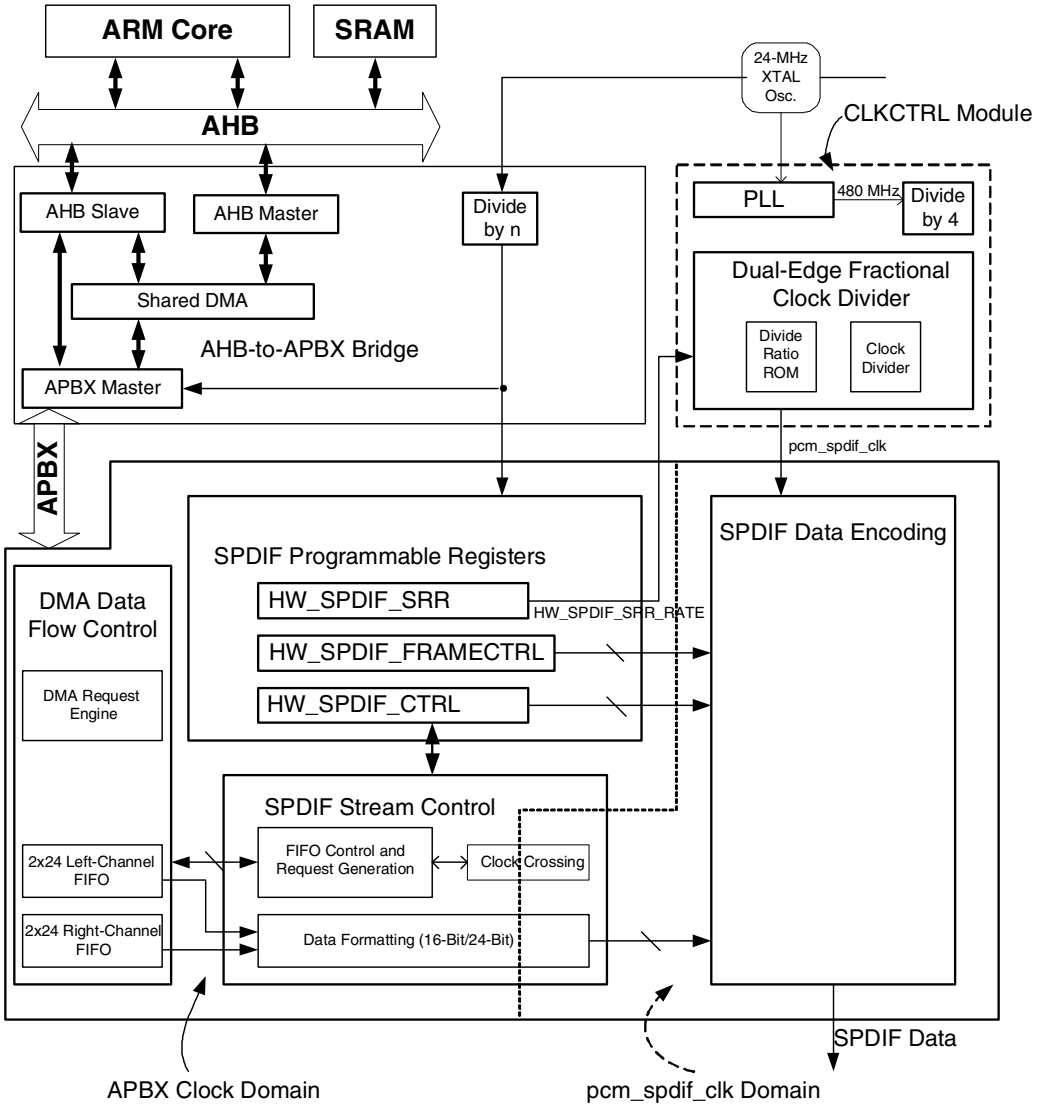


Figure 30-1. SPDIF Transmitter Block Diagram

The behavior of the SPDIF module during or after a FIFO underflow is programmable. On detection of an underflow event, the SPDIF module sends the current sample for four frames before muting (sending zeros) the data stream based on the configuration of HW_SPDIF_FRAMECTRL_AUTO_MUTE. The final validity unit embedded within each frame dictates whether the receiver processes the data within that frame. HW_SPDIF_FRAMECTRL determines the behavior of this bit.

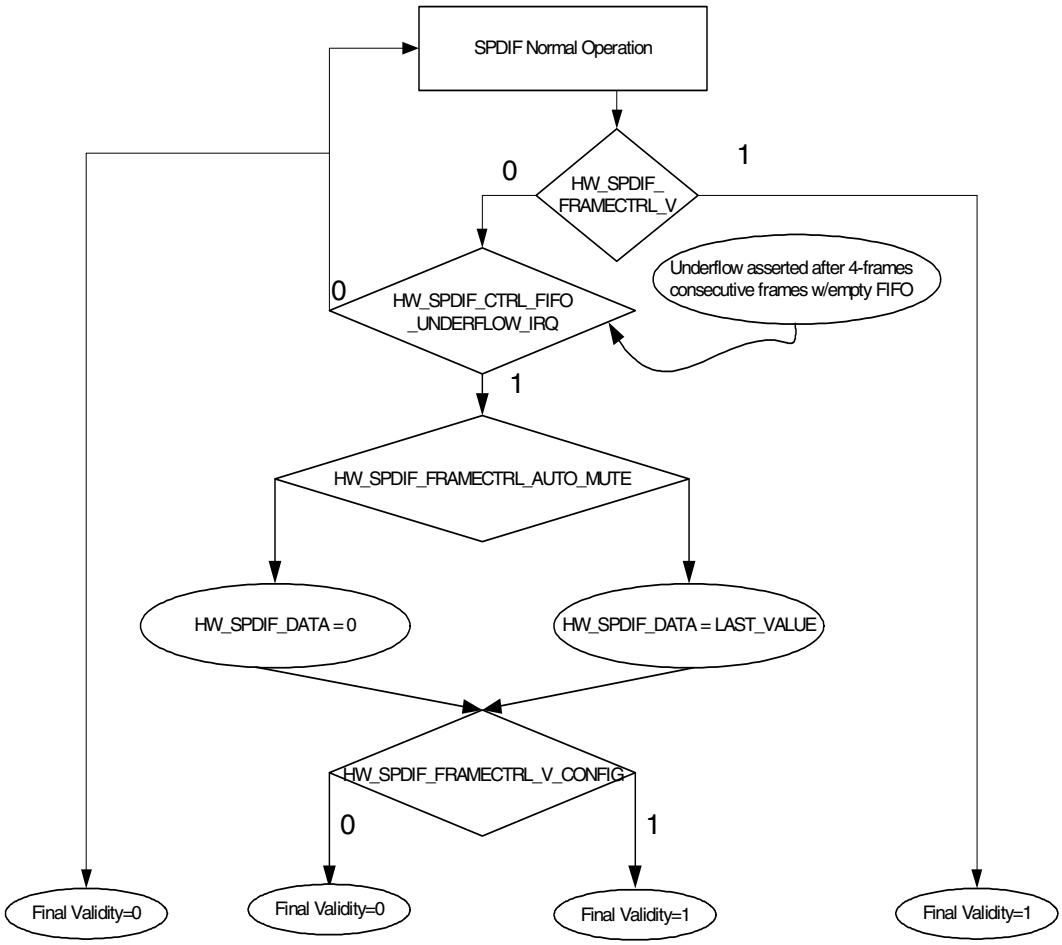


Figure 30-2. SPDIF Flow Chart

SPDIF data can be transmitted in one of two modes: 32-bit mode and 16-bit mode. Selection between these modes is done with the WORD_LENGTH bit in the HW_SPDIF_CTR register. In either case, data samples must be interleaved in main memory for proper behavior, although in 32-bit mode, 32-bit words are interleaved and in 16-bit mode, 16-bit words are interleaved.

- When WORD_LENGTH = 0, 32-bit mode is enabled, and HW_SPDIF_DATA contains either the left or right data sample. Since the SPDIF frame allows for transmission of only 24 bits, only the 24 MSBs stored in the HW_SPDIF_DATA register will be transmitted.
- Alternately, when WORD_LENGTH = 1, 16-bit mode is enabled, and the HW_SPDIF_DATA register will contain one of each left AND right samples. The data transmitted in the SPDIF frame will be these 16 MSBs with 8 zeros appended in the LSB positions.

NOTE: If the data supplied actually represents a lower resolution analog-to-digital conversion, this information is not captured by the SPDIF transmitter, which always reports a 24-bit sample-size.

30.2.1 Interrupts

The SPDIF module contains a single interrupt source that is asserted on FIFO overflows and/or FIFO underflows. This interrupt is enabled by setting HW_SPDIF_CTRL_FIFO_ERROR_IRQ_EN. On interrupt detection, the HW_SPDIF_CTRL_FIFO_UNDERFLOW_IRQ and HW_SPDIF_CTRL_FIFO_OVERFLOW_IRQ fields can be polled for the exact cause of the interrupt and appropriate action taken.

Note: These bits remain valid for polling, regardless of the state of the interrupt enable.

30.2.2 Clocking

The IEC-60958 specification outlines the requirements for SPDIF clocking. The SPDIF module is designed according to the Consumer Audio requirements. These dictate that:

- Average Sample-Rate Error must not exceed 1000 ppm
- Maximum Instantaneous Jitter must not exceed ~4.4 ns.

The jitter requirement implies either a single-phase of a >240-MHz clock or both phases of a 120-MHz clock. It also implies the use of a fractional divider for which the divisors are maintained to sufficient significant digits to yield the required ppm tolerance. The SPDIF module in the i.MX23 uses nine-bit fractional coefficients that yield an average frequency error of 52 ppm. These coefficients are determined according to the required clock-rates that are dictated by the sample rates implemented. The required clock frequencies provided by the CLKCTRL module for the implemented sample-rates are:

F(48 kHz) ≥ 6.144 MHz

F(44.1 kHz) ≥ 5.6448 MHz

F(32 kHz) ≥ 4.096 MHz

F(96 kHz) ≥ 12.288 MHz

F(88.2 kHz) ≥ 11.2896 MHz

F(64 kHz) ≥ 8.192 MHz

All clocks within the SPDIF module are gated according to the state of HW_SPDIF_CTRL_CLKGATE. When set, all clocks derived from the apb_clk are gated. Gating of the pcm_spdif_clk is accomplished through HW_CLKCTRL_SPDIF_CLKGATE. A module-level reset is also provided in HW_SPDIF_CTRL_SFTRST. Setting this bit performs a module-wide reset and subsequent assertion of the HW_SPDIF_CTRL_CLKGATE.

NOTE: A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, "Correct Way to Soft Reset a Block," for additional information on using the SFTRST and CLKGATE bit fields.

30.2.3 DMA Operation

Using the SPDIF module in DMA mode involves configuring the appropriate DMA channel to provide the interleaved data blocks stored in memory. See Chapter 11, "AHB-to-APBX Bridge with DMA," for detailed information on DMA programming. Once programmed, the DMA engine references a set of linked DMA descriptors stored by the CPU in main memory. These descriptors point to data blocks stored in system memory and also provide a mechanism for automated PIO writes before transfer of a data-block. Figure 30-3 describes a typical set of descriptors required to transmit two data-blocks.

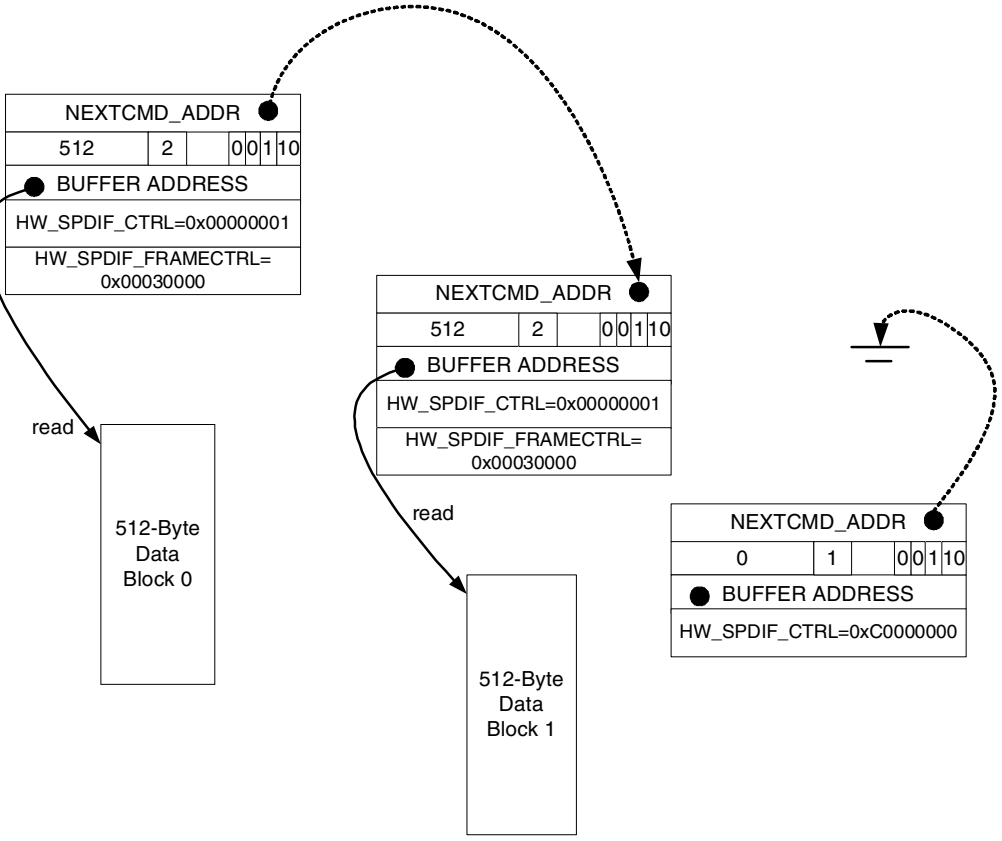


Figure 30-3. SPDIF DMA Two-Block Transmit Example

Here, the DMA is instructed to perform two PIO writes prior to toggling the DMA_PCMDKICK signal**:**

- HW_SPDIF_CTRL_FIFO_UNDERFLOW_IRQ_EN is set to enable interrupts on FIFO underflow detect
- HW_SPDIF_FRAMECTRL_AUTO_MUTE and HW_SPDIF_FRAMECTRL_V_CONFIG are set to mute and tag the data stream as invalid on a FIFO underflow.

The DMA engine is then programmed to transfer 512-bytes to the SPDIF module.

Additionally, the SPDIF module contains a mechanism for “throttling” DMA requests to the DMA engine. This circuit is programmed using the HW_SPDIF_CTRL_DMAWAIT_COUNT field and corresponds to the number of cycles of the apb_clk to wait before toggling the DMA_PREQ signal to the DMA engine.

NOTE: Considering that the bandwidth requirements of the SPDIF module are minimal (not in excess of 96 kHz) and burst requests occur only in pairs, this field can be ignored for most, if not all, applications.

There is a floor APBX frequency below which the SPDIF cannot work without errors. That frequency can be calculated as follows:

- Assume that there are 6 other blocks apart from SPDIF on the APBX bus, and it takes 4 APBX clock cycles to service each block. If the number of clock cycles required to service each block changes, change the equations accordingly.
- Assume that HW_SPDIF_CTRL_DMAWAIT_COUNT is less than DMA LATENCY. If this is not true, then even DMA WAIT has to be added to the calculation and the floor APBX frequency increases further.

In 16-bit Mode:

```
Floor APBX freq = (DMA latency + 9) * sample rate.
For max DMA latency = (6 blocks) x (4 cycles per block) = 24 cycles and max SPDIF
sample rate = 96 kHz,
min APBX freq = 3.168 MHz.
```

In 32-bit Mode:

(A) Ideal Calculation:

```
min freq = [2*(DMA latency+4) + 7] * sample rate.
For max DMA latency = 24 cycles and max SPDIF sample rate = 96 kHz,
min APBX freq = 6.048 MHz.
```

(B) Simpler Calculation:

```
Floor APBX freq = 2*(latency + 9) * sample rate = twice that of 16-bit mode.
For max latency = 24 cycles and max sample rate = 96 kHz,
min APBX freq = 6.336 MHz.
```

Option A is ideal as it allows a lower floor frequency; option B can be used to keep it simple and avoid confusion.

30.2.4 PIO Debug Mode

The block is connected only as a PIO device to the APBX bus. Even though it is designed to work with the DMA controller integrated in the APBX bridge, all transfers to and from the block are programmed I/O (PIO) read or write cycles. When the DMA is ready to write to the HW_SPDIF_DATA register, it does so with standard APB write cycles. There *are* four DMA related signals that connect the SPDIF transmitter to the DMA, *but* all data transfers are standard PIO cycles on the APB. The state of these four signals can be seen in the HW_SPDIF_DEBUG register.

Thus, it is possible to completely exercise the SPDIF block for diagnostic purposes, using only load and store instructions from the CPU without ever starting the DMA controller. This section describes how to interact with the block using PIO operations, and also defines the block's detailed behavior.

Whenever the HW_SPDIF_CTRL register is written to, by either the CPU or the DMA, it establishes the basic operation mode for the block. If the HW_SPDIF_CTRL register is written with a 1 in the RUN bit, then the operation begins and the SPDIF attempts to read the data block by toggling its PDMAREQ signal to the DMA. Notice that the PDMAREQ signal is defined as a "toggle" signal. This changes state to signify either a request for another DMA word or a notification that the current command transfer has been ended by the SPDIF. Diagnostic software should poll these signals to determine when the SPDIF is ready for another DMA write, and can then supply data by storing a 32-bit word to the HW_SPDIF_DATA register, just as the DMA would do in normal operation.

To perform SPDIF transfers in PIO debug mode, diagnostic software should perform the following:

1. Clear CLKGATE in the HW_CLKCTRL_SPDIF register.
2. Turn off the Soft Reset bit, HW_SPDIF_CTRL_SFTRST**,** and the Clock Gate bit, HW_SPDIF_CTRL_CLKGATE.
3. Properly configure the subcode information by writing the HW_SPDIF_FRAMECTRL register. NOTE: See IEC-60958 for proper coding of these fields.
4. Enable the SPDIF transmitter by setting HW_SPDIF_CTRL_RUN.
5. Wait for HW_SPDIF_DEBUG_DMA_PREQ status bit to toggle.
6. Write one sample of the left/right DATA block data to the HW_SPDIF_DATA register.
7. Repeat 5 and 6 until all samples have been written to HW_SPDIF_DATA.

30.3 Programmable Registers

The following registers provide control for programmable elements of the SPDIF module.

30.3.1 SPDIF Control Register Description

The SPDIF Control Register provides overall control of the SPDIF converter.

HW_SPDIF_CTRL	0x000
HW_SPDIF_CTRL_SET	0x004
HW_SPDIF_CTRL_CLR	0x008
HW_SPDIF_CTRL_TOG	0x00C

Table 30-1. HW_SPDIF_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	RSRVD1									DMAWAIT_COUNT					RSRVD0										WAIT_END_XFER	WORD_LENGTH	FIFO_UNDERFLOW_IRQ	FIFO_OVERFLOW_IRQ	FIFO_ERROR_IRQ_EN	RUN

Table 30-2. HW_SPDIF_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	Setting this bit to one forces a reset to the entire block and then gates the clocks off. This bit must be set to zero for normal operation.
30	**CLKGATE**	RW	0x1	This bit must be set to zero for normal operation. When set to one it gates off the clocks to the block. WARNING: First set the CLKGATE bit in the HW_CLKCTRL_SPDIF register to 1. Only then, set this bit to 1 to prevent any extra samples from being transmitted. When removing clock gating, follow the reverse order: First reset this CLKGATE bit to 0, and then reset the CLKGATE bit in the HW_CLKCTRL_SPDIF register to 0 .
29:21	**RSRVD1**	RO	0x00	Reserved
20:16	**DMAWAIT_COUNT**	RW	0x00	DMA Request Delay Count. This bit field specifies the number of APBX clock cycles (0 to 31) to delay before each DMA request. This field acts as a throttle on the bandwidth consumed by the SPDIF block. This field can be loaded by the DMA.
15:6	**RSRVD0**	RO	0x000	Reserved
5	**WAIT_END_XFER**	RW	0x1	Set this bit to a one if the SPDIF Transmitter should wait until the internal FIFO is empty before halting transmission based on deassertion of RUN. Use in conjuntion with HW_SPDIF_STAT_END_XFER to determine transfer completion
4	**WORD_LENGTH**	RW	0x0	Set this bit to one to enable 16-bit mode. Set this bit to zero for 32-bit mode. In either case, the SPDIF frame allows transmission of only 24 bits. In 16-bit mode, eight zeros will be appended to the LSB's of the input sample; in 32-bit mode, the 24 MSB's of HW_SPDIF_DATA will be transmitted.
3	**FIFO_UNDERFLOW_IRQ**	RW	0x0	This bit is set by hardware if the FIFO underflows during SPDIF transmission. Reset this bit by writing a one to the SCT clear address space and not by a general write.

Table 30-2. HW_SPDIF_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**FIFO_OVERFLOW_IRQ**	RW	0x0	This bit is set by hardware if the FIFO overflows during SPDIF transmission. Reset this bit by writing a one to the SCT clear address space and not by a general write.
1	**FIFO_ERROR_IRQ_EN**	RW	0x0	Set this bit to one to enable a SPDIF interrupt request on FIFO overflow or underflow status conditions.
0	**RUN**	RW	0x0	Setting this bit to one causes the SPDIF to begin converting data. The actual conversion will begin when the SPDIF FIFO is filled (4 or 8 words written, depending upon sample word format, i.e., 16 or 32 bits).

DESCRIPTION:

The SPDIF Control Register contains the overall control for SPDIF sample formats, loopback mode, and interrupt controls.

EXAMPLE:

```
HW_SPDIF_CTRL.RUN = 1; // start SPDIF conversion
```

30.3.2 SPDIF Status Register Description

The SPDIF Status Register reflects overall status of the SPDIF converter.

HW_SPDIF_STAT	0x010
HW_SPDIF_STAT_SET	0x014
HW_SPDIF_STAT_CLR	0x018
HW_SPDIF_STAT_TOG	0x01C

Table 30-3. HW_SPDIF_STAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PRESENT	RSRVD1																														END_XFER

Table 30-4. HW_SPDIF_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**PRESENT**	RO	0x1	This bit is set to 1 in products in which SPDIF is present.
30:1	**RSRVD1**	RO	0x0	Reserved
0	**END_XFER**	RO	0x0	When set, indicates that the SPDIF module has completed transfer of all data, including data stored in internal FIFOs. Used in conjunction with HW_SPDIF_CTRL_WAIT_END_XFER.

DESCRIPTION:

The SPDIF Status Register provides the status of the SPDIF converter.

EXAMPLE:

```
unsigned TestBit = HW_SPDIF_STAT.PRESENT;
```

30.3.3 SPDIF Frame Control Register Description

The SPDIF Frame Control Register provides direct control of the control bits transmitted over an SPDIF frame.

HW_SPDIF_FRAMECTRL	0x020
HW_SPDIF_FRAMECTRL_SET	0x024
HW_SPDIF_FRAMECTRL_CLR	0x028
HW_SPDIF_FRAMECTRL_TOG	0x02C

Table 30-5. HW_SPDIF_FRAMECTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2														V_CONFIG	AUTO_MUTE	RSRVD1	USER_DATA	V	L	RSRVD0	CC							PRE	COPY	AUDIO	PRO

Table 30-6. HW_SPDIF_FRAMECTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:18	**RSRVD2**	RO	0x0	Reserved
17	**V_CONFIG**	RW	0x1	Defines SPDIF behavior when sending invalid frames. 0:Do NOT tag frame as invalid. 1: Tag frame as invalid.
16	**AUTO_MUTE**	RW	0x0	Auto-Mute Stream on stream-suspend detect.
15	**RSRVD1**	RO	0x0	Reserved
14	**USER_DATA**	RW	0x0	User data transmitted during each sub-frame. Consult IEC Standard for additional details.
13	**V**	RW	0x0	Indicates that a sub-frame's samples are invalid. If V=0, the sub-frame is indicated as valid, that is, correctly transmitted and received by the interface. If V=1, the subframe is indicated as invalid.
12	**L**	RW	0x0	Generation level is defined by the IEC standard, or as appropriate.
11	**RSRVD0**	RO	0x0	Reserved
10:4	**CC**	RW	0x0	Category code is defined by the IEC standard, or as appropriate.
3	**PRE**	RW	0x0	0: No Pre-Emphasis. 1: Pre-Emphasis is 50/15 usec.
2	**COPY**	RW	0x0	0: Copyright bit NOT asserted. 1: Copyright bit asserted.
1	**AUDIO**	RW	0x0	AUDIO=0:PCM Data;1. AUDIO=Non-PCM Data
0	**PRO**	RW	0x0	0: Consumer use of the channel. 1: Professional use of the channel.

DESCRIPTION:

The SPDIF Frame Control Register provides direct control of the control bits transmitted over an SPDIF frame.

EXAMPLE:

```
HW_SPDIF_FRAMECTRL.COPY=1 //SPDIF frame contains copyrighted material
```

30.3.4 SPDIF Sample Rate Register Description

The SPDIF Sample Rate Register controls the sample rate of the data stream played back from the circular buffer.

HW_SPDIF_SRR	0x030
HW_SPDIF_SRR_SET	0x034
HW_SPDIF_SRR_CLR	0x038
HW_SPDIF_SRR_TOG	0x03C

Table 30-7. HW_SPDIF_SRR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1	BASEMULT			RSRVD0								RATE																			

Table 30-8. HW_SPDIF_SRR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSRVD1**	RO	0x0	Reserved
30:28	**BASEMULT**	RW	0x1	Base-Rate Multiplier. 1 = Single-Rate (48 kHz). 2 = Double-Rate (96 kHz).
27:20	**RSRVD0**	RO	0x0	Reserved
19:0	**RATE**	RW	0x00000	Sample-Rate Conversion Factor. The only valid entries are: 0x07D00, 0x0AC44, 0x0BB80 // 32k, 44.1k, 48k

DESCRIPTION:

The SPDIF Sample Rate Register provides a RATE field for specifying the sample rate conversion factor to use in outputting the current SPDIF stream.

EXAMPLE:

```
HW_SPDIF_SRR.B.RATE = 0x0AC44; // 44.1KHz
```

30.3.5 SPDIF Debug Register Description

The SPDIF Debug Register provides read-only access to various internal state information that may be useful for block debugging and validation.

HW_SPDIF_DEBUG	0x040
HW_SPDIF_DEBUG_SET	0x044

HW_SPDIF_DEBUG_CLR	0x048
HW_SPDIF_DEBUG_TOG	0x04C

Table 30-9. HW_SPDIF_DEBUG

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																														DMA_PREQ	FIFO_STATUS

Table 30-10. HW_SPDIF_DEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:2	**RSRVD1**	RO	0x00	Reserved
1	**DMA_PREQ**	RO	0x0	DMA request status. This read-only bit reflects the current state of the SPDIF's DMA request signal. DMA requests are issued any time the request signal toggles. This bit can be polled by software, in order to manually move samples to the SPDIF's FIFO from a memory buffer when the SPDIF's DMA channel is not used
0	**FIFO_STATUS**	RO	0x1	This bit is set when the FIFO has empty space. This reflects a DMA request being generated.

DESCRIPTION:

This is a read-only register used for checking FIFO status and PIO mode of operation.

EXAMPLE:

```
unsigned TestBit = HW_SPDIF_DEBUG.DMA_PREQ;
```

30.3.6 SPDIF Write Data Register Description

The SPDIF Write Data Register receives 32-bit data transfers from the DMA. It deposits the data into an internal FIFO and from there into the SPDIF stream. These 32-bit writes contain either one 32-bit sample or two 16-bit samples.

HW_SPDIF_DATA	0x050
HW_SPDIF_DATA_SET	0x054
HW_SPDIF_DATA_CLR	0x058
HW_SPDIF_DATA_TOG	0x05C

Table 30-11. HW_SPDIF_DATA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
HIGH																LOW															

Table 30-12. HW_SPDIF_DATA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**HIGH**	RW	0x0000	For 16-bit mode, this field contains the entire right channel sample. For 32-bit mode, this field contains the 16 MSBs of the 32-bit sample (either left or right).
15:0	**LOW**	RW	0x0000	For 16-bit mode, this field contains the entire left channel sample. For 32-bit mode, this field contains the 16 LSBs of the 32-bit sample (either left or right).

DESCRIPTION:

Writing a 32-bit value to the register corresponds to pushing that 32-bit value into the SPDIF FIFO. The DMA writes 32-bit values to this register. In 32-bit-per-sample mode, the DMA is writing either one full left sample or one full right sample for each write to this register. For 16-bit mode, the DMA is writing a 16-bit left sample and a 16-bit right sample for each 32-bit write to this register.

EXAMPLE:

```
HW_SPDIF_DATA = 0x12345678; // write 0x1234 to the right channel and 0x5678 to the left channel in
16-bit mode
HW_SPDIF_DATA = 0x12345678; // write 0x12345678 to either the left or right channel in 32-bit per
sample mode.
```

30.3.7 SPDIF Version Register Description

The SPDIF Version Register is read-only and is used for debug to determine the implementation version number for the block.

HW_SPDIF_VERSION 0x060

Table 30-13. HW_SPDIF_VERSION

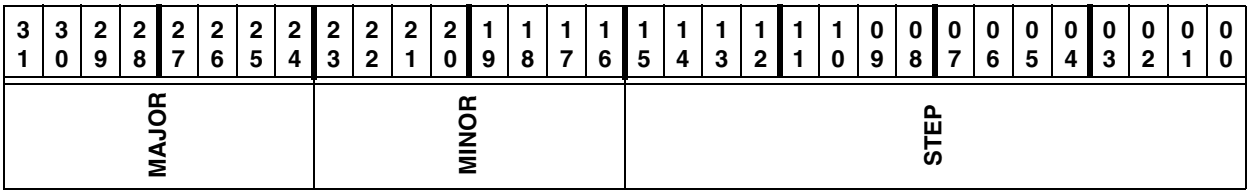


Table 30-14. HW_SPDIF_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x01	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x01	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
if (HW_SPDIF_VERSION.B.MAJOR != 1)
```

```
Error();
```

SPDIF Block v1.1, Revision 1.26

Chapter 31
Serial Audio Interface (SAIF) (BGA169 Only)

This chapter describes the two instances of the serial audio interface (SAIF) included on the i.MX23 and how to use them. Programmable registers are described in Section 31.3, "Programmable Registers."

31.1 Overview

The SAIF provides the following functions:

- 3-, 4-, or 5-wire serial interface to industry's most common analog codecs.
- Transmit or receive (half-duplex).
- 16-bit to 24-bit serial stereo digital audio PCM receive/transmit.
- Two, four, or six channels supported—three stereo pairs (mono supported in two-channel mode).
- Generic frame control supports I^2S, left- and right-justified frame formats, as well as other non-standard variants of these formats.
- Master and slave BITCLK and LRCLK modes (clocks driven to codec or received from codec), as well as optional master MCLK mode.
- Supports a continuous range of sample rates from 8 kHz to 192 kHz using a high-resolution fractional divider driven by the PLL.
- Programmable over-sample rate for MCLK output (32x, 48x, 64x, 96x, 128x, 192x, 256x, 384x, and 512x) supports codecs found in systems with both audio and video.
- Four-entry FIFOs (per sample pair) buffer either two-channel sample pairs (17-bit through 24-bit PCM) or four-packed-channel sample pairs (16-bit PCM).
- Samples transferred to/from the FIFO via the APBX DMA interface, a FIFO service interrupt, or software polling.

Figure 31-1 shows the major functional blocks within the SAIF.

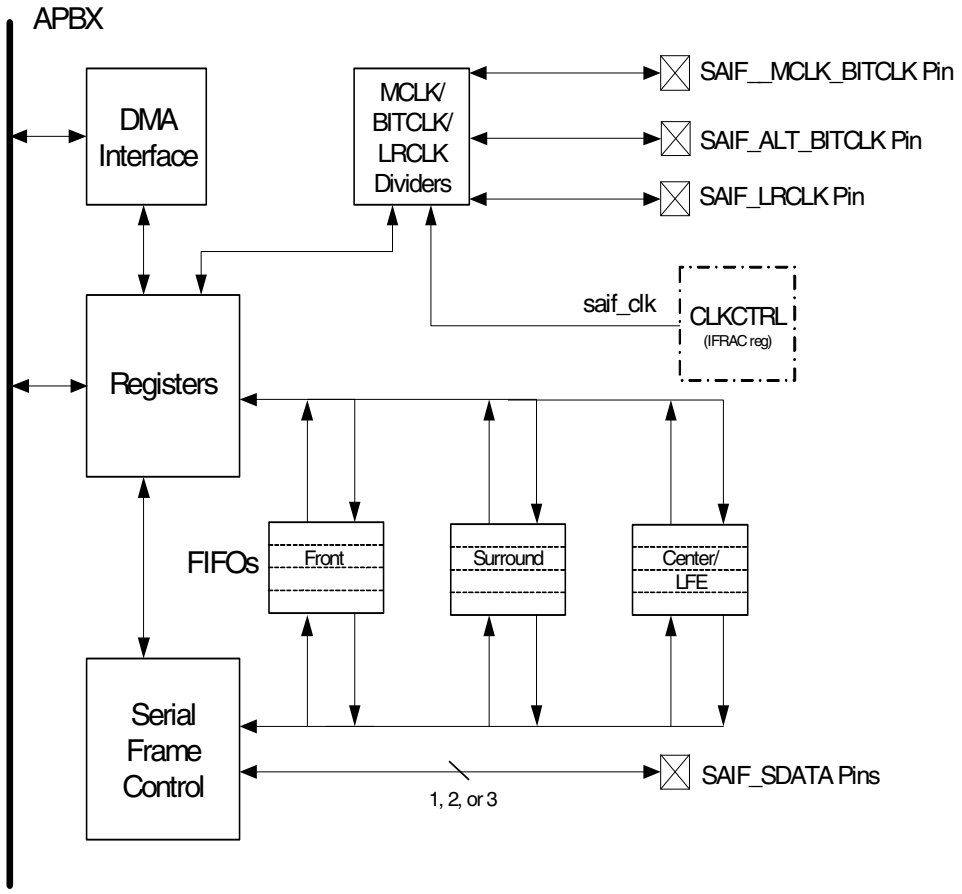


Figure 31-1. Serial Audio Interface (SAIF) Block Diagram

31.2 Operation

The SAIF is a half-duplex port, meaning it can either transmit or receive PCM audio, but not simultaneously. Data is communicated serially one sample at a time, alternating between left and right samples. One to three serial data lines (SDATA0–SDATA2) can be used to transmit either two (stereo/mono), four (stereo/surround), or six (stereo/surround/center/LFE) channels of digital PCM audio data. Samples boundaries are delineated by a left/right clock (LRCLK) pin, and individual bits within each sample are delineated by a bit clock (BITCLK) pin.

The LRCLK can be programmed to toggle every 16, 24, or 32 BITCLK transitions, and, because data ranges from 16 to 24 bits, serial data within each LRCLK period can either fully occupy the LRCLK cycle or cause the LRCLK period to contain BITCLK cycles in which no data is being communicated. Because of this, three basic types of sample frame formats can be programmed: I^2S, left-justified, and right-justified. However, many programming options exist to alter these basic frame types, such as the LRCLK signal polarity, BITCLK edge selection to drive/sample serial data, and sample justification/delay within an LRCLK period.

For codecs that do not contain their own PLL, or in applications where including a crystal oscillator to drive the codec is not desired, the SAIF can provide a master clock (MCLK) reference that can be configured from 512x down to 32x the audio data's sample rate. This master clock is used by the off-chip codec for all of its internal logic and to synchronize the BITCLK/LRCLK/SDATA inputs for DAC operation.

The digital PCM audio sample rate is determined by programming a fractional divider within the clock controller module.

31.2.1 Sample Rate Programming and Codec Clocking Operation

SAIF clocking is programmed in three blocks of the i.MX23:

- Clock control module (CLKCTRL)
- SAIF module
- Digital control module (DIGCTL)

The saif_clk (shown in Figure 31-1) is generated by the CLKCTRL block with a 16-bit fractional divider that divides down the 480-MHz PLL reference. The fractional divider minimizes system cost and power by eliminating the need for a second on-chip PLL. This fractional divider continuously selects which edge of the PLL reference clock (positive or negative) to use to best represent the oversample rate, such that less than 2 ns of correlated jitter occurs in saif_clk (jitter of the PLL plus periodic jitter of the fractional divide). This low level of jitter is required by most codec manufacturers to ensure a high SNR.

Table 31-1 shows the values to program the HW_CLKCTRL_SAIF_DIV bit field for all standard sample rates with either a base oversample rate of 512 or 384 times the sample rate.

These two base oversample values are the base rates typically required by codecs for the master clock (MCLK). An additional divider exists within the SAIF to generate sub-multiples of these two base rates if MCLK is required by the codec.

- The sub-rates that can be generated from 512x are: 256x, 128x, 64x, and 32x.
- The sub-rates for the 384x base rate are: 192x, 96x, and 48x.

The 384xFs base rate is common among systems that include MPEG1/2/4 audio and video, and AAC and AC-3 (Dolby Digital) audio. These MCLK rates are generated by programming the HW_SAIF_CTRL_BITCLK_BASE_RATE and HW_SAIF_CTRL_BITCLK_MULT_RATE bit fields.

Because there is a small amount of error in the sample rates generated by the fractional divider, software needs to periodically alter the HW_CLKCTRL_SAIF_DIV value higher or lower, depending on the difference between the required and actual sample rates. For an audio-only playback application, this adjustment need not ever be performed, because the frequency phase shift is imperceptible. However, it is needed for applications such as locking to and playing from an FM tuner, or audio/video applications where AV synchronization is required. For an audio/video application, it is typically accepted that audio and video cannot diverge any greater than ± 20 ms. Table 31-1 contains a column that lists the average elapsed time between each DIV adjustment required to keep within this 20-ms limit. Note that the small-

est elapsed time is just under 20 seconds. Typically, software can be configured to periodically monitor the read and write pointers for the digital PCM audio buffer in OCRAM or SDRAM. When the pointers either become close to overrunning each other or close to the 20-ms AV divergence point, the DIV value should be increased or decreased incrementally (LSBs).

Table 31-1. HW_CLKCTRL_SAIF_DIV Values for Standard Sample Rates/ Oversample Base Rates

Sample Rate (Hz)	Oversample Base Rate Multiplier	saif_clk Required (MHz)	Desired Fractional Divisor	Closest Actual Fractional Divisor	DIV Binary Value	DIV Hex Value	Error (ppm)	DIV Adjustment Frequency (seconds) (Notes 1 and 2)	
								DIV < Actual	DIV > Actual
192000	512	98.304	0.2048000000	0.2048034668	0011010001101110	0x346E	16.9	347.3	1181.5
	384	73.728	0.1536000000	0.1535949707	0010011101010010	0x2752	32.7	610.8	300.3
176200	512	90.2144	0.1879466667	0.1879425049	0011000000011101	0x301D	22.1	903.2	338.8
	384	67.6608	0.1409600000	0.1409606934	0010010000010110	0x2416	4.9	193.5	4066.0
128000	512	65.536	0.1365333333	0.1365356445	0010001011110100	0x22F4	16.9	210.9	1181.5
	384	49.152	0.1024000000	0.1024017334	0001101000110111	0x1A37	16.9	151.4	1181.5
96000	512	49.152	0.1024000000	0.1024017334	0001101000110111	0x1A38	16.9	151.4	1181.5
	384	36.864	0.0768000000	0.0767974854	0001001110101001	0x13A9	32.7	610.8	120.5
88100	512	45.1072	0.0939733333	0.0939788818	0001100000001111	0x180F	59.0	193.5	338.8
	384	33.8304	0.0704800000	0.0704803467	0001001000001011	0x120B	4.9	94.5	4066.0
64000	512	32.768	0.0682666667	0.0682678223	0001000101111010	0x117A	16.9	96.8	1181.5
	384	24.576	0.0512000000	0.0511932373	0000110100011011	0x0D1B	132.1	151.4	120.5
48000	512	24.576	0.0512000000	0.0511932373	0000110100011011	0x0D1B	132.1	151.4	120.5
	384	18.432	0.0384000000	0.0384063721	0000100111010101	0x09D5	165.9	86.4	120.5
44100	512	22.5792	0.0470400000	0.0470428467	0000110000001011	0x0C0B	60.5	75.8	330.5
	384	16.9344	0.0352800000	0.0352783203	0000100100001000	0x0908	47.6	420.1	52.0
32000	512	16.384	0.0341333333	0.0341339111	0000100010111101	0x08BD	16.9	46.5	1181.5
	384	12.288	0.0256000000	0.0256042480	0000011010001110	0x068E	165.9	46.5	120.5
24000	512	12.288	0.0256000000	0.0256042480	0000011010001110	0x068E	165.9	46.5	120.5
	384	9.216	0.0192000000	0.0191955566	0000010011101010	0x04EA	231.4	86.4	35.5
22050	512	11.2896	0.0235200000	0.0235137939	0000011000000101	0x0605	263.9	75.8	52.0
	384	8.4672	0.0176400000	0.0176391602	0000010010000100	0x0484	47.6	420.1	24.5
16000	512	8.192	0.0170666667	0.0170593262	0000010001011110	0x045E	430.1	46.5	43.1
	384	6.144	0.0128000000	0.0128021240	0000001101000111	0x0347	165.9	19.5	120.5
12000	512	6.144	0.0128000000	0.0128021240	0000001101000111	0x0347	165.9	19.5	120.5
	384	4.608	0.0096000000	0.0095977783	0000001001110101	0x0275	231.4	86.4	14.7

Table 31-1. HW_CLKCTRL_SAIF_DIV Values for Standard Sample Rates/ Oversample Base Rates (continued)

11025	512	5.6448	0.0117600000	0.0117645264	0000001100000011	0x0303	384.9	21.9	52.0
	384	4.2336	0.0088200000	0.0088195801	0000001001000010	0x0242	47.6	420.1	11.9
8000	512	4.096	0.0085333333	0.0085296631	0000001000101111	0x022F	430.1	46.5	14.7
	384	3.072	0.0064000000	0.0063934326	0000000110100011	0x01A3	1026.2	19.5	14.7

1 HW_CLKCTRL_SAIF_DIV_FRAC_EN=1

2 Average elapsed time between each DIV adjustment to maintain audio within ± 20 ms of sample rate.

BITCLK is used to launch or capture each bit of the serial PCM data, while LRCLK clocks the individual left/right samples (transitions at every sample boundary). For transmit, BITCLK and LRCLK are always driven by the SAIF. However, there are two different clocking modes for receive. In master mode, the SAIF drives both BITCLK and LRCLK, while in slave clock mode it is the responsibility of the codec to drive BITCLK and LRCLK to the SAIF. Note that, for any of these modes, an alternate MCLK reference can be multiplexed out to a pin to drive the codec's main system clock.

In slave clocking mode, the SAIF configures the BITCLK and LRCLK pins as inputs, and the off-chip codec is responsible for driving both clocks to the SAIF. The SAIF synchronizes these inputs and uses them to determine when to latch serial PCM data from the ADC for receive. The codec must be configured to run BITCLK either at 32xFs for 16-bit operation, 64xFs for 17-bit through 24-bit operation, or 48xFs for certain codecs for 16-bit through 24-bit operation.

In master clocking mode, it is the responsibility of the SAIF to drive both BITCLK and LRCLK out to the off-chip codec. In this mode, BITCLK is again programmed to transition at a the standard 32x, 48x, or 64x the sample rate.

On the i.MX23, two SAIF modules are instantiated on-chip. However, due to pin multiplexing constraints, only one set of clock pins is provided for both ports. This means that only one of the two SAIFs can master or drive the clock pins at a time, and the other SAIF becomes a slave to the master. This also means that both SAIFs must operate at the same sample rate. Following are the valid configurations for SAIF1 and SAIF2 on the i.MX23:

- One SAIF in TX mode (is the default clock master) while the other SAIF is in RX slave mode and is internally controlled by the TX SAIF's BITCLK and LRCLK.
- One SAIF in RX master mode while the other SAIF is in RX slave mode and again is internally controlled by the RX master SAIF's BITCLK and LRCLK.
- Both SAIFs in RX slave mode, with BITCLK and LRCLK controlled by the off-chip codec.
- Only one SAIF used (any configuration).

For any of these configurations, MCLK can also be output via a multiplexed pin to provide the clock reference for the off-chip codec.

Configuring the SAIF for transmit makes it the master by default. However, for receive, the SAIF can either be master or slave. For master mode, it drives BITCLK and LRCLK to the pins the off-chip codec,

which uses the clocks to time when to serially transmit PCM data back to the SAIF. For slave mode, the BITCLK and LRCLK are pin inputs driven by the off-chip codec, and the SAIF uses these clocks to determine when to latch each incoming bit and push the assembled PCM data to the FIFO.

The DIGCTL module contains a number of register bits to define which of the two SAIFs is master, as well as the pin configuration for BITCLK, LRCLK, and the optional MCLK output. The HW_DIGCTL_CTRL_SAIF_CLKMST_SEL bit simply selects which of the two SAIFs is to function as clock master (0 = SAIF1 is clock master, 1 = SAIF2 is clock master). The HW_DIGCTL_CTRL_SAIF_CLKMUX_SEL bit field selects the clock direction as well as what pins (either BITCLK-only or both BITCLK and MCLK) are used (see Table 31-2).

Table 31-2. HW_DIGCTL_CTRL_SAIF_CLKMUX_SEL Programming

HW_SAIF_CLKMUX_SEL	PIN			MODE
	SAIF_MCLK_BITCLK	SAIF_ALT_BITCLK	SAIF_LRCLK	
00	MCLK OUT	BITCLK OUT	LRCLK OUT	TX/RX Master
01	BITCLK OUT	-	LRCLK OUT	TX/RX Master
10	MCLK OUT	BITCLK IN	LRCLK IN	RX Slave
11	BITCLK IN	-	LRCLK IN	RX Slave

HW_DIGCTL_CTRL_SAIF_ALT_BITCLK_SEL selects what pin BITCLK uses when both BITCLK and MCLK are selected.

See Chapter 37, “Pin Control and GPIO,” for instructions on which pins the two SAIFs use and how to configure them for operation.

The SAIF contains a four-entry FIFO for each channel pair that buffers data between the SAIF and the on-chip or off-chip RAM buffer used to supply or collect serial PCM audio data. Both the SAIF’s register and DMA interface are clocked by APBX clock. To ensure the SAIF FIFO does not overrun or underrun, the minimum APBX clock frequency to sample rate frequency ratio is 22:1. Therefore, if the sample rate is configured to 192 kHz, then APBX must be set to 4.224 MHz or greater.

31.2.2 Transmit Operation

If the APBX DMA is to supply PCM data to the SAIF FIFO, the user first configures its corresponding DMA channel and initializes the buffer(s) of PCM data that are to be played. Next the SAIF control register is initialized, selecting the frame format and number of channel pairs, clearing the CLKGATE bit, and setting the RUN bit.

Once running, the BITCLK, LRCLK, and optional MCLK pins begin to transition, and null data is output to the off-chip analog DAC. The SAIF DMA interface requests PCM samples until the FIFO(s) is/are filled, and continues to request a sample (or sample pair for 16-bit operation) whenever an empty FIFO

entry is available. Once valid PCM data resides within the bottom of the front channel pair FIFO, the current null sample left/right pair(s) are allowed to complete. At this point, the serialization frame control logic begins to output the first valid left sample.

In 16-bit operation, sample pairs are packed with the right samples occupying the upper halfword and left samples the lower halfword of the FIFO entries.

For 17-bit to 24-bit operation, each FIFO entry contains a sample that is right-justified (LSB in bit 0).

The first sample DMA-ed to the FIFO at the start of operation should always be a left sample, followed by a right, and so on. If four or six channel pairs are enabled, samples should be grouped with all left samples first, followed by all right samples (e.g., front left, surround left, then center, followed by front right, surround right then LFE, and so on). As long as data resides within the FIFO(s), valid sample pairs continue to be output. If the FIFO(s) ever underflow or overflow, an interrupt occurs. At this point, the system software should shut down the SAIF, clear the FIFO(s), and then cleanly resume operation because there is not a way to prevent left/right swap of the PCM channels after this point. If the FIFO does underflow, null samples are output until valid data once again resides within the bottom of the FIFO. Any PCM value that is written to a full FIFO is discarded, preventing the top entry from be overwritten.

When the RUN bit is cleared at the end of operation, all PCM data corresponding to one sample collection (either two, four, or six channel pairs) that are currently being transmitted are allowed to complete before operation ceases and the BITCLK, LRCLK, and SDATA pins stop transitioning.

Alternately, software can be used to maintain the FIFO(s) if the DMA is not used, either by responding to an interrupt that is issued whenever an empty FIFO entry exists, or by polling a FIFO status bit.

31.2.3 Receive Operation

If the APBX DMA is to collect PCM data from the SAIF FIFO(s), the user first configures its corresponding DMA channel and allocates the buffer(s) where PCM data is to be recorded. Next the SAIF control register is initialized, selecting the frame format and number of channel pairs, selecting whether the SAIF is BITCLK/LRCLK master or slave, clearing the CLKGATE bit, and setting the RUN bit.

Once running, the SAIF either waits until the BITCLK and LRCLK pins begin to transition (slave mode) or begins to toggle BITCLK and LRCLK (master mode). In either case, once an LRCLK edge that corresponds to the start of a left sample is detected, the SAIF frame-control logic begins to assemble the sample in its serial shift register. Each time the LRCLK pin toggles, a new sample is pushed to the FIFO(s).

In 16-bit operation, sample pairs are packed with the right samples occupying the upper halfword and left samples the lower halfword of the FIFO entries.

For 17-bit to 24-bit operation, samples are placed in individual FIFO entries, are right justified (LSB in bit 0), and the unused MSBs are zero-filled.

The first sample pushed to the FIFO at the start of operation is always a left sample, followed by a right, and so on. If 4 or 6 channel pairs are enabled, sample pairs are grouped when pushed to the FIFO with all left samples first, followed by all right samples (e.g., front left, surround left, then center, followed by front right, surround right then LFE, and so on). As long as the BITCLK and LRCLK pins continue to transition, data is collected within the FIFO. If the FIFO ever overflows or underflows, an interrupt occurs. At this point, the system software should shut down the SAIF, clear the FIFO(s), and then cleanly resume operation because there is not a way to determine left from right PCM channel data within the FIFO after this point. If the FIFO does overflow, any PCM value that is pushed to the full FIFO is discarded, not allowing the top entry to be overwritten. If the FIFO underflows, the read of the empty FIFO returns all zeros.

When the run bit is cleared at the end of operation, all PCM data corresponding to one sample collection (either two, four, or six channel pairs) that are currently being serially received are allowed to complete and are stored to the FIFO before operation ceases.

Software can be used to empty the FIFO if the DMA is not used, either by responding to an interrupt that is issued whenever an empty FIFO entry exists, or by polling a FIFO status bit.

31.2.4 DMA Interface

Both SAIFs on the i.MX23 are assigned to APBX DMA channels. See Chapter 11, “AHB-to-APBX Bridge with DMA,” for the DMA channel assignments.

Once the DMA channel and SAIF are programmed (except for the RUN bit), operation can be initiated either by setting the SAIF’s RUN bit or by signaling a kick from the DMA channel.

The HW_SAIF_CTRL_DMAWAIT_COUNT bit field can be programmed to wait 0 to 31 APBX clock cycles between each DMA request. This feature acts as a throttle on the bandwidth required by the SAIF to allow delays such that DMA requests from other modules can be serviced by the DMA controller.

31.2.5 PCM Data FIFO

The SAIF contains three 4-entry by 32-bit wide FIFOs. These FIFOs serve as a buffer to ensure data is not corrupted if the DMA cannot service the SAIF before the next sample or sample pair is processed by the SAIF. Access to the FIFOs is achieved via read/writes of the 32-bit HW_SAIF_DATA register. Writes place PCM values at the top of the FIFOs, and reads take them from the bottom of the FIFOs. Each FIFO is used to store different sets of left/right channel pairs. FIFO1 stores the stereo or front channels, FIFO2 the surround or rear channels, and FIFO3 the center and low frequency effect (LFE) or subwoofer channels (see Figure 31-1). Read/write accesses are made to the FIFOs in round-robin fashion such that all left channel samples are transferred first including the center channel, followed by all right channel samples including the LFE channel.

In 16-bit operation, sample pairs are packed with the right samples occupying the upper halfword and left samples the lower halfword of the FIFO entries.

For 17-bit to 24-bit operation, sample pairs are placed in individual FIFO entries, left channel first then followed by the right channel, and are right justified (LSB in bit 0) in each FIFO entry.

The FIFO underflow, overflow, and service interrupt status bits reside within the HW_SAIF_STAT register.

31.2.6 Serial Frame Formats

There are six types of serial PCM frames that can be transmitted or received. The three basic formats are I^2S, left-justified, and right-justified. Because there are two variations of a frame based on whether or not the data consumes the entire frame width, these three formats have two frame types each that make up the six basic frame types. One variation exists for 16-bit and 24-bit serial PCM data that consumes the whole frame width, and the other for 17-bit to 24-bit serial PCM data that does not. Recall that for 16-bit operation, BITCLK is either 32xFs or 48xFs, while for 17-bit to 24-bit, it is either 48xFs or 64xFs. These six types of serial PCM frames are shown in Figure 31-2.

- In left-justified (LJ) format, the serial PCM data is left-justified within the sample's start and end point indicated by the LRCLK. The first bit of PCM data is coincident with the first BITCLK period after LRCLK transitions.
- Conversely, in right-justified (RJ) format, the last bit of PCM data in a sample is coincident with the last BITCLK period before LRCLK transitions, indicating the start of the next sample.
- I^2S format is simply a variant of LJ format, in that the first BITCLK period after the LRCLK transitions, is a null wait state, followed by the first serial PCM bit during the next BITCLK period.

For both LJ and RJ formats, LRCLK is high for left samples and low for right samples. For I^2S, it is the opposite: LRCLK is low for left samples and high for right samples.

When the programmed data size is smaller than the frame size or number of BITCLKs per LRCLK, there are BITCLK periods within the sample frame in which no data is being transmitted or received. This occurs in 48xFs mode when that data is less than 24 bits, and for all data sizes allowed in 64xFs mode. For LJ and I^2S modes, this occurs after the sample has been transmitted or received, while for RJ mode, this occurs prior to the sample being transmitted or received.

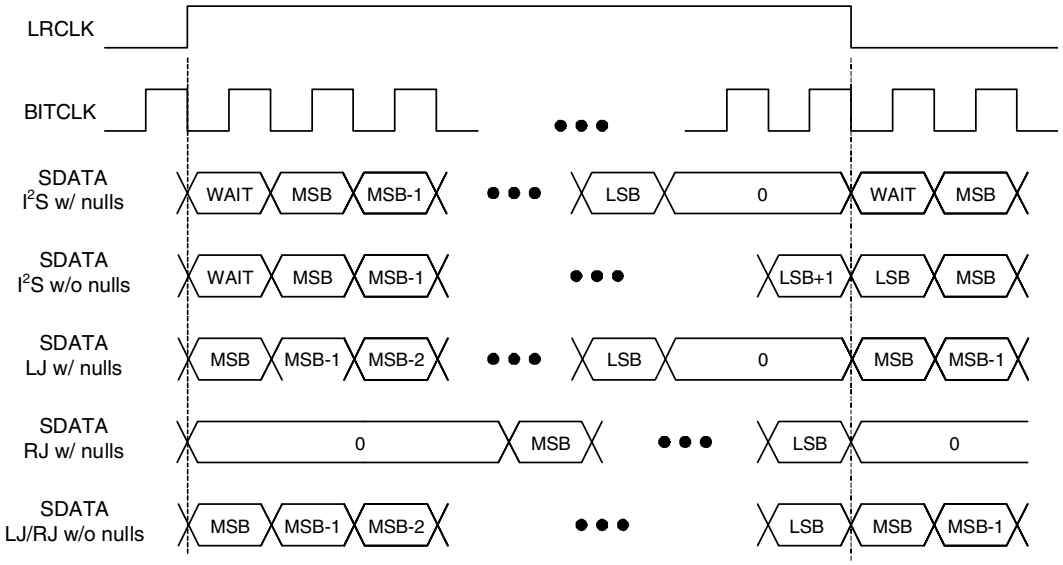
For 16-bit (32xFs mode) operation and 24-bit (48xFs mode) operation, data is always being transmitted, so that LJ and RJ modes are identical. I^2S is a special case for these modes. At the start of transmit or receive, the first BITCLK period after LRCLK transitions is a null or wait state cycle in which no PCM data is present. However, this means one too few BITCLK periods remain to transmit or receive data before LRCLK transitions. As a result, the protocol dictates that the last serial PCM bit of each sample be transmitted or received during the BITCLK wait state at the start of the *next* sample.

Additionally data can be programmed to be transmitted or received MSB or LSB first.

The bits to program frame format reside within the HW_SAIF_CTRL register.

31.2.7 Pin Timing

Figure 31-2 shows the six basic frame formats supported by the SAIF. Keep in mind that for 16-bit operation, BITCLK runs at either 32x or 48x the sample rate, and for 17-bit through 24-bit operation, it runs at either 48x or 64x the sample rate (i.e., clock frequency relationship of differing data sizes is not shown here).



Note: LRCLK_POLARITY=1 and BITCLK_EDGE=0 for this example.

Figure 31-2. Frame Formats Supported by SAIF

31.3 Programmable Registers

The following registers are available for CPU programmer access and control of the serial audio interface.

31.3.1 SAIF Control Register Description

The SAIF Control Register controls the frame format and operation of the three-wire serial audio interface.

HW_SAIF_CTRL	0x000
HW_SAIF_CTRL_SET	0x004
HW_SAIF_CTRL_CLR	0x008
HW_SAIF_CTRL_TOG	0x00C

Table 31-3. HW_SAIF_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	BITCLK_MULT_RATE			BITCLK_BASE_RATE	FIFO_ERROR_IRQ_EN	FIFO_SERVICE_IRQ_EN	RSRVD2			DMAWAIT_COUNT					CHANNEL_NUM_SELECT		RSRVD1	BIT_ORDER	DELAY	JUSTIFY	LRCLK_POLARITY	BITCLK_EDGE	WORD_LENGTH				BITCLK_48XFS_ENABLE	SLAVE_MODE	READ_MODE	RUN

Table 31-4. HW_SAIF_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	Setting this bit to 1 forces a reset to the entire block. SFTRST has no effect on CLKGATE. Also, the SFTRST bit may be written when CLKGATE=1. This bit must be cleared to 0 for normal operation.
30	**CLKGATE**	RW	0x1	This bit gates the clocks to the SAIF to save power when the clocks are not in use. When set to 1, this bit gates off the clocks to the block. When this bit is cleared to 0, the block receives its clocks for normal operation.
29:27	**BITCLK_MULT_RATE**	RW	0x0	BITCLK Mutiplier Rate. This bit field selects the multiple of the base frequency rate of BITCLK for transmit mode and receive master clock mode (READ_MODE=1, SLAVE_MODE=0), or if the alternate BITCLK pin is used, it selects the multiple of the base frequency rate of MCLK (any mode). When BITCLK_BASE_RATE = 0 (32x base rate): 000=512 x Fs, 001=256 x Fs, 010=128 x Fs, 011=64 x Fs, 100=32 x Fs, 101-111=reserved. When BITCLK_BASE_RATE = 1 (48x base rate): 000=384 x Fs, 001=192 x Fs, 010=96 x Fs, 011=48 x Fs, 100-111=reserved. When the SAIF_BITCLK_MCLK pin is used as BITCLK, this field should be programmed to 32x for 16-bit data, 48x for 16-bit to 24-bit data (BITCLK_48XFS_ENABLE=1), and 64x for 17-bit to 24-bit data, depending on the modes supported by the off-chip codec. When the SAIF_BITCLK_MCLK pin is used as MCLK (and the alternate BITCLK pin mux function is enabled), any oversample rate can be selected as dictated by the codec's required MCLK frequency. Note that the clock controller block should be programmed with the correct DIV value to produce the correct oversample clock base frequency (either 512x or 384x the sample rate) to the SAIF.

Table 31-4. HW_SAIF_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
26	**BITCLK_BASE_RATE**	RW	0x0	BITCLK Base Rate. This bit selects the base frequency rate at which the BITCLK pin toggles when configured as an output (either in transmit mode or in receive master clock mode), or if the alternate BITCLK pin is used, it selects the base frequency rate at which both the MCLK and alternate BITCLK pin toggles. 0 = BITCLK/MCLK base frequency rate is in multiples of 32x the sample rate. 1 = BITCLK/MCLK base frequency rate in in multiples of 48x the sample rate. This bit field is used in conjunction with the BITCLK_MULT_RATE field to program the BITCLK/MCLK output frequency.
25	**FIFO_ERROR_IRQ_EN**	RW	0x0	Set this bit to one to enable a SAIF interrupt request on FIFO overflow or underflow status condition.
24	**FIFO_SERVICE_IRQ_EN**	RW	0x0	Set this bit to one to enable a SAIF interrupt request to service the FIFO when it contains an empty entry (for transmit) or a full entry (for receive).
23:21	**RSRVD2**	RO	0x0	Reserved.
20:16	**DMAWAIT_COUNT**	RW	0x00	DMA Request Delay Count. This bit field specifies the number of APBX clock cycles (0 to 31) to delay after a DMA request has been serviced and before the next request is granted. This field acts as a throttle on the bandwidth consumed by the SAIF block. This field can be loaded by the DMA.
15:14	**CHANNEL_NUM_SELECT**	RW	0x0	Channel Number Select. This bit field selects the number of channel pairs (left and right) that are transmitted and/or received by the SAIF. 00 = One channel pair (stereo) 01 = Two channel pairs (front, surround) 10 = Three channel pairs (front, surround, center/lfe) 11 = Reserved
13	**RSRVD1**	RO	0x0	Reserved.
12	**BIT_ORDER**	RW	0x0	SAIF PCM Data Serial Bit Order. This bit selects whether PCM data is serially transmitted or received LSB or MSB first. 0 = MSB first 1 = LSB first Note that the two's complement audio data written to and read from the FIFO is always ordered MSB to LSB (LSB located in bit 0 for 17-bit through 24-bit operation, and in bits 15 and 0 for 16-bit operation).
11	**DELAY**	RW	0x0	SAIF Data Delay. In left-justified mode, this bit selects whether or not serial PCM data transmission/reception is delayed by one BITCLK period each LRCLK frame (to generate I2S serial operation). 0 = Data is not delayed. MSB of serial sample is output/input coincident with LRCLK transition (left-justified mode) 1 = Data is delayed one BITCLK period. MSB of serial serial sample is output/input one BITCLK period after LRCLK transitions (I2S mode). Note that this bit is ignored in right-justified mode (JUSTIFY=1).

Table 31-4. HW_SAIF_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
10	**JUSTIFY**	RW	0x0	SAIF Data Justification. This bit selects whether serial PCM data is left- or right-justified within each sample's LRCLK frame. 0 = Data is left-justified (start or MSB of serial sample transmission/reception coincides with LRCLK transition) 1 = Data is right-justified (end or LSB of serial sample transmission/reception coincides with LRCLK transition).
9	**LRCLK_POLARITY**	RW	0x0	SAIF LRCLK Polarity Select. This bit selects which LRCLK levels (high/low) correspond to left and right PCM samples. 0 = Left low/right high 1 = Left high/right low.
8	**BITCLK_EDGE**	RW	0x0	SAIF BITCLK Edge Select. For both transmit and receive, this bit selects the BITCLK edge upon which serial PCM data changes. For receive, data is sampled and stored to the receive FIFO on the opposite edge as selected by BITCLK_EDGE that corresponds to the midpoint of the data. 0 = TX: data is driven (changes) on falling-edges of BITCLK; RX: data is sampled on rising-edges of BITCLK 1 = TX: data is driven (changes) on rising-edges of BITCLK; RX: data is sampled on falling-edges of BITCLK.
7:4	**WORD_LENGTH**	RW	0x0	SAIF data size. Selects one of nine PCM data widths from 16-bit to 24-bit to serially input or output from/to a codec. 17-bit to 24-bit PCM data should be right-justified (LSB in bit 0) when it is DMAed or written to the HW_SAIF_DATA register. These samples should be interleaved starting with a left sample first, followed by a right, then left and so on. For 16-bit PCM data, stereo pairs should be constructed with the right sample in the upper half-word (bits 31-16) and the left sample in the lower half word (bits 15-0). 0000 = 16-bit 0001 = 17-bit 0010 = 18-bit 0011 = 19-bit 0100 = 20-bit 0101 = 21-bit 0110 = 22-bit 0111 = 23-bit 1000 = 24-bit 1001-1111 = Reserved but defaults to 24-bit.
3	**BITCLK_48XFS_ENABLE**	RW	0x0	BITCLK 48x Sample Rate Enable. For 384x base frequency multiples, this bit enables generation of 48 BITCLKs per sample pair (24 BITCLKs per channel or LRCLK transition) when the SAIF is BITCLK master. This bit is ignored for the following cases: BITCLK_BASE_RATE=0, or READ_MODE=1, or READ_MODE=0 and SLAVE_MODE=1.

Table 31-4. HW_SAIF_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**SLAVE_MODE**	RW	0x0	SAIF Receive Master/Slave Clock Mode Select. For receive operation, this bit selects whether BITCLK and LRCLK are driven to the off-chip codec or uses the two clock pins as inputs to determine when to receive data from the codec. In receive master mode (SLAVE_MODE=0), BITCLK and LRCLK are output to the codec. When SLAVE_MODE=0, both BITCLK and LRCLK start to transition immediately after the RUN bit is set. Note that when in transmit mode or receive master mode, the user must configure the SAIF's clock controls within the clock controller to the correct oversample rate (see BITCLK_BASE_RATE/BITCLK_MULT_RATE above). 0 = Master mode. SAIF drives BITCLK and LRCLK 1 = Slave mode. SAIF uses BITCLK and LRCLK as inputs to determine when to sample input PCM data. Note that is bit is ignored in transmit operation (READ_MODE=0).
1	**READ_MODE**	RW	0x0	SAIF Transmit/Receive Select. This bit selects whether the SAIF block transmits to an off-chip DAC (write mode) or receives from an off-chip ADC (read mode). The selected mode (TX or RX) starts operation once the RUN bit is set. 0 = TX or write mode 1 = RX or read mode
0	**RUN**	RW	0x0	Setting this bit to one causes the SAIF to begin transmitting or receiving serial PCM data, depending on the programming of the READ_MODE bit. For transmit, when this bit is cleared, operation ends after transmission of the current active channel set (pairs from all enabled channels) from the FIFO. If the FIFO is already empty and the RUN bit is cleared, operation halts immediately and the LRCLK and BITCLK pins stop transitioning. For receive, when the RUN bit is cleared, reception ends after the current active channel set (pairs from all enabled channels) are pushed to the FIFO. Note for 4- and 6-channel operation, clearing the RUN bit means that the SAIF does not stop until the corresponding audio samples for all 4 or 6 channels have been transmitted or received.

DESCRIPTION:

The SAIF Control Register is used to configure the SAIF's input/output frame format, MCLK, BITCLK and LRCLK, and interrupt enables.

EXAMPLE:

```
HW_SAIF_CTRL.RUN = 1; // start SAIF operation
```

31.3.2 SAIF Status Register Description

The SAIF Status Register provides status of key hardware components required by software of the SAIF module.

HW_SAIF_STAT	0x010
HW_SAIF_STAT_SET	0x014
HW_SAIF_STAT_CLR	0x018
HW_SAIF_STAT_TOG	0x01C

Table 31-5. HW_SAIF_STAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PRESENT	RSRVD2														DMA_PREQ	RSRVD1									FIFO_UNDERFLOW_IRQ	FIFO_OVERFLOW_IRQ	FIFO_SERVICE_IRQ	RSRVD0			BUSY

Table 31-6. HW_SAIF_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**PRESENT**	RO	0x1	This bit is set to 1 in products in which SAIF is present.
30:17	**RSRVD2**	RO	0x0	Reserved.
16	**DMA_PREQ**	RO	0x0	DMA Request Status. This read-only bit reflects the current state of the SAIF's DMA request signal. DMA requests are issued any time the request signal toggles.
15:7	**RSRVD1**	RO	0x0	Reserved.
6	**FIFO_UNDERFLOW_IRQ**	RW	0x0	This bit is set by hardware if the FIFO underflows during SAIF operation. Underflow occurs whenever a read or pop is attempted on an empty FIFO. This occurs in transmit mode when the FIFO is not filled in time and the hardware tries to pop a sample from the bottom of the FIFO. It also occurs in receive mode when the DMA or software attempts to read data from an empty FIFO. Reset this bit by writing a one to the SCT clear address space and not by a general write.
5	**FIFO_OVERFLOW_IRQ**	RW	0x0	This bit is set by hardware if the FIFO overflows during SAIF operation. Overflow occurs whenever a write or push is attempted to a full FIFO. This occurs in transmit mode if the FIFO is full and software or the DMA attempts to write additional data to the top of the FIFO. It also occurs in receive mode when the DMA or software does not respond in time to a service request, and the hardware attempts to push data to a full FIFO. Reset this bit by writing a one to the SCT clear address space and not by a general write.

Table 31-6. HW_SAIF_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
4	**FIFO_SERVICE_IRQ**	RO	0x0	This bit is set by hardware when the FIFO requires service. FIFO service requests are made when an empty entry exists during transmit or a full entry exists during receive. A DMA request is generated (DMA_PREQ toggles) each time this bit is set. Once the DMA or software has serviced the request and the FIFO is filled (TX) or emptied (RX), this bit is automatically cleared. This interrupt can be used by software to trigger the manual movement of samples from/to the SAIF's FIFO to/from a memory buffer when the SAIF's DMA channel is not used.
3:1	**RSRVD0**	RO	0x0	Reserved.
0	**BUSY**	RO	0x0	This bit indicates when the SAIF is actively transmitting/receiving serial PCM audio data from/to its FIFO(s). For transmit, it is automatically set when the first sample from the FIFO begins to be output by the serial shifter. For receive, it is set coincident with the RUN bit beging set as serial receive begins immediately. After the RUN bit is cleared and the serial shifter becomes inactive (end of the current sample set), this bit is automatically cleared.

DESCRIPTION:

The SAIF Status Register provides the status of interrupt requests and active operation of the SAIF.

EXAMPLE:

```
unsigned TestBit = HW_SAIF_STAT.PRESENT;
```

31.3.3 SAIF Data Register Description

The SAIF Data Register is used to either write PCM data samples to the top of the SAIF FIFOs for transmit, or read PCM samples from the bottom of the FIFOs during receive. 32-bit values written/read to/from this register contain either one 17-bit to 24-bit sample or two 16-bit samples.

HW_SAIF_DATA	0x020
HW_SAIF_DATA_SET	0x024
HW_SAIF_DATA_CLR	0x028
HW_SAIF_DATA_TOG	0x02C

Table 31-7. HW_SAIF_DATA

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PCM_RIGHT																PCM_LEFT															

Table 31-8. HW_SAIF_DATA Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**PCM_RIGHT**	RW	0x0000	For 16-bit mode, this field contains the entire right channel sample. For 17-bit through 24-bit modes, this field contains 1 through 8 of the MSBs of the sample (either left or right).
15:0	**PCM_LEFT**	RW	0x0000	For 16-bit mode, this field contains the entire left channel sample. For 17-bit through 24-bit modes, this field contains the 16 LSBs of the sample (either left or right).

DESCRIPTION:

In transmit mode, writing a value to this register causes it to push the value to the top of the FIFO. In receive mode, reads cause values to be popped from the bottom of the FIFO. Writing to a full FIFO does not change the contents of the FIFO's top entry, and reading from an empty FIFO returns all zeros. For 16-bit operation, the left sample should be written to the register's lower half word, and the right to the upper half word. For all other data sizes, PCM audio values should be right-justified within the 32-bit word. Three FIFOs exist, one for each channel pair. If two-channel operation is enabled, only the stereo or front FIFO is accessed. Four-channel opearation causes both the front and surround FIFOs to be accessed. Six-channel operation uses all three FIFOs: front, surround, and center/LFE FIFOs. For both transmit and receive FIFO accesses, left and right samples should be interleaved, starting with all left samples for a given sample time, followed by all right samples (e.g., left front, left surround, center, right front, right surround, LFE (subwoofer), and so on). Operation should always start with the left samples. All left or all right channels for a given sample collection in time are received/transmitted simultaneously (e.g., for 6-channel mode, three PCM audio samples are popped from the FIFO prior to serialization and transmission). For mono operation, use the left channel to transmit/receive PCM audio, while the right channel should be zero-filled (TX) or discarded (RX). For transmit, if the FIFO is empty when operation begins, null (zero) data is output until the FIFO contains valid PCM data. Once a complete set of left samples resides within the bottom of the active FIFOs, the SAIF waits for the current collection of null samples (all left and right samples for a given time) to complete before transmitting the left sample collection from the FIFOs.

EXAMPLE:

```
HW_SAIF_DATA = 0x12345678; // write 0x1234 to the right channel and 0x5678 to the left channel in 16
bit mode
HW_SAIF_DATA = 0x12345678; // write 0x12345678 to either the left or right channel in 32 bit per
sample mode.
```

31.3.4 SAIF Version Register Description

The SAIF Version Register is read-only and is used for debug to determine the implementation version number for the block.

HW_SAIF_VERSION 0x030

Table 31-9. HW_SAIF_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 31-10. HW_SAIF_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x01	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x01	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
if (HW_SAIF_VERSION.B.MAJOR != 1)
Error();
```

SAIF Block v1.1, Revision 2.1

Chapter 32
Power Supply

This chapter describes the power supply subsystem provided on the i.MX23. It includes sections on the DC-DC converter, linear regulators, PSWITCH pin functions, battery monitor and charger, and silicon speed sensor. Programmable registers are described in Section 32.11, "Programmable Registers."

32.1 Overview

The i.MX23 integrates a comprehensive power supply subsystem, including the following features:

- One integrated DC-DC converter that supports Li-Ion batteries.
- Four linear regulators directly power the supply rails from 5 V.
- Linear battery charger for Li-Ion cells.
- Battery voltage and brownout detection monitoring for VDDD, VDDA, VDDIO, VDD4P2 and 5V supplies.
- Integrated current limiter from 5 V power source.
- Reset controller.
- System monitors for temperature and speed.
- Generates USB-Host 5V from Li-Ion battery (using PWM).
- Support for on-the-fly transitioning between 5V and battery power.
- VDD4P2, a nominal 4.2 V supply, is available when the i.MX23 is connected to a 5 V source and allows the DCDC to run from a 5 V source with a depleted battery.
- The 4.2 V regulated output also allows for programmable current limits:
  - Battery Charge current + DCDC input current < the 5 V current limit
  - DCDC input current (which ultimately provides current to the on-chip and off-chip loads) as the priority and battery charge current will be automatically reduced if the 5V current limit is reached

The i.MX23 power supply is designed to offer maximum flexibility and performance, while minimizing external component requirements. Figure 32-1 shows a functional block diagram of the power supply components including switching converters, five linear regulators, battery charge support, as well as battery monitoring, supply brownout detection, and silicon process/temperature sensors. This figure can be used to understand which register and status bits relate to which subsystems, but it is not intended to be a complete architecture description.

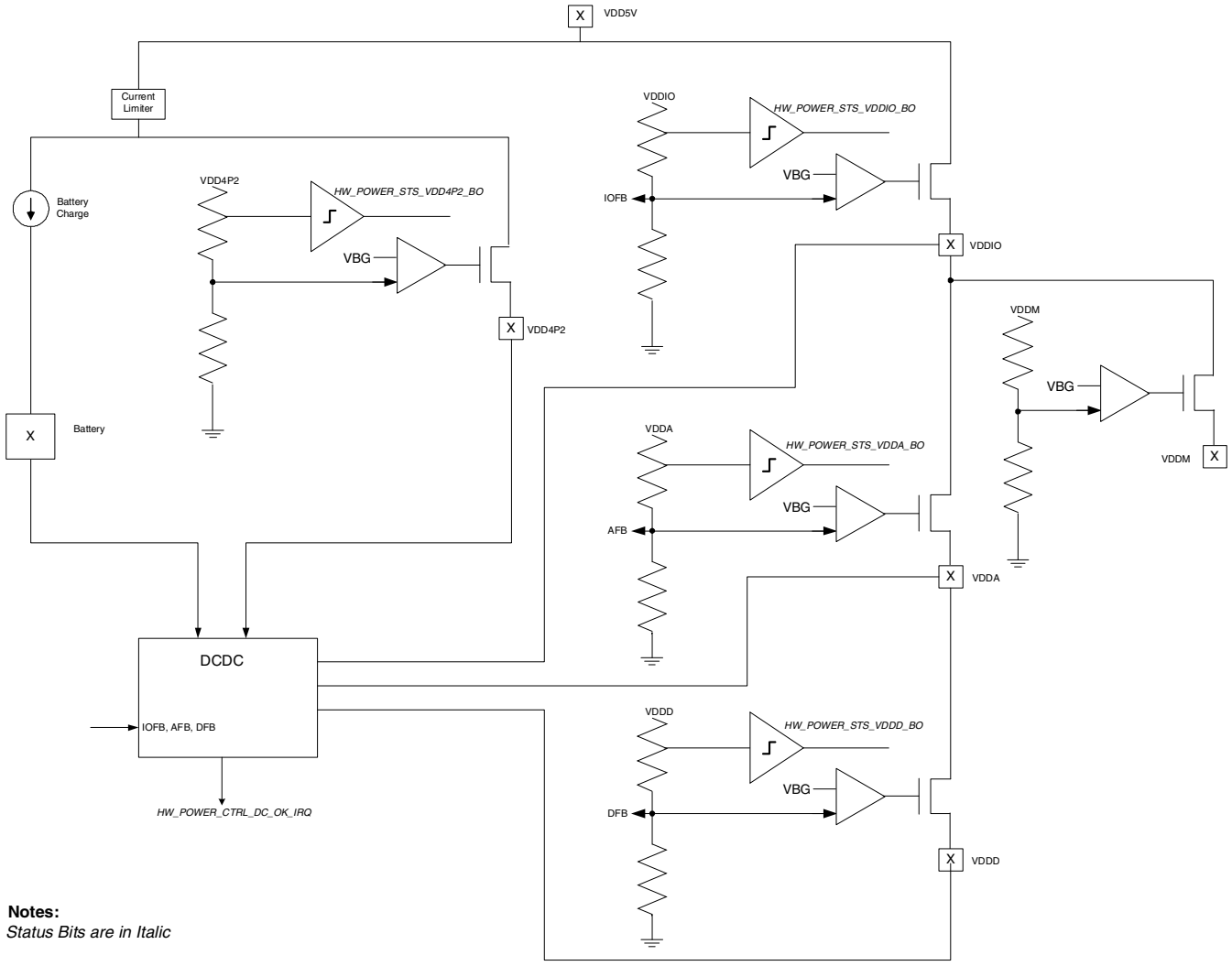


Figure 32-1. Power Supply Block Diagram

32.2 DC-DC Converters

The DC-DC converters efficiently scale battery voltage (or a regulated 4.2 V derived from a 5 V source) to the required supply voltages. The DC-DC converters include several advanced features:

- Single inductor architecture
- Programmable output voltages
- Programmable brownout detection thresholds
- Pulse frequency modulation (PFM) mode for low-current load operation

32.2.1 DC-DC Operation

The i.MX23 DC-DC converter enables a low-power system and features programmable output voltages and control modes. Most products adjust VDDD dynamically to provide the minimum voltage required

for proper system operation. VDDIO and VDDA are typically set once during system initialization and not changed during operation.

32.2.1.1 Brownout/Error Detection

The power subsystem has several mechanisms active by default that safely return the device to the off state if any one of the following errors or brownouts occur:

- The crystal oscillator frequency is detected below a certain threshold—This threshold is process- and voltage-sensitive, but will always be between 100 kHz and 2 MHz. This feature can be disabled in the DISABLE_XTALOK field in the HW_RTC_PERSISTENT0 register.
- The battery voltage falls below the battery brownout level (field BRWNOUT_LVL in HW_POWER_BATTMONITOR)—This feature is disabled by clearing PWDN_BATTBRNOUT in the same register.
- 5 V is detected, then removed—This feature is disabled by clearing HW_POWER_5VCTRL_PWDN_5VBRNOUT.

All three mechanisms are active by default to ensure that the device always has a valid transition to a known state in case the power source is unexpectedly removed before software has completed system configuration. Software will typically disable the functionality of PWDN_5VBRNOUT and PWDN_BATTBRNOUT after system configuration is complete, as shown in Figure 32-2. System configuration generally includes setting up brownout detection thresholds on the supply voltages, battery, etc. to obtain the desired system operation as the battery or power source is depleted or removed.

Typically, each output target voltage is set to some voltage margin above the minimum operating level via the TRG field in the HW_POWER_VDDDCTRL, HW_POWER_VDDACTRL, and HW_POWER_VDDIOCTRL registers. The brownout detection threshold is also set via the BO_OFFSET field in the same three registers. The BO_OFFSET field determines how far the brownout voltage is below the output target voltage for each supply and might typically be set 75–100 mV below the target voltage. If the voltage drops to the brownout detector's level, then it optionally triggers a CPU Fast Interrupt (FIQ). The CPU can then alleviate the problem and/or shut down the system elegantly.

To eliminate false detection, the brownout circuit filters transient noise above 1 MHz. Any system with an i.MX23 should include at least 33µF of decoupling capacitance on the VDDIO, VDDA, and VDDD power rails. The capacitors should be arranged to filter supply noise in the 1-MHz and higher frequencies. See Figure 32-2.

32.2.1.2 DC-DC Extended Battery Life Features

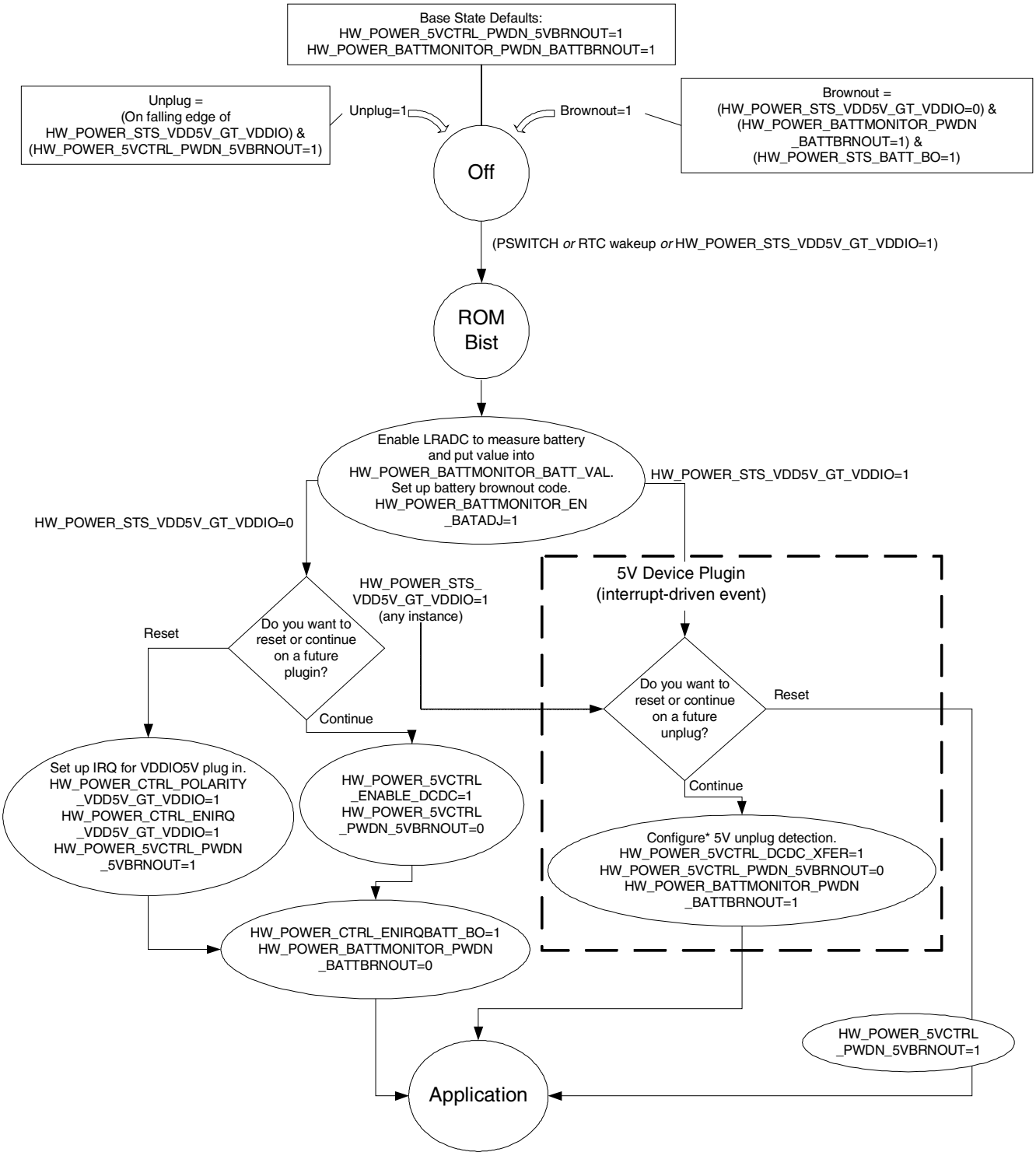
The DC-DC converter has several other power-reducing programmable modes that are useful for maximizing battery life:

- **Li-Ion Buck/Boost**—The Li-Ion battery configuration supports buck/boost operation, which means that a VDDIO voltage can be supported that is higher than the input Li-Ion battery voltage.

This is important to maximize battery life in all applications, but is crucial in hard drives that have large transient current requirements.

- **Transient Loading Optimizations**—Several new incremental improvements have been made to the control architecture of the switching converters. At this time, Freescale recommends setting the following bits via software in HW_POWER_LOOPCTRL to obtain maximum efficiency and minimum supply ripple: TOGGLE_DIF, EN_CM_HYST, and EN_RCSCALE=0x1**.** Also, set HW_POWER_BATTMONITOR_EN_BATADJ. The complete settings to optimize transient loading will be dependent on the application, component selection, and board layout.
- **Pulse Frequency Modulation (PFM)**—PFM, also known as pulse-skipping mode, is used to reduce power consumed by the DC-DC converter when the voltage outputs are lightly-loaded at a cost of higher transient noise. The DC-DC converter can be separately placed in PFM mode via the EN_DC_PFM bit in HW_POWER_MINPWR.
- **DC-DC Switching Frequency**—The standard DC-DC switching frequency is 1.5 MHz, which provides a good mix of efficiency and power output. The frequency can be reduced to 750 kHz to reduce operating current in some light load situations via DC_HALFCLK in HW_POWER_MINPWR. The DC-DC converter can also be programmed to a frequency that is based on the PLL using the SEL_PLLCLK and FREQSEL fields in the HW_POWER_MISC register.
- **DC-DC Converter Power Down**—If the system is to operate from linear regulators or an external power supply, then the internal DC-DC converters can be powered down via DC_STOPCLK in HW_POWER_MINPWR. This bit is not intended to power down the whole system. Use the HW_POWER_RESET bits to power the system off.

The DC-DC converter can be programmed to run off of the battery or the 4.2 V regulated voltage supply. It can be set up to oppportunistically draw power from either source. This allows the system to draw up to the USB current limit from the 5 V supply and take the balance of the required power from the battery.



Note: See Section 32.3.2.1, “Battery Power to 5-V Power”.

Figure 32-2. Brownout Detection Flowchart

32.3 Linear Regulators

The i.MX23 integrates four linear regulators that can be used to directly power the supply rails when 5 V is present. All of these regulators have an output impedance of approximately one-quarter ohm. This means that the measured output voltage will be slightly dependent on the current consumed on each supply. Architecturally, one regulator generates VDDIO from the VDD5V pin, one generates VDDA from VDDIO, one generates VDDM from VDDIO, and the other generates VDDD from the VDDA supply. Therefore, all of the current is supplied by the VDD5V->VDDIO regulator.

In addition, the i.MX23 integrates a 4.2 V regulator enables the DCDC converter to run off the power supplied by 5 V. This allows DCDC converter operation when 5 V is present and the battery is exhausted. In addition, the path can provide better power consumption as compared to the other linear regulator generated supplies. a current limiter is implemented in series with the 4.2 V regulator and the battery supply. The intent is to limit the $I_{battery}$ plus I_{4P2} to be less than I_{limit}. The 4.2 V regulator has priority on the current and will dynamically reduce the battery charge current in order to preserve the I_{limit} and the regulated 4.2 V.

It should also be noted that the VDD5V voltage can dynamically change as the product is plugged into a USB port or other 5-V supply. The i.MX23 is programmable to provide a variety of behaviors when the VDD5V supply becomes valid or invalid, as well as to support operation via USB or external power.

32.3.1 USB Compliance Features

Upon connection of 5 V to the powered-down device, the linear regulators will automatically power up the device. To meet USB inrush specifications, linear regulators have a current limit which is <100 mA, nominally 50 mA, and is active by default. This current limit is disabled in hardware after power-up, but can be re-enabled via the HW_POWER_5VCTRL_ENABLE_ILIMIT by software. System designers must understand that the current inrush limit during 5-V power-up places restrictions on application current consumption until it is disabled. Specifically, after connection to 5V, if the system draws more current than the current limit allows, the startup sequence does not complete and the ROM code does not execute.

Further, USB Host implies that B-devices must draw very little current from 5V. This requirement can be met by setting HW_POWER_5VCTRL_ILIMIT_EQ_ZERO when the Host application is active. The comparators required for Host capability can be enabled by HW_POWER_5VCTRL_PWRUP_VBUS_CMPS**.** It is also possible to change the threshold of the Vbus valid comparator using HW_POWER_5VCTRL_VBUSVALID_TRSH**.**

If very low power operation is required, as in USB suspend, then the circuits required to elegantly switch to the DC-DC converter may have to be powered off. In those cases, the system must fully power down after VDD5V becomes invalid. It can auto restart with the DC-DC converter if HW_RTC_PERSISTENT0_AUTO_RESTART is set.

32.3.2 5V to Battery Power Interaction

The i.MX23 supports several different options related to the interaction of the switching converters with the linear regulators. The two primary options are a reset on 5V insertion/removal or a handoff to the DC-DC converters that is invisible to the end-user of the application. Figure 32-3 includes these two options as the two system architecture decision boxes.

32.3.2.1 Battery Power to 5-V Power

By default, the DC-DC converter turns off when VDD5V becomes valid and the system does not reset. If the system is operating from the DC-DC converter and using more current than the linear regulators can supply, then the VDDD, VDDA, and VDDIO rails will droop when 5V is attached and the system may brownout and shut down. To avoid this issue, set the ENABLE_DCDC bit and set the LINREG_OFFSET fields to 0b2 in anticipation of VDD5V becoming present. The ENABLE_DCDC bit will cause the DC-DC converter to remain on even after 5V is connected and, thus, guarantee a stable supply voltage until the system is configured for removal of 5V. The LINREG_OFFSET fields = 0b2 cause the linear regulators to regulate to a lower target voltage than the switching converter and prevent unwanted interaction between the two power supplies. After the system is configured for removal of 5V, ENABLE_DCDC can be set low and ENABLE_ILIMIT set low in register HW_POWER_5VCTRL to allow the linear regulators to supply the system power, if desired.

32.3.2.2 5-V Power to Battery Power

Configuring the system for a 5-V-to-battery power handoff requires setup code to monitor the battery voltage as well as detect the removal of 5V.

Monitoring the battery voltage is performed by the LRADC. Typically, this involves programming the LRADC registers to periodically monitor the battery voltage as described in Chapter 33, “Low-Resolution ADC and Touch-Screen Interface.” The measured battery voltage should be written into the HW_POWER_BATTMONITOR register field BATT_VAL using the AUTOMATIC field in the HW_LRADC_CONVERSION register. Also**,** configuring battery brownout should be performed so that the system behaves as desired when 5V is no longer present and the battery is low.

The recommended method to detect removal of 5V requires setting VBUSVALID_5VDETECT and programming the detection threshold VBUSVALID_TRSH to 0x1 in HW_POWER_5VCTRL**.** Next, in order to minimize linear regulator and DC-DC converter interaction, it is necessary to set the LINREG_OFFSET = 0b2 in the HW_POWER_VDDIOCTRL, HW_POWER_VDDACTRL, and HW_POWER_VDDDCTRL registers**.** Finally, set DCDC_XFER and clear PWDN_5VBRNOUT in the HW_POWER_5VCTRL register. This sequence is important because it is safe to disable the powerdown-on-unplug functionality of the device only after the system is completely ready for a transition to battery power.

32.3.2.3 5-V Power and Battery Power

Battery charge can also be enabled to provide additional power efficiency when connected to 5V. If the DCDC switching converter is also enabled, the buck switching converters will efficiently convert the battery voltage to the desired VDDA, VDDD, and VDDIO voltages (instead of using the less efficient internal linear regulators).

32.3.3 Power-Up Sequence

The DC-DC converter controls the power-up and reset of the i.MX23. The power-up sequence begins when the battery is connected to the BATT pin of the device (or a 5V source is connected to the VDD5V pin). Either the BATT pin or VDD5V provides power to the DC-DC startup circuitry, the crystal oscillator, and the real-time clock. This means that the crystal oscillator can be running, if desired, whenever a battery is connected to BATT pin. This feature allows the real-time clock to operate when the chip is in the off state. The crystal oscillator/RTC is the only power drain on the battery in this state and consumes only a very small amount of power. During this time, the VDDIO, VDDD and VDDA supplies are held at ground. This is the off state that continues until the system power up begins.

Power-up can be started with one of several events:

- PSWITCH pin >= minimum MID level PSWITCH for 100 ms (see “Characteristics & Specifications” chapter)
- VDD5V power pin >= minimum VDD5V voltage for 100 ms (see “Characteristics & Specifications” chapter)
- Real-time clock alarm wakeup

When a power-up event has occurred, if VDD5V is valid, then the on-chip linear regulators charge the VDDD, VDDA and VDDIO rails to their default voltages. If VDD5V is not valid, then the DC-DC supplies the VDDD, VDDA, and VDDIO rails. When the voltage rails have reached their target values, the digital logic reset is deasserted and the CPU begins executing code. If the power supplies do not reach the target values by the time PSWITCH is deasserted or 5V is removed, the system returns to the off state.

The power-up time is dependent on the VDDD/VDDA/VDDIO load and battery or VDD5V voltage, but should be less than 100 ms. The VDDD//VDDAVDDIO load should be minimal during power up to ensure proper startup of the DC-DC converter.

There is an integrated 5-KΩ resistor that can be switched in between the VDDXTAL pin and PSWITCH. If enabled using HW_RTC_PERSISTENT0_AUTO_RESTART, then the device immediately begins the power-up sequence after power-down.

32.3.4 Power-Down Sequence

Power-down is also controlled by the DC-DC converters. When the DC-DC converter detects a power-down event, it returns the player to the off state described above. The power-down sequence is started when one of these events occurs:

- HW_POWER_RESET_PWD bit set while the register is unlocked.
- Error condition occurs, as described in Section 32.2.1.1, "Brownout/Error Detection."
- The watchdog timer expires while enabled.
- Fast falling edge (<10 ns) on PSWITCH pin.

The HW_POWER_RESET_PWD_OFF bit disables all power-down paths except for the watchdog timer when it is set.

The lower 16 bits of the HW_POWER_RESET register can only be written if the value 0x3E77 is placed in the unlock field.

An external capacitor on the PSWITCH can be used to prevent an unwanted power down due to falling edges. This can also be disabled in register HW_POWER_RESET_PWD_OFF.

32.3.4.1 Powered-Down State

While the chip is powered down, the VDDIO, VDDD and VDDA rail are pulled down to ground. The crystal oscillator and the RTC can continue to operate by drawing power from the BATT pin. See Chapter 23, "Real-Time Clock, Alarm, Watchdog, Persistent Bits," for more information about operating a crystal and the RTC in the powered-down state.

32.3.5 Reset Sequence

Figure 32-3 shows a flowchart for the power-up, power-down, and reset sequences. A reset event can be triggered by unlocking the HW_POWER_RESET register and setting the HW_CLKCTRL_RESET_DIG bit. This reset affects the digital logic only, although the digital logic also includes most of the registers that control the analog portions of the chip. The persistent bits within the RTC block and the power module control bits are not reset using this method. To reset the analog as well as the digital logic, set the HW_CLKCTRL_RESET_CHIP bit. The DC-DC converter and/or linear regulators continue to maintain the power supply rails during the reset.

32.4 PSWITCH Pin Functions

The PSWITCH pin has several functions whose operation is determined by the i.MX23-based product's hardware and software design.

32.4.1 Power On

When the PSWITCH pin voltage is higher than the minimum MID level (as specified in Table 2-3., "Recommended Power Supply Operating Conditions") for >100 ms, the DC-DC converter begins its startup routine. This is the primary method of starting the system via PSWITCH. All products based on the i.MX23 must have a mechanism of bringing PSWITCH high to power up via an always-present supply (e.g, battery or VDDXTAL).

32.4.2 Power Down

If the PSWITCH pin voltage has a falling edge faster than 15 ns, then this sends a power-down request to the DC-DC converter. The fast-falling-edge power-down may be blocked by the HW_POWER_RESET_PWD_OFF function. The fast-falling edge can also be prevented by placing an RC filter on the PSWITCH pin. Most i.MX23-based systems do not use the PSWITCH fast-falling-edge power-down and include the RC filter to prevent it from occurring accidentally.

32.4.3 Software Functions/Recovery Mode

When the PSWITCH pin voltage is pulled up to the minimum MID level (as specified in Table 2-3., "Recommended Power Supply Operating Conditions") the lower HW_POWER_STS_PSWITCH bit is set. Software can poll this bit and perform a function as desired by the product designer. Example functions include a play/pause/power-down button, delay for startup, etc.

When the PSWITCH pin is connected to VDDIO through a current limiting resistor, the upper HW_POWER_STS_PSWITCH bit is also set. If this bit is set for more than five seconds during ROM boot, the system executes the Freescale USB Firmware Recovery function, as described in Chapter 35, "Boot Modes." If the product designer does not wish to use Freescale USB Firmware Recovery, the product can be designed to not assert a voltage higher than the maximum MID level (as specified in Table 2-3., "Recommended Power Supply Operating Conditions") on the PSWITCH pin.

Refer to the Freescale i.MX23 reference schematics for example configurations of the PSWITCH pin.

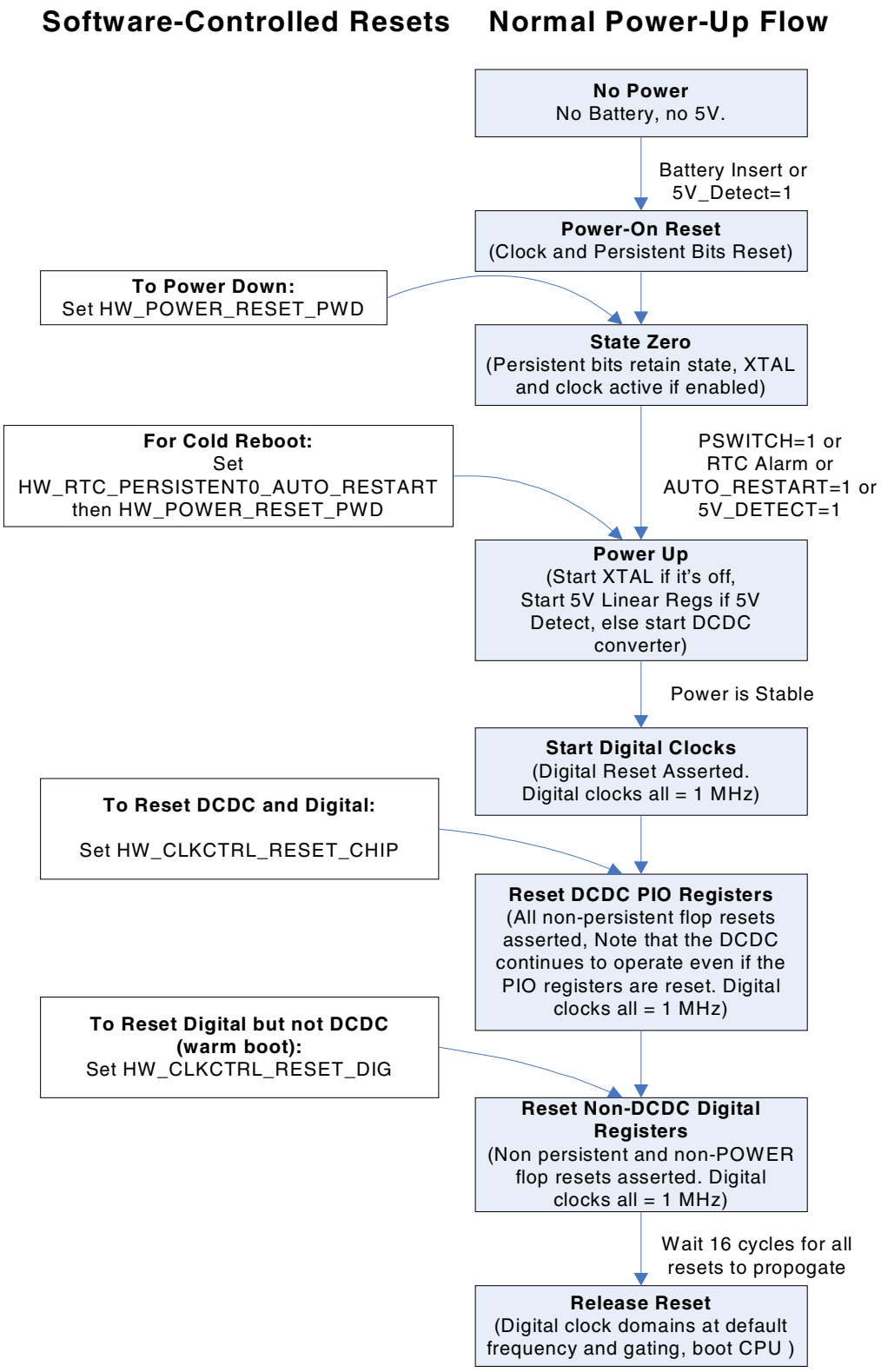


Figure 32-3. Power-Up, Power-Down, and Reset Flow Chart

32.5 Battery Monitor

The power control system includes a battery monitor. The battery monitor has two functions: battery brownout detection and battery voltage feedback to the DC-DC converter.

If the battery voltage drops below the programmable brownout, then a fast interrupt (FIQ) can be generated for the CPU. Software typically uses the LRADC to monitor the battery voltage and shut down elegantly while there is a minimal operating margin. But, if an unexpected event (such as a battery removal) occurs, then the system needs to be placed immediately in the off state to ensure that it can restart properly. The brownout is controlled in the HW_POWER_BATTMONITOR register. The IRQ must also be enabled in the interrupt collector.

To enable optimum performance over the battery range, the DC-DC converter needs to be provided with the battery voltage, which is measured by the battery pin LRADC. Normally, LRADC channel 7 is dedicated to periodically measuring the battery voltage with a period in the millisecond range for most applications. The voltage is automatically placed into the BATT_VAL field of the HW_POWER_BATTMONITOR register via the HW_LRADC_CONVERSION register. If necessary, software can turn off the automatic battery voltage update and set the BATT_VAL field manually.

32.6 Battery Charger

The battery charger is essentially a linear regulator that has current and voltage limits.

Charge current is software-programmable within the HW_POWER_CHARGE register. The maximum Li-Ion battery charge current will be the lower of 1C, 785 mA, or the VDD5V current limit. USB charging is typically limited to 500 mA or less to meet compliance requirements. Also, battery charge current will be automatically reduced if the current demands from VDD4P2 and the battery charger exceed the CHARGE_4P2_ILIMIT. Full battery charge current will be restored once the VDD4P2 + battery charge current falls below the CHARGE_4P2_ILIMIT value.

Typical charge times for a Li-Ion battery are 1.5 to 3 hours with >70% of the charge delivered in the first hour.

The battery charge voltage limit is 4.2 V for the Li-Ion batteries.

The Li-Ion charge is typically stopped after a certain time limit OR when the charging current drops below 10% of the charge current setting. The HW_POWER_CHARGE register includes controls for the maximum charge current and for the stop charge current. While the charger is delivering current greater than the stop charge limit, the HW_POWER_STS_CHRGSTS bit will be high. This bit should be polled (a low rate of 1 second or greater is fine) during charge. When the bit goes low, the charging is complete. It would be good practice to check that this bit is low for two consecutive checks, as the DC-DC switching might cause a spurious “low” result. Once this bit goes low, the charger can either be stopped immediately or stopped after a “top-off” time limit. Although the charger will avoid exceeding the charge

voltage limit on the battery, it is NOT recommended to leave the charger active indefinitely. It should be turned off when the charge is complete.

One can programatically monitor the battery voltage using the LRADC. The charger has its own (very robust) voltage limiting that operates independently of the LRADC. But monitoring the battery voltage during the charge might be helpful for reporting the charge progress.

The battery charger is capable of generating a large amount of heat within the i.MX23, especially at currents above 400 mA. The dissipated power can be estimated as: (5V – battery_volt) * current. At max current (785 mA) and a 3-V battery, the charger can dissipate 1.57 W on chip, which can cause a dramatic increase in the die temperature. To ensure that the system operates correctly, the die temperature sensor should be monitored periodically (every 100 ms is recommended). If the die temperature exceeds (or closely approaches) the maximum junction temperature in the Characteristics and Specifications chapter, then the battery charge current should be reduced. The LRADC can also be used to monitor the battery temperature or chip temperature. There is an integrated current source for the external temperature sensor that can be configured and enabled via HW_LRADC_CTRL2 register.

32.7 Silicon Speed Sensor

The i.MX23 integrates a silicon speed sensor to measure the performance characteristics of an individual die at its ambient temperature and process parametrics. It consists of a ring oscillator and a frequency counter. The ring oscillator runs on the VDDD power rail. Therefore, its frequency tracks the silicon performance as it changes in response to changes in operating voltage and temperature. The crystal oscillator is directly used as the precision time base for measuring the frequency of a ring oscillator. The ring oscillator is normally disabled. There is a 8-bit counter connected to the ring oscillator that performs the frequency measurement. See the HW_POWER_SPEED register.

Thus, the counter holds the number of cycles the ring oscillator was able to generate during one crystal clock period. The natural frequency of the ring oscillator strongly tracks the silicon process parametrics, i.e., faster silicon processes yield ring oscillators that run faster and thereby yield larger count values. The natural frequency tracks junction temperature effects on silicon speed as well.

The information given by the speed sensor can be used with the silicon temperature and process parameters, which can also be monitored by system software. Freescale can provide a power management application note and firmware that takes full advantage of the on-chip monitoring functions to enable minimum-voltage operation.

32.8 Interrupts

The power system supports nine CPU interrupt events that are programmable within the HW_POWER_CTRL register. The interrupts are listed in Table 32-1.

Table 32-1. Power System Interrupts

HW_POWER_CTRL Interrupt Bit	Description
VDDA_BO_IRQ	VDDA Brownout
VDDD_BO_IRQ	VDDD Brownout
VDDIO_BO_IRQ	VDDIO Brownout
BATT_BO_IRQ	Battery Brownout
VDD5V_GT_VDDIO_IRQ	VDD5V > (VDDIO + 0.6)
DC_OK_IRQ	Voltage Supplies Ok after target voltage change
VBUSVALID_IRQ	VDD5V > Vbusvalid Threshold
LINREG_OK_IRQ	Debug use only
PSWITCH_IRQ	PSWITCH Status

The VDDA_BO_IRQ, VDDD_BO_IRQ, VDDIO_BO_IRQ, and BATT_BO_IRQ each have their own interrupt line back to the interrupt collector. However, the remaining five interrupts—VDD5V_GT_VDDIO_IRQ, DC_OK_IRQ, VBUSVALID_IRQ, LINREG_OK_IRQ, and PSWITCH_IRQ—all share a single interrupt line. In this case, software must read the interrupt status bits to discover which event caused the interrupt.

32.9 Proper Power Supply Protection

To fully comprehend this section, first read the above power supply sections which discuss the basic operation of the power supply and

Chapter 3 "Characteristics and Specifications" which provides the operational limits of the integrated DC-DC converter and power supply. Great care must be taken when programming of the internal PMU to ensure proper protection and operation.

The integrated power supply provides undervoltage-protection mechanisms on the input and output voltage rails. The power supply does not provide overcurrent protection with the exception of the VDD4P2 rail. The LRADC peripheral can be used to take die temperature measurements and external thermistor measurements which software can monitor for further system protection against overheating.

The RTC peripheral's watchdog functionality can be used to monitor the health of the processor to guarantee software reliability.

32.9.1 Power Supply Protection Goal

The primary goal of the power supply system protection should be to keep both the supply input voltages and the regulated output voltages within their proper operational limits. Voltages outside these limits could result in undefined behavior. When voltages outside the limits are detected, the device should be

completely shut down. Proper protection of the power supply input voltage requires at least one supply source to be within its valid operating voltage range.

32.9.2 Power Supply Input Voltage Protection

On boot up, the PWDN_BATTBRNOUT and the PWDN_5VBRNOUT functions are enabled by default to allow hardware the ability to shutdown the device in the case of a battery or 5V supply brownout, respectively. After the system is booted and initialized, these hardware protection functionalities are commonly replaced by software protection mechanisms and then disabled. For example, PWDN_5VBRNOUT is disabled once sufficient battery voltage is available and the 5V-to-Battery transfer has been properly configured. PWDN_BATTBRNOUT is disabled and replaced by a battery brownout FIQ handler.

To ensure proper power supply operation, the developer must take great care to ensure proper protection of the system by using software interrupts to handle supply and output rail brownouts. Detailed knowledge of the power supply behavior is necessary to achieve proper coordination of the protection functionalities of the supply.

32.9.3 PWDN_BATTBRNOUT and PWDN_5VBRNOUT Details

The hardware battery brownout shutdown functionality is only enabled when PWDN_BATTBRNOUT is set and when the chip's 5V status is false. This allows the hardware to keep this register bit enabled by default and still allow 5V to boot up the device when the battery is completely discharged. Then the application can start and charge the battery. Note the PWDN_BATTBRNOUT only uses the VDD5V_GT_VDDIO 5V detection method to determine 5V status.

The hardware 5V brownout shutdown functionality is only enabled after the VDD5V voltage has reached the 5V connection voltage threshold. The threshold is established by the 5V detection method and in the case of VBUSVALID, can be configured by firmware. This allows the PWDN_5VBRNOUT bit to be enabled by default in hardware while still booting a device from battery when 5V is not present.

It is necessary to keep the PWDN_5VBRNOUT and PWD_BATTBRNOUT behavior details in mind to understand how these functionalities can affect the system protection implementation.

32.9.4 VDD5V Input Protection

The integrated power supply has two methods for determining the 5V connection status of the chip: VBUSVALID and VDD5V_GT_VDDIO. Software can poll status bits to determine 5V connection status for either or both of these methods. The hardware can only use one of the 5V detection methods and the choice is programmed via the VBUSVALID_5VDETECT bit. Note the PWDN_BATTBRNOUT functionality always uses VDD5V_GT_VDDIO to determine 5V status regardless of the VBUSVALID_5VDETECT setting.

On boot up, the hardware defaults to the VDD5V_GT_VDDIO 5V detection method. Even though it is the default, VDD5V_GT_VDDIO is only intended as the chip startup 5V detection method. Firmware should immediately switch to the VBUSVALID detection method to avoid some potential issues.

First, the VDD5V_GT_VDDIO method depends on a fixed offset between VDDIO and the VDD5V voltage. So the VDD5V_GT_VDDIO status will change as the VDDIO level changes.

Second, as with any linear regulator, the VDDIO linear regulator supply strength is based upon its supply voltage which is VDD5V. As the VDD5V level decreases, the VDDIO supply strength decreases. So if the VDDIO rail has too much load and the VDD5V level decreases, the VDDIO output voltage can decrease as well. If the VDDIO voltage drops with VDD5V, then a scenario arises where VDD5V_GT_VDDIO never becomes false. One method for countering this scenario is to enable and properly handle VDDIO brownouts. But this brings other complications such as causing the device to shutdown on a 5V disconnect when it could have performed a 5V-to-battery transfer, or having to add functionality to the VDDIO interrupt handler to determine whether a brownout was caused by a 5V disconnect.

To avoid these issues with the VDD5V_GT_VDDIO functionality, the hardware should be programmed to use the VBUSVALID functionality for detecting 5V by setting VBUSVALID_5VDETECT. But, great care must be taken to coordinate this change properly. Remember the PWD_BATTBRNOUT functionality always uses VDD5V_GT_VDDIO as an internal enable/disable mechanism. A problem can occur when the 5V detection is changed from VDD5V_GT_VDDIO to VBUSVALID if the VDD5V_GT_VDDIO status is true but VBUSVALID status is false due to the VBUSVALID_TRSH level selected. In this case, VDD5V level is higher than the set threshold. A false VBUSVALID will register as a 5V disconnection. A hardware 5V disconnection event causes the DCDC to automatically turn on regardless of the battery voltage level. This could result in the DCDC operating from a voltage lower than the minimum operating voltage specified in the Characteristics and Specifications section and physical damage to the DCDC converter itself. To avoid this situation, follow these rules.

- Before switching to VBUSVALID detection method, ensure the VDD5V voltage level is higher than the VBUSVALID_TRSH level.
- When raising the VBUSVALID_TRSH from one level to another, be sure that the VDD5V is higher than the target threshold level being programmed.

In addition to the power supply 5V detection methods, VDD5V voltage measurements can be taken by the LRADC peripheral.

32.9.5 DCDC Input Protection

The DCDC converter must only be allowed to operate within the valid operational range provided in the "Characteristics and Specifications" section of the datasheet. Failure to do so could result in damage to the DCDC converter.

As mentioned above, PWD_BATTBRNOUT is the hardware protection mechanism for shutting down the device if the VBATT rail voltage falls too low during DCDC activity. This hardware mechanism can be disabled and replaced with a software FIQ mechanism. The FIQ handler can then take the appropriate action such as shutting down the device.

The VDD4P2 rail must also be protected so that the DCDC gets a valid input voltage. To do this, configure the system to shutdown on a VDD4P2 brownout FIQ if battery voltage is insufficient.

32.9.6 DCDC Output Protection

The output rails voltage brownout functionality provides both software FIQ handling and automatic hardware shutdown to protect against undervoltage events. The software FIQ handling provides flexibility on what actions are taken when a brownout is detected, but it is vulnerable to software problems that could lead to it not functioning as intended. Also, the time needed to handle the brownout will be greater than if the automatic hardware brownouts are used.

32.9.7 PWD_OFF Bit Usage

Some applications set the PWD_OFF bit in an attempt to raise the ESD tolerance level, but the PWD_OFF bit disables all automatic hardware shutdown functions. Before setting the PWD_OFF bit, all hardware power supply protections for input and output power rails must be replaced by a software FIQ mechanism.

32.9.8 Power Supply Protection Summary

Given the above information, here are some basic rules to follow:

1. Never allow the power supply to operate from an input source that is not at a sufficient voltage level
    - When a 5V disconnection event is detected, the DCDC automatically attempts to operate from the battery source. To avoid the possibility of causing the DCDC to start sourcing from a voltage that is below its specified minimum operating voltage, the PWDN_5VBRNOUT (or similar software FIQ functionality) must remain active until the battery has sufficient charge and the 5V-to-battery transfer has been enabled.
    - The switch from VDD5V_GT_VDDIO to VBUSVALID 5V detection method can cause a 5V disconnected hardware status event if not handled properly.
    - Never manually enable the DCDC in software while the battery voltage or VDD4P2 voltage is too low. Also ensure battery voltage and VDD4P2 protection mechanisms are enabled prior to enabling the DCDC.
2. Shutdown the device if the regulated output voltage rails fall below the brownout level.

3. Proper power supply protection and operation can depend on immediate FIQ handling of power supply events.
    - When depending on FIQ handlers for proper system protection and operation, never disable FIQ handling.
    - FIQ handling code and the context the FIQ runs in should be located in OCRAM for fastest and most reliable operation. When enabling virtual memory, take care to not require a page table walk that would cause an external memory access. In many applications, it is sometimes necessary to disable the external memory access or vastly reduce its performance. Special fixed TLB cache entries exist in the ARM926ej-s core to allow the lockdown of particular page entries which can be used to avoid a page table access.

32.10 DC-DC Converter Efficiency

Graphs showing estimates of typical efficiencies of the DC-DC converter under nominal conditions will be provided after silicon is validated and characterized.

32.11 Programmable Registers

32.11.1 Power Control Register Description

The Power Control Register contains control bits specific to the digital section.

HW_POWER_CTRL	0x000
HW_POWER_CTRL_SET	0x004
HW_POWER_CTRL_CLR	0x008
HW_POWER_CTRL_TOG	0x00C

Table 32-2. HW_POWER_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD3	CLKGATE	RSRVD2		PSWITCH_MID_TRAN	RSRVD1		DCDC4P2_BO_IRQ	ENIRQ_DCDC4P2_BO	VDD5V_DROOP_IRQ	ENIRQ_VDD5V_DROOP	PSWITCH_IRQ	PSWITCH_IRQ_SRC	POLARITY_PSWITCH	ENIRQ_PSWITCH	POLARITY_DC_OK	DC_OK_IRQ	ENIRQ_DC_OK	BATT_BO_IRQ	ENIRQBATT_BO	VDDIO_BO_IRQ	ENIRQ_VDDIO_BO	VDDA_BO_IRQ	ENIRQ_VDDA_BO	VDDD_BO_IRQ	ENIRQ_VDDD_BO	POLARITY_VBUSVALID	VBUSVALID_IRQ	ENIRQ_VBUS_VALID	POLARITY_VDD5V_GT_VDDIO	VDD5V_GT_VDDIO_IRQ	ENIRQ_VDD5V_GT_VDDIO

Table 32-3. HW_POWER_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSRVD3**	RO	0x0	Empty Description.
30	**CLKGATE**	RW	0x1	This bit must be set to zero for normal operation. When set to one, it gates off the clocks to the block. This bit has no effect on the RTC analog section.
29:28	**RSRVD2**	RO	0x0	Empty Description.
27	**PSWITCH_MID_TRAN**	RW	0x0	When this bit is set, it selects the from-mid-transition interrupt functionality for PSWITCH. When set, in conjuction with ENIRQ_PSWITCH, it will set PSWITCH_IRQ when either a 01 to 11 or 01 to 00 transition is detected on the PSWITCH comparators. When this bit is set, PSWITCH_IRQ_SRC and POLARITY_PSWITCH have no effect.
26:25	**RSRVD1**	RO	0x0	Empty Description.
24	**DCDC4P2_BO_IRQ**	RW	0x0	Interrupt Status for 4P2_BO. Reset this bit by writing a one to the SCT clear address space and not by a general write.
23	**ENIRQ_DCDC4P2_BO**	RW	0x0	Enable interrupt for 4P2_BO.
22	**VDD5V_DROOP_IRQ**	RW	0x0	Interrupt Status for VDD5V_DROOP. Reset this bit by writing a one to the SCT clear address space and not by a general write.
21	**ENIRQ_VDD5V_DROOP**	RW	0x0	Enable interrupt for VDD5V_DROOP.
20	**PSWITCH_IRQ**	RW	0x0	Interrupt status for PSWITCH signals. Interrupt polarity is set using POLARITY_PSWITCH. Comparator bit is selected via PSWITCH_IRQ_SRC. Reset this bit by writing a one to the SCT clear address space and not by a general write.
19	**PSWITCH_IRQ_SRC**	RW	0x0	Set to 1 to use HW_POWER_STS_PSWITCH bit 1 as source, 0 for HW_POWER_STS_PSWITCH bit 0
18	**POLARITY_PSWITCH**	RW	0x0	Set to 1 to interupt when the interrupt source is high, 0 for low
17	**ENIRQ_PSWITCH**	RW	0x0	Interrupt status for PSWITCH signal. Interrupt polarity is set using POLARITY_PWITCH and source selected by PSWITCH_SRC.
16	**POLARITY_DC_OK**	RW	0x1	Debug use only. Set to 1 to check for linear regulators ok. Set to 0 to check for 5V disconnected.
15	**DC_OK_IRQ**	RW	0x0	Interrupt Status for DC_OK. Reset this bit by writing a one to the SCT clear address space and not by a general write. The IRQ will be asserted when switching DC-DC converter control loop has stabilized after a voltage target change. When linear regulators are active, interrupt will get asserted when the actual voltage is above the target voltage. Therefore, DC_OK_IRQ will assert when changing a linear regulator output to a lower value before the actual voltage decreases to the new target value.
14	**ENIRQ_DC_OK**	RW	0x0	Enable interrupt for DC_OK.
13	**BATT_BO_IRQ**	RW	0x0	Interrupt Status for BATT_BO. Reset this bit by writing a one to the SCT clear address space and not by a general write.
12	**ENIRQBATT_BO**	RW	0x0	Enable interrupt for battery brownout.
11	**VDDIO_BO_IRQ**	RW	0x0	Interrupt Status for VDDIO_BO. Reset this bit by writing a one to the SCT clear address space and not by a general write.

Table 32-3. HW_POWER_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
10	**ENIRQ_VDDIO_BO**	RW	0x0	Enable interrupt for VDDIO brownout.
9	**VDDA_BO_IRQ**	RW	0x0	Interrupt Status for VDDA_BO. Reset this bit by writing a one to the SCT clear address space and not by a general write.
8	**ENIRQ_VDDA_BO**	RW	0x0	Enable interrupt for VDDA brownout.
7	**VDDD_BO_IRQ**	RW	0x0	Interrupt Status for VDDD_BO. Reset this bit by writing a one to the SCT clear address space and not by a general write.
6	**ENIRQ_VDDD_BO**	RW	0x0	Enable interrupt for VDDD brownout.
5	**POLARITY_VBUSVALID**	RW	0x1	Set to 1 to check for 5V connected using VBUSVALID status bit. Set to 0 to check for 5V disconnected.
4	**VBUSVALID_IRQ**	RW	0x0	Interrupt status for VBUSVALID signal. Interrupt polarity is set using POLARITY_VBUSVALID. Reset this bit by writing a one to the SCT clear address space and not by a general write.
3	**ENIRQ_VBUS_VALID**	RW	0x0	Enable interrupt for 5V detect using VBUSVALID.
2	**POLARITY_VDD5V_GT_VDDIO**	RW	0x1	Set to 1 to check for 5V connected. Set to 0 to check for 5V disconnected.
1	**VDD5V_GT_VDDIO_IRQ**	RW	0x0	Interrupt status for VDD5V_GT_VDDIO signal. Interrupt polarity is set using POLARITY_VDD5V_GT_VDDIO. Reset this bit by writing a one to the SCT clear address space and not by a general write.
0	**ENIRQ_VDD5V_GT_VDDIO**	RW	0x0	Enable interrupt for 5V detect.

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.2 DC-DC 5V Control Register Description

This register contains the configuration options of the power management subsystem that are available when external 5V is applied.

HW_POWER_5VCTRL	0x010
HW_POWER_5VCTRL_SET	0x014
HW_POWER_5VCTRL_CLR	0x018
HW_POWER_5VCTRL_TOG	0x01C

Table 32-4. HW_POWER_5VCTRL

31 30	29 28	27	26 25 24	23 22 21	20	19 18	17 16	15 14 13 12	11	10 09 08	07	06	05	04	03	02	01	00
RSRVD6	VBUSDROOP_TRSH	RSRVD5	HEADROOM_ADJ	RSRVD4	PWD_CHARGE_4P2	RSRVD3	CHARGE_4P2_ILIMIT		RSRVD2	VBUSVALID_TRSH	PWDN_5VBRNOUT	ENABLE_LINREG_ILIMIT	DCDC_XFER	VBUSVALID_5VDETECT	VBUSVALID_TO_B	ILIMIT_EQ_ZERO	PWRUP_VBUS_CMPS	ENABLE_DCDC

Table 32-5. HW_POWER_5VCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD6**	RO	0x0	Empty Description.
29:28	**VBUSDROOP_TRSH**	RW	0x0	Set the threshold for the VBUSDROOP comparator. This comparator should typically be programmed at least 200mV above VVBUSVALID, and can be used to terminate battery charge when the voltage on VDD5V falls below the programmed threshold. This comparator has ~50mV of hystersis to prevent chattering at the comparator trip point. 00: 4.3V 01: 4.4V 10: 4.5V 11: 4.7V
27	**RSRVD5**	RO	0x0	Empty Description.
26:24	**HEADROOM_ADJ**	RW	0x0	Adjustment to optimize the performance of the battery charge and 4.2V regulation circuit at low 5v voltages.
23:21	**RSRVD4**	RO	0x0	Empty Description.
20	**PWD_CHARGE_4P2**	RW	0x1	Controls the power down of both the battery charger and 4.2V regulation circuit. Default is powered down.
19:18	**RSRVD3**	RO	0x0	Empty Description.
17:12	**CHARGE_4P2_ILIMIT**	RW	0x00	Limits the combined current from 5V that the battery charger and DCDC_4P2 circuit consume. Current represented by each bits is as follows: (400 mA, 200 mA, 100 mA, 50 mA, 20 mA, 10 mA) = (bit 5,bit 4, bit 3, bit 2, bit 1, bit 0). The DCDC_4P2 circuit is given priority as the sum of the currents exceeds the limit.
11	**RSRVD2**	RO	0x0	Empty Description.

Table 32-5. HW_POWER_5VCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
10:8	**VBUSVALID_TRSH**	RW	0x0	Set the threshold for the VBUSVALID comparator. This comparator is the most accurate method to determine the presence of 5v, and includes hystersis to minimize the need for software debounce of the detection. This comparator has ~50mV of hystersis to prevent chattering at the comparator trip point. 000: 2.9V 001: 4.0V 010: 4.1V 011: 4.2V 100= 4.3V 101: 4.4V 110: 4.5V 111: 4.6V
7	**PWDN_5VBRNOUT**	RW	0x1	The purpose of this bit is to power down the device if 5V is removed before the system is completely initialized. Clear this bit to disable automatic hardware powerdown AFTER the system is configured for 5v removal. This bit should not be set if DCDC_XFER is set.
6	**ENABLE_LINREG_ILIMIT**	RW	0x0	Enable the current limit in the linear regulators. The current limit is active during powerup from 5v and automatically disables before the ROM executes. This limit is needed to meet the USB in rush current specification of 100mA + 50uC. Note this linear regulator current limit is not related to the CHARGE_4P2_ILIMIT.
5	**DCDC_XFER**	RW	0x0	Enable automatic transition to switching DC-DC converter when VDD5V is removed. The LRADC must be operational and the BATT_VAL field must be written with the battery voltage using 8-mV step-size. It is also important to set the EN_BATADJ field.
4	**VBUSVALID_5VDETECT**	RW	0x0	Power up and use VBUSVALID comparator as detection circuit for 5V in the switching converter. Default is for the switching converter to use the VDD5V_GT_VDDIO status bit to determine the presence of 5V in the system. The VBUSVALID comparator provides a more accurate and adjustable threshold to determine the presence of 5V in the system, and is the recommended method of detecting 5V.
3	**VBUSVALID_TO_B**	RW	0x0	This bit muxes the Bvalid comparator to the VBUSVALID comparator and is used for test purposes only.
2	**ILIMIT_EQ_ZERO**	RW	0x0	The amount of current the device will consume from the 5V rail is minimized. The VDDIO linear regulator current limit is set to zero mA. Also, the source of current for the crystal oscillator and RTC is switched to the battery. Note that this functionality does not affect battery charge.

Table 32-5. HW_POWER_5VCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
1	**PWRUP_VBUS_CMPS**	RW	0x0	Powers up comparators for 5v
0	**ENABLE_DCDC**	RW	0x0	Enables the switching DC-DC converter when 5V is present. It is recommended to set ILIMIT_EQ_ZERO, ENABLE_ILIMIT, and all LINREG_OFFSET bits when enabling this functionality.

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.3 DC-DC Minimum Power and Miscellaneous Control Register Description

This register controls options to drop the power used by the switching DC-DC converter. These bits should only be modified with guidance from Freescale.

HW_POWER_MINPWR	0x020
HW_POWER_MINPWR_SET	0x024
HW_POWER_MINPWR_CLR	0x028
HW_POWER_MINPWR_TOG	0x02C

Table 32-6. HW_POWER_MINPWR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																	LOWPWR_4P2	VDAC_DUMP_CTRL	PWD_BO	USE_VDDXTAL_VBG	PWD_ANA_CMPS	ENABLE_OSC	SELECT_OSC	VBG_OFF	DOUBLE_FETS	HALF_FETS	LESSANA_I	PWD_XTAL24	DC_STOPCLK	EN_DC_PFM	DC_HALFCLK

Table 32-7. HW_POWER_MINPWR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:15	**RSRVD1**	RO	0x0	Empty Description.
14	**LOWPWR_4P2**	RW	0x0	Enable low power regulation of DCDC_4P2 limited to 2.5mA. This bit should always be set to 0.
13	**VDAC_DUMP_CTRL**	RW	0x0	Dumps extra Video DAC current into the VDDD supply rail only when the VDDD supply voltage is below it's target value and XX bit is set. Default is to always dump the current to VDDD when XX bit is set.
12	**PWD_BO**	RW	0x0	Powers down supply brownout comparators. This should only be used when monitoring supply brownouts is not needed.

Table 32-7. HW_POWER_MINPWR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
11	**USE_VDDXTAL_VBG**	RW	0x0	Change the reference used in the dcdc converter to a low power, less accurate reference. This should only be used when achieving minimum system power is more important than supply voltage accuracy.
10	**PWD_ANA_CMPS**	RW	0x0	Powers down analog comparators used in the power module, including the VDD and VDDA brownout comparators, low power reference. This bit should only be set to reach absolute minimum system power when running from the linear regulators and supply accuracy and VDDD/VDDA brownout detection is not important.
9	**ENABLE_OSC**	RW	0x0	Enables the internal oscillator. This oscillator is less accurate , but lower power than the 24mhz xtal.
8	**SELECT_OSC**	RW	0x0	Switch internal 24mhz clock reference to the less accurate internal oscillator. This bit should be set only when accuracy of the 24mhz clock is not important. The value of this bit should only be changed when ENABLE_OSC=1 and PWD_XTAL24=0.
7	**VBG_OFF**	RW	0x0	Powers down the bandgap reference. This should only be used in 5v-powered applications when absolute minimum power is more important than supply voltage accuracy. USB_I_SUSPEND must be set before this bit is set.
6	**DOUBLE_FETS**	RW	0x0	Approximately doubles the size of power transistors in DC-DC converter. This maybe be useful in high power conditions.
5	**HALF_FETS**	RW	0x0	Disable half the power transistors in DC-DC converter. This maybe be useful in low-power conditions when the increased resistance of the power FETs is acceptable.
4	**LESSANA_I**	RW	0x0	Reduce DC-DC analog bias current 20%. This bit is intended to reduce power in low-performance operating modes, such as USB suspend.
3	**PWD_XTAL24**	RW	0x0	Powers down the 24mhz oscillator, even when the system is still operating. This can be used with ENABLE_OSC and SELECT_OSC to reduce current in usb suspend and other low power modes where the accuracy of the clock is not important.
2	**DC_STOPCLK**	RW	0x0	Stop the clock to internal logic of switching DC-DC converter. This bit will take effect only after the switching FETs are off, due to battery configuration, PFM mode, or internal linear regulator operation.
1	**EN_DC_PFM**	RW	0x0	Forces DC-DC to operate in a Pulse Frequency Modulation mode. Intended to allow minimum system power in very low-power configurations when increased ripple on the supplies is acceptable. Also, HYST_SIGN in HW_POWER_LOOPCTRL should be set high when using PFM mode.
0	**DC_HALFCLK**	RW	0x0	Slow down DC-DC clock from 1.5 MHz to 750 kHz. This maybe be useful to improve efficiency at light loads or improve EMI radiation, although peak-to-peak voltage on the supplies will increase.

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.4 Battery Charge Control Register Description

This register cotrols the battery charge features for both NiMH slow charge and Li-Ion charge.

HW_POWER_CHARGE	0x030
HW_POWER_CHARGE_SET	0x034
HW_POWER_CHARGE_CLR	0x038
HW_POWER_CHARGE_TOG	0x03C

Table 32-8. HW_POWER_CHARGE

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD5					ADJ_VOLT			RSRVD3	ENABLE_LOAD	ENABLE_CHARGER_RESISTORS	ENABLE_FAULT_DETECT	CHRG_STS_OFF	RSVD4	RSVD3	PWD_BATTCHRG	RSVD2				STOP_ILIMIT				RSVD1		BATTCHRG_I					

Table 32-9. HW_POWER_CHARGE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:27	**RSVD5**	RO	0x0	Empty Description.
26:24	**ADJ_VOLT**	RW	0x0	Adjustments to the final LiIon final voltage. These bits shouldn't be set unless recommended by Freescale. 0b000: no change 0b001: -0.25% 0b010: +0.50% 0b011: -0.75% 0b100: +0.25% 0b101: -0.50% 0b110: +0.75% 0b111: -1.00%
23	**RSRVD3**	RW	0x0	Empty Description.
22	**ENABLE_LOAD**	RW	0x0	Enable 100ohm load on the regulated 4.2V output. This bit shouldn't be set unless recommended by Freescale.

Table 32-9. HW_POWER_CHARGE Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
21	**ENABLE_CHARGER_RESISTORS**	RW	0x0	Enable 125k pullup on USB_DP and 375k on USB_DN to provice USB_CHARGER functionality. This functionality is a new USB spec and should not be enabled unless recommended by Freescale.
20	**ENABLE_FAULT_DETECT**	RW	0x0	Enable fault detection in the battery charger. When enabled, this bit will power down the battery charger when the VDD5V voltage falls below the battery voltage. The fault can be cleared by cycling PWD_CHARGE_4P2. The fault detection is visible via HW_POWER_STS_VDD5V_FAULT.
19	**CHRG_STS_OFF**	RW	0x0	Setting this bit disables the CHRGSTS status bit. Disabling CHRGSTS should only be done when the switching converter is enabled during battery charge if noise from the switching converter causes CHRGSTS to toggle excessively.
18	**RSVD4**	RW	0x0	Program this field to 0x0.
17	**RSVD3**	RO	0x0	Empty Description.
16	**PWD_BATTCHRG**	RW	0x1	Power-down the battery charge circuitry. This should only be set low when 5V is present
15:12	**RSVD2**	RO	0x0	Empty Description.
11:8	**STOP_ILIMIT**	RW	0x0	Current threshold at which the Li-Ion battery charger signals to stop charging. The current represented by each bits is as follows: (100 mA, 50 mA, 20 mA, 10 mA) = (bit 3, bit 2, bit 1, bit 0) It is recommended to set this value to 10% of the charge current.
7:6	**RSVD1**	RO	0x0	Empty Description.
5:0	**BATTCHRG_I**	RW	0x00	Magnitude of the battery charge current, the current represented by each bits is as follows: (400 mA, 200 mA, 100 mA, 50 mA, 20 mA, 10 mA) = (bit 5,bit 4, bit 3, bit 2, bit 1, bit 0)

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.5 VDDD Supply Targets and Brownouts Control Register Description

This register controls the voltage targets and brownout targets for the VDDD supply generated from the switching DC-DC converter and integrated linear regulators. The brownout comparators default enabled.

HW_POWER_VDDDCTRL 0x040

Table 32-10. HW_POWER_VDDDCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADJTN				RSRVD4				PWDN_BRNOUT	DISABLE_STEPPING	ENABLE_LINREG	DISABLE_FET	RSRVD3		LINREG_OFFSET		RSRVD2					BO_OFFSET			RSRVD1				TRG			

Table 32-11. HW_POWER_VDDDCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**ADJTN**	RW	0x0	Two's complement number that can be used to adjust the duty cycle of VDDD. This is intended for test purposes only
27:24	**RSRVD4**	RO	0x0	Empty Description.
23	**PWDN_BRNOUT**	RW	0x0	Powers down the device after the DC-DC converter completes startup if a VDDD brownout occurs. This function is only active when 5V is not present. Additionally, software should clear this bit and disable this function after the system is configured for a VDDD brownout and the VDDD brownout interrupt is enabled.
22	**DISABLE_STEPPING**	RW	0x0	Disables the default behavior of the voltage stepping algorithm when the TRG field is updated. By default, the control loop steps sequentially through 25mV steps on both target and brownout voltage to minimize transients during TRG adjustments. When this bit is set, the entire target voltage and brownout adjustment takes place at one time, which will speed transitions, but will result in increased supply transients as well as possible brownouts when the new (TRG - BO_OFFSET) >= old TRG. Thus, it is recommended to change TRG by less than the (BO_OFFSET-2) value when using DISABLE_STEPPING. This bit should be set high when powering VDDD from the integrated linear regulators.
21	**ENABLE_LINREG**	RW	0x1	Enables the VDDD linear regulator converter when the switching converter is active. By default, the linear regulator is not active when the switching converter is active.
20	**DISABLE_FET**	RW	0x1	Disable the VDDD switching converter output.
19:18	**RSRVD3**	RO	0x0	Empty Description.

Table 32-11. HW_POWER_VDDDCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
17:16	**LINREG_OFFSET**	RW	0x1	Number of 25mV steps between linear regulator output voltage and switching converter target. 00b = 0 steps, recommended when powering VDDD only from linear regulator and ENABLE_DCDC=DCDC_XFER=0. It is also recommended to set DISABLE_STEPPING when powering VDDD from the linear regulators. 01b = 1 step above, default. 1Xb = 1 step below, important when powering VDDD from DC-DC converter and linear regulator simultaneously.
15:11	**RSRVD2**	RO	0x0	Empty Description.
10:8	**BO_OFFSET**	RW	0x7	Brownout voltage offset in 25mV steps below the TRG value. Note that the hardware only supports brownout voltages between 0.8V and 1.475V, and values outside this range should not be programmed. The brownout trip voltage will adjust as the target voltage changes.
7:5	**RSRVD1**	RO	0x0	Empty Description.
4:0	**TRG**	RW	0x10	Voltage level of the VDDD supply. The step size of this field is 25 mV. 0x00 = 0.8 V, 0x1F = 1.575 V, and the reset value = 1.2 V. This field should not be set above the operating maximum as specified in the Recommended Operating Conditions table. It is also recommended to set DISABLE_STEPPING when powering VDDD from the integrated linear regulators. Setting DISABLE_STEPPING does set additional restrictions on TRG adjusments.

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.6 VDDA Supply Targets and Brownouts Control Register Description

This register controls the voltage targets and brownout targets for the VDDA supply generated from the switching DC-DC converter and integrated linear regulators. The brownout comparators default enabled.

HW_POWER_VDDACTRL 0x050

Table 32-12. HW_POWER_VDDACTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD4												PWDN_BRNOUT	DISABLE_STEPPING	ENABLE_LINREG	DISABLE_FET	RSRVD3		LINREG_OFFSET		RSRVD2	BO_OFFSET			RSRVD1				TRG			

Table 32-13. HW_POWER_VDDACTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:20	**RSRVD4**	RO	0x0	Empty Description.
19	**PWDN_BRNOUT**	RW	0x0	Powers down the device after the DC-DC converter completes startup if a VDDA brownout occurs. This function is only active when 5V is not present. Additionally, software should clear this bit and disable this function after the system is configured for a VDDA brownout and the VDDA brownout interrupt is enabled.
18	**DISABLE_STEPPING**	RW	0x0	Disables the default behavior of the voltage stepping algorithm when the TRG field is updated. By default, the control loop steps sequentially through 25mV steps on both target and brownout voltage to minimize transients during TRG adjustments. When this bit is set, the entire target voltage and brownout adjustment takes place at one time, which will speed transitions, but will result in increased supply transients as well as possible brownouts when the new (TRG - BO_OFFSET) >= old TRG. Thus, it is recommended to change TRG by less than the (BO_OFFSET-2) value when using DISABLE_STEPPING. This bit should be set high when powering VDDA from the integrated linear regulators.
17	**ENABLE_LINREG**	RW	0x0	Enables the VDDA linear regulator converter when the switching converter is active. By default, the linear regulator is not active when the switching converter is active.
16	**DISABLE_FET**	RW	0x0	Disable the VDDA switching converter output. The switching converter is enabled by default when the battery is not present or the ENABLE_DCDC is set.
15:14	**RSRVD3**	RO	0x0	Empty Description.

Table 32-13. HW_POWER_VDDACTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
13:12	**LINREG_OFFSET**	RW	0x1	Number of 25mV steps between linear regulator output voltage and switching converter target. 00b = 0 steps, recommended when powering VDDA from linear regulator and ENABLE_DCDC=DCDC_XFER=0. It is also recommended to set DISABLE_STEPPING when powering VDDA from the linear regulators. 01b = 1 step above, default. 1Xb = 1 step below, important when powering VDDA from DC-DC converter and linear regulator simultaneously.
11	**RSRVD2**	RO	0x0	Empty Description.
10:8	**BO_OFFSET**	RW	0x7	Brownout voltage offset in 25mV steps below the TRG value. Note that the hardware only supports brownout voltages between 1.4V and 2.175V, and values outside this range should not be programmed. The brownout trip voltage will adjust as the target voltage changes.
7:5	**RSRVD1**	RO	0x0	Empty Description.
4:0	**TRG**	RW	0xA	Voltage level of the VDDA supply. The step size of this field is 25 mV. 0x00 = 1.5 V, 0x1F = 2.275 V, and the reset value = 1.75 V. This field should not be set above 1.95V as this may cause damage to the device. It is also recommended to set DISABLE_STEPPING when powering VDDA from the integrated linear regulators. Setting DISABLE_STEPPING does set additional restrictions on TRG adjusments.

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.7 VDDIO Supply Targets and Brownouts Control Register Description

This register controls the voltage targets and brownout targets for the VDDIO supply generated from the switching DC-DC converter and integrated linear regulators. The brownout comparators default enabled.

HW_POWER_VDDIOCTRL 0x060

Table 32-14. HW_POWER_VDDIOCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD5								ADJTN				RSRVD4	PWDN_BRNOUT	DISABLE_STEPPING	DISABLE_FET	RSRVD3		LINREG_OFFSET		RSRVD2	BO_OFFSET			RSRVD1				TRG			

Table 32-15. HW_POWER_VDDIOCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSRVD5**	RO	0x0	Empty Description.
23:20	**ADJTN**	RW	0x0	Two's complement number that can be used to adjust the duty cycle of VDDIO. This is intended for test purposes only
19	**RSRVD4**	RO	0x0	Empty Description.
18	**PWDN_BRNOUT**	RW	0x0	Powers down the device after the DC-DC converter completes startup if a VDDIO brownout occurs. This function is only active when 5V is not present. Additionally, software should clear this bit and disable this function after the system is configured for a VDDIO brownout and the VDDIO brownout interrupt is enabled.
17	**DISABLE_STEPPING**	RW	0x0	Disables the default behavior of the voltage stepping algorithm when the TRG field is updated. By default, the control loop steps sequentially through 25mV steps on both target and brownout voltage to minimize transients during TRG adjustments. When this bit is set, the entire target voltage and brownout adjustment takes place at one time, which will speed transitions, but will result in increased supply transients as well as possible brownouts when the new (TRG - BO_OFFSET) >= old TRG. Thus, it is recommended to change TRG by less than the (BO_OFFSET-2) value when using DISABLE_STEPPING. This bit should be set high when powering VDDIO from the integrated linear regulators.
16	**DISABLE_FET**	RW	0x0	Disable the VDDIO switching converter output. The switching converter is enabled by default when the battery is not present or the ENABLE_DCDC is set.
15:14	**RSRVD3**	RO	0x0	Empty Description.

Table 32-15. HW_POWER_VDDIOCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
13:12	**LINREG_OFFSET**	RW	0x1	Number of 25mV steps between linear regulator output voltage and switching converter target. 00b = 0 steps, recommended when powering VDDIO from linear regulator and ENABLE_DCDC=DCDC_XFER=0. It is also recommended to set DISABLE_STEPPING when powering VDDIO from the linear regulators. 01b = 1 step above, default. 1Xb = 1 step below, important when powering VDDIO from DC-DC converter and linear regulator simultaneously.
11	**RSRVD2**	RO	0x0	Empty Description.
10:8	**BO_OFFSET**	RW	0x7	Brownout voltage offset in 25mV steps below the TRG value. Note that the hardware only supports brownout voltages between 2.7V and 3.475V, and values outside this range should not be programmed. The brownout trip voltage will adjust as the target voltage changes.
7:5	**RSRVD1**	RO	0x0	Empty Description.
4:0	**TRG**	RW	0x0C	Voltage level of the VDDIO supply. The step size of this field is 25 mV. 0x00 = 2.8 V, 0x1F = 3.575 V, and the reset value = 3.1 V. It is also recommended to set DISABLE_STEPPING when powering VDDIO from the integrated linear regulators. Setting DISABLE_STEPPING does set additional restrictions on TRG adjusments.

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.8 VDDMEM Supply Targets Control Register Description

This register controls the voltage target for a supply generated from the VDDIO. This supply is intended for use with external memories such as DDR that have unique voltage requirements not compatible with VDDIO, VDDA , or VDDD.

HW_POWER_VDDMEMCTRL 0x070

Table 32-16. HW_POWER_VDDMEMCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2																					PULLDOWN_ACTIVE	ENABLE_ILIMIT	ENABLE_LINREG	RSRVD1				TRG			

Table 32-17. HW_POWER_VDDMEMCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:11	**RSRVD2**	RO	0x0	Empty Description.
10	**PULLDOWN_ACTIVE**	RW	0x0	Activates pulldown on external memory supply. This bit should be set before the regulator is enabled to be sure the supply voltage powers up from ground. Default is pulldown inactive.
9	**ENABLE_ILIMIT**	RW	0x1	Controls the inrush limit (~10mA) for the memory supply voltage. Default is active. This should remain active until the supply settles after enabling the linreg. This should be disabled before accessing the memory.
8	**ENABLE_LINREG**	RW	0x0	Enables the regulator that creates the external memory supply voltage. After enabling the linreg need to wait until the VDDMEM rail is up before disabling the linreg current limit and accessing the memories. 500uS is usually an adequate delay, but it can be longer if VDDMEM cap is >1uF.
7:5	**RSRVD1**	RO	0x0	Empty Description.
4:0	**TRG**	RW	0x0C	Voltage level of the External memory supply. The step size of this field is 50 mV. 0x00 = 1.7 V, 0x1F = 3.25 V, and the reset value = 1.7 V.

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.9 DC-DC Converter 4.2V Control Register Description

This register contains controls that need to be adjusted to select the 4.2V source as the input for the dcdc converter

HW_POWER_DCDC4P2 0x080

Table 32-18. HW_POWER_DCDC4P2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DROPOUT_CTRL				RSRVD5		ISTEAL_THRESH		ENABLE_4P2	ENABLE_DCDC	HYST_DIR	HYST_THRESH	RSRVD3	TRG			RSRVD2				BO				RSRVD1				CMPTRIP			

Table 32-19. HW_POWER_DCDC4P2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**DROPOUT_CTRL**	RW	0x0	Adjusts the behavior of the dcdc converter and 4.2V regulation circuit. The two msbs control the VDD4P2 brownout below the target set by DCDC4p2_trg before the regulation circuit steals battery charge current to support the voltage on VDD4P2. The two lsbs control which power source is selected by the dcdc converter after ENABLE_DCDC is set. 0b11XX: 200mV 0b10XX: 100mV 0b01XX: 50mV 0b00XX: 25mV 0bXX00: DcDc Converter power source is DCDC_4P2 regardless of BATTERY voltage 0bXX01: DcDc converter uses DCDC_4P2 always, and only enables DCDC_BATT when VDD4P2 is less than BATTERY. 0bXX1X: DcDc converter selects either VDD4P2 or BATTERY, which ever is higher.
27:26	**RSRVD5**	RO	0x0	Empty Description.
25:24	**ISTEAL_THRESH**	RW	0x0	Has no effect.
23	**ENABLE_4P2**	RW	0x0	Enables the DCDC_4P2 regulation circuitry. The 4p2V load current has priority over the battery charge current when the sum of the two tries to exceed the limit set with CHARGE_4P2_ILIMIT.
22	**ENABLE_DCDC**	RW	0x0	Enable the dcdc converter to use the DCDC_4P2 pin as a power source based on a voltage comparison between the BATTERY pin voltage and the VDD4P2 pin voltage. The trip point of this comparator is controlled by the CMPTRIP bitfield.
21	**HYST_DIR**	RW	0x0	Enable hysteresis in analog comparator.
20	**HYST_THRESH**	RW	0x0	Increase the threshold detection for DCDC_4P2/BATTERY analog comparator.
19	**RSRVD3**	RO	0x0	Empty Description.
18:16	**TRG**	RW	0x0	Regulation voltage of the DCDC_4P2 pin. 0b000 : 4.2V 0b001 : 4.1V 0b010 : 4.0V 0b011 : 3.9V 0b1XX : BATTERY
15:13	**RSRVD2**	RO	0x0	Empty Description.

Table 32-19. HW_POWER_DCDC4P2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
12:8	**BO**	RW	0x0	Brownout voltage in 25mV steps for the DCDC_4P2 pin. 0b00000 : 3.6V .. 0b11111 : 4.375V
7:5	**RSRVD1**	RO	0x0	Empty Description.
4:0	**CMPTRIP**	RW	0x18	Sets the trip point for the comparison between the DCDC_4P2 and BATTERY pin. When the comparator output is high then, the switching converter may use the DCDC_4P2 pin as the source for the switching converter, otherwise it will use the DCDC_BATT pin. 0b00000 DCDC_4P2 pin >= 0.85 * BATTERY pin 0b00001 DCDC_4P2 pin >= 0.86 * BATTERY pin 0b11000 DCDC_4P2 pin >= BATTERY pin (default) 0b11111 DCDC_4P2 pin >= 1.05 * BATTERY pin

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.10 DC-DC Miscellaneous Register Description

This register contains controls that may need to be adjusted to optimize DC-DC converter performance using the battery voltage information

HW_POWER_MISC 0x090

Table 32-20. HW_POWER_MISC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2																									FREQSEL			RSRVD1	DELAY_TIMING	TEST	SEL_PLLCLK

Table 32-21. HW_POWER_MISC Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:7	**RSRVD2**	RO	0x0	Empty Description.
6:4	**FREQSEL**	RW	0x0	This register will select the PLL-based frequency that the dcdc uses when SEL_PLLCLK is set high. The decode is as follows: 0x0=Reserved 0x1=20MHz 0x2=24MHz 0x3=19.2MHz 0x4=14.4MHz 0x5=18MHz 0x6=21.6MHz 0x7=17.28Mhz .
3	**RSRVD1**	RW	0x0	Empty Description.
2	**DELAY_TIMING**	RW	0x0	This bit delays the timing of the output fets in the switching dcdc converter. This may provide improved ground noise performance in high power applications.
1	**TEST**	RW	0x0	Reserved. Do not set.
0	**SEL_PLLCLK**	RW	0x0	This bit selects the source of the clock used for the DC-DC converter. The default is to use the 24-MHz clock. Setting this bit selects the PLL clock as a clock source for the DC-DC converter. It is required to program FREQSEL before setting this bit.

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.11 DC-DC Duty Cycle Limits Control Register Description

This register defines the upper and lower duty cycle limits of DC-DC. These values depend on details of switching converter implementation and should not be changed without guidance from Freescale.

HW_POWER_DCLIMITS 0x0A0

Table 32-22. HW_POWER_DCLIMITS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD3																RSRVD2	POSLIMIT_BUCK							RSRVD1	NEGLIMIT						

Table 32-23. HW_POWER_DCLIMITS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**RSRVD3**	RO	0x0	Empty Description.
15	**RSRVD2**	RO	0x0	Empty Description.
14:8	**POSLIMIT_BUCK**	RW	0xC	Upper limit duty cycle limit in DC-DC converter. This field will limit the maximum VDDIO acheivable for a given battery voltage, and it's value may be increased if very low battery operation is desired.
7	**RSRVD1**	RO	0x0	Empty Description.
6:0	**NEGLIMIT**	RW	0x5F	Negative duty cycle limit of DC-DC converter.

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.12 Converter Loop Behavior Control Register Description

This register defines the control loop parameters available for the DC-DC converter.

HW_POWER_LOOPCTRL	0x0B0
HW_POWER_LOOPCTRL_SET	0x0B4
HW_POWER_LOOPCTRL_CLR	0x0B8
HW_POWER_LOOPCTRL_TOG	0x0BC

Table 32-24. HW_POWER_LOOPCTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD3											TOGGLE_DIF	HYST_SIGN	EN_CM_HYST	EN_DF_HYST	CM_HYST_THRESH	DF_HYST_THRESH	RCSCALE_THRESH	EN_RCSCALE	RSRVD2	DC_FF			DC_R				RSRVD1		DC_C	

Table 32-25. HW_POWER_LOOPCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:21	**RSRVD3**	RO	0x0	Empty Description.
20	**TOGGLE_DIF**	RW	0x0	Set high to enable supply stepping to change only after the differential control loop has toggled as well. This should eliminate any chance of large transients when supply voltage changes are made.
19	**HYST_SIGN**	RW	0x0	Invert the sign of the hysteresis in DC-DC analog comparators. This bit should set when using PFM mode.

Table 32-25. HW_POWER_LOOPCTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
18	**EN_CM_HYST**	RW	0x0	Enable hysteresis in switching converter common mode analog comparator. This feature will improve transient supply ripple and efficiency.
17	**EN_DF_HYST**	RW	0x0	Enable hysteresis in switching converter differential mode analog comparators. This feature will improve transient supply ripple and efficiency.
16	**CM_HYST_THRESH**	RW	0x0	Increase the threshold detection for common mode analog comparator.
15	**DF_HYST_THRESH**	RW	0x0	Increase the threshold detection for common mode analog comparator.
14	**RCSCALE_THRESH**	RW	0x0	Increase the threshold detection for RC scale circuit.
13:12	**EN_RCSCALE**	RW	0x0	Enable analog circuit of DC-DC converter to respond faster under transient load conditions. 00: disabled 01: 2X increase 10: 4X increase 11: 8X increase
11	**RSRVD2**	RO	0x0	Empty Description.
10:8	**DC_FF**	RW	0x0	Two's complement feed forward step in duty cycle in the switching DC-DC converter. Each time this field makes a transition from 0x0, the loop filter of the DC-DC converter is stepped once by a value proportional to the change. This can be used to force a certain control loop behavior, such as improving response under known heavy load transients.
7:4	**DC_R**	RW	0x2	Magnitude of proportional control parameter in the switching DC-DC converter control loop.
3:2	**RSRVD1**	RO	0x0	Empty Description.
1:0	**DC_C**	RW	0x1	Ratio of integral control parameter to proportional control parameter in the switching DC-DC converter, and can be used to optimize efficiency and loop response. 00: Maximum 01: Decrease ratio 2X 10: Decrease ratio 4X 11: Lowest ratio.

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.13 Power Subsystem Status Register Description

This register contains status information for the battery charger, DCDC converter and USB/OTG connections.

HW_POWER_STS 0x0C0

Table 32-26. HW_POWER_STS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSVD4		PWRUP_SOURCE						RSVD3		PSWITCH		RSVD2		AVALID_STATUS	BVALID_STATUS	VBUSVALID_STATUS	SESSEND_STATUS	BATT_BO	VDD5V_FAULT	CHRGSTS	DCDC_4P2_BO	RSVD1	VDDIO_BO	VDDA_BO	VDDD_BO	VDD5V_GT_VDDIO	VDD5V_DROOP	AVALID	BVALID	VBUSVALID	SESSEND

Table 32-27. HW_POWER_STS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSVD4**	RO	0x0	Empty Description.
29:24	**PWRUP_SOURCE**	RO	0x0	These read-only bits determine which source was active when the dcdc converter powerup sequence was complete. This can be used to determine what event caused the device to powerup. bit5 : five volts bit4 : rtc wakeup bit3 : reserved bit2 : reserved bit1 : high level pswitch voltage bit0 : midlevel pswitch voltage
23:22	**RSVD3**	RO	0x0	Empty Description.
21:20	**PSWITCH**	RO	0x0	These read-only bits reflect the current state of the pswitch comparators. The lsb is high when voltage on the PSWITCH pin is above 0.8V, and the msb is high when the voltage on the PSWITCH pin is above 1.75V
19:18	**RSVD2**	RO	0x0	Empty Description.
17	**AVALID_STATUS**	RO	0x0	Indicates VBus is valid for a A-peripheral. This bit is a read only version of the state of the analog signal. It can not be overritten by software like the AVALID bit below.
16	**BVALID_STATUS**	RO	0x0	Indicates VBus is valid for a B-peripheral. This bit is a read only version of the state of the analog signal. It can not be overwritten by software like the BVALID bit below.
15	**VBUSVALID_STATUS**	RO	0x0	VBus valid for USB OTG. This bit is a read only version of the state of the analog signal. It can not be overwritten by software like the VBUSVALID bit below.
14	**SESSEND_STATUS**	RO	0x0	Session End for USB OTG. This bit is a read only version of the state of the analog signal. It can not be overwritten by software like the SESSEND bit below.
13	**BATT_BO**	RO	0x0	Output of battery brownout comparator.

Table 32-27. HW_POWER_STS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
12	**VDD5V_FAULT**	RO	0x0	Battery charging fault status. If the battery charger is not powered down, the bit is high when the 5V supply falls below the battery voltage. If the charger is powered down, the bit asserts high when 5V falls to below roughly VDDIO/2. If the charger is not powered down, the bit is sticky and remains set until the PWD_CHARGE_4P2 bit is cycled. Otherwise, the bit is cleared when 5V is restored.
11	**CHRGSTS**	RO	0x0	Battery charging status. High during Li-Ion battery charge until the charging current falls below the STOP_ILIMIT threshold.
10	**DCDC_4P2_BO**	RO	0x0	Output of the brownout comparator on the DCDC_4P2 pin
9	**RSVD1**	RO	0x0	Program this field to 0x0.
8	**VDDIO_BO**	RO	0x0	Output of VDDIO brownout comparator. High when a brownout is detected. This comparator defaults powered up, but can be powered down via the POWER_MINPWR register.
7	**VDDA_BO**	RO	0x0	Output of VDDA brownout comparator. High when a brownout is detected. It is not possible to power-down this comparator.
6	**VDDD_BO**	RO	0x0	Output of VDDD brownout comparator. High when a brownout is detected. It is not possible to power-down this comparator.
5	**VDD5V_GT_VDDIO**	RO	0x0	Indicates the voltage on the VDD5V pin is higher than VDDIO by a Vt voltage, nominally 500 mV.
4	**VDD5V_DROOP**	RO	0x0	Indicates the voltage on the VDD5V pin is below the VBUSDROOP_TRSH defined in the 5VCTRL register.
3	**AVALID**	RW	0x0	Indicates VBus is above the VA_SESS_VLD threshold, i.e. high if VBus greater than 2.0, low if VBus less than 0.8, otherwise unknown.
2	**BVALID**	RW	0x0	Indicates VBus is above the VB_SESS_VLD threshold, high if VBus greater than 4.0, low if VBus less than 0.8, otherwise unknown.
1	**VBUSVALID**	RW	0x0	Accurate detection of the presence of 5v power. This can be used for detection of 5v in all modes of operation including USB OTG. See POWER_5VCTRL to enable and set threshold for comparison.
0	**SESSEND**	RW	0x0	Indicates VBus is below the VB_SESS_END threshold, i.e. 0 if VBus is greater than 0.8 V, 1 if VBus is less than 0.2 V, otherwise unknown. See POWER_5VCTRL to enable comparators.

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.14 Transistor Speed Control and Status Register Description

This register contains the setup and controls needed to measure silicon speed.

HW_POWER_SPEED	0x0D0
HW_POWER_SPEED_SET	0x0D4
HW_POWER_SPEED_CLR	0x0D8
HW_POWER_SPEED_TOG	0x0DC

Table 32-28. HW_POWER_SPEED

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1								STATUS								RSRVD0														CTRL	

Table 32-29. HW_POWER_SPEED Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSRVD1**	RO	0x0	Empty Description.
23:16	**STATUS**	RO	0x0	Result from the speed sensor. This result is only valid when SPEEDCTRL=0b11; otherwise this field contains debug information from the switching DC-DC converter.
15:2	**RSRVD0**	RO	0x0	Empty Description.
1:0	**CTRL**	RW	0x0	Speed Control bits. 00: Speed sensor off, 0b01: Speed sensor enabled, 11: Enable speed sensor measurement. Every time a measurement is taken, the sequence of 0x00 ; 01 ; 11 must be repeated. This sequence should proceed no faster than 1.5 MHz to ensure proper operation.

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.15 Battery Level Monitor Register Description

This register provides brownout controls and monitors the battery voltage.

HW_POWER_BATTMONITOR	0x0E0

Table 32-30. HW_POWER_BATTMONITOR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD3						BATT_VAL										RSRVD2					EN_BATADJ	PWDN_BATTBRNOUT	BRWNOUT_PWD	RSRVD1			BRWNOUT_LVL				

Table 32-31. HW_POWER_BATTMONITOR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSRVD3**	RO	0x0	Empty Description.
25:16	**BATT_VAL**	RW	0x0	Software should be configured to place the battery voltage in this register measured with an 8-mV LSB resolution via the LRADC. This value is used by the DC-DC converter and must be correct before setting EN_BATADJ.
15:11	**RSRVD2**	RO	0x0	Empty Description.
10	**EN_BATADJ**	RW	0x0	This bit enables the DC-DC to improve efficiency and minimize ripple using the information from the BATT_VAL field. It is very important that BATT_VAL contain accurate information before setting EN_BATADJ.
9	**PWDN_BATTBRNOUT**	RW	0x1	Powers down the device after the DC-DC converter completeS startup if a battery brownout occurs. This function is only active when 5V is not present. Additionally, software should clear this bit and disable this function after the system is configured for a battery brownout and the battery brownout interrupt is enabled.
8	**BRWNOUT_PWD**	RW	0x0	Power-down circuitry for battery brownout detection. This bit should only be set when it is not important to montior battery brownouts and minimum system power consumption is required.
7:5	**RSRVD1**	RO	0x0	Empty Description.
4:0	**BRWNOUT_LVL**	RW	0x0	The default setting of the brownout settings decode to a voltage as follows: Li-Ion = 2.4 V The voltage level can be calculated for other values by the following equation: Li-Ion brownout voltage = 2.4 V + 0.04 * BRWNOUT_LVL

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.16 Power Module Reset Register Description

This register allows software to put the chip into the off state.

HW_POWER_RESET	0x100
HW_POWER_RESET_SET	0x104
HW_POWER_RESET_CLR	0x108
HW_POWER_RESET_TOG	0x10C

Table 32-32. HW_POWER_RESET

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
UNLOCK																RSRVD1														PWD_OFF	PWD

Table 32-33. HW_POWER_RESET Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**UNLOCK**	RW	0x0	Write 0x3E77 to unlock this register and allow other bits to be changed. NOTE: This register must be unlocked on a write-by-write basis, so the UNLOCK bitfield must contain the correct key value during all writes to this register in order to update any other bitfield values in the register. KEY = 0x3E77 Key needed to unlock HW_POWER_RESET register.
15:2	**RSRVD1**	RO	0x0000	Empty Description.
1	**PWD_OFF**	RW	0x0	Optional bit to disable all paths to power off the chip except the watchdog timer. Setting this bit will be useful for preventing fast falling edges on the PSWITCH pin from resetting the chip. It may also be useful increasing system tolerance of noisy EMI environments. Note that setting this bit disables most automatic hardware shutdown protection mechanisms such as PWDN_BATTBRNOUT and the PWDN_5VBRNOUT. Before setting PWD_OFF, the application must configure the proper FIQ handled replacements for these protection mechanisms.
0	**PWD**	RW	0x0	Powers down the chip.

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.17 Power Module Debug Register Description

Debug Register.

HW_POWER_DEBUG	0x110
HW_POWER_DEBUG_SET	0x114
HW_POWER_DEBUG_CLR	0x118
HW_POWER_DEBUG_TOG	0x11C

Table 32-34. HW_POWER_DEBUG

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD0																												VBUSVALIDPIOLOCK	AVALIDPIOLOCK	BVALIDPIOLOCK	SESSENDPIOLOCK

Table 32-35. HW_POWER_DEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:4	**RSRVD0**	RO	0x0	Empty Description.
3	**VBUSVALIDPIOLOCK**	RW	0x0	Empty Description.
2	**AVALIDPIOLOCK**	RW	0x0	Empty Description.
1	**BVALIDPIOLOCK**	RW	0x0	Empty Description.
0	**SESSENDPIOLOCK**	RW	0x0	Empty Description.

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.18 Power Module Special Register Description

Special test functionality.

HW_POWER_SPECIAL	0x120
HW_POWER_SPECIAL_SET	0x124
HW_POWER_SPECIAL_CLR	0x128
HW_POWER_SPECIAL_TOG	0x12C

Table 32-36. HW_POWER_SPECIAL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TEST																															

Table 32-37. HW_POWER_SPECIAL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**TEST**	RW	0x0	

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.19 Power Module Version Register Description

This register always returns a known read value for debug purposes it indicates the version of the block.

HW_POWER_VERSION 0x130

Table 32-38. HW_POWER_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 32-39. HW_POWER_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x03	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x01	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
Empty Example.
```

POWER Block v3.1, Revision 1.0

Chapter 33
Low-Resolution ADC and Touch-Screen Interface

This chapter describes the low-resolution analog-to-digital converters and touch-screen interface included on the i.MX23. It includes sections on scheduling conversions and delay channels. Programmable registers are described in Section 33.4, "Programmable Registers."

33.1 Overview

The sixteen-channel low-resolution ADC (LRADC) block is used for voltage measurement Figure 33-1 shows a block diagram of the LRADC. Eight "virtual" channels can be used at one time. Each of the eight virtual channels can be mapped to any of the 16 physical channels using HW_LRADC_CTRL4. Six physical channels are available for general use.

- Channel 15 is dedicated to measuring the voltage on the VDD5V pin and can be used to detect possible issues with 5V rail drooping.
- Channel 14 is dedicated to measuring the bandgap reference voltage and can be used to calibrate out a portion of the LRADC measurement error (comparator offset, buffer amp offset, and DAC offset). In most cases, the bandgap reference error (specified to ±1%) will dominate the total LRADC error, and this calibration will not be helpful. But if the bandgap reference is calibrated using the fuses, then it is possible that LRADC accuracy will be limited by these other sources and that using the VBG input for calibration of the LRADC can improve accuracy further.
- Channel 12 and 13 are dedicated to measuring the voltage on the USB_DP and USB_DN pins. This is to be used only in non-USB mode and can be used for special peripheral circuitry detection. HW_USBPHY_CTRL_DATA_ON_LRADC must be set to measure these inputs.
- Channel 10 and 11 are reserved inputs for analog testing.
- Channel 8 and 9 are dedicated to measuring the internal die temperature. HW_LRADC_CTRL2_TEMPSENSE_PWD must be cleared for these inputs to function. See Section 33.2.2, "Internal Die Temperature Sensing."
- Channel 7 is dedicated to measuring the voltage on the BATT pin and can be used to sense the amount of battery life remaining.
- Channel 6 is dedicated to measuring the voltage on the VDDIO Rail and is used to calibrate the voltage levels measured on the auxiliary channels when those inputs are resistor divided from the VDDIO rail.
- On the 128 -pin LQFP package, LRADC4 is bonded to the VDDM 2.5 V regulator output. To use this pin as an LRADC input, the VDDM 2.5 V LDO must be disabled.

The LRADC has 12 bits of resolution and an absolute accuracy of 1.3% limited primarily by the bandgap voltage reference accuracy. If the bandgap voltage reference is calibrated with the fuses, the LRADC

absolute accuracy might be improved to better than 0.5%. All channels sample on the same divided clock rate from the 24.0-MHz crystal clock. The LRADC controller includes an integrated touch-screen controller with drive voltage generation for touch-screen coordinate measurement, as well as a touch-detection interrupt circuit. The LRADC controller also contains four delay-control channels that can be used to automatically time and schedule control events within the LRADC.

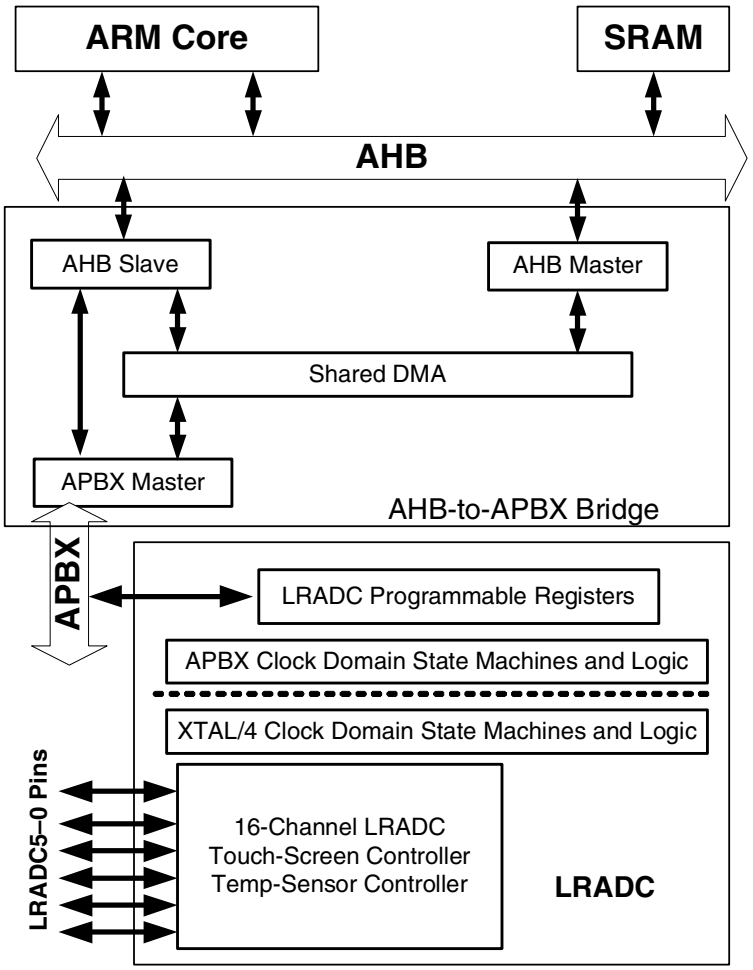


Figure 33-1. Low-Resolution ADC and Touch-Screen Interface Block Diagram

33.2 Operation

All channels of the LRADC share a common successive approximation style analog-to-digital converter through a common analog mux front end (see Figure 33-1).

- The BATT pin has a built-in 4:1 voltage divider on its analog multiplexer input that is activated only in Li-Ion battery mode.
- The Channel 15 5V input also has a built-in 4:1 divider on its input.
- The Channel 6 VDDIO input has a built-in 2:1 divider on its input. The maximum analog input voltage into the LRADC is 1.85 V.

- For input channels (other than BATT, 5V, or VDDIO) with signals larger than 1.85 V, the divide-by-two option should be set (HW_LRADC_CTRL2_DIVIDE_BY_TWO). With the DIVIDE_BY_TWO option set, the maximum input voltage is VDDIO – 50mv.

The touch-screen driver works for typical touch-screen impedances of 200–900 ohm and for high impedance touch-panels with impedances in the 50-Kohm range.

The LRADC channels 0 and 1 have optional current source outputs to allow these channels to be used with an external themistor (or an external diode) for temperature sensing. The controls for these current sources are in HW_LRADC_CTRL2. The current source values can be changed to allow significant temperature sensing range using a thermistor or to use a diode for temperature sensing. The currents are derived using the on-chip 1% accurate bandgap voltage reference and an optionally tuned on-chip poly resistor. The accuracy of the current source is limited by the on-chip resistor, which should be 5% accurate and optionally tuned for higher accuracy (with efuses). Most thermistors are no more than 5% accurate, so this level of current source accuracy is acceptable for most applications. For temperature sensing with higher accuracy, customers can use a 1% resistor divider from VDDIO with the thermistor. In this case, the thermistor will be the dominant source of error.

33.2.1 External Temperature Sensing with a Diode

Using a diode instead of a thermistor for external temperature sensing can be cheaper and provide greater temperature range for a given accuracy level. A cheap diode like a 1N4148 is connected between ground and either LRADC 0 or 1. Two voltage measurements are taken—first with the HW_LRADC_CTRL2_TEMP_ISRC current source set at 300 µA, then another voltage measurement with the current source set at 20 µA. The temperature will be roughly:

degrees Kelvin = (Vmax – Vmin) / 0.409 mV

or, from the LRADC conversion (LSB=0.45mV):

degrees Kelvin = (Codemax – Codemin) * 1.104

Freescale recommends taking 5–10 samples for the min and max and then averaging them to get a good reading.

The temperature reading error will likely be dominated by part-to-part matching of the diodes. Some manufacturers' diodes show substantially less variation than others. Freescale has shown 3sigma accuracy of +/-7.5C using Fairchild MMBD914 (from multiple batches of diodes).

If better accuracy is required, Freescale recommends using a thermistor for external temperature sensing. The thermistor will be more accurate, but over a smaller temperature range than the diode method.

Any routing impedance to the diode will cause a shift in the temperature reading. This can be measured and corrected in software for each design.

Two ohms of routing impedance would cause (2 * (300µA – 20µA) error of 0.56 mV or 1.25 degrees C error.

33.2.2 Internal Die Temperature Sensing

The i.MX23 has a new internal die temperature sensor that uses two of the sixteen physical LRADC channels. To use the internal die temperature sensor, HW_LRADC_CTRL2_TEMPSENSE_PWD should be cleared. (This bit can be left cleared after power up. There is no need to toggle it on and off.) Two of the eight virtual LRADC channels need to be mapped to the temperature sensing channels 8 and 9 using HW_LRADC_CTRL4. Then, these virtual LRADC channels should BOTH be converted using the LRADC conversion scheduler described below. The temperature in degrees Kelvin will be equal to:

T = (Channel9 – Channel8) * Gain_correction/4

The Gain_correction corrects a mean gain error in the temperature conversion and should be 1.012. After this correction factor, the three-sigma error of the temperature sensor should be within ± 1.5% in degrees Kelvin. Additionally, the temperature sampling has a three-sigma sample-to-sample variation of 2 degrees Kelvin. If desired, this error can be removed by oversampling and averaging the temperature result.

Prior to starting a battery charge cycle, the internal die temperature sensing could be used for an approximate ambient temperature. During high-current battery charging, the temperature sensor can be used as extra protection to avoid excessive die temperatures (to throttle the charging current).

33.2.3 Scheduling Conversions

The APBX clock domain logic schedules conversions on a per-channel basis and handles interrupt processing back to the CPU. Each of the eight virtual channels has its own interrupt request enable bit and its own interrupt request status bit.

A schedule request bit, HW_LRADC_CTRL0_SCHEDULE, exists for each virtual channel. Setting this bit causes the LRADC to schedule a conversion for that virtual channel. Each virtual channel schedule bit is sequentially checked and, if scheduled, causes a conversion. The schedule bit is cleared upon completion of a successive approximation conversion, and its corresponding interrupt request status bit is set. Thus, software controls how often a conversion is requested. As each scheduled channel is converted, its interrupt status bit is set and its schedule bit is reset.

There is a mechanism to continuously reschedule a conversion for a particular virtual channel. With set/clear/toggle addressing modes, independent threads can request conversions without needing any information from unrelated threads using other channels. Setting a schedule bit can be performed in an atomic way. Setting a “gang” of four channel-schedule bits can also be performed atomically. The LRADC scheduler is round-robin. It snapshots all schedule bits at once, and then processes them in sequence until all are converted. It then monitors the schedule bits. If any schedule bits are set, it snap-

shots them and starts a new conversion operation for all scheduled channels. Thus, one can set the schedule bits for four channels on the same clock edge. The channel with the largest channel number is converted last and has its interrupt status bit set last. If that channel is the only one of the four with an interrupt enable bit set, then it interrupts the ARM after all four channels have been converted, effectively ganging four channels together.

33.2.4 Delay Channels

To minimize the interrupt load on the ARM processor, four delay channels are provided. Each has an 11-bit counter that increments at 2 kHz. A delay channel can be kicked off either by an ARM store instruction or at the completion of a delay channel time-out. At time-out, each channel has the option of kicking off any combination of LRADC conversions, as well as any combination of delay channels.

NOTE: The DELAY fields in HW_LRADC_DELAY0, HW_LRADC_DELAY1, HW_LRADC_DELAY2, and HW_LRADC_DELAY3 must be non-zero; otherwise, the LRADC will not trigger the delay group. The ACCUMULATE bit in the appropriate channel register HW_LRADC_CHn must be set to 1 if NUM_SAMPLES is greater then 0; otherwise, the IRQs will not fire.

Consider the case of a touch-screen that requires 4x oversampling of its coordinate values. Further, suppose you wish to receive an oversampled X or Y coordinate approximately every 5 ms and that the oversampling should be spaced at 1-ms intervals.

- In the touch-screen, first select either X or Y drive, then set up the appropriate LRADC.
- In setting up the LRADC, clear the accumulator associated with it by setting the ACCUMULATE bit and set the NUM_SAMPLES field to 3 (4 samples before interrupt request).
- Next, set up two delay channels.
  - Delay Channel 1 is set to delay 1 ms (DELAY = 1, two ticks) and then kick the schedule bit for LRADC 4. Its LOOP_COUNT bit field is also set to 3, so that four kicks of LRADC 4 occur, each spaced by 1 ms.
  - Delay Channel 0 is set to delay 1 ms with LOOP_COUNT = 0, i.e., one time. Its TRIGER_DELAYS field is set to trigger Delay Channel 1 when it times out. The ISR routine kicks off Delay Channel 0 immediately before it does its return from interrupt. Another interrupt (LRADC4_IRQ) is asserted once the entire 4x oversample data capture is complete. A sample timeline for such a sequence is shown in Figure 33-3.

NOTE: If a delay group schedules channels to be sampled and a manual write to the schedule field in CTRL0 occurs while the block is discarding samples, the LRADC will switch to the new schedule and will not sample the channels that were previously scheduled. The time window for this to happen is very small and lasts only while the LRADC is discarding samples.

WARNING: The pad ESD protection limits the voltage on the LRADC0-LRADC6 inputs to VDDIO. The BATT and 5V inputs to the LRADC have built-in dividers to handle the higher voltages.

33.3 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, “Correct Way to Soft Reset a Block,” for additional information on using the SFTRST and CLKGATE bit fields.

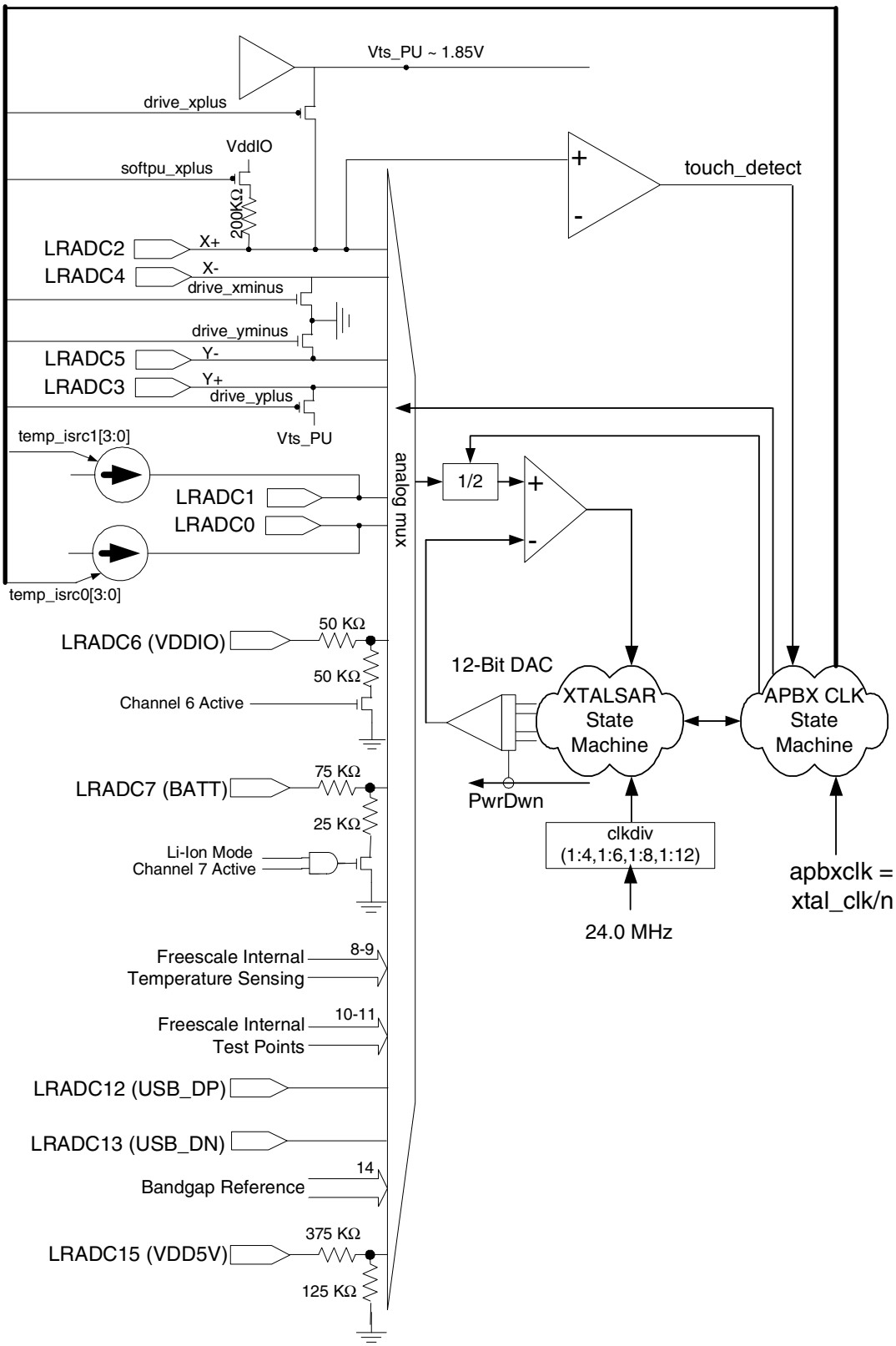


Figure 33-2. Low-Resolution ADC Successive Approximation Unit

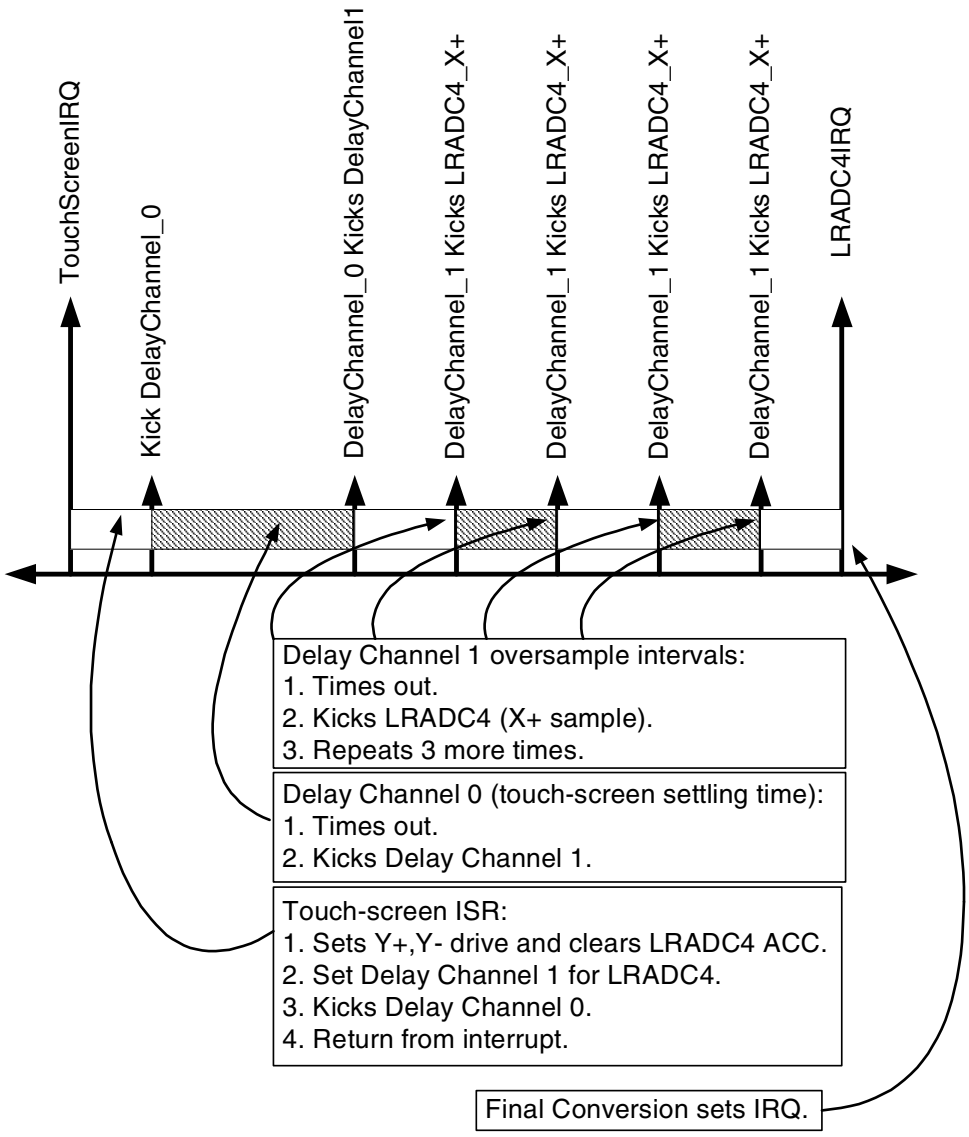


Figure 33-3. Using Delay Channels to Oversample a Touch-Screen

33.4 Programmable Registers

The following registers describe the programming interfaces for the Low-Resolution ADC and Touch-Screen Interface.

33.4.1 LRADC Control Register 0 Description

The LRADC Control Register 0 provides overall control of the eight low resolution analog to digital converters.

HW_LRADC_CTRL0	0x000
HW_LRADC_CTRL0_SET	0x004

HW_LRADC_CTRL0_CLR 0x008
HW_LRADC_CTRL0_TOG 0x00C

Table 33-1. HW_LRADC_CTRL0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	RSRVD2								ONCHIP_GROUNDREF	TOUCH_DETECT_ENABLE	YMINUS_ENABLE	XMINUS_ENABLE	YPLUS_ENABLE	XPLUS_ENABLE	RSRVD1								SCHEDULE							

Table 33-2. HW_LRADC_CTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	When set to one, this bit causes a reset to the entire LRADC block. In addition, it turns off the converter clock and powers down the analog portion of the LRADC. Set this bit to zero for normal operation.
30	**CLKGATE**	RW	0x1	This bit must be set to zero for normal operation. When set to one it gates off the clocks to the block.
29:22	**RSRVD2**	RO	0x0	Reserved
21	**ONCHIP_GROUNDREF**	RW	0x0	Set this bit to one to use the on-chip ground as reference for conversions. OFF = 0x0 Turn it off. ON = 0x1 Turn it on.
20	**TOUCH_DETECT_ENABLE**	RW	0x0	Set this bit to one to enable touch panel touch detector. OFF = 0x0 Turn it off. ON = 0x1 Turn it on.
19	**YMINUS_ENABLE**	RW	0x0	Set this bit to one to enable yminus pull down on the LRADC5 pin. OFF = 0x0 Turn it off. ON = 0x1 Turn it on.
18	**XMINUS_ENABLE**	RW	0x0	Set this bit to one to enable xminus pull down on the LRADC4 pin. OFF = 0x0 Turn it off. ON = 0x1 Turn it on.
17	**YPLUS_ENABLE**	RW	0x0	Set this bit to one to enable yplus pull up on the LRADC3 pin. . OFF = 0x0 Turn it off. ON = 0x1 Turn it on.
16	**XPLUS_ENABLE**	RW	0x0	Set this bit to one to enable xplus pull up on the LRADC2 pin. OFF = 0x0 Turn it off. ON = 0x1 Turn it on.

Table 33-2. HW_LRADC_CTRL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:8	**RSRVD1**	RO	0x00	Reserved
7:0	**SCHEDULE**	RW	0x00	Setting a bit to one schedules the corresponding LRADC channel to be converted. When the conversion of a scheduled channel is completed the corresponding schedule bit is reset by the hardware and the corresponding interrupt request is set to one. Thus any thread can request a conversion asynchronously from any other thread.

DESCRIPTION:

The LRADC control register provides control over the shared eight channel LRADC converter. It allows software to independently schedule conversion cycles on any number of any size subsets of the eight channels. In addition it allows software to manage the interrupt reporting for the channel conversion sets.

EXAMPLE:

```
BW_LRADC_CTRL0_YPLUS_ENABLE(BV_LRADC_CTRL0_YPLUS_ENABLE__ON);
```

33.4.2 LRADC Control Register 1 Description

The LRADC Control Register 1 provides overall control of the eight low resolution analog to digital converters.

HW_LRADC_CTRL1	0x010
HW_LRADC_CTRL1_SET	0x014
HW_LRADC_CTRL1_CLR	0x018
HW_LRADC_CTRL1_TOG	0x01C

Table 33-3. HW_LRADC_CTRL1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2							TOUCH_DETECT_IRQ_EN	LRADC7_IRQ_EN	LRADC6_IRQ_EN	LRADC5_IRQ_EN	LRADC4_IRQ_EN	LRADC3_IRQ_EN	LRADC2_IRQ_EN	LRADC1_IRQ_EN	LRADC0_IRQ_EN	RSRVD1								TOUCH_DETECT_IRQ	LRADC7_IRQ	LRADC6_IRQ	LRADC5_IRQ	LRADC4_IRQ	LRADC3_IRQ	LRADC2_IRQ	LRADC1_IRQ	LRADC0_IRQ

Table 33-4. HW_LRADC_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSRVD2**	RO	0x00	Reserved
24	**TOUCH_DETECT_IRQ_EN**	RW	0x0	Set to one to enable an interrupt for the touch detector comparator. DISABLE = 0x0 Disable Interrupt request. ENABLE = 0x1 Enable Interrupt request.

Table 33-4. HW_LRADC_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
23	**LRADC7_IRQ_EN**	RW	0x0	Set to one to enable an interrupt for channel 7 (BATT) conversions. DISABLE = 0x0 Disable Interrupt request. ENABLE = 0x1 Enable Interrupt request.
22	**LRADC6_IRQ_EN**	RW	0x0	Set to one to enable an interrupt for channel 6 (VddIO) conversions. DISABLE = 0x0 Disable Interrupt request. ENABLE = 0x1 Enable Interrupt request.
21	**LRADC5_IRQ_EN**	RW	0x0	Set to one to enable an interrupt for channel 5 conversions. DISABLE = 0x0 Disable Interrupt request. ENABLE = 0x1 Enable Interrupt request.
20	**LRADC4_IRQ_EN**	RW	0x0	Set to one to enable an interrupt for channel 4 conversions. DISABLE = 0x0 Disable Interrupt request. ENABLE = 0x1 Enable Interrupt request.
19	**LRADC3_IRQ_EN**	RW	0x0	Set to one to enable an interrupt for channel 3 conversions. DISABLE = 0x0 Disable Interrupt request. ENABLE = 0x1 Enable Interrupt request.
18	**LRADC2_IRQ_EN**	RW	0x0	Set to one to enable an interrupt for channel 2 conversions. DISABLE = 0x0 Disable Interrupt request. ENABLE = 0x1 Enable Interrupt request.
17	**LRADC1_IRQ_EN**	RW	0x0	Set to one to enable an interrupt for channel 1 conversions. DISABLE = 0x0 Disable Interrupt request. ENABLE = 0x1 Enable Interrupt request.
16	**LRADC0_IRQ_EN**	RW	0x0	Set to one to enable an interrupt for channel 0 conversions. DISABLE = 0x0 Disable Interrupt request. ENABLE = 0x1 Enable Interrupt request.
15:9	**RSRVD1**	RO	0x00	Reserved
8	**TOUCH_DETECT_IRQ**	RW	0x0	This bit is set to one upon detection of a touch condition in the touch panel attached to LRADC2-LRADC5. It is ANDed with its corresponding interrupt enable bit to request an interrrupt. Once set by the conversion hardware, this bit remains set until cleared by software. CLEAR = 0x0 Interrupt request cleared. PENDING = 0x1 Interrupt request pending.
7	**LRADC7_IRQ**	RW	0x0	This bit is set to one upon completion of a scheduled conversion for channel 7(BATT). It is ANDed with its corresponding interrupt enable bit to request an interrrupt. Once set by the conversion hardware, this bit remains set until cleared by software. CLEAR = 0x0 Interrupt request cleared. PENDING = 0x1 Interrupt request pending.
6	**LRADC6_IRQ**	RW	0x0	This bit is set to one upon completion of a scheduled conversion for channel 6(VDDIO). It is ANDed with its corresponding interrupt enable bit to request an interrrupt. Once set by the conversion hardware, this bit remains set until cleared by software. CLEAR = 0x0 Interrupt request cleared. PENDING = 0x1 Interrupt request pending.

Table 33-4. HW_LRADC_CTRL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
5	**LRADC5_IRQ**	RW	0x0	This bit is set to one upon completion of a scheduled conversion for channel 5. It is ANDed with its corresponding interrupt enable bit to request an interrrupt. Once set by the conversion hardware, this bit remains set until cleared by software. CLEAR = 0x0 Interrupt request cleared. PENDING = 0x1 Interrupt request pending.
4	**LRADC4_IRQ**	RW	0x0	This bit is set to one upon completion of a scheduled conversion for channel 4. It is ANDed with its corresponding interrupt enable bit to request an interrrupt. Once set by the conversion hardware, this bit remains set until cleared by software. CLEAR = 0x0 Interrupt request cleared. PENDING = 0x1 Interrupt request pending.
3	**LRADC3_IRQ**	RW	0x0	This bit is set to one upon completion of a scheduled conversion for channel 3. It is ANDed with its corresponding interrupt enable bit to request an interrrupt. Once set by the conversion hardware, this bit remains set until cleared by software. CLEAR = 0x0 Interrupt request cleared. PENDING = 0x1 Interrupt request pending.
2	**LRADC2_IRQ**	RW	0x0	This bit is set to one upon completion of a scheduled conversion for channel 2. It is ANDed with its corresponding interrupt enable bit to request an interrrupt. Once set by the conversion hardware, this bit remains set until cleared by software. CLEAR = 0x0 Interrupt request cleared. PENDING = 0x1 Interrupt request pending.
1	**LRADC1_IRQ**	RW	0x0	This bit is set to one upon completion of a scheduled conversion for channel 1. It is ANDed with its corresponding interrupt enable bit to request an interrrupt. Once set by the conversion hardware, this bit remains set until cleared by software. CLEAR = 0x0 Interrupt request cleared. PENDING = 0x1 Interrupt request pending.
0	**LRADC0_IRQ**	RW	0x0	This bit is set to one upon completion of a scheduled conversion for channel 0. It is ANDed with its corresponding interrupt enable bit to request an interrrupt. Once set by the conversion hardware, this bit remains set until cleared by software. CLEAR = 0x0 Interrupt request cleared. PENDING = 0x1 Interrupt request pending.

DESCRIPTION:

The LRADC control register 1 provides control over the shared eight channel LRADC converter. It allows software to independently schedule conversion cycles on any number of any size subsets of the eight channels. In addition it allows software to manage the interrupt reporting for the channel conversion sets.

EXAMPLE:

```
if(HW_LRADC_CTRL1.TOUCH_DETECT_IRQ == BV_LRADC_CTRL1_TOUCH_DETECT_IRQ__PENDING){
// Then handle the interrupt.
HW_LRADC_CTRL1.TOUCH_DETECT_IRQ_EN = BV_LRADC_CTRL1_TOUCH_DETECT_IRQ_EN__DISABLE;
}
```

33.4.3 LRADC Control Register 2 Description

The LRADC Control Register 2 provides overall control of the eight low resolution analog to digital converters.

HW_LRADC_CTRL2	0x020
HW_LRADC_CTRL2_SET	0x024
HW_LRADC_CTRL2_CLR	0x028
HW_LRADC_CTRL2_TOG	0x02C

Table 33-5. HW_LRADC_CTRL2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DIVIDE_BY_TWO								BL_AMP_BYPASS	BL_ENABLE	BL_MUX_SELECT	BL_BRIGHTNESS					TEMPSENSE_PWD	RSRVD1	EXT_EN1	EXT_EN0	RSRVD2		TEMP_SENSOR_IENABLE1	TEMP_SENSOR_IENABLE0	TEMP_ISRC1				TEMP_ISRC0			

Table 33-6. HW_LRADC_CTRL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**DIVIDE_BY_TWO**	RW	0x0	Each bit of this eight bit field corresponds to a channel of an LRADC. Setting the bit to one caused the A/D converter to use its analog divide by two circuit for the conversion of the corresponding channel.
23	**BL_AMP_BYPASS**	RW	0x0	The analog feedback control signal is normally gained up by 4X. When this bit is one, the feedback control signal bypasses the gain of 4 stage. DISABLE = 0x0 . ENABLE = 0x1 .
22	**BL_ENABLE**	RW	0x0	Enables the back light.
21	**BL_MUX_SELECT**	RW	0x0	0 - Use pin LRADC4 for feedback control. 1 - Use pin LRADC1 for feedback control.
20:16	**BL_BRIGHTNESS**	RW	0x0	Sets the voltage comparison level for the analog feedback control. Each step is -1.293dB with the max voltage of 1.212V. BL_AMP_BYPASS 1 0 11111 = 1.213V 0.303 11110 = 1.046V 0.262 ..... 00001 = 0.0136V 0.0034 00000 = 0.0117V 0.0029
15	**TEMPSENSE_PWD**	RW	0x1	PWD the tempsense block. ENABLE = 0x0 When this is low the tempsense gain block muxes to LRADC channel 8 and 9. DISABLE = 0x1 .
14	**RSRVD1**	RO	0x00	Reserved

Table 33-6. HW_LRADC_CTRL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
13	**EXT_EN1**	RW	0x0	These bits are not supported. DISABLE = 0x0 . ENABLE = 0x1 .
12	**EXT_EN0**	RW	0x0	When set to zero(default) the mux amp is bypassed when the LRADC input channel is not using the divide-by-two. When set to one the mux amp is never bypassed (old behavior).
11:10	**RSRVD2**	RO	0x00	Reserved
9	**TEMP_SENSOR_IENABLE1**	RW	0x0	Set this bit to one to enable the current source onto LRADC1. DISABLE = 0x0 Disable Temperature Sensor Current Source. ENABLE = 0x1 Enable Temperature Sensor Current Source.
8	**TEMP_SENSOR_IENABLE0**	RW	0x0	Set this bit to one to enable the current source onto LRADC0. DISABLE = 0x0 Disable Temperature Sensor Current Source. ENABLE = 0x1 Enable Temperature Sensor Current Source.
7:4	**TEMP_ISRC1**	RW	0x0	When the output voltage is lower than 1V the output current is 1uA higher than the decode shown above. This extra current drops to zero as the output voltage raises above 1.5V. This four bit field encodes the current magnitude to inject into an external temperature sensor attached to LRADC1. 300 = 0xF 300uA. 280 = 0xE 280uA. 260 = 0xD 260uA. 240 = 0xC 240uA. 220 = 0xB 220uA. 200 = 0xA 200uA. 180 = 0x9 180uA. 160 = 0x8 160uA. 140 = 0x7 140uA. 120 = 0x6 120uA. 100 = 0x5 100uA. 80 = 0x4 80uA. 60 = 0x3 60uA. 40 = 0x2 40uA. 20 = 0x1 20uA. ZERO = 0x0 0uA.
3:0	**TEMP_ISRC0**	RW	0x0	When the output voltage is lower than 1V the output current is 1uA higher than the decode shown above. This extra current drops to zero as the output voltage raises above 1.5V. This four bit field encodes the current magnitude to inject into an external temperature sensor attached to LRADC0. 300 = 0xF 300uA. 280 = 0xE 280uA. 260 = 0xD 260uA. 240 = 0xC 240uA. 220 = 0xB 220uA. 200 = 0xA 200uA. 180 = 0x9 180uA. 160 = 0x8 160uA. 140 = 0x7 140uA. 120 = 0x6 120uA. 100 = 0x5 100uA. 80 = 0x4 80uA. 60 = 0x3 60uA. 40 = 0x2 40uA. 20 = 0x1 20uA. ZERO = 0x0 0uA.

DESCRIPTION:

The LRADC control register 2 provides control over the shared eight channel LRADC converter. It allows software to independently schedule conversion cycles on any number of any size subsets of the eight channels. In addition it allows software to manage the interrupt reporting for the channel conversion sets.

EXAMPLE:

```
BW_LRADC_CTRL2_TEMP_SENSOR_IENABLE1(BV_LRADC_CTRL2_TEMP_SENSOR_IENABLE1__DISABLE);
```

33.4.4 LRADC Control Register 3 Description

The LRADC touch panel control register specifies the voltages at which a touch detect interrupt is generated.

HW_LRADC_CTRL3	0x030
HW_LRADC_CTRL3_SET	0x034
HW_LRADC_CTRL3_CLR	0x038
HW_LRADC_CTRL3_TOG	0x03C

Table 33-7. HW_LRADC_CTRL3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD5						DISCARD		FORCE_ANALOG_PWUP	FORCE_ANALOG_PWDN	RSRVD4								RSRVD3				CYCLE_TIME		RSRVD2		HIGH_TIME		RSRVD1		DELAY_CLOCK	INVERT_CLOCK

Table 33-8. HW_LRADC_CTRL3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:26	**RSRVD5**	RO	0x0	Reserved
25:24	**DISCARD**	RW	0x0	This bit field specifies the number of samples to discard whenever the LRADC analog is first powered up. 00= discard first 3 samples 01= discard first sample 10= discard first 2 samples 11= discard first 3 samples 1_SAMPLE = 0x1 discard first sample before first capture. 2_SAMPLES = 0x2 discard 2 samples before first capture. 3_SAMPLES = 0x3 discard 3 samples before first capture.
23	**FORCE_ANALOG_PWUP**	RW	0x0	Set this bit to zero for normal operation. Setting it to one forces an analog power up, regardless of where the digital state machine may be. OFF = 0x0 Turn it off. ON = 0x1 Turn it on.

Table 33-8. HW_LRADC_CTRL3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
22	**FORCE_ANALOG_PWDN**	RW	0x0	Set this bit to zero for normal operation. Setting it to one forces an analog power down, regardless of where the digital state machine may be. ON = 0x0 Turn it on. OFF = 0x1 Turn it off.
21:14	**RSRVD4**	RO	0x0	Reserved
13:10	**RSRVD3**	RO	0x0	Reserved
9:8	**CYCLE_TIME**	RW	0x0	Changes the LRADC clock frequency. Note: the sample rate is one thirteenth of the frequency selected here. 00= 6MHz 01= 4MHz 10= 3MHz 11= 2MHz 6MHZ = 0x0 6MHz clock. 4MHZ = 0x1 4MHz clock. 3MHZ = 0x2 3MHz clock. 2MHZ = 0x3 2MHz clock.
7:6	**RSRVD2**	RO	0x0	Reserved
5:4	**HIGH_TIME**	RW	0x0	When CYCLE_TIME=00 only 00 and 01 are valid for HIGH_TIME. When CYCLE_TIME=01 only 00,01,and 10 are valid Changes the duty cycle (time high) for the LRADC clock. 00= 41.66ns 01= 83.33ns 10= 125ns 11= 250ns 42NS = 0x0 Duty cycle high time to 41.66ns. 83NS = 0x1 Duty cycle high time to 83.33ns. 125NS = 0x2 Duty cycle high time to 125ns. 250NS = 0x3 Duty cycle high time to 250ns.
3:2	**RSRVD1**	RO	0x0	Reserved
1	**DELAY_CLOCK**	RW	0x0	Set this bit to one to delay the 24MHz clock used in the LRADC even further away from the predominant rising edge used within the digital section. The delay inserted is approximately 400pS. NORMAL = 0x0 Normal operation, that is no delay. DELAYED = 0x1 Delay the clock.
0	**INVERT_CLOCK**	RW	0x0	Set this bit field to one to invert the 24MHz clock where it comes into the LRADC analog section. This moves it away from the predominant digital rising edge. Setting this bit to one causes the A/D converter to run from the negative edge of the divided clock, effectively shifting the conversion point away from the edge used by the DCDC converter. NORMAL = 0x0 Run the clock in normal that is not inverted mode. INVERT = 0x1 Inver the clock.

DESCRIPTION:

The LRADC touch detect control and status register controls the voltage at which a touch detection interrupt is generated. This register also contains the interrupt request status bit and enable bit for the touch detection interrupt request to the CPU's IRQ interrupt input.

EXAMPLE:

```
BW_LRADC_CTRL3_HIGH_TIME(BV_LRADC_CTRL3_HIGH_TIME__83NS);
BW_LRADC_CTRL3_INVERT_CLOCK(BV_LRADC_CTRL3_INVERT_CLOCK__NORMAL);
```

33.4.5 LRADC Status Register Description

The LRADC status register returns various read-only status bit field values.

HW_LRADC_STATUS	0x040
HW_LRADC_STATUS_SET	0x044
HW_LRADC_STATUS_CLR	0x048
HW_LRADC_STATUS_TOG	0x04C

Table 33-9. HW_LRADC_STATUS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD3					TEMP1_PRESENT	TEMP0_PRESENT	TOUCH_PANEL_PRESENT	CHANNEL7_PRESENT	CHANNEL6_PRESENT	CHANNEL5_PRESENT	CHANNEL4_PRESENT	CHANNEL3_PRESENT	CHANNEL2_PRESENT	CHANNEL1_PRESENT	CHANNEL0_PRESENT	RSRVD2																TOUCH_DETECT_RAW

Table 33-10. HW_LRADC_STATUS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:27	**RSRVD3**	RO	0x0	Reserved
26	**TEMP1_PRESENT**	RO	0x1	This read-only bit returns a one when the temperature sensor 1 current source is present on the chip.
25	**TEMP0_PRESENT**	RO	0x1	This read-only bit returns a one when the temperature sensor 0 current source is present on the chip.
24	**TOUCH_PANEL_PRESENT**	RO	0x1	This read-only bit returns a one when the touch panel controller function is present on the chip.
23	**CHANNEL7_PRESENT**	RO	0x1	This read-only bit returns a one when the LRADC channel 7 converter function is present on the chip.
22	**CHANNEL6_PRESENT**	RO	0x1	This read-only bit returns a one when the LRADC channel 6 converter function is present on the chip.
21	**CHANNEL5_PRESENT**	RO	0x1	This read-only bit returns a one when the LRADC channel 5 converter function is present on the chip.
20	**CHANNEL4_PRESENT**	RO	0x1	This read-only bit returns a one when the LRADC channel 4 converter function is present on the chip.
19	**CHANNEL3_PRESENT**	RO	0x1	This read-only bit returns a one when the LRADC channel 3 converter function is present on the chip.
18	**CHANNEL2_PRESENT**	RO	0x1	This read-only bit returns a one when the LRADC channel 2 converter function is present on the chip.
17	**CHANNEL1_PRESENT**	RO	0x1	This read-only bit returns a one when the LRADC channel 1 converter function is present on the chip.
16	**CHANNEL0_PRESENT**	RO	0x1	This read-only bit returns a one when the LRADC channel 0 converter function is present on the chip.

Table 33-10. HW_LRADC_STATUS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:1	**RSRVD2**	RO	0x0	Reserved
0	**TOUCH_DETECT_RAW**	RO	0x0	This read-only bit shows the status of the Touch Detect Comparator in the analog section. OPEN = 0x0 No contact, i.e. open connection. HIT = 0x1 Someone is touching the panel.

DESCRIPTION:

The status register returns the value of a number of status bit fields.

EXAMPLE:

```
if(HW_LRADC_STATUS.TOUCH_DETECT_RAW == BV_LRADC_STATUS_TOUCH_DETECT_RAW__HIT){
// Then something is touching the screen.
}
```

33.4.6 LRADC 0 Result Register Description

The LRADC result register returns the 12-bit result for low resolution analog to digital converter channel zero.

HW_LRADC_CH0	0x050
HW_LRADC_CH0_SET	0x054
HW_LRADC_CH0_CLR	0x058
HW_LRADC_CH0_TOG	0x05C

Table 33-11. HW_LRADC_CH0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TOGGLE	RSRVD2	ACCUMULATE	NUM_SAMPLES					RSRVD1								VALUE															

Table 33-12. HW_LRADC_CH0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**TOGGLE**	RW	0x0	This bit toggles at every completed conversion so software can detect a missed or duplicated sample.
30	**RSRVD2**	RO	0x0	Reserved
29	**ACCUMULATE**	RW	0x0	Set this bit to one to add successive samples to the 18 bit accumulator.
28:24	**NUM_SAMPLES**	RW	0x0	This bit field contains the number of conversion cycles to sum together before reporting operation complete interrupt status. Set this field to zero for a single conversion per interrupt.

Table 33-12. HW_LRADC_CH0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
23:18	**RSRVD1**	RO	0x000	Reserved
17:0	**VALUE**	RW	0x0000	This bit field contains the most recent 12-bit conversion value for this channel. If automatic oversampling is enbled this bit field contains the sum of the most recent N oversampled values, where N is set in the NUM_SAMPLES field for this channel. When 32 full-scale samples are added together, the 12-bit results can sum up to 256K. Software is responsible for dividing this value by the number of samples summed together. Software must clear this register in preparation for a multi-cycle accumulation.

DESCRIPTION:

The Result register contains the most recent conversion results for one channel of the LRADC. Note that each channel can be converted at an independent rate. The TOGGLE bit is used to debug missed conversion cycles. When using oversampling, the channel must be individualy scheduled for conversion N times for when N samples are required before an interrupt is generated. This is most easily accomplished by using one of the LRADC Delay Channels.

EXAMPLE:

```
if (HW_LRADC_CHn(0).TOGGLE == 1) { } // Toggle is high.
// ...
unsigned long channelAverage;
HW_LRADC_CHn_WR(0, (BF_LRADC_CHn_ACCUMULATE(1) | // Enable accumulation mode.
BF_LRADC_CHn_NUM_SAMPLES(5) | // Set samples to five.
BF_LRADC_CHn_VALUE(0) ) ); // Clear accumulator.
// ... setup Delay channel (see HW_LRADC_DELAY0 through 3)
while (HW_LRADC_CTRL1.LRADC0_IRQ != BV_LRADC_CTRL1_LRADC0_IRQ__PENDING)
{
// Wait for interrupt.
}
channelAverage = HW_LRADC_CHn(0).VALUE / 5;
```

33.4.7 LRADC 1 Result Register Description

The LRADC result register returns the 12-bit result for low resolution analog to digital converter channel one.

HW_LRADC_CH1	0x060
HW_LRADC_CH1_SET	0x064
HW_LRADC_CH1_CLR	0x068
HW_LRADC_CH1_TOG	0x06C

Table 33-13. HW_LRADC_CH1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TOGGLE	RSRVD2	ACCUMULATE	NUM_SAMPLES					RSRVD1								VALUE															

Table 33-14. HW_LRADC_CH1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**TOGGLE**	RW	0x0	This toggles at every completed conversion so software can detect a missed or duplicated sample.
30	**RSRVD2**	RO	0x0	Reserved
29	**ACCUMULATE**	RW	0x0	Set this bit to one to add successive samples to the 18 bit accumulator.
28:24	**NUM_SAMPLES**	RW	0x0	This bit field contains the number of conversion cycles to sum together before reporting operation complete interrupt status. Set this field to zero for a single conversion per interrupt.
23:18	**RSRVD1**	RO	0x000	Reserved
17:0	**VALUE**	RW	0x0000	This bit field contains the most recent 12-bit conversion value for this channel. If automatic oversampling is enbled this bit field contains the sum of the most recent N oversampled values, where N is set in the NUM_SAMPLES field for this channel. When 32 full-scale samples are added together, the 12-bit results can sum up to 256K. Software is responsible for dividing this value by the number of samples summed together. Software must clear this register in preparation for a multi-cycle accumulation.

DESCRIPTION:

The Result register contains the most recent conversion results for one channel of the LRADC. Note that each channel can be converted at an independent rate. The TOGGLE bit is used to debug missed conversion cycles. When using oversampling, the channel must be individualy scheduled for conversion N times for when N samples are required before an interrupt is generated. This is most easily accomplished by using one of the LRADC Delay Channels.

EXAMPLE:

```
if (HW_LRADC_CHn(1).TOGGLE == 1) { } // Toggle is high.
// ...
unsigned long channelAverage;
HW_LRADC_CHn_WR(1, (BF_LRADC_CHn_ACCUMULATE(1) | // Enable accumulation mode.
BF_LRADC_CHn_NUM_SAMPLES(5) | // Set samples to five.
BF_LRADC_CHn_VALUE(0) ) ); // Clear accumulator.
// ... setup Delay channel (see HW_LRADC_DELAY0 through 3)
while (HW_LRADC_CTRL1.LRADC1_IRQ != BV_LRADC_CTRL1_LRADC1_IRQ__PENDING)
{
// Wait for interrupt.
}
channelAverage = HW_LRADC_CHn(1).VALUE / 5;
```

33.4.8 LRADC 2 Result Register Description

The LRADC result register returns the 12-bit result for low resolution analog to digital converter channel two.

HW_LRADC_CH2	0x070
HW_LRADC_CH2_SET	0x074
HW_LRADC_CH2_CLR	0x078
HW_LRADC_CH2_TOG	0x07C

Table 33-15. HW_LRADC_CH2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TOGGLE	RSRVD2	ACCUMULATE	NUM_SAMPLES					RSRVD1								VALUE															

Table 33-16. HW_LRADC_CH2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**TOGGLE**	RW	0x0	This bit toggles at every completed conversion so software can detect a missed or duplicated sample.
30	**RSRVD2**	RO	0x0	Reserved
29	**ACCUMULATE**	RW	0x0	Set this bit to one to add successive samples to the 18 bit accumulator.
28:24	**NUM_SAMPLES**	RW	0x0	This bit field contains the number of conversion cycles to sum together before reporting operation complete interrupt status. Set this field to zero for a single conversion per interrupt.
23:18	**RSRVD1**	RO	0x000	Reserved
17:0	**VALUE**	RW	0x0000	This bit field contains the most recent 12-bit conversion value for this channel. If automatic oversampling is enbled this bit field contains the sum of the most recent N oversampled values, where N is set in the NUM_SAMPLES field for this channel. When 32 full-scale samples are added together, the 12-bit results can sum up to 256K. Software is responsible for dividing this value by the number of samples summed together. Software must clear this register in preparation for a multi-cycle accumulation.

DESCRIPTION:

The Result register contains the most recent conversion results for one channel of the LRADC. Note that each channel can be converted at an independent rate. The TOGGLE bit is used to debug missed conversion cycles. When using oversampling, the channel must be individualy scheduled for conversion N times for when N samples are required before an interrupt is generated. This is most easily accomplished by using one of the LRADC Delay Channels.

EXAMPLE:

```
if (HW_LRADC_CHn(2).TOGGLE == 1) { } // Toggle is high.
// ...
unsigned long channelAverage;
HW_LRADC_CHn_WR(2, (BF_LRADC_CHn_ACCUMULATE(1) | // Enable accumulation mode.
BF_LRADC_CHn_NUM_SAMPLES(5) | // Set samples to five.
BF_LRADC_CHn_VALUE(0) ) ); // Clear accumulator.
// ... setup Delay channel (see HW_LRADC_DELAY0 through 3)
while (HW_LRADC_CTRL1.LRADC2_IRQ != BV_LRADC_CTRL1_LRADC2_IRQ__PENDING)
{
// Wait for interrupt.
}
channelAverage = HW_LRADC_CHn(2).VALUE / 5;
```

33.4.9 LRADC 3 Result Register Description

The LRADC result register returns the 12-bit result for low resolution analog to digital converter channel three.

HW_LRADC_CH3	0x080
HW_LRADC_CH3_SET	0x084
HW_LRADC_CH3_CLR	0x088
HW_LRADC_CH3_TOG	0x08C

Table 33-17. HW_LRADC_CH3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TOGGLE	RSRVD2	ACCUMULATE	NUM_SAMPLES					RSRVD1								VALUE															

Table 33-18. HW_LRADC_CH3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**TOGGLE**	RW	0x0	This bit toggles at every completed conversion so software can detect a missed or duplicated sample.
30	**RSRVD2**	RO	0x0	Reserved
29	**ACCUMULATE**	RW	0x0	Set this bit to one to add successive samples to the 18 bit accumulator.
28:24	**NUM_SAMPLES**	RW	0x0	This bit field contains the number of conversion cycles to sum together before reporting operation complete interrupt status. Set this field to zero for a single conversion per interrupt.

Table 33-18. HW_LRADC_CH3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
23:18	**RSRVD1**	RO	0x000	Reserved
17:0	**VALUE**	RW	0x0000	This bit field contains the most recent 12-bit conversion value for this channel. If automatic oversampling is enbled this bit field contains the sum of the most recent N oversampled values, where N is set in the NUM_SAMPLES field for this channel. When 32 full-scale samples are added together, the 12-bit results can sum up to 256K. Software is responsible for dividing this value by the number of samples summed together. Software must clear this register in preparation for a multi-cycle accumulation.

DESCRIPTION:

The Result register contains the most recent conversion results for one channel of the LRADC. Note that each channel can be converted at an independent rate. The TOGGLE bit is used to debug missed conversion cycles. When using oversampling, the channel must be individualy scheduled for conversion N times for when N samples are required before an interrupt is generated. This is most easily accomplished by using one of the LRADC Delay Channels.

EXAMPLE:

```
if (HW_LRADC_CHn(3).TOGGLE == 1) { } // Toggle is high.
// ...
unsigned long channelAverage;
HW_LRADC_CHn_WR(3, (BF_LRADC_CHn_ACCUMULATE(1) | // Enable accumulation mode.
BF_LRADC_CHn_NUM_SAMPLES(5) | // Set samples to five.
BF_LRADC_CHn_VALUE(0) ) ); // Clear accumulator.
// ... setup Delay channel (see HW_LRADC_DELAY0 through 3)
while (HW_LRADC_CTRL1.LRADC3_IRQ != BV_LRADC_CTRL1_LRADC3_IRQ__PENDING)
{
// Wait for interrupt.
}
channelAverage = HW_LRADC_CHn(3).VALUE / 5;
```

33.4.10 LRADC 4 Result Register Description

The LRADC result register returns the 12-bit result for low resolution analog to digital converter channel four.

HW_LRADC_CH4	0x090
HW_LRADC_CH4_SET	0x094
HW_LRADC_CH4_CLR	0x098
HW_LRADC_CH4_TOG	0x09C

Table 33-19. HW_LRADC_CH4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TOGGLE	RSRVD2	ACCUMULATE	NUM_SAMPLES					RSRVD1								VALUE															

Table 33-20. HW_LRADC_CH4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**TOGGLE**	RW	0x0	This bit toggles at every completed conversion so software can detect a missed or duplicated sample.
30	**RSRVD2**	RO	0x0	Reserved
29	**ACCUMULATE**	RW	0x0	Set this bit to one to add successive samples to the 18 bit accumulator.
28:24	**NUM_SAMPLES**	RW	0x0	This bit field contains the number of conversion cycles to sum together before reporting operation complete interrupt status. Set this field to zero for a single conversion per interrupt.
23:18	**RSRVD1**	RO	0x000	Reserved
17:0	**VALUE**	RW	0x0000	This bit field contains the most recent 12-bit conversion value for this channel. If automatic oversampling is enbled this bit field contains the sum of the most recent N oversampled values, where N is set in the NUM_SAMPLES field for this channel. When 32 full-scale samples are added together, the 12-bit results can sum up to 256K. Software is responsible for dividing this value by the number of samples summed together. Software must clear this register in preparation for a multi-cycle accumulation.

DESCRIPTION:

The Result register contains the most recent conversion results for one channel of the LRADC. Note that each channel can be converted at an independent rate. The TOGGLE bit is used to debug missed conversion cycles. When using oversampling, the channel must be individualy scheduled for conversion N times for when N samples are required before an interrupt is generated. This is most easily accomplished by using one of the LRADC Delay Channels.

EXAMPLE:

```
if (HW_LRADC_CHn(4).TOGGLE == 1) { } // Toggle is high.
// ...
unsigned long channelAverage;
HW_LRADC_CHn_WR(4, (BF_LRADC_CHn_ACCUMULATE(1) | // Enable accumulation mode.
BF_LRADC_CHn_NUM_SAMPLES(5) | // Set samples to five.
BF_LRADC_CHn_VALUE(0) ) ); // Clear accumulator.
// ... setup Delay channel (see HW_LRADC_DELAY0 through 3)
while (HW_LRADC_CTRL1.LRADC4_IRQ != BV_LRADC_CTRL1_LRADC4_IRQ__PENDING)
{
// Wait for interrupt.
}
channelAverage = HW_LRADC_CHn(4).VALUE / 5;
```

33.4.11 LRADC 5 Result Register Description

The LRADC result register returns the 12-bit result for low resolution analog to digital converter channel five.

HW_LRADC_CH5	0x0A0
HW_LRADC_CH5_SET	0x0A4
HW_LRADC_CH5_CLR	0x0A8
HW_LRADC_CH5_TOG	0x0AC

Table 33-21. HW_LRADC_CH5

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TOGGLE	RSRVD2	ACCUMULATE	NUM_SAMPLES					RSRVD1								VALUE															

Table 33-22. HW_LRADC_CH5 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**TOGGLE**	RW	0x0	This bit toggles at every completed conversion so software can detect a missed or duplicated sample.
30	**RSRVD2**	RO	0x0	Reserved
29	**ACCUMULATE**	RW	0x0	Set this bit to one to add successive samples to the 18 bit accumulator.
28:24	**NUM_SAMPLES**	RW	0x0	This bit field contains the number of conversion cycles to sum together before reporting operation complete interrupt status. Set this field to zero for a single conversion per interrupt.
23:18	**RSRVD1**	RO	0x000	Reserved
17:0	**VALUE**	RW	0x0000	This bit field contains the most recent 12-bit conversion value for this channel. If automatic oversampling is enbled this bit field contains the sum of the most recent N oversampled values, where N is set in the NUM_SAMPLES field for this channel. When 32 full-scale samples are added together, the 12-bit results can sum up to 256K. Software is responsible for dividing this value by the number of samples summed together. Software must clear this register in preparation for a multi-cycle accumulation.

DESCRIPTION:

The Result register contains the most recent conversion results for one channel of the LRADC. Note that each channel can be converted at an independent rate. The TOGGLE bit is used to debug missed conversion cycles. When using oversampling, the channel must be individualy scheduled for conversion N times for when N samples are required before an interrupt is generated. This is most easily accomplished by using one of the LRADC Delay Channels.

EXAMPLE:

```
if (HW_LRADC_CHn(5).TOGGLE == 1) { } // Toggle is high.
// ...
unsigned long channelAverage;
HW_LRADC_CHn_WR(5, (BF_LRADC_CHn_ACCUMULATE(1) | // Enable accumulation mode.
BF_LRADC_CHn_NUM_SAMPLES(5) | // Set samples to five.
BF_LRADC_CHn_VALUE(0) ) ); // Clear accumulator.
// ... setup Delay channel (see HW_LRADC_DELAY0 through 3)
while (HW_LRADC_CTRL1.LRADC5_IRQ != BV_LRADC_CTRL1_LRADC5_IRQ__PENDING)
{
// Wait for interrupt.
}
channelAverage = HW_LRADC_CHn(5).VALUE / 5;
```

33.4.12 LRADC 6 (VddIO) Result Register Description

The LRADC result register returns the 12-bit result for low resolution analog to digital converter channel six (VddIO).

HW_LRADC_CH6	0x0B0
HW_LRADC_CH6_SET	0x0B4
HW_LRADC_CH6_CLR	0x0B8
HW_LRADC_CH6_TOG	0x0BC

Table 33-23. HW_LRADC_CH6

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TOGGLE	RSRVD2	ACCUMULATE	NUM_SAMPLES					RSRVD1								VALUE															

Table 33-24. HW_LRADC_CH6 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**TOGGLE**	RW	0x0	This bit toggles at every completed conversion so software can detect a missed or duplicated sample.
30	**RSRVD2**	RO	0x0	Reserved
29	**ACCUMULATE**	RW	0x0	Set this bit to one to add successive samples to the 18 bit accumulator.
28:24	**NUM_SAMPLES**	RW	0x0	This bit field contains the number of conversion cycles to sum together before reporting operation complete interrupt status. Set this field to zero for a single conversion per interrupt.

Table 33-24. HW_LRADC_CH6 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
23:18	**RSRVD1**	RO	0x000	Reserved
17:0	**VALUE**	RW	0x0000	This bit field contains the most recent 12-bit conversion value for this channel. If automatic oversampling is enbled this bit field contains the sum of the most recent N oversampled values, where N is set in the NUM_SAMPLES field for this channel. When 32 full-scale samples are added together, the 12-bit results can sum up to 256K. Software is responsible for dividing this value by the number of samples summed together. Software must clear this register in preparation for a multi-cycle accumulation.

DESCRIPTION:

The Result register contains the most recent conversion results for one channel of the LRADC. Note that each channel can be converted at an independent rate. The TOGGLE bit is used to debug missed conversion cycles. When using oversampling, the channel must be individualy scheduled for conversion N times for when N samples are required before an interrupt is generated. This is most easily accomplished by using one of the LRADC Delay Channels.

EXAMPLE:

```
if (HW_LRADC_CHn(6).TOGGLE == 1) { } // Toggle is high.
// ...
unsigned long channelAverage;
HW_LRADC_CHn_WR(6, (BF_LRADC_CHn_ACCUMULATE(1) | // Enable accumulation mode.
BF_LRADC_CHn_NUM_SAMPLES(5) | // Set samples to five.
BF_LRADC_CHn_VALUE(0) ) ); // Clear accumulator.
// ... setup Delay channel (see HW_LRADC_DELAY0 through 3)
while (HW_LRADC_CTRL1.LRADC6_IRQ != BV_LRADC_CTRL1_LRADC6_IRQ__PENDING)
{
// Wait for interrupt.
}
channelAverage = HW_LRADC_CHn(6).VALUE / 5;
```

33.4.13 LRADC 7 (BATT) Result Register Description

The LRADC result register returns the 12-bit result for low resolution analog to digital converter channel seven (BATT).

HW_LRADC_CH7	0x0C0
HW_LRADC_CH7_SET	0x0C4
HW_LRADC_CH7_CLR	0x0C8
HW_LRADC_CH7_TOG	0x0CC

Table 33-25. HW_LRADC_CH7

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TOGGLE	TESTMODE_TOGGLE	ACCUMULATE	NUM_SAMPLES					RSRVD1								VALUE															

Table 33-26. HW_LRADC_CH7 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**TOGGLE**	RW	0x0	This bit toggles at every completed conversion so software can detect a missed or duplicated sample.
30	**TESTMODE_TOGGLE**	RO	0x0	This read-only bit toggles at every completed conversion of interest in test mode so software can synchornize to the desired sample. When the test mode count is loaded with a value of 7, this will toggle every eighth conversion on channel 7. If testmode operation for channel 5 and or 6 are set then the sample rate will be lower for channel 7.
29	**ACCUMULATE**	RW	0x0	Set this bit to one to add successive samples to the 18 bit accumulator.
28:24	**NUM_SAMPLES**	RW	0x0	This bit field contains the number of conversion cycles to sum together before reporting operation complete interrupt status. Set this field to zero for a single conversion per interrupt.
23:18	**RSRVD1**	RO	0x000	Reserved
17:0	**VALUE**	RW	0x0000	This bit field contains the most recent 12-bit conversion value for this channel. If automatic oversampling is enbled this bit field contains the sum of the most recent N oversampled values, where N is set in the NUM_SAMPLES field for this channel. When 32 full-scale samples are added together, the 12-bit results can sum up to 256K. Software is responsible for dividing this value by the number of samples summed together. Software must clear this register in preparation for a multi-cycle accumulation.

DESCRIPTION:

The Result register contains the most recent conversion results for one channel of the LRADC. Note that each channel can be converted at an independent rate. The TOGGLE bit is used to debug missed conversion cycles. When using oversampling, the channel must be individualy scheduled for conversion N times for when N samples are required before an interrupt is generated. This is most easily accomplished by using one of the LRADC Delay Channels.

EXAMPLE:

```
if (HW_LRADC_CHn(7).TOGGLE == 1) { } // Toggle is high.
// ...
unsigned long channelAverage;
HW_LRADC_CHn_WR(7, (BF_LRADC_CHn_ACCUMULATE(1) | // Enable accumulation mode.
BF_LRADC_CHn_NUM_SAMPLES(5) | // Set samples to five.
BF_LRADC_CHn_VALUE(0) ) ); // Clear accumulator.
// ... setup Delay channel (see HW_LRADC_DELAY0 through 3)
while (HW_LRADC_CTRL1.LRADC7_IRQ != BV_LRADC_CTRL1_LRADC7_IRQ__PENDING)
{
// Wait for interrupt.
}
channelAverage = HW_LRADC_CHn(7).VALUE / 5;
```

33.4.14 LRADC Scheduling Delay 0 Description

The LRADC scheduling delay 0 register controls one delay operation. At the end of this delay, this channel can trigger one or more LRADC channels or one or more Scheduling delay channels .

HW_LRADC_DELAY0	0x0D0
HW_LRADC_DELAY0_SET	0x0D4
HW_LRADC_DELAY0_CLR	0x0D8
HW_LRADC_DELAY0_TOG	0x0DC

Table 33-27. HW_LRADC_DELAY0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TRIGGER_LRADCS								RSRVD2			KICK	TRIGGER_DELAYS				LOOP_COUNT					DELAY										

Table 33-28. HW_LRADC_DELAY0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**TRIGGER_LRADCS**	RW	0x00	Setting a bit in this bit field to one causes the delay controller to trigger the corresponding LRADC channel. This trigger occurs when the delay count of this delay channel reaches zero. Note that all eight LRADC channels can be triggered at the same time. Any channel with its corresponding bit set in this field is triggered. The HW accomplishes this by setting the corresponding bit(s) in HW_LRADC_CTRL0_SCHEDULE.
23:21	**RSRVD2**	RO	0x0	Reserved
20	**KICK**	RW	0x0	Setting this bit to one initiates a delay cycle. At the end of that cycle, any TRIGGER_LRADCS or TRIGGER_DELAYS will start.

Table 33-28. HW_LRADC_DELAY0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:16	**TRIGGER_DELAYS**	RW	0x0	Setting a bit in this bit field to one causes the delay controller to trigger the corresponding delay channel. This trigger occurs when the delay count of this delay channel reaches zero. Note that all four delay channels can be triggered at the same time, including the one that issues the trigger. This can have the effect of automatically retriggering a delay channel.
15:11	**LOOP_COUNT**	RW	0x00	This bit field specifies the number of times this delay counter will count down and then trigger its designated targets. This is particularly useful for scheduling multiple samples of an LRADC channel set. If this field is set to 0x0, then exactly one delay loop will be generated with exactly one event triggering the target LRADC and/or delay channels. Note setting the loop count to 0x01 will yield two conversions.
10:0	**DELAY**	RW	0x000	This 11-bit field counts down to zero. At zero it triggers either a set of LRADC channel conversions or another delay channel, or both. It can trigger up to all eight LRADCs and all four delay channels in a single even. This counter operates on a 2KHz clock derived from crystal clock.

DESCRIPTION:

The LRADC Delay Channel provides control by which LRADC channels and delay channels (including itself) may be triggered. The triggering of the selected delay and LRADC channel(s) is delayed by the DELAY field value which counts down on a 2 kHz clock. It is possible to use delay channels chained together to configure dependent timing of channel conversions as in the example provided in introduction to this block. A delay channel may also be configured to trigger itself. In this case, it could be used to simultaneously trigger an LRADC channel, providing continuous acquisitions of the conversions executed, delayed by the value specified in the DELAY field. The delay channel is started by setting the KICK bit to one.

EXAMPLE:

```
HW_LRADC_DELAYn_WR(0, (BF_LRADC_DELAYn_TRIGGER_LRADCS(0x05) | // LRADC channel 0 and 2
BF_LRADC_DELAYn_KICK(1) | // Start the Delay channel
BF_LRADC_DELAYn_TRIGGER_DELAYS(0x1) | // restart delay channel 0 each time
BF_LRADC_DELAYn_DELAY(0x0E45) ) ); // delay 3653 periods of 2 kHz clock
// ... do other things until the triggered LRADC channels report an interrupt.
```

33.4.15 LRADC Scheduling Delay 1 Description

The LRADC scheduling delay 1 register controls one delay operation. At the end of this delay, this channel can trigger one or more LRADC channels or one or more Scheduling delay channels .

HW_LRADC_DELAY1	0x0E0
HW_LRADC_DELAY1_SET	0x0E4
HW_LRADC_DELAY1_CLR	0x0E8
HW_LRADC_DELAY1_TOG	0x0EC

Table 33-29. HW_LRADC_DELAY1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TRIGGER_LRADCS								RSRVD2			KICK	TRIGGER_DELAYS				LOOP_COUNT					DELAY										

Table 33-30. HW_LRADC_DELAY1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**TRIGGER_LRADCS**	RW	0x00	Setting a bit in this bit field to one causes the delay controller to trigger the corresponding LRADC channel. This trigger occurs when the delay count of this delay channel reaches zero. Note that all eight LRADC channels can be triggered at the same time. Any channel with its corresponding bit set in this field is triggered. The HW accomplishes this by setting the corresponding bit(s) in HW_LRADC_CTRL0_SCHEDULE.
23:21	**RSRVD2**	RO	0x0	Reserved
20	**KICK**	RW	0x0	Setting this bit to one initiates a delay cycle. At the end of that cycle, any TRIGGER_LRADCS or TRIGGER_DELAYS will start.
19:16	**TRIGGER_DELAYS**	RW	0x0	Setting a bit in this bit field to one causes the delay controller to trigger the corresponding delay channel. This trigger occurs when the delay count of this delay channel reaches zero. Note that all four delay channels can be triggered at the same time, including the one that issues the trigger. This can have the effect of automatically retriggering a delay channel.
15:11	**LOOP_COUNT**	RW	0x00	This bit field specifies the number of times this delay counter will count down and then trigger its designated targets. This is particularly useful for scheduling multiple samples of an LRADC channel set. If this field is set to 0x0, then exactly one delay loop will be generated with exactly one event triggering the target LRADC and/or delay channels.
10:0	**DELAY**	RW	0x000	This 11-bit field counts down to zero. At zero it triggers either a set of LRADC channel conversions or another delay channel, or both. It can trigger up to all eight LRADCs and all four delay channels in a single even. This counter operates on a 2KHz clock derived from crystal clock.

DESCRIPTION:

The LRADC Delay Channel provides control by which LRADC channels and delay channels (including itself) may be triggered. The triggering of the selected delay and LRADC channel(s) is delayed by the DELAY field value which counts down on a 2 kHz clock. It is possible to use delay channels chained

together to configure dependent timing of channel conversions as in the example provided in introduction to this block. A delay channel may also be configured to trigger itself. In this case, it could be used to simultaneously trigger an LRADC channel, providing continuous acquisitions of the conversions executed, delayed by the value specified in the DELAY field. The delay channel is started by setting the KICK bit to one.

EXAMPLE:

```
HW_LRADC_DELAYn_WR(1, (BF_LRADC_DELAYn_TRIGGER_LRADCS(0x05) | // LRADC channel 0 and 2
BF_LRADC_DELAYn_KICK(1) | // Start the Delay channel
BF_LRADC_DELAYn_TRIGGER_DELAYS(0x2) | // restart delay channel 1 each time
BF_LRADC_DELAYn_DELAY(0x0E45) ) ); // delay 3653 periods of 2 kHz clock
// ... do other things until the triggered LRADC channels report an interrupt.
```

33.4.16 LRADC Scheduling Delay 2 Description

The LRADC scheduling delay 2 register controls one delay operation. At the end of this delay, this channel can trigger one or more LRADC channels or one or more Scheduling delay channels .

HW_LRADC_DELAY2	0x0F0
HW_LRADC_DELAY2_SET	0x0F4
HW_LRADC_DELAY2_CLR	0x0F8
HW_LRADC_DELAY2_TOG	0x0FC

Table 33-31. HW_LRADC_DELAY2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TRIGGER_LRADCS								RSRVD2			KICK	TRIGGER_DELAYS				LOOP_COUNT				DELAY											

Table 33-32. HW_LRADC_DELAY2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**TRIGGER_LRADCS**	RW	0x00	Setting a bit in this bit field to one causes the delay controller to trigger the corresponding LRADC channel. This trigger occurs when the delay count of this delay channel reaches zero. Note that all eight LRADC channels can be triggered at the same time. Any channel with its corresponding bit set in this field is triggered. The HW accomplishes this by setting the corresponding bit(s) in HW_LRADC_CTRL0_SCHEDULE.
23:21	**RSRVD2**	RO	0x0	Reserved
20	**KICK**	RW	0x0	Setting this bit to one initiates a delay cycle. At the end of that cycle, any TRIGGER_LRADCS or TRIGGER_DELAYS will start.

Table 33-32. HW_LRADC_DELAY2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:16	**TRIGGER_DELAYS**	RW	0x0	Setting a bit in this bit field to one causes the delay controller to trigger the corresponding delay channel. This trigger occurs when the delay count of this delay channel reaches zero. Note that all four delay channels can be triggered at the same time, including the one that issues the trigger. This can have the effect of automatically retriggering a delay channel.
15:11	**LOOP_COUNT**	RW	0x00	This bit field specifies the number of times this delay counter will count down and then trigger its designated targets. This is particularly useful for scheduling multiple samples of an LRADC channel set. If this field is set to 0x0, then exactly one delay loop will be generated with exactly one event triggering the target LRADC and/or delay channels. ERRATA: TA1 and TA2 silicon revisions do not correctly support the LOOP_COUNT field, do not use.
10:0	**DELAY**	RW	0x000	This 11-bit field counts down to zero. At zero it triggers either a set of LRADC channel conversions or another delay channel, or both. It can trigger up to all eight LRADCs and all four delay channels in a single even. This counter operates on a 2KHz clock derived from crystal clock.

DESCRIPTION:

The LRADC Delay Channel provides control by which LRADC channels and delay channels (including itself) may be triggered. The triggering of the selected delay and LRADC channel(s) is delayed by the DELAY field value which counts down on a 2 kHz clock. It is possible to use delay channels chained together to configure dependent timing of channel conversions as in the example provided in introduction to this block. A delay channel may also be configured to trigger itself. In this case, it could be used to simultaneously trigger an LRADC channel, providing continuous acquisitions of the conversions executed, delayed by the value specified in the DELAY field. The delay channel is started by setting the KICK bit to one.

EXAMPLE:

```
HW_LRADC_DELAYn_WR(2, (BF_LRADC_DELAYn_TRIGGER_LRADCS(0x05) | // LRADC channel 0 and 2
BF_LRADC_DELAYn_KICK(1) | // Start the Delay channel
BF_LRADC_DELAYn_TRIGGER_DELAYS(0x4) | // restart delay channel 2 each time
BF_LRADC_DELAYn_DELAY(0x0E45) ) ); // delay 3653 periods of 2 kHz clock
// ... do other things until the triggered LRADC channels report an interrupt.
```

33.4.17 LRADC Scheduling Delay 3 Description

The LRADC scheduling delay 3 register controls one delay operation. At the end of this delay, this channel can trigger one or more LRADC channels or one or more Scheduling delay channels .

HW_LRADC_DELAY3	0x100
HW_LRADC_DELAY3_SET	0x104
HW_LRADC_DELAY3_CLR	0x108
HW_LRADC_DELAY3_TOG	0x10C

Table 33-33. HW_LRADC_DELAY3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
TRIGGER_LRADCS								RSRVD2			KICK	TRIGGER_DELAYS				LOOP_COUNT					DELAY										

Table 33-34. HW_LRADC_DELAY3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**TRIGGER_LRADCS**	RW	0x00	Setting a bit in this bit field to one causes the delay controller to trigger the corresponding LRADC channel. This trigger occurs when the delay count of this delay channel reaches zero. Note that all eight LRADC channels can be triggered at the same time. Any channel with its corresponding bit set in this field is triggered. The HW accomplishes this by setting the corresponding bit(s) in HW_LRADC_CTRL0_SCHEDULE.
23:21	**RSRVD2**	RO	0x0	Reserved
20	**KICK**	RW	0x0	Setting this bit to one initiates a delay cycle. At the end of that cycle, any TRIGGER_LRADCS or TRIGGER_DELAYS will start.
19:16	**TRIGGER_DELAYS**	RW	0x0	Setting a bit in this bit field to one causes the delay controller to trigger the corresponding delay channel. This trigger occurs when the delay count of this delay channel reaches zero. Note that all four delay channels can be triggered at the same time, including the one that issues the trigger. This can have the effect of automatically retriggering a delay channel.
15:11	**LOOP_COUNT**	RW	0x00	This bit field specifies the number of times this delay counter will count down and then trigger its designated targets. This is particularly useful for scheduling multiple samples of an LRADC channel set. If this field is set to 0x0, then exactly one delay loop will be generated with exactly one event triggering the target LRADC and/or delay channels. ERRATA: TA1 and TA2 silicon revisions do not correctly support the LOOP_COUNT field, do not use.
10:0	**DELAY**	RW	0x000	This 11-bit field counts down to zero. At zero it triggers either a set of LRADC channel conversions or another delay channel, or both. It can trigger up to all eight LRADCs and all four delay channels in a single even. This counter operates on a 2KHz clock derived from crystal clock.

DESCRIPTION:

The LRADC Delay Channel provides control by which LRADC channels and delay channels (including itself) may be triggered. The triggering of the selected delay and LRADC channel(s) is delayed by the DELAY field value which counts down on a 2 kHz clock. It is possible to use delay channels chained together to configure dependent timing of channel conversions as in the example provided in introduction to this block. A delay channel may also be configured to trigger itself. In this case, it could be used to simultaneously trigger an LRADC channel, providing continuous acquisitions of the conversions executed, delayed by the value specified in the DELAY field. The delay channel is started by setting the KICK bit to one.

EXAMPLE:

```
HW_LRADC_DELAYn_WR(3, (BF_LRADC_DELAYn_TRIGGER_LRADCS(0x05) | // LRADC channel 0 and 2
BF_LRADC_DELAYn_KICK(1) | // Start the Delay channel
BF_LRADC_DELAYn_TRIGGER_DELAYS(0x8) | // restart delay channel 3 each time
BF_LRADC_DELAYn_DELAY(0x0E45) ) ); // delay 3653 periods of 2 kHz clock
// ... do other things until the triggered LRADC channels report an interrupt.
```

33.4.18 LRADC Debug Register 0 Description

The LRADC debug register provides read-only access to various internal states and other debug information.

HW_LRADC_DEBUG0	0x110
HW_LRADC_DEBUG0_SET	0x114
HW_LRADC_DEBUG0_CLR	0x118
HW_LRADC_DEBUG0_TOG	0x11C

Table 33-35. HW_LRADC_DEBUG0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
READONLY																RSRVD1				STATE											

Table 33-36. HW_LRADC_DEBUG0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:16	**READONLY**	RO	0x4321	LRADC internal state machine current state.
15:12	**RSRVD1**	RO	0x0	Reserved
11:0	**STATE**	RO	0x0	LRADC internal state machine current state.

DESCRIPTION:

The LRADC debug register contains read-only diagnostic information regarding the internal state machine. This only used in debugging.

EXAMPLE:

```
if (HW_LRADC_DEBUG0.STATE == 0X33) {} // some action based on this state.
```

33.4.19 LRADC Debug Register 1 Description

The LRADC debug register provides read-only access to various internal states and other debug information.

HW_LRADC_DEBUG1	0x120
HW_LRADC_DEBUG1_SET	0x124
HW_LRADC_DEBUG1_CLR	0x128
HW_LRADC_DEBUG1_TOG	0x12C

Table 33-37. HW_LRADC_DEBUG1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD3								REQUEST								RSRVD2			TESTMODE_COUNT					RSRVD1					TESTMODE6	TESTMODE5	TESTMODE

Table 33-38. HW_LRADC_DEBUG1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSRVD3**	RO	0x0	Reserved
23:16	**REQUEST**	RO	0x0	LRADC internal request register.
15:13	**RSRVD2**	RO	0x0	Reserved
12:8	**TESTMODE_COUNT**	RW	0x0	When in test mode, the value in this register will be loaded in to a counter which is decremented upon each Channel 7 conversion. When that counter decrements to zero, the HW_LRADC_CH7.TESTMODE_TOGGLE field will be toggled, indicating that the conversion value of interest is available in the HW_LRADC_CH7.VALUE bit field.
7:3	**RSRVD1**	RO	0x0	Reserved
2	**TESTMODE6**	RW	0x0	Force dummy conversion cycles on channel 6 during test mode. NORMAL = 0x0 Normal operation. TEST = 0x1 Put it in test mode, i.e. continuously sample channel 6.
1	**TESTMODE5**	RW	0x0	Force dummy conversion cycles on channel 5 during test mode. NORMAL = 0x0 Normal operation. TEST = 0x1 Put it in test mode, i.e. continuously sample channel 5.
0	**TESTMODE**	RW	0x0	Place the LRADC in a special test mode in which the analog section is free-running at its clock rate. LRADC_CH7 result is continuously updated every N conversions from the analog source selected in CTRL2, where N is determined by TESTMODE_COUNT. NORMAL = 0x0 Normal operation. TEST = 0x1 Put it in test mode, i.e. continuously sample channel 7.

DESCRIPTION:

The LRADC DEBUG1 register provides, read-only diagnostic information and control over the test modes of LRADC channels 5, 6 and 7. This is only used in debugging the LRADC.

EXAMPLE:

```
BW_LRADC_DEBUG1_TESTMODE(BV_LRADC_DEBUG1_TESTMODE__TEST);
```

33.4.20 LRADC Battery Conversion Register Description

The LRADC battery conversion register provides access to the battery voltage scale multiplier.

HW_LRADC_CONVERSION	0x130
HW_LRADC_CONVERSION_SET	0x134
HW_LRADC_CONVERSION_CLR	0x138
HW_LRADC_CONVERSION_TOG	0x13C

Table 33-39. HW_LRADC_CONVERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD3											AUTOMATIC	RSRVD2		SCALE_FACTOR		RSRVD1					SCALED_BATT_VOLTAGE										

Table 33-40. HW_LRADC_CONVERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:21	**RSRVD3**	RO	0x0	Reserved
20	**AUTOMATIC**	RW	0x0	Control the automatic update mode of the BATT_VAL bit field in the HW_POWER_BATTMONITOR register. DISABLE = 0x0 No automatic update of the scaled value. ENABLE = 0x1 Automatically compute the scaled battery voltage each time an LRADC Channel 7 (BATT) conversion takes place.
19:18	**RSRVD2**	RO	0x0	Reserved
17:16	**SCALE_FACTOR**	RW	0x0	Scale factors of 29/512, 29/256 or 29/128 are selected here. NIMH = 0x0 Single NiMH Battery operation, 29/512. DUAL_NIMH = 0x1 Two NiMH Battery operation, 29/256. LI_ION = 0x2 Lithium Ion Battery operation, 29/128. ALT_LI_ION = 0x3 Lithium Ion Battery operation, 29/128.
15:10	**RSRVD1**	RO	0x0	Reserved
9:0	**SCALED_BATT_VOLTAGE**	RW	0x80	LRADC Battery Voltage Divided by approximately 17.708. The actual scale factor is (battery voltage) times 29 divided by 512, 256 or 128.

DESCRIPTION:

This register controls the voltage scaling multiplier which is used to multiply the LRADC battery voltage by 29 divided by 512 for NiMH, battery voltage times 29 divided by 256 for dual NiMH and battery voltage times 29 divided by 128 for Lithium Ion batteries.

EXAMPLE:

```
HW_LRADC_CONVERSION.AUTOMATIC = 1;
```

33.4.21 LRADC Control Register 4 Description

LRADC control register 4 specifies the analog mux selector values used for channels 0 through channel 7.

HW_LRADC_CTRL4	0x140
HW_LRADC_CTRL4_SET	0x144
HW_LRADC_CTRL4_CLR	0x148
HW_LRADC_CTRL4_TOG	0x14C

Table 33-41. HW_LRADC_CTRL4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
LRADC7SELECT				LRADC6SELECT				LRADC5SELECT				LRADC4SELECT				LRADC3SELECT				LRADC2SELECT				LRADC1SELECT				LRADC0SELECT			

Table 33-42. HW_LRADC_CTRL4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**LRADC7SELECT**	RW	0x7	This bit field selects which analog mux input is used for conversion on LRADC channel 7. CHANNEL0 = 0x0 CHANNEL1 = 0x1 CHANNEL2 = 0x2 CHANNEL3 = 0x3 CHANNEL4 = 0x4 CHANNEL5 = 0x5 CHANNEL6 = 0x6 VDDIO CHANNEL7 = 0x7 BATTERY CHANNEL8 = 0x8 PMOS THIN CHANNEL9 = 0x9 NMOS THIN CHANNEL10 = 0xA NMOS THICK CHANNEL11 = 0xB PMOS THICK CHANNEL12 = 0xC USB_DP (must also set usb_data_on_lradc to use this) CHANNEL13 = 0xD USB_DN (must also set usb_data_on_lradc to use this) CHANNEL14 = 0xE VBG (Can be used to calibrate the LRADC) CHANNEL15 = 0xF 5V input
27:24	**LRADC6SELECT**	RW	0x6	This bit field selects which analog mux input is used for conversion on LRADC channel 6. CHANNEL0 = 0x0 CHANNEL1 = 0x1 CHANNEL2 = 0x2 CHANNEL3 = 0x3 CHANNEL4 = 0x4 CHANNEL5 = 0x5 CHANNEL6 = 0x6 VDDIO CHANNEL7 = 0x7 BATTERY CHANNEL8 = 0x8 PMOS THIN CHANNEL9 = 0x9 NMOS THIN CHANNEL10 = 0xA NMOS THICK CHANNEL11 = 0xB PMOS THICK CHANNEL12 = 0xC USB_DP (must also set usb_data_on_lradc to use this) CHANNEL13 = 0xD USB_DN (must also set usb_data_on_lradc to use this) CHANNEL14 = 0xE VBG (Can be used to calibrate the LRADC) CHANNEL15 = 0xF 5V input
23:20	**LRADC5SELECT**	RW	0x5	This bit field selects which analog mux input is used for conversion on LRADC channel 5. CHANNEL0 = 0x0 CHANNEL1 = 0x1 CHANNEL2 = 0x2 CHANNEL3 = 0x3 CHANNEL4 = 0x4 CHANNEL5 = 0x5 CHANNEL6 = 0x6 VDDIO CHANNEL7 = 0x7 BATTERY CHANNEL8 = 0x8 PMOS THIN CHANNEL9 = 0x9 NMOS THIN CHANNEL10 = 0xA NMOS THICK CHANNEL11 = 0xB PMOS THICK CHANNEL12 = 0xC USB_DP (must also set usb_data_on_lradc to use this) CHANNEL13 = 0xD USB_DN (must also set usb_data_on_lradc to use this) CHANNEL14 = 0xE VBG (Can be used to calibrate the LRADC) CHANNEL15 = 0xF 5V input

Table 33-42. HW_LRADC_CTRL4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:16	**LRADC4SELECT**	RW	0x4	This bit field selects which analog mux input is used for conversion on LRADC channel 4. CHANNEL0 = 0x0 CHANNEL1 = 0x1 CHANNEL2 = 0x2 CHANNEL3 = 0x3 CHANNEL4 = 0x4 CHANNEL5 = 0x5 CHANNEL6 = 0x6 VDDIO CHANNEL7 = 0x7 BATTERY CHANNEL8 = 0x8 PMOS THIN CHANNEL9 = 0x9 NMOS THIN CHANNEL10 = 0xA NMOS THICK CHANNEL11 = 0xB PMOS THICK CHANNEL12 = 0xC USB_DP (must also set usb_data_on_lradc to use this) CHANNEL13 = 0xD USB_DN (must also set usb_data_on_lradc to use this) CHANNEL14 = 0xE VBG (Can be used to calibrate the LRADC) CHANNEL15 = 0xF 5V input
15:12	**LRADC3SELECT**	RW	0x3	This bit field selects which analog mux input is used for conversion on LRADC channel 3. CHANNEL0 = 0x0 CHANNEL1 = 0x1 CHANNEL2 = 0x2 CHANNEL3 = 0x3 CHANNEL4 = 0x4 CHANNEL5 = 0x5 CHANNEL6 = 0x6 VDDIO CHANNEL7 = 0x7 BATTERY CHANNEL8 = 0x8 PMOS THIN CHANNEL9 = 0x9 NMOS THIN CHANNEL10 = 0xA NMOS THICK CHANNEL11 = 0xB PMOS THICK CHANNEL12 = 0xC USB_DP (must also set usb_data_on_lradc to use this) CHANNEL13 = 0xD USB_DN (must also set usb_data_on_lradc to use this) CHANNEL14 = 0xE VBG (Can be used to calibrate the LRADC) CHANNEL15 = 0xF 5V input
11:8	**LRADC2SELECT**	RW	0x2	This bit field selects which analog mux input is used for conversion on LRADC channel 2. CHANNEL0 = 0x0 CHANNEL1 = 0x1 CHANNEL2 = 0x2 CHANNEL3 = 0x3 CHANNEL4 = 0x4 CHANNEL5 = 0x5 CHANNEL6 = 0x6 VDDIO CHANNEL7 = 0x7 BATTERY CHANNEL8 = 0x8 PMOS THIN CHANNEL9 = 0x9 NMOS THIN CHANNEL10 = 0xA NMOS THICK CHANNEL11 = 0xB PMOS THICK CHANNEL12 = 0xC USB_DP (must also set usb_data_on_lradc to use this) CHANNEL13 = 0xD USB_DN (must also set usb_data_on_lradc to use this) CHANNEL14 = 0xE VBG (Can be used to calibrate the LRADC) CHANNEL15 = 0xF 5V input

Table 33-42. HW_LRADC_CTRL4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
7:4	**LRADC1SELECT**	RW	0x1	This bit field selects which analog mux input is used for conversion on LRADC channel 1. CHANNEL0 = 0x0 CHANNEL1 = 0x1 CHANNEL2 = 0x2 CHANNEL3 = 0x3 CHANNEL4 = 0x4 CHANNEL5 = 0x5 CHANNEL6 = 0x6 VDDIO CHANNEL7 = 0x7 BATTERY CHANNEL8 = 0x8 PMOS THIN CHANNEL9 = 0x9 NMOS THIN CHANNEL10 = 0xA NMOS THICK CHANNEL11 = 0xB PMOS THICK CHANNEL12 = 0xC USB_DP (must also set usb_data_on_lradc to use this) CHANNEL13 = 0xD USB_DN (must also set usb_data_on_lradc to use this) CHANNEL14 = 0xE VBG (Can be used to calibrate the LRADC) CHANNEL15 = 0xF 5V input
3:0	**LRADC0SELECT**	RW	0x0	This bit field selects which analog mux input is used for conversion on LRADC channel 0. CHANNEL0 = 0x0 CHANNEL1 = 0x1 CHANNEL2 = 0x2 CHANNEL3 = 0x3 CHANNEL4 = 0x4 CHANNEL5 = 0x5 CHANNEL6 = 0x6 VDDIO CHANNEL7 = 0x7 BATTERY CHANNEL8 = 0x8 PMOS THIN CHANNEL9 = 0x9 NMOS THIN CHANNEL10 = 0xA NMOS THICK CHANNEL11 = 0xB PMOS THICK CHANNEL12 = 0xC USB_DP (must also set usb_data_on_lradc to use this) CHANNEL13 = 0xD USB_DN (must also set usb_data_on_lradc to use this) CHANNEL14 = 0xE VBG (Can be used to calibrate the LRADC) CHANNEL15 = 0xF 5V input

DESCRIPTION:

Each virtual channel of the LRADC can be directed to use any of the 16 analog mux input sources for it conversion. This register specifies the analog mux to

channels to be used for LRADC virtual channels 0 through 7.

EXAMPLE:

```
BW_LRADC_CTRL4_LRADC3SELECT(BV_LRADC_CTRL4_LRADC3SELECT__CHANNEL11);
```

33.4.22 LRADC Version Register Description

This register always returns a known read value for debug purposes it indicates the version of the block.

HW_LRADC_VERSION 0x150

Table 33-43. HW_LRADC_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

Table 33-44. HW_LRADC_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x01	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x01	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
if (HW_ICOLL_VERSION.B.MAJOR != 1) Error();
```

LRADC Block v1.1, Revision 1.48

Chapter 34
Serial JTAG (SJTAG)

This chapter describes the one-wire serial JTAG (SJTAG) function included on the i.MX23 and how to use it. There are no registers in this module.

34.1 Overview

The i.MX23 provides a one-wire serial JTAG (SJTAG) interface to connect to various external JTAG debugger dongles through a Freescale-defined FPGA/CPLD. SJTAG supports the Green Hills Slingshot and ETM probe debugger dongles, as well as those made by ARM, Abatron, and Lauterbach.

The SJTAG block provides the following functions.

- Maps one-wire protocol to six-wire JTAG interface on the ARM926 core.
- Detects presence of external debugger when it is connected to the one-wire SJTAG pin (DEBUG) and it issues clock or JTAG reset commands.
- Signals JTAG presence to external FPGA/CPLD translator.
- Detects glitches and false JTAG clocks to improve noise immunity.

The one-wire SJTAG interface is illustrated in Figure 34-1.

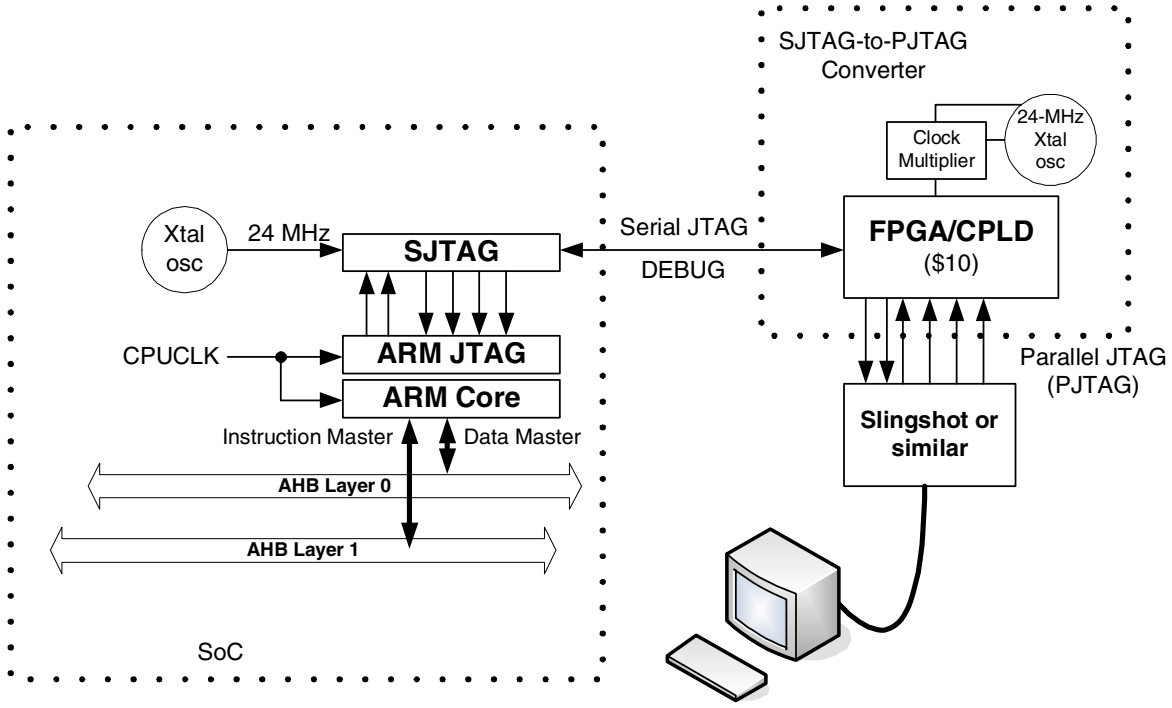


Figure 34-1. Serial JTAG (SJTAG) Block Diagram

34.2 Operation

The architecture of the one-wire serial JTAG interface depends on the FPGA/CPLD to always be present and for it to do all of the "heavy lifting" associated with synchronizing data back and forth across the interface. To that end, the FPGA/CPLD is programmed to use its digital clock modules to generate an 8x oversample clock to interface with the 24-MHz on-chip crystal oscillator circuits. Figure 34-2 defines the clock relationships.

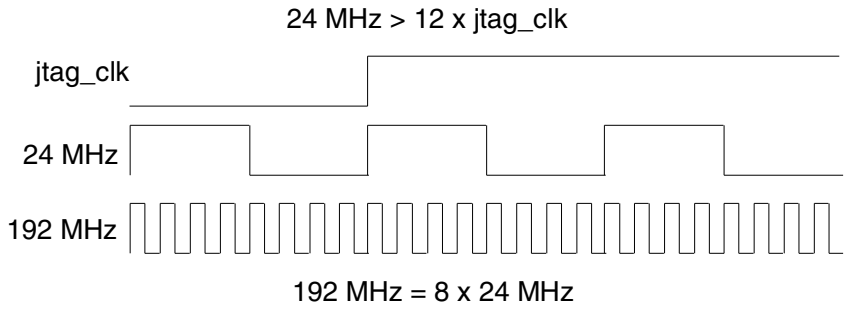


Figure 34-2. SJTAG Clock Relationships

The following high-level steps describe using the FPGA/CPLD and the debugger with the i.MX23 SJTAG interface:

- The FPGA/CPLD synchronizes the debugger's JTAG clock into its 192-MHz domain and edge-detects it.
- The rising edge of the debugger's clock triggers an asynchronous start phase in which the FPGA/CPLD drives the DEBUG pin high for one FPGA/CPLD clock period.
- The DEBUG pin is pulled lightly high until the SJTAG block on the chip recognizes the start condition on the wire.
- The timing mark phase is entered at that point, and the chip pulls the DEBUG line back low.
- This falling edge is detected by the oversample clock in the FPGA/CPLD, and it uses this timing mark to drive all subsequent transfers to the chip.
- The 8x oversample clock then works on pseudo-synchronous timing points derived by counting out the 8x clocks.

The following sections describe these phases illustrated in Figure 34-3 in more detail.

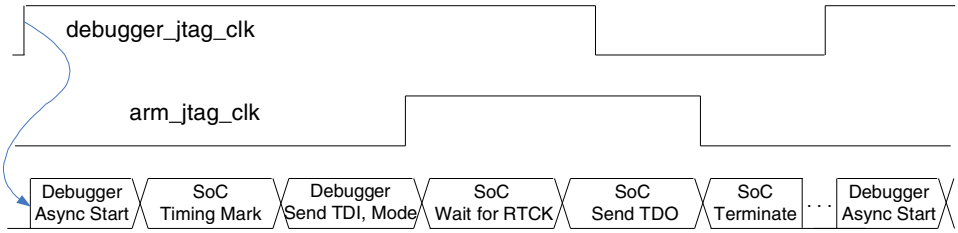


Figure 34-3. SJTAG Phases of Operation for One JTAG Clock

34.2.1 Debugger Async Start Phase

- This phase begins when a rising edge is detected on the JTAG clock signal coming from the debugger.
- This phase lasts a fixed number of 192-MHz clock periods. That is, the DEBUG pin is driven high for a period corresponding to half a 24-MHz clock, i.e., 20.8 ns. This drive is completely asynchronous to the on-chip 24-MHz clock, i.e., the relationship is unknown at this point.
- The soft pullup is turned on by the FPGA/CPLD and left on as the phase ends.
- This phase ends when the strong driver is turned off and the timing mark phase is entered.

34.2.2 i.MX23 Timing Mark Phase

- This phase is entered when the FPGA/CPLD releases its strong driver, i.e., when it three-states the driver.
- The SJTAG controller on the chip detects the rising edge on the DEBUG pin and synchronizes it.
- This action starts a shift register timing chain that runs through this and the next phase.

- When the synchronized edge is recognized by the on-chip SJTAG controller, it pulls the DEBUG line back down clock to Q to pad after the rising edge of its 24-MHz clock. This is the critical timing mark that is detected in the FPGA/CPLD and used to time data in next phase.
- The timing mark phase ends when the on-chip SJTAG stops driving the serial JTAG wire low for one cycle.

34.2.3 Debugger Send TDI, Mode Phase

- During the first 24-MHz clock period of this phase, the FPGA/CPLD sends a one clock-wide signal that either tells the on-chip SJTAG that it is present and a JTAG clock begins, or it tells the on-chip SJTAG to do a JTAG reset operation to the ARM JTAG TAP controller.
- If a noise glitch falsely triggered the ASYNC Start Phase, then the on-chip SJTAG will treat it as a TAP controller reset in most cases.
- If the debugger is performing a JTAG clock cycle operation then, it next sends the state of the debugger TDI and MODE pins sequentially on the wire, i.e., one in each of the following two 24-MHz clocks. Notice that for this phase, the FPGA/CPLD knows the correct timing to change these three data elements on the wire because of the timing information it learned from the Timing Mark Phase.
- This phase ends after the FPGA/CPLD drives the serial wire low on the fourth 24-MHz clock of this phase.

To review, the first data element sent is the signal that distinguishes clock cycles from TAP reset cycles. The next two bits sent are the JTAG MODE and TDI bits from the debugger. Finally, the line is driven low and pulled down for half a 24-MHz clock and the driver is turned off and the pulldown left on. This phase ends when the half-clock pulldown is complete. The rising edge of the JTAG clock is sent to the ARM TAP controller during this phase.

34.2.4 i.MX23 Wait For Return Clock Phase

During this phase:

- The on-chip SJTAG controller waits for the ARM TAP controller to send back the return clock. This is an asynchronous event for both the on-chip TAP controller and the FPGA/CPLD controller.
- The on-chip controller drives the serial wire high for one 24-MHz clock period to tell the FPGA/CPLD that the variable length wait for return clock period is complete.
- This phase ends when the on-chip SJTAG detects the return clock going high and drives the serial high for one 24-MHz clock.

34.2.5 i.MX23 Sends TDO and Return Clock Timing Phase

During this phase:

- The on-chip SJTAG controller sends the value of the ARM TAP controller's TDO signal on the wire for one full 24-MHz period that begins on the rising edge of the on-chip 24-MHz clock.
- This phase ends when the TDO value has been sent.

34.2.6 i.MX23 Terminate Phase

The primary purpose of this phase to leave the SJTAG serial wire in the low state.

- The on-chip SJTAG controller accomplishes this by driving it low for half a 24-MHz clock, releasing it at the falling edge of its internal 24-MHz clock.
- This allows the next JTAG cycle to be started by the FPGA/CPLD around ¾ of a 24-MHz clock after this phase is entered.
- When this phase ends, the on-chip SJTAG controller resets its "Active" flip-flop and returns to its idle state in both its timing chain and its state machine.

The on-chip SJTAG always drives out the serial wire at the rising edge of the 24-MHz clock. It may drive for one 24-MHz clock cycle (return clock and TDO) or half cycle (terminate phase).

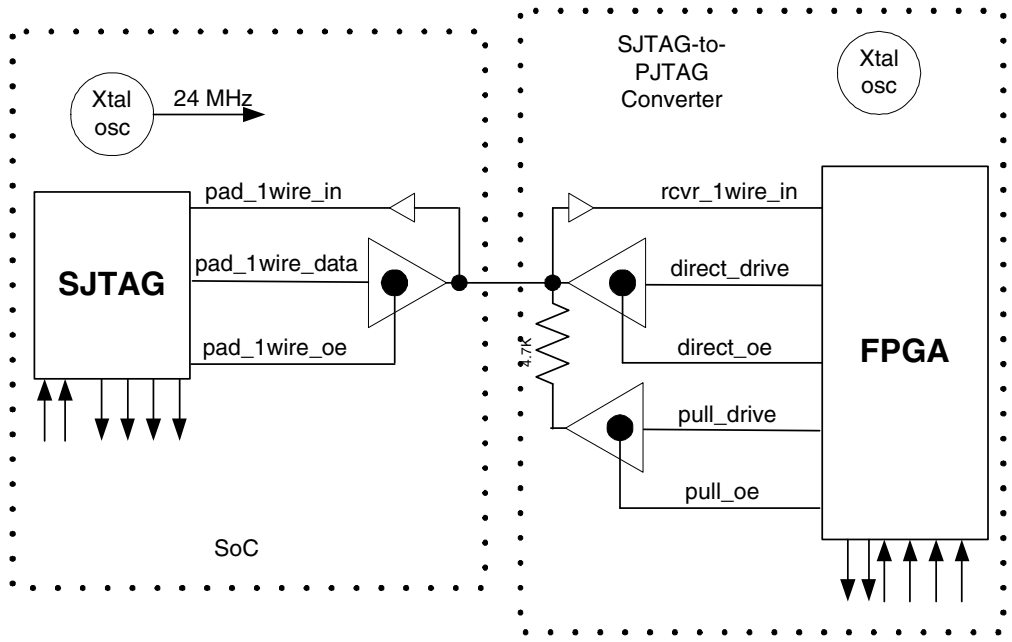


Figure 34-4. SJTAG Drivers

34.2.7 SJTAG External Pin

The SJTAG interface uses a single bidirectional interface pin (DEBUG) running at VDDIO to communicate with external debuggers. The DEBUG pad itself is wired as a conventional 8-mA 3.3-V driver. The DEBUG pin is completely dedicated to this one function. The signal on this interface is actively pulled up or down by the SOC as well as by the external debugger. However, the SOC will never drive this interface until it is first driven high by the external debugger..

The external JTAG debugger interface circuit includes a switched 4.7 Kohm pulldown resistor on its board. The DEBUG pin has a Schmitt trigger.

If the DEBUG pin is unused, Freescale recommends pulling the DEBUG pin to ground through a 100K resistor. It is also possible to short the DEBUG pin to ground directly, but doing this will prohibit debugging on a production player.

34.2.8 Selecting Serial JTAG or Six-Wire JTAG Mode

The HW_DIGCTL_CTRL_USE_SERIAL_JTAG bit in the digital control block selects whether the serial JTAG interface or the alternative six-wire parallel JTAG interface is used.

- When this bit is cleared to 0, parallel six-wire JTAG is enabled and is mapped to a collection of module pins that must be enabled by programming their PINMUXSEL bits in the PINCTRL block.
- When this bit is set to 1, serial JTAG is enabled and uses the dedicated DEBUG pin.

The ROM bootcode writes this field prior to enabling JTAG, selecting which type of JTAG pin signaling to use.

Chapter 35
Boot Modes

This chapter describes the boot modes implemented on the i.MX23.

35.1 Boot Modes

Table 35-1 lists all of the boot modes supported by the i.MX23 ROM. The boot mode can be selected either through external resistors or via OTP eFuse bit programming.

Table 35-1. ROM Supported Boot Modes

PORT	BOOT MODE
USB	Encrypted/unencrypted USB slave boot mode.
I^2C	Encrypted/unencrypted I^2C master—Boots from 3.3-V EEPROM.
SPI1	Encrypted/unencrypted SPI master from SSP1—Boots from 3.3-V flash memory.
SPI2	Encrypted/unencrypted SPI master from SSP2—Boots from 3.3-V flash and EEPROM.
SSP1	Encrypted/unencrypted SD/MMC master from SSP1—Boots from 3.3-V 1-bit, 4-bit, and 8-bit SD/MMC cards.
SSP2	Encrypted/unencrypted SD/MMC master from SSP2—Boots from 3.3-V 1-bit, 4-bit, and 8-bit SD/MMC cards.
GPMI	Encrypted/unencrypted NAND, 3.3-V, 8-bit wide and ECC4 and ECC8.
JTAG_WAIT	Unencrypted startup —Waits for JTAG debugger connection.

35.1.1 Boot Pins Definition and Mode Selection

Boot pins are located on LCD_RS, LCD_DATA[5] and LCD_DATA[3:0]. To enable boot mode selection from the LCD data pins, pull up LCD_RS. The ROM probes the LCD_RS pin and then, if valid, decodes the boot mode vector from the pins LCD_DATA[5] and LCD_DATA[3:0].

If LCD_RS is pulled down, then the boot mode is determined by OTP eFuse bits, as defined in Table 35-4.

Table 35-2 shows the boot pins.

Table 35-2. Boot Pins

PIN NAME	BOOT FUNCTION	BIT NAME
LCD_RS	Determines if there is a need to examine the other boot pins.	

Table 35-2. Boot Pins

PIN NAME	BOOT FUNCTION	BIT NAME
LCD_DATA[5]	ETM Enable	TBM1
LCD_DATA[3]	Boot Mode Bit 3	BM3
LCD_DATA[2]	Boot Mode Bit 2	BM2
LCD_DATA[1]	Boot Mode Bit 1	BM1
LCD_DATA[0]	Boot Mode Bit 0	BM0

These pads are powered during the initial startup sequence. The pads are enabled as GPIOs for sensing and then disabled. However, the pads remain powered. The TBMx pins are not powered or configured as GPIOs unless BM3:0=0xF. The ETM sets drive strength to 8mA on the ETM pins.

35.1.2 Boot Mode Selection Map

Table 35-3. Boot Mode Selection Map

ETM Enable/ LCD_ DATA[5]	BM3/ LCD_ DATA[3]	BM2/ LCD_ DATA[2]	BM1/ LCD_ DATA[1]	BM0/ LCD_ DATA[0]	PORT	BOOT MODE
0/1	0	0	0	0	USB	USB (unencrypted vs. encrypted is under OTP control)
0/1	0	0	0	1	I^2C	I^2C master
0/1	0	0	1	0	SPI	SPI master SSP1 boot from flash
0/1	0	0	1	1	SPI	SPI master SSP2 boot from flash
0/1	0	1	0	0	GPMI	NAND
0/1	0	1	0	1	–	Reserved
0/1	0	1	1	0	JTAG_WAIT	Startup waits for JTAG debugger connection
x	0	1	1	1	–	Reserved
0/1	1	0	0	0	SPI	SPI master SSP2 boot from EEPROM
0/1	1	0	0	1	SSP1	SD/MMC master on SSP1
0/1	1	0	1	0	SSP2	SD/MMC master on SSP2
x	1	0	1	1	–	Reserved
0/1	1	1	0	0	–	Reserved
0/1	1	1	0	1	–	Reserved
x	1	1	1	0	–	Reserved
x	1	1	1	1	–	Reserved

35.2 OTP eFuse and Persistent Bit Definitions

35.2.1 OTP eFuse

The i.MX23 contains a 1-Kbit array of OTP eFuse bits, some of which are used by the ROM. The bits listed in Table 35-4 through Table 35-6 can be configured by customers and are typically programmed on the customer's board assembly line. For more information about the OTP bits, see Chapter 7, "On-Chip OTP (OCOTP) Controller."

Table 35-4. General ROM Bits

eFuse Bank:Address:Bit	eFuse Function
HW_OCOTP_ROM0:0x8002C1A0:31:29	Undefined
HW_OCOTP_ROM0:0x8002C1A0:28	TBM0
HW_OCOTP_ROM0:0x8002C1A0:27	BM3
HW_OCOTP_ROM0:0x8002C1A0:26	BM2
HW_OCOTP_ROM0:0x8002C1A0:25	BM1
HW_OCOTP_ROM0:0x8002C1A0:24	BM0
HW_OCOTP_ROM0:0x8002C1A0:23	ENABLE_PJTAG_12MA_DRIVE - This bit is used to drive pins for 6-wire parallel JTAG at 12ma.
HW_OCOTP_ROM0:0x8002C1A0:22	USE_PARALLEL_JTAG - The default is serial jtag, this bit can be used to enable 6-wire parallel JTAG.
HW_OCOTP_ROM0:0x8002C1A0:21:20	POWER_GATE_GPIO-SD/MMC card power gate GPIO pin select. 00 = PWM0 01 = LCD_DOTCK 10 = PWM3 11 = NO_GATE=
HW_OCOTP_ROM0:0x8002C1A0:19:14	POWER_UP_DELAY—SD/MMC card power up delay required after enabling GPIO power gate. 000000 = 20 ms (default) 000001 = 10 ms 000010 = 20 ms …. 111111 = 630 ms
HW_OCOTP_ROM0:0x8002C1A0:13:12	SD_BUS_WIDTH—SD/MMC card bus width. 00 = 4-bit 01 = 1-bit 10 = 8-bit 11 = Reserved
HW_OCOTP_ROM0:0x8002C1A0:11:8	Index to SSP clock speed. By default (0x0), the clock speed is set to 12 MHz. The value of the index will modify the SPI clock speed accordingly.
HW_OCOTP_ROM0:0x8002C1A0:6	DISABLE_SPI_NOR_FAST_READ—Blow to disable SPI NOR fast reads, which are used by default. Some SPI NORs do not support this functionality.

Table 35-4. General ROM Bits (continued)

eFuse Bank:Address:Bit	eFuse Function
HW_OCOTP_ROM0:0x8002C1A0:5	ENABLE_USB_BOOT_SERIAL_NUMBER—If set, the device serial number is reported to the host during USB boot mode initialization, else no serial number is reported.
HW_OCOTP_ROM0:0x8002C1A0:4	ENABLE_UNENCRYPTED_BOOT—If clear, allows only booting of encrypted images. If set, both encrypted/unencrypted images are valid.
HW_OCOTP_ROM0:0x8002C1A0:3	SD_MBR_BOOT—Set to enable SD Master Boot Record (MBR) mode. The SD/MMC card should have a valid MBR to boot successfully in this mode. If this bit is not set, ROM will try to boot in default mode, BCB (Boot Control Block).
HW_OCOTP_ROM0:0x8002C1A0:2	DISABLE_RECOVERY_MODE—If set, does not allow booting in recovery mode.

Table 35-5. NAND/SD-MMC Related Bits

eFuse Bank:Address:Bit	eFuse Function
HW_OCOTP_ROM1:0x8002C1B0:29:28	USE_ALT_GPMI_RDY3 - These bits are used by ROM NAND driver to enable one of 3 alternate pins for GPMI_RDY3. 00-GPMI_RDY3 01-PWM2 10-LCD_DOTCK
HW_OCOTP_ROM1:0x8002C1B0:27:26	USE_ALT_GPMI_CE3 - These bits are used by ROM NAND driver to enable one of 4 alternate pins for GPMI_CE3. 00-GPMI_D15 01-LCD_RESET 10-SSP_DETECT 11-ROTARYB
HW_OCOTP_ROM1:0x8002C1B0:25	USE_ALT_GPMI_RDY2 - If the bit is blown then ROM NAND driver will enable alternate pins for GPMI_RDY2
HW_OCOTP_ROM1:0x8002C1B0:24	USE_ALT_GPMI_CE2 If the bit is blown then ROM NAND driver will enable alternate pins for GPMI_CE2
HW_OCOTP_ROM1:0x8002C1B0:23	ENABLE_NAND3_CE_RDY_PULLUP - If the bit is blown then ROM NAND driver will enable internal pull-up for GPMI_CE3 and GPMI_RDY3 pins.
HW_OCOTP_ROM1:0x8002C1B0:22	ENABLE_NAND2_CE_RDY_PULLUP - If the bit is blown then ROM NAND driver will enable internal pull-up for GPMI_CE2 and GPMI_RDY2 pins.
HW_OCOTP_ROM1:0x8002C1B0:21	ENABLE_NAND1_CE_RDY_PULLUP - If the bit is blown then ROM NAND driver will enable internal pull-up for GPMI_CE1 and GPMI_RDY1 pins.
HW_OCOTP_ROM1:0x8002C1B0:20	ENABLE_NAND0_CE_RDY_PULLUP - If the bit is blown then ROM NAND driver will enable internal pull-up for GPMI_CE0 and GPMI_RDY0 pins.
HW_OCOTP_ROM1:0x8002C1B0:19	UNTOUCH_INTERNAL_SSP_PULLUP - If this bit is blown then internal pull-ups for SSP are neither enabled nor disabled. This bit is used only if external pull-ups are implemented and ROM1:18 and/or ROM1:17 are blown.
HW_OCOTP_ROM1:0x8002C1B0:18	SSP2_EXT_PULLUP - Blow to indicate external pull-ups implemented for SSP2

Table 35-5. NAND/SD-MMC Related Bits (continued)

HW_OCOTP_ROM1:0x8002C1B0:17	SSP1_EXT_PULLUP - Blow to indicate external pull-ups implemented for SSP1
HW_OCOTP_ROM1:0x8002C1B0:12	USE_ALT_SSP1_DATA4-7 - This bit is blown to enable alternate pin use for SSP1 data lines 4-7
HW_OCOTP_ROM1:0x8002C1B0:11:8	BOOT_SEARCH_COUNT - Number of 64-page blocks skipped by the NAND driver when searching for information saved into the NAND (see Section 35.8, "NAND Boot Mode" for details). Default value of 0 means 4 blocks to skip.
HW_OCOTP_ROM1:0x8002C1B0:2:0	NUMBER_OF_NANDS - Indicates the number of external NANDs. A 0 value means that the NAND driver will scan the chip selects to dynamically find the correct number of NANDs.

Table 35-6. USB-Related Bits

eFuse Bank:Address:Bit	eFuse Function
HW_OCOTP_ROM2:0x8002C1C0:31:16	USB_VID—Vendor ID used in recovery mode. If the field is 0, Freescale vendor ID is used.
HW_OCOTP_ROM2:0x8002C1C0:15:0	USB_PID—Product ID used in recovery mode.

35.2.2 Persistent Bits

Persistent bits are used to control certain features in the ROM, as shown in Table 35-7. For more information on the persistent bits, see Chapter 23, "Real-Time Clock, Alarm, Watchdog, Persistent Bits."

Table 35-7. Persistent Bits

PERSISTENT BIT	FUNCTION
HW_RTC_PERSISTENT1:0x8005C070:3	SD_SPEED_ENABLE—If this bit is set, ROM will put the SD/MMC card in high-speed mode.
HW_RTC_PERSISTENT1:0x8005C070:2	NAND_SDK_BLOCK_REWRITE—The NAND driver sets this bit to indicate to the SDK that the boot image has ECC errors that reached the warning threshold. The SDK must regenerate the firmware by copying it from the backup image. The SDK will clear this bit.
HW_RTC_PERSISTENT1:0x8005C070:1	NAND_SECONDARY_BOOT—When this bit is set, the ROM attempts to boot from the secondary image if the boot driver supports it. This bit is set by the ROM boot driver and cleared by the SDK after repair.
HW_RTC_PERSISTENT1:0x8005C070:0	FORCE_RECOVERY—When this bit is set, the ROM code forces the system to boot in recovery mode, regardless of the selected mode. The ROM will clear the bit.

35.3 Memory Map

Figure 35-1 shows the memory map for the boot loader.

ROM boot code resides in the top 64K address space. The boot code uses the top 16K of OCRAM for data, and the 4K just below it should be reserved for patching the ROM. This leaves 16K of OCRAM for loading application data.

If a system uses external memory, then a boot image may be created that first loads a small SDRAM initialization program into OCRAM. The program will set up the SDRAM, and then the rest of the boot image may continue to load, overwriting the initialization program in OCRAM.

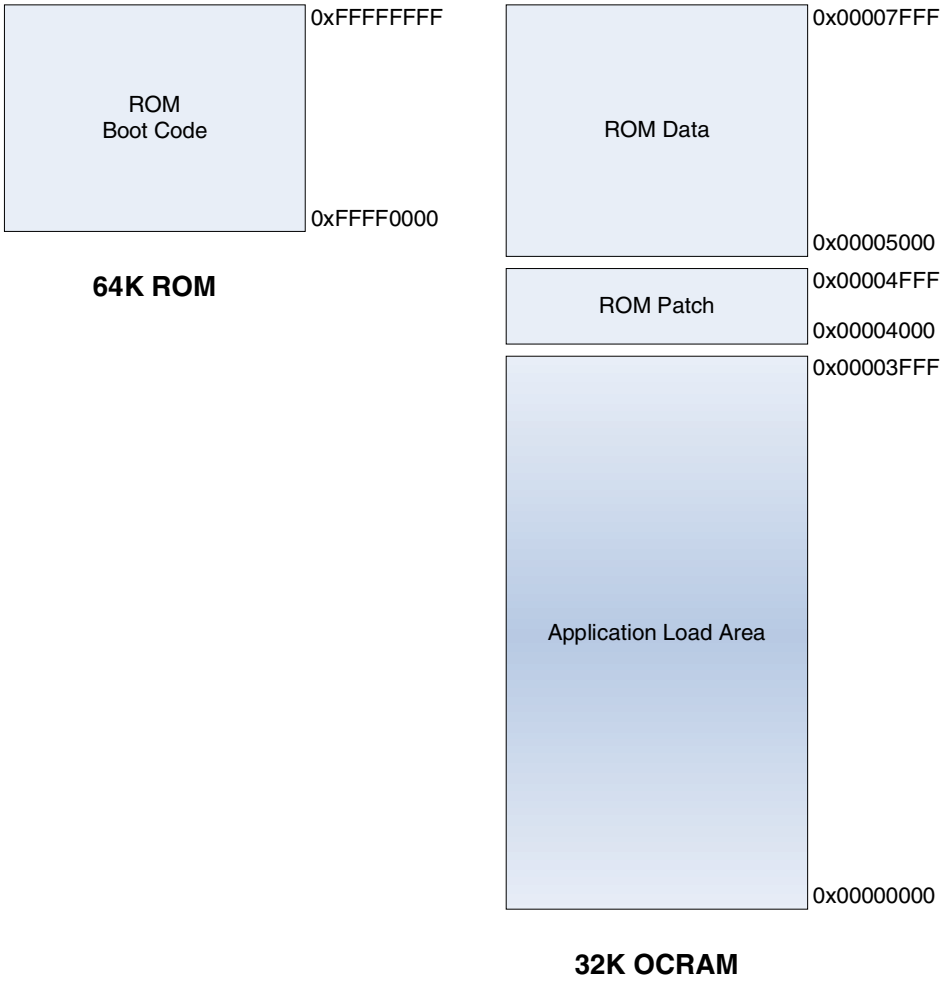


Figure 35-1. Boot Loader Memory Map

35.4 General Boot Procedure

During ROM startup, the boot mode is determined, and then control is passed to the boot loader. The loader first calls an init function for the boot driver responsible for reading boot images from the target boot port. The driver initializes the hardware port and external device, and then finds the boot image on

that device. The loader then requests boot image data from the driver. The boot image contains commands for loading code and data into memory, so the loader will interpret these commands and load the boot image into memory. At the end of a boot image, the loader passes control to the code that was loaded.

Boot images are encrypted, and customers have full control over the encryption keys by setting the CRYPTO_KEY TOP bits. Boot images are created by the Freescale-supplied *elftosb* application.

35.4.1 Preparing Bootable Images

Preparing a bootable image for all boot modes includes the following high-level steps:

- Prepare the ELF file for the firmware that is to be booted by the i.MX23 ROM.
- Run the ELF file through the *elftosb* program (available from Freescale), which generates an encrypted SB file that can be booted from ROM.

Any additional requirements for individual boot modes are identified in the following sections.

35.4.2 Constructing Image to Be Loaded by Boot Loader

The image is stored in an encrypted form that includes an authenticating hash. Freescale supplies a program called *elftosb* to convert a fully resolved executable binary image into a boot image usable by the boot loader. A key set must be input to the *elftosb* program to properly encrypt and authenticate the image. A default key set is supplied with *elftosb*. The process of creating a boot loader image is shown in Figure 35-2.

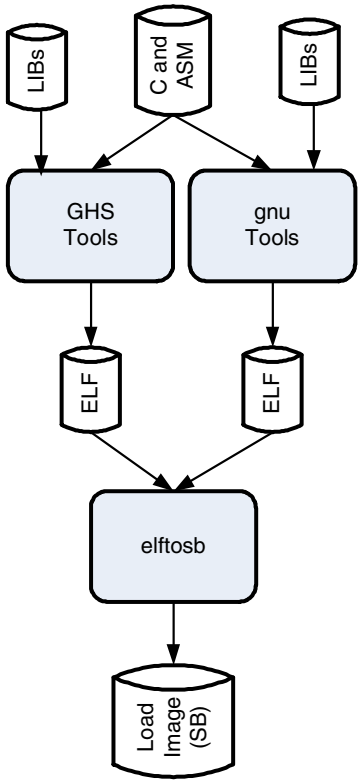


Figure 35-2. Creating a Boot Loader Image

35.5 I²C Boot Mode

EEPROMs must have the slave address 0xA0 (i.e., '1010000R', where R indicates a read if 1 and a write if 0). Also, the EEPROM must have exactly a two-byte “sub” address.

Boot images must start at address 0x0 of the EEPROM and cannot exceed 64 Kbytes in size. Note that because of the boot image size limitation, I2C boot mode will likely not be useful for OS boot.

The I^2C port is set to run at 400 kHz.

35.6 SPI Boot Mode

SPI memories are either EEPROMs or NORs.

By default, the SPI serial clock is set to 0.9 MHz for EEPROMs and 12 MHz for NORs. The SSP_SCK_INDEX OTP bits are used to change the SPI serial clock from defaults. These bits serve as the index for the SSP HAL clock rate array (see Section 35.6.2, “SSP,” for details on the clock rate array).

The defaults may also be changed by using the ConfigBlock.Clocks field (see ,”). If Config-Block.Clocks.SspClockConfig is non-zero, then that struct will be used to change the SPI SCK rate and will override the SSP_SCK_INDEX OTP setting.

This driver supports only 2-byte addresses for EEPROMs and 3-byte addresses for NORs.

This boot mode supports SPI mode 0 only.

35.6.1 Media Format

The media is arbitrarily partitioned into 128-byte “sectors”. An optional configuration block may reside on the media at byte address 0. This block has the following format:

```
//! \brief Spi media configuration block structs
typedef struct _spi_ConfigBlockFlags
{
    uint32_t  DisableFastRead:1; // Ignored for Spis
                                 // 0 - Do not disable fast reads
                                 // 1 - Disable fast reads
} spi_ConfigBlockFlags_t;

typedef struct _spi_ConfigBlockClocks
{
    uint32_t           SizeOfSspClockConfig; // sizeof(ssp_ClockConfig_t)
    ssp_ClockConfig_t  SspClockConfig;       // SSP clock configuration structure. A null
                                             //   structure indicates no clock change.
} spi_ConfigBlockClocks_t;

typedef struct _spi_ConfigBlock // Little Endian
{
    uint32_t  Signature;                     // 0x4D454D53, or "SMEM"
    uint32_t  BootStartAddr;                 // Start address of boot image. Must be >=
                                             //   sizeof(spi_ConfigBlock_t)
    uint32_t  SectorSize;                    // Sector size in bytes. Overrides ROM default
                                             //   of 128-bytes. Max is 1024-bytes.  0 is
                                             //   default 128-bytes.
    spi_ConfigBlockFlags_t Flags;            // Various flags. See spi_ConfigBlockFlags
    spi_ConfigBlockClocks_t Clocks;          // SCK clock update structure.
} spi_ConfigBlock_t;
```

If the block is present, then the boot image is found at the address specified by BootStartAddr. If the block is not present, then it assumes that the boot image resides on the media starting at byte address 0.

35.6.2 SSP

The SSP is used for the SPI boot mode.

The following table is used to look up a requested speed. If the speed is not an exact match, the boot ROM picks the next lowest value. Speed values can range from 1 to 50 MHz. A speed value of 0 is not allowed.

```
//  Lookup Table entry
typedef struct _ssp_clockConfig
{
    int    clkSel    :1;      //!< Clock Select (0=io_ref 1=xtal_ref)
    int    io_frac   :6;      //!< IO FRAC 18-35 (io_frac+16)
    int    ssp_frac  :9;      //!< SSP FRAC (1=default)
    int    ssp_div   :8;      //!< Divider: Must be an even value 2-254
    int    ssp_rate  :8;      //!< Serial Clock Rate
}
ssp_clockConfig_t;
```

The table is loaded with the clock rates listed in Table 35-8.

Table 35-8. SCK Clock Standard Values Lookup Table

SCK	N/A	.9	1	2	4	6	8	10	12	16	20	24	40	48	X	X
INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CLK_SEL	X	1	1	1	1	1	0	0	1	0	0	0	0	0		X
IOFRAC	X	X	X	X	X	X	18	18	X	18	18	18	18	18		X
SSP_FRAC	X	X	X	X	X	X	6	6	X	5	6	10	6	5		X
SSP_CLK		24	24	24	24	24	80	80	24	96	80	48	80	96		X
SSP_DIV	X	26	24	12	6	4	10	8	2	6	4	2	2	2		X
SSP_RATE	X	0	0	0	0	0	0	0	0	0	0	0	0	0		X

35.7 SD/MMC Boot Mode

SD/MMC boot mode supports booting from SD/MMC cards adhering to the following specifications:

- iNand Product Manual, Version 2.1
- SD Specifications, Part 1, Physical Layer Specification, Version 1.10
- SD Specifications, Part 1, Physical Layer Specifications, Version 2.0 Draft
- MultiMediaCard System Specification, Version 4.1
- MultiMediaCard System Specification, Version 4.2

Note, however, that this mode does not support dynamic insertion removal, so systems will typically not include removable cards.

The following modes are supported:

- SD/MMC on SSP1
- SD/MMC on SSP2

The SD_POWER_GATE_GPIO eFuse bits indicate which GPIO pin to use for controlling an external power gate for the connected device.

Table 35-9. GPIO Pin Selection

SD_POWER_GATE_GPIO	Gate GPIO
00b	PWM0
01b	LCD_DOTCK
10b	PWM3
11b	NONE

If a gate GPIO is used, then the driver will use the SD_POWER_UP_DELAY eFuse to determine the amount of time, in 10-ms increments, to wait until starting the 1-ms initialization sequence. This eFuse field is 6-bits wide, providing from 10–600 ms of delay. If the field is 000000b, then the delay is a default 20 ms. If no gate GPIO is specified in SD_POWER_GATE_GPIO, then the delay is skipped.

The SSP ports on the i.MX23 top out at 50 MHz with 20–40 pF loading. By default, the serial clock is set to 12 MHz. If the SD_SPEED_ENABLE persistent bit is set, then the driver will use a maximum speed based on the results of device identification and limited by choices available in the SSP clock index.

This mode supports the 1-bit, 4-bit, and 8-bit data MMC/SD buses. The SD_BUS_WIDTH efuse bits selects how many bus pins are physically available for the SSP port. Bus width will be limited based on these bits, as well as the bus width capabilities indicated by the connected device.

Table 35-10. Bus Pin Selection

SD_BUS_WIDTH	Width
00b	4-bit
01b	1-bit
10b	8-bit
11b	Reserved

The SD/MMC boot mode requires either a Boot Control Block (BCB) or Master Boot Record (MBR) on the device. The boot loader will first search for a MBR. If found, it will use the MBR data to find the boot image. If the MBR is not present, the boot loader will search for a BCB. If found, the BCB will provide the boot image located and size to the boot loader. If neither structure is fount, the boot loader will return an error to the ROM.

35.7.1 Boot Control Block (BCB)

The last physical sector of the device contains a media configuration block. This block contains the sector address of the boot image. The config block has the following format:

```
typedef struct {
    drive_type_t    eDriveType;
    uint32_t        Tag;
    uint32_t        Reserved[5];
} media_regions_t;

typedef struct {
    uint32_t        Signature;
    uint32_t        Version
    uint32_t        Reserved[1];
    uint32_t        NumRegions;
    media_regions_t Regions[];
} media_config_block_t;
```

The driver will first verify the Signature and Version, then search all NumRegions for the appropriate Tag. Table 35-11 shows the expected values for these parameters.

Table 35-11. Media Config Block Parameters

Field	Value
Signature	0x4D454D53
Version	0x00000001
Tag	0x00000050

35.7.2 Master Boot Record (MBR)

The first block of the device contains the master boot record (MBR). The MBR is identified by its signature located at offset 0x1FE of the first sector. The partition table is stored at address 0x1BE. The Freescale partition is identified by MBR_SIGMATEL_ID at an offset 0x04 from partition table. The boot image address is stored at offset 0x08 of partition table and size of the image at offset 0x10 of partition table.

Table 35-12. MBR Signature Bits

Field	Value
MBR Signature	0x55AA
MBR_SIGMATEL_ID	‘S’

35.7.3 Device Identification

SD/MMC boot mode uses the identification processes specified in SD Specifications, Part 1, Physical Layer Specifications, Version 2.0 Draft and MultiMediaCard System Specification, Version 4.2.

35.8 NAND Boot Mode

35.8.1 NAND Control Block (NCB)

As part of the NAND media initialization, the ROM driver uses safe NAND timings to search for a NAND Control Block (NCB) that contains the optimum NAND timings and the Factory Marked Bad Block Table. A flowchart is shown in Figure 35-3.

In the i.MX23, there are no separate boot modes for each type of ECC level. The hardware ECC level to use is embedded inside NCB block. The NCB data structure is itself protected using software ECC (SEC-DED Hamming Codes). Driver reads raw 2112 bytes of first sector and runs through software ECC engine that determines whether NCB data is valid or not.

If the NCB is found, the optimum NAND timings are loaded for further reads. If the software ECC fails, or the fingerprints do not match, the Block Search state machine increments 64 pages and reads that page

until 2n efNANDBootSearchLimit pages have been read. The 64-page stride value was picked as the smallest block size in many of the current NAND devices.

If a NCB cannot be found, the driver sets the NAND_SDK_BLOCK_REWRITE persistent bit and retries the Block Search looking for NCB2. If the second search fails, the NAND driver responds with an error and the boot ROM enters recovery mode.

Upon finding the NCB, the NAND parameters are read from the NCB, and the NAND driver updates the fail-safe parameters with the new parameters. These parameters include NAND timing, page size, and block size.

The next step is to find the Logical Drive Layout Block (LDLB), which uses the same search method as the NCB but the starting address will be different. The LDLB contains the Media Table, a pointer to the Discovered Bad Block Table (DBBT), and the sector information for the initial boot image. After the LDLB is read, the DBBT is loaded, and the initial boot image is loaded using the initial boot image starting address pointer.

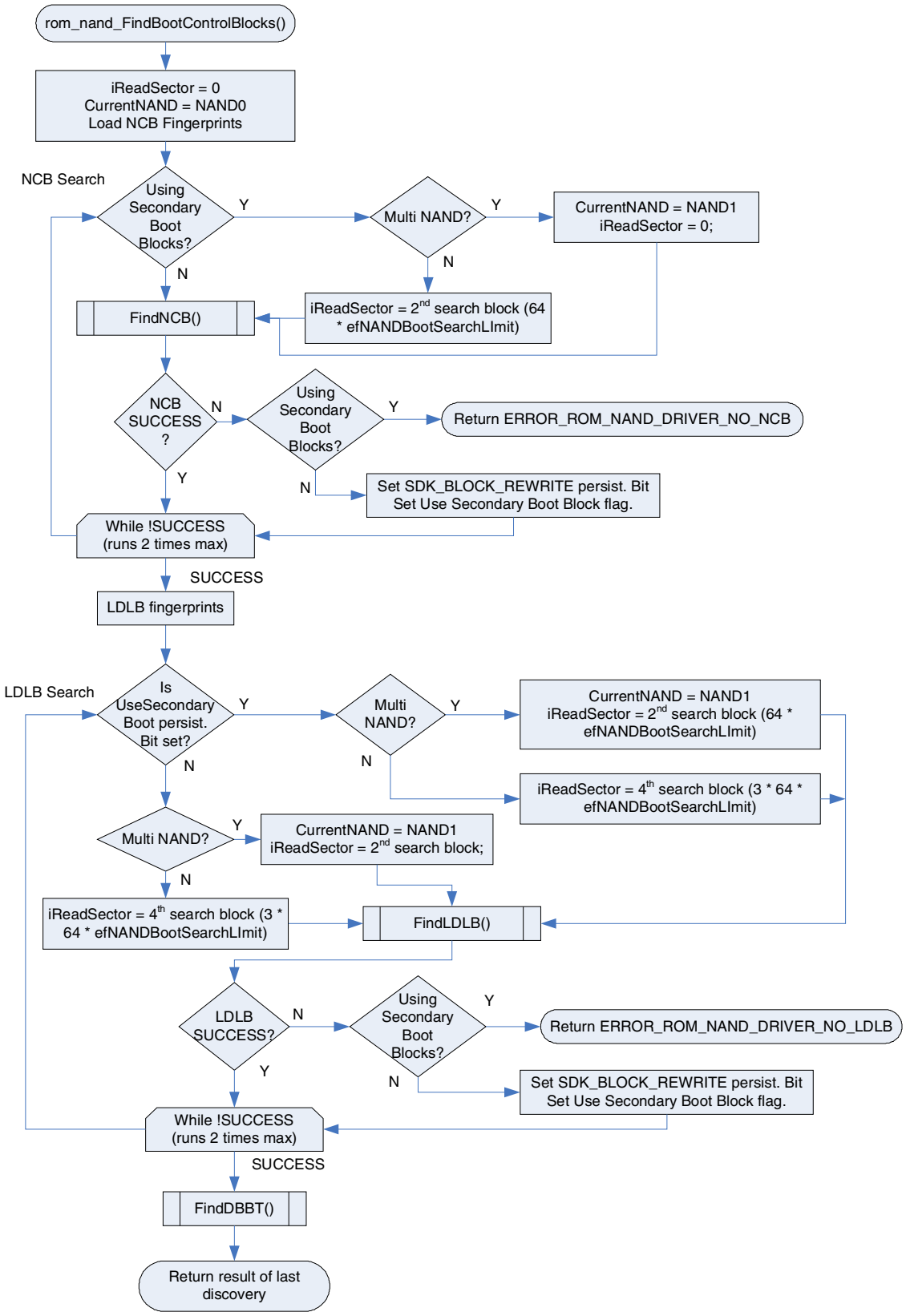


Figure 35-3. FindBootControlBlocks Flowchart

If the ECC fails or the fingerprints do not match, the Block Search state machine will increment 64 pages and read that page until 2^n efNANDBootSearchLimit pages have been read. If a LDLB cannot be found, the driver sets the NAND_SDK_BLOCK_REWRITE persistent bit and retries the Block Search. If the second search fails, the NAND driver responds with an error, and the boot ROM enters recovery mode. The block search flowchart is shown in Figure 35-4.

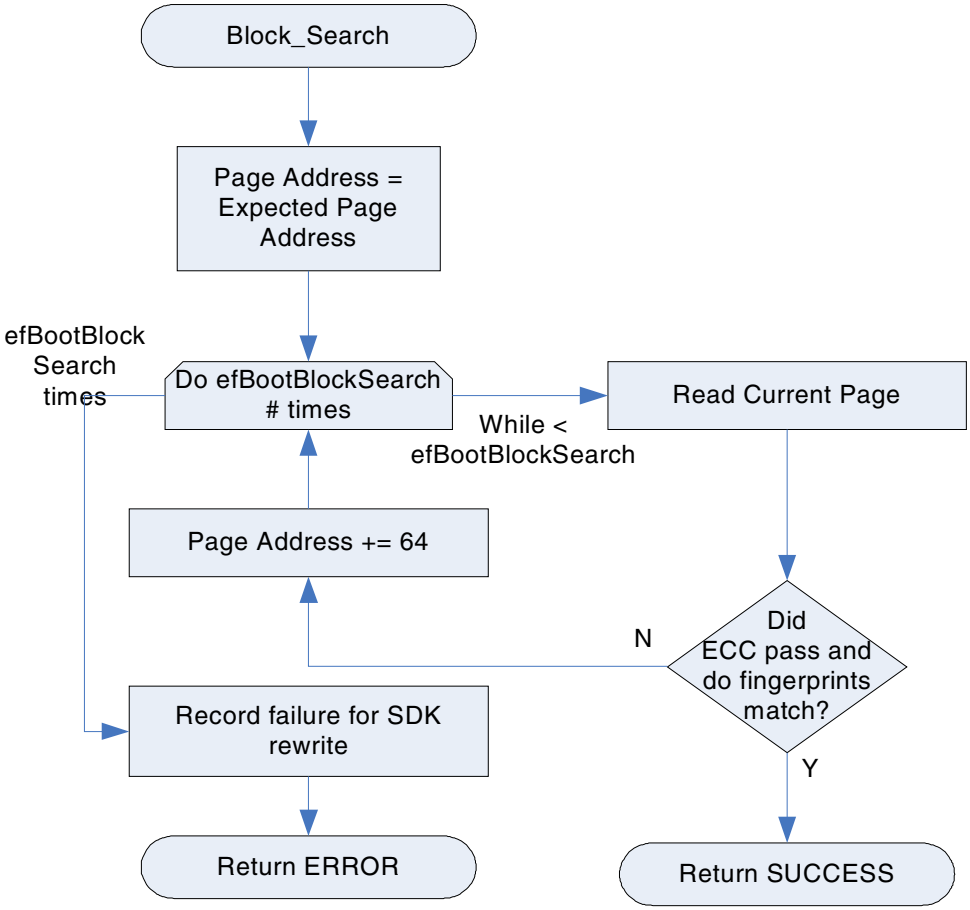


Figure 35-4. Block Search Flowchart

Once the LDLB is identified, more information about the boot image geometry is determined, including information required to calculate the appropriate boot image starting NAND, starting sector, stride, etc.

The NCB and LDLB search and load function also monitors the ECC correction threshold and sets the NAND_SDK_BLOCK_REWRITE persistent bit if the threshold exceeds three corrections in the four-bit ECC case and seven corrections in the eight-bit ECC case. One bit is used for all boot block images. If the NAND_SDK_BLOCK_REWRITE bit is set, the ROM continues loading the image, but the SDK will need to “refresh” the boot blocks at a later time.

35.8.2 NAND Patch Boot using NCB

A secondary mechanism to boot NAND patch image is implemented in the ROM. A flag is used in NCB data structure that would indicate to the ROM to boot the patch binary image present on the second page of first good block of NAND. If this flag is set, ROM will not try to locate LDLB and other boot blocks and would start loading the patch image.

35.8.3 Expected NAND Layout

The ROM expects the layout described in this section and illustrated in Figure 35-5 when booting from a single NAND configuration. The first search area contains the NAND Configuration Block (NCB) followed by an alternate NCB in the second search area. The third search area contains the Logical Drive Layout Block (LDLB), which is followed by an alternate LDLB in the fourth search area.

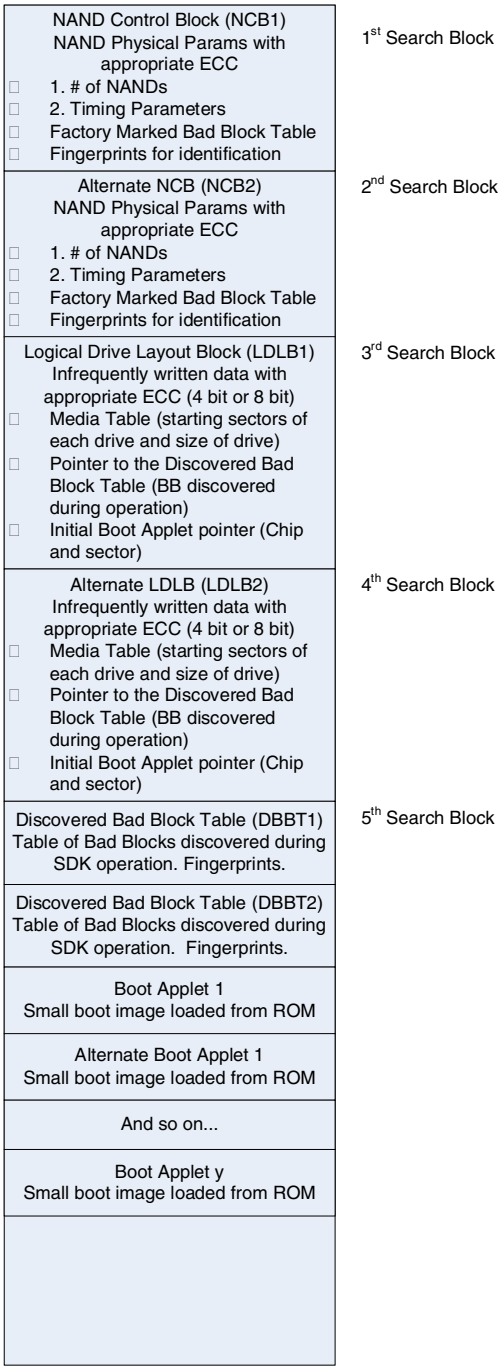


Figure 35-5. Expected NAND Layout

Search areas are defined as 64 pages * efSearchSize. They are defined in such a way that there is at least one extra block to hedge against a Boot Control Block (BCB) going bad during operation. If a Boot Control Block goes bad during operation, the data is copied from the original BCB to the extra BCB, and the original BCB is written with zeros. Since we rely upon both the ECC being correct and the fingerprints, overwriting the BCB with zeros should invalidate all the data. The search algorithm will search an entire search area before looking for the backup or secondary Boot Control Block.

35.8.3.1 NAND Config Block

The primary NAND Config Block (NCB) resides on the NAND attached to GPMI_CE0. The NCB is the first sector in the first good block. In the single NAND configuration, a copy of the NCB also resides on the NAND—its location is immediately after NCB1 - 2nd search area. In the case of multiple NAND devices, copies of the NCB, LDLB, and DBBT are located on the first two NANDs. This case is shown in more detail in Section 35.8.3.4, "Firmware Layout on the NAND."

The layout of first good page containing NCB is described in Figure 35-6.

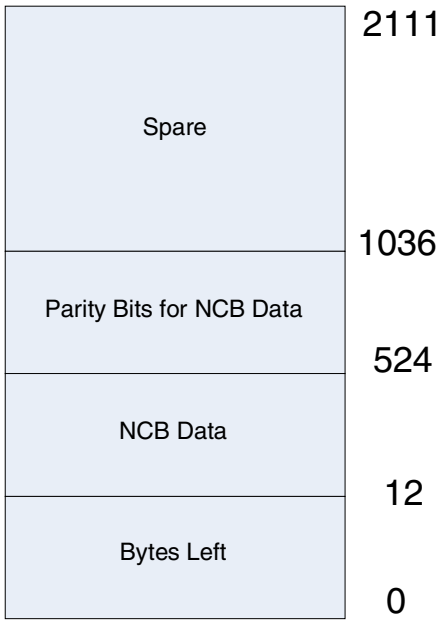


Figure 35-6. Layout of Boot Page Containing NCB

The NCB is located on the first good page of the NAND; minimum size of a page is 2112 bytes. The first 12 bytes of NCB page are reserved and left blank (all 0s), next 512 bytes are reserved for NCB data. The remaining 512 bytes are available for software ECC and the rest are all 0s. NCB is protected using SEC-DED Hamming codes.

35.8.3.2 Single Error Correct and Double Error Detect (SEC-DED) Hamming

For each 8 bits of data in 512 byte NCB, 5-bit parity is used. Each byte of parity area contains 5 parity bits (LSB) and 3 unused bits (MSB).

For each 8 bits of NCB data, parity is calculated and compared with corresponding parity bits read from the parity area of NCB page to detect errors and correct single error.

If errors are more than one then flag shall be raised against the block.

To determine a good NCB, all three fingerprints must match and the ECC must not fail.

The data held in the NCB includes:

- NAND Timing parameters.
- Which NANDs are in the system (GPMI_CE0, GPMI_CE1, GPMI_CE2, GPMI_CE3)—this is also in the eFuse.
- Geometry information (sectors per block, sectors per page, etc.).
- Additionally, the NCB is marked with three fingerprints in the sector.
- Factory Marked Bad Block Table.
- Pointer to physical sector of LDLB (on this chip) and, in a single NAND configuration, a pointer to the physical sector of LDLB2.
- A flag to boot NAND patch image located at sector 2 of first block.

35.8.3.3 Logical Drive Layout Block

A copy of the Logical Drive Layout Block (LDLB) resides on each NAND. A copy of the LDLB also resides on the NAND if it is a single NAND configuration, or on the first two NANDs if it is a multi-NAND configuration (See Figure 35-7). Its location is determined using the search area described above. The starting page will be the beginning of the 3rd search area. The LDLB has the same ECC as the rest of the system drives. The data held in the LDLB includes:

- Start Sector and stride of boot region.
- Start Sector and stride of alternate boot region.
- Tag of boot region.
- The LDLB is marked with three fingerprints in the sector.
- Media Table (Layout of drives on the NAND—used for SDK only).
- Pointer to the Discovered Bad Block Table and Alternate Discovered Bad Block Table (if a single NAND).

35.8.3.4 Firmware Layout on the NAND

The boot image firmware is stored in the NAND as shown in Figure 35-7.

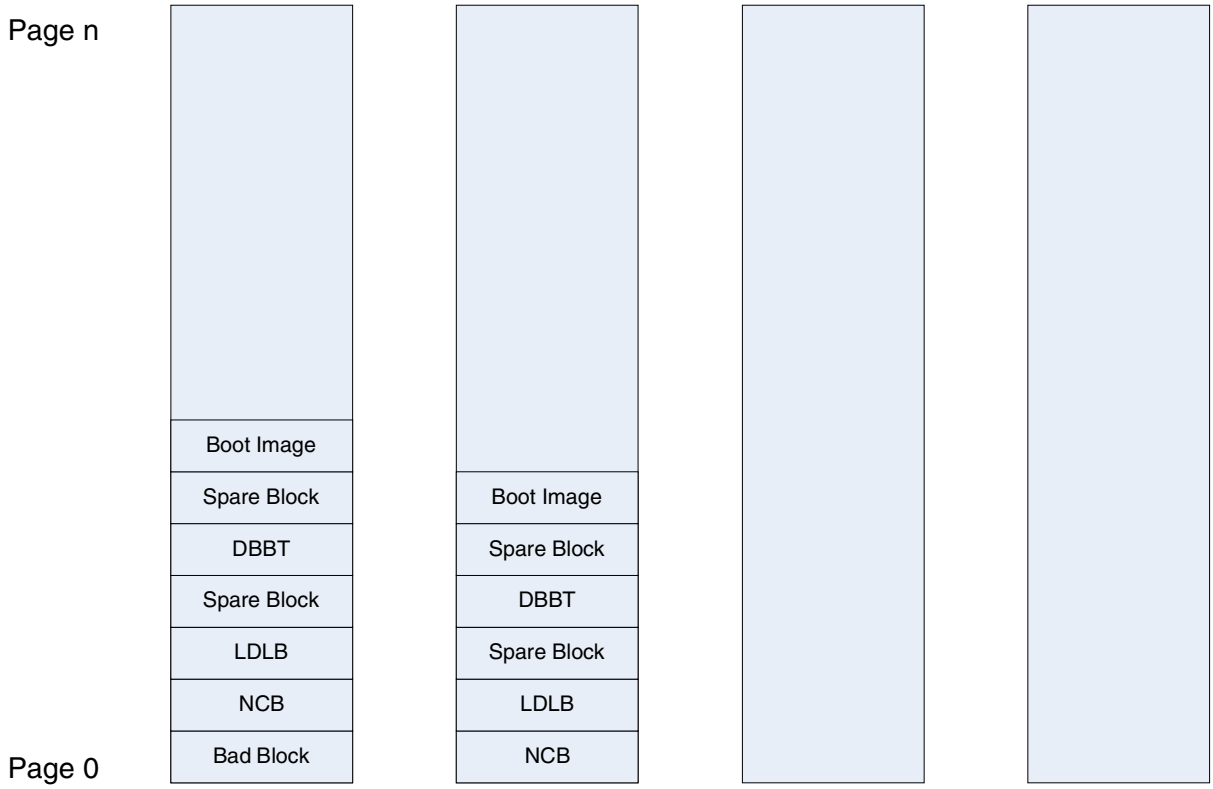


Figure 35-7. NAND Layout—Multiple NANDs

A pointer in the LDLB contains the physical sector address (on that NAND) of the boot image. The boot image is typically located on the first good block after the NCB, LDLB, and DBBT blocks and the additional blocks reserved for Boot Control Blocks that become Bad Blocks. The next sector will be drawn from the same block, until all sectors are drawn from the block. Then, the first sector of the subsequent block on NAND0 will be read. All boot blocks are located on the first NAND. If a block is determined to be bad (from the Discovered Bad Block Table or fails the ECC), then the algorithm will skip to the next block on NAND0, etc.

In a multiple NAND configuration, the first two NANDs will have the NCB, LDLB, DBBT, and boot images.

The DBBT will be duplicated across all the NANDs.

35.8.3.5 Recovery From a Failed Boot Firmware Image Read

The mechanism for recovering from a failed NCB or LDLB read is covered in Section 35.8.1, “NAND Control Block (NCB).” The SDK is warned about impending errors with the

NAND_SDK_BLOCK_REWRITE persistent bit and is notified of firmware boot errors with the NAND_SECONDARY_BOOT persistent bit.

In the case of a warning, the read routine will monitor the ECC threshold and set the NAND_SDK_BLOCK_REWRITE bit if the threshold is within one symbol of the maximum correctable number of symbols. The ROM continues to load from the primary boot image. At a later time, the SDK will refresh the primary boot images by copying and rewriting the primary boot image blocks.

If an error is discovered while reading the boot firmware image, the NAND driver sets the NAND_SECONDARY_BOOT persistent bit and resets the device. Booting will proceed the second time from the secondary boot images. If booting continues uninterrupted, no “unwinding” needs to take place at the loader level.

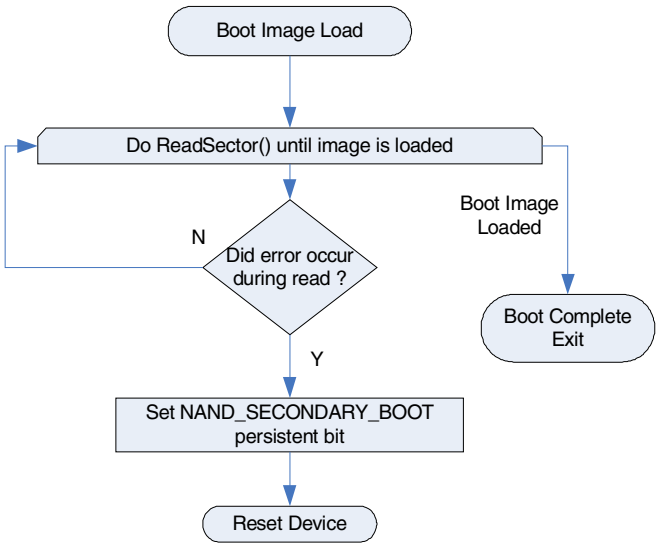


Figure 35-8. Boot Image Recovery

35.8.3.6 Bad Block Handling in the ROM

Bad blocks are not an issue for the Boot Control Blocks of the NAND because the NCB and LDLB Boot Control Blocks are found from a search as described in Section 35.8.1, “NAND Control Block (NCB).” The search for the Discovered Bad Block Table Header Control Block works with a similar mechanism. The search starts where the LDLB indicates the DBBT should be and progresses for 2^n efNANDBoot-SearchLimit times in the same fashion as the search described in Figure 35-4.

If the DBBT is not found, the ROM will search for the DBBT2. It will not change to boot from the secondary boot blocks even if the DBBT2 is found. Because the DBBT2 must be duplicated across all the NANDs, this is a safe assumption.

If no DBBTs are found, the ROM will assume there are no Bad Blocks and will continue to boot with no Bad Blocks.

During firmware boot, at the block boundary, the Bad Block table is searched for a match to the next block. If no match is found, the next block can be loaded. If a match is found, the block must be skipped and the next block checked.

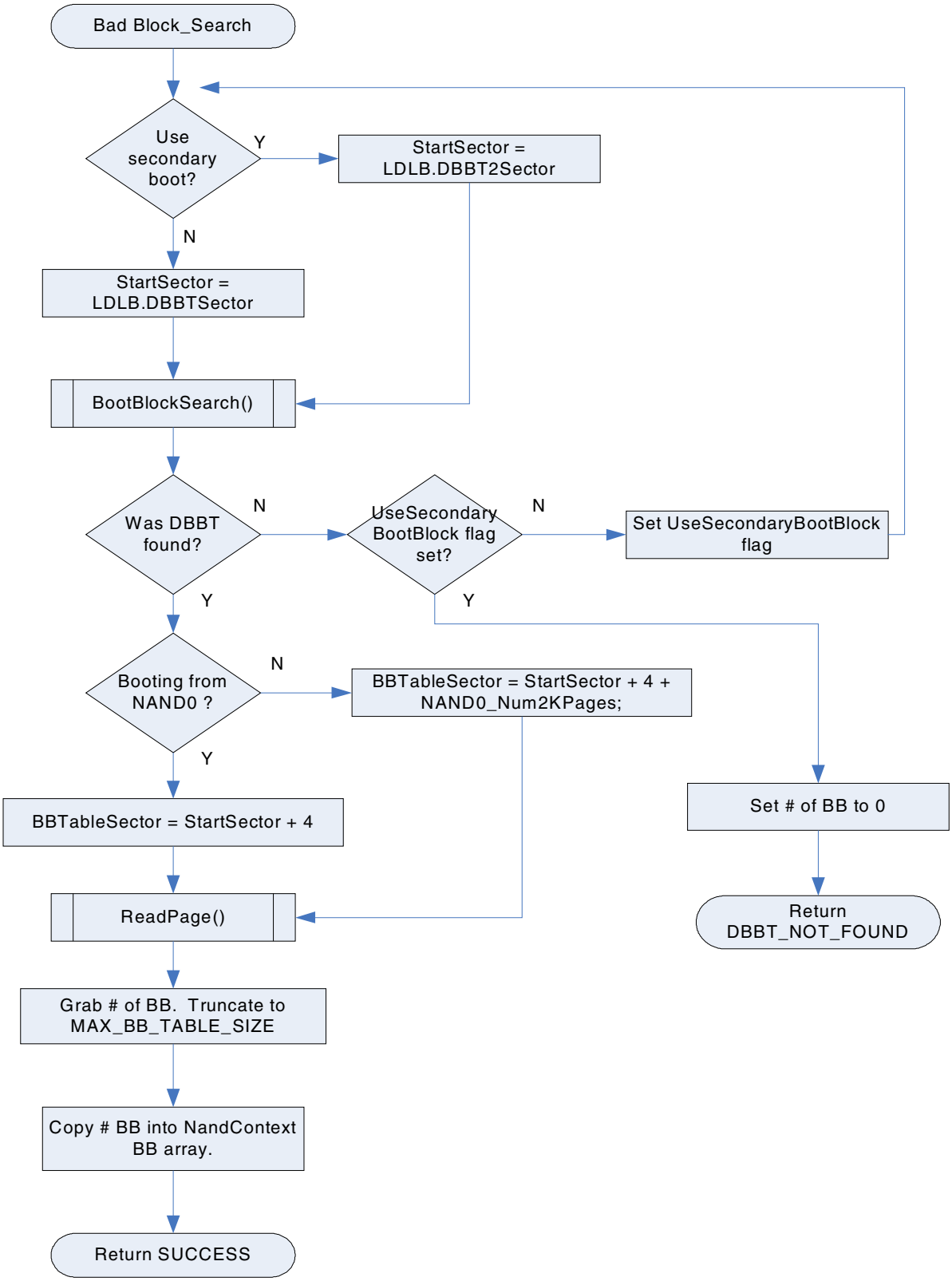


Figure 35-9. Bad Block Search

The DBBT structure is contained within a block and is discussed in more detail below. Figure 35-10 depicts the layout of the Discovered Bad Block Table block. The first 8K are reserved for the DBBT Header. The following pages are used by the DBBT for each NAND.

The Bad Block table for each NAND begins on a 2KB boundary that coincides with current NAND page sizes and is a subset of future NAND page sizes. The BBT can extend beyond 2K, which is the purpose of the #2K_num, and translates to the number of 2K pages that NAND0 through NAND3 require. In this way, the ROM can quickly index to the appropriate NAND table.

Only the Bad Blocks for the current NAND are loaded. In other words, in a dual NAND configuration, if the primary boot blocks are being used, then NAND0's BBTable will be loaded. Conversely, if the secondary boot blocks are being used, then NAND1's BB table will be loaded into the ROM's NandContext Bad Block array in NAND.

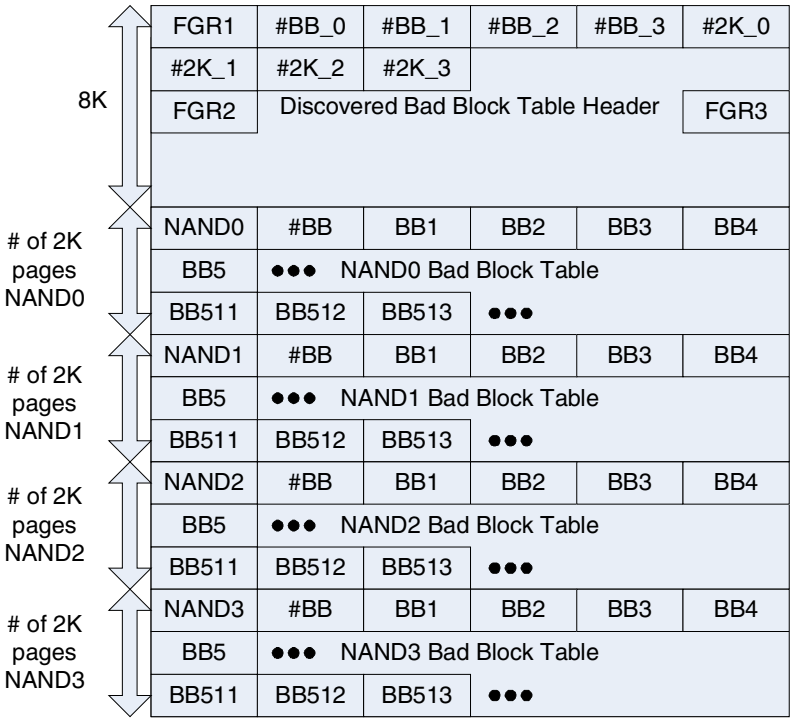


Figure 35-10. DBBT Layout

35.8.3.7 NAND Control Block Structure and Definitions

The NCB structure is as follows:

```
typedef struct _NAND_Control_Block
{
    uint32_t    m_u32Fingerprint1;
        struct
    {
        uint8_t m_u8DataSetup;
        uint8_t m_u8DataHold;
        uint8_t m_u8AddressSetup;
        uint8_t m_u8DSAMPLE_TIME;
    } NAND_Timing;
```

```
    uint32_t m_u32DataPageSize;      //!< 2048 for 2K pages, 4096 for 4K pages.
    uint32_t m_u32TotalPageSize;     //!< 2112 for 2K pages, 4314 for 4K pages.
    uint32_t m_u32SectorsPerBlock;   //!< Number of 2K sections per block.
    uint32_t m_u32SectorInPageMask;  //!< Mask for handling pages > 2K.
    uint32_t m_u32SectorToPageShift; //!< Address shift for handling pages > 2K.
    uint32_t m_u32NumberOfNANDs;     //!< Total Number of NANDs - not used by ROM.

    uint32_t m_u32Fingerprint2;      // @ word offset 10

    uint32_t m_u32NumRowBytes;       //!< Number of row bytes in read/write transactions.
    uint32_t m_u32NumColumnBytes;    //!< Number of row bytes in read/write transactions.
    uint32_t m_u32TotalInternalDie;  //!< Number of separate chips in this NAND.
    uint32_t m_u32InternalPlanesPerDie; //!<Number of internal planes-treat like separate
                                        //!<chips
    uint32_t m_u32CellType;  //!< MLC or SLC.
    uint32_t m_u32ECCType;   //!< Type of ECC, can be one of RS-4,8 or BCH-2-20
    uint32_t m_u32EccBlock0Size; //!< Number of bytes for Block0 - BCH
    uint32_t m_u32EccBlockNSize; //!< Block size in bytes for all blocks other than
                                 //!< Block0 - BCH
    uint32_t m_u32EccBlock0EccLevel;     //!< Ecc level for Block 0 - BCH
    uint32_t m_u32NumEccBlocksPerPage;   //!< Number of blocks per page - BCH
    uint32_t m_u32MetadataBytes;         //!< Metadata size - BCH
    uint32_t m_u32EraseThreshold;        //!< To set into BCH_MODE register.
    uint32_t m_u32Read1stCode;    //!< First value sent to initiate a NAND Read sequence.
    uint32_t m_u32Read2ndCode;    //!< Second value sent to initiate a NAND Read sequence.
    uint32_t m_u32BootPatch; //!< 0 for normal boot and 1 to load patch starting next to NCB.
    uint32_t m_u32PatchSectors;   //!< Size of patch in sectors.
    uint32_t m_u32Firmware_startingNAND2;    //!< duplicate required for patch boot.

    uint32_t    m_u32Fingerprint3; // @ word offset 20
} NAND_Control_Block;
```

Where:

m_u32Fingerprint1—32 bit word consisting of STMP (stored as 0x504d5453)

m_u32Fingerprint2—32 bit word consisting of NCB (stored as 0x2042434E)

m_u32Fingerprint3—32 bit word consisting of RBIN (stored as 0x4E494252)

m_u32_NumberOfNANDs—32 bit word detailing the number and configuration of NANDs that are in the system.

NAND_Timing structure

m_u8DataSetup—Number of nanoseconds required for NAND Data Setup.

m_u8DataHold—Number of nanoseconds required for NAND Data Hold.

m_u8AddressSetup—Number of nanoseconds required for NAND Data Setup.

m_u8DSAMPLE_TIME—Number of ½ GPMI cycles to wait before latching Read.

m_u32DataPageSize—Number of bytes in Data section of a page.

m_u32TotalPageSize—Number of bytes in a page - total.

m_ u32SectorsPerBlock—Number of 2K sections per block.
(4K pages = 2 x #pages)

m_ u32SectorInPageMask—Mask for handling pages > 2K.

m_ u32SectorToPageShift—Address shift for handling pages > 2K.

m_ u32NumRowBytes—Number of row bytes in read/write transactions.

m_ u32NumColumnBytes—Number of column bytes in read/write transactions.

m_u32TotalInternalDie—Number of die in this chip.

m_u32InternalPlanesPerDie—Number of planes or districts per die.

m_u32CellType—Type of NAND Cell—SLC, MLC.

m_u32ECCType—Type of ECC for this NAND—

0 – RS-4 bit ECC per 512 bytes
1 – RS-8 bit ECC per 512 bytes
2 – BCH-0 bit ECC per 512 bytes
3 – BCH-2 bit ECC per 512 bytes
4 – BCH-4 bit ECC per 512 bytes
5 – BCH-6 bit ECC per 512 bytes
6 – BCH-8 bit ECC per 512 bytes
7 – BCH-10 bit ECC per 512 bytes
8 – BCH-12 bit ECC per 512 bytes
9 – BCH-14 bit ECC per 512 bytes
10 – BCH-16 bit ECC per 512 bytes
11 – BCH-18 bit ECC per 512 bytes
12 – BCH-20 bit ECC per 512 bytes

m_32EccBlock0Size—Number of bytes for BCH block 0 of a page.

m_32EccBlockNSize—Number of bytes for BCH blocks N of a page; N=BCH blocks in a page except block 0.

m_u32EccBlock0EccLevel—ECC for BCH Block 0, Please refer to m_u32ECCType description for ECC values applicable to this field.

m_u32NumEccBlocksPerPage—Number of BCH blocks per page.

m_u32MetadataBytes—Metadata size – BCH only

m_u32EraseThreshold—This goes into BCH_MODE register.

m_ u32Read1stCode—Read command code—1st byte.

m_ u32Read2ndCode—Read command code—2nd byte.

m_u32BootPatch—To load patch image data located on 2nd page of NCB block.

m_u32PatchSectors—size of patch image data in sectors.

m_u32Firmware_startingNAND2—required for patch image boot for location of duplicate copy of firmware.

35.8.3.8 Logical Drive Layout Block Structure and Definitions

The first 512 bytes of the LDLB structure are as follows:

```
typedef struct _LogicalDriveLayoutBlock
{
    uint32_t    m_u32Fingerprint1;
    struct
    {
        uint16_t    m_u16Major;
        uint16_t    m_u16Minor;
        uint16_t    m_u16Sub;
    } LDLB_Version;
    uint32_t    m_u32Fingerprint2;
    uint32_t    m_u32Firmware_startingNAND;
    uint32_t    m_u32Firmware_startingSector;
    uint32_t    m_u32Firmware_sectorStride;
    uint32_t    m_u32SectorsInFirmware;
    uint32_t    m_u32Firmware_StartingNAND2;
    uint32_t    m_u32Firmware_StartingSector2;
    uint32_t    m_u32Firmware_SectorStride2;
    uint32_t    m_uSectorsInFirmware2;
    struct
    {
        uint16_t    m_u16Major;
        uint16_t    m_u16Minor;
        uint16_t    m_u16Sub;
    } FirmwareVersion;
    uint32_t    Rsvd[10];
    uint32_t    m_u32DiscoveredBBTableSector;
    uint32_t    m_u32DiscoveredBBTableSector2;
    uint32_t    m_u32Fingerprint3;
    uint32_t    RSVD[100];     // Region configuration used by SDK only.

} LogicalDriveLayoutBlock_t;;
```

Where:

m_u32Fingerprint1—32 bit word consisting of STMP (stored as 0x504d5453)

m_u32Fingerprint2—32 bit word consisting of LDLB (stored as 0x424C444C)

m_u32Fingerprint3—32 bit word consisting of RBIL (stored as 0x4C494252)

LDLBVersion structure—must be set to Major = 1, Minor = 0; Sub = anything.

m_u32Firmware_startingNAND/m_u32Firmware_startingNAND2—Which NAND holds the firmware that will get loaded. This is zero-based, so NAND0 will be 0, NAND1 will be 1, etc.

m_u32Firmware_startingSector/m_u32Firmware_StartingSector2—Which sector on the NAND to start with. Remember that sectors from the ROM's perspective are 512 bytes. Since the supported NANDs store data in 2K pages, this conversion will need a <<2 (or * 4). This is zero-based on the specific NAND (i.e., Sector0 of NAND3 will be 0 which may be physical sector 2048 (assuming NANDs are 1024 sectors per NAND) of the entire group of NANDs. Use 0 instead of 2048.)

m_u32Firmware_sectorStride/m_u32Firmware_SectorStride2—This 32 bit word describes how consecutive pages are read. Are the pages concatenated (read sequentially from the same block, then the next block on the same NAND), striped across NANDs on a sector basis (in NAND2 case -> sector 0 on NAND0, sector 0 on NAND1, sector 1 on NAND0, sector 1 on NAND1, etc.), striped across NANDs on a Block basis (sector 0-63 on NAND0, then sector 0-63 on NAND1, etc.)? Not used on current ROM.

m_u32SectorsInFirmware—Number of sectors (2K byte chunks) that will be read by the ROM.

m_ u32DiscoveredBBTableSector—Physical sector of Discovered Bad Block Table.

m_ u32DiscoveredBBTableSector—Physical sector of secondary Discovered Bad Block Table.

35.8.3.9 Discovered Bad Block Table Header Layout Block Structure and Definitions

The first 512 bytes of the DBBT header structure are as follows:

```
typedef struct _LogicalDriveLayoutBlock
{
    uint32_t    m_u32Fingerprint1;
    uint32_t        m_u32NumberBB_NAND0;
    uint32_t        m_u32NumberBB_NAND1;
    uint32_t        m_u32NumberBB_NAND2;
    uint32_t        m_u32NumberBB_NAND3;
    uint32_t        m_u32Number2KPagesBB_NAND0;
    uint32_t        m_u32Number2KPagesBB_NAND1;
    uint32_t        m_u32Number2KPagesBB_NAND2;
    uint32_t        m_u32Number2KPagesBB_NAND3;
    uint32_t    m_u32Fingerprint2;
    uint32_t    RSVD[20];     //
    uint32_t    m_u32Fingerprint3;
} LogicalDriveLayoutBlock_t;
```

Where:

m_u32Fingerprint1—32 bit word consisting of STMP (stored as 0x504d5453)

m_u32Fingerprint2—32 bit word consisting of DBBT (stored as 0x54424244)

m_u32Fingerprint3—32 bit word consisting of RBId (stored as 0x44494252)

m_u32NumberBB_NAND0—The number of Bad Blocks stored in this table for NAND0.

m_u32NumberBB_NAND1—The number of Bad Blocks stored in this table for NAND1.

m_u32NumberBB_NAND2—The number of Bad Blocks stored in this table for NAND2.

m_u32NumberBB_NAND3—The number of Bad Blocks stored in this table for NAND3.

m_u32Number2KPagesBB_NAND0—The number of 2K pages that the Bad Block table for NAND0 consumes.

m_u32Number2KPagesBB_NAND1—The number of 2K pages that the Bad Block table for NAND1 consumes.

m_u32Number2KPagesBB_NAND2—The number of 2K pages that the Bad Block table for NAND2 consumes.

m_u32Number2KPagesBB_NAND3—The number of 2Kpages that the Bad Block table for NAND3 consumes.

35.8.3.10 Discovered Bad Block Table Layout Block Structure and Definitions

The DBBT structure is as follows:

```
typedef struct _BadBlockTableNand_t
{
            uint32_t            uNAND;
uint32_t                        uNumberBB;
int16_t                         u16BadBlock[];
} BadBlockTableNand_t;
```

Where:

uNAND– 32 bit word denoting which NAND the table describes.

uNumberBB—32 bit word holding the number of Bad Blocks.

u16BadBlock—Array of 16 bit words holding the Bad Blocks for this particular NAND.

35.8.4 Typical NAND Page Organization

35.8.4.1 BCH ECC Page Organization.

For NAND boot, ROM restricts the size of a BCH data block to 512 bytes. The first data block is called block 0 and the rest of the blocks are called block N. Separate ECC levels can be used for block 0 and block N. The metadata bytes should be located at the beginning of a page, starting at byte 0, followed by data block 0, followed by ECC bytes for data block 0, followed by block 1 and its ECC bytes, and so on until N data blocks. The ECC level for block 0 can be different from the ECC level of rest of the blocks. Metadata bytes can be 0.

For NAND boot, with page size restrictions and data block size restricted to 512 bytes, only few combinations of ECC for block 0 and block N are possible.

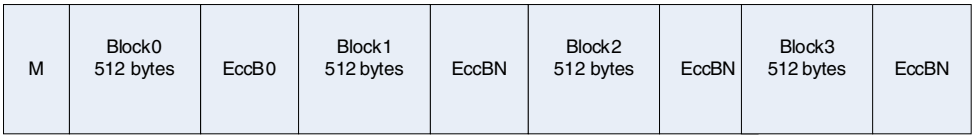


Figure 35-11. Valid layout for 2112 bytes sized page

The number of ECC bits required for a data block is calculated using (ECC_Correction_Level * 13) bits.

In the above layout the ECC size for EccB0 and EccBN should be selected to not exceed a total page size of 2112 bytes. EccB0 and EccBN can be one of 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 bits ECC correction level. The total bytes would then be:

[M + (data_block_size * 4) + ([EccB0 + (EccBN * 3)] * 13) / 8] <= 2112;

M = metadata bytes and data_block_size is 512.

There are 4 data blocks of 512 bytes each in a page of 2k page sized NAND. The values of EccB0 and EccBN should be such that the above calculation would not result in a value greater than 2112 bytes.

M	Block0 512 bytes	EccB0	Block1 512 bytes	EccBN	Block2 512 bytes	EccBN	Block3 512 bytes	EccBN
	Block4 512 bytes	EccBN	Block5 512 bytes	EccBN	Block6 512 bytes	EccBN	Block7 512 bytes	EccBN

Figure 35-12. Valid layout for 4K bytes sized page

Different NAND manufacturers have different sizes for a 4K page, 4314 bytes is typical.

[M + (data_block_size * 8) + ([EccB0 + (EccBN * 7)] * 13) / 8] <= 4314;

M = metadata bytes and data_block_size is 512.

There are 8 data blocks of 512 bytes each in a page of a 4k page sized NAND. The values of EccB0 and EccBN should be such that above calculation should not result in a value greater than the size of a page in a 4k page NAND.

35.8.4.2 2K Page Organization on the NAND for RS-4 Bit ECC

The ECC engine expects the following layout of data in a 2K page. Note that this configuration conflicts with the factory-marked bad block byte.

Each 512 byte group has an ECC and the metadata has its own ECC.

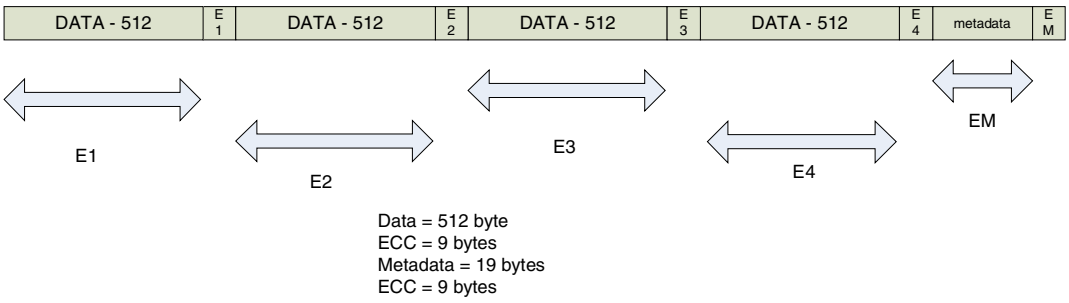


Figure 35-13. 2K Page Layout in NAND

35.8.4.3 In-Memory Organization for RS-4 Bit ECC

A data read or data write does not have the ECC interspersed in the data as shown in Figure 35-13. Instead, on a read, the data is stored as follows in the i.MX23 on-chip RAM memory.

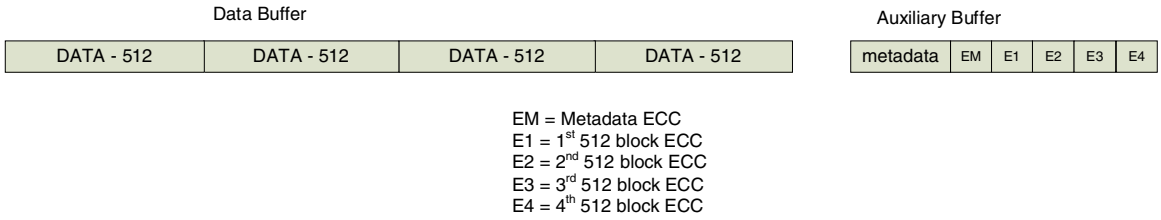


Figure 35-14. 2K Page Layout in On-Chip Memory

The ECC is stripped off and placed in the Auxiliary buffer. This Auxiliary buffer is 188 bytes for a 2K page NAND. In the write case, the 2048 byte buffer is sent to the GPMI, but the Auxiliary buffer must be allocated for scratch pad space.

35.8.4.4 Metadata

The Metadata consists of 19 bytes of the 2112 bytes in a page. Currently unused bytes are set to 0, but they can be any value as long as they are included in the ECC calculation. Currently, the ROM does not care about the metadata. Only the ECC bytes are important.

The ROM expects the boot image sectors to have metadata as described in Figure 35-15.

The EM shown in Figure 35-14 is the ECC for the metadata.

512	RS_1_ECC1
513	RS_1_ECC2
514	RS_1_ECC3
515	RS_1_ECC4
516	RS_1_ECC5
517	RS_1_ECC6
518	RS_1_ECC7
519	RS_1_ECC8
520	RS_1_ECC9

1554	RS_3_ECC1
1555	RS_3_ECC2
1556	RS_3_ECC3
1557	RS_3_ECC4
1558	RS_3_ECC5
1559	RS_3_ECC6
1560	RS_3_ECC7
1561	RS_3_ECC8
1562	RS_3_ECC9

1033	RS_2_ECC1
1034	RS_2_ECC2
1035	RS_2_ECC3
1036	RS_2_ECC4
1037	RS_2_ECC5
1038	RS_2_ECC6
1039	RS_2_ECC7
1040	RS_2_ECC8
1041	RS_2_ECC9

2075	RS_4_ECC1
2076	RS_4_ECC2
2078	RS_4_ECC3
2079	RS_4_ECC4
2080	RS_4_ECC5
2081	RS_4_ECC6
2082	RS_4_ECC7
2083	RS_4_ECC8
2084	RS_4_ECC9

Figure 35-15. Redundant Area—2K

35.8.4.5 4K Page Organization on the NAND for RS-8 Bit ECC

The ECC engine expects the following layout of data in a 4K page. Note that this configuration conflicts with the factory marked bad block byte.

Each 512 byte group has an ECC and the metadata has its own ECC.

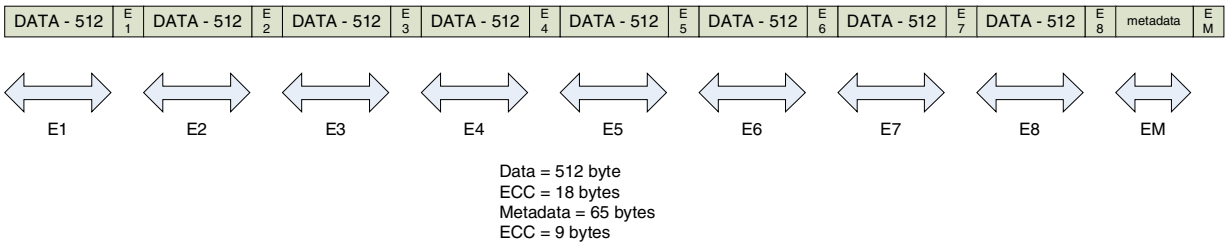


Figure 35-16. 4K Page in NAND

35.8.4.6 In-Memory Organization for RS-8 Bit ECC

A data read or data write does not have the ECC interspersed in the data as shown in Figure 35-16. Instead, on a read, the data is stored as follows in the i.MX23 on-chip RAM memory.

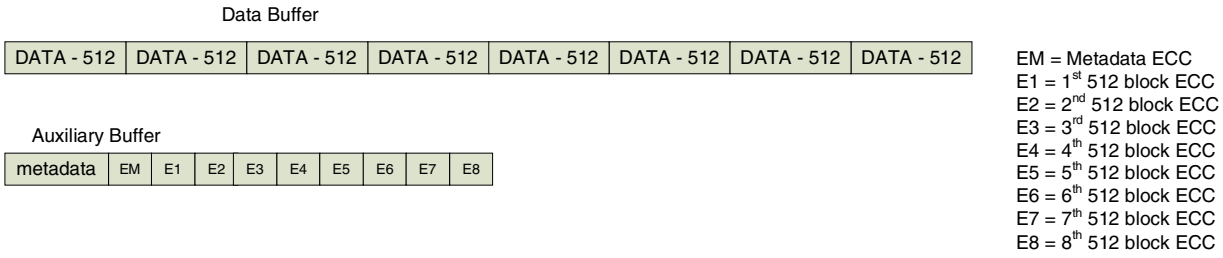


Figure 35-17. 4K Page Layout in On-Chip Memory

The ECC is stripped off and placed in the Auxiliary buffer. This Auxiliary buffer is 412 bytes in the 4K page case. In the write case, the 4096 byte buffer is sent to the GPMI, but the Auxiliary buffer must be allocated for scratch pad space.

35.9 USB Boot Driver

The USB Boot Driver is implemented as a USB HID class and is referred to as the Recovery HID, or RHID. The RHID serves as a fail-safe mechanism for downloading and communicating with application-specific code.

The system is based on two HID Application collections: the Boot Loader Transfer Controller (BLTC) and the Plug-in Transfer Controller (PITC). Each collection has its own set of HID reports.

35.9.1 Boot Loader Transfer Controller (BLTC)

The BLTC provides a tunnel to download application-specific PITCs to local system memory. The BLTC runs completely from ROM and interfaces directly to the ROM Loader. Typically, a PITC will be packaged on the host and downloaded through the BLTC and ROM Loader straight to OCRAM (or SDRAM).

Four HID reports are provided for communication with the BLTC:

- BLTC Command Out (BLCO)
- BLTC Data Out (BLDO)
- BLTC Data In (BLDI)
- BLTC Status In (BLSI)

The BLTC provides a command/data protocol for downloading code to the ROM Loader.

The BLTC has no knowledge of the contents of the data passing through so it is really possible to download anything (not just PITCs).

A list of USB bootloader commands is available upon request.

35.9.2 Plug-in Transfer Controller (PITC)

The PITC is a generic command/data/status tunnel that may be used for any type of application. The implementation only specifies the HID report structure and ROM HID-stack installation for a PITC—the protocol implementation is specific to a given PITC. Typically, a PITC will be downloaded to memory via the BLTC.

Four HID reports are provided for communication with a PITC:

- PITC Command Out (PICO)
- PITC Data Out (PIDO)
- PITC Data In (PIDI)
- PITC Status In (PISI)

The command/data protocol is specific to any given PITC.

Only one PITC may be installed at any given time.

35.9.3 USB IDs and Serial Number

By default the USB Device Descriptor Vendor ID, Product ID, and serial number are reported as follows:

- Vendor ID—0x066F
- Product ID—0x3700
- Serial Number String—none

If any of the HW_OCOTP_ROM2 bits are blown, then the full contents of that OTP register are used to report the Vendor ID and Product ID. If the ENABLE_USB_BOOT_SERIAL_NUMBER OTP is blown, then a unique serial number will be generated from the factory-programmed SGTL_OPS3 OTP registers.

35.9.4 USB Recovery Mode

USB boot mode is provided as a fail-safe mechanism for writing system firmware to the boot media. The boot mode is not usually entered by the normal methods of setting the boot pins or OTP: the other methods of entering USB boot mode are referred to generally as recovery mode.

An end user can manually start recovery mode by holding the recovery switch for several seconds while plugging in USB. Holding the recovery switch is defined as reading the i.MX23 PSWITCH input as a 0x3. There are several switch circuits that will produce this input. The loader also automatically starts recovery mode if a non-recoverable error is detected from any boot mode other than USB.

If the DISABLE_RECOVERY_MODE OTP bit is blown, then USB boot mode is disabled completely. Attempts to enter USB boot mode via boot pins, OTP, or recovery methods will result in a chip power-down.

Chapter 36
Pin Descriptions

This chapter provides various views of the pinout for the i.MX23.

- Pin definitions for the 128-Pin LQFP are in Section 36.1, “Pin Definitions for 128-Pin LQFP.”
- Pin definitions for the 169-Pin BGA are in Section 36.2, “Pin Definitions for 169-Pin BGA.”

The pin tables in this chapter include columns with the headings “Description 1”, “Description 2”, and “Description 3”. These columns refer to the different functions that can be enabled for each individual pin by programming the pin control registers (HW_PINCTRL_MUXSEL*x*). For example:

- Enable the function listed in the “Description 1” column by programming the BANK*x*_PIN*x* bit field to 00.
- Enable the function listed in the “Description 2” column by programming the BANK*x*_PIN*x* bit field to 01.
- Enable the function listed in the “Description 3” column by programming the BANK*x*_PIN*x* bit field to 10.

See Chapter 37, “Pin Control and GPIO.” for more information.

Table 36-1 lists the abbreviations used in the pin tables in this chapter.

Table 36-1. Nomenclature for Pin Tables

TYPE	DESCRIPTION
A	Analog pin
I	Input pin
I/O	Input/output pin
O	Output pin
P	Power pin

MODULE	DESCRIPTION
ADC	ADC analog pins
CLOCK	Clock pins
DCDC	DC-DC Converter pins
EMI	External Memory Interface pins
ETM	Embedded Trace Macrocell
GPIO	General-Purpose Input/Output pins
GPMI	General-Purpose Media Interface (NAND) pins
HP	Headphone pins
I^2C	I^2C pins
LCDIF	LCD Interface pins
LineOut	Line Out pins
LRADC	Low-Resolution ADC/Touch-Screen pins
POWER	Power pins
PWM	Pulse Width Modulator pins
RTC	Real-Time Clock pins
SSP	Synchronous Serial Port pins
SYSTEM	System pins
TIMER	Timer/Rotary Encoder pins
UART	Either debug or application UART pins
USB	USB pins

Note: Almost all digital pins are powered down (i.e., high-impedance) at reset, until reprogrammed. The only exceptions are DEBUG. This pin is always active.

36.1 Pin Definitions for 128-Pin LQFP

This section includes the following pin information for the 128-pin LQFP package:

- Table 36-2, “128-Pin LQFP Pin Definitions by Pin Name,” on page 2
- Table 36-3, “128-Pin LQFP Pin Definitions by Pin Number,” on page 5
- Table 36-4, “128-Pin LQFP Connection Diagram—Top View,” on page 9

Table 36-2. 128-Pin LQFP Pin Definitions by Pin Name

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
103	BATT	DCDC	A	Battery Input		
100	DCDC_BATTERY	DCDC	A	DCDC Battery		
98	DCDC_GND	DCDC	A	DCDC Ground		
97	DCDC_LN1	DCDC	A	DCDC Inductor N 1		
99	DCDC_LP	DCDC	A	DCDC Inductor P		
96	DCDC_VDDA	DCDC	A	DCDC Analog Power		

Table 36-2. 128-Pin LQFP Pin Definitions by Pin Name (continued)

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
94	DCDC_VDDD	DCDC	A	DCDC Digital Core Power		
95	DCDC_VDDIO	DCDC	A	DCDC I/O Power		
76	EMI_A00	EMI	DIO	EMI Address 0		
75	EMI_A01	EMI	DIO	EMI Address 1		
74	EMI_A02	EMI	DIO	EMI Address 2		
73	EMI_A03	EMI	DIO	EMI Address 3		
72	EMI_A04	EMI	DIO	EMI Address 4		
71	EMI_A05	EMI	DIO	EMI Address 5		
70	EMI_A06	EMI	DIO	EMI Address 6		
69	EMI_A07	EMI	DIO	EMI Address 7		
68	EMI_A08	EMI	DIO	EMI Address 8		
67	EMI_A09	EMI	DIO	EMI Address 9		
66	EMI_A10	EMI	DIO	EMI Address 10		
65	EMI_A11	EMI	DIO	EMI Address 11		
64	EMI_A12	EMI	DIO	EMI Address 12		
79	EMI_BA0	EMI	DIO	DRAM Bank Address 0		
80	EMI_BA1	EMI	DIO	DRAM Bank Address 1		
61	EMI_CASN	EMI	DIO	EMI Column Address Strobe#		
63	EMI_CE0N	EMI	DIO	EMI Chip Enable 0#		
78	EMI_CKE	EMI	DIO	EMI Clock Enable		
36	EMI_CLK	EMI	DIO	EMI Clock		
37	EMI_CLKN	EMI	DIO	EMI Clock#		
41	EMI_D00	EMI	DIO	EMI Data 0		
43	EMI_D01	EMI	DIO	EMI Data 1		
42	EMI_D02	EMI	DIO	EMI Data 2		
44	EMI_D03	EMI	DIO	EMI Data 3		
47	EMI_D04	EMI	DIO	EMI Data 4		
48	EMI_D05	EMI	DIO	EMI Data 5		
49	EMI_D06	EMI	DIO	EMI Data 6		
50	EMI_D07	EMI	DIO	EMI Data 7		
51	EMI_D08	EMI	DIO	EMI Data 8		
52	EMI_D09	EMI	DIO	EMI Data 9		
54	EMI_D10	EMI	DIO	EMI Data 10		
55	EMI_D11	EMI	DIO	EMI Data 11		
57	EMI_D12	EMI	DIO	EMI Data 12		
58	EMI_D13	EMI	DIO	EMI Data 13		
60	EMI_D14	EMI	DIO	EMI Data 14		
59	EMI_D15	EMI	DIO	EMI Data 15		
46	EMI_DQM0	EMI	DIO	EMI Data Mask 0		
56	EMI_DQM1	EMI	DIO	EMI Data Mask1		
39	EMI_DQS0	EMI	DIO	EMI mDDR-DDR Data Strobe 0 (Low Byte)		
40	EMI_DQS1	EMI	DIO	EMI mDDR-DDR Data Strobe 1 (High Byte)		
62	EMI_RASN	EMI	DIO	EMI Row Address Strobe#		
77	EMI_WEN	EMI	DIO	EMI Write Enable#		
20	GPMI_ALE	GPMI	DIO	NAND ALE	LCD_D17	
82	GPMI_CE0N	GPMI	DIO	GPMI Chip Enable 0#		

Table 36-2. 128-Pin LQFP Pin Definitions by Pin Name (continued)

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
81	GPMI_CE1N	GPMI	DIO	GPMI Chip Enable 1#		
19	GPMI_CLE	GPMI	DIO	NAND CLE	LCD_D16	
22	GPMI_D00	GPMI	DIO	NAND Data 0	LCD_D8	SSP2_DATA0
23	GPMI_D01	GPMI	DIO	NAND Data 1	LCD_D9	SSP2_DATA1
24	GPMI_D02	GPMI	DIO	NAND Data 2	LCD_D10	SSP2_DATA2
25	GPMI_D03	GPMI	DIO	NAND Data 3	LCD_D11	SSP2_DATA3
27	GPMI_D04	GPMI	DIO	NAND Data 4	LCD_D12	SSP2_DATA4
26	GPMI_D05	GPMI	DIO	NAND Data 5	LCD_D13	SSP2_DATA5
29	GPMI_D06	GPMI	DIO	NAND Data 6	LCD_D14	SSP2_DATA6
28	GPMI_D07	GPMI	DIO	NAND Data 7	LCD_D15	SSP2_DATA7
31	GPMI_RDN	GPMI	DIO	NAND Read Strobe		
32	GPMI_RDY0	GPMI	DIO	NAND0 Ready-Busy#		SSP2_DETECT
21	GPMI_RDY1	GPMI	DIO	NAND1 Ready-Busy#		SSP2_CMD
34	GPMI_WPN	GPMI	DIO	NAND Write Protect		
33	GPMI_WRN	GPMI	DIO	NAND Write Strobe		SSP2_SCK
113	HPL	HP	A	Headphone Left		
109	HPR	HP	A	Headphone Right		
111	HP_VGND	HP	A	Direct Coupled Headphone Virtual Ground		
127	I2C_SCL	I2C	DIO	I2C Serial Clock	GPMI_RDY2	AUART1_TX
128	I2C_SDA	I2C	DIO	I2C Serial Data	GPMI_CE2N	AUART1_RX
10	LCD_CS	LCDIF	DIO	LCD Interface Chip Select		
2	LCD_D00	LCDIF	DIO	LCD Interface Data 0		
3	LCD_D01	LCDIF	DIO	LCD Interface Data 1		
4	LCD_D02	LCDIF	DIO	LCD Interface Data 2		
5	LCD_D03	LCDIF	DIO	LCD Interface Data 3		
6	LCD_D04	LCDIF	DIO	LCD Interface Data 4		
7	LCD_D05	LCDIF	DIO	LCD Interface Data 5		
8	LCD_D06	LCDIF	DIO	LCD Interface Data 6		
9	LCD_D07	LCDIF	DIO	LCD Interface Data 7		
17	LCD_DOTCK	LCDIF	DIO	LCD Interface DOT clock	GPMI_RDY3	
11	LCD_ENABLE	LCDIF	DIO	LCD Interface Enable	I2C_SCL	
15	LCD_HSYNC	LCDIF	DIO	LCD Interface horizontal sync	I2C_SDA	
12	LCD_RESET	LCDIF	DIO	LCD Interface Reset Out		GPMI_CE3N
14	LCD_RS	LCDIF	DIO	LCD Interface Register Select / LCD CCIR clock		
16	LCD_VSYNC	LCDIF	DIO	LCD Interface Vertical Sync	LCD_BUSY	
13	LCD_WR	LCDIF	DIO	LCD Interface 6800 E / 8800 WR		
115	LINE1_INL	ADC	A	Line-in 1 Left		
114	LINE1_INR	ADC	A	Line-in 1 Right		
108	LRADC0	LRADC	A	LRADC0 (Button 1 or Temp or MicBias)		
107	LRADC1	LRADC	A	LRADC1 (Button 2 or Temp or MicBias)		
116	MIC	ADC	A	Microphone Input		
119	PSWITCH	DCDC	A	Power On / Recovery		
125	PWM0	PWM	DIO	PWM 0	ROTARYA	DUART RX
126	PWM1	PWM	DIO	PWM 1	ROTARYB	DUART TX
91	PWM2	PWM	DIO	PWM 2	GPMI_RDY3	
83	SSP1_CMD	SSP	DIO	SD-MMC CMD / SPI MOSI		JTAG_TDO

Table 36-2. 128-Pin LQFP Pin Definitions by Pin Name (continued)

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
84	SSP1_DATA0	SSP	DIO	SD-MMC Data0 / SPI MISO		JTAG_TDI
85	SSP1_DATA1	SSP	DIO	SD-MMC Data1	I2C_SCL	JTAG_TCLK
86	SSP1_DATA2	SSP	DIO	SD-MMC Data2	I2C_SDA	JTAG_RTCK
87	SSP1_DATA3	SSP	DIO	SD-MMC Data3 / SPI Slave Select 0		JTAG_TMS
88	SSP1_DETECT	SSP	DIO	Removable Card Detect	GPMI_CE3N	USB_ID
90	SSP1_SCK	SSP	DIO	SSP Serial Clock		JTAG_TRST
123	USB_DM	USB	A	USB Negative Data Line		
124	USB_DP	USB	A	USB Positive Data Line		
117	VAG	HP	A	Analog Reference Capacitor		
104	VDAC1	VDAC	A	VDAC composite out		
101	VDD4P2	DCDC	A	DCDC 4.2V Regulated Output		
102	VDD5V	DCDC	A	5V Power Input		
110	VDDA1	POWER	A	Analog Power 1		
1	VDDD1	POWER	P	Digital Core Power 1		
93	VDDD3	POWER	P	Digital Core Power 3		
106	VDDM	DCDC	A	2.5V Supply for external DDR DRAM	LRADC4	
45	VDDIO_EMI1	POWER	P	Digital I/O Power 1 - EMI		
53	VDDIO_EMI2	POWER	P	Digital I/O Power 2 - EMI		
38	VDDIO_EMIQ	POWER	P	EMI quiet supply		
18	VDDIO33_1	POWER	P	Digital I/O 3.3v Power 1		
92	VDDIO33_3	POWER	P	Digital I/O 3.3v Power 3		
120	VDDXTAL	CLOCK	A	Crystal Power Filter Cap - Cross bond to side 4		
112	VSSA1	POWER	A	Analog Ground 1		
118	VSSA2	POWER	A	Analog Ground 2 (quiet)		
105	VSSA4	POWER	A	Shared Analog Ground 1c		
89	DEBUG	SYSTEM	I/O	Serial JTAG Debug Port		
30	VSSD1	POWER	D	Ground		
35	VSSD2	POWER	D	Ground		
122	XTALI	CLOCK	A	Crystal In - 24MHz		
121	XTALO	CLOCK	A	Crystal Out - 24MHz		

Table 36-3. 128-Pin LQFP Pin Definitions by Pin Number

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
1	VDDD1	POWER	P	Digital Core Power 1		
2	LCD_D00	LCDIF	DIO	LCD Interface Data 0		
3	LCD_D01	LCDIF	DIO	LCD Interface Data 1		
4	LCD_D02	LCDIF	DIO	LCD Interface Data 2		
5	LCD_D03	LCDIF	DIO	LCD Interface Data 3		
6	LCD_D04	LCDIF	DIO	LCD Interface Data 4		
7	LCD_D05	LCDIF	DIO	LCD Interface Data 5		
8	LCD_D06	LCDIF	DIO	LCD Interface Data 6		
9	LCD_D07	LCDIF	DIO	LCD Interface Data 7		
10	LCD_CS	LCDIF	DIO	LCD Interface Chip Select		
11	LCD_ENABLE	LCDIF	DIO	LCD Interface Enable	I2C_SCL	
12	LCD_RESET	LCDIF	DIO	LCD Interface Reset Out		GPMI_CE3N

Table 36-3. 128-Pin LQFP Pin Definitions by Pin Number (continued)

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
13	LCD_WR	LCDIF	DIO	LCD Interface 6800 E / 8800 WR		
14	LCD_RS	LCDIF	DIO	LCD Interface Register Select / LCD CCIR clock		
15	LCD_HSYNC	LCDIF	DIO	LCD Interface horizontal sync	I2C_SDA	
16	LCD_VSYNC	LCDIF	DIO	LCD Interface Vertical Sync	LCD_BUSY	
17	LCD_DOTCK	LCDIF	DIO	LCD Interface DOT clock	GPMI_RDY3	
18	VDDIO33_1	POWER	P	Digital I/O 3.3v Power 1		
19	GPMI_CLE	GPMI	DIO	NAND CLE	LCD_D16	
20	GPMI_ALE	GPMI	DIO	NAND ALE	LCD_D17	
21	GPMI_RDY1	GPMI	DIO	NAND1 Ready-Busy#		SSP2_CMD
22	GPMI_D00	GPMI	DIO	NAND Data 0	LCD_D8	SSP2_DATA0
23	GPMI_D01	GPMI	DIO	NAND Data 1	LCD_D9	SSP2_DATA1
24	GPMI_D02	GPMI	DIO	NAND Data 2	LCD_D10	SSP2_DATA2
25	GPMI_D03	GPMI	DIO	NAND Data 3	LCD_D11	SSP2_DATA3
26	GPMI_D05	GPMI	DIO	NAND Data 5	LCD_D13	SSP2_DATA5
27	GPMI_D04	GPMI	DIO	NAND Data 4	LCD_D12	SSP2_DATA4
28	GPMI_D07	GPMI	DIO	NAND Data 7	LCD_D15	SSP2_DATA7
29	GPMI_D06	GPMI	DIO	NAND Data 6	LCD_D14	SSP2_DATA6
30	VSSD1	POWER	P	Ground		
31	GPMI_RDN	GPMI	DIO	NAND Read Strobe		
32	GPMI_RDY0	GPMI	DIO	NAND0 Ready-Busy#		SSP2_DETECT
33	GPMI_WRN	GPMI	DIO	NAND Write Strobe		SSP2_SCK
34	GPMI_WPN	GPMI	DIO	NAND Write Protect		
35	VSSD2	POWER	P	Ground		
36	EMI_CLK	EMI	DIO	EMI Clock		
37	EMI_CLKN	EMI	DIO	EMI Clock#		
38	VDDIO_EMIQ	POWER	P	EMI quiet supply		
39	EMI_DQS0	EMI	DIO	EMI mDDR-DDR Data Strobe 0 (Low Byte)		
40	EMI_DQS1	EMI	DIO	EMI mDDR-DDR Data Strobe 1 (High Byte)		
41	EMI_D00	EMI	DIO	EMI Data 0		
42	EMI_D02	EMI	DIO	EMI Data 2		
43	EMI_D01	EMI	DIO	EMI Data 1		
44	EMI_D03	EMI	DIO	EMI Data 3		
45	VDDIO_EMI1	POWER	P	Digital I/O Power 1 - EMI		
46	EMI_DQM0	EMI	DIO	EMI Data Mask 0		
47	EMI_D04	EMI	DIO	EMI Data 4		
48	EMI_D05	EMI	DIO	EMI Data 5		
49	EMI_D06	EMI	DIO	EMI Data 6		
50	EMI_D07	EMI	DIO	EMI Data 7		
51	EMI_D08	EMI	DIO	EMI Data 8		
52	EMI_D09	EMI	DIO	EMI Data 9		
53	VDDIO_EMI2	POWER	P	Digital I/O Power 2 - EMI		
54	EMI_D10	EMI	DIO	EMI Data 10		
55	EMI_D11	EMI	DIO	EMI Data 11		
56	EMI_DQM1	EMI	DIO	EMI Data Mask1		
57	EMI_D12	EMI	DIO	EMI Data 12		
58	EMI_D13	EMI	DIO	EMI Data 13		

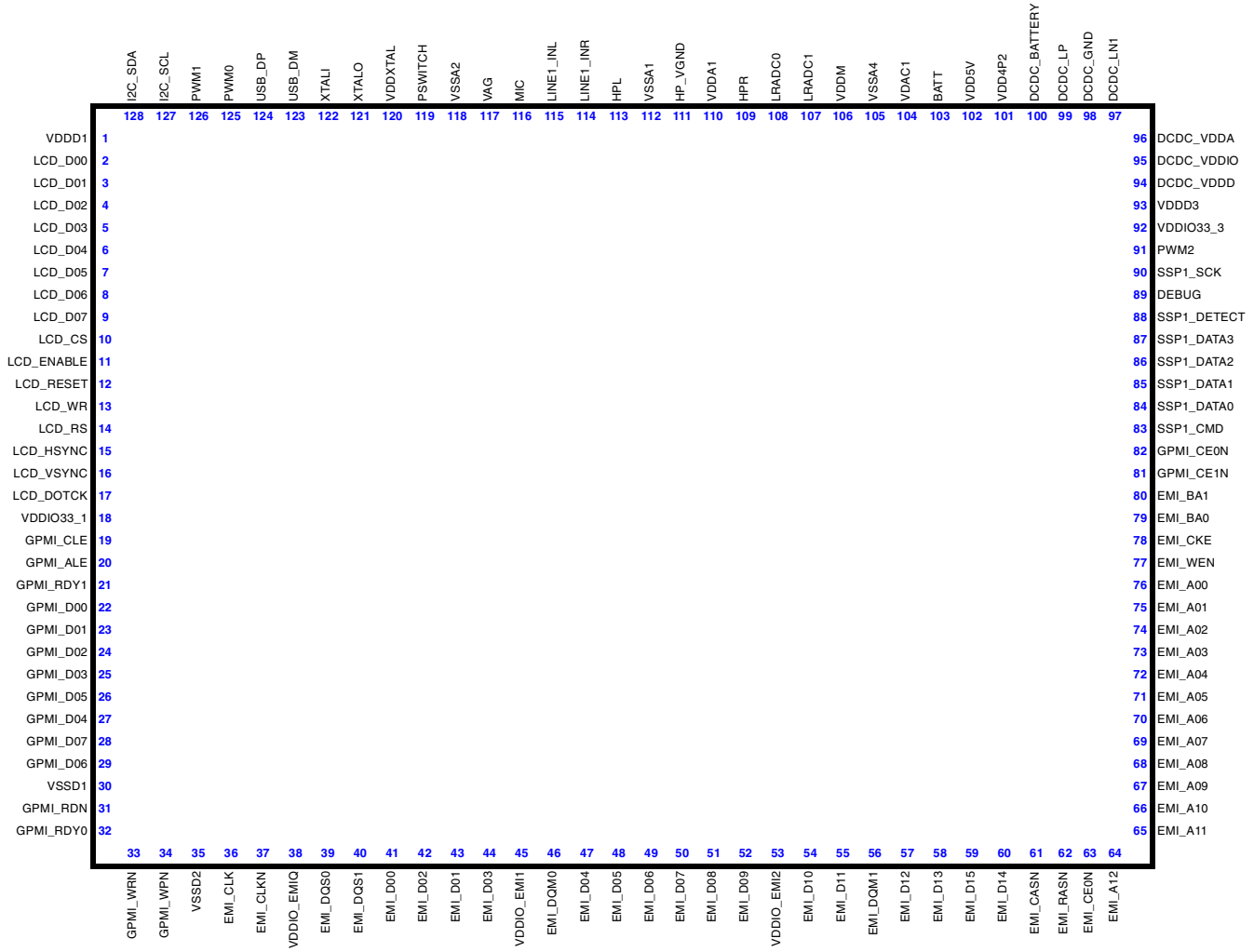
Table 36-3. 128-Pin LQFP Pin Definitions by Pin Number (continued)

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
59	EMI_D15	EMI	DIO	EMI Data 15		
60	EMI_D14	EMI	DIO	EMI Data 14		
61	EMI_CASN	EMI	DIO	EMI Column Address Strobe#		
62	EMI_RASN	EMI	DIO	EMI Row Address Strobe#		
63	EMI_CE0N	EMI	DIO	EMI Chip Enable 0#		
64	EMI_A12	EMI	DIO	EMI Address 12		
65	EMI_A11	EMI	DIO	EMI Address 11		
66	EMI_A10	EMI	DIO	EMI Address 10		
67	EMI_A09	EMI	DIO	EMI Address 9		
68	EMI_A08	EMI	DIO	EMI Address 8		
69	EMI_A07	EMI	DIO	EMI Address 7		
70	EMI_A06	EMI	DIO	EMI Address 6		
71	EMI_A05	EMI	DIO	EMI Address 5		
72	EMI_A04	EMI	DIO	EMI Address 4		
73	EMI_A03	EMI	DIO	EMI Address 3		
74	EMI_A02	EMI	DIO	EMI Address 2		
75	EMI_A01	EMI	DIO	EMI Address 1		
76	EMI_A00	EMI	DIO	EMI Address 0		
77	EMI_WEN	EMI	DIO	EMI Write Enable#		
78	EMI_CKE	EMI	DIO	EMI Clock Enable		
79	EMI_BA0	EMI	DIO	DRAM Bank Address 0		
80	EMI_BA1	EMI	DIO	DRAM Bank Address 1		
81	GPMI_CE1N	GPMI	DIO	GPMI Chip Enable 1#		
82	GPMI_CE0N	GPMI	DIO	GPMI Chip Enable 0#		
83	SSP1_CMD	SSP	DIO	SD-MMC CMD / SPI MOSI		JTAG_TDO
84	SSP1_DATA0	SSP	DIO	SD-MMC Data0 / SPI MISO		JTAG_TDI
85	SSP1_DATA1	SSP	DIO	SD-MMC Data1	I2C_SCL	JTAG_TCLK
86	SSP1_DATA2	SSP	DIO	SD-MMC Data2	I2C_SDA	JTAG_RTCK
87	SSP1_DATA3	SSP	DIO	SD-MMC Data3 / SPI Slave Select 0		JTAG_TMS
88	SSP1_DETECT	SSP	DIO	Removable Card Detect	GPMI_CE3N	USB_ID
89	DEBUG	SYSTEM	I/O	Serial JTAG Debug Port		
90	SSP1_SCK	SSP	DIO	SSP Serial Clock		JTAG_TRST
91	PWM2	PWM	DIO	PWM 2	GPMI_RDY3	
92	VDDIO33_3	POWER	P	Digital I/O 3.3v Power 3		
93	VDDD3	POWER	P	Digital Core Power 3		
94	DCDC_VDDD	DCDC	A	DCDC Digital Core Power		
95	DCDC_VDDIO	DCDC	A	DCDC I/O Power		
96	DCDC_VDDA	DCDC	A	DCDC Analog Power		
97	DCDC_LN1	DCDC	A	DCDC Inductor N 1		
98	DCDC_GND	DCDC	A	DCDC Ground		
99	DCDC_LP	DCDC	A	DCDC Inductor P		
100	DCDC_BATTERY	DCDC	A	DCDC Battery		
101	VDD4P2	DCDC	A	DCDC 4.2V Regulated Output		
102	VDD5V	DCDC	A	5V Power Input		
103	BATT	DCDC	A	Battery Input		
104	VDAC1	VDAC	A	VDAC composite out		

Table 36-3. 128-Pin LQFP Pin Definitions by Pin Number (continued)

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
105	VSSA4	POWER	A	Shared Analog Ground 1c		
106	VDDM	DCDC	A	2.5V Supply for external DDR DRAM	LRADC4	
107	LRADC1	LRADC	A	LRADC1 (Button 2 or Temp or MicBias)		
108	LRADC0	LRADC	A	LRADC0 (Button 1 or Temp or MicBias)		
109	HPR	HP	A	Headphone Right		
110	VDDA1	POWER	A	Analog Power 1		
111	HP_VGND	HP	A	Direct Coupled Headphone Virtual Ground		
112	VSSA1	POWER	A	Analog Ground 1		
113	HPL	HP	A	Headphone Left		
114	LINE1_INR	ADC	A	Line-in 1 Right		
115	LINE1_INL	ADC	A	Line-in 1 Left		
116	MIC	ADC	A	Microphone Input		
117	VAG	HP	A	Analog Reference Capacitor		
118	VSSA2	POWER	A	Analog Ground 2 (quiet)		
119	PSWITCH	DCDC	A	Power On / Recovery		
120	VDDXTAL	CLOCK	A	Crystal Power Filter Cap - Cross bond to side 4		
121	XTALO	CLOCK	A	Crystal Out - 24MHz		
122	XTALI	CLOCK	A	Crystal In - 24MHz		
123	USB_DM	USB	A	USB Negative Data Line		
124	USB_DP	USB	A	USB Positive Data Line		
125	PWM0	PWM	DIO	PWM 0	ROTARYA	DUART RX
126	PWM1	PWM	DIO	PWM 1	ROTARYB	DUART TX
127	I2C_SCL	I2C	DIO	I2C Serial Clock	GPMI_RDY2	AUART1_TX
128	I2C_SDA	I2C	DIO	I2C Serial Data	GPMI_CE2N	AUART1_RX

Table 36-4. 128-Pin LQFP Connection Diagram—Top View



36.2 Pin Definitions for 169-Pin BGA

This section includes the following pin information for the 169-pin BGA package:

- Table 36-5, “169-Pin BGA Pin Definitions by Pin Name,” on page 9
- Table 36-6, “169-Pin BGA Pin Definitions by Pin Number,” on page 13
- Table 36-7, “169-Pin BGA Ball Map,” on page 18

Table 36-5. 169-Pin BGA Pin Definitions by Pin Name

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
G7	AUART1_CTS	UART	DIO	Application UART 1 Clear To Send Flow Control		SSP1_DATA4
H7	AUART1_RTS	UART	DIO	Application UART 1 Ready To Send Flow Control		SSP1_DATA5
H8	AUART1_RX	UART	DIO	Application UART 1 Receive		SSP1_DATA6
J8	AUART1_TX	UART	DIO	Application UART 1 Transmit		SSP1_DATA7

Table 36-5. 169-Pin BGA Pin Definitions by Pin Name (continued)

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
A11	BATT	DCDC	A	Battery Input		
B11	DCDC_BATTERY	DCDC	A	DCDC Battery		
A13	DCDC_GND	DCDC	A	DCDC Ground		
B13	DCDC_LN1	DCDC	A	DCDC Inductor N 1		
A12	DCDC_LP	DCDC	A	DCDC Inductor P		
B12	DCDC_VDDA	DCDC	A	DCDC Analog Power		
D13	DCDC_VDDD	DCDC	A	DCDC Digital Core Power		
C13	DCDC_VDDIO	DCDC	A	DCDC I/O Power		
E12	DEBUG	SYSTEM	DIO	1-Wire Debug Port		
K13	EMI_A00	EMI	DIO	EMI Address 0		
K12	EMI_A01	EMI	DIO	EMI Address 1		
J13	EMI_A02	EMI	DIO	EMI Address 2		
K11	EMI_A03	EMI	DIO	EMI Address 3		
G10	EMI_A04	EMI	DIO	EMI Address 4		
G12	EMI_A05	EMI	DIO	EMI Address 5		
F12	EMI_A06	EMI	DIO	EMI Address 6		
G11	EMI_A07	EMI	DIO	EMI Address 7		
G13	EMI_A08	EMI	DIO	EMI Address 8		
F13	EMI_A09	EMI	DIO	EMI Address 9		
J12	EMI_A10	EMI	DIO	EMI Address 10		
H10	EMI_A11	EMI	DIO	EMI Address 11		
H13	EMI_A12	EMI	DIO	EMI Address 12		
L13	EMI_BA0	EMI	DIO	DRAM Bank Address 0		
L12	EMI_BA1	EMI	DIO	DRAM Bank Address 1		
M13	EMI_CASN	EMI	DIO	EMI Column Address Strobe#		
J10	EMI_CE0N	EMI	DIO	EMI Chip Enable 0#		
H12	EMI_CE1N	EMI	DIO	EMI Chip Enable 1#		
F11	EMI_CKE	EMI	DIO	EMI Clock Enable		
M6	EMI_CLK	EMI	DIO	EMI Clock		
N6	EMI_CLKN	EMI	DIO	EMI Clock#		
K10	EMI_D00	EMI	DIO	EMI Data 0		
M10	EMI_D01	EMI	DIO	EMI Data 1		
N11	EMI_D02	EMI	DIO	EMI Data 2		
N10	EMI_D03	EMI	DIO	EMI Data 3		
M11	EMI_D04	EMI	DIO	EMI Data 4		
K9	EMI_D05	EMI	DIO	EMI Data 5		
L11	EMI_D06	EMI	DIO	EMI Data 6		
L9	EMI_D07	EMI	DIO	EMI Data 7		
N9	EMI_D08	EMI	DIO	EMI Data 8		
N8	EMI_D09	EMI	DIO	EMI Data 9		
M8	EMI_D10	EMI	DIO	EMI Data 10		
K7	EMI_D11	EMI	DIO	EMI Data 11		
L7	EMI_D12	EMI	DIO	EMI Data 12		
K6	EMI_D13	EMI	DIO	EMI Data 13		
N7	EMI_D14	EMI	DIO	EMI Data 14		
M7	EMI_D15	EMI	DIO	EMI Data 15		

Table 36-5. 169-Pin BGA Pin Definitions by Pin Name (continued)

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
M12	EMI_DQM0	EMI	DIO	EMI Data Mask 0		
M9	EMI_DQM1	EMI	DIO	EMI Data Mask1		
N12	EMI_DQS0	EMI	DIO	EMI mDDR-DDR Data Strobe 0 (Low Byte)		
K8	EMI_DQS1	EMI	DIO	EMI mDDR-DDR Data Strobe 1 (High Byte)		
N13	EMI_RASN	EMI	DIO	EMI Row Address Strobe#		
J11	EMI_WEN	EMI	DIO	EMI Write Enable#		
K4	GPMI_ALE	GPMI	DIO	NAND ALE	LCD_D17	
G8	GPMI_CE0N	GPMI	DIO	GPMI Chip Enable 0#		
F8	GPMI_CE1N	GPMI	DIO	GPMI Chip Enable 1#		
K5	GPMI_CE2N	GPMI	DIO	NAND2 Chip Enable#		
K3	GPMI_CLE	GPMI	DIO	NAND CLE	LCD_D16	
K2	GPMI_D00	GPMI	DIO	NAND Data 0	LCD_D8	SSP2_DATA0
L1	GPMI_D01	GPMI	DIO	NAND Data 1	LCD_D9	SSP2_DATA1
L2	GPMI_D02	GPMI	DIO	NAND Data 2	LCD_D10	SSP2_DATA2
L3	GPMI_D03	GPMI	DIO	NAND Data 3	LCD_D11	SSP2_DATA3
L5	GPMI_D04	GPMI	DIO	NAND Data 4	LCD_D12	SSP2_DATA4
L4	GPMI_D05	GPMI	DIO	NAND Data 5	LCD_D13	SSP2_DATA5
M2	GPMI_D06	GPMI	DIO	NAND Data 6	LCD_D14	SSP2_DATA6
M1	GPMI_D07	GPMI	DIO	NAND Data 7	LCD_D15	SSP2_DATA7
M3	GPMI_D08	GPMI	DIO	NAND Data 8	LCD_D18	SSP1_DATA4
M4	GPMI_D09	GPMI	DIO	NAND Data 9	LCD_D19	SSP1_DATA5
M5	GPMI_D10	GPMI	DIO	NAND Data 10	LCD_D20	SSP1_DATA6
N1	GPMI_D11	GPMI	DIO	NAND Data 11	LCD_D21	SSP1_DATA7
N2	GPMI_D12	GPMI	DIO	NAND Data 12	LCD_D22	
N3	GPMI_D13	GPMI	DIO	NAND Data 13	LCD_D23	
N4	GPMI_D14	GPMI	DIO	NAND Data 14	AUART2_RX	
N5	GPMI_D15	GPMI	DIO	NAND Data 15	AUART2_TX	GPMI_CE3N
H6	GPMI_RDN	GPMI	DIO	NAND Read Strobe		
G6	GPMI_RDY0	GPMI	DIO	NAND0 Ready-Busy#		SSP2_DETECT
J6	GPMI_RDY1	GPMI	DIO	NAND1 Ready-Busy#		SSP2_CMD
F6	GPMI_RDY2	GPMI	DIO	NAND2 Ready-Busy#		
E6	GPMI_RDY3	GPMI	DIO	NAND3 Ready-Busy#		
F7	GPMI_WPN	GPMI	DIO	NAND Write Protect		
E7	GPMI_WRN	GPMI	DIO	NAND Write Strobe		SSP2_SCK
A7	HP_VGND	HP	A	Direct Coupled Headphone Virtual Ground		
B7	HPL	HP	A	Headphone Left		
B8	HPR	HP	A	Headphone Right		
C5	I2C_SCL	I2C	DIO	I2C Serial Clock	GPMI_RDY2	AUART1_TX
C6	I2C_SDA	I2C	DIO	I2C Serial Data	GPMI_CE2N	AUART1_RX
H4	LCD_CS	LCDIF	DIO	LCD Interface Chip Select		
D3	LCD_D00	LCDIF	DIO	LCD Interface Data 0	ETM_DA8	
E2	LCD_D01	LCDIF	DIO	LCD Interface Data 1	ETM_DA9	
E4	LCD_D02	LCDIF	DIO	LCD Interface Data 2	ETM_DA10	
F1	LCD_D03	LCDIF	DIO	LCD Interface Data 3	ETM_DA11	
F3	LCD_D04	LCDIF	DIO	LCD Interface Data 4	ETM_DA12	
F5	LCD_D05	LCDIF	DIO	LCD Interface Data 5	ETM_DA13	

Table 36-5. 169-Pin BGA Pin Definitions by Pin Name (continued)

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
G2	LCD_D06	LCDIF	DIO	LCD Interface Data 6	ETM_DA14	
G5	LCD_D07	LCDIF	DIO	LCD Interface Data 7	ETM_DA15	
H2	LCD_D08	LCDIF	DIO	LCD Interface Data 8	ETM_DA0	SAIF2_SDATA0
H3	LCD_D09	LCDIF	DIO	LCD Interface Data 9	ETM_DA1	SAIF1_SDATA0
H1	LCD_D10	LCDIF	DIO	LCD Interface Data 10	ETM_DA2	SAIF_MCLK_BITCLK
G4	LCD_D11	LCDIF	DIO	LCD Interface Data 11	ETM_DA3	SAIF_LRCLK
G1	LCD_D12	LCDIF	DIO	LCD Interface Data 12	ETM_DA4	SAIF2_SDATA1
F4	LCD_D13	LCDIF	DIO	LCD Interface Data 13	ETM_DA5	SAIF2_SDATA2/SAIF_ALT_BITCLK
F2	LCD_D14	LCDIF	DIO	LCD Interface Data 14	ETM_DA6	SAIF1_SDATA2
E5	LCD_D15	LCDIF	DIO	LCD Interface Data 15	ETM_DA7	SAIF1_SDATA1
E3	LCD_D16	LCDIF	DIO	LCD Interface Data 16		SAIF_ALT_BITCLK
E1	LCD_D17	LCDIF	DIO	LCD Interface Data 17		
K1	LCD_DOTCK	LCDIF	DIO	LCD Interface DOT clock	GPMI_RDY3	
H5	LCD_ENABLE	LCDIF	DIO	LCD Interface Enable	I2C_SCL	
J2	LCD_HSYNC	LCDIF	DIO	LCD Interface horizontal sync	I2C_SDA	
J5	LCD_RESET	LCDIF	DIO	LCD Interface Reset Out	ETM_TCTL	GPMI_CE3N
J3	LCD_RS	LCDIF	DIO	LCD Interface Register Select / LCD CCIR clock	ETM_TCLK	
J1	LCD_VSYNC	LCDIF	DIO	LCD Interface Vertical Sync	LCD_BUSY	
J4	LCD_WR	LCDIF	DIO	LCD Interface 6800 E / 8800 WR		
B5	LINE1_INL	ADC	A	Line-in 1 Left		
A6	LINE1_INR	ADC	A	Line-in 1 Right		
A8	LRADC0	LRADC	A	LRADC0 (Button 1 or Temp or MicBias)		
B9	LRADC1	LRADC	A	LRADC1 (Button 2 or Temp or MicBias)		
C9	LRADC2	LRADC	A	LRADC2 (Touchscreen 0)	LINE2_INL	
D9	LRADC3	LRADC	A	LRADC3 (Touchscreen 1)	LINE2_INR	
A10	LRADC4	LRADC	A	LRADC4 (Touchscreen 2)		
B10	LRADC5	LRADC	A	LRADC5 (Touchscreen 3)		
B6	MIC	ADC	A	Microphone Input		
B3	PSWITCH	DCDC	A	Power On / Recovery		
C3	PWM0	PWM	DIO	PWM 0	ROTARYA	DUART RX
D2	PWM1	PWM	DIO	PWM 1	ROTARYB	DUART TX
H11	PWM2	PWM	DIO	PWM 2	GPMI_RDY3	
D11	PWM3	PWM	DIO	PWM 3	ETM_TCTL	AUART1_CTS
C12	PWM4	PWM	DIO	PWM 4	ETM_TCLK	AUART1_RTS
D7	ROTARYA	TIMER	DIO	Rotary Encoder A	AUART2_RTS	SPDIF
E8	ROTARYB	TIMER	DIO	Rotary Encoder B	AUART2_CTS	GPMI_CE3N
B4	RTC_XTALI	CLOCK	A	32.768 KHz Xtal In		
C4	RTC_XTALO	CLOCK	A	32.768 KHz Xtal Out		
C1	SPEAKERP	SPKR	A	Speaker out P		
D1	SPEAKERN	SPKR	A	Speaker out N		
H9	SSP1_CMD	SSP	DIO	SD-MMC CMD / SPI MOSI		JTAG_TDO
G9	SSP1_DATA0	SSP	DIO	SD-MMC Data0 / SPI MISO		JTAG_TDI
F9	SSP1_DATA1	SSP	DIO	SD-MMC Data1	I2C_SCL	JTAG_TCLK
F10	SSP1_DATA2	SSP	DIO	SD-MMC Data2	I2C_SDA	JTAG_RTCK
E10	SSP1_DATA3	SSP	DIO	SD-MMC Data3 / SPI Slave Select 0		JTAG_TMS
D10	SSP1_DETECT	SSP	DIO	Removable Card Detect	GPMI_CE3N	USB_ID

Table 36-5. 169-Pin BGA Pin Definitions by Pin Name (continued)

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
D12	SSP1_SCK	SSP	DIO	SSP Serial Clock		JTAG_TRST
A1	USB_DP	USB	A	USB Negative Data Line		
B2	USB_DM	USB	A	USB Positive Data Line		
A3	VAG	HP	A	Analog Reference Capacitor		
C11	VDAC1	VDAC	A	VDAC composite out		
A9	VDD4P2	DCDC	A	DCDC 4.2V Regulated Output		
E13	VDD5V	DCDC	A	5V Power Input		
C7	VDDA1	POWER	A	Analog Power 1		
D5	VDDD1/3	POWER	P	Digital Core Power 1		
D8	VDDD1/3	POWER	P	Digital Core Power 3		
L8	VDDIO_EMI1/2	POWER	P	Digital I/O Power 1 - EMI		
L10	VDDIO_EMI1/2	POWER	P	Digital I/O Power 2 - EMI		
L6	VDDIO_EMIQ	POWER	P	EMI quiet supply		
G3	VDDIO33_1/2/3	POWER	P	Digital I/O 3.3v Power 1		
E11	VDDIO33_1/2/3	POWER	P	Digital I/O 3.3v Power 3		
C10	VDDM	DCDC	A	2.5V Supply for external DDR DRAM	LRADC4	
B1	VDDS	SPKR	A	Speaker out voltage source		
C2	VDDXTAL	CLOCK	A	Crystal Power Filter Cap - Cross bond to side 4		
C8	VSSA1	LCDIF	A	Analog Ground 1		
A2	VSSA2	LCDIF	A	Analog Ground 2 (quiet)		
D4	VSSA5	POWER	A	Shared Analog Ground 1d (speaker)		
D6	VSSD1/5	POWER	P	Digital Ground 1		
E9	VSSD1/5	POWER	P	Digital Ground 2		
J7	VSSIO_EMI1/2	POWER	P	EMI IO Ground 1		
J9	VSSIO_EMI1/2	POWER	P	EMI IO Ground 2		
A4	XTALI	CLOCK	A	Crystal In - 24MHz		
A5	XTALO	CLOCK	A	Crystal Out - 24MHz		

Table 36-6. 169-Pin BGA Pin Definitions by Pin Number

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
A1	USB_DP	USB	A	USB Negative Data Line		
A2	VSSA2	LCDIF	A	Analog Ground 2 (quiet)		
A3	VAG	HP	A	Analog Reference Capacitor		
A4	XTALI	CLOCK	A	Crystal In - 24MHz		
A5	XTALO	CLOCK	A	Crystal Out - 24MHz		
A6	LINE1_INR	ADC	A	Line-in 1 Right		
A7	HP_VGND	HP	A	Direct Coupled Headphone Virtual Ground		
A8	LRADC0	LRADC	A	LRADC0 (Button 1 or Temp or MicBias)		
A9	VDD4P2	DCDC	A	DCDC 4.2V Regulated Output		
A10	LRADC4	LRADC	A	LRADC4 (Touchscreen 2)		
A11	BATT	DCDC	A	Battery Input		
A12	DCDC_LP	DCDC	A	DCDC Inductor P		
A13	DCDC_GND	DCDC	A	DCDC Ground		
B1	VDDS	SPKR	A	Speaker out voltage source		
B2	USB_DM	USB	A	USB Positive Data Line		

Table 36-6. 169-Pin BGA Pin Definitions by Pin Number (continued)

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
B3	PSWITCH	DCDC	A	Power On / Recovery		
B4	RTC_XTALI	CLOCK	A	32.768 KHz Xtal In		
B5	LINE1_INL	ADC	A	Line-in 1 Left		
B6	MIC	ADC	A	Microphone Input		
B7	HPL	HP	A	Headphone Left		
B8	HPR	HP	A	Headphone Right		
B9	LRADC1	LRADC	A	LRADC1 (Button 2 or Temp or MicBias)		
B10	LRADC5	LRADC	A	LRADC5 (Touchscreen 3)		
B11	DCDC_BATTERY	DCDC	A	DCDC Battery		
B12	DCDC_VDDA	DCDC	A	DCDC Analog Power		
B13	DCDC_LN1	DCDC	A	DCDC Inductor N 1		
C1	SPEAKERP	SPKR	A	Speaker out P		
C2	VDDXTAL	CLOCK	A	Crystal Power Filter Cap - Cross bond to side 4		
C3	PWM0	PWM	DIO	PWM 0	ROTARYA	DUART RX
C4	RTC_XTALO	CLOCK	A	32.768 KHz Xtal Out		
C5	I2C_SCL	I2C	DIO	I2C Serial Clock	GPMI_RDY2	AUART1_TX
C6	I2C_SDA	I2C	DIO	I2C Serial Data	GPMI_CE2N	AUART1_RX
C7	VDDA1	POWER	A	Analog Power 1		
C8	VSSA1	LCDIF	A	Analog Ground 1		
C9	LRADC2	LRADC	A	LRADC2 (Touchscreen 0)	LINE2_INL	
C10	VDDM	DCDC	A	2.5V Supply for external DDR DRAM	LRADC4	
C11	VDAC1	VDAC	A	VDAC composite out		
C12	PWM4	PWM	DIO	PWM 4	ETM_TCLK	AUART1_RTS
C13	DCDC_VDDIO	DCDC	A	DCDC I/O Power		
D1	SPEAKERN	SPKR	A	Speaker out N		
D2	PWM1	PWM	DIO	PWM 1	ROTARYB	DUART TX
D3	LCD_D00	LCDIF	DIO	LCD Interface Data 0	ETM_DA8	
D4	VSSA5	POWER	A	Shared Analog Ground 1d (speaker)		
D5	VDDD1/3	POWER	P	Digital Core Power 1		
D6	VSSD1/5	POWER	P	Digital Ground 1		
D7	ROTARYA	TIMER	DIO	Rotary Encoder A	AUART2_RTS	SPDIF
D8	VDDD1/3	POWER	P	Digital Core Power 3		
D9	LRADC3	LRADC	A	LRADC3 (Touchscreen 1)	LINE2_INR	
D10	SSP1_DETECT	SSP	DIO	Removable Card Detect	GPMI_CE3N	USB_ID
D11	PWM3	PWM	DIO	PWM 3	ETM_TCTL	AUART1_CTS
D12	SSP1_SCK	SSP	DIO	SSP Serial Clock		JTAG_TRST
D13	DCDC_VDDD	DCDC	A	DCDC Digital Core Power		
E1	LCD_D17	LCDIF	DIO	LCD Interface Data 17		
E2	LCD_D01	LCDIF	DIO	LCD Interface Data 1	ETM_DA9	
E3	LCD_D16	LCDIF	DIO	LCD Interface Data 16		SAIF_ALT_BITCLK
E4	LCD_D02	LCDIF	DIO	LCD Interface Data 2	ETM_DA10	
E5	LCD_D15	LCDIF	DIO	LCD Interface Data 15	ETM_DA7	SAIF1_SDATA1
E6	GPMI_RDY3	GPMI	DIO	NAND3 Ready-Busy#		
E7	GPMI_WRN	GPMI	DIO	NAND Write Strobe		SSP2_SCK
E8	ROTARYB	TIMER	DIO	Rotary Encoder B	AUART2_CTS	GPMI_CE3N
E9	VSSD1/5	POWER	P	Digital Ground 2		

Table 36-6. 169-Pin BGA Pin Definitions by Pin Number (continued)

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
E10	SSP1_DATA3	SSP	DIO	SD-MMC Data3 / SPI Slave Select 0		JTAG_TMS
E11	VDDIO33_1/2/3	POWER	P	Digital I/O 3.3v Power 3		
E12	DEBUG	SYSTEM	DIO	1-Wire Debug Port		
E13	VDD5V	DCDC	A	5V Power Input		
F1	LCD_D03	LCDIF	DIO	LCD Interface Data 3	ETM_DA11	
F2	LCD_D14	LCDIF	DIO	LCD Interface Data 14	ETM_DA6	SAIF1_SDATA2
F3	LCD_D04	LCDIF	DIO	LCD Interface Data 4	ETM_DA12	
F4	LCD_D13	LCDIF	DIO	LCD Interface Data 13	ETM_DA5	SAIF2_SDATA2/SAIF_ALT_BITCLK
F5	LCD_D05	LCDIF	DIO	LCD Interface Data 5	ETM_DA13	
F6	GPMI_RDY2	GPMI	DIO	NAND2 Ready-Busy#		
F7	GPMI_WPN	GPMI	DIO	NAND Write Protect		SSP2_SCK
F8	GPMI_CE1N	GPMI	DIO	GPMI Chip Enable 1#		
F9	SSP1_DATA1	SSP	DIO	SD-MMC Data1	I2C_SCL	JTAG_TCLK
F10	SSP1_DATA2	SSP	DIO	SD-MMC Data2	I2C_SDA	JTAG_RTCK
F11	EMI_CKE	EMI	DIO	EMI Clock Enable		
F12	EMI_A06	EMI	DIO	EMI Address 6		
F13	EMI_A09	EMI	DIO	EMI Address 9		
G1	LCD_D12	LCDIF	DIO	LCD Interface Data 12	ETM_DA4	SAIF2_SDATA1
G2	LCD_D06	LCDIF	DIO	LCD Interface Data 6	ETM_DA14	
G3	VDDIO33_1/2/3	POWER	P	Digital I/O 3.3v Power 1		
G4	LCD_D11	LCDIF	DIO	LCD Interface Data 11	ETM_DA3	SAIF_LRCLK
G5	LCD_D07	LCDIF	DIO	LCD Interface Data 7	ETM_DA15	
G6	GPMI_RDY0	GPMI	DIO	NAND0 Ready-Busy#		SSP2_DETECT
G7	AUART1_CTS	UART	DIO	Application UART 1 Clear To Send Flow Control		SSP1_DATA4
G8	GPMI_CE0N	GPMI	DIO	GPMI Chip Enable 0#		
G9	SSP1_DATA0	SSP	DIO	SD-MMC Data0 / SPI MISO		JTAG_TDI
G10	EMI_A04	EMI	DIO	EMI Address 4		
G11	EMI_A07	EMI	DIO	EMI Address 7		
G12	EMI_A05	EMI	DIO	EMI Address 5		
G13	EMI_A08	EMI	DIO	EMI Address 8		
H1	LCD_D10	LCDIF	DIO	LCD Interface Data 10	ETM_DA2	SAIF_MCLK_BITCLK
H2	LCD_D08	LCDIF	DIO	LCD Interface Data 8	ETM_DA0	SAIF2_SDATA0
H3	LCD_D09	LCDIF	DIO	LCD Interface Data 9	ETM_DA1	SAIF1_SDATA0
H4	LCD_CS	LCDIF	DIO	LCD Interface Chip Select		
H5	LCD_ENABLE	LCDIF	DIO	LCD Interface Enable	I2C_SCL	
H6	GPMI_RDN	GPMI	DIO	NAND Read Strobe		SSP2_DETECT
H7	AUART1_RTS	UART	DIO	Application UART 1 Ready To Send Flow Control		SSP1_DATA5
H8	AUART1_RX	UART	DIO	Application UART 1 Receive		SSP1_DATA6
H9	SSP1_CMD	SSP	DIO	SD-MMC CMD / SPI MOSI		JTAG_TDO
H10	EMI_A11	EMI	DIO	EMI Address 11		
H11	PWM2	PWM	DIO	PWM 2	GPMI_RDY3	
H12	EMI_CE1N	EMI	DIO	EMI Chip Enable 1#		
H13	EMI_A12	EMI	DIO	EMI Address 12		
J1	LCD_VSYNC	LCDIF	DIO	LCD Interface Vertical Sync	LCD_BUSY	
J2	LCD_HSYNC	LCDIF	DIO	LCD Interface horizontal sync	I2C_SDA	
J3	LCD_RS	LCDIF	DIO	LCD Interface Register Select / LCD CCIR clock	ETM_TCLK	

Table 36-6. 169-Pin BGA Pin Definitions by Pin Number (continued)

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
J4	LCD_WR	LCDIF	DIO	LCD Interface 6800 E / 8800 WR		
J5	LCD_RESET	LCDIF	DIO	LCD Interface Reset Out	ETM_TCTL	GPMI_CE3N
J6	GPMI_RDY1	GPMI	DIO	NAND1 Ready-Busy#		SSP2_CMD
J7	VSSIO_EMI1/2	POWER	P	EMI IO Ground 1		
J8	AUART1_TX	UART	DIO	Application UART 1 Transmit		SSP1_DATA7
J9	VSSIO_EMI1/2	POWER	P	EMI IO Ground 2		
J10	EMI_CE0N	EMI	DIO	EMI Chip Enable 0#		
J11	EMI_WEN	EMI	DIO	EMI Write Enable#		
J12	EMI_A10	EMI	DIO	EMI Address 10		
J13	EMI_A02	EMI	DIO	EMI Address 2		
K1	LCD_DOTCK	LCDIF	DIO	LCD Interface DOT clock	GPMI_RDY3	
K2	GPMI_D00	GPMI	DIO	NAND Data 0	LCD_D8	SSP2_DATA0
K3	GPMI_CLE	GPMI	DIO	NAND CLE	LCD_D16	
K4	GPMI_ALE	GPMI	DIO	NAND ALE	LCD_D17	
K5	GPMI_CE2N	GPMI	DIO	NAND2 Chip Enable#		
K6	EMI_D13	EMI	DIO	EMI Data 13		
K7	EMI_D11	EMI	DIO	EMI Data 11		
K8	EMI_DQS1	EMI	DIO	EMI mDDR-DDR Data Strobe 1 (High Byte)		
K9	EMI_D05	EMI	DIO	EMI Data 5		
K10	EMI_D00	EMI	DIO	EMI Data 0		
K11	EMI_A03	EMI	DIO	EMI Address 3		
K12	EMI_A01	EMI	DIO	EMI Address 1		
K13	EMI_A00	EMI	DIO	EMI Address 0		
L1	GPMI_D01	GPMI	DIO	NAND Data 1	LCD_D9	SSP2_DATA1
L2	GPMI_D02	GPMI	DIO	NAND Data 2	LCD_D10	SSP2_DATA2
L3	GPMI_D03	GPMI	DIO	NAND Data 3	LCD_D11	SSP2_DATA3
L4	GPMI_D05	GPMI	DIO	NAND Data 5	LCD_D13	SSP2_DATA5
L5	GPMI_D04	GPMI	DIO	NAND Data 4	LCD_D12	SSP2_DATA4
L6	VDDIO_EMIQ	POWER	P	EMI quiet supply		
L7	EMI_D12	EMI	DIO	EMI Data 12		
L8	VDDIO_EMI1/2	POWER	P	Digital I/O Power 1 - EMI		
L9	EMI_D07	EMI	DIO	EMI Data 7		
L10	VDDIO_EMI1/2	POWER	P	Digital I/O Power 2 - EMI		
L11	EMI_D06	EMI	DIO	EMI Data 6		
L12	EMI_BA1	EMI	DIO	DRAM Bank Address 1		
L13	EMI_BA0	EMI	DIO	DRAM Bank Address 0		
M1	GPMI_D07	GPMI	DIO	NAND Data 7	LCD_D15	SSP2_DATA7
M2	GPMI_D06	GPMI	DIO	NAND Data 6	LCD_D14	SSP2_DATA6
M3	GPMI_D08	GPMI	DIO	NAND Data 8	LCD_D18	SSP1_DATA4
M4	GPMI_D09	GPMI	DIO	NAND Data 9	LCD_D19	SSP1_DATA5
M5	GPMI_D10	GPMI	DIO	NAND Data 10	LCD_D20	SSP1_DATA6
M6	EMI_CLK	EMI	DIO	EMI Clock		
M7	EMI_D15	EMI	DIO	EMI Data 15		
M8	EMI_D10	EMI	DIO	EMI Data 10		
M9	EMI_DQM1	EMI	DIO	EMI Data Mask1		
M10	EMI_D01	EMI	DIO	EMI Data 1		

Table 36-6. 169-Pin BGA Pin Definitions by Pin Number (continued)

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
M11	EMI_D04	EMI	DIO	EMI Data 4		
M12	EMI_DQM0	EMI	DIO	EMI Data Mask 0		
M13	EMI_CASN	EMI	DIO	EMI Column Address Strobe#		
N1	GPMI_D11	GPMI	DIO	NAND Data 11	LCD_D21	SSP1_DATA7
N2	GPMI_D12	GPMI	DIO	NAND Data 12	LCD_D22	
N3	GPMI_D13	GPMI	DIO	NAND Data 13	LCD_D23	
N4	GPMI_D14	GPMI	DIO	NAND Data 14	AUART2_RX	
N5	GPMI_D15	GPMI	DIO	NAND Data 15	AUART2_TX	GPMI_CE3N
N6	EMI_CLKN	EMI	DIO	EMI Clock#		
N7	EMI_D14	EMI	DIO	EMI Data 14		
N8	EMI_D09	EMI	DIO	EMI Data 9		
N9	EMI_D08	EMI	DIO	EMI Data 8		
N10	EMI_D03	EMI	DIO	EMI Data 3		
N11	EMI_D02	EMI	DIO	EMI Data 2		
N12	EMI_DQS0	EMI	DIO	EMI mDDR-DDR Data Strobe 0 (Low Byte)		
N13	EMI_RASN	EMI	DIO	EMI Row Address Strobe#		

Table 36-7. 169-Pin BGA Ball Map

	1	2	3	4	5	6	7	8	9	10	11	12	13	
A	USB_DP	VSSA2	VAG	XTALI	XTALO	LINE1_INR	HP_VGND	LRADC0	VDD4P2	LRADC4	BATT	DCDC_LP	DCDC_GND	A
B	VDDS	USB_DM	PSWITCH	RTC_XTALI	LINE1_INL	MIC	HPL	HPR	LRADC1	LRADC5	DCDC_BATTERY	DCDC_VDDA	DCDC_LN1	B
C	SPEAKERP	VDDXTAL	PWM0	RTC_XTALO	I2C_SCL	I2C_SDA	VDDA1	VSSA1	LRADC2	VDDM	VDAC1	PWM4	DCDC_VDDIO	C
D	SPEAKERN	PWM1	LCD_D00	VSSA5	VDDD1/3	VSSD1/5	ROTARYA	VDDD1/3	LRADC3	SSP1_DETECT	PWM3	SSP1_SCK	DCDC_VDDD	D
E	LCD_D17	LCD_D01	LCD_D16	LCD_D02	LCD_D15	GPMI_RDY3	GPMI_WRN	ROTARYB	VSSD1/5	SSP1_DATA3	VDDIO33_1/2/3	DEBUG	VDD5V	E
F	LCD_D03	LCD_D14	LCD_D04	LCD_D13	LCD_D05	GPMI_RDY2	GPMI_WPN	GPMI_CE1N	SSP1_DATA1	SSP1_DATA2	EMI_CKE	EMI_A06	EMI_A09	F
G	LCD_D12	LCD_D06	VDDIO33_1/2/3	LCD_D11	LCD_D07	GPMI_RDY0	AUART1_CTS	GPMI_CE0N	SSP1_DATA0	EMI_A04	EMI_A07	EMI_A05	EMI_A08	G
H	LCD_D10	LCD_D08	LCD_D09	LCD_CS	LCD_ENABLE	GPMI_RDN	AUART1_RTS	AUART1_RX	SSP1_CMD	EMI_A11	PWM2	EMI_CE1N	EMI_A12	H
J	LCD_VSYNC	LCD_HSYNC	LCD_RS	LCD_WR	LCD_RESET	GPMI_RDY1	VSSIO_EMI1/2	AUART1_TX	VSSIO_EMI1/2	EMI_CE0N	EMI_WEN	EMI_A10	EMI_A02	J
K	LCD_DOTCK	GPMI_D00	GPMI_CLE	GPMI_ALE	GPMI_CE2N	EMI_D13	EMI_D11	EMI_DQS1	EMI_D05	EMI_D00	EMI_A03	EMI_A01	EMI_A00	K
L	GPMI_D01	GPMI_D02	GPMI_D03	GPMI_D05	GPMI_D04	VDDIO_EMIQ	EMI_D12	VDDIO_EMI1/2	EMI_D07	VDDIO_EMI1/2	EMI_D06	EMI_BA1	EMI_BA0	L
M	GPMI_D07	GPMI_D06	GPMI_D08	GPMI_D09	GPMI_D10	EMI_CLK	EMI_D15	EMI_D10	EMI_DQM1	EMI_D01	EMI_D04	EMI_DQM0	EMI_CASN	M
N	GPMI_D11	GPMI_D12	GPMI_D13	GPMI_D14	GPMI_D15	EMI_CLKN	EMI_D14	EMI_D09	EMI_D08	EMI_D03	EMI_D02	EMI_DQS0	EMI_RASN	N
	1	2	3	4	5	6	7	8	9	10	11	12	13	

Chapter 37
Pin Control and GPIO

This chapter describes the pin control and general-purpose input/output (GPIO) pin interface implemented on the i.MX23. It includes sections on pin multiplexing and configuration, including color-coded pin multiplexing tables (see Tables 37-1–37-3), followed by a description of the GPIO interface operation. Programmable registers are described in Section 37.4.

37.1 Overview

The pin control interface on the i.MX23 has the following features:

- The i.MX23 has four banks of pins, three of which can serve as GPIOs. The last bank contains the EMI high speed pins which are not muxed with other functions due to tight timing requirements to memory and the need to skew match the timing of the pins.
- All digital pins have selectable output drive strengths as described in Section 37.2.2.1.
- All EMI pins have 1.8/2.5-V and 3.3-V selects as described in Section 37.2.2.1.1.
- All digital pins have weak internal keepers to minimize power loss due to undriven pins.
- The following pin interfaces have selectable pull up resistors:
  - SSP data - 47 kΩ
  - SSP command - 10 kΩ
  - GPMI chip enable - 47 kΩ
  - GPMI ready/busy - 10 kΩ
- All EMI data and DQM pins' internal keepers can be disabled to allow them to change to a high-impedance state (as required by some DRAM manufacturers).
- Slow transitioning pin interfaces contain internal Schmidt triggers for noise immunity.

NOTE

In the context of this chapter, "digital pin" means the standard digital interface pins. This does *not* include the DEBUG pin.

37.2 Operation

Each individual digital pin supporting GPIO operation may be dynamically programmed at any time to be in one of the following states:

- High-impedance (for input, three-state, or open-drain applications)
- Low
- High

- Controlled by one to three selectable on-chip peripheral module interfaces, on a pin by pin basis, as described in Section 37.2.3.

All non-EMI digital pins operate at a fixed nominal voltage of 3.3V. Whereas the EMI pins can be programmed for 1.8/2.5V operation.

Selected pins have pullups that can be configured using register settings. When pullups are enabled, the pin's weak keeper devices are disabled.

Additionally, the state of each pin may be read at any time (no matter how it is configured), and its drive strength may be configured as described in Section 37.2.2.1.

Each GPIO pin may also be used as an interrupt input, and the interrupt trigger type may be configured to be low level-sensitive, high level-sensitive, rising edge-sensitive, or falling edge-sensitive.

For programming purposes, these 120 digital interface pins are divided into four banks of up to 32 pins each. The following sections show how to use all the features of each pin.

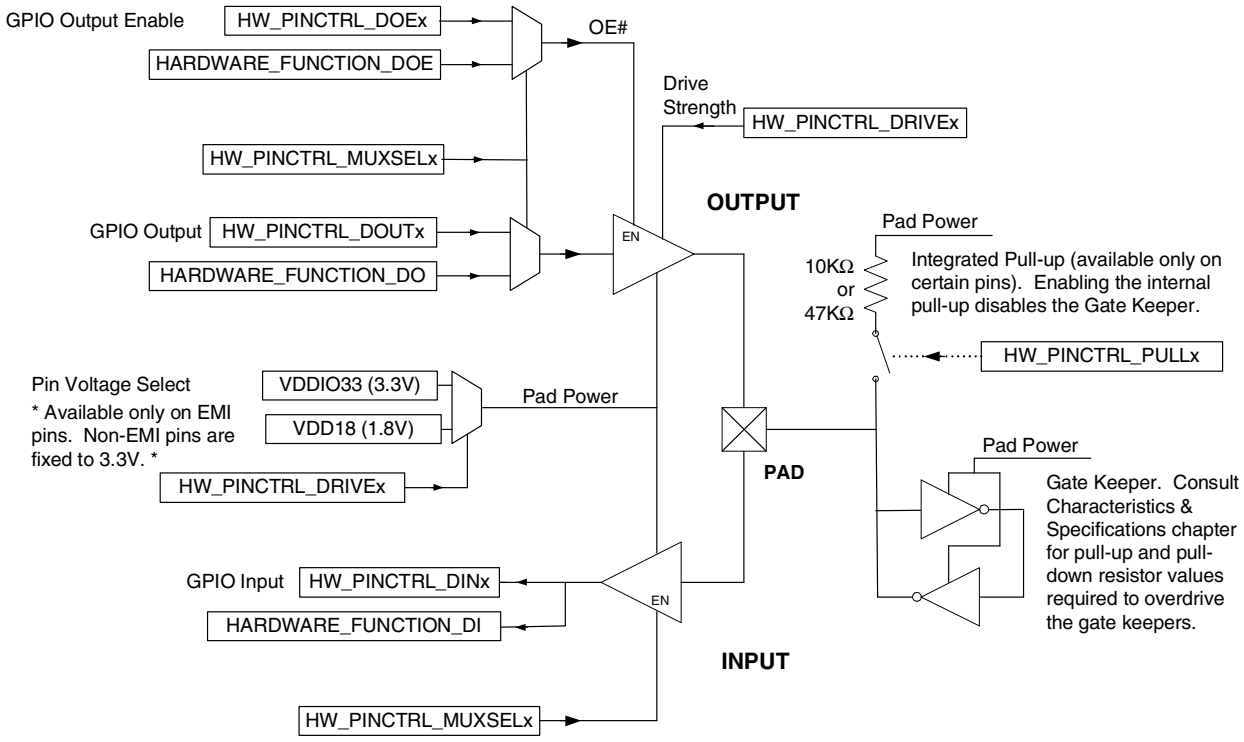


Figure 37-1. Pad Diagram

37.2.1 Reset Configuration

Out of reset, all non-EMI pins (with the exception of those required for boot) are configured as 3.3 V GPIO inputs with gate keepers enabled.

37.2.2 Pin Interface Multiplexing

The i.MX23 is designed for cost sensitive applications. It contains a rich set of specialized hardware interfaces (mDDR, NAND flash, LCD panels, many types of insertable media, etc.), but does not have enough pins to allow use of all signals of all interfaces simultaneously. Consequently, a pin multiplexing scheme is used to allow customers to choose which specialized interfaces to enable for their application. In addition to these specialized hardware interfaces, the i.MX23 allows many digital pins to be used as GPIOs. This capability supports custom interfacing requirements, such as the ability to communicate with LEDs, digital buttons, and other devices that are not directly supported by any of the i.MX23 specialized hardware interfaces.

Each pin is connected to one, two, or three specialized hardware interfaces, in addition to the GPIO function available on banks 0, 1, and 2. The description of each pin contains full details on which specialized hardware interfaces are attached to that pin. For example, the package pin named PWM0 is shared between the PWM, rotary, and debug UART hardware interfaces.

Users define which of the available hardware interfaces controls each pin by writing a two-bit field for that pin into one of the HW_PINCTRL_MUXSELx registers.

Tables 37-1–37-3 illustrate the pin multiplexing on the i.MX23.

- Table 37-1 shows the color mapping used in the tables.
- Table 37-2 shows the multiplexing used in the 169-pin package.
- Table 37-3 shows the multiplexing used in both 128-pin packages.

Table 37-1. Color Mapping for Pin Control Bank Tables

EMI	GPIO	I^2C	JTAG	PWM	SPDIF	Timers and Rotary	Application UART
ETM	GPMI		LCD	SAIF	SSP	Debug UART	USB

Table 37-2. Pin Multiplexing for 169-Pin BGA Package

Bank 0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mux Reg 0	31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
select = 00	gpmi_d15	gpmi_d14	gpmi_d13	gpmi_d12	gpmi_d11	gpmi_d10	gpmi_d9	gpmi_d8	gpmi_d7	gpmi_d6	gpmi_d5	gpmi_d4	gpmi_d3	gpmi_d2	gpmi_d1	gpmi_d0
select = 01	auart2_tx	auart2_rx	lcd_d23	lcd_d22	lcd_d21	lcd_d20	lcd_d19	lcd_d18	lcd_d15	lcd_d14	lcd_d13	lcd_d12	lcd_d11	lcd_d10	lcd_d9	lcd_d8
select = 10	gpmi_ce3n				ssp1_d7	ssp1_d6	ssp1_d5	ssp1_d4	ssp2_d7	ssp2_d6	ssp2_d5	ssp2_d4	ssp2_d3	ssp2_d2	ssp2_d1	ssp2_d0
select = 11	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO

Bank 0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Mux Reg 1	31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
select = 00	i2c_sd	i2c_clk	auart1_tx	auart1_rx	auart1_rts	auart1_cts	gpmi_rdn	gpmi_wrn	gpmi_wpn	gpmi_rb3	gpmi_rb2	gpmi_rb1	gpmi_rb0	gpmi_ce2n	gpmi_ale	gpmi_cle
select = 01	gpmi_ce2n	gpmi_rb2													lcd_d17	lcd_d16
select = 10	auart1_rx	auart1_tx	ssp1_d7	ssp1_d6	ssp1_d5	ssp1_d4		ssp2_sck				ssp2_cmd	ssp2_det			
select = 11	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO

Bank 1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mux Reg 2	31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
select = 00	lcd_d15	lcd_d14	lcd_d13	lcd_d12	lcd_d11	lcd_d10	lcd_d9	lcd_d8	lcd_d7	lcd_d6	lcd_d5	lcd_d4	lcd_d3	lcd_d2	lcd_d1	lcd_d0
select = 01	etm_da7	etm_da5	etm_da6	etm_da4	etm_da3	etm_da2	etm_da1	etm_da0	etm_da15	etm_da14	etm_da13	etm_da12	etm_da11	etm_da10	etm_da9	etm_da8
select = 10	saif1_d1	saif1_d2	saif2_d2	saif2_d1	saif_lrclk	saif_bitclk	saif1_d0	saif2_d0								
select = 11	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO

Bank 1	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Mux Reg 3	31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
select = 00		pwm4	pwm3	pwm2	pwm1	pwm0	lcd_vsync	lcd_hsync	lcd_enable	lcd_dotclk	lcd_cs	lcd_wr	lcd_rs	lcd_reset	lcd_d17	lcd_d16
select = 01		etm_tclk	etm_tctl	gpmi_rb3	timrot2	timrot1	lcd_busy	i2c_sd	i2c_clk	gpmi_rb3			etm_tclk	etm_tctl		
select = 10		auart1_rts	auart1_cts		duart1_tx	duart1_rx								gpmi_ce3n		saif1_alt_bitclk
select = 11		GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO

Table 37-2. Pin Multiplexing for 169-Pin BGA Package

Bank 2	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mux Reg 4	31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
select = 00	emi_a06	emi_a05	emi_a04	emi_a03	emi_a02	emi_a01	emi_a00	timrot2	timrot1	ssp1_sck	ssp1_d3	ssp1_d2	ssp1_d1	ssp1_d0	ssp1_det	ssp1_cmd
select = 01								auart2_cts	auart2_rts			i2c_sd	i2c_clk		gpmi_ce3n	
select = 10								gpmi_ce3n	spdif	jtag_trst_n	jtag_tms	jtag_rtck	jtag_tck	jtag_tdi	usb_id	jtag_tdo
select = 11	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO

Bank 2	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Mux Reg 5	31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
select = 00	emi_wen	emi_rasn	emi_cke	gpmi_ce0n	gpmi_ce1n	emi_ce1n	emi_ce0n	emi_casn	emi_ba1	emi_ba0	emi_a12	emi_a11	emi_a10	emi_a09	emi_a08	emi_a07
select = 01																
select = 10																
select = 11	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO

Bank 3	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mux Reg 6	31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
select = 00	emi_d15	emi_d14	emi_d13	emi_d12	emi_d11	emi_d10	emi_d9	emi_d8	emi_d7	emi_d6	emi_d5	emi_d4	emi_d3	emi_d2	emi_d1	emi_d0
select = 01																
select = 10																
select = 11	disabled	disabled	disabled	disabled	disabled	disabled	disabled	disabled	disabled	disabled	disabled	disabled	disabled	disabled	disabled	disabled

Bank 3	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Mux Reg 7	31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
select = 00											emi_clkn	emi_clk	emi_dqs1	emi_dqs0	emi_dqm1	emi_dqm0
select = 01																
select = 10																
select = 11											disabled	disabled	disabled	disabled	disabled	disabled

Table 37-3. Pin Multiplexing for 128-Pin QFP Packages

Bank 0	15		14		13		12		11		10		9		8		7		6		5		4		3		2		1		0	
Mux Reg 0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
select = 00																	gpmi_data07		gpmi_data06		gpmi_data05		gpmi_data04		gpmi_data03		gpmi_data02		gpmi_data01		gpmi_data00	
select = 01																	lcd_d15		lcd_d14		lcd_d13		lcd_d12		lcd_d11		lcd_d10		lcd_d9		lcd_d8	
select = 10																	ssp2_d7		ssp2_d6		ssp2_d5		ssp2_d4		ssp2_d3		ssp2_d2		ssp2_d1		ssp2_d0	
select = 11																	GPIO		GPIO		GPIO		GPIO		GPIO		GPIO		GPIO		GPIO	

Bank 0	31		30		29		28		27		26		25		24		23		22		21		20		19		18		17		16	
Mux Reg 1	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
select = 00	i2c_sd		i2c_clk										gpmi_rdn		gpmi_wrn		gpmi_wpn						gpmi_rb1		gpmi_rb0				gpmi_ale		gpmi_cle	
select = 01	gpmi_ce2n		gpmi_rb2																										lcd_d17		lcd_d16	
select = 10	auart1_rx		auart1_tx												ssp2_sck								ssp2_cmd		ssp2_det							
select = 11	GPIO		GPIO										GPIO		GPIO		GPIO						GPIO		GPIO				GPIO		GPIO	

Bank 1	15		14		13		12		11		10		9		8		7		6		5		4		3		2		1		0	
Mux Reg 2	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
select = 00																	lcd_d7		lcd_d6		lcd_d5		lcd_d4		lcd_d3		lcd_d2		lcd_d1		lcd_d0	
select = 01																																
select = 10																																
select = 11																	GPIO		GPIO		GPIO		GPIO		GPIO		GPIO		GPIO		GPIO	

Bank 1	31		30		29		28		27		26		25		24		23		22		21		20		19		18		17		16	
Mux Reg 3	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
select = 00							pwm2		pwm1		pwm0		lcd_vsync		lcd_hsync		lcd_enable		lcd_dotclk		lcd_cs		lcd_wr		lcd_rs		lcd_reset					
select = 01							gpmi_rb3		timrot2		timrot1		lcd_busy		i2c_sd		i2c_clk		gpmi_rb3													
select = 10									duart1_tx		duart1_rx																gpmi_ce3n					
select = 11							GPIO		GPIO		GPIO		GPIO		GPIO		GPIO		GPIO		GPIO		GPIO		GPIO		GPIO					

Table 37-3. Pin Multiplexing for 128-Pin QFP Packages

Bank 2	15		14		13		12		11		10		9		8		7		6		5		4		3		2		1		0	
Mux Reg 4	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
select = 00	emi_a06		emi_a05		emi_a04		emi_a03		emi_a02		emi_a01		emi_a00						ssp1_sck		ssp1_d3		ssp1_d2		ssp1_d1		ssp1_d0		ssp1_det		ssp1_cmd	
select = 01																							i2c_sd		i2c_clk				gpmi_ce3n			
select = 10																			jtag_trst_n		jtag_tms		jtag_rtck		jtag_tck		jtag_tdi		usb_id		jtag_tdo	
select = 11	GPIO		GPIO		GPIO		GPIO		GPIO		GPIO		GPIO						GPIO		GPIO		GPIO		GPIO		GPIO		GPIO		GPIO	

Bank 2	31		30		29		28		27		26		25		24		23		22		21		20		19		18		17		16	
Mux Reg 5	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
select = 00	emi_wen		emi_rasn		emi_cke		gpmi_ce0n		gpmi_ce1n				emi_ce0n		emi_casn		emi_ba1		emi_ba0		emi_a12		emi_a11		emi_a10		emi_a09		emi_a08		emi_a07	
select = 01																																
select = 10																																
select = 11	GPIO		GPIO		GPIO		GPIO		GPIO				GPIO		GPIO		GPIO		GPIO				GPIO		GPIO		GPIO		GPIO		GPIO	

Bank 3	15		14		13		12		11		10		9		8		7		6		5		4		3		2		1		0	
Mux Reg 6	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
select = 00	emi_d15		emi_d14		emi_d13		emi_d12		emi_d11		emi_d10		emi_d9		emi_d8		emi_d7		emi_d6		emi_d5		emi_d4		emi_d3		emi_d2		emi_d1		emi_d0	
select = 01																																
select = 10																																
select = 11	disabled		disabled		disabled		disabled		disabled		disabled		disabled		disabled		disabled		disabled		disabled		disabled		disabled		disabled		disabled		disabled	

Bank 3	31		30		29		28		27		26		25		24		23		22		21		20		19		18		17		16	
Mux Reg 7	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
select = 00																					emi_clkn		emi_clk		emi_dqs1		emi_dqs0		emi_dqm1		emi_dqm0	
select = 01																																
select = 10																																
select = 11																					disabled		disabled		disabled		disabled		disabled		disabled	

Readback registers are never affected by the operation of the HW_PINCTRL_MUXSELx registers and always sense the actual value on the data pin.

For example, if a pin is programmed to be a GPIO output and then driven high, any specialized hardware interfaces that are actively monitoring that pin will read the high logic value. Conversely, if the pin mux is programmed to give a specialized hardware interface such as the EMI block control of a particular pin, the current state of that pin can be read through its GPIO read register at any time, even while active EMI cycles are in progress.

Because the pin mux configuration is independent for each individual pin, many pins not required for a given active interface can be reused as GPIO pins. For example, the LCD_RESET pin can be configured and controlled as a GPIO pin, while the other LCD interface pins are controlled by the LCD block.

37.2.2.1 Pin Drive Strength Selection

The drive strength for each digital pin can be programmed by setting the bit corresponding to that pin in one of the HW_PINCTRL_DRIVEx registers. All digital pins have selectable output drive strengths of 4, 8, and 12 mA, with the following exceptions:

- PWM4 has 8, 16, and 24 mA drive strengths.
- EMI_D[15:0], EMI_DQS[1:0], EMI_DQM[1:0], EMI_CLK/N have 4, 8, 12, and 16 mA drive strengths.

Note: *The HW_PINCTRL_DRIVEx registers must be configured prior to operation of the pins and cannot be changed mid-course during active operation. Drive-strength options are provided to optimize simultaneous switching output (SSO) noise. The majority of GPIO pins must be programmed in 4 mA mode. For EMI pins, the weakest mode should be used as long as timing is met.*

Note: *It is recommended that the drive strength of GPMI_RDn and GPMI_WRn output pins be set to 8 mA. This will reduce the transition time under heavy loads. Low transition times will be important when NAND interface read and write cycle times are below 30 ns. The other GPMI pins may remain at 4 mA, since their frequency is only up to half that of GPMI_RDn and GPMI_WRn.*

37.2.2.1.1 Pin Voltage Selection

Each EMI pin can be programmed to operate at either 1.8/2.5 V or 3.3 V by setting the bit corresponding to that pin in one of the HW_PINCTRL_DRIVEx registers.

Note: The I/O pad driver has two PMOS pullup drivers directly connected to a 1.8/2.5 V or 3.3 V power supply. Drive voltage selection for i.MX23 EMI pins is handled in the chip by switching N-well circuits. When the I/O driver is configured in 1.8/2.5 V mode, it is not 3.3 V signal-compatible and will cause a DC current flow if the input is driven by a 3.3 V signal.

37.2.2.2 Pullup/Pulldown Selection

Several digital pins can be programmed to enable pullups by setting the appropriate bit in one of the HW_PINCTRL_PULLx registers. Note that enabling the pullup will also disable the internal gate keeper on that pin.

The pullups are tied to the physical pin pad and not the function. So, for example, if the AUART1_TX pullup is enabled, that pullup will be present on the AUART1_TX pin regardless of what function (AUART1_TX, IR_TX, or SSP1_DATA7) is pin multiplexed out of that pin.

The table below lists which function (not which pin name) that an internal pullup has implemented, to assist in the hardware interface operation.

Table 37-4. i.MX23 Functions with Pullup Resistors

FUNCTION	TYPE	VALUE	PRESENT ON 169BGA	PRESENT ON 128QFP
SSP1_DATA0	Pullup	47K	Y	Y
SSP1_DATA1	Pullup	47K	Y	Y
SSP1_DATA2	Pullup	47K	Y	Y
SSP1_DATA3	Pullup	47K	Y	Y
SSP1_DATA4	Pullup	47K	Y	N
SSP1_DATA5	Pullup	47K	Y	N
SSP1_DATA6	Pullup	47K	Y	N
SSP1_DATA7	Pullup	47K	Y	N
SSP1_CMD	Pullup	10K	Y	Y
SSP2_DATA0	Pullup	47K	Y	Y
SSP2_DATA1	Pullup	47K	Y	Y
SSP2_DATA2	Pullup	47K	Y	Y
SSP2_DATA3	Pullup	47K	Y	Y
SSP2_DATA4	Pullup	47K	Y	Y
SSP2_DATA5	Pullup	47K	Y	Y
SSP2_DATA6	Pullup	47K	Y	Y
SSP2_DATA7	Pullup	47K	Y	Y
SSP2_CMD	Pullup	10K	Y	Y
GPMI_CE0N	Pullup	47K	Y	Y
GPMI_CE1N	Pullup	47K	Y	Y
GPMI_CE2N	Pullup	47K	Y	Y
GPMI_CE3N	Pullup	47K	Y	Y
GPMI_RDY0	Pullup	10K	Y	Y
GPMI_RDY1	Pullup	10K	Y	Y
GPMI_RDY2	Pullup	10K	Y	Y
GPMI_RDY3	Pullup	10K	Y	Y

37.2.3 GPIO Interface

The registers discussed in the following sections exist within each of the three GPIO banks to configure the chip's digital pins. Some pins exist in the 169-pin package only. The registers that control those pins exist but perform no useful function when in a 128-pin package.

37.2.3.1 Output Operation

Programming and controlling a digital pin as a GPIO output is accomplished by programming the appropriate bits in four registers, as shown in Figure 37-2.

- After setting the field in the HW_PINCTRL_MUXSELx to program for GPIO control, the HW_PINCTRL_DRIVEx register bit is set for the desired drive strength and pin voltage. Set bits in HW_PINCTRL_PULLx as required to enable pullups.
- The HW_PINCTRL_DOUTx register bit is then loaded with the level that will initially be driven on the pin.
- Finally, the HW_PINCTRL_DOEx register bit is set.
- Once set, the logic value the HW_PINCTRL_DOUTx bit will be driven on the pin and the value can be toggled with repeated writes.

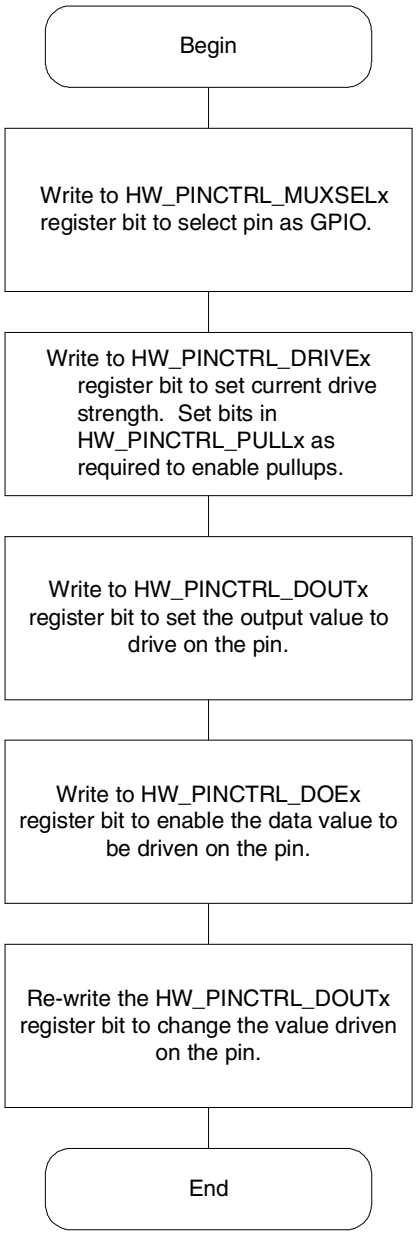


Figure 37-2. GPIO Output Setup Flowchart

37.2.3.2 Input Operation

Any (non-EMI high speed) digital pin may be used as a GPIO input by programming its HW_PINCTRL_MUXSELx field to 3 to enable GPIO mode, programming its HW_PINCTRL_DOEx field to 0 to disable output, and then reading from the HW_PINCTRL_DINx register, as shown in Figure 37-3. Note that because of clock synchronization issues, the logic levels read from the HW_PINCTRL_DINx registers are delayed from the pins by two APBX clock cycles.

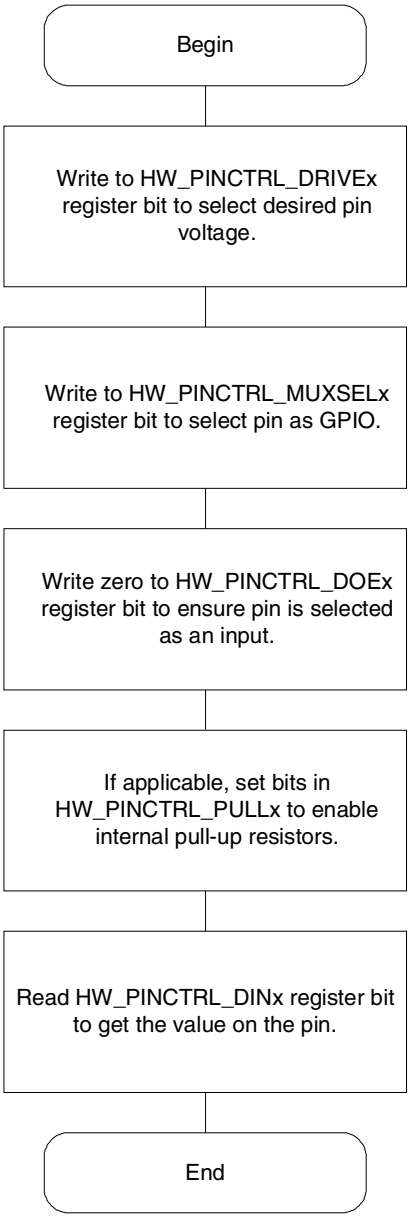


Figure 37-3. GPIO Input Setup Flowchart

37.2.3.3 Input Interrupt Operation

Programming and controlling a digital pin as a GPIO interrupt input is accomplished by programming the appropriate bits in six registers, as shown in Figure 37-4.

- After setting the HW_PINCTRL_MUXSELx register for GPIO, the HW_PINCTRL_IRQLEVELx and HW_PINCTRL_IRQPOLx registers set the interrupt trigger mode. A GPIO interrupt pin can be programmed in one of four trigger detect modes: positive edge, negative edge, positive level, and negative level triggered.

- The HW_PINCTRL_IRQSTATx register bit should then be cleared to ensure that there are no interrupts pending when enabled.
- Setting the HW_PINCTRL_PIN2IRQx register bit will then set up the pin to be an interrupt pin.
- At this point, if an interrupt event occurs on the pin, it will be sensed and recorded in the appropriate HW_PINCTRL_IRQSTATx bit.
- However, the interrupt will not be communicated back to the interrupt collector until the HW_PINCTRL_IRQENx register bit is enabled.

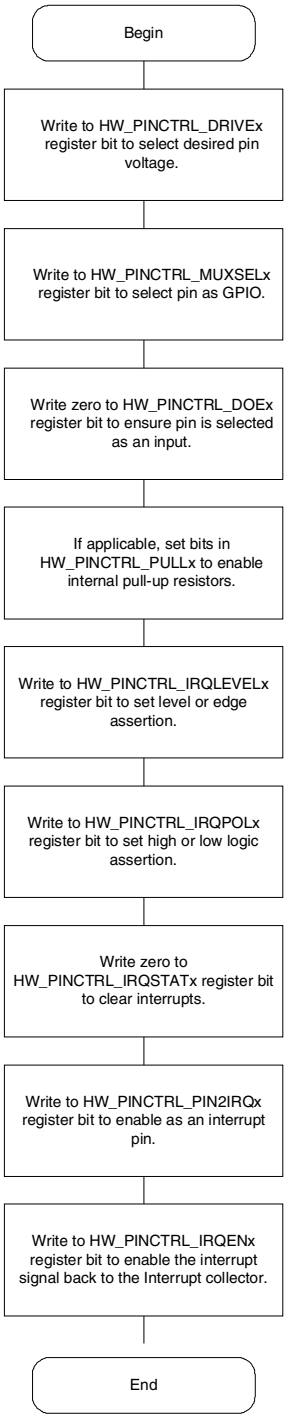


Figure 37-4. GPIO Interrupt Flowchart

Figure 37-5 shows the logic diagram for the interrupt-generation circuit.

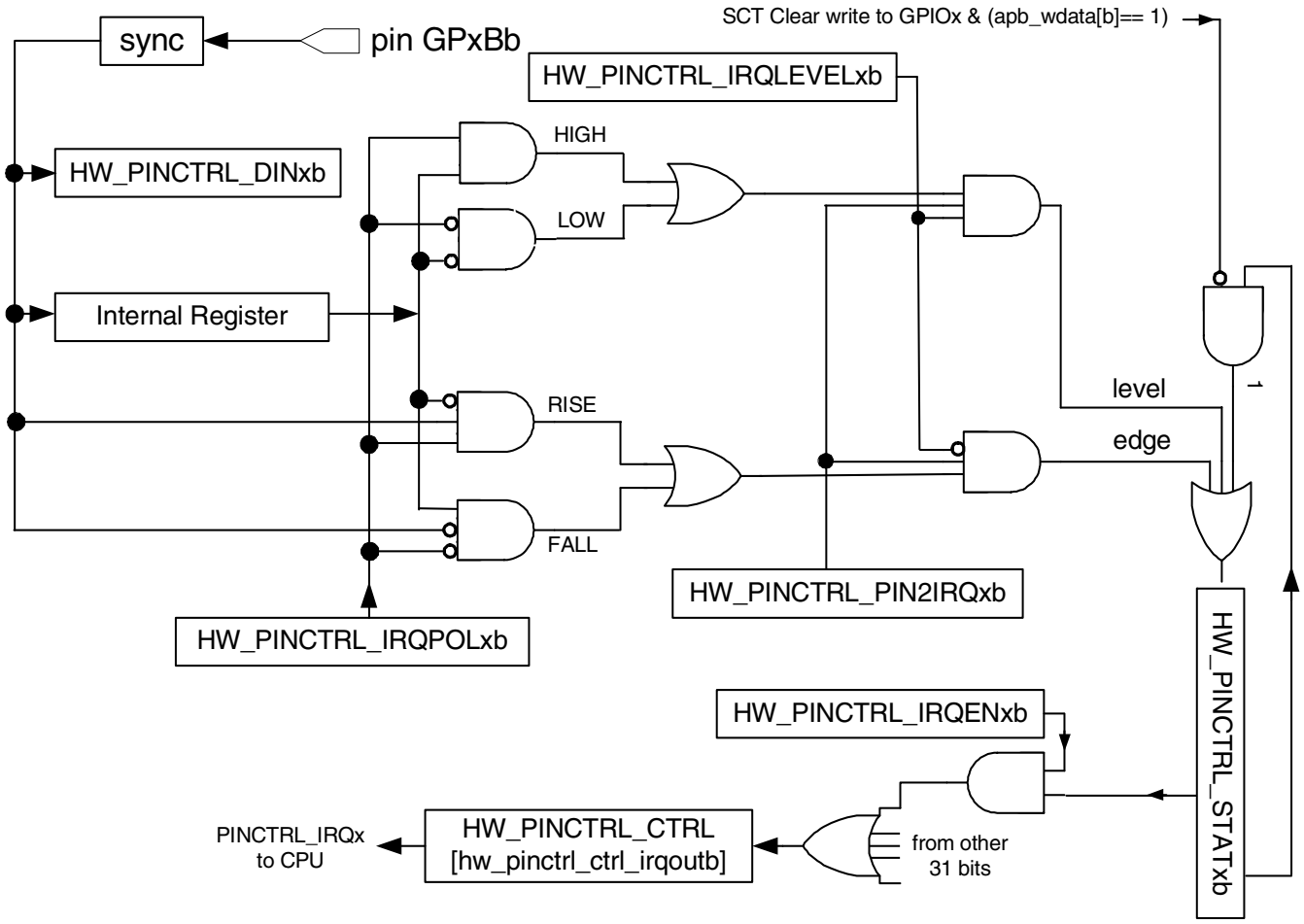


Figure 37-5. GPIO Interrupt Generation

37.3 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10 “Correct Way to Soft Reset a Block” on page 39-7, for additional information on using the SFTRST and CLKGATE bit fields.

37.4 Programmable Registers

The following programmable registers are available for controlling the pin control and GPIO interface of the i.MX23.

37.4.1 PINCTRL Block Control Register Description

The PINCTRL Block Control Register contains the block control bits and combined interrupt output status for each PINCTRL bank.

HW_PINCTRL_CTRL	0x000
HW_PINCTRL_CTRL_SET	0x004
HW_PINCTRL_CTRL_CLR	0x008
HW_PINCTRL_CTRL_TOG	0x00C

Table 37-5. HW_PINCTRL_CTRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
SFTRST	CLKGATE	RSRVD2		PRESENT3	PRESENT2	PRESENT1	PRESENT0	RSRVD1																					IRQOUT2	IRQOUT1	IRQOUT0

Table 37-6. HW_PINCTRL_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**SFTRST**	RW	0x1	This bit must be set to zero to enable operation of any of the PINCTRL banks. When set to one, it forces a block-level reset.
30	**CLKGATE**	RW	0x1	This bit must be set to zero for normal operation. When set to one, it disables the block clock.
29:28	**RSRVD2**	RO	0x0	Always write zeroes to this field.
27	**PRESENT3**	RO	0x1	GPIO Functionality Present. 0: GPIO functionality for Pin Control Bank 3 is not present in this product. 1: GPIO functionality for Bank 3 is present.
26	**PRESENT2**	RO	0x1	GPIO Functionality Present. 0: GPIO functionality for Pin Control Bank 2 is not present in this product. 1: GPIO functionality for Bank 2 is present.
25	**PRESENT1**	RO	0x1	GPIO Functionality Present. 0: GPIO functionality for Pin Control Bank 1 is not present in this product. 1: GPIO functionality for Bank 1 is present.
24	**PRESENT0**	RO	0x1	GPIO Functionality Present. 0: GPIO functionality for Pin Control Bank 0 is not present in this product. 1: GPIO functionality for Bank 0 is present.
23:3	**RSRVD1**	RO	0x00000	Always write zeroes to this field.
2	**IRQOUT2**	RO	0x0	Read-only view of the interrupt collector GPIO2 signal, sourced from the combined IRQ outputs from bank 2.
1	**IRQOUT1**	RO	0x0	Read-only view of the interrupt collector GPIO1 signal, sourced from the combined IRQ outputs from bank 1.
0	**IRQOUT0**	RO	0x0	Read-only view of the interrupt collector GPIO0 signal, sourced from the combined IRQ outputs from bank 0.

DESCRIPTION:

This register contains block-wide control bits and combined bank interrupt status bits. For normal operation, write a 0x00000000 into this register.

EXAMPLE:

```
Empty Example.
```

37.4.2 PINCTRL Pin Mux Select Register 0 Description

The PINCTRL Pin Mux Select Register provides pin function selection for 16 pins in bank0.

HW_PINCTRL_MUXSEL0	0x100
HW_PINCTRL_MUXSEL0_SET	0x104
HW_PINCTRL_MUXSEL0_CLR	0x108
HW_PINCTRL_MUXSEL0_TOG	0x10C

Table 37-7. HW_PINCTRL_MUXSEL0

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	09 08	07 06	05 04	03 02	01 00
BANK0_PIN15	BANK0_PIN14	BANK0_PIN13	BANK0_PIN12	BANK0_PIN11	BANK0_PIN10	BANK0_PIN09	BANK0_PIN08	BANK0_PIN07	BANK0_PIN06	BANK0_PIN05	BANK0_PIN04	BANK0_PIN03	BANK0_PIN02	BANK0_PIN01	BANK0_PIN00

Table 37-8. HW_PINCTRL_MUXSEL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**BANK0_PIN15**	RW	0x3	Pin 59, GPMI_D15 pin function selection: 00= gpmi_data15; 01= auart2_tx; 10= gpmi_ce3n; 11= GPIO.
29:28	**BANK0_PIN14**	RW	0x3	Pin 58, GPMI_D14 pin function selection: 00= gpmi_data14; 01= auart2_rx; 10= reserved; 11= GPIO.
27:26	**BANK0_PIN13**	RW	0x3	Pin 57, GPMI_D13 pin function selection: 00= gpmi_data13; 01= lcd_d23; 10= reserved; 11= GPIO.
25:24	**BANK0_PIN12**	RW	0x3	Pin 56, GPMI_D12 pin function selection: 00= gpmi_data12; 01= lcd_d22; 10= reserved; 11= GPIO.
23:22	**BANK0_PIN11**	RW	0x3	Pin 55, GPMI_D11 pin function selection: 00= gpmi_data11; 01= lcd_d21; 10= ssp1_d7; 11= GPIO.

Table 37-8. HW_PINCTRL_MUXSEL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
21:20	**BANK0_PIN10**	RW	0x3	Pin 54, GPMI_D10 pin function selection: 00= gpmi_data10; 01= lcd_d20; 10= ssp1_d6; 11= GPIO.
19:18	**BANK0_PIN09**	RW	0x3	Pin 53, GPMI_D09 pin function selection: 00= gpmi_data09; 01= lcd_d19; 10= ssp1_d5; 11= GPIO.
17:16	**BANK0_PIN08**	RW	0x3	Pin 52, GPMI_D08 pin function selection: 00= gpmi_data08; 01= lcd_d18; 10= ssp1_d4; 11= GPIO.
15:14	**BANK0_PIN07**	RW	0x3	Pin 50, GPMI_D07 pin function selection: 00= gpmi_data07; 01= lcd_d15; 10= ssp2_d7; 11= GPIO.
13:12	**BANK0_PIN06**	RW	0x3	Pin 51, GPMI_D06 pin function selection: 00= gpmi_data06; 01= lcd_d14; 10= ssp2_d6; 11= GPIO.
11:10	**BANK0_PIN05**	RW	0x3	Pin 48, GPMI_D05 pin function selection: 00= gpmi_data05; 01= lcd_d13; 10= ssp2_d5; 11= GPIO.
9:8	**BANK0_PIN04**	RW	0x3	Pin 49, GPMI_D04 pin function selection: 00= gpmi_data04; 01= lcd_d12; 10= ssp2_d4; 11= GPIO.
7:6	**BANK0_PIN03**	RW	0x3	Pin 47, GPMI_D03 pin function selection: 00= gpmi_data03; 01= lcd_d11; 10= ssp2_d3; 11= GPIO.
5:4	**BANK0_PIN02**	RW	0x3	Pin 46, GPMI_D02 pin function selection: 00= gpmi_data02; 01= lcd_d10; 10= ssp2_d2; 11= GPIO.

Table 37-8. HW_PINCTRL_MUXSEL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
3:2	**BANK0_PIN01**	RW	0x3	Pin 45, GPMI_D01 pin function selection: 00= gpmi_data01; 01= lcd_d9; 10= ssp2_d1; 11= GPIO.
1:0	**BANK0_PIN00**	RW	0x3	Pin 44, GPMI_D00 pin function selection: 00= gpmi_data00; 01= lcd_d8; 10= ssp2_d0; 11= GPIO.

DESCRIPTION:

This register allows the programmer to select which hardware interface blocks drive the 16 pins shown above.

EXAMPLE:

```
Empty Example.
```

37.4.3 PINCTRL Pin Mux Select Register 1 Description

The PINCTRL Pin Mux Select Register provides pin function selection for 16 pins in bank0.

HW_PINCTRL_MUXSEL1	0x110
HW_PINCTRL_MUXSEL1_SET	0x114
HW_PINCTRL_MUXSEL1_CLR	0x118
HW_PINCTRL_MUXSEL1_TOG	0x11C

Table 37-9. HW_PINCTRL_MUXSEL1

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	09 08	07 06	05 04	03 02	01 00
BANK0_PIN31	BANK0_PIN30	BANK0_PIN29	BANK0_PIN28	BANK0_PIN27	BANK0_PIN26	BANK0_PIN25	BANK0_PIN24	BANK0_PIN23	BANK0_PIN22	BANK0_PIN21	BANK0_PIN20	BANK0_PIN19	BANK0_PIN18	BANK0_PIN17	BANK0_PIN16

Table 37-10. HW_PINCTRL_MUXSEL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**BANK0_PIN31**	RW	0x3	Pin 4, I2C_SDA pin function selection: 00= i2c_sd; 01= gpmi_ce2n; 10= auart1_rx; 11= GPIO.
29:28	**BANK0_PIN30**	RW	0x3	Pin 2, I2C_SCL pin function selection: 00= i2c_clk; 01= gpmi_ready2; 10= auart1_tx; 11= GPIO.

Table 37-10. HW_PINCTRL_MUXSEL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
27:26	**BANK0_PIN29**	RW	0x3	Pin 69, AUART1_TX pin function selection: 00= auart1_tx; 01= reserved; 10= ssp1_d7; 11= GPIO.
25:24	**BANK0_PIN28**	RW	0x3	Pin 68, AUART1_RX pin function selection: 00= auart1_rx; 01= reserved; 10= ssp1_d6; 11= GPIO.
23:22	**BANK0_PIN27**	RW	0x3	Pin 67, AUART1_RTS pin function selection: 00= auart1_rts; 01= reserved; 10= ssp1_d5; 11= GPIO.
21:20	**BANK0_PIN26**	RW	0x3	Pin 66, AUART1_CTS pin function selection: 00= auart1_cts; 01= reserved; 10= ssp1_d4; 11= GPIO.
19:18	**BANK0_PIN25**	RW	0x3	Pin 60, GPMI_RDN pin function selection: 00= gpmi_rdn; 01= reserved; 10= reserved; 11= GPIO.
17:16	**BANK0_PIN24**	RW	0x3	Pin 65, GPMI_WRN pin function selection: 00= gpmi_wrn; 01= reserved; 10= ssp2_sck; 11= GPIO.
15:14	**BANK0_PIN23**	RW	0x3	Pin 64, GPMI_WPN pin function selection: 00= gpmi_wpn; 01= reserved; 10= reserved; 11= GPIO.
13:12	**BANK0_PIN22**	RW	0x3	Pin 63, GPMI_RDY3 pin function selection: 00= gpmi_ready3; 01= reserved; 10= reserved; 11= GPIO.
11:10	**BANK0_PIN21**	RW	0x3	Pin 62, GPMI_RDY2 pin function selection: 00= gpmi_ready2; 01= reserved; 10= reserved; 11= GPIO.
9:8	**BANK0_PIN20**	RW	0x3	Pin 43, GPMI_RDY1 pin function selection: 00= gpmi_ready1; 01= reserved; 10= ssp2_cmd; 11= GPIO.

Table 37-10. HW_PINCTRL_MUXSEL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
7:6	**BANK0_PIN19**	RW	0x3	Pin 61, GPMI_RDY0 pin function selection: 00= gpmi_ready0; 01= reserved; 10= ssp2_detect; 11= GPIO.
5:4	**BANK0_PIN18**	RW	0x3	Pin 42, GPMI_CE2N pin function selection: 00= gpmi_ce2n; 01= reserved; 10= reserved; 11= GPIO.
3:2	**BANK0_PIN17**	RW	0x3	Pin 41, GPMI_ALE pin function selection: 00= gpmi_ale; 01= lcd_d17; 10= reserved; 11= GPIO.
1:0	**BANK0_PIN16**	RW	0x3	Pin 40, GPMI_CLE pin function selection: 00= gpmi_cle; 01= lcd_d16; 10= reserved; 11= GPIO.

DESCRIPTION:

This register allows the programmer to select which hardware interface blocks drive the 16 pins shown above.

EXAMPLE:

```
Empty Example.
```

37.4.4 PINCTRL Pin Mux Select Register 2 Description

The PINCTRL Pin Mux Select Register provides pin function selection for 16 pins in bank1.

HW_PINCTRL_MUXSEL2	0x120
HW_PINCTRL_MUXSEL2_SET	0x124
HW_PINCTRL_MUXSEL2_CLR	0x128
HW_PINCTRL_MUXSEL2_TOG	0x12C

Table 37-11. HW_PINCTRL_MUXSEL2

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	09 08	07 06	05 04	03 02	01 00
BANK1_PIN15	BANK1_PIN14	BANK1_PIN13	BANK1_PIN12	BANK1_PIN11	BANK1_PIN10	BANK1_PIN09	BANK1_PIN08	BANK1_PIN07	BANK1_PIN06	BANK1_PIN05	BANK1_PIN04	BANK1_PIN03	BANK1_PIN02	BANK1_PIN01	BANK1_PIN00

Table 37-12. HW_PINCTRL_MUXSEL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**BANK1_PIN15**	RW	0x3	Pin 15, LCD_D15 pin function selection: 00= lcd_d15; 01= etm_da7; 10= saif1_sdata1; 11= GPIO.
29:28	**BANK1_PIN14**	RW	0x3	Pin 17, LCD_D14 pin function selection: 00= lcd_d14; 01= etm_da6; 10= saif1_sdata2; 11= GPIO.
27:26	**BANK1_PIN13**	RW	0x3	Pin 19, LCD_D13 pin function selection: 00= lcd_d13; 01= etm_da5; 10= saif2_sdata2; 11= GPIO.
25:24	**BANK1_PIN12**	RW	0x3	Pin 22, LCD_D12 pin function selection: 00= lcd_d12; 01= etm_da4; 10= saif2_sdata1; 11= GPIO.
23:22	**BANK1_PIN11**	RW	0x3	Pin 24, LCD_D11 pin function selection: 00= lcd_d11; 01= etm_da3; 10= saif_lrclk; 11= GPIO.
21:20	**BANK1_PIN10**	RW	0x3	Pin 26, LCD_D10 pin function selection: 00= lcd_d10; 01= etm_da2; 10= saif_bitclk; 11= GPIO.
19:18	**BANK1_PIN09**	RW	0x3	Pin 28, LCD_D09 pin function selection: 00= lcd_d9; 01= etm_da1; 10= saif1_sdata0; 11= GPIO.
17:16	**BANK1_PIN08**	RW	0x3	Pin 27, LCD_D08 pin function selection: 00= lcd_d8; 01= etm_da0; 10= saif2_sdata0; 11= GPIO.
15:14	**BANK1_PIN07**	RW	0x3	Pin 25, LCD_D07 pin function selection: 00= lcd_d7; 01= etm_da15; 10= reserved; 11= GPIO.
13:12	**BANK1_PIN06**	RW	0x3	Pin 23, LCD_D06 pin function selection: 00= lcd_d6; 01= etm_da14; 10= reserved; 11= GPIO.

Table 37-12. HW_PINCTRL_MUXSEL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
11:10	**BANK1_PIN05**	RW	0x3	Pin 21, LCD_D05 pin function selection: 00= lcd_d5; 01= etm_da13; 10= reserved; 11= GPIO.
9:8	**BANK1_PIN04**	RW	0x3	Pin 18, LCD_D04 pin function selection: 00= lcd_d4; 01= etm_da12; 10= reserved; 11= GPIO.
7:6	**BANK1_PIN03**	RW	0x3	Pin 16, LCD_D03 pin function selection: 00= lcd_d3; 01= etm_da11; 10= reserved; 11= GPIO.
5:4	**BANK1_PIN02**	RW	0x3	Pin 14, LCD_D02 pin function selection: 00= lcd_d2; 01= etm_da10; 10= reserved; 11= GPIO.
3:2	**BANK1_PIN01**	RW	0x3	Pin 12, LCD_D01 pin function selection: 00= lcd_d1; 01= etm_da9; 10= reserved; 11= GPIO.
1:0	**BANK1_PIN00**	RW	0x3	Pin 10, LCD_D00 pin function selection: 00= lcd_d0; 01= etm_da8; 10= reserved; 11= GPIO.

DESCRIPTION:

This register allows the programmer to select which hardware interface blocks drive the 16 pins shown above.

EXAMPLE:

```
Empty Example.
```

37.4.5 PINCTRL Pin Mux Select Register 3 Description

The PINCTRL Pin Mux Select Register provides pin function selection for 15 pins in bank1.

HW_PINCTRL_MUXSEL3	0x130
HW_PINCTRL_MUXSEL3_SET	0x134
HW_PINCTRL_MUXSEL3_CLR	0x138
HW_PINCTRL_MUXSEL3_TOG	0x13C

Table 37-13. HW_PINCTRL_MUXSEL3

31:30	29:28	27:26	25:24	23:22	21:20	19:18	17:16	15:14	13:12	11:10	09:08	07:06	05:04	03:02	01:00
RSRVD0	BANK1_PIN30	BANK1_PIN29	BANK1_PIN28	BANK1_PIN27	BANK1_PIN26	BANK1_PIN25	BANK1_PIN24	BANK1_PIN23	BANK1_PIN22	BANK1_PIN21	BANK1_PIN20	BANK1_PIN19	BANK1_PIN18	BANK1_PIN17	BANK1_PIN16

Table 37-14. HW_PINCTRL_MUXSEL3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD0**	RO	0x0	Always write zeroes to this field.
29:28	**BANK1_PIN30**	RW	0x3	Pin 131, PWM4 pin function selection: 00= pwm4; 01= etm_tclk; 10= auart1_rts; 11= GPIO.
27:26	**BANK1_PIN29**	RW	0x3	Pin 130, PWM3 pin function selection: 00= pwm3; 01= etm_tctl; 10= auart1_cts; 11= GPIO.
25:24	**BANK1_PIN28**	RW	0x3	Pin 129, PWM2 pin function selection: 00= pwm2; 01= gpmi_ready3; 10= reserved; 11= GPIO.
23:22	**BANK1_PIN27**	RW	0x3	Pin 3, PWM1 pin function selection: 00= pwm1; 01= timrot2; 10= duart_tx; 11= GPIO.
21:20	**BANK1_PIN26**	RW	0x3	Pin 1, PWM0 pin function selection: 00= pwm0; 01= timrot1; 10= duart_rx; 11= GPIO.
19:18	**BANK1_PIN25**	RW	0x3	Pin 35, LCD_VSYNC pin function selection: 00= lcd_vsync; 01= lcd_busy; 10= reserved; 11= GPIO.
17:16	**BANK1_PIN24**	RW	0x3	Pin 34, LCD_HSYNC pin function selection: 00= lcd_hsync; 01= i2c_sd; 10= reserved; 11= GPIO.
15:14	**BANK1_PIN23**	RW	0x3	Pin 30, LCD_ENABLE pin function selection: 00= lcd_enable; 01= i2c_clk; 10= reserved; 11= GPIO.

Table 37-14. HW_PINCTRL_MUXSEL3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
13:12	**BANK1_PIN22**	RW	0x3	Pin 36, LCD_DOTCK pin function selection: 00= lcd_dotck; 01= gpmi_ready3; 10= reserved; 11= GPIO.
11:10	**BANK1_PIN21**	RW	0x3	Pin 29, LCD_CS pin function selection: 00= lcd_cs; 01= reserved; 10= reserved; 11= GPIO.
9:8	**BANK1_PIN20**	RW	0x3	Pin 32, LCD_WR pin function selection: 00= lcd_wr; 01= reserved; 10= reserved; 11= GPIO.
7:6	**BANK1_PIN19**	RW	0x3	Pin 33, LCD_RS pin function selection: 00= lcd_rs; 01= etm_tclk; 10= reserved; 11= GPIO.
5:4	**BANK1_PIN18**	RW	0x3	Pin 31, LCD_RESET pin function selection: 00= lcd_reset; 01= etm_tctl; 10= gpmi_ce3n; 11= GPIO.
3:2	**BANK1_PIN17**	RW	0x3	Pin 11, LCD_D17 pin function selection: 00= lcd_d17; 01= reserved; 10= reserved; 11= GPIO.
1:0	**BANK1_PIN16**	RW	0x3	Pin 13, LCD_D16 pin function selection: 00= lcd_d16; 01= reserved; 10= saif_alt_bitclk; 11= GPIO.

DESCRIPTION:

This register allows the programmer to select which hardware interface blocks drive the 15 pins shown above.

EXAMPLE:

```
Empty Example.
```

37.4.6 PINCTRL Pin Mux Select Register 4 Description

The PINCTRL Pin Mux Select Register provides pin function selection for 16 pins in bank2.

HW_PINCTRL_MUXSEL4	0x140
HW_PINCTRL_MUXSEL4_SET	0x144
HW_PINCTRL_MUXSEL4_CLR	0x148
HW_PINCTRL_MUXSEL4_TOG	0x14C

Table 37-15. HW_PINCTRL_MUXSEL4

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	09 08	07 06	05 04	03 02	01 00
BANK2_PIN15	BANK2_PIN14	BANK2_PIN13	BANK2_PIN12	BANK2_PIN11	BANK2_PIN10	BANK2_PIN09	BANK2_PIN08	BANK2_PIN07	BANK2_PIN06	BANK2_PIN05	BANK2_PIN04	BANK2_PIN03	BANK2_PIN02	BANK2_PIN01	BANK2_PIN00

Table 37-16. HW_PINCTRL_MUXSEL4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**BANK2_PIN15**	RW	0x3	Pin 108, EMI_A06 pin function selection: 00= emi_addr06; 01= reserved; 10= reserved; 11= GPIO.
29:28	**BANK2_PIN14**	RW	0x3	Pin 107, EMI_A05 pin function selection: 00= emi_addr05; 01= reserved; 10= reserved; 11= GPIO.
27:26	**BANK2_PIN13**	RW	0x3	Pin 109, EMI_A04 pin function selection: 00= emi_addr04; 01= reserved; 10= reserved; 11= GPIO.
25:24	**BANK2_PIN12**	RW	0x3	Pin 110, EMI_A03 pin function selection: 00= emi_addr03; 01= reserved; 10= reserved; 11= GPIO.
23:22	**BANK2_PIN11**	RW	0x3	Pin 111, EMI_A02 pin function selection: 00= emi_addr02; 01= reserved; 10= reserved; 11= GPIO.
21:20	**BANK2_PIN10**	RW	0x3	Pin 112, EMI_A01 pin function selection: 00= emi_addr01; 01= reserved; 10= reserved; 11= GPIO.
19:18	**BANK2_PIN09**	RW	0x3	Pin 113, EMI_A00 pin function selection: 00= emi_addr00; 01= reserved; 10= reserved; 11= GPIO.
17:16	**BANK2_PIN08**	RW	0x3	Pin 38, ROTARYB pin function selection: 00= timrot2; 01= auart2_cts; 10= gpmi_ce3n; 11= GPIO.

Table 37-16. HW_PINCTRL_MUXSEL4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
15:14	**BANK2_PIN07**	RW	0x3	Pin 37, ROTARYA pin function selection: 00= timrot1; 01= auart2_rts; 10= spdif; 11= GPIO.
13:12	**BANK2_PIN06**	RW	0x3	Pin 127, SSP1_SCK pin function selection: 00= ssp1_sck; 01= reserved; 10= alt_jtag_trst_n; 11= GPIO.
11:10	**BANK2_PIN05**	RW	0x3	Pin 125, SSP1_DATA3 pin function selection: 00= ssp1_d3; 01= reserved; 10= alt_jtag_tms; 11= GPIO.
9:8	**BANK2_PIN04**	RW	0x3	Pin 124, SSP1_DATA2 pin function selection: 00= ssp1_d2; 01= i2c_sd; 10= alt_jtag_rtck; 11= GPIO.
7:6	**BANK2_PIN03**	RW	0x3	Pin 123, SSP1_DATA1 pin function selection: 00= ssp1_d1; 01= i2c_clk; 10= alt_jtag_tck; 11= GPIO.
5:4	**BANK2_PIN02**	RW	0x3	Pin 122, SSP1_DATA0 pin function selection: 00= ssp1_d0; 01= reserved; 10= alt_jtag_tdi; 11= GPIO.
3:2	**BANK2_PIN01**	RW	0x3	Pin 126, SSP1_DETECT pin function selection: 00= ssp1_detect; 01= gpmi_ce3n; 10= usb_id; 11= GPIO.
1:0	**BANK2_PIN00**	RW	0x3	Pin 121, SSP1_CMD pin function selection: 00= ssp1_cmd; 01= reserved; 10= alt_jtag_tdo; 11= GPIO.

DESCRIPTION:

This register allows the programmer to select which hardware interface blocks drive the 16 pins shown above.

EXAMPLE:

```
Empty Example.
```

37.4.7 PINCTRL Pin Mux Select Register 5 Description

The PINCTRL Pin Mux Select Register provides pin function selection for 16 pins in bank2.

HW_PINCTRL_MUXSEL5	0x150
HW_PINCTRL_MUXSEL5_SET	0x154
HW_PINCTRL_MUXSEL5_CLR	0x158
HW_PINCTRL_MUXSEL5_TOG	0x15C

Table 37-17. HW_PINCTRL_MUXSEL5

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BANK2_PIN31		BANK2_PIN30		BANK2_PIN29		BANK2_PIN28		BANK2_PIN27		BANK2_PIN26		BANK2_PIN25		BANK2_PIN24		BANK2_PIN23		BANK2_PIN22		BANK2_PIN21		BANK2_PIN20		BANK2_PIN19		BANK2_PIN18		BANK2_PIN17		BANK2_PIN16	

Table 37-18. HW_PINCTRL_MUXSEL5 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**BANK2_PIN31**	RW	0x3	Pin 114, EMI_WEN pin function selection: 00= emi_wen; 01= reserved; 10= reserved; 11= GPIO.
29:28	**BANK2_PIN30**	RW	0x3	Pin 98, EMI_RASN pin function selection: 00= emi_rasn; 01= reserved; 10= reserved; 11= GPIO.
27:26	**BANK2_PIN29**	RW	0x3	Pin 115, EMI_CKE pin function selection: 00= emi_cke; 01= reserved; 10= reserved; 11= GPIO.
25:24	**BANK2_PIN28**	RW	0x3	Pin 120, GPMI_CE0N pin function selection: 00= gpmi_ce0n; 01= reserved; 10= reserved; 11= GPIO.
23:22	**BANK2_PIN27**	RW	0x3	Pin 118, GPMI_CE1N pin function selection: 00= gpmi_ce1n; 01= reserved; 10= reserved; 11= GPIO.
21:20	**BANK2_PIN26**	RW	0x3	Pin 99, EMI_CE1N pin function selection: 00= emi_ce1n; 01= reserved; 10= reserved; 11= GPIO.
19:18	**BANK2_PIN25**	RW	0x3	Pin 100, EMI_CE0N pin function selection: 00= emi_ce0n; 01= reserved; 10= reserved; 11= GPIO.

Table 37-18. HW_PINCTRL_MUXSEL5 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
17:16	**BANK2_PIN24**	RW	0x3	Pin 97, EMI_CASN pin function selection: 00= emi_casn; 01= reserved; 10= reserved; 11= GPIO.
15:14	**BANK2_PIN23**	RW	0x3	Pin 117, EMI_BA1 pin function selection: 00= emi_ba1; 01= reserved; 10= reserved; 11= GPIO.
13:12	**BANK2_PIN22**	RW	0x3	Pin 116, EMI_BA0 pin function selection: 00= emi_ba0; 01= reserved; 10= reserved; 11= GPIO.
11:10	**BANK2_PIN21**	RW	0x3	Pin 101, EMI_A12 pin function selection: 00= emi_addr12; 01= reserved; 10= reserved; 11= GPIO.
9:8	**BANK2_PIN20**	RW	0x3	Pin 102, EMI_A11 pin function selection: 00= emi_addr11; 01= reserved; 10= reserved; 11= GPIO.
7:6	**BANK2_PIN19**	RW	0x3	Pin 104, EMI_A10 pin function selection: 00= emi_addr10; 01= reserved; 10= reserved; 11= GPIO.
5:4	**BANK2_PIN18**	RW	0x3	Pin 103, EMI_A09 pin function selection: 00= emi_addr09; 01= reserved; 10= reserved; 11= GPIO.
3:2	**BANK2_PIN17**	RW	0x3	Pin 106, EMI_A08 pin function selection: 00= emi_addr08; 01= reserved; 10= reserved; 11= GPIO.
1:0	**BANK2_PIN16**	RW	0x3	Pin 105, EMI_A07 pin function selection: 00= emi_addr07; 01= reserved; 10= reserved; 11= GPIO.

DESCRIPTION:

This register allows the programmer to select which hardware interface blocks drive the 16 pins shown above.

EXAMPLE:

```
Empty Example.
```

37.4.8 PINCTRL Pin Mux Select Register 6 Description

The PINCTRL Pin Mux Select Register provides pin function selection for 16 pins in bank3.

HW_PINCTRL_MUXSEL6	0x160
HW_PINCTRL_MUXSEL6_SET	0x164
HW_PINCTRL_MUXSEL6_CLR	0x168
HW_PINCTRL_MUXSEL6_TOG	0x16C

Table 37-19. HW_PINCTRL_MUXSEL6

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BANK3_PIN15		BANK3_PIN14		BANK3_PIN13		BANK3_PIN12		BANK3_PIN11		BANK3_PIN10		BANK3_PIN09		BANK3_PIN08		BANK3_PIN07		BANK3_PIN06		BANK3_PIN05		BANK3_PIN04		BANK3_PIN03		BANK3_PIN02		BANK3_PIN01		BANK3_PIN00	

Table 37-20. HW_PINCTRL_MUXSEL6 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**BANK3_PIN15**	RW	0x3	Pin 95, EMI_D15 pin function selection: 00= emi_data15; 01= reserved; 10= reserved; 11= disabled.
29:28	**BANK3_PIN14**	RW	0x3	Pin 96, EMI_D14 pin function selection: 00= emi_data14; 01= reserved; 10= reserved; 11= disabled.
27:26	**BANK3_PIN13**	RW	0x3	Pin 94, EMI_D13 pin function selection: 00= emi_data13; 01= reserved; 10= reserved; 11= disabled.
25:24	**BANK3_PIN12**	RW	0x3	Pin 93, EMI_D12 pin function selection: 00= emi_data12; 01= reserved; 10= reserved; 11= disabled.
23:22	**BANK3_PIN11**	RW	0x3	Pin 91, EMI_D11 pin function selection: 00= emi_data11; 01= reserved; 10= reserved; 11= disabled.
21:20	**BANK3_PIN10**	RW	0x3	Pin 89, EMI_D10 pin function selection: 00= emi_data10; 01= reserved; 10= reserved; 11= disabled.

Table 37-20. HW_PINCTRL_MUXSEL6 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:18	**BANK3_PIN09**	RW	0x3	Pin 87, EMI_D09 pin function selection: 00= emi_data09; 01= reserved; 10= reserved; 11= disabled.
17:16	**BANK3_PIN08**	RW	0x3	Pin 86, EMI_D08 pin function selection: 00= emi_data08; 01= reserved; 10= reserved; 11= disabled.
15:14	**BANK3_PIN07**	RW	0x3	Pin 85, EMI_D07 pin function selection: 00= emi_data07; 01= reserved; 10= reserved; 11= disabled.
13:12	**BANK3_PIN06**	RW	0x3	Pin 84, EMI_D06 pin function selection: 00= emi_data06; 01= reserved; 10= reserved; 11= disabled.
11:10	**BANK3_PIN05**	RW	0x3	Pin 83, EMI_D05 pin function selection: 00= emi_data05; 01= reserved; 10= reserved; 11= disabled.
9:8	**BANK3_PIN04**	RW	0x3	Pin 82, EMI_D04 pin function selection: 00= emi_data04; 01= reserved; 10= reserved; 11= disabled.
7:6	**BANK3_PIN03**	RW	0x3	Pin 79, EMI_D03 pin function selection: 00= emi_data03; 01= reserved; 10= reserved; 11= disabled.
5:4	**BANK3_PIN02**	RW	0x3	Pin 77, EMI_D02 pin function selection: 00= emi_data02; 01= reserved; 10= reserved; 11= disabled.
3:2	**BANK3_PIN01**	RW	0x3	Pin 76, EMI_D01 pin function selection: 00= emi_data01; 01= reserved; 10= reserved; 11= disabled.
1:0	**BANK3_PIN00**	RW	0x3	Pin 75, EMI_D00 pin function selection: 00= emi_data00; 01= reserved; 10= reserved; 11= disabled.

DESCRIPTION:

This register allows the programmer to select which hardware interface blocks drive the 16 pins shown above.

EXAMPLE:

```
Empty Example.
```

37.4.9 PINCTRL Pin Mux Select Register 7 Description

The PINCTRL Pin Mux Select Register provides pin function selection for 6 pins in bank3.

HW_PINCTRL_MUXSEL7	0x170
HW_PINCTRL_MUXSEL7_SET	0x174
HW_PINCTRL_MUXSEL7_CLR	0x178
HW_PINCTRL_MUXSEL7_TOG	0x17C

Table 37-21. HW_PINCTRL_MUXSEL7

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD0																				BANK3_PIN21		BANK3_PIN20		BANK3_PIN19		BANK3_PIN18		BANK3_PIN17		BANK3_PIN16	

Table 37-22. HW_PINCTRL_MUXSEL7 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:12	**RSRVD0**	RO	0x0	Always write zeroes to this field.
11:10	**BANK3_PIN21**	RW	0x3	Pin 72, EMI_CLKN pin function selection: 00= emi_clkn; 01= reserved; 10= reserved; 11= disabled.
9:8	**BANK3_PIN20**	RW	0x3	Pin 70, EMI_CLK pin function selection: 00= emi_clk; 01= reserved; 10= reserved; 11= disabled.
7:6	**BANK3_PIN19**	RW	0x3	Pin 74, EMI_DQS1 pin function selection: 00= emi_dqs1; 01= reserved; 10= reserved; 11= disabled.
5:4	**BANK3_PIN18**	RW	0x3	Pin 73, EMI_DQS0 pin function selection: 00= emi_dqs0; 01= reserved; 10= reserved; 11= disabled.

Table 37-22. HW_PINCTRL_MUXSEL7 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
3:2	**BANK3_PIN17**	RW	0x3	Pin 92, EMI_DQM1 pin function selection: 00= emi_dqm1; 01= reserved; 10= reserved; 11= disabled.
1:0	**BANK3_PIN16**	RW	0x3	Pin 81, EMI_DQM0 pin function selection: 00= emi_dqm0; 01= reserved; 10= reserved; 11= disabled.

DESCRIPTION:

This register allows the programmer to select which hardware interface blocks drive the 6 pins shown above.

EXAMPLE:

```
Empty Example.
```

37.4.10 PINCTRL Drive Strength and Voltage Register 0 Description

The PINCTRL Drive Strength and Voltage Register selects the current drive strength for 8 pins of bank 0.

HW_PINCTRL_DRIVE0	0x200
HW_PINCTRL_DRIVE0_SET	0x204
HW_PINCTRL_DRIVE0_CLR	0x208
HW_PINCTRL_DRIVE0_TOG	0x20C

Table 37-23. HW_PINCTRL_DRIVE0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD7		BANK0_PIN07_MA		RSRVD6		BANK0_PIN06_MA		RSRVD5		BANK0_PIN05_MA		RSRVD4		BANK0_PIN04_MA		RSRVD3		BANK0_PIN03_MA		RSRVD2		BANK0_PIN02_MA		RSRVD1		BANK0_PIN01_MA		RSRVD0		BANK0_PIN00_MA	

Table 37-24. HW_PINCTRL_DRIVE0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD7**	RO	0x0	Always write zeroes to this field.
29:28	**BANK0_PIN07_MA**	RW	0x0	Pin 50, GPMI_D07 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
27:26	**RSRVD6**	RO	0x0	Always write zeroes to this field.

Table 37-24. HW_PINCTRL_DRIVE0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
25:24	**BANK0_PIN06_MA**	RW	0x0	Pin 51, GPMI_D06 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
23:22	**RSRVD5**	RO	0x0	Always write zeroes to this field.
21:20	**BANK0_PIN05_MA**	RW	0x0	Pin 48, GPMI_D05 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
19:18	**RSRVD4**	RO	0x0	Always write zeroes to this field.
17:16	**BANK0_PIN04_MA**	RW	0x0	Pin 49, GPMI_D04 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
15:14	**RSRVD3**	RO	0x0	Always write zeroes to this field.
13:12	**BANK0_PIN03_MA**	RW	0x0	Pin 47, GPMI_D03 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
11:10	**RSRVD2**	RO	0x0	Always write zeroes to this field.
9:8	**BANK0_PIN02_MA**	RW	0x0	Pin 46, GPMI_D02 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
7:6	**RSRVD1**	RO	0x0	Always write zeroes to this field.
5:4	**BANK0_PIN01_MA**	RW	0x0	Pin 45, GPMI_D01 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
3:2	**RSRVD0**	RO	0x0	Always write zeroes to this field.
1:0	**BANK0_PIN00_MA**	RW	0x0	Pin 44, GPMI_D00 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.

DESCRIPTION:

The Drive Strength and Voltage Register selects the drive strength and voltage for pins that are configured for output.

EXAMPLE:

```
Empty Example.
```

37.4.11 PINCTRL Drive Strength and Voltage Register 1 Description

The PINCTRL Drive Strength and Voltage Register selects the current drive strength for 8 pins of bank 0.

HW_PINCTRL_DRIVE1	0x210
HW_PINCTRL_DRIVE1_SET	0x214
HW_PINCTRL_DRIVE1_CLR	0x218
HW_PINCTRL_DRIVE1_TOG	0x21C

Table 37-25. HW_PINCTRL_DRIVE1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD7		BANK0_PIN15_MA		RSRVD6		BANK0_PIN14_MA		RSRVD5		BANK0_PIN13_MA		RSRVD4		BANK0_PIN12_MA		RSRVD3		BANK0_PIN11_MA		RSRVD2		BANK0_PIN10_MA		RSRVD1		BANK0_PIN09_MA		RSRVD0		BANK0_PIN08_MA	

Table 37-26. HW_PINCTRL_DRIVE1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD7**	RO	0x0	Always write zeroes to this field.
29:28	**BANK0_PIN15_MA**	RW	0x0	Pin 59, GPMI_D15 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
27:26	**RSRVD6**	RO	0x0	Always write zeroes to this field.
25:24	**BANK0_PIN14_MA**	RW	0x0	Pin 58, GPMI_D14 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
23:22	**RSRVD5**	RO	0x0	Always write zeroes to this field.
21:20	**BANK0_PIN13_MA**	RW	0x0	Pin 57, GPMI_D13 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
19:18	**RSRVD4**	RO	0x0	Always write zeroes to this field.
17:16	**BANK0_PIN12_MA**	RW	0x0	Pin 56, GPMI_D12 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
15:14	**RSRVD3**	RO	0x0	Always write zeroes to this field.
13:12	**BANK0_PIN11_MA**	RW	0x0	Pin 55, GPMI_D11 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
11:10	**RSRVD2**	RO	0x0	Always write zeroes to this field.

Table 37-26. HW_PINCTRL_DRIVE1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
9:8	**BANK0_PIN10_MA**	RW	0x0	Pin 54, GPMI_D10 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
7:6	**RSRVD1**	RO	0x0	Always write zeroes to this field.
5:4	**BANK0_PIN09_MA**	RW	0x0	Pin 53, GPMI_D09 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
3:2	**RSRVD0**	RO	0x0	Always write zeroes to this field.
1:0	**BANK0_PIN08_MA**	RW	0x0	Pin 52, GPMI_D08 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.

DESCRIPTION:

The Drive Strength and Voltage Register selects the drive strength and voltage for pins that are configured for output.

EXAMPLE:

```
Empty Example.
```

37.4.12 PINCTRL Drive Strength and Voltage Register 2 Description

The PINCTRL Drive Strength and Voltage Register selects the current drive strength for 8 pins of bank 0.

HW_PINCTRL_DRIVE2	0x220
HW_PINCTRL_DRIVE2_SET	0x224
HW_PINCTRL_DRIVE2_CLR	0x228
HW_PINCTRL_DRIVE2_TOG	0x22C

Table 37-27. HW_PINCTRL_DRIVE2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD7		BANK0_PIN23_MA		RSRVD6		BANK0_PIN22_MA		RSRVD5		BANK0_PIN21_MA		RSRVD4		BANK0_PIN20_MA		RSRVD3		BANK0_PIN19_MA		RSRVD2		BANK0_PIN18_MA		RSRVD1		BANK0_PIN17_MA		RSRVD0		BANK0_PIN16_MA	

Table 37-28. HW_PINCTRL_DRIVE2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD7**	RO	0x0	Always write zeroes to this field.
29:28	**BANK0_PIN23_MA**	RW	0x0	Pin 64, GPMI_WPN pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
27:26	**RSRVD6**	RO	0x0	Always write zeroes to this field.
25:24	**BANK0_PIN22_MA**	RW	0x0	Pin 63, GPMI_RDY3 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
23:22	**RSRVD5**	RO	0x0	Always write zeroes to this field.
21:20	**BANK0_PIN21_MA**	RW	0x0	Pin 62, GPMI_RDY2 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
19:18	**RSRVD4**	RO	0x0	Always write zeroes to this field.
17:16	**BANK0_PIN20_MA**	RW	0x0	Pin 43, GPMI_RDY1 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
15:14	**RSRVD3**	RO	0x0	Always write zeroes to this field.
13:12	**BANK0_PIN19_MA**	RW	0x0	Pin 61, GPMI_RDY0 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
11:10	**RSRVD2**	RO	0x0	Always write zeroes to this field.
9:8	**BANK0_PIN18_MA**	RW	0x0	Pin 42, GPMI_CE2N pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
7:6	**RSRVD1**	RO	0x0	Always write zeroes to this field.
5:4	**BANK0_PIN17_MA**	RW	0x0	Pin 41, GPMI_ALE pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
3:2	**RSRVD0**	RO	0x0	Always write zeroes to this field.
1:0	**BANK0_PIN16_MA**	RW	0x0	Pin 40, GPMI_CLE pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.

DESCRIPTION:

The Drive Strength and Voltage Register selects the drive strength and voltage for pins that are configured for output.

EXAMPLE:

```
Empty Example.
```

37.4.13 PINCTRL Drive Strength and Voltage Register 3 Description

The PINCTRL Drive Strength and Voltage Register selects the current drive strength for 8 pins of bank 0.

HW_PINCTRL_DRIVE3	0x230
HW_PINCTRL_DRIVE3_SET	0x234
HW_PINCTRL_DRIVE3_CLR	0x238
HW_PINCTRL_DRIVE3_TOG	0x23C

Table 37-29. HW_PINCTRL_DRIVE3

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	09 08	07 06	05 04	03 02	01 00
RSRVD7	BANK0_PIN31_MA	RSRVD6	BANK0_PIN30_MA	RSRVD5	BANK0_PIN29_MA	RSRVD4	BANK0_PIN28_MA	RSRVD3	BANK0_PIN27_MA	RSRVD2	BANK0_PIN26_MA	RSRVD1	BANK0_PIN25_MA	RSRVD0	BANK0_PIN24_MA

Table 37-30. HW_PINCTRL_DRIVE3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD7**	RO	0x0	Always write zeroes to this field.
29:28	**BANK0_PIN31_MA**	RW	0x0	Pin 4, I2C_SDA pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
27:26	**RSRVD6**	RO	0x0	Always write zeroes to this field.
25:24	**BANK0_PIN30_MA**	RW	0x0	Pin 2, I2C_SCL pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
23:22	**RSRVD5**	RO	0x0	Always write zeroes to this field.
21:20	**BANK0_PIN29_MA**	RW	0x0	Pin 69, AUART1_TX pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
19:18	**RSRVD4**	RO	0x0	Always write zeroes to this field.

Table 37-30. HW_PINCTRL_DRIVE3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
17:16	**BANK0_PIN28_MA**	RW	0x0	Pin 68, AUART1_RX pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
15:14	**RSRVD3**	RO	0x0	Always write zeroes to this field.
13:12	**BANK0_PIN27_MA**	RW	0x0	Pin 67, AUART1_RTS pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
11:10	**RSRVD2**	RO	0x0	Always write zeroes to this field.
9:8	**BANK0_PIN26_MA**	RW	0x0	Pin 66, AUART1_CTS pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
7:6	**RSRVD1**	RO	0x0	Always write zeroes to this field.
5:4	**BANK0_PIN25_MA**	RW	0x0	Pin 60, GPMI_RDN pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
3:2	**RSRVD0**	RO	0x0	Always write zeroes to this field.
1:0	**BANK0_PIN24_MA**	RW	0x0	Pin 65, GPMI_WRN pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.

DESCRIPTION:

The Drive Strength and Voltage Register selects the drive strength and voltage for pins that are configured for output.

EXAMPLE:

```
Empty Example.
```

37.4.14 PINCTRL Drive Strength and Voltage Register 4 Description

The PINCTRL Drive Strength and Voltage Register selects the current drive strength for 8 pins of bank 1.

HW_PINCTRL_DRIVE4	0x240
HW_PINCTRL_DRIVE4_SET	0x244
HW_PINCTRL_DRIVE4_CLR	0x248
HW_PINCTRL_DRIVE4_TOG	0x24C

Table 37-31. HW_PINCTRL_DRIVE4

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	09 08	07 06	05 04	03 02	01 00
RSRVD7	BANK1_PIN07_MA	RSRVD6	BANK1_PIN06_MA	RSRVD5	BANK1_PIN05_MA	RSRVD4	BANK1_PIN04_MA	RSRVD3	BANK1_PIN03_MA	RSRVD2	BANK1_PIN02_MA	RSRVD1	BANK1_PIN01_MA	RSRVD0	BANK1_PIN00_MA

Table 37-32. HW_PINCTRL_DRIVE4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD7**	RO	0x0	Always write zeroes to this field.
29:28	**BANK1_PIN07_MA**	RW	0x0	Pin 25, LCD_D07 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
27:26	**RSRVD6**	RO	0x0	Always write zeroes to this field.
25:24	**BANK1_PIN06_MA**	RW	0x0	Pin 23, LCD_D06 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
23:22	**RSRVD5**	RO	0x0	Always write zeroes to this field.
21:20	**BANK1_PIN05_MA**	RW	0x0	Pin 21, LCD_D05 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
19:18	**RSRVD4**	RO	0x0	Always write zeroes to this field.
17:16	**BANK1_PIN04_MA**	RW	0x0	Pin 18, LCD_D04 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
15:14	**RSRVD3**	RO	0x0	Always write zeroes to this field.
13:12	**BANK1_PIN03_MA**	RW	0x0	Pin 16, LCD_D03 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
11:10	**RSRVD2**	RO	0x0	Always write zeroes to this field.
9:8	**BANK1_PIN02_MA**	RW	0x0	Pin 14, LCD_D02 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
7:6	**RSRVD1**	RO	0x0	Always write zeroes to this field.

Table 37-32. HW_PINCTRL_DRIVE4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
5:4	**BANK1_PIN01_MA**	RW	0x0	Pin 12, LCD_D01 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
3:2	**RSRVD0**	RO	0x0	Always write zeroes to this field.
1:0	**BANK1_PIN00_MA**	RW	0x0	Pin 10, LCD_D00 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.

DESCRIPTION:

The Drive Strength and Voltage Register selects the drive strength and voltage for pins that are configured for output.

EXAMPLE:

```
Empty Example.
```

37.4.15 PINCTRL Drive Strength and Voltage Register 5 Description

The PINCTRL Drive Strength and Voltage Register selects the current drive strength for 8 pins of bank 1.

HW_PINCTRL_DRIVE5	0x250
HW_PINCTRL_DRIVE5_SET	0x254
HW_PINCTRL_DRIVE5_CLR	0x258
HW_PINCTRL_DRIVE5_TOG	0x25C

Table 37-33. HW_PINCTRL_DRIVE5

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	09 08	07 06	05 04	03 02	01 00
RSRVD7	BANK1_PIN15_MA	RSRVD6	BANK1_PIN14_MA	RSRVD5	BANK1_PIN13_MA	RSRVD4	BANK1_PIN12_MA	RSRVD3	BANK1_PIN11_MA	RSRVD2	BANK1_PIN10_MA	RSRVD1	BANK1_PIN09_MA	RSRVD0	BANK1_PIN08_MA

Table 37-34. HW_PINCTRL_DRIVE5 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD7**	RO	0x0	Always write zeroes to this field.
29:28	**BANK1_PIN15_MA**	RW	0x0	Pin 15, LCD_D15 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
27:26	**RSRVD6**	RO	0x0	Always write zeroes to this field.

Table 37-34. HW_PINCTRL_DRIVE5 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
25:24	**BANK1_PIN14_MA**	RW	0x0	Pin 17, LCD_D14 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
23:22	**RSRVD5**	RO	0x0	Always write zeroes to this field.
21:20	**BANK1_PIN13_MA**	RW	0x0	Pin 19, LCD_D13 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
19:18	**RSRVD4**	RO	0x0	Always write zeroes to this field.
17:16	**BANK1_PIN12_MA**	RW	0x0	Pin 22, LCD_D12 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
15:14	**RSRVD3**	RO	0x0	Always write zeroes to this field.
13:12	**BANK1_PIN11_MA**	RW	0x0	Pin 24, LCD_D11 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
11:10	**RSRVD2**	RO	0x0	Always write zeroes to this field.
9:8	**BANK1_PIN10_MA**	RW	0x0	Pin 26, LCD_D10 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
7:6	**RSRVD1**	RO	0x0	Always write zeroes to this field.
5:4	**BANK1_PIN09_MA**	RW	0x0	Pin 28, LCD_D09 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
3:2	**RSRVD0**	RO	0x0	Always write zeroes to this field.
1:0	**BANK1_PIN08_MA**	RW	0x0	Pin 27, LCD_D08 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.

DESCRIPTION:

The Drive Strength and Voltage Register selects the drive strength and voltage for pins that are configured for output.

EXAMPLE:

```
Empty Example.
```

37.4.16 PINCTRL Drive Strength and Voltage Register 6 Description

The PINCTRL Drive Strength and Voltage Register selects the current drive strength for 8 pins of bank 1.

HW_PINCTRL_DRIVE6	0x260
HW_PINCTRL_DRIVE6_SET	0x264
HW_PINCTRL_DRIVE6_CLR	0x268
HW_PINCTRL_DRIVE6_TOG	0x26C

Table 37-35. HW_PINCTRL_DRIVE6

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	09 08	07 06	05 04	03 02	01 00
RSRVD7	BANK1_PIN23_MA	RSRVD6	BANK1_PIN22_MA	RSRVD5	BANK1_PIN21_MA	RSRVD4	BANK1_PIN20_MA	RSRVD3	BANK1_PIN19_MA	RSRVD2	BANK1_PIN18_MA	RSRVD1	BANK1_PIN17_MA	RSRVD0	BANK1_PIN16_MA

Table 37-36. HW_PINCTRL_DRIVE6 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD7**	RO	0x0	Always write zeroes to this field.
29:28	**BANK1_PIN23_MA**	RW	0x0	Pin 30, LCD_ENABLE pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
27:26	**RSRVD6**	RO	0x0	Always write zeroes to this field.
25:24	**BANK1_PIN22_MA**	RW	0x0	Pin 36, LCD_DOTCK pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
23:22	**RSRVD5**	RO	0x0	Always write zeroes to this field.
21:20	**BANK1_PIN21_MA**	RW	0x0	Pin 29, LCD_CS pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
19:18	**RSRVD4**	RO	0x0	Always write zeroes to this field.
17:16	**BANK1_PIN20_MA**	RW	0x0	Pin 32, LCD_WR pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
15:14	**RSRVD3**	RO	0x0	Always write zeroes to this field.

Table 37-36. HW_PINCTRL_DRIVE6 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
13:12	**BANK1_PIN19_MA**	RW	0x0	Pin 33, LCD_RS pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
11:10	**RSRVD2**	RO	0x0	Always write zeroes to this field.
9:8	**BANK1_PIN18_MA**	RW	0x0	Pin 31, LCD_RESET pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
7:6	**RSRVD1**	RO	0x0	Always write zeroes to this field.
5:4	**BANK1_PIN17_MA**	RW	0x0	Pin 11, LCD_D17 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
3:2	**RSRVD0**	RO	0x0	Always write zeroes to this field.
1:0	**BANK1_PIN16_MA**	RW	0x0	Pin 13, LCD_D16 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.

DESCRIPTION:

The Drive Strength and Voltage Register selects the drive strength and voltage for pins that are configured for output.

EXAMPLE:

```
Empty Example.
```

37.4.17 PINCTRL Drive Strength and Voltage Register 7 Description

The PINCTRL Drive Strength and Voltage Register selects the current drive strength for 7 pins of bank 1.

HW_PINCTRL_DRIVE7	0x270
HW_PINCTRL_DRIVE7_SET	0x274
HW_PINCTRL_DRIVE7_CLR	0x278
HW_PINCTRL_DRIVE7_TOG	0x27C

Table 37-37. HW_PINCTRL_DRIVE7

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	09 08	07 06	05 04	03 02	01 00
RSRVD7	RSRVD6	BANK1_PIN30_MA	RSRVD5	BANK1_PIN29_MA	RSRVD4	BANK1_PIN28_MA	RSRVD3	BANK1_PIN27_MA	RSRVD2	BANK1_PIN26_MA	RSRVD1	BANK1_PIN25_MA	RSRVD0	BANK1_PIN24_MA	

Table 37-38. HW_PINCTRL_DRIVE7 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**RSRVD7**	RO	0x0	Always write zeroes to this field.
27:26	**RSRVD6**	RO	0x0	Always write zeroes to this field.
25:24	**BANK1_PIN30_MA**	RW	0x0	Pin 131, PWM4 pin output drive strength selection: 00= 8 mA; 01= 16 mA; 10= 24 mA; 11= reserved.
23:22	**RSRVD5**	RO	0x0	Always write zeroes to this field.
21:20	**BANK1_PIN29_MA**	RW	0x0	Pin 130, PWM3 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
19:18	**RSRVD4**	RO	0x0	Always write zeroes to this field.
17:16	**BANK1_PIN28_MA**	RW	0x0	Pin 129, PWM2 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
15:14	**RSRVD3**	RO	0x0	Always write zeroes to this field.
13:12	**BANK1_PIN27_MA**	RW	0x0	Pin 3, PWM1 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
11:10	**RSRVD2**	RO	0x0	Always write zeroes to this field.
9:8	**BANK1_PIN26_MA**	RW	0x0	Pin 1, PWM0 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
7:6	**RSRVD1**	RO	0x0	Always write zeroes to this field.
5:4	**BANK1_PIN25_MA**	RW	0x0	Pin 35, LCD_VSYNC pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.

Table 37-38. HW_PINCTRL_DRIVE7 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
3:2	**RSRVD0**	RO	0x0	Always write zeroes to this field.
1:0	**BANK1_PIN24_MA**	RW	0x0	Pin 34, LCD_HSYNC pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.

DESCRIPTION:

The Drive Strength and Voltage Register selects the drive strength and voltage for pins that are configured for output.

EXAMPLE:

```
Empty Example.
```

37.4.18 PINCTRL Drive Strength and Voltage Register 8 Description

The PINCTRL Drive Strength and Voltage Register selects the current drive strength for 8 pins of bank 2.

HW_PINCTRL_DRIVE8	0x280
HW_PINCTRL_DRIVE8_SET	0x284
HW_PINCTRL_DRIVE8_CLR	0x288
HW_PINCTRL_DRIVE8_TOG	0x28C

Table 37-39. HW_PINCTRL_DRIVE8

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	09 08	07 06	05 04	03 02	01 00
RSRVD7	BANK2_PIN07_MA	RSRVD6	BANK2_PIN06_MA	RSRVD5	BANK2_PIN05_MA	RSRVD4	BANK2_PIN04_MA	RSRVD3	BANK2_PIN03_MA	RSRVD2	BANK2_PIN02_MA	RSRVD1	BANK2_PIN01_MA	RSRVD0	BANK2_PIN00_MA

Table 37-40. HW_PINCTRL_DRIVE8 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD7**	RO	0x0	Always write zeroes to this field.
29:28	**BANK2_PIN07_MA**	RW	0x0	Pin 37, ROTARYA pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
27:26	**RSRVD6**	RO	0x0	Always write zeroes to this field.

Table 37-40. HW_PINCTRL_DRIVE8 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
25:24	**BANK2_PIN06_MA**	RW	0x0	Pin 127, SSP1_SCK pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
23:22	**RSRVD5**	RO	0x0	Always write zeroes to this field.
21:20	**BANK2_PIN05_MA**	RW	0x0	Pin 125, SSP1_DATA3 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
19:18	**RSRVD4**	RO	0x0	Always write zeroes to this field.
17:16	**BANK2_PIN04_MA**	RW	0x0	Pin 124, SSP1_DATA2 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
15:14	**RSRVD3**	RO	0x0	Always write zeroes to this field.
13:12	**BANK2_PIN03_MA**	RW	0x0	Pin 123, SSP1_DATA1 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
11:10	**RSRVD2**	RO	0x0	Always write zeroes to this field.
9:8	**BANK2_PIN02_MA**	RW	0x0	Pin 122, SSP1_DATA0 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
7:6	**RSRVD1**	RO	0x0	Always write zeroes to this field.
5:4	**BANK2_PIN01_MA**	RW	0x0	Pin 126, SSP1_DETECT pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
3:2	**RSRVD0**	RO	0x0	Always write zeroes to this field.
1:0	**BANK2_PIN00_MA**	RW	0x0	Pin 121, SSP1_CMD pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.

DESCRIPTION:

The Drive Strength and Voltage Register selects the drive strength and voltage for pins that are configured for output.

EXAMPLE:

```
Empty Example.
```

37.4.19 PINCTRL Drive Strength and Voltage Register 9 Description

The PINCTRL Drive Strength and Voltage Register selects the current drive strength for 8 pins of bank 2.

HW_PINCTRL_DRIVE9	0x290
HW_PINCTRL_DRIVE9_SET	0x294
HW_PINCTRL_DRIVE9_CLR	0x298
HW_PINCTRL_DRIVE9_TOG	0x29C

Table 37-41. HW_PINCTRL_DRIVE9

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD7	BANK2_PIN15_V	BANK2_PIN15_MA		RSRVD6	BANK2_PIN14_V	BANK2_PIN14_MA		RSRVD5	BANK2_PIN13_V	BANK2_PIN13_MA		RSRVD4	BANK2_PIN12_V	BANK2_PIN12_MA		RSRVD3	BANK2_PIN11_V	BANK2_PIN11_MA		RSRVD2	BANK2_PIN10_V	BANK2_PIN10_MA		RSRVD1	BANK2_PIN09_V	BANK2_PIN09_MA		RSRVD0		BANK2_PIN08_MA	

Table 37-42. HW_PINCTRL_DRIVE9 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSRVD7**	RO	0x0	Always write zeroes to this field.
30	**BANK2_PIN15_V**	RW	0x1	Pin 108, EMI_A06 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
29:28	**BANK2_PIN15_MA**	RW	0x0	Pin 108, EMI_A06 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
27	**RSRVD6**	RO	0x0	Always write zeroes to this field.
26	**BANK2_PIN14_V**	RW	0x1	Pin 107, EMI_A05 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
25:24	**BANK2_PIN14_MA**	RW	0x0	Pin 107, EMI_A05 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
23	**RSRVD5**	RO	0x0	Always write zeroes to this field.
22	**BANK2_PIN13_V**	RW	0x1	Pin 109, EMI_A04 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
21:20	**BANK2_PIN13_MA**	RW	0x0	Pin 109, EMI_A04 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.

Table 37-42. HW_PINCTRL_DRIVE9 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19	**RSRVD4**	RO	0x0	Always write zeroes to this field.
18	**BANK2_PIN12_V**	RW	0x1	Pin 110, EMI_A03 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
17:16	**BANK2_PIN12_MA**	RW	0x0	Pin 110, EMI_A03 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
15	**RSRVD3**	RO	0x0	Always write zeroes to this field.
14	**BANK2_PIN11_V**	RW	0x1	Pin 111, EMI_A02 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
13:12	**BANK2_PIN11_MA**	RW	0x0	Pin 111, EMI_A02 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
11	**RSRVD2**	RO	0x0	Always write zeroes to this field.
10	**BANK2_PIN10_V**	RW	0x1	Pin 112, EMI_A01 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
9:8	**BANK2_PIN10_MA**	RW	0x0	Pin 112, EMI_A01 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
7	**RSRVD1**	RO	0x0	Always write zeroes to this field.
6	**BANK2_PIN09_V**	RW	0x1	Pin 113, EMI_A00 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
5:4	**BANK2_PIN09_MA**	RW	0x0	Pin 113, EMI_A00 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
3:2	**RSRVD0**	RO	0x0	Always write zeroes to this field.
1:0	**BANK2_PIN08_MA**	RW	0x0	Pin 38, ROTARYB pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.

DESCRIPTION:

The Drive Strength and Voltage Register selects the drive strength and voltage for pins that are configured for output.

EXAMPLE:

```
Empty Example.
```

37.4.20 PINCTRL Drive Strength and Voltage Register 10 Description

The PINCTRL Drive Strength and Voltage Register selects the current drive strength for 8 pins of bank 2.

HW_PINCTRL_DRIVE10	0x2a0
HW_PINCTRL_DRIVE10_SET	0x2a4
HW_PINCTRL_DRIVE10_CLR	0x2a8
HW_PINCTRL_DRIVE10_TOG	0x2aC

Table 37-43. HW_PINCTRL_DRIVE10

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD7	BANK2_PIN23_V	BANK2_PIN23_MA		RSRVD6	BANK2_PIN22_V	BANK2_PIN22_MA		RSRVD5	BANK2_PIN21_V	BANK2_PIN21_MA		RSRVD4	BANK2_PIN20_V	BANK2_PIN20_MA		RSRVD3	BANK2_PIN19_V	BANK2_PIN19_MA		RSRVD2	BANK2_PIN18_V	BANK2_PIN18_MA		RSRVD1	BANK2_PIN17_V	BANK2_PIN17_MA		RSRVD0	BANK2_PIN16_V	BANK2_PIN16_MA	

Table 37-44. HW_PINCTRL_DRIVE10 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSRVD7**	RO	0x0	Always write zeroes to this field.
30	**BANK2_PIN23_V**	RW	0x1	Pin 117, EMI_BA1 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
29:28	**BANK2_PIN23_MA**	RW	0x0	Pin 117, EMI_BA1 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
27	**RSRVD6**	RO	0x0	Always write zeroes to this field.
26	**BANK2_PIN22_V**	RW	0x1	Pin 116, EMI_BA0 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
25:24	**BANK2_PIN22_MA**	RW	0x0	Pin 116, EMI_BA0 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
23	**RSRVD5**	RO	0x0	Always write zeroes to this field.
22	**BANK2_PIN21_V**	RW	0x1	Pin 101, EMI_A12 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
21:20	**BANK2_PIN21_MA**	RW	0x0	Pin 101, EMI_A12 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
19	**RSRVD4**	RO	0x0	Always write zeroes to this field.
18	**BANK2_PIN20_V**	RW	0x1	Pin 102, EMI_A11 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.

Table 37-44. HW_PINCTRL_DRIVE10 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
17:16	**BANK2_PIN20_MA**	RW	0x0	Pin 102, EMI_A11 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
15	**RSRVD3**	RO	0x0	Always write zeroes to this field.
14	**BANK2_PIN19_V**	RW	0x1	Pin 104, EMI_A10 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
13:12	**BANK2_PIN19_MA**	RW	0x0	Pin 104, EMI_A10 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
11	**RSRVD2**	RO	0x0	Always write zeroes to this field.
10	**BANK2_PIN18_V**	RW	0x1	Pin 103, EMI_A09 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
9:8	**BANK2_PIN18_MA**	RW	0x0	Pin 103, EMI_A09 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
7	**RSRVD1**	RO	0x0	Always write zeroes to this field.
6	**BANK2_PIN17_V**	RW	0x1	Pin 106, EMI_A08 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
5:4	**BANK2_PIN17_MA**	RW	0x0	Pin 106, EMI_A08 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
3	**RSRVD0**	RO	0x0	Always write zeroes to this field.
2	**BANK2_PIN16_V**	RW	0x1	Pin 105, EMI_A07 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
1:0	**BANK2_PIN16_MA**	RW	0x0	Pin 105, EMI_A07 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.

DESCRIPTION:

The Drive Strength and Voltage Register selects the drive strength and voltage for pins that are configured for output.

EXAMPLE:

```
Empty Example.
```

37.4.21 PINCTRL Drive Strength and Voltage Register 11 Description

The PINCTRL Drive Strength and Voltage Register selects the current drive strength for 8 pins of bank 2.

HW_PINCTRL_DRIVE11	0x2b0
HW_PINCTRL_DRIVE11_SET	0x2b4
HW_PINCTRL_DRIVE11_CLR	0x2b8
HW_PINCTRL_DRIVE11_TOG	0x2bC

Table 37-45. HW_PINCTRL_DRIVE11

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD7	BANK2_PIN31_V	BANK2_PIN31_MA		RSRVD6	BANK2_PIN30_V	BANK2_PIN30_MA		RSRVD5	BANK2_PIN29_V	BANK2_PIN29_MA		RSRVD4		BANK2_PIN28_MA		RSRVD3		BANK2_PIN27_MA		RSRVD2	BANK2_PIN26_V	BANK2_PIN26_MA		RSRVD1	BANK2_PIN25_V	BANK2_PIN25_MA		RSRVD0	BANK2_PIN24_V	BANK2_PIN24_MA	

Table 37-46. HW_PINCTRL_DRIVE11 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSRVD7**	RO	0x0	Always write zeroes to this field.
30	**BANK2_PIN31_V**	RW	0x1	Pin 114, EMI_WEN pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
29:28	**BANK2_PIN31_MA**	RW	0x0	Pin 114, EMI_WEN pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
27	**RSRVD6**	RO	0x0	Always write zeroes to this field.
26	**BANK2_PIN30_V**	RW	0x1	Pin 98, EMI_RASN pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
25:24	**BANK2_PIN30_MA**	RW	0x0	Pin 98, EMI_RASN pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
23	**RSRVD5**	RO	0x0	Always write zeroes to this field.
22	**BANK2_PIN29_V**	RW	0x1	Pin 115, EMI_CKE pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
21:20	**BANK2_PIN29_MA**	RW	0x0	Pin 115, EMI_CKE pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
19:18	**RSRVD4**	RO	0x0	Always write zeroes to this field.

Table 37-46. HW_PINCTRL_DRIVE11 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
17:16	**BANK2_PIN28_MA**	RW	0x0	Pin 120, GPMI_CE0N pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
15:14	**RSRVD3**	RO	0x0	Always write zeroes to this field.
13:12	**BANK2_PIN27_MA**	RW	0x0	Pin 118, GPMI_CE1N pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= reserved.
11	**RSRVD2**	RO	0x0	Always write zeroes to this field.
10	**BANK2_PIN26_V**	RW	0x1	Pin 99, EMI_CE1N pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
9:8	**BANK2_PIN26_MA**	RW	0x0	Pin 99, EMI_CE1N pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
7	**RSRVD1**	RO	0x0	Always write zeroes to this field.
6	**BANK2_PIN25_V**	RW	0x1	Pin 100, EMI_CE0N pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
5:4	**BANK2_PIN25_MA**	RW	0x0	Pin 100, EMI_CE0N pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
3	**RSRVD0**	RO	0x0	Always write zeroes to this field.
2	**BANK2_PIN24_V**	RW	0x1	Pin 97, EMI_CASN pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
1:0	**BANK2_PIN24_MA**	RW	0x0	Pin 97, EMI_CASN pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.

DESCRIPTION:

The Drive Strength and Voltage Register selects the drive strength and voltage for pins that are configured for output.

EXAMPLE:

```
Empty Example.
```

37.4.22 PINCTRL Drive Strength and Voltage Register 12 Description

The PINCTRL Drive Strength and Voltage Register selects the current drive strength for 8 pins of bank 3.

HW_PINCTRL_DRIVE12	0x2c0
HW_PINCTRL_DRIVE12_SET	0x2c4
HW_PINCTRL_DRIVE12_CLR	0x2c8
HW_PINCTRL_DRIVE12_TOG	0x2cC

Table 37-47. HW_PINCTRL_DRIVE12

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD7	BANK3_PIN07_V	BANK3_PIN07_MA		RSRVD6	BANK3_PIN06_V	BANK3_PIN06_MA		RSRVD5	BANK3_PIN05_V	BANK3_PIN05_MA		RSRVD4	BANK3_PIN04_V	BANK3_PIN04_MA		RSRVD3	BANK3_PIN03_V	BANK3_PIN03_MA		RSRVD2	BANK3_PIN02_V	BANK3_PIN02_MA		RSRVD1	BANK3_PIN01_V	BANK3_PIN01_MA		RSRVD0	BANK3_PIN00_V	BANK3_PIN00_MA	

Table 37-48. HW_PINCTRL_DRIVE12 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSRVD7**	RO	0x0	Always write zeroes to this field.
30	**BANK3_PIN07_V**	RW	0x1	Pin 85, EMI_D07 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
29:28	**BANK3_PIN07_MA**	RW	0x0	Pin 85, EMI_D07 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
27	**RSRVD6**	RO	0x0	Always write zeroes to this field.
26	**BANK3_PIN06_V**	RW	0x1	Pin 84, EMI_D06 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
25:24	**BANK3_PIN06_MA**	RW	0x0	Pin 84, EMI_D06 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
23	**RSRVD5**	RO	0x0	Always write zeroes to this field.
22	**BANK3_PIN05_V**	RW	0x1	Pin 83, EMI_D05 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
21:20	**BANK3_PIN05_MA**	RW	0x0	Pin 83, EMI_D05 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
19	**RSRVD4**	RO	0x0	Always write zeroes to this field.
18	**BANK3_PIN04_V**	RW	0x1	Pin 82, EMI_D04 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.

Table 37-48. HW_PINCTRL_DRIVE12 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
17:16	**BANK3_PIN04_MA**	RW	0x0	Pin 82, EMI_D04 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
15	**RSRVD3**	RO	0x0	Always write zeroes to this field.
14	**BANK3_PIN03_V**	RW	0x1	Pin 79, EMI_D03 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
13:12	**BANK3_PIN03_MA**	RW	0x0	Pin 79, EMI_D03 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
11	**RSRVD2**	RO	0x0	Always write zeroes to this field.
10	**BANK3_PIN02_V**	RW	0x1	Pin 77, EMI_D02 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
9:8	**BANK3_PIN02_MA**	RW	0x0	Pin 77, EMI_D02 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
7	**RSRVD1**	RO	0x0	Always write zeroes to this field.
6	**BANK3_PIN01_V**	RW	0x1	Pin 76, EMI_D01 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
5:4	**BANK3_PIN01_MA**	RW	0x0	Pin 76, EMI_D01 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
3	**RSRVD0**	RO	0x0	Always write zeroes to this field.
2	**BANK3_PIN00_V**	RW	0x1	Pin 75, EMI_D00 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
1:0	**BANK3_PIN00_MA**	RW	0x0	Pin 75, EMI_D00 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.

DESCRIPTION:

The Drive Strength and Voltage Register selects the drive strength and voltage for pins that are configured for output.

EXAMPLE:

```
Empty Example.
```

37.4.23 PINCTRL Drive Strength and Voltage Register 13 Description

The PINCTRL Drive Strength and Voltage Register selects the current drive strength for 8 pins of bank 3.

HW_PINCTRL_DRIVE13	0x2d0
HW_PINCTRL_DRIVE13_SET	0x2d4
HW_PINCTRL_DRIVE13_CLR	0x2d8
HW_PINCTRL_DRIVE13_TOG	0x2dC

Table 37-49. HW_PINCTRL_DRIVE13

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD7	BANK3_PIN15_V	BANK3_PIN15_MA		RSRVD6	BANK3_PIN14_V	BANK3_PIN14_MA		RSRVD5	BANK3_PIN13_V	BANK3_PIN13_MA		RSRVD4	BANK3_PIN12_V	BANK3_PIN12_MA		RSRVD3	BANK3_PIN11_V	BANK3_PIN11_MA		RSRVD2	BANK3_PIN10_V	BANK3_PIN10_MA		RSRVD1	BANK3_PIN09_V	BANK3_PIN09_MA		RSRVD0	BANK3_PIN08_V	BANK3_PIN08_MA	

Table 37-50. HW_PINCTRL_DRIVE13 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSRVD7**	RO	0x0	Always write zeroes to this field.
30	**BANK3_PIN15_V**	RW	0x1	Pin 95, EMI_D15 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
29:28	**BANK3_PIN15_MA**	RW	0x0	Pin 95, EMI_D15 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
27	**RSRVD6**	RO	0x0	Always write zeroes to this field.
26	**BANK3_PIN14_V**	RW	0x1	Pin 96, EMI_D14 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
25:24	**BANK3_PIN14_MA**	RW	0x0	Pin 96, EMI_D14 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
23	**RSRVD5**	RO	0x0	Always write zeroes to this field.
22	**BANK3_PIN13_V**	RW	0x1	Pin 94, EMI_D13 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
21:20	**BANK3_PIN13_MA**	RW	0x0	Pin 94, EMI_D13 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
19	**RSRVD4**	RO	0x0	Always write zeroes to this field.
18	**BANK3_PIN12_V**	RW	0x1	Pin 93, EMI_D12 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.

Table 37-50. HW_PINCTRL_DRIVE13 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
17:16	**BANK3_PIN12_MA**	RW	0x0	Pin 93, EMI_D12 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
15	**RSRVD3**	RO	0x0	Always write zeroes to this field.
14	**BANK3_PIN11_V**	RW	0x1	Pin 91, EMI_D11 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
13:12	**BANK3_PIN11_MA**	RW	0x0	Pin 91, EMI_D11 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
11	**RSRVD2**	RO	0x0	Always write zeroes to this field.
10	**BANK3_PIN10_V**	RW	0x1	Pin 89, EMI_D10 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
9:8	**BANK3_PIN10_MA**	RW	0x0	Pin 89, EMI_D10 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
7	**RSRVD1**	RO	0x0	Always write zeroes to this field.
6	**BANK3_PIN09_V**	RW	0x1	Pin 87, EMI_D09 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
5:4	**BANK3_PIN09_MA**	RW	0x0	Pin 87, EMI_D09 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
3	**RSRVD0**	RO	0x0	Always write zeroes to this field.
2	**BANK3_PIN08_V**	RW	0x1	Pin 86, EMI_D08 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
1:0	**BANK3_PIN08_MA**	RW	0x0	Pin 86, EMI_D08 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.

DESCRIPTION:

The Drive Strength and Voltage Register selects the drive strength and voltage for pins that are configured for output.

EXAMPLE:

```
Empty Example.
```

37.4.24 PINCTRL Drive Strength and Voltage Register 14 Description

The PINCTRL Drive Strength and Voltage Register selects the current drive strength for 6 pins of bank 3.

HW_PINCTRL_DRIVE14	0x2e0
HW_PINCTRL_DRIVE14_SET	0x2e4
HW_PINCTRL_DRIVE14_CLR	0x2e8
HW_PINCTRL_DRIVE14_TOG	0x2eC

Table 37-51. HW_PINCTRL_DRIVE14

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD6								RSRVD5	BANK3_PIN21_V	BANK3_PIN21_MA		RSRVD4	BANK3_PIN20_V	BANK3_PIN20_MA		RSRVD3	BANK3_PIN19_V	BANK3_PIN19_MA		RSRVD2	BANK3_PIN18_V	BANK3_PIN18_MA		RSRVD1	BANK3_PIN17_V	BANK3_PIN17_MA		RSRVD0	BANK3_PIN16_V	BANK3_PIN16_MA	

Table 37-52. HW_PINCTRL_DRIVE14 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**RSRVD6**	RO	0x0	Always write zeroes to this field.
23	**RSRVD5**	RO	0x0	Always write zeroes to this field.
22	**BANK3_PIN21_V**	RW	0x1	Pin 72, EMI_CLKN pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
21:20	**BANK3_PIN21_MA**	RW	0x0	Pin 72, EMI_CLKN pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
19	**RSRVD4**	RO	0x0	Always write zeroes to this field.
18	**BANK3_PIN20_V**	RW	0x1	Pin 70, EMI_CLK pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
17:16	**BANK3_PIN20_MA**	RW	0x0	Pin 70, EMI_CLK pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
15	**RSRVD3**	RO	0x0	Always write zeroes to this field.
14	**BANK3_PIN19_V**	RW	0x1	Pin 74, EMI_DQS1 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
13:12	**BANK3_PIN19_MA**	RW	0x0	Pin 74, EMI_DQS1 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
11	**RSRVD2**	RO	0x0	Always write zeroes to this field.

Table 37-52. HW_PINCTRL_DRIVE14 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
10	**BANK3_PIN18_V**	RW	0x1	Pin 73, EMI_DQS0 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
9:8	**BANK3_PIN18_MA**	RW	0x0	Pin 73, EMI_DQS0 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
7	**RSRVD1**	RO	0x0	Always write zeroes to this field.
6	**BANK3_PIN17_V**	RW	0x1	Pin 92, EMI_DQM1 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
5:4	**BANK3_PIN17_MA**	RW	0x0	Pin 92, EMI_DQM1 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.
3	**RSRVD0**	RO	0x0	Always write zeroes to this field.
2	**BANK3_PIN16_V**	RW	0x1	Pin 81, EMI_DQM0 pin voltage selection: 0= 1.8V (mDDR) or 2.5V (DDR1); 1= reserved.
1:0	**BANK3_PIN16_MA**	RW	0x0	Pin 81, EMI_DQM0 pin output drive strength selection: 00= 4 mA; 01= 8 mA; 10= 12 mA; 11= 16 mA.

DESCRIPTION:

The Drive Strength and Voltage Register selects the drive strength and voltage for pins that are configured for output.

EXAMPLE:

```
Empty Example.
```

37.4.25 PINCTRL Bank 0 Pull Up Resistor Enable Register Description

The PINCTRL Bank 0 PULL Register enables/disables the internal pull up resistors for those pins in bank 0 which support this operation.

HW_PINCTRL_PULL0	0x400
HW_PINCTRL_PULL0_SET	0x404
HW_PINCTRL_PULL0_CLR	0x408
HW_PINCTRL_PULL0_TOG	0x40C

Table 37-53. HW_PINCTRL_PULL0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
BANK0_PIN31	BANK0_PIN30	BANK0_PIN29	BANK0_PIN28	BANK0_PIN27	BANK0_PIN26	RSRVD2			BANK0_PIN22	BANK0_PIN21	BANK0_PIN20	BANK0_PIN19	BANK0_PIN18	RSRVD1		BANK0_PIN15	RSRVD0			BANK0_PIN11	BANK0_PIN10	BANK0_PIN09	BANK0_PIN08	BANK0_PIN07	BANK0_PIN06	BANK0_PIN05	BANK0_PIN04	BANK0_PIN03	BANK0_PIN02	BANK0_PIN01	BANK0_PIN00

Table 37-54. HW_PINCTRL_PULL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**BANK0_PIN31**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 4, I2C_SDA.
30	**BANK0_PIN30**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 2, I2C_SCL.
29	**BANK0_PIN29**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 69, AUART1_TX.
28	**BANK0_PIN28**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 68, AUART1_RX.
27	**BANK0_PIN27**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 67, AUART1_RTS.
26	**BANK0_PIN26**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 66, AUART1_CTS.
25:23	**RSRVD2**	RO	0x0	Always write zeroes to this field.
22	**BANK0_PIN22**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 63, GPMI_RDY3.
21	**BANK0_PIN21**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 62, GPMI_RDY2.
20	**BANK0_PIN20**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 43, GPMI_RDY1.
19	**BANK0_PIN19**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 61, GPMI_RDY0.
18	**BANK0_PIN18**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 42, GPMI_CE2N.
17:16	**RSRVD1**	RO	0x0	Always write zeroes to this field.
15	**BANK0_PIN15**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 59, GPMI_D15.
14:12	**RSRVD0**	RO	0x0	Always write zeroes to this field.

Table 37-54. HW_PINCTRL_PULL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
11	**BANK0_PIN11**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 55, GPMI_D11.
10	**BANK0_PIN10**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 54, GPMI_D10.
9	**BANK0_PIN09**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 53, GPMI_D09.
8	**BANK0_PIN08**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 52, GPMI_D08.
7	**BANK0_PIN07**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 50, GPMI_D07.
6	**BANK0_PIN06**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 51, GPMI_D06.
5	**BANK0_PIN05**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 48, GPMI_D05.
4	**BANK0_PIN04**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 49, GPMI_D04.
3	**BANK0_PIN03**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 47, GPMI_D03.
2	**BANK0_PIN02**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 46, GPMI_D02.
1	**BANK0_PIN01**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 45, GPMI_D01.
0	**BANK0_PIN00**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 44, GPMI_D00.

DESCRIPTION:

The Pull register enables/disables integrated on-chip pull up resistors or pad keepers for the pins.

EXAMPLE:

```
Empty Example.
```

37.4.26 PINCTRL Bank 1 Pull Up Resistor Enable Register Description

The PINCTRL Bank 1 PULL Register enables/disables the internal pull up resistors for those pins in bank 1 which support this operation.

HW_PINCTRL_PULL1	0x410
HW_PINCTRL_PULL1_SET	0x414
HW_PINCTRL_PULL1_CLR	0x418

HW_PINCTRL_PULL1_TOG 0x41C

Table 37-55. HW_PINCTRL_PULL1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD3			BANK1_PIN28	RSRVD2					BANK1_PIN22	RSRVD1			BANK1_PIN18	RSRVD0																	

Table 37-56. HW_PINCTRL_PULL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSRVD3**	RO	0x0	Always write zeroes to this field.
28	**BANK1_PIN28**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 129, PWM2.
27:23	**RSRVD2**	RO	0x0	Always write zeroes to this field.
22	**BANK1_PIN22**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 36, LCD_DOTCK.
21:19	**RSRVD1**	RO	0x0	Always write zeroes to this field.
18	**BANK1_PIN18**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 31, LCD_RESET.
17:0	**RSRVD0**	RO	0x0	Always write zeroes to this field.

DESCRIPTION:

The Pull register enables/disables integrated on-chip pull up resistors or pad keepers for the pins.

EXAMPLE:

```
Empty Example.
```

37.4.27 PINCTRL Bank 2 Pull Up Resistor Enable Register Description

The PINCTRL Bank 2 PULL Register enables/disables the internal pull up resistors for those pins in bank 2 which support this operation.

HW_PINCTRL_PULL2 0x420
HW_PINCTRL_PULL2_SET 0x424
HW_PINCTRL_PULL2_CLR 0x428
HW_PINCTRL_PULL2_TOG 0x42C

Table 37-57. HW_PINCTRL_PULL2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD2			BANK2_PIN28	BANK2_PIN27	RSRVD1																		BANK2_PIN08	RSRVD0		BANK2_PIN05	BANK2_PIN04	BANK2_PIN03	BANK2_PIN02	BANK2_PIN01	BANK2_PIN00

Table 37-58. HW_PINCTRL_PULL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:29	**RSRVD2**	RO	0x0	Always write zeroes to this field.
28	**BANK2_PIN28**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 120, GPMI_CE0N.
27	**BANK2_PIN27**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 118, GPMI_CE1N.
26:9	**RSRVD1**	RO	0x0	Always write zeroes to this field.
8	**BANK2_PIN08**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 38, ROTARYB.
7:6	**RSRVD0**	RO	0x0	Always write zeroes to this field.
5	**BANK2_PIN05**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 125, SSP1_DATA3.
4	**BANK2_PIN04**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 124, SSP1_DATA2.
3	**BANK2_PIN03**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 123, SSP1_DATA1.
2	**BANK2_PIN02**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 122, SSP1_DATA0.
1	**BANK2_PIN01**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 126, SSP1_DETECT.
0	**BANK2_PIN00**	RW	0x0	Set this bit to one to disable the internal pad keeper and enable the internal pull up resistor on pin 121, SSP1_CMD.

DESCRIPTION:

The Pull register enables/disables integrated on-chip pull up resistors or pad keepers for the pins.

EXAMPLE:

```
Empty Example.
```

37.4.28 PINCTRL Bank 3 Pad Keeper Disable Register Description

The PINCTRL Bank 3 PULL Register enables/disables the pad keepers for those pins in bank 3 which support this operation.

HW_PINCTRL_PULL3	0x430
HW_PINCTRL_PULL3_SET	0x434
HW_PINCTRL_PULL3_CLR	0x438
HW_PINCTRL_PULL3_TOG	0x43C

Table 37-59. HW_PINCTRL_PULL3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD0														BANK3_PIN17	BANK3_PIN16	BANK3_PIN15	BANK3_PIN14	BANK3_PIN13	BANK3_PIN12	BANK3_PIN11	BANK3_PIN10	BANK3_PIN09	BANK3_PIN08	BANK3_PIN07	BANK3_PIN06	BANK3_PIN05	BANK3_PIN04	BANK3_PIN03	BANK3_PIN02	BANK3_PIN01	BANK3_PIN00

Table 37-60. HW_PINCTRL_PULL3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:18	**RSRVD0**	RO	0x0	Always write zeroes to this field.
17	**BANK3_PIN17**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 92, EMI_DQM1.
16	**BANK3_PIN16**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 81, EMI_DQM0.
15	**BANK3_PIN15**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 95, EMI_D15.
14	**BANK3_PIN14**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 96, EMI_D14.
13	**BANK3_PIN13**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 94, EMI_D13.
12	**BANK3_PIN12**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 93, EMI_D12.
11	**BANK3_PIN11**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 91, EMI_D11.
10	**BANK3_PIN10**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 89, EMI_D10.
9	**BANK3_PIN09**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 87, EMI_D09.
8	**BANK3_PIN08**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 86, EMI_D08.
7	**BANK3_PIN07**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 85, EMI_D07.
6	**BANK3_PIN06**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 84, EMI_D06.
5	**BANK3_PIN05**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 83, EMI_D05.
4	**BANK3_PIN04**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 82, EMI_D04.
3	**BANK3_PIN03**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 79, EMI_D03.

Table 37-60. HW_PINCTRL_PULL3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**BANK3_PIN02**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 77, EMI_D02.
1	**BANK3_PIN01**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 76, EMI_D01.
0	**BANK3_PIN00**	RW	0x0	Set this bit to one to disable the internal pad keeper on pin 75, EMI_D00.

DESCRIPTION:

The Pull register enables/disables integrated on-chip pull up resistors or pad keepers for the pins.

EXAMPLE:

```
Empty Example.
```

37.4.29 PINCTRL Bank 0 Data Output Register Description

The Bank 0 Data Output register provides data for all pins in bank 0 that are configured for GPIO output mode.

HW_PINCTRL_DOUT0	0x500
HW_PINCTRL_DOUT0_SET	0x504
HW_PINCTRL_DOUT0_CLR	0x508
HW_PINCTRL_DOUT0_TOG	0x50C

Table 37-61. HW_PINCTRL_DOUT0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DOUT																															

Table 37-62. HW_PINCTRL_DOUT0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**DOUT**	RW	0x00000000	This field selects the output value (0 or 1) for pins configured as GPIO outputs. Each bit in this register corresponds to one of the 32 pins in bank 0.

DESCRIPTION:

This register contains the data that will be driven out all bank 0 pins which are configured for GPIO output mode. For example, if HW_PINCTRL_MUXSEL0 contains 0x0000000F and HW_PINCTRL_DOE0 contains 0x00000001, then GPIO0[0] will be driven with the value from bit 0 of this register, GPIO0[1] will not be driven, and GPIO0[2:15] will be controlled by the associated primary interfaces.

EXAMPLE:

```
Empty Example.
```

37.4.30 PINCTRL Bank 1 Data Output Register Description

The Bank 1 Data Output register provides data for all pins in bank 1 that are configured for GPIO output mode.

HW_PINCTRL_DOUT1	0x510
HW_PINCTRL_DOUT1_SET	0x514
HW_PINCTRL_DOUT1_CLR	0x518
HW_PINCTRL_DOUT1_TOG	0x51C

Table 37-63. HW_PINCTRL_DOUT1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1	DOUT																														

Table 37-64. HW_PINCTRL_DOUT1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSRVD1**	RO	0x0	Reserved - write 0 to this bit-field.
30:0	**DOUT**	RW	0x00000000	This field selects the output value (0 or 1) for pins configured as GPIO outputs. Each bit in this register corresponds to one of the 31 pins in bank 1.

DESCRIPTION:

This register contains the data that will be driven out all bank 0 pins which are configured for GPIO output mode. For example, if HW_PINCTRL_MUXSEL0 contains 0x0000000F and HW_PINCTRL_DOE0 contains 0x00000001, then GPIO0[0] will be driven with the value from bit 0 of this register, GPIO0[1] will not be driven, and GPIO0[2:15] will be controlled by the associated primary interfaces.

EXAMPLE:

```
Empty Example.
```

37.4.31 PINCTRL Bank 2 Data Output Register Description

The Bank 2 Data Output register provides data for all pins in bank 2 that are configured for GPIO output mode.

HW_PINCTRL_DOUT2	0x520
HW_PINCTRL_DOUT2_SET	0x524
HW_PINCTRL_DOUT2_CLR	0x528
HW_PINCTRL_DOUT2_TOG	0x52C

Table 37-65. HW_PINCTRL_DOUT2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DOUT																															

Table 37-66. HW_PINCTRL_DOUT2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**DOUT**	RW	0x00000000	This field selects the output value (0 or 1) for pins configured as GPIO outputs. Each bit in this register corresponds to one of the 32 pins in bank 2.

DESCRIPTION:

This register contains the data that will be driven out all bank 0 pins which are configured for GPIO output mode. For example, if HW_PINCTRL_MUXSEL0 contains 0x0000000F and HW_PINCTRL_DOE0 contains 0x00000001, then GPIO0[0] will be driven with the value from bit 0 of this register, GPIO0[1] will not be driven, and GPIO0[2:15] will be controlled by the associated primary interfaces.

EXAMPLE:

```
Empty Example.
```

37.4.32 PINCTRL Bank 0 Data Input Register Description

The current value of all bank 0 pins may be read from the PINCTRL Bank 0 Data Input Register.

HW_PINCTRL_DIN0	0x600
HW_PINCTRL_DIN0_SET	0x604
HW_PINCTRL_DIN0_CLR	0x608
HW_PINCTRL_DIN0_TOG	0x60C

Table 37-67. HW_PINCTRL_DIN0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DIN																															

Table 37-68. HW_PINCTRL_DIN0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**DIN**	RO	0x00000000	Each bit in this read-only register corresponds to one of the 32 pins in bank 0. The current state of each pin in bank 0, synchronized to HCLK, may be read here.

DESCRIPTION:

This register reflects the current values of all the bank 0 pins. The register accurately reflects the state of the pin regardless of the setting of the HW_PINCTRL_MUXSELx or HW_PINCTRL_DOEx registers, but generally, if it is desired to use a pin as a general purpose input, the pin's two bits in the HW_PINCTRL_MUXSELx register should be set to 3 (GPIO mode) and the pin's bit in the HW_PINCTRL_DOEx register should be set to 0 (disabled) to insure that the chip is not driving the pin.

EXAMPLE:

```
Empty Example.
```

37.4.33 PINCTRL Bank 1 Data Input Register Description

The current value of all bank 1 pins may be read from the PINCTRL Bank 1 Data Input Register.

HW_PINCTRL_DIN1	0x610
HW_PINCTRL_DIN1_SET	0x614
HW_PINCTRL_DIN1_CLR	0x618
HW_PINCTRL_DIN1_TOG	0x61C

Table 37-69. HW_PINCTRL_DIN1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1	DIN																														

Table 37-70. HW_PINCTRL_DIN1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSRVD1**	RO	0x0	Reserved - write 0 to this bit-field.
30:0	**DIN**	RO	0x00000000	Each bit in this read-only register corresponds to one of the 31 pins in bank 1. The current state of each pin in bank 1, synchronized to HCLK, may be read here.

DESCRIPTION:

This register reflects the current values of all the bank 1 pins. The register accurately reflects the state of the pin regardless of the setting of the HW_PINCTRL_MUXSELx or HW_PINCTRL_DOEx registers, but generally, if it is desired to use a pin as a general purpose input, the pin's two bits in the HW_PINCTRL_MUXSELx register should be set to 3 (GPIO mode) and the pin's bit in the HW_PINCTRL_DOEx register should be set to 0 (disabled) to insure that the chip is not driving the pin.

EXAMPLE:

```
Empty Example.
```

37.4.34 PINCTRL Bank 2 Data Input Register Description

The current value of all bank 2 pins may be read from the PINCTRL Bank 2 Data Input Register.

HW_PINCTRL_DIN2	0x620
HW_PINCTRL_DIN2_SET	0x624
HW_PINCTRL_DIN2_CLR	0x628
HW_PINCTRL_DIN2_TOG	0x62C

Table 37-71. HW_PINCTRL_DIN2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DIN																															

Table 37-72. HW_PINCTRL_DIN2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**DIN**	RO	0x00000000	Each bit in this read-only register corresponds to one of the 32 pins in bank 2. The current state of each pin in bank 2, synchronized to HCLK, may be read here.

DESCRIPTION:

This register reflects the current values of all the bank 2 pins. The register accurately reflects the state of the pin regardless of the setting of the HW_PINCTRL_MUXSELx or HW_PINCTRL_DOEx registers, but generally, if it is desired to use a pin as a general purpose input, the pin's two bits in the HW_PINCTRL_MUXSELx register should be set to 3 (GPIO mode) and the pin's bit in the HW_PINCTRL_DOEx register should be set to 0 (disabled) to insure that the chip is not driving the pin.

EXAMPLE:

```
Empty Example.
```

37.4.35 PINCTRL Bank 0 Data Output Enable Register Description

The PINCTRL Bank 0 Output Enable Register controls the output enable signal for all pins in bank 0 that are configured for GPIO mode.

HW_PINCTRL_DOE0	0x700
HW_PINCTRL_DOE0_SET	0x704
HW_PINCTRL_DOE0_CLR	0x708
HW_PINCTRL_DOE0_TOG	0x70C

Table 37-73. HW_PINCTRL_DOE0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DOE																															

Table 37-74. HW_PINCTRL_DOE0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**DOE**	RW	0x00000000	Each bit in this register corresponds to one of the 32 pins in bank 0. Setting a bit in this register to one allows the chip to drive the corresponding pin in GPIO mode.

DESCRIPTION:

For pins in bank 0 that are configured as GPIOs, a 1 in this register will enable the corresponding bit value from HW_PINCTRL_DOUTxx register to be driven out the pin, and a 0 in this register will disable the corresponding driver.

EXAMPLE:

```
Empty Example.
```

37.4.36 PINCTRL Bank 1 Data Output Enable Register Description

The PINCTRL Bank 1 Output Enable Register controls the output enable signal for all pins in bank 1 that are configured for GPIO mode.

HW_PINCTRL_DOE1	0x710
HW_PINCTRL_DOE1_SET	0x714
HW_PINCTRL_DOE1_CLR	0x718
HW_PINCTRL_DOE1_TOG	0x71C

Table 37-75. HW_PINCTRL_DOE1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1	DOE																														

Table 37-76. HW_PINCTRL_DOE1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSRVD1**	RO	0x0	Reserved - write 0 to this bit-field.
30:0	**DOE**	RW	0x00000000	Each bit in this register corresponds to one of the 31 pins in bank 1. Setting a bit in this register to one allows the chip to drive the corresponding pin in GPIO mode.

DESCRIPTION:

For pins in bank 1 that are configured as GPIOs, a 1 in this register will enable the corresponding bit value from HW_PINCTRL_DOUTxx register to be driven out the pin, and a 0 in this register will disable the corresponding driver.

EXAMPLE:

```
Empty Example.
```

37.4.37 PINCTRL Bank 2 Data Output Enable Register Description

The PINCTRL Bank 2 Output Enable Register controls the output enable signal for all pins in bank 2 that are configured for GPIO mode.

HW_PINCTRL_DOE2	0x720
HW_PINCTRL_DOE2_SET	0x724
HW_PINCTRL_DOE2_CLR	0x728
HW_PINCTRL_DOE2_TOG	0x72C

Table 37-77. HW_PINCTRL_DOE2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
DOE																															

Table 37-78. HW_PINCTRL_DOE2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**DOE**	RW	0x00000000	Each bit in this register corresponds to one of the 32 pins in bank 2. Setting a bit in this register to one allows the chip to drive the corresponding pin in GPIO mode.

DESCRIPTION:

For pins in bank 2 that are configured as GPIOs, a 1 in this register will enable the corresponding bit value from HW_PINCTRL_DOUTxx register to be driven out the pin, and a 0 in this register will disable the corresponding driver.

EXAMPLE:

```
Empty Example.
```

37.4.38 PINCTRL Bank 0 Interrupt Select Register Description

The Bank 0 Interrupt Select register selects which of the bank 0 pins may be used as interrupt sources.

HW_PINCTRL_PIN2IRQ0	0x800
HW_PINCTRL_PIN2IRQ0_SET	0x804
HW_PINCTRL_PIN2IRQ0_CLR	0x808
HW_PINCTRL_PIN2IRQ0_TOG	0x80C

Table 37-79. HW_PINCTRL_PIN2IRQ0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PIN2IRQ																															

Table 37-80. HW_PINCTRL_PIN2IRQ0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**PIN2IRQ**	RW	0x00000000	Each bit in this register corresponds to one of the 32 pins in bank 0: 0= Deselect the pin's interrupt functionality. 1= Select the pin to be used as an interrupt source.

DESCRIPTION:

As described earlier in this chapter, any digital I/O pin can be used as an interrupt source. This register selects which pins in bank 0 can be used to generate interrupts. If the pin is selected in this register by setting its bit to 1, then detection of the correct level or edge on the pin (as chosen by the HW_PINCTRL_IRQLEVEL0 and HW_PINCTRL_IRQPOL0 registers) will set the corresponding bit in the HW_PINCTRL_IRQSTAT0 register. If the pin is additionally enabled in the HW_PINCTRL_IRQEN0 register, then the interrupt will be propagated to the interrupt collector as interrupt GPIO0.

For example, if this register contains 0x00000014, then pins GPIO0[2] and GPIO0[4] can be used as interrupt pins, and no other pins in bank 0 will cause bits to be set in the HW_PINCTRL_IRQSTAT0 register.

EXAMPLE:

```
Empty Example.
```

37.4.39 PINCTRL Bank 1 Interrupt Select Register Description

The Bank 1 Interrupt Select register selects which of the bank 1 pins may be used as interrupt sources.

HW_PINCTRL_PIN2IRQ1	0x810
HW_PINCTRL_PIN2IRQ1_SET	0x814
HW_PINCTRL_PIN2IRQ1_CLR	0x818
HW_PINCTRL_PIN2IRQ1_TOG	0x81C

Table 37-81. HW_PINCTRL_PIN2IRQ1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1	PIN2IRQ																														

Table 37-82. HW_PINCTRL_PIN2IRQ1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSRVD1**	RO	0x0	Reserved - write 0 to this bit-field.
30:0	**PIN2IRQ**	RW	0x00000000	Each bit in this register corresponds to one of the 31 pins in bank 1: 0= Deselect the pin's interrupt functionality. 1= Select the pin to be used as an interrupt source.

DESCRIPTION:

As described earlier in this chapter, any digital I/O pin can be used as an interrupt source. This register selects which pins in bank 1 can be used to generate interrupts. If the pin is selected in this register by setting its bit to 1, then detection of the correct level or edge on the pin (as chosen by the HW_PINCTRL_IRQLEVEL1 and HW_PINCTRL_IRQPOL1 registers) will set the corresponding bit in the HW_PINCTRL_IRQSTAT1 register. If the pin is additionally enabled in the HW_PINCTRL_IRQEN0 register, then the interrupt will be propagated to the interrupt collector as interrupt GPIO1.

For example, if this register contains 0x00000014, then pins GPIO1[2] and GPIO1[4] can be used as interrupt pins, and no other pins in bank 0 will cause bits to be set in the HW_PINCTRL_IRQSTAT1 register.

EXAMPLE:

```
Empty Example.
```

37.4.40 PINCTRL Bank 2 Interrupt Select Register Description

The Bank 2 Interrupt Select register selects which of the bank 2 pins may be used as interrupt sources.

HW_PINCTRL_PIN2IRQ2	0x820
HW_PINCTRL_PIN2IRQ2_SET	0x824
HW_PINCTRL_PIN2IRQ2_CLR	0x828

HW_PINCTRL_PIN2IRQ2_TOG 0x82C

Table 37-83. HW_PINCTRL_PIN2IRQ2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
PIN2IRQ																															

Table 37-84. HW_PINCTRL_PIN2IRQ2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**PIN2IRQ**	RW	0x00000000	Each bit in this register corresponds to one of the 32 pins in bank 2: 0= Deselect the pin's interrupt functionality. 1= Select the pin to be used as an interrupt source.

DESCRIPTION:

As described earlier in this chapter, any digital I/O pin can be used as an interrupt source. This register selects which pins in bank 2 can be used to generate interrupts. If the pin is selected in this register by setting its bit to 1, then detection of the correct level or edge on the pin (as chosen by the HW_PINCTRL_IRQLEVEL2 and HW_PINCTRL_IRQPOL2 registers) will set the corresponding bit in the HW_PINCTRL_IRQSTAT2 register. If the pin is additionally enabled in the HW_PINCTRL_IRQEN0 register, then the interrupt will be propagated to the interrupt collector as interrupt GPIO2.

For example, if this register contains 0x00000014, then pins GPIO2[2] and GPIO2[4] can be used as interrupt pins, and no other pins in bank 0 will cause bits to be set in the HW_PINCTRL_IRQSTAT2 register.

EXAMPLE:

```
Empty Example.
```

37.4.41 PINCTRL Bank 0 Interrupt Mask Register Description

The PINCTRL Bank 0 Interrupt Mask Register contains interrupt enable masks for the pins in bank 0.

HW_PINCTRL_IRQEN0 0x900
HW_PINCTRL_IRQEN0_SET 0x904
HW_PINCTRL_IRQEN0_CLR 0x908
HW_PINCTRL_IRQEN0_TOG 0x90C

Table 37-85. HW_PINCTRL_IRQEN0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IRQEN																															

Table 37-86. HW_PINCTRL_IRQEN0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**IRQEN**	RW	0x00000000	Each bit in this register corresponds to one of the 32 pins in bank 0: 1= Enable interrupts from the corresponding bit in HW_PINCTRL_IRQSTAT0. 0= Disable interrupts from the corresponding bit in HW_PINCTRL_IRQSTAT0.

DESCRIPTION:

As described earlier in this chapter, any digital I/O pin can be used as an interrupt source. This register masks the interrupt sources from the pins in bank 0. If a bit is set in this register and the same bit is set in HW_PINCTRL_IRQSTAT0, an interrupt will be propagated to the interrupt collector as interrupt GPIO0.

For example, if this register contains 0x00000014, then only bits 2 and 4 in HW_PINCTRL_IRQSTAT0 (corresponding to pins GPIO0[2] and GPIO0[4]) will cause interrupts from bank 0.

EXAMPLE:

```
Empty Example.
```

37.4.42 PINCTRL Bank 1 Interrupt Mask Register Description

The PINCTRL Bank 1 Interrupt Mask Register contains interrupt enable masks for the pins in bank 1.

HW_PINCTRL_IRQEN1	0x910
HW_PINCTRL_IRQEN1_SET	0x914
HW_PINCTRL_IRQEN1_CLR	0x918
HW_PINCTRL_IRQEN1_TOG	0x91C

Table 37-87. HW_PINCTRL_IRQEN1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1	IRQEN																														

Table 37-88. HW_PINCTRL_IRQEN1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSRVD1**	RO	0x0	Reserved - write 0 to this bit-field.
30:0	**IRQEN**	RW	0x00000000	Each bit in this register corresponds to one of the 31 pins in bank 1: 1= Enable interrupts from the corresponding bit in HW_PINCTRL_IRQSTAT1. 0= Disable interrupts from the corresponding bit in HW_PINCTRL_IRQSTAT1.

DESCRIPTION:

As described earlier in this chapter, any digital I/O pin can be used as an interrupt source. This register masks the interrupt sources from the pins in bank 1. If a bit is set in this register and the same bit is set in HW_PINCTRL_IRQSTAT1, an interrupt will be propagated to the interrupt collector as interrupt GPIO1.

For example, if this register contains 0x00000014, then only bits 2 and 4 in HW_PINCTRL_IRQSTAT1 (corresponding to pins GPIO1[2] and GPIO1[4]) will cause interrupts from bank 1.

EXAMPLE:

```
Empty Example.
```

37.4.43 PINCTRL Bank 2 Interrupt Mask Register Description

The PINCTRL Bank 2 Interrupt Mask Register contains interrupt enable masks for the pins in bank 2.

HW_PINCTRL_IRQEN2	0x920
HW_PINCTRL_IRQEN2_SET	0x924
HW_PINCTRL_IRQEN2_CLR	0x928
HW_PINCTRL_IRQEN2_TOG	0x92C

Table 37-89. HW_PINCTRL_IRQEN2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IRQEN																															

Table 37-90. HW_PINCTRL_IRQEN2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**IRQEN**	RW	0x00000000	Each bit in this register corresponds to one of the 32 pins in bank 2: 1= Enable interrupts from the corresponding bit in HW_PINCTRL_IRQSTAT2. 0= Disable interrupts from the corresponding bit in HW_PINCTRL_IRQSTAT2.

DESCRIPTION:

As described earlier in this chapter, any digital I/O pin can be used as an interrupt source. This register masks the interrupt sources from the pins in bank 2. If a bit is set in this register and the same bit is set in HW_PINCTRL_IRQSTAT2, an interrupt will be propagated to the interrupt collector as interrupt GPIO2.

For example, if this register contains 0x00000014, then only bits 2 and 4 in HW_PINCTRL_IRQSTAT2 (corresponding to pins GPIO2[2] and GPIO2[4]) will cause interrupts from bank 2.

EXAMPLE:

```
Empty Example.
```

37.4.44 PINCTRL Bank 0 Interrupt Level/Edge Register Description

The PINCTRL Bank 0 Interrupt Level/Edge Register selects level or edge sensitivity for interrupt requests for the pins in bank 0.

HW_PINCTRL_IRQLEVEL0	0xa00
HW_PINCTRL_IRQLEVEL0_SET	0xa04
HW_PINCTRL_IRQLEVEL0_CLR	0xa08
HW_PINCTRL_IRQLEVEL0_TOG	0xa0C

Table 37-91. HW_PINCTRL_IRQLEVEL0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IRQLEVEL																															

Table 37-92. HW_PINCTRL_IRQLEVEL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**IRQLEVEL**	RW	0x00000000	Each bit in this register corresponds to one of the 32 pins in bank 0: 1= Level detection; 0= Edge detection.

DESCRIPTION:

This register selects level or edge detection for interrupt generation. Each pin in bank 0 that is configured for interrupt generation can be independently set to interrupt on low level, high level, rising edge, or falling edge by setting bits in this register and HW_PINCTRL_IRQPOL0 (see below) appropriately.

EXAMPLE:

```
Empty Example.
```

37.4.45 PINCTRL Bank 1 Interrupt Level/Edge Register Description

The PINCTRL Bank 1 Interrupt Level/Edge Register selects level or edge sensitivity for interrupt requests for the pins in bank 1.

HW_PINCTRL_IRQLEVEL1	0xa10
HW_PINCTRL_IRQLEVEL1_SET	0xa14
HW_PINCTRL_IRQLEVEL1_CLR	0xa18
HW_PINCTRL_IRQLEVEL1_TOG	0xa1C

Table 37-93. HW_PINCTRL_IRQLEVEL1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1	IRQLEVEL																														

Table 37-94. HW_PINCTRL_IRQLEVEL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSRVD1**	RO	0x0	Reserved - write 0 to this bit-field.
30:0	**IRQLEVEL**	RW	0x00000000	Each bit in this register corresponds to one of the 31 pins in bank 1: 1= Level detection; 0= Edge detection.

DESCRIPTION:

This register selects level or edge detection for interrupt generation. Each pin in bank 1 that is configured for interrupt generation can be independently set to interrupt on low level, high level, rising edge, or falling edge by setting bits in this register and HW_PINCTRL_IRQPOL1 (see below) appropriately.

EXAMPLE:

```
Empty Example.
```

37.4.46 PINCTRL Bank 2 Interrupt Level/Edge Register Description

The PINCTRL Bank 2 Interrupt Level/Edge Register selects level or edge sensitivity for interrupt requests for the pins in bank 2.

HW_PINCTRL_IRQLEVEL2	0xa20
HW_PINCTRL_IRQLEVEL2_SET	0xa24
HW_PINCTRL_IRQLEVEL2_CLR	0xa28
HW_PINCTRL_IRQLEVEL2_TOG	0xa2C

Table 37-95. HW_PINCTRL_IRQLEVEL2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IRQLEVEL																															

Table 37-96. HW_PINCTRL_IRQLEVEL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**IRQLEVEL**	RW	0x00000000	Each bit in this register corresponds to one of the 32 pins in bank 2: 1= Level detection; 0= Edge detection.

DESCRIPTION:

This register selects level or edge detection for interrupt generation. Each pin in bank 2 that is configured for interrupt generation can be independently set to interrupt on low level, high level, rising edge, or falling edge by setting bits in this register and HW_PINCTRL_IRQPOL2 (see below) appropriately.

EXAMPLE:

```
Empty Example.
```

37.4.47 PINCTRL Bank 0 Interrupt Polarity Register Description

The PINCTRL Bank 0 Interrupt Polarity Register selects the polarity for interrupt requests for the pins in bank 0.

HW_PINCTRL_IRQPOL0	0xb00
HW_PINCTRL_IRQPOL0_SET	0xb04
HW_PINCTRL_IRQPOL0_CLR	0xb08
HW_PINCTRL_IRQPOL0_TOG	0xb0C

Table 37-97. HW_PINCTRL_IRQPOL0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IRQPOL																															

Table 37-98. HW_PINCTRL_IRQPOL0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**IRQPOL**	RW	0x00000000	Each bit in this register corresponds to one of the 32 pins in bank 0: 0= Low or falling edge; 1= High or rising edge.

DESCRIPTION:

This register selects the polarity for interrupt generation. Each pin in bank 0 which is configured for interrupt generation can be independently set to interrupt on low level, high level, rising edge, or falling edge by setting this register and HW_PINCTRL_IRQLEVEL0 (see above) appropriately.

EXAMPLE:

```
Empty Example.
```

37.4.48 PINCTRL Bank 1 Interrupt Polarity Register Description

The PINCTRL Bank 1 Interrupt Polarity Register selects the polarity for interrupt requests for the pins in bank 1.

HW_PINCTRL_IRQPOL1	0xb10
HW_PINCTRL_IRQPOL1_SET	0xb14
HW_PINCTRL_IRQPOL1_CLR	0xb18
HW_PINCTRL_IRQPOL1_TOG	0xb1C

Table 37-99. HW_PINCTRL_IRQPOL1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1	IRQPOL																														

Table 37-100. HW_PINCTRL_IRQPOL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSRVD1**	RO	0x0	Reserved - write 0 to this bit-field.
30:0	**IRQPOL**	RW	0x00000000	Each bit in this register corresponds to one of the 31 pins in bank 1: 0= Low or falling edge; 1= High or rising edge.

DESCRIPTION:

This register selects the polarity for interrupt generation. Each pin in bank 1 which is configured for interrupt generation can be independently set to interrupt on low level, high level, rising edge, or falling edge by setting this register and HW_PINCTRL_IRQLEVEL1 (see above) appropriately.

EXAMPLE:

```
Empty Example.
```

37.4.49 PINCTRL Bank 2 Interrupt Polarity Register Description

The PINCTRL Bank 2 Interrupt Polarity Register selects the polarity for interrupt requests for the pins in bank 2.

HW_PINCTRL_IRQPOL2	0xb20
HW_PINCTRL_IRQPOL2_SET	0xb24
HW_PINCTRL_IRQPOL2_CLR	0xb28
HW_PINCTRL_IRQPOL2_TOG	0xb2C

Table 37-101. HW_PINCTRL_IRQPOL2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IRQPOL																															

Table 37-102. HW_PINCTRL_IRQPOL2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**IRQPOL**	RW	0x00000000	Each bit in this register corresponds to one of the 32 pins in bank 2: 0= Low or falling edge; 1= High or rising edge.

DESCRIPTION:

This register selects the polarity for interrupt generation. Each pin in bank 2 which is configured for interrupt generation can be independently set to interrupt on low level, high level, rising edge, or falling edge by setting this register and HW_PINCTRL_IRQLEVEL2 (see above) appropriately.

EXAMPLE:

```
Empty Example.
```

37.4.50 PINCTRL Bank 0 Interrupt Status Register Description

The PINCTRL Bank 0 Interrupt Status Register reflects pending interrupt status for the pins in bank 0.

HW_PINCTRL_IRQSTAT0	0xc00
HW_PINCTRL_IRQSTAT0_SET	0xc04
HW_PINCTRL_IRQSTAT0_CLR	0xc08
HW_PINCTRL_IRQSTAT0_TOG	0xc0C

Table 37-103. HW_PINCTRL_IRQSTAT0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IRQSTAT																															

Table 37-104. HW_PINCTRL_IRQSTAT0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**IRQSTAT**	RW	0x00000000	Each bit in this register corresponds to one of the 32 pins in bank 0: 0= No interrupt pending; 1= Interrupt pending.

DESCRIPTION:

This register reflects the pending interrupt status for pins in bank 0. Bits in this register are automatically set by hardware when an interrupt condition (level high, level low, rising edge, or falling edge) occurs on a bank 0 pin which has been enabled as an interrupts source in the HW_PINCTRL_PIN2IRQ0 register. Software may clear any bit in this register by writing a 1 to the bit at the SCT clear address, e.g., HW_PINCTRL_IRQSTAT0_CLR. Status bits for pins configured as level sensitive interrupts cannot be cleared unless either the actual pin is in the non-interrupting state, or the pin has been disabled as an interrupt source by clearing its bit in HW_PINCTRL_PIN2IRQ0.

If a bit is set in this register, and the corresponding bit is also set in the HW_PINCNTRL_IRQEN0 mask register, then the GPIO0 interrupt will be asserted to the interrupt collector.

EXAMPLE:

```
Empty Example.
```

37.4.51 PINCTRL Bank 1 Interrupt Status Register Description

The PINCTRL Bank 1 Interrupt Status Register reflects pending interrupt status for the pins in bank 1.

HW_PINCTRL_IRQSTAT1	0xc10
HW_PINCTRL_IRQSTAT1_SET	0xc14
HW_PINCTRL_IRQSTAT1_CLR	0xc18
HW_PINCTRL_IRQSTAT1_TOG	0xc1C

Table 37-105. HW_PINCTRL_IRQSTAT1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1	IRQSTAT																														

Table 37-106. HW_PINCTRL_IRQSTAT1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSRVD1**	RO	0x0	Reserved - write 0 to this bit-field.
30:0	**IRQSTAT**	RW	0x00000000	Each bit in this register corresponds to one of the 31 pins in bank 1: 0= No interrupt pending; 1= Interrupt pending.

DESCRIPTION:

This register reflects the pending interrupt status for pins in bank 1. Bits in this register are automatically set by hardware when an interrupt condition (level high, level low, rising edge, or falling edge) occurs on a bank 1 pin which has been enabled as an interrupts source in the HW_PINCTRL_PIN2IRQ1 register. Software may clear any bit in this register by writing a 1 to the bit at the SCT clear address, e.g., HW_PINCTRL_IRQSTAT1_CLR. Status bits for pins configured as level sensitive interrupts cannot be cleared unless either the actual pin is in the non-interrupting state, or the pin has been disabled as an interrupt source by clearing its bit in HW_PINCTRL_PIN2IRQ1.

If a bit is set in this register, and the corresponding bit is also set in the HW_PINCNTRL_IRQEN1 mask register, then the GPIO1 interrupt will be asserted to the interrupt collector.

EXAMPLE:

```
Empty Example.
```

37.4.52 PINCTRL Bank 2 Interrupt Status Register Description

The PINCTRL Bank 2 Interrupt Status Register reflects pending interrupt status for the pins in bank 2.

HW_PINCTRL_IRQSTAT2	0xc20
HW_PINCTRL_IRQSTAT2_SET	0xc24
HW_PINCTRL_IRQSTAT2_CLR	0xc28
HW_PINCTRL_IRQSTAT2_TOG	0xc2C

Table 37-107. HW_PINCTRL_IRQSTAT2

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
IRQSTAT																															

Table 37-108. HW_PINCTRL_IRQSTAT2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**IRQSTAT**	RW	0x00000000	Each bit in this register corresponds to one of the 32 pins in bank 2: 0= No interrupt pending; 1= Interrupt pending.

DESCRIPTION:

This register reflects the pending interrupt status for pins in bank 2. Bits in this register are automatically set by hardware when an interrupt condition (level high, level low, rising edge, or falling edge) occurs on a bank 2 pin which has been enabled as an interrupts source in the HW_PINCTRL_PIN2IRQ2 register. Software may clear any bit in this register by writing a 1 to the bit at the SCT clear address, e.g., HW_PINCTRL_IRQSTAT2_CLR. Status bits for pins configured as level sensitive interrupts cannot be cleared unless either the actual pin is in the non-interrupting state, or the pin has been disabled as an interrupt source by clearing its bit in HW_PINCTRL_PIN2IRQ2.

If a bit is set in this register, and the corresponding bit is also set in the HW_PINCNTRL_IRQEN2 mask register, then the GPIO2 interrupt will be asserted to the interrupt collector.

EXAMPLE:

```
Empty Example.
```

PINCTRL Block pinctrl, Revision 1.19

Chapter 38 Digital Video Encoder Programmers' Manual

38.1 Functional Overview

The DVE receives pixel data in 8-bit Cb,Cr,Y coding representing either interlaced or progressive image and produces 10-bit outputs to drive video DACs. A number of different NTSC and PAL formats are supported. The input data can be received on a single 8-bit port clocked at 27 MHz in CbYCrY order or on a pair of such ports, one luma and one chroma, both clocked at 27MHz. Only the latter can support the data rate for progressive. When the 16-bit mode is used for interlaced input, the data is expected to change at 13.5 MHz, but it is still clocked in at 27 MHz.

Synchronization of internal line and pixel counters to the incoming stream can use input strobes (horizontal and vertical/field), or embedded D1 SAV and EAV codes. Alternatively, the internal counters can free-run and output horizontal and vertical, and field signals can be generated to provide sync to the picture source.

The video output in interlaced mode always includes the composite, CVBS, signal. The remaining three video output channels can provide the three color components of component-video or can provide the Y and C signals for S-video. The color components are produced by a fully programmable matrixing. Thus either RGB, YUV, YPbPc or some other combination can be produced. A sync signal can optionally be inserted on one or more of these component signals. (This sync signal can include, as desired, closed caption, CGMS, WSS and macrovision elements.)

The DVE is controlled through 21 read/writeable registers, most of which are 32-bits wide or close to it. These are addressed from the ASIC's host_interface via 32-bit wide data ports and a 4-bit address. Since there are more than 16 registers, an indirect addressing scheme is necessary.

The registers provide complete control over the video filtering, output format, and synchronization. Closed caption is supported with data writes to a dedicated register to enter the byte-pair to be sent on line 21 of either field. Handshaking signals are provided to permit timely entry of the data for one or both fields. CGMS (NTSC interlaced and progressive) and WSS (interlaced PAL) are supported by writes to another register that includes enable bits and the 14-bit payload. Macrovision is fully supported both for interlaced and progressive modes.

38.2 Block Diagram and Implementation Overview

Figure 38-1 is a block diagram of the DVE. Discussion of the individual blocks follows. the figure.

This page ɔration.

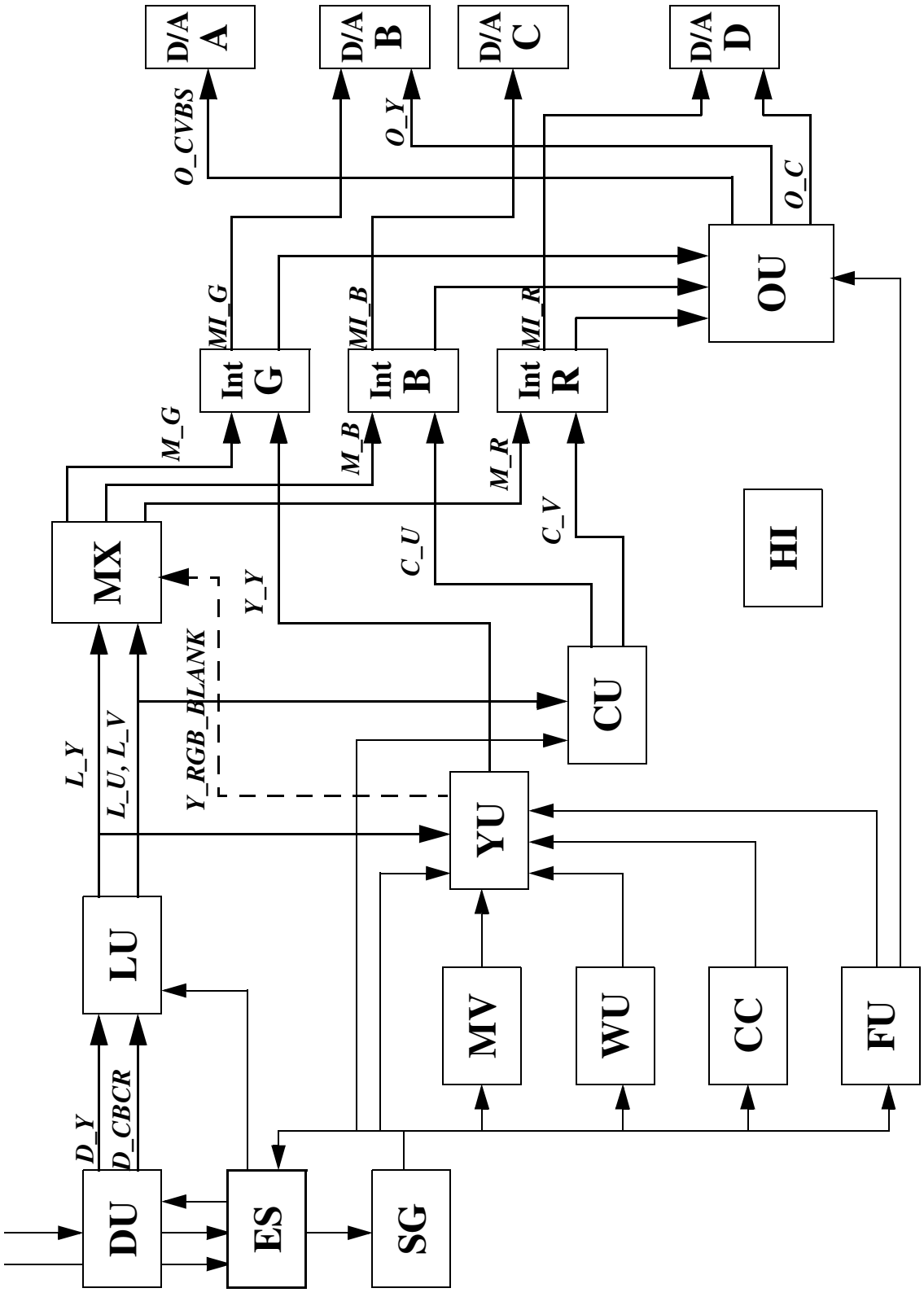


Figure 38-1. Block Diagram of DVE

This page ɔration.

38.2.1 DU -- Data Input Unit

This block receives the input video data, separates luma from chroma and sends them to the L_unit. If enabled, it generates color bars instead. In the case of the D1 input mode, it parses the control codes and generates appropriate horizontal and vertical blanking signals.

38.2.2 ES -- External Sync Unit

This block receives the input sync signals (slave mode) or the sync signals generated by DU (D1 mode) and creates the horizontal and vertical reset signals that drive the sync block. These can be derived from rising or falling edges of input blanking signals, can be delayed by programmable amounts and accommodate the fixed pipeline-delay of the L-unit so that the sync signals produced by SG are defined in relation to the outputs of the L-unit.

38.2.3 SG -- Sync Generation Unit

This block produces all the major timing and synchronization signals including sync, blanking and active video. The signals required to format closed caption and macrovision are also produced. Additional timing for special purposed is based on pixel and line counts outputted by this block. Signals used to format the activities of other blocks are shown in the diagram as unlabeled lines. In addition, all blocks use a pair of 27-MHz signals that toggle at 13.5 and 6.75 MHz rates. These are used to strobe interlaced pixels and to distinguish the color components, Cb or U and Cr or V.

38.2.4 FU -- Frequency Generation Unit

This block generates the color subcarrier and also the sinusoid used in closed caption lead-in. The frequency and phase of the color subcarrier are programmable; the frequency of the closed caption waveform is hard-wired.

38.2.5 LU -- Low-pass and Other Signal Conditioning Filter Unit

This block separates and up-samples the color components, performs low-pass filtering on both luma and chroma, and performs sharpness enhancement on the luma.

38.2.6 MX -- RGB Matrix Unit

This block matrixes the filtered Y,U, and V to produce RGB or YPbPr component outputs. The 12-bit matrix coefficients are fully programmable on the range [-4,4). (N.B., one of the YUV to RGB matrix coefficients is greater than 2.0.) A signal is received from YU (dashed line labeled Y_RGB_BLANK in the diagram) that comprises sync, blanking and other signals (e.g., CGMS, macrovision, etc.) that are placed on the Y signal in CVBS or S-video. Under program control these luma-related signals can be imposed on one or more of the component outputs (e.g., Y of YPbPr).

This page ›ration.

38.2.7 YU -- Y(luma)-main Unit

This block produces the interlaced luma output (apart from interpolation to twice the pixel rate.) It scales the luma as appropriate for NTSC or PAL (rather for 714:286 vs 700:300 systems), and adds it to a base signal containing sync, blanking, pedestal and various special signals. It edge-shapes so that rise/fall constraints are met on sync edges, special signals and envelopes. This base signal is sent to the MX block, as discussed above, except that there may be some differences between the version appropriate for component and composite signals. (E.g, component might be 700:300 and composite 714:286.)

38.2.8 CU -- Chroma-main Unit

This block produces the interlaced, base-band chroma output (apart from interpolation to twice the pixel rate.) It scales the chroma as appropriate for NTSC or PAL and adds the base-band form of the color burst. It insures that rise/fall constraints are met on burst edges and signal envelope.

38.2.9 Int -- Interpolation Block

There are three copies of this block. In progressive mode, they interpolate the 3 component signals to 54 MHz. In interlaced modes, they interpolate both the component and composite signals to 27 MHz. They do this by time-share multiplexing the two groups of signals. Tthe main filter pipeline runs at 54 MHz in the progressive and 27 MHz in the interlaced case..

38.2.10 OU -- (Composite) Output Unit

This block produces the chroma part of the composite video signal by multiplying in-phase and quadrature base-band chroma signals by the corresponding sinusoids received from FU. The final composite output is then obtained by adding in the interpolated luma signal.

38.2.11 D/A -- D/A Selection Muxes

There are 4 of these, one for each DAC output. As shown in the diagram, DAC-A outputs composite in interlaced or 0 in progressive modes. The others can output either component video or S-video under program control. (In the latter case, only 2 of the 3 are needed, so "D/A-C" continues to output one component output. Despite its being labeled as "B," it could in fact be any linear combination of YUV or even just Y_RGB_BLANK.

38.2.12 MV -- Macrovision Unit

This block supports macrovision and will not be discussed in detail here.

38.2.13 WU -- WSS and CGMS Unit

This block supports WSS for interlaced-PAL (625-line systems) and CGMS for both interlaced and progressive NTSC (525-line systems.) In both cases a special frequency is generated by phase accumulation and the requisite signal constructed on the proper line. The WSS signal is 14-data bits (bi-phase encoded) following a fixed run-in and start-code sequence. The CGMS is 14-data bits following

This page ɔration.

a leading "10" pair and followed by a 6-bit CRC. The CRC is generated by logic in the WU block. Thus apart from enables, the payload in each case is 14-bits which are programmed via the host interface.

38.2.14 CC -- Closed Caption Unit

The timing signals and clock for the closed caption are produced by the SU. The function of the cc-block is simply to enable closed caption output on top and/or bottom field and to receive and shift out the appropriate 16-bit data words.

38.2.15 HI -- Host Interface Unit

This block holds the registers that are written by the external host to set-up and control the DVE. All registers can be read as well as written. Many registers have default values that can be used simply by identifying the picture format (e.g, NTSC, PAL-B, etc.), of which 8 are pre-defined. Other registers provide support for macrovision, closed-caption, WSS or CGMS. The host interfaces with this set of registers via a 32-bit data bus and a 4-bit address bus, plus strobe and R/W control.

38.3 Registers

The host reads and writes registers in the HI_unit, "HI" in Figure 38-1. The DVE pins prefixed "hi_" are inputs to this unit used to write its registers. The signals prefixed "dv_hi" are driven by the HI_unit, as is dv_illeg_acc. Table 38-1 lists the registers writeable by the host in the DVE. An address such as "3" means that this register is written directly at address 3. An address such as "8.1" means that this register is written in a two step operation: first register "7" is written with the "subaddress", "1" in the example. Then the desired register value is written to address "8." This indirect addressing is necessary because there are more registers than the 16 permitted by a 4-bit address. Most registers concatenate a number of fields having different functions. The field column indicates what portion of the corresponding address holds the data described in the right column. Fields named with an initial "H_" are outputs of the HI_unit with that name.

Unless otherwise noted fields marked "reserved" are "don't care" when written and are read as 0.

Table 38-1. Registers in the HI_unit Writeable by Host

Address	Field	Register/Field Contents
0	9:0	H_cnfg_s[9:0]: vector of controls for the ES block (see detailed description below)
	11:10	H_clk_phs[1:0]: sent to SG block; permits adjusting phase of pixel clock set at line-end
	13:12	H_cc_enbl[1:0] : closed caption enable (line 21) [0]: enables insertion in odd (top) field [1]: enables insertion in even (bottom) field
	14	H_color_bar_en: enable insertion of internally generated color bars

This page ›ration.

Table 38-1. Registers in the HI_unit Writeable by Host (continued)

Address	Field	Register/Field Contents
	23:15	H_cnfg_l[8:0]: vector of controls for the LU block (see detailed description below)
	25:24	H_cgain[1:0]: controls chroma gain for composite 00: NTSC gain 01: PAL gain 1x: no gain
	30:26	H_cnfg_y[4:0]: vector of controls for the YU block (see detailed description below)
	31	default_picform
1	9:0	H_cnfg_m[9:0]: vector of controls for the MX block (see detailed description below)
	10	H_Svideo: enables S-video output on DACs B and D
	13:11	H_ydel_adj: delays luma versus chroma for composite output. Luma lags chroma by - 4 + H_ydel_adj cycles of 27 MHz clock In other words, if H_ydel_adj is zero, the luma leads by 4 cycles and if H_ydel_adj is 7, luma lags by 3 cycles
	14	reserved
	15	H_ysharp_bw: controls the filter bandwidth inside the ysharp block (see detailed description below)
	31:16	reserved
2	9:0	H_HLC[9:0]: pixel count (pixels per line minus 1)
	19:10	H_VSO[9:0]: offset of internal vertical (frame) reset from external vertical sync
	30:20	H_HSO[10:0]: offset of internal horizontal (line) reset from external horizontal sync
	31	reserved
3	31:0	H_phase_inc[31:0]: defines the frequency of the color subcarrier (see below)
4	31:0	H_phase_offset[31:0]: phase offset for the color subcarrier
5	13:0	wss_cgms_data[13:0]: payload for either wss or cgms
	14	cgms_enbl: enable cgms
	15	wss_enbl: enable wss
	31:16	reserved
6	15:0	cc_data[15:0]: data to be outputted on line 21 (see discussion below)

This page ›ration.

Table 38-1. Registers in the HI_unit Writeable by Host (continued)

Address	Field	Register/Field Contents
	17:16	cc_fill[1:0]: flags to control and monitor data insertion on upper and lower fields
	31:18	reserved
7	3:0	subaddr[3:0]: sets subaddress (after ".") in accessing next three register groups
	31:4	reserved
8.0	11:0	bcoef_cb[11:0]: cb coefficient in b output
	23:12	bcoef_cr[11:0]: cr coefficient in b output
	31:24	bcoef_y[7:0]: y coefficient in b output
8.1	3:0	bcoef_y[11:8]: y coefficient in b output
	15:4	gcoef_cb[11:0]: cb coefficient in g output
	27:16	gcoef_cr[11:0]: cr coefficient in g output
	31:28	gcoef_y[3:0]: y coefficient in g output
8.2	7:0	gcoef_y[11:4]: y coefficient in g output
	19:8	rcoef_cb{11:0]: cb coefficient in r output
	31:20	rcoef_cr[11:0]: cr coefficient in r output
8.3	11:0	rcoef_y[11:0]: y coefficient in r output
	19:12	H_PBA[7:0]: defines V component of color burst
	27:20	H_NBA[7:0]: defines U component of color burst
	31:28	reserved
9.0	31:0	{N7,N6,N5,N4,N3,N2,N1}: registers defined in macrovision specs
9.1	31:0	{N11[13:0],N10,N9,N8}: registers defined in macrovision specs
9.2	31:0	{N14,N13,N12,N11[14]}: registers defined in macrovision specs
9.3	31:0	{N0,N20,N19,N18,N17,N16,N15}: registers defined in macrovision specs
9.4	10:0	{N22,N21}: registers defined in macrovision specs
	11	reserved
	19:12	macv_tst[7:0]: reserved -- must be written to 0
	31:20	reserved
10.0	9:0	sync_strt[9:0]: start of sync pulse in line or half-line (pixel count less 1)

This page ›ration.

Table 38-1. Registers in the HI_unit Writeable by Host (continued)

Address	Field	Register/Field Contents
	19:10	sync_end[9:0]: normal end of sync pulse in first half-line of video line (pixel count less 1)
	29:20	sync_srend[9:0]: end of sync pulse in each half line in serration region of vertical blanking (pixel count less 1)
	31:30	reserved
10.1	9:0	sync_eqend[9:0]: end of sync puls in each half line in equalization regions of vertical blanking (pixel count less 1)
	19:10	actv_strt[9:0]: (horizontal) start of active video (pixel count less 1)
	29:20	actv_end[9:0]: (horizontal) end of active video (pixel count less 1)
	31:30	reserved
10.2	9:0	nbrst_strt[9:0]: start of normal color burst (pixel count less 1)
	19:10	wbrst_strt[9:0]: start of wide color burst for macrovision (pixel count less 1)
	29:20	brst_end[9:0]: end of normal or wide color burst (pixel count less 1)
	31:30	reserved
10.3	5:0	vstrt_subph[5:0]: last half-line of post-equalization; followed by "sub-phase" -- vertical blanking lines after post-equalization
	11:6	vstrt_actv[5:0]: last half-line of sub_phase; followed by active video
	21:12	vstrt_preeq[9:0]: last half-line of active video; followed by pre-equalization
	31:22	last_fld_ln[9:0]: last half-line of field -- usually same as vstrt_preeq
10.4	5:0	vstrt_serra[5:0]: last half-line of pre-equalization; followed by serration
	11:6	vstrt_posteq[5:0]:last half-line of serration; followed by post-equalization
	13:12	H_y_blank_ctrl[1:0]: goes to YU (see discussion below)
	14	H_cs_invert_ctrl: goes to MV (see discussion below)
	16:15	H_agc_lvl_ctrl[1:0]: goes to MV (see discussion below)
	18:17	H_bruchb[1:0]: goes to SG (see discussion below)
	20:19	H_fsc_phase_rst[1:0]: goes to SG (see discussion below)

This page ɔration.

Table 38-1. Registers in the HI_unit Writeable by Host (continued)

Address	Field	Register/Field Contents
	21	H_pal_fsc_phase_alt: goes to SG (see discussion below)
	22	H_ntsc_ln_cnt: goes to SG (see discussion below)
	31:23	H_lpf_rst_off[8:0]: goes to ES (see discussion below)

Apart from subaddr[3:0], discussed above, and default_picform which is used within the HI unit and will be discussed below, the register fields without a "H_" prefix" are assembed into larger buses for distribution. These buses are identified as follows:

H_wss_cgms_word = {wss_enbl,cgms_enbl,wss_cgms_data[13:0]}

H_cc_word = {cc_fill,cc_data[15:0]}

H_mx_coef_bus ={rcoef_y, rcoef_cr, rcoef_cb, gcoef_y, gcoef_cr, gcoef_cb, bcoef_y, bcoef_cr, bcoef_cb}

H_MACV_SU = {N20,N19,N18,N17,N16}

H_MACV_EU={N21,N15,N14,N13,N12,N11,N10,N9,N8,N7,N6,N5,N4,N3,N2,N1}

H_MACV_CTRL = N0

H_MACV_RGB = N22

H_PICFORM[133:0] = {vstrt_posteq, vstrt_serra, ast_fld_ln, vstrt_preeq, vstrt_actv, vstrt_subph, brst_end, wbrst_strt, nbrst_strt, actv_end, actv_strt, sync_eqend, sync_srend, sync_end, sync_strt}

H_wss_cgms_word combines the controls and data for CGMS/WSS and is sent to WU.

H_cc_word combines the controls and data for closed caption and is sent to CC.

H_mx_coef_bus combines all the matrix coefficient for creating the component outputs and is sent to MX.

H_MACV_SU contains those macrovision registers that relate to generation of timing signals and is sent to SG.

H_MACV_EU contains most of the other macrovision registers and is sent to MV.

H_MACV_CTRL holds the master control register for macrovision and is sent to MV.

H_MACV_RGB holds a special control bit for insertion of macrovision on the component output and is sent to YU.

H_PICFORM[133:0] combines all the parameters (apart from H_HLC) for defining the video format -- half-lines per field and the locations of sync, blanking, burst and active video. It is sent to SG.

This page ›ration.

38.4 Function and Programming of Controls

38.4.1 Register 0

In register 0, H_cnfg_s[9:0], contains the following signals:

fsync_enbl = H_cnfg_s[9],

fsync_phs = H_cnfg_s[8],

hsync_phs = H_cnfg_s[7],

vsync_phs = H_cnfg_s[6],

T_SYNC_MODE = H_cnfg_s[5:3];

T_ENCD_MODE = H_cnfg_s[2:0].

T_ENCD_MODE identifies the video mode:

000: NTSC-M Mode

001: PAL-B Mode

010: PAL-M Mode

011: PAL-N Mode

100: PAL-CN Mode

101: NTSC with 700:300 scaling on "G"

110: PAL-60 Mode

111: NTSC progressive

T_SYNC_MODE identifies the manner in which the input is synchronized to the display

000: Ext slave: 8-bit Y/C in, SYNC in

001: Ext slave: 16-bit Y/C in, SYNC in

010: Master: 8-bit Y/C in, SYNC out

011: Master: 16-bit Y/C in, SYNC out

1xx: D1 mode: 8-bit Y/C in, SYNC out

In external-sync ("slave") mode, the rising edge of an external P_HSYNC_IN is used to derive the horizontal sync for the DVE block if hsync_phs = 0; otherwise the falling edge is used.. Similarly vsync_phs, selects the active edge of P_VSYNC_IN to generate the internal vertical timing in external sync mode. The actual horizontal reset is delayed relative to the designated edge of P_HSYNC_IN by H_HSO cycles of the 27-MHz clock, and the vertical reset is delayed by H_VSO lines relative to P_VSYNC_IN. (H_HSO and H_VSO are in register 2.) (In D1 mode, horizontal sync is taken from the leading edge of EAV and vertical sync from that of SFB. H_HSO and H_VSO are not used to delay the horizontal and vertical resets in this case.)

This page ɔration.

The fsync_enbl and fsync_phs bits relate the internal field polarity to those of the input and/or output signals in interlaced modes. In the ASIC application, fsync_enbl can be taken to be 0; fsync_phs can be set to 1 for NTSC and to 0 for PAL.

The 27 MHz clock is divided to generate 13.5 and 6.75 MHz timing signals. In 8-bit input mode these distinguish Cb,Y1,Cr,Y2. The H_clk_phs[1:0] can be used to adjust this phase at the beginning of a line. The value 00 has been used in all cases simulated to date.

Closed caption is enabled for the NTSC upper field by H_cc_enbl[0] and for the lower field by H_cc_enbl[1]. (H_cc_word[17:0] is used to by the host to program the specific closed caption data, field by field.)

When H_color_bar_en = 1, internally generated color bars are used as the video source instead of the video input data.

The filtering of the video input is controlled by H_cnfg_l[8:0] = {ys_gainsel[1:0], ys_gainsgn, coefsel_clpf, ylpf_coefsel, sel_ysharp, sel_clpf, sel_ylpf, yd_offsetsel}. These parameters have the following significance:

ys_gainsel[1:0] controls the degree of luma sharpness enhancement by luma sharpness filter: 00=3dB, 01=6dB, 10=9dB, 11=12dB.

ys_gainsgn controls the sign of sharpness modification, 1 = negative, 0 = positive.

coefsel_clpf controls the chroma low pass filter bandwidth: 1=0.6MHz, 0=1.3MHz.

ylpf_coefsel controls the luma low pass filter bandwidth: 1=4.2MHz, 0=5.5MHz.

sel_ysharp enables the luma sharpness filter.

sel_clpf enables the chroma low pass filter

sel_ylpf enables the luma low pass filter

yd_offsetsel controls the luma offset: 1 = subtract 16 from luma, 0 = do not subtract

The filtering is discussed further in a subsequent section.

The next two fields mainly affect the composite output: H_cnfg_y[4:0] acts on the luma and H_cgain[1:0] on the chroma:

cgain[1:0] controls the chroma gain: 00 = NTSC, 01 = PAL, 1x = no gain.

H_cnfg_y consists of the following fields:

tst_ygain_sel[1:0] = H_cnfg_y[1:0] controls the luma gain: 00 = NTSC, 01 = PAL, 1x = no gain.

no_ped = H_cnfg_y[2]can be set to prevent insertion of a black pedestal as required by NTSC-J

pal_shape = H_cnfg_y[3] is set to impose a 250 nSec edge shape as required by PAL, otherwise the steeper edges specified by NTSC are used.

add_YPbPr_ped = H_cnfg_y[4] permits inserting a black pedestal when sync is inserted on one (or more) of the component signals.

This page ɔration.

Finally, the msb, "default_picform," permits use of a set of default parameters, tailored to the mode defined by T_ENCD_MODE to be used in place of the registers a.0-a.4. This will be discussed further below.

38.4.2 Register 1

Register 1, holds additional key controls that must always be specified.

H_cnfg_m controls the offset of the components signals. (The rgbmatrix outputs are signed 10-bit numbers. An offset is required to make the DAC inputs positive.) The offset can be either a fixed number or a sync/banking signal generated for this purpose by YU labeled Y_RGB_BLANK in Figure 1..

H_cnfg_m = {g_use_sync,rb_use_sync,rgb_blank_val[7:0]}

g_use_sync: use the sync/blanking signal to offset the green output (This is the Y of YPbPr if the matrix coefficents are the unit matrix.)

rb_use_sync: use the sync/blanking signal to offset the red and blue outputs.

rgb_blank_va[7:0]: this defines the fixed offset used when the sync/blanking signal is not used as the offset. (This 8-bit value is multipled by 4 to create the 10-bit positive offset.)

The other controls in this register are ysharp_bw, ydel_adj and Svideo. Their function was identified in the Table above.

38.4.3 Register 2

Register 2 holds values used to define the video line and field and to accommodate the format of the video input. These values must be set. However, in the ASIC environment, once they are calibrated for a particular format they never have to be changed.

H_HSO[10:0] defines the horizontal linestart in counts of CK27 following the external sync edge (or EAV code in the case of D1) as discussed above in relation to hsync_phs.

H_VSO[9:0] similarly defines the vertical fieldstart in units of lines following the designated edge of the external vertical sync (or SFB rising edge in D1).

H_HLC[9:0] is a pixel count (pixels per line minus 1). It is set to 857 for NTSC (interlaced and progressive) and to 863 for PAL.

38.4.4 Registers 3 and 4

In register 3, H_phase_inc[31:0] defines the frequency of the color subcarrier. It is 32'h 21f07c1f for NTSC and 32'h 2a098acb for PAL-B. It is relevant only to the composite output. The units are such that a phase increment of 360^{o} is entered as 32'h ffffffff.

in register 4, H_phase_offset[31:0]; this is added to the phase as otherwise generated. The unit are the same as for phase_inc. It can ordinarily be set to 0;

This page ɔration.

38.4.5 Register 5

Register 5 is used to enable either WSS or CGMS, and to specify the data to be transmitted. The register contents are as follows:

wss_enbl = enables WSS insertion for 625-line modes

cgms_enbl enables CGMS insertion for 525-line modes (both interlaced and progresive.)

wss_cgms_data[13:0] contains the data to be inserted.

The run-in and start code for WSS are generated by the logic and do not have to be entered. Similarly the leading 2'b10 and the final CRC check word for the CGMS are automatically generated by the logic.

38.4.6 Register 6

Register 6,is used to load the closed caption data. Insertion of this data is only supported for 525-line interlaced systems.) The fields have the following significance on a write

cc_fill[1:0] holds bits used to determine whether the two-bytes in cc_data are to be inserted in the odd (upper) field or the even (lower) field

cc_data[15:0] is the actual data to be inserted.

When this register is read, the cc_fill field holds flag bits indicating whether the corresponding data transfer has taken place.

The host would use this register as follows: To program line 21 of the upper field, poll register 6 until cc_fill[0] is zero. Then write the required closed caption data to cc_data[15:0] with cc_fill[0] set to 1. This write to register 6 causes the data to be copied into a register in the cc_block that holds the next upper-field pair of bytes. Reading register 6 at this point would echo the data just written in bits 15:0 and a "1" in bit 16. This bit would continue to be read as a 1 until the data is transferred to a shift register at the beginning of the line 21 on which it is to be outputted. Any time thereafter the data for the corresponding line in the upper field of the next frame can be entered.

The data for line 21 of the even field is similarly associated with c_fill[1]: it is set on a write to cause the data to be entered in a holding register; read as a one so long as that register should not be overwritten; and cleared to a 0 when the output on the corresponding line has started and the register is free for reprogramming.

38.4.7 Register group 8

The register 8 group must be written to define the rgbmatrix for the component output.

The matrix coefficients are 12-bit signed quantities defined on the ranges (-2,+2). Together they form an 108-bit bus, H_mx_coef_bus[107:0], organized as {rcoef_y, rcoef_cr, rcoef_cb, gcoef_y, gcoef_cr, gcoef_cb,bcoef_y, bcoef_cr, bcoef_cb}. Here rcoef_y, for example, is the coefficient of y in forming the red output.

The coefficient bus is entered as follows:

H_mx_coef_bus[107:0] = {reg_8.3[11:0],reg_8.2,reg_8.1,reg_8.0}.

This page ›ration.

The matrices used include the unit matrix for YPbPr

12'h0,12'h200,12'h0,
12'h200,12'h0,12'h0,
12'h0,12'h0,12'h200};

and the matrix used for RGB output

12'h260,12'h341,12'h0,
12'h260,12'he58,12'hf34,
12'h260,12'h0,12'h41d}.

The remaining fields in register 8.3 are used to define the color burst. They are

nba[7:0] = reg_8.3[27:20] = 8'h c8 (-56)
pba[7:0] = reg_8.3[19:12] = 8'h 00

for NTSC and

nba[7:0] = reg_8.3[27:20] = 8'h d6 (-42)
pba[7:0] = reg_8.3[19:12] = 8'h 2a (+42)

for PAL-B.

38.4.8 Macrovision Registers

Macrovision (register group 9) is on by default, configured for NTSC interlaced. This has no effect on the component outputs so long as the sync does not appear on them, i.e., so long as g_use_sync = 0 and rb_use_sync = 0.

38.4.9 Register group 10

The first group of parameters in the register 10 group define the features of a video line either in active video region or in the vertical sync regions. The features are defined on the basis of "pixel_cnt[9:0]". For interlaced formats, pixel_cnt[8:0] is the pixel count in the half line incremented at the 13.5 MHz rate, and pixel_cnt[9] is a flag set to 0 for the first half-line of a video line and set to 1 for the second half line. The pixel_cnt[8:0] is reset to zero after H_HLC cycles of 27 MHz. (That is the total length of the half-line is H_HLC + 1 cycles of 27 MHz.

For progressive NTSC, pixel_cnt[9:0] is incremented for full line which has H_HLC + 1 pixels at 27 MHz.

The location of the zero of pixel_cnt defines the start of the front-porch region of the video line. In all cases, the falling edge of sync is defined by sync_strt; the falling edge occurs immediately after pixel number "sync_strt." These falling edges are thus absolutely regular throughout the active video and vertical blanking regions. For interlaced formats, there is a sync pulse in each half-line of the pre-equalization, serration and post-equalization segmets of vertical blanking. Otherwise the sync pulse occurs only in the first half-line. (There is no second half line in the pregressive case.)

This page ɔration.

All other horizontal features are located in the same way: a pixel count after the beginning of the half-line (interlaced) or line (progressive.) Thus on active video lines, the last pixel of the sync pulse is given by "sync_end." The end for the serration pulse is "sync_srend" and that for the equalization pulse is "sync_eqend." The start and end of the color burst, and the start of active video are given in exactly the same manner. Normally all these quantities refer to the first half-line so their bit-9 is zero. For interlaced, the end of active video, specified by actv_end[9:0], is typically in the second half line. Thus actv_end[9] is "1" and does not represent an increment of 512 pixels but rather the number in a half line. (For NTSC the number of pixels per half-line is 429.)

Vertical formatting uses a half-line counter for interlaced formats, a line counter for progressive. Each region is programmed by specifying the last half-line of the preceding region. Field begins with the pre-equalization region, then comes serration followed by post-equalization, sub-phase and active video. The field size is defined by last_fld_ln. Ordinarily the pre-equalization region begins with the first line of the field so that last_fld_ln and vstrt_preeq are the same quantity.

In NTSC for example, pre-equalization begins with the first half line of the field and occupies 3 full lines. The comes serration. Thus vstrt_serra is 5 which is the last half line of the 6 occupied by the pre-equalization regions. Similarly, the serration also occupies 3 lines, so its last half-line is 11. It is followed by the post-equalization region, so that vstrt_posteq = 11, and so forth for the subsequent regions. In particular, there are 525 half lines in the field, so last_fld_ln = vstrt_preeq = 524.

The remaining parameters in register 10.4 form a rather miscellaneous group of controls

H_y_blank_ctrl[1:0] goes to YU where it controls the blanking level

00: 700:300 blanking for progressive

01: 714:286 blanking on both composite and component

10: 714:286 for blanking for composite and 700:300 blanking for component

11: 700:300 blanking for PAL systems

H_agc_lvl_ctrl[1:0] goes to MV for control of AGC levels

00: for "mixed NTSC" -i.e., 714:286 on composite and 700:300 on component

01: for NTSC

10: for PAL

11: for progressive

H_cs_invert_ctrl goes to MV to disable illegal modes of color-stripe inversion

H_bruchb[1:0] goes to SG to control mode of Bruch blanking

00: for progressive

01: for 525 line cases with NTSC color

10: for PAL-M and Pal-60

11: other PAL cases

H_fsc_phase_rst[1:0] goes to SG to control timing of color subcarrier phase reset

00: for progressive

This page ɔration.

01: for NTSC

10: for PAL-M, PAL-N and Pal-60

11: other PAL cases

H_pal_fsc_phase_alt goes to SG to enable PAL-manner of phase alternation by field of color subcarrier

H_ntsc_ln_cnt goes to SG to align even-odd field identification with internal field count; expected value is "1" for ntsc and "0" for PAL-B

H_lpf_rst_off[8:0] goes to ES to program the time to generate a pulse so as to preload a pipeline in the y_delay module of the L_unit. The value is found by simulation to be 272 for NTSC in D1 mode, 284 for PAL in D1 mode, and 136 for progressive in external sync mode.

If "default_picform" in register 0 is set to1, all the parameters programmable through the register-10 group are instead assigned hardwired values determined by T_ENCD_MODE. However, if even one of these parameters needs to be given a different value, the entire register-10 group must be written and default_picform set to 0. To make this process easier, the hardwired values for all registers and modes are given in Table 38-2.

Table 38-2. Hardwired Registers Values

mode	10.0	10.1	10.2	10.3	10.4
000 NTSC	17b1340e	3a42242d	07b14459	8320ca51	884a92c5
001 PAL-B	17c1340e	3a72642d	07714058	9c270c4e	0471f244
010 PAL-M	17b1340e	3a72642d	07f1505c	8320ca51	0443a2c5
011 PAL-N	17b1340e	3a725c2d	07e1505c	9c270c51	0471b2c5
100 PAL-CN	17c1340e	3a72642d	07a14058	9c270c4e	0471f244
101 mixed NTSC	17b1340e	3a42242d	07b14459	8320ca51	044252c5
110 PAL-60	17b1340e	3a72642d	07f1505c	8320ca51	0443a2c5
111 prog NTSC	32a1380f	359224xx (*)	xxxxxxxx (*)	8320bacx (*)	022202c5

(*) The parameters sync_eqend, wbrst_strt, nbrst_strt, brst_end, vstrt_subph are not used for progressive. The corresponding fields are "don't care."

This page ıration.

Chapter 39
Register Macro Usage

This chapter provides background on the i.MX23 register set and illustrates a consistent use of the C macros for registers. The examples provided here show how to use the hardware register macros generated from the chip database.

39.1 Background

The i.MX23 SOC is built on a 32-bit architecture using an ARM926 core. All hardware blocks are controlled and accessed through 32-bit wide registers. The design of these registers is maintained in a database that is part of the overall chip design. As part of the chip build process, a set of C include files are generated from the register descriptions. These include files provide a consistent set of C defines and macros that should be used to access the hardware registers.

The i.MX23 SOC has a complex architecture that uses multiple buses to segment I/O traffic and clock domains. To facilitate low power consumption, clocks are set to just meet application demands. In general, the I/O buses and associated hardware blocks run at speeds much slower than the CPU. As a result, reading a hardware register incurs a potentially large number of wait cycles, as the CPU must wait for the register data to travel multiple buses and bridges. The SOC does provide write buffering, meaning the CPU does not wait for register write transactions to complete. From the CPU perspective, register writes occur much faster than reads.

Most of the 32-bit registers are subdivided into smaller functional fields. These bit fields can be any number of bits wide and are usually packed. Thus, most fields do not align on byte or half-word boundaries.

A common operation is to update one field without disturbing the contents of the remaining fields in the register. Normally, this requires a read-modify-write (RMW) operation, where the CPU reads the register, modifies the target field, then writes the results back to the register. As already noted, this is an expensive operation in terms of CPU cycles, because of the initial register read.

To address this issue, most hardware registers are implemented as a set, including registers that can be used to either set, clear, or toggle (SCT) individual bits of the primary register. When writing to an SCT register, all bits set to 1 perform the associated operation on the primary register, while all bits set to 0 are not affected. The SCT registers always read back 0, and should be considered write-only. The SCT registers are not implemented if the primary register is read-only.

With this architecture, it is possible to update one or more fields using only register writes. First, all bits of the target fields are cleared by a write to the associated clear register, then the desired value of the tar-

get fields is written to the set register. This sequence of two writes is referred to as a clear-set (CS) operation.

A CS operation does have one potential drawback. Whenever a field is modified, the hardware sees a value of 0 before the final value is written. For most fields, passing through the 0 state is not a problem. Nonetheless, this behavior is something to consider when using a CS operation.

Also, a CS operation is not required for fields that are one bit wide. While the CS operation works in this case, it is more efficient to simply set or clear the target bit (i.e., one write instead of two). A simple set or clear operation is also atomic, while a CS operation is not.

Note that not all macros for set, clear, or toggle (SCT) are atomic. For registers that do not provide hardware support for this functionality, these macros are implemented as a sequence of read/modify/write operations. When atomic operation is required, the developer should pay attention to this detail, because unexpected behavior might result if an interrupt occurs in the middle of the critical section comprising the update sequence.

39.2 Naming Convention

The generated include files and macros follow a consistent naming convention that matches the SOC documentation. This prevents name-space collisions and makes the macros easier to remember.

```
//
// The include file for a specific hardware module is named:
//
//     regs<module>.h
//
// Every register has an associated typedef that provides a C definition of
// the register. The definition is always a union of a 32-bit unsigned int
// (i.e., reg32_t), and an anonymous bit field structure.
//
//     hw_<module>_<regname>_t
//
// Macros and defines that relate to a register as a whole are named:
//
//     HW_<module>_<regname>_ADDR
//     HW_<module>_<regname>_<SET | CLR | TOG>_ADDR
//         - defines for the indicated register address
//
//     HW_<module>_<regname>
//         - a define for accessing the primary register using the typedef.
//           Should be used as an rvalue (i.e., for reading), but avoided as
//           an lvalue (i.e., for writing). Will usually generate RMW when
//           used as an lvalue.
//
//     HW_<module>_<regname>_RD()
//     HW_<module>_<regname>_WR()
//         - macros for reading/writing the primary register as a whole
//
//     HW_<module>_<regname>_<SET | CLR | TOG>()
//         - macros for writing the associated set | clear | toggle registers
//
```

```
// Macros and defines that relate to the fields of a register are named:
//
//     BM_<module>_<regname>_<field>
//     BP_<module>_<regname>_<field>
//         - defines for the field's bit mask and bit position
//
//     BF_<module>_<regname>_<field>()
//     BF_<module>_<regname>_<field>_V(<valuename>)
//         - macros for generating a bit field value. The parameter is masked
//           and shifted to the field position.
//
//     BW_<module>_<regname>_<field>()
//         - macro for writing a bit field. Usually expands to a CS operation.
//           Not generated for read-only fields.
//
//     BV_<module>_<regname>_<field>__<valuename>
//         - define equates to an unshifted named value for the field
//
// Some hardware modules repeat the same register definition multiple times. An
// example is a block that implements multiple channels. For these registers,
// the name adds a lowercase 'n' after the module, and the HW_ macros take a
// numbered parameter to select the channel (or instance). This allows these
// macros to be used in for loops.
//
//     HW_<module>n_<regname><macrotype>(n,...)
//         - the n parameter must evaluate to an integer, and selects the channel
//           or instance number.
//
// The regs.h include file provides several "generic" macros that can be used
// as an alternate syntax for the various register operations. Because most
// operations involve using two or more of the above defines/macros, the <module>,
// <regname> and <field> are often repeated in a C expression. The generic
// macros provide shorthand to avoid the repetition. Refer to the following
// examples for the alternate syntax.
```

The C++ style comments above represent a single-instance block. For multiple-instance blocks, the macros are similar, but have an instance number as the first parameter where needed. (Differences only shown below.)

Note: x is the block instance number. If shown, v is the value field for the macro.

```
// HW_<module>_<regname>_ADDR(x)
// HW_<module>_<regname>_<SET | CLR | TOG>_ADDR(x)
// - defines for the indicated register address
//
// HW_<module>_<regname>
// - a define for accessing the primary register using the typedef.
// Should be used as an rvalue (i.e., for reading), but avoided as
// an lvalue (i.e., for writing). Will usually generate RMW when
// used as an lvalue.
//
// HW_<module>_<regname>_RD(x)
// HW_<module>_<regname>_WR(x)
// - macros for reading/writing the primary register as a whole
//
// HW_<module>_<regname>_<SET | CLR | TOG>(x)
// - macros for writing the associated set | clear | toggle registers
```

```
//
// Macros and defines that relate to the fields of a register are named:
//
// BW_<module>_<regname>_<field>(x, v)
// - macro for writing a bit field. Usually expands to a CS operation.
// Not generated for read-only fields.
```

39.2.1 Multi-Instance Blocks

Additionally, newer silicon architecture adds the concept of Multi-Instance Blocks which is similar to Multi-Instance Registers, although a Multi-Instance Block may also contain Multi-Instance Registers (There are none in the i.MX23). In order to accommodate that (and additional chips going forward) Multi-Instance Blocks have a required additional parameter specifying the block number but use otherwise identical nomenclature. This also allows runtime usage of different blocks (perhaps unifying driver models) without recompilation or near-duplication of code and run-time selection of macros.

The i.MX23 SOC starts all block instances from 1.

39.2.1.1 Examples

The SSP has two instances (numbered 1 and 2). To access the CTRL0 register in that block, instead of having two separate include files with hard coded macros, one can use the following:

```
HW_SSP_CTRL0_WR(instance, value);
```

where instance is 1 or 2, and value is (in this case) the 32 bit value to be written to the SSP_CTRL0 register in the block specified by instance.

39.3 Examples

The following examples show how to code common register operations using the predefined include files. Each example shows preferred and alternate syntax and also shows constructs to avoid. Summaries are provided toward the end.

The examples are valid C and will compile without errors. The reader is encouraged to compile this file and examine the resulting assembly code.

39.3.1 Setting 1-Bit Wide Field

```
// Preferred (one atomic write to SET register)
HW_GPMI_CTRL0_SET(BM_GPMI_CTRL0_UDMA);

// Alternate (same as above, just different syntax)
BF_SET(GPMI_CTRL0, UDMA);

// Avoid
BW_GPMI_CTRL0_UDMA(1);              // writes 1 to _CLR then 1 to _SET register
BF_WR(GPMI_CTRL0, UDMA, 1);         // same as above, just different syntax
HW_GPMI_CTRL0.B.UDMA = 1;           // RMW
```

39.3.2 Clearing 1-Bit Wide Field

```
// Preferred (one atomic write to _CLR register)
HW_GPMI_CTRL0_CLR(BM_GPMI_CTRL0_DEV_IRQ_EN);

// Alternate (same as above, just different syntax)
BF_CLR(GPMI_CTRL0, DEV_IRQ_EN);

// Avoid
BW_GPMI_CTRL0_DEV_IRQ_EN(0);        // writes 1 to _CLR then 0 to _SET register
BF_WR(GPMI_CTRL0, DEV_IRQ_EN, 0);     // same as above, just different syntax
HW_GPMI_CTRL0.B.DEV_IRQ_EN = 0;       // RMW
```

39.3.3 Toggling 1-Bit Wide Field

```
// Preferred (one atomic write to _TOG register)
HW_GPMI_CTRL0_TOG(BM_GPMI_CTRL0_RUN);

// Alternate (same as above, just different syntax)
BF_TOG(GPMI_CTRL0, RUN);

// Avoid
HW_GPMI_CTRL0.B.RUN ^= 1;             // RMW
```

39.3.4 Modifying n-Bit Wide Field

```
// Preferred (does CS operation or byte/halfword write if the field is
//   8 or 16 bits wide and properly aligned)
BW_GPMI_CTRL0_COMMAND_MODE(BV_GPMI_CTRL0_COMMAND_MODE__READ_AND_COMPARE);
BW_GPMI_CTRL0_COMMAND_MODE(iMode);
BW_GPMI_CTRL0_XFER_COUNT(2);        // this does a halfword write

// Alternate (same as above, just different syntax)
BF_WR(GPMI_CTRL0, COMMAND_MODE, BV_GPMI_CTRL0_COMMAND_MODE__READ_AND_COMPARE);
BF_WR(GPMI_CTRL0, COMMAND_MODE, iMode);
BF_WR(GPMI_CTRL0, XFER_COUNT, 2);   // this does a halfword write

// Avoid (RMW)
HW_GPMI_CTRL0.B.COMMAND_MODE = BV_GPMI_CTRL0_COMMAND_MODE__READ_AND_COMPARE;
HW_GPMI_CTRL0.B.COMMAND_MODE = iMode;
```

39.3.5 Modifying Multiple Fields

```
// Preferred (explicit CS operation)
HW_GPMI_CTRL0_CLR( OR3(BM_GPMI_CTRL0, RUN, DEV_IRQ_EN, COMMAND_MODE) );
HW_GPMI_CTRL0_SET( OR3(BF_GPMI_CTRL0, RUN(iRun), DEV_IRQ_EN(1),
                                              COMMAND_MODE_V(READ_AND_COMPARE)) );

// Alternate (same as above, just different syntax)
BF_CS3(GPMI_CTRL0, RUN, iRun, DEV_IRQ_EN, 1, COMMAND_MODE,
                                    BV_GPMI_CTRL0_COMMAND_MODE__READ_AND_COMPARE);

// Avoid (multiple RMW - the C compiler does NOT merge into one RMW)
HW_GPMI_CTRL0.B.RUN          = iRun;
HW_GPMI_CTRL0.B.DEV_IRQ_EN   = 1;
HW_GPMI_CTRL0.B.COMMAND_MODE = BV_GPMI_CTRL0_COMMAND_MODE__READ_AND_COMPARE;
```

39.3.6 Writing Entire Register (All Fields Updated at Once)

```
// Preferred
HW_GPMI_CTRL0_WR(BM_GPMI_CTRL0_SFTRST); // all other fields are set to 0

// Alternate (same as above, just different syntax)
HW_GPMI_CTRL0.U = BM_GPMI_CTRL0_SFTRST;
```

39.3.7 Reading a Bit Field

```
// Preferred
iRun = HW_GPMI_CTRL0.B.RUN;

// Alternate (same as above, just different syntax)
iRun = BF_RD(GPMI_CTRL0, RUN);

// Verbose Alternate (example of using bit position (BP_) define)
iRun = (HW_GPMI_CTRL0_RD() & BM_GPMI_CTRL0_RUN) >> BP_GPMI_CTRL0_RUN;
```

39.3.8 Reading Entire Register

```
0 // Preferred
i = HW_GPMI_CTRL0_RD();

// Alternate (same as above, just different syntax)
i = HW_GPMI_CTRL0.U;
```

39.3.9 Accessing Multiple Instance Register

```
// Preferred
for (i = 0; i < HW_TIMROT_TIMCTRLn_COUNT; i++)
{
    // Set 1-bit wide field
    HW_TIMROT_TIMCTRLn_SET(i, BM_TIMROT_TIMCTRLn_IRQ_EN);

    // Write n-bit wide field
    BW_TIMROT_TIMCTRLn_PRESCALE(i, BV_TIMROT_TIMCTRLn_PRESCALE__DIV_BY_1);

    // Write multiple fields
    HW_TIMROT_TIMCTRLn_CLR(i, OR2(BM_TIMROT_TIMCTRLn, RELOAD, SELECT));
HW_TIMROT_TIMCTRLn_CLR(i, OR2(BF_TIMROT_TIMCTRLn, RELOAD(1), SELECT_V(1KHZ_XTAL)));

    // Read a field
    iRun = HW_TIMROT_TIMCTRLn(i).B.IRQ;
}

// Alternate (same as above, just different syntax)
for (i = 0; i < HW_TIMROT_TIMCTRLn_COUNT; i++)
{
    // Set 1-bit wide field
    BF_SETn(TIMROT_TIMCTRLn, i, IRQ_EN);

    // Write n-bit wide field
    BF_WRn(TIMROT_TIMCTRLn, i, PRESCALE, BV_TIMROT_TIMCTRLn_PRESCALE__DIV_BY_1);

    // Write multiple fields
```

```
  BF_CS2n(TIMROT_TIMCTRLn, i, RELOAD, 1, SELECT, BV_TIMROT_TIMCTRLn_SELECT__1KHZ_XTAL);

    // Read a field
    iRun = BF_RDn(TIMROT_TIMCTRLn, i, IRQ);
}
```

39.3.10 Correct Way to Soft Reset a Block

```
// Prepare for soft-reset by making sure that SFTRST is not currently
// asserted. Also clear CLKGATE so we can wait for its assertion below.
HW_GPMI_CTRL0_CLR(BM_GPMI_CTRL0_SFTRST);

// Wait at least a microsecond for SFTRST to deassert. In actuality, we
// need to wait 3 GPMI clocks, but this is much easier to implement.
musecs = hw_profile_GetMicroseconds();
while (HW_GPMI_CTRL0.B.SFTRST || (hw_profile_GetMicroseconds() - musecs <
DDI_NAND_HAL_GPMI_SOFT_RESET_LATENCY));

// Also clear CLKGATE so we can wait for its assertion below.
HW_GPMI_CTRL0_CLR(BM_GPMI_CTRL0_CLKGATE);

// Now soft-reset the hardware.
HW_GPMI_CTRL0_SET(BM_GPMI_CTRL0_SFTRST);

// Poll until clock is in the gated state before subsequently
// clearing soft reset and clock gate.
while (!HW_GPMI_CTRL0.B.CLKGATE)
{
    ; // busy wait
}

// bring GPMI_CTRL0 out of reset
HW_GPMI_CTRL0_CLR(BM_GPMI_CTRL0_SFTRST);

// Wait at least a microsecond for SFTRST to deassert. In actuality, we
// need to wait 3 GPMI clocks, but this is much easier to implement.
musecs = hw_profile_GetMicroseconds();
while (HW_GPMI_CTRL0.B.SFTRST || (hw_profile_GetMicroseconds() - musecs <
DDI_NAND_HAL_GPMI_SOFT_RESET_LATENCY));

HW_GPMI_CTRL0_CLR(BM_GPMI_CTRL0_CLKGATE);

// Poll until clock is in the NON-gated state before returning.
while (HW_GPMI_CTRL0.B.CLKGATE)
{
    ; // busy wait
}
```

39.3.10.1 Pinmux Selection During Reset

For proper I^2C operation, the appropriate pinmux(s) must be selected before taking the block out of reset. Failure to select the I^2C pinmux selections before taking the block out of reset will cause the I^2C clock to operate incorrectly and will require another I^2C hardware reset.

39.3.10.1.1 Correct and Incorrect Reset Examples

Incorrect:

Clear I^2C SFTRST/CLKGATE
... Setup ...
I^2C PinMux Selections
*** I^2C will not operate.*

Correct:

I^2C PinMux Selections
Clear I^2C SFTRST/CLKGATE
... Setup ...
*** I^2C operates correctly.*

39.4 Summary Preferred

```
// Setting, clearing, toggling 1-bit wide field
HW_GPMI_CTRL0_SET(BM_GPMI_CTRL0_UDMA);
HW_GPMI_CTRL0_CLR(BM_GPMI_CTRL0_DEV_IRQ_EN);
HW_GPMI_CTRL0_TOG(BM_GPMI_CTRL0_RUN);

// Modifying n-bit wide field
BW_GPMI_CTRL0_XFER_COUNT(2);

// Modifying multiple fields
HW_GPMI_CTRL0_CLR( OR3(BM_GPMI_CTRL0, RUN, DEV_IRQ_EN, COMMAND_MODE) );
HW_GPMI_CTRL0_SET( OR3(BF_GPMI_CTRL0, RUN(iRun), DEV_IRQ_EN(1),
                                          COMMAND_MODE_V(READ_AND_COMPARE)) );

// Reading a bit field
iRun = HW_GPMI_CTRL0.B.RUN;

// Writing or reading entire register (all fields updated at once)
HW_GPMI_CTRL0_WR(BM_GPMI_CTRL0_SFTRST);
i = HW_GPMI_CTRL0_RD();
```

39.5 Summary Alternate Syntax

```
// Setting, clearing, toggling 1-bit wide field
BF_SET(GPMI_CTRL0, UDMA);
BF_CLR(GPMI_CTRL0, DEV_IRQ_EN);
BF_TOG(GPMI_CTRL0, RUN);

// Modifying n-bit wide field
BF_WR(GPMI_CTRL0, XFER_COUNT, 2);

// Modifying multiple fields
BF_CS3(GPMI_CTRL0, RUN, iRun, DEV_IRQ_EN, 1, COMMAND_MODE,
                                  BV_GPMI_CTRL0_COMMAND_MODE__READ_AND_COMPARE);

// Reading a bit field
iRun = BF_RD(GPMI_CTRL0, RUN);
```

```
// Writing or reading entire register (all fields updated at once)
HW_GPMI_CTRL0.U = BM_GPMI_CTRL0_SFTRST;
i = HW_GPMI_CTRL0.U;
```

39.6 Assembly Example

```
// The generated include files are safe to use with assembly code as well. Not
// all of the defines make sense in the assembly context, but many should prove
// useful.
//
//     HW_<module>_<regname>_ADDR
//     HW_<module>_<regname>_<SET | CLR | TOG>_ADDR
//         - defines for the indicated register address
//
//     BM_<module>_<regname>_<field>
//     BP_<module>_<regname>_<field>
//         - defines for the field's bit mask and bit position
//
//     BF_<module>_<regname>_<field>()
//     BF_<module>_<regname>_<field>_V(<valuename>)
//         - macros for generating a bit field value. The parameter is masked
//           and shifted to the field position.
//
//     BV_<module>_<regname>_<field>__<valuename>
//         - define equates to an unshifted named value for the field
//
// 6.1 Take GPMI block out of reset and remove clock gate.
// 6.2 Write a value to GPMI CTRL0 register. All other fields are set to 0.
#pragma asm
    ldr    r0, =HW_GPMI_CTRL0_CLR_ADDR
    ldr    r1, =BM_GPMI_CTRL0_SFTRST | BM_GPMI_CTRL0_CLKGATE
    str    r1, [r0]

    ldr    r0, =HW_GPMI_CTRL0_ADDR
    ldr    r1, =BF_GPMI_CTRL0_COMMAND_MODE_V(READ_AND_COMPARE)
    str    r1, [r0]
#pragma endasm
}

////////////////////////////////////////////////////////////////////////////////
//! \brief Standalone application main entry point.
//!
//! \fntype Function
//!
//! Provides main entry point when building as a standalone application.
//! Simply calls the example register access function.
////////////////////////////////////////////////////////////////////////////////
void main(void)
{
    hw_regs_Example();
}
////////////////////////////////////////////////////////////////////////////////
// End of file
////////////////////////////////////////////////////////////////////////////////
//! }@
```

Chapter 40
Memory Map

The following table shows the memory map in the i.MX23 as seen by the processor. Any blank entries indicate that nothing is mapped at that address. No accesses should be made to these addresses since the results are indeterminate. The Decode Block column indicates the decode group to which each peripheral belongs. Most peripherals reside on the APBH or APBX peripheral busses.

Table 40-1. Address Map for i.MX23

DECODE BLOCK	DEVICE	MNEMONIC	START ADDRESS	END ADDRESS	SIZE
AHB	On-chip RAM	OCRAM	0x00000000	0x00007FFF	32KB
	On-chip RAM alias	OCRAM	0x00008000	0x3FFFFFFF	
	External Memory		0x40000000	0x5FFFFFFF	512MB
	Default Slave		0x60000000	0x7FFFFFFF	512MB

Table 40-1. Address Map for i.MX23 (continued)

DECODE BLOCK	DEVICE	MNEMONIC	START ADDRESS	END ADDRESS	SIZE
APBH	Interrupt Controller	ICOLL	0x80000000	0x80001FFF	8KB
			0x80002000	0x80003FFF	8KB
	APBH DMA	APBHDMA	0x80004000	0x80005FFF	8KB
			0x80006000	0x80007FFF	8KB
	Reed-Solomon ECC	ECC8	0x80008000	0x80009FFF	8KB
	BCH ECC	BCH	0x8000A000	0x8000BFFF	8KB
	General Purpose Media Interface	GPMI	0x8000C000	0x8000DFFF	8KB
			0x8000E000	0x8000FFFF	8KB
	Sync Serial Port 1	SSP1	0x80010000	0x80011FFF	8KB
			0x80012000	0x80013FFF	8KB
	Embedded Trace Module	ETM	0x80014000	0x80015FFF	8KB
			0x80016000	0x80017FFF	8KB
	Pin Control	PINCTRL	0x80018000	0x80019FFF	8KB
			0x8001A000	0x8001BFFF	8KB
	Digital Control	DIGCTL	0x8001C000	0x8001DFFF	8KB
			0x8001E000	0x8001FFFF	8KB
	External Memory Interface	EMI	0x80020000	0x80021FFF	8KB
			0x80022000	0x80023FFF	8KB
	APBX DMA	APBXDMA	0x80024000	0x80025FFF	8KB
			0x80026000	0x80027FFF	8KB
	Data CoProcessor	DCP	0x80028000	0x80029FFF	8KB
	Pixel Pipeline	PXP	0x8002A000	0x8002BFFF	8KB
	One-time Programmable Array Controller	OCOTP	0x8002C000	0x8002DFFF	8KB
	AXI Control	AXI	0x8002E000	0x8002FFFF	8KB
	LCD Interface	LCDIF	0x80030000	0x80031FFF	8KB
			0x80032000	0x80033FFF	8KB
	Sync Serial Port 2	SSP2	0x80034000	0x80035FFF	8KB
			0x80036000	0x80037FFF	8KB
	TV Encoder	TVENC	0x80038000	0x80039FFF	8KB
			0x8003A000	0x8003BFFF	8KB
	Reserved		0x8003C000	0x8003DFFF	8KB
			0X8003E000	0X8003FFFF	8KB

Table 40-1. Address Map for i.MX23 (continued)

DECODE BLOCK	DEVICE	MNEMONIC	START ADDRESS	END ADDRESS	SIZE
APBX	Clock Controller	CLKCTRL	0x80040000	0x80041FFF	8KB
	Sync Audio Interface	SAIF1	0x80042000	0x80043FFF	8KB
	Power Control	PWR	0x80044000	0x80045FFF	8KB
	Sync Audio Interface	SAIF2	0x80046000	0x80047FFF	8KB
	Digital Audio Filter Output	AUDIOOUT	0x80048000	0x80049FFF	8KB
			0x8004A000	0x8004BFFF	8KB
	Digital Audio Filter Input	AUDIOIN	0x8004C000	0x8004DFFF	8KB
			0x8004E000	0x8004FFFF	8KB
	Low Resolution ADC	LRADC	0x80050000	0x80051FFF	8KB
			0x80052000	0x80053FFF	8KB
	Sony/Phillips Digital Audio Interface	SPDIF	0x80054000	0x80055FFF	8KB
			0x80056000	0x80057FFF	8KB
	I^2C	I2C	0x80058000	0x80059FFF	8KB
			0x8005A000	0x8005BFFF	8KB
	Real Time Clock	RTC	0x8005C000	0x8005DFFF	8KB
			0x8005E000	0x8005FFFF	8KB
			0x80060000	0x80061FFF	8KB
			0x80062000	0x80063FFF	8KB
	Pulse Width Modulation	PWM	0x80064000	0x80065FFF	8KB
			0x80066000	0x80067FFF	8KB
	Timers/Rotary Interface	TIMROT	0x80068000	0x80069FFF	8KB
			0x8006A000	0x8006BFFF	8KB
	Application UART 1	APPUART1	0x8006C000	0x8006DFFF	8KB
	Application UART 2	APPUART2	0x8006E000	0x8006FFFF	8KB
	Debug UART	DBGUART	0x80070000	0x80071FFF	8KB
			0x80072000	0x80073FFF	8KB
			0x80074000	0x80075FFF	8KB
			0x80076000	0x80077FFF	8KB
			0x80078000	0x80079FFF	8KB
			0x8007A000	0x8007BFFF	8KB
	USB Physical Interface	USBPHY	0x8007C000	0x8007DFFF	8KB
			0x8007E000	0x8007FFFF	8KB
AHB	USB Controller	USB	0x80080000	0x800BFFFF	256KB
			0x800C0000	0x800CFFFF	64KB
			0x800D0000	0x800DFFFF	64KB
	DRAM Registers	DRAM	0x800E0000	0x800EFFFF	64KB

Table 40-1. Address Map for i.MX23 (continued)

DECODE BLOCK	DEVICE	MNEMONIC	START ADDRESS	END ADDRESS	SIZE
	DRAM Registers	DRAM	0x800F0000	0x800FFFFF	64KB
			0x80100000	0xBFFFFFFF	
	ROM	OCROM	0xC0000000	0xC000FFFF	64KB
	ROM alias	OCROM	0xC0010000	0xFFFFFFFF	

Chapter 41
i.MX23 Part Numbers and Ordering Information

The i.MX23 family comprises a set of parts targeted at specific applications and customers. Table 41-1 summarizes the family members and provides part numbers for order placement.

Table 41-1. Part Numbers for i.MX23 Family Members

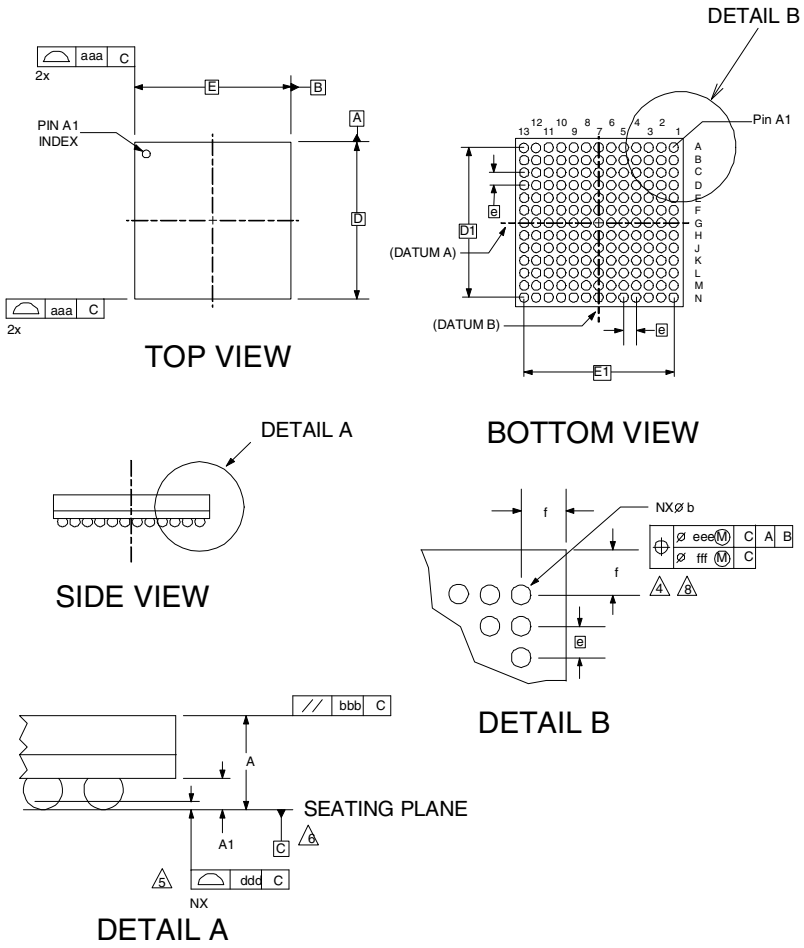
Description	Part Number	Package	Speed	Ambient Temperature Range
i.MX233, Industrial, 169BGA	MCIMX233CJM4B	169-pin BGA	454 MHz	-40 to 85
i.MX233, Industrial, 128QFP	MCIMX233CAG4B	128-pin LQFP	454 MHz	-40 to 85
i.MX233, Commercial, 169BGA	MCIMX233DJM4B	169-pin BGA	454 MHz	-10 to 70
i.MX233, Commercial, 128QFP	MCIMX233DAG4B	128-pin LQFP	454 MHz	-10 to 70

Chapter 42
Package Drawings

The i.MX23 is offered in two different packages, which are illustrated in this chapter:

- Section 42.1, “169-Pin Ball Grid Array (BGA)”
- Section 42.2, “128-Pin Low-Profile Quad Flat Package (LQFP)”

42.1 169-Pin Ball Grid Array (BGA)



169 fpBGA (11 x 11 mm)

DIMENSIONAL REFERENCES			
REF.	MIN.	NOM.	MAX.
A	1.14	1.30	1.43
A1	0.21	0.28	0.35
D	10.80	11.00	11.20
D1		9.60 BSC	
E	10.80	11.00	11.20
E1		9.60 BSC	
b	0.37	0.43	0.49
e		0.80 BSC	
f	0.60	0.70	0.80
aaa			0.10
bbb			0.10
ddd			0.15
eee			0.15
fff			0.08
M		13	
N		169	

ALL DIMENSIONS ARE IN MILLIMETERS.
'e' REPRESENTS THE BASIC SOLDER BALL GRID PITCH.
'M' REPRESENTS THE BASIC SOLDER BALL MATRIX SIZE. SYMBOL 'N'
IS THE NUMBER OF BALLS IN THE BALL MATRIX.
'b' IS MEASURABLE AT THE MAXIMUM SOLDER BALL DIAMETER PARALLEL
TO PRIMARY DATUM C.
DIMENSION 'ddd' IS MEASURED PARALLEL TO PRIMARY DATUM C.
PRIMARY DATUM C AND SEATING PLANE ARE DEFINED BY THE SPHERICAL
CROWNS OF THE SOLDER BALLS.
SOLDER BALL DIAMETER 'b' REFERS TO POST REFLOW CONDITION. THE
PRE-REFLOW DIAMETER IS 0.40mm.
SUBSTRATE MATERIAL BASE IS BT RESIN.
THE OVERALL PACKAGE THICKNESS 'A' ALREADY CONSIDERS COLLAPSE BALLS.
DIMENSIONING AND TOLERENCING PER ASME Y 14.5-1994.
PACKAGE DIMENSIONS TAKE REFERENCE TO JEDEC MO-205 F.

Figure 42-1. 169-Pin BGA Package Drawing

42.2 128-Pin Low-Profile Quad Flat Package (LQFP)

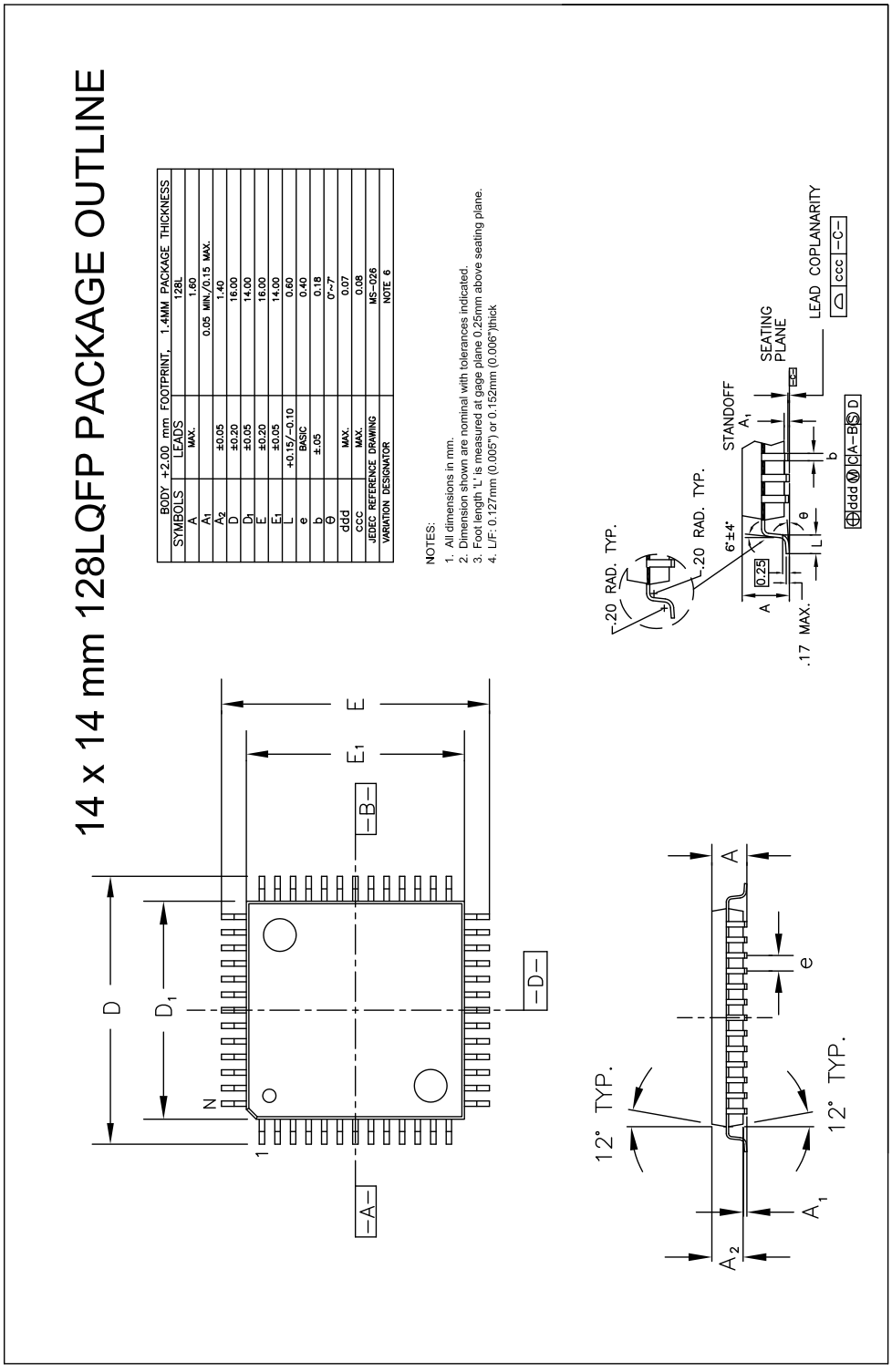


Figure 42-2. 128-Pin Low-Profile Quad Flat Pack (LQFP) Package Drawing

Appendix A
Revision History

This appendix provides a list of the major differences between the *i.MX23 Applications Processor Reference Manual Reference Manual*, Revision 0 through Revision 1.

A.1 Changes From Revision 0 to Revision 1

Major changes to the *i.MX23 Applications Processor Reference Manual Reference Manual*, from Revision 0 to Revision 1, are as follows:

Section, Page	Changes
Book	Added Sarnoff DVE Chapter. Added revision history as Appendix A.
Chapter 2/2-1	Updated 169BGA Operating States Table
	Added 128QFP Operating States Table
	Added 128QFP EMICLK data.
	Updated Power Supply Characteristics table
	Updated VDD5V Absolute Maximum specification.
	Updated 169-BGA Package Thermal Impedance value.
	Added DCDC Efficiency curves.
Chapter 12/12-1	Added Denali Databahn User Manual content:

Appendix B
Register Names

Table B-1 lists the i.MX23 register names and addresses in alphabetical order by register mnemonic.

Table B-1. Register Names and Addresses

Register Name	Address
DFLPT_PTE_2048	0x800C2000
HW_APBH_CH0_BAR	0x80004070
HW_APBH_CH0_CMD	0x80004060
HW_APBH_CH0_CURCMDAR	0x80004040
HW_APBH_CH0_DEBUG1	0x80004090
HW_APBH_CH0_DEBUG2	0x800040A0
HW_APBH_CH0_NXTCMDAR	0x80004050
HW_APBH_CH0_SEMA	0x80004080
HW_APBH_CH1_BAR	0x800040E0
HW_APBH_CH1_CMD	0x800040D0
HW_APBH_CH1_CURCMDAR	0x800040B0
HW_APBH_CH1_DEBUG1	0x80004100
HW_APBH_CH1_DEBUG2	0x80004110
HW_APBH_CH1_NXTCMDAR	0x800040C0
HW_APBH_CH1_SEMA	0x800040F0
HW_APBH_CH2_BAR	0x80004150
HW_APBH_CH2_CMD	0x80004140
HW_APBH_CH2_CURCMDAR	0x80004120
HW_APBH_CH2_DEBUG1	0x80004170
HW_APBH_CH2_DEBUG2	0x80004180
HW_APBH_CH2_NXTCMDAR	0x80004130
HW_APBH_CH2_SEMA	0x80004160
HW_APBH_CH3_BAR	0x800041C0
HW_APBH_CH3_CMD	0x800041B0
HW_APBH_CH3_CURCMDAR	0x80004190
HW_APBH_CH3_DEBUG1	0x800041E0

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_APBH_CH3_DEBUG2	0x800041F0
HW_APBH_CH3_NXTCMDAR	0x800041A0
HW_APBH_CH3_SEMA	0x800041D0
HW_APBH_CH4_BAR	0x80004230
HW_APBH_CH4_CMD	0x80004220
HW_APBH_CH4_CURCMDAR	0x80004200
HW_APBH_CH4_DEBUG1	0x80004250
HW_APBH_CH4_DEBUG2	0x80004260
HW_APBH_CH4_NXTCMDAR	0x80004210
HW_APBH_CH4_SEMA	0x80004240
HW_APBH_CH5_BAR	0x800042A0
HW_APBH_CH5_CMD	0x80004290
HW_APBH_CH5_CURCMDAR	0x80004270
HW_APBH_CH5_DEBUG1	0x800042C0
HW_APBH_CH5_DEBUG2	0x800042D0
HW_APBH_CH5_NXTCMDAR	0x80004280
HW_APBH_CH5_SEMA	0x800042B0
HW_APBH_CH6_BAR	0x80004310
HW_APBH_CH6_CMD	0x80004300
HW_APBH_CH6_CURCMDAR	0x800042E0
HW_APBH_CH6_DEBUG1	0x80004330
HW_APBH_CH6_DEBUG2	0x80004340
HW_APBH_CH6_NXTCMDAR	0x800042F0
HW_APBH_CH6_SEMA	0x80004320
HW_APBH_CH7_BAR	0x80004380
HW_APBH_CH7_CMD	0x80004370
HW_APBH_CH7_CURCMDAR	0x80004350
HW_APBH_CH7_DEBUG1	0x800043A0
HW_APBH_CH7_DEBUG2	0x800043B0
HW_APBH_CH7_NXTCMDAR	0x80004360
HW_APBH_CH7_SEMA	0x80004390
HW_APBH_CTRL0	0x80004000
HW_APBH_CTRL0_CLR	0x80004008

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_APBH_CTRL0_SET	0x80004004
HW_APBH_CTRL0_TOG	0x8000400C
HW_APBH_CTRL1	0x80004010
HW_APBH_CTRL1_CLR	0x80004018
HW_APBH_CTRL1_SET	0x80004014
HW_APBH_CTRL1_TOG	0x8000401C
HW_APBH_CTRL2	0x80004020
HW_APBH_CTRL2_CLR	0x80004028
HW_APBH_CTRL2_SET	0x80004024
HW_APBH_CTRL2_TOG	0x8000402C
HW_APBH_DEVSEL	0x80004030
HW_APBH_VERSION	0x800043F0
HW_APBX_CH0_BAR	0x80024130
HW_APBX_CH0_CMD	0x80024120
HW_APBX_CH0_CURCMDAR	0x80024100
HW_APBX_CH0_DEBUG1	0x80024150
HW_APBX_CH0_DEBUG2	0x80024160
HW_APBX_CH0_NXTCMDAR	0x80024110
HW_APBX_CH0_SEMA	0x80024140
HW_APBX_CH1_BAR	0x800241A0
HW_APBX_CH1_CMD	0x80024190
HW_APBX_CH1_CURCMDAR	0x80024170
HW_APBX_CH1_DEBUG1	0x800241C0
HW_APBX_CH1_DEBUG2	0x800241D0
HW_APBX_CH1_NXTCMDAR	0x80024180
HW_APBX_CH1_SEMA	0x800241B0
HW_APBX_CH10_BAR	0x80024590
HW_APBX_CH10_CMD	0x80024580
HW_APBX_CH10_CURCMDAR	0x80024560
HW_APBX_CH10_DEBUG1	0x800245B0
HW_APBX_CH10_DEBUG2	0x800245C0
HW_APBX_CH10_NXTCMDAR	0x80024570
HW_APBX_CH10_SEMA	0x800245A0

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_APBX_CH11_BAR	0x80024600
HW_APBX_CH11_CMD	0x800245F0
HW_APBX_CH11_CURCMDAR	0x800245D0
HW_APBX_CH11_DEBUG1	0x80024620
HW_APBX_CH11_DEBUG2	0x80024630
HW_APBX_CH11_NXTCMDAR	0x800245E0
HW_APBX_CH11_SEMA	0x80024610
HW_APBX_CH12_BAR	0x80024670
HW_APBX_CH12_CMD	0x80024660
HW_APBX_CH12_CURCMDAR	0x80024640
HW_APBX_CH12_DEBUG1	0x80024690
HW_APBX_CH12_DEBUG2	0x800246A0
HW_APBX_CH12_NXTCMDAR	0x80024650
HW_APBX_CH12_SEMA	0x80024680
HW_APBX_CH13_BAR	0x800246E0
HW_APBX_CH13_CMD	0x800246D0
HW_APBX_CH13_CURCMDAR	0x800246B0
HW_APBX_CH13_DEBUG1	0x80024700
HW_APBX_CH13_DEBUG2	0x80024710
HW_APBX_CH13_NXTCMDAR	0x800246C0
HW_APBX_CH13_SEMA	0x800246F0
HW_APBX_CH14_BAR	0x80024750
HW_APBX_CH14_CMD	0x80024740
HW_APBX_CH14_CURCMDAR	0x80024720
HW_APBX_CH14_DEBUG1	0x80024770
HW_APBX_CH14_DEBUG2	0x80024780
HW_APBX_CH14_NXTCMDAR	0x80024730
HW_APBX_CH14_SEMA	0x80024760
HW_APBX_CH15_BAR	0x800247C0
HW_APBX_CH15_CMD	0x800247B0
HW_APBX_CH15_CURCMDAR	0x80024790
HW_APBX_CH15_DEBUG1	0x800247E0
HW_APBX_CH15_DEBUG2	0x800247F0

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_APBX_CH15_NXTCMDAR	0x800247A0
HW_APBX_CH15_SEMA	0x800247D0
HW_APBX_CH2_BAR	0x80024210
HW_APBX_CH2_CMD	0x80024200
HW_APBX_CH2_CURCMDAR	0x800241E0
HW_APBX_CH2_DEBUG1	0x80024230
HW_APBX_CH2_DEBUG2	0x80024240
HW_APBX_CH2_NXTCMDAR	0x800241F0
HW_APBX_CH2_SEMA	0x80024220
HW_APBX_CH3_BAR	0x80024280
HW_APBX_CH3_CMD	0x80024270
HW_APBX_CH3_CURCMDAR	0x80024250
HW_APBX_CH3_DEBUG1	0x800242A0
HW_APBX_CH3_DEBUG2	0x800242B0
HW_APBX_CH3_NXTCMDAR	0x80024260
HW_APBX_CH3_SEMA	0x80024290
HW_APBX_CH4_BAR	0x800242F0
HW_APBX_CH4_CMD	0x800242E0
HW_APBX_CH4_CURCMDAR	0x800242C0
HW_APBX_CH4_DEBUG1	0x80024310
HW_APBX_CH4_DEBUG2	0x80024320
HW_APBX_CH4_NXTCMDAR	0x800242D0
HW_APBX_CH4_SEMA	0x80024300
HW_APBX_CH5_BAR	0x80024360
HW_APBX_CH5_CMD	0x80024350
HW_APBX_CH5_CURCMDAR	0x80024330
HW_APBX_CH5_DEBUG1	0x80024380
HW_APBX_CH5_DEBUG2	0x80024390
HW_APBX_CH5_NXTCMDAR	0x80024340
HW_APBX_CH5_SEMA	0x80024370
HW_APBX_CH6_BAR	0x800243D0
HW_APBX_CH6_CMD	0x800243C0
HW_APBX_CH6_CURCMDAR	0x800243A0

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_APBX_CH6_DEBUG1	0x800243F0
HW_APBX_CH6_DEBUG2	0x80024400
HW_APBX_CH6_NXTCMDAR	0x800243B0
HW_APBX_CH6_SEMA	0x800243E0
HW_APBX_CH7_BAR	0x80024440
HW_APBX_CH7_CMD	0x80024430
HW_APBX_CH7_CURCMDAR	0x80024410
HW_APBX_CH7_DEBUG1	0x80024460
HW_APBX_CH7_DEBUG2	0x80024470
HW_APBX_CH7_NXTCMDAR	0x80024420
HW_APBX_CH7_SEMA	0x80024450
HW_APBX_CH8_BAR	0x800244B0
HW_APBX_CH8_CMD	0x800244A0
HW_APBX_CH8_CURCMDAR	0x80024480
HW_APBX_CH8_DEBUG1	0x800244D0
HW_APBX_CH8_DEBUG2	0x800244E0
HW_APBX_CH8_NXTCMDAR	0x80024490
HW_APBX_CH8_SEMA	0x800244C0
HW_APBX_CH9_BAR	0x80024520
HW_APBX_CH9_CMD	0x80024510
HW_APBX_CH9_CURCMDAR	0x800244F0
HW_APBX_CH9_DEBUG1	0x80024540
HW_APBX_CH9_DEBUG2	0x80024550
HW_APBX_CH9_NXTCMDAR	0x80024500
HW_APBX_CH9_SEMA	0x80024530
HW_APBX_CHANNEL_CTRL	0x80024030
HW_APBX_CHANNEL_CTRL_CLR	0x80024038
HW_APBX_CHANNEL_CTRL_SET	0x80024034
HW_APBX_CHANNEL_CTRL_TOG	0x8002403C
HW_APBX_CTRL0	0x80024000
HW_APBX_CTRL0_CLR	0x80024008
HW_APBX_CTRL0_SET	0x80024004
HW_APBX_CTRL0_TOG	0x8002400C

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_APBX_CTRL1	0x80024010
HW_APBX_CTRL1_CLR	0x80024018
HW_APBX_CTRL1_SET	0x80024014
HW_APBX_CTRL1_TOG	0x8002401C
HW_APBX_CTRL2	0x80024020
HW_APBX_CTRL2_CLR	0x80024028
HW_APBX_CTRL2_SET	0x80024024
HW_APBX_CTRL2_TOG	0x8002402C
HW_APBX_DEVSEL	0x80024040
HW_APBX_VERSION	0x80024800
HW_AUDIOIN_ADCDEBUG	0x8004C040
HW_AUDIOIN_ADCDEBUG_CLR	0x8004C048
HW_AUDIOIN_ADCDEBUG_SET	0x8004C044
HW_AUDIOIN_ADCDEBUG_TOG	0x8004C04C
HW_AUDIOIN_ADCSRR	0x8004C020
HW_AUDIOIN_ADCSRR_CLR	0x8004C028
HW_AUDIOIN_ADCSRR_SET	0x8004C024
HW_AUDIOIN_ADCSRR_TOG	0x8004C02C
HW_AUDIOIN_ADCVOL	0x8004C050
HW_AUDIOIN_ADCVOL_CLR	0x8004C058
HW_AUDIOIN_ADCVOL_SET	0x8004C054
HW_AUDIOIN_ADCVOL_TOG	0x8004C05C
HW_AUDIOIN_ADCVOLUME	0x8004C030
HW_AUDIOIN_ADCVOLUME_CLR	0x8004C038
HW_AUDIOIN_ADCVOLUME_SET	0x8004C034
HW_AUDIOIN_ADCVOLUME_TOG	0x8004C03C
HW_AUDIOIN_ANACLKCTRL	0x8004C070
HW_AUDIOIN_ANACLKCTRL_CLR	0x8004C078
HW_AUDIOIN_ANACLKCTRL_SET	0x8004C074
HW_AUDIOIN_ANACLKCTRL_TOG	0x8004C07C
HW_AUDIOIN_CTRL	0x8004C000
HW_AUDIOIN_CTRL_CLR	0x8004C008
HW_AUDIOIN_CTRL_SET	0x8004C004

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_AUDIOIN_CTRL_TOG	0x8004C00C
HW_AUDIOIN_DATA	0x8004C080
HW_AUDIOIN_DATA_CLR	0x8004C088
HW_AUDIOIN_DATA_SET	0x8004C084
HW_AUDIOIN_DATA_TOG	0x8004C08C
HW_AUDIOIN_MICLINE	0x8004C060
HW_AUDIOIN_MICLINE_CLR	0x8004C068
HW_AUDIOIN_MICLINE_SET	0x8004C064
HW_AUDIOIN_MICLINE_TOG	0x8004C06C
HW_AUDIOIN_STAT	0x8004C010
HW_AUDIOIN_STAT_CLR	0x8004C018
HW_AUDIOIN_STAT_SET	0x8004C014
HW_AUDIOIN_STAT_TOG	0x8004C01C
HW_AUDIOOUT_ANACLKCTRL	0x800480e0
HW_AUDIOOUT_ANACLKCTRL_CLR	0x800480e8
HW_AUDIOOUT_ANACLKCTRL_SET	0x800480e4
HW_AUDIOOUT_ANACLKCTRL_TOG	0x800480eC
HW_AUDIOOUT_ANACTRL	0x80048090
HW_AUDIOOUT_ANACTRL_CLR	0x80048098
HW_AUDIOOUT_ANACTRL_SET	0x80048094
HW_AUDIOOUT_ANACTRL_TOG	0x8004809C
HW_AUDIOOUT_BISTCTRL	0x800480b0
HW_AUDIOOUT_BISTCTRL_CLR	0x800480b8
HW_AUDIOOUT_BISTCTRL_SET	0x800480b4
HW_AUDIOOUT_BISTCTRL_TOG	0x800480bC
HW_AUDIOOUT_BISTSTAT0	0x800480c0
HW_AUDIOOUT_BISTSTAT0_CLR	0x800480c8
HW_AUDIOOUT_BISTSTAT0_SET	0x800480c4
HW_AUDIOOUT_BISTSTAT0_TOG	0x800480cC
HW_AUDIOOUT_BISTSTAT1	0x800480d0
HW_AUDIOOUT_BISTSTAT1_CLR	0x800480d8
HW_AUDIOOUT_BISTSTAT1_SET	0x800480d4
HW_AUDIOOUT_BISTSTAT1_TOG	0x800480dC

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_AUDIOOUT_CTRL	0x80048000
HW_AUDIOOUT_CTRL_CLR	0x80048008
HW_AUDIOOUT_CTRL_SET	0x80048004
HW_AUDIOOUT_CTRL_TOG	0x8004800C
HW_AUDIOOUT_DACDEBUG	0x80048040
HW_AUDIOOUT_DACDEBUG_CLR	0x80048048
HW_AUDIOOUT_DACDEBUG_SET	0x80048044
HW_AUDIOOUT_DACDEBUG_TOG	0x8004804C
HW_AUDIOOUT_DACSRR	0x80048020
HW_AUDIOOUT_DACSRR_CLR	0x80048028
HW_AUDIOOUT_DACSRR_SET	0x80048024
HW_AUDIOOUT_DACSRR_TOG	0x8004802C
HW_AUDIOOUT_DACVOLUME	0x80048030
HW_AUDIOOUT_DACVOLUME_CLR	0x80048038
HW_AUDIOOUT_DACVOLUME_SET	0x80048034
HW_AUDIOOUT_DACVOLUME_TOG	0x8004803C
HW_AUDIOOUT_DATA	0x800480f0
HW_AUDIOOUT_DATA_CLR	0x800480f8
HW_AUDIOOUT_DATA_SET	0x800480f4
HW_AUDIOOUT_DATA_TOG	0x800480fC
HW_AUDIOOUT_HPVOL	0x80048050
HW_AUDIOOUT_HPVOL_CLR	0x80048058
HW_AUDIOOUT_HPVOL_SET	0x80048054
HW_AUDIOOUT_HPVOL_TOG	0x8004805C
HW_AUDIOOUT_PWRDN	0x80048070
HW_AUDIOOUT_PWRDN_CLR	0x80048078
HW_AUDIOOUT_PWRDN_SET	0x80048074
HW_AUDIOOUT_PWRDN_TOG	0x8004807C
HW_AUDIOOUT_REFCTRL	0x80048080
HW_AUDIOOUT_REFCTRL_CLR	0x80048088
HW_AUDIOOUT_REFCTRL_SET	0x80048084
HW_AUDIOOUT_REFCTRL_TOG	0x8004808C
HW_AUDIOOUT_RESERVED	0x80048060

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_AUDIOOUT_RESERVED_CLR	0x80048068
HW_AUDIOOUT_RESERVED_SET	0x80048064
HW_AUDIOOUT_RESERVED_TOG	0x8004806C
HW_AUDIOOUT_SPEAKERCTRL	0x80048100
HW_AUDIOOUT_SPEAKERCTRL_CLR	0x80048108
HW_AUDIOOUT_SPEAKERCTRL_SET	0x80048104
HW_AUDIOOUT_SPEAKERCTRL_TOG	0x8004810C
HW_AUDIOOUT_STAT	0x80048010
HW_AUDIOOUT_STAT_CLR	0x80048018
HW_AUDIOOUT_STAT_SET	0x80048014
HW_AUDIOOUT_STAT_TOG	0x8004801C
HW_AUDIOOUT_TEST	0x800480a0
HW_AUDIOOUT_TEST_CLR	0x800480a8
HW_AUDIOOUT_TEST_SET	0x800480a4
HW_AUDIOOUT_TEST_TOG	0x800480aC
HW_AUDIOOUT_VERSION	0x80048200
HW_BCH_BLOCKNAME	0x8000A150
HW_BCH_CTRL	0x8000A000
HW_BCH_CTRL_CLR	0x8000A008
HW_BCH_CTRL_SET	0x8000A004
HW_BCH_CTRL_TOG	0x8000A00C
HW_BCH_DATAPTR	0x8000A040
HW_BCH_DBGAHBMREAD	0x8000A140
HW_BCH_DBGCSFEREAD	0x8000A120
HW_BCH_DBGKESREAD	0x8000A110
HW_BCH_DBGSYNDGENREAD	0x8000A130
HW_BCH_DEBUG0	0x8000A100
HW_BCH_DEBUG0_CLR	0x8000A108
HW_BCH_DEBUG0_SET	0x8000A104
HW_BCH_DEBUG0_TOG	0x8000A10C
HW_BCH_ENCODEPTR	0x8000A030
HW_BCH_FLASH0LAYOUT0	0x8000a080
HW_BCH_FLASH0LAYOUT1	0x8000a090

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_BCH_FLASH1LAYOUT0	0x8000a0a0
HW_BCH_FLASH1LAYOUT1	0x8000a0b0
HW_BCH_FLASH2LAYOUT0	0x8000a0c0
HW_BCH_FLASH2LAYOUT1	0x8000a0d0
HW_BCH_FLASH3LAYOUT0	0x8000a0e0
HW_BCH_FLASH3LAYOUT1	0x8000a0f0
HW_BCH_LAYOUTSELECT	0x8000A070
HW_BCH_METAPTR	0x8000A050
HW_BCH_MODE	0x8000A020
HW_BCH_STATUS0	0x8000A010
HW_BCH_VERSION	0x8000A160
HW_CLKCTRL_CLKSEQ	0x80040110
HW_CLKCTRL_CLKSEQ_CLR	0x80040118
HW_CLKCTRL_CLKSEQ_SET	0x80040114
HW_CLKCTRL_CLKSEQ_TOG	0x8004011c
HW_CLKCTRL_CPU	0x80040020
HW_CLKCTRL_CPU_CLR	0x80040028
HW_CLKCTRL_CPU_SET	0x80040024
HW_CLKCTRL_CPU_TOG	0x8004002c
HW_CLKCTRL_EMI	0x800400a0
HW_CLKCTRL_ETM	0x800400e0
HW_CLKCTRL_FRAC	0x800400f0
HW_CLKCTRL_FRAC_CLR	0x800400f8
HW_CLKCTRL_FRAC_SET	0x800400f4
HW_CLKCTRL_FRAC_TOG	0x800400fC
HW_CLKCTRL_FRAC1	0x80040100
HW_CLKCTRL_FRAC1_CLR	0x80040108
HW_CLKCTRL_FRAC1_SET	0x80040104
HW_CLKCTRL_FRAC1_TOG	0x8004010C
HW_CLKCTRL_GPMI	0x80040080
HW_CLKCTRL_HBUS	0x80040030
HW_CLKCTRL_HBUS_CLR	0x80040038
HW_CLKCTRL_HBUS_SET	0x80040034

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_CLKCTRL_HBUS_TOG	0x8004003c
HW_CLKCTRL_IR	0x800400b0
HW_CLKCTRL_PIX	0x80040060
HW_CLKCTRL_PLLCTRL0	0x80040000
HW_CLKCTRL_PLLCTRL0_CLR	0x80040008
HW_CLKCTRL_PLLCTRL0_SET	0x80040004
HW_CLKCTRL_PLLCTRL0_TOG	0x8004000C
HW_CLKCTRL_PLLCTRL1	0x80040010
HW_CLKCTRL_RESET	0x80040120
HW_CLKCTRL_SAIF	0x800400c0
HW_CLKCTRL_SPDIF	0x80040090
HW_CLKCTRL_SSP	0x80040070
HW_CLKCTRL_STATUS	0x80040130
HW_CLKCTRL_TV	0x800400d0
HW_CLKCTRL_VERSION	0x80040140
HW_CLKCTRL_XBUS	0x80040040
HW_CLKCTRL_XTAL	0x80040050
HW_CLKCTRL_XTAL_CLR	0x80040058
HW_CLKCTRL_XTAL_SET	0x80040054
HW_CLKCTRL_XTAL_TOG	0x8004005C
HW_DCP_CAPABILITY0	0x80028030
HW_DCP_CAPABILITY1	0x80028040
HW_DCP_CH0CMDPTR	0x80028100
HW_DCP_CH0OPTS	0x80028130
HW_DCP_CH0OPTS_CLR	0x80028138
HW_DCP_CH0OPTS_SET	0x80028134
HW_DCP_CH0OPTS_TOG	0x8002813C
HW_DCP_CH0SEMA	0x80028110
HW_DCP_CH0STAT	0x80028120
HW_DCP_CH0STAT_CLR	0x80028128
HW_DCP_CH0STAT_SET	0x80028124
HW_DCP_CH0STAT_TOG	0x8002812C
HW_DCP_CH1CMDPTR	0x80028140

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_DCP_CH1OPTS	0x80028170
HW_DCP_CH1OPTS_CLR	0x80028178
HW_DCP_CH1OPTS_SET	0x80028174
HW_DCP_CH1OPTS_TOG	0x8002817C
HW_DCP_CH1SEMA	0x80028150
HW_DCP_CH1STAT	0x80028160
HW_DCP_CH1STAT_CLR	0x80028168
HW_DCP_CH1STAT_SET	0x80028164
HW_DCP_CH1STAT_TOG	0x8002816C
HW_DCP_CH2CMDPTR	0x80028180
HW_DCP_CH2OPTS	0x800281B0
HW_DCP_CH2OPTS_CLR	0x800281B8
HW_DCP_CH2OPTS_SET	0x800281B4
HW_DCP_CH2OPTS_TOG	0x800281BC
HW_DCP_CH2SEMA	0x80028190
HW_DCP_CH2STAT	0x800281A0
HW_DCP_CH2STAT_CLR	0x800281A8
HW_DCP_CH2STAT_SET	0x800281A4
HW_DCP_CH2STAT_TOG	0x800281AC
HW_DCP_CH3CMDPTR	0x800281C0
HW_DCP_CH3OPTS	0x800281F0
HW_DCP_CH3OPTS_CLR	0x800281F8
HW_DCP_CH3OPTS_SET	0x800281F4
HW_DCP_CH3OPTS_TOG	0x800281FC
HW_DCP_CH3SEMA	0x800281D0
HW_DCP_CH3STAT	0x800281E0
HW_DCP_CH3STAT_CLR	0x800281E8
HW_DCP_CH3STAT_SET	0x800281E4
HW_DCP_CH3STAT_TOG	0x800281EC
HW_DCP_CHANNELCTRL	0x80028020
HW_DCP_CHANNELCTRL_CLR	0x80028028
HW_DCP_CHANNELCTRL_SET	0x80028024
HW_DCP_CHANNELCTRL_TOG	0x8002802C

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_DCP_CONTEXT	0x80028050
HW_DCP_CSCCHROMAU	0x80028360
HW_DCP_CSCCHROMAV	0x80028370
HW_DCP_CSCCLIP	0x800283D0
HW_DCP_CSCCOEFF0	0x80028380
HW_DCP_CSCCOEFF1	0x80028390
HW_DCP_CSCCOEFF2	0x800283A0
HW_DCP_CSCCTRL0	0x80028300
HW_DCP_CSCCTRL0_CLR	0x80028308
HW_DCP_CSCCTRL0_SET	0x80028304
HW_DCP_CSCCTRL0_TOG	0x8002830C
HW_DCP_CSCINBUFPARAM	0x80028330
HW_DCP_CSCLUMA	0x80028350
HW_DCP_CSCOUTBUFPARAM	0x80028320
HW_DCP_CSCRGB	0x80028340
HW_DCP_CSCSTAT	0x80028310
HW_DCP_CSCSTAT_CLR	0x80028318
HW_DCP_CSCSTAT_SET	0x80028314
HW_DCP_CSCSTAT_TOG	0x8002831C
HW_DCP_CSCXSCALE	0x800283E0
HW_DCP_CSCYSCALE	0x800283F0
HW_DCP_CTRL	0x80028000
HW_DCP_CTRL_CLR	0x80028008
HW_DCP_CTRL_SET	0x80028004
HW_DCP_CTRL_TOG	0x8002800C
HW_DCP_DBGDATA	0x80028410
HW_DCP_DBGSELECT	0x80028400
HW_DCP_KEY	0x80028060
HW_DCP_KEYDATA	0x80028070
HW_DCP_PACKET0	0x80028080
HW_DCP_PACKET1	0x80028090
HW_DCP_PACKET2	0x800280A0
HW_DCP_PACKET3	0x800280B0

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_DCP_PACKET4	0x800280C0
HW_DCP_PACKET5	0x800280D0
HW_DCP_PACKET6	0x800280E0
HW_DCP_PAGETABLE	0x80028420
HW_DCP_STAT	0x80028010
HW_DCP_STAT_CLR	0x80028018
HW_DCP_STAT_SET	0x80028014
HW_DCP_STAT_TOG	0x8002801C
HW_DCP_VERSION	0x80028430
HW_DIGCTL_AHB_STATS_SELECT	0x8001C330
HW_DIGCTL_ARMCACHE	0x8001C2B0
HW_DIGCTL_CHIPID	0x8001C310
HW_DIGCTL_CTRL	0x8001C000
HW_DIGCTL_CTRL_CLR	0x8001C008
HW_DIGCTL_CTRL_SET	0x8001C004
HW_DIGCTL_CTRL_TOG	0x8001C00C
HW_DIGCTL_DBG	0x8001C0E0
HW_DIGCTL_DBGRD	0x8001C0D0
HW_DIGCTL_DEBUG_TRAP_ADDR_HIGH	0x8001C2D0
HW_DIGCTL_DEBUG_TRAP_ADDR_LOW	0x8001C2C0
HW_DIGCTL_EMICLK_DELAY	0x8001C500
HW_DIGCTL_ENTROPY	0x8001C090
HW_DIGCTL_ENTROPY_LATCHED	0x8001C0A0
HW_DIGCTL_HCLKCOUNT	0x8001C020
HW_DIGCTL_HCLKCOUNT_CLR	0x8001C028
HW_DIGCTL_HCLKCOUNT_SET	0x8001C024
HW_DIGCTL_HCLKCOUNT_TOG	0x8001C02C
HW_DIGCTL_L0_AHB_ACTIVE_CYCLES	0x8001C340
HW_DIGCTL_L0_AHB_DATA_CYCLES	0x8001C360
HW_DIGCTL_L0_AHB_DATA_STALLED	0x8001C350
HW_DIGCTL_L1_AHB_ACTIVE_CYCLES	0x8001C370
HW_DIGCTL_L1_AHB_DATA_CYCLES	0x8001C390
HW_DIGCTL_L1_AHB_DATA_STALLED	0x8001C380

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_DIGCTL_L2_AHB_ACTIVE_CYCLES	0x8001C3A0
HW_DIGCTL_L2_AHB_DATA_CYCLES	0x8001C3C0
HW_DIGCTL_L2_AHB_DATA_STALLED	0x8001C3B0
HW_DIGCTL_L3_AHB_ACTIVE_CYCLES	0x8001C3D0
HW_DIGCTL_L3_AHB_DATA_CYCLES	0x8001C3F0
HW_DIGCTL_L3_AHB_DATA_STALLED	0x8001C3E0
HW_DIGCTL_MICROSECONDS	0x8001C0C0
HW_DIGCTL_MICROSECONDS_CLR	0x8001C0C8
HW_DIGCTL_MICROSECONDS_SET	0x8001C0C4
HW_DIGCTL_MICROSECONDS_TOG	0x8001C0CC
HW_DIGCTL_MPTE0_LOC	0x8001C400
HW_DIGCTL_MPTE1_LOC	0x8001C410
HW_DIGCTL_MPTE10_LOC	0x8001C4A0
HW_DIGCTL_MPTE11_LOC	0x8001C4B0
HW_DIGCTL_MPTE12_LOC	0x8001C4C0
HW_DIGCTL_MPTE13_LOC	0x8001C4D0
HW_DIGCTL_MPTE14_LOC	0x8001C4E0
HW_DIGCTL_MPTE15_LOC	0x8001C4F0
HW_DIGCTL_MPTE2_LOC	0x8001C420
HW_DIGCTL_MPTE3_LOC	0x8001C430
HW_DIGCTL_MPTE4_LOC	0x8001C440
HW_DIGCTL_MPTE5_LOC	0x8001C450
HW_DIGCTL_MPTE6_LOC	0x8001C460
HW_DIGCTL_MPTE7_LOC	0x8001C470
HW_DIGCTL_MPTE8_LOC	0x8001C480
HW_DIGCTL_MPTE9_LOC	0x8001C490
HW_DIGCTL_OCRAM_BIST_CSR	0x8001C0F0
HW_DIGCTL_OCRAM_BIST_CSR_CLR	0x8001C0F8
HW_DIGCTL_OCRAM_BIST_CSR_SET	0x8001C0F4
HW_DIGCTL_OCRAM_BIST_CSR_TOG	0x8001C0FC
HW_DIGCTL_OCRAM_STATUS0	0x8001C110
HW_DIGCTL_OCRAM_STATUS0_CLR	0x8001C118
HW_DIGCTL_OCRAM_STATUS0_SET	0x8001C114

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_DIGCTL_OCRAM_STATUS0_TOG	0x8001C11C
HW_DIGCTL_OCRAM_STATUS1	0x8001C120
HW_DIGCTL_OCRAM_STATUS1_CLR	0x8001C128
HW_DIGCTL_OCRAM_STATUS1_SET	0x8001C124
HW_DIGCTL_OCRAM_STATUS1_TOG	0x8001C12C
HW_DIGCTL_OCRAM_STATUS10	0x8001C1B0
HW_DIGCTL_OCRAM_STATUS10_CLR	0x8001C1B8
HW_DIGCTL_OCRAM_STATUS10_SET	0x8001C1B4
HW_DIGCTL_OCRAM_STATUS10_TOG	0x8001C1BC
HW_DIGCTL_OCRAM_STATUS11	0x8001C1C0
HW_DIGCTL_OCRAM_STATUS11_CLR	0x8001C1C8
HW_DIGCTL_OCRAM_STATUS11_SET	0x8001C1C4
HW_DIGCTL_OCRAM_STATUS11_TOG	0x8001C1CC
HW_DIGCTL_OCRAM_STATUS12	0x8001C1D0
HW_DIGCTL_OCRAM_STATUS12_CLR	0x8001C1D8
HW_DIGCTL_OCRAM_STATUS12_SET	0x8001C1D4
HW_DIGCTL_OCRAM_STATUS12_TOG	0x8001C1DC
HW_DIGCTL_OCRAM_STATUS13	0x8001C1E0
HW_DIGCTL_OCRAM_STATUS13_CLR	0x8001C1E8
HW_DIGCTL_OCRAM_STATUS13_SET	0x8001C1E4
HW_DIGCTL_OCRAM_STATUS13_TOG	0x8001C1EC
HW_DIGCTL_OCRAM_STATUS2	0x8001C130
HW_DIGCTL_OCRAM_STATUS2_CLR	0x8001C138
HW_DIGCTL_OCRAM_STATUS2_SET	0x8001C134
HW_DIGCTL_OCRAM_STATUS2_TOG	0x8001C13C
HW_DIGCTL_OCRAM_STATUS3	0x8001C140
HW_DIGCTL_OCRAM_STATUS3_CLR	0x8001C148
HW_DIGCTL_OCRAM_STATUS3_SET	0x8001C144
HW_DIGCTL_OCRAM_STATUS3_TOG	0x8001C14C
HW_DIGCTL_OCRAM_STATUS4	0x8001C150
HW_DIGCTL_OCRAM_STATUS4_CLR	0x8001C158
HW_DIGCTL_OCRAM_STATUS4_SET	0x8001C154
HW_DIGCTL_OCRAM_STATUS4_TOG	0x8001C15C

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_DIGCTL_OCRAM_STATUS5	0x8001C160
HW_DIGCTL_OCRAM_STATUS5_CLR	0x8001C168
HW_DIGCTL_OCRAM_STATUS5_SET	0x8001C164
HW_DIGCTL_OCRAM_STATUS5_TOG	0x8001C16C
HW_DIGCTL_OCRAM_STATUS6	0x8001C170
HW_DIGCTL_OCRAM_STATUS6_CLR	0x8001C178
HW_DIGCTL_OCRAM_STATUS6_SET	0x8001C174
HW_DIGCTL_OCRAM_STATUS6_TOG	0x8001C17C
HW_DIGCTL_OCRAM_STATUS7	0x8001C180
HW_DIGCTL_OCRAM_STATUS7_CLR	0x8001C188
HW_DIGCTL_OCRAM_STATUS7_SET	0x8001C184
HW_DIGCTL_OCRAM_STATUS7_TOG	0x8001C18C
HW_DIGCTL_OCRAM_STATUS8	0x8001C190
HW_DIGCTL_OCRAM_STATUS8_CLR	0x8001C198
HW_DIGCTL_OCRAM_STATUS8_SET	0x8001C194
HW_DIGCTL_OCRAM_STATUS8_TOG	0x8001C19C
HW_DIGCTL_OCRAM_STATUS9	0x8001C1A0
HW_DIGCTL_OCRAM_STATUS9_CLR	0x8001C1A8
HW_DIGCTL_OCRAM_STATUS9_SET	0x8001C1A4
HW_DIGCTL_OCRAM_STATUS9_TOG	0x8001C1AC
HW_DIGCTL_RAMCTRL	0x8001C030
HW_DIGCTL_RAMCTRL_CLR	0x8001C038
HW_DIGCTL_RAMCTRL_SET	0x8001C034
HW_DIGCTL_RAMCTRL_TOG	0x8001C03C
HW_DIGCTL_RAMREPAIR	0x8001C040
HW_DIGCTL_RAMREPAIR_CLR	0x8001C048
HW_DIGCTL_RAMREPAIR_SET	0x8001C044
HW_DIGCTL_RAMREPAIR_TOG	0x8001C04C
HW_DIGCTL_ROMCTRL	0x8001C050
HW_DIGCTL_ROMCTRL_CLR	0x8001C058
HW_DIGCTL_ROMCTRL_SET	0x8001C054
HW_DIGCTL_ROMCTRL_TOG	0x8001C05C
HW_DIGCTL_SCRATCH0	0x8001C290

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_DIGCTL_SCRATCH1	0x8001C2A0
HW_DIGCTL_SGTL	0x8001C300
HW_DIGCTL_SJTAGDBG	0x8001C0B0
HW_DIGCTL_SJTAGDBG_CLR	0x8001C0B8
HW_DIGCTL_SJTAGDBG_SET	0x8001C0B4
HW_DIGCTL_SJTAGDBG_TOG	0x8001C0BC
HW_DIGCTL_STATUS	0x8001C010
HW_DIGCTL_STATUS_CLR	0x8001C018
HW_DIGCTL_STATUS_SET	0x8001C014
HW_DIGCTL_STATUS_TOG	0x8001C01C
HW_DIGCTL_WRITEONCE	0x8001C060
HW_DRAM_CTL00	0x800E0000
HW_DRAM_CTL01	0x800E0004
HW_DRAM_CTL02	0x800E0008
HW_DRAM_CTL03	0x800E000C
HW_DRAM_CTL04	0x800E0010
HW_DRAM_CTL05	0x800E0014
HW_DRAM_CTL06	0x800E0018
HW_DRAM_CTL07	0x800E001C
HW_DRAM_CTL08	0x800E0020
HW_DRAM_CTL09	0x800E0024
HW_DRAM_CTL10	0x800E0028
HW_DRAM_CTL11	0x800E002C
HW_DRAM_CTL12	0x800E0030
HW_DRAM_CTL13	0x800E0034
HW_DRAM_CTL14	0x800E0038
HW_DRAM_CTL15	0x800E003C
HW_DRAM_CTL16	0x800E0040
HW_DRAM_CTL17	0x800E0044
HW_DRAM_CTL18	0x800E0048
HW_DRAM_CTL19	0x800E004C
HW_DRAM_CTL20	0x800E0050
HW_DRAM_CTL21	0x800E0054

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_DRAM_CTL22	0x800E0058
HW_DRAM_CTL23	0x800E005C
HW_DRAM_CTL24	0x800E0060
HW_DRAM_CTL25	0x800E0064
HW_DRAM_CTL26	0x800E0068
HW_DRAM_CTL27	0x800E006C
HW_DRAM_CTL28	0x800E0070
HW_DRAM_CTL29	0x800E0074
HW_DRAM_CTL30	0x800E0078
HW_DRAM_CTL31	0x800E007C
HW_DRAM_CTL32	0x800E0080
HW_DRAM_CTL33	0x800E0084
HW_DRAM_CTL34	0x800E0088
HW_DRAM_CTL35	0x800E008C
HW_DRAM_CTL36	0x800E0090
HW_DRAM_CTL37	0x800E0094
HW_DRAM_CTL38	0x800E0098
HW_DRAM_CTL39	0x800E009C
HW_DRAM_CTL40	0x800E00A0
HW_ECC8_BLOCKNAME	0x80008080
HW_ECC8_CTRL	0x80008000
HW_ECC8_CTRL_CLR	0x80008008
HW_ECC8_CTRL_SET	0x80008004
HW_ECC8_CTRL_TOG	0x8000800C
HW_ECC8_DBGAHBMREAD	0x80008070
HW_ECC8_DBGCSFEREAD	0x80008050
HW_ECC8_DBGKESREAD	0x80008040
HW_ECC8_DBGSYNDGENREAD	0x80008060
HW_ECC8_DEBUG0	0x80008030
HW_ECC8_DEBUG0_CLR	0x80008038
HW_ECC8_DEBUG0_SET	0x80008034
HW_ECC8_DEBUG0_TOG	0x8000803C
HW_ECC8_STATUS0	0x80008010

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ECC8_STATUS1	0x80008020
HW_ECC8_VERSION	0x800080a0
HW_EMI_CTRL	0x80020000
HW_EMI_CTRL_CLR	0x80020008
HW_EMI_CTRL_SET	0x80020004
HW_EMI_CTRL_TOG	0x8002000C
HW_EMI_DDR_TEST_MODE_CSR	0x80020030
HW_EMI_DDR_TEST_MODE_CSR_CLR	0x80020038
HW_EMI_DDR_TEST_MODE_CSR_SET	0x80020034
HW_EMI_DDR_TEST_MODE_CSR_TOG	0x8002003C
HW_EMI_DDR_TEST_MODE_STATUS0	0x80020090
HW_EMI_DDR_TEST_MODE_STATUS1	0x800200A0
HW_EMI_DDR_TEST_MODE_STATUS2	0x800200B0
HW_EMI_DDR_TEST_MODE_STATUS3	0x800200C0
HW_EMI_VERSION	0x800200F0
HW_GPMI_AUXILIARY	0x8000C050
HW_GPMI_COMPARE	0x8000C010
HW_GPMI_CTRL0	0x8000C000
HW_GPMI_CTRL0_CLR	0x8000C008
HW_GPMI_CTRL0_SET	0x8000C004
HW_GPMI_CTRL0_TOG	0x8000C00C
HW_GPMI_CTRL1	0x8000C060
HW_GPMI_CTRL1_CLR	0x8000C068
HW_GPMI_CTRL1_SET	0x8000C064
HW_GPMI_CTRL1_TOG	0x8000C06C
HW_GPMI_DATA	0x8000C0A0
HW_GPMI_DEBUG	0x8000C0C0
HW_GPMI_DEBUG2	0x8000C0E0
HW_GPMI_DEBUG3	0x8000C0F0
HW_GPMI_ECCCOUNT	0x8000C030
HW_GPMI_ECCCTRL	0x8000C020
HW_GPMI_ECCCTRL_CLR	0x8000C028
HW_GPMI_ECCCTRL_SET	0x8000C024

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_GPMI_ECCCTRL_TOG	0x8000C02C
HW_GPMI_PAYLOAD	0x8000C040
HW_GPMI_STAT	0x8000C0B0
HW_GPMI_TIMING0	0x8000C070
HW_GPMI_TIMING1	0x8000C080
HW_GPMI_TIMING2	0x8000C090
HW_GPMI_VERSION	0x8000C0D0
HW_I2C_CTRL0	0x80058000
HW_I2C_CTRL0_CLR	0x80058008
HW_I2C_CTRL0_SET	0x80058004
HW_I2C_CTRL0_TOG	0x8005800C
HW_I2C_CTRL1	0x80058040
HW_I2C_CTRL1_CLR	0x80058048
HW_I2C_CTRL1_SET	0x80058044
HW_I2C_CTRL1_TOG	0x8005804C
HW_I2C_DATA	0x80058060
HW_I2C_DEBUG0	0x80058070
HW_I2C_DEBUG0_CLR	0x80058078
HW_I2C_DEBUG0_SET	0x80058074
HW_I2C_DEBUG0_TOG	0x8005807C
HW_I2C_DEBUG1	0x80058080
HW_I2C_DEBUG1_CLR	0x80058088
HW_I2C_DEBUG1_SET	0x80058084
HW_I2C_DEBUG1_TOG	0x8005808C
HW_I2C_STAT	0x80058050
HW_I2C_TIMING0	0x80058010
HW_I2C_TIMING0_CLR	0x80058018
HW_I2C_TIMING0_SET	0x80058014
HW_I2C_TIMING0_TOG	0x8005801C
HW_I2C_TIMING1	0x80058020
HW_I2C_TIMING1_CLR	0x80058028
HW_I2C_TIMING1_SET	0x80058024
HW_I2C_TIMING1_TOG	0x8005802C

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_I2C_TIMING2	0x80058030
HW_I2C_TIMING2_CLR	0x80058038
HW_I2C_TIMING2_SET	0x80058034
HW_I2C_TIMING2_TOG	0x8005803C
HW_I2C_VERSION	0x80058090
HW_ICOLL_CTRL	0x80000020
HW_ICOLL_CTRL_CLR	0x80000028
HW_ICOLL_CTRL_SET	0x80000024
HW_ICOLL_CTRL_TOG	0x8000002C
HW_ICOLL_DBGFLAG	0x80001150
HW_ICOLL_DBGFLAG_CLR	0x80001158
HW_ICOLL_DBGFLAG_SET	0x80001154
HW_ICOLL_DBGFLAG_TOG	0x8000115C
HW_ICOLL_DBGREAD0	0x80001130
HW_ICOLL_DBGREAD0_CLR	0x80001138
HW_ICOLL_DBGREAD0_SET	0x80001134
HW_ICOLL_DBGREAD0_TOG	0x8000113C
HW_ICOLL_DBGREAD1	0x80001140
HW_ICOLL_DBGREAD1_CLR	0x80001148
HW_ICOLL_DBGREAD1_SET	0x80001144
HW_ICOLL_DBGREAD1_TOG	0x8000114C
HW_ICOLL_DBGREQUEST0	0x80001160
HW_ICOLL_DBGREQUEST0_CLR	0x80001168
HW_ICOLL_DBGREQUEST0_SET	0x80001164
HW_ICOLL_DBGREQUEST0_TOG	0x8000116C
HW_ICOLL_DBGREQUEST1	0x80001170
HW_ICOLL_DBGREQUEST1_CLR	0x80001178
HW_ICOLL_DBGREQUEST1_SET	0x80001174
HW_ICOLL_DBGREQUEST1_TOG	0x8000117C
HW_ICOLL_DBGREQUEST2	0x80001180
HW_ICOLL_DBGREQUEST2_CLR	0x80001188
HW_ICOLL_DBGREQUEST2_SET	0x80001184
HW_ICOLL_DBGREQUEST2_TOG	0x8000118C

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ICOLL_DBGREQUEST3	0x80001190
HW_ICOLL_DBGREQUEST3_CLR	0x80001198
HW_ICOLL_DBGREQUEST3_SET	0x80001194
HW_ICOLL_DBGREQUEST3_TOG	0x8000119C
HW_ICOLL_DEBUG	0x80001120
HW_ICOLL_DEBUG_CLR	0x80001128
HW_ICOLL_DEBUG_SET	0x80001124
HW_ICOLL_DEBUG_TOG	0x8000112C
HW_ICOLL_INTERRUPT0	0x80000120
HW_ICOLL_INTERRUPT0_CLR	0x80000128
HW_ICOLL_INTERRUPT0_SET	0x80000124
HW_ICOLL_INTERRUPT0_TOG	0x8000012C
HW_ICOLL_INTERRUPT1	0x80000130
HW_ICOLL_INTERRUPT1_CLR	0x80000138
HW_ICOLL_INTERRUPT1_SET	0x80000134
HW_ICOLL_INTERRUPT1_TOG	0x8000013C
HW_ICOLL_INTERRUPT10	0x800001C0
HW_ICOLL_INTERRUPT10_CLR	0x800001C8
HW_ICOLL_INTERRUPT10_SET	0x800001C4
HW_ICOLL_INTERRUPT10_TOG	0x800001CC
HW_ICOLL_INTERRUPT100	0x80000760
HW_ICOLL_INTERRUPT100_CLR	0x80000768
HW_ICOLL_INTERRUPT100_SET	0x80000764
HW_ICOLL_INTERRUPT100_TOG	0x8000076C
HW_ICOLL_INTERRUPT101	0x80000770
HW_ICOLL_INTERRUPT101_CLR	0x80000778
HW_ICOLL_INTERRUPT101_SET	0x80000774
HW_ICOLL_INTERRUPT101_TOG	0x8000077C
HW_ICOLL_INTERRUPT102	0x80000780
HW_ICOLL_INTERRUPT102_CLR	0x80000788
HW_ICOLL_INTERRUPT102_SET	0x80000784
HW_ICOLL_INTERRUPT102_TOG	0x8000078C
HW_ICOLL_INTERRUPT103	0x80000790

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ICOLL_INTERRUPT103_CLR	0x80000798
HW_ICOLL_INTERRUPT103_SET	0x80000794
HW_ICOLL_INTERRUPT103_TOG	0x8000079C
HW_ICOLL_INTERRUPT104	0x800007A0
HW_ICOLL_INTERRUPT104_CLR	0x800007A8
HW_ICOLL_INTERRUPT104_SET	0x800007A4
HW_ICOLL_INTERRUPT104_TOG	0x800007AC
HW_ICOLL_INTERRUPT105	0x800007B0
HW_ICOLL_INTERRUPT105_CLR	0x800007B8
HW_ICOLL_INTERRUPT105_SET	0x800007B4
HW_ICOLL_INTERRUPT105_TOG	0x800007BC
HW_ICOLL_INTERRUPT106	0x800007C0
HW_ICOLL_INTERRUPT106_CLR	0x800007C8
HW_ICOLL_INTERRUPT106_SET	0x800007C4
HW_ICOLL_INTERRUPT106_TOG	0x800007CC
HW_ICOLL_INTERRUPT107	0x800007D0
HW_ICOLL_INTERRUPT107_CLR	0x800007D8
HW_ICOLL_INTERRUPT107_SET	0x800007D4
HW_ICOLL_INTERRUPT107_TOG	0x800007DC
HW_ICOLL_INTERRUPT108	0x800007E0
HW_ICOLL_INTERRUPT108_CLR	0x800007E8
HW_ICOLL_INTERRUPT108_SET	0x800007E4
HW_ICOLL_INTERRUPT108_TOG	0x800007EC
HW_ICOLL_INTERRUPT109	0x800007F0
HW_ICOLL_INTERRUPT109_CLR	0x800007F8
HW_ICOLL_INTERRUPT109_SET	0x800007F4
HW_ICOLL_INTERRUPT109_TOG	0x800007FC
HW_ICOLL_INTERRUPT11	0x800001D0
HW_ICOLL_INTERRUPT11_CLR	0x800001D8
HW_ICOLL_INTERRUPT11_SET	0x800001D4
HW_ICOLL_INTERRUPT11_TOG	0x800001DC
HW_ICOLL_INTERRUPT110	0x80000800
HW_ICOLL_INTERRUPT110_CLR	0x80000808

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ICOLL_INTERRUPT110_SET	0x80000804
HW_ICOLL_INTERRUPT110_TOG	0x8000080C
HW_ICOLL_INTERRUPT111	0x80000810
HW_ICOLL_INTERRUPT111_CLR	0x80000818
HW_ICOLL_INTERRUPT111_SET	0x80000814
HW_ICOLL_INTERRUPT111_TOG	0x8000081C
HW_ICOLL_INTERRUPT112	0x80000820
HW_ICOLL_INTERRUPT112_CLR	0x80000828
HW_ICOLL_INTERRUPT112_SET	0x80000824
HW_ICOLL_INTERRUPT112_TOG	0x8000082C
HW_ICOLL_INTERRUPT113	0x80000830
HW_ICOLL_INTERRUPT113_CLR	0x80000838
HW_ICOLL_INTERRUPT113_SET	0x80000834
HW_ICOLL_INTERRUPT113_TOG	0x8000083C
HW_ICOLL_INTERRUPT114	0x80000840
HW_ICOLL_INTERRUPT114_CLR	0x80000848
HW_ICOLL_INTERRUPT114_SET	0x80000844
HW_ICOLL_INTERRUPT114_TOG	0x8000084C
HW_ICOLL_INTERRUPT115	0x80000850
HW_ICOLL_INTERRUPT115_CLR	0x80000858
HW_ICOLL_INTERRUPT115_SET	0x80000854
HW_ICOLL_INTERRUPT115_TOG	0x8000085C
HW_ICOLL_INTERRUPT116	0x80000860
HW_ICOLL_INTERRUPT116_CLR	0x80000868
HW_ICOLL_INTERRUPT116_SET	0x80000864
HW_ICOLL_INTERRUPT116_TOG	0x8000086C
HW_ICOLL_INTERRUPT117	0x80000870
HW_ICOLL_INTERRUPT117_CLR	0x80000878
HW_ICOLL_INTERRUPT117_SET	0x80000874
HW_ICOLL_INTERRUPT117_TOG	0x8000087C
HW_ICOLL_INTERRUPT118	0x80000880
HW_ICOLL_INTERRUPT118_CLR	0x80000888
HW_ICOLL_INTERRUPT118_SET	0x80000884

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ICOLL_INTERRUPT118_TOG	0x8000088C
HW_ICOLL_INTERRUPT119	0x80000890
HW_ICOLL_INTERRUPT119_CLR	0x80000898
HW_ICOLL_INTERRUPT119_SET	0x80000894
HW_ICOLL_INTERRUPT119_TOG	0x8000089C
HW_ICOLL_INTERRUPT12	0x800001E0
HW_ICOLL_INTERRUPT12_CLR	0x800001E8
HW_ICOLL_INTERRUPT12_SET	0x800001E4
HW_ICOLL_INTERRUPT12_TOG	0x800001EC
HW_ICOLL_INTERRUPT120	0x800008A0
HW_ICOLL_INTERRUPT120_CLR	0x800008A8
HW_ICOLL_INTERRUPT120_SET	0x800008A4
HW_ICOLL_INTERRUPT120_TOG	0x800008AC
HW_ICOLL_INTERRUPT121	0x800008B0
HW_ICOLL_INTERRUPT121_CLR	0x800008B8
HW_ICOLL_INTERRUPT121_SET	0x800008B4
HW_ICOLL_INTERRUPT121_TOG	0x800008BC
HW_ICOLL_INTERRUPT122	0x800008C0
HW_ICOLL_INTERRUPT122_CLR	0x800008C8
HW_ICOLL_INTERRUPT122_SET	0x800008C4
HW_ICOLL_INTERRUPT122_TOG	0x800008CC
HW_ICOLL_INTERRUPT123	0x800008D0
HW_ICOLL_INTERRUPT123_CLR	0x800008D8
HW_ICOLL_INTERRUPT123_SET	0x800008D4
HW_ICOLL_INTERRUPT123_TOG	0x800008DC
HW_ICOLL_INTERRUPT124	0x800008E0
HW_ICOLL_INTERRUPT124_CLR	0x800008E8
HW_ICOLL_INTERRUPT124_SET	0x800008E4
HW_ICOLL_INTERRUPT124_TOG	0x800008EC
HW_ICOLL_INTERRUPT125	0x800008F0
HW_ICOLL_INTERRUPT125_CLR	0x800008F8
HW_ICOLL_INTERRUPT125_SET	0x800008F4
HW_ICOLL_INTERRUPT125_TOG	0x800008FC

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ICOLL_INTERRUPT126	0x80000900
HW_ICOLL_INTERRUPT126_CLR	0x80000908
HW_ICOLL_INTERRUPT126_SET	0x80000904
HW_ICOLL_INTERRUPT126_TOG	0x8000090C
HW_ICOLL_INTERRUPT127	0x80000910
HW_ICOLL_INTERRUPT127_CLR	0x80000918
HW_ICOLL_INTERRUPT127_SET	0x80000914
HW_ICOLL_INTERRUPT127_TOG	0x8000091C
HW_ICOLL_INTERRUPT13	0x800001F0
HW_ICOLL_INTERRUPT13_CLR	0x800001F8
HW_ICOLL_INTERRUPT13_SET	0x800001F4
HW_ICOLL_INTERRUPT13_TOG	0x800001FC
HW_ICOLL_INTERRUPT14	0x80000200
HW_ICOLL_INTERRUPT14_CLR	0x80000208
HW_ICOLL_INTERRUPT14_SET	0x80000204
HW_ICOLL_INTERRUPT14_TOG	0x8000020C
HW_ICOLL_INTERRUPT15	0x80000210
HW_ICOLL_INTERRUPT15_CLR	0x80000218
HW_ICOLL_INTERRUPT15_SET	0x80000214
HW_ICOLL_INTERRUPT15_TOG	0x8000021C
HW_ICOLL_INTERRUPT16	0x80000220
HW_ICOLL_INTERRUPT16_CLR	0x80000228
HW_ICOLL_INTERRUPT16_SET	0x80000224
HW_ICOLL_INTERRUPT16_TOG	0x8000022C
HW_ICOLL_INTERRUPT17	0x80000230
HW_ICOLL_INTERRUPT17_CLR	0x80000238
HW_ICOLL_INTERRUPT17_SET	0x80000234
HW_ICOLL_INTERRUPT17_TOG	0x8000023C
HW_ICOLL_INTERRUPT18	0x80000240
HW_ICOLL_INTERRUPT18_CLR	0x80000248
HW_ICOLL_INTERRUPT18_SET	0x80000244
HW_ICOLL_INTERRUPT18_TOG	0x8000024C
HW_ICOLL_INTERRUPT19	0x80000250

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ICOLL_INTERRUPT19_CLR	0x80000258
HW_ICOLL_INTERRUPT19_SET	0x80000254
HW_ICOLL_INTERRUPT19_TOG	0x8000025C
HW_ICOLL_INTERRUPT2	0x80000140
HW_ICOLL_INTERRUPT2_CLR	0x80000148
HW_ICOLL_INTERRUPT2_SET	0x80000144
HW_ICOLL_INTERRUPT2_TOG	0x8000014C
HW_ICOLL_INTERRUPT20	0x80000260
HW_ICOLL_INTERRUPT20_CLR	0x80000268
HW_ICOLL_INTERRUPT20_SET	0x80000264
HW_ICOLL_INTERRUPT20_TOG	0x8000026C
HW_ICOLL_INTERRUPT21	0x80000270
HW_ICOLL_INTERRUPT21_CLR	0x80000278
HW_ICOLL_INTERRUPT21_SET	0x80000274
HW_ICOLL_INTERRUPT21_TOG	0x8000027C
HW_ICOLL_INTERRUPT22	0x80000280
HW_ICOLL_INTERRUPT22_CLR	0x80000288
HW_ICOLL_INTERRUPT22_SET	0x80000284
HW_ICOLL_INTERRUPT22_TOG	0x8000028C
HW_ICOLL_INTERRUPT23	0x80000290
HW_ICOLL_INTERRUPT23_CLR	0x80000298
HW_ICOLL_INTERRUPT23_SET	0x80000294
HW_ICOLL_INTERRUPT23_TOG	0x8000029C
HW_ICOLL_INTERRUPT24	0x800002A0
HW_ICOLL_INTERRUPT24_CLR	0x800002A8
HW_ICOLL_INTERRUPT24_SET	0x800002A4
HW_ICOLL_INTERRUPT24_TOG	0x800002AC
HW_ICOLL_INTERRUPT25	0x800002B0
HW_ICOLL_INTERRUPT25_CLR	0x800002B8
HW_ICOLL_INTERRUPT25_SET	0x800002B4
HW_ICOLL_INTERRUPT25_TOG	0x800002BC
HW_ICOLL_INTERRUPT26	0x800002C0
HW_ICOLL_INTERRUPT26_CLR	0x800002C8

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ICOLL_INTERRUPT26_SET	0x800002C4
HW_ICOLL_INTERRUPT26_TOG	0x800002CC
HW_ICOLL_INTERRUPT27	0x800002D0
HW_ICOLL_INTERRUPT27_CLR	0x800002D8
HW_ICOLL_INTERRUPT27_SET	0x800002D4
HW_ICOLL_INTERRUPT27_TOG	0x800002DC
HW_ICOLL_INTERRUPT28	0x800002E0
HW_ICOLL_INTERRUPT28_CLR	0x800002E8
HW_ICOLL_INTERRUPT28_SET	0x800002E4
HW_ICOLL_INTERRUPT28_TOG	0x800002EC
HW_ICOLL_INTERRUPT29	0x800002F0
HW_ICOLL_INTERRUPT29_CLR	0x800002F8
HW_ICOLL_INTERRUPT29_SET	0x800002F4
HW_ICOLL_INTERRUPT29_TOG	0x800002FC
HW_ICOLL_INTERRUPT3	0x80000150
HW_ICOLL_INTERRUPT3_CLR	0x80000158
HW_ICOLL_INTERRUPT3_SET	0x80000154
HW_ICOLL_INTERRUPT3_TOG	0x8000015C
HW_ICOLL_INTERRUPT30	0x80000300
HW_ICOLL_INTERRUPT30_CLR	0x80000308
HW_ICOLL_INTERRUPT30_SET	0x80000304
HW_ICOLL_INTERRUPT30_TOG	0x8000030C
HW_ICOLL_INTERRUPT31	0x80000310
HW_ICOLL_INTERRUPT31_CLR	0x80000318
HW_ICOLL_INTERRUPT31_SET	0x80000314
HW_ICOLL_INTERRUPT31_TOG	0x8000031C
HW_ICOLL_INTERRUPT32	0x80000320
HW_ICOLL_INTERRUPT32_CLR	0x80000328
HW_ICOLL_INTERRUPT32_SET	0x80000324
HW_ICOLL_INTERRUPT32_TOG	0x8000032C
HW_ICOLL_INTERRUPT33	0x80000330
HW_ICOLL_INTERRUPT33_CLR	0x80000338
HW_ICOLL_INTERRUPT33_SET	0x80000334

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ICOLL_INTERRUPT33_TOG	0x8000033C
HW_ICOLL_INTERRUPT34	0x80000340
HW_ICOLL_INTERRUPT34_CLR	0x80000348
HW_ICOLL_INTERRUPT34_SET	0x80000344
HW_ICOLL_INTERRUPT34_TOG	0x8000034C
HW_ICOLL_INTERRUPT35	0x80000350
HW_ICOLL_INTERRUPT35_CLR	0x80000358
HW_ICOLL_INTERRUPT35_SET	0x80000354
HW_ICOLL_INTERRUPT35_TOG	0x8000035C
HW_ICOLL_INTERRUPT36	0x80000360
HW_ICOLL_INTERRUPT36_CLR	0x80000368
HW_ICOLL_INTERRUPT36_SET	0x80000364
HW_ICOLL_INTERRUPT36_TOG	0x8000036C
HW_ICOLL_INTERRUPT37	0x80000370
HW_ICOLL_INTERRUPT37_CLR	0x80000378
HW_ICOLL_INTERRUPT37_SET	0x80000374
HW_ICOLL_INTERRUPT37_TOG	0x8000037C
HW_ICOLL_INTERRUPT38	0x80000380
HW_ICOLL_INTERRUPT38_CLR	0x80000388
HW_ICOLL_INTERRUPT38_SET	0x80000384
HW_ICOLL_INTERRUPT38_TOG	0x8000038C
HW_ICOLL_INTERRUPT39	0x80000390
HW_ICOLL_INTERRUPT39_CLR	0x80000398
HW_ICOLL_INTERRUPT39_SET	0x80000394
HW_ICOLL_INTERRUPT39_TOG	0x8000039C
HW_ICOLL_INTERRUPT4	0x80000160
HW_ICOLL_INTERRUPT4_CLR	0x80000168
HW_ICOLL_INTERRUPT4_SET	0x80000164
HW_ICOLL_INTERRUPT4_TOG	0x8000016C
HW_ICOLL_INTERRUPT40	0x800003A0
HW_ICOLL_INTERRUPT40_CLR	0x800003A8
HW_ICOLL_INTERRUPT40_SET	0x800003A4
HW_ICOLL_INTERRUPT40_TOG	0x800003AC

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ICOLL_INTERRUPT41	0x800003B0
HW_ICOLL_INTERRUPT41_CLR	0x800003B8
HW_ICOLL_INTERRUPT41_SET	0x800003B4
HW_ICOLL_INTERRUPT41_TOG	0x800003BC
HW_ICOLL_INTERRUPT42	0x800003C0
HW_ICOLL_INTERRUPT42_CLR	0x800003C8
HW_ICOLL_INTERRUPT42_SET	0x800003C4
HW_ICOLL_INTERRUPT42_TOG	0x800003CC
HW_ICOLL_INTERRUPT43	0x800003D0
HW_ICOLL_INTERRUPT43_CLR	0x800003D8
HW_ICOLL_INTERRUPT43_SET	0x800003D4
HW_ICOLL_INTERRUPT43_TOG	0x800003DC
HW_ICOLL_INTERRUPT44	0x800003E0
HW_ICOLL_INTERRUPT44_CLR	0x800003E8
HW_ICOLL_INTERRUPT44_SET	0x800003E4
HW_ICOLL_INTERRUPT44_TOG	0x800003EC
HW_ICOLL_INTERRUPT45	0x800003F0
HW_ICOLL_INTERRUPT45_CLR	0x800003F8
HW_ICOLL_INTERRUPT45_SET	0x800003F4
HW_ICOLL_INTERRUPT45_TOG	0x800003FC
HW_ICOLL_INTERRUPT46	0x80000400
HW_ICOLL_INTERRUPT46_CLR	0x80000408
HW_ICOLL_INTERRUPT46_SET	0x80000404
HW_ICOLL_INTERRUPT46_TOG	0x8000040C
HW_ICOLL_INTERRUPT47	0x80000410
HW_ICOLL_INTERRUPT47_CLR	0x80000418
HW_ICOLL_INTERRUPT47_SET	0x80000414
HW_ICOLL_INTERRUPT47_TOG	0x8000041C
HW_ICOLL_INTERRUPT48	0x80000420
HW_ICOLL_INTERRUPT48_CLR	0x80000428
HW_ICOLL_INTERRUPT48_SET	0x80000424
HW_ICOLL_INTERRUPT48_TOG	0x8000042C
HW_ICOLL_INTERRUPT49	0x80000430

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ICOLL_INTERRUPT49_CLR	0x80000438
HW_ICOLL_INTERRUPT49_SET	0x80000434
HW_ICOLL_INTERRUPT49_TOG	0x8000043C
HW_ICOLL_INTERRUPT5	0x80000170
HW_ICOLL_INTERRUPT5_CLR	0x80000178
HW_ICOLL_INTERRUPT5_SET	0x80000174
HW_ICOLL_INTERRUPT5_TOG	0x8000017C
HW_ICOLL_INTERRUPT50	0x80000440
HW_ICOLL_INTERRUPT50_CLR	0x80000448
HW_ICOLL_INTERRUPT50_SET	0x80000444
HW_ICOLL_INTERRUPT50_TOG	0x8000044C
HW_ICOLL_INTERRUPT51	0x80000450
HW_ICOLL_INTERRUPT51_CLR	0x80000458
HW_ICOLL_INTERRUPT51_SET	0x80000454
HW_ICOLL_INTERRUPT51_TOG	0x8000045C
HW_ICOLL_INTERRUPT52	0x80000460
HW_ICOLL_INTERRUPT52_CLR	0x80000468
HW_ICOLL_INTERRUPT52_SET	0x80000464
HW_ICOLL_INTERRUPT52_TOG	0x8000046C
HW_ICOLL_INTERRUPT53	0x80000470
HW_ICOLL_INTERRUPT53_CLR	0x80000478
HW_ICOLL_INTERRUPT53_SET	0x80000474
HW_ICOLL_INTERRUPT53_TOG	0x8000047C
HW_ICOLL_INTERRUPT54	0x80000480
HW_ICOLL_INTERRUPT54_CLR	0x80000488
HW_ICOLL_INTERRUPT54_SET	0x80000484
HW_ICOLL_INTERRUPT54_TOG	0x8000048C
HW_ICOLL_INTERRUPT55	0x80000490
HW_ICOLL_INTERRUPT55_CLR	0x80000498
HW_ICOLL_INTERRUPT55_SET	0x80000494
HW_ICOLL_INTERRUPT55_TOG	0x8000049C
HW_ICOLL_INTERRUPT56	0x800004A0
HW_ICOLL_INTERRUPT56_CLR	0x800004A8

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ICOLL_INTERRUPT56_SET	0x800004A4
HW_ICOLL_INTERRUPT56_TOG	0x800004AC
HW_ICOLL_INTERRUPT57	0x800004B0
HW_ICOLL_INTERRUPT57_CLR	0x800004B8
HW_ICOLL_INTERRUPT57_SET	0x800004B4
HW_ICOLL_INTERRUPT57_TOG	0x800004BC
HW_ICOLL_INTERRUPT58	0x800004C0
HW_ICOLL_INTERRUPT58_CLR	0x800004C8
HW_ICOLL_INTERRUPT58_SET	0x800004C4
HW_ICOLL_INTERRUPT58_TOG	0x800004CC
HW_ICOLL_INTERRUPT59	0x800004D0
HW_ICOLL_INTERRUPT59_CLR	0x800004D8
HW_ICOLL_INTERRUPT59_SET	0x800004D4
HW_ICOLL_INTERRUPT59_TOG	0x800004DC
HW_ICOLL_INTERRUPT6	0x80000180
HW_ICOLL_INTERRUPT6_CLR	0x80000188
HW_ICOLL_INTERRUPT6_SET	0x80000184
HW_ICOLL_INTERRUPT6_TOG	0x8000018C
HW_ICOLL_INTERRUPT60	0x800004E0
HW_ICOLL_INTERRUPT60_CLR	0x800004E8
HW_ICOLL_INTERRUPT60_SET	0x800004E4
HW_ICOLL_INTERRUPT60_TOG	0x800004EC
HW_ICOLL_INTERRUPT61	0x800004F0
HW_ICOLL_INTERRUPT61_CLR	0x800004F8
HW_ICOLL_INTERRUPT61_SET	0x800004F4
HW_ICOLL_INTERRUPT61_TOG	0x800004FC
HW_ICOLL_INTERRUPT62	0x80000500
HW_ICOLL_INTERRUPT62_CLR	0x80000508
HW_ICOLL_INTERRUPT62_SET	0x80000504
HW_ICOLL_INTERRUPT62_TOG	0x8000050C
HW_ICOLL_INTERRUPT63	0x80000510
HW_ICOLL_INTERRUPT63_CLR	0x80000518
HW_ICOLL_INTERRUPT63_SET	0x80000514

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ICOLL_INTERRUPT63_TOG	0x8000051C
HW_ICOLL_INTERRUPT64	0x80000520
HW_ICOLL_INTERRUPT64_CLR	0x80000528
HW_ICOLL_INTERRUPT64_SET	0x80000524
HW_ICOLL_INTERRUPT64_TOG	0x8000052C
HW_ICOLL_INTERRUPT65	0x80000530
HW_ICOLL_INTERRUPT65_CLR	0x80000538
HW_ICOLL_INTERRUPT65_SET	0x80000534
HW_ICOLL_INTERRUPT65_TOG	0x8000053C
HW_ICOLL_INTERRUPT66	0x80000540
HW_ICOLL_INTERRUPT66_CLR	0x80000548
HW_ICOLL_INTERRUPT66_SET	0x80000544
HW_ICOLL_INTERRUPT66_TOG	0x8000054C
HW_ICOLL_INTERRUPT67	0x80000550
HW_ICOLL_INTERRUPT67_CLR	0x80000558
HW_ICOLL_INTERRUPT67_SET	0x80000554
HW_ICOLL_INTERRUPT67_TOG	0x8000055C
HW_ICOLL_INTERRUPT68	0x80000560
HW_ICOLL_INTERRUPT68_CLR	0x80000568
HW_ICOLL_INTERRUPT68_SET	0x80000564
HW_ICOLL_INTERRUPT68_TOG	0x8000056C
HW_ICOLL_INTERRUPT69	0x80000570
HW_ICOLL_INTERRUPT69_CLR	0x80000578
HW_ICOLL_INTERRUPT69_SET	0x80000574
HW_ICOLL_INTERRUPT69_TOG	0x8000057C
HW_ICOLL_INTERRUPT7	0x80000190
HW_ICOLL_INTERRUPT7_CLR	0x80000198
HW_ICOLL_INTERRUPT7_SET	0x80000194
HW_ICOLL_INTERRUPT7_TOG	0x8000019C
HW_ICOLL_INTERRUPT70	0x80000580
HW_ICOLL_INTERRUPT70_CLR	0x80000588
HW_ICOLL_INTERRUPT70_SET	0x80000584
HW_ICOLL_INTERRUPT70_TOG	0x8000058C

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ICOLL_INTERRUPT71	0x80000590
HW_ICOLL_INTERRUPT71_CLR	0x80000598
HW_ICOLL_INTERRUPT71_SET	0x80000594
HW_ICOLL_INTERRUPT71_TOG	0x8000059C
HW_ICOLL_INTERRUPT72	0x800005A0
HW_ICOLL_INTERRUPT72_CLR	0x800005A8
HW_ICOLL_INTERRUPT72_SET	0x800005A4
HW_ICOLL_INTERRUPT72_TOG	0x800005AC
HW_ICOLL_INTERRUPT73	0x800005B0
HW_ICOLL_INTERRUPT73_CLR	0x800005B8
HW_ICOLL_INTERRUPT73_SET	0x800005B4
HW_ICOLL_INTERRUPT73_TOG	0x800005BC
HW_ICOLL_INTERRUPT74	0x800005C0
HW_ICOLL_INTERRUPT74_CLR	0x800005C8
HW_ICOLL_INTERRUPT74_SET	0x800005C4
HW_ICOLL_INTERRUPT74_TOG	0x800005CC
HW_ICOLL_INTERRUPT75	0x800005D0
HW_ICOLL_INTERRUPT75_CLR	0x800005D8
HW_ICOLL_INTERRUPT75_SET	0x800005D4
HW_ICOLL_INTERRUPT75_TOG	0x800005DC
HW_ICOLL_INTERRUPT76	0x800005E0
HW_ICOLL_INTERRUPT76_CLR	0x800005E8
HW_ICOLL_INTERRUPT76_SET	0x800005E4
HW_ICOLL_INTERRUPT76_TOG	0x800005EC
HW_ICOLL_INTERRUPT77	0x800005F0
HW_ICOLL_INTERRUPT77_CLR	0x800005F8
HW_ICOLL_INTERRUPT77_SET	0x800005F4
HW_ICOLL_INTERRUPT77_TOG	0x800005FC
HW_ICOLL_INTERRUPT78	0x80000600
HW_ICOLL_INTERRUPT78_CLR	0x80000608
HW_ICOLL_INTERRUPT78_SET	0x80000604
HW_ICOLL_INTERRUPT78_TOG	0x8000060C
HW_ICOLL_INTERRUPT79	0x80000610

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ICOLL_INTERRUPT79_CLR	0x80000618
HW_ICOLL_INTERRUPT79_SET	0x80000614
HW_ICOLL_INTERRUPT79_TOG	0x8000061C
HW_ICOLL_INTERRUPT8	0x800001A0
HW_ICOLL_INTERRUPT8_CLR	0x800001A8
HW_ICOLL_INTERRUPT8_SET	0x800001A4
HW_ICOLL_INTERRUPT8_TOG	0x800001AC
HW_ICOLL_INTERRUPT80	0x80000620
HW_ICOLL_INTERRUPT80_CLR	0x80000628
HW_ICOLL_INTERRUPT80_SET	0x80000624
HW_ICOLL_INTERRUPT80_TOG	0x8000062C
HW_ICOLL_INTERRUPT81	0x80000630
HW_ICOLL_INTERRUPT81_CLR	0x80000638
HW_ICOLL_INTERRUPT81_SET	0x80000634
HW_ICOLL_INTERRUPT81_TOG	0x8000063C
HW_ICOLL_INTERRUPT82	0x80000640
HW_ICOLL_INTERRUPT82_CLR	0x80000648
HW_ICOLL_INTERRUPT82_SET	0x80000644
HW_ICOLL_INTERRUPT82_TOG	0x8000064C
HW_ICOLL_INTERRUPT83	0x80000650
HW_ICOLL_INTERRUPT83_CLR	0x80000658
HW_ICOLL_INTERRUPT83_SET	0x80000654
HW_ICOLL_INTERRUPT83_TOG	0x8000065C
HW_ICOLL_INTERRUPT84	0x80000660
HW_ICOLL_INTERRUPT84_CLR	0x80000668
HW_ICOLL_INTERRUPT84_SET	0x80000664
HW_ICOLL_INTERRUPT84_TOG	0x8000066C
HW_ICOLL_INTERRUPT85	0x80000670
HW_ICOLL_INTERRUPT85_CLR	0x80000678
HW_ICOLL_INTERRUPT85_SET	0x80000674
HW_ICOLL_INTERRUPT85_TOG	0x8000067C
HW_ICOLL_INTERRUPT86	0x80000680
HW_ICOLL_INTERRUPT86_CLR	0x80000688

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ICOLL_INTERRUPT86_SET	0x80000684
HW_ICOLL_INTERRUPT86_TOG	0x8000068C
HW_ICOLL_INTERRUPT87	0x80000690
HW_ICOLL_INTERRUPT87_CLR	0x80000698
HW_ICOLL_INTERRUPT87_SET	0x80000694
HW_ICOLL_INTERRUPT87_TOG	0x8000069C
HW_ICOLL_INTERRUPT88	0x800006A0
HW_ICOLL_INTERRUPT88_CLR	0x800006A8
HW_ICOLL_INTERRUPT88_SET	0x800006A4
HW_ICOLL_INTERRUPT88_TOG	0x800006AC
HW_ICOLL_INTERRUPT89	0x800006B0
HW_ICOLL_INTERRUPT89_CLR	0x800006B8
HW_ICOLL_INTERRUPT89_SET	0x800006B4
HW_ICOLL_INTERRUPT89_TOG	0x800006BC
HW_ICOLL_INTERRUPT9	0x800001B0
HW_ICOLL_INTERRUPT9_CLR	0x800001B8
HW_ICOLL_INTERRUPT9_SET	0x800001B4
HW_ICOLL_INTERRUPT9_TOG	0x800001BC
HW_ICOLL_INTERRUPT90	0x800006C0
HW_ICOLL_INTERRUPT90_CLR	0x800006C8
HW_ICOLL_INTERRUPT90_SET	0x800006C4
HW_ICOLL_INTERRUPT90_TOG	0x800006CC
HW_ICOLL_INTERRUPT91	0x800006D0
HW_ICOLL_INTERRUPT91_CLR	0x800006D8
HW_ICOLL_INTERRUPT91_SET	0x800006D4
HW_ICOLL_INTERRUPT91_TOG	0x800006DC
HW_ICOLL_INTERRUPT92	0x800006E0
HW_ICOLL_INTERRUPT92_CLR	0x800006E8
HW_ICOLL_INTERRUPT92_SET	0x800006E4
HW_ICOLL_INTERRUPT92_TOG	0x800006EC
HW_ICOLL_INTERRUPT93	0x800006F0
HW_ICOLL_INTERRUPT93_CLR	0x800006F8
HW_ICOLL_INTERRUPT93_SET	0x800006F4

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ICOLL_INTERRUPT93_TOG	0x800006FC
HW_ICOLL_INTERRUPT94	0x80000700
HW_ICOLL_INTERRUPT94_CLR	0x80000708
HW_ICOLL_INTERRUPT94_SET	0x80000704
HW_ICOLL_INTERRUPT94_TOG	0x8000070C
HW_ICOLL_INTERRUPT95	0x80000710
HW_ICOLL_INTERRUPT95_CLR	0x80000718
HW_ICOLL_INTERRUPT95_SET	0x80000714
HW_ICOLL_INTERRUPT95_TOG	0x8000071C
HW_ICOLL_INTERRUPT96	0x80000720
HW_ICOLL_INTERRUPT96_CLR	0x80000728
HW_ICOLL_INTERRUPT96_SET	0x80000724
HW_ICOLL_INTERRUPT96_TOG	0x8000072C
HW_ICOLL_INTERRUPT97	0x80000730
HW_ICOLL_INTERRUPT97_CLR	0x80000738
HW_ICOLL_INTERRUPT97_SET	0x80000734
HW_ICOLL_INTERRUPT97_TOG	0x8000073C
HW_ICOLL_INTERRUPT98	0x80000740
HW_ICOLL_INTERRUPT98_CLR	0x80000748
HW_ICOLL_INTERRUPT98_SET	0x80000744
HW_ICOLL_INTERRUPT98_TOG	0x8000074C
HW_ICOLL_INTERRUPT99	0x80000750
HW_ICOLL_INTERRUPT99_CLR	0x80000758
HW_ICOLL_INTERRUPT99_SET	0x80000754
HW_ICOLL_INTERRUPT99_TOG	0x8000075C
HW_ICOLL_LEVELACK	0x80000010
HW_ICOLL_RAW0	0x800000A0
HW_ICOLL_RAW0_CLR	0x800000A8
HW_ICOLL_RAW0_SET	0x800000A4
HW_ICOLL_RAW0_TOG	0x800000AC
HW_ICOLL_RAW1	0x800000B0
HW_ICOLL_RAW1_CLR	0x800000B8
HW_ICOLL_RAW1_SET	0x800000B4

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_ICOLL_RAW1_TOG	0x800000BC
HW_ICOLL_RAW2	0x800000C0
HW_ICOLL_RAW2_CLR	0x800000C8
HW_ICOLL_RAW2_SET	0x800000C4
HW_ICOLL_RAW2_TOG	0x800000CC
HW_ICOLL_RAW3	0x800000D0
HW_ICOLL_RAW3_CLR	0x800000D8
HW_ICOLL_RAW3_SET	0x800000D4
HW_ICOLL_RAW3_TOG	0x800000DC
HW_ICOLL_STAT	0x80000070
HW_ICOLL_VBASE	0x80000040
HW_ICOLL_VBASE_CLR	0x80000048
HW_ICOLL_VBASE_SET	0x80000044
HW_ICOLL_VBASE_TOG	0x8000004C
HW_ICOLL_VECTOR	0x80000000
HW_ICOLL_VECTOR_CLR	0x80000008
HW_ICOLL_VECTOR_SET	0x80000004
HW_ICOLL_VECTOR_TOG	0x8000000C
HW_ICOLL_VERSION	0x800011E0
HW_IR_CTRL	0x80078000
HW_IR_CTRL_CLR	0x80078008
HW_IR_CTRL_SET	0x80078004
HW_IR_CTRL_TOG	0x8007800C
HW_IR_DATA	0x80078050
HW_IR_DBGCTRL	0x80078030
HW_IR_DBGCTRL_CLR	0x80078038
HW_IR_DBGCTRL_SET	0x80078034
HW_IR_DBGCTRL_TOG	0x8007803C
HW_IR_DEBUG	0x80078090
HW_IR_INTR	0x80078040
HW_IR_INTR_CLR	0x80078048
HW_IR_INTR_SET	0x80078044
HW_IR_INTR_TOG	0x8007804C

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_IR_RXDMA	0x80078020
HW_IR_RXDMA_CLR	0x80078028
HW_IR_RXDMA_SET	0x80078024
HW_IR_RXDMA_TOG	0x8007802C
HW_IR_SI_READ	0x80078080
HW_IR_STAT	0x80078060
HW_IR_TCCTRL	0x80078070
HW_IR_TCCTRL_CLR	0x80078078
HW_IR_TCCTRL_SET	0x80078074
HW_IR_TCCTRL_TOG	0x8007807C
HW_IR_TXDMA	0x80078010
HW_IR_TXDMA_CLR	0x80078018
HW_IR_TXDMA_SET	0x80078014
HW_IR_TXDMA_TOG	0x8007801C
HW_IR_VERSION	0x800780a0
HW_LCDIF_BM_ERROR_STAT	0x800301c0
HW_LCDIF_CSC_COEFF0	0x80030110
HW_LCDIF_CSC_COEFF1	0x80030120
HW_LCDIF_CSC_COEFF2	0x80030130
HW_LCDIF_CSC_COEFF3	0x80030140
HW_LCDIF_CSC_COEFF4	0x80030150
HW_LCDIF_CSC_LIMIT	0x80030170
HW_LCDIF_CSC_OFFSET	0x80030160
HW_LCDIF_CTRL	0x80030000
HW_LCDIF_CTRL_CLR	0x80030008
HW_LCDIF_CTRL_SET	0x80030004
HW_LCDIF_CTRL_TOG	0x8003000C
HW_LCDIF_CTRL1	0x80030010
HW_LCDIF_CTRL1_CLR	0x80030018
HW_LCDIF_CTRL1_SET	0x80030014
HW_LCDIF_CTRL1_TOG	0x8003001C
HW_LCDIF_CUR_BUF	0x80030030
HW_LCDIF_DATA	0x800301b0

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_LCDIF_DEBUG0	0x800301f0
HW_LCDIF_DEBUG1	0x80030200
HW_LCDIF_DVICTRL0	0x800300c0
HW_LCDIF_DVICTRL1	0x800300d0
HW_LCDIF_DVICTRL2	0x800300e0
HW_LCDIF_DVICTRL3	0x800300f0
HW_LCDIF_DVICTRL4	0x80030100
HW_LCDIF_NEXT_BUF	0x80030040
HW_LCDIF_PAGETABLE	0x80030050
HW_LCDIF_PIN_SHARING_CTRL0	0x80030180
HW_LCDIF_PIN_SHARING_CTRL0_CLR	0x80030188
HW_LCDIF_PIN_SHARING_CTRL0_SET	0x80030184
HW_LCDIF_PIN_SHARING_CTRL0_TOG	0x8003018C
HW_LCDIF_PIN_SHARING_CTRL1	0x80030190
HW_LCDIF_PIN_SHARING_CTRL2	0x800301a0
HW_LCDIF_STAT	0x800301d0
HW_LCDIF_TIMING	0x80030060
HW_LCDIF_TRANSFER_COUNT	0x80030020
HW_LCDIF_VDCTRL0	0x80030070
HW_LCDIF_VDCTRL0_CLR	0x80030078
HW_LCDIF_VDCTRL0_SET	0x80030074
HW_LCDIF_VDCTRL0_TOG	0x8003007C
HW_LCDIF_VDCTRL1	0x80030080
HW_LCDIF_VDCTRL2	0x80030090
HW_LCDIF_VDCTRL3	0x800300a0
HW_LCDIF_VDCTRL4	0x800300b0
HW_LCDIF_VERSION	0x800301e0
HW_LRADC_CH0	0x80050050
HW_LRADC_CH0_CLR	0x80050058
HW_LRADC_CH0_SET	0x80050054
HW_LRADC_CH0_TOG	0x8005005C
HW_LRADC_CH1	0x80050060
HW_LRADC_CH1_CLR	0x80050068

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_LRADC_CH1_SET	0x80050064
HW_LRADC_CH1_TOG	0x8005006C
HW_LRADC_CH2	0x80050070
HW_LRADC_CH2_CLR	0x80050078
HW_LRADC_CH2_SET	0x80050074
HW_LRADC_CH2_TOG	0x8005007C
HW_LRADC_CH3	0x80050080
HW_LRADC_CH3_CLR	0x80050088
HW_LRADC_CH3_SET	0x80050084
HW_LRADC_CH3_TOG	0x8005008C
HW_LRADC_CH4	0x80050090
HW_LRADC_CH4_CLR	0x80050098
HW_LRADC_CH4_SET	0x80050094
HW_LRADC_CH4_TOG	0x8005009C
HW_LRADC_CH5	0x800500A0
HW_LRADC_CH5_CLR	0x800500A8
HW_LRADC_CH5_SET	0x800500A4
HW_LRADC_CH5_TOG	0x800500AC
HW_LRADC_CH6	0x800500B0
HW_LRADC_CH6_CLR	0x800500B8
HW_LRADC_CH6_SET	0x800500B4
HW_LRADC_CH6_TOG	0x800500BC
HW_LRADC_CH7	0x800500C0
HW_LRADC_CH7_CLR	0x800500C8
HW_LRADC_CH7_SET	0x800500C4
HW_LRADC_CH7_TOG	0x800500CC
HW_LRADC_CONVERSION	0x80050130
HW_LRADC_CONVERSION_CLR	0x80050138
HW_LRADC_CONVERSION_SET	0x80050134
HW_LRADC_CONVERSION_TOG	0x8005013C
HW_LRADC_CTRL0	0x80050000
HW_LRADC_CTRL0_CLR	0x80050008
HW_LRADC_CTRL0_SET	0x80050004

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_LRADC_CTRL0_TOG	0x8005000C
HW_LRADC_CTRL1	0x80050010
HW_LRADC_CTRL1_CLR	0x80050018
HW_LRADC_CTRL1_SET	0x80050014
HW_LRADC_CTRL1_TOG	0x8005001C
HW_LRADC_CTRL2	0x80050020
HW_LRADC_CTRL2_CLR	0x80050028
HW_LRADC_CTRL2_SET	0x80050024
HW_LRADC_CTRL2_TOG	0x8005002C
HW_LRADC_CTRL3	0x80050030
HW_LRADC_CTRL3_CLR	0x80050038
HW_LRADC_CTRL3_SET	0x80050034
HW_LRADC_CTRL3_TOG	0x8005003C
HW_LRADC_CTRL4	0x80050140
HW_LRADC_CTRL4_CLR	0x80050148
HW_LRADC_CTRL4_SET	0x80050144
HW_LRADC_CTRL4_TOG	0x8005014C
HW_LRADC_DEBUG0	0x80050110
HW_LRADC_DEBUG0_CLR	0x80050118
HW_LRADC_DEBUG0_SET	0x80050114
HW_LRADC_DEBUG0_TOG	0x8005011C
HW_LRADC_DEBUG1	0x80050120
HW_LRADC_DEBUG1_CLR	0x80050128
HW_LRADC_DEBUG1_SET	0x80050124
HW_LRADC_DEBUG1_TOG	0x8005012C
HW_LRADC_DELAY0	0x800500D0
HW_LRADC_DELAY0_CLR	0x800500D8
HW_LRADC_DELAY0_SET	0x800500D4
HW_LRADC_DELAY0_TOG	0x800500DC
HW_LRADC_DELAY1	0x800500E0
HW_LRADC_DELAY1_CLR	0x800500E8
HW_LRADC_DELAY1_SET	0x800500E4
HW_LRADC_DELAY1_TOG	0x800500EC

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_LRADC_DELAY2	0x800500F0
HW_LRADC_DELAY2_CLR	0x800500F8
HW_LRADC_DELAY2_SET	0x800500F4
HW_LRADC_DELAY2_TOG	0x800500FC
HW_LRADC_DELAY3	0x80050100
HW_LRADC_DELAY3_CLR	0x80050108
HW_LRADC_DELAY3_SET	0x80050104
HW_LRADC_DELAY3_TOG	0x8005010C
HW_LRADC_STATUS	0x80050040
HW_LRADC_STATUS_CLR	0x80050048
HW_LRADC_STATUS_SET	0x80050044
HW_LRADC_STATUS_TOG	0x8005004C
HW_LRADC_VERSION	0x80050150
HW_OCOTP_CRYPTO0	0x8002C060
HW_OCOTP_CRYPTO1	0x8002C070
HW_OCOTP_CRYPTO2	0x8002C080
HW_OCOTP_CRYPTO3	0x8002C090
HW_OCOTP_CTRL	0x8002C000
HW_OCOTP_CTRL_CLR	0x8002C008
HW_OCOTP_CTRL_SET	0x8002C004
HW_OCOTP_CTRL_TOG	0x8002C00C
HW_OCOTP_CUST0	0x8002C020
HW_OCOTP_CUST1	0x8002C030
HW_OCOTP_CUST2	0x8002C040
HW_OCOTP_CUST3	0x8002C050
HW_OCOTP_CUSTCAP	0x8002C110
HW_OCOTP_DATA	0x8002C010
HW_OCOTP_HWCAP0	0x8002C0A0
HW_OCOTP_HWCAP1	0x8002C0B0
HW_OCOTP_HWCAP2	0x8002C0C0
HW_OCOTP_HWCAP3	0x8002C0D0
HW_OCOTP_HWCAP4	0x8002C0E0
HW_OCOTP_HWCAP5	0x8002C0F0

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_OCOTP_LOCK	0x8002C120
HW_OCOTP_OPS0	0x8002C130
HW_OCOTP_OPS1	0x8002C140
HW_OCOTP_OPS2	0x8002C150
HW_OCOTP_OPS3	0x8002C160
HW_OCOTP_ROM0	0x8002C1A0
HW_OCOTP_ROM1	0x8002C1B0
HW_OCOTP_ROM2	0x8002C1C0
HW_OCOTP_ROM3	0x8002C1D0
HW_OCOTP_ROM4	0x8002C1E0
HW_OCOTP_ROM5	0x8002C1F0
HW_OCOTP_ROM6	0x8002C200
HW_OCOTP_ROM7	0x8002C210
HW_OCOTP_SWCAP	0x8002C100
HW_OCOTP_UN0	0x8002C170
HW_OCOTP_UN1	0x8002C180
HW_OCOTP_UN2	0x8002C190
HW_OCOTP_VERSION	0x8002C220
HW_PINCTRL_CTRL	0x80018000
HW_PINCTRL_CTRL_CLR	0x80018008
HW_PINCTRL_CTRL_SET	0x80018004
HW_PINCTRL_CTRL_TOG	0x8001800C
HW_PINCTRL_DIN0	0x80018600
HW_PINCTRL_DIN0_CLR	0x80018608
HW_PINCTRL_DIN0_SET	0x80018604
HW_PINCTRL_DIN0_TOG	0x8001860C
HW_PINCTRL_DIN1	0x80018610
HW_PINCTRL_DIN1_CLR	0x80018618
HW_PINCTRL_DIN1_SET	0x80018614
HW_PINCTRL_DIN1_TOG	0x8001861C
HW_PINCTRL_DIN2	0x80018620
HW_PINCTRL_DIN2_CLR	0x80018628
HW_PINCTRL_DIN2_SET	0x80018624

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_PINCTRL_DIN2_TOG	0x8001862C
HW_PINCTRL_DOE0	0x80018700
HW_PINCTRL_DOE0_CLR	0x80018708
HW_PINCTRL_DOE0_SET	0x80018704
HW_PINCTRL_DOE0_TOG	0x8001870C
HW_PINCTRL_DOE1	0x80018710
HW_PINCTRL_DOE1_CLR	0x80018718
HW_PINCTRL_DOE1_SET	0x80018714
HW_PINCTRL_DOE1_TOG	0x8001871C
HW_PINCTRL_DOE2	0x80018720
HW_PINCTRL_DOE2_CLR	0x80018728
HW_PINCTRL_DOE2_SET	0x80018724
HW_PINCTRL_DOE2_TOG	0x8001872C
HW_PINCTRL_DOUT0	0x80018500
HW_PINCTRL_DOUT0_CLR	0x80018508
HW_PINCTRL_DOUT0_SET	0x80018504
HW_PINCTRL_DOUT0_TOG	0x8001850C
HW_PINCTRL_DOUT1	0x80018510
HW_PINCTRL_DOUT1_CLR	0x80018518
HW_PINCTRL_DOUT1_SET	0x80018514
HW_PINCTRL_DOUT1_TOG	0x8001851C
HW_PINCTRL_DOUT2	0x80018520
HW_PINCTRL_DOUT2_CLR	0x80018528
HW_PINCTRL_DOUT2_SET	0x80018524
HW_PINCTRL_DOUT2_TOG	0x8001852C
HW_PINCTRL_DRIVE0	0x80018200
HW_PINCTRL_DRIVE0_CLR	0x80018208
HW_PINCTRL_DRIVE0_SET	0x80018204
HW_PINCTRL_DRIVE0_TOG	0x8001820C
HW_PINCTRL_DRIVE1	0x80018210
HW_PINCTRL_DRIVE1_CLR	0x80018218
HW_PINCTRL_DRIVE1_SET	0x80018214
HW_PINCTRL_DRIVE1_TOG	0x8001821C

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_PINCTRL_DRIVE10	0x800182a0
HW_PINCTRL_DRIVE10_CLR	0x800182a8
HW_PINCTRL_DRIVE10_SET	0x800182a4
HW_PINCTRL_DRIVE10_TOG	0x800182aC
HW_PINCTRL_DRIVE11	0x800182b0
HW_PINCTRL_DRIVE11_CLR	0x800182b8
HW_PINCTRL_DRIVE11_SET	0x800182b4
HW_PINCTRL_DRIVE11_TOG	0x800182bC
HW_PINCTRL_DRIVE12	0x800182c0
HW_PINCTRL_DRIVE12_CLR	0x800182c8
HW_PINCTRL_DRIVE12_SET	0x800182c4
HW_PINCTRL_DRIVE12_TOG	0x800182cC
HW_PINCTRL_DRIVE13	0x800182d0
HW_PINCTRL_DRIVE13_CLR	0x800182d8
HW_PINCTRL_DRIVE13_SET	0x800182d4
HW_PINCTRL_DRIVE13_TOG	0x800182dC
HW_PINCTRL_DRIVE14	0x800182e0
HW_PINCTRL_DRIVE14_CLR	0x800182e8
HW_PINCTRL_DRIVE14_SET	0x800182e4
HW_PINCTRL_DRIVE14_TOG	0x800182eC
HW_PINCTRL_DRIVE2	0x80018220
HW_PINCTRL_DRIVE2_CLR	0x80018228
HW_PINCTRL_DRIVE2_SET	0x80018224
HW_PINCTRL_DRIVE2_TOG	0x8001822C
HW_PINCTRL_DRIVE3	0x80018230
HW_PINCTRL_DRIVE3_CLR	0x80018238
HW_PINCTRL_DRIVE3_SET	0x80018234
HW_PINCTRL_DRIVE3_TOG	0x8001823C
HW_PINCTRL_DRIVE4	0x80018240
HW_PINCTRL_DRIVE4_CLR	0x80018248
HW_PINCTRL_DRIVE4_SET	0x80018244
HW_PINCTRL_DRIVE4_TOG	0x8001824C
HW_PINCTRL_DRIVE5	0x80018250

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_PINCTRL_DRIVE5_CLR	0x80018258
HW_PINCTRL_DRIVE5_SET	0x80018254
HW_PINCTRL_DRIVE5_TOG	0x8001825C
HW_PINCTRL_DRIVE6	0x80018260
HW_PINCTRL_DRIVE6_CLR	0x80018268
HW_PINCTRL_DRIVE6_SET	0x80018264
HW_PINCTRL_DRIVE6_TOG	0x8001826C
HW_PINCTRL_DRIVE7	0x80018270
HW_PINCTRL_DRIVE7_CLR	0x80018278
HW_PINCTRL_DRIVE7_SET	0x80018274
HW_PINCTRL_DRIVE7_TOG	0x8001827C
HW_PINCTRL_DRIVE8	0x80018280
HW_PINCTRL_DRIVE8_CLR	0x80018288
HW_PINCTRL_DRIVE8_SET	0x80018284
HW_PINCTRL_DRIVE8_TOG	0x8001828C
HW_PINCTRL_DRIVE9	0x80018290
HW_PINCTRL_DRIVE9_CLR	0x80018298
HW_PINCTRL_DRIVE9_SET	0x80018294
HW_PINCTRL_DRIVE9_TOG	0x8001829C
HW_PINCTRL_IRQEN0	0x80018900
HW_PINCTRL_IRQEN0_CLR	0x80018908
HW_PINCTRL_IRQEN0_SET	0x80018904
HW_PINCTRL_IRQEN0_TOG	0x8001890C
HW_PINCTRL_IRQEN1	0x80018910
HW_PINCTRL_IRQEN1_CLR	0x80018918
HW_PINCTRL_IRQEN1_SET	0x80018914
HW_PINCTRL_IRQEN1_TOG	0x8001891C
HW_PINCTRL_IRQEN2	0x80018920
HW_PINCTRL_IRQEN2_CLR	0x80018928
HW_PINCTRL_IRQEN2_SET	0x80018924
HW_PINCTRL_IRQEN2_TOG	0x8001892C
HW_PINCTRL_IRQLEVEL0	0x80018a00
HW_PINCTRL_IRQLEVEL0_CLR	0x80018a08

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_PINCTRL_IRQLEVEL0_SET	0x80018a04
HW_PINCTRL_IRQLEVEL0_TOG	0x80018a0C
HW_PINCTRL_IRQLEVEL1	0x80018a10
HW_PINCTRL_IRQLEVEL1_CLR	0x80018a18
HW_PINCTRL_IRQLEVEL1_SET	0x80018a14
HW_PINCTRL_IRQLEVEL1_TOG	0x80018a1C
HW_PINCTRL_IRQLEVEL2	0x80018a20
HW_PINCTRL_IRQLEVEL2_CLR	0x80018a28
HW_PINCTRL_IRQLEVEL2_SET	0x80018a24
HW_PINCTRL_IRQLEVEL2_TOG	0x80018a2C
HW_PINCTRL_IRQPOL0	0x80018b00
HW_PINCTRL_IRQPOL0_CLR	0x80018b08
HW_PINCTRL_IRQPOL0_SET	0x80018b04
HW_PINCTRL_IRQPOL0_TOG	0x80018b0C
HW_PINCTRL_IRQPOL1	0x80018b10
HW_PINCTRL_IRQPOL1_CLR	0x80018b18
HW_PINCTRL_IRQPOL1_SET	0x80018b14
HW_PINCTRL_IRQPOL1_TOG	0x80018b1C
HW_PINCTRL_IRQPOL2	0x80018b20
HW_PINCTRL_IRQPOL2_CLR	0x80018b28
HW_PINCTRL_IRQPOL2_SET	0x80018b24
HW_PINCTRL_IRQPOL2_TOG	0x80018b2C
HW_PINCTRL_IRQSTAT0	0x80018c00
HW_PINCTRL_IRQSTAT0_CLR	0x80018c08
HW_PINCTRL_IRQSTAT0_SET	0x80018c04
HW_PINCTRL_IRQSTAT0_TOG	0x80018c0C
HW_PINCTRL_IRQSTAT1	0x80018c10
HW_PINCTRL_IRQSTAT1_CLR	0x80018c18
HW_PINCTRL_IRQSTAT1_SET	0x80018c14
HW_PINCTRL_IRQSTAT1_TOG	0x80018c1C
HW_PINCTRL_IRQSTAT2	0x80018c20
HW_PINCTRL_IRQSTAT2_CLR	0x80018c28
HW_PINCTRL_IRQSTAT2_SET	0x80018c24

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_PINCTRL_IRQSTAT2_TOG	0x80018c2C
HW_PINCTRL_MUXSEL0	0x80018100
HW_PINCTRL_MUXSEL0_CLR	0x80018108
HW_PINCTRL_MUXSEL0_SET	0x80018104
HW_PINCTRL_MUXSEL0_TOG	0x8001810C
HW_PINCTRL_MUXSEL1	0x80018110
HW_PINCTRL_MUXSEL1_CLR	0x80018118
HW_PINCTRL_MUXSEL1_SET	0x80018114
HW_PINCTRL_MUXSEL1_TOG	0x8001811C
HW_PINCTRL_MUXSEL2	0x80018120
HW_PINCTRL_MUXSEL2_CLR	0x80018128
HW_PINCTRL_MUXSEL2_SET	0x80018124
HW_PINCTRL_MUXSEL2_TOG	0x8001812C
HW_PINCTRL_MUXSEL3	0x80018130
HW_PINCTRL_MUXSEL3_CLR	0x80018138
HW_PINCTRL_MUXSEL3_SET	0x80018134
HW_PINCTRL_MUXSEL3_TOG	0x8001813C
HW_PINCTRL_MUXSEL4	0x80018140
HW_PINCTRL_MUXSEL4_CLR	0x80018148
HW_PINCTRL_MUXSEL4_SET	0x80018144
HW_PINCTRL_MUXSEL4_TOG	0x8001814C
HW_PINCTRL_MUXSEL5	0x80018150
HW_PINCTRL_MUXSEL5_CLR	0x80018158
HW_PINCTRL_MUXSEL5_SET	0x80018154
HW_PINCTRL_MUXSEL5_TOG	0x8001815C
HW_PINCTRL_MUXSEL6	0x80018160
HW_PINCTRL_MUXSEL6_CLR	0x80018168
HW_PINCTRL_MUXSEL6_SET	0x80018164
HW_PINCTRL_MUXSEL6_TOG	0x8001816C
HW_PINCTRL_MUXSEL7	0x80018170
HW_PINCTRL_MUXSEL7_CLR	0x80018178
HW_PINCTRL_MUXSEL7_SET	0x80018174
HW_PINCTRL_MUXSEL7_TOG	0x8001817C

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_PINCTRL_PIN2IRQ0	0x80018800
HW_PINCTRL_PIN2IRQ0_CLR	0x80018808
HW_PINCTRL_PIN2IRQ0_SET	0x80018804
HW_PINCTRL_PIN2IRQ0_TOG	0x8001880C
HW_PINCTRL_PIN2IRQ1	0x80018810
HW_PINCTRL_PIN2IRQ1_CLR	0x80018818
HW_PINCTRL_PIN2IRQ1_SET	0x80018814
HW_PINCTRL_PIN2IRQ1_TOG	0x8001881C
HW_PINCTRL_PIN2IRQ2	0x80018820
HW_PINCTRL_PIN2IRQ2_CLR	0x80018828
HW_PINCTRL_PIN2IRQ2_SET	0x80018824
HW_PINCTRL_PIN2IRQ2_TOG	0x8001882C
HW_PINCTRL_PULL0	0x80018400
HW_PINCTRL_PULL0_CLR	0x80018408
HW_PINCTRL_PULL0_SET	0x80018404
HW_PINCTRL_PULL0_TOG	0x8001840C
HW_PINCTRL_PULL1	0x80018410
HW_PINCTRL_PULL1_CLR	0x80018418
HW_PINCTRL_PULL1_SET	0x80018414
HW_PINCTRL_PULL1_TOG	0x8001841C
HW_PINCTRL_PULL2	0x80018420
HW_PINCTRL_PULL2_CLR	0x80018428
HW_PINCTRL_PULL2_SET	0x80018424
HW_PINCTRL_PULL2_TOG	0x8001842C
HW_PINCTRL_PULL3	0x80018430
HW_PINCTRL_PULL3_CLR	0x80018438
HW_PINCTRL_PULL3_SET	0x80018434
HW_PINCTRL_PULL3_TOG	0x8001843C
HW_POWER_5VCTRL	0x80044010
HW_POWER_5VCTRL_CLR	0x80044018
HW_POWER_5VCTRL_SET	0x80044014
HW_POWER_5VCTRL_TOG	0x8004401C
HW_POWER_BATTMONITOR	0x800440E0

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_POWER_CHARGE	0x80044030
HW_POWER_CHARGE_CLR	0x80044038
HW_POWER_CHARGE_SET	0x80044034
HW_POWER_CHARGE_TOG	0x8004403C
HW_POWER_CTRL	0x80044000
HW_POWER_CTRL_CLR	0x80044008
HW_POWER_CTRL_SET	0x80044004
HW_POWER_CTRL_TOG	0x8004400C
HW_POWER_DCDC4P2	0x80044080
HW_POWER_DCLIMITS	0x800440A0
HW_POWER_DEBUG	0x80044110
HW_POWER_DEBUG_CLR	0x80044118
HW_POWER_DEBUG_SET	0x80044114
HW_POWER_DEBUG_TOG	0x8004411C
HW_POWER_LOOPCTRL	0x800440B0
HW_POWER_LOOPCTRL_CLR	0x800440B8
HW_POWER_LOOPCTRL_SET	0x800440B4
HW_POWER_LOOPCTRL_TOG	0x800440BC
HW_POWER_MINPWR	0x80044020
HW_POWER_MINPWR_CLR	0x80044028
HW_POWER_MINPWR_SET	0x80044024
HW_POWER_MINPWR_TOG	0x8004402C
HW_POWER_MISC	0x80044090
HW_POWER_RESET	0x80044100
HW_POWER_RESET_CLR	0x80044108
HW_POWER_RESET_SET	0x80044104
HW_POWER_RESET_TOG	0x8004410C
HW_POWER_SPECIAL	0x80044120
HW_POWER_SPECIAL_CLR	0x80044128
HW_POWER_SPECIAL_SET	0x80044124
HW_POWER_SPECIAL_TOG	0x8004412C
HW_POWER_SPEED	0x800440D0
HW_POWER_SPEED_CLR	0x800440D8

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_POWER_SPEED_SET	0x800440D4
HW_POWER_SPEED_TOG	0x800440DC
HW_POWER_STS	0x800440C0
HW_POWER_VDDACTRL	0x80044050
HW_POWER_VDDDCTRL	0x80044040
HW_POWER_VDDIOCTRL	0x80044060
HW_POWER_VDDMEMCTRL	0x80044070
HW_POWER_VERSION	0x80044130
HW_PWM_ACTIVE0	0x80064010
HW_PWM_ACTIVE0_CLR	0x80064018
HW_PWM_ACTIVE0_SET	0x80064014
HW_PWM_ACTIVE0_TOG	0x8006401C
HW_PWM_ACTIVE1	0x80064030
HW_PWM_ACTIVE1_CLR	0x80064038
HW_PWM_ACTIVE1_SET	0x80064034
HW_PWM_ACTIVE1_TOG	0x8006403C
HW_PWM_ACTIVE2	0x80064050
HW_PWM_ACTIVE2_CLR	0x80064058
HW_PWM_ACTIVE2_SET	0x80064054
HW_PWM_ACTIVE2_TOG	0x8006405C
HW_PWM_ACTIVE3	0x80064070
HW_PWM_ACTIVE3_CLR	0x80064078
HW_PWM_ACTIVE3_SET	0x80064074
HW_PWM_ACTIVE3_TOG	0x8006407C
HW_PWM_ACTIVE4	0x80064090
HW_PWM_ACTIVE4_CLR	0x80064098
HW_PWM_ACTIVE4_SET	0x80064094
HW_PWM_ACTIVE4_TOG	0x8006409C
HW_PWM_CTRL	0x80064000
HW_PWM_CTRL_CLR	0x80064008
HW_PWM_CTRL_SET	0x80064004
HW_PWM_CTRL_TOG	0x8006400C
HW_PWM_PERIOD0	0x80064020

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_PWM_PERIOD0_CLR	0x80064028
HW_PWM_PERIOD0_SET	0x80064024
HW_PWM_PERIOD0_TOG	0x8006402C
HW_PWM_PERIOD1	0x80064040
HW_PWM_PERIOD1_CLR	0x80064048
HW_PWM_PERIOD1_SET	0x80064044
HW_PWM_PERIOD1_TOG	0x8006404C
HW_PWM_PERIOD2	0x80064060
HW_PWM_PERIOD2_CLR	0x80064068
HW_PWM_PERIOD2_SET	0x80064064
HW_PWM_PERIOD2_TOG	0x8006406C
HW_PWM_PERIOD3	0x80064080
HW_PWM_PERIOD3_CLR	0x80064088
HW_PWM_PERIOD3_SET	0x80064084
HW_PWM_PERIOD3_TOG	0x8006408C
HW_PWM_PERIOD4	0x800640A0
HW_PWM_PERIOD4_CLR	0x800640A8
HW_PWM_PERIOD4_SET	0x800640A4
HW_PWM_PERIOD4_TOG	0x800640AC
HW_PWM_VERSION	0x800640b0
HW_PXP_CSCCOEFF0	0x8002A0D0
HW_PXP_CSCCOEFF1	0x8002A0E0
HW_PXP_CSCCOEFF2	0x8002A0F0
HW_PXP_CTRL	0x8002A000
HW_PXP_CTRL_CLR	0x8002A008
HW_PXP_CTRL_SET	0x8002A004
HW_PXP_CTRL_TOG	0x8002A00C
HW_PXP_DEBUG	0x8002A1E0
HW_PXP_DEBUGCTRL	0x8002A1D0
HW_PXP_NEXT	0x8002A100
HW_PXP_NEXT_CLR	0x8002A108
HW_PXP_NEXT_SET	0x8002A104
HW_PXP_NEXT_TOG	0x8002A10C

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_PXP_OL0	0x8002a200
HW_PXP_OL0PARAM	0x8002a220
HW_PXP_OL0PARAM2	0x8002a230
HW_PXP_OL0SIZE	0x8002a210
HW_PXP_OL1	0x8002a240
HW_PXP_OL1PARAM	0x8002a260
HW_PXP_OL1PARAM2	0x8002a270
HW_PXP_OL1SIZE	0x8002a250
HW_PXP_OL2	0x8002a280
HW_PXP_OL2PARAM	0x8002a2a0
HW_PXP_OL2PARAM2	0x8002a2b0
HW_PXP_OL2SIZE	0x8002a290
HW_PXP_OL3	0x8002a2c0
HW_PXP_OL3PARAM	0x8002a2e0
HW_PXP_OL3PARAM2	0x8002a2f0
HW_PXP_OL3SIZE	0x8002a2d0
HW_PXP_OL4	0x8002a300
HW_PXP_OL4PARAM	0x8002a320
HW_PXP_OL4PARAM2	0x8002a330
HW_PXP_OL4SIZE	0x8002a310
HW_PXP_OL5	0x8002a340
HW_PXP_OL5PARAM	0x8002a360
HW_PXP_OL5PARAM2	0x8002a370
HW_PXP_OL5SIZE	0x8002a350
HW_PXP_OL6	0x8002a380
HW_PXP_OL6PARAM	0x8002a3a0
HW_PXP_OL6PARAM2	0x8002a3b0
HW_PXP_OL6SIZE	0x8002a390
HW_PXP_OL7	0x8002a3c0
HW_PXP_OL7PARAM	0x8002a3e0
HW_PXP_OL7PARAM2	0x8002a3f0
HW_PXP_OL7SIZE	0x8002a3d0
HW_PXP_OLCOLORKEYHIGH	0x8002A1B0

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_PXP_OLCOLORKEYLOW	0x8002A1A0
HW_PXP_PAGETABLE	0x8002A170
HW_PXP_RGBBUF	0x8002A020
HW_PXP_RGBBUF2	0x8002A030
HW_PXP_RGBSIZE	0x8002A040
HW_PXP_S0BACKGROUND	0x8002A090
HW_PXP_S0BUF	0x8002A050
HW_PXP_S0COLORKEYHIGH	0x8002A190
HW_PXP_S0COLORKEYLOW	0x8002A180
HW_PXP_S0CROP	0x8002A0A0
HW_PXP_S0OFFSET	0x8002A0C0
HW_PXP_S0PARAM	0x8002A080
HW_PXP_S0SCALE	0x8002A0B0
HW_PXP_S0UBUF	0x8002A060
HW_PXP_S0VBUF	0x8002A070
HW_PXP_STAT	0x8002A010
HW_PXP_STAT_CLR	0x8002A018
HW_PXP_STAT_SET	0x8002A014
HW_PXP_STAT_TOG	0x8002A01C
HW_PXP_VERSION	0x8002A1F0
HW_RTC_ALARM	0x8005C040
HW_RTC_ALARM_CLR	0x8005C048
HW_RTC_ALARM_SET	0x8005C044
HW_RTC_ALARM_TOG	0x8005C04C
HW_RTC_CTRL	0x8005C000
HW_RTC_CTRL_CLR	0x8005C008
HW_RTC_CTRL_SET	0x8005C004
HW_RTC_CTRL_TOG	0x8005C00C
HW_RTC_DEBUG	0x8005C0C0
HW_RTC_DEBUG_CLR	0x8005C0C8
HW_RTC_DEBUG_SET	0x8005C0C4
HW_RTC_DEBUG_TOG	0x8005C0CC
HW_RTC_MILLISECONDS	0x8005C020

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_RTC_MILLISECONDS_CLR	0x8005C028
HW_RTC_MILLISECONDS_SET	0x8005C024
HW_RTC_MILLISECONDS_TOG	0x8005C02C
HW_RTC_PERSISTENT0	0x8005C060
HW_RTC_PERSISTENT0_CLR	0x8005C068
HW_RTC_PERSISTENT0_SET	0x8005C064
HW_RTC_PERSISTENT0_TOG	0x8005C06C
HW_RTC_PERSISTENT1	0x8005C070
HW_RTC_PERSISTENT1_CLR	0x8005C078
HW_RTC_PERSISTENT1_SET	0x8005C074
HW_RTC_PERSISTENT1_TOG	0x8005C07C
HW_RTC_PERSISTENT2	0x8005C080
HW_RTC_PERSISTENT2_CLR	0x8005C088
HW_RTC_PERSISTENT2_SET	0x8005C084
HW_RTC_PERSISTENT2_TOG	0x8005C08C
HW_RTC_PERSISTENT3	0x8005C090
HW_RTC_PERSISTENT3_CLR	0x8005C098
HW_RTC_PERSISTENT3_SET	0x8005C094
HW_RTC_PERSISTENT3_TOG	0x8005C09C
HW_RTC_PERSISTENT4	0x8005C0A0
HW_RTC_PERSISTENT4_CLR	0x8005C0A8
HW_RTC_PERSISTENT4_SET	0x8005C0A4
HW_RTC_PERSISTENT4_TOG	0x8005C0AC
HW_RTC_PERSISTENT5	0x8005C0B0
HW_RTC_PERSISTENT5_CLR	0x8005C0B8
HW_RTC_PERSISTENT5_SET	0x8005C0B4
HW_RTC_PERSISTENT5_TOG	0x8005C0BC
HW_RTC_SECONDS	0x8005C030
HW_RTC_SECONDS_CLR	0x8005C038
HW_RTC_SECONDS_SET	0x8005C034
HW_RTC_SECONDS_TOG	0x8005C03C
HW_RTC_STAT	0x8005C010
HW_RTC_STAT_CLR	0x8005C018

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_RTC_STAT_SET	0x8005C014
HW_RTC_STAT_TOG	0x8005C01C
HW_RTC_VERSION	0x8005C0D0
HW_RTC_WATCHDOG	0x8005C050
HW_RTC_WATCHDOG_CLR	0x8005C058
HW_RTC_WATCHDOG_SET	0x8005C054
HW_RTC_WATCHDOG_TOG	0x8005C05C
HW_SAIF_CTRL	0x80042000
HW_SAIF_CTRL_CLR	0x80042008
HW_SAIF_CTRL_SET	0x80042004
HW_SAIF_CTRL_TOG	0x8004200C
HW_SAIF_DATA	0x80042020
HW_SAIF_DATA_CLR	0x80042028
HW_SAIF_DATA_SET	0x80042024
HW_SAIF_DATA_TOG	0x8004202C
HW_SAIF_STAT	0x80042010
HW_SAIF_STAT_CLR	0x80042018
HW_SAIF_STAT_SET	0x80042014
HW_SAIF_STAT_TOG	0x8004201C
HW_SAIF_VERSION	0x80042030
HW_SPDIF_CTRL	0x80054000
HW_SPDIF_CTRL_CLR	0x80054008
HW_SPDIF_CTRL_SET	0x80054004
HW_SPDIF_CTRL_TOG	0x8005400C
HW_SPDIF_DATA	0x80054050
HW_SPDIF_DATA_CLR	0x80054058
HW_SPDIF_DATA_SET	0x80054054
HW_SPDIF_DATA_TOG	0x8005405C
HW_SPDIF_DEBUG	0x80054040
HW_SPDIF_DEBUG_CLR	0x80054048
HW_SPDIF_DEBUG_SET	0x80054044
HW_SPDIF_DEBUG_TOG	0x8005404C
HW_SPDIF_FRAMECTRL	0x80054020

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_SPDIF_FRAMECTRL_CLR	0x80054028
HW_SPDIF_FRAMECTRL_SET	0x80054024
HW_SPDIF_FRAMECTRL_TOG	0x8005402C
HW_SPDIF_SRR	0x80054030
HW_SPDIF_SRR_CLR	0x80054038
HW_SPDIF_SRR_SET	0x80054034
HW_SPDIF_SRR_TOG	0x8005403C
HW_SPDIF_STAT	0x80054010
HW_SPDIF_STAT_CLR	0x80054018
HW_SPDIF_STAT_SET	0x80054014
HW_SPDIF_STAT_TOG	0x8005401C
HW_SPDIF_VERSION	0x80054060
HW_SSP_CMD0	0x80010010
HW_SSP_CMD0_CLR	0x80010018
HW_SSP_CMD0_SET	0x80010014
HW_SSP_CMD0_TOG	0x8001001C
HW_SSP_CMD1	0x80010020
HW_SSP_COMPMASK	0x80010040
HW_SSP_COMPREF	0x80010030
HW_SSP_CTRL0	0x80010000
HW_SSP_CTRL0_CLR	0x80010008
HW_SSP_CTRL0_SET	0x80010004
HW_SSP_CTRL0_TOG	0x8001000C
HW_SSP_CTRL1	0x80010060
HW_SSP_CTRL1_CLR	0x80010068
HW_SSP_CTRL1_SET	0x80010064
HW_SSP_CTRL1_TOG	0x8001006C
HW_SSP_DATA	0x80010070
HW_SSP_DEBUG	0x80010100
HW_SSP_SDRESP0	0x80010080
HW_SSP_SDRESP1	0x80010090
HW_SSP_SDRESP2	0x800100A0
HW_SSP_SDRESP3	0x800100B0

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_SSP_STATUS	0x800100C0
HW_SSP_TIMING	0x80010050
HW_SSP_VERSION	0x80010110
HW_TIMROT_ROTCOUNT	0x80068010
HW_TIMROT_ROTCTRL	0x80068000
HW_TIMROT_ROTCTRL_CLR	0x80068008
HW_TIMROT_ROTCTRL_SET	0x80068004
HW_TIMROT_ROTCTRL_TOG	0x8006800C
HW_TIMROT_TIMCOUNT0	0x80068030
HW_TIMROT_TIMCOUNT1	0x80068050
HW_TIMROT_TIMCOUNT2	0x80068070
HW_TIMROT_TIMCOUNT3	0x80068090
HW_TIMROT_TIMCTRL0	0x80068020
HW_TIMROT_TIMCTRL0_CLR	0x80068028
HW_TIMROT_TIMCTRL0_SET	0x80068024
HW_TIMROT_TIMCTRL0_TOG	0x8006802C
HW_TIMROT_TIMCTRL1	0x80068040
HW_TIMROT_TIMCTRL1_CLR	0x80068048
HW_TIMROT_TIMCTRL1_SET	0x80068044
HW_TIMROT_TIMCTRL1_TOG	0x8006804C
HW_TIMROT_TIMCTRL2	0x80068060
HW_TIMROT_TIMCTRL2_CLR	0x80068068
HW_TIMROT_TIMCTRL2_SET	0x80068064
HW_TIMROT_TIMCTRL2_TOG	0x8006806C
HW_TIMROT_TIMCTRL3	0x80068080
HW_TIMROT_TIMCTRL3_CLR	0x80068088
HW_TIMROT_TIMCTRL3_SET	0x80068084
HW_TIMROT_TIMCTRL3_TOG	0x8006808C
HW_TIMROT_VERSION	0x800680a0
HW_TVENC_CLOSEDCAPTION	0x800380f0
HW_TVENC_CLOSEDCAPTION_CLR	0x800380f8
HW_TVENC_CLOSEDCAPTION_SET	0x800380f4
HW_TVENC_CLOSEDCAPTION_TOG	0x800380fC

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_TVENC_COLORBURST	0x80038140
HW_TVENC_COLORBURST_CLR	0x80038148
HW_TVENC_COLORBURST_SET	0x80038144
HW_TVENC_COLORBURST_TOG	0x8003814C
HW_TVENC_COLORSUB0	0x800380c0
HW_TVENC_COLORSUB0_CLR	0x800380c8
HW_TVENC_COLORSUB0_SET	0x800380c4
HW_TVENC_COLORSUB0_TOG	0x800380cC
HW_TVENC_COLORSUB1	0x800380d0
HW_TVENC_COLORSUB1_CLR	0x800380d8
HW_TVENC_COLORSUB1_SET	0x800380d4
HW_TVENC_COLORSUB1_TOG	0x800380dC
HW_TVENC_CONFIG	0x80038010
HW_TVENC_CONFIG_CLR	0x80038018
HW_TVENC_CONFIG_SET	0x80038014
HW_TVENC_CONFIG_TOG	0x8003801C
HW_TVENC_COPYPROTECT	0x800380e0
HW_TVENC_COPYPROTECT_CLR	0x800380e8
HW_TVENC_COPYPROTECT_SET	0x800380e4
HW_TVENC_COPYPROTECT_TOG	0x800380eC
HW_TVENC_CTRL	0x80038000
HW_TVENC_CTRL_CLR	0x80038008
HW_TVENC_CTRL_SET	0x80038004
HW_TVENC_CTRL_TOG	0x8003800C
HW_TVENC_DACCTRL	0x800381a0
HW_TVENC_DACCTRL_CLR	0x800381a8
HW_TVENC_DACCTRL_SET	0x800381a4
HW_TVENC_DACCTRL_TOG	0x800381aC
HW_TVENC_DACSTATUS	0x800381b0
HW_TVENC_DACSTATUS_CLR	0x800381b8
HW_TVENC_DACSTATUS_SET	0x800381b4
HW_TVENC_DACSTATUS_TOG	0x800381bC
HW_TVENC_FILTCTRL	0x80038020

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_TVENC_FILTCTRL_CLR	0x80038028
HW_TVENC_FILTCTRL_SET	0x80038024
HW_TVENC_FILTCTRL_TOG	0x8003802C
HW_TVENC_HTIMINGACTIVE	0x80038060
HW_TVENC_HTIMINGACTIVE_CLR	0x80038068
HW_TVENC_HTIMINGACTIVE_SET	0x80038064
HW_TVENC_HTIMINGACTIVE_TOG	0x8003806C
HW_TVENC_HTIMINGBURST0	0x80038070
HW_TVENC_HTIMINGBURST0_CLR	0x80038078
HW_TVENC_HTIMINGBURST0_SET	0x80038074
HW_TVENC_HTIMINGBURST0_TOG	0x8003807C
HW_TVENC_HTIMINGBURST1	0x80038080
HW_TVENC_HTIMINGBURST1_CLR	0x80038088
HW_TVENC_HTIMINGBURST1_SET	0x80038084
HW_TVENC_HTIMINGBURST1_TOG	0x8003808C
HW_TVENC_HTIMINGSYNC0	0x80038040
HW_TVENC_HTIMINGSYNC0_CLR	0x80038048
HW_TVENC_HTIMINGSYNC0_SET	0x80038044
HW_TVENC_HTIMINGSYNC0_TOG	0x8003804C
HW_TVENC_HTIMINGSYNC1	0x80038050
HW_TVENC_HTIMINGSYNC1_CLR	0x80038058
HW_TVENC_HTIMINGSYNC1_SET	0x80038054
HW_TVENC_HTIMINGSYNC1_TOG	0x8003805C
HW_TVENC_MACROVISION0	0x80038150
HW_TVENC_MACROVISION0_CLR	0x80038158
HW_TVENC_MACROVISION0_SET	0x80038154
HW_TVENC_MACROVISION0_TOG	0x8003815C
HW_TVENC_MACROVISION1	0x80038160
HW_TVENC_MACROVISION1_CLR	0x80038168
HW_TVENC_MACROVISION1_SET	0x80038164
HW_TVENC_MACROVISION1_TOG	0x8003816C
HW_TVENC_MACROVISION2	0x80038170
HW_TVENC_MACROVISION2_CLR	0x80038178

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_TVENC_MACROVISION2_SET	0x80038174
HW_TVENC_MACROVISION2_TOG	0x8003817C
HW_TVENC_MACROVISION3	0x80038180
HW_TVENC_MACROVISION3_CLR	0x80038188
HW_TVENC_MACROVISION3_SET	0x80038184
HW_TVENC_MACROVISION3_TOG	0x8003818C
HW_TVENC_MACROVISION4	0x80038190
HW_TVENC_MACROVISION4_CLR	0x80038198
HW_TVENC_MACROVISION4_SET	0x80038194
HW_TVENC_MACROVISION4_TOG	0x8003819C
HW_TVENC_MISC	0x800380b0
HW_TVENC_MISC_CLR	0x800380b8
HW_TVENC_MISC_SET	0x800380b4
HW_TVENC_MISC_TOG	0x800380bC
HW_TVENC_SYNCOFFSET	0x80038030
HW_TVENC_SYNCOFFSET_CLR	0x80038038
HW_TVENC_SYNCOFFSET_SET	0x80038034
HW_TVENC_SYNCOFFSET_TOG	0x8003803C
HW_TVENC_VDACTEST	0x800381c0
HW_TVENC_VDACTEST_CLR	0x800381c8
HW_TVENC_VDACTEST_SET	0x800381c4
HW_TVENC_VDACTEST_TOG	0x800381cC
HW_TVENC_VERSION	0x800381d0
HW_TVENC_VTIMING0	0x80038090
HW_TVENC_VTIMING0_CLR	0x80038098
HW_TVENC_VTIMING0_SET	0x80038094
HW_TVENC_VTIMING0_TOG	0x8003809C
HW_TVENC_VTIMING1	0x800380a0
HW_TVENC_VTIMING1_CLR	0x800380a8
HW_TVENC_VTIMING1_SET	0x800380a4
HW_TVENC_VTIMING1_TOG	0x800380aC
HW_UARTAPP_CTRL0	0x8006C000
HW_UARTAPP_CTRL0_CLR	0x8006C008

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_UARTAPP_CTRL0_SET	0x8006C004
HW_UARTAPP_CTRL0_TOG	0x8006C00C
HW_UARTAPP_CTRL1	0x8006C010
HW_UARTAPP_CTRL1_CLR	0x8006C018
HW_UARTAPP_CTRL1_SET	0x8006C014
HW_UARTAPP_CTRL1_TOG	0x8006C01C
HW_UARTAPP_CTRL2	0x8006C020
HW_UARTAPP_CTRL2_CLR	0x8006C028
HW_UARTAPP_CTRL2_SET	0x8006C024
HW_UARTAPP_CTRL2_TOG	0x8006C02C
HW_UARTAPP_DATA	0x8006C060
HW_UARTAPP_DEBUG	0x8006C080
HW_UARTAPP_INTR	0x8006C050
HW_UARTAPP_INTR_CLR	0x8006C058
HW_UARTAPP_INTR_SET	0x8006C054
HW_UARTAPP_INTR_TOG	0x8006C05C
HW_UARTAPP_LINECTRL	0x8006C030
HW_UARTAPP_LINECTRL_CLR	0x8006C038
HW_UARTAPP_LINECTRL_SET	0x8006C034
HW_UARTAPP_LINECTRL_TOG	0x8006C03C
HW_UARTAPP_LINECTRL2	0x8006C040
HW_UARTAPP_LINECTRL2_CLR	0x8006C048
HW_UARTAPP_LINECTRL2_SET	0x8006C044
HW_UARTAPP_LINECTRL2_TOG	0x8006C04C
HW_UARTAPP_STAT	0x8006C070
HW_UARTAPP_VERSION	0x8006C090
HW_UARTDBGCR	0x80070030
HW_UARTDBGDMACR	0x80070048
HW_UARTDBGDR	0x80070000
HW_UARTDBGFBRD	0x80070028
HW_UARTDBGFR	0x80070018
HW_UARTDBGIBRD	0x80070024
HW_UARTDBGICR	0x80070044

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_UARTDBGIFLS	0x80070034
HW_UARTDBGILPR	0x80070020
HW_UARTDBGIMSC	0x80070038
HW_UARTDBGLCR_H	0x8007002C
HW_UARTDBGMIS	0x80070040
HW_UARTDBGRIS	0x8007003C
HW_UARTDBGRSR_ECR	0x80070004
HW_USBCTRL_ASYNCLISTADDR	0x80080158
HW_USBCTRL_BURSTSIZE	0x80080160
HW_USBCTRL_CAPLENGTH	0x80080100
HW_USBCTRL_DCCPARAMS	0x80080124
HW_USBCTRL_DCIVERSION	0x80080120
HW_USBCTRL_DEVICEADDR	0x80080154
HW_USBCTRL_ENDPOINTLISTADDR	0x80080158
HW_USBCTRL_ENDPTCOMPLETE	0x800801bc
HW_USBCTRL_ENDPTCTRL0	0x800801c0
HW_USBCTRL_ENDPTCTRL1	0x800801c4
HW_USBCTRL_ENDPTCTRL2	0x800801c8
HW_USBCTRL_ENDPTCTRL3	0x800801cc
HW_USBCTRL_ENDPTCTRL4	0x800801d0
HW_USBCTRL_ENDPTFLUSH	0x800801b4
HW_USBCTRL_ENDPTNAK	0x80080178
HW_USBCTRL_ENDPTNAKEN	0x8008017c
HW_USBCTRL_ENDPTPRIME	0x800801b0
HW_USBCTRL_ENDPTSETUPSTAT	0x800801ac
HW_USBCTRL_ENDPTSTAT	0x800801b8
HW_USBCTRL_FRINDEX	0x8008014c
HW_USBCTRL_GPTIMER0CTRL	0x80080084
HW_USBCTRL_GPTIMER0LD	0x80080080
HW_USBCTRL_GPTIMER1CTRL	0x8008008c
HW_USBCTRL_GPTIMER1LD	0x80080088
HW_USBCTRL_HCCPARAMS	0x80080108
HW_USBCTRL_HCSPARAMS	0x80080104

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_USBCTRL_HWDEVICE	0x8008000c
HW_USBCTRL_HWGENERAL	0x80080004
HW_USBCTRL_HWHOST	0x80080008
HW_USBCTRL_HWRXBUF	0x80080014
HW_USBCTRL_HWTXBUF	0x80080010
HW_USBCTRL_IC_USB	0x8008016c
HW_USBCTRL_ID	0x80080000
HW_USBCTRL_OTGSC	0x800801a4
HW_USBCTRL_PERIODICLISTBASE	0x80080154
HW_USBCTRL_PORTSC1	0x80080184
HW_USBCTRL_SBUSCFG	0x80080090
HW_USBCTRL_TTCTRL	0x8008015c
HW_USBCTRL_TXFILLTUNING	0x80080164
HW_USBCTRL_ULPI	0x80080170
HW_USBCTRL_USBCMD	0x80080140
HW_USBCTRL_USBINTR	0x80080148
HW_USBCTRL_USBMODE	0x800801a8
HW_USBCTRL_USBSTS	0x80080144
HW_USBPHY_CTRL	0x8007c030
HW_USBPHY_CTRL_CLR	0x8007c038
HW_USBPHY_CTRL_SET	0x8007c034
HW_USBPHY_CTRL_TOG	0x8007c03c
HW_USBPHY_DEBUG	0x8007c050
HW_USBPHY_DEBUG_CLR	0x8007c058
HW_USBPHY_DEBUG_SET	0x8007c054
HW_USBPHY_DEBUG_TOG	0x8007c05c
HW_USBPHY_DEBUG0_STATUS	0x8007c060
HW_USBPHY_DEBUG1	0x8007c070
HW_USBPHY_DEBUG1_CLR	0x8007c078
HW_USBPHY_DEBUG1_SET	0x8007c074
HW_USBPHY_DEBUG1_TOG	0x8007c07c
HW_USBPHY_IP	0x8007c090
HW_USBPHY_IP_CLR	0x8007c098

Table B-1. Register Names and Addresses (continued)

Register Name	Address
HW_USBPHY_IP_SET	0x8007c094
HW_USBPHY_IP_TOG	0x8007c09c
HW_USBPHY_PWD	0x8007c000
HW_USBPHY_PWD_CLR	0x8007c008
HW_USBPHY_PWD_SET	0x8007c004
HW_USBPHY_PWD_TOG	0x8007c00c
HW_USBPHY_RX	0x8007c020
HW_USBPHY_RX_CLR	0x8007c028
HW_USBPHY_RX_SET	0x8007c024
HW_USBPHY_RX_TOG	0x8007c02c
HW_USBPHY_STATUS	0x8007c040
HW_USBPHY_TX	0x8007c010
HW_USBPHY_TX_CLR	0x8007c018
HW_USBPHY_TX_SET	0x8007c014
HW_USBPHY_TX_TOG	0x8007c01c
HW_USBPHY_VERSION	0x8007c080

Appendix C
Acronyms and Abbreviations

This appendix includes definitions for many of the acronyms and abbreviations found in this product data sheet.

AAC: Advanced Audio Coding
AC: Audio Coding
ADC: Analog-to-Digital Converter
AES: Advanced Encryption Standard
AHB: Advanced High-performance Bus
AIO: Analog Input/Output
AMBA: Advanced Microcontroller Bus Architecture
APB: Advanced Peripheral Bus
APBH: Advanced Peripheral Bus—HCLK Domain
APBX: Advanced Peripheral Bus—XCLK Domain
ARC: ARC International (corporate name)
ARM: Advanced RISC Machine (formerly Acorn RISC Machine)
AVC: Adaptive Voltage Control
BATT: Battery
BCB: Boot Control Block
BGA: Ball Grid Array
BIST: Built-In Self-Test
BKPT: Breakpoint
BLTC: Boot Loader Transaction Controller
CBC Cipher Block Chaining
CCS: Command Completion Signaling
CE: Consumer Electronics
CLKCTRL: Clock Control
CP: Charge Pump
CPUCLK: Processor (ARM CPU) Clock
CRC: Cyclic Redundancy Check
CSC: Color-Space Conversion
CTS: Clear To Send
DABT: Data Abort
DAC: Digital-to-Analog Converter
dB: Decibel

DCP: Data Co-Processor
DBBT: Discovered Bad Block Table
DES: Data Encryption Standard
DFD: Digital Fractional Divider
DIGCTL: Digital Control
DIO: Digital Input/Output
DiVX: Digital video codec created by DivXNetworks, Inc.
DMA: Direct Memory Access
DVI: Digital Video Interface (ITU-R BT.656 mode)
ECB: Electronic Book Code
ECC: Error Correction Code
EL: Electroluminescent
EMI: External Memory Interface
EMICLK: EMI Clock
ETM: Embedded Trace Macrocell
FIQ: Fast Peripheral Interrupt
FLPT: First-Level Page Table
FREQ: Frequency
FS: Full-Speed
FSM: Finite State Machine
GPIO: General-Purpose Input/Output
GPMI: General-Purpose Media Interface
GPMICLK: GMPI Clock
HCLK: Main and HBUS Peripherals Clock
HID: Human Interface Device
HS: High-Speed
HW: Hardware
H.264: High-Compression Digital Video Codec
ICOLL: Interrupt Collector
IrDA: Infrared Data Association
IROVCLK: IR Clock (sourced from PLL)
IRCLK: IR Clock (source from IROVCLK)
IRQ: Normal Peripheral Interrupt
ISR: Interrupt Service Register
JEDEC: Joint Electron Device Engineering Council
JPEG: Joint Photographic Experts Group (computer image format)
JTAG: Joint Test Action Group
LDLB: Logical Drive Layout Block
LFE: Low Frequency Effects
Li-Ion: Lithium Ion (battery type)

LJ: Left-Justified
LQFP: Low-Profile Quad Flat Pack
LRADC: Low Resolution ADC
LSB: Least Significant Bit
MATT: Multi-chip Attachment mode
MBR: Master Boot Record
MMC: Multi-Media Card
MPEG4: Motion Picture Experts Group 4 (standard for compressed video at 64 kbps)
MP3: Moving Picture Experts Group Layer-3 Audio
MSB: Most Significant Bit
Mux: Multiplexer
NCB: NAND Control Block
NiMH: Nickel Metal Hydride
NRZI: Non-Return to Zero Inverted
NTSC: National Television Systems Committee
OCRAM: On-chip Random Access Memory
OCOTP: On-chip, One Time Programmable
OTP: One Time Programmable
PABT: Instruction Pre-Fetch Abort
PAL: Phase Alternating Line
PCM: Pulse Code Modulation
PDA: Personal Digital Assistant
PDDRM: Portable Device Digital Rights Management (DRM9)
PFD: Phase Fractional Divider
PFM: Pulse Frequency Modulation
PHY: Physical Layer Protocol
PLL: Phase-Locked Loop
PITC: Plug-In Transfer Controller
PWM: Pulse Width Modulation
RHID: Recovery Human Interface Device
RJ: Right-Justified
RTC: Real-Time Clock
RTS: Request To Send
RMW: Read-Modify-Write
SAIF: Serial Audio Interface
SB: Safe Boot
SDIO: Secure Digital Input/Output
SDK: Software Development Kit
SHA: Secure Hash Algorithm
SJTAG: Serial JTAG

SNR:	Signal-to-Noise Ratio
SOC:	System-on-a-Chip
SPDIF:	Sony-Philips Digital Interface Format
SPDIFCLK:	SPDIF Clock
SPI:	Serial Peripheral Interface
SSI:	Synchronous Serial Interface
SSP:	Synchronous Serial Port
SWI:	Software Interrupt
TAP:	Test Access Port
TBD:	To Be Determined
TDEA	Triple Data Encryption Algorithm
THD:	Total Harmonic Distortion
TPC:	Transfer Protocol Commands
UNDEF:	Undefined instruction
UDMA:	Ultra Direct Memory Access
UTMI:	USB 2.0 Transceiver Macrocell Interface
VAG:	Analog Ground Voltage
VBG:	Internal Bandgap Voltage
VCO:	Variable Crystal Oscillator
VDDA:	Analog Power
VDDD:	Digital Power
VMI:	Virtual Memory Interface
WMDRM10:	Windows Media® Digital Rights Management 10 (Janus)
WMA:	Windows Media® Audio
XCLK:	XBUS Peripherals Clock