Triscend A7S Configurable System-on-Chip Platform

■ **Industry's first complete 32-bit Configurable System-on-Chip (CSoC)**

- High-performance, low-power consumption, 32-bit RISC processor (ARM7TDMI™)
- 8Kbyte mixed instruction/data cache
- 16Kbyte internal scratchpad RAM
- Next-generation embedded programmable logic architecture (up to 25,600 ASIC gates)
- High-performance dedicated internal bus (up to 455Mbytes per second at 60 MHz)
- External memory interface supporting Flash, EEPROM, SRAM, and SDRAM
- Advanced real-time, in-system debugging capability
- Stand-alone operation from a single external memory (code + initialization)
- 2.5-volt core with 3.3- or 2.5-volt I/Os
- Four independent high-performance DMA channels

■ **High-performance, 32-bit ARM7TDMI RISC Processor**

- Popular, industry-standard 32-bit RISC processor
- Binary and source code compatible with other ARM7/ARM7TDMI variants
- Widespread C/C++ compiler, source-level debugger, and RTOS support
- Superior code density using the Thumb® instruction set
- 54 MIPS (Dhrystone 2.1) at 60 MHz
- Low latency, real-time interrupt response
- Fast hardware multiplier
- 32-bit register bank and ALU
- 32-bit addressing — 4Gbyte linear address
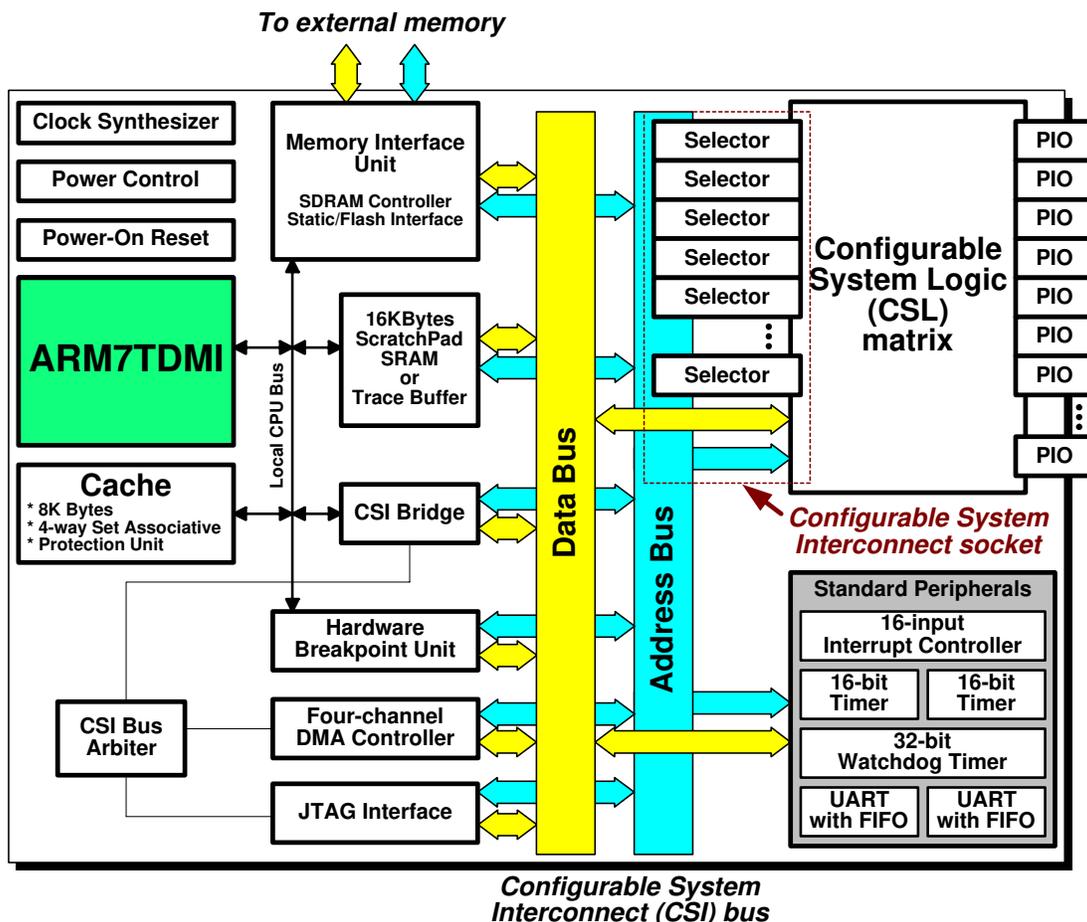- 32-bit barrel shifter
- EmbeddedICE™ on-chip debugger

Figure 1. Block diagram of the Triscend A7S Configurable System-on-Chip (CSoC).

TCH305-0001-002

Table 1. Triscend A7S Configurable System-on-Chip Family

Device	Embedded Processor Core	Dedicated Resources	System RAM	Configurable System Logic (CSL) Cells	CSI Address Selectors	PIO* Pins (Max)
TA7S04	**ARM7TDMI** 32-bit RISC CPU 8K unified cache Barrel shifter Hardware multiplier Thumb extensions Debug extensions	Flash memory interface SDRAM memory interface 4-channel DMA controller Two 16C550-style UARTs Two 16-bit timers 32-bit watchdog timer 16-input interrupt controller Power management Power-on reset Hardware breakpoint unit JTAG debugger	4Kx32	448	32	124
TA7S20				2,048	128	252

* Maximum PIO on each base device, actual PIO count depends on package style and initialization mode. See <u>Table 61</u>.

■ **Rich set of embedded support peripherals**

- 4-channel high-performance DMA controller
  - fly-by performance
  - memory-to-memory transfers
  - linked-list DMA
  - frame transfer support
- Memory Subsystem Interface Unit (MSSIU) for flexible, glueless interface to external memories (ROM, EEPROM, Flash, SRAM, and SDRAM)
- Two 16C550-style serial ports (UART) with modem interface
- Two 16-bit timers/counters
- 32-bit Watchdog timer
- 16-input interrupt controller with fast interrupt response
- IEEE 1149.1 enhanced JTAG interface
- In-system debug/breakpoint unit
- Power-on reset
- Power-down and power-management modes

■ **Full-Featured Memory Interface Unit**

- Simultaneous support for independent external Flash and SDRAM memory subsystems using x8 or x16 memory devices
- Expandable external data bus: 8-bit, 16-bit and 32-bit support
- Up to two external SDRAM banks
- Automatic support for self-refresh, auto-refresh and initialization of SDRAM
- Programmable SDRAM parameters for optimal memory bandwidth

■ **Embedded SRAM-based Configurable System Logic (CSL) matrix**

- Next-generation embedded programmable logic architecture, optimized with processor and bus interface
- Over 2,600 flip-flops and 190 programmable inputs and outputs (PIOs)
- Abundant, flexible interconnect structure with easy access to and from system bus
- Dedicated circuitry for fast adders, counters, and multipliers
- CSL cells optionally used as distributed memory, including true dual-port operation
- Six independent low-skew clock or global signal distribution buffers plus bus clock
- Supported by standard logic design tools
  - VHDL and Verilog logic synthesis
  - Schematic entry
  - VHDL and Verilog simulation

■ **High performance dedicated system bus**

- Configurable System Interconnect (CSI) bus integrates CSL matrix, CSoC system
- 455Mbytes per second peak transfer rate
- 32-bit address bus and 32-bit data bus
- Programmable wait-state support
- Openly-defined CSI Socket bus interface to CSL matrix
  - CSL peripheral addresses independent of placement in CSL matrix
  - CSL peripherals compatible with past and future CSoC families
- Ten bus masters and built-in arbitration
  - ARM7TDMI™ CPU
  - Four-channel DMA controller
  - JTAG interface

System Overview

The Triscend A7S Configurable System-on-Chip (CSoC) device is a complete, high-performance user-programmable system. The A7S contains

- an embedded 32-bit ARM7TDMI RISC processor
- a next generation embedded programmable logic architecture, optimized for processor and bus interface
- a high-performance 32-bit internal bus supporting up to 455Mbytes per second peak transfer rates
- 16Kbytes of internal scratchpad SRAM memory and a separate 8Kbyte cache.

The ARM7TDMI is a general-purpose 32-bit RISC microprocessor that supports the complete ARM 32-bit instruction set and the reduced 16-bit instruction set, referred to as Thumb. The ARM7TDMI processor offers the following advantages:

- High-performance for very low power consumption and price
- Excellent code density using the Thumb instruction set
- Low-latency interrupt response

ARM7TDMI Processor System with Cache, Scratchpad RAM

The processor is paired with an 8Kbyte unified code/data cache and a 16Kbyte (4Kx32) scratchpad RAM for storing timing critical code or data. The scratchpad is accessible over the Configurable System Interconnect (CSI) bus by other CSI bus masters, primarily for DMA transfers. The ARM processor is integrated with other system components and the Configurable System Logic (CSL) matrix to provide a complete configurable system, as illustrated in Figure 1.

Next-Generation Embedded Programmable Logic Architecture

The embedded SRAM-based Configurable System Logic (CSL) matrix provides full, easy-to-use system customization. The high-performance programmable logic architecture consists of a highly interconnected matrix of CSL cells. Resources within the matrix provide seamless access to and from the internal CSI bus. Each CSL cell performs various potential functions, including combinatorial and sequential logic. The combinatorial portion performs Boolean logic operations, arithmetic functions, and memory. The sequential element performs independently or in tandem with the combinatorial function. The abundant programmable input/output blocks (PIOs) provide a highly flexible interface between external functions and the internal system bus or configurable system logic. Each PIO offers advanced I/O capabilities including selectable output drive current, optional input hysteresis, and programmable low-power functionality during power-down mode.

Internal, High-Performance Bus

A high-performance internal system bus—called the Configurable System Interconnect (CSI) bus— interconnects the embedded processor, its peripherals, and the CSL matrix at a maximum speed of 60MHz. The bus simultaneously provides 32 bits of read data, 32 bits of write data, and a 32-bit address. Multiple bus masters arbitrate for bus access. Potential bus masters include the ARM7TDMI processor, the read and write channels of all four DMA channels, and the JTAG interface. CSL-based devices become CSI bus masters using DMA services. The CSI bus and the local CPU bus following the little endian format.
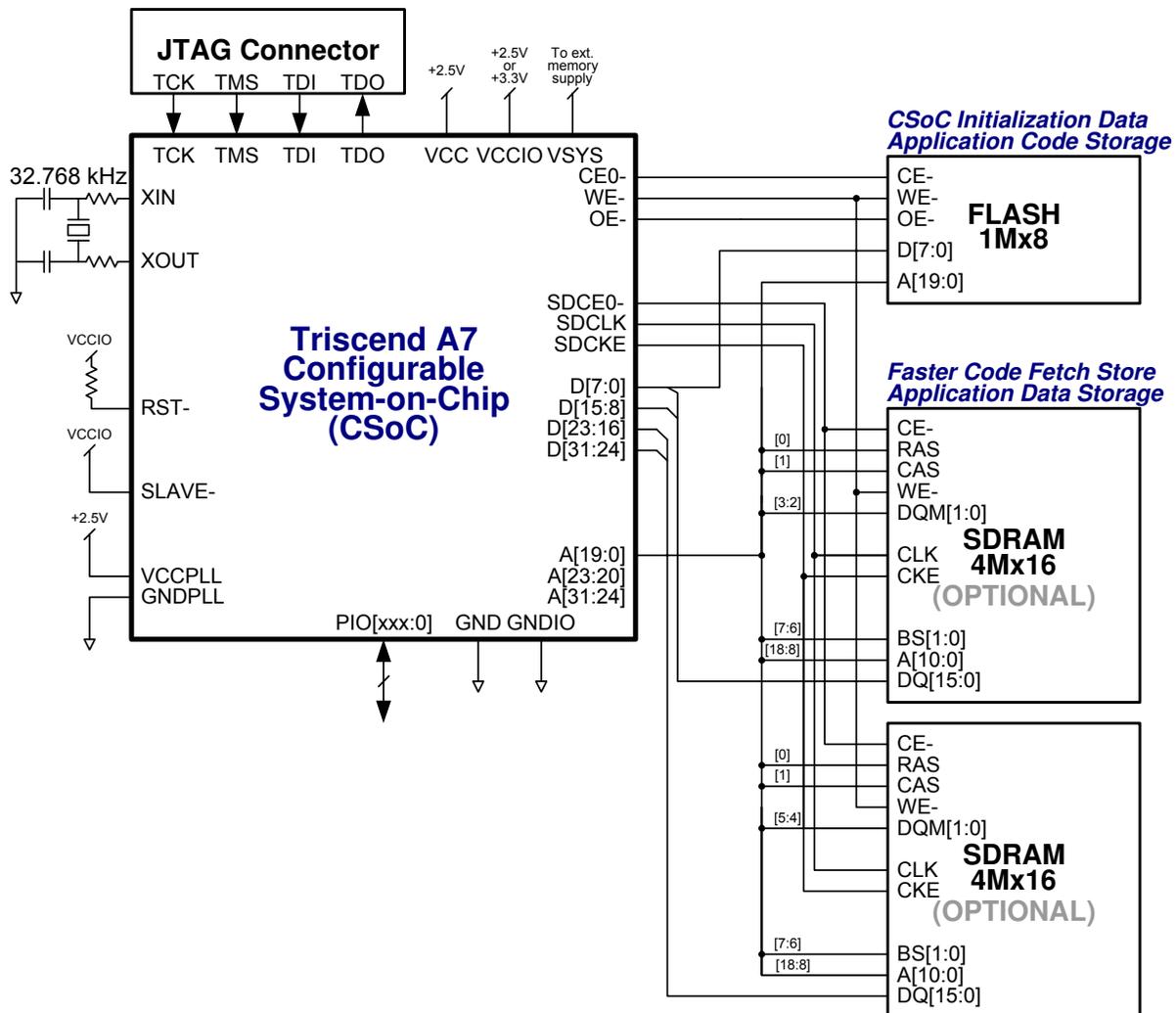
Figure 2. A typical A7S-based system.

External Interface to Flash, SDRAM

A static memory interface unit seamlessly connects the A7S device to external static memories such as Flash or SRAM, as shown in Figure 2. An external Flash memory device contains the A7S's initialization boot program plus the system application code. The external memory interface has programmable read/write control and chip-select signals that provide flexible set-up, strobe, and hold timing. The CPU connects directly to external memory, eliminating any potential latency incurred by using the CSI bus. For low frequency or minimal applications, the ARM7TDMI processor directly fetches its instructions from external Flash.

The A7S optionally supports external SDRAM, offering additional affordable and high-density memory to the system. The SDRAM interface connects an A7S-based system to a variety of SDRAM types and configurations, including 100-pin DIMMs. The SDRAM interface operates at up to 60 MHz and provides options to optimize the interface timing for slower system clocks. SDRAM memory is ideal for DMA buffers. Similarly, the application program can be stored in slow, cheap, byte-wide Flash and copied into SDRAM at power-up. Then, the CPU starts executing code from the wider and faster SDRAM memory. The Flash and SDRAM interfaces share device pins, as shown in Figure 2.

Four-Channel DMA

The four-channel DMA controller provides high-bandwidth communication between CSL-based I/O devices, at up to 228M bytes per second, per direction. The easy-to-use DMA handshake simplifies interface and control logic within the CSL. The DMA controller provides advanced capabilities such as linked-list and frame-transfer support.

Dedicated Peripherals

The A7S also offers a set of common dedicated peripherals including

- two 16-bit timers with pre-scalers,
- two 16C450/550-like serial controllers (UART), with an optional modem interface
- a 32-bit watchdog timer, and
- an interrupt controller.

Complete Single-Chip System

The majority of the system, including the CPU, operates from a single clock signal. The clock source is typically driven directly via an external pin or connected to the on-chip PLL clock synthesizer. The clock synthesizer operates from an external 32.768 kHz watch crystal. Additionally, an internal ring oscillator is provided. Six other global buffers provide high-fanout signals to CSL functions. The bus clock and the global buffers are optionally stopped upon a breakpoint event and shut off during power-down mode.

Power management controls provide selectable power-down options over internal functions. Furthermore, each PIO provides pin-by-pin power-down settings.

An internal initialization boot ROM controls device initialization after power-on or after the reset pin is released. The initialization boot ROM locates user's initialization data and code stored in external Flash or other non-volatile memory. The Triscend FastChip development system programs external Flash via the A7S's JTAG port.

Additionally, the JTAG interface provides real-time, in-system debugging capabilities, eliminating the need for an external emulator. The JTAG interface has full access and control over the CPU, peripherals, and CSL functions during debugging.

When debugging application software, the A7S employs the rich set of standard ARM7TDMI debugging tools. The A7S fully supports the standard ARM internal breakpoint and watchpoint capabilities. In addition, the A7S's breakpoint unit monitors both the CPU local bus or the CSI bus. Upon a predefined set of conditions, the breakpoint unit halts or interrupts the execution of the application program. The breakpoint unit also supports real-time tracing of local CPU bus or the CSI bus transactions.

All together, the Triscend A7S Configurable System-on-Chip (CSoC) platform offers unparalleled time-to-market and performance advantages for embedded system designs.

A7S Development Support

The Triscend A7S Configurable System-on-Chip (CSoC) platform is supported by a variety of third-party development tools including compilers, debuggers, real-time operating systems (RTOS), and in-system debuggers/emulators as shown in Table 2. Most compilers that support the ARM7 architecture also support the Triscend A7S CSoC device. To accelerate development, there are multiple development boards available, shown in Table 3. Additionally, Triscend provides a free Software Development Kit (SDK) that includes board support packages (BSPs) for leading RTOS environments and a source-level driver library.

The A7S's Configurable System Logic (CSL) matrix is well supported by a variety of logic design entry solutions, including both VHDL or Verilog logic synthesis and schematic entry as shown in Table 2. Likewise, there are VHDL and Verilog simulation models available for popular logic verification tools.

Table 2. Supported Development Tools for A7S CSoC.

ARM7TDMI Software Development	CSoC/Logic Design Development
Triscend Software Development Kit (SDK)	***Triscend FastChip CSoC Development System***
▪ Source-level device driver library	▪ Graphical development/integration environment, Windows-based
▪ Board Support Packages (BSPs)	▪ Drag-and-drop Soft Module library
Compilers	▪ Create initialization images for Triscend CSoC devices
▪ Wind River Diab C/C++™ Compiler	▪ Download directly or program external Flash via JTAG
▪ ARM® Developer Suite (ADS) C/C++ Compiler	▪ Seamless integration with third-party microprocessor and logic design tools
▪ GNU C Compiler (GCC)	▪ Powerful real-time, in-system debugging
Source-Level Debuggers	***VHDL/Verilog Logic Synthesis***
▪ Wind River visionCLICK, in-system support using Wind River visionPROBE II	▪ Synplicity® Synplify®
▪ ARM eXtended Debugger (AXD)	▪ Synopsys® FPGA Compiler II
▪ GNU gdb Debugger	***Schematic Entry***
Real-Time Operating System (RTOS) Support	▪ Cadence/OrCAD Capture
▪ Wind River Tornado/VxWorks®	▪ SpinCircuit eCapture
▪ Red Hat eCos™	▪ Innoveda ViewDraw
▪ Red Hat µClinux	***VHDL Logic Simulation***
JTAG-Based Hardware Emulators/Debuggers	▪ Model Technology™ ModelSim
▪ Wind River visionPROBE II (ARM and CSoC debugging)	▪ Innoveda Fusion/Speedwave
▪ ARM Mutli-ICE™ (ARM only debugging)	▪ VITAL/SDF support
▪ EPI JEENI and MAJIC™ (ARM only debugging)	***Verilog Logic Simulation***
	▪ Model Technology™ ModelSim
	▪ Cadence® Verilog XL®
	▪ Synopsys® VCS
	▪ Triscend CSI Bus Functional Model
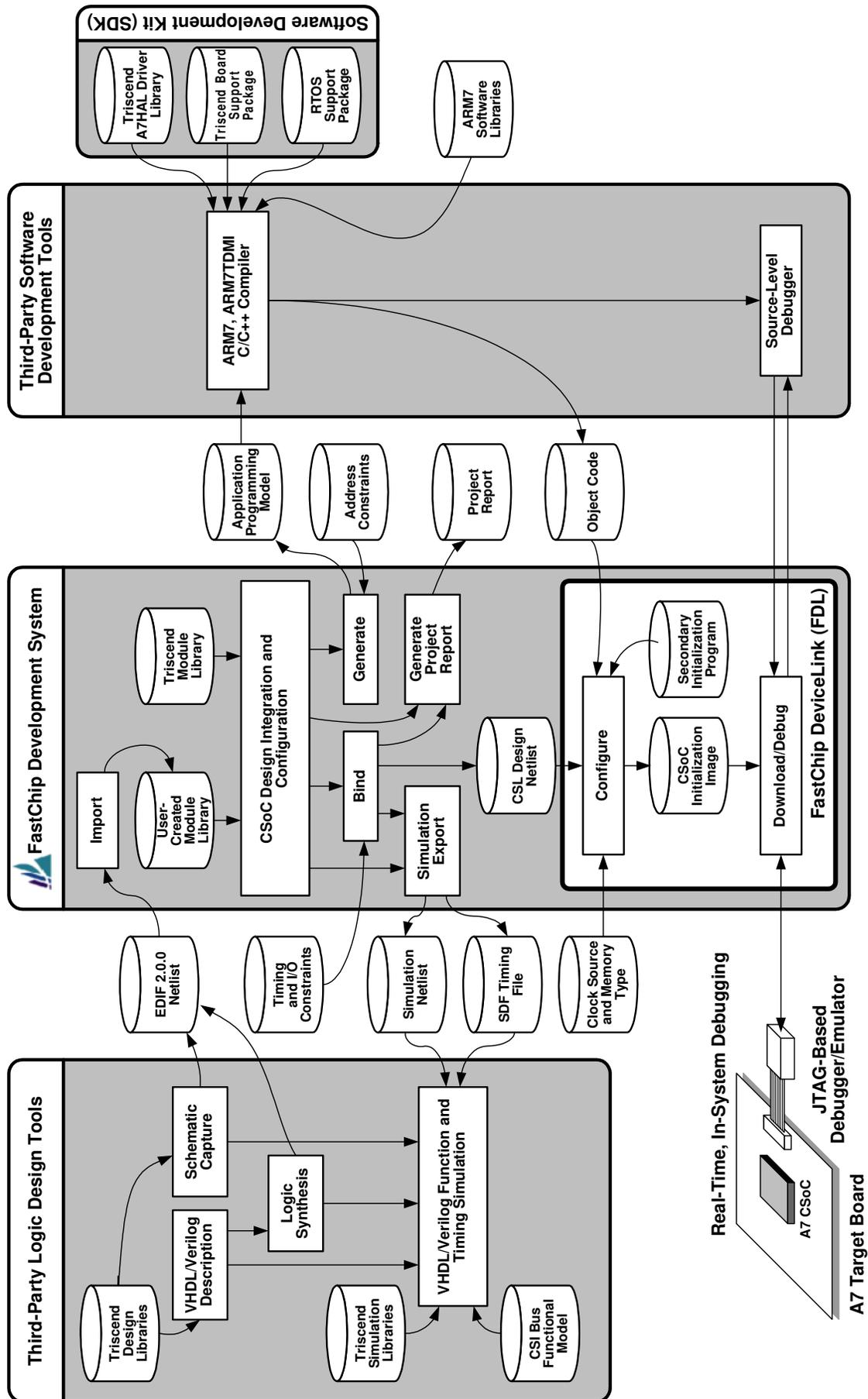
Figure 3. Detailed Triscend A7S Development Flow.

Table 3. A7S Development Boards.

Supplier	Part Number
Triscend Corporation	THW-KIT-720
Embedded Performance Inc. (EPI)	Dev-A7

PC-Based Development Platform

Figure 3 presents a detailed view of the entire Triscend A7S development flow. FastChip is a Windows-based application and operates on most PC-compatible computers with the recommended minimum 192Mbytes of RAM memory. The Triscend FastChip development system provides design integration and configuration capabilities, working in conjunction with third-party logic design and software development tools.

Powerful FastChip CSoC Development System

FastChip includes a powerful Soft Module library of commonly used embedded systems functions like additional UARTs, timers, various bus interfaces, etc. Likewise, FastChip includes libraries that allow designers to create custom functions using third-part logic design and simulation tools. Designs imported into FastChip via an EDIF 2.0.0 netlist become FastChip modules.

FastChip also exports a CSoC designs for either VHDL or Verilog logic simulation purposes. A Triscend-provided bus functional model simulates traffic on the A7S's internal CSI bus.

Seamless Integration with ARM7TDMI Compiler

After defining the A7S's logic, FastChip's Bind utility creates the physical hardware implementation for the CSoC device. Similarly, FastChip's Generate utility allocates addresses for any functions attached to the Configurable System Interconnect (CSI) bus and creates an application programming model for a third-party ARM compiler. This model includes register definitions for both standard ARM7TDMI functions and any custom hardware.

FastChip combines the output from the Bind utility and the object code from the ARM7TDMI compiler to create a CSoC initialization image. Using this image, FastChip either directly downloads to an A7S device or programs external Flash memory attached to the A7. Optionally, the initialization image can be saved as an Intel Hex file four use with an external device programmer.

Real-time, In-system, Full-Speed Debugging

Furthermore, FastChip provides a real-time, in-system debugging environment using the actual A7S production silicon with the actual system hardware and application software. FastChip drives a supported JTAG-based debugger/emulator and provides interfaces to third-party source-level debuggers. Via a source-level debugger, software developers have register-level access to the A7S device, complete with breakpoints and trace. FastChip's Debug utility also provides logic debugging capabilities, including the ability to probe flip-flop values and the outputs of CSL cells.

FastChip's Configure and Download/Debug utilities are packaged as a separate, stand-alone application called FastChip Device Link (FDL), providing software developers with necessary software development capabilities without the complexity of the entire FastChip CSoC development system.

The Triscend A7S Configurable System-on-Chip family and the FastChip development system are supported by a world-wide network of factory-trained field applications engineers. Additionally, the Triscend SupportCenter provides online support via the world-wide web at http://support.triscend.com or via E-mail at SupportCenter@Triscend.com.
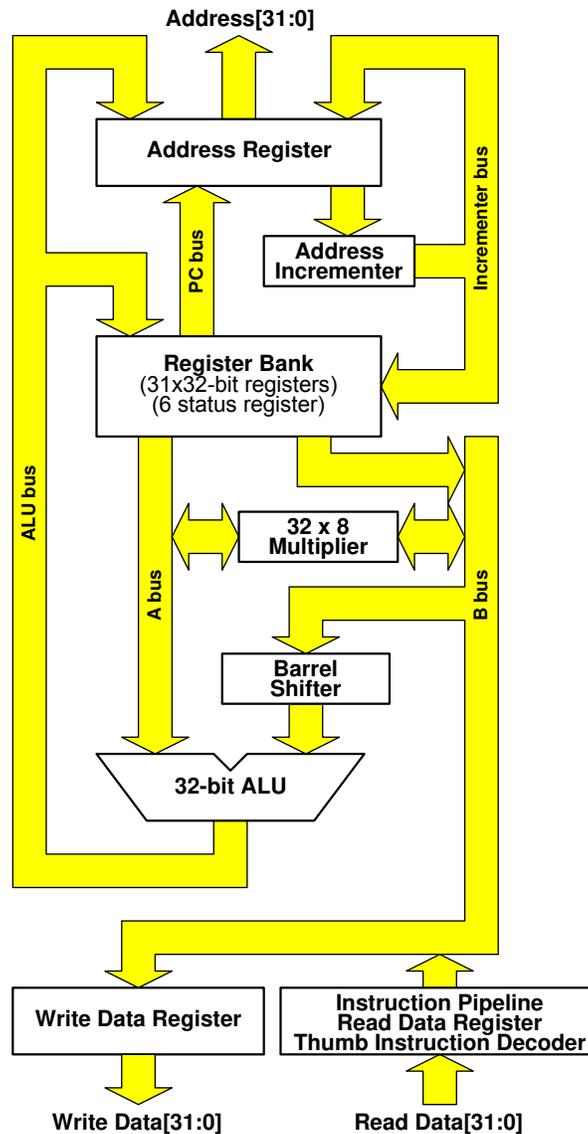
Figure 4. ARM7TDMI CPU Block Diagram.

ARM7TDMI Processor Overview

The A7S Configurable System-on-Chip family includes an embedded ARM7TDMI 32-bit RISC processor. The A7S is binary compatible with other ARM7-based devices. Figure 4 shows the major architectural features within the ARM7TDMI processor and the following text provides a brief overview. Please refer to the ARM7TDMI data sheet or *Resources* for more additional information.

Notable Architectural Features

Registers

The ARM7TDMI CPU has sixteen active 32-bit general-purpose registers at any given instance. There are a total of 31 such registers but some are only available during exception handling.

Arithmetic Logic Unit

The arithmetic logic unit (ALU) performs 32-bit arithmetic and logic instructions in a single clock cycle.

Barrel Shifter

The 32-bit barrel shifter allows a general shift operation to be combined with a general ALU operation in a single instruction that executes in a single clock cycle.

Hardware Multiplier

The ARM7TDMI processor includes a dedicated 32 x 8 hardware multiplier. Additionally, the multiplier supports multiply-accumulate functions, which are central to many digital signal processing (DSP) applications.

The performance of the multiplier depends on the data values and the type of data multiplied, as shown in Table 4. The multiplier terminates the instruction immediately upon computing the result, regardless of the data width.

Table 4. ARM7TDMI Multiplier Performance.

Multiplier Operation	Clock Cycles
32 x 32 = 32 Multiply two 32-bit values with a 32-bit result	2 to 5
32 x 32 = 64 Multiply two 32-bit values with a 64-bit result	3 to 6
32 x 32 + 32 = 32 Multiply two 32-bit values, add the result with a 32-bit value, producing a 32-bit result	3 to 6
32 x 32 + 64 = 64 Multiply two 32-bit values, add the result with a 64-bit value, producing a 64-bit result	4 to 7

Conditional Code Execution

Each ARM instruction is conditionally executed, based on the current status flags. The capability minimizes short branches, which might otherwise reduce system performance.

Three-Address Data Processing Instructions

The two source operand registers and the result register are independently specified, which aids performance and improves code density.

Thumb Instruction Set

The Thumb instruction set provides an extremely dense 16-bit representation of the most commonly used instructions. Thumb offers cost advantages for smaller systems and performance advantages in systems with 8-bit or 16-bit external memory subsystems.

CISC-like Instructions

Load and store multiple instructs allow an application to quickly and easily save and restore registers

Operating States

The ARM7TDMI processor provides two operating states. ***ARM state*** executes 32-bit, word-aligned ARM instructions, providing the full richness of the ARM instruction set. The alternate ***THUMB state*** operates with 16-bit, half-word aligned THUMB instructions, offering significant code size reductions. An application can switch between the two states providing the optimum mix of performance and code density.

Operating Modes

The ARM7TDMI processor supports seven different operating modes, as shown in Table 5. Mode changes may occur under software control or by external interrupts or exception processing. Most application programs execute in User mode. The non-user modes, known as privileged modes, are designed to service interrupts or exceptions, or to access protected system resources.

Table 5. ARM7TDMI Operating Modes.

Mode	Purpose
User (usr)	The normal ARM program execution state.
FIQ (fiq)	Designed to support a data transfer or channel processes.
IRQ (irq)	Used for general-purpose interrupt handling.
Supervisor (svc)	Protected mode for the operating system.
Abort (abt)	Entered after a data or instruction prefetch abort.
System (sys)	A privileged user mode for the operating system.
Undefined (und)	Entered when an undefined instruction is executed.

Registers

The ARM7TDMI has 37 registers, consisting of 31 general-purpose 32-bit registers and six status registers. However, not all registers are viewable at once. The visibility of a particular register depends on the processor state and operating mode.

ARM State Register Set

In ARM state, 16 general registers and one or two status registers are visible at any one time. In privileged (non-user) mode, various mode-specific banked registers become available. Table 6 shows which registers are available in each operating mode. Banked registers are shaded.

The ARM state register set contains 16 directly accessible registers, named R0 through R15. All of these, except R15, are general-purpose registers and may store either data or address values.

Additionally, there is a seventeenth register used to store status information, named CPSR (Current ***Program Status Register***).

FIQ mode supports seven banked registers mapped to R8 through R14 (R8_fiq through R14_fiq). In ARM state, many FIQ handlers do not need to save any registers. User, IRQ, Supervisor, Abort, and Undefined each have two banked registers mapped to R13 and R14, allowing each of these modes to have a private stack pointer and link registers.

R14 Used as the subroutine link register. Receives a copy of R15 when the Branch and Link (BL) instruction is executed. At all other times, R14 performs as a general-purpose register. Similarly, the corresponding banked registers— R14_svc, R14_irq, R14_fiq, R14_abt, and R14_und—hold the return values of R15 when interrupts or exceptions occur, or when a Branch and Link (BL) instruction is executed within an interrupt or exception handling routine.

R15 Holds the Program Counter (PC). In ARM state, R15 bits [1:0] are zero while R15 bits [31:2] contain the program counter (PC). In THUMB state, R15 bit 0 is zero and R15 bits [31:1] contain the PC.

CPSR The Current *Program Status Registers* (CPSR) contains condition code flags and the current mode bits.

ARM Thumb Extensions

The ARM7TDMI processor includes the Thumb extension to the 32-bit ARM architecture. The Thumb instruction set features a subset of the most commonly used 32-bit ARM instructions, which have been compressed into 16-bit wide op codes. When executed, these 16-bit instructions are decompressed transparently into full 32-bit ARM instructions, in real time, without degrading performance.

Designers can use both 16-bit Thumb and 32-bit ARM instructions sets in an application, providing an optimal mix of code density, performance, and instruction richness.

Table 6. Register organization in ARM state.

ARM State General Registers and Program Counter

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

ARM State Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

☐ indicates a banked register.

Thumb offers better code density than common 8- and 16-bit CISC/RISC controllers. Thumb application programs are merely a fraction of the code size of traditional 32-bit architectures. Consequently, program memory is smaller and hence cost reduced.

THUMB State Register Set

The THUMB state register set is a subset of the ARM state set. The programmer has direct access to eight general-purpose registers, named R0 through R7, plus the Program Counter (PC), a stack pointer register (SP), a link register (LR), and the Current Program Status Register (CPSR).

There are banked Stack Pointers, Link Registers, and Saved Process Status Registers (SPRs) for each privileged mode, as shown in Table 7.

In THUMB state, registers R8 through R15—called the Hi registers—are not part of the standard THUMB register set. However, the assembly-language programmer has limited access to the Hi registers and can use them for fast temporary storage.

Values are transferred from a Lo register (R0 through R8) to a Hi register, and *vice versa*, using special variants of the MOV instruction. Hi register values can also be compared against or added to Lo register values using the CMP and ADD instructions.

Program Status Registers

The ARM7TDMI processor contains a Current **Program Status Register** (CPSR) plus five Saved Program Status Registers (SPSRs) used by exception handlers. These registers

- Hold information about the most recently-performed ALU operation
- Control the enabling and disabling of interrupts
- Set the processor operating mode

The bottom 8 bits of a program status register, consisting of the I, F, T, and M[4:0] bits, are known collectively as the control bits. These bits change when an exception occurs.

Table 7. Register organization in THUMB state.
THUMB State General Registers and Program Counter

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
SP	SP_fiq	SP_svc	SP_abt	SP_irq	SP_und
LR	LR_fiq	LR_svc	LR_abt	LR_irq	LR_und
PC	PC	PC	PC	PC	PC

THUMB State Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

[ ] indicates a banked register.

If the processor is operating in a privileged mode, these bits may be manipulated by software.

When changing a program status register value, you must ensure that the reserved bits are not altered. Furthermore, the application program should not rely on the reserved bits containing a specific value.

Program Status Register Format

Bit	Description/Function
31	Negative, Less Than (N):
30	Zero (Z):
29	Carry, Borrow, or Extend (C):
28	Overflow (V):
27:8	Reserved
7	**IRQ Disable (I):** 0: IRQs enabled 1: IRQs disabled
6	**FIQ Disable (F):** 0: FIQs enabled 1: FIQs disabled
5	**THUMB State Enable (T):** 0: Operating in ARM state, using 32-bit instructions 1: Operating in THUMB state, using 16-bit instructions
4:0	**Operating Mode (M[4:0]):** These bits determine the processor's operating mode, as shown in Table 8. Only use values explicitly defined. These values are typically set by the real-time operating system.

Table 8. Processor Operating Mode Settings.

Mode	M[4:0]
User	10000
FIQ	10001
IRQ	10010
Supervisor	10011
Abort	10111
Undefined	11011
System	11111

Exception Vectors

When an exception occurs, such as a reset or an interrupt, the processor branches to a predefined vector address as shown in Table 9. The table shows the source or cause of the exception, the priority of each exception should they happen simultaneously, the CPU operating state entered by the exception, and the vector address.

Table 9. ARM7TDMI Exception Types and Vectors.

Exception Type	Priority	Cause/Source	CPU State	Vector Address
Fast Interrupt (FIQ)	3	FIQ interrupt to the Interrupt Controller	FIQ	0x1C
Interrupt (IRQ)	4	Any of the 15 IRQ interrupts to the Interrupt Controller	IRQ	0x18
Reserved				0x14
Data Abort	2	Memory access violations from the Protection Unit	Abort	0x10
Prefetch Abort	5	The CPU attempts to execute an invalid instruction or a `BKPT` instruction	Abort	0x0C
Software Interrupt	6	The CPU executes a `SWI` instruction	Supervisor	0x08
Undefined Instruction	6	The CPU executes a coprocessor instruction and no coprocessor responds	Undefined	0x04
Reset	1	Any reset condition	Supervisor	0x00

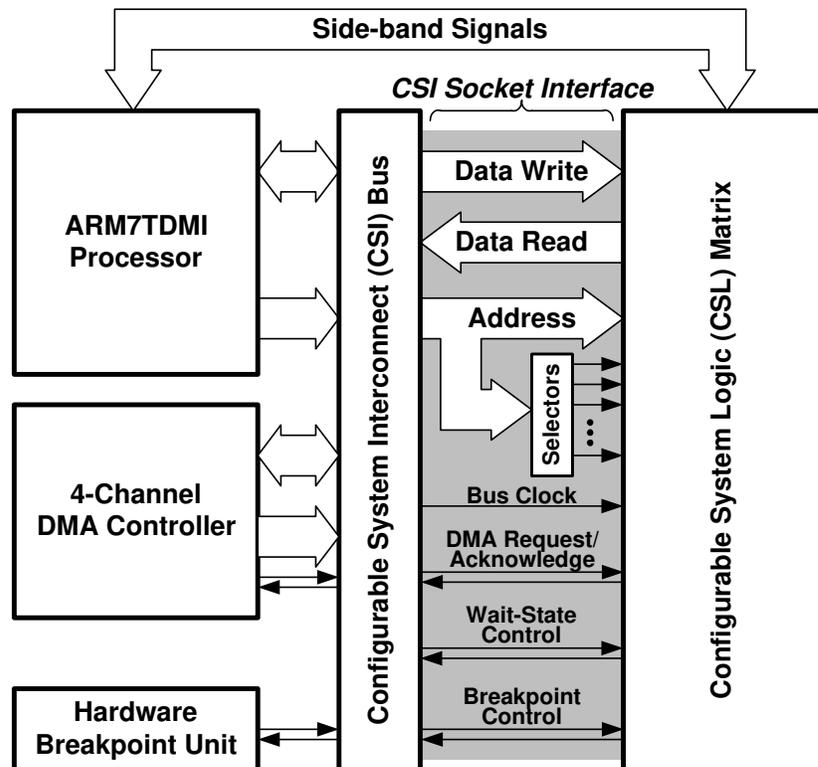**Figure 5.** **The Configurable System Interconnect (CSI) bus and the socket interface to user-defined logic functions in the CSL matrix.**

Configurable System Interconnect (CSI) Bus

The Configurable System Interconnect (CSI) bus, shown in Figure 5, bridges the processor with its peripherals including the Configurable System Logic (CSL) matrix.

The CSI bus socket provides a simple, synchronous interface to custom logic functions or peripherals implemented in the CSL matrix, as shown in Figure 6. The CSI bus interface socket consists of the following signals.

- A 32-bit write data port
- A 32-bit read data port, including a read enable signal and read return path from the CSL matrix onto the CSI system bus
- A 32-bit address port
- A set of address Selector functions to decode CSI bus transactions. The number of Selectors varies according to device size as shown in Table 10. The Selectors also optionally steer DMA request and acknowledge signals to and from CSL-based devices.
- The bus clock. All CSI bus events occur on the rising edge of the bus clock.
- Wait-state control and monitor signals
- Hardware breakpoint control and monitor signals

Data Write Port

After being granted the bus by the CSI bus arbiter, the current CSI bus master presents up to 32 bits of write data on the CSI socket during every active bus cycle.
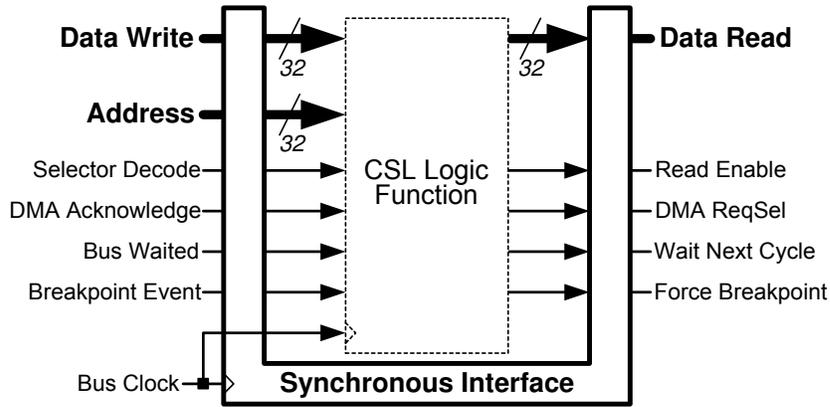
Figure 6. **The CSI bus socket represents a simple-to-use, synchronous interface to custom logic functions implemented in the CSL logic matrix.**

Data Read Port

An decoded or acknowledged CSL-based function asserts its read enable signal to present up to 32 bits of read data onto the CSI bus socket. All unselected CSL functions drive the read port with logic Low. The read data values from all CSL-based functions are logically OR-ed together before appearing on the CSI bus. Read data can be presented during every active bus cycle.

Byte, Half-Word, and Word Operations and Data Alignment

The CSI bus provides automatic handling for byte, half-word, and word operations. Figure 7 shows the supported data types and the associated byte lane alignment for each type. A 32-bit register can optionally be addressed as a single 32-bit word, two 16-bit half-words, or as four individual bytes.

Address Port

The bus master, granted the bus by the CSI bus arbiter, presents 32 bits of address on the CSI socket during every active bus cycle.

All 32 address bits appear the CSI interface socket. Typically, only a few if any of the address signals are used by functions in the CSL matrix. The actual address decoding for a bus transaction is usually performed using the on-chip address Selectors.
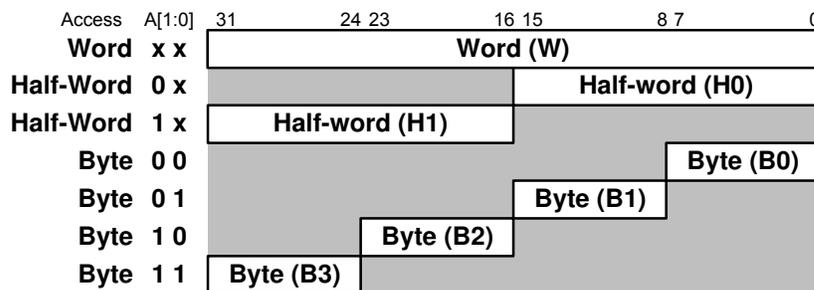

Figure 7. **CSI bus data types and byte-lane alignment. The FastChip byte-lane specifier is shown in parentheses.**

Address Selectors

The CSI socket interface practically eliminates the need to use any CSL matrix resources to decode bus transactions. One of the more important elements in the CSI socket interface is the programmable address decoder function, generically called a Selector. A Selector performs functions similar to a chip-select unit or an address decoder built using a PLD. As shown in Figure 8, a Selector decodes a range of addresses and produces separate read and write decode outputs, based on the bus address, and the size of the data transaction, commonly referred to as byte enables. By specifying the matching conditions, a Selector decodes a range of addresses, stretching from a single byte up to a 4Gbyte region in memory.
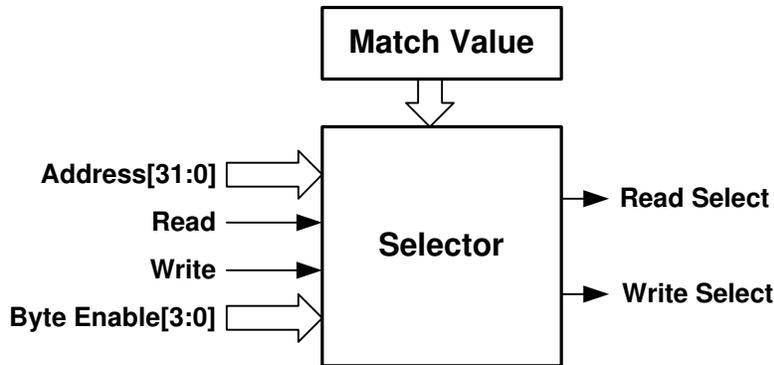
Figure 8. A simplified view of a Selector.

Address Selector Operation

A Selector detects transactions to a specified range of CSI bus addresses by decoding the full 32-bit CSI address bus. If a transaction targets its address range, the Selector asserts one of its read or write decode outputs coincident with the appearance of address and data on the bus socket, all synchronized to the bus clock. This approach dramatically simplifies the logic and timing of CSL logic functions attached to the CSI bus.

The number of available Selectors depends on the particular device. The number of Selectors grows with the increasing size of the CSL matrix. There is one Selector located above every column of sixteen CSL cells in a bank, as shown in Table 10 and Figure 16.

Table 10. Number of selectors by device.

Device	Selectors
TA7S04	32
TA7S20	128

Functionally, each Selector is similar to diagram shown in Figure 9. Each Selector contains two 32-bit registers that define the target address. The MATCH0 register defines which particular address bits match when the address bit is Low. The MATCH1 register defines which particular address bits match when the address bit is High. If the same bit location is set in both registers, then the corresponding address bit is a "don't care", matching regardless if the address bit is High or Low. For the A7, bits A[1:0] are typically programmed for "don't care" because most transactions are word-oriented and word aligned.

If all the address bits match the values defined in the MATCH0 and MATCH1 register, then the Selector further decodes read or write operations and the byte-lane alignment setting. During a read operation, if the address matches to the correct target address and

byte-lane alignment, then the Selector asserts its read-select output, RDSEL. Likewise, if the transfer is a write, then the Selector asserts its write-select output, WRSEL.

Address Specification

The MATCH0 and MATCH1 register values are automatically defined by the Triscend FastChip development system at design time and are loaded into the Selectors during device initialization. These register values are not changed by application software.

The addresses loaded into MATCH0 and MATCH1 registers and the byte-lane alignment are symbolically defined during hardware design by defining the following parameters.

- The **symbolic name** for the address range. This is the name used in application software to refer to the target address range.
- The **size** of the addressed range, which must be a power of two, ranging from '1' indicating a single byte to '4G' indicating a 4G byte region.
- The **byte alignment** for the selector, whether defined a word-wide, half-word-wide, or byte-wide access.
- The **address space** or spaces to which the selector should respond. For the A7, there is a single 32-bit linear data space.

> *The Triscend FastChip Development System allows you to specify only a **symbolic address**. The FastChip Generate utility allocates and assigns the physical address for you, based on the resource requirements that you define. The Generate utility also passes the symbolic and physical assignments to your compiler via a header file. By assigning only a symbolic address, all of your functions are re-useable and can be mixed and matched without address conflicts.*
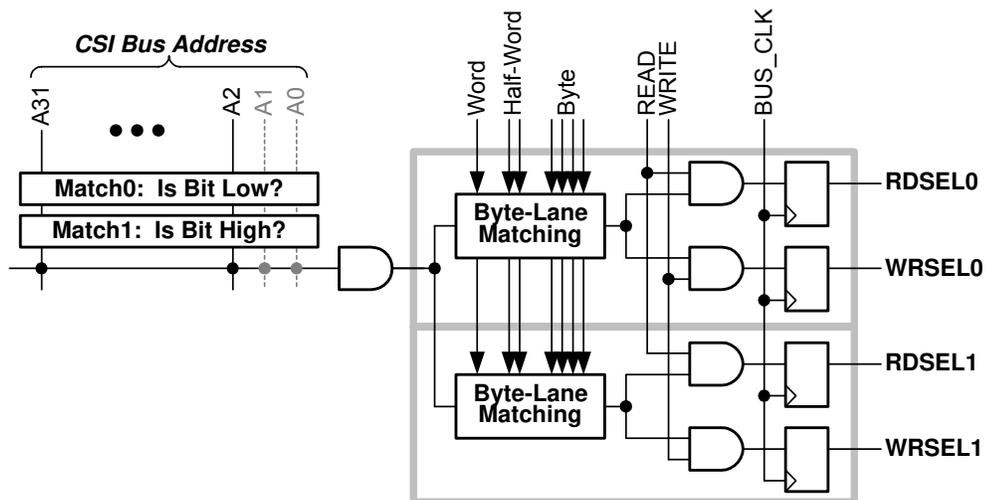>
> **FastChip**

Figure 9. The distributed address selector functions decode read and write transactions to a target address range. The Selectors eliminate the need to build decoding logic using CSL resources.
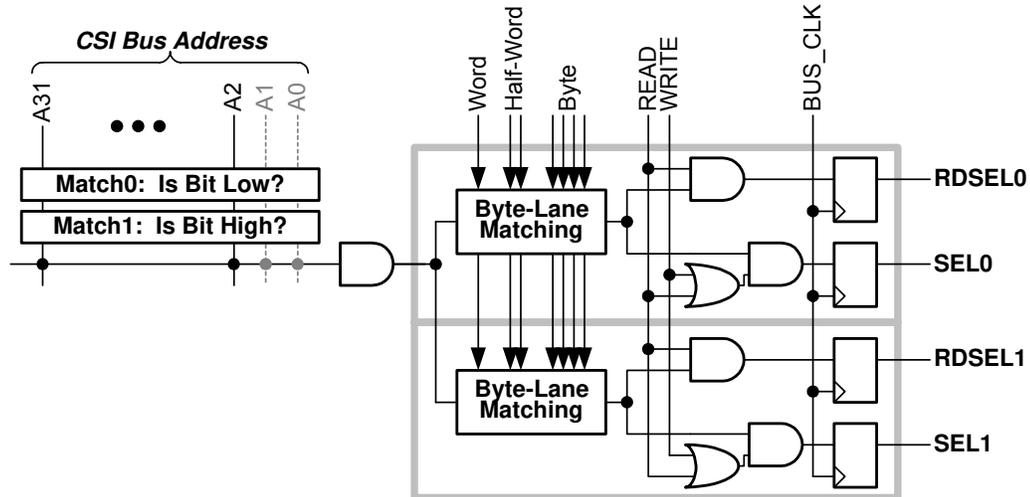
Figure 10. In chip select mode, an address selector decodes any read or write transaction to the target address range.

The Triscend FastChip development system examines these settings from all of the Selectors defined in the hardware design. It then allocates physical addresses to each Selector. The assigned addresses are defined and specified in a header file, used during software development.

Address Selector Modes

A Selector performs one of three potential functions as shown in Table 11.

Table 11. Address Selector Types.

Selector Modes	Ports	Function
Selector	RDSEL0 RDSEL1	Read decode
	WRSEL0 WRSEL1	Write decode
Chip Select	RDSEL0 RDSEL1	R/W- control
	SEL0 SEL1	Chip select
DMA Control Register	REQSEL	DMA request
	ACKSEL	DMA acknowledge
	DMASTAT	Early termination request
	DMACTL	Early termination acknowledge

Selector

A Selector separately decodes read and write operations to the target address range, as shown in Figure 9. The RDSEL[1:0] output indicates a read operation, the WRSEL[1:0] output indicates a write operation.

Chip Select

A Selector in chip select mode decodes any read or write transaction to the target address range. Figure 10 shows a functional drawing of a chip select function. The SEL[1:0] performs like a chip-select function, decoding both read and write transactions. The

RDSEL[1:0] output is asserted only during read operations and indicates the direction of a data transfer.

DMA Control Register

A Selector provides a relocatable control register for CSL-based I/O devices requiring DMA access. The DMA Control Register enables requests and steers the request and acknowledge signals to the selected DMA channel. See the *DMA Controller* section for more information.

Data Access and Selectors Required

In a 32-bit system, data is optionally accessed as a single 32-bit word, two 16-bit half-words, or as four individual bytes. Each Selector provides two write-enable outputs and two read-enable outputs to decode bus transactions to a specific address. The number of Selectors to decode a CSL-based function depends on the data width of the function and how it is accessed, as shown in Table 12.

For example, if the CSL-based function is a 32-bit wide register, then the CPU may access it as a word, a half-word, or a byte. If the application always accesses the CSL register as a word, never as a half-word or byte, then only half the Selector's outputs are required to decode the register. However, if the CPU accesses the 32-bit register as four bytes, then two Selectors are required.

Table 12. Number of Selectors required to access a CSL-based function of a given data width, depending on data access type.

Function Data Width	Data Access Type		
	Word	Half-Word	Byte
Word (32 bits)	½	1	2
Half-Word (16 bits)	½	½	1
Byte (8 bits)	½	½	½

CSI-to-CSL Bus Interface Design Example

Figure 11 illustrates how a 32-bit read/write register, implemented in the CSL logic, connects to the CSI bus socket. Physically, the 32-bit register requires 32 CSL cells. Because a CSL bank is just 16 CSL cells high, a 32-bit register requires two CSL banks, but is just one column wide. The actual physical implementation may be different, depending on the application.

The CPU might access the 32-bit register as a 32-bit word, as two 16-bit half-words, or as four individual bytes, as shown in Figure 12. Consequently, a 32-bit register may require up to two address Selectors, which provide four separate read and write byte-enables. The number of Selectors required depends on the data width of the CSL-based function and the desired data access types, as shown in Table 12.

The signals shown in Figure 12 are relative to the CSL side of the CSI bus socket. In this idealized series of transfers, the CPU performs various write and read transactions to the CSL-based register.

① The CPU writes 0xFFFFFFFF as a word-wide transfer to the data register. All four byte-enables are asserted by their respective Selectors. The data register captures the value 0xFFFFFFFF on the next clock edge.
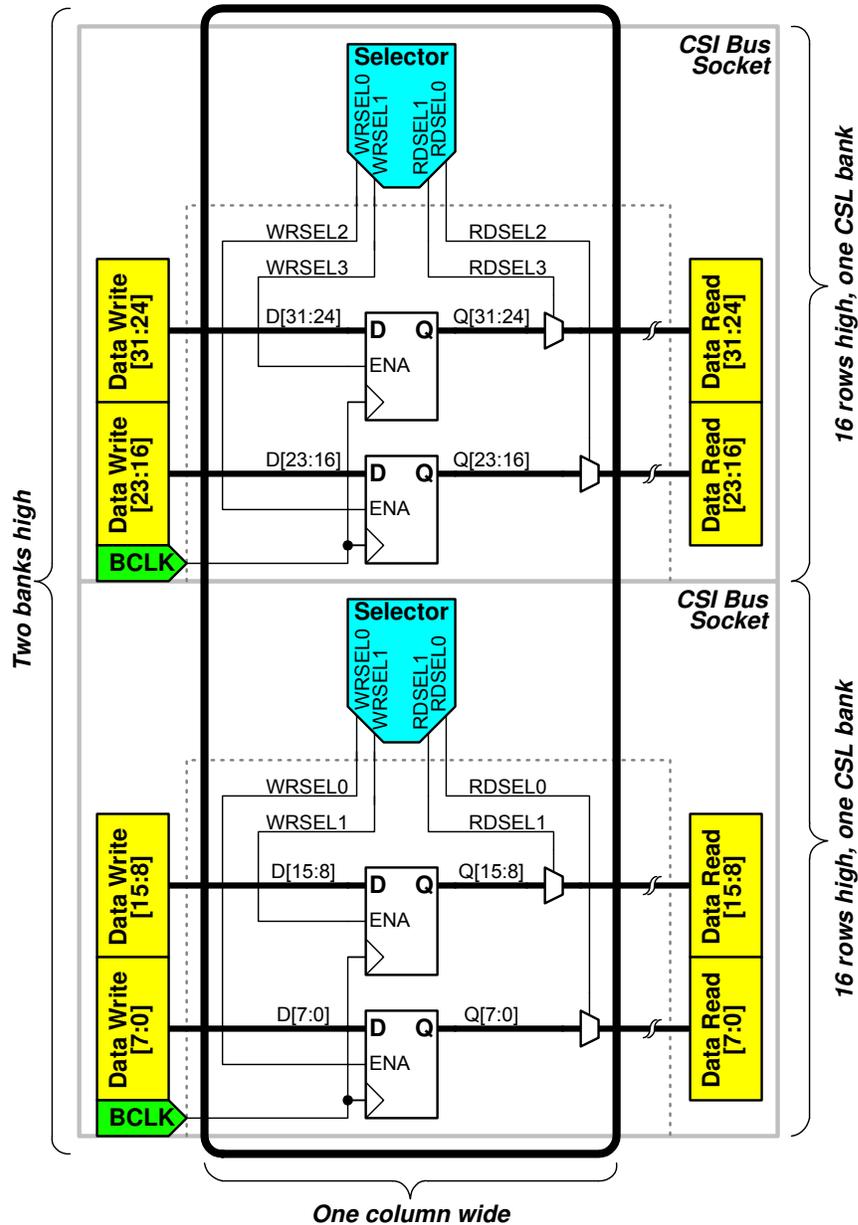
Figure 11. **A possible physical implementation of a 32-bit read/write data register connected to the CSI bus.**

② The CPU writes 0x0000 to the upper half-word of the data register. The CSI bus automatically duplicates the upper half-word onto the lower half-word of the CSI Data Write bus, making the bus value 0x00000000. Only the two byte-enables controlling the upper 16 bits of the register are asserted. The data register captures the data and the register output becomes 0x0000FFFF on the next clock edge.

③ The CPU writes 0x0000 to the lower half-word of the data register. As before, the CSI bus automatically duplicates the half-word on the CSI Data Write bus. Only the two byte-enables controlling the lower 16 bits of the register are asserted. The data register captures the data and the register output becomes 0x0000000 on the next clock edge.
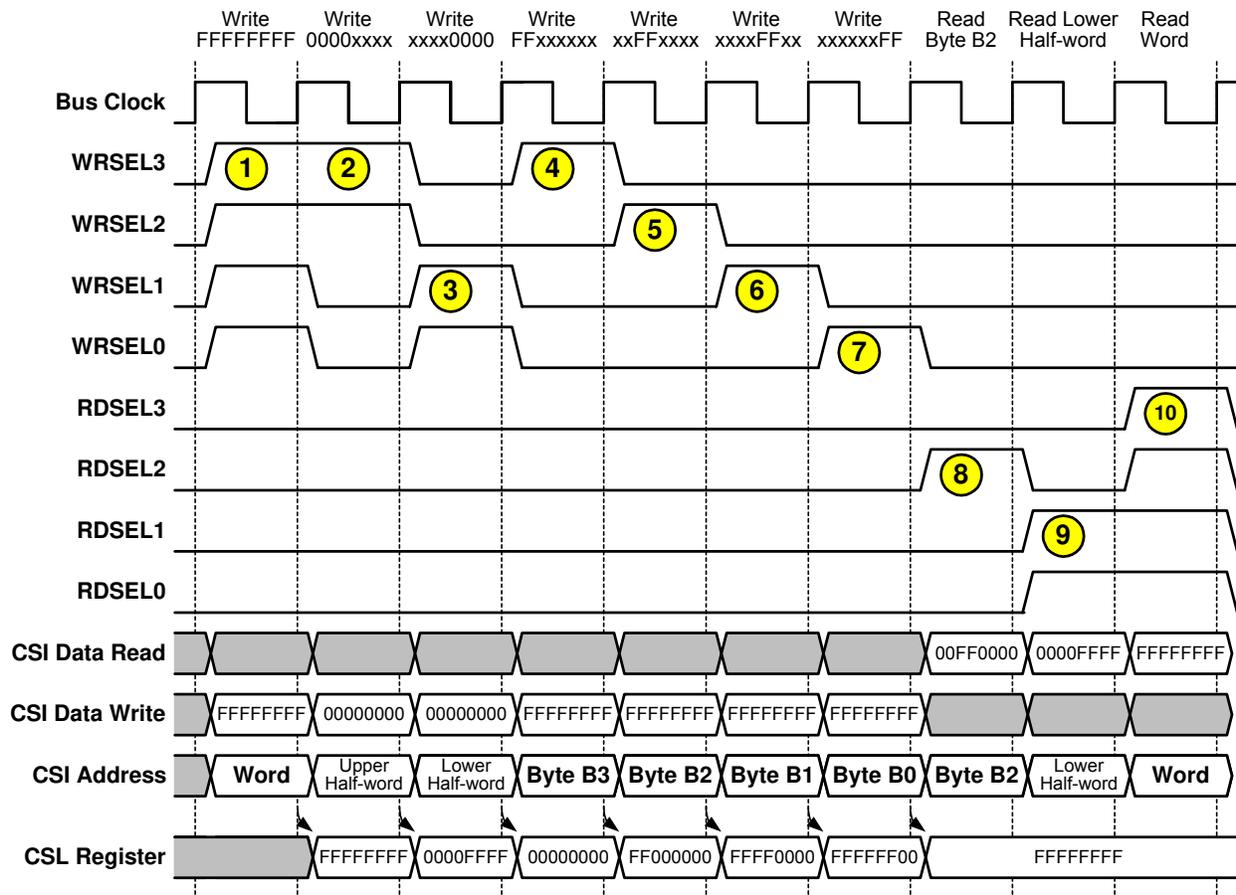
Figure 12. Idealized data transfers to a 32-bit register shown in Figure 11.

④ The CPU writes 0xFF to the upper-most byte, byte B3, of the data register. The CSI bus automatically duplicates the byte across all four byte lanes, making the CSI Data Write bus 0xFFFFFFFF. Only a single byte-enable is asserted, controlling the upper 8 bits of the register. The data register captures the data and the register output becomes 0xFF000000 on the next clock edge.

⑤ through ⑦ are similar to ④ except that different byte-enables are asserted. After ⑦, the register value becomes 0xFFFFFFFF on the next clock edge.

⑧ The CPU reads from byte B2. Only the RDSEL2 byte-enable is asserted, placing bits Q[23:16] onto the CSI Data Read bus. All other byte lanes are zero because they are not selected in this transaction. The value 0x00FF0000 appears on the CSI Data Read bus.

⑨ The CPU reads the lower half-word from the data register. The two byte-enables controlling the lower 16 bits of the register place bits Q[15:0] onto the CSI Data Read bus and the value 0x0000FFFF is captured on the next rising clock edge. All other byte lanes are zero.

⑩ The CPU reads the entire 32-bit data register as a word-wide transaction. All four byte-enables are active placing bits Q[31:0] onto the CSI Data Read bus. The 0xFFFFFFFF value is captured on the next rising clock edge.

Wait-State Monitor and Control Signals

The CSI socket interface includes signals to monitor and control wait-states on the internal system bus.

WAITED (output from bus socket)

The WAITED signal indicates that a wait-state was asserted during the previous CSI bus cycle. Though rarely used, this signal is typically used in FIFO control logic.

Initial Wait-State Insertion

Some CSL logic functions implemented in the CSL matrix may require wait-states, either because the CSL logic function handshakes with another asynchronous device or if the CSL logic function is too slow to respond in a single bus cycle.

If a CSL logic function requires *any* wait-states, then a Selector must assert the first wait-state. The Selector only asserts a wait-state if the system is accessing the Selector's target address space.

Should a CSL logic function require additional wait-states beyond the first wait-state asserted by the selector, then the CSL logic function inserts additional wait-states by asserting the WAITNEXT signal on the CSI socket interface.

> **NOTE:**
> *If a CSL-based function requires any CSI bus wait-states, then a Selector must assert the first wait-state cycle.*

WAITNEXT (input to bus socket)

If a CSL function requires more than one wait-state, then it inserts additional wait-states by asserting the WAITNEXT signal before the next rising clock edge on Bus Clock. Again, a Selector must always insert the first wait-state. When valid on a rising clock edge, the WAITNEXT signal causes a wait-state on the next bus cycle.

A CSL-based function may insert any number of wait-states. There is not built-in time-out mechanism. Functions that insert wait-states while waiting for an external event should always consider the case where the external event never happens and release the bus.

Breakpoint Event Monitor and Control Signals

The CSI socket interface includes signals to monitor and control hardware breakpoint events. These signals can be used to aid system-level debugging.

EVENT (output from bus socket)

CSL functions can monitor hardware breakpoint events using the EVENT signal. When EVENT is asserted, a hardware breakpoint event has occurred, either caused by the built-in hardware breakpoint unit or by another function in the CSL matrix that asserted the BREAK signal.

The EVENT signal allows a CSL-based function to freeze in conjunction with the remainder of the system. A function that uses Bus Clock as its clock source is automatically frozen during a breakpoint event.

BREAK (input to bus socket)

CSL functions can force a hardware breakpoint event by asserting the BREAK signal. The built-in hardware breakpoint unit typically only monitors transactions on the CSI bus. The BREAK signal allows CSL functions to interact with the hardware breakpoint unit.

For example, a CSL function could be monitoring a serial communications stream that rarely interacts with the system bus. Upon detecting a particular pattern or error condition, the CSL function could force a breakpoint event, stopping the system. The state of the system or CSL functions could then be monitored through the JTAG port.

CSI Bus Transactions

The following section provides some example CSI bus transactions, demonstrating the interaction of a CSL logic function and the CSI bus socket, including wait-states.

Data Write Transactions

During a data write transaction, the system sends data to a CSL logic function. The system presents both write data and address.

Figure 13 shows a single-cycle write transaction to a CSL logic function. Write data and address are presented on the CSI socket interface. The address is decoded using a Selector. If the transaction is to the Selector's address range, then the Selector asserts its WRSEL signal. The CSL function uses WRSEL to enable a register, as shown in Figure 11, and captures the write data on the next rising clock edge.
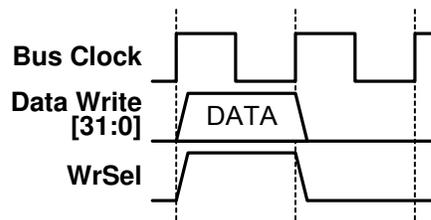
Figure 13. A single-cycle write transaction to a CSL logic function.

Figure 14 demonstrates a similar transaction, except that the CSL logic function requires two clock cycles to capture the write data. The CSL logic function's Selector is configured to assert the initial wait-state. When the Selector detects that the system is addressing it, it asserts its WRSEL signal and automatically asserts the initial wait-state.

During the wait-state, the bus master continues presenting write data and address and the Selector continues to assert its WRSEL output. The CSL function will capture the write data during the second bus cycle so it does not assert WAITNEXT. The WAITED signal indicates the wait-state inserted by the Selector during the first bus cycle. The transaction ends after the second bus cycle and the Selector de-asserts its WRSEL output.
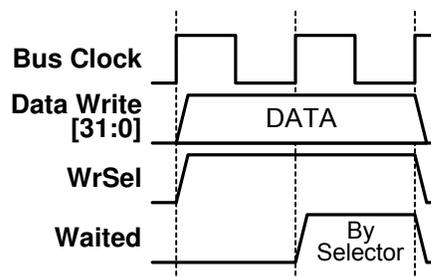
Figure 14. A two-cycle write transaction to a CSL logic function.

See the "Designing with Triscend Selectors" technical document within FastChip for additional information on creating custom logic functions that connect to the CSI bus.

Data Read Transactions

During a data read transaction, the system presents the read address and the targeted CSL function presents the data.

Figure 15 shows a single-cycle read transaction from a CSL function. The read address is presented on the CSI socket interface. The address is decoded using a Selector. If addressed, then the Selector asserts its RDSEL signal. The CSL function uses RDSEL to enable data onto the Data Read output port on the CSI socket, as shown in Figure 11.

Figure 15. A single-cycle read transaction from a CSL logic function.

Figure 16 demonstrates a similar transaction, except that the CSL logic function requires two clock cycles to provide the read data. The CSL logic function's Selector is configured to assert the initial wait-state. When the Selector detects that the system is addressing it, it asserts its RDSEL signal and automatically asserts the initial wait-state.

During the wait-state, the bus master continues presenting address and the Selector continues to assert its RDSEL output. The CSL function will present valid read data during the second bus cycle so it does not assert WAITNEXT. The WAITED signal indicates the wait-state inserted by the Selector during the first bus cycle. The transaction ends after the second bus cycle and the Selector de-asserts its RDSEL output.
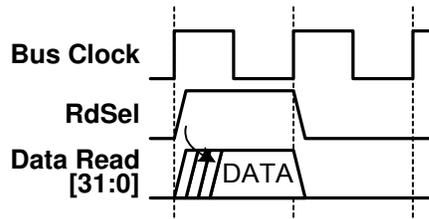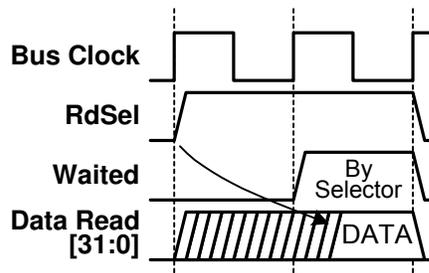

Figure 16. A two-cycle read transaction from a CSL logic function.

Using WAITNEXT during a transaction

There are four general rules for asserting a CSI bus wait-state, depending on the CSL function's response time.

1. If the CSL logic function can respond within a single bus cycle, the no wait-states are required.

2. If the CSL logic function can respond by the second bus cycle, then it associated Selector must insert a single wait-state.

3. If three or more bus cycles are required before the CSL logic function can respond, then the Selector is configured to insert the initial wait-state and the CSL function must insert additional wait-states using the WAITNEXT signal.

4. If any wait-states are required on one side of a transaction, either read or write, then the other side of the transaction requires at least one wait-state. The initial wait-state inserted by a Selector occurs on both read and write transactions.

Figure 17 shows an example transaction where the selected CSL function insert wait-states using the WAITNEXT signal. During the first bus cycle, the addressed CSL function's Selector asserts it RDSEL output. In this example, the function always requires at least one wait-state, so the Selector inserts the initial wait-state. The CSL function is not ready to respond in the first bus cycle, so the function asserts WAITNEXT, inserting a wait state during the next bus cycle.

During the second bus cycle, the system continues providing address and the Selector continues asserting RDSEL. The CSL function determines that it is ready to respond during the next bus cycle and de-asserts WAITNEXT. The WAITED signal shows the wait-state inserted by the Selector during the first bus cycle.

Finally, during the third bus cycle, the CSL logic function is ready to respond. The CSL logic function provides data on the Read Data port on the CSI socket. The WAITED signal shows the wait-state inserted when the CSL function asserted WAITNEXT. The wait-state occurred on the second bus cycle but WAITED reports the wait-state on the following bus cycle.
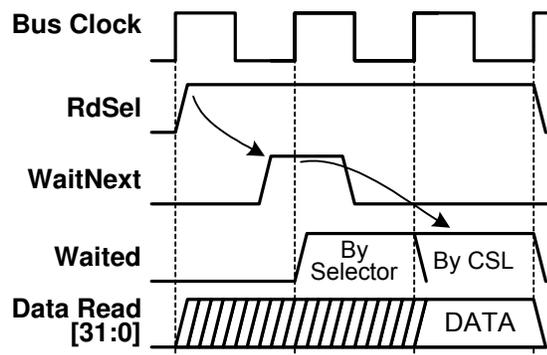
Figure 17. A multi-cycle transaction using WAITNEXT to insert wait-states.

CSI Bus Arbiter

The CSI bus arbiter schedules and manages traffic on the CSI bus.

Bus Masters

There are up to ten independent bus masters on the CSI bus, as shown below. Each DMA channel separately requests read and write transactions because these requests can sometimes be combined with other transfers on the CSI bus, as discussed below.

1. ARM7TDMI CPU
2. JTAG unit
3. DMA 0 Read
4. DMA 0 Write
5. DMA 1 Read
6. DMA 1 Write
7. DMA 2 Read
8. DMA 2 Write
9. DMA 3 Read
10. DMA 3 Write

Arbitration Scheme

All active bus masters arbitrate for the bus using a round-robin arbitration scheme with fixed prioritization. The arbitration priority rotates between the various active bus masters. The bus arbiter attempts to fully utilize the CSI bus bandwidth by intermixing non-conflicting transactions, as discussed below.

Simultaneous Transactions

The CSI bus arbiter can schedule two transactions during the same bus cycle provided that …

- Of the two bus requests, only one transfer is to a memory-mapped location. There is only one address bus.
- The two transfer requests are in opposite directions. In other words, one must be a read transfer, the other a write transfer. There are separate read and write busses.

Table 13 shows the type of traffic carried over the CSI bus and which portions of the CSI bus are used during each type of transaction. The conditions outlined above allow a variety of simultaneous transactions. For example, the CPU can write to a memory location while a DMA channel acknowledges a read request and gathers data from a CSL-based device.

Table 13. CSI Bus Transaction Types.

Transaction Type	CSI Bus		
	Address	**Read Data**	**Write Data**
CPU Read Cycle (over CSI-to-local bus bridge)	✓	✓	
CPU Write Cycle (over local-to-CSI bus bridge)	✓		✓
DMA Acknowledge Read Cycle (read data from device to DMA FIFO)		✓	
DMA Acknowledge Write Cycle (write from DMA FIFO to device)			✓
DMA Addressed Read Cycle (read from memory to DMA FIFO)	✓	✓	
DMA Addressed Write Cycle (write from DMA FIFO to memory)	✓		✓
JTAG Read Cycle	✓	✓	
JTAG Write Cycle	✓		✓

Sideband Signals

All of the signals on the CSI socket interface are designed to be processor independent. CSL logic functions designed using this interface can be re-used with future Triscend configurable system-on-chip families.

> **NOTE:** ✓
>
> *The SysRstN sideband signal is active-Low. All other sideband signals are active-High.*
>
> *The modem control sideband signals, DTR, RTS, CTS, DSR, DCD, and RI connect to either UART_0 or UART_1, but not both simultaneously. The selection is controlled by the UART's MODEM_EN_BIT.*

However, some signals are processor specific. The signals are called "sideband" signals. The sideband signals for the A7S family are shown in Table 14.

On most processors, these sideband signals would be assigned to dedicated pins. The A7S CSoC is more flexible and these signals are optionally routed to any available PIO pin or to logic in the CSL matrix.

Table 14. A7S Family Sideband Signals.

A7S Function	Direction	Signal	Active Polarity
Alternate clock, output of the phase-locked loop (PLL) multiplexer	PLL → CSL	ACLK	High
Phase-locked loop (PLL) lock indicator	PLL → CSL	Plock	High
System reset signal, **active-Low**. Active upon any system reset condition	Reset → CSL	SysRstN	Low
Serial port 0 transmit data	CSL → UART_0	SOUT0	High
Serial port 1 transmit data	CSL → UART_1	SOUT1	High
Serial port 0 baud rate clock	UART_0 → CSL	BDCLK0	High
Serial port 1 baud rate clock	UART_1 → CSL	BDCLK1	High
Modem Data Terminal Ready (DTR) signal	UART → CSL	DTR	High
Modem Request to Send (RTS) signal	UART → CSL	RTS	High
Application reset from CSL matrix	CSL → CPU	AppRst	High
ARM7 Fast Interrupt (FIQ) signal	CSL → CPU	FIQ	High
ARM7 Interrupt Request 2 (IRQ2)	CSL → CPU	IRQ2	High
ARM7 Interrupt Request 1 (IRQ1)	CSL → CPU	IRQ1	High
ARM7 Interrupt Request 0 (IRQ0)	CSL → CPU	IRQ0	High
Serial port 0 receive data	UART_0 → CSL	SIN1	High
Serial port 1 receive data	UART_1 → CSL	SIN2	High
Modem Clear to Send (CTS) signal	CSL → UART	CTS	High
Modem Data Set Ready (DSR) signal	CSL → UART	DSR	High
Data Carrier Detect (DCD)	CSL → UART	DCD	High
Modem Ring Indicator (RI)	CSL → UART	RI	High

FastChip

The sideband signals are available as global signal names within FastChip.
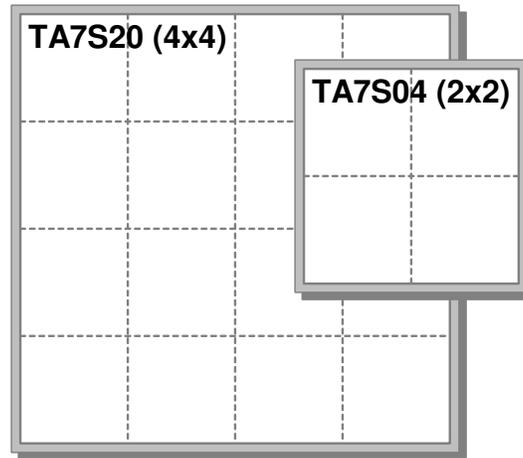
Figure 18. The two members of the A7S CSoC device family range in density from four CSL banks (448 CSL cells) up to 16 CSL banks (2,048 CSL cells).
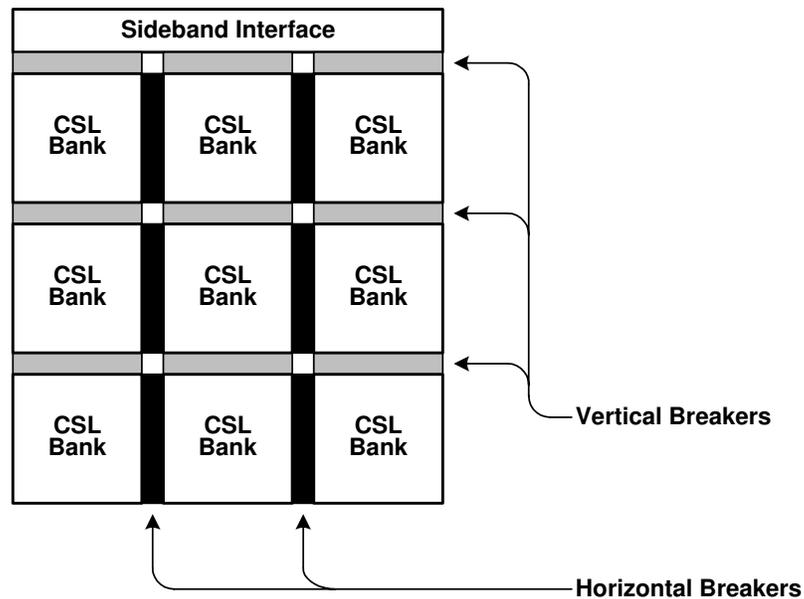
Figure 19. Vertical and horizontal breakers separate the individual CSL banks with Configurable System Interconnect (CSI) bus resources.

Configurable System Logic (CSL)

The Configurable System Logic (CSL) matrix provides flexible, programmable resources to build almost any digital logic function. Because it is intimately connected to the CSI bus, the CSL matrix is ideal for building any custom peripherals or logic functions required by the CPU. The matrix consists of multiple CSL banks. Each bank is a rectangular array of individual CSL cells.
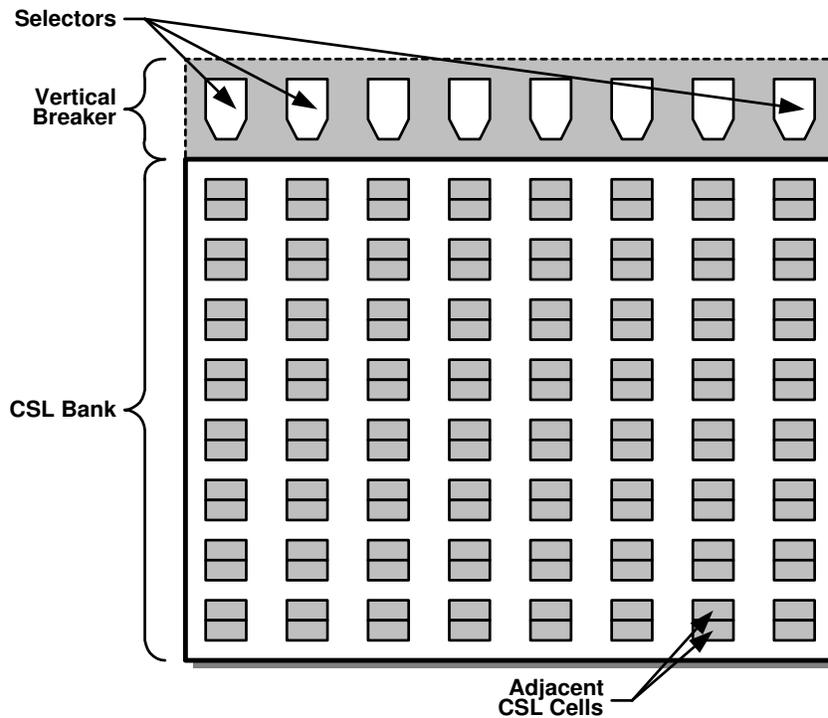
Figure 20. A CSL bank consists of multiple columns, each with 16 rows of CSL cells.

The number of CSL banks varies by part number. The highest-density A7S family device, the TA7S20, contains 16 CSL banks, arranged in a 4x4 array as shown in Figure 18 and Figure 20. The smallest member, the TA7S04, has just four CSL banks, arranged in a 2x2 array.

Table 15. CSL Banks by Device.

Device	CSL Cells Per Bank			Banks Per Device			Total CSL Cells
	Columns	Rows	CSL Cells/ Bank	Columns	Rows	Total Banks	
TA7S04	7	16	112	2	2	4	448
TA7S20	8	16	128	4	4	16	2,048

Vertical and horizontal breakers separate the individual CSL banks on a device, as shown in Figure 19. Vertical breakers appear at the top of every CSL bank. Horizontal breakers appear between adjacent columns of CSL banks. The breakers contain Configurable System Interconnect (CSI) bus resources. The horizontal breakers distribute CSI bus address signals to the CSL banks. The vertical breakers distribute the Selector input and output signals, breakpoint control signals, the global buffer signals, and the wait-state control signals. The CSI read data return path is also located in the vertical breakers.

Signals from one CSL bank can cross into other banks via the breakers, though crossing a breaker adds delay to the signal.

Sideband signals originate and terminate in resources along the top edge of the device.

Each CSL bank consists of multiple columns, each with 16 rows of CSL cells. Figure 20 shows the basic layout of the cells within a bank. Pairs of adjacent CSL cells share resources to build more complex cell functions. The Selectors, located in the vertical breaker above the bank, distribute any decoded address signals. There is one address Selector per column of 16 CSL cells.

Programmable interconnect surrounds the CSL cells. These programmable "wires" allow a signal originating from one CSL cell to communicate with one or more CSL cells, even those in other CSL banks. Likewise, signals to and from the CSI bus are distributed to and from individual CSL cells.

General-purpose Interconnect

The general-purpose interconnect, shown in Figure 22, distributes signals within a CSL bank. Metal lines of various lengths and purposes connect to individual CSL cells, to the horizontal and vertical breakers, and to the distributed array of routing matrices. Each routing matrix provides connections between the various lines entering or exiting the segment. The various interconnect resources are described below.

- **8 Short Segment lines** in each vertical and horizontal channel, connecting adjacent routing matrices.

- **8 Long Lines** in each vertical and horizontal channel. These long lines traverse the width or breadth of the CSL bank. The vertical long lines optionally distribute the outputs from the Selectors located in the vertical breaker. The horizontal long lines optionally distribute address signals from CSI bus.

- **Bus clock and 3 of 6 global buffer signal lines** in every vertical channel, as shown in Figure 21. The bus clock signal is distributed globally to all resources on an A7S CSoC device. Within a CSL bank, any three of the six global buffer signals are available.

- **A carry/cascade signal** between adjacent pairs of CSL cells, for faster arithmetic functions and for wide logic functions.
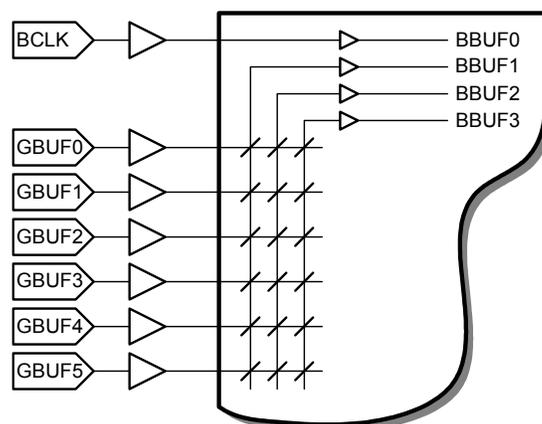

Figure 21. The bus clock signal and any 3 of the 6 global buffer signals are available within a CSL bank.
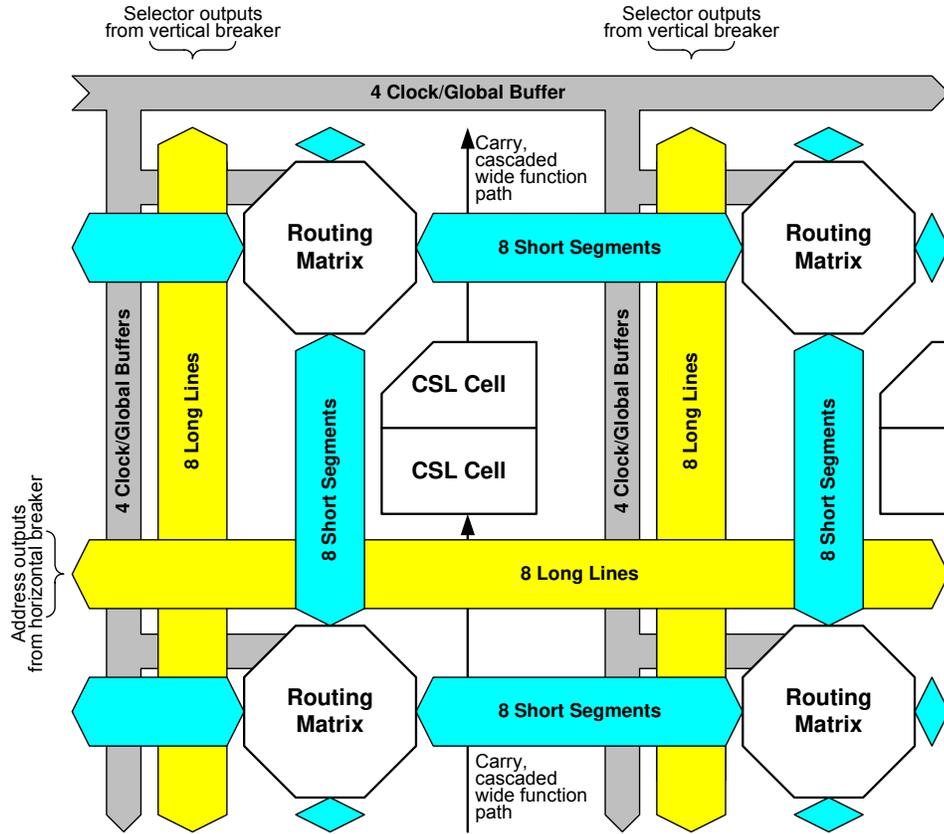
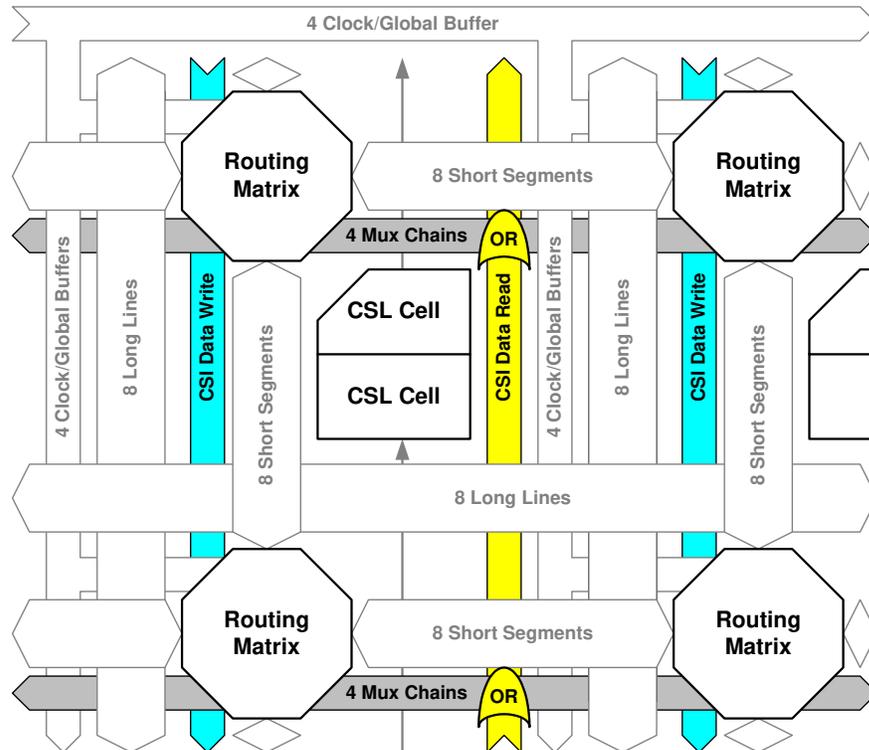**Figure 22. The general-purpose interconnect surrounds a pair of adjacent CSL cells.**

Figure 23. CSI bus write data is available at each routing matrix. Read data returns to the CSI bus.

CSI Bus Read and Write Data Distribution

Beyond the general-purpose signals, the programmable interconnect also distributed data signals to and from the CSI bus, as shown in Figure 23.

▪ **CSI Write Data** is accessible at every routing matrix, distributed throughout the CSL bank.

▪ **4 Multiplexer Chains** for distributing bi-directional data across a CSL bank. The multiplexer chains behave much like a bi-directional, three-state bus but avoids the potential data-contention problems and associated power consumption of a three-state bus because all signals are unidirectional.

▪ **CSI Read Data** paths gather the values of individual data lines from throughout the device. Ultimately, all the signals return to the CSI bus. The signals from individual bit lines are gathered via an OR-chain.

Signal-Flow Directional Preferences

Though the interconnect was designed to minimize directionality, there are few inherent preferences inspired by the architecture, as shown in Figure 24.
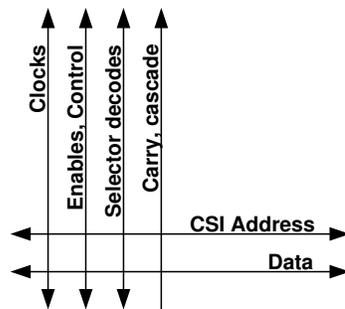
Figure 24. The interconnect architecture inspires a few signal-flow biases.

Clock signals prefer the vertical channels, either to use the four clock buffers signals available within a CSL bank or to use the direct connections between the vertical long lines and the clock inputs on CSL cell.

Likewise, other control signals and enable signals prefer vertical channels. Addresses decoded using Selectors also prefer vertical channels because vertical long lines distribute these signals from the vertical breakers.

Wide arithmetic functions or wide logic functions benefit from the built-in carry/wide interconnect, which prefers a vertical orientation, from bottom to top. Other orientations are possible, though with decreased performance.

Individual CSI bus address signals are distributed using the horizontal long lines and consequently prefer horizontal channels.

The multiplexer chains, designed to distribute bi-directional data, traverse a CSL bank horizontally. Consequently, data flow prefers the horizontal channels.
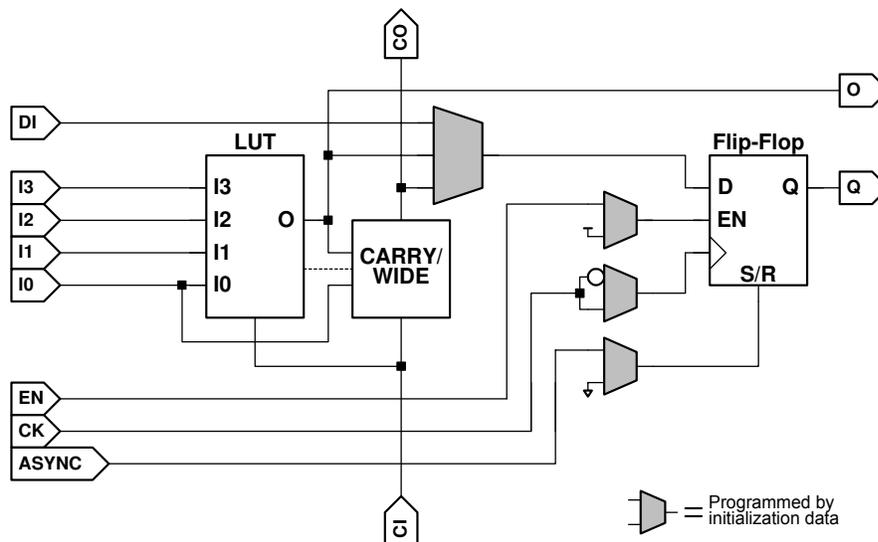
Figure 25. A basic CSL cell of both combinatorial and sequential logic.

CSL Cell Capabilities

A CSL cell, as shown in Figure 25, consists of a flip-flop plus combinatorial functions capable of performing various logic, arithmetic, or memory operations. Both these resources operate independently or in tandem, depending on the specific function implemented. An individual CSL cell is capable of implementing the following types of functions.

- Logic
- Arithmetic
- Memory
- Bus
- Sequential

Most logic functions are implemented using the CSL cell's four-input look-up table (LUT4). Any four-input, single-output function fits within a single four-input LUT, regardless of its logical complexity. A special mode allows two adjacent CSL cells to implement a five-input LUT (LUT5) and some logic functions of up to nine inputs.

The shaded multiplexers in Figure 25 represent static signal flow options, defined by the initialization data loaded into the device at power-on or after a Configuration Reset.

Logic Functions

In logic mode, an individual CSL cell performs a variety of combinatorial functions of the available inputs, as shown in Table 16. A single CSL cell performs any possible combinatorial function of four or less inputs, regardless of complexity. Likewise, two CSL cells working in tandem implement any possible function of five inputs. Two CSL cells also implement some functions of between six to nine inputs, with limitations. A sequence of four- or five-input functions, chained together, create wide gate functions of practically any width.

Arithmetic Functions

In arithmetic mode, a CSL cell performs simple arithmetic functions such as add, subtract, or multiply. Various functions, as shown in Table 17, provide common structures for building adders, subtracters, comparators, accumulators, incrementers, decrementers, binary counters, multipliers, and other arithmetic-based operations.

Memory Functions

In memory mode, a CSL cell performs various memory functions, including single- and dual-ported RAM, read-only memory (ROM), and an 8-bit serial-in/serial-out shift register. The small amount of RAM inside each CSL cell is ideal for building small register files and FIFOs. Should larger quantities of RAM be required, the A7S's large scratchpad RAM provides fast and efficient storage.

Single-Port RAM (RAM16X1, RAM32X1)

As a single-port RAM, a CSL cell provides

- Four address inputs for a 16x1 RAM block, five address inputs for a 32x1 block.
- A data input
- An active-High write enable input
- An invertible write clock input

The initial contents of the RAM can be pre-defined and loaded during initialization. However, the contents are *not* reloaded after a system reset. All write operations are synchronized to the clock input, simplifying the timing relationship of the data, address, and write-enable signal. Also, there are no hold time requirements for any of the RAM inputs after the active clock edge.

Read operations are asynchronous and depend only on the address inputs.

NOTE:	*The initial contents for RAMs, dual-port RAMs, and flip-flops are only loaded by a power-on reset or configuration reset condition. A system reset preserves the current state of these functions.*

Dual-Port RAM (RAMDUAL)

In dual-port RAM mode, two CSL cells provide true dual-port capabilities, supporting simultaneous read and write operations from both ports. As a dual-port RAM, two CSL cells offer

- Two, four-input address input ports
- Two data input ports
- Two active-High write enable input ports
- A single shared, invertible write clock input
- A daisy-chained error monitor that detects simultaneous write operations to the same address with different data

The initial contents of the RAM can be pre-defined and loaded during initialization. However, the contents are *not* reloaded after a system reset. All write operations are synchronized to the clock input, simplifying the timing relationship of the data, address, and write-enable signal. Also, there are no hold time requirements for any of the RAM inputs after the active clock edge.

Read operations are asynchronous and depend only on the address inputs.

Table 16. Logic functions implemented in a CSL cell.

Class	Function	CSL Cells	Application
Logic	f(4) Any 4-input logic function	1	Any combinatorial logic function with four or less inputs. Includes any mixture of AND, NAND, OR, NOR, XOR, XNOR, and INVERT.
	f(5) Any 5-input logic function	2	Any combinatorial logic function with five inputs. Includes any mixture of AND, NAND, OR, NOR, XOR, XNOR, and INVERT.
	Some functions of 6 to 9 inputs	2	Some combinatorial logic functions of between six to nine inputs. Includes a mixture of AND, NAND, OR, NOR, XOR, XNOR, and INVERT.
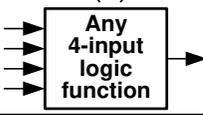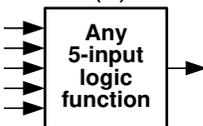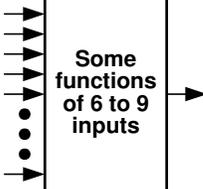
Table 17. Arithmetic functions implemented in a CSL cell.

Class	Function	CSL Cells	Application
Arithmetic	ADD	1	A one-bit full adder where SUM=X+Y. The output of the adder can be multiplexed with another input via the load (LD) input. Useful for building adders, incrementers, accumulators, and binary up counters.
	SUB	1	A one-bit full subtracter where SUM=X-Y. The output of the subtracter can be multiplexed with another input via the load (LD) input. Useful for building subtracters, decrementers, comparators, and binary down counters.
	ADDSUB	1	A one-bit full adder/subtracter where SUM=X±Y, depending on the SUB control. The SUB input controls whether the function is X+Y or X-Y. Useful for building adder/subtracters, incrementer/decrementers, and binary up/down counters.
	MULT	1	A one-bit multiply function. By combining MULT functions with ADD functions, large multipliers can be created.
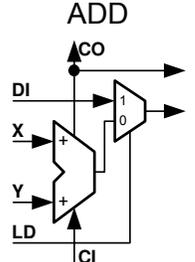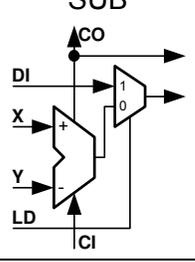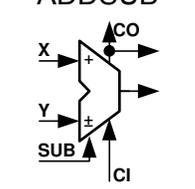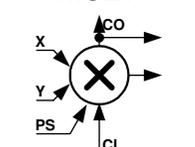
Table 18. Memory functions implemented in a CSL cell.

Class	Function	CSL Cells	Application
Memory	**RAM16x1** DI, WE, CK, A3, A2, A1, A0 → 16x1 RAM	1	A 16-deep by one-bit wide, clocked write, random-access memory (RAM).
	RAM32x1 DI, WE, CK, A4, A3, A2, A1, A0 → 32x1 RAM	2	A 32-deep by one-bit wide, clocked write, random-access memory (RAM).
	RAMDUAL DA, WEA, A3, A2, A1, A0, CK, DB, WEB, B3, B2, B1, B0 → 16x1 Dual-Port RAM, Contention Detection	2	A 16-deep by one-bit wide, clocked write, dual-port RAM supporting simultaneous read and write operations from both ports. Also includes write contention circuitry to detect simultaneous write operations to the same location with different data.
	ROM16x1 A3, A2, A1, A0 → 16x1 ROM	1	A 16-deep by one-bit wide read-only memory (ROM).
	ROM32x1 A4, A3, A2, A1, A0 → 32x1 ROM	2	A 32-deep by one-bit wide read-only memory (ROM).
	SHIFT8 SDO, DI, SH, EN, CK, A2, A1, A0, SDI	1	An 8-bit serial-in/serial-out shift register with selectable output tap and shift/load control.
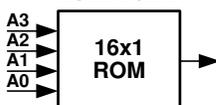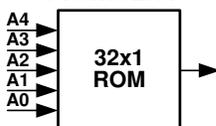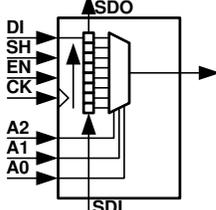
SUBJECT TO CHANGE

ROM (ROM16X1, ROM32X1)

The ROM function is available in two versions; a 16-deep by 1-bit wide ROM (ROM16X1) and a 32-deep by 1-bit ROM (ROM32X1)

A 16-deep ROM has four address lines to address the 16 memory locations. Likewise, a 32-deep ROM provides five address lines. The value on the address lines directly affects the output value.

A 16x1 ROM consumes a single CSL cell while a 32x1 requires two CSL cells operating in tandem.

The ROM's initial contents are specified in the user's design, loaded during initialization, and cannot be changed during operation.

8-bit Shift Register (SHIFT8)

Another operating mode offered by a CSL cell is an 8-bit, serial-in/serial-out shift register. Serial data arrives on the SDI input. When Low, the shift/load signal, SH, loads data on DI into the shift register location specified by the address lines, A[2:0]. All other shifting halts.

When SH is High, the SDI data is shifted in the first register location. Likewise, all subsequent register values are shifted one position toward the most-significant location (Location 7). The value in location 7 appears on the SDO output. The address lines, A[2:0] select the register location presented on the asynchronous output, O.

The enable signal, EN, disables all shifting and loading operations when Low.

Sequential Functions

Each CSL cell contains a 'D'-type flip-flop. The flip-flop provides the following controls

- An optional active-High clock enable input
- An invertible, edge-triggered clock input
- An optional asynchronous input to set or clear the flip-flop, defined at design time.

During the initialization process, each flip-flop is loaded with a '1' or '0' as defined in the design. This value is protected against potential spurious writes until the end of the initialization process.

**Table 19. Sequential Functionality
(no optional inversions used).**

Inputs				Output
D	EN	CK	ASYNC	Q
During initialization				INITV
X	X	X	1	ASYNCV
X	0	X	0*	Q
D	1*	↑	0*	D

INITV = Initial value loaded during initialization, defined at design time.

ASYNCV = Asynchronous preload value, defined at design time.

0* = Active Low, default value if left unconnected

1* = Active High, default value if left unconnected

Programmable Input/Output (PIO) Pins

Programmable input/output blocks (PIOs) interface external package pins to internal functions such as the processor, its peripherals, and the CSL matrix. Each PIO connects to a bonding pad, which may or may not attach to an external package pin, depending on the packaging option used. Each PIO can be configured as an input, an output, or a bi-directional signal, as shown in Figure 26.

The following list illustrates additional features available within each PIO pin.

- Weak pull-up or pull-down resistor, or bus follower
- Multiple voltage I/O support, 2.5 or 3.3 volts
- 3.3-volt PCI compliance
- Selectable output drive current
- 3.3-volt I/O tolerance, even from a 2.5-volt supply
- Optional input switching hysteresis
- Slew rate control
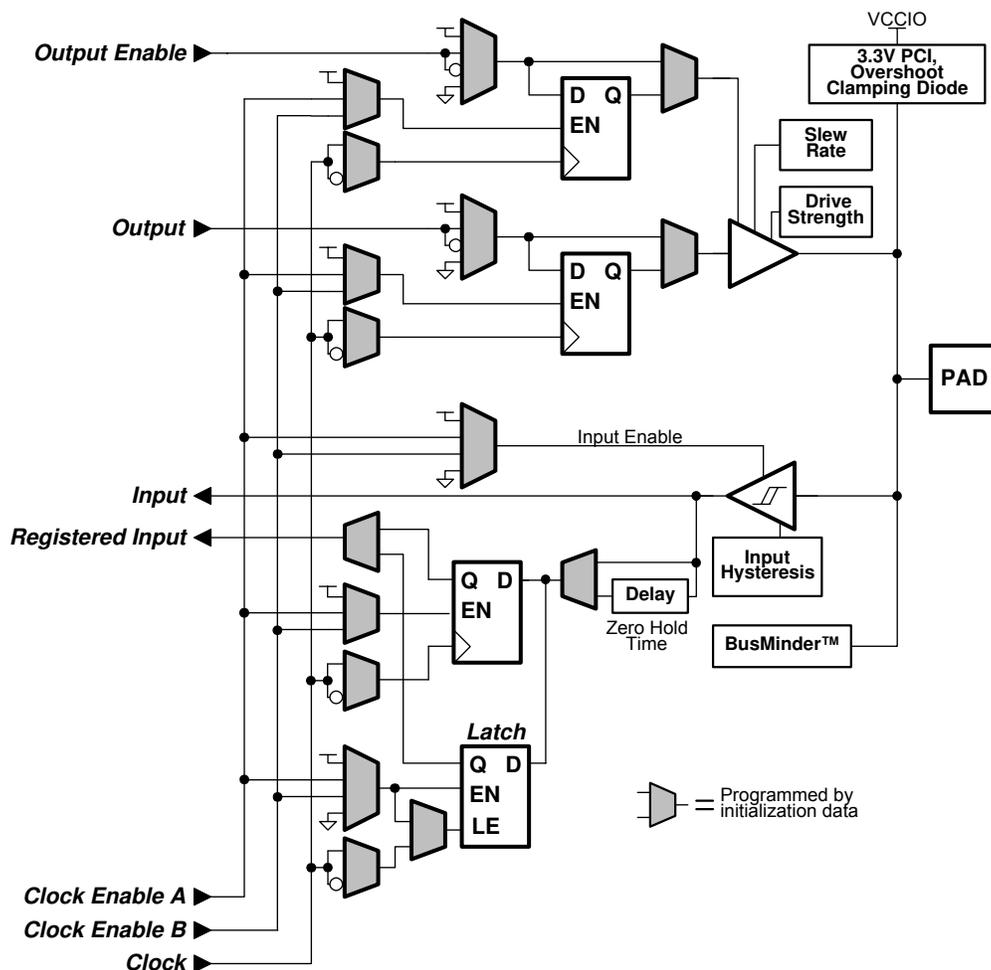
Figure 26. Programmable Input/Output block (PIO).

Creating a PIO Port for the Processor

The Triscend A7S offers nearly unparalleled I/O flexibility. Each of the configurable system-on-chip's PIO pins optionally connects to the processor via the CSI bus or operates independently in unassociated CSL logic functions.

Creating a memory-mapped I/O port for the processor requires CSI socket resources, including connections to the Data Read and Data Write busses, plus one or two address selectors.

Storage Elements

There are three storage elements in each PIO:

▪ An edge-triggered input flip-flop or a level-sensitive input latch

▪ An output flip-flop

▪ An output-enable flip-flop

The functionality of each storage element is defined in Table 20. Only the input register provides the latch option. Each storage element is a 'D'-type element and all share a common invertible clock control input. An individual flip-flop is either permanently enabled or selectively enabled using one of the two common clock-enable inputs.

Table 20. Flip-Flop/Input Latch Operation (no optional inversions used).

Mode	Inputs			Output
	D	EN	CK LE	Q
During initialization				INITV
Flip-flop	D	1*	↑	D
	X	X	0	Q
Input Latch (input only)	X	1*	1	Q
	D	1*	0	D
Both	X	0	X	Q

X = don't care

1* = logic High (default value)

↑ = rising clock edge (assuming no clock inversions)

INITV = preloaded initialization value defined in user's design

The polarity of the common clock signal is individual controlled for each flip-flop within a PIO.

A user-defined initial value of a register is loaded during initialization.

PIO Input Side

Input signals flow into the device through two possible paths. One path is a direct logical input while the other is a registered input, programmed as either an edge-triggered flip-flop or a level-sensitive latch.

Optional Input Switching Hysteresis

Each PIO input has optional input hysteresis. When enabled, 0.05•VCCIO to 0.2•VCCIO of hysteresis, centered around the input switching voltage without hysteresis.

Registered Input

Unlike the other storage elements, the input register can be configured either as an edge-triggered flip-flop or as a level-sensitive latch as shown in Table 20.

Guaranteed Zero Hold Time on Input Register

The data input to the input register is optionally delayed by several nanoseconds to guarantee that there is no positive hold time requirements. With the delay enabled, the setup time of the input flip-flop is increased to negate the internal clock distribution delay. This guarantees that the input register hold time is always zero or negative, never a positive hold time.

The delay guarantees a zero hold-time with respect to the clock provides by the bus clock buffer.

PIO Output Side

Output signals can be optionally inverted within the PIO, and can pass directly to the pad or be stored in an edge-triggered output flip-flop.

A logical Low on the Output-Enable signal forces the output into a high-impedance (Hi-Z) state, as shown in Table 21. Consequently, a PIO functions as a three-state output or bi-directional I/O. Conversely, a logical High enables the output buffer. Statically defined by initialization, the Output and Output-Enable signals are optionally inverted. The polarity of these signals is independently configured for each PIO. In addition, each can be internally tied High or Low independently.

Table 21. Output Buffer Operation
(no optional inversions used).

Inputs		Output
O	OE	PAD
X	0*	Hi-Z (floating)
0	1*	0
1	1*	1

X = don't care

O* = logic Low (default value)

1* = logic High (default value, if OE is unconnected but O is connected)

The polarity of the O and OE ports is individual controlled within each PIO.

The outputs on each PIO are full CMOS outputs. The switching threshold is a product of the I/O supply voltage (VCCIO).

An output can be configured as open-drain (open-collector), as shown in Figure 27, by tying the output path to ground and driving the output-enabled signal, OE. However, the voltage applied to the pin should never exceed the maximum values defined for VCCIO.
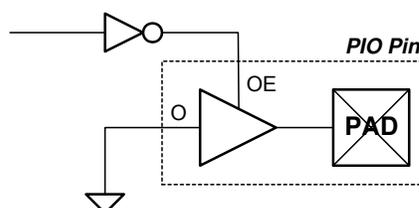
Figure 27. On open-drain (open-collector) output using a PIO pin.

Selectable Slew Rate Control

Each output buffer has an optional slew-rate control, which reduces noise generation and ground bounce. The slew-rate control provides a tradeoff between low-noise and maximum performance. The fast slew-rate should be used for speed critical outputs in systems that are adequately protected against noise.

TCH305-0001-002 — 43 — SUBJECT TO CHANGE

Selectable Output Drive Current

Each PIO has selectable output drive current, independent of the slew-rate control. The output drive options are shown in Table 22. Reduced drive current results in lower power consumption, lower EMI emissions, and lower ground bounce.

Table 22. PIO Output Drive Current.

Drive Strength	Drive Low (IOL)	Drive High (IOH)
Low Current	4 mA	2 mA
High Current	16 mA	8 mA

For even higher drive current, two or more PIO pins can be ganged together. However, watch out for differences in routing delays between the pins as this may cause contention.

Other PIO Options

There are a number of other programmable options available for PIO blocks.

BusMinder™

The BusMinder feature allows each PIO to have an optional

- Pull-up resistor (pulls un-driven inputs High)
- Pull-down resistor (pulls un-driven inputs Low)
- Weak follower (forces un-driven inputs to the last value that appeared on the bus)

Triscend A7S configurable system-on-chip devices are fabricated on leading-edge CMOS processes. Like all CMOS devices, input pins should never be left floating. An unused input might float unless tied to High or Low. In addition, an input connected to a bi-directional bus might float if the bus is three-stated (high impedance).

> *Set the electrical properties on one or more PIO pins using FastChip's I/O Editor. From the I/O Editor, you can specify the power-down characteristics of each pin, its drive strength, and other options.*
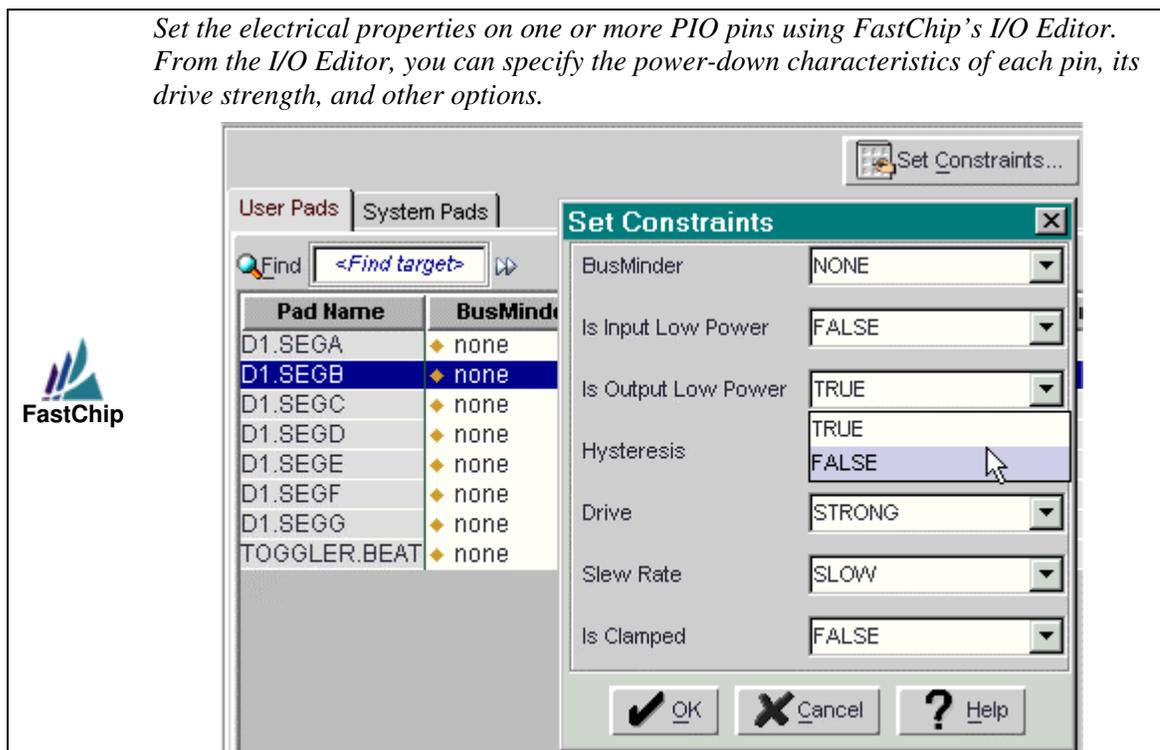
The BusMinder's programmable pull-up/pull-down resistor and weak follower are useful for tying unused or floating pins High or Low to minimize power consumption and reduce noise sensitivity.

The configurable pull-up resistor is a p-channel transistor that pulls to VCC. The configurable pull-down resistor is an n-channel transistor that pulls to Ground. The value of these resistors is between 25 kΩ to 100 kΩ. This high resistance value makes them unsuitable as wired-AND pull-up resistors, so an external resistor is required in these cases.

The pull-up resistors for PIOs are active during the initialization process. This pulls all as-yet-unprogrammed PIOs High, preventing them from floating. Devices connected to the PIO see a logic High. Other devices driving into the PIO can easily overdrive the weak pull-up resistor.

After initialization, voltage levels of unused pads—both bonded or unbonded—must be valid logic levels to reduce noise sensitivity and avoid excess current. Therefore, by default, unused pads are configured with the internal pull-up resistor active.

The weak follower is used on PIOs that connect to a bi-directional bus. Instead of pulling the floating input High or Low, the weak follower remembers the last value that appeared on the bus before the bus signal was three-stated.

3.3-Volt PCI Clamping Diode

3.3V PCI compliance requires a clamping diode to VCC, which is optionally available inside every PIO pin. In other low voltage applications where 3.3V-tolerance is not required, the VCC clamping diode can be used to limit the overshoot. During configuration, the VCC clamping diode is disabled to avoid any inadvertent current flow from pad to the power supply providing VCCIO.

Low-Power Mode

The Triscend A7S device supports a low-power mode that dramatically reduces system power consumption. While in power-down mode, each PIO pin can optionally be configured for low-power operation or to remain active during power-down. The PD_IO_EN_BIT must be set in the *Power Down Control Register* before entering power-down mode, which allows all PIOs to enter their assigned low-power configuration.

Output Options

If enabled for low-power operation, a PIO output is forced into a high-impedance state (disabled) during power-down mode. The BusMinder function switches to weak follower mode, regardless if configured for a pull-up or pull-down resistor or left floating. The weak follower function keeps the PIO at the voltage level last applied to the pin. This helps to reduce overall power consumption.

Input Options

Likewise, a PIO input can be enabled for low-power operation. A PIO input is optionally forced High during power-down mode.

JTAG Support

Embedded logic attached to the PIOs contains test structures compatible with IEEE Standard 1149.1 for boundary-scan testing, permitting easy chip and board-level testing.

Default, Unconfigured State

During device configuration, both the output buffer and input buffer are disabled to reduce power consumption and inadvertent switching. The input data is pulled high and the VCC

clamping diode is disabled. The weak pull-up stays active during configuration to prevent the pin from floating.

Before configuration, all PIO pins function via the JTAG interface but with the following default conditions:

- The input hysteresis is turned on
- The output drive is 4 mA
- The BusMinder is a pull-up resistor
- The clamping diode is disabled

Default, Configured State

All unused but configured PIO blocks are programmed as inputs with the soft BusMinder's pull-up resistor enabled. This prevents unused PIO pins from floating.

Electro-static Discharge (ESD) Protection

Each PIO has built-in ESD protection capable of protecting against a minimum discharge of 2,000 volts, using the human body model.

Multi-Voltage Support

Each A7S I/O buffer is 3.3-volt tolerant for multiple-voltage applications with a mixture of 3.3-volt devices and 2.5-volt devices. Independent of the power supply voltage to VCCIO, each I/O pin sustains up to 3.6 volts without reliability concerns. Special circuit techniques prevent the 3.3-volt voltage from forcing excessive current into the 2.5-volt VCCIO.

Table 23 shows the voltage standards supported when VCCIO is connected to 3.3 volts or 2.5 volts. When connected to 3.3 volts, each PIO supports a variety of standards, including TTL. At 2.5 volts, each PIO supports only 2.5V CMOS signaling levels.

Table 23. I/O Signaling Standard Compliance.

VCC	VCCIO	Compatible Standards
2.5 V	3.3 V	TTL, LVTTL 3.3V PCI 3.3V CMOS
	2.5 V	2.5V CMOS
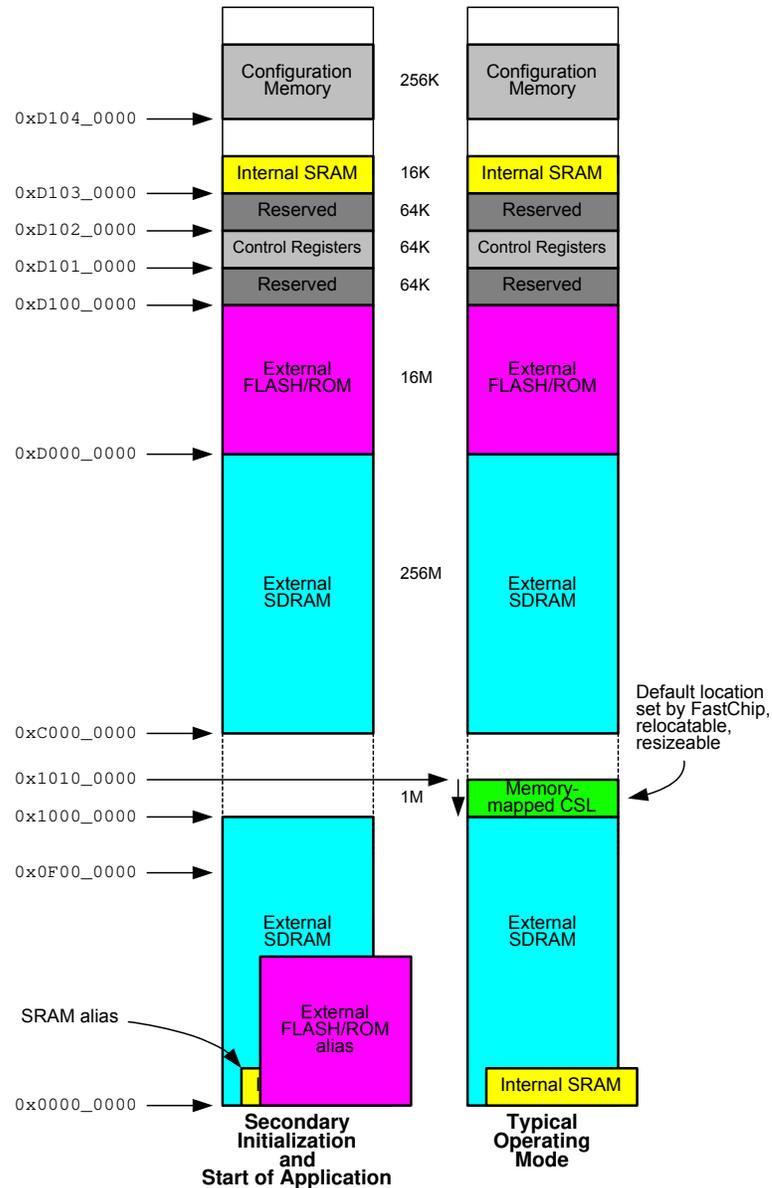
Figure 28. A7S System Memory Map.

System Memory Map

An A7S-based system supports a 32-bit or 4G byte address space. Some of the 4G bytes are already allocated to A7S internal resources. Table 24 summarizes all those resources and their respective memory space allocation.

All of the resources are mapped to the bottom of the memory space. The Flash, the internal scratchpad SRAM and the SDRAM spaces have one possible additional alias at address 0. These aliases are present by default following a Configuration Reset. They overlay each other with the Flash having the highest priority, followed by the internal scratchpad SRAM and finally the SDRAM.

47

Table 24. A7S Memory-Mapped Resources.

Resource	Allocated Space	Base Address
Control Registers (CRU)	64K bytes	Fixed
Configuration Memory	256K bytes	Fixed
Memory-mapped CSL Functions	1M bytes	Relocatable, resizable in FastChip
Scratch Pad	16K bytes	Re-locatable with alias at 0
External Flash	16M bytes	Fixed with alias at 0
External Dynamic Memory	256M bytes	Fixed with alias at 0

Figure 28 illustrates the memory map in two different phases of operation—just after secondary initialization completes and a more typical normal operating mode, where the Flash alias is disabled.

The external Flash alias, starting at address 0, can be disabled by the RTOS via the ***Clear Reset Memory Map Register***. However, before the RTOS disables the Flash, it must branch to the Flash image at the top of memory. The internal scratchpad SRAM alias is now available at the bottom of memory. The user application can copy all timing critical code to the scratchpad RAM including functions such as the interrupt vector table and the fast interrupt service routine.

For some applications where the Flash must reside at the bottom of memory, the scratchpad RAM can be overlaid over the Flash alias at the bottom of memory by changing its overlay priority.

All the re-mapping control is performed through registers and is explained in the Remap and Pause Registers section.

A complete memory map, minus any user-created functions, is available in ***Appendix A***.

Memory-Mapped CSL Functions

This region is where CSL functions containing memory-mapped registers are located. The region is defined in the FastChip address allocation file. By default, the region is 1M byte in size, starting at address 0x1000_0000, and allocated down toward the bottom of memory. This region in resizable and relocatable elsewhere in the system memory map. The default address allocation settings are shown below.

```
ALLOCATE START=0x1000_0000 SIZE=1M DIRECTION=DOWN;
```

Cache and Memory Protection Support

The cache, cache controller, and protection unit provide increased system performance as well as provide basic memory protection capabilities.

The cache controller/protection unit includes eight control registers. The control registers are normally accessed through the coprocessor interface, using `MCR` and `MRC` instructions to CP15. **The processor must be in privileged mode to access these registers.**

Cache Description

The cache is an 8Kbyte mixed instruction/data 4-way set associative cache. It holds 2K words, divided into 512 lines, with 4 words per line. The cache is transparent to software.

The cache is a write-through cache. During write operations, data is always written to its destination. If the data is present in the cache, then the cache is updated as well. The CPU interface unit provides a write buffer for increased performance.

> **NOTE:** *There are two write buffers from the CPU, one to external memory, the other to the CSI bus. Every CPU write operation is buffered. When clearing interrupts and leaving an interrupt service routing (ISR), flush the write buffer by reading from any CSI location.*
>
> *The Triscend driver library already handles this situation.*

The cache is setup through the protection unit registers by defining cacheable regions.

The cache is not recoverable as regular SRAM, although 16K bytes of extra on-chip scratchpad are available for that purpose. All applications should use the cache because it increases overall performance and decreases power consumption.

Only code residing in the external Flash or SDRAM, as defined by the memory map shown in Figure 28, can be cached by the system. The content of any additional external memories connected to additional chip select lines generated in the CSL cannot be cached. However, if the contents of the auxiliary memory are copied to SDRAM, then the contents can be cached.

Using the Cache

The cache is enabled through the coprocessor #15. The following sequence of ARM7TDMI assembly language instruction gives an example of how to enable the cache. The example below defines the first 4K of Flash as cacheable.

Example:

```
; Value for memory area definition (base pointer + size)
LDR R0, 0xd0000017

; Value for cacheable area #0 enable
LDR R1, 0x00000001

; Value for cache enable
LDR R2, 0x00000004

; Setup first 4K of Flash as cacheable
MCR p15, R0, c6, c0, 0

; Enable cacheable area #0
MCR p15, R1, c2, c0, 0

; Enable cache
MCR p15, R2, c1, c0, 0
```

Clearing the cache enable bit is not enough to disable the cache. It first must be invalidated. The next sequence of instruction shows the proper procedure to disable the cache.

Example:

```
; Value for cache disable
LDR R0, 0x00000000

; Enable cache
MCR p15, R0, c1, c0, 0

; Invalidate cache
MCR p15, R0, c7, c0, 0
```

Protection Unit

The Protection Unit controls all ARM7TDMI memory accesses with up to eight separate memory areas, as shown in Figure 29. These memory areas are named in order of priority with Area 7 having the highest priority, and Area 0 having the lowest priority. The Pro-

tection Unit allows the memory to be partitioned and individual attributes to be set for each region. With the Protection Unit enabled, any memory access that falls outside the range of all eight memory areas is aborted.

NOTE:

*Program the Protection Unit with valid protection regions **before** enabling the Protection Unit.*

If the memory regions are not defined prior to enabling the Protection Unit, the ARM processor can enter a state that is recoverable only through a reset. When the Protection Unit is disabled, all accesses—instruction fetches or data reads and writes—are non-cacheable.

The instruction and data address space may be partitioned into a maximum of eight regions. Each region is specified by the following attributes:

- Base address
- Region size
- Cache configuration (cacheable or not)
- Access permissions

If two or more regions overlap, then any attributes conflicts are resolved through a priority scheme; memory area 7 has the highest priority and area 0 has the lowest.

NOTE:

Only the contents of external Flash or SDRAM memory are cacheable. There is no need to cache internal SRAM because the CPU can execute directly from internal SRAM without wait-states.
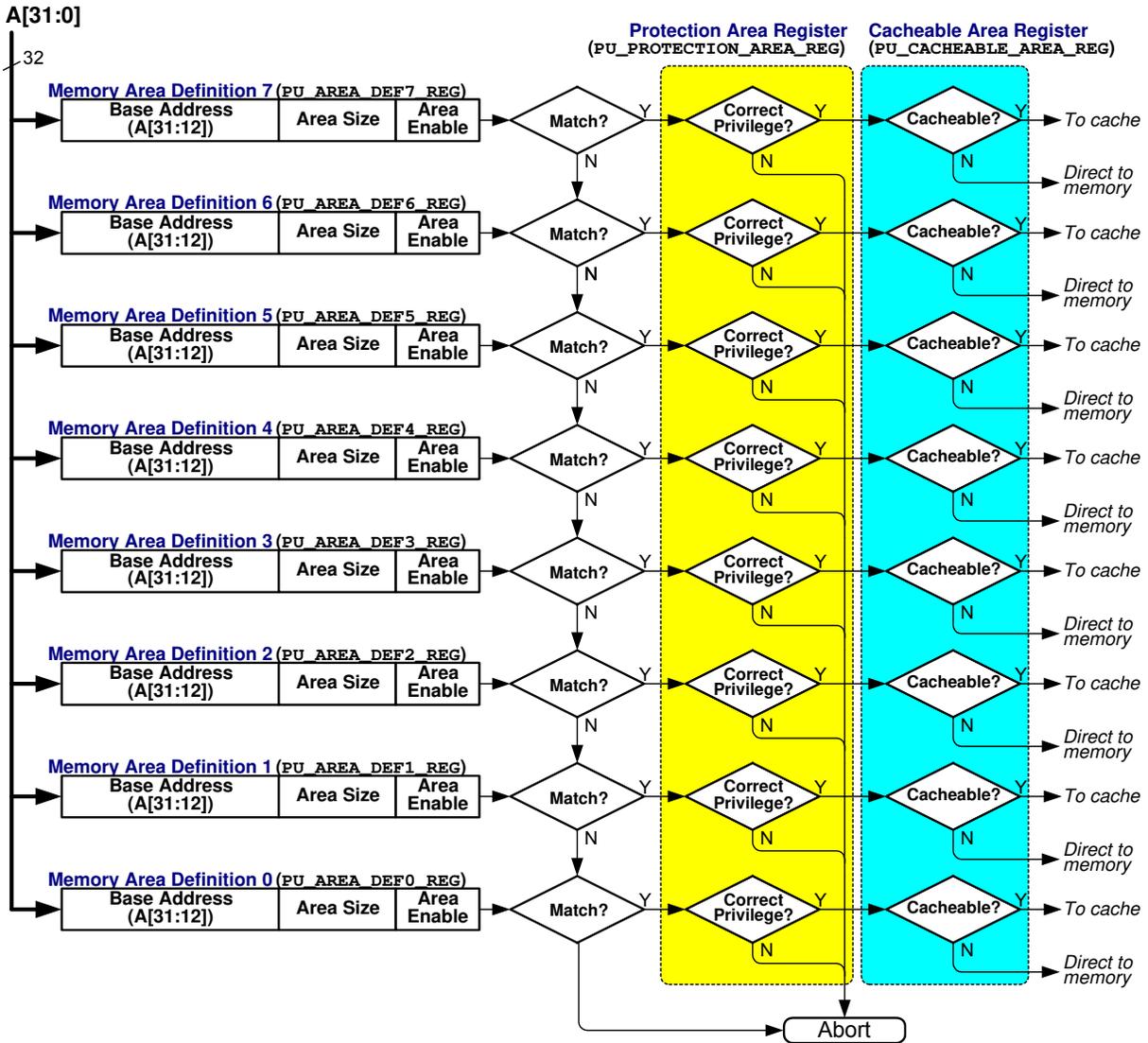
Figure 29. The A7S Memory Protection Unit supports up to eight separate protected memory areas. Each area has a user-definable base address, size, enable, privilege level, and cache enable.

Protection Unit Register Map, CP15 Register Map

Register No.	Function	Access
0	Reserved	-
1	Control	R/W
2	Cacheable Area	R/W
3	Reserved	-
4	Reserved	-
5	Protection	R/W
6	Memory Area Definition	R/W
7	Cache Invalidate	W

Register 1: Protection Unit Control register (PU_CONTROL_REG)

Bit	Description/Function
31:3	Reserved
2	**Cache Enable (CACHE_EN_BIT):** 0: Disable 1: Enable
1	Reserved
0	**Protection Enable (PROT_EN_BIT):** 0: Disable 1: Enable

Default reset value: 0 (cache and protection unit disabled)

Register 2: Cacheable Area Register (PU_CACHEABLE_AREA_REG)

Bit	Description/Function
31:8	Reserved
7	**Cacheable Area 7 Enable (C_7_BIT):** 0: Non-cacheable area 1: Cacheable area
6	**Cacheable Area 6 Enable (C_6_BIT):** 0: Non-cacheable area 1: Cacheable area
5	**Cacheable Area 5 Enable (C_5_BIT):** 0: Non-cacheable area 1: Cacheable area
4	**Cacheable Area 4 Enable (C_4_BIT):** 0: Non-cacheable area 1: Cacheable area
3	**Cacheable Area 3 Enable (C_3_BIT):** 0: Non-cacheable area 1: Cacheable area
2	**Cacheable Area 2 Enable (C_2_BIT):** 0: Non-cacheable area 1: Cacheable area
1	**Cacheable Area 1 Enable (C_1_BIT):** 0: Non-cacheable area 1: Cacheable area
0	**Cacheable Area 0 Enable (C_0_BIT):** 0: Non-cacheable area 1: Cacheable area

Default reset value: Undefined

Register 5: Protection Area Register (PU_PROTECTION_AREA_REG)

Bit	Description/Function
31:16	Reserved
15:14	**Memory Area 7 Access Permission (AP_7_FIELD):** (see Table 25)
13:12	**Memory Area 6 Access Permission (AP_6_FIELD):** (see Table 25)
11:10	**Memory Area 5 Access Permission (AP_5_FIELD):** (see Table 25)
9:8	**Memory Area 4 Access Permission (AP_4_FIELD):** (see Table 25)
7:6	**Memory Area 3 Access Permission (AP_3_FIELD):** (see Table 25)
5:4	**Memory Area 2 Access Permission (AP_2_FIELD):** (see Table 25)
3:2	**Memory Area 1 Access Permission (AP_1_FIELD):** (see Table 25)
1:0	**Memory Area 0 Access Permission (AP_0_FIELD):** (see Table 25)

Default reset value: Undefined

Table 25. Memory Area Access Permission Settings.

Function	Setting
No access to Supervisor and User	0 0
Full access to Supervisor, no access to User	0 1
Full access to Supervisor, read only access to User	1 0
Full access to Supervisor and User	1 1

Register 7: Cache Invalidate register (PU_CACHE_INVALIDATE_REG)

Writing any value to this register invalidates all lines of the cache.

Bit	Description/Function
31:0	Reserved

Register 6: Memory Area Definition Register (PU_AREA_DEFx_REG)

These registers define up to eight separate programmable regions in memory. To access the individual area registers, the CRm field selects the region number when using the MCR or MRC ARM instruction.

Example:

```
MCR/MRC p15,0,Rd,c6,c0
```
accesses register #6 sub-register #0.

Bit	Description/Function
31:12	**Memory Area Base Address (AREA_BASE_ADR_FIELD[19:0]):**
11:6	Reserved
5:1	**Memory Area Size (AREA_SIZE_FIELD[4:0]):** (see Table 26)
0	**Memory Area Enable (AREA_EN_BIT):** 0: Disable 1: Enable

Default reset value: Bit 0=0, all others undefined (disabled).

Table 26. Memory Area Size Settings.

Area Size	Setting
4KB	01011
8KB	01100
16KB	01101
32KB	01110
64KB	01111
128KB	10000
256KB	10001
512KB	10010
1MB	10011
2MB	10100
4MB	10101
8MB	10110
16MB	10111
32MB	11000
64MB	11001
128MB	11010
256MB	11011
512MB	11100
1GB	11101
2GB	11110
4GB	11111
Reserved	00000 to 01010

Scratchpad RAM

A 16Kbyte static RAM memory connects to both the CPU local bus and the CSI bus, as shown in Figure 65. On the local CPU bus, the scratchpad serves as a zero wait-state memory for the CPU. Critical code and data stored in the scratchpad have fast, guaranteed performance. The scratchpad is also accessible over the CSI bus by any CSI master (JTAG or DMA controller). DMA transfers to or from the scratchpad through the CSI bus incur no wait states.

Simultaneous accesses by both the CPU and DMA to the same half of the scratchpad memory are arbitrated in an equal manner. A basic round-robin arbitration results in half of the total scratchpad bandwidth being allocated to each the CPU and DMA. The bandwidth allocation is performed in two bus-cycle increments. Some dead cycles might be present if the CPU does not use the scratchpad memory after it has been granted access.

For increased performance, the scratchpad is separated into two halves, a lower 8Kbyte block and an upper 8Kbyte block. Each half is simultaneously accessible without wait states. While the CPU accesses the bottom half of the scratchpad, a DMA transfer to the upper half happens without performance loss. The application can use the two portions of the scratchpad as a ping-pong memory. While the DMA transfers data to one half, the CPU can process the data in the other half of the scratchpad at full speed.

The scratchpad base address is also programmable in 16K byte increments. Although not necessary in most cases, this option provides more flexibility to the system. However, the scratchpad is not relocatable over an already-defined region, though aliasing to address 0 is the exception because it is activated by other means (see *Alias Enable Register*).

Basic memory protection is available within the scratchpad to protect the contents from being corrupted by CSI accesses. CSI writes to the scratchpad can be selectively disabled, into four independent 4Kbyte areas.   This feature protects critical code or data from errant CSI writes.

During debugging, the scratchpad optionally serves as a trace buffer. When enable during debugging, only the lower 8Kbyte regions is accessible by the application while the upper 8Kbyte region captures real-time bus traces, controlled by the breakpoint unit.

Scratchpad Control Registers

Scratchpad Memory Map

Address		Register Name	Access
Base	**Offset**		
REMAP_BASE	+ 0x44	Scratchpad Configuration Register	R/W
	+ 0x48	Scratchpad Base Address Register	R/W

Scratchpad Configuration Register (REMAP_SRAM_CONFIG_REG)

This register enables to configure the internal SRAM.

Bit	Description/Function
31:6	Reserved
5:2	**SRAM Area Write Protect (SRAM_PROTECT_FIELD[3:0]):** This field defines 4K areas in the scratchpad that can be protected from any CSI write accesses. (see Table 27)
1	**SRAM Size (SRAM_SIZE_BIT):** 0: SRAM is 16K 1: SRAM is 8K, upper 8Kbytes used as trace buffer by breakpoint unit
0	**SRAM Alias Priority (SRAM_PRIO_BIT):** 0: Flash alias has priority over SRAM alias 1: SRAM alias has priority over Flash alias

Default reset value: 0x00000000 (SRAM alias has lower priority than Flash alias)

Table 27. Protected SRAM Area Settings.

Protected SRAM Area	Setting
Scratchpad is open for CSI writes	0000
Protect lower 4K	xxx1
Protect second 4K	xx1x
Protect third 4K	x1xx
Protect upper 4K	1xxx

Scratchpad Base Address Register (REMAP_SRAM_BASE_ADR_REG)

The internal SRAM can be relocated through this register.

Bit	Description/Function
31:14	**SRAM Base Address (SRAM_BASE_ADR_FIELD [17:0]):**
13:0	Reserved

Default reset value: 0xD1030000

External Flash, SDRAM Memory Support

The A7S architecture provides support for external static and dynamic memories through the MSSIU (Memory Subsystem Interface Unit) unit. All memory subsystem configurations share a common set of pins—address, data, and control. The Flash and SDRAM memory configurations are programmed using the **MSS Configuration Register**. The width of the Flash and SDRAM memory subsystems can be defined independently.

> *Configure the external memory interface using FastChip's I/O Editor. From the I/O Editor, you can specify the width and density of the external static memory and SDRAM memory attached to the A7.*

NOTE:

To operate the memory subsystem at frequencies greater than 40MHz, an additional internal memory interface pipeline is activated by the FastChip development software.

Flash or Static Memory Organization

The external memory interface seamlessly connects an A7S CSoC device to standard static memories such as Flash, EEPROM or SRAM. The external static memory subsystem is configurable as an 8-, 16-, or 32-bit system, built from byte-wide or 16-bit static memories. The external interface responds to internal byte, half-word, or word transactions regardless of the width of the external memory subsystem. The memory subsystem is byte addressable and efficiently bridges the 32-bit CSI bus or local CPU buses to the external memory bus. In an 8-bit subsystem, the external interface generates four external transactions to provide the correct data for a word read. In response to a byte write on a 32-bit external memory subsystem, the memory interface masks the unwanted bytes and writes the correct byte. To expand the memory subsystem from an 8-bit to 16-bit or 32-bit, the correct number of chip-enable pins—one additional for 16-bit, three additional for 32-bit—must be reclaimed for system use. The optional chip-enable pins include CE3-/PIO, CE2-/PIO, and CE1-/PIO. Each chip-enable pin drives a one-byte slice of the mem-

ory subsystem (see Table 28). FastChip automatically reclaims the pins as user-defined PIO pins if not used as chip-enable signals.

> **NOTE:**
> ✔ *The memory interface only supports external interfaces built using byte- wide and 16-bit wide memory devices. If using 16-bit wide memories, be sure to set the MSS_FLASH_X16 bit in the MSSIU Configuration Register.*

The amount of addressable external static memory (i.e., not SDRAM) begins at a default minimum of 1M bytes, or 20 address lines. Up to four additional PIO pins can be reclaimed as address lines to expand to amount of external memory, up to 16M bytes maximum.

If less than the maximum external static memory is attached, then multiple copies of the attached memory appear, duplicated at every multiple of the size of the memory. Examples are shown in Figure 30. For instance, if only 8Mbytes of Flash are attached, then two copies of the memory appear in the Flash address region. One copy appears at 0xD0000_0000, another at 0xD080_0000. The same applies if the static memory is aliased into the bottom of memory (see **Alias Enable Register**).
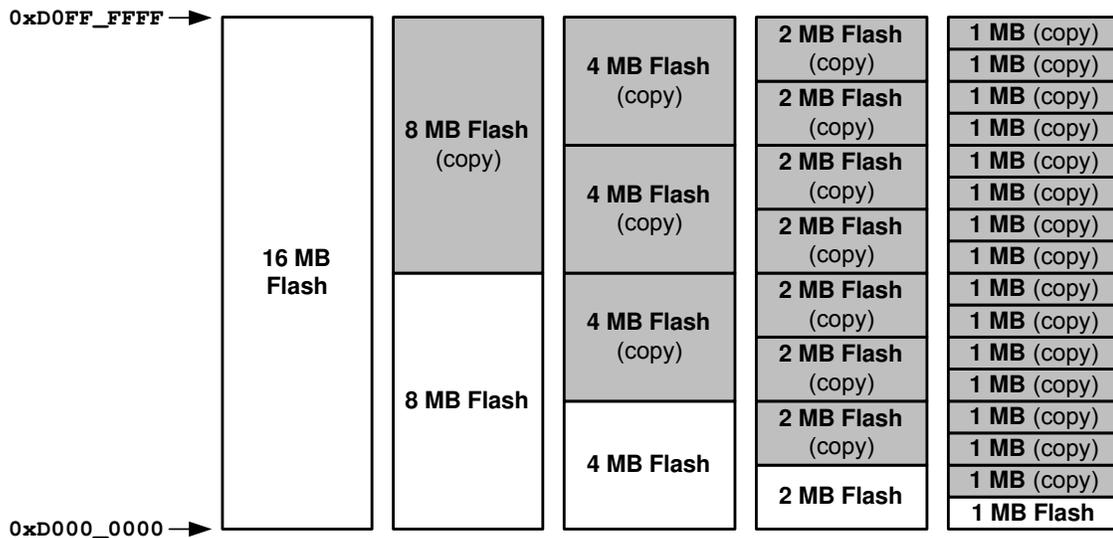
Figure 30. If less than the maximum static memory is attached, copies of the memory appear at every boundary of the attached memory size.

The external memory is mapped linearly in the system memory map starting at the predefined base address (see Figure 28).

The external memory content follows the little endian format. Figure 31, Figure 32, and Figure 33 illustrate the external memory content for each of the possible data-width organizations.

> **NOTE:**
> ✔ *If executing directly from an 8-bit or 16-bit wide external memory, the ARM7TMDI's Thumb mode offers improved performance. By using a 16-bit instruction instead of a 32-bit instruction, Thumb mode reduces the number of fetches required from external memory.*

(msb)
31

(lsb)
0

Upper Address	Byte *n*	Byte *n*-1	Byte *n*-2	Byte *n*-3
	⋮			
	Byte 7	Byte 6	Byte 5	Byte 4
Lower Address	Byte 3	Byte 2	Byte 1	Byte 0
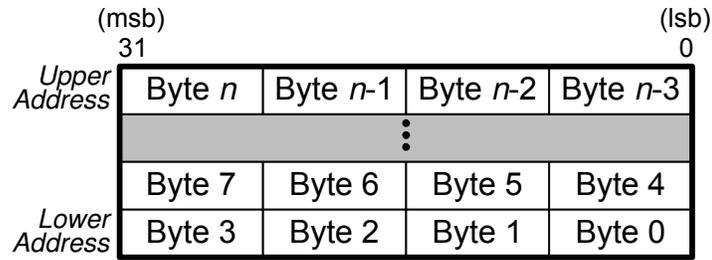
Figure 31. 32-bit Memory Organization.

(msb)
15

(lsb)
0

Upper Address	Byte *n*	Byte *n*-1
	⋮	
	Byte 7	Byte 6
	Byte 5	Byte 4
	Byte 3	Byte 2
Lower Address	Byte 1	Byte 0
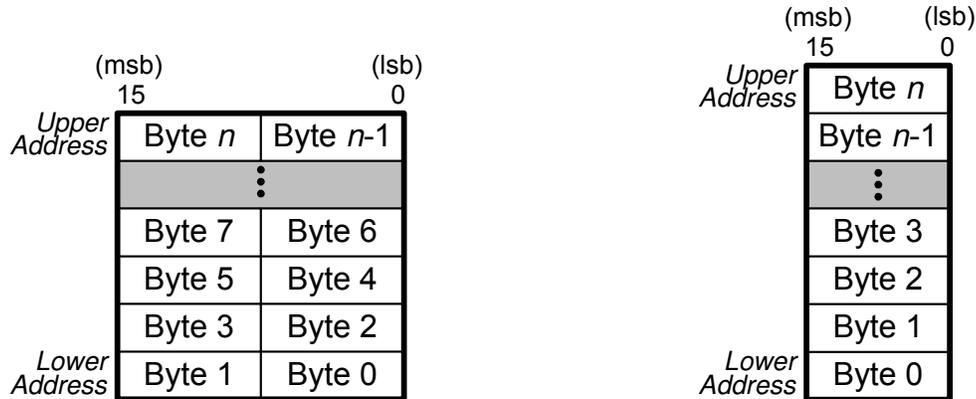
Figure 32. 16-bit memory Organization.

(msb)   (lsb)
15   0

Upper Address	Byte *n*
	Byte *n*-1
	⋮
	Byte 3
	Byte 2
	Byte 1
Lower Address	Byte 0
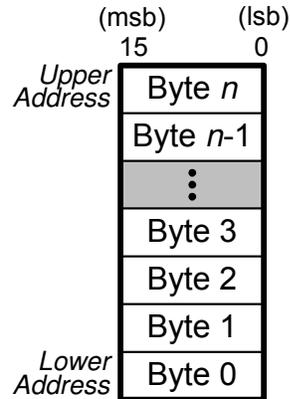
Figure 33. 8-bit Memory Organization.

Using Thumb mode in narrow memory systems

By default, the ARM7TDMI CPU executes 32-bit ARM instructions. ARM instructions offer the highest execution performance when the A7S connects to a 32-bit wide instruction memory. When operating with an 8-bit wide or 16-bit wide instruction memory, each instruction fetch involves multiple memory accesses.

To enhance performance in narrow memory systems, the ARM7TDMI CPU offers Thumb mode, which compresses a subset commonly used instructions into a 16-bit op-code. The benefits are even higher code density, reducing the cost of the instruction memory, but also offering higher performance in narrower memory systems. Figure 34 shows the relative performance of an ARM mode application versus a Thumb mode application, executing directly from external memory, *i.e.*, no cache. The width of the external memory affects ARM mode performance more dramatically than for Thumb mode.
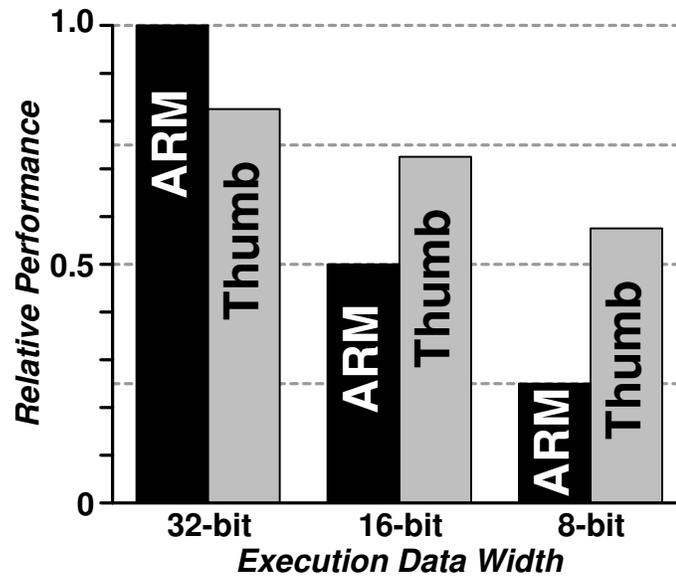
Figure 34. Relative performance of ARM mode versus Thumb mode executing from 32, 16, and 8 bit wide external memory.

The A7S's cache memory and internal SRAM are both 32 bits wide, which benefits ARM mode instructions. Likewise, code can be copied from an 8-bit or 16-bit Flash memory into a 32-bit SDRAM memory and executed from SDRAM.

Interface timing

The interface timing of the memory signals is programmable to accommodate memories with different access times. The *MSSIU Control Timing Register* defines the timing of each portion of the static memory read and write cycles. The basic external-memory read and write waveforms are illustrated in Figure 35 and Figure 36.
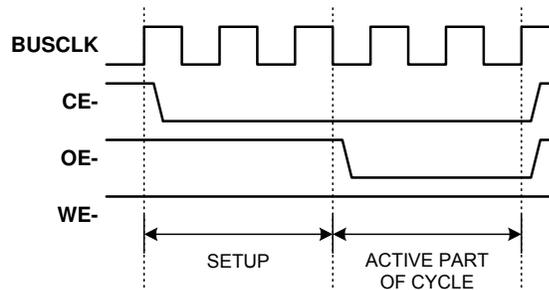
Figure 35. Generalized read cycle using the static memory interface.

FastChip Device Link (FDL) automatically configures the MSSIU Control Timing Register whenever you create a configuration image and specify an external memory device. FDL uses the memory's interface timing information and your operating frequency to determine the proper values.

FastChip Device Link
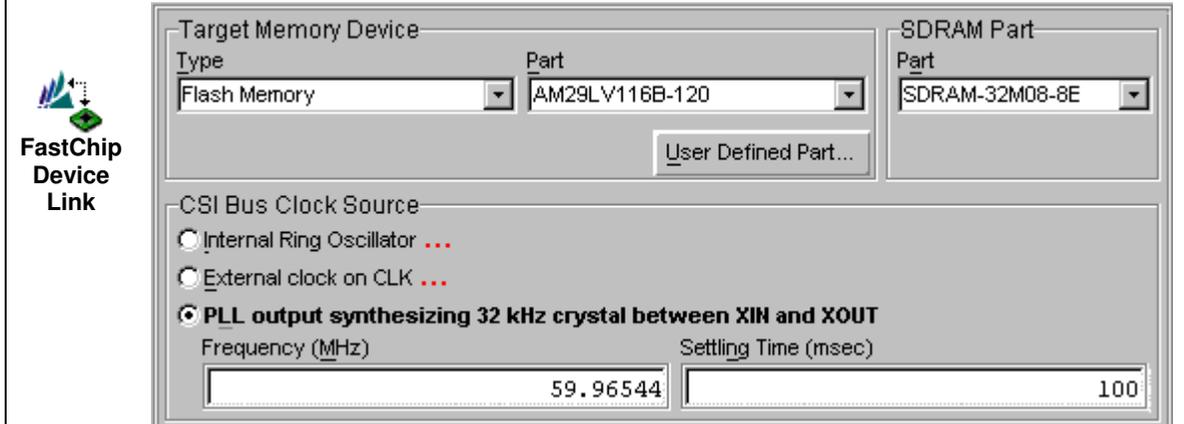
During a read cycle, the output enable (OE-) signal is asserted at the falling edge of the system clock and is de-asserted at the rising edge of the system clock. The read cycle is divided between a *setup* and an *active* portion. Each of them is controlled independently and can have values between ½ and 15.5 times the system clock, by increments of one. At lower system-clock frequencies, one-cycle read operations are possible.
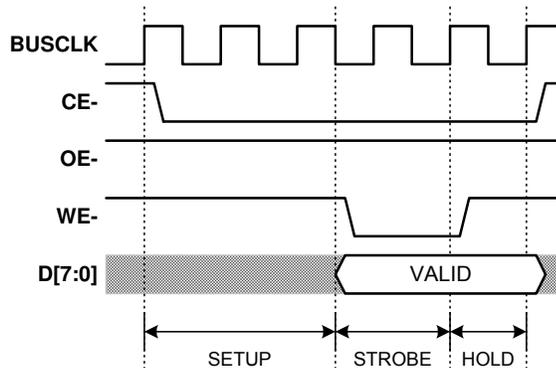
Figure 36. Generalized write cycle using the static memory interface.

During a write cycle, the write enable (WE-) signal is asserted at the falling edge of the system clock and is de-asserted at the rising edge of the system clock. The write cycle is divided between a *setup*, an *active,* and a *hold* portion. The *setup* and *active* portions are controlled independently and can have values between ½ and 15.5 times the system clock, by increments of one. The hold portion can have a value between 0 and 15 times the system clock. At lower system-clock frequencies, one-cycle write operations are possible.

It is possible to execute from external memory with zero wait-states by setting all the external memory timing values to zero. If hold time is set to zero, i.e.—a one-cycle write, watch the timing of the WE- signal at the board level. Be sure to set the electrical characteristics of the WE- pin to high current drive and fast slew rate.

SDRAM Memory Controller

The A7S CSoC device provides an optional interface supporting up to 256Mbytes of synchronous DRAM (SDRAM). A variety of SDRAM types are supported—64M-bit, 128M-bit and 256M-bit—in their most common configuration (x8 and x16) as well as 100-pin SDRAM modules with 4M to 64MBytes configurations. The SDRAM controller supports an external SDRAM subsystem with an 8-, 16- or 32-bit data path, operating as a single or dual bank. The 100-pin SDRAM modules are configured as a 32-bit system. The SDRAM interface allows up to four SDRAM chips per banks in single or dual bank mode, or one 100-pin module to be connected to the external memory bus. In any of these configurations, the SDRAM interface accepts byte, half-word, and word transactions. It generates the appropriate external cycles to serve these requests. For example, if the external dynamic memory is configured as an 8-bit system, the SDRAM controller generates four equivalent SDRAM read cycles in response to a word transaction. For a 32-bit memory subsystem, byte and half-word transactions are supported through byte masking.

If less than the maximum external SDRAM memory is attached, then multiple copies of the attached memory appear, duplicated at every multiple of the size of the memory. Examples are shown in Figure 37. For instance, if only 64Mbytes of SDRAM are attached, then four copies of the memory appear in the SDRAM address region. Copies appear at 0xC0000_0000, 0xC400_0000, 0xC800_0000, and 0xCC00_0000. The same applies if the static memory is aliased into the bottom of memory (see *Alias Enable Register*).

Figure 37. If less than the maximum SDRAM memory is attached, copies of the memory appear at every boundary of the attached memory size.

The SDRAM interface operates up to 60 MHz, requiring SDRAMs with a 8 ns or less access time. The SDRAM timing parameters—CAS latency, precharge cycles, etc.—are controlled by the interface and are transparent to the user.

The auto-refresh feature retains the data inside the SDRAM external memory. An internal 12-bit counter generates refresh requests at regular intervals. The count is programmable in multiples of the system clock period. To achieve a refresh every 15.625 μs—a typical requirement for 64M-bit parts— set the count value to 1,031 when operating at 60 MHz (15.625us x 60MHz). The self-refresh feature of the SDRAM is used during power-down to retain memory content.

SDRAM Organization

The SDRAM address space is organized into columns, rows and banks and different types of SDRAM have varying numbers of each. The linear and contiguous address generated by the system must be mapped to the column, row, and bank address lines appropriate for the specific type of SDRAM. Figure 38 through Figure 40 show the relation between the A7S's linear addresses lines and the columns, rows, banks addresses for SDRAM memory subsystems with bus widths of 8, 16, and 32 bits.

The upper four address bits, A[31:28] are always 1100'b because the external SDRAM is mapped starting at 0xC000_0000 in the system memory map (see Figure 28). The SDRAM chip-select inputs connect to the A7S's SDCE[1:0]- chip-select signals. Two external banks must be defined in the *MSS Configuration Register* and the appropriate settings made in the FastChip I/O Editor for SDCE1- to be valid.

Optimized Performance

The SDRAM controller is optimized for DMA transfers and Cache fills. DMA packets and Cache lines are four words deep. When using DMA, the DMA transfer buffers should be enabled for best system performance (see *DMA SDRAM Transfers*).

To maximize performance when accessing SDRAM, it is important to understand the memory organization. For a particular bank, accessing a row that is different than the active row results in precharge and activation cycles, followed later by the actual access. For maximum performance, avoid accessing different rows within the same internal bank. For example, placing the application code into the first SDRAM bank, and performing DMA transfers to or from the other bank allows cache fills without the memory precharge penalty, except at page boundaries.

Power-up Sequence

Regardless of the physical presence of external SDRAM, the A7S device automatically performs a SDRAM power-up sequence during its power-on initialization cycle. The SDCLK, SDCKE, and SDCE[1:0]- all toggle during initialization.

If physically connected, the external SDRAMs are powered-up every time the A7S executes its initialization sequence. The A7S configures the MODE REGISTER within the SDRAM devices using the values specified in the *SDRAM Mode Register*. In a typical application, the initialization sequence switches the SDRAM from the disabled state to the active state. Application code then specifies the correct refresh requirements and then either has the SDRAM remain in active mode or places the SDRAM subsystem in stand-by mode to reduce power consumption.

SDRAM Timing Parameters

The basic SDRAM timing parameters are programmable and can be optimized corresponding to the system clock frequency. Each of the following options controls an SDRAM timing parameter as a multiple of the system clock period. An example waveform is shown in Figure 44.

- Precharge period (TRP_FIELD)
- Activate to read/write command (TRCD_FIELD)
- CAS latency, although CAS=2 should work for all applications (MODE_REG_FIELD[6:4])
- Write recovery time (TWR_FIELD)
- Refresh to Active period (TRC_FIELD)

All other SDRAM parameters are hard-coded within the SDRAM controller hardware.

Memory Organization	CS	Bank	Row	Column (high)	Column (low)	Byte mask
64M-bit (4x4Kx512x8)	CS	B[1:0]	R[11:0]	C[8:2]	C[1:0]	M[1:0]
64M-bit (4x4Kx256x16)	CS	B[1:0]	R[11:0]	C[7:2]	C[1:0]	M[1:0]
128M-bit (4x4Kx1Kx8)	CS	B[1:0]	R[11:0]	C[9:2]	C[1:0]	M[1:0]
128M-bit (4x4Kx512x16)	CS	B[1:0]	R[11:0]	C[8:2]	C[1:0]	M[1:0]
256M-bit (4x8Kx1Kx8)	CS	B[1:0]	R[12:0]	C[9:2]	C[1:0]	M[1:0]
256M-bit (4x8Kx512x16)	CS	B[1:0]	R[12:0]	C[8:2]	C[1:0]	M[1:0]
1Mx32 DIMM	CS	B0	R[10:0]	C[7:2]	C[1:0]	M[1:0]
2Mx32 DIMM	CS	B0	R[10:0]	C[7:2]	C[1:0]	M[1:0]
4Mx32 DIMM	CS	B[1:0]	R[11:0]	C[7:2]	C[1:0]	M[1:0]
8Mx32 DIMM	CS	B[1:0]	R[11:0]	C[7:2]	C[1:0]	M[1:0]
16Mx32 DIMM	CS	B[1:0]	R[11:0]	C[8:2]	C[1:0]	M[1:0]

Figure 38. SDRAM Address Mapping for 32-bit External Memory Subsystems.

Memory Organization	CS	Bank	Row	Column (high)	Column (low)	Byte mask
64M-bit (4x4Kx512x8)	CS	B[1:0]	R[11:0]	C[8:3]	C[2:0]	M0
64M-bit (4x4Kx256x16)	CS	B[1:0]	R[11:0]	C[7:3]	C[2:0]	M0
128M-bit (4x4Kx1Kx8)	CS	B[1:0]	R[11:0]	C[9:3]	C[2:0]	M0
128M-bit (4x4Kx512x16)	CS	B[1:0]	R[11:0]	C[8:3]	C[2:0]	M0
256M-bit (4x8Kx1Kx8)	CS	B[1:0]	R[12:0]	C[9:3]	C[2:0]	M0
256M-bit (4x8Kx512x16)	CS	B[1:0]	R[12:0]	C[8:3]	C[2:0]	M0

Figure 39. SDRAM Address Mapping for 16-bit External Memory Subsystems.

Memory Organization	CS	Bank	Row	Column (high)	Column (low)
64M-bit (4x4Kx512 x8)	CS	B[1:0]	R[11:0]	C[8:4]	C[3:0]
128M-bit (4x4Kx1Kx8)	CS	B[1:0]	R[11:0]	C[9:4]	C[3:0]
256M-bit (4x8Kx1Kx8)	CS	B[1:0]	R[12:0]	C[9:4]	C[3:0]

Figure 40. SDRAM Address Mapping for 8-bit External Memory Subsystems.

In Figure 38 through Figure 40:
M = byte mask, **C** = column, **R** = row, **B** = internal bank, **CS** = external bank select
Memory Organization = Banks x Rows x Columns x Data Width = M-bits

Refresh

SDRAM must be refreshed periodically to retain its data content. The refresh rate is specified in the ***SDRAM Control Register*** as the number of bus clock cycles between refresh cycles. During normal or stand-by operation, the SDRAM controller issues an auto-refresh command, which cycles the refresh address counter inside the SDRAM. Long burst refresh sequences are avoided. An example waveform is shown in Figure 45.

In full power-down mode, the SDRAMs use their self-refresh feature where the SDRAM device generates its own clock.

Power Management

The A7S provides the following SDRAM modes. Writing to the Power Management field of the ***SDRAM Control Register*** transfers the SDRAM memory from one level to another.

1. ***Disable state***. Writing 000 (default value) disables the SDRAM controller. No read or write accesses or refresh cycles are possible. In this mode, the SDRAM does not retain data.

2. ***Active state***. Writing 010'b activates the SDRAM controller. Activating the controller from the disabled state causes the controller to apply a PRECHARGE ALL command to the SDRAM memory, followed by eight AUTO REFRESH cycles. The SDRAM controller then programs the MODE REGISTER in the SDRAM using the values specified in the ***SDRAM Mode Register***. The SDRAM is ready for full-speed access.

3. ***Stand-by***. Writing 011'b activates the stand-by mode. This state is similar to the active state, except that the SDRAM clock enable pin (SDCKE) is forced Low as soon as no SDRAM accesses are required. By doing so, the SDRAM enters a power-saving mode without being de-activated. The SDRAM leaves the power-saving mode automatically upon a refresh or a read or write request. The performance loss is minimal because only one NOP command is inserted when exiting power saving mode.

4. ***Power-down***. Writing 100'b causes the SDRAM controller to generate a self-refresh sequence. The SDRAM device retains its content by generating refresh cycles internally without the A7S's SDRAM controller. To exit this mode, return to the active or stand-by states.

Connecting Flash and SDRAM to the A7

Table 28 lists how the A7S's pins connect to different memory configurations. Both Flash and SDRAM can be connected simultaneously, although SDRAM is optional.

Addresses are shifted internally to the A7S for 16-bit and 32-bit Flash configurations. Consequently, in the 16-bit configuration, address pins A[22:0] represent a half-word address. Similarly, in 32-bit configuration, address pins A[21:0] represent a word address. There is a total of 16M bytes of external Flash or static memory address space.

> *When using 16-bit Flash devices, watch out for differences in terminology used between various Flash suppliers. In a 16-bit wide memory subsystem, the A7S internally shifts the address presented to the memory interface. The A7S's A0 address line must connect to the half-word address input on the Flash device. On some Flash devices, the half-word address input is labeled A0, with a separate input labeled D15/A-1 for byte addressing. Other vendors label their half-word address input as A1 and the byte address input as A0.*
>
> *For example, the diagram below shows how to connect the A7S device to an Intel StrataFlash or an AMD Am29F800B Flash. Tie the Flash BYTE# input to select 16-bit or byte-wide mode.*
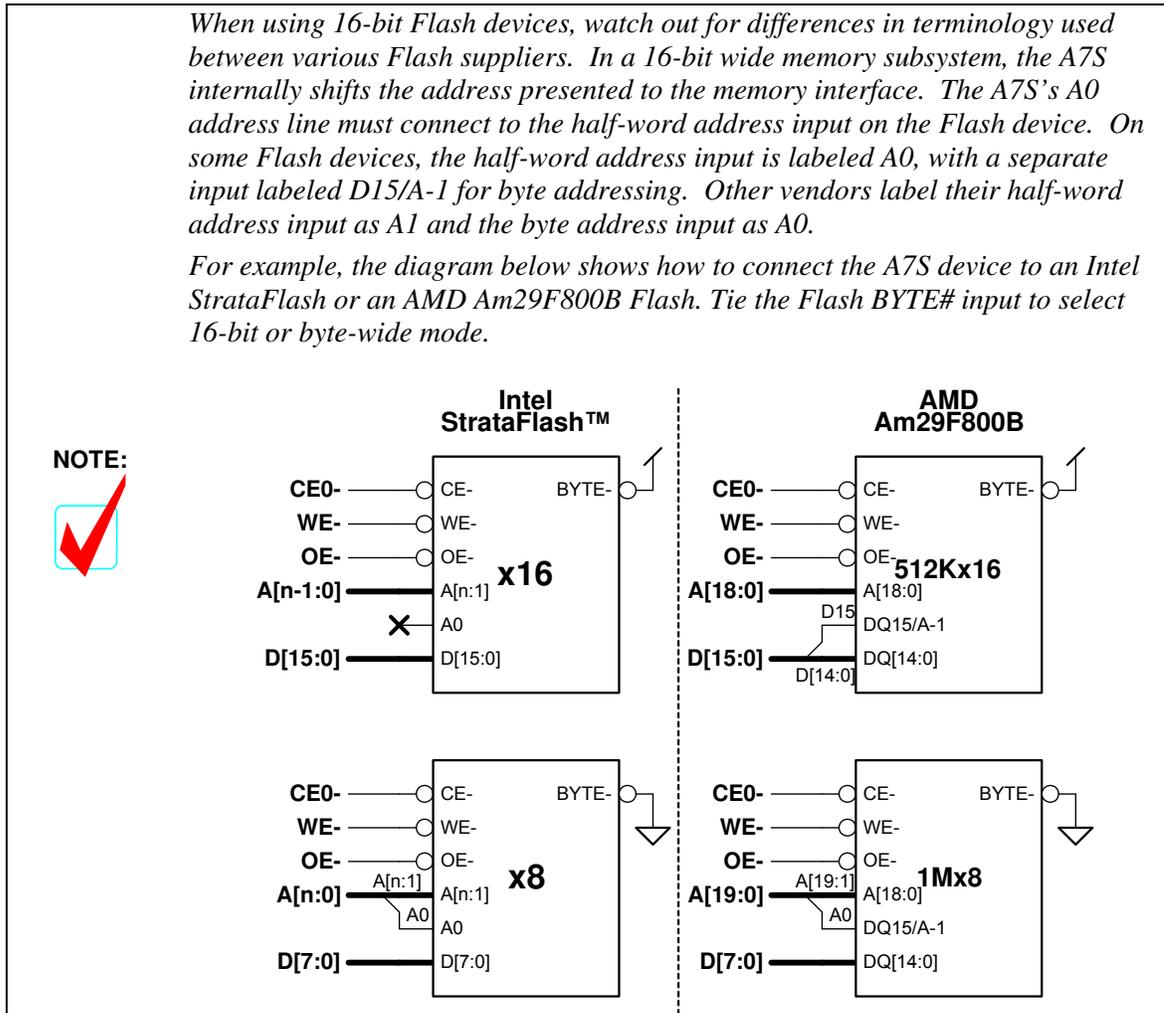
Figure 41 through Figure 42 illustrate the possible Flash configurations and connections, ranging from a byte-wide interface to a 32-bit wide interface, built using byte-wide and 16-bit wide memories.

Table 28. External Memory Connectivity.

Device Pins	Static Memory (Flash)					SDRAM		
	8 bits wide	16 bits wide		32 bits wide		8 bits wide	16 bits wide	32 bits wide
Memory Width	8-bit	8-bit	16-bit	8-bit	16-bit			
CE0-	CE-	CE- Byte 0	CE-	CE- Byte 0	CE- Lower Half-Word			
CE1-		CE- Byte 1		CE- Byte 1				
CE2-				CE- Byte 2	CE- Upper Half-Word			
CE3-				CE- Byte 3				
WE-	WE-	WE-	WE-	WE-	WE-	WE	WE	WE
OE-	OE-	OE-	OE-	OE-	OE-			
SDCLK						CLK	CLK	CLK
SDCKE						CKE	CKE	CKE
SDCE[1:0]-						CS[1:0]	CS[1:0]	CS[1:0]
A[0]	A[0]	A[0]	A[0]	A[0]	A[0]	RAS	RAS	RAS
A[1]	A[1]	A[1]	A[1]	A[1]	A[1]	CAS	CAS	CAS
A[2]	A[2]	A[2]	A[2]	A[2]	A[2]	DQM[0]	DQM[0]	DQM[0]
A[3]	A[3]	A[3]	A[3]	A[3]	A[3]		DQM[1]	DQM[1]
A[5:4]	A[5:4]	A[5:4]	A[5:4]	A[5:4]	A[5:4]			DQM[3:2]
A[7:6]	A[7:6]	A[7:6]	A[7:6]	A[7:6]	A[7:6]	BS[1:0]	BS[1:0]	BS[1:0]
A[20:8]	A[20:8]	A[20:8]	A[20:8]	A[20:8]	A[20:8]	A[12:0]	A[12:0]	A[12:0]
A[21]	A[21]	A[21]	A[21]	A[21]	A[21]			
A[22]	A[22]	A[22]	A[22]					
A[23]	A[23]							
A[31:24]								
D[7:0]	D[7:0]	D[7:0]	D[7:0]	D[7:0]	D[7:0]	DQ[7:0]	DQ[7:0]	DQ[7:0]
D[15:8]		D[15:8]	D[15:8]	D[15:8]	D[15:8]		DQ[15:8]	DQ[15:8]
D[31:16]				D[31:16]	D[31:16]			DQ[31:16]

Figure 41. An 8-bit Flash memory interface.

a.) 32-bit Subsystem using four 8-bit Flash devices.

b.) 32-bit Subsystem using two 16-bit Flash devices. Set MSS_FLASH_X16 bit.

Figure 42. A 32-bit Flash memory interface.

a.) 16-bit Subsystem using two 8-bit Flash memories.
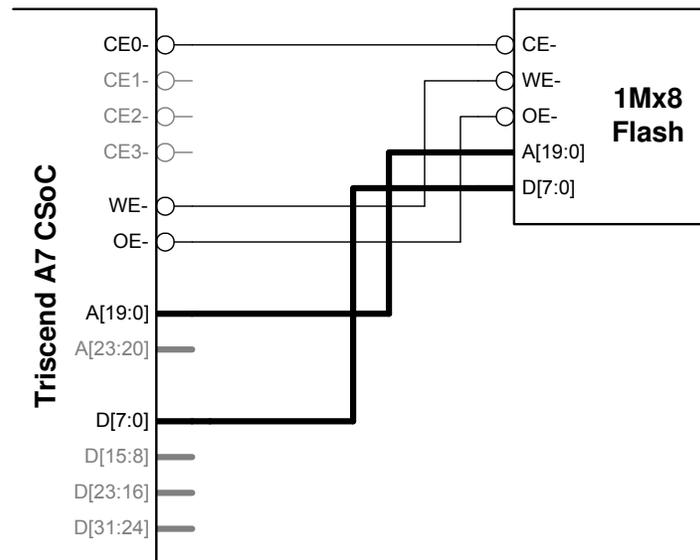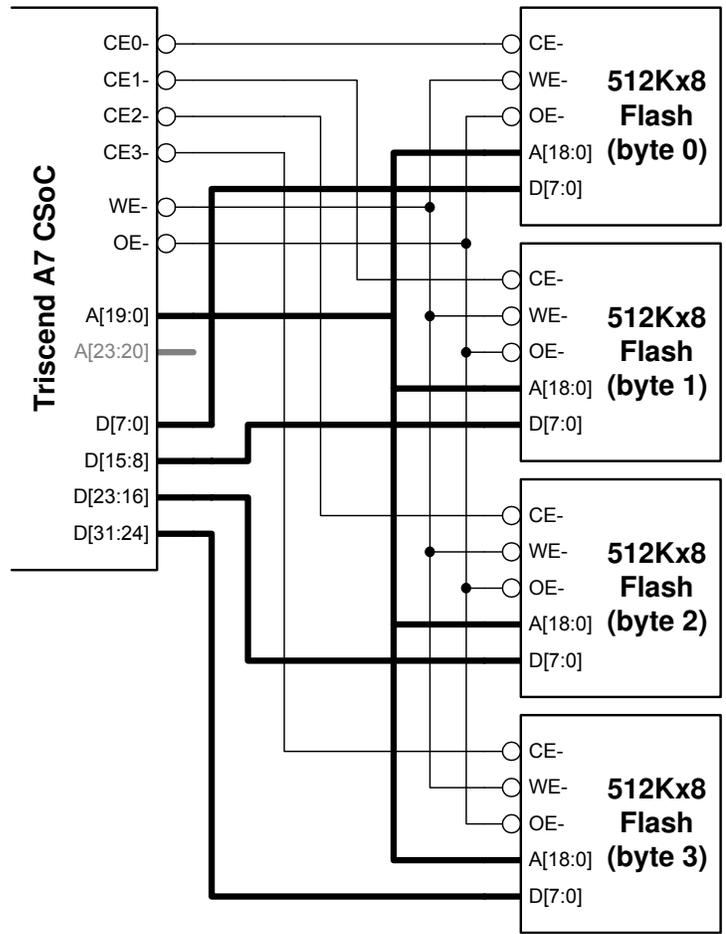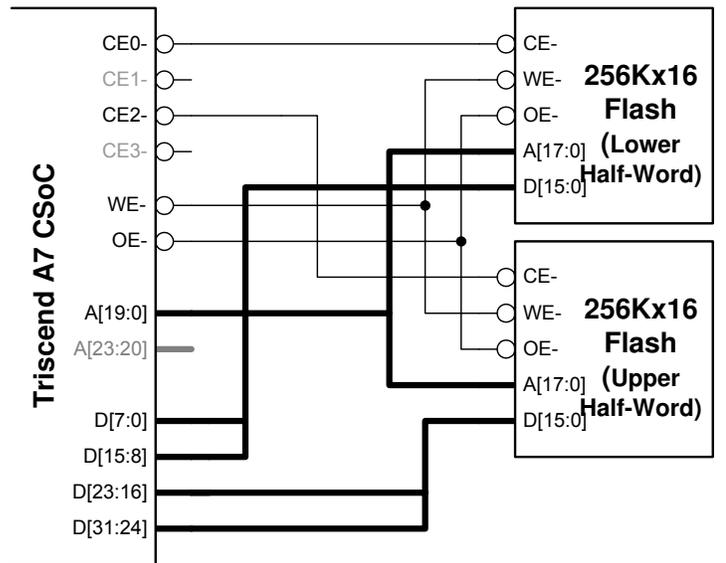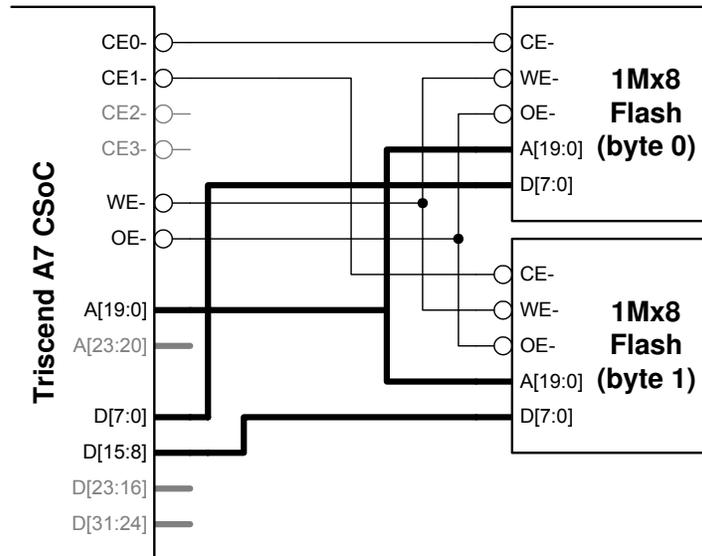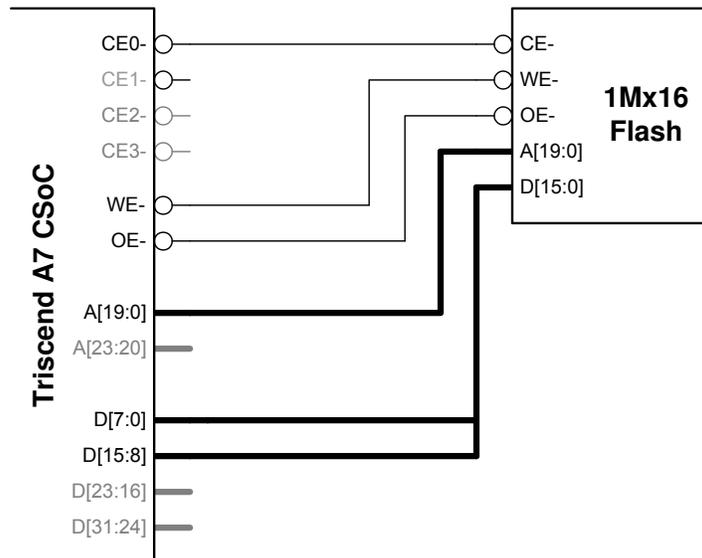

b.) 16-bit Subsystem using a 16-bit Flash memory. Set MSS_FLASH_X16 bit.

Figure 43. A 16-bit Flash memory interface (1Mx16).

Memory Subsystem Arbitration

The memory subsystem serves requests from different sources including the CPU, the DMA buffers, the CSI bus (JTAG or non-buffered DMA transactions) and internal SDRAM refresh request. The arbitration is round-robin style where the arbitration priority rotates between the various sources, except for the internal SDRAM refresh request. The internal refresh is a low priority request and usually gives way to any other request. However, after accumulating too many un-serviced refresh requests, the refresh priority is promoted to the same level as the other sources, guaranteeing service after a few cycles of latency.

In a heavily loaded application, where DMA transfers use the full CSI bus bandwidth and where the cache requests fills, half the memory interface bandwidth is allocated to cache fills while the other half is allocated to the DMA transfers. The cache fills cannot use the full memory bandwidth anyway because there is a minimum of three clock cycles between the end of a cache fill and the next cache fill request. These three cycles can be used by any other sources without impacting on the cache fill rate.

DMA Data Transfers to and from External Memory

DMA SDRAM Transfers

The SDRAM subsystem is designed and optimized for four-cycle burst transfers on the memory bus.

Transferring just one piece of data at a time to or from SDRAM is inefficient. In addition to the CAS latency, which is typically two cycles, the burst is aborted for every piece of data. Therefore, every single data transfer incurs several clock cycles of overhead.

The A7S includes FIFOs within the SDRAM interface to maximize the burst nature of SDRAM. Each DMA channel has a 8x32 buffer, which is individually enabled in the DMA_BUFFER_EN_FIELD in the *Memory Subsystem Configuration Register*.

When writing to memory, the DMA FIFO accumulates four words and then transfers these four words to SDRAM efficiently. Upon receiving the last piece of write data, the FIFO automatically writes out all the FIFO contents to the SDRAM, flushing the FIFO.

When reading, data is burst from SDRAM (prefetched) and stored in the FIFO until the DMA requests the data. When the transfer is completed, the FIFO is automatically cleared by hardware to discard any unused pre-fetched data.

There are no data coherency issues because, upon receiving the last piece of data of a particular DMA transfer, the FIFO receives the highest priority for flushing itself to external SDRAM memory.

DMA Flash Transfers

The DMA FIFOs also marginally improve data transfers to and from external Flash. As with SDRAM, the individual buffers are enabled in the DMA_BUFFER_EN_FIELD.

Additional Latency at Higher Frequencies

To guarantee performance at higher bus clock frequencies, an additional pipeline is inserted between the CPU memory requests and the memory interface when operating at frequencies over 40 MHz.

This pipeline increases latency during cache fills and code execution from external memory.

Setting PIPE_BIT enables the bus pipeline.

Memory Map

Address		Register Name	Access
Base	**Offset**		
MSS_BASE	+ 0x00	Memory Subsystem Configuration	R/W
	+ 0x04	Static Memory Interface Timing Control	R/W
	+ 0x08	SDRAM Mode	R/W
	+ 0x0C	SDRAM Control	R/W
	+ 0x10	Status	R
	+ 0x14	Status Clear	W

FastChip Device Link

FastChip Device Link (FDL) automatically configures the memory interface control registers whenever you create a configuration image and specify an external memory device. FDL uses the memory's interface timing information and your operating frequency to determine the proper values.

Memory Subsystem Configuration Register (MSS_CONFIG_REG)

FastChip Device Link automatically configures most options. The DMA Buffer Enables must be configured by application software.

Bit	Description/Function
31:25	Reserved
24	**Flash Devices are x16 (MSS_FLASH_X16):** This bit identifies the data width of individual static memory devices in order to generate the correct chip enable sequence for byte accesses. 0: Byte-wide Flash devices 1: 16-bit-wide Flash devices
23:16	**DMA Buffer Enables (DMA_BUF_EN_FIELD [7:0]):** This field enables DMA channel buffers independently of each other. 0: Disable channel 1: Enable channel *Flash Buffer Enables* <table><tr><td>Bit</td><td>23</td><td>22</td><td>21</td><td>20</td></tr><tr><td>Channel</td><td>DMA 3</td><td>DMA 2</td><td>DMA 1</td><td>DMA 0</td></tr></table> *SDRAM Buffer Enables* <table><tr><td>Bit</td><td>19</td><td>18</td><td>17</td><td>16</td></tr><tr><td>Channel</td><td>DMA 3</td><td>DMA 2</td><td>DMA 1</td><td>DMA 0</td></tr></table>

Bit	Description/Function
15	**CPU Interface Pipeline Enable (PIPE_BIT):** This bit enables the CPU interface pipeline. This bit should be set under most conditions. The bit should be cleared if the system clock is low frequency. 0: Disable pipeline 1: Enable pipeline (required if operating at 40 MHz or faster)
14	**Number of External SDRAM Banks (NE_BANK_BIT):** This field indicates how many external banks are present in the SDRAM memory subsystem. 0: One external bank 1: Two external banks
13	**Number of Banks within SDRAM Device (N_BANK_BIT):** This field indicates how many internal banks are present within each SDRAM device. The 16M-bit SDRAM devices have 2 banks while 64M-bit devices and larger have 4 banks. 0: Two internal banks 1: Four internal banks (default)
12:10	**SDRAM Memory Subsystem Bank Mapping (B_MAP_FIELD [2:0]):** This field indicates how the linear address is mapped onto the bank select lines. The number of rows, columns and the bus-width determines the shifting of the bank select lines (Bank) relative to the system address (Addr). (see Table 29)
9:8	**SDRAM Memory Subsystem Row Mapping (R_MAP_FIELD [1:0]):** This register indicates how the linear address is mapped onto the row address, R. The number of columns and the bus-width determines the shifting of the row address (Row) relative to the system address (Addr). (see Table 30)
7:6	**SDRAM Memory Subsystem Datapath Width (SDIU_DEV_WIDTH_FIELD [1:0]):** See Figure 38, Figure 39, and Figure 40.
5:4	**Static Memory Subsystem Datapath Width (MIU_DEV_WIDTH_FIELD [1:0]):** See Figure 38, Figure 39, and Figure 40.
3:0	**Bus Mode (BUS_MODE_FIELD [3:0]):** This field indicates the major bus modes of the memory subsystem as it appears to the internal CSI Bus (see Table 32).

Configuration reset value: 0x00004002

Table 29. SDRAM Memory Subsystem Bank Mapping Settings.

SDRAM Memory Subsystem Bank Mapping	Settings
Bank[1:0] = Addr[22:21]	000
Bank[1:0] = Addr[23:22]	001
Bank[1:0] = Addr[24:23]	010
Bank[1:0] = Addr[25:24]	011
Bank[1:0] = Addr[26:25]	100

See Figure 38, Figure 39, and Figure 40.

Table 30. SDRAM Memory Subsystem Row Mapping Settings.

Row Mapping	Setting
Row = Addr[I+9]	00
Row = Addr[I+10]	01
Row = Addr[I+11]	10
Row = Addr[I+12]	11

See Figure 38, Figure 39, and Figure 40.

Table 31. Memory Subsystem Datapath Width Settings.

Datapath Width	Setting
Byte -- 8-bit	00
Half-Word -- 16-bit	01
Word -- 32-bit	10
Reserved	All Others

Table 32. CSI Bus Mode Settings.

CSI Bus Mode	Setting
Slave – CSI Slave Mode	0001
Master – CSI Master Mode	0010
Reserved	All Others

Static Memory Interface Timing Control Register (MSS_TIM_CTRL_REG)

This register specifies the read and write timing values of the static memory subsystem. FastChip Device Link automatically configures these options. However, the timing values may be modified when changing operating frequency or when entering or existing power-down mode.

Bit	Description/Function
31:20	Reserved
19:16	**Read Cycle Pulse Width (RC_WIDTH_FIELD [3:0]):** This field specifies the pulse width of OE- when generating a memory read sequence. The pulse width is calculated using the following formula: $T_{RPW} = (RcWidth[3:0] + 0.5) * T_{CLK}$
15:12	**Read Cycle Setup Time (RC_SETUP_FIELD [3:0]):** This field specifies the width of the set-up portion of a read cycle in addition to the half-cycle mandatory width of OE-. The setup time is calculated using the following formula: $T_{RSU} = (RcSetup[3:0] + 0.5) * T_{CLK}$
11:8	**Write Cycle Hold Time (WC_HOLD_FIELD [3:0]):** This field specifies the width of the hold portion of a write cycle. The hold time is calculated using the following formula: $T_{WHD} = WcHold[3:0] * T_{CLK}$
7:4	**Write Cycle Pulse Width (WC_WIDTH_FIELD [3:0]):** This field specifies the pulse width of WE- when generating a memory write sequence. The pulse width is calculated using the following formula: $T_{WPW} = (WcWidth[3:0] + 0.5) * T_{CLK}$
3:0	**Write Cycle Setup Time (WC_SETUP_FIELD [3:0]):** This field specifies the WE- assertion setup time for a memory write cycle. The setup time is calculated using the following formula: $T_{WSU} = (WcSetup[3:0] + 0.5) * T_{CLK}$

Configuration reset value: 0x00077777
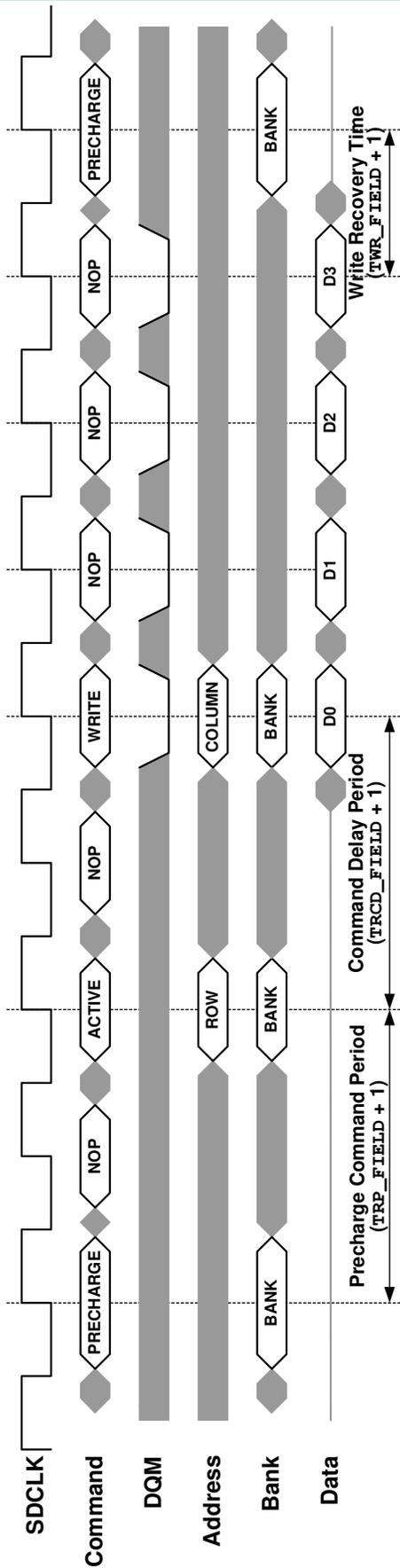
Figure 44. Example SDRAM Burst Write Waveform.
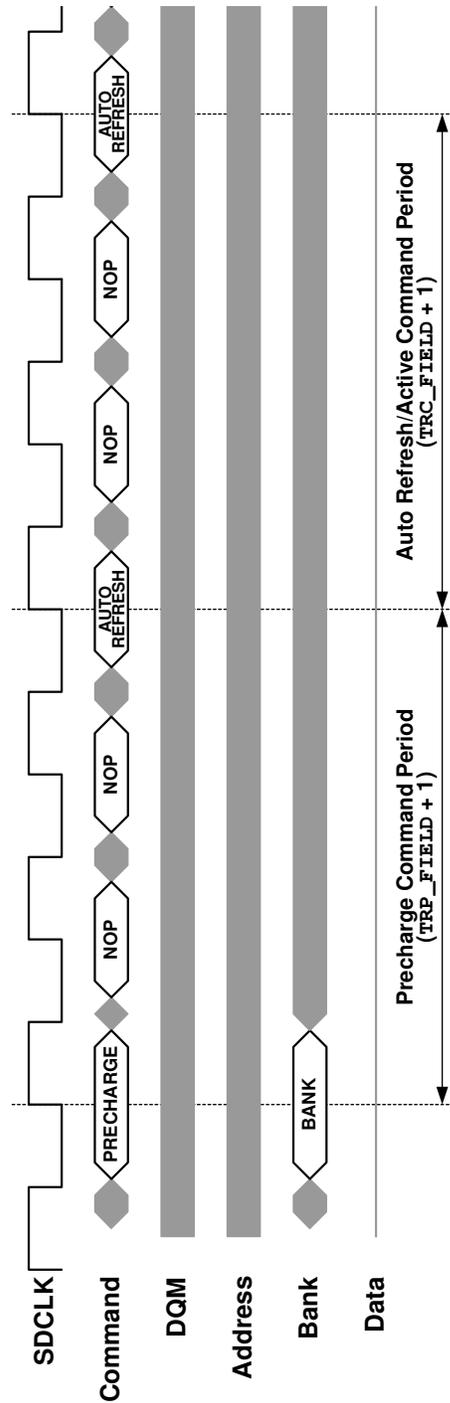
Figure 45. Example SDRAM Auto-Refresh Waveform.

SDRAM Mode Register (MSS_SDR_MODE_REG)

This register configures the different operating modes and timing of the SDRAM subsystem. FastChip Device Link automatically configures these options. However, the timing values may be modified when changing operating frequency or when entering or existing power-down mode. See Figure 44 and Figure 45 for waveforms showing timing values.

Bit	Description/Function
31:30	Reserved
29:23	**SDRAM Mode Register (MODE_REG_FIELD [13:7]):** ***Set these bits to 0000100'b to program the external SDRAM for single location write access.*** This field defines the content of the SDRAM mode register loaded into each external SDRAM device at power-on or device reset. The mapping to pins is as follows: MODE_REG_FIELD[13:0] = {BS[1:0], A[11:0]} The SDRAM controller expects a pre-defined mode of operation (sequential bursts of 4 and single writes). The only programmable value is the CAS latency. This field is applied to the pins during the LOAD MODE REGISTER command.
22:20	**SDRAM Mode Register, CAS Latency (MODE_REG_FIELD [6:4]):** These bits control the CAS latency. The legal values are shown below. 010: 2-cycle latency (most applications use this value) 011: 3-cycle latency
19:16	**SDRAM Mode Register (MODE_REG_FIELD [3:0]):** ***Set these bits to 0010'b.*** These bits define the SDRAM burst access. The SDRAM must always be configured for sequential bursts of four.
15	Reserved
14:12	**Refresh/Active Period (TRC_FIELD [2:0]):** This field defines the refresh period to be applied by the SDRAM controller. The number of clock cycles applied to the refresh cycle is (TRC_FIELD+1). This value also represents the active cycle. However, it is only used for refresh. The active cycle is a function of the minimum SDRAM cycle (precharge + activate + command ~ 5 clock cycles).
11	Reserved
10:8	**Write Recovery Time (TWR_FIELD [2:0]):** This field defines the write recovery time applied by the SDRAM controller. The number of clock cycles applied is equal to (TWR_FIELD+1). The write recovery time is applied between a write and a precharge command.
7	Reserved
6:4	**Command Delay Period (TRCD_FIELD [2:0]):** This field defines the number of clocks to be applied between an active command and a read/write command. The period, measured in clock cycles, is equal to (TRCD_FIELD+1).
3	Reserved
2:0	**Precharge Period (TRP_FIELD [2:0]):** This field defines the precharge period to be applied by the SDRAM controller. The number of clock cycles applied to the precharge cycle is (TRP_FIELD+1).

Configuration reset value: 0x02223222

SDRAM Control Register (MSS_SDR_CTRL_REG)

This register controls and configures the SDRAM controller state and the refresh mechanism.

Bit	Description/Function
31:28	**Refresh Burst Size (RFSH_BURST_FIELD [3:0]):** This field specifies the number of auto refresh bursts, minus one, to generate per refresh request. Load burst size *minus 1* in this field.
27:16	**Refresh Rate (RFSH_RATE_FIELD [11:0]):** The refresh rate specifies the desired number of clock periods between two refreshes. This field is loaded into the Refresh Interval Counter. Loading 0 in this field disables the SDRAM refresh.
15:3	Reserved
2:0	**SDRAM Power Management (PWR_MAN_FIELD [2:0]:** (see Table 33)

System reset value: 0

Table 33. SDRAM Power Management Mode Settings.

SDRAM Power Management Mode	Setting
SDRAM disable	000
SDRAM normal operation	010
SDRAM normal operation with stand-by	011
Power down (self-refresh) mode. In this mode, the SDRAM retains its data.	100

MSSIU Status Register (MSS_STATUS_REG)

This register is read only.

Bit	Description/Function
31:4	Reserved
3	**Refresh Request Counter Overflow (RFSH_OVF_BIT):** The refresh request counter is limited to seven pending requests. However, an overflow condition is very unlikely, possibly resulting from an excessively high refresh rate.
2:0	**SDRAM Controller Status (SD_STATUS_FIELD [2:0]:** (see Table 34)

System reset value: 1

Table 34. SDRAM Controller Status.

Setting	SDRAM Controller Status
001	SDRAM controller is disabled
010	SDRAM controller is in self-refresh mode
100	SDRAM controller is in stand-by mode
000	SDRAM controller is in normal operation

MSSIU Status Clear Register (MSS_STATUS_CLEAR_REG)

This register is write only.

Bit	Description/Function
31:4	Reserved
3	**Clear Refresh Request Counter Overflow (RFSH_OVF_CLR_BIT):** Write a '1' to this bit to clear the refresh request overflow bit in the status register.
2:0	Reserved

System Performance

Maximizing System Performance

The techniques outlined in Table 35 guarantee maximum system using an A7S CSoC device. Some techniques incur a cost penalty, such as requiring SDRAM. Others use features already available on every A7S device.

In general, wider external memory data paths boost perform by reducing the number of access per each instruction fetch or data transfer.

Table 35. A7S Performance Optimization Techniques.

Feature	Performance Enhancement	Cost Penalty/Benefit
Code Execution		
Cache	When enabled, cached instructions are fetched in a single clock cycle.	None. Cache available on every A7S device.
Internal SRAM	Copy and execute critical code from internal SRAM. Instructions are fetched in a single clock cycle.	Reduces available internal SRAM for data storage.
SDRAM, Cache	Copy code from Flash to SDRAM and execute from SDRAM. Performance enhancement is greatest for a 32 bit wide SDRAM interface. SDRAM interface optimized for cache fills.	Requires external SDRAM but reduces the cost of external Flash. Use a slow, cheap, 8-bit Flash to store code.
CPU-to-external memory pipeline	Guarantees system performance when operating above 40 MHz.	None.
Thumb mode	If system cost goals requires executing directly from a byte-wide or 16-bit wide Flash, Thumb mode instructions improve system performance by reducing number of memory accesses for each instruction fetch.	May reduce overall system cost while maintaining moderately high performance.
Data Transfers		
DMA	Use DMA to transfer blocks of memory, offloading the CPU. DMA descriptor mode allows DMA to execute complex transfers. Ideal for transfers between memory and device functions implemented in CSL logic.	Each DMA channel competes for bandwidth on the CSI bus.
DMA, Internal SRAM	Use half of internal SRAM as a DMA data buffer. DMA accesses to internal SRAM are zero wait-state. CPU accesses other half of internal SRAM without wait-states. Ideal for "ping-pong" data buffers.	Reduces available internal SRAM for data or code storage.
DMA, SDRAM	The SDRAM interface is optimized for DMA transfers. Enable transfer buffer for best performance.	

CPU Code Execution

The ARM7TDMI CPU can execute code from any of the following resources.

- Scratchpad RAM
- Cache
- External Static Memory (Flash)
- External SDRAM
- Over the CSI bus

Code execution is optimized when fetching from the Scratchpad RAM, the cache, or the Flash subsystem operating at lower speeds. Code execution implies code fetch and data read or write transactions by the CPU.

Executing from Scratchpad RAM or Cache

The CPU executes from the Scratchpad RAM and Cache at full speed, without any wait states, regardless of sequential or non-sequential (branching) accesses.

This is true for code fetching and data read or write accesses.

Additional Pipeline Delay at Higher Frequencies

To guarantee performance when operating at higher frequencies, an additional pipeline delay is inserted between the CPU memory requests and the external memory interface when operating at 40 MHz or more.

This pipeline only increases latency during cache fills from external memory and when executing code from external memory.

Executing from External Flash or SRAM

Executing code from external static memory, typically Flash or SRAM, is optimized for lower operating frequencies. Table 36 shows the number of bus cycles required when executing from external Flash or SRAM, by access or instruction type. Values are shown when executing with and without the additional high-performance pipeline delay. The values shown in Table 36 assume a 32-bit static memory interface with Flash or SRAM memory capable of operating with single-cycle accesses.

Table 36. Bus Cycles When Executing from External Flash or SRAM.

Access Type	No Pipeline	Pipeline Enabled
Non-sequential	2	3
Sequential	1	2
Load/store multiple	1	2

Table 40 presents similar data, but for other static memory interface data widths and memory speeds.

Executing from External SDRAM

Table 37 shows the number of bus cycles required when executing from external SDRAM, by access or instruction type. Values are shown when executing with and without the additional high-performance pipeline delay. The values shown in Table 37 assume a 32-bit SDRAM interface, a CAS latency of 2, and that all accesses are to the same SDRAM page—no pre-charge and activation required.

Table 37 shows that four bus cycles are required when reading from external SDRAM, without the pipeline enabled. This number includes one cycle for request generation, one cycle for the SDRAM command generation, and two cycles for SDRAM read data latency.

Table 37. Bus Cycles When Executing from External SDRAM.

Access Type	No Pipeline	Pipeline Enabled
Read	4	5
Write	2	3

Table 41 presents similar data, but for other SDRAM memory interface data widths.

Cache Fills

The cache can only be filled from external memory, and not from internal SRAM or CSI memory locations. Only memory located in the SDRAM and External Flash address regions between 0xC000_0000 and 0xD100_0000, or their aliases, can be cached.

Memories connected through the CSL or external memory request mechanism cannot be used to fill the Cache.

Each cache line is four words.

Cache Fills from External Static Memory

Table 38 shows the number of bus cycles required to fill a cache line from external static memory. Values are shown with and without the additional high-performance pipeline delay. The values shown in Table 38 assume a 32-bit static memory interface with Flash or SRAM memory capable of operating with single-cycle accesses.

Table 38. Bus Cycles When Filling Cache Line from External Static Memory (Flash, SRAM).

No Pipeline	Pipeline Enabled
7	8

Table 42 presents similar data, but for other memory interface data widths and static memory speeds.

Cache Fills from External SDRAM

Table 39 shows the number of bus cycles required to fill a cache line from external SDRAM memory. Values are shown with and without the additional high-performance pipeline delay. The values shown in Table 39 assume a 32-bit SDRAM interface, a CAS latency of 2, and that all accesses are to the same SDRAM page—no pre-charge and activation required.

Table 39. Bus Cycles When Filling Cache Line from External SDRAM.

No Pipeline	Pipeline Enabled
9	10

DMA Transfers

DMA transfers are performed over the Configurable System Interconnect (CSI) bus.

Un-buffered DMA transactions to or from the external static memory or SDRAM memory subsystems incur one wait state per transaction.

Buffered DMA transactions provide efficient transfers with the SDRAM subsystem.

DMA Transfer to/from Scratchpad RAM

DMA transfers to or from the Scratchpad occur without wait states.

Because each transaction occurs in a single cycle, the A7S is capable up supporting 228 Mbytes per second data rates to or from Scratchpad while operating at 60 MHz.

DMA Transfer to/from External Static Memory

DMA transfers to or from external memory are optimal, assuming a 32-bit, single-cycle external Flash or SRAM memory and buffered DMA transactions.

A single-channel or four-channel DMA write occurs in a single bus cycle.

A single-channel or four-channel DMA read also occurs in a single bus cycle, but with some initial latency for each channel.

DMA Transfer to/from External SDRAM

DMA transfers to or from external SDRAM memory are nearly optimal, assuming a 32-bit interface to external SDRAM memory with a CAS latency of two cycles, with all transactions to the same SDRAM page, and buffered DMA transactions.

A single-channel or four-channel DMA write occurs in a single bus cycle, with some initial latency.

Optimizing SDRAM Performance

When partitioning SDRAM memory between the CPU code and transfer buffers for the various DMA channels, place the data in different SDRAM banks to avoid the overhead of constant pre-charge and activate cycles.

For example, place the CPU code within in the first SDRAM internal bank, while DMA channels 0 and 1 transfer data to or from SDRAM internal Bank 1 and Bank 2 respectively.

External Memory Turn-Around Cycles

A read transaction to external memory followed by a write transaction to the same external memory incurs one dead cycle to avoid contention on the A7S's data lines.

Table 40. CPU Code Fetch Performance by Static Memory Interface Type.

| | | Flash/SRAM Memory Subsystem Data Width | | | | | |
| | | 8-bit | | 16-bit | | 32-bit | |
		Pipeline disabled	Pipeline enabled	Pipeline disabled	Pipeline enabled	Pipeline disabled	Pipeline enabled
Byte	Sequential						
	Non-sequential	1 + t	2 + t	1 + t	2 + t	1 + t	2 + t
Half-word	Sequential	2t	1 + 2t	t	1 + t	t	1 + t
	Non-sequential	1 + 2t	2 + 2t	1 + t	2 + t	1 + t	2 + t
Word	Sequential	4t	1 + 4t	2t	1 + 2t	t	1 + t
	Non-sequential	1 + 4t	2 + 4t	1 + 2t	2 + 2t	1 + t	2 + t

t = Flash or SRAM access time in clock cycles.

Table 41. CPU Code Fetch Performance by SDRAM Memory Interface Type (CAS Latency 2).

| | | SDRAM Memory Subsystem Data Width | | | | | |
| | | 8-bit | | 16-bit | | 32-bit | |
		Pipeline disabled	Pipeline enabled	Pipeline disabled	Pipeline enabled	Pipeline disabled	Pipeline Enabled
Byte	Read	4	5	4	5	4	5
	Write	2	3	2	3	2	3
Half-word	Read	5	6	4	5	4	5
	Write	2	3	2	3	2	3
Word	Read	7	8	5	6	4	5
	Write	2	3	2	3	2	3

Table 42. Cache Fill Performance by Memory Interface Type.

| | Memory Subsystem Data Width | | | | | |
| | 8-bit | | 16-bit | | 32-bit | |
	Pipeline disabled	Pipeline enabled	Pipeline disabled	Pipeline enabled	Pipeline disabled	Pipeline Enabled
Cache fill from Flash	3 + 16t	4 + 16t	3 + 8t	4 + 8t	3 + 4t	4 + 4t
Cache fill from SDRAM	21	22	13	14	9	10

t = Flash or SRAM access time in clock cycles.
SDRAM CAS latency 2.
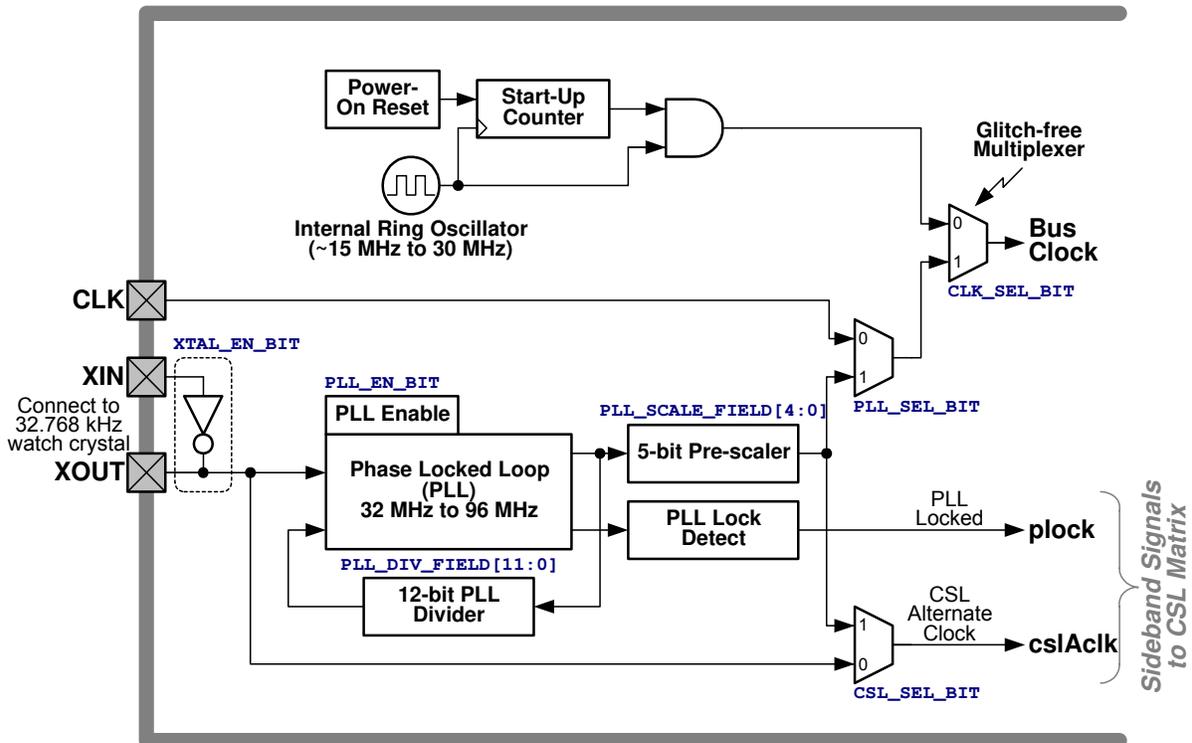
Figure 46. A7S clock control circuit, including PLL clock synthesizer, crystal oscillator amplifier, and internal ring oscillator.

Clocking

Clock Operation

The system clock, which drives most of the system, has three possible sources. The clock source is selectable through the *Clock Control Register*.

1. Internal ring oscillator

2. External clock source

3. Phase-Locked Loop (PLL) output

The internal ring oscillator the default clock source at power-up. The ring oscillator generates a clock with a typical frequency of 20 MHz. However, due to process variations as well as varying operating conditions, the frequency will vary between 15 MHz to 30 MHz.

The system clock frequency can also be synthesized from an external 32.768 kHz watch crystal, shown in Figure 47, using the phase-locked loop (PLL) circuit.
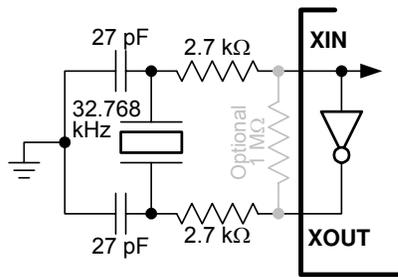
Figure 47. Crystal oscillator circuit.

Optionally, an external clock source can be applied to the dedicated external clock pin, CLK.

When not used, the PLL and crystal oscillator should be shut down so that they consume no power. Likewise, the six CSL global buffers can be shutdown to save power or to freeze functions during debugging.

Finally, an additional clock is provided to the CSL by a sideband signal. This signal originates either from the output of the PLL or the output of the 32.768 kHz crystal oscillator.

FastChip Device Link (FDL) automatically configures the clock source and the PLL based on your settings when you create a configuration image. The specified settings are applied when the device powers up or is re-initialized. However, application code can change the values dynamically as potentially required when entering or exiting power-down mode.

FastChip Device Link

PLL

The phase-locked loop receives a 32.768-kHz input from the crystal oscillator and generates a frequency-programmable clock output in steps of 32.768 kHz, as shown in Equation 1. The output frequency from the PLL ranges from 32 MHz to 96 MHz at 2.5V. The PLL output must be divided down by the pre-scaler before driving the bus clock input to the CSoC device. It acquires lock from a dead start within a few milliseconds.

$$\text{Bus Clock Frequency} = \frac{\text{PLL\_DIV\_FIELD} \bullet 32.768\text{kHz}}{\text{PLL\_SCALE\_FIELD}} \qquad (1)$$

where:

PLL_DIV_FIELD = PLL divider count, defined in the Clock Control Register. This value must be between 1,000 and 3,000, inclusive.

PLL_SCALE_FIELD = PLL pre-scaler value, defined in the Clock Control Register, either 1, 2, 4, 8, 16, or 32.

The output of the PLL (VCO output) is divided down, and the output of the divider is connected to the PLL feedback input. The divider provides 12-bits of divide count. To provide a wider range of system clock frequencies, the PLL output can be further divided

down by a 5-bit pre-scaler. The pre-scaler provides the following set of scaling values: 1, 2, 4, 8, 16 and 32. The resulting range of frequency that can be synthesized from a 32.768 kHz crystal is 1 MHz to 60 MHz—the maximum operating frequency of the device.

A lock detector indicates when the PLL achieves phase lock. Anytime the frequency divide count is changed or the PLL output is selected, the application code must first switch to the internal ring oscillator, as described below. Then, the application must wait for the PLL re-acquire lock by monitoring the PLL status register. The status register contains both a "not locked" flag and "locked" flag. The "not locked" flag is set when the PLL circuit loses phase lock. The "locked" flag is set whenever the PLL circuit locks to the specified frequency. The user application program can clear these flags.

The output of the PLL can drive the system clock and can separately drive the CSL matrix through the alternate clock sideband signal, ACLK. When an external clock source on the CLK drives the bus clock, the PLL is bypassed completely.

The PLL consumes no power when shut down.

Changing PLL Clock Frequency or Switching Clocks

Some applications might dynamically change the PLL clock frequency to reduce system power or to use another base clock frequency.

The A7S's clock structure allows for this by providing a glitch-free clock multiplexer as shown in Figure 46. The procedures below describe how to switch PLL frequencies.

1. If the PLL is operating slower than 30 MHz, modify the memory interface timing values to operate from the internal ring oscillator at the maximum oscillator frequency of 30 MHz.

2. If using SDRAM and the PLL is operating faster than 15 MHz, modify the SDRAM refresh timer for the internal ring oscillator's minimum frequency of 15 MHz.

3. Write the CLK_SEL_BIT with '0', switching to internal ring oscillator.

4. Modify the PLL parameters for the new desired frequency.

5. Write a '1' to the PLL_LOCK_CLEAR_BIT to reset the PLL Locked Flag.

6. Monitor the PLL_LOCK_BIT and wait for it to be set, indicating that the PLL is locked to the specified frequency.

7. Once the PLL_LOCK_BIT is set, write the CLK_SEL_BIT with '1', switching back to the PLL clock synthesizer.

8. Modify the memory interface and SDRAM timing values for the new PLL clock frequency.

Changing Clock Sources

There are three potential clock sources including the internal ring oscillator, the PLL clock synthesizer, and an external clock source. The following procedure describes how to change from one source to another.

Switching to the Internal Ring Oscillator

1. Modify the memory interface timing values to operate from the internal ring oscillator at the maximum oscillator frequency of 30 MHz.

2. Modify the SDRAM refresh timer for the internal ring oscillator's minimum frequency of 15 MHz.

3. Write the CLK_SEL_BIT with '0', switching to internal ring oscillator.

Switching Between the PLL and the CLK Input Pin

1. If the current clock source is slower than 30 MHz, modify the memory interface timing values to operate from the internal ring oscillator at the maximum oscillator frequency of 30 MHz.

2. If using SDRAM and the current clock sources is faster than 15 MHz, modify the SDRAM refresh timer for the internal ring oscillator's minimum frequency of 15 MHz.

3. Write the CLK_SEL_BIT with '0', switching temporarily to the internal ring oscillator.

4. If the CLK pin is the desired clock source, write a '0' to the PLL_SEL_BIT. If the PLL is the desired clock source, write a '1' to the PLL_SEL_BIT.

5. If the new clock source is the PLL, modify the PLL parameters for the new desired frequency.

    a. Write a '1' to the PLL_LOCK_CLEAR_BIT to reset the PLL Locked Flag.

    b. Monitor the PLL_LOCK_BIT and wait for it to be set, indicating that the PLL is locked to the specified frequency.

6. Switch back to the PLL or CLK input pin by writing a '1' to the CLK_SEL_BIT.

7. Modify the memory interface and SDRAM timing values for the new clock source.

Alternate Clock Sideband Signal

The alternate clock sideband signal, ACLK, is only available to the CSL matrix and provides the direct output from the 32.786 kHz crystal oscillator or the output from the PLL pre-scaler, as shown in Figure 46.

The alternate clock signal, shown in Figure 48, is typically used during power-down mode where a small "time-keeper" function in the CSL matrix operates directly from the 32.687 kHz crystal oscillator and wakes the A7S device periodically.
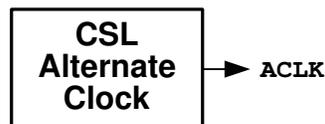
Figure 48. The ACLK sideband signal drives directly and only into the CSL matrix.

Clock and PLL Control Registers Description

System control Memory Map

Address		Register Name	Access
Base	Offset		
SYS_BASE	+ 0x00	Clock Control	R/W
	+ 0x04	PLL Status	R
	+ 0x08	PLL Status Clear	W

Triscend</image>

Clock Control Register *(SYS_CLOCK_CONTROL_REG)*

This register controls the PLL frequency and other clock-related options. FastChip Device Link automatically configures most of these options.

Bit	Description/Function
31:29	Reserved
28:24	**PLL Pre-Scaler Value (PLL_SCALE_FIELD [4:0]):** 00000: PLL output (no divider) 00001: PLL output ÷ 2 00010: PLL output ÷ 4 00100: PLL output ÷ 8 01000: PLL output ÷ 16 10000: PLL output ÷ 32
23:20	Reserved
19:8	**PLL Divider Count (PLL_DIV_FIELD [11:0]):** This field holds the divide count for the PLL divider. The PLL output frequency is determined as follow. The recommended range for this value is between 1,000 and 3,000, inclusive. PLL Frequency = PLL_DIV_FIELD x 32.768 kHz
7	Reserved. Must be 0
6	Reserved. Must be 0.
5	**PLL Enable (PLL_EN_BIT):** The PLL consumes no power when disabled. 0: Disable PLL 1: Enable PLL
4	**Crystal Oscillator Enable (XTAL_EN_BIT):** The crystal oscillator consumes no power when disabled. 0: Disable 1: Enable
3	**CSL Global Clocks Enable (CK_EN_BIT):** 0: Disable all six global buffers, GBUF5:0 1: Enable all six global buffers, GBUF5:0, into the CSL
2	**CSL Clock Source Select (CSL_SEL_BIT):** This bit selects which clock drives the CSL alternate clock sideband signal. 0: Output of crystal oscillator 1: PLL pre-scaler output
1	**PLL Output Mux Select (PLL_SEL_BIT):** This bit selects the clock source for PLL output multiplexer. 0: External clock source on the CLK pin 1: PLL pre-scaler output
0	**Clock Source Select (CLK_SEL_BIT):** This bit selects the clock source for the internal system clock. 0: Internal ring oscillator 1: PLL output multiplexer

Configuration reset value: 0x00040000

PLL Status Register *(SYS_PLL_STATUS_REG)*

This register is read-only.

Bit	Description/Function
31:2	Reserved
1	**PLL Locked Flag (PLL_LOCK_BIT):** A "1" indicates that the PLL achieved lock permanently or temporarily.
0	**PLL Not Locked Flag (PLL_NOT_LOCK_BIT):** A "1" indicates that the PLL has *lost* lock permanently or temporarily.

Configuration reset value: 1

PLL Status Clear Register *(SYS_PLL_STATUS_CLEAR_REG)*

This register is write only.

Bit	Description/Function
31:2	Reserved
1	**Clear PLL Locked Flag (PLL_LOCK_CLEAR_BIT):** Writing a '1' to this bit clears the **Lock** bit in the PLL Status Register. Writing '0' has no effect.
0	**Clear PLL Not Locked Flag (PLL_NOT_LOCK_CLEAR_BIT):** Writing a '1' to this bit clears the **NotLock** bit in the PLL Status Register. Writing '0' has no effect.

Reset

The A7S is reset to a pre-defined state via four potential reset types.

1. Power-on reset

2. Configuration reset by asserting the RST- pin

3. System reset by asserting the AppRst sideband signal or by a watchdog timer reset.

4. CPU reset, which is only generated from the JTAG interface during debugging.

Reset Hierarchy

Not all reset conditions are equal in an A7S system. There is a hierarchy within the different reset types as highlighted in Figure 49. For example, any condition that causes a Configuration Reset, also generates a System Reset and a CPU Reset. The various conditions that cause a reset event are shown on the left-hand side of the diagram. A reset event also sets specific bits within the **_Reset Status Register_**, as indicated in the diagram..
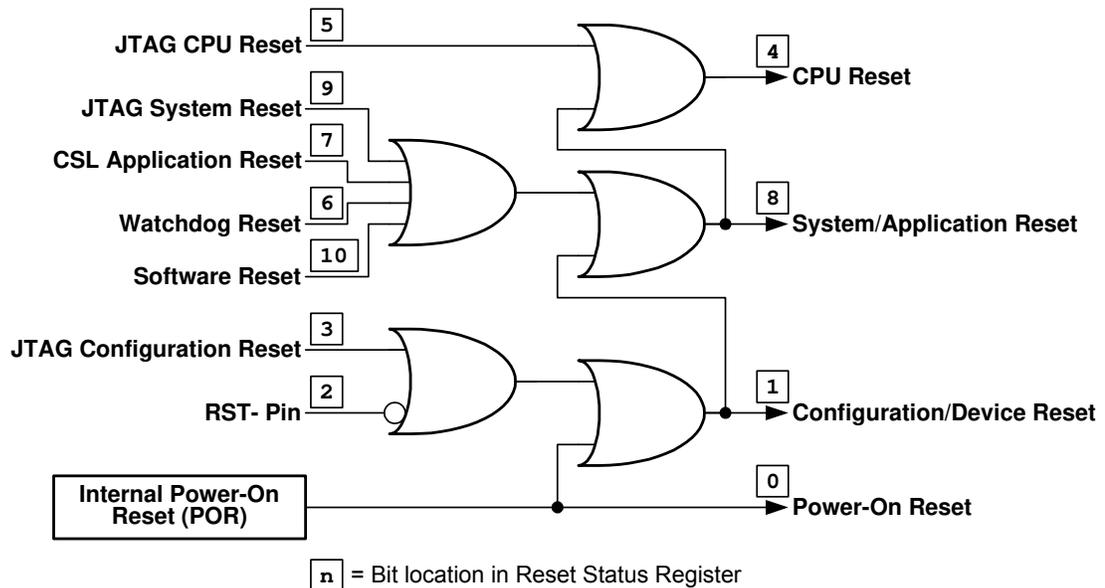
Figure 49. A7S Reset Hierarchy.

Power-On Reset (POR)

The **power-on reset** is the ultimate reset condition that asynchronously places the entire device, including the JTAG port, into a known state. At power-up, a status bit in the **_Reset Status Register_** is set indicating that a power-on condition occurred. Following a power-on reset, the device will initialize itself. The secondary initialization code, created by the FastChip development system, resets this bit during the initialization process. After power-up, software can disable the power-on logic to save power.

At power-on, the system begins operating from the internal ring oscillator. To allow the CSoC system to stabilize, the ring oscillator is temporarily blocked to the remainder of the system for 16 ring oscillator cycles. At the end of 16 cycles, the ring oscillator clock is supplied to the remainder of the system.

Configuration Reset

Next in the hierarchy is the configuration reset. It forces a system configuration by reset-ting everything except the JTAG unit. A configuration reset forces the internal ring oscilla-tor as the active clock source, resets the memory interface timing values, and causes the A7S device to re-initialize. The configuration reset is activated directly through the RST-reset pin or via a JTAG Configuration Reset during debugging. Following a configuration reset, the memory map is reset with all aliases at address 0.

System Reset

Next in the hierarchy is the system reset, which only resets the user application and does not affect configuration information such as the CSL matrix, PIO pins, clock settings, and static settings in the memory subsystem. It resets the CPU, the CSI bus, and all the pe-ripherals. This reset is activated through the CSL application reset sideband signal, by the watchdog reset, or through software (SYS_RESET_BIT). Following a system reset, the memory map is reset with all aliases at address 0, except for internal primary initializa-tion ROM alias, which is disabled.

CPU Reset

Finally, the CPU reset generates an ARM reset exception, which forces the ARM7TDMI processor to clear itself and start executing from address 0. A CPU Reset is only applied through the JTAG interface for debugging purposes.

Reset Sideband Signals

Logic in the CSL matrix can observe when a System Reset occurs or force a System Re-set condition via the AppRst and SysRstN sideband signals, shown in Figure 50. A CSL function generates a System Reset by asserting the AppRst signal High.

When a System Reset occurs, for any reason, the SysRstN signal is asserted Low. The SysRstN signal is typically used to clear CSL-based registers, along with the remainder of the system.
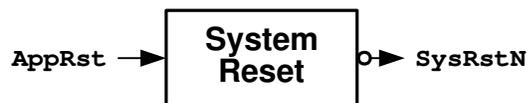
Figure 50. System Reset sideband signal connections.

During initialization, a System Reset occurs just before the system begins executing the application program.

Reset Control Registers Description

Reset Control Memory Map

Address		Register Name	Access
Base	Offset		
REMAP_BASE	+ 0x30	Reset Status	R
	+ 0x34	Reset Status Clear	W
SYS_BASE	+ 0x0C	Reset Control	R/W

Reset Control Register (SYS_RESET_CONTROL_REG)

Bit	Description/Function
31:3	Reserved
2	**System Reset Enable (SYS_RESET_BIT):** This bit is used by application software to generate a System Reset. It resets the CPU, the bus, and all the peripherals without affecting the configuration of the device. The bit is self-clearing. 0: No effect. 1: Generate a system reset.
1	**Slave Pin Reset Disable (SLAVE_DIS_BIT):** This function is used only for testing. Set this bit to 0.
0	Reserved

Configuration reset value: 0

Reset Status Register (REMAP_RESET_STATUS_REG)

This register indicates the cause of the most-recent reset. This register is read-only.

Once a status bit is set, it remains set until cleared by software.

Bit	Description/Function
31:11	Reserved
10	**Software System Reset Flag (SOFT_RST_BIT):** This bit is set whenever system software generates an A7S system reset by setting the SYS_RESET_BIT.
9	**JTAG System Reset Flag (J_SYS_RST_BIT):** This bit is set whenever the JTAG controller issues a system reset.
8	**System Reset Flag (SYS_RST_BIT):** This bit is set whenever an A7S system reset occurs, for any reason, including a configuration reset.
7	**CSL Application Reset Flag (APP_RST_BIT):** This bit is set whenever the system is reset via the application reset sideband signal, AppRst, from the CSL logic.
6	**Watchdog Reset Flag (WD_RST_BIT):** This bit is set whenever a watchdog timer reset occurs.
5	**JTAG CPU Reset Flag (J_CPU_RST_BIT):** This bit is set whenever the JTAG controller generates a CPU reset.
4	**CPU Reset Flag (CPU_RST_BIT):** This bit is set whenever an ARM processor reset occurs, for any reason, including a configuration or system reset.
3	**JTAG Configuration Reset Flag (J_CFG_RST_BIT):** This bit is set whenever a JTAG configuration reset command occurs.
2	**RST- Pin Flag (RST_PIN_BIT):** This bit is set whenever the device reset pin, RST-, is asserted.
1	**Configuration Reset Flag (CFG_RST_BIT):** This bit is set whenever a configuration reset condition occurs.
0	**Power-On Reset Flag (POR_BIT):** This bit is set whenever a power on reset occurs.

Default reset value: Depends on reset source.

Reset Status Clear Register (REMAP_RESET_STATUS_CLEAR_REG)

This register is used to clear the bits from the **_Reset Status Register_**. This register is write only. Software can clear any bits of the *Reset Status Register* by writing to the corresponding *Reset Status Clear Register* location. Writing a 1 in to a corresponding bit clears the status. Writing 0 has no effect.

Bit	Description/Function
31:11	Reserved
10	**Clear Software System Reset Flag (SOFT_RST_CLR_BIT):** 0: No effect 1: Clear bit location
9	**Clear JTAG System Reset Flag (J_SYS_RST_CLR_BIT):** 0: No effect 1: Clear bit location
8	**Clear System Reset Flag (SYS_RST_CLR_BIT):** 0: No effect 1: Clear bit location
7	**Clear CSL Application Reset Flag (APP_RST_CLR_BIT):** 0: No effect 1: Clear bit location
6	**Clear Watchdog Reset Flag (WD_RST_CLR_BIT):** 0: No effect 1: Clear bit location
5	**Clear JTAG CPU Reset Flag (J_CPU_RST_CLR_BIT):** 0: No effect 1: Clear bit location
4	**Clear CPU Reset Flag (CPU_RST_CLR_BIT):** 0: No effect 1: Clear bit location
3	**Clear JTAG Configuration Reset Flag (J_CFG_RST_CLR_BIT):** 0: No effect 1: Clear bit location
2	**Clear RST- Pin Flag (RST_PIN_CLR_BIT):** 0: No effect 1: Clear bit location
1	**Clear Configuration Reset Flag (CFG_RST_CLR_BIT):** 0: No effect 1: Clear bit location
0	**Clear Power-On Reset Flag (POR_CLR_BIT):** 0: No effect 1: Clear bit location

Power Down Mode

A power saving mode allows an application to selectively shut down portions of the device. Different power-saving levels are possible, all the way to complete power down. Application code shuts down selected functions via the ***Power Down Control Register***, choosing which feature to turn off before the actual power-down event. Other portions of the system are turned-off dynamically through hardware upon a power down command. Because the device is a fully static CMOS integrated circuit, the only power consumed in the lowest-power mode is from substrate leakage, which is about 100 µA.

Table 43 summarizes the different functions that can be independently shut down. The items marked as "Software Shut Down" are selectively disabled by application code prior to setting the PD_BIT. When the PD_BIT is set, the items marked as "Dynamic Shut Down" take effect. For example, the power-down behavior for each PIO pin can be individually configured. Likewise, the system clock can be shut off to the system bus or the CSL logic.

Table 43. Functions with Independent Power-Down Control.

Feature	Software Shut Down	Dynamic Shut Down
Power-On Reset (POR)	✓	
PLL	✓	
Crystal Oscillator Amplifier	✓	
Internal Ring Oscillator		✓
PIO pins		✓
System clock		✓
CSL user system clock		✓
CSL clocks (ENCK)	✓	
SDRAM	✓	

If an application uses an external crystal to drive the system clock and wants to shut down the crystal oscillator amplifier, then the application software must first switch to the internal ring oscillator and then disable the crystal oscillator. When exiting power down mode, the internal ring oscillator drives the system clock and the application code must re-enable the crystal oscillator and wait for it to stabilize before switching the clock source back to the crystal oscillator.

In most applications, the crystal oscillator feeds the PLL clock synthesizer circuitry, as shown in Figure 46. If the PLL is the primary clock source, shutting down the crystal oscillator also affects the PLL. Similarly, if the application shuts down the PLL prior to entering power-down mode, application software must first switch to the internal ring oscillator, then disable the PLL circuit.

Some applications may decide to leave the 32.768 kHz crystal oscillator active during power down. The output of the crystal oscillator is available to the CSL matrix on the alternate clock sideband signal, ACLK. This alternate clock signal can drive a counter that, after a pre-determined amount of time, wakes the A7S CSoC device from power-down by generating an interrupt using one of the CSL IRQ sideband signals. Because this counter operates are a relatively low frequency and uses only a few CSL cells, it consumes very little power while active.

The standard ARM ***Pause Register*** shuts down the ARM7TDMI processor only. If system level shutdown is required, the Pause Register can be ignored because the processor also enters a low power state when the system clock is shutdown.

Entering power down mode

Power down mode is initiated by writing a '1' to the power down bit (PD_BIT) of the ***Power Down Control Register***. Depending on the portions of the device selected for shut down, the effect is immediate. One case, however, requires additional application code.

When the external crystal drives the internal system clock, merely shutting it down is not sufficient. If the application expects to fetch instructions from external memory when exiting power down, there are potential timing problems if the frequency of the internal ring oscillator is greater than the frequency of the external crystal. Before switching to the internal ring oscillator, the application code should ensure that the external-memory timing settings are correct by reprogramming the memory interfaces to more conservative values. Likewise, if using SDRAM and if the minimum internal ring oscillator frequency is slower than the external crystal, then modify the SDRAM refresh rate accordingly.

The following steps illustrate the complete power down sequence.

1. Shut-off or wait for any DMA transfers to complete

2. If executing from the SDRAM, branch to either internal SRAM or external Flash

3. Turn-off the SDRAM memory subsystem if there is no need to retain data. This forces the SDRAM self-refresh command and powers it down

4. Slow down external static memory timing if necessary

5. Switch to the internal ring oscillator

6. Turn-off the POR logic by writing a '1' to the POR_DIS_BIT

7. Turn-off the PLL by writing a '0' to the PLL_EN_BIT

8. Turn-off the crystal oscillator by writing a '0' to the XTAL_EN_BIT

9. If desired, turn-off global buffers to the CSL matrix by writing a '1' to CK_EN_BIT.

10. Select the following features for automatic shut-down during power-down mode when the PD_BIT is set. Select the internal ring oscillator for shut-down by writing a '1' to PD_OSC_EN_BIT. Select the PIOs for shut-down by writing a '1' to PD_IO_EN_BIT. Shut down the bus clock to the CSL matrix by writing a '1' to PD_CSL_BCK_EN_BIT. Shut down bus clock to the CPU and the CSI bus by writing a '1' to PD_BCK_EN_BIT.

11. Write a '1' to the power-down bit (PD_BIT) to shut-down the features selected in Step 10.

Exiting power down mode

There are two different application methods to exit of power down mode. The first method is to cause a system reset by asserting the application reset sideband signal, AppRst. In this case, the ***Power Control Register*** and the entire system are reset, but initialization data remains intact. The CPU starts executing from external Flash, and the application starts again.

The second method to exit power down mode is via an interrupt on one of the CSL interrupt sideband signals, IRQ2, IRQ1, or IRQ0. Actually, any interrupt causes the device to exit power down. Upon receiving an interrupt, the power down bit is asynchronously reset to "0". In turn, and enables all the circuits that were selected for power down.

Power-Down Control Registers Description

System control Memory Map

Address		Register Name	Access
Base	**Offset**		
SYS_BASE	+ 0x10	Power Down Control	R/W

Power Down Control Register (SYS_POWER_CONTROL_REG)

Application code defines which portions of the device are turned off or that are to remain active during power-down mode. If a particular bit is set, then the corresponding circuit is turned off during power down mode. A '0' keeps the circuit active at all times.

Bit	Description/Function
31:6	Reserved
5	**Power-On Reset Disabled during Power Down (POR_DIS_BIT):** 0: Power-on reset (POR) circuit active 1: POR circuit disabled. The circuit consumes no power in this mode.
4	**Power Down Mode Enable (PD_BIT):** 0: Device is fully active 1: Force device into power down mode
3	**Selected PIOs Disabled during Power Down (PD_IO_EN_BIT):** 0: All PIOs active during power down 1: Selectively disables PIOs during power down. Individual PIO controls set during design time
2	**Internal Ring Oscillator Disabled during Power Down (PD_OSC_EN_BIT):** 0: Active during power down 1: Disabled during power down
1	**Block BusClock Signal to CSL Matrix during Power Down (PD_CSL_BCK_EN_BIT):** 0: BusClock distributed to CSL matrix 1: BusClock blocked to CSL during power down
0	**Block BusClock to CPU and CSI Bus during Power Down (PD_BCK_EN_BIT):** 0: Bus Clock drives CPU and CSI bus 1: Bus Clock blocked to CPU and CSI bus during power down

System reset value: 0

System Initialization

Like most processors and SRAM-based programmable logic devices, the Triscend A7S requires initialization data to configure programmable options within the device before it becomes functional. The Triscend A7S offers two different system initialization options to best fit various target applications, as shown in Table 44. Both active and passive initialization methods are available.

In active initialization modes, the Triscend A7S provides the control signals, directing data transfers and controlling external devices.

In passive modes, an external controller directs data transfers.

Table 44. Initialization Modes.

Initialization Mode	Method	Data Source
Parallel	Active	Parallel static memory (typically Flash)
JTAG	Passive	Downloaded by intelligent host through JTAG port

System Initialization

System Initialization Overview

Figure 51 outlines the general parallel mode initialization procedure.

① Upon power-up, unless the chip is held in reset or the SLAVE- pin held Low, the CPU starts executing from the internal primary initialization boot ROM. The purpose of the primary initialization boot ROM is to locate the application's initialization data and code stored in the secondary boot device, usually held in external non-volatile memory.
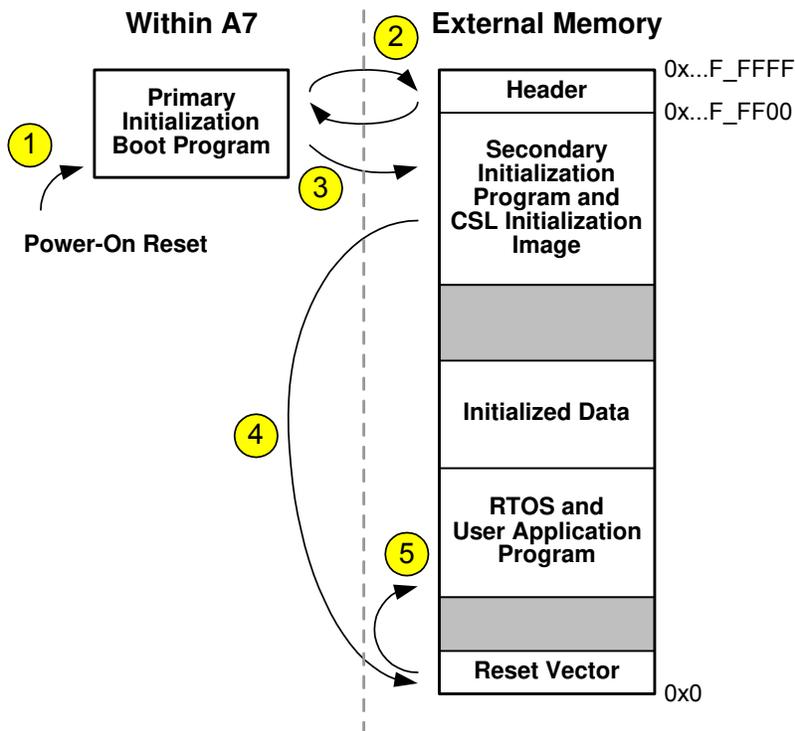
Figure 51. Parallel Mode Initialization Flow.

② A four-word header, located in the top 64 words of memory marks valid secondary initialization data.

③ After locating a valid header in external memory, the boot ROM program branches to the associated secondary initialization program.

The secondary initialization code configures the device by programming the CSL and PIO memory cells, and defines the basic control registers controlling the clock and memory subsystem static settings.

Once secondary initialization is complete, the initialization code enables the CSL matrix and disables the configuration mode.

④ The secondary initialization program then issues a System Reset, which causes the CPU to execute code located at its reset vector, location 0x0.

⑤ At this point, the program branches to the application code and begins executing.

> **FastChip Device Link** *FastChip Device Link (FDL) automatically creates a secondary initialization program whenever you create a configuration image. This information on initialization is provided as reference.*

The initialization process is triggered at power-up or whenever the RST- pin is asserted Low. The JTAG controller can also issue a JTAG Configuration Reset command during debugging.

During the initialization procedure, any un-initialized PIO pins are forced into a high-impedance state with a weak pull-up resistor. Even the system pins, when not driven, are in this weak pull-up mode.

During the initialization process, heavy activity occurs on the system pins such as the Flash control, address and data pins. One of the first operations performed by the boot ROM is to initialize the external SDRAM memories, regardless if any are actually attached. This initialization step is done to avoid possible contention on the external memory data bus. During the SDRAM setup procedure, the A7S actively drives the SDCLK, SDCKE, SDCE-[1:0] and A[20:19] pins. Although these pins are potentially reclaimable as user-defined PIO pins, the behavior of these pins at reset and start-up should be carefully considered.

Primary Initialization Procedure

Figure 52 illustrates the complete primary initialization process. When the A7S devices leaves the reset state, either after a power-on event or when the RST- pin is released, the embedded ARM7TDMI processor branches to the internal, primary initialization boot ROM. The internal ROM contains a small ARM7TDMI application program that guarantees an orderly device start-up and searches for an external, secondary initialization memory, typically an external Flash memory device.

The ARM7TDMI CPU programs each of the PIO pins so that they are high-impedance (i.e., floating) but with soft pull-up resistors. This causes the PIO pins to drift High.

Then, the CPU programs the external memory interface and sets all timing values to their slowest, default settings.

Out of Reset

Jump to boot ROM

Program all PIO memory cells
(low drive, slow slew rate)

Setup MSSIU in slave mode

Sample VSYS pin

VSYS
stable ?

Increment
trial count

Trial count
reached ?

Power down device

N

Y

Y

N

Compare VSYS sample
with original sample

Sample VSYS pin

Initialize SDRAMs

De-activate SDRAM controller in case it
was activated

Wait a minimum of 200us (the SDRAM
clock pin is enabled at reset)

Activate SDRAM controller

Setup external FLASH memory sub-system
width, recovering appropriate number of
data pins. Increment from 8-bit, to 16-bit, to
32-bit data width.

Header search

Search the top 64 words of FLASH for a
valid header. Headers are 4 words long and
are aligned to a 4-word boundaries. The
search ends after exhaustively detecting 16
incorrect headers. A valid header is found
if it has been successfullly found 16
consecutive times.

Increment
width

Increment success
count

N

Search
successful ?

Success count
reached ?

Y

N

Disable ROM alias

Y

Y

More possible
memory width
?

**Jump to secondary
initialization code**

N

VSYS Loop

Trial Loop

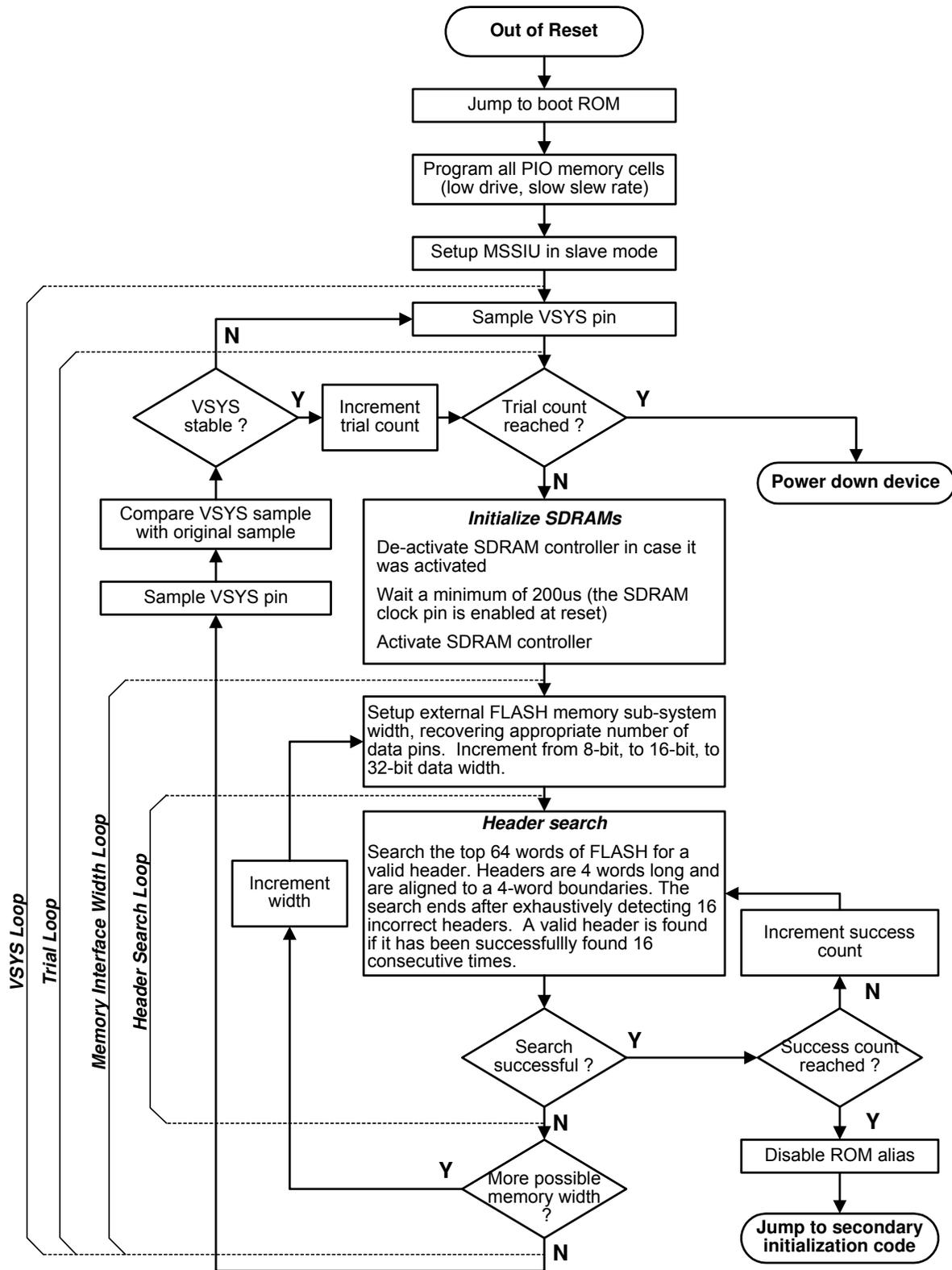*Memory Interface Width Loop*

Header Search Loop

**Figure 52.   The A7S executes its primary initialization boot procedure from internal ROM
at power-up or after a configuration reset.**

The CPU then samples the VSYS pin to see if the power supply to any external devices is stable. The external devices may be operating at a different voltage level or from a separate power supply. If the external supply is not stable before VSYS reaches its limit of trials (about 300 ms), the device automatically enters the power-down mode to minimize power consumption. If VSYS is tied to Ground, then the A7S retries the initialization loop until it succeeds.

The CPU performs an initialization sequence for external SDRAMs, even if none are connected to the A7. The CPU first de-activates the SDRAM controller, in case it was activated during a previous initialization attempt. After waiting 1.2 ms, the CPU activates the SDRAM controller.

The CPU then configures the external static memory interface to access various data widths. On the first attempt, the initialization procedure attempts to find a valid header in an 8-bit external device. If a valid header is not found using an 8-bit interface, the initialization procedure then tries a 16-bit interface. If the 16-bit interface is unsuccessful, then a 32-bit interface is tried.

To determine if external memory is correctly accessed, the primary initialization routine attempts to locate a valid header within the top 64 words of external static memory. A valid header is four words long and aligned to a four-word boundary. The search ends either when the CPU detects a valid header or when the CPU unsuccessfully tested all 16 potential header locations. Once a valid header is found, the CPU retests the header 16 consecutive times to guarantee that the first successful search was not caused by a transient state on the power supply.

If the header search was unsuccessful, the CPU then increments the width on the external static memory interface—first 8 bits wide, then 16, and finally 32 bits wide.

If a valid header is still not found after trying every data width, the CPU re-samples the VSYS pin and compares it against the value captured at the start of the process, in case there might have been a transient on the power supply. If the two VSYS samples do not match, the CPU restarts the entire process again. If the two VSYS samples do match, then the CPU increments a counter that tracks the number of attempts.

If no valid header is found after about 300 ms, and VSYS is sampled High, then the device automatically enters power-down mode to conserve power. If VSYS is Low, the device continues to search for initialization data.

If a valid header is found, then the CPU disables the alias to the internal ROM and branches to the secondary initialization code stored in external static memory.

Secondary Initialization Procedure

For most applications, FastChip generates the secondary initialization code automatically. No coding is required unless an application has some very special requirements.

The following describes the FastChip-created secondary initialization procedure. The ARM7TDMI CPU begins executing instructions located at the top of the external memory device, typically Flash.

After performing some initial set up, the CPU configures one of the A7S's DMA channels to copy configuration data from the external static memory to internal memory cells that control the Configurable System Logic matrix, within the A7.

After copying the CSL configuration image, the CPU uses a DMA channel to program the A7S's Configuration Register Unit.

The CPU then reclaims any unused system pins and user-PIO pins, as per the final application. Then, the CPU configures the A7S's clock control and external memory interface

timing values, and enables the CSL logic matrix. The application code then resets the *Reset Status Register*. As its final initialization act, the CPU sets the SYS_RESET_BIT in the *Reset Control Register*. This action resets the ARM7TDMI, which branches to location 0x0. Because of the default settings in the *Alias Enable Register*, location 0x0 points to external memory, where the user application program resides and the CPU starts executing the code there.

Post Initialization Options

After initialization, the application program typically performs a variety of system optimization setup steps. For example, code stored in external Flash can be copied to and executed from SDRAM. Executing from SDRAM offers higher system performance.

Likewise, the CPU can enable the cache, set up the memory protection mechanism, and enable additional pipelining.

Parallel Mode

Using parallel mode initialization, shown in Figure 53, the A7S's initialization data and the application program are stored in external static memory, typically Flash. The CPU can fetch and execute instructions directly from external memory or, for increased performance, copy the application code to SDRAM and execute from there.

The external memory interface signals are:

D[n:0] – the 8-, 16-, or 32-bit data bus from the external memory.

A[19:0] – the lower 20 bits of the address bus. If additional address bits are required, the upper address lines—A20 and beyond—can be enabled using the FastChip I/O Editor.

WE- - write enable signal provided to writeable memories such as Flash, EEPROM, and SRAM

OE- - enables the data output from the external memory during a read or fetch operation.

CE- - chip enable to the external memory.
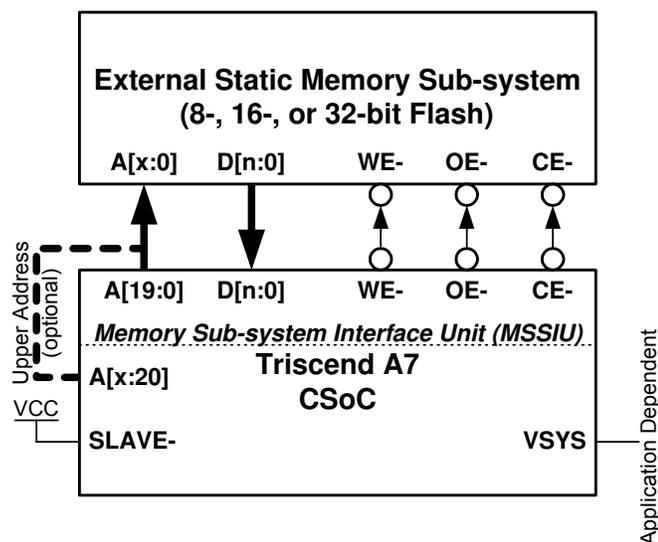
Figure 53. Parallel-mode initialization from external Flash.

Address lines A[19:0] sufficiently address up to 1M byte of external memory. With a larger memory subsystem, additional optional PIO pins serve as high-order address lines.

Once system initialization is complete, the CPU starts executing application code from location 0, which is aliased to external memory. Parallel mode requires that at least some application code reside in external static memory. At the beginning of application program execution, the contents of the external static memory can be copied into and executed from SDRAM.

During the parallel initialization phase, all PIO and MIU pins that do not participate in the parallel memory interface, are pulled High by weak pull-up resistors. These pins are properly initialized to their user-defined configurations at the end of initialization.

JTAG Initialization

The JTAG mode initializes the Triscend A7S using an external tester, personal computer (PC), or other intelligent host. The external host performs the secondary-initialization process, loading the A7S device. The initialization image downloaded directly into an A7S device does not include the secondary initialization code or headers.

The JTAG port is also used for debugging application code or to program external memory devices via the external static memory interface. When a PC acts as a host, the JTAG unit is controlled by a series of commands entered from the Triscend FastChip development system or third-party hardware debugger, such as the WindRiver visionPROBE II.

External memories are not required in JTAG configuration mode. The JTAG link can directly initialize the CSL matrix, download the necessary application code into the internal RAM, and direct the CPU to execute code from the internal system RAM. However, the entire application program must fit within the 16K bytes of internal RAM.

Figure 54 depicts a typical JTAG interface to a Triscend A7. In the figure, the CSoC device already connects to an external parallel memory. The JTAG port is compliant with IEEE standard 1149.1. The four JTAG pins are dedicated only to the JTAG function.

Figure 54. A JTAG interface to the Triscend A7.

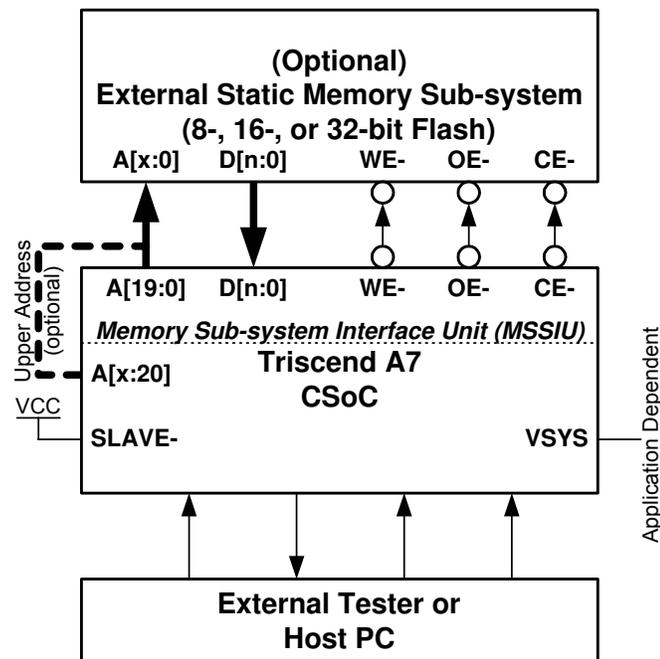The external memory interface signals are:

TCK – Test clock input. If unused should be tied high.

TMS – Test mode select input. If unused should be tied high.

TDI – Test data input. If unused should be tied high.

TDO – Test data output.

The JTAG unit also serves as a master on the CSI bus and can read and write every addressable entity in the system.

The JTAG port optionally controls the ARM7TDMI processor by setting breakpoint events that freeze the CPU. Once the processor is frozen, the JTAG unit can single-step the processor, examine the internal registers of the processor, and then resume the processor operation.

With the help of the CPU, the JTAG port can programs external Flash memory. In the Flash programming mode, the JTAG unit downloads program / erase / verify algorithms into the internal RAM. The new Flash programming data is also buffered in the internal RAM. The ARM7TDMI CPU executes the programming algorithms required to write new data into an external Flash memory device. The Flash is programmed with a parallel-mode initialization image. The processor interacts with the JTAG unit through flags and shared variables or via interrupts.

Size of Initialization Data

The size of the initialization data file depends on the specific A7S family device used and the initialization method. Most applications use parallel mode.

Table 45 provides details on the size of the secondary initialization image for an A7S family device by its various components. The size of the data is independent of the size of the logic function loaded into the CSL matrix. A blank CSL matrix requires the same amount of data as a fully utilized matrix. The size of the secondary initialization program may vary in future versions of the Triscend FastChip development system.

Table 45. Size of Secondary Initialization Image for A7S Family.

Initialization Data	Size (Bytes)
CSL initialization data structure	85,792
Secondary initialization program*	8,084
Header, system register start-up values, various data structures	160
TOTAL	**89,036**

* created using FastChip 2.2.0

VSYS Control and Affects on Initialization

A power-on reset or configuration reset causes the Triscend A7S to start the initialization process as shown in Figure 52. The VSYS input pin directs the system initialization state machine how to behave should the initialization process fail to find valid external initialization data.

If the VSYS pin is held High through the initialization phase and no valid configuration pattern is found, then the CSoC device automatically powers down to conserve power. Another Power-On Reset or System Reset event is required to restart the initialization process.

However, if VSYS is sampled Low during the initialization process, the A7S CSoC device attempts to re-start the initialization process following a failed initialization attempt. If

VSYS is tied to GND, the CSoC device attempts to initialize itself indefinitely until it finds valid data. This option is useful when the external memory operates from a separate supply or the system operates in noisy environments.

> **NOTE:**
>
> *VSYS must connect to a valid logic level. Do not leave VSYS unconnected or floating!*

In applications using a system supervisory chip, connect the "VCC good" output from the supervisory device to the A7S's VSYS input, ensuring that initialization only takes place when VCC is within the proper operating voltage range.

If using a non-Triscend supplied JTAG debugger, allow provisions on the A7S target so that VSYS can optionally be strapped Low. This will prevent the A7S device from powering down during debugging. Many non-Triscend JTAG debuggers are unable to wake the A7S device from power-down mode.

What If Initialization Fails?

If properly connected, an A7S CSoC device always correctly initializes. However, should initialization fail during development, use this debugging checklist to diagnose the problem.

Parallel Mode Initialization Checklist

☐ Are all VCC and VCCIO pins connected to their proper values? VCC must be connected to +2.5 volts. VCCIO may be connected either to +2.5 volts or +3.3 volts. If initializing from an external memory device, check that the correct voltage is applied.

☐ Are all GND and GNDIO pins connected?

☐ Is the VSYS pin connected to a valid logic level? Do not leave VSYS unconnected or floating.

☐ Is the SLAVE- pin connected to VCCIO?

☐ Is the RST- pin connected to a valid logic level? Do not leave RST- unconnected or floating. Is RST- a logic High? RST- must be High before initialization can begin.

☐ If using the crystal oscillator or an external clock as the source for bus clock, temporarily use the internal ring oscillator as the clock source. Create a new configuration image in FastChip Device Link (FDL) using the internal ring oscillator. Download the design to the A7. Does the device initialize from the internal ring oscillator? If so, then verify that the crystal oscillator or external clock is operating. Verify that the crystal oscillator is externally configured correctly (see Figure 47).

☐ Try downloading an application with a small amount of code that executes from internal SRAM. Does it function correctly? If so, this indicates that the external memory is either incorrectly connected or misconfigured.

☐ If executing from external Flash or if copying to and executing code from external SDRAM, verify that the connections are correct between the A7S and the external memory. Also, verify that the correct memory types and speed grades were specified when creating the configuration image in FastChip Device Link (FDL).

☐ Temporarily pull-down the RST- pin to ground using a 4.7kΩ resistor. Re-apply power to the board and wait a few seconds. Disconnect the pull-up resistor. Does the system initialize correctly? If so, this indicates a possible power-supply ramp-rate or supply stability problem in the system.

JTAG Initialization Checklist

The items listed under ***Parallel Mode Initialization Checklist*** also apply to JTAG initialization. Here are additional items unique to JTAG initialization.

☐ Check that all four JTAG pins are correctly connected to the intelligent host.

☐ Is there activity on the JTAG pins during the initialization process? If not, then the A7S device is not receiving initialization data.

☐ If using FastChip Device Link (FDL) or the command-line `csoc download` utility, then the download software should indicate whether it could successfully connect to the A7S device. If not, the software displays further debugging information.

☐ If using the WindRiver visionPROBE II JTAG download/debug cable, be sure that the cable is properly installed on your computer, including setting your parallel port in Enhanced Capabilities Port (ECP) mode.

Additional Initialization Debugging Support

Additional details and information is available on the Triscend SupportCenter web site at `http://support.triscend.com`.
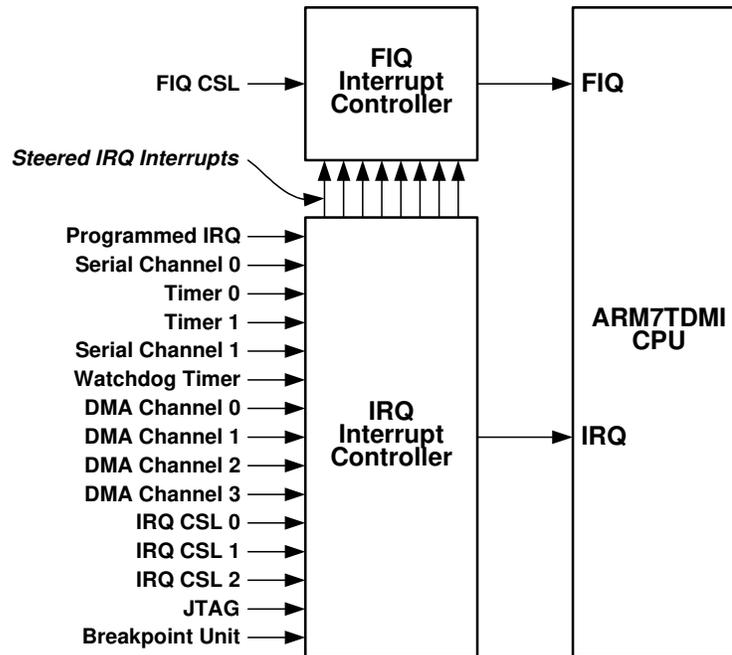
Figure 55. Interrupt Control Structure for the A7S Family.

Interrupts

The interrupt controller consolidates all the system interrupts into the two standard ARM interrupts, FIQ (Fast Interrupt Request) and IRQ (Interrupt Request), as shown in Figure 55. The interrupt controller provides a simple software interface to the interrupt system. All interrupt sources are active High and level sensitive. Priority and interrupt vectoring is handled by software.

Each interrupt is identically implemented. A status bit is available to monitor the source status regardless if the specific interrupt is enabled. An enable bit for each source selectively masks a source independently of the others.

Interrupt Control

The interrupt controller provides both a raw interrupt source status and a masked interrupt request status, which is individually controlled via an interrupt enable register. The interrupt control structure is shown in Figure 56. The control logic for the FIQ interrupt is similar, but there is no associated steering logic. FIQ interrupt requests always appear in the associated FIQ status registers.

The enable bit permits an active interrupt source to generate an interrupt request to the ARM7TDMI processor. Prior to masking, the interrupt source status indicates if a particular interrupt source is active, regardless if the interrupt is enabled or not.

The enable register allows an individual bit to be set or cleared by software without prior knowledge of the state of the other bits in the register. When writing to the FIQ or IRQ interrupt enable register, each data bit that is '1' in the register sets the corresponding bit in the enable register—all other bits are unaffected. The FIQ or IRQ clear interrupt register similarly clears individual bits of the enable register.

For debugging or performance reasons, any of the IRQ interrupts can be steered to the FIQ interrupts using the *IRQ Steering Register*. Once an IRQ interrupt is steered to the FIQ interrupt, its raw and masked status appear in the corresponding FIQ status registers and no longer in the IRQ status registers.

Figure 56. The Triscend A7S interrupt control logic selectively enables inputs to the FIQ and IRQ interrupt controller. IRQ interrupt requests can also be steered to the FIQ interrupt input.

Interrupt Sources

The FIQ interrupt typically originates from the CSL as a sideband signal. However, any IRQ interrupt source can be steered to the FIQ interrupt via the *IRQ Steering Register*.

The IRQ interrupt has 15 potential sources as listed below. Table 46 shows the source, source enable control, and source clear control for each IRQ interrupt.

- 4 interrupts from the DMA unit (one per channel)
- 2 timer interrupts
- 2 serial port interrupts
- Watchdog timer
- JTAG unit
- Breakpoint unit
- Software interrupt (ARM7TDMI instruction `SWI`)
- 3 user-defined interrupt inputs, IRQ2, IRQ1, and IRQ0, that originate in the CSL matrix as sideband signals

> **NOTE:**
>
> *A CSoC design practically supports an unlimited number of interrupts. Each user-defined interrupt input—IRQ2, IRQ1, IRQ0—can be expanded in the CSL matrix. FastChip provides a soft module for this function.*

ARM FIQ and IRQ Interrupts

The ARM7TDMI processor supports two levels of interrupts, FIQ and IRQ. Separate interrupt controllers are used between FIQ and IRQ.

The FIQ interrupt is used for handling fast, low-latency interrupts. The IRQ interrupt handles general interrupts. Typically, a single source for FIQ should be used at any particular time to ensure low latency interrupt handling. All IRQ interrupts are also available as FIQ interrupts. They can be steered individually by programming the *IRQ Steering Register*.

When an IRQ is steered, its status appears in the *__FIQ Status Register__* and not the IRQ register anymore. However, the associated interrupt is still enabled using the *__IRQ Enable Register__*.

A fast interrupt request (FIQ) has higher priority than an IRQ request because is it serviced first when multiple interrupts occur and servicing an FIQ interrupt disables an IRQ requests, thereby preventing any IRQ requests from being serviced until the FIQ interrupt handler has re-enabled them.

The FIQ interrupt structure is designed to service interrupt requests as quickly as possible. The FIQ vector address is the last in the vector table, which allows the FIQ interrupt handles to be located at the vector address, avoiding an additional jump. Furthermore, the FIQ exception has five additional banked registers, which reduces the number of registers that must be preserved by the handler.

Finally, for best performance, there should be just one source for the FIQ interrupt, reducing software overhead in the interrupt handler.

The worst-case interrupt latency for FIQ is 28 processor cycles or 420 ns at 60 MHz, assuming that the processor is executing from internal scratchpad RAM or cache.

Interrupt Controller Sideband Signals

There are four sideband input signals to the interrupt controller, as shown in . The fast interrupt input, FIQ, and the three external interrupt inputs IRQ2, IRQ1, and IRQ0 all originate from logic in the CSL matrix or directly from a PIO pin.

All the interrupt inputs are active High. If unused, connect these sideband signals Low.
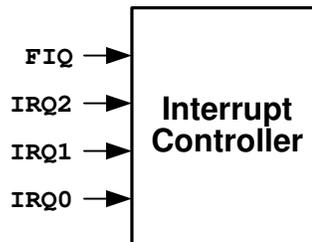

Figure 57. Interrupt Controller sideband inputs.

Control Registers Description

For the interrupt registers, the individual bit correspondence is described in Table 46. The FIQ interrupt is also available in the IRQ registers at the same bit position.

Table 46. Interrupt Register Bit Declarations.

Interrupt	Bit Location	Interrupt Source	Enable Interrupt	Clear Interrupt
Breakpoint IRQ_BREAKPOINT_BIT	15	Breakpoint unit	Controlled by Fast-Chip Device Link	Controlled by FastChip Device Link
JTAG IRQ_JTAG_BIT	14	JTAG unit	Controlled by Fast-Chip Device Link	Controlled by FastChip Device Link
IRQ 2 (from CSL) IRQ_CSL_USER_2_BIT	13	CSL application	Controlled via CSL application	Controlled via CSL application
IRQ 1 (from CSL) IRQ_CSL_USER_1_BIT	12	CSL application	Controlled via CSL application	Controlled via CSL application
IRQ 0 (from CSL) IRQ_CSL_USER_0_BIT	11	CSL application	Controlled via CSL application	Controlled via CSL application
DMA Channel 3 IRQ_DMA_3_BIT	10	Various sources recorded in DMA3_INT_REG	Set appropriate bits in DMA3_INT_ENABLE_REG	Write '1' to appropriate bit in DMA3_INT_CLEAR_REG
DMA Channel 2 IRQ_DMA_2_BIT	9	Various sources recorded in DMA2_INT_REG	Set appropriate bits in DMA2_INT_ENABLE_REG	Write '1' to appropriate bit in DMA2_INT_CLEAR_REG
DMA Channel 1 IRQ_DMA_1_BIT	8	Various sources recorded in DMA1_INT_REG	Set appropriate bits in DMA1_INT_ENABLE_REG	Write '1' to appropriate bit in DMA1_INT_CLEAR_REG
DMA Channel 0 IRQ_DMA_0_BIT	7	Various sources recorded in DMA0_INT_REG	Set appropriate bits in DMA0_INT_ENABLE_REG	Write '1' to appropriate bit in DMA0_INT_CLEAR_REG
Watchdog Timer IRQ_WATCHDOG_BIT	6	Watchdog timer decrements to 0	Set WD_ENABLE_BIT in WATCHDOG_CONTROL_REG	Write '1' to WATCHDOG_CLEAR_REG
Serial Channel 1 IRQ_SERIAL_1_BIT	5	Various sources recorded in UART1_INT_ID_REG	Set appropriate bits in UART1_INT_ENABLE_REG	Various actions, as defined in Table 50
Timer 1 IRQ_TIMER_1_BIT	4	Timer 1 decrements to 0	Set TIM_ENABLE_BIT in TIMER1_CONTROL_REG	Write '1' to TIMER1_CLEAR_REG
Timer 0 IRQ_TIMER_0_BIT	3	Timer 0 decrements to 0	Set TIM_ENABLE_BIT in TIMER0_CONTROL_REG	Write '1' to TIMER0_CLEAR_REG
Serial Channel 0 IRQ_SERIAL_0_BIT	2	Various sources recorded in UART0_INT_ID_REG	Set appropriate bits in UART0_INT_ENABLE_REG	Various actions, as defined in Table 50
Programmed IRQ IRQ_SOFTWARE_BIT	1	Write '1' to INT_IRQ_SOFT_REG	N/A. Always enabled.	Write '0' to INT_IRQ_SOFT_REG
FIQ (from CSL) FIQ_BIT	0	CSL application	Controlled via CSL application	Controlled via CSL application

> **NOTE:** ✔
> *Every CPU write to the CSI bus is buffered. When clearing interrupts and leaving an interrupt service routing (ISR), flush the write buffer by reading from any CSI location that it unaffected by a read.*
>
> *The Triscend driver library already handles this situation.*

Interrupt Controller Memory Map

Address		Register Name	Access
Base	**Offset**		
INT_BASE	+ 0x00	IRQ Status	R
	+ 0x04	IRQ Raw Status	R
	+ 0x08	IRQ Enable	R/W
	+ 0x0C	IRQ Enable Clear	W
	+ 0x10	IRQ Soft Request	W
	+ 0x100	FIQ Status	R
	+ 0x104	FIQ Raw Status	R
	+ 0x108	FIQ Enable	R/W
	+ 0x10C	FIQ Enable Clear	W
	+ 0x110	IRQ Steering	W

IRQ Status Register (INT_IRQ_STATUS_REG)

This register provides the status of the interrupt sources *after* masking by the interrupt enable register. It is a read-only register.

Bit	Description/Function
31:16	Reserved
15:1	**General Interrupt Requests Status:** 0: No request. 1: An active and enabled IRQ request from the associated interrupt source. (See Table 46)
0	**Fast Interrupt Request Status:** 0: No active FIQ interrupt request. 1: FIQ interrupt request.

System reset value: 0

IRQ Raw Status Register (INT_IRQ_RAW_STATUS_REG)

This register provides the status of the interrupt sources before masking, regardless of the interrupt enable register. It is a read-only register.

Bit	Description/Function
31:16	Reserved
15:1	**General Interrupts Source Status:** 0: No request. 1: An active IRQ request from the associated interrupt source. (See Table 46)
0	**Fast Interrupt Source Status:** 0: No active FIQ interrupt request. 1: FIQ interrupt request.

System reset value: 0

IRQ Enable Register (INT_IRQ_ENABLE_REG)

This register selectively allows individual interrupt sources to generate an IRQ interrupt to the ARM7TDMI processor.

Bit	Description/Function
31:16	Reserved
15:1	**General Interrupts Enable:** 0: No effect. 1: Enable associated interrupt request. When source generates an interrupt, the request is recorded in either the IRQ or FIQ status register, depending on the IRQ steering register. (See Table 46)
0	Reserved

System reset value: 0

IRQ Enable Clear Register (INT_IRQ_ENABLE_CLEAR_REG)

This register provides the control to clear the interrupt enables. This register is write-only.

Bit	Description/Function
31:16	Reserved
15:1	**Clear General Interrupts Enable:** 0: No effect. 1: Disable associated interrupt request. When source generates an interrupt, the request is not recorded. However, the raw, unmasked status appears in either the IRQ or FIQ raw status register, depending on the IRQ steering register. (See Table 46)
0	Reserved

Software Interrupt Request Register (INT_IRQ_SOFT_REG)

This register provides a software mechanism to generate an interrupt. This register is write-only.

Bit	Description/Function
31:2	Reserved
1	**Software Interrupt Request:** 0: Clear software interrupt. 1: Generate a software interrupt request.
0	Reserved

FIQ Status Register (INT_FIQ_STATUS_REG)

This register provides the status of the fast interrupt sources (FIQ or "steered" IRQ) after masking via the appropriate interrupt enable register. This register is read-only.

Bit	Description/Function
31:16	Reserved
15:1	**Steered IRQ Status:** 0: No request. 1: An active and enabled IRQ request from the associated interrupt source, if the associated bit is set in the IRQ steering register. (See Table 46)
0	**Fast Interrupt Request Status (FiqStat):** 0: No active FIQ interrupt request. 1: FIQ interrupt request.

System reset value: 0

FIQ Raw Status Register (INT_FIQ_RAW_STATUS_REG)

This register provides the status of the interrupt sources (FIQ or "steered" IRQ) before masking via the appropriate interrupt enable register. This register is read-only.

Bit	Description/Function
31:16	Reserved
15:1	**Steered IRQ Source Status:** 0: No request. 1: An active IRQ request from the associated interrupt source, if the associated bit is set in the IRQ steering register. (See Table 46)
0	**Fast Interrupt Source Status:** 0: No active FIQ interrupt request. 1: FIQ interrupt request.

System reset value: 0

FIQ Enable Register (INT_FIQ_ENABLE_REG)

This register provides the status and partial control for the fast interrupt enable.

Bit	Description/Function
31:1	Reserved
0	**Fast Interrupt Enable:** 0: No effect. 1: Enable the FIQ interrupt request. When the source generates an interrupt, either via the FIQ interrupt input or a steered IRQ request, the request is recorded in the FIQ status register.

System reset value: 0

FIQ Enable Clear Register (INT_FIQ_ENABLE_CLEAR_REG)

This register provides the control to clear the fast interrupt enable. This register is write-only.

Bit	Description/Function
31:1	Reserved
0	**Clear Fast Interrupt Enable:** 0: No effect. 1: Disable the FIQ interrupt request. When the source generates an interrupt, the request is not recorded. However, the raw, unmasked status appears in FIQ raw status register.

IRQ Steering Register (INT_IRQ_STEER_REG)

This register enables any individual IRQ to be steered to the FIQ.

Bit	Description/Function
31:16	Reserved
15:1	**IRQ Steer Enable:** 0: The associated IRQ interrupt request appears in the IRQ status registers (See Table 46) 1: The associated IRQ interrupt request is steered to the FIQ interrupt and appears in the FIQ status registers (See Table 46)
0	Reserved

System reset value: 0

System Control Registers

Remap and Pause Registers

The remap and pause control registers provide general system control. Some of these registers are required for minimum ARM7TDMI real-time operating system (RTOS) support.

Address		Register Name	Access
Base	**Offset**		
RMAP_BASE	+ 0x00	Pause	W
	+ 0x10	Identification	R
	+ 0x14	Revision	R
	+ 0x20	Clear Reset Map	W
	+ 0x38	Pin Status	R
	+ 0x3C	Pin Status Clear	W
	+ 0x40	Alias Enable	R/W
	+ 0x44	Scratchpad Configuration	R/W
	+ 0x48	Scratchpad Base Address	R/W
	+ 0x4C	Access Protection	R/W

Pause Register (REMAP_PAUSE_REG)

The *Pause Register* places only the ARM7TDMI in a low-power state. Use the **Power Down Control Register** to place other portions of the system into a lower-power state.

Writing any value to the Pause register causes the ARM processor, and only the processor, to enter the low-power state. The processor remains in this mode until it receives an IRQ interrupt or a system reset.

Bit	Description/Function
31:0	Reserved

Identification Register (REMAP_IDENTIFICATION_REG)

This register identifies the specific A7S CSoC family member. This register is read-only.

Bit	Description/Function
31:0	**Device Identification Type:** A 32-bit constant identifying an A7S family member.

Not reset. Value = 0x1803_D2FF.

Revision Register (REMAP_REVISION_REG)

This register identifies the mask revision for the A7S device. This register is read-only.

Bit	Description/Function
31:0	**Mask Revision Number:** A 32-bit constant identifying the mask revision for the device type indicated in the REMAP_IDENTIFICATION_REG register. <table><tr><td>Device</td><td>Revision Code</td><td>Revision Status</td></tr><tr><td>A7S04</td><td>0x0_01_13</td><td>Production</td></tr><tr><td>A7S20</td><td>0x0_01_13</td><td>Production</td></tr><tr><td>A7S20</td><td>0x0_01_11</td><td>Prototype, no cache support</td></tr><tr><td>A7S20</td><td>0x0_00_00</td><td>Obsolete</td></tr></table>

Not reset. Undefined.

Clear Reset Memory Map Register (REMAP_CLEAR_RESET_MAP_REG)

This register provides a means to change the system memory map from the user's current memory map to the one used during normal operation. Writing this register causes the memory map switch. It effectively clears the Flash alias bit in the **_Alias Enable Register_**.

This register is provided to maintain compatibility with other software created for the ARM7TDMI processor. The same functionality is provided by the A7S's **_Alias Enable Register_**.

Bit	Description/Function
31:0	Reserved

Pin Status Register (REMAP_PIN_STATUS_REG)

This register enables software to monitor the status of some static pins of the device. This register is read-only.

Bit	Description/Function
31:5	Reserved
4	**VSYS Bad Flag (VSYS_BAD_BIT):** A '1' indicates that the voltage level on the VSYS pin dropped to a "bad" level permanently or temporarily.
3	**VSYS Good Flag (VSYS_GOOD_BIT):** A '1' indicates that the voltage level on the VSYS pin reached a "good" level permanently or temporarily.
2	**SLAVE- Pin Status (SLAVEN_BIT)** Should always be '1'. SLAVE- pin connects to VCCIO.
1	**RST- Pin Status (RSTN_BIT)**
0	**VSYS Pin Status (VSYS_BIT)**

Configuration reset value: bits [4:3] = '11'
Bits [2:0] are not reset. Undefined.

Pin Status Clear Register (REMAP_PIN_STATUS_CLEAR_REG)

This register is write only. Software clears bits of the **Pin Status Register** by writing '1' to the corresponding *Pin Status Clear Register* location. Writing 0 has no effect.

Bit	Description/Function
31:5	Reserved
4	**Clear VSYS Bad Flag (VSYS_BAD_CLR_BIT):** 0: No effect 1: Clear bit.
3	**Clear VSYS Good Flag (VSYS_GOOD_CLR_BIT):** 0: No effect 1: Clear bit.
2:0	Reserved

Alias Enable Register (REMAP_ALIAS_ENABLE_REG)

This register defines which alias is enabled at the bottom of the memory starting at address 0. If more than one alias is enabled, they are overlaid over each other with the following priority (from highest to the lowest priority). See Figure 28.

1. Internal boot ROM

2. External Flash or SRAM static memories

3. Internal SRAM

4. External SDRAM

Bit	Description/Function
31:4	Reserved
3	**SDRAM Alias Enable (SDRAM_AEN_BIT):** 0: Disable alias 1: Alias external SDRAM to address 0
2	**Internal SRAM Alias Enable (SRAM_AEN_BIT):** 0: Disable alias 1: Alias internal SRAM to address 0
1	**Flash Alias Enable (FLASH_AEN_BIT):** 0: Disable alias 1: Alias external Flash to address 0
0	**Internal Boot ROM Alias Enable (ROM_AEN_BIT):** This alias is automatically disabled during the device initialization process. 0: Disable alias 1: Alias external Flash to address 0

Power-on or configuration reset value: 0xF (All aliases enabled).

System reset value: 0xE (All aliases enabled, except for primary initialization ROM).

After secondary initialization complete: 0xC (SDRAM and internal SRAM aliased to address 0).

Access Protection Register (REMAP_ACC_PROTECT_REG)

Bit	Description/Function
31:1	Reserved
0	**DMA Register Write Disable (DMA_DIS_BIT):** When set, protects the A7S control registers against accidental DMA write accesses. 0: DMA write accesses allowed to addresses between 0xD101_0000 and 0xD101_FFFF. 1: The memory region between 0xD101_000 and 0xD101_FFFF is protected against DMA write transactions.

System reset value: 1

Dedicated 16-bit Timers

The A7S system has two dedicated 16-bit timers that follow the exact model specified by the ARM "Reference Peripherals" document. If required by an application, timers with different features can be added in using the CSL matrix.

Each timer is 16 bits wide with an 8-stage programmable pre-scaler and operates from the system clock. The system clock is used directly, or divided by 16 or 256 using the pre-scaler. Two operating modes are available, free-running and periodic timer. In periodic timer mode, the counter generates a raw interrupt at a constant time interval. In free-running mode, the timer overflows after reaching zero and continues to countdown from the maximum value.

The pre-scale values, the timer mode, and enable control are configured using the ***Timer Control Register***. Writing '1' to the ***Timer Clear Register*** clears the timer interrupt flag. The current timer value is available in the ***Timer Value Register***.

Free-Running Mode

Figure 58 shows a block diagram of the free-running mode. Writing to the ***Timer Load Register*** loads the timer. Once enabled, the timer decrements until it reaches 0. Upon reaching zero, the timer generates a raw interrupt and the counter wraps around to its maximum value, 0xFFFF, and continues to decrement.

Periodic Timer Mode

Figure 59 shows a block diagram of the periodic mode. Writing to the ***Timer Load Register*** loads the timer. Once enabled, the timer decrements until it reaches 0. Upon reaching zero, the timer generates a raw interrupt and the counter reloads itself from the ***Timer Load Register*** and continues to decrement.
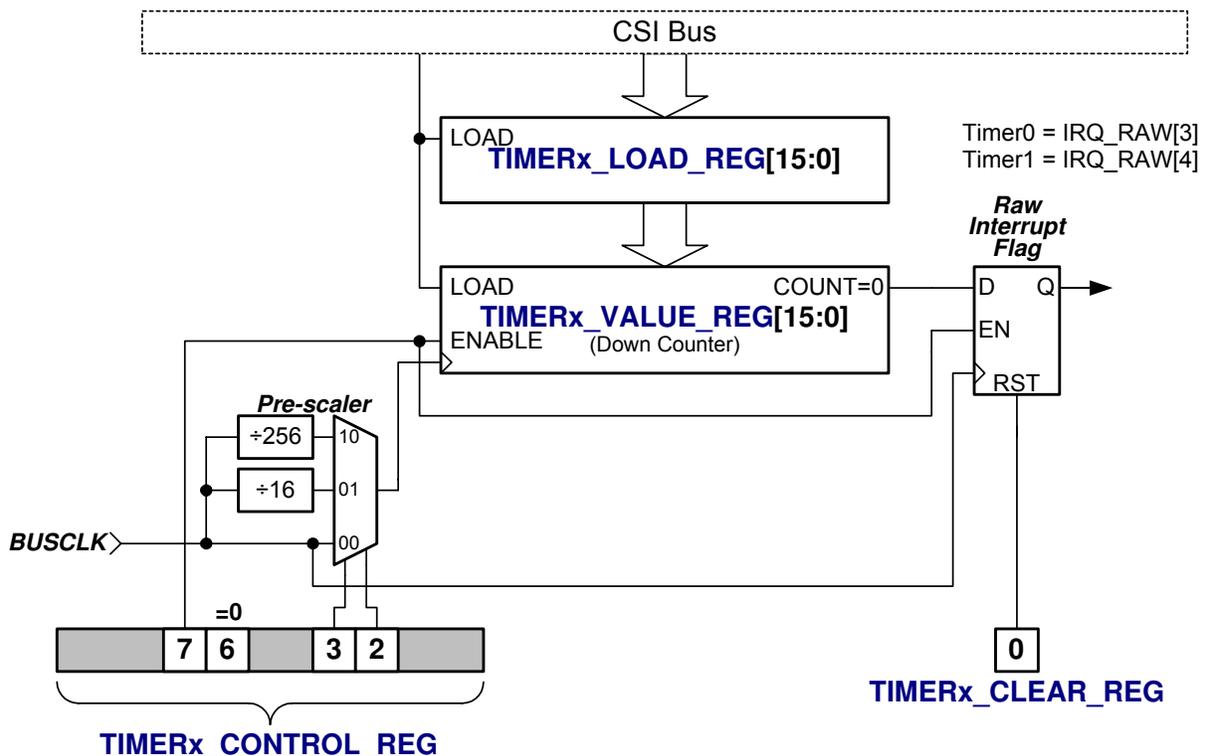

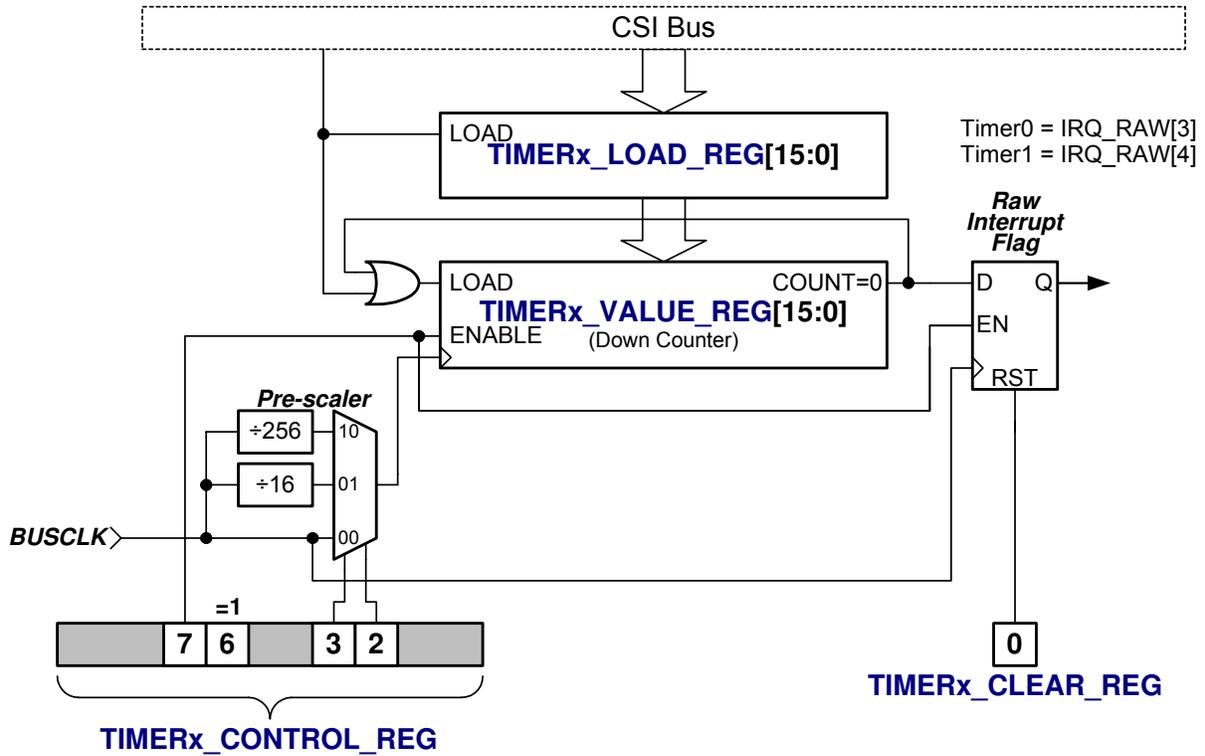**Figure 58. Timer 0 or Timer 1 in Free-Running Mode.**

Figure 59. Timer 0 or Timer 1 in Periodic Mode.

Clock Source and Pre-scaling

The source for the timer clock is the system clock. The clock is pre-scaled by the values 1, 16 or 256. The pre-scaler unit generates an enable signal to decrement the counter.

Control Registers Description

Address		Register Name	Access
Base	**Offset**		
TIMER_BASE	+ 0x00	Timer 0 Load	R/W
	+ 0x04	Timer 0 Value	R
	+ 0x08	Timer 0 Control	R/W
	+ 0x0C	Timer 0 Clear	W
	+ 0x20	Timer 1 Load	R/W
	+ 0x24	Timer 1 Value	R
	+ 0x28	Timer 1 Control	R/W
	+ 0x2C	Timer 1 Clear	W

Timer Load Register (TIMERx_LOAD_REG)

Bit	Description/Function
31:16	Reserved
15:0	**Load Timer Value (LOAD_FIELD [15:0]):** Writing to this register loads the timer. Defines the initial value of the timer. Also used as the reload value in period mode.

Not reset. Undefined.

Timer Value Register (TIMERx_VALUE_REG)

This register is read-only.

Bit	Description/Function
31:16	Reserved
15:0	**Current Timer Value (VALUE_FIELD [15:0])**

System reset value: 0

Timer Control Register (TIMERx_CONTROL_REG)

This register configures the timer's operation.

Bit	Description/Function
31:8	Reserved
7	**Timer Enable (TIM_ENABLE_BIT):** 0: Disable 1: Enable
6	**Timer Mode (TIM_MODE_BIT):** 0: Free-running 1: Periodic
5:4	Reserved
3:2	**Pre-scale Value (PRESCALE_FIELD[1:0]):** 00: Bus Clock 01: Bus Clock ÷ 16 10: Bus Clock ÷ 256 11: Reserved
1:0	Reserved

System reset value: 0

Timer Clear Register (TIMERx_CLEAR_REG)

This register is used to clear the timer interrupt. This register is write-only.

Bit	Description/Function
31:1	Reserved
0	**Clear Timer Interrupt (TIM_INT_CLEAR_BIT):** 1: Clear timer interrupt 0: No effect

Watchdog Timer

The watchdog timer is a free-running 32-bit counter programmed to serve as an event timer. The watchdog timer operates from the internal system clock. The time-out interval is programmable. When a time-out occurs, the watchdog timer generates a raw interrupt and optionally generates a system reset. Both operations are independent of one another.

The watchdog timer is primarily used as a system monitor but also serves as a simple timer. As a system monitor, the watchdog timer generates a interrupt or optionally generates a system reset. As a simple timer, the watchdog timer again generates an interrupt, but with system reset option disabled. Similarly, the CPU can poll the raw interrupt flag or directly read the value of the counter. The watchdog timer is initialized upon a system reset or via a control register bit.

The time-out interval is programmable using the ***Watchdog Timeout Value Register***. The watchdog counter is loaded automatically when software writes to the time-out interval. Once enabled, the watchdog counter decrements indefinitely or until it is disabled. Upon reaching zero, the counter reloads the time-out value and continues decrementing.

When the counter reaches zero, a time-out occurs which sets the interrupt flag. If the watchdog interrupt source is not enabled in the ***IRQ Enable Register*** (bit 6), then it can be polled as a status line in the ***IRQ Raw Status Register*** (bit 6). If the watchdog-reset bit is enabled, then the watchdog timer generates a system reset 4,096 clock cycles after the time-out occurs. A system reset also resets the watchdog timer, clearing and disabling it. The 4096-cycle delay provides ample time for the software to reset the watchdog timer before a watchdog reset occurs, avoiding an unwanted reset condition. If the system ever entered an unexpected state, then the CPU would fail to reset the watchdog timer when required. The watchdog timer would cause a system reset, forcing the system back into a known state.
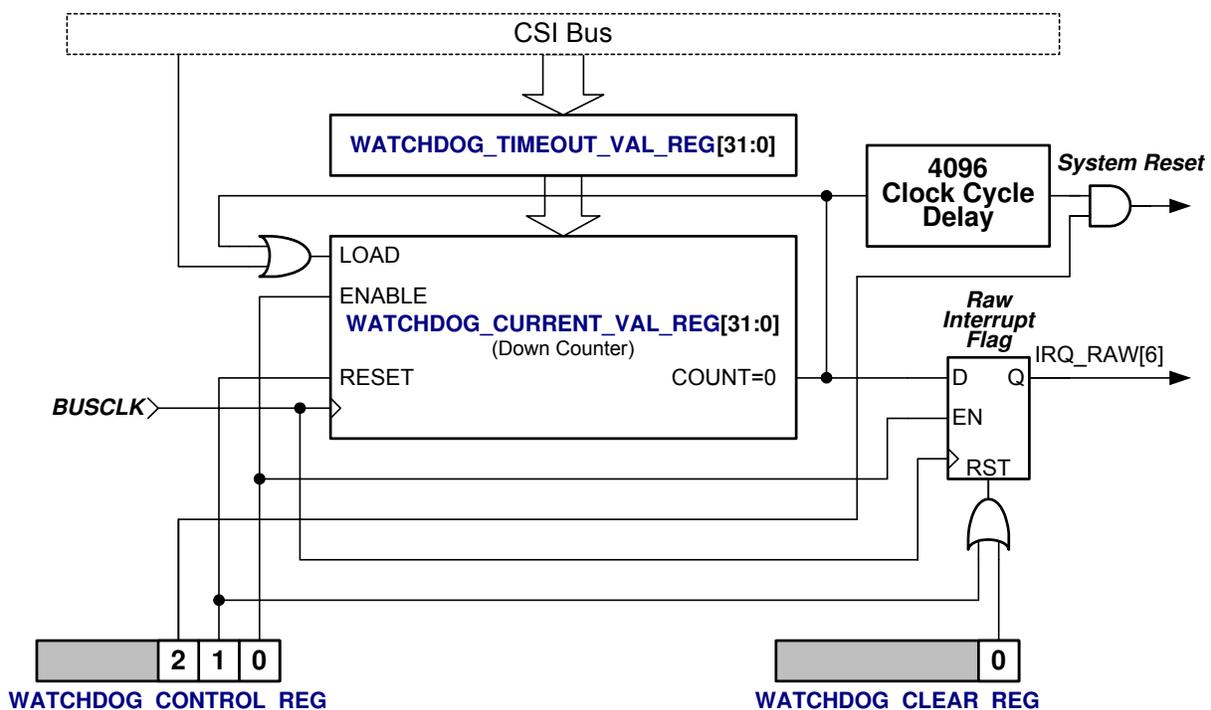

Figure 60. The A7S's 32-bit Watchdog Timer.

The application program resets the watchdog timer by setting the self-clearing WD_RESET_BIT in the ***Watchdog Control Register***.

Watchdog Registers Memory Map

Address		Register Name	Access
Base	Offset		
WD_BASE	+ 0x00	Watchdog Control	R/W
	+ 0x04	Watchdog Timeout Value	R/W
	+ 0x08	Watchdog Current Value	R
	+ 0x0C	Watchdog Clear	W

Watchdog Control Register (WATCHDOG_CONTROL_REG)

Bit	Description/Function
31:1	Reserved
2	**Enable Watchdog Reset (EN_WD_RST_BIT):** 0: Disable watchdog timer reset 1: Enable watchdog timer reset. A system reset will occur 4096 clock cycles after the Watchdog timer time-out, unless the application program resets the watchdog timer within the 4096 clock cycle window.
1	**Watchdog Reset (WD_RESET_BIT):** 0: No effect 1: Reset watchdog timer, including its interrupt flag and reset logic. This bit is self-clearing.
0	**Watchdog Enable (WD_ENABLE_BIT):** 0: Disable watchdog timer 1: Enable watchdog timer

System reset value: 0

Watchdog Time-out Value Register (WATCHDOG_TIMEOUT_VAL_REG)

When this register is updated, the counter is also automatically loaded with this register value.

Bit	Description/Function
31:0	**Watchdog Time-out Value**

Not reset. Undefined.

Watchdog Current Value Register (WATCHDOG_CURRENT_VAL_REG)

This register is read only.

Bit	Description/Function
31:0	**Watchdog Current Value**

Not reset. Undefined.

Watchdog Clear Register (WATCHDOG_CLEAR_REG)

This register is used to clear the watchdog interrupt. It is a write-only register.

Bit	Description/Function
31:1	Reserved
0	**Clear Watchdog Timer Interrupt (WD_INT_CLR_BIT):** 0: No effect. 1: Clear timer interrupt

Re-

ceiver/Transmitters communi-

Serial Ports (UARTs)

Each A7S CSoC device contains two dedicated Universal Asynchronous Receiver/Transmitters (UARTs). Each UART, shown in Figure 62, is "register and feature compatible" with a 16C450/550-style device and is a full duplex asynchronous communication module that supports the following features.

- 5-, 6,- 7-, or 8 bit data transmission
- Even, odd, or no parity bit generation and checking
- Start and stop bit generation and checking
- Line break detection and generation
- Receiver overrun and framing error checking
- Communications rates exceeding 1M baud
- Internal programmable baud-rate generator
- FIFO (16C550-style) or non-FIFO (16C450-style) operating modes
  - Transmitter is buffered with 16-byte FIFO
  - Receiver is buffered with 16-byte FIFO plus three error bits per data byte
- Exception handling using interrupt/polled modes
- Internal diagnostic capabilities with loop-back
- Modem handshake capability

In addition to the regular set of 16C550 registers, a few additional registers provide extra flexibility to the serial ports.

Baud Rate Generation

The baud-rate generator provides the clock for the transmitter and the receiver. The generated clock is 16 times the baud rate of the UART. The baud rate-generator output from each UART is also available in the CSL through the BDCLKx sideband signals.

> **NOTE:**
>
> *The BDCLKx sideband signal is a pulse that repeats at the baud-rate frequency and is High for one Bus Clock cycle, Low for the remainder of the period. It is **not** a 50% duty cycle output.*

Dividing Bus Clock, the system clock, generates the baud-rate clock. The clock division is performed in two steps. First, the system clock is divided down using an 8-bit pre-scaler, defined in the *UART Control Register*. The available pre-scaler ranges between 1 to 256. Then pre-scaler output is further divided by a fully programmable 16-bit value.

Together, the 8-bit pre-scaler and the UART's 16-bit divider provide a 24-bit divider. Any effective baud rate can be synthesized from a 60 MHz clock using a 24-bit divisor value, as shown in Equation 2. The output of the BDCLKx baud-rate clock sideband signals is sixteen times the baud-rate frequency, as shown in Equation 3.
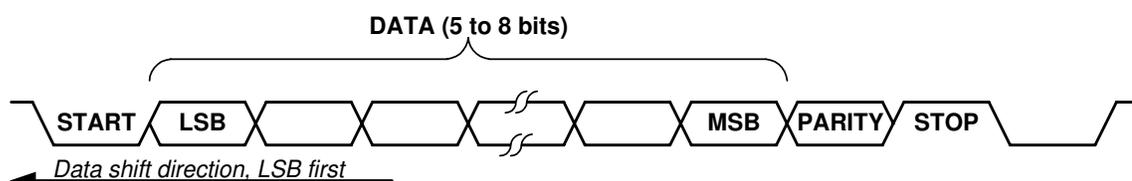
DATA (5 to 8 bits)

Figure 61. UART serial data format.

$$F_{BaudRate} = \frac{UART\_ENABLE \bullet F_{BusClock}}{16 \bullet (UART\_PRESCALE + 1) \bullet (UART\_DIVISOR)} \tag{2}$$

where:

UART_ENABLE	=	the PRESCALE_EN_BIT bit in the UART control register.
$F_{BusClock}$	=	the frequency of Bus Clock.
UART_PRESCALE	=	The 8-bit UART_PRESCALE_FIELD[7:0] field in the UART control register. Values 0 and 1 are illegal.
UART_DIVISOR	=	the 16-bit divider value stored in the UARTx_DIVISOR_MSB_REG and UARTx_DIVISOR_LSB_REG registers.

$$F_{Baud\,Rate\,Clock} = F_{BaudRate} \bullet 16 \tag{3}$$

NOTE:

The 16550/16450-style UARTs embedded within an A7S CSoC device are generally binary compatible with the original discrete UART components. However, the baud-rate equation is different.

Transmitting Data

Data is transmitted by writing a data byte into the ***Transmit Holding Register*** or the transmit FIFO in FIFO mode. The data byte is either written directly through a CSI memory access or by a memory-to-device DMA transaction. In DMA mode, the UART generates the proper requests to prompt the system for more data. In normal, non-DMA mode, data requests are prompted through interrupts.

Once transmitted, the data byte is then shifted out through the SOUTx CSL sideband signal. The LSB is shifted out first and the MSB last, as shown in Figure 61. The proper framing bits—start, stop, and parity—are automatically added by the transmitter.

The transmitter clock is the output of the baud rate generator. Data bits are presented on the serial output data line once every 16 transmitter clock cycles.

Receiving Data

The UART receives bytes from the SINx CSL sideband signal. Once a byte is received, the UART either generates an interrupt or generates a DMA device-to-memory request to an associated DMA channel. The byte is stored in the ***Receiver Buffer Register*** or in the *Receiver Buffer* in FIFO mode. When using the DMA to transfer received data; the FIFO mode is unnecessary because a DMA request is generated immediately when a word is received. Realistically, the DMA response latency is fast enough to eliminate buffering in the UART.

The receiver clock is the output of the baud rate generator. The receiver synchronizes the shift clock using the falling edge of the start bit on the SINx sideband input. It then receives a complete byte according to the parameters set in the ***Line Control Register***.
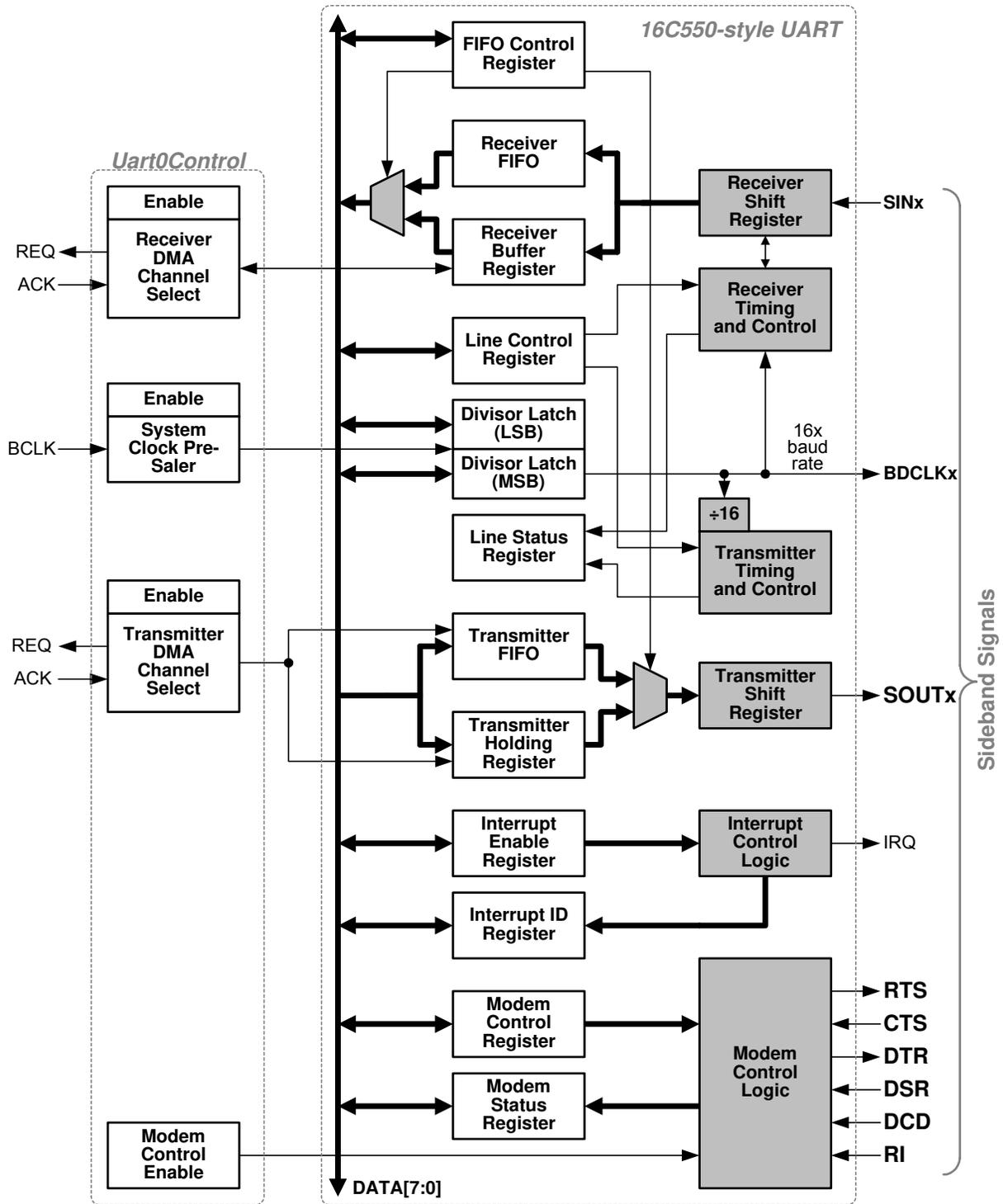
Figure 62. UART block diagram showing one of the 16C550-style serial ports and the modem control logic, assigned to one of the serial ports.

UART Sideband Signals

The serial data input and output signals, as well as the baud-rate clock, are available as sideband signals into the CSL matrix, as shown in Figure 63. These signals can connect to other logic within the CSL matrix or to PIO pins. There are no PIO pins dedicated for these signals, any PIO pins will do.
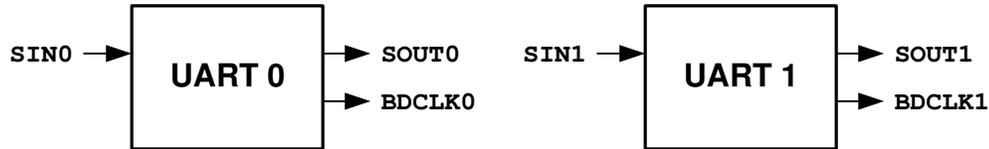

Figure 63. UART sideband signals.

DMA Feature

Each UART optionally connects to any of the four CSI DMA channels. The transmitter and the receiver of each UART can be paired independently to a DMA channel using the *UART Control Register*.

When associated with a DMA channel, the receiver or transmitter issues DMA requests and is treated as an I/O device. For example, if the receiver of a UART is paired to a DMA channel, then the receiver issues a device-to-memory DMA request whenever data is available. Application software previously configures the DMA channel to perform de-vice-to-memory transfers. The DMA performs a DMA Acknowledge Read cycle to gather the data from the receiver then stores the data away in memory at the current DMA desti-nation address.

When a UART transmitter is associated with a DMA channel, it similarly issues memory-to-device requests.

Using DMA transfers frees the CPU from polling for data or handling interrupts.

Modem Feature and Sideband Control Signals

Signals required to interface a UART with a modem are available through the CSL side-band signals, as shown in Figure 64. Each UART has a full set of modem handshake signals. However, only one set of signals can be used at a time. Only one set at a time is statically connected to the CSL sideband signals by setting the MODEM_EN_BIT.
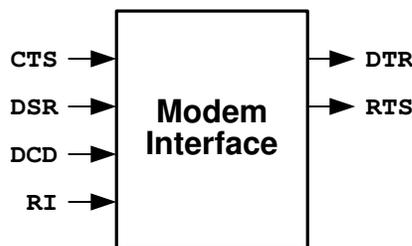

Figure 64. Modem control sideband signals.

Table 47 describes the modem control signals, including their signal direction and active polarity. On the A7S CSoC device, all the modem control signals are active High, which differs from a stand-alone 16C450/550 UART device. On the original 16C450/550, the RTS-, DTR-, DCD-, CTS-, and RI- signals are all active Low. On the A7S device, these are active High. However, these signals can be inverted using CSL logic to maintain com-plete compatibility with the original 16C450/550.

Table 47. Modem Control Sideband Signals.

Signal	Description	Direction	Active Polarity
DTR	Modem Data Terminal Ready (DTR) signal	UART → CSL	High
RTS	Modem Request to Send (RTS) signal	UART → CSL	High
CTS	Modem Clear to Send (CTS) signal	CSL → UART	High
DSR	Modem Data Set Ready (DSR) signal	CSL → UART	High
DCD	Data Carrier Detect (DCD)	CSL → UART	High
RI	Modem Ring Indicator (RI)	CSL → UART	High

Serial Ports Register Memory Map

Address Base	Offset	Register Name	Access
UART_BASE	+ 0x00	UART 0 Control	R/W
	+ 0x20	UART 0 RxTx or UART 0 Divisor LSB	R/W
	+ 0x24	UART 0 Interrupt Enable or UART 0 Divisor MSB	R/W
	+ 0x28	UART 0 Interrupt Identification or UART 0 FIFO Control	R or W
	+ 0x2C	UART 0 Line Control	R/W
	+ 0x30	UART 0 Modem Control	R/W
	+ 0x34	UART 0 Line Status	R
	+ 0x38	UART 0 Modem Status	R
	+ 0x3C	UART 0 Scratchpad	R/W
	+ 0x40	UART 1 Control	R/W
	+ 0x60	UART 1 RxTx or UART 1 Divisor LSB	R/W
	+ 0x64	UART 1 Interrupt Enable or UART 1 Divisor MSB	R/W
	+ 0x68	UART 1 Interrupt Identification Or UART 1 FIFO Control	R or W
	+ 0x6C	UART 1 Line Control	R/W
	+ 0x70	UART 1 Modem Control	R/W
	+ 0x74	UART 1 Line Status	R
	+ 0x78	UART 1 Modem Status	R
	+ 0x7C	UART 1 Scratchpad	R/W

To preserve full compatibility with the 16C550-style UART, some registers share the same address. Registers sharing the same address are differentiated by the DLAB_BIT, bit 7 in the UART's *Line Control Register*. Set DLAB_BIT=1 to access the baud-rate divisor values.

Each serial channel contains a set of the following registers.

UART Control Register (UARTx_CONTROL_REG)

Bit	Description/Function
31:23	Reserved
22:21	**Receive DMA Channel Select (RX_DMA_SEL_FIELD [1:0]):** These bits specify the DMA channel that services the UART's receive transfers. (see Table 48)
20	**Receive DMA Channel Enable (RX_DMA_EN_BIT)** Setting this bit enables the UART receive channel to communicate with the select DMA channel via a device-to-memory device transaction. 0: Disabled 1: Enabled
19	Reserved
18:17	**Transmit DMA Channel Select (TX_DMA_SEL_FIELD [1:0]):** These bits specify the DMA channel that services the UART's transmit transfers. (see Table 48)
16	**Transmit DMA Channel Enable (TX_DMA_EN_BIT):** Setting this bit enables the UART transmit channel to communicate with the selected DMA channel via a memory-to-device transaction. 0: Disabled 1: Enabled
15:10	Reserved
9	**Modem Control Enable (MODEM_EN_BIT):** Setting this bit enables the serial channel's modem control lines, which are available to the CSL matrix via sideband signals. 0: Disabled 1: Enabled *NOTE:* Because both channels share the same sideband lines for the modem handshake, only one serial channel can have its modem enabled.
8	**Pre-scaler Enable (PRESCALE_EN_BIT):** When disabled, the output of the pre-scaler is forced to 0. 0: Disabled 1: Enabled
7:0	**System Clock Pre-Scale Value (UART_PRESCALE_FIELD [7:0]):** Divide the system clock frequency from 1 to 256.

System reset value: 0

Table 48. DMA Channel Select settings.

DMA Channel	Setting
Channel 0	0 0
Channel 1	0 1
Channel 2	1 0
Channel 3	1 1

UART Receive Buffer/Transmit Holding Register (UARTx_RX_TX_REG)

Writing to this register sends data to the transmitter. Reading from this register captures data from the UART receiver. This register is only accessible when DLAB_BIT=0 in the UART's Line Control Register.

Bit	Description/Function
31:8	Reserved
7:0	**Receive Buffer/Transmit Holding Register (Data[7:0])**

Not reset. Undefined.

UART Interrupt Enable Register (UARTx_INT_ENABLE_REG)

This register is only accessible when DLAB_BIT=0 in the UART's Line Control Register.

Bit	Description/Function
31:4	Reserved
3	**Modem Status Interrupt Enable (MSE_BIT):** This bit enables the modem status interrupt sources (bits DELTA_CTS_BIT, DELTA_DSR_BIT, TERI_BIT and DELTA_DCD_BIT of the ***Modem Status Register***). 0: Disable 1: Enable
2	**Receiver Line Status Interrupt Enable (RLSE_BIT):** This bit enables the line status interrupt sources (bits OE_BIT, PE_BIT, FE_BIT and BI_BIT of the ***Line Status Register***). 0: Disable 1: Enable
1	**Transmit Holding Register Empty Interrupt Enable (THREE_BIT):** This bit enables the UART interrupt that signals when the transmit holding register is empty, indicated by the TEMT_BIT in the ***Line Status Register***. 0: Disable 1: Enable
0	**Receiver Data Ready Interrupt Enable (RDRE_BIT):** This bit enables the UART's Data Ready, Trigger Level and Timeout interrupts. 0: Disable 1: Enable

System reset value: 0

UART Line Control Register (UARTx_LINE_CONTROL_REG)

Bit	Description/Function
31:8	Reserved
7	**Divisor Latch Select (DLAB_BIT):** This bit must be set to access the baud-rate divisor registers. 0: Normal UART operation 1: Access baud-rate divisor registers
6	**Set Break (BREAK_BIT):** When set, this bit transmits a break condition to the receiving UART. When set, the serial output line (SOUT) is forced Low (the Spacing state). Clearing this bit releases the break condition. The break control bit acts only on the SOUT line and has no other effect on the transmitter. 0: Release break condition 1: Force break condition
5	**Stick Parity (STICK_PARITY_BIT):** When in the stick parity mode (SP=1 and PEN=1), the parity bit transmitted is the opposite polarity of the EPS bit programmed.
4	**Even Parity Select (EPS_BIT):** If set, the total number of 1's in the transmitted data bits + parity bit is even. If cleared, the total number of 1's is odd. 0: odd parity 1: even parity
3	**Parity Enable (PEN_BIT):** 0: No parity 1: Enable parity This bit controls the generation and transmission of parity bits in the transmitter and reception and checking of parity bits in the receiver.
2:0	**Stop Bits, Word Length Select (STB_WLS_FIELD)** (see Table 49)

System reset value: 0

Table 49. UART Data Format Settings.

STB_WLS_FIELD [2:0]			Data Format	
STB	WLS1	WLS0	Data Bits	Stop Bits
0	0	0	5	1
0	0	1	6	1
0	1	0	7	1
0	1	1	8	1
1	0	0	5	1.5
1	0	1	6	2
1	1	0	7	2
1	1	1	8	2

UART Line Status Register (UARTx_LINE_STATUS_REG)

This register is read-only.

Bit	Description/Function
31:8	Reserved
7	**Error Bit (ERROR_BIT):** This bit is valid only in FIFO mode. When set, it indicates that there is at least one parity, framing or break indication in the FIFO. It is reset when the *Line Status Register* is read and there are no subsequent errors in the FIFO.
6	**Transmitter Empty (TEMT_BIT):** When set, it indicates that the transmitter is idle (buffer or FIFO and transmitter shift register are empty).
5	**Transmit Holding Register Empty (THRE_BIT):** If set, the bit indicates that the UART is ready to accept a new character for transmission. This bit is cleared when a new data byte is loaded into the transmitter buffer or FIFO.
4	**Break Indicator (BI_BIT):** This bit is set if the duration of break transmission is longer than one word transmission time on all word boundaries.
3	**Framing Error (FE_BIT):** This bit is set whenever the stop bit following the data/parity bit is logic 0. It is reset by reading the Line Status Register.
2	**Parity Error (PE_BIT):** This bit is set upon detection of a parity error. It is reset by reading the *Line Status Register*.
1	**Overrun Error (OE_BIT):** If set, this bit indicates that the receiver buffer or FIFO has overflowed. The data causing the overflow is lost. The bit is reset when the *Line Status Register* is read.
0	**Data Ready (DR_BIT):** This bit is set when a complete incoming character is transferred into the receiver buffer. Reading the receiver buffer clears this bit. In FIFO mode, this bit is set whenever a character is received and transferred to the receiver FIFO. It is reset after reading all the bytes from the receiver FIFO.

System reset value: 0x60

UART Modem Control Register (UARTx_MODEM_CONTROL_REG)

Bit	Description/Function
31:5	Reserved
4	**Loopback Feature Enable (LOOP_BIT):** This bit enables a local loopback feature for diagnostic testing of the UART. The output of the transmitter is looped back into the receiver. 1: Enable 0: Disable
3:2	Reserved
1	**Request To Send (RTS_BIT):** This bit controls the RTS sideband signal level.
0	**Data Terminal Ready (DTR_BIT):** This bit controls the DTR sideband signal level.

System reset value: 0

UART Modem Status Register (UARTx_MODEM_STATUS_REG)

This register is read-only.

Bit	Description/Function
31:8	Reserved
7	**Data Carrier Detect Flag (DCD_BIT):** This bit reflects the state of the DCD sideband signal.
6	**Ring Indicator Flag (RI_BIT):** This bit reflects the state of the RI sideband signal.
5	**Data Set Ready Flag (DSR_BIT):** This bit reflects the state of the DSR sideband signal.
4	**Clear To Send Flag (CTS_BIT):** This bit reflects the state of the CTS sideband signal.
3	**Delta DCD Bit (DELTA_DCD_BIT):** This bit is set whenever the DCD sideband signal changes state.
2	**Trailing Edge of RI Bit (TERI_BIT):** This bit is set whenever a falling edge on the RI sideband signal is detected.
1	**Delta DSR Bit (DELTA_DSR_BIT):** This bit is set whenever the DSR sideband signal changes state.
0	**Delta CTS Bit (DELTA_CTS_BIT):** This bit is set whenever the CTS sideband signal changes state.

Default reset value: 0x000000(xxxx0000)

UART Interrupt Identification Register (UARTx_INT_ID_REG)

This register shares the same address as the FIFO Control Register.

Bit	Description/Function
31:8	Reserved
7:6	**FIFO Mode Indicator (FIFO_MODE_FIELD [1:0]):** 00: FIFO mode disabled (16C450-style mode) 11: FIFO mode enabled (16C550-style mode)
5:4	Reserved
3:0	**Interrupt Identification (INT_ID_FIELD [3:0]):** (see Table 50)

System reset value: 1

UART FIFO Control Register (UARTx_FIFO_CTRL_REG)

This register is write-only. It shares the same address as the Interrupt Identification Register.

Bit	Description/Function
31:8	Reserved
7:6	**Receiver FIFO Trigger Level (F FIFO_TRIG_LEVEL_FIELD[1:0]):** These bits define the trigger level of the receiver FIFO interrupt. 00: Trigger Level = 1 01: Trigger Level = 4 10: Trigger Level = 8 11: Trigger Level = 14
5:3	Reserved
2	**Transmit FIFO Clear (TX_FIFO_CLR_BIT):** Writing a '1' to this bit clears all bytes in the transmit FIFO. This bit is self-clearing.
1	**Receiver FIFO Clear (RX_FIFO_CLR_BIT):** Writing a '1' to this bit clears all bytes in the receiver FIFO. This bit is self-clearing.
0	**FIFO Mode Enable (FIFO_MODE_ENABLE_BIT):** Writing a 1 enables both the transmitter and receiver FIFOs. Resetting this bit clears all bytes in both FIFOs. This bit must be '1' in order to write to the other bits of the *FIFO Control Register*.

UART Scratchpad Register (UARTx_SCRATCHPAD_REG)

This register is used as a simple one-byte scratchpad. It is provided for compatibility purposes.

Bit	Description/Function
31:8	Reserved
7:0	**UART Scratchpad Register (Scratchpad[7:0])**

Not reset. Undefined.

UART Divisor Latch LSB Register (UARTx_DIVISOR_LSB_REG)

This register is only accessible when DLAB_BIT=1 in the UART's Line Control Register.

Bit	Description/Function
31:8	Reserved
7:0	**Baud Rate Generator Divisor Least Significant Byte (DIVISOR_LSB_FIELD [7:0])**

Not reset. Undefined.

UART Divisor Latch MSB Register (UARTx_DIVISOR_MSB_REG)

This register is only accessible when DLAB_BIT=1 in the UART's Line Control Register.

Bit	Description/Function
31:8	Reserved
7:0	**Baud Rate Generator Divisor Most Significant Byte (DIVISOR_MSB_FIELD [15:8])**

Not reset. Undefined.

Table 50. Interrupt Identification.

Interrupt ID				Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset Control
3	2	1	0				
0	0	0	1	—	None	None	—
0	1	1	0	1 (Highest)	Receiver Line Status	Overrun Error, Parity Error, Framing Error, or Break Interrupt	Read from **_Line Status Register_**
0	1	0	0	2	Received Data Available	Receiver data available or FIFO trigger level reached	Read from **_Receiver Buffer Register_** or the FIFO drops below the trigger level
1	1	0	0	2	Character Timeout (FIFO mode only)	No characters have been removed from or input to the receiver FIFO during the last 4 character times and there is at least 1 character in the FIFO during this time.	Read from the **_Receiver Buffer Register_**
0	0	1	0	3	Transmitter Holding Register Empty	Transmitter Holding Register Empty	Write to the **_Transmitter Holding Register_** or read from the **_IIR Register_**, if this condition was the source of the interrupt
0	0	0	0	4 (Lowest)	Modem Status	Clear to Send, Data Set Ready, Ring Indicator, or Data Carrier Detect	Read from the **_Modem Status Register_**

NOTE:

Setting the MSB and LSB of the UARTx_DIVISOR_xxx_REG to zero yields an undefined result.

DMA Controller

The DMA unit is composed of four independent channels. Via program settings, each channel performs a series of transfers between an I/O device and memory or performs memory-to-memory transfers. A set of parameters defines the operation of a given transfer including the memory source or destination addresses, the transfer count and a variety of other transfer characteristics. These parameters are either …

- programmed directly into the corresponding DMA channel control registers by the CPU, or
- assembled into a series of two-word or four-word descriptors, which operate independently of the CPU.

DMA Interaction with A7S System

The DMA controller interacts with the other subsystems that comprise the A7S system, as shown in Figure 65. The DMA interface to each subsystem has been optimized in some manners, as described below. Each of the four channels operates independently over the CSI bus.
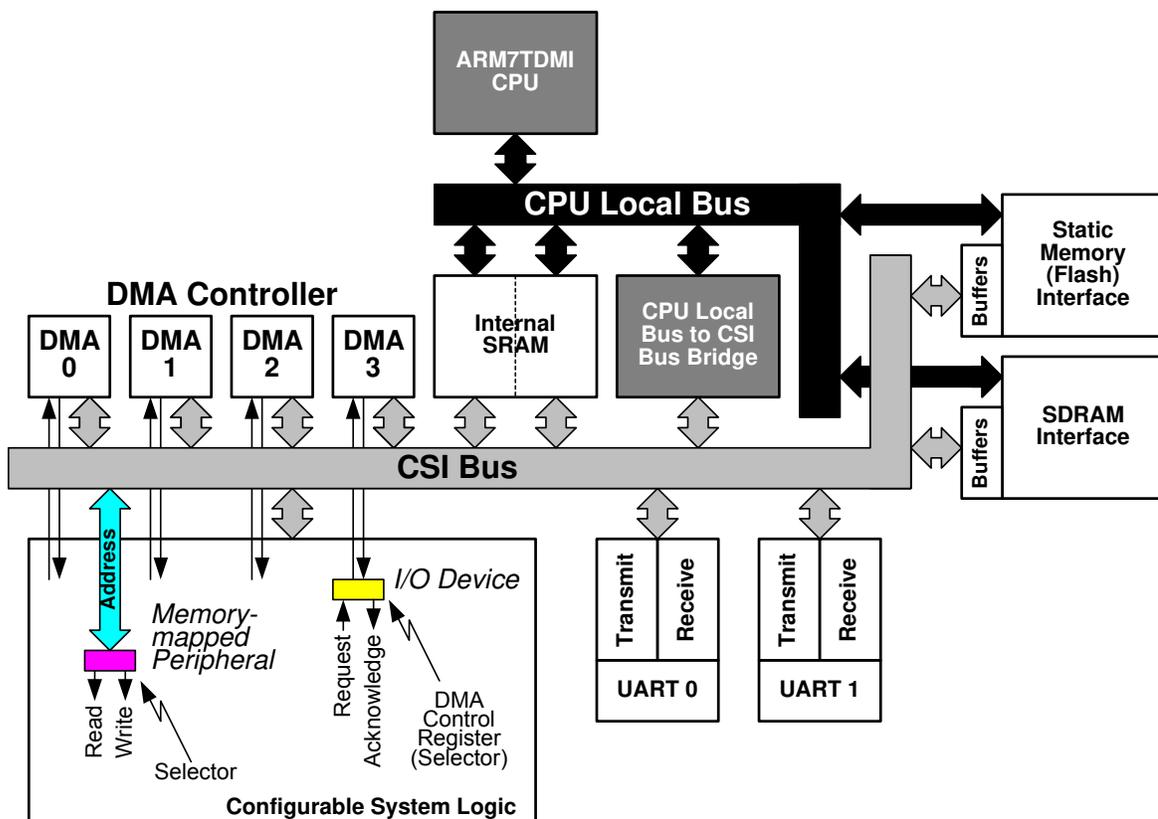
Figure 65. The DMA Controller communicates with functions over the CSI bus and directly with I/O devices implemented in the CSL matrix.

External Memory Interface

There are optional transfer buffers available when the DMA communicates with external memory. These buffers help maximize system performance. Each DMA channel has optional buffers for transfers to and from SDRAM and Flash, controlled by the DMA_BUF_EN_FIELD.

UARTs

The receiver and transmitter in both UARTs can be optionally assigned to a specific DMA channel via the *__UART Control Register__* for each UART. The RX_DMA_SEL_FIELD controls which DMA channel communicates with the receiver, enabled by the RX_DMA_EN_BIT. Similarly, the TX_DMA_SEL_FIELD defines which DMA channel communicates with the transmitter, enabled by the TX_DMA_EN_BIT.

DMA requests from a UART receiver are handled as device-to-memory transfers. Conversely, requests from a UART transmitter are handles as memory-to-device transfers.

Internal SRAM

The DMA can transfer data between the CSI bus and internal SRAM without wait-states. By restricting the CPU to one half of the internal SRAM and the DMA to the other half, then both the CPU and DMA access internal SRAM without causing wait-states. This technique is powerful for building ping-pong buffers where the DMA transfers data to or from one half of internal SRAM while the CPU processes data stored in the other buffer. When the DMA and CPU finish their tasks, both switch to the opposite half and continue the task.

The SRAM_PROTECT_FIELD optionally provides write-protection for 4Kbyte portions of internal SRAM. This capability exists because the DMA can write into SRAM and the SRAM can contain executable code.

Configurable System Logic (CSL)

The DMA communicates with both memory-mapped peripherals and input/output (I/O) devices implemented in the CSL matrix. In both cases, the DMA works in conjunction with a Selector to access the CSL-based function. If the DMA communicates with a memory-mapped function, then the Selector decodes the address presented on the CSI bus. If the DMA communicates with an I/O device, then the Selector acts as a DMA Control Register for the device, steering request and acknowledge signals to and from the selected DMA channel.

A7S Control Registers Protected Against DMA Writes

The A7S control register are protected against DMA write access by default, via the REMAP_ACC_PROTECT_REG register. This is to prevent an incorrectly specified DMA write transaction from accidentally corrupting the control registers and potentially placing the A7S in an unexpected state.

Restricting DMA Access

There is no hardware protection that prevents the DMA channels from accessing other restricted areas of memory. If the operating system needs to restrict an application from using DMA read and write transactions to some regions in memory space, then the DMA control registers should only be made available via supervisor-mode calls.

DMA Transfer Types

Each DMA channel supports three different types of transfers, as summarized in Table 51.

Table 51. DMA Transfer Types.

Transfer Type	Source	Destination	Typical Request Source	Acknowledge Destination
Memory-to-memory	Memory	Memory	CPU sets the Software Request Bit	No acknowledge signal generated
Memory-to-device	Memory	CSL	REQSEL input to DMA Control Register	ACKSEL output from DMA Control Register
Device-to-memory	CSL	Memory	REQSEL input to DMA Control Register	ACKSEL output from DMA Control Register

Memory-to-memory transfers

Most memory-to-memory transfers begin when the CPU sets the Software Request Bit for a specific channel. It is possible to request a memory-to-memory transfer from within the CSL matrix using the REQSEL input on a DMA Control Register. However, no acknowledge signal is generated for this type of transfer.

Unlike the Triscend E5 CSoC family, memory-to-memory transfers use only a single A7S DMA channel.

Device transfers

The device side of a memory-to-device or device-to memory DMA transfer resides either in a UART or in the CSL matrix.

By setting bits in the ***UART Control Register***, a DMA channel services requests for the UART transmitter and another channel services the receiver.

With CSL-based devices, a DMA Control Register associated with the requesting device forwards any requests to the selected DMA channel and steers the acknowledge signal back to the CSL logic, as shown in Figure 66. Additional control signals provide early termination and retransmit request capabilities.

Transfer Data Widths

A DMA channel transfers three possible data widths, including word, half-word, and byte transfers. The transfer data width is controlled by the TRANS_SIZE_FIELD in the channel control register. The size of the transfer also controls the auto-increment or auto-decrement value of the address pointers.

If the data width of the device is different than the width of the external memory subsystem, and the transfer is to external memory, then the external memory interface reacts accordingly. For example, a word-wide transfer from a device to byte-wide external static memory results in four individual write operations by the memory controller.

Automatic Address Generation

Each DMA channel provides automatic address generation, based on the source and destination address registers. The current source and destination address pointers are automatically and independently updated after each transfer.

Prior to starting a DMA request, the source address for a transfer is loaded into the ***Transfer Source Address Register***. The address loaded must be correctly aligned based on the transaction data width, as indicated in Table 52. The source address is only appropriate for transfers that originate from memory, *i.e.*, memory-to-memory or memory-to-device transfers, as shown in Table 53.

The destination address for a transfer is loaded into the ***Transfer Destination Address Register***. Again, the address loaded must be correctly aligned on the transfer size, as indicated in Table 52. The destination address is only appropriate for transfers that conclude in memory, *i.e.*, memory-to-memory or device-to-memory transfers, as shown in Table 53.

Table 52. Address Alignment and Auto Increment/Decrement Value by Transfer Size.

Transfer Width	Source/Destination Address Alignment	Auto Increment/ Decrement Value
Word (32 bits)	Word (A[1:0]=00	± 4
Half-Word (16 bits)	Half-Word (A[0]=0)	± 2
Byte (8 bits)	Byte	± 1

Table 53. Source and Destination Address Requirements by Transfer Type.

Transfer Type	Source Address	Destination Address
Memory-to-Memory	✓	✓
Memory-to-Device	✓	
Device-to-Memory		✓

When a DMA channel is initialized, the contents of the source register is copied to the ***Current Source Address Register*** and the destination register copied to the ***Current Destination Address Register***.

After each valid transfer, the DMA channel updates the current source and destination registers, as appropriate. The DMA controller provides auto increment and decrement capabilities. Based on the settings in the SRC_ADDR_MODE_FIELD, the current source address is either …

▪ incremented,

▪ decremented, or

▪ remains constant

… after the end of each DMA transaction. Similarly, the current destination address is modified according to the DEST_ADDR_MODE_FIELD.

The amount that an address pointer is incremented or decremented also depends on the DMA transaction data width, indicated in the TRANS_SIZE_FIELD. For example, a word-

wide transfer increments or decrements an address register by 4, pointing to the next word location. See Table 52.

Controlling Device-Side DMA Transfers

I/O devices, implemented in CSL logic, attach to a particular DMA channel via a DMA Control Register, as shown in Figure 66. A DMA Control Register is actually an alternate function provided by each CSI bus address Selector and these are distributed throughout the CSL matrix.
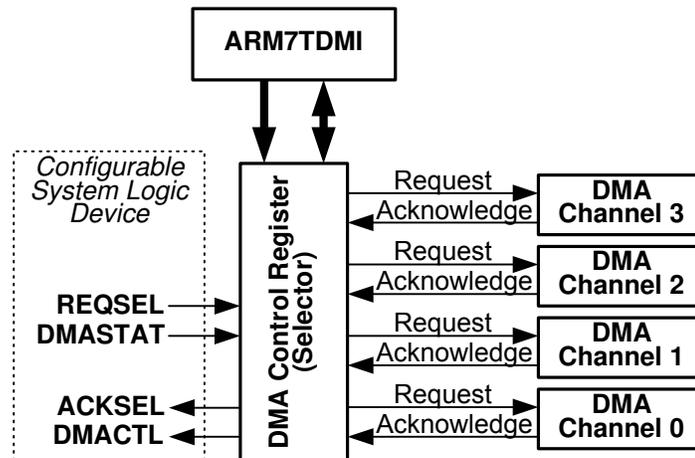

Figure 66. DMA Control Registers steer control signals between a CSL-based device and a specified DMA channel.

In most applications, application software executing on the CPU assigns a DMA Control Register to a specific DMA channel. Each DMA Control Register is individually addressed, located in memory. The address for a DMA Control Register is programmable, just like any Selector address. Typically, only the symbolic address name is specified for a DMA Control Register. The actual address assignment is usually left to the Triscend FastChip development system.

A DMA control register is enabled for DMA access by setting the associated enable bit. Until enabled, all DMA requests from a DMA Control Register are ignored. When enabled, the DMA Control Register forwards any requests on the REQSEL input to the assigned DMA channel and steers any acknowledge signals from the DMA back to the ACKSEL output. Similarly, the DMACTL and DMASTAT signals are also steered to the assigned DMA channel.

Most transfers to or from a device begin when a device requests a transfer by asserting the REQSEL input on its associated DMA Control Register. The DMA channel indicates that it is ready to send or receive data by asserting the ACKSEL output on the associated DMA Control Register.

In standard use, only one DMA Control Register is enabled per channel, per transfer direction at any time. Other DMA Control Registers can share a DMA channel, but must only be enabled via software after first disabling the active control register.

DMA Control Register Connections

Table 54 shows the inputs to a DMA Control Registers from the CSL matrix. By appropriately driving the inputs, a CSL-based device can request a new DMA transfer using the REQSEL input or terminate a currently-active transfer using the DMASTAT input.

The DMASTAT input has no meaning for memory-to-memory transfers.

Table 54. Device-Side DMA Request and Status Signals.

REQSEL	DMASTAT	Action
0	0	No Request
1	0	Request
0	1	Retransmit
1	1	Last request

Shaded entries do not apply for memory-to-memory transfers.

Similarly, Table 55 shows the outputs from a DMA Control Register to the CSL matrix. A CSL-based device uses these outputs to control the data phase of a DMA transfer. When the DMA Control Register asserts the ACKSEL output, the CSL device either accepts or presents data, depending on the direction of the DMA transfer. The DMACTL signal acknowledges any early termination request.

The DMACTL input has no function for memory-to-memory transfers.

Table 55. Device-Side DMA Control and Acknowledge Signals.

ACKSEL	DMACTL	Action
0	0	No Acknowledge
1	0	Acknowledge
0	1	Retransmit Acknowledge
1	1	Last Acknowledge

Shaded entries do not apply for memory-to-memory transfers.

E5 Backward Compatibility Mode

To maintain backward compatibility with Triscend E5 family designs, a bit (AUX_DIS_BIT) in each DMA channel selectively enables or disables the DMASTAT and DMACTL control signals for all associated DMA Control Registers.

Distributed DMA Control Register

Bit	Description/Function
7:4	Duplicate bits 3:0
3	Reserved
2:1	**DMA Channel Select:** Steers control signals between a CSL-based device and the selected DMA controller. The DMA Channel Select bits must be written to all nibbles. When read, all nibbles must be OR-ed together. 00: Channel 0 01: Channel 1 10: Channel 2 11: Channel 3
0	**Enable:** Allows a CSL-based device to access DMA services via the selected DMA channel. The Enable control bit must be written to all nibbles. When read, all nibbles must be OR-ed together to determine the actual value. 0: Disable DMA services for this device. 1: Enable DMA services for this device.

Configuration reset value: 0 (Disabled)

The register mnemonic for a distributed DMA Control Register is user defined as a symbolic address. FastChip's Generate utility allocates a DMA Control Register on a word boundary. The control register can be accessed as a word, half-word, or byte but be sure to duplicate all nibbles during writes and OR together all bits of a nibble during reads.

NOTE:	*The distributed DMA Control registers only connect to one of the eight nibbles on the CSI data bus. Consequently, application code must write duplicate copies of the high and low nibble. When reading, only one of the eight nibbles will contain valid data. All nibbles should be OR-ed together to determine the actual settings. For example, to enable a DMA Control Register to access DMA channel 2 using a word-wide write, write the value 0x55555555 to the DMA Control Register address.*

Device-to-Memory Handshake

Figure 67 shows an example of a device-to-memory transfer, originating in the CSL matrix. A logic function in the CSL indicates that it is ready to provide data by asserting the REQSEL input on its associated DMA Control Register. Bits within the DMA Control Register dictate whether the request is forwarded to the DMA controller and to which specific channel. Upon receiving a request, the assigned DMA channel indicates that it is ready to receive the data by asserting its acknowledge signal. The acknowledge signal is forwarded to the DMA Control Register, which asserts its ACKSEL output.

In most applications, the ACKSEL signal enables data on the CSI Data Read bus. Upon receiving the data, the DMA stores the data in memory at the current location defined in the DMA channel transfer parameters.
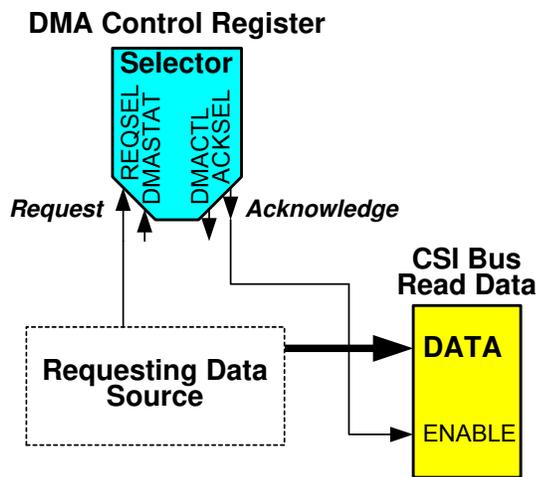
Figure 67. Device-to-memory transfer.

An example device-to-memory transfer appears in Figure 68. The example waveform assumes minimum latency. Prior to the request, one of the DMA channels is configured for a device-to-memory operation and the proper values loaded into its configuration registers.

① The CSL-based device requests to transfer data to memory, by asserting REQSEL before the next rising edge of Bus Clock.

② The selected and enabled DMA channel asserts ACKSEL and expects the device to present its data on the CSI Data Read port. Data must be set-up and valid before the next rising clock edge. For any given REQSEL, the corresponding ACKSEL is always one bus clock-cycle in duration. There are no wait-states allowed on a device-side data transfer.

③ The DMA channel writes the data to the memory location specified in the DMA channel's configuration registers.
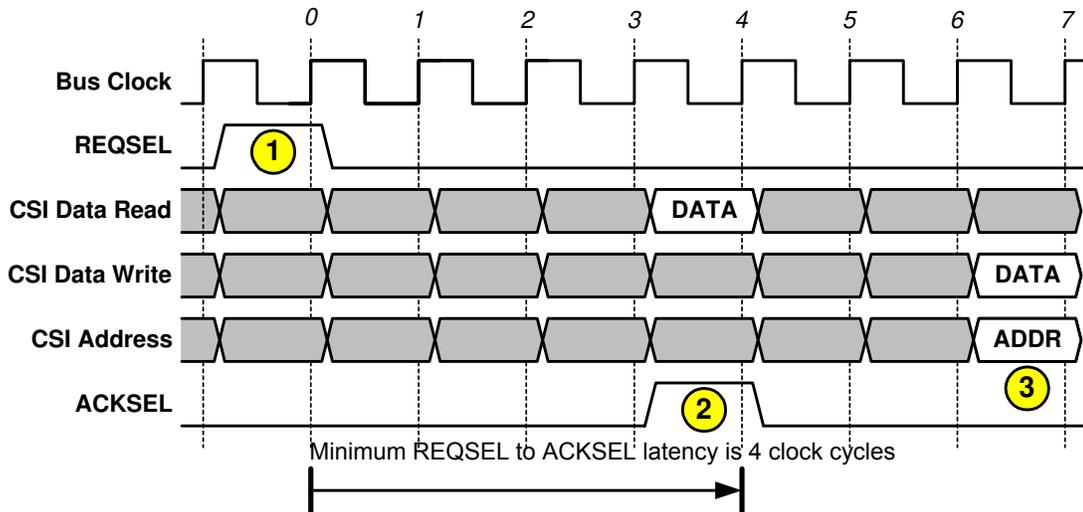
Figure 68. Example device-to-memory DMA transfer, assuming minimum latency.

Memory-to-Device Handshake

Figure 69 shows an example of a memory-to-device transfer, ending in a CSL-based device. A logic function in the CSL indicates that it is ready to receive data by asserting the REQSEL input on its associated DMA Control Register. Bits within the DMA Control Register dictate whether the request is forwarded to the DMA controller and to which specific channel. Upon receiving a request, the assigned DMA channel grabs data from memory at the location defined by the DMA channel's current transfer parameters. The DMA then presents valid data on the CSI Data Write port and indicates that data is ready by asserting its acknowledge signal. The acknowledge signal is steered back to the appropriate DMA Control Register and appears on its ACKSEL output.

In most applications, the ACKSEL signal drives an enable signal on a register that is to capture the data. The clock to the register is the Bus Clock, which is also available on the CSI bus socket interface. After capturing the DMA transfer, the data is available to other CSL logic on the 'Q' output of the data register.
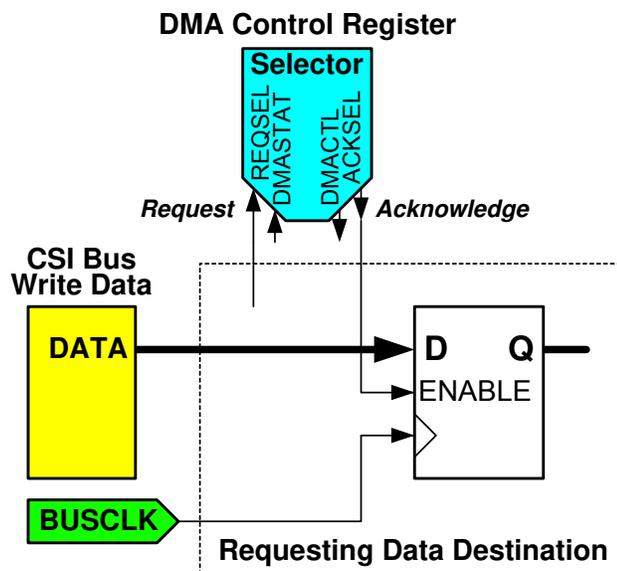

Figure 69. Memory-to-device transfer.

An example memory-to-device transfer appears in Figure 68. The example waveform assumes minimum latency. Prior to the request, one of the DMA channels is configured for a memory-to-device operation and the proper values loaded into its configuration registers.

Likewise, a DMA control register—part of a CSL logic function—is enabled and the channel select bits are set to steer signals to the proper DMA channel.

① The CSL-based device requests data from memory by asserting REQSEL before the next rising edge of Bus Clock.

② The selected and enabled DMA channel reads the requested data from the memory location specified in the DMA channel configuration registers.

③ The DMA presents data on the CSI Data Write bus and asserts ACKSEL. For any given REQSEL, the corresponding ACKSEL is always one bus clock-cycle in duration. There are no wait-states allowed on a device-side data transfer. The receiving CSL-based device must accept the data on or before the next rising edge of Bus Clock.
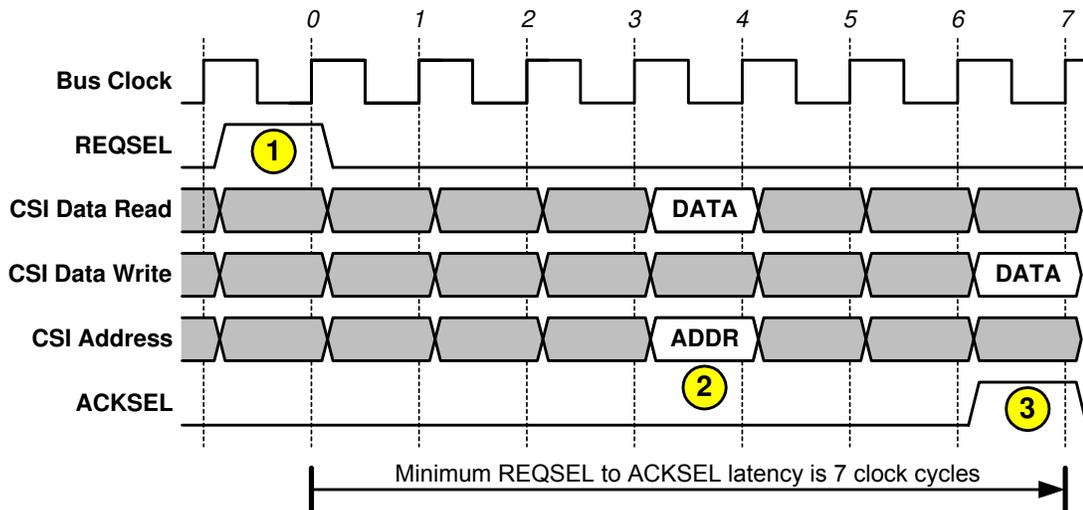
Figure 70. Example memory-to-device DMA transfer, assuming minimum latency.

DMA Requests

A DMA transfer begins with a request either by the CPU or from a device implemented in the CSL matrix. The DMA controller treats both types of requests equally.

CPU Requests via Software Request Bit

The CPU requests DMA services by setting the Software Request bit (SFT_REQ_BIT) for a specific DMA channel. This bit is self-clearing.

The CPU generally begins memory-to-memory transfers in this manner, though transfers to and from a device and memory are also supported.

Device-Side Requests

Devices implemented in CSL logic request a DMA transfer by asserting the REQSEL input on an enabled DMA Control Register. The request is forwarded to the appropriate DMA channel.

A valid request is recognized under the following conditions.

▪ The REQSEL input to the DMA Controller Register is a logical High at the rising-edge of Bus Clock.
▪ The DMA Control Register is enabled and associated with one of the four DMA channels.
▪ The DMA channel is enabled and initialized.

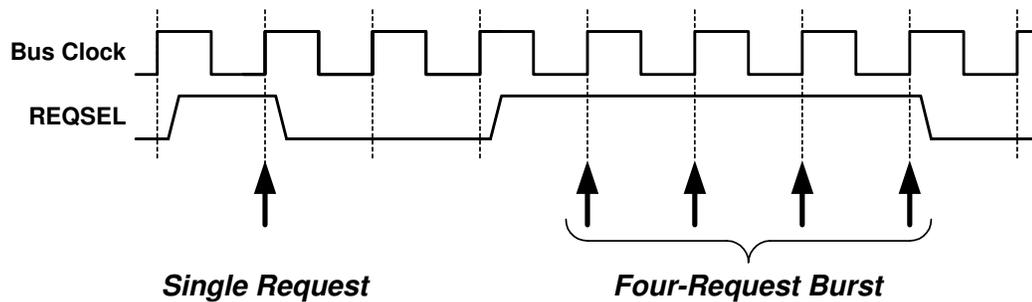A separate request is recognized on every clock edge where REQSEL is asserted, as shown in Figure 71.

Figure 71. Example DMA Requests.

Single versus Block Requests

In most cases, a single request results in a single data transfer by the DMA channel. However, if the DMA channel is in block request mode, then each request causes the DMA to transfer a block of data, the size of which is specified in the ***Transfer Count Register***.

The device requesting a DMA transfer must be ready to send or receive data in a single Bus Clock cycle, in response to an ACKSEL acknowledge signal. There are no wait-states allowed for device-side transfers. Only memory-mapped CSL functions may have wait-states.

Pending Request Counter

Once appropriately configured, a DMA channel receives all valid requests but does not guarantee immediate service. Any un-serviced requests—for which the DMA channel has not yet transferred data—are tracked in the DMA channel's 10-bit ***Pending Request Counter***. The counter accumulates requests only if the corresponding DMA channel is enabled. Software can read the value of the counter at any time. The pending requests counter is cleared by setting the DMA clear bit or by enabling the DMA channel.

Up to 1,023 un-service requests can be accumulated. If a DMA channel accumulates too many un-serviced requests, the channel can generate an interrupt, if so enabled.

The counter only tracks the number of requests signaled. If Block Request is enabled, then each block transfer counts as one request.

If the auxiliary handshake signals are enabled, then a device can terminate the DMA transaction early by signaling a "last" request. The position of that last request is stored, so that the transaction can be terminated early, after service any requests that occurred prior to the last request. The DMA controller maintains a separate "last" counter internally as well as the pending request counter.

The "last" request functionality is not available if Block Request is enabled.

DMA Acknowledge Cycles

Once a CSL-based device requests a DMA transfer, control over the transaction shifts to the DMA channel. The DMA channel requests the CSI bus. Once granted the bus by the CSI bus arbiter, the DMA channel asserts its acknowledge signal High during each bus cycle in which the DMA expects to send or receive data. On the device side, the ACKSEL output on a DMA Control Register steers this acknowledge signal back to the requesting CSL-based device.

> **NOTE:**
>
> *The ACKSEL signal is only generated for device-to-memory or memory-to-device transfers. No acknowledge signal is generated for memory-to-memory transfers.*

When ACKSEL is asserted High, the CSL-base device must respond appropriately. During a device-to-memory transfer, the CSL-based device presents data on the Data Read port, controlled by ACKSEL. During a memory-to-device transfer, the CSL-based device accepts data from the Data Write port, again controlled by ACKSEL. When accepting data, the ACKSEL signal typically connects to the enable inputs of flip-flops that capture the write data from the CSI bus socket.

A valid acknowledge occurs under the following conditions.

▪ A device issued a valid request.

▪ The DMA channel is enabled and initialized.

▪ The DMA Control Register is enabled and associated with one of the four DMA channels.

▪ The ACKSEL output from the DMA Controller Register is a logical High at the rising-edge of Bus Clock.

A separate data acknowledge occurs on every clock edge where ACKSEL is asserted, as shown in Figure 72.
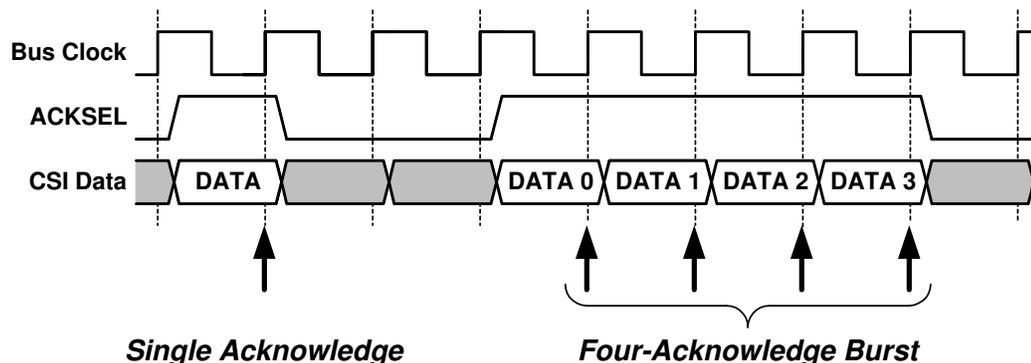

Figure 72. Example DMA acknowledges cycles.

There may not be a direct correlation between the number of DMA requests to a channel and the DMA acknowledge responses from a channel. This is because a singe request might ask for a block transfer whereas an acknowledge signal occurs for every individual data transfer.

Also, there is no correlation between the relative timing of requests and the corresponding acknowledge signals. A group of single-data requests, spaced every five clock cycles might result in a group of acknowledge signals in five consecutive clock cycles. The only correlation is that the data first requested is also the data first transferred and first acknowledged.

The DMA may acknowledge a series of transfers on a consecutive clock cycles, up to the number of pending requests tracked in the ***Pending Request Counter***.

The device requesting a DMA transfer must be ready to send or receive data in a single Bus Clock cycle, in response to an ACKSEL acknowledge signal. There are no wait-states allowed for device-side transfers. Only memory-mapped CSL functions may have wait-states.

REQSEL to ACKSEL Latency

The minimum REQSEL to ACKSEL latency is between four to seven CSI bus clock cycles, as shown in Table 56. It is difficult to specify absolute latency numbers because the latency depends on a variety of factors. The CSI bus supports multiple bus masters, controlled by a round-robin bus arbitration scheme. There can be just one or up to ten bus masters active on the CSI bus, depending on the application. Furthermore, other CSL functions that connect to the CSI bus may be in the middle of a transaction and may require wait-states.

Table 56. Minimum REQSEL to ACKSEL Latency.

DMA Transfer Direction	Latency	Units
Memory-to-Device	7	CSI bus
Device-to-Memory	4	clock cycles

Terminating a Transfer

A DMA transfer terminates by one of four possible means, as described below.

1. Normal Termination

2. Early Termination via a "Last" Request

3. Retransmit Termination Request

4. Early Termination by Software

Normal termination is the most common and works best for data transfers with a known size. However, the A7S provides enhanced capabilities for transferring blocks with an unknown size using auxiliary control signals available on each DMA Control Register.

Normal Termination

A DMA transfer terminates normally when a DMA channel's ***Transfer Counter*** reaches zero.

Early Termination via a "Last" Request

In many applications, the size of a DMA data transfer may not be known in advance. Data packets or frames may be of varying length. With the A7, CSL-based devices can terminate the current DMA transfer before reaching the maximum terminal count.

This early termination capability is not available for block transfers, *i.e.*, if the BLOCK_EN_BIT is enabled. Likewise, this capability is not available for memory-to-memory transfers.

When transferring data of unknown length, allocate a buffer in memory, large enough to accommodate the largest expected data transfer—*e.g.*, a complete frame. A frame might consist of a single data transfer or enough data to completely fill the buffer. The requesting device controls the amount of data transferred by notifying the DMA channel when the transfer ends.

To notify the DMA channel of the last transfer request, the CSL-based device simultaneously asserts both REQSEL and the DMASTAT signal. This action terminates the current transfer and the DMA controller tags the transfer as completed. In descriptor mode, the transfer count and status are updated in the corresponding descriptor by the DMA, and the DMA controller continues on to the next indicated action.

Figure 73 shows an idealized device-to-memory transfer with early termination, when the CSL-based device issues a "Last" request.

① The CSL-based device starts requesting that the DMA channel transfer data to memory by asserting the REQSEL input on the associated DMA Control Register. The requests do not need to occur on consecutive clock cycles.

② To notify the DMA controller that the third request is also the "Last" request, the device asserts the DMASTAT input along with the REQSEL input on the DMA Control Register.

③ After being granted the CSI bus, the enabled and initialized DMA channel acknowledges the data transfer by asserting ACKSEL on the DMA Control Register. The acknowledge signals may or may not occur on consecutive clock cycles. The CSL-based device uses the ACKSEL to enable the requested data onto the CSI Data Read port of the CSI bus socket.

④ The DMA notifies the CSL-based device of the "Last" acknowledge by asserting both the ACKSEL and DMACTL outputs on the DMA Control Register. Again, the CSL-based device provides data onto the CSI Data Read port of the CSI bus socket.

⑤ The DMA channel stores the requested data away in memory. In this example, the data is stored at incrementing addresses, controlled by the DMA channel's configuration registers. The transfers to memory may or may not occur on consecutive clock cycles.
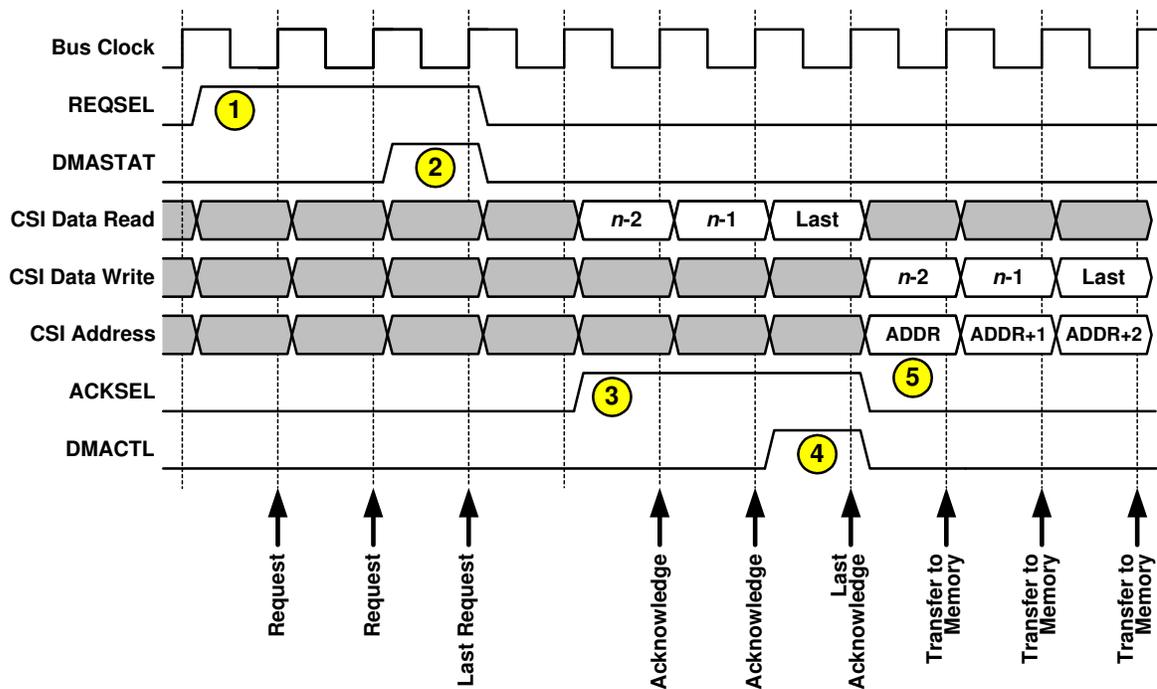

Figure 73. An idealized device-to-memory transfer showing early termination via a "Last" request.

Retransmit Termination Request

A CSL-based device can notify the DMA channel that it wishes to receive the data again, usually because of a data transmission error. To request retransmission, the device assert the DMASTAT input on a DMA Control Register, without asserting the REQSEL input. In response to such a request, the selected DMA channel re-executes the current transfer. There is no need for software interaction to restart the transfer. **Before requesting retransmission, the device must wait for the DMA unit to acknowledge all pending requests. Additionally, the device must wait for the DMA unit to acknowledge the retransmission before issuing any new requests.**

In Descriptor mode, the DMA channel restarts the existing description table entry. The DMA channel does not re-fetch the current descriptor table entry.

The DMA channel always services any pending requests accumulated in the Pending Request Counter before proceeding with the retransmit request. The DMA channel acknowledges the request by asserting the DMACTL output, without asserting the ACKSEL output. Upon receiving the DMACTL signal, the device can safely issue new requests.

This retransmission capability is not available for memory-to-memory transfers.

The DMA channel generates an interrupt if a device requests retransmission during a linked transfer, in the period between consecutive descriptor entries.

A device must not issue a retransmission request after receiving a Last Acknowledge (ACKSEL=High, DMACTL=High) nor after the device issues a Last Request (REQSEL=High, DMASTAT=High).

Figure 74 shows an idealized device-to-memory transfer with a retransmit termination request.

① The CSL-based device starts requesting that the DMA channel transfer data to memory by asserting the REQSEL input on the DMA Control Register associated with the device.

② After being granted the CSI bus, the enabled and initialized DMA channel acknowledges the data transfer by asserting ACKSEL on the DMA Control Register. The acknowledge signals may or may not occur on consecutive clock cycles. The CSL-based device uses the ACKSEL to enable the requested data onto the CSI Data Read port.

③ The CSL-based device must wait until after all pending requests have been acknowledged before requesting retransmission of the current transfer. The device issues a retransmit termination request to the DMA channel by asserting DMASTAT High but holding REQSEL Low.

④ The DMA channel stores the previously requested data away in memory.

⑤ The DMA acknowledges the retransmission request by asserting the DMACTL output on the DMA Control Register. The ACKSEL output is Low.

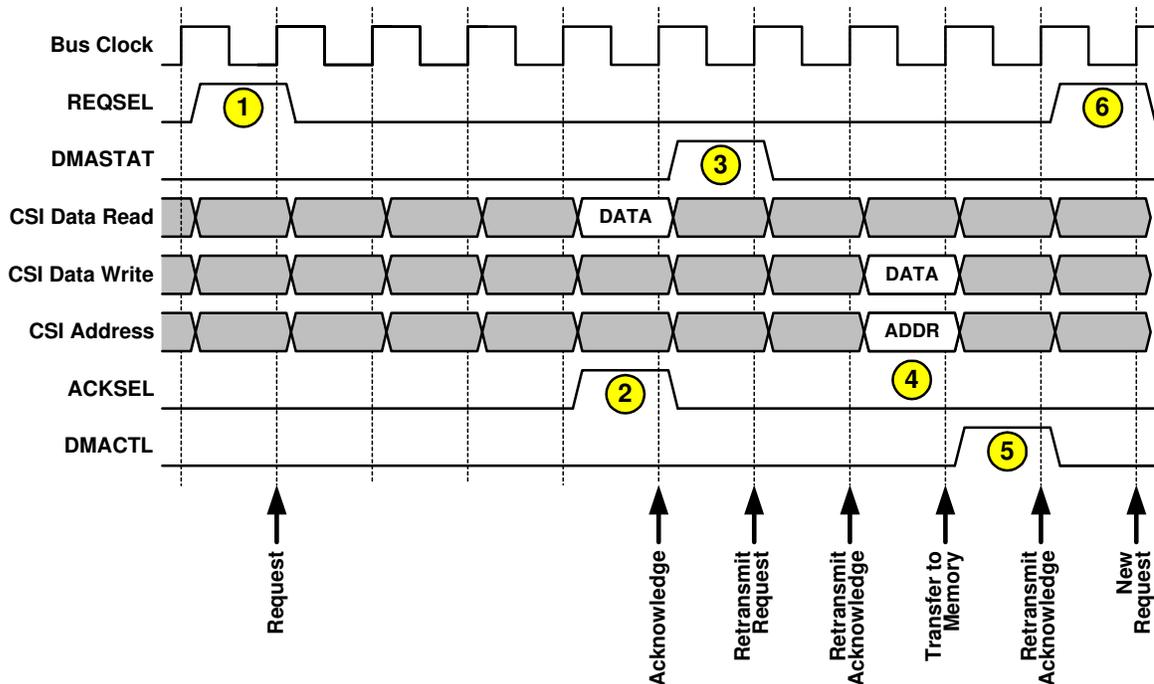⑥ After the retransmit request is acknowledged (Step ⑤), the device is allowed to issue a new request.

Figure 74. Idealized device-to-memory transfer showing a retransmit request.

Early Termination by Software

The CPU terminates an active DMA transfer by clearing the DMA channel's DMA_ENABLE_BIT. This method should be used only as a last resort.

Direct Mode

In direct mode, the CPU or other master writes to a DMA channel's control registers to configure the transfer parameters and mode of operation. This method requires that the CPU update these registers on any subsequent transfers. The address and transfer count are doubled-buffered inside each DMA channel. Once a transfer starts, the CPU can setup the next transfer, offering linked-list transfers while operating in direct mode.

Initializing a direct-mode DMA transfer entails programming the control registers for a particular DMA channel.

Prepare DMA Channel

1. Clear the DMA channel setting the CLEAR_BIT. Allow enough bus cycles for the longest possible memory access to complete.

2. Release the CLEAR_BIT.

3. The METHOD_BIT must be 0 for direct mode transfers.

4. If not using the auxiliary DMA signals to signal early termination requests or retransmission requests, set the AUX_DIS_BIT.

5. If, after completing this transfer, the DMA channel should repeat this same transfer again, using the same transfer parameters, set the CONT_BIT for continuous initialization. If a single transfer or if the DMA channel will be loaded with different parameters after completing the current transfer, clear the CONT_BIT.

Define the Transfer Width, Size, and Direction

6. Define the width of the data transferred by the DMA using the TRANS_SIZE_FIELD.

7. Define the type of DMA transfer using the TRANS_DIR_FIELD.

8. Load the DMAx_TRANS_CNT_REG with the number of bytes to transfer, minus one. For example, if transferring 64 pieces of data, load the transfer count register with 63.

9. If each incoming request asks for a block transfer, set the BLOCK_EN_BIT. The DMAx_TRANS_CNT_REG defines the size of the block transfer. Otherwise, if each request asks for a single transfer, clear the BLOCK_EN_BIT.

Define the Source Address and Automatic Addressing Options

10. If performing device-to-memory transfers (TRANS_DIR_FIELD=10'b), skip to Step 13.

11. If performing memory-to-memory or memory-to-device transfers, load the DMAx_SRC_ADDR_REG with the starting memory address where the data is located.

12. Define in automatic addressing options for the source address using the SRC_ADDR_MODE_FIELD. After every transfer, the DMA channel modifies the DMAx_CUR_SRC_ADDR_REG according to the defined addressing option.

Define the Destination Address and Automatic Addressing Options

13. If performing memory-to-device transfers (TRANS_DIR_FIELD=00'b), skip to Step 16.

14. If performing memory-to-memory or device-to-memory transfers, load the DMAx_DST_ADDR_REG with the starting memory address where the data will be stored.

15. Define in automatic addressing options for the source address using the DEST_ADDR_MODE_FIELD. After every transfer, the DMA channel modifies the DMAx_CUR_DEST_ADDR_REG according to the defined addressing option.

Define DMA Interrupt Conditions

16. Clear the current DMA interrupt register value by writing 0xF to the DMAx_INT_CLEAR_REG.

17. Enable the desired DMA interrupt conditions in the DMAx_INT_ENABLE_REG. Once a selected interrupt condition occurs, the corresponding bit location is set in the DMAx_INT_REG and the DMA channel generates an IRQ interrupt.

18. Enable the specific IRQ request associated with the DMA channel via the INT_IRQ_ENABLE_REG. The interrupt register bit locations are shown in Table 46.

Start the DMA Transfer

19. Enable the DMA channel to receive requests by setting the DMA_ENABLE_BIT. Setting this bit also clears the DMAx_PEND_REQ_REG. Any incoming requests are now recognized but they will not be serviced until the DMA channel is initialized. The incoming requests are accumulated in the DMAx_PEND_REQ_REG.

20. Initialize the DMA channel by setting the DMA_INIT_BIT. Setting this bit loads the various starting operating values into the appropriate control registers. Once the transfer has started, the DMA_INIT_BIT is automatically cleared by hardware. After completing the transfer, the DMA channel waits for a new initialization command. If none are present, the DMA stops, otherwise it continues with the next transfer. While the DMA is busy with a transfer, the CPU can set up the next transfer and set the DMA_INIT_BIT, after the CPU checks that the current transfer has actually started. If

the DMA_INIT_BIT is zero or its corresponding interrupt is set, then it is safe to update the parameters.

Descriptor Mode

Descriptor mode allows a DMA channel to perform complex linked transfers without much CPU interaction. Setting the METHOD_BIT in a DMA channel's control register activates descriptor mode. Linked transfers are a series of single transfers where the DMA controller automatically deduces the parameters for the next transfer from the values in the descriptor table. This method frees the CPU from the tedious task of constantly monitoring and managing the DMA transfers.

Figure 75 shows an example linked transfer using descriptor mode. A Descriptor Table is created somewhere in memory and a DMA channel is configured to perform descriptor mode transfers. The METHOD_BIT is set and the channel's other static parameters are defined. The DMA channel is enabled and initialized. A descriptor table entry is loaded when either …

- the CPU sets the DMA_INIT_BIT and the DMA channel is in descriptor mode, or
- the previous entry in the descriptor table has completed.

① When the DMA channel is initialized, the DMA controller begins fetching a command at the beginning of the Descriptor Table, located at the Descriptor Table Base Address.

② The DMA controller reads the command parameters and programs the DMA channel's configuration registers. The controller loads the source and destination addresses, as required by the transaction, and the transfer count.

③ After the transfer completes, the DMA controller examines the Buffer Status and the Action When Transfer Complete bits in the table entry to determine the next course of action. Assuming that the next action to continue, the DMA fetches the next entry from the Descriptor Table and performs the next transfer.
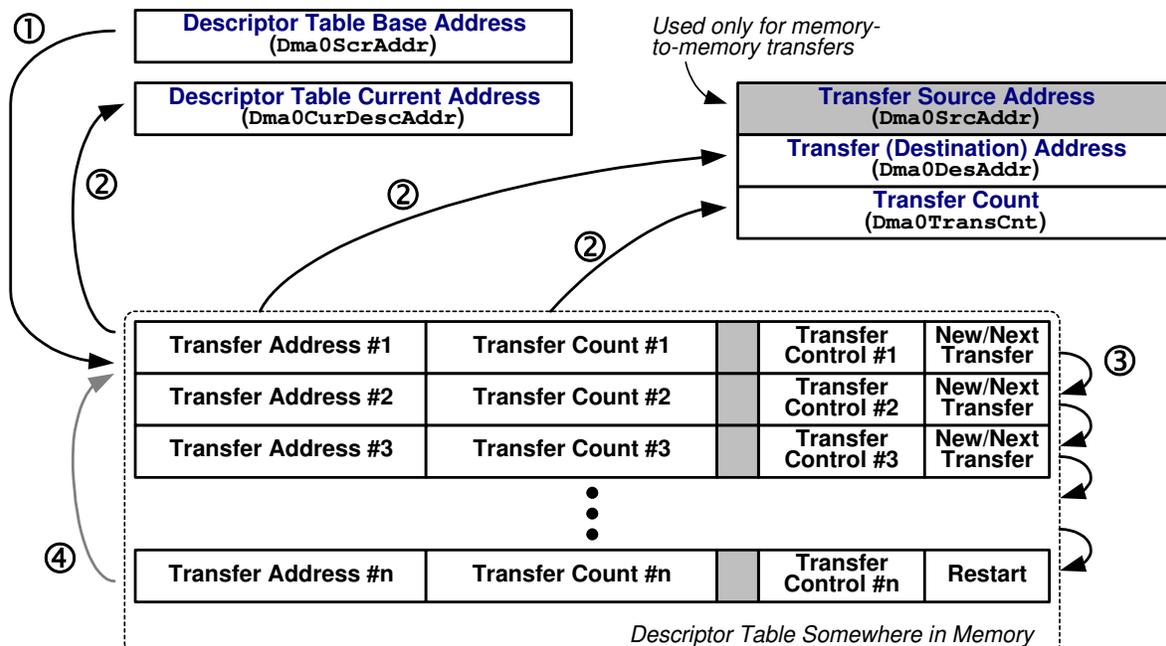

Figure 75. A descriptor mode DMA transfer using DMA Channel 0.

④ The process continues until a descriptor entry commands the DMA channel to halt or to restart the linked transfer chain again.

The descriptor table resides anywhere in the system memory map, including memory-mapped functions in the CSL matrix. Two or more DMA channels can share Descriptor Table values by setting the ***Descriptor Table Base Address*** to the same value in multiple DMA channels.

Descriptor Table Format

> **NOTE:**
> ✔
> *Not all DMA controls are accessible through a descriptor table entry. Some static settings, such as the DMA channel's Transfer Direction, must be set when the DMA is first initialized.*

The Transfer Descriptor Table, located at the address defined in the DMAx_DES_TABLE_ADDR_REG register has *n* entries. Each entry has two or four command words depending on the type of transfer. In Table 57 and Table 58, *n* refers to the "n^{th}" entry in the Descriptor Table.

If calculating offsets from the current descriptor table address, set *n*=0.

Table 57 Transfer Descriptor Table Format for Memory-to-Device or Device-to-Memory Transfers

Address		Field
Base	**Offset**	**Field**
DMAx_DES_TABLE_ADDR_REG	+ (*n* • 8) + 0	Transfer Start Address
	+ (*n* • 8) + 4	Transfer Control/Status

Table 58 Transfer Descriptor Table Format for Memory-to-Memory Transfers

Address		Field
Base	**Offset**	**Field**
DMAx_DES_TABLE_ADDR_REG	+ (*n* • 16) + 0	Transfer Source Address
	+ (*n* • 16) + 4	Reserved
	+ (*n* • 16) + 8	Transfer Destination Address
	+ (*n* • 16) + 12	Transfer Control/Status

Memory-to-device or device-to-memory transfers only require a start address. Memory-to-memory transfers require both source and destination addresses.

Descriptor Definition

Each descriptor is composed of two or four 32-bit command words, as defined below.

Bit	Description/Function
127:96	**Transfer Source Address:** This field defines the source address for memory-to-memory transfers. This field is not required for memory-to-device or device-to-memory transfers.
96:64	Reserved This field is not required for memory-to-device or device-to-memory transfers.
63:32	**Transfer Start/Destination Address:** This field defines the destination address during memory-to-memory transfers. Otherwise, it is used as the transfer address.
31:16	**Transfer Count:** The requested number of bytes, **minus one**, to transfer. The transfer count is always measured in bytes, regardless if the transferred data is 16 or 32 bits wide. The DMA channel updates this field with the actual number of bytes transferred, **minus one**, when the transfer ends.
15:6	Reserved
5	**Buffer Status:** Does not apply to memory-to-memory transfers. 0: **Buffer empty**. The buffer, pointed to by the descriptor, does not contain valid data. This bit is cleared after completing a memory-to-device descriptor transfer, indicating that the buffer is empty. It is now safe for a DMA channel to write to the buffer. A DMA channel cannot read from the buffer until this bit is set. 1: **Buffer full**. The buffer, pointed to by the descriptor, contains valid data. This bit is set after completing a device-to-memory descriptor transfer. It is now safe for a DMA channel to read from the buffer. A DMA channel cannot write to the buffer until this bit is cleared.
4	**Continuous Descriptor:** When set, instructs the DMA unit that a continuous descriptor is required. The DMA logic automatically executes the next entry in the Descriptor table after the current transfer completes. There is no need for a software or device request to initiate the next transfer. The first transfer entry in the Descriptor table *always* requires an explicit request to start the DMA transfer. 0: Wait for a new DMA request before continuing with the next table entry. 1: Continue to next entry without waiting for another DMA request.
3	**CRC Clear Disable:** This bit has no effect on the CRC Register in <u>Direct Transfer Mode</u>. 0: Reset the ***CRC Register*** on each transfer. 1: Do not reset the ***CRC Register*** on each transfer.
2	**Descriptor Interrupt:** 0: Disable descriptor interrupt. 1: Generate an interrupt upon completing the current descriptor transfer.
1:0	**Action When Transfer Complete:** This field specifies the DMA channel's next action upon completion of the current descriptor transfer. A list of possible actions follows. 00: *Halt* – Do nothing. 01: *New/Next* – Start new DMA transfer from the next entry of the Descriptor Table. 10: *Restart* – Start new DMA transfer from the first entry of the Descriptor Table. 11: *Repeat* – Repeat this transfer again.

Descriptor Table Fetching

This section describes in more details on linked DMA transfers using a Descriptor Table as described above. The following is only applicable when using Descriptor mode.

The DMA controller performs the following operations when in Linked Transfer method:

1. When the DMA_INIT_BIT is set to initiate the DMA channel, the DMA controller loads the ***Descriptor Table Base Address Register*** into ***Current Descriptor Address Register***. The transfer starts at beginning of descriptor table and the first transfer is initiated.

2. The DMA channel fetches the descriptor entry pointed to by the ***Current Descriptor Address Register*** and loads the associated registers for the DMA channel. The DMA controller performs some internal housecleaning functions before starting the next transfer, which requires between 5 to 15 bus cycles.

3. The DMA channel checks the Buffer Status bit in the descriptor table entry to check if it is safe to read from or write to the buffer. If not, the DMA monitors the BROADCAST BIT to establish when to re-fetch the descriptor. If the BROADCAST_BIT is set, the channel returns to Step 2, otherwise the channel waits.

4. Once the descriptor is loaded into the DMA channel, the controller is idle until it receives a DMA request. Upon receiving the request(s), the DMA performs the required transaction(s).

5. When the transfer completes or is terminated, the DMA updates the descriptor counter and buffer status entries in memory.

6. The DMA channel examines the transfer control bits in the descriptor table entry to determine what action, if any, to perform next. The DMA controller either loads the next contiguous descriptor, repeats the current descriptor, halts, or returns back to the beginning of the descriptor table. The DMA clears the Buffer Status bit in the current descriptor just before moving on to the next descriptor.

If the DMA controller attempts to load a new descriptor and finds the Buffer Status bit is not ready yet, then the DMA controller waits until a broadcast event is detected. When the broadcast event occurs, the entire descriptor is reloaded, including the address, count, control, and Buffer Status. At this point, the Buffer Status bit is checked again.

"Broadcast" events are generated in one of two ways.

 i. Another DMA channel completes a transaction.

 ii. The CPU writes the BROADCAST_BIT in the DMA control register.

DMA Buffer Management

Buffer management is only relevant in descriptor mode.

In descriptor mode, the DMA operates with a pre-defined set of buffers in memory, defined in the descriptor table entry. Before writing or reading to or from one of the buffers, the DMA controller checks the status of the buffer, indicated by status bits in the corresponding descriptor. After completing a descriptor transfer, the DMA controller also updates the transfer count within the descriptor, which handles those cases where the device marks the end of a transfer via a "Last" request. Updating the transfer count value tracks the size of the transfer.

It is possible to bridge two devices in the CSL using two DMA channels, as shown in Figure 76. Both DMA channels operate in descriptor mode and use the same descriptor table. Both channels transfer data in and out of the same transfer buffer, located somewhere in memory. When both channels are initialized, the respective DMA controllers check the buffer status to see if the buffer is full or empty. DMA 1, seeing that the buffer is empty, cannot transfer data from the buffer so it waits.
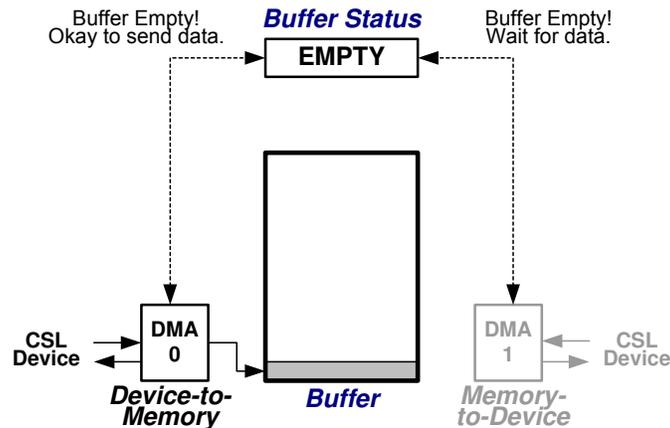
a.) When the transfer buffer is empty, DMA 0 fills the buffer from one device. DMA 1 must wait for the buffer to fill..
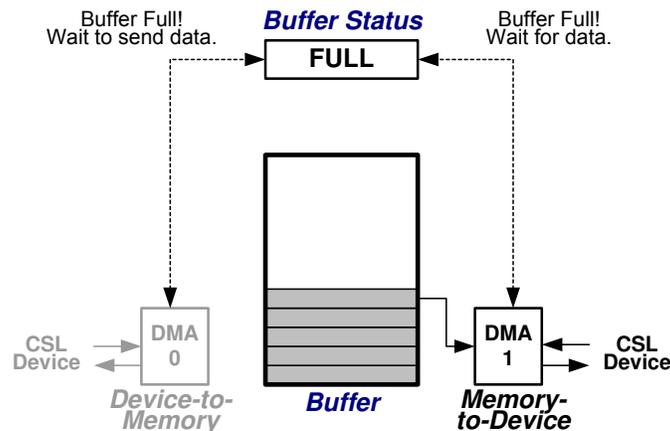
b.) After DMA 0 fills the buffer, it marks the buffer as FULL. DMA 1 now empties the buffer to another device. DMA 0 must wait for the buffer to empty.

Figure 76. An example of a buffer management. Two DMA channels bridge two CSL-based devices using a desriptor table.

DMA 0, seeing that the buffer is empty, gathers data from a device and sends it to the transfer buffer. Once DMA 0 completes the descriptor transfer, the DMA channel automatically updates the length count in the descriptor table entry and marks the buffer as full. It then fetches the next descriptor. If the descriptor indicates that the buffer is already full, then DMA 0 cannot proceed and waits for an empty buffer.

DMA 1 uses the same descriptor table entry for its transfer. Now that DMA 0 updated the length and marked the transfer buffer as full, DMA 1 uses the updated descriptor table entry to perform the opposite transfer, transferring data from the buffer to another device. The DMA channel reads the first descriptor and examines the status bits. If the buffer is full, it proceeds with the transfer as indicated by the updated transfer count. At the end of the transfer, it clears the buffer status bit, again marking the transfer buffer as empty.

DMA 1, seeing that the buffer is empty, again waits for a full buffer. DMA 0, seeing that the buffer is empty, continues with its next transfer. The transfers continue indefinitely or until both channels traverse through the descriptor table.

The DMA unit not only executes the data transfers but also handles the buffer management, off-loading the CPU from that task.

It is also possible for the CPU to fill or empty the transfer buffer. However, after filling or emptying the buffer, the CPU must update the buffer status bit and set the BROADCAST_BIT, which notifies all DMA channels that the buffer status has changed. A DMA channel automatically sets the BROADCAST_BIT whenever it updates the buffer status.

Clearing a DMA Channel

In case a DMA channel is not responding properly, it is possible to clear the channel by setting the CLEAR_BIT in the ***DMA Channel Control Register***. Once this bit is set, the DMA channel is in a reset state and remains so until after the bit is cleared.

In Descriptor mode, the procedure is slightly different. Clear the DMA channel's enable bit first. Then, clear the transfer method bit, which returns the channel to direct mode. Then, reset the DMA channel by setting the CLEAR_BIT.

DMA Interrupts

Each DMA channel optionally generates interrupts upon the following events.

- **Transfer Terminal Count**: this event is generated when the terminal count is reached (i.e., the ***Transfer Count*** reaches zero).

- **DMA Initialization Flag**: this event indicates that the DMA channel was initialized.

- **Pending Request Overflow**: this event is generated when the ***Pending Request Counter*** overflows, *i.e.*, there are greater than 1023 un-serviced requests. The DMA channel can no longer keep track of all the incoming requests. The Pending Request Counter has two parts: the number of requests not yet serviced and the position of the last transaction of a transfer within these pending requests. This interrupt is also asserted if there is already one request marked as "last" and the DMA controller receives a second "last" request with "last" tag.

- **"Last" Request Received Interrupt**: this event is generated upon reception of the "last" request from a device. Not applicable for memory-to-memory transfers.

- **Descriptor Interrupt**: the DMA controller asserts this interrupt, only in descriptor mode, if the Descriptor Table's entry specifies to generate an interrupt.

- **Linked Buffer Full Interrupt**: the DMA controller asserts this interrupt, only in descriptor mode, if the descriptor of the next transfer indicates that the buffer is full. Not applicable for memory-to-memory transfers.
- **Linked Buffer Empty Interrupt**: the DMA controller asserts this interrupt, only in descriptor mode, if the descriptor of the next transfer indicates that the buffer is empty. Not applicable for memory-to-memory transfers.
- **Retransmit Interrupt**: this event is generated upon a reception of a retransmit request from a DMA device. Not applicable for memory-to-memory transfers.
- **Bad Retransmit Interrupt**: the DMA unit asserts an interrupt and sets this bit, only in descriptor mode, if a device requests a retransmit in the period between consecutive descriptor entries. Not applicable for memory-to-memory transfers.

For each channel, each event is recorded separately in each channel's ***DMA Interrupt Register***. These bits form a combined single interrupt signal for each channel to the interrupt controller.

Cyclic Redundancy Check

All the DMA channels can perform Cyclic Redundancy Checking (CRC) on their data stream at any time. The CRC logic is shared between channels so the CRC checker must be assigned to only one channel by setting the CRC_EN_BIT in only one channel. A 0-to-1 transition on the CRC enable bit in the ***DMA Channel Control Register*** resets the CRC shift-register to all ones. Once enabled, the CRC checker is activated any time data is transferred. Once a transfer is complete, software can read the output of the CRC register and can compare the resulting signature against the expected signature.

The divisor polynomial used in the implementation is the standard CRC-32 32-bit polynomial shown in Equation 4.

$$CRC\text{-}32 = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1 \quad (4)$$

DMA/Cache Fill Interaction

In a high bandwidth application, where DMA transfers use the full CSI bus bandwidth and the Cache is being filled, half the bandwidth is allocated to the DMA and half the bandwidth to the Cache fills.

The arbitration between CPU and CSI accesses is done with no dead cycles.

DMA Channel 2 Control Registers Description

Address		Register Name	Access
Base	**Offset**		
DMA_BASE	+ 0x80	DMA 2 Control	R/W
	+ 0x84	DMA 2 Interrupt Enable	R/W
	+ 0x88	DMA 2 Interrupt	R
	+ 0x8C	DMA 2 Interrupt Clear	W
	+ 0x90	DMA 2 Descriptor Table Base Address	R/W
	+ 0x94	DMA 2 Source Address	R/W
	+ 0x98	DMA 2 Destination Address	R/W
	+ 0x9C	DMA 2 Transfer Counter	R/W
	+ 0xA0	DMA 2 Current Descriptor Address	R
	+ 0xA4	DMA 2 Current Source Address	R
	+ 0xA8	DMA 2 Current Destination Address	R
	+ 0xAC	DMA 2 Current Transfer Count	R
	+ 0xB0	DMA 2 Pending Requests Counter	R
	+ 0xB4	Reserved	-
	+ 0xB8	Reserved	-
	+ 0xBC	Reserved	-

DMA Channel 3 Control Registers Description

Address		Register Name	Access
Base	**Offset**		
DMA_BASE	+ 0xC0	DMA 3 Control	R/W
	+ 0xC4	DMA 3 Interrupt Enable	R/W
	+ 0xC8	DMA 3 Interrupt	R
	+ 0xCC	DMA 3 Interrupt Clear	W
	+ 0xD0	DMA 3 Descriptor Table Base Address	R/W
	+ 0xD4	DMA 3 Source Address	R/W
	+ 0xD8	DMA 3 Destination Address	R/W
	+ 0xDC	DMA 3 Transfer Counter	R/W
	+ 0xE0	DMA 3 Current Descriptor Address	R
	+ 0xE4	DMA 3 Current Source Address	R
	+ 0xE8	DMA 3 Current Destination Address	R
	+ 0xEC	DMA 3 Current Transfer Count	R
	+ 0xF0	DMA 3 Pending Requests Counter	R
	+ 0xF4	Reserved	-
	+ 0xF8	Reserved	-

CRC Register Description

Address		Register Name	Access
Base	**Offset**		
DMA_BASE	+ 0xFC	DMA CRC	R

DMA Channel Control Register (DMAx_CONTROL_REG)

Bit	Description/Function
31:18	Reserved
17	**Buffer RD/WR Broadcast Flag (BROADCAST_BIT):** This bit informs the DMA controller that it is now safe to read from or write to the transfer buffer. In descriptor mode, the CPU must set this bit whenever it reads or writes a memory buffer associated with a memory-to-device or device-to-memory descriptor transfer and sets or clears the associated Buffer Status bit. The bit is self-clearing. 0: No effect. 1: Notify DMA controller that CPU has updated transfer buffer.
16	**CRC Enable (CRC_EN_BIT):** A 0-to-1 transition on this bit resets the CRC logic to 0xFFFF_FFFF. 0: No effect. 1: Activate CRC logic for this channel.
15	**Auxiliary Flags Disable (AUX_DIS_BIT):** 0: Allows a device attached to the DMA channel to signal a "Retransmit" and "Last" condition to the DMA controller using the DMASTAT and DMACTL auxiliary input signals. The "Last" functionality is only applicable to modes where block mode is inactive. This capability is ignored if BLOCK_EN_BIT=1. 1: E5-compatibility mode. The auxiliary input signals are ignored.
14	**Transfer Method (METHOD_BIT):** 0: *Direct Mode* – the CPU programs the DMA controller with the transfer parameters 1: *Descriptor Mode* – The DMA channel fetches the DMA transfer parameters from descriptor table, built in memory.
13:12	**Transaction Size (TRANS_SIZE_FIELD [1:0]):** This field defines the width of data transferred by the DMA and controls the increment or decrement value for automatic address generation. 00: *Byte* (8 bits wide) 01 *Half-word* (16 bits wide) 10: *Word* (32 bits wide) 11: Reserved
11:10	**Destination Addressing Mode (DEST_ADDR_MODE_FIELD [1:0]):** This field defines the behavior of the destination address during a DMA transfer. The increment or decrement value is defined by the transaction size. 00: *Increment* – The destination address pointer increments after each transaction 01: *Decrement* – The destination address pointer decrements after each transaction 10: *Single* – The destination address pointer remains constant 11: Reserved

Bit	Description/Function
9:8	**Source Addressing Mode (SRC_ADDR_MODE_FIELD [1:0]):** This field defines the behavior of the source address during a DMA transfer. The increment or decrement value is defined by the [transaction size](#). 00: ***Increment*** – The source address pointer increments after each transaction 01: ***Decrement*** – The source address pointer decrements after each transaction 10: ***Single*** – The source address pointer remains constant 11: Reserved
7:6	**Transfer Direction (TRANS_DIR_FIELD [1:0]):** 00: ***Memory to Device*** 01: ***Device to Memory*** 10: ***Memory to Memory*** 11: Reserved
5	**Block Transfer Enable (BLOCK_EN_BIT):** 0: No effect. 1: The DMA performs a complete block transfer upon receiving a single request from a device. The ***[Transfer Count Register](#)*** defines the size of the block. The early termination requests, "Last" requests, are ignored from a device
4	**Software Request (SFT_REQ_BIT):** 0: No effect. 1: Issue a DMA request. Typically used by the software to begin DMA requests during memory-to-memory transfers. This bit is self-clearing.
3	**Continuous Transfer Initialization (CONT_BIT):** 0: No effect. 1: The DMA transfer continues to use same transfer parameters until this bit is cleared, which is equivalent to having the [DMA_INIT_BIT](#) always set. Ignored when the [METHOD_BIT](#) indicates Descriptor Mode.
2	**Transfer Initialization (DMA_INIT_BIT):** 0: If a transfer is active, stop after the current transfer completes. Otherwise, no effect. 1: Initialize a DMA transfer. When this bit is set, the starting operation values are loaded into their corresponding counters at the beginning of a transfer. Once the transfer has started, the bit is cleared by hardware. Software can set it again during the current transfer to prepare the DMA channel for the next transfer. However, software should not do this if Retransmit requests are allowed because the DMA controller requires the original start address and transfer count should a retransmit request occur.
1	**DMA Enable (DMA_ENABLE_BIT):** 0: Ignore incoming DMA transfer requests. 1: Enable the DMA channel to receive transfer requests. When first set, the pending requests counter is cleared and the DMA channel is ready to accept requests.

Bit	Description/Function
0	**DMA Channel Clear (CLEAR_BIT)**: 0: Release DMA channel reset. 1: Resets the associated DMA channel and clears the pending request counter and last counter. Must be held long enough for any active memory accesses to complete. Fifteen clock cycles is the maximum required.

System reset value: 0

DMA Interrupt Enable Register (DMAx_INT_ENABLE_REG)

This register is used to enable the various DMA interrupt conditions described in **DMA Interrupts**.

Bit	Description/Function
31:9	Reserved
8	**Bad Retransmit Interrupt Enable (BAD_RETR_BIT):** Not applicable for memory-to-memory transfers. 0: Disable interrupt source 1: Enable interrupt source
7	**Descriptor Interrupt Enable (DESC_BIT):** 0: Disable interrupt source 1: Enable interrupt source
6	**Retransmit Interrupt Enable (RETRANS_BIT):** Not applicable for memory-to-memory transfers. 0: Disable interrupt source 1: Enable interrupt source
5	**"Last" Flag Interrupt Enable (LAST_BIT):** Not applicable for memory-to-memory transfers. 0: Disable interrupt source 1: Enable interrupt source
4	**Buffer Empty Interrupt Enable (EMPTY_BIT):** Not applicable for memory-to-memory transfers. 0: Disable interrupt source 1: Enable interrupt source
3	**Buffer Full Interrupt Enable (FULL_BIT):** Not applicable for memory-to-memory transfers. 0: Disable interrupt source 1: Enable interrupt source
2	**Pending Request Overflow Interrupt Enable (OVF_BIT):** 0: Disable interrupt source 1: Enable interrupt source
1	**Initialization Interrupt Enable (INIT_BIT):** 0: Disable interrupt source 1: Enable interrupt source
0	**Terminal Count Interrupt Enable (TC_BIT):** 0: Disable interrupt source 1: Enable interrupt source

System reset value: 0

DMA Interrupt Register (DMAx_INT_REG)

This register reflects the various enabled interrupt conditions described in **_DMA Interrupts_**. This register is read-only.

Bit	Description/Function
31:9	Reserved
8	**Bad Retransmit Interrupt (BAD_RETR_BIT):** Not applicable for memory-to-memory transfers. 0: Not active 1: Active
7	**Descriptor Interrupt (DESC_BIT):** 0: Not active 1: Active
6	**Retransmit Interrupt (RETRANS_BIT):** Not applicable for memory-to-memory transfers. 0: Not active 1: Active
5	**"Last" Flag Interrupt (LAST_BIT):** Not applicable for memory-to-memory transfers. 0: Not active 1: Active
4	**Buffer Empty Interrupt (EMPTY_BIT):** Not applicable for memory-to-memory transfers. 0: Not active 1: Active
3	**Buffer Full Interrupt (FULL_BIT):** Not applicable for memory-to-memory transfers. 0: Not active 1: Active
2	**Pending Request Overflow Interrupt (OVF_BIT):** 0: Not active 1: Active
1	**Initialization Interrupt (INIT_BIT):** 0: Not active 1: Active
0	**Terminal Count Interrupt (TC_BIT):** 0: Not active 1: Active

System reset value: 0

DMA Interrupt Clear Register (DMAx_INT_CLEAR_REG)

This register is used to clear the various DMA interrupts described in *DMA Interrupts*. This register is write-only

Bit	Description/Function
31:9	Reserved
8	**Clear Bad Retransmit Interrupt (BAD_RETR_BIT):** 0: No effect 1: Clear interrupt
7	**Clear Descriptor Interrupt (DESC_BIT):** 0: No effect 1: Clear interrupt
6	**Clear Retransmit Interrupt (RETRANS_BIT):** 0: No effect 1: Clear interrupt
5	**Clear "Last" Flag Interrupt (LAST_BIT):** 0: No effect 1: Clear interrupt
4	**Clear Buffer Empty Interrupt (EMPTY_BIT):** 0: No effect 1: Clear interrupt
3	**Clear Buffer Full Interrupt (FULL_BIT):** 0: No effect 1: Clear interrupt
2	**Clear Pending Request Overflow Interrupt (OVF_BIT):** 0: No effect 1: Clear interrupt
1	**Clear Initialization Interrupt (INIT_BIT):** 0: No effect 1: Clear interrupt
0	**Clear Terminal Count Interrupt (TC_BIT):** 0: No effect 1: Clear interrupt

Not reset. Undefined.

Descriptor Table Base Address Register (DMAx_DES_TABLE_ADDR_REG)

This register points to the DMA descriptor table built somewhere in memory.

Bit	Description/Function
31:2	**Base Address of the Descriptor Table (A[31:2]):** This pointer is always aligned to word boundaries.
1:0	Reserved

Not reset. Undefined.

Transfer Source Address Register (DMAx_SRC_ADDR_REG)

This register is used during memory-to-device or memory-to-memory transfers.

Bit	Description/Function
31:0	**Transfer Source Address (A[31:0])**

Not reset. Undefined.

Transfer Destination Address Register (DMAx_DST_ADDR_REG)

This register is used during device-to-memory or memory-to-memory transfers.

Bit	Description/Function
31:0	**Transfer Destination Address (A[31:0])**

Not reset. Undefined.

Transfer Count Register (DMAx_TRANS_CNT_REG)

Bit	Description/Function
31:16	Reserved
15:0	**Transfer Count (TRANS_CNT_FIELD [15:0]):** This field specifies the number of bytes to transfer ***minus one***.

Not reset. Undefined.

Descriptor Table Current Address Register (DMAx_CUR_DESC_ADDR_REG)

This register is read-only.

Bit	Description/Function
31:2	**Base Address of the Current Descriptor (A[31:2]):** This pointer is always aligned to word boundaries.
1:0	Reserved

Not reset. Undefined.

Current Transfer Source Address Register (DMAx_CUR_SRC_ADDR_REG)

This register is used during memory-to-device or memory-to-memory transfers. This register is read-only.

Bit	Description/Function
31:0	**Current Transfer Source Address (A[31:0])**

Not reset. Undefined.

Current Transfer Destination Address Register (DMAx_CUR_DEST_ADDR_REG)

This register is used during device-to-memory or memory-to-memory transfers. This register is read-only.

Bit	Description/Function
31:0	**Current Transfer Destination Address (A[31:0])**

Not reset. Undefined.

Current Transfer Count Register (DMAx_CUR_TRANS_CNT_REG)

This register is read only.

Bit	Description/Function
31:16	Reserved
15:0	**Current Transfer Count (CUR_TRANS_CNT_FIELD [15:0]):** This field specifies the number of bytes remaining to transfer.

Not reset. Undefined.

DMA Pending Request Counter (DMAx_PEND_REQ_REG)

This register is read-only.

Bit	Description/Function
31:26	Reserved
25:16	**Position of "last" transaction (LAST_POST_FIELD [9:0]):** The DMA track the position of the request that has the tag "last" within the pending requests. It can only tracks one "last" flag position.
15:10	Reserved
9:0	**Pending Request Count (PEND_REQ_CTRL_FIELD [9:0]):** This field specifies the number of DMA requests yet to be serviced.

System reset value: 0.

DMA Cyclic Redundancy Check (DMA_CRC_REG)

This register is read-only.

Bit	Description/Function
31:0	**DMA Cycle Redundancy Check:** This field holds the result on the CRC-32 computation.

Not reset. Undefined.

Debugging Infrastructure

To aid real-time, in-system debugging, the A7S provides four dedicated on-chip resources.

IEEE 1149.1 JTAG Interface

An industry-standard, four-wire JTAG interface connects an A7S CSoC device to an external computer or tester. The JTAG controller within the A7S has full bus mastering capabilities on the CSI bus, offering complete observability and controllability over the entire device. The JTAG unit, along with debugging software, also controls the basic state of the device such as resetting it or freezing the state of the system after the next instruction finishes.

The JTAG port also directly initializes the CSoC device or updates external Flash memory devices connected to the A7S's external memory interface. When programming external Flash memory, the JTAG unit downloads a Flash-programming algorithm to the CSoC's internal RAM. It then interacts with the internal CPU, allowing the processor to perform the actual program/erase/verify algorithms while the JTAG port supplies the programming data.

Full ARM Debugging Support

The ARM7TDMI development environment offers a rich set of debugging tools for debugging a software application. The A7S device provides the standard ARM EmbeddedICE™ debugging support required to leverage ARM7TDMI debugging tools, in-
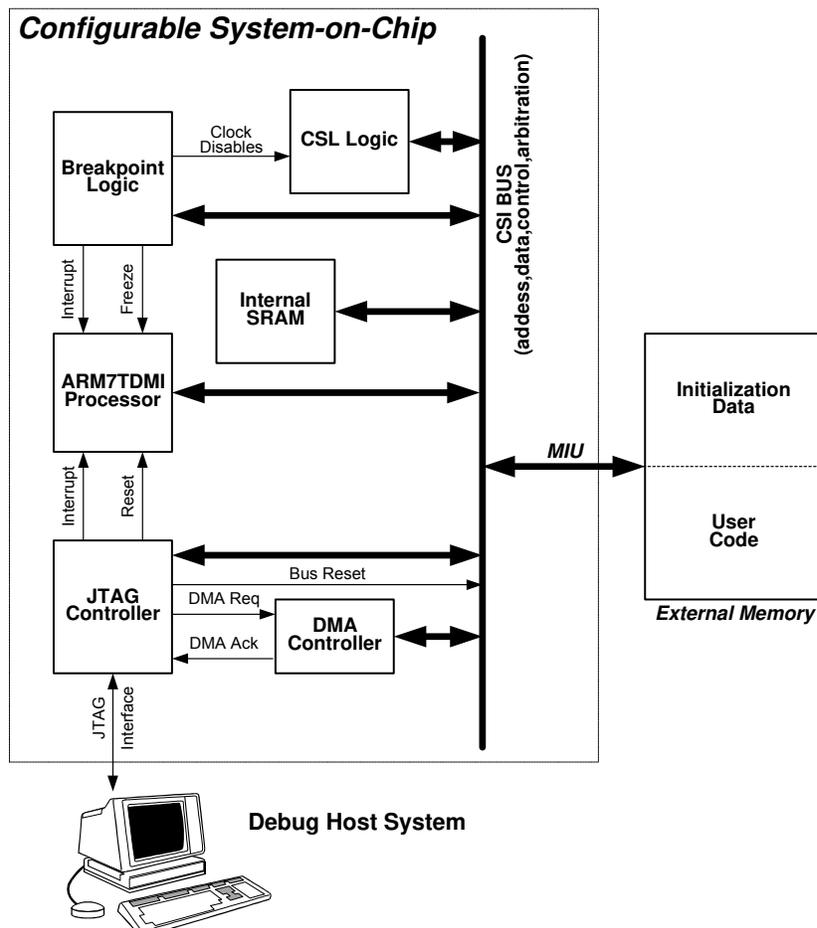

Figure 77. System debugging block diagram.

cluding internal breakpoint and watchpoint capabilities.

NOTE: ✓	*If using a third-party JTAG debugger or emulator, make sure that the target board allows the A7S's VSYS pin to be strapped Low. If VSYS is tied High, the A7S device will automatically power-down during the first power-on cycle because the A7S will be unable to locate a valid initialization image. Third-party debuggers are not fully Triscend aware and do not wake an A7S device from power-down mode. Strapping VSYS Low during debugging prevents the A7S from automatically entering power-down mode.*

Hardware Breakpoint Unit

An on-chip hardware breakpoint unit monitors the CSI or CPU local bus, and can interrupt or freeze the system when either bus matches a predefined set of conditions. The breakpoint unit provides two independent programmable breakpoints.

When a breakpoint condition occurs, the CPU either freezes at the end of the current instruction or receives a breakpoint interrupt and branches to the debugger interrupt routines, depending on its current configuration. Following a breakpoint freeze, CSL clocks or global signals can be blocked to aid system debugging. The Triscend FastChip software enables the user to specify which CSL clocks or global signals are affected following a breakpoint freeze.

FastChip Device Link, working with the Wind River visionPROBE II JTAG debugging cable, provides an advanced debugging environment including real-time system monitoring and control, logic probes, breakpoint capabilities, and trace.

While the CPU is frozen, the JTAG unit can poll any addressable location residing inside or outside the CSoC and sends all requested information to the host PC for further display and analysis.

At the end of the debugging session, the JTAG unit clears the breakpoint freeze condition and the ARM processor resumes executing code from where it left off. Alternatively, the JTAG unit can restart the software application by issuing a CPU reset over the JTAG connection.

At any point in time, the JTAG unit can single-step the processor and send pertinent display information to the host PC.

Internal Trace Buffer

The upper half of the internal scratchpad memory optionally serves as a trace buffer for the CSI or the CPU local bus. The trace window is defined by a breakpoint unit event.

> **NOTE:**
>
> *When the trace buffer is active, any data or code stored in the upper 8K bytes of internal RAM is corrupted.*

The A7S's debugging hardware is capable of tracing the following transactions.

CSI Bus Transactions

- **Trace All Cycles** – Capture all CSI bus cycles including all wait-state and idle cycles.
- **Trace All Valid Cycles** – Capture all the valid CSI bus activity, discarding all wait-state and idle cycles
- **Trace Specific Cycles**
  - DMA Channel Number – Trace cycles associated with a particular DMA channel
  - ARM CPU CSI cycles – Trace all ARM CPU activity on the CSI bus.

CPU Local Bus Transactions

- **Trace All Cycles** – Capture all CPU bus cycles including all Idle, Coprocessor, Non-Sequential, Sequential, and wait-state cycles.
- **Trace All Valid Cycles** – Capture all the valid CSI bus activity, discarding all wait-state and all idle cycles
- **Trace All Instruction Fetches**
- **Trace All Data Transfers**

Glossary

A7 – Triscend's configurable system-on-chip (CSoC) family based around an embedded 32-bit ARM7TDMI RISC processor.

ARM7TDMI – The word's best selling 32-bit RISC processor architecture, designed and supported by ARM Ltd. It is know for its high-performance for its relatively low cost and low power consumption.

BGA – Ball Grid Array. A type of surface-mount packaging technology where the package leads are solder balls located on the bottom-side of the package.

BPU – Breakpoint Unit. The hardware breakpoint unit embedded within the A7.

BSP – Board Support Package.

CSoC – Configurable System-on-Chip. A configurable device containing a dedicated processor, configurable logic, on-chip RAM, and a dedicated internal bus.

CPU – Central Processing Unit. The ARM7TDMI processor embedded within the Triscend A7S configurable system-on-chip.

CRU – Configuration Register Unit. Registers that define special functions on the Triscend A7S device.

CSL – Configurable System Logic. The peripheral configurable logic matrix providing user-programmable functions to the microcontroller and its peripherals.

CSI – Configurable System Interconnect. The on-chip bus that bridges the Configurable System Logic (CSL) matrix, the ARM7TDMI processor, the dedicated peripherals, and the Memory Subsystem Interface Unit (MSSIU).

DIMM – Dual In-line Memory Module. A small circuit board that holds memory chips. The A7S's SDRAM interface supports 100-pin DIMMs.

DMA – Direct Memory Access. A controller that offloads memory and I/O transfers from the microcontroller.

DSP – Digital Signal Processing.

E5 – Triscend's configurable system-on-chip (CSoC) family based on a high-performance version of the 8-bit 8051 microcontroller.

EMI – Electro-Magnetic Interference. The CSoC's internal system bus and I/O features help reduce EMI.

FastChip – The Triscend CSoC development system.

FDL – FastChip Device Link, the FastChip configuration, download, and debugging utility.

FIFO – First-In, First-Out memory.

Flash – A type of programmable, non-volatile memory that can be bulk erased.

ISR – Interrupt Service Routine. The program that handles a specific interrupt to the CPU.

JTAG – Joint Test Action Group. A general name given to the four-wire serial interface described in IEEE 1149.1.

LSB – Least-Significant Byte or Bit.

LUT– Look-Up Table. The basic Boolean logic function inside each CSL logic cell.

MSB – Most-Significant Byte or Bit.

MSSIU – Memory Subsystem Interface Unit. Connects the ARM7TDMI processor, its peripherals, and the internal system bus to external memory resources, including both Flash and SDRAM.

N.C. – No Connection. Two potential usages. Either a signal that should be left unconnected or a device package pin that is not connected to a PIO for a specific device.

PIO – Programmable Input/Output.

PQFP – Plastic Quad Flat Pack. A type of surface-mount packaging technology.

RISC – Reduced Instruction Set Computer. An architecture that favors performance and simplicity over a rich instruction set.

RTOS – Real-Time Operating System.

SDK – Software Development Kit.

SDRAM – Synchronous Dynamic Random-Access Memory.

TBD – To Be Determined. Unknown at the time this document was prepared.

Thumb – A technique to encode and compress a 32-bit ARM instruction into a code-efficient 16-bit instruction, with minimal loss in performance.

UART – Universal Asynchronous Receiver/Transmitter, a serial port.

Life Support Policy

Triscend's products are not authorized for use as critical components in life support devices or systems unless a specific written agreement pertaining to such intended use is executed between the manufacturer and the President of Triscend.

1. Life support devices or systems are devices which (a) are intended for surgical implant into the body or (b) support or sustain life and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in significant injury to the user.

2. Critical component in any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device system, or affect its safety or effectiveness.

Resources

Web Sites

ARM7TDMI Information

www.arm.com

ARM Development Guide

www.embedded.com/directories/embedded/arm2000/index.html

ARM Resources Links

www.bluewaternz.com/links.htm

Books
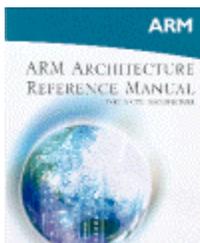
ARM Architecture Reference Manual

D. Jaggar, David Seal

Prentice Hall:
ISBN 0201737191
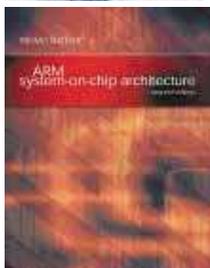
ARM System-on-Chip Architecture (Second Edition)

S. Furber

Addison-Wesley:
ISBN 0201675196
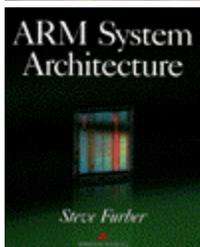
ARM System Architecture

S. Furber

Addison-Wesley:
ISBN 0201403528

Japanese translation available:
Book title: ARM Processor
Publishing company: CQ Publishing Co., Ltd.
ISBN 4789833518
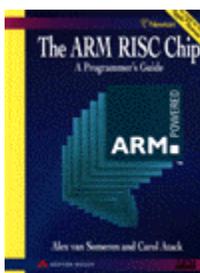
The ARM RISC Chip, A Programmer's Guide

A. van Someron and C. Atack

Addison-Wesley:
ISBN 0201624109

Pin Description

describes the pins available on an A7S configurable system-on-chip (CSoC) device. The directionality of each pin is described during parallel mode initialization. After initialization, all PIO pins and the indicated system pins are available as user-defined I/O signals.

Table 59. Pins available on an A7S configurable system-on-chip and their function.

| Pin Name	Pin Description	Parallel
A[19:0]	Address bus bits. These pins cannot be reclaimed as user-defined PIO pins.	O
CE0-	Active-Low chip-enable output to external memory. Enables all bytes in an 8-bit subsystem, or the first byte of a 16-bit or 32-bit subsystem, or the first half-word of a 32-bit subsystem using 16-bit wide memories. This pin cannot be reclaimed as a user-defined PIO pin.	O
CE[3:1]- /PIO	Optional active-Low byte-lane chip-enable signals. Used to enable external memory during byte or half-word addressing in 16-bit or 32-bit Flash memory subsystems. If not used as chip-enables, these pins may be reclaimed as user-defined PIO pins.	O
CLK/PIO	External clock source input. Can be reclaimed as a PIO pin if not used as the system clock source.	I
D[7:0]	Data bus bits. These pins cannot be reclaimed as user-defined PIO pins.	I
DQ[31:8]	Optional upper 24 bits of the Data bus. These pins may be reclaimed as user-defined PIO pins. Until initialization is complete, these pins have an internal high-impedance pull-up resistor.	I
GND	Ground connection for internal (non-PIO) logic. All must be connected.	I
GNDIO	Ground connection for PIO functions. All must be connected.	I
GNDPLL	Ground connection for the phase-locked loop (PLL). Must be connected.	I
N.C.	No connect. There is no function on this pin.	N.C.
OE-	Active-Low output-enable signal to read from external memory.	O
PIO	General-purpose input, output, or bi-directional signal pin after initialization is complete. Before initialization is complete, these pins have internal high-impedance pull-up resistors that pull the signal pin to a High logic level.	I (pull-up)
PIO/ A[23:20]	Typically, a general-purpose input, output, or bi-directional signal pin after initialization is complete. Before initialization is complete, these pins have an internal high-impedance pull-up resistor that pulls the signal pin to a High logic level. These pins optionally extend the Flash subsystem address up to the maximum 16Mbytes allowed after initialization. Any unused address lines may be used as user-defined PIO pins.	I (pull-up)
PIO/ A[31:24]	Typically, a general-purpose input, output, or bi-directional signal pin after initialization is complete. Before initialization is complete, these pins have an internal high-impedance pull-up resistor that pulls the signal pin to a High logic level. These pins are optionally used as the upper 8-bit of the 32-bit CSI bus address during testing.	I (pull-up)
PIO/GBUF5 PIO/GBUF4 PIO/GBUF3 PIO/GBUF2 PIO/GBUF1 PIO/GBUF0	Global buffer input. Typically, a general-purpose input, output, or bi-directional signal pin after initialization is complete. Before initialization is complete, this pin has an internal high-impedance pull-up resistor that pulls the signal pin to a High logic level.	I (pull-up)

Pin Name	Pin Description	Parallel
PIO/XDONE	General-purpose input, output, or bi-directional signal pin after initialization is complete. Before initialization is complete, this pin has an internal high-impedance pull-up resistor that pulls the signal pin to a High logic level. This pin is optionally used during testing.	I (pull-up)
RST-	Active-Low device reset input. Connect to VCCIO with a pull-up resistor. Do not allow it to float.	I
SDCE[1:0]-	Active-Low SDRAM chip-enable signals. If not used, leave unconnected.	O
SDCKE	SDRAM clock enable signal. Disables SDRAM during power-down conditions. If not used, leave unconnected.	O
SDCLK	SDRAM clock source. All SDRAM operations are synchronized to the rising edge of this signal. If not used, leave unconnected.	O
SLAVE-	Test slave mode. Connect to VCCIO. Do not allow it to float.	I
TCK	JTAG Test Clock input. Tie High if unused.	I
TDI	JTAG Test Data Input. Tie High if unused.	I
TDO	JTAG Test Data Output.	O
TMS	JTAG Test Mode Select input. Tie High if unused.	I
VCC	Supply voltage for internal logic functions, separate from I/O. Connect to a +2.5-volt supply. DO NOT CONNECT TO +3.3-VOLT SUPPLY. All must be connected and each must be decoupled with a 0.01 to 0.1 µF capacitor to ground.	I
VCCIO	Supply voltage for I/O functions, separate from internal logic. Connect all to a +2.5-volt or +3.3-volt supply, depending on external interface requirements. All must be connected and each must be decoupled with a 0.01 to 0.1 µF capacitor to ground.	I
VCCPLL	Supply voltage for phase-locked loop circuitry. Connect to a +2.5-volt supply. Must be connected and decoupled with a 0.01 to 0.1 µF capacitor to ground.	I
VSYS	External 'system voltage-good' indicator input. Indicates when external devices, such as the external Flash, have sufficient operating voltage. In most applications, connect VSYS to VCCIO supply source. If VSYS is High during initialization and no valid configuration is found, the CSoC device automatically powers down to conserve power. If VSYS is tied Low, the CSoC device attempts to initialize itself until it finds valid initialization data. This pin must be connected. Do not allow it to float.	I
WE-	Active-Low write-enable signal to write to external memory. Used to program external Flash-based devices.	O
XIN	The input from an external 32.768 kHz watch crystal into the internal crystal oscillator amplifier. Connect to one side of the external crystal as shown in Figure 47. Connect to GNDIO if not using an external crystal.	I
XOUT	Output from the 32.768 kHz internal crystal oscillator amplifier. Connect to one side of the external Hz watch crystal. Leave unconnected if not using an external crystal.	I/O

Available Packages and Package Codes

Each Triscend A7S family member is available in different packaging options. The ordering code for an A7S device contains a single-character package code that defines the package style, as shown in Table 60.

Table 60. A7S Family Package Options.

Package Code	Leads	Style	Max. Width	Max. Length	Max. Height	Units
Q	208	PQFP	30.85	30.85	4.07	mm
G	324	BGA	19.20	19.20	1.80	mm

Footprint-Compatibility

Although the Triscend A7S device family and the Triscend E5 device family share some common packages, the A7S and E5 families are *not* pin compatible.

Available PIOs by Package

The number of user-configurable PIO pins depends on the base device type and the package in which it is packaged. Table 61 shows the available PIOs by package style. Two values are indicated for each device type, one for a 32-bit data, 24-bit address configuration and one for 8-bit data, 20-bit address—the minimum configuration. In the latter case, all spare data, address, and control lines are available as user-defined PIO pins.

Table 61. Available PIOs by Package

	Package Code	Q	G
	Leads	208	324
TA7S04	32 Data 24 Address	60	
	8 Data 20 Address	91	
TA7S20	32 Data 24 Address	86	158
	8 Data 20 Address	117	189

Package Power Dissipation/Thermal Characteristics

Package Thermal Resistance

Table 62 shows the junction-to-ambient thermal resistance (Θ_{JA}) for the various A7S packaging options as a function of airflow over the package. Systems without a fan can expect only minor airflow due to convection in the system. Fans help reduce the thermal resistance by carrying away heat.

Table 62. Thermal Resistance (Theta-JA) for A7S Package Options.

Package		Airflow				Units
Code	Leads	*0*	*0.5*	*1.0*	*2.0*	*m/sec*
Q	208	28.8	26.9	26.1	24.9	°C/W
G	324	18.0	TBD	16.5	15.3	°C/W

Estimates provided by packaging subcontractor.

Maximum Power Dissipation

Table 63 displays the maximum power dissipation allowed by an application operating at either 70° C ambient temperature for commercial-grade devices or 85° C for industrial-grade parts without exceeding the specified maximum operating junction temperature. This data is calculated using Table 62 according to Equation 5.

It is possible to exceed these values by operating at lower ambient temperatures or by improving heat dissipation with heat sinks or additional airflow.

$$P_{max} < \frac{T_J - T_A}{\Theta_{JA}} \qquad (5)$$

where:

P_{max} = the maximum amount of power that the package can dissipate without exceeding the recommended maximum operating junction temperature, T_J

T_J = the recommended maximum operating junction temperature
T_J = +85° C for commercial grade devices
T_J = +100° C for industrial grade devices

T_A = the maximum operating ambient temperature, assumptions shown below
T_A = +70° C for commercial grade devices
T_A = +85° C for industrial grade devices

Θ_{JA} = the thermal resistance of the package at the specified airflow

**Table 63. Maximum Power Dissipation for A7S Package Options
(operating at high ambient temperatures).**

Package		Airflow				Units
Code	Leads	*0*	*0.5*	*1.0*	*2.0*	*m/sec*
Q	208	520	555	575	600	mW
G	324	830	TBD	905	980	mW

208-pin PQFP Package Footprint Diagram
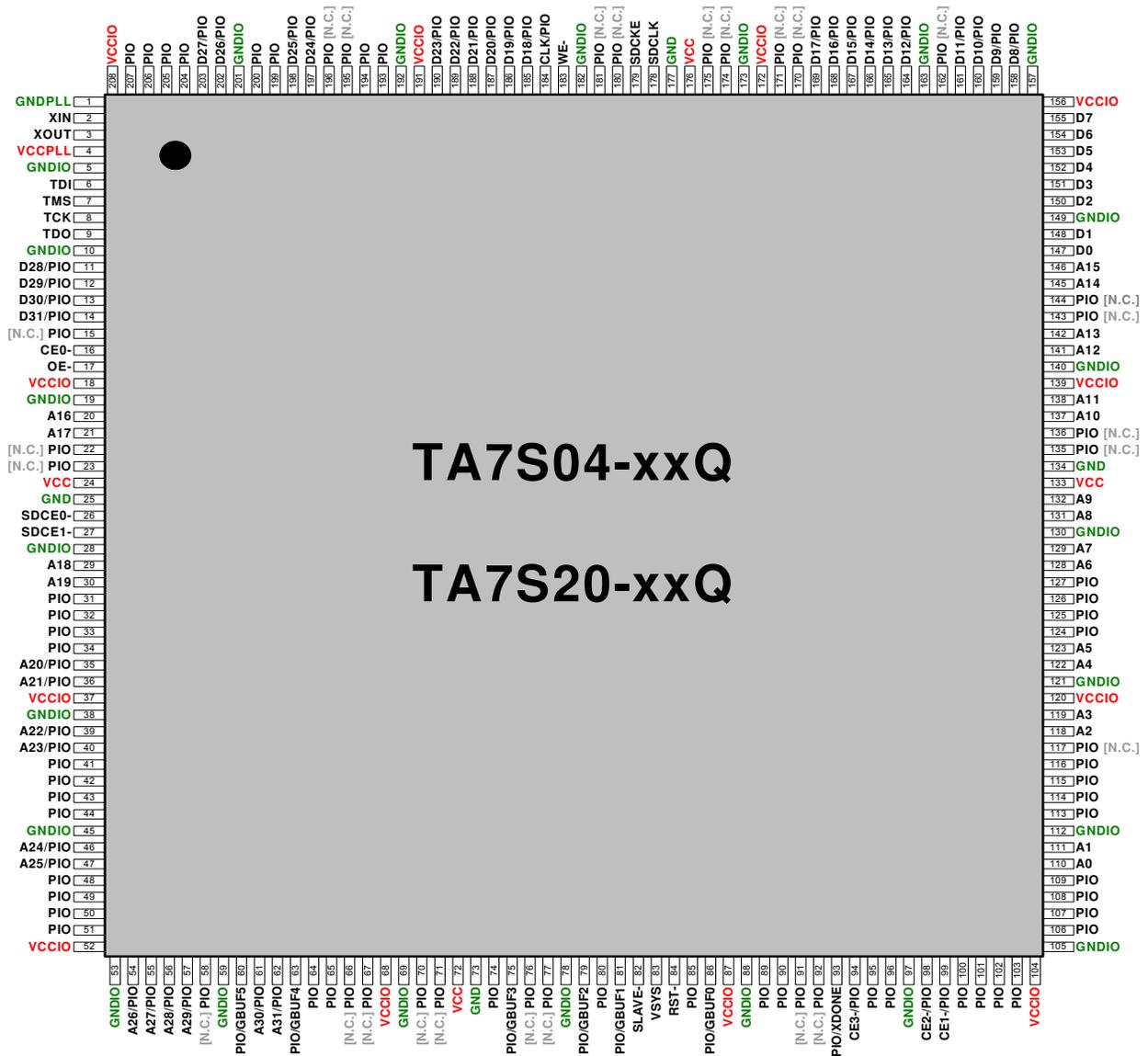
(Top View)

208-pin PQFP Package Pinout Tables

	(left edge)	
Pin	**TA7S04**	**TA7S20**
1	GNDPLL	GNDPLL
2	XIN	XIN
3	XOUT	XOUT
4	VCCPLL	VCCPLL
5	GNDIO	GNDIO
6	TDI	TDI
7	TMS	TMS
8	TCK	TCK
9	TDO	TDO
10	GNDIO	GNDIO
11	D28/PIO	D28/PIO
12	D29/PIO	D29/PIO
13	D30/PIO	D30/PIO
14	D31/PIO	D31/PIO
15	N.C.	PIO
16	CE0-	CE0-
17	OE-	OE-
18	VCCIO	VCCIO
19	GNDIO	GNDIO
20	A16	A16
21	A17	A17
22	N.C.	PIO
23	N.C.	PIO
24	VCC	VCC
25	GND	GND
26	SDCE0-	SDCE0-
27	SDCE1-	SDCE1-
28	GNDIO	GNDIO
29	A18	A18
30	A19	A19
31	PIO	PIO
32	PIO	PIO
33	PIO	PIO
34	PIO	PIO
35	A20/PIO	A20/PIO
36	A21/PIO	A21/PIO
37	VCCIO	VCCIO
38	GNDIO	GNDIO
39	A22/PIO	A22/PIO
40	A23/PIO	A23/PIO
41	PIO	PIO
42	PIO	PIO
43	PIO	PIO
44	PIO	PIO
45	GNDIO	GNDIO
46	A24/PIO	A24/PIO
47	A25/PIO	A25/PIO
48	PIO	PIO
49	PIO	PIO
50	PIO	PIO
51	PIO	PIO
52	VCCIO	VCCIO

	(bottom edge)	
Pin	**TA7S04**	**TA7S20**
53	GNDIO	GNDIO
54	A26/PIO	A26/PIO
55	A27/PIO	A27/PIO
56	A28/PIO	A28/PIO
57	A29/PIO	A29/PIO
58	N.C.	PIO
59	GNDIO	GNDIO
60	PIO/GBUF5	PIO/GBUF5
61	A30/PIO	A30/PIO
62	A31/PIO	A31/PIO
63	PIO/GBUF4	PIO/GBUF4
64	PIO	PIO
65	PIO	PIO
66	N.C.	PIO
67	N.C.	PIO
68	VCCIO	VCCIO
69	GNDIO	GNDIO
70	N.C.	PIO
71	N.C.	PIO
72	VCC	VCC
73	GND	GND
74	PIO	PIO
75	PIO/GBUF3	PIO/GBUF3
76	N.C.	PIO
77	N.C.	PIO
78	GNDIO	GNDIO
79	PIO/GBUF2	PIO/GBUF2
80	PIO	PIO
81	PIO/GBUF1	PIO/GBUF1
82	SLAVE-	SLAVE-
83	VSYS	VSYS
84	RST-	RST-
85	PIO	PIO
86	PIO/GBUF0	PIO/GBUF0
87	VCCIO	VCCIO
88	GNDIO	GNDIO
89	PIO	PIO
90	PIO	PIO
91	N.C.	PIO
92	N.C.	PIO
93	PIO/XDONE	PIO/XDONE
94	CE3-/PIO	CE3-/PIO
95	PIO	PIO
96	PIO	PIO
97	GNDIO	GNDIO
98	CE2-/PIO	CE2-/PIO
99	CE1-/PIO	CE1-/PIO
100	PIO	PIO
101	PIO	PIO
102	PIO	PIO
103	PIO	PIO
104	VCCIO	VCCIO

	(right edge)	
Pin	**TA7S04**	**TA7S20**
105	GNDIO	GNDIO
106	PIO	PIO
107	PIO	PIO
108	PIO	PIO
109	PIO	PIO
110	A0	A0
111	A1	A1
112	GNDIO	GNDIO
113	PIO	PIO
114	PIO	PIO
115	PIO	PIO
116	PIO	PIO
117	N.C.	PIO
118	A2	A2
119	A3	A3
120	VCCIO	VCCIO
121	GNDIO	GNDIO
122	A4	A4
123	A5	A5
124	PIO	PIO
125	PIO	PIO
126	PIO	PIO
127	PIO	PIO
128	A6	A6
129	A7	A7
130	GNDIO	GNDIO
131	A8	A8
132	A9	A9
133	VCC	VCC
134	GND	GND
135	N.C.	PIO
136	N.C.	PIO
137	A10	A10
138	A11	A11
139	VCCIO	VCCIO
140	GNDIO	GNDIO
141	A12	A12
142	A13	A13
143	N.C.	PIO
144	N.C.	PIO
145	A14	A14
146	A15	A15
147	D0	D0
148	D1	D1
149	GNDIO	GNDIO
150	D2	D2
151	D3	D3
152	D4	D4
153	D5	D5
154	D6	D6
155	D7	D7
156	VCCIO	VCCIO

(top edge)

Pin	TA7S04	TA7S20
157	GNDIO	GNDIO
158	D8/PIO	D8/PIO
159	D9/PIO	D9/PIO
160	D10/PIO	D10/PIO
161	D11/PIO	D11/PIO
162	N.C.	PIO
163	GNDIO	GNDIO
164	D12/PIO	D12/PIO
165	D13/PIO	D13/PIO
166	D14/PIO	D14/PIO
167	D15/PIO	D15/PIO
168	D16/PIO	D16/PIO
169	D17/PIO	D17/PIO
170	N.C.	PIO
171	N.C.	PIO
172	VCCIO	VCCIO
173	GNDIO	GNDIO
174	N.C.	PIO
175	N.C.	PIO
176	VCC	VCC
177	GND	GND
178	SDCLK	SDCLK
179	SDCKE	SDCKE
180	N.C.	PIO
181	N.C.	PIO
182	GNDIO	GNDIO
183	WE-	WE-
184	CLK/PIO	CLK/PIO
185	D18/PIO	D18/PIO
186	D19/PIO	D19/PIO
187	D20/PIO	D20/PIO
188	D21/PIO	D21/PIO
189	D22/PIO	D22/PIO
190	D23/PIO	D23/PIO
191	VCCIO	VCCIO
192	GNDIO	GNDIO
193	PIO	PIO
194	PIO	PIO
195	N.C.	PIO
196	N.C.	PIO
197	D24/PIO	D24/PIO
198	D25/PIO	D25/PIO
199	PIO	PIO
200	PIO	PIO
201	GNDIO	GNDIO
202	D26/PIO	D26/PIO
203	D27/PIO	D27/PIO
204	PIO	PIO
205	PIO	PIO
206	PIO	PIO
207	PIO	PIO
208	VCCIO	VCCIO

208-pin PQFP Pins by Type

32 Data, 24 Address Configuration

Type	TA7S04Q	TA7S20Q
PIO*	60	86
VCC	4	4
VCCIO	12	12
VCCPLL	1	1
GND	4	4
GNDIO	24	24
GNDPLL	1	1
D[31:0]	32	32
A[31:0]	24	24
JTAG	4	4
N.C.	26	0
Others/ Control	16	16

8 Data, 20 Address Configuration

Type	TA7S04Q	TA7S20Q
PIO*	91	117
VCC	4	4
VCCIO	12	12
VCCPLL	1	1
GND	4	4
GNDIO	24	24
GNDPLL	1	1
D[7:0]	8	8
A[19:0]	20	20
JTAG	4	4
N.C.	26	0
Others/ Control	13	13

PIO* includes pure PIO pins and those with alternate functions.

208-pin PQFP Package Mechanical Drawing
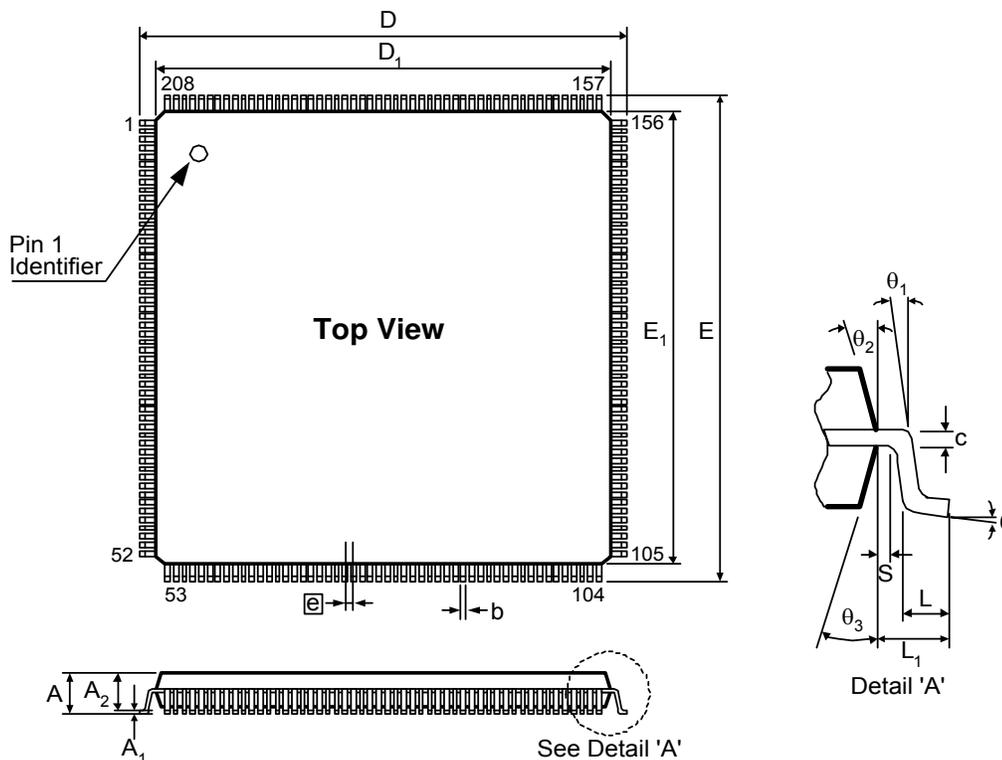
	Millimeters		
Symbol	**Min.**	**Nom.**	**Max.**
A	3.92	—	4.07
A_1	0.25	—	—
A_2	3.15	3.23	3.30
b	0.18	—	0.28
c	0.13	—	0.23
D	30.35	30.60	30.85
D_1	27.90	28.00	28.10
E	30.35	30.60	30.85
E_1	27.90	28.00	28.10
e	0.50 BSC		
L	0.35	0.50	0.65
L_1	1.30 REF		
S	0.20	—	—
θ	0º	—	7º
$θ_1$	0º	—	—
$θ_2$	12º TYP		
$θ_3$	12º TYP		

1. All dimensions and tolerances conform to ASME Y14.5M-1994.
2. All dimensions are in millimeters.
3. Dimensions D1 and E1 do not include mold protrusion. Allowable mold protrusion shall not exceed 0.25 mm per side.
4. Package top dimensions may be smaller than bottom dimensions.

JEDEC Reference Drawing:	MS-029A-FA-1
JEDEC File Location:	http://www.jedec.org/DOWNLOAD/pub95/ms-029a.pdf
Land Pattern Dimensions:	See 208-pin PQFP Package Land Pattern Dimensions
Assembly Vendor Drawing:	SPIL, Q208-SW2-C

208-pin PQFP Package Land Pattern Dimensions
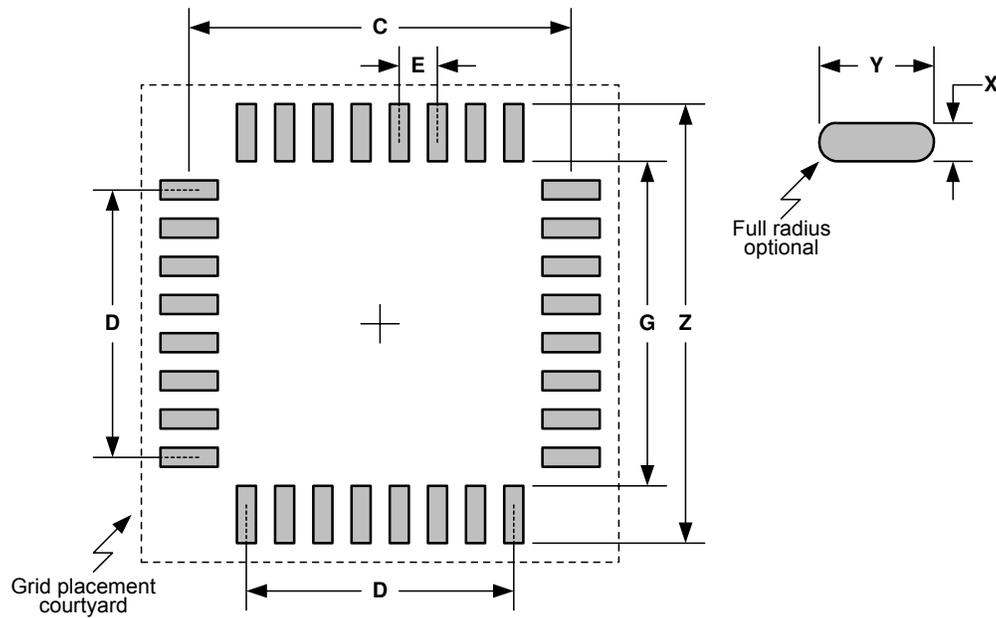
Symbol	208-pinPQFP (Code=Q)			Units
	Min	Ref	Max	
Z	—	—	30.8	mm
G	27.6	—	—	
X	—	—	0.3	
Y	—	1.6	—	
C	—	29.2	—	
D	—	25.5	—	
E	—	0.5	—	

SUBJECT TO CHANGE

324-ball BGA Package Footprint Diagram

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A	GND PLL	PIO	PIO	D22/PIO	D21/PIO	D20/PIO	D19/PIO	WE-	SD CKE	SD CLK	D17/PIO	D15/PIO	D13/PIO	D11/PIO	D10/PIO	D9/PIO	D8/PIO	D5
B	XIN	D25/PIO	PIO	D27/PIO	D26/PIO	D24/PIO	D23/PIO	D18/PIO	CLK/PIO	PIO	D16/PIO	D14/PIO	D12/PIO	PIO	PIO	PIO	D7	D3
C	XOUT	TDI	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	D6	D2
D	VCC PLL	TMS	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	D4	D1
E	GNDIO	TCK	PIO	PIO	VCCIO	VCCIO	VCCIO	PIO	PIO	PIO	PIO	VCCIO	VCCIO	VCCIO	PIO	PIO	D0	A14
F	D28/PIO	TDO	PIO	PIO	VCCIO	VCCIO	GNDIO	GNDIO	VCC	GND	GNDIO	GNDIO	VCCIO	VCCIO	PIO	PIO	A15	PIO
G	D30/PIO	D29/PIO	PIO	PIO	VCCIO	GNDIO	GNDIO	GNDIO	VCC	GND	GNDIO	GNDIO	GNDIO	VCCIO	PIO	PIO	PIO	PIO
H	CE0-	D31/PIO	PIO	PIO	PIO	GNDIO	GNDIO	GNDIO	GNDIO	GNDIO	GNDIO	GNDIO	GNDIO	PIO	PIO	PIO	PIO	A12
J	A16	OE-	PIO	PIO	PIO	VCC	VCC	GNDIO	GNDIO	GNDIO	GNDIO	GND	GND	PIO	PIO	PIO	A8	A10
K	A17	PIO	PIO	PIO	PIO	GND	GND	GNDIO	GNDIO	GNDIO	GNDIO	VCC	VCC	PIO	PIO	PIO	A6	A13
L	SD CE0-	PIO	PIO	PIO	PIO	GNDIO	GNDIO	GNDIO	GNDIO	GNDIO	GNDIO	GNDIO	GNDIO	PIO	PIO	PIO	PIO	A11
M	SD CE1-	PIO	PIO	PIO	VCCIO	GNDIO	GNDIO	GNDIO	VCC	GND	GNDIO	GNDIO	GNDIO	VCCIO	PIO	PIO	PIO	A9
N	A18	A20/PIO	PIO	PIO	VCCIO	VCCIO	GNDIO	GNDIO	VCC	GND	GNDIO	GNDIO	VCCIO	VCCIO	PIO	PIO	A4	A7
P	PIO	A19	PIO	PIO	VCCIO	VCCIO	VCCIO	PIO	PIO	PIO	PIO	VCCIO	VCCIO	VCCIO	PIO	PIO	PIO	A5
R	A22/PIO	A21/PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	A3
T	A23/PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	A2
U	A24/PIO	A26/PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO	PIO/XDONE	CE2-/PIO	A1
V	A25/PIO	A27/PIO	A28/PIO	A29/PIO	PIO/GBUF5	A30/PIO	A31/PIO	PIO/GBUF4	PIO/GBUF3	PIO/GBUF2	PIO/GBUF1	SLAVE-	VSYS	RST-	PIO/GBUF0	CE3-/PIO	CE1-/PIO	A0

(Top view, through top of package to solder balls underneath)

324-ball BGA Package Pinout Tables

Ball	TA7S20G	Ball	TA7S20G	Ball	TA7S20G	Ball	TA7S20G	Ball	TA7S20G
A1	GNDPLL	C15	PIO	F11	GNDIO	J7	VCC	M3	PIO
A2	PIO	C16	PIO	F12	GNDIO	J8	GNDIO	M4	PIO
A3	PIO	C17	D6	F13	VCCIO	J9	GNDIO	M5	VCCIO
A4	D22/PIO	C18	D2	F14	VCCIO	J10	GNDIO	M6	GNDIO
A5	D21/PIO	D1	VCCPLL	F15	PIO	J11	GNDIO	M7	GNDIO
A6	D20/PIO	D2	TMS	F16	PIO	J12	GND	M8	GNDIO
A7	D19/PIO	D3	PIO	F17	A15	J13	GND	M9	VCC
A8	WE-	D4	PIO	F18	PIO	J14	PIO	M10	GND
A9	SDCKE	D5	PIO	G1	D30/PIO	J15	PIO	M11	GNDIO
A10	SDCLK	D6	PIO	G2	D29/PIO	J16	PIO	M12	GNDIO
A11	D17/PIO	D7	PIO	G3	PIO	J17	A8	M13	GNDIO
A12	D15/PIO	D8	PIO	G4	PIO	J18	A10	M14	VCCIO
A13	D13/PIO	D9	PIO	G5	VCCIO	K1	A17	M15	PIO
A14	D11/PIO	D10	PIO	G6	GNDIO	K2	PIO	M16	PIO
A15	D10/PIO	D11	PIO	G7	GNDIO	K3	PIO	M17	PIO
A16	D9/PIO	D12	PIO	G8	GNDIO	K4	PIO	M18	A9
A17	D8/PIO	D13	PIO	G9	VCC	K5	PIO	N1	A18
A18	D5	D14	PIO	G10	GND	K6	GND	N2	A20/PIO
B1	XIN	D15	PIO	G11	GNDIO	K7	GND	N3	PIO
B2	D25/PIO	D16	PIO	G12	GNDIO	K8	GNDIO	N4	PIO
B3	PIO	D17	D4	G13	GNDIO	K9	GNDIO	N5	VCCIO
B4	D27/PIO	D18	D1	G14	VCCIO	K10	GNDIO	N6	VCCIO
B5	D26/PIO	E1	GNDIO	G15	PIO	K11	GNDIO	N7	GNDIO
B6	D24/PIO	E2	TCK	G16	PIO	K12	VCC	N8	GNDIO
B7	D23/PIO	E3	PIO	G17	PIO	K13	VCC	N9	VCC
B8	D18/PIO	E4	PIO	G18	PIO	K14	PIO	N10	GND
B9	CLK/PIO	E5	VCCIO	H1	CE0-	K15	PIO	N11	GNDIO
B10	PIO	E6	VCCIO	H2	D31/PIO	K16	PIO	N12	GNDIO
B11	D16/PIO	E7	VCCIO	H3	PIO	K17	A6	N13	VCCIO
B12	D14/PIO	E8	PIO	H4	PIO	K18	A13	N14	VCCIO
B13	D12/PIO	E9	PIO	H5	PIO	L1	SDCE0-	N15	PIO
B14	PIO	E10	PIO	H6	GNDIO	L2	PIO	N16	PIO
B15	PIO	E11	PIO	H7	GNDIO	L3	PIO	N17	A4
B16	PIO	E12	VCCIO	H8	GNDIO	L4	PIO	N18	A7
B17	D7	E13	VCCIO	H9	GNDIO	L5	PIO	P1	PIO
B18	D3	E14	VCCIO	H10	GNDIO	L6	GNDIO	P2	A19
C1	XOUT	E15	PIO	H11	GNDIO	L7	GNDIO	P3	PIO
C2	TDI	E16	PIO	H12	GNDIO	L8	GNDIO	P4	PIO
C3	PIO	E17	D0	H13	GNDIO	L9	GNDIO	P5	VCCIO
C4	PIO	E18	A14	H14	PIO	L10	GNDIO	P6	VCCIO
C5	PIO	F1	D28/PIO	H15	PIO	L11	GNDIO	P7	VCCIO
C6	PIO	F2	TD0	H16	PIO	L12	GNDIO	P8	PIO
C7	PIO	F3	PIO	H17	PIO	L13	GNDIO	P9	PIO
C8	PIO	F4	PIO	H18	A12	L14	PIO	P10	PIO
C9	PIO	F5	VCCIO	J1	A16	L15	PIO	P11	PIO
C10	PIO	F6	VCCIO	J2	OE-	L16	PIO	P12	VCCIO
C11	PIO	F7	GNDIO	J3	PIO	L17	PIO	P13	VCCIO
C12	PIO	F8	GNDIO	J4	PIO	L18	A11	P14	VCCIO
C13	PIO	F9	VCC	J5	PIO	M1	SDCE1-	P15	PIO
C14	PIO	F10	GND	J6	VCC	M2	PIO	P16	PIO

Ball	TA7S20G	Ball	TA7S20G	Ball	TA7S20G	Ball	TA7S20G	Ball	TA7S20G
P17	PIO	R15	PIO	T13	PIO	U11	PIO	V9	PIO/GBUF3
P18	A5	R16	PIO	T14	PIO	U12	PIO	V10	PIO/GBUF2
R1	A22/PIO	R17	PIO	T15	PIO	U13	PIO	V11	PIO/GBUF1
R2	A21/PIO	R18	A3	T16	PIO	U14	PIO	V12	SLAVE-
R3	PIO	T1	A23/PIO	T17	PIO	U15	PIO	V13	VSYS
R4	PIO	T2	PIO	T18	A2	U16	PIO/XDONE	V14	RST-
R5	PIO	T3	PIO	U1	A24/PIO	U17	CE2-/PIO	V15	PIO/GBUF0
R6	PIO	T4	PIO	U2	A26/PIO	U18	A1	V16	CE3-/PIO
R7	PIO	T5	PIO	U3	PIO	V1	A25/PIO	V17	CE1-/PIO
R8	PIO	T6	PIO	U4	PIO	V2	A27/PIO	V18	A0
R9	PIO	T7	PIO	U5	PIO	V3	A28/PIO		
R10	PIO	T8	PIO	U6	PIO	V4	A29/PIO		
R11	PIO	T9	PIO	U7	PIO	V5	PIO/GBUF5		
R12	PIO	T10	PIO	U8	PIO	V6	A30/PIO		
R13	PIO	T11	PIO	U9	PIO	V7	A31/PIO		
R14	PIO	T12	PIO	U10	PIO	V8	PIO/GBUF4		

324-ball BGA Package Pins by Type

32 Data, 24 Address Configuration

Type	TA7S20G
PIO*	158
VCC	8
VCCIO	24
VCCPLL	1
GND	8
GNDIO	45
GNDPLL	1
D[31:0]	32
A[23:0]	24
JTAG	4
N.C.	0
Others/Control	16

8 Data, 20 Address Configuration

Type	TA7S20G
PIO*	189
VCC	8
VCCIO	24
VCCPLL	1
GND	8
GNDIO	45
GNDPLL	1
D[7:0]	8
A[19:0]	20
JTAG	4
N.C.	0
Others/Control	13

PIO* includes pure PIO pins and those with alternate functions.

![Triscend logo]

324-ball BGA Package Mechanical Drawing

Top View

Side View

See detail 'A'

Detail 'A'

Bottom View

See detail 'B'

Detail 'B'
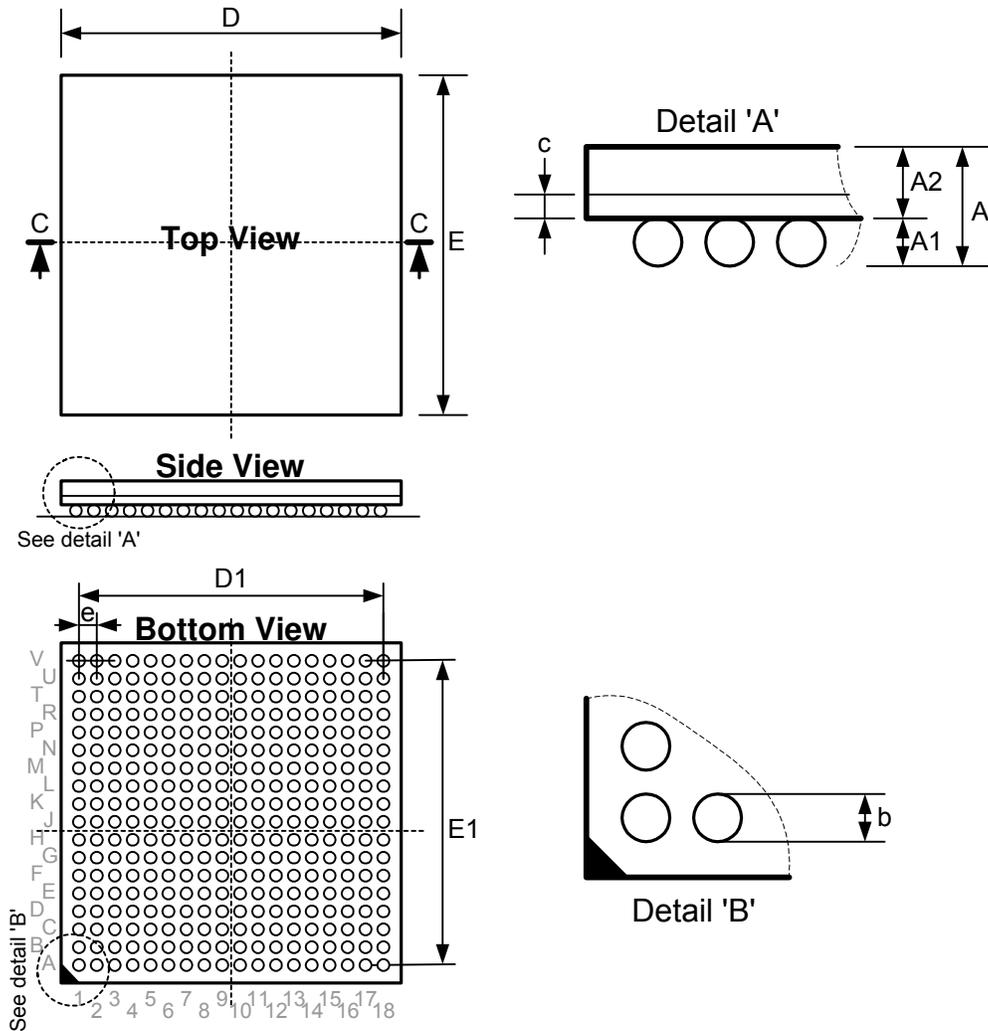
Symbol	Millimeters		
	Min.	Nom.	Max.
A	—	—	1.80
A1	0.40	0.50	0.60
A2	1.00	1.06	1.12
b	0.50	0.60	0.70
c	—	0.36	—
D	18.80	19.00	19.20
D1	—	17.00	—
E	18.80	19.00	19.20
E1	—	17.00	—
e	—	1.00	—

1. All dimensions and tolerances conform to ASME Y14.5M-1994.

2. All dimensions are in millimeters.

3. Dimension 'b' is measured at the maximum solder ball diameter, parallel to the plane of the package body.

4. There shall be a minimum clearance of 0.25 mm between the edge of the solder ball and body edge.

JEDEC Reference Drawing:	MO-205
JEDEC File Location:	http://www.jedec.org/download/search/MO-205d.pdf
Assembly Vendor Drawing:	SPIL, TF324-SW3

SUBJECT TO CHANGE

Electrical and Timing Characteristics

Absolute Maximum Ratings

Symbol	Parameter	Min	Max	Units
V_{CC}	Core logic supply voltage relative to GND	-0.5	3.0	V
V_{CCIO}	PIO supply voltage relative to GND	-0.5	4.0	V
V_{IN}	Input voltage relative to GND [1]	-0.5	4.0	V
V_{TS}	Voltage applied to three-state output [1]	-0.5	4.0	V
T_{STG}	Storage temperature (ambient)	-60	+150	° C
T_{SOL}	Maximum soldering temperature (10 sec at 1/16 in. = 1.5 mm)		+260	° C
T_J	Junction temperature, plastic packages		+125	° C

Recommended Operating Conditions/DC Characteristics

Symbol	Parameter	Min	Max	Units
V_{CC}	Core logic supply voltage relative to GND	2.3	2.7	V
V_{CCIO}	PIO supply voltage relative to GND	2.3	3.6	V
T_J	Operating junction temperature, commercial [3]	0	+85	° C
	Operating junction temperature, industrial [3]	-40	+100	° C
V_{CCR}	Longest supply voltage rise time from 1V to 3V [4]		200	ms
V_{IL}	Input Low voltage	0	30% V_{CC}	V
V_{IH}	Input High voltage	50% V_{CC}	V_{CC}+0.5	V
V_S	Schmitt Hysteresis, hysteresis mode	5% VCC	20% VCC	V
V_{ESD}	Electro-static discharge protection, human body model	2 000		V
T_{IN}	Input signal transition time		250	ns
V_{OL}	Output Low voltage, I_{OL} = -16 mA, V_{CC} min (TTL) [5]		0.4	V
	Output Low voltage, I_{OL} = -1.5 mA, V_{CC} min (LVCMOS)		10% VCC	V
V_{OH}	Output High voltage, I_{OH} = 8 mA, V_{CC} min (TTL) [5]	2.4		V
	Output High voltage, I_{OH} = 0.5 mA, V_{CC} min (LVCMOS)	90% VCC		V
V_{DR}	Data retention supply voltage of internal core logic (below which initialization data may be lost)	2.5		V
V_{DRIO}	Data retention supply voltage of PIO (below which initialization data may be lost)	2.1		V
I_{OL}	Output Low current, output in highest drive strength mode (TTL), V_{OL} max, V_{CC} min		-16.0	mA
	Output Low current, output in lowest current drive mode (TTL), V_{OL} max, V_{CC} min		-4.0	mA
	Output Low current (LVCMOS), V_{OL} max, V_{CC} min		-1.5	mA
I_{OH}	Output High current, output in highest drive strength mode (TTL), V_{OH} min, V_{CC} min		+8.0	mA
	Output High current, output in lowest drive strength mode (TTL), V_{OH} min, V_{CC} min		+2.0	mA
	Output High current (LVCMOS), V_{OH} min, V_{CC} min		+0.5	mA
I_{IL}	Input leakage current	-10	+10	µA
C_{IO}	Pin capacitance [7]		10	pF
L_{IO}	Pin inductance [7]		20	nH

Note 1: Maximum DC overshoot above VCC or undershoot below GND must be limited to either 0.5 V or 10 mA, whichever is easier to achieve. During transitions, device pins may undershoot to -2.0 V or overshoot to +5.0 V, provided this condition lasts less than 20 ns and with less than 100 mA forcing current.

Note 2: Stresses beyond those listed under Absolute Maximum Ratings may cause permanent device damage. The values listed are stress ratings only. Functional operation of the device at these or any conditions exceeding those listed under Recommended Operating Conditions is not implied. Exposure to Absolute Maximum Ratings conditions for extended periods may affect device reliability.

Note 3: Typical ambient operating conditions are between 0 to +70° C for commercial devices and -40 to +85° C for industrial devices, depending on the application's power consumption, package style, and airflow.

Note 4: Ramp rate can be extended by asserting RST- until V_{CC} reaches the minimum specified value.

Note 5: Output in highest drive strength mode.

Note 6: Continuous static loads must fall within the I_{OH}, I_{OL} limits in this section.

Note 7: Capacitance and inductance is sample-tested only.

All Triscend devices are 100% functionally tested. These parameters are modeled after the testing methods described by MIL-M-38510/605. The values listed below are ***representative, guideline values*** extracted from measured internal test patterns. Actual values may depend on application-specific use. ***The FastChip development system reports specific, worst-case guaranteed values in the Timing Analysis section of the project report.***

All timing values shown assume worst-case operating conditions, including process technology, power supply voltage, and junction temperature.

Product Status Definitions

These specifications include a status designation as defined below.

Preview:	Initial estimates based on simulation or extrapolated data from other speed grades, devices, families, or process technologies. These values are subject to change without notice. These values are estimates and should not be used for production.
Preliminary:	Based on preliminary or partial device characterization. Though further changes are not expected, these values are subject to change without notice. These values are safe for producing prototypes but should not be used for high-volume production.
Final or Unmarked:	Specifications are final. These specifications may be used for volume production. Values cannot change without notice.
Guideline:	A worst-case value or a range of worst-case values based on example applications under a variety of conditions. Actual worst-case values are reported for the specific use within an application by the FastChip development system in the Timing Analysis section of the project report. The value reported by FastChip may not match the values shown herein. The value reported by FastChip supercedes any value shown below.

General A7S Timing Characteristics

			Preview			
		Speed Grade	All	-33	-60	
Description	Symbol	Device	Min	Max	Max	Units
Bus Clock frequency	F_{BCLK}	All	0	33.0	60.0	MHz
Bus Clock high time	T_{BCH}	All				ns
Bus Clock low time	T_{BCL}	All				ns
Bus Clock rise time	T_{BCRT}	All		5.0	5.0	ns
Bus Clock fall time	T_{BCFT}	All		5.0	5.0	ns
32.786 kHz crystal start-up time	T_{32STU}	All		TBD	TBD	ms
Internal ring oscillator frequency	F_{ROSC}	All	15.0	30.0	30.0	MHz
RST- pulse width	T_{RSTW}	All	TBD			ns
Power-on reset delay after power valid	T_{RSTW}	All	0.5	1.1	1.1	µs
RST- de-asserted to first access to external secondary initialization memory [1]	T_{RSTF}	All	1.0	3.0	3.0	ms
			Preview			

Note 2: VSYS=High, SLAVE-=Low. Variation is due to differences in internal ring oscillator frequencies between devices.

JTAG Interface Timing Characteristics
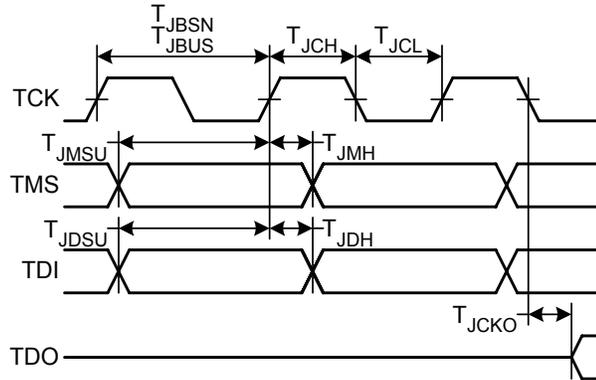
Figure 78. JTAG timing diagram.

JTAG Timing Characteristics

Description	Symbol	Fig.	Device	Preview -33 Min	Preview -33 Max	Preview -60 Min	Preview -60 Max	Units
TCK frequency, boundary scan	F_{JBSN}		All	0	10.0	0	10.0	MHz
TCK boundary scan cycle time	T_{JBSN}		All	100	∞	100	∞	ns
TCK frequency, bus access	F_{JBUS}		All	0	33.0	0	60.0	MHz
TCK bus access cycle time	T_{JBUS}		All	30.3	∞	16.7	∞	ns
TCK High time	T_{JCH}		All	11.0		11.0		ns
TCK Low time	T_{JCL}		All	11.0		11.0		ns
TCK rise time	T_{JCRT}		All		3.0		3.0	ns
TCK fall time	T_{JCFT}		All		3.0		3.0	ns
TDI setup time before TCK rising edge	T_{JDSU}		All	5.0		5.0		ns
TDI hold time after TCK rising edge	T_{JDH}		All	2.0		2.0		ns
TMS setup time before TCK rising edge	T_{JMSU}		All	5.0		5.0		ns
TMS hold time after TCK rising edge	T_{JMH}		All	2.0		2.0		ns
TDO valid after TCK falling edge	T_{JCKO}		All	1.0		1.0		ns
				Preview		*Preview*		

SDRAM Interface Timing Characteristics

Example SDRAM Memory Interface Waveforms
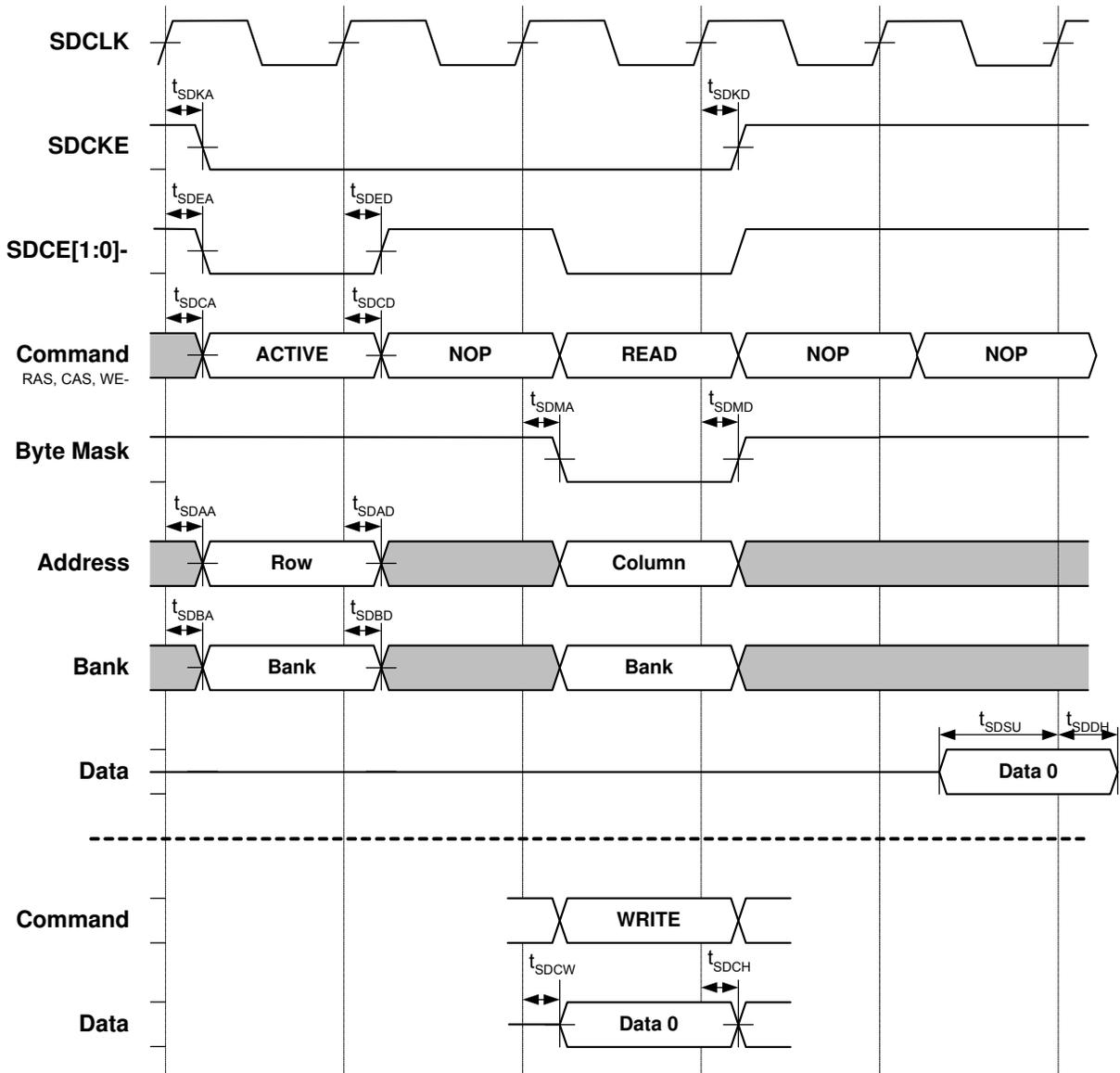
Figure 79.   Stylized SDRAM timing waveform showing typical Read and Write transfers. The specific SDRAM interface pins involved depend on the MSSIU configuration. See Figure 38, Figure 39, and Figure 40.

Power Consumption Characteristics

The A7S CSoC family devices are currently undergoing full characterization. The following data represents estimated power consumption based on limited bench testing.

Typical Dynamic Current Consumption Estimates

Table 65 shows the estimated range of dynamic current consumed by a typical A7S application. Actual dynamic power consumption depends on the operating frequency, cache usage, and the style of design implemented in the configurable system logic (CSL). The following values represent data measured from a range of typical 33 and 60 MHz applications.

Table 65. Typical A7S Dynamic Current Consumption.

Device	Dynamic Current Consumption	Units
TA7S20	100 to 500	mA

Typical Power-Down Mode Current Consumption Estimates

The data in Table 66 represents the estimated typical current consumed by an A7S device while in power-down mode, with all power-down options selected (i.e., the lowest power mode). The data assumes that VCC and VCCIO are both 2.5 volts, and the device operating at room temperature.

Table 66. Typical A7S Power-Down Current Consumption.

Device	Power-Down Current	Units
TA7S20	100	µA

APPENDIX A: A7S Memory Map

Absolute Address	Symbolic Address	Register Name	Access
0x1000_0000 to 0x100F_FFFF	Various, defined by user with FastChip	User-defined, memory-mapped functions implemented in CSL matrix	User-defined
0xC000_0000	**EXTERNAL_SDRAM_BASE**	**External SDRAM**	
0xC000_0000 to 0xCFFF_FFFF		External SDRAM	Read/Write
0xD000_0000	**EXTERNAL_FLASH_BASE**	**External Flash**	
0xD000_0000 to 0xD0FF_FFFF		External Flash	Read/Write
0xD100_0000 to 0xD100_FFFF		Primary Initialization Boot ROM (Reserved)	Read
0xD101_0000	**MSS_BASE**	**External Memory Interface**	
0xD101_0000	MSS_BASE + 0x00	MSS_CONFIG_REG	Read/Write
0xD101_0004	MSS_BASE + 0x04	MSS_TIM_CTRL_REG	Read/Write
0xD101_0008	MSS_BASE + 0x08	MSS_SDR_MODE_REG	Read/Write
0xD101_000C	MSS_BASE + 0x0C	MSS_SDR_CTRL_REG	Read/Write
0xD101_0010	MSS_BASE + 0x10	MSS_STATUS_REG	Read
0xD101_0014	MSS_BASE + 0x14	MSS_STATUS_CLEAR_REG	Write
0xD101_0100	**SYS_BASE**	**Clock, Reset, Power Management**	
0xD101_0100	SYS_BASE + 0x00	SYS_CLOCK_CONTROL_REG	Read/Write
0xD101_0104	SYS_BASE + 0x04	SYS_PLL_STATUS_REG	Read
0xD101_0108	SYS_BASE + 0x08	SYS_PLL_STATUS_CLEAR_REG	Write
0xD101_010C	SYS_BASE + 0x0C	SYS_RESET_CONTROL_REG	Read/Write
0xD101_0110	SYS_BASE + 0x10	SYS_POWER_CONTROL_REG	Read/Write
0xD101_0200	**INT_BASE**	**Interrupt Controller**	
0xD101_0200	INT_BASE + 0x00	INT_IRQ_STATUS_REG	Read
0xD101_0204	INT_BASE + 0x04	INT_IRQ_RAW_STATUS_REG	Read
0xD101_0208	INT_BASE + 0x08	INT_IRQ_ENABLE_REG	Read/Write
0xD101_020C	INT_BASE + 0x0C	INT_IRQ_ENABLE_CLEAR_REG	Write
0xD101_0210	INT_BASE + 0x10	INT_IRQ_SOFT_REG	Write
0xD101_0300	INT_BASE + 0x100	INT_FIQ_STATUS_REG	Read
0xD101_0304	INT_BASE + 0x104	INT_FIQ_RAW_STATUS_REG	Read
0xD101_0308	INT_BASE + 0x108	INT_FIQ_ENABLE_REG	Read/Write
0xD101_030C	INT_BASE + 0x10C	INT_FIQ_ENABLE_CLEAR_REG	Write
0xD101_0310	INT_BASE + 0x110	INT_IRQ_STEER_REG	Read/Write
0xD101_0400	**REMAP_BASE**	**Pause and Remap Registers**	
0xD101_0400	REMAP_BASE + 0x00	REMAP_PAUSE_REG	Write
0xD101_0410	REMAP_BASE + 0x10	REMAP_IDENTIFICATION_REG	Read
0xD101_0414	REMAP_BASE + 0x14	REMAP_REVISION_REG	Read
0xD101_0420	REMAP_BASE + 0x20	REMAP_CLEAR_RESET_MAP_REG	Write
0xD101_0430	REMAP_BASE + 0x30	REMAP_RESET_STATUS_REG	Read
0xD101_0434	REMAP_BASE + 0x34	REMAP_RESET_STATUS_CLEAR_REG	Write
0xD101_0438	REMAP_BASE + 0x38	REMAP_PIN_STATUS_REG	Read
0xD101_043C	REMAP_BASE + 0x3C	REMAP_PIN_STATUS_CLEAR_REG	Write
0xD101_0440	REMAP_BASE + 0x40	REMAP_ALIAS_ENABLE_REG	Read/Write
0xD101_0444	REMAP_BASE + 0x44	REMAP_SRAM_CONFIG_REG	Read/Write
0xD101_0448	REMAP_BASE + 0x48	REMAP_SRAM_BASE_ADR_REG	Read/Write
0xD101_044C	REMAP_BASE + 0x4C	REMAP_ACC_PROTECT_REG	Read/Write

Absolute Address	Symbolic Address	Register Name	Access
0xD101_0500	**TIMER_BASE**	**Dedicated 16-bit Timers**	
0xD101_0500	TIMER_BASE + 0x00	TIMER0_LOAD_REG	Read/Write
0xD101_0504	TIMER_BASE + 0x04	TIMER0_VALUE_REG	Read
0xD101_0508	TIMER_BASE + 0x08	TIMER0_CONTROL_REG	Read/Write
0xD101_050C	TIMER_BASE + 0x0C	TIMER0_CLEAR_REG	Write
0xD101_0520	TIMER_BASE + 0x20	TIMER1_LOAD_REG	Read/Write
0xD101_0524	TIMER_BASE + 0x24	TIMER1_VALUE_REG	Read
0xD101_0528	TIMER_BASE + 0x28	TIMER1_CONTROL_REG	Read/Write
0xD101_052C	TIMER_BASE + 0x2C	TIMER1_CLEAR_REG	Write
0xD101_0600	**WD_BASE**	**Watchdog Timer**	
0xD101_0600	WD_BASE + 0x00	WATCHDOG_CONTROL_REG	Read/Write
0xD101_0604	WD_BASE + 0x04	WATCHDOG_TIMEOUT_VAL_REG	Read/Write
0xD101_0608	WD_BASE + 0x08	WATCHDOG_CURRENT_VAL_REG	Read
0xD101_060C	WD_BASE + 0x0C	WATCHDOG_CLEAR_REG	Write
0xD101_0700	**CFG_BASE**	**Configuration Control**	
0xD101_0700 to 0XD101_070C		Configuration Control	Read/Write
0xD101_0800	**DMA_BASE**	**DMA Controller**	
0xD101_0800	DMA_BASE + 0x00	DMA0_CONTROL_REG	Read/Write
0xD101_0804	DMA_BASE + 0x04	DMA0_INT_ENABLE_REG	Read/Write
0xD101_0808	DMA_BASE + 0x08	DMA0_INT_REG	Read
0xD101_080C	DMA_BASE + 0x0C	DMA0_INT_CLEAR_REG	Write
0xD101_0810	DMA_BASE + 0x10	DMA0_DES_TABLE_ADDR_REG	Read/Write
0xD101_0814	DMA_BASE + 0x14	DMA0_SRC_ADDR_REG	Read/Write
0xD101_0818	DMA_BASE + 0x18	DMA0_DST_ADDR_REG	Read/Write
0xD101_081C	DMA_BASE + 0x1C	DMA0_TRANS_CNT_REG	Read/Write
0xD101_0820	DMA_BASE + 0x20	DMA0_CUR_DESC_ADDR_REG	Read
0xD101_0824	DMA_BASE + 0x24	DMA0_CUR_SRC_ADDR_REG	Read
0xD101_0828	DMA_BASE + 0x28	DMA0_CUR_DEST_ADDR_REG	Read
0xD101_082C	DMA_BASE + 0x2C	DMA0_CUR_TRANS_CNT_REG	Read
0xD101_0830	DMA_BASE + 0x30	DMA0_PEND_REQ_REG	Read
0xD101_0840	DMA_BASE + 0x40	DMA1_CONTROL_REG	Read/Write
0xD101_0844	DMA_BASE + 0x44	DMA1_INT_ENABLE_REG	Read/Write
0xD101_0848	DMA_BASE + 0x48	DMA1_INT_REG	Read
0xD101_084C	DMA_BASE + 0x4C	DMA1_INT_CLEAR_REG	Write
0xD101_0850	DMA_BASE + 0x50	DMA1_DES_TABLE_ADDR_REG	Read/Write
0xD101_0854	DMA_BASE + 0x54	DMA1_SRC_ADDR_REG	Read/Write
0xD101_0858	DMA_BASE + 0x58	DMA1_DST_ADDR_REG	Read/Write
0xD101_085C	DMA_BASE + 0x5C	DMA1_TRANS_CNT_REG	Read/Write
0xD101_0860	DMA_BASE + 0x60	DMA1_CUR_DESC_ADDR_REG	Read
0xD101_0864	DMA_BASE + 0x64	DMA1_CUR_SRC_ADDR_REG	Read
0xD101_0868	DMA_BASE + 0x68	DMA1_CUR_DEST_ADDR_REG	Read
0xD101_086C	DMA_BASE + 0x6C	DMA1_CUR_TRANS_CNT_REG	Read
0xD101_0870	DMA_BASE + 0x70	DMA1_PEND_REQ_REG	Read
0xD101_0880	DMA_BASE + 0x80	DMA2_CONTROL_REG	Read/Write
0xD101_0884	DMA_BASE + 0x84	DMA2_INT_ENABLE_REG	Read/Write
0xD101_0888	DMA_BASE + 0x88	DMA2_INT_REG	Read
0xD101_084C	DMA_BASE + 0x8C	DMA2_INT_CLEAR_REG	Write
0xD101_0890	DMA_BASE + 0x90	DMA2_DES_TABLE_ADDR_REG	Read/Write
0xD101_0894	DMA_BASE + 0x94	DMA2_SRC_ADDR_REG	Read/Write
0xD101_0898	DMA_BASE + 0x98	DMA2_DST_ADDR_REG	Read/Write
0xD101_089C	DMA_BASE + 0x9C	DMA2_TRANS_CNT_REG	Read/Write
0xD101_08A0	DMA_BASE + 0xA0	DMA2_CUR_DESC_ADDR_REG	Read
0xD101_08A4	DMA_BASE + 0xA4	DMA2_CUR_SRC_ADDR_REG	Read
0xD101_08A8	DMA_BASE + 0xA8	DMA2_CUR_DEST_ADDR_REG	Read
0xD101_08AC	DMA_BASE + 0xAC	DMA2_CUR_TRANS_CNT_REG	Read
0xD101_08B0	DMA_BASE + 0xB0	DMA2_PEND_REQ_REG	Read

Absolute Address	Symbolic Address	Register Name	Access
0xD101_08C0	DMA_BASE + 0xC0	DMA3_CONTROL_REG	Read/Write
0xD101_08C4	DMA_BASE + 0xC4	DMA3_INT_ENABLE_REG	Read/Write
0xD101_08C8	DMA_BASE + 0xC8	DMA3_INT_REG	Read
0xD101_08CC	DMA_BASE + 0xCC	DMA3_INT_CLEAR_REG	Write
0xD101_08D0	DMA_BASE + 0xD0	DMA3_DES_TABLE_ADDR_REG	Read/Write
0xD101_08D4	DMA_BASE + 0xD4	DMA3_SRC_ADDR_REG	Read/Write
0xD101_08D8	DMA_BASE + 0xD8	DMA3_DST_ADDR_REG	Read/Write
0xD101_08DC	DMA_BASE + 0xDC	DMA3_TRANS_CNT_REG	Read/Write
0xD101_08E0	DMA_BASE + 0xE0	DMA3_CUR_DESC_ADDR_REG	Read
0xD101_08E4	DMA_BASE + 0xE4	DMA3_CUR_SRC_ADDR_REG	Read
0xD101_08E8	DMA_BASE + 0xE8	DMA3_CUR_DEST_ADDR_REG	Read
0xD101_08EC	DMA_BASE + 0xEC	DMA3_CUR_TRANS_CNT_REG	Read
0xD101_08F0	DMA_BASE + 0xF0	DMA3_PEND_REQ_REG	Read
0xD101_08FC	DMA_BASE + 0xFC	DMA_CRC_REG	Read
0xD101_0900	**UART_BASE**	**UART**	
0xD101_0900	UART_BASE + 0x00	UART0_CONTROL_REG	Read/Write
0xD101_0920	UART_BASE + 0x20	UART0_RX_TX_REG or UART0_DIVISOR_LSB_REG	DLAB_BIT=0 Read/Write DLAB_BIT=1
0xD101_0924	UART_BASE + 0x24	UART0_INT_ENABLE_REG or UART0_DIVISOR_MSB_REG	DLAB_BIT=0 Read/Write DLAB_BIT=1
0xD101_0928	UART_BASE + 0x28	UART0_INT_ID_REG or UART0_FIFO_CTRL_REG	Read or Write
0xD101_092C	UART_BASE + 0x2C	UART0_LINE_CONTROL_REG	Read/Write
0xD101_0930	UART_BASE + 0x30	UART0_MODEM_CONTROL_REG	Read/Write
0xD101_0934	UART_BASE + 0x34	UART0_LINE_STATUS_REG	Read
0xD101_0938	UART_BASE + 0x38	UART0_MODEM_STATUS_REG	Read
0xD101_093C	UART_BASE + 0x3C	UART0_SCRATCHPAD_REG	Read/Write
0xD101_0940	UART_BASE + 0x40	UART1_CONTROL_REG	Read/Write
0xD101_0960	UART_BASE + 0x60	UART1_RX_TX_REG or UART1_DIVISOR_LSB_REG	Read/Write or Read/Write
0xD101_0964	UART_BASE + 0x64	UART1_INT_ENABLE_REG or UART1_DIVISOR_MSB_REG	Read/Write or Read/Write
0xD101_0968	UART_BASE + 0x68	UART1_INT_ID_REG or UART1_FIFO_CTRL_REG	Read or Write
0xD101_096C	UART_BASE + 0x6C	UART1_LINE_CONTROL_REG	Read/Write
0xD101_0970	UART_BASE + 0x70	UART1_MODEM_CONTROL_REG	Read/Write
0xD101_0974	UART_BASE + 0x74	UART1_LINE_STATUS_REG	Read
0xD101_0978	UART_BASE + 0x78	UART1_MODEM_STATUS_REG	Read
0xD101_097C	UART_BASE + 0x7C	UART1_SCRATCHPAD_REG	Read/Write
0xD101_0A00	**BPU_BASE**	**Breakpoint Control**	
0xD101_0A00 to 0xD101_0A54		Breakpoint Control	Read/Write

Absolute Address	Symbolic Address	Register Name	Access
0xD101_1100	**PU_BASE**	**Protection Unit**	
0xD101_1104	PU_BASE + 0x04	PU_CONTROL_REG	Read/Write
0xD101_1108	PU_BASE + 0x08	PU_CACHEABLE_AREA_REG	Read/Write
0xD101_1114	PU_BASE + 0x14	PU_PROTECTION_AREA_REG	Read/Write
0xD101_111C	PU_BASE + 0x1C	PU_CACHE_INVALIDATE_REG	Write
0xD101_1120	PU_BASE + 0x20	PU_AREA_DEF0_REG	Read/Write
0xD101_1124	PU_BASE + 0x24	PU_AREA_DEF1_REG	Read/Write
0xD101_1128	PU_BASE + 0x28	PU_AREA_DEF2_REG	Read/Write
0xD101_112C	PU_BASE + 0x2C	PU_AREA_DEF3_REG	Read/Write
0xD101_1130	PU_BASE + 0x30	PU_AREA_DEF4_REG	Read/Write
0xD101_1134	PU_BASE + 0x34	PU_AREA_DEF5_REG	Read/Write
0xD101_1138	PU_BASE + 0x38	PU_AREA_DEF6_REG	Read/Write
0xD101_113C	PU_BASE + 0x3C	PU_AREA_DEF7_REG	Read/Write
0xD102_0000 to 0xD102_FFFF		Test Area (Cache + Control)	Read/Write
0xD103_0000	**INTERNAL_RAM_BASE**	**Scratchpad RAM**	
0xD103_0000 to 0xD103_3FFF		Internal Scratchpad	Read/Write
0xD103_4000 to 0xD103_FFFF		Unused (Reserved)	
0xD104_0000 to 0xD107_FFFF		Configuration Memory	Read/Write

SUBJECT TO CHANGE

Ordering Information

Device Type — **A7S04**
A7S20

TA7 S 20 - 60 Q C

Triscend A7
Configurable
System-on-Chip
Family

Configurable
System Logic
(CSL) Cells x 100
(approximate)

Temperature Range

C = Commercial
I = Industrial

Package Style

Q = 208-pin Plastic Quad Flat Pack
G = 324-ball Ball Grid Array

Maximum processor
and CSI bus frequency

33 MHz
60 MHz
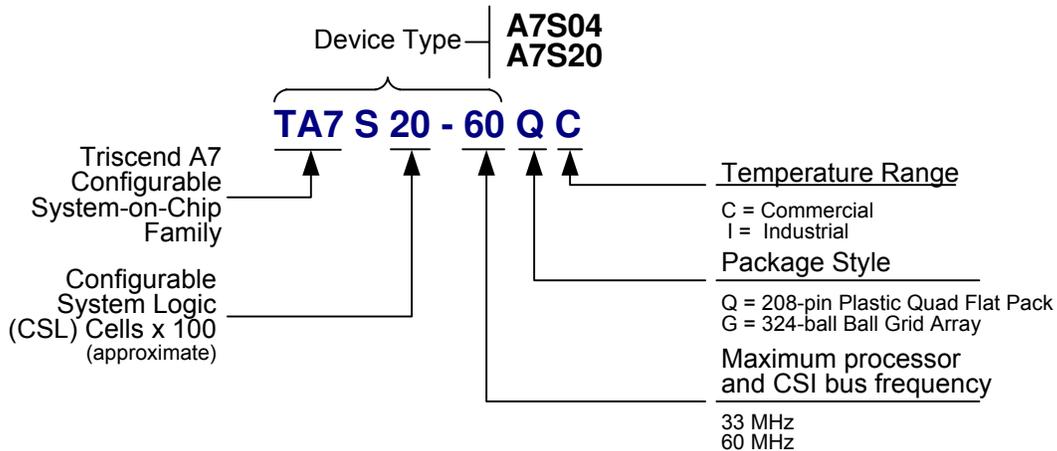
Sales Offices

Headquarters

Triscend Corporation
301 North Whisman Road
Mountain View, CA 94043
USA

Tel: **1-650-968-8668**
Fax: 1-650-934-9393

Sales Representatives

Visit the Triscend web site for the Triscend sales representative in your area.
Web: **www.triscend.com**

U.S. Distribution

All American Semiconductor
Web: **www.allamerican.com**

Unique Technologies
Web:
www.unique-technologies.com

Europe, Middle-East, South Africa

Unique
Web: **www.unique.memec.com**

Asia-Pacific

Unique Asia Pacific
Web: **www.unique-ap.com**

Japan

Internix Incorporated

Tel: **(03) 3369-1101**
Fax: (03) 3366-8566
Web: **www.internix.co.jp**

Triscend Corporation
301 N. Whisman Rd.
Mountain View, CA 94043-3969
Tel: 1-650-968-8668
Fax: 1-650-934-9393
E-mail: **SupportCenter@Triscend.com**
Web: **www.triscend.com**

Triscend A7S Configurable System-on-Chip Platform

C O N T E N T S