ATtiny88 Automotive

8-bit AVR Microcontroller with 8K Bytes In-System Programmable Flash

DATASHEET

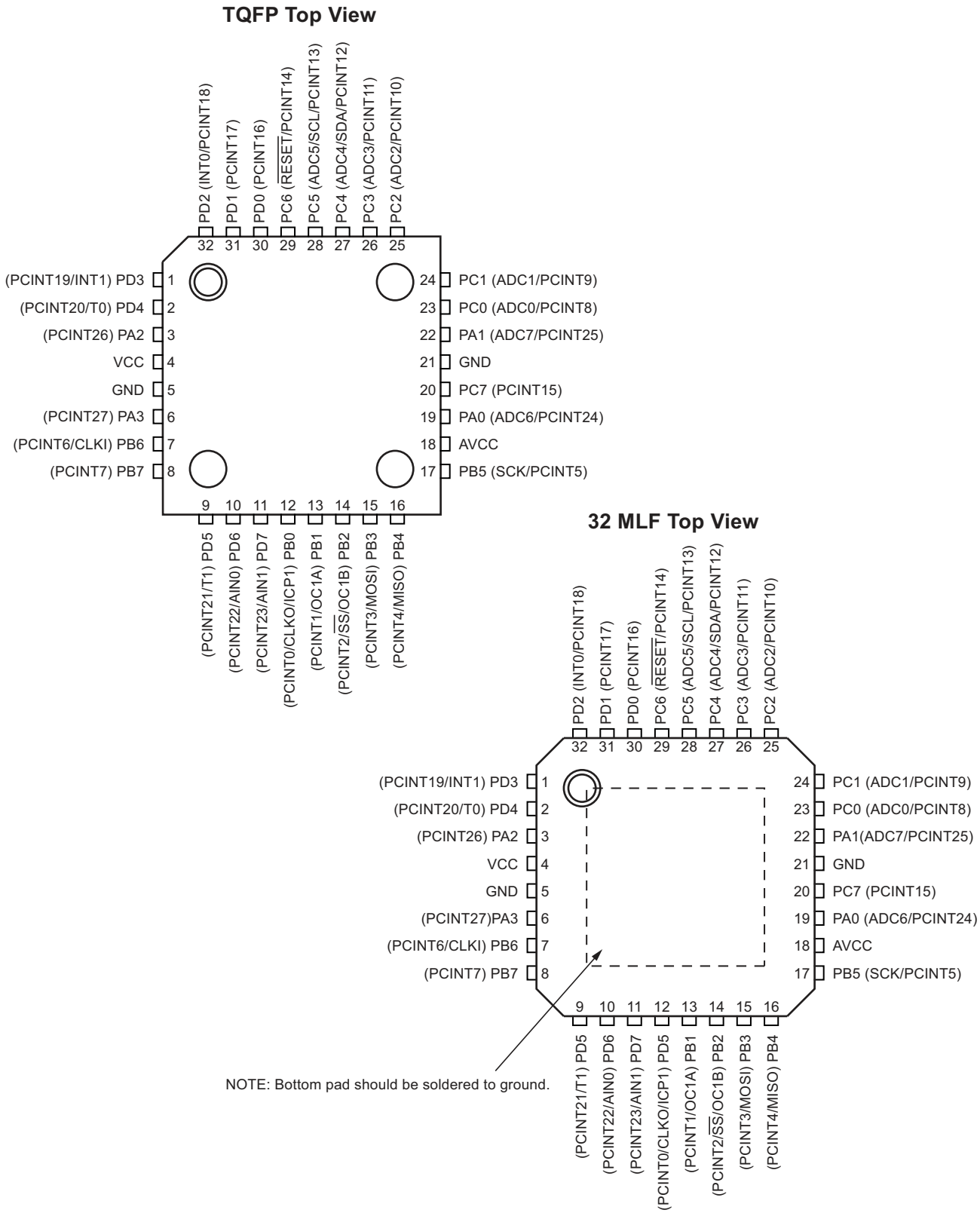
Features

- High performance, low power AVR® 8-Bit microcontroller
- Advanced RISC architecture
  - 123 powerful instructions – most single clock cycle execution
  - 32 x 8 general purpose working registers
  - Fully static operation
- High endurance non-volatile memory segments
  - 8K bytes of in-system self-programmable flash program memory(ATtiny88)
  - 64 bytes EEPROM
  - 512 bytes internal SRAM
  - Write/erase cycles: 10,000 Flash/100,000 EEPROM
  - Programming lock for software security
- Peripheral features
  - One 8-bit Timer/Counter with separate prescaler and compare mode
  - One 16-bit Timer/Counter with prescaler, and compare and capture modes
  - 8-channel 10-bit ADC in 32-lead TQFP and 32-pad QFN package
  - Master/slave SPI serial interface
  - Byte-oriented 2-wire serial interface (Phillips I^2C compatible)
  - Programmable watchdog timer with separate on-chip oscillator
  - On-chip analog comparator
  - Interrupt and wake-up on pin change
- Special microcontroller features
  - debugWIRE on-chip debug system
  - In-system programmable via SPI port
  - Power-on reset and programmable brown-out detection
  - Internal calibrated oscillator
  - External and internal interrupt sources
  - Three sleep modes: Idle, ADC noise reduction and power-down
- I/O and packages
  - 28 programmable I/O lines in 32-lead TQFP and 32-pad QFN package
- Operating voltage:
  - 2.7– 5.5V
- Automotive temperature range:
  - –40°C to +125°C

- Speed grade:
  - 0 to 8MHz at 2.7 – 5.5V
  - 0 to 16MHz at 4.5 – 5.5V
- Low Power Consumption
  - Active mode: 8MHz at 5V – 4.4mA
  - Power-down mode: at5V – 6uA

1. Pin Configurations

Figure 1-1. Pinout of ATtiny88



1.1 Disclaimer

Typical values contained in this data sheet are based on simulations and characterization of actual ATtiny88 AVR® microcontrollers manufactured on the typical process technology. Applicable automotive min. and max. values are based on characterization of devices representative of the whole process excursion (corner run).

1.2 Pin Descriptions

1.2.1 VCC

Digital supply voltage.

1.2.2 GND

Ground.

1.2.3 Port A (PA3:0)

Port A is a 4-bit bi-directional I/O port with internal pull-up resistors (selected for each bit) in 32-lead TQFP and 32-pad QFN package. The PA3..0 output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, port A pins that are externally pulled low will source current if the pull-up resistors are activated. The port A pins are tri-stated when a reset condition becomes active, even if the clock is not running.

1.2.4 Port B (PB7:0)

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the internal clock operating circuit.

The various special features of port B are elaborated in Section 10.3.2 “Alternate Functions of Port B” on page 59 and Section 6. “System Clock and Clock Options” on page 25.

1.2.5 Port C (PC7, PC5:0)

Port C is a 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC7 and PC5..0 output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

1.2.6 PC6/$\overline{\text{RESET}}$

If the RSTDISBL fuse is programmed, PC6 is used as an input pin.

If the RSTDISBL fuse is unprogrammed, PC6 is used as a reset input. A low level on this pin for longer than the minimum pulse width will generate a reset, even if the clock is not running. The minimum pulse length is given in Table 21-4 on page 186. Shorter pulses are not guaranteed to generate a reset.

The various special features of port C are elaborated in Section 10.3.3 “Alternate Functions of Port C” on page 61.

1.2.7 Port D (PD7:0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PD7..4 output buffers have symmetrical drive characteristics with both high sink and source capabilities, while the PD3..0 output buffers have stronger sink capabilities. As inputs, port D pins that are externally pulled low will source current if the pull-up resistors are activated. The port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

The various special features of port D are elaborated in Section 10.3.4 “Alternate Functions of Port D” on page 63.

1.2.8 AV_{CC}

AV_{CC} is the supply voltage pin for the A/D converter and a selection of I/O pins. This pin should be externally connected to V_{CC} even if the ADC is not used. If the ADC is used, it is recommended this pin is connected to V_{CC} through a low-pass filter, as described in Section 17.9 “Analog Noise Canceling Techniques” on page 152.

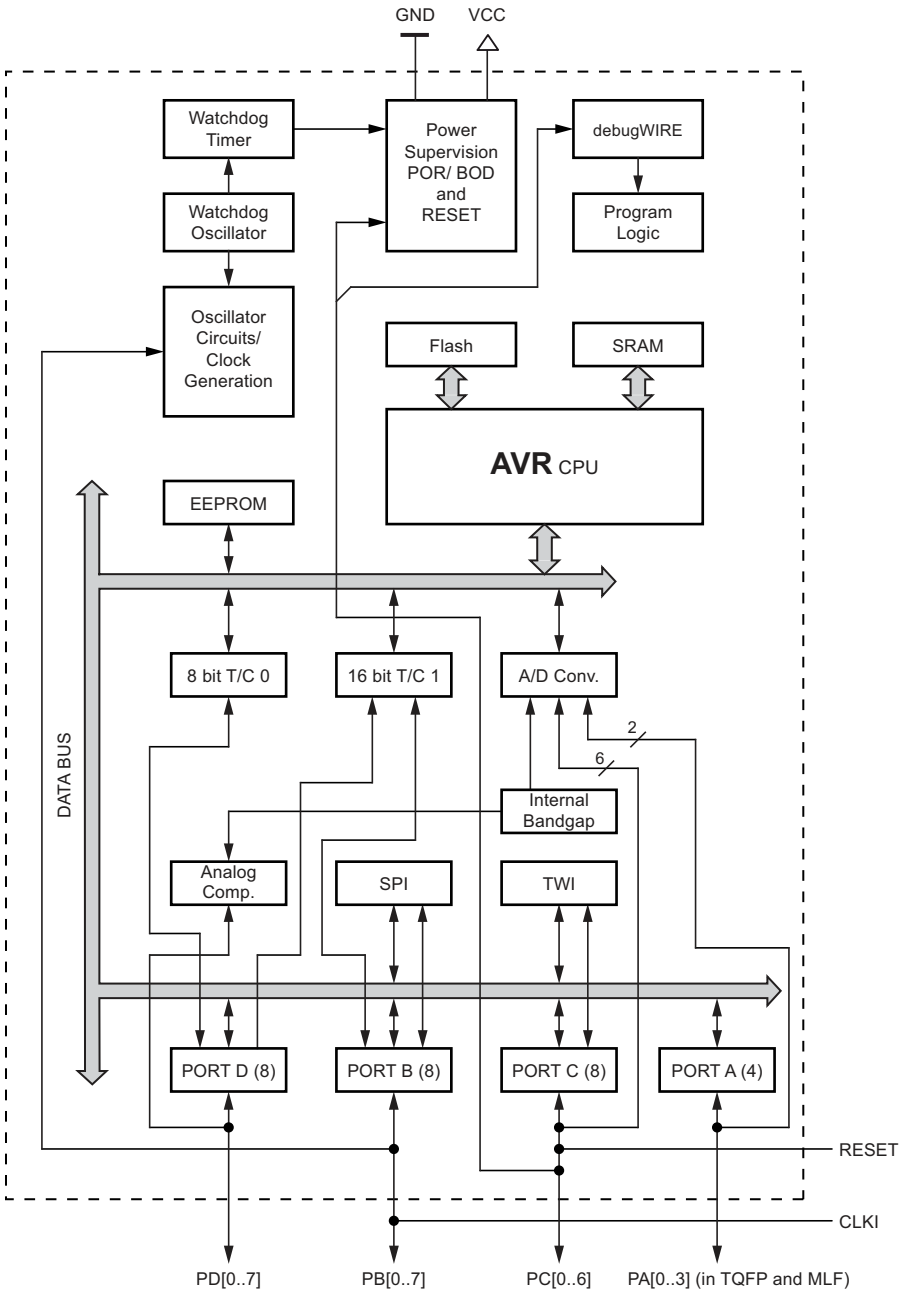
The following pins receive their supply voltage from AV_{CC}: PC7, PC5:0 and (in 32-lead packages) PA1:0. All other I/O pins take their supply voltage from V_{CC}.

2. Overview

The Atmel® ATtiny88 is a low-power CMOS 8-bit microcontroller based on the AVR® enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the Atmel ATtiny88 achieves throughputs approaching 1MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

2.1 Block Diagram

Figure 2-1. Block Diagram



The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the arithmetic logic unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The Atmel®ATtiny88 provides the following features: 8Kbytes of in-system programmable flash, 64 bytes EEPROM, 512 bytes SRAM, 28 general purpose I/O lines, 32 general purpose working registers, two flexible Timer/Counters with compare modes, internal and external interrupts, a byte-oriented 2-wire serial interface, an SPI serial port, a 6-channel 10-bit ADC (8 channels in 32-lead TQFP and 32-pad QFN packages), a programmable watchdog timer with internal oscillator, and three software selectable power saving modes. Idle mode stops the CPU while allowing Timer/Counters, 2-wire serial interface, SPI port, and interrupt system to continue functioning. Power-down mode saves the register contents but freezes the oscillator, disabling all other chip functions until the next interrupt or hardware reset. ADC noise reduction mode stops the CPU and all I/O modules except ADC, and helps to minimize switching noise during ADC conversions.

The device is manufactured using Atmel high density non-volatile memory technology. The on-chip ISP flash allows the program memory to be reprogrammed in-system through an SPI serial interface, by a conventional non-volatile memory programmer, or by an on-chip boot program running on the AVR® core. The boot program can use any interface to download the application program in the flash memory. By combining an 8-bit RISC CPU with in-system self-programmable flash on a monolithic chip, the Atmel ATtiny88 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The Atmel ATtiny88 AVR is supported by a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators and evaluation kits.

2.2 Automotive Quality Grade

The Atmel ATtiny88 have been developed and manufactured according to the most stringent requirements of the international standard ISO-TS-16949 grade 1. This data sheet contains limit values extracted from the results of extensive characterization (temperature and voltage). The quality and reliability of the ATtiny88 have been verified during regular product qualification as per AEC-Q100.

As indicated in the ordering information paragraph, the product is available in only one temperature grade,

Table 2-1. Temperature Grade Identification for Automotive Products

Temperature	Temperature Identifier	Comments
–40; +125	Z	Full automotive temperature range

3. Additional Information

3.1 Resources

A comprehensive set of development tools, application notes and datasheets are available for download at http://www.atmel.com/avr.

3.2 About Code Examples

This documentation contains simple code examples that briefly show how to use various parts of the device. These code examples assume that the part specific header file is included before compilation. Be aware that not all C compiler vendors include bit definitions in the header files and interrupt handling in C is compiler dependent. Please confirm with the C compiler documentation for more details.

For I/O registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

3.3 Data Retention

Reliability qualification results show that the projected data retention failure rate is much less than 1 PPM over 20 years at 125°C or 100 years at 25°C.

3.4 Disclaimer

Typical values contained in this data sheet are based on simulations and characterization of actual Atmel® ATtiny88 AVR® microcontrollers manufactured on the typical process technology. Applicable automotive min. and max. values are based on characterization of devices representative of the whole process excursion (corner run).

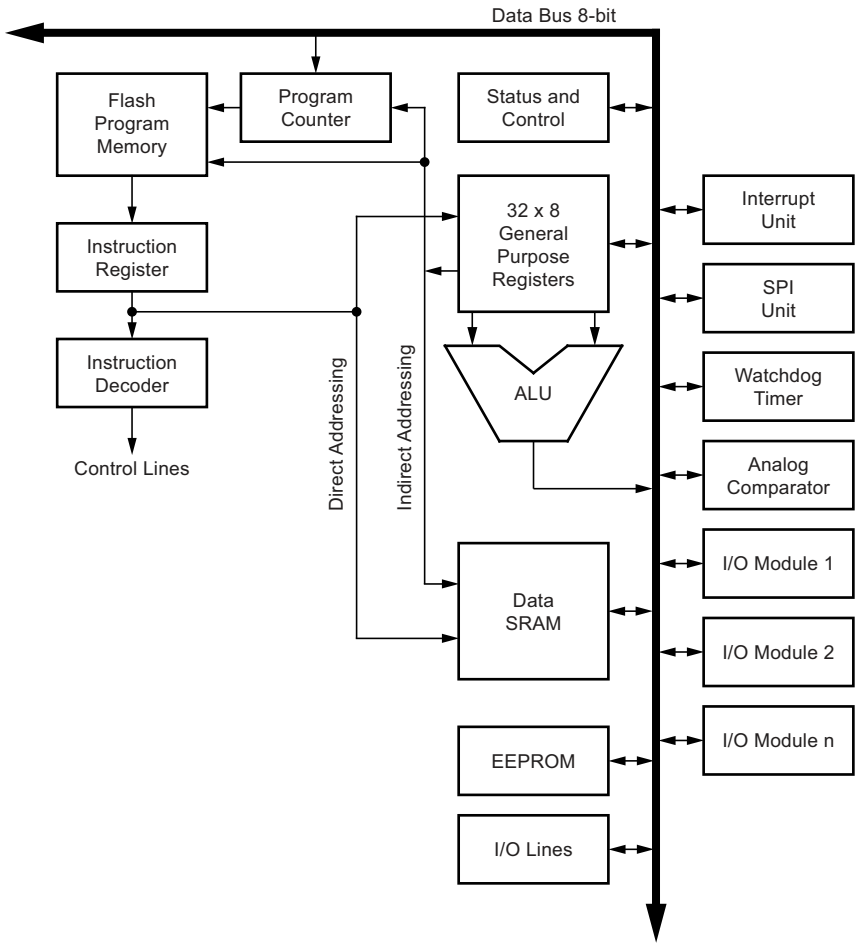
4. AVR CPU Core

4.1 Introduction

This section discusses the AVR® core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupts.

4.2 Architectural Overview

Figure 4-1. Block Diagram of the AVR Architecture



In order to maximize performance and parallelism, the AVR uses a harvard architecture – with separate memories and buses for program and data. Instructions in the program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is in-system reprogrammable flash memory.

The fast-access register file contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle arithmetic logic unit (ALU) operation. In a typical ALU operation, two operands are output from the register file, the operation is executed, and the result is stored back in the register file – in one clock cycle.

Six of the 32 registers can be used as three 16-bit indirect address register pointers for data space addressing – enabling efficient address calculations. One of the these address pointers can also be used as an address pointer for look up tables in flash program memory. These added function registers are the 16-bit X-, Y-, and Z-register, described later in this section.

The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After an arithmetic operation, the status register is updated to reflect information about the result of the operation.

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR® instructions have a single 16-bit word format, but there are also 32-bit instructions.

During interrupts and subroutine calls, the return address program counter (PC) is stored on the stack. The stack is effectively allocated in the general data SRAM, and consequently the stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the SP in the reset routine (before subroutines or interrupts are executed). The stack pointer (SP) is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture.

The memory spaces in the AVR architecture are all linear and regular memory maps.

A flexible interrupt module has its control registers in the I/O space with an additional global interrupt enable bit in the status register. All interrupts have a separate interrupt vector in the interrupt vector table. The interrupts have priority in accordance with their interrupt vector position. The lower the interrupt vector address, the higher the priority.

The I/O memory space contains 64 addresses for CPU peripheral functions as control registers, SPI, and other I/O functions. The I/O memory can be accessed directly, or as the data space locations following those of the register file, 0x20 – 0x5F. In addition, the Atmel® ATtiny88 has extended I/O space from 0x60 – 0xFF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

4.3 ALU – Arithmetic Logic Unit

The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format. See Section 24. “Instruction Set Summary” on page 209 for a detailed description.

4.4 Status Register

The status register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the status register is updated after all ALU operations, as specified in the instruction set reference. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

The status register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

The AVR® status register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – I: Global Interrupt Enable**

The global interrupt enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the global interrupt enable register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

- **Bit 6 – T: Bit Copy Storage**

The bit copy instructions BLD (bit LoaD) and BST (bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the register file can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the register file by the BLD instruction.

- **Bit 5 – H: Half Carry Flag**

The half carry flag H indicates a half carry in some arithmetic operations. Half carry Is useful in BCD arithmetic. See the “Instruction Set Description” for detailed information.

- **Bit 4 – S: Sign Bit, S = N ⊕ V**

The S-bit is always an exclusive or between the negative flag N and the two’s complement overflow flag V. See the “Instruction Set Description” for detailed information.

- **Bit 3 – V: Two’s Complement Overflow Flag**

The two’s complement overflow flag V supports two’s complement arithmetics. See the “Instruction Set Description” for detailed information.

- **Bit 2 – N: Negative Flag**

The negative flag N indicates a negative result in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

- **Bit 1 – Z: Zero Flag**

The zero flag Z indicates a zero result in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

- **Bit 0 – C: Carry Flag**

The carry flag C indicates a carry in an arithmetic or logic operation. See the “Instruction Set Description” for detailed information.

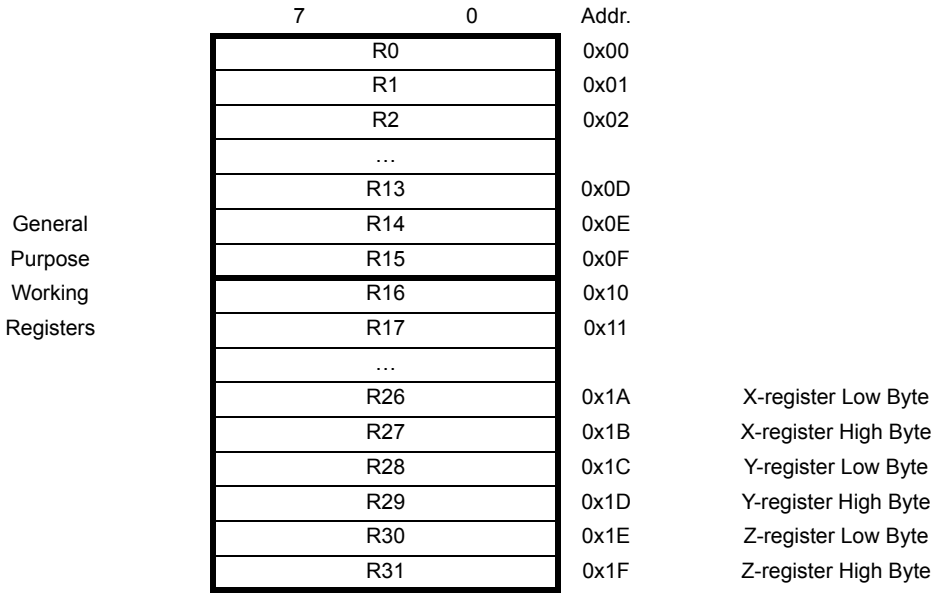
4.5 General Purpose Register File

The register file is optimized for the AVR® enhanced RISC instruction set. In order to achieve the required performance and flexibility, the following input/output schemes are supported by the register file:

- One 8-bit output operand and one 8-bit result input
- Two 8-bit output operands and one 8-bit result input
- Two 8-bit output operands and one 16-bit result input
- One 16-bit output operand and one 16-bit result input

Figure 4-2 shows the structure of the 32 general purpose working registers in the CPU.

Figure 4-2. AVR CPU General Purpose Working Registers



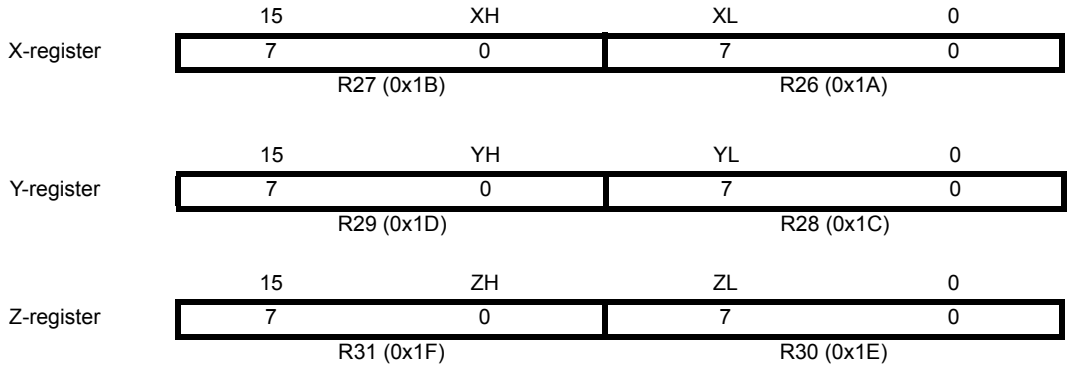
Most of the instructions operating on the register file have direct access to all registers, and most of them are single cycle instructions.

As shown in Figure 4-2, each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user data space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X-, Y- and Z-pointer registers can be set to index any register in the file.

4.5.1 The X-register, Y-register, and Z-register

The registers R26..R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the data space. The three indirect address registers X, Y, and Z are defined as described in Figure 4-3.

Figure 4-3. The X-, Y-, and Z-Registers



In the different addressing modes these address registers have functions as fixed displacement, automatic increment, and automatic decrement (see the instruction set reference for details).

4.6 Stack Pointer

The stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The stack pointer register always points to the top of the stack. Note that the stack is implemented as growing from higher memory locations to lower memory locations. This implies that a stack PUSH command decreases the stack pointer.

The stack pointer points to the data SRAM stack area where the subroutine and interrupt stacks are located. This stack space in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. The stack pointer should be set to point to RAMEND. The stack pointer is decremented by one when data is pushed onto the stack with the PUSH instruction, and it is decremented by two when the return address is pushed onto the stack with subroutine call or interrupt. The stack pointer is incremented by one when data is popped from the stack with the POP instruction, and it is incremented by two when data is popped from the stack with return from subroutine RET or return from interrupt RETI.

The AVR® stack pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent. Note that the data space in some implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH register will not be present.

Bit	15	14	13	12	11	10	9	8	
	SP15	**SP14**	**SP13**	**SP12**	**SP11**	**SP10**	**SP9**	**SP8**	**SPH**
	SP7	**SP6**	**SP5**	**SP4**	**SP3**	**SP2**	**SP1**	**SP0**	**SPL**
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	
	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	

4.7 Instruction Execution Timing

This section describes the general access timing concepts for instruction execution. The AVR® CPU is driven by the CPU clock clk_{CPU}, directly generated from the selected clock source for the chip. No internal clock division is used.

Figure 4-4 shows the parallel instruction fetches and instruction executions enabled by the harvard architecture and the fast-access register file concept. This is the basic pipelining concept to obtain up to 1MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

Figure 4-4. The Parallel Instruction Fetches and Instruction Executions

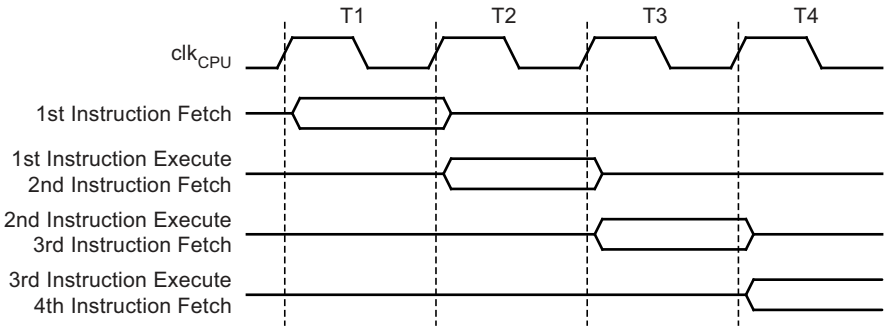
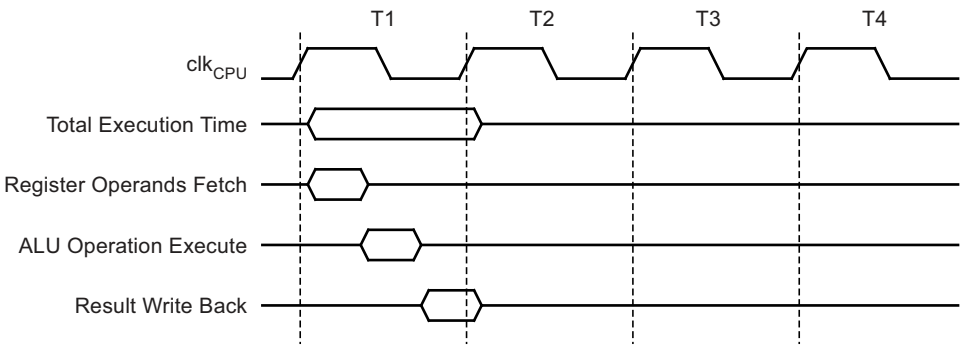


Figure 4-5 shows the internal timing concept for the register file. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

Figure 4-5. Single Cycle ALU Operation



4.8 Reset and Interrupt Handling

The AVR provides several different interrupt sources. These interrupts and the separate reset vector each have a separate program vector in the program memory space. All interrupts are assigned individual enable bits which must be written logic one together with the global interrupt enable bit in the status register in order to enable the interrupt. Depending on the program counter value, interrupts may be automatically disabled when lock bits LB2 or LB1 are programmed. This feature improves software security. See Section 20. “Memory Programming” on page 168 for details.

The lowest addresses in the program memory space are by default defined as the reset and interrupt vectors. The complete list of vectors is shown in Section 9. “Interrupts” on page 44. The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0 – the external interrupt request 0. Refer to Section 9. “Interrupts” on page 44 for more information.

When an interrupt occurs, the global interrupt enable I-bit is cleared and all interrupts are disabled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a return from interrupt instruction – RETI – is executed.

There are basically two types of interrupts. The first type is triggered by an event that sets the interrupt flag. For these interrupts, the program counter is vectored to the actual interrupt vector in order to execute the interrupt handling routine, and hardware clears the corresponding interrupt flag. Interrupt flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the interrupt flag will be set and remembered until the interrupt is enabled, or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while the global interrupt enable bit is cleared, the corresponding interrupt flag(s) will be set and remembered until the global interrupt enable bit is set, and will then be executed by order of priority.

The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have interrupt flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered.

When the AVR® exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.

Note that the status register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software.

When using the CLI instruction to disable interrupts, the interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction. The following example shows how this can be used to avoid interrupts during the timed EEPROM write sequence.

Assembly Code Example

```
in     r16, SREG     ; store SREG value
cli    ; disable interrupts during timed sequence
sbi    EECR, EEMPE   ; start EEPROM write
sbi    EECR, EEPE
out    SREG, r16     ; restore SREG value (I-bit)
```

C Code Example

```
char cSREG;
cSREG = SREG; /* store SREG value */
/* disable interrupts during timed sequence */
_CLI();
EECR |= (1<<EEMPE); /* start EEPROM write */
EECR |= (1<<EEPE);
SREG = cSREG; /* restore SREG value (I-bit) */
```

When using the SEI instruction to enable interrupts, the instruction following SEI will be executed before any pending interrupts, as shown in this example.

Assembly Code Example

```
sei    ; set Global Interrupt Enable
sleep  ; enter sleep, waiting for interrupt
; note: will enter sleep before any pending interrupt(s)
```

C Code Example

```
__enable_interrupt(); /* set Global Interrupt Enable */
__sleep(); /* enter sleep, waiting for interrupt */
/* note: will enter sleep before any pending interrupt(s) */
```

4.8.1 Interrupt Response Time

The interrupt execution response for all the enabled AVR interrupts is four clock cycles minimum. After four clock cycles the program vector address for the actual interrupt handling routine is executed. During this four clock cycle period, the program counter is pushed onto the stack. The vector is normally a jump to the interrupt routine, and this jump takes three clock cycles. If an interrupt occurs during execution of a multi-cycle instruction, this instruction is completed before the interrupt is served. If an interrupt occurs when the MCU is in sleep mode, the interrupt execution response time is increased by four clock cycles. This increase comes in addition to the start-up time from the selected sleep mode.

A return from an interrupt handling routine takes four clock cycles. During these four clock cycles, the program counter (two bytes) is popped back from the stack, the stack pointer is incremented by two, and the I-bit in SREG is set.

5. Memories

This section describes the different memories in the Atmel® ATtiny88. The AVR® architecture has two main memory spaces, the data memory and the program memory space. In addition, the Atmel ATtiny88 features an EEPROM memory for data storage. All three memory spaces are linear and regular.

5.1 In-System Reprogrammable Flash Program Memory

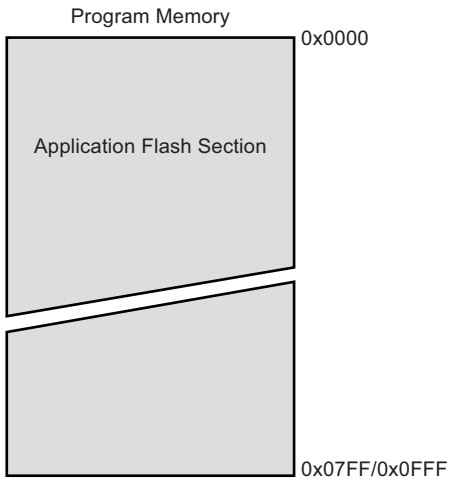
The Atmel ATtiny88 contains 8Kbytes on-chip in-system reprogrammable flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the flash is organized as 4K x 16. Atmel ATtiny88 does not have separate boot loader and application program sections, and the SPM instruction can be executed from the entire flash. See SELFPRGEN description in Section 19.5.1 “SPMCSR – Store Program Memory Control and Status Register” on page 167 for more details.

The flash memory has an endurance of at least 10,000 write/erase cycles. The Atmel ATtiny88 program counter (PC) is 11/12 bits wide, thus addressing the 4K program memory locations. Section 20. “Memory Programming” on page 168 contains a detailed description on flash programming in SPI- or parallel programming mode.

Constant tables can be allocated within the entire program memory address space (see instructions LPM – load program memory and SPM – store program memory).

Timing diagrams for instruction fetch and execution are presented in Section 4.7 “Instruction Execution Timing” on page 14.

Figure 5-1. Program Memory Map of ATtiny88



5.2 SRAM Data Memory

Figure 5-2 shows how the Atmel® ATtiny88 SRAM memory is organized.

The Atmel ATtiny88 is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in the opcode for the IN and OUT instructions. For the extended I/O space from 0x60 – 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

The lower 512/768 data memory locations address both the register file, the I/O memory, extended I/O memory, and the internal data SRAM. The first 32 locations address the register file, the next 64 location the standard I/O memory, then 160 locations of extended I/O memory, and the next 512 locations address the internal data SRAM.

The five different addressing modes for the data memory cover: Direct, indirect with displacement, indirect, indirect with pre-decrement, and indirect with post-increment. In the register file, registers R26 to R31 feature the indirect addressing pointer registers.

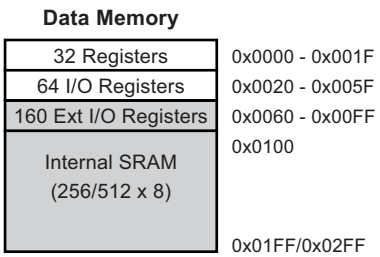
The direct addressing reaches the entire data space.

The indirect with displacement mode reaches 63 address locations from the base address given by the Y- or Z-register.

When using register indirect addressing modes with automatic pre-decrement and post-increment, the address registers X, Y, and Z are decremented or incremented.

The 32 general purpose working registers, 64 I/O registers, 160 extended I/O registers, and the 512 bytes of internal data SRAM in the Atmel ATtiny88 are all accessible through all these addressing modes. The register file is described in Section 4.5 “General Purpose Register File” on page 12.

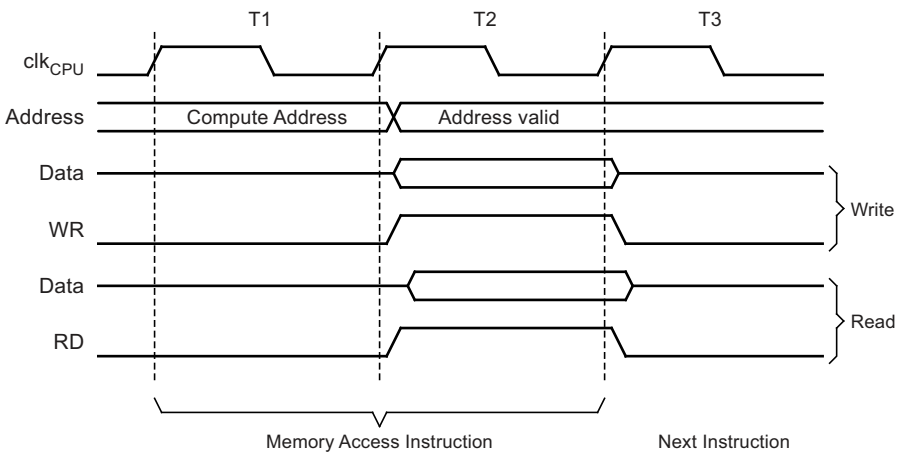
Figure 5-2. Data Memory Map



5.2.1 Data Memory Access Times

This section describes the general access timing concepts for internal memory access. The internal data SRAM access is performed in two clk_{CPU} cycles as described in Figure 5-3.

Figure 5-3. On-chip Data SRAM Access Cycles



5.3 EEPROM Data Memory

Atmel® ATtiny88 devices contain 64 bytes of data EEPROM memory, organized as a separate data space in which single bytes can be read and written. The EEPROM has an endurance of at least 100,000 write/erase cycles. The access between the EEPROM and the CPU is described in the following, specifying the EEPROM Address registers, the EEPROM data register, and the EEPROM control register.

Section 20. “Memory Programming” on page 168 contains a detailed description on EEPROM programming in SPI or parallel programming mode.

5.3.1 EEPROM Read/Write Access

The EEPROM access registers are located in I/O space.

The write access time for the EEPROM is given in Table 5-2 on page 23. A self-timing function, however, lets the user software detect when the next byte can be written. If the user code contains instructions that write the EEPROM, some precautions must be taken. In heavily filtered power supplies, V_{CC} is likely to rise or fall slowly on power-up/down. This causes the device for some period of time to run at a voltage lower than specified as minimum for the clock frequency used. See Section 5.3.6 “Preventing EEPROM Corruption” on page 20 for details on how to avoid problems in these situations.

In order to prevent unintentional EEPROM writes, a specific write procedure must be followed. Refer to Section 5.3.2 “Atomic Byte Programming” on page 18 and Section 5.3.3 “Split Byte Programming” on page 18 for details on this.

When the EEPROM is read, the CPU is halted for four clock cycles before the next instruction is executed. When the EEPROM is written, the CPU is halted for two clock cycles before the next instruction is executed.

5.3.2 Atomic Byte Programming

The simplest programming method is called atomic byte programming. When writing a byte to the EEPROM, the user must write the address into register EEAR and data into register EEDR. If the EEPMn bits are zero, writing EEPE (within four cycles after EEMPE is written) will trigger the erase/write operation. Both the erase and write cycle are done in one operation and the total programming time is given in Table 5-1 on page 22. The EEPE bit remains set until the erase and write operations are completed. While the device is busy with programming, it is not possible to do any other EEPROM operations.

5.3.3 Split Byte Programming

It is possible to split the erase/write cycle in two different operations. This may be useful if the system requires short access time for some limited period of time (typically if the power supply voltage falls). In order to take advantage of this method, it is required that the locations to be written have been erased before the write operation.

5.3.4 Erase

To erase a byte, the address must be written to EEAR. If the EEPMn bits are 0b01, writing the EEPE (within four cycles after EEMPE is written) will trigger the erase operation only (programming time is given in Table 5-1 on page 22). The EEPE bit remains set until the erase operation completes. While the device is busy programming, it is not possible to do any other EEPROM operations.

5.3.5 Write

To write a location, the user must write the address into EEAR and the data into EEDR. If the EEPMn bits are 0b10, writing the EEPE (within four cycles after EEMPE is written) will trigger the write operation only (programming time is given in Table 5-1 on page 22). The EEPE bit remains set until the write operation completes. If the location to be written has not been erased before write, the data that is stored must be considered as lost. While the device is busy with programming, it is not possible to do any other EEPROM operations.

The calibrated oscillator is used to time the EEPROM accesses. Make sure the oscillator frequency is within the requirements described in Section 6.8.1 “OSCCAL – Oscillator Calibration Register” on page 30.

The following code examples show one assembly and one C function for erase, write, or atomic write of the EEPROM. The examples assume that interrupts are controlled (e.g., by disabling interrupts globally) so that no interrupts will occur during execution of these functions.

Assembly Code Example

```
EEPROM_write:
        ; Wait for completion of previous write
        sbic    EECR,EEPE
        rjmp    EEPROM_write
        ; Set up address (r17) in address register
        out     EEARL, r17
        ; Write data (r19) to Data Register
        out     EEDR,r19
        ; Write logical one to EEMPE
        sbi     EECR,EEMPE
        ; Start eeprom write by setting EEPE
        sbi     EECR,EEPE
        ret
```

C Code Example

```
void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{
        /* Wait for completion of previous write */
        while(EECR & (1<<EEPE))
                ;
        /* Set up address and Data Registers */
        EEAR = uiAddress;
        EEDR = ucData;
        /* Write logical one to EEMPE */
        EECR |= (1<<EEMPE);
        /* Start eeprom write by setting EEPE */
        EECR |= (1<<EEPE);
}
```

The next code examples show assembly and C functions for reading the EEPROM. The examples assume that interrupts are controlled so that no interrupts will occur during execution of these functions.

Assembly Code Example

```
EEPROM_read:
        ; Wait for completion of previous write
        sbic    EECR,EEPE
        rjmp    EEPROM_read
        ; Set up address (r17) in address register
        out     EEARL, r17
        ; Start eeprom read by writing EERE
        sbi     EECR,EERE
        ; Read data from Data Register
        in      r16,EEDR
        ret
```

C Code Example

```
unsigned char EEPROM_read(unsigned int uiAddress)
{
        /* Wait for completion of previous write */
        while(EECR & (1<<EEPE))
        ;
        /* Set up address register */
        EEAR = uiAddress;
        /* Start eeprom read by writing EERE */
        EECR |= (1<<EERE);
        /* Return data from Data Register */
        return EEDR;
}
```

5.3.6 Preventing EEPROM Corruption

During periods of low V_{CC} the EEPROM data can be corrupted because the supply voltage is too low for the CPU and the EEPROM to operate properly. These issues are the same as for board level systems using EEPROM, and the same design solutions should be applied.

An EEPROM data corruption can be caused by two situations when the voltage is too low. First, a regular write sequence to the EEPROM requires a minimum voltage to operate correctly. Secondly, the CPU itself can execute instructions incorrectly, if the supply voltage is too low.

EEPROM data corruption can easily be avoided by keeping the RESET active (low) during periods of insufficient power supply voltage. This can be done by enabling the internal brown-out detector (BOD). If the detection level of the internal BOD does not match the needed detection level, an external low V_{CC} reset protection circuit can be used. If a reset occurs while a write operation is in progress, the write operation will be completed provided that the power supply voltage is sufficient.

5.4 I/O Memory

The I/O space definition of the Atmel® ATtiny88 is shown in Section 23. “Register Summary” on page 202.

All Atmel ATtiny88 I/Os and peripherals are placed in the I/O space. All I/O locations may be accessed by the LD/LDS/LDD and ST/STS/STD instructions, transferring data between the 32 general purpose working registers and the I/O space. I/O registers within the address range 0x00 – 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions. Refer to the instruction set section for more details. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 – 0x3F must be used. When addressing I/O registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The Atmel ATtiny88 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in opcode for the IN and OUT instructions. For the extended I/O space from 0x60 – 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.

Some of the status flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such status flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.

The I/O and peripherals control registers are explained in later sections.

5.4.1 General Purpose I/O Registers

ATtiny88 contains three general purpose I/O registers. These registers can be used for storing any information, and they are particularly useful for storing global variables and status flags. General purpose I/O registers within the address range 0x00 – 0x1F are directly bit-accessible using the SBI, CBI, SBIS, and SBIC instructions.

5.5 Register Description

5.5.1 EEARH and EEARL – EEPROM Address Register

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	–	-	**EEARH**
	-	-	**EEAR5**	**EEAR4**	**EEAR3**	**EEAR2**	**EEAR1**	**EEAR0**	EEARL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	X	X	X	X	X	X	

- **Bits 15..6 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bits 5..0 – EEAR5..0: EEPROM Address**

The EEPROM address registers – EEARH and EEARL specify the EEPROM address in the 64 bytes EEPROM space. The EEPROM data bytes are addressed linearly between 0 and 63. The initial value of EEAR is undefined. A proper value must be written before the EEPROM may be accessed.

5.5.2 EEDR – EEPROM Data Register

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	EEDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..0 – EEDR7.0: EEPROM Data**

For the EEPROM write operation, the EEDR register contains the data to be written to the EEPROM in the address given by the EEAR register. For the EEPROM read operation, the EEDR contains the data read out from the EEPROM at the address given by EEAR.

5.5.3 EECR – EEPROM Control Register

Bit	7	6	5	4	3	2	1	0	
	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	EECR
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	X	X	0	0	X	0	

- **Bits 7..6 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bits 5, 4 – EEPM1 and EEPM0: EEPROM Programming Mode Bits**

The EEPROM programming mode bit setting defines which programming action that will be triggered when writing EEPE. It is possible to program data in one atomic operation (erase the old value and program the new value) or to split the erase and write operations in two different operations. The programming times for the different modes are shown in Table 5-1. While EEPE is set, any write to EEPMn will be ignored. During reset, the EEPMn bits will be reset to 0b00 unless the EEPROM is busy programming.

Table 5-1. EEPROM Mode Bits

EEPM1	EEPM0	Programming Time	Operation
0	0	3.4ms	Erase and write in one operation (atomic operation)
0	1	1.8ms	Erase only
1	0	1.8ms	Write only
1	1	–	Reserved for future use

- **Bit 3 – EERIE: EEPROM Ready Interrupt Enable**

Writing EERIE to one enables the EEPROM ready interrupt if the I bit in SREG is set. Writing EERIE to zero disables the interrupt. The EEPROM ready interrupt generates a constant interrupt when EEPE is cleared. The interrupt will not be generated during EEPROM write or SPM.

- **Bit 2 – EEMPE: EEPROM Master Write Enable**

The EEMPE bit determines whether setting EEPE to one causes the EEPROM to be written. When EEMPE is set, setting EEPE within four clock cycles will write data to the EEPROM at the selected address If EEMPE is zero, setting EEPE will have no effect. When EEMPE has been written to one by software, hardware clears the bit to zero after four clock cycles. See the description of the EEPE bit for an EEPROM write procedure.

- **Bit 1 – EEPE: EEPROM Write Enable**

The EEPROM write enable signal EEPE is the write strobe to the EEPROM. When address and data are correctly set up, the EEPE bit must be written to one to write the value into the EEPROM. The EEMPE bit must be written to one before a logical one is written to EEPE, otherwise no EEPROM write takes place. The following procedure should be followed when writing the EEPROM (the order of steps 3 and 4 is not essential):

1. Wait until EEPE becomes zero.
2. Wait until SELFPRGEN in SPMCSR becomes zero.
3. Write new EEPROM address to EEAR (optional).
4. Write new EEPROM data to EEDR (optional).
5. Write a logical one to the EEMPE bit while writing a zero to EEPE in EECR.
6. Within four clock cycles after setting EEMPE, write a logical one to EEPE.

The EEPROM can not be programmed during a CPU write to the flash memory. The software must check that the flash programming is completed before initiating a new EEPROM write. If the flash is never being updated by the CPU, step 2 can be omitted.

Caution: An interrupt between step 5 and step 6 will make the write cycle fail, since the EEPROM master write enable will time-out. If an interrupt routine accessing the EEPROM is interrupting another EEPROM access, the EEAR or EEDR register will be modified, causing the interrupted EEPROM access to fail. It is recommended to have the global interrupt flag cleared during all the steps to avoid these problems.

When the write access time has elapsed, the EEPE bit is cleared by hardware. The user software can poll this bit and wait for a zero before writing the next byte. When EEPE has been set, the CPU is halted for two cycles before the next instruction is executed.

- **Bit 0 – EERE: EEPROM Read Enable**

The EEPROM read enable signal EERE is the read strobe to the EEPROM. When the correct address is set up in the EEAR register, the EERE bit must be written to a logic one to trigger the EEPROM read. The EEPROM read access takes one instruction, and the requested data is available immediately. When the EEPROM is read, the CPU is halted for four cycles before the next instruction is executed.

The user should poll the EEPE bit before starting the read operation. If a write operation is in progress, it is neither possible to read the EEPROM, nor to change the EEAR register.

The calibrated oscillator is used to time the EEPROM accesses. Table 5-2 lists the typical programming time for EEPROM access from the CPU.

Table 5-2. EEPROM Programming Time

Symbol	Number of Calibrated Oscillator Cycles	Typ Programming Time
EEPROM write (from CPU)	26,368	3.4ms

5.5.4 GPIOR2 – General Purpose I/O Register 2

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	GPIOR2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

This register may be used freely for storing any kind of data.

5.5.5 GPIOR1 – General Purpose I/O Register 1

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	GPIOR1
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

This register may be used freely for storing any kind of data.

5.5.6 GPIOR0 – General Purpose I/O Register 0

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	GPIOR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

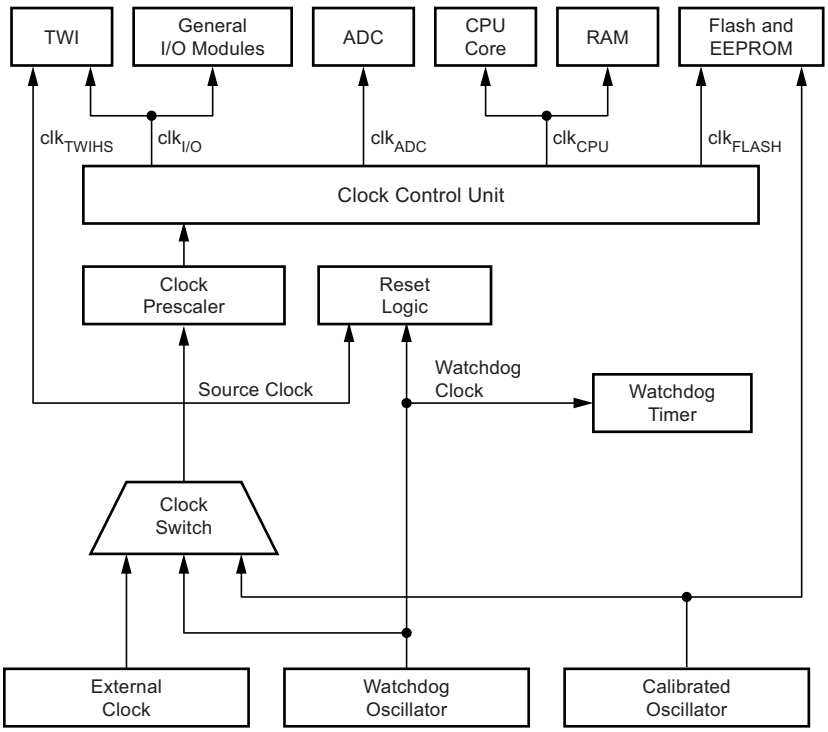
This register may be used freely for storing any kind of data.

6. System Clock and Clock Options

6.1 Clock Systems and their Distribution

Figure 6-1 presents the principal clock systems in the AVR® and their distribution. All of the clocks need not be active at a given time. In order to reduce power consumption, the clocks to modules not being used can be halted by using different sleep modes, as described in Section 7. “Power Management and Sleep Modes” on page 32. The clock systems are detailed below.

Figure 6-1. Clock Distribution



6.1.1 CPU Clock – clk_{CPU}

The CPU clock is routed to parts of the system concerned with operation of the AVR core. Examples of such modules are the general purpose register file, the status register and the data memory holding the stack pointer. Halting the CPU clock inhibits the core from performing general operations and calculations.

6.1.2 I/O Clock – $clk_{I/O}$

The I/O clock is used by the majority of the I/O modules such as Timer/Counters, the serial peripheral interface and the external interrupt module. Note, that some external interrupts are detected by asynchronous logic, meaning they are recognized even if the I/O clock is halted. Also note that the start condition detection of the two-wire interface module is asynchronous, meaning TWI address recognition works in all sleep modes (even when $clk_{I/O}$ is halted).

6.1.3 Flash Clock – clk_{FLASH}

The flash clock controls operation of the flash interface. The flash clock is usually active simultaneously with the CPU clock.

6.1.4 Analog to Digital Converter Clock – clk_{ADC}

The ADC is provided with a dedicated clock domain. This allows halting the CPU and I/O clocks in order to reduce noise generated by digital circuitry. This gives more accurate ADC conversion results.

6.1.5 High-Speed Two-Wire Interface Clock – clk_{TWIHS}

The TWI clock controls the operation of the two-wire interface module, when operated in high-speed mode. In practice, this clock is identical to the source clock of the device. See Section 15.5.2 “Bit Rate Generator Unit” on page 117.

6.2 Clock Sources

The device has the following clock source options, selectable by flash fuse bits as shown below. The clock from the selected source is input to the AVR® clock generator, and routed to the appropriate modules.

Table 6-1. Device Clocking Options(1)

CKSEL1..0	Device Clocking Option
00	External clock
01	Reserved
10	Calibrated internal oscillator
11	Internal 128kHz oscillator

Note: 1. For all fuses “1” means unprogrammed while “0” means programmed.

6.2.1 Default Clock Source

The device is shipped with internal oscillator at 8.0MHz and with the fuse CKDIV8 programmed, resulting in 1.0MHz system clock. The startup time is set to maximum and time-out period enabled (CKSEL = 0b10, SUT = 0b10, CKDIV8 = 0). The default setting ensures that all users can make their desired clock source setting using any available programming interface.

6.2.2 Clock Startup Sequence

Any clock source needs a sufficient V_{CC} to start oscillating and a minimum number of oscillating cycles before it can be considered stable.

To ensure sufficient V_{CC}, the device issues an internal reset with a time-out delay (t_{TOUT}) after the device reset is released by all other reset sources. Section 8. “System Control and Reset” on page 37 describes the start conditions for the internal reset. The delay (t_{TOUT}) is timed from the watchdog oscillator and the number of cycles in the delay is set by the SUTx and CKSELx fuse bits. The selectable delays are shown in Table 6-2. The frequency of the Watchdog oscillator is voltage and temperature dependent, as shown in Section 22.8 “Internal Oscillator Speed” on page 200 and Figure 22-17 on page 200.

Table 6-2. Length of Startup Sequence.

CKSEL1:0	SUT1:0	Number of WDT Cycles	Typical Time-out
00 10 11	00	0	0ms
	01	4K (4,096)	4ms
	10	8K (8,192)	64ms
	11	Reserved	Reserved
01	XX	Reserved	Reserved

The main purpose of the delay is to keep the AVR® in reset until it is supplied with minimum V_{CC}. The delay will not monitor the actual voltage and it will be required to select a delay longer than the V_{CC} rise time. If this is not possible, an internal or external brown-out detection circuit should be used. A BOD circuit will ensure sufficient V_{CC} before it releases the reset, and the time-out delay can be disabled. Disabling the time-out delay without utilizing a brown-out detection circuit is not recommended.

The oscillator is required to oscillate for a minimum number of cycles before the clock is considered stable. An internal ripple counter monitors the oscillator output clock, and keeps the internal reset active for a given number of clock cycles. The reset is then released and the device will start to execute.

The start-up sequence for the clock includes both the time-out delay and the start-up time when the device starts up from reset. When starting up from power-down mode, V_{CC} is assumed to be at a sufficient level and only the start-up time is included.

6.3 Calibrated Internal Oscillator

By default, the internal oscillator provides an approximate 8.0MHz clock. Though voltage and temperature dependent, this clock can be very accurately calibrated by the user. See Table 21-1 on page 185 for more details. The device is shipped with the CKDIV8 fuse programmed. See Section 6.7 “System Clock Prescaler” on page 29 for more details.

This clock may be selected as the system clock by programming the CKSEL fuses as shown in Table 6-3 on page 27. If selected, it will operate with no external components. During reset, hardware loads the pre-programmed calibration value into the OSCCAL register and thereby automatically calibrates the oscillator. The accuracy of this calibration is shown as factory calibration in Table 21-1 on page 185.

By changing the OSCCAL register from SW, see Section 6.8.1 “OSCCAL – Oscillator Calibration Register” on page 30, it is possible to get a higher calibration accuracy than by using the factory calibration. The accuracy of this calibration is shown as user calibration in Table 21-1 on page 185.

When this oscillator is used as the chip clock, the watchdog oscillator will still be used for the watchdog timer and for the reset time-out. For more information on the pre-programmed calibration value, see the section Section 20.4 “Calibration Byte” on page 170.

Table 6-3. Selecting Internal Calibrated Oscillator Mode(1)(2)

CKSEL1..0	Nominal Frequency (MHz)
10	8.0

Notes:
1. The device is shipped with this option selected.
2. If 8MHz frequency exceeds the specification of the device (depends on V_{CC}), the CKDIV8 fuse can be programmed in order to divide the internal frequency by 8.

When this oscillator is selected, start-up times are determined by the SUT fuses as shown in the table below.

Table 6-4. Start-up Times for the Internal Calibrated Oscillator Clock Selection

SUT1..0	Power Conditions	Start-up Time from Power-down	Additional Delay from Reset (V_{CC} = 5.0V)
00	BOD enabled	6CK	14CK(1)
01	Fast rising power	6CK	14CK + 4ms
10	Slowly rising power	6CK	14CK + 64ms(2)
11	Reserved		

Notes:
1. If the RSTDISBL fuse is programmed, this start-up time will be increased to 14CK + 4ms to ensure programming mode can be entered.
2. The device is shipped with this option selected.

6.4 128kHz Internal Oscillator

The 128kHz internal oscillator is a low power oscillator providing a clock of 128kHz. The frequency is nominal at 3V and 25°C. This clock may be select as the system clock by programming the CKSEL fuses to “11” as shown in Table 6-5.

.

Table 6-5. Selecting 128kHz Internal Oscillator Modes

CKSEL1..0	Nominal Frequency
11	128kHz

When this clock source is selected, start-up times are determined by the SUT fuses as shown in Table 6-6.

Table 6-6. Start-up Times for the 128kHz Internal Oscillator

SUT1..0	Power Conditions	Start-up Time from Power-down	Additional Delay from Reset
00	BOD enabled	6CK	14CK[1]
01	Fast rising power	6CK	14CK + 4ms
10	Slowly rising power	6CK	14CK + 64ms
11	Reserved		

Note: 1. If the RSTDISBL fuse is programmed, this start-up time will be increased to 14CK + 4ms to ensure programming mode can be entered.

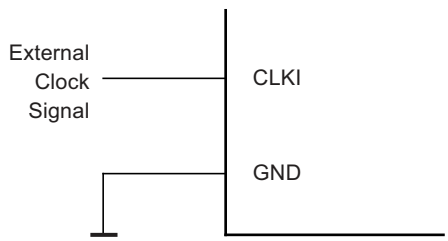
6.5 External Clock

To drive the device from an external clock source, CLKI should be driven as shown in Figure 6-2. To run the device on an external clock, the CKSEL fuses must be programmed to “00” (see Table 6-7).

Table 6-7. Selecting External Clock

CKSEL1..0	Frequency
00	0 – 16MHz

Figure 6-2. External Clock Drive Configuration



When this clock source is selected, start-up times are determined by the SUT fuses as shown in Table 6-8.

Table 6-8. Start-up Times for the External Clock Selection

SUT1..0	Power Conditions	Start-up Time from Power-down	Additional Delay from Reset (V_{CC} = 5.0V)
00	BOD enabled	6CK	14CK
01	Fast rising power	6CK	14CK + 4ms
10	Slowly rising power	6CK	14CK + 64ms
11	Reserved		

When applying an external clock, it is required to avoid sudden changes in the applied clock frequency to ensure stable operation of the MCU. A variation in frequency of more than 2% from one clock cycle to the next can lead to unpredictable behavior. If changes of more than 2% is required, ensure that the MCU is kept in reset during the changes.

Note that the system clock prescaler can be used to implement run-time changes of the internal clock frequency while still ensuring stable operation. Refer to Section 6.7 “System Clock Prescaler” on page 29 for details.

6.6 Clock Output Buffer

The device can output the system clock on the CLKO pin. To enable the output, the CKOUT fuse has to be programmed. This mode is suitable when the chip clock is used to drive other circuits on the system. The clock also will be output during reset, and the normal operation of I/O pin will be overridden when the fuse is programmed. Any clock source, including the internal oscillator, can be selected when the clock is output on CLKO. If the system clock prescaler is used, it is the divided system clock that is output.

6.7 System Clock Prescaler

The Atmel® ATtiny88 has a system clock prescaler, and the system clock can be divided by setting the Section 6.8.2 “CLKPR – Clock Prescale Register” on page 30. This feature can be used to decrease the system clock frequency and the power consumption when the requirement for processing power is low. This can be used with all clock source options, and it will affect the clock frequency of the CPU and all synchronous peripherals. $clk_{I/O}$, clk_{ADC}, clk_{CPU}, and clk_{FLASH} are divided by a factor as shown in Table 6-9 on page 31.

When switching between prescaler settings, the system clock prescaler ensures that no glitches occur in the clock system. It also ensures that no intermediate frequency is higher than neither the clock frequency corresponding to the previous setting, nor the clock frequency corresponding to the new setting. The ripple counter that implements the prescaler runs at the frequency of the undivided clock, which may be faster than the CPU's clock frequency. Hence, it is not possible to determine the state of the prescaler – even if it were readable, and the exact time it takes to switch from one clock division to the other cannot be exactly predicted. From the time the CLKPS values are written, it takes between T1 + T2 and T1 + 2 × T2 before the new clock frequency is active. In this interval, 2 active clock edges are produced. Here, T1 is the previous clock period, and T2 is the period corresponding to the new prescaler setting.

To avoid unintentional changes of clock frequency, a special write procedure must befollowed to change the CLKPS bits:

1. Write the clock prescaler change enable (CLKPCE) bit to one and all other bitsin CLKPR to zero.
2. Within four cycles, write the desired value to CLKPS while writing a zero to CLKPCE.

Interrupts must be disabled when changing prescaler setting to make sure the write procedure is not interrupted.

6.8 Register Description

6.8.1 OSCCAL – Oscillator Calibration Register

Bit	7	6	5	4	3	2	1	0	
	CAL7	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0	OSCCAL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	Device Specific Calibration Value								

The oscillator calibration register is used to trim the internal oscillator to remove process variations from the oscillator frequency. A pre-programmed calibration value is automatically written to this register during chip reset, giving the factory calibrated frequency as specified in Table 21-1 on page 185. The application software can write to the OSCCAL register to change the oscillator frequency. The oscillator can be calibrated to frequencies as specified in Table 21-1 on page 185. calibration outside the given range is not guaranteed.

Note that this oscillator is used to time EEPROM and flash write accesses, and the write times will be affected accordingly. If the EEPROM or flash are written, do not calibrate to more than 8.8MHz. Otherwise, the EEPROM or flash write may fail.

All register bits are in use for frequency. A setting of 0x00 gives the lowest frequency and a setting of 0xFF gives the highest frequency.

6.8.2 CLKPR – Clock Prescale Register

Bit	7	6	5	4	3	2	1	0	
	CLKPCE	–	–	–	**CLKPS3**	**CLKPS2**	**CLKPS1**	**CLKPS0**	**CLKPR**
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	See Bit Description				

- **Bit 7 – CLKPCE: Clock Prescaler Change Enable**

The CLKPCE bit must be written to logic one to enable change of the CLKPS bits. The CLKPCE bit is only updated when the other bits in CLKPR are simultaneously written to zero. CLKPCE is cleared by hardware four cycles after it is written or when CLKPS bits are written. Rewriting the CLKPCE bit within this time-out period does neither extend the time-out period, nor clear the CLKPCE bit.

- **Bits 6..4 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bits 3..0 – CLKPS3..0: Clock Prescaler Select Bits 3 – 0**

These bits define the division factor between the selected clock source and the internal system clock. These bits can be written run-time to vary the clock frequency to suit the application requirements. As the divider divides the master clock input to the MCU, the speed of all synchronous peripherals is reduced when a division factor is used. The division factors are given in Table 6-9 on page 31.

The CKDIV8 fuse determines the initial value of the CLKPS bits. If CKDIV8 is unprogrammed, the CLKPS bits will be reset to 0b0000. If CKDIV8 is programmed, CLKPS bits are reset to 0b0011, giving a division factor of 8 at start up.

This feature should be used if the selected clock source has a higher frequency than the maximum frequency of the device at the present operating conditions. Note that any value can be written to the CLKPS bits regardless of the CKDIV8 fuse setting.

The application software must ensure that a sufficient division factor is chosen if the selected clock source has a higher frequency than the maximum frequency of the device at the present operating conditions. The device is shipped with the CKDIV8 fuse programmed.

Table 6-9. Clock Prescaler Select

CLKPS3	CLKPS2	CLKPS1	CLKPS0	Clock Division Factor
0	0	0	0	1
0	0	0	1	2
0	0	1	0	4
0	0	1	1	8
0	1	0	0	16
0	1	0	1	32
0	1	1	0	64
0	1	1	1	128
1	0	0	0	256
1	0	0	1	Reserved
1	0	1	0	Reserved
1	0	1	1	Reserved
1	1	0	0	Reserved
1	1	0	1	Reserved
1	1	1	0	Reserved
1	1	1	1	Reserved

7. Power Management and Sleep Modes

Sleep modes enable the application to shut down unused modules in the MCU, thereby saving power. The AVR® provides various sleep modes allowing the user to tailor the power consumption to the application's requirements.

When enabled, the brown-out detector (BOD) actively monitors the power supply voltage during the sleep periods. To further save power, it is possible to disable the BOD in some sleep modes. See Section 7.2 "Software BOD Disable" on page 33 for more details.

7.1 Sleep Modes

Figure 6-1 on page 25 presents the different clock systems in the Atmel® ATtiny88, and their distribution. The figure is helpful in selecting an appropriate sleep mode. Table 7-1 shows the different sleep modes, their wake up sources and the BOD disable ability.

Table 7-1. Active Clock Domains and Wake-up Sources in the Different Sleep Modes

Sleep Mode	Active Clock Domain				Oscillator	Wake-up Source					
	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	Main Clock Source Enabled	INT1, INT0 and Pin Change	TWI Address Match	EEPROM Ready	ADC	WDT	Other I/O
Idle			X	X	X	X	X	X	X	X	X
ADC noise reduction				X	X	X(1)	X	X	X	X	
Power-down						X(1)	X			X	

Note: 1. For INT1 and INT0, only level interrupt

To enter any of the sleep modes, the SE bit in SMCR must be written to logic one and a SLEEP instruction must be executed. The SM1, and SM0 bits in the SMCR register select which sleep mode (Idle, ADC noise reduction, or power-down) will be activated by the SLEEP instruction. See Table 7-2 on page 35 for a summary.

If an enabled interrupt occurs while the MCU is in a sleep mode, the MCU wakes up. The MCU is then halted for four cycles in addition to the start-up time, executes the interrupt routine, and resumes execution from the instruction following SLEEP. The contents of the register file and SRAM are unaltered when the device wakes up from sleep. If a reset occurs during sleep mode, the MCU wakes up and executes from the reset vector.

Note that if a level triggered interrupt is used for wake-up the changed level must be held for some time to wake up the MCU (and for the MCU to enter the interrupt service routine). See Section 9.2 "External Interrupts" on page 45 for details.

7.1.1 Idle Mode

When the SM1..0 bits are written to 00, the SLEEP instruction makes the MCU enter idle mode, stopping the CPU but allowing the SPI, analog comparator, ADC, 2-wire serial interface, Timer/Counters, watchdog, and the interrupt system to continue operating. This sleep mode basically halts clk_{CPU} and clk_{FLASH}, while allowing the other clocks to run.

Idle mode enables the MCU to wake up from external triggered interrupts as well as internal ones like the SPI interrupts. If wake-up from the analog comparator interrupt is not required, the analog comparator can be powered down by setting the ACD bit in the analog comparator control and status register – ACSR. This will reduce power consumption in Idle mode. If the ADC is enabled, a conversion starts automatically when this mode is entered.

7.1.2 ADC Noise Reduction Mode

When the SM1..0 bits are written to 01, the SLEEP instruction makes the MCU enter ADC noise reduction mode, stopping the CPU but allowing the ADC, the external interrupts, the 2-wire serial interface address watch and the watchdog to continue operating (if enabled). This sleep mode basically halts $clk_{I/O}$, clk_{CPU}, and clk_{FLASH}, while allowing the other clocks to run.

This improves the noise environment for the ADC, enabling higher resolution measurements. If the ADC is enabled, a conversion starts automatically when this mode is entered. Apart from the ADC conversion complete interrupt, only an external reset, a watchdog system reset, a watchdog interrupt, a brown-out reset, a 2-wire serial interface address match, an EEPROM ready interrupt, an external level interrupt on INT0 or INT1 or a pin change interrupt can wake up the MCU from ADC noise reduction mode.

7.1.3 Power-down Mode

When the SM1..0 bits are written to 10, the SLEEP instruction makes the MCU enter power-down mode. In this mode, the external oscillator is stopped, while the external interrupts, the 2-wire serial interface address watch, and the watchdog continue operating (if enabled). Only an external reset, a watchdog system reset, a watchdog interrupt, a brown-out reset, a 2-wire serial interface address match, an external level interrupt on INT0 or INT1, or a pin change interrupt can wake up the MCU. This sleep mode basically halts all generated clocks, allowing operation of asynchronous modules only.

Note that if a level triggered interrupt is used for wake-up from power-down mode, the changed level must be held for some time to wake up the MCU. Refer to Section 9.2 “External Interrupts” on page 45 for details.

When waking up from power-down mode, there is a delay from the wake-up condition occurs until the wake-up becomes effective. This allows the clock to restart and become stable after having been stopped. The wake-up period is defined by the same CKSEL fuses that define the reset time-out period, as described in Section 6.2 “Clock Sources” on page 26.

7.2 Software BOD Disable

When the brown-out detector (BOD) is enabled by BODLEVEL fuses (see Table 20-4 on page 169), the BOD is actively monitoring the power supply voltage during a sleep period. To save power, it is possible for software to disable the BOD in power-down mode. The sleep mode power consumption will then be at the same level as when BOD is globally disabled by fuses. If disabled by software, the BOD is turned off immediately after entering the sleep mode and automatically turned on upon wake-up. This ensures safe operation in case the V_{CC} level has dropped during the sleep period.

When the BOD has been disabled the wake-up time from sleep mode will be the same as the wake-up time from RESET. This is in order to ensure the BOD is working correctly before the MCU continues executing code.

BOD disable is controlled by bit 6, BODS (BOD Sleep) in the control register MCUCR, see Section 7.4.2 “MCUCR – MCU Control Register” on page 35. Writing this bit to one turns off the BOD in power-down mode, while a zero in this bit keeps BOD active. The default setting is zero, i.e. BOD active.

Writing to the BODS bit is controlled by a timed sequence and an enable bit, see Section 7.4.2 “MCUCR – MCU Control Register” on page 35.

7.3 Minimizing Power Consumption

There are several issues to consider when trying to minimize the power consumption in an AVR® controlled system. In general, sleep modes should be used as much as possible, and the sleep mode should be selected so that as few as possible of the device’s functions are operating. All functions not needed should be disabled. In particular, the following modules may need special consideration when trying to achieve the lowest possible power consumption.

7.3.1 Analog to Digital Converter

If enabled, the ADC will be enabled in all sleep modes. To save power, the ADC should be disabled before entering any sleep mode. When the ADC is turned off and on again, the next conversion will be an extended conversion. Refer to Section 17. “Analog-to-Digital Converter” on page 145 for details on ADC operation.

7.3.2 Analog Comparator

When entering Idle mode, the analog comparator should be disabled if not used. When entering ADC noise reduction mode, the analog comparator should be disabled. In other sleep modes, the analog comparator is automatically disabled. However, if the analog comparator is set up to use the internal voltage reference as input, the analog comparator should be disabled in all sleep modes. Otherwise, the internal voltage reference will be enabled, independent of sleep mode. Refer to Section 16. “Analog Comparator” on page 142 for details on how to configure the analog comparator.

7.3.3 Brown-out Detector

If the brown-out detector is not needed by the application, this module should be turned off. If the brown-out detector is enabled by the BODLEVEL fuses, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to Section 8.2.3 “Brown-out Detection” on page 39 for details on how to configure the brown-out detector.

7.3.4 Internal Voltage Reference

The internal voltage reference will be enabled when needed by the brown-out detection, the analog comparator or the ADC. If these modules are disabled as described in the sections above, the internal voltage reference will be disabled and it will not be consuming power. When turned on again, the user must allow the reference to start up before the output is used. If the reference is kept on in sleep mode, the output can be used immediately. Refer to Section 8.3 “Internal Voltage Reference” on page 40 for details on the start-up time.

7.3.5 Watchdog Timer

If the watchdog timer is not needed in the application, the module should be turned off. If the watchdog timer is enabled, it will be enabled in all sleep modes and hence always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to Section 8.4 “Watchdog Timer” on page 40 for details on how to configure the watchdog timer.

7.3.6 Port Pins

When entering a sleep mode, all port pins should be configured to use minimum power. The most important is then to ensure that no pins drive resistive loads. In sleep modes where both the I/O clock ($clk_{I/O}$) and the ADC clock (clk_{ADC}) are stopped, the input buffers of the device will be disabled. This ensures that no power is consumed by the input logic when not needed. In some cases, the input logic is needed for detecting wake-up conditions, and it will then be enabled. Refer to the section Section 10.2.6 “Digital Input Enable and Sleep Modes” on page 55 for details on which pins are enabled. If the input buffer is enabled and the input signal is left floating or have an analog signal level close to $V_{CC}/2$, the input buffer will use excessive power.

For analog input pins, the digital input buffer should be disabled at all times. An analog signal level close to $V_{CC}/2$ on an input pin can cause significant current even in active mode. Digital input buffers can be disabled by writing to the digital input disable registers (DIDR1 and DIDR0). Refer to Section 16.2.3 “DIDR1 – Digital Input Disable Register 1” on page 144 and Section 17.13.5 “DIDR0 – Digital Input Disable Register 0” on page 159 for details.

7.3.7 On-chip Debug System

If the on-chip debug system is enabled by the DWEN fuse and the chip enters sleep mode, the main clock source is enabled and hence always consumes power. In the deeper sleep modes, this will contribute significantly to the total current consumption.

7.4 Register Description

7.4.1 SMCR – Sleep Mode Control Register

The sleep mode control register contains control bits for power management.

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	SM1	SM0	SE	SMCR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..3 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bits 2..1 – SM1..0: Sleep Mode Select Bits 1 and 0**

These bits select between the available sleep modes as shown in Table 7-2.

Table 7-2. Sleep Mode Select

SM1	SM0	Sleep Mode
0	0	Idle
0	1	ADC noise reduction
1	0	Power-down
1	1	Reserved

- **Bit 0 – SE: Sleep Enable**

The SE bit must be written to logic one to make the MCU enter the sleep mode when the SLEEP instruction is executed. To avoid the MCU entering the sleep mode unless it is the programmer's purpose, it is recommended to write the sleep enable (SE) bit to one just before the execution of the SLEEP instruction and to clear it immediately after waking up.

7.4.2 MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
	–	BODS	BODSE	PUD	–	–	–	–	MCUCR
Read/Write	R	R/W	R/W	R/W	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7, 3..0 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 6 – BODS: BOD Sleep**

The BODS bit must be written to logic one in order to turn off BOD during sleep, see Table 7-2. Writing to the BODS bit is controlled by a timed sequence and an enable bit, BODSE in MCUCR. To disable BOD in relevant sleep modes, both BODS and BODSE must first be set to one. Then, to set the BODS bit, BODS must be set to one and BODSE must be set to zero within four clock cycles.

The BODS bit is active three clock cycles after it is set. A sleep instruction must be executed while BODS is active in order to turn off the BOD for the actual sleep mode. The BODS bit is automatically cleared after three clock cycles.

- **Bit 5 – BODSE: BOD Sleep Enable**

BODSE enables setting of BODS control bit, as explained in BODS bit description. BOD disable is controlled by a timed sequence.

7.4.3 PRR – Power Reduction Register

The power reduction register (PRR) provides a method to stop the clock to individual peripherals to reduce power consumption. The current state of the peripheral is frozen and the I/O registers can not be read or written. Resources used by the peripheral when stopping the clock will remain occupied, hence the peripheral should in most cases be disabled before stopping the clock. Waking up a module, which is done by clearing the bit in PRR, puts the module in the same state as before shutdown.

Module shutdown can be used in Idle mode and active mode to significantly reduce the overall power consumption. In all other sleep modes, the clock is already stopped.

Bit	7	6	5	4	3	2	1	0	
	PRTWI	**–**	**PRTIM0**	**–**	**PRTIM1**	**PRSPI**	**–**	**PRADC**	**PRR**
Read/Write	R/W	R	R/W	R	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – PRTWI: Power Reduction TWI**

Writing a logic one to this bit shuts down the TWI by stopping the clock to the module. When waking up the TWI again, the TWI should be re initialized to ensure proper operation.

- **Bits 6, 4, 1 – Res: Reserved**

These bits are reserved and will always read zero.

- **Bit 5 – PRTIM0: Power Reduction Timer/Counter0**

Writing a logic one to this bit shuts down the Timer/Counter0 module. When the Timer/Counter0 is enabled, operation will continue like before the shutdown.

- **Bit 3 – PRTIM1: Power Reduction Timer/Counter1**

Writing a logic one to this bit shuts down the Timer/Counter1 module. When the Timer/Counter1 is enabled, operation will continue like before the shutdown.

- **Bit 2 – PRSPI: Power Reduction Serial Peripheral Interface**

If using debugWIRE on-chip debug system, this bit should not be written to one.

Writing a logic one to this bit shuts down the serial peripheral interface by stopping the clock to the module. When waking up the SPI again, the SPI should be re initialized to ensure proper operation.

- **Bit 0 – PRADC: Power Reduction ADC**

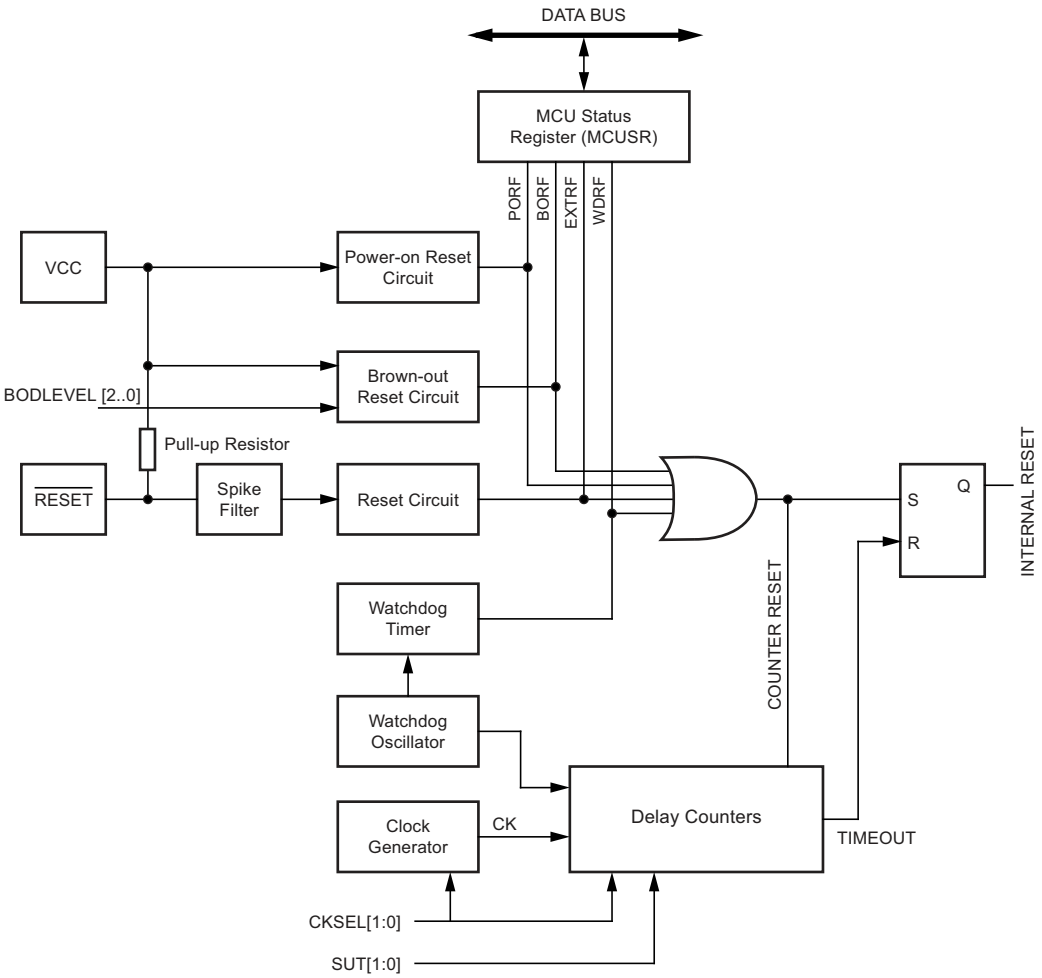
Writing a logic one to this bit shuts down the ADC. The ADC must be disabled before shut down. The analog comparator cannot be used when the ADC is shut down.

8. System Control and Reset

8.1 Resetting the AVR

During reset, all I/O registers are set to their initial values, and the program starts execution from the reset vector. The instruction placed at the reset vector must be an RJMP – relative jump – instruction to the reset handling routine. If the program never enables an interrupt source, the interrupt vectors are not used, and regular program code can be placed at these locations. The circuit diagram in Figure 8-1 shows the reset circuit. Table 21-4 on page 186 shows the electrical parameters of the reset circuitry.

Figure 8-1. Reset Logic



The I/O ports of the AVR® are immediately reset to their initial state when a reset source goes active. This does not require any clock source to be running.

After all reset sources have gone inactive, a delay counter is invoked, stretching the internal reset. This allows the power to reach a stable level before normal operation starts. The time-out period of the delay counter is defined by the user through the SUT and CKSEL fuses. The different selections for the delay period are presented in Section 6.2 “Clock Sources” on page 26.

8.2 Reset Sources

The ATtiny88 has four sources of reset:

- Power-on reset. The MCU is reset when the supply voltage is below the power-on reset threshold (V_{POT}), or when the supply voltage falls rapidly.
- External reset. The MCU is reset when a low level is present on the $\overline{\text{RESET}}$ pin for longer than the required pulse length.
- Watchdog system reset. The MCU is reset when the watchdog timer period expires and the watchdog system reset mode is enabled.
- Brown-out reset. The MCU is reset when the supply voltage V_{CC} is below the brown-out reset threshold (V_{BOT}) and the brown-out detector is enabled.

8.2.1 Power-on Reset

A power-on reset (POR) pulse is generated by an on-chip detection circuit. The detection level is defined in Table 21-4 on page 186. The POR is activated whenever V_{CC} is below the detection level. The POR circuit can be used to trigger the start-up Reset, as well as to detect a failure in supply voltage.

A power-on reset (POR) circuit ensures that the device is reset from power-on. Reaching the power-on reset threshold voltage invokes the delay counter, which determines how long the device is kept in RESET after V_{CC} rise. The RESET signal is activated again, without any delay, when V_{CC} decreases below the detection level.

Figure 8-2. MCU Start-up, $\overline{\text{RESET}}$ Tied to V_{CC}

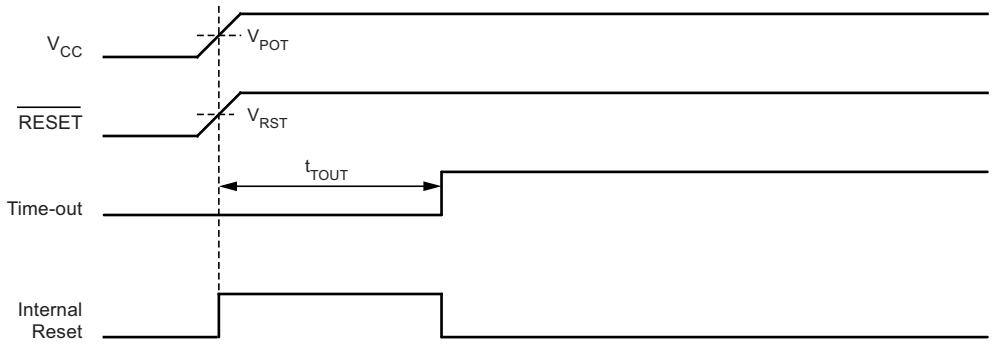
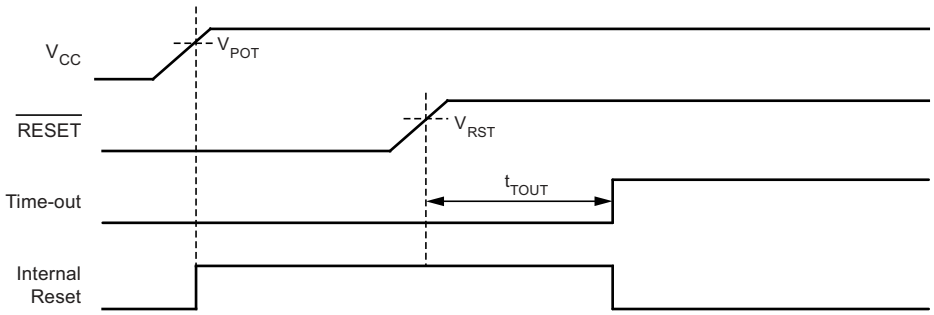


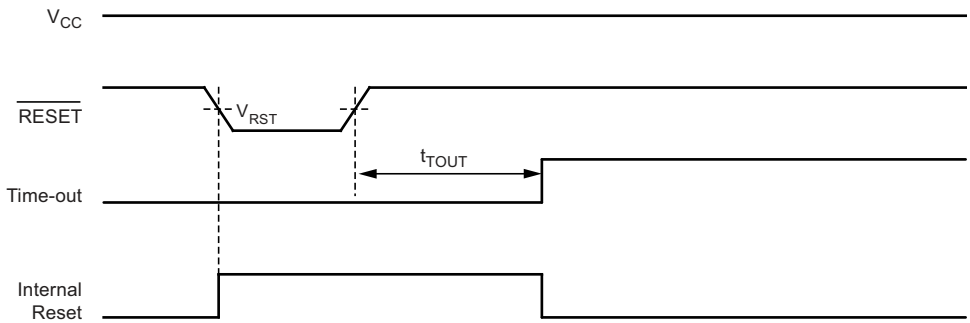
Figure 8-3. MCU Start-up, $\overline{\text{RESET}}$ Extended Externally



8.2.2 External Reset

An external reset is generated by a low level on the $\overline{\text{RESET}}$ pin. Reset pulses longer than the minimum pulse width (see Table 21-4 on page 186) will generate a reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a reset. When the applied signal reaches the Reset Threshold Voltage – V_{RST} – on its positive edge, the delay counter starts the MCU after the time-out period – t_{TOUT} – has expired. The external reset can be disabled by the RSTDISBL fuse, see Table 20-4 on page 169.

Figure 8-4. External Reset During Operation

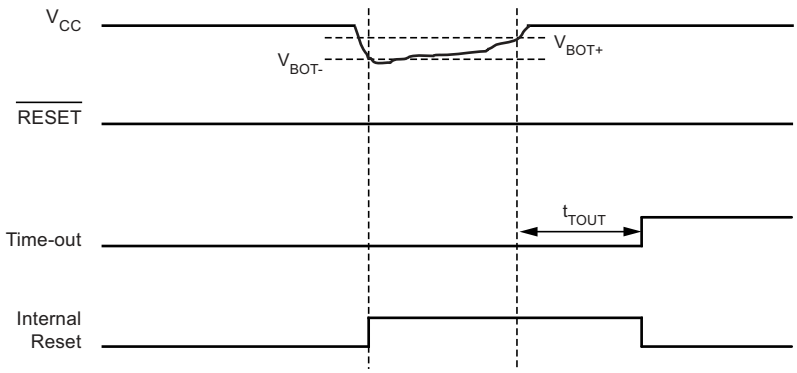


8.2.3 Brown-out Detection

Atmel® ATtiny88 has an on-chip brown-out detection (BOD) circuit for monitoring the V_{CC} level during operation by comparing it to a fixed trigger level. The trigger level for the BOD can be selected by the BODLEVEL fuses. The trigger level has a hysteresis to ensure spike free brown-out detection. The hysteresis on the detection level should be interpreted as $V_{BOT+} = V_{BOT} + V_{HYST}/2$ and $V_{BOT-} = V_{BOT} - V_{HYST}/2$. When the BOD is enabled, and V_{CC} decreases to a value below the trigger level (V_{BOT-} in Figure 8-5), the brown-out reset is immediately activated. When V_{CC} increases above the trigger level (V_{BOT+} in Figure 8-5), the delay counter starts the MCU after the time-out period t_{TOUT} has expired.

The BOD circuit will only detect a drop in V_{CC} if the voltage stays below the trigger level for longer than t_{BOD} given in Section 21.5 “System and Reset Characterizations” on page 186.

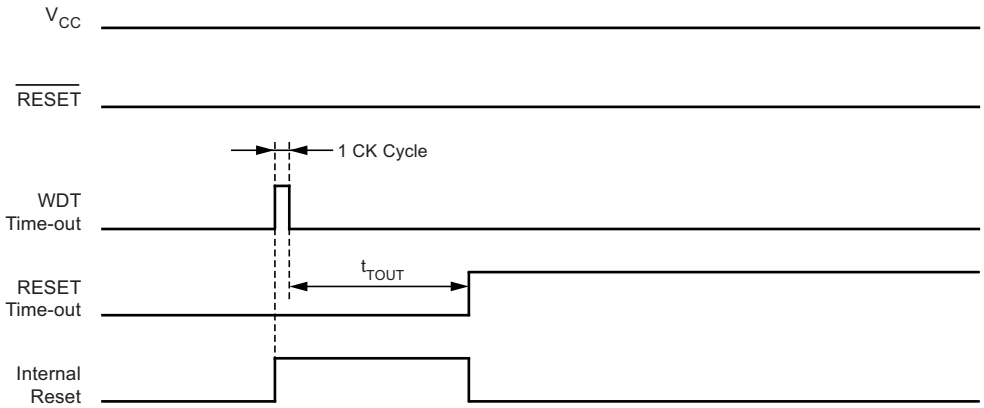
Figure 8-5. Brown-out Reset During Operation



8.2.4 Watchdog Reset

When the watchdog times out, it will generate a short reset pulse of one CK cycle duration. On the falling edge of this pulse, the delay timer starts counting the time-out period t_{TOUT}. Refer to Section 8.4 “Watchdog Timer” on page 40 for details on operation of the watchdog timer.

Figure 8-6. Watchdog System Reset During Operation



8.3 Internal Voltage Reference

Atmel® ATtiny88 features an internal bandgap reference. This reference is used for brown-out detection, and it can be used as an input to the analog comparator or the ADC.

8.3.1 Voltage Reference Enable Signals and Start-up Time

The voltage reference has a start-up time that may influence the way it should be used. The start-up time is given in Section 21.5 “System and Reset Characterizations” on page 186. To save power, the reference is not always turned on. The reference is on during the following situations:

1. When the BOD is enabled (by programming the BODLEVEL [2:0] fuses).
2. When the internal reference is connected to the analog comparator (by setting the ACBG bit in ACSR).
3. When the ADC is enabled.

Thus, when the BOD is not enabled, after setting the ACBG bit or enabling the ADC, the user must always allow the reference to start up before the output from the analog comparator or ADC is used. To reduce power consumption in power-down mode, the user can avoid the three conditions above to ensure that the reference is turned off before entering power-down mode.

8.4 Watchdog Timer

The watchdog timer is clocked from an on-chip oscillator which runs at 128kHz. By controlling the watchdog timer prescaler, the watchdog reset interval can be adjusted as shown in Table 8-3 on page 43. The WDR – watchdog reset – instruction resets the watchdog timer. The watchdog timer is also reset when it is disabled and when a chip reset occurs. Ten different clock cycle periods can be selected to determine the reset period. If the reset period expires without another watchdog reset, the Atmel ATtiny88 resets and executes from the reset vector. For timing details on the watchdog reset, refer to Table 8-3 on page 43.

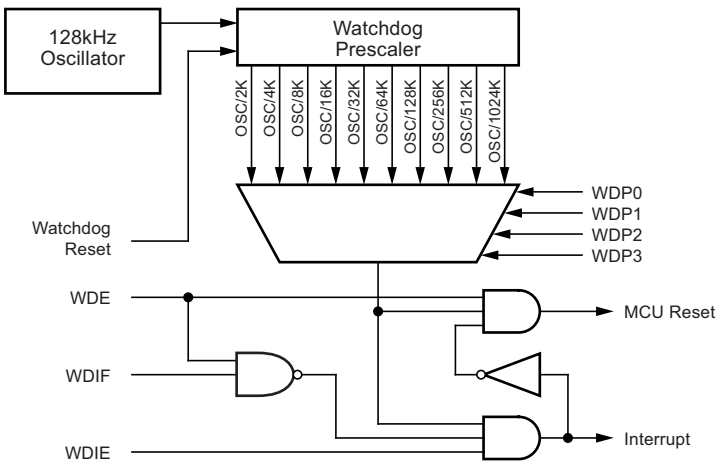
The watchdog timer can also be configured to generate an interrupt instead of a reset. This can be very helpful when using the watchdog to wake-up from power-down.

To prevent unintentional disabling of the watchdog or unintentional change of time-out period, two different safety levels are selected by the fuse WDTON as shown in Table 8-1 See Section 8.4.1 “Timed Sequences for Changing the Configuration of the Watchdog Timer” on page 41 for details.

Table 8-1. WDT Configuration as a Function of the WDTON Fuse Setting

WDTON	Safety Level	WDT Initial State	Disable WDT	Change Time-out
Unprogrammed	1	Disabled	Timed sequence	No limitations
Programmed	2	Enabled	Always enabled	Timed sequence

Figure 8-7. Watchdog Timer



8.4.1 Timed Sequences for Changing the Configuration of the Watchdog Timer

The sequence for changing configuration differs slightly between the two safety levels. Separate procedures are described for each level.

8.4.2 Safety Level 1

In this mode, the watchdog timer is initially disabled, but can be enabled by writing the WDE bit to one without any restriction. A timed sequence is needed when disabling an enabled watchdog timer. To disable an enabled watchdog timer, the following procedure must be followed:

1. In the same operation, write a logic one to WDCE and WDE. A logic one must be written to WDE regardless of the previous value of the WDE bit.
2. Within the next four clock cycles, in the same operation, write the WDE and WDP bits as desired, but with the WDCE bit cleared.

8.4.3 Safety Level 2

In this mode, the watchdog timer is always enabled, and the WDE bit will always read as one. A timed sequence is needed when changing the watchdog time-out period. To change the watchdog time-out, the following procedure must be followed:

1. In the same operation, write a logical one to WDCE and WDE. Even though the WDE always is set, the WDE must be written to one to start the timed sequence.
2. Within the next four clock cycles, in the same operation, write the WDP bits as desired, but with the WDCE bit cleare d. The value written to the WDE bit is irrelevant.

8.5 Register Description

8.5.1 MCUSR – MCU Status Register

The MCU status register provides information on which reset source caused an MCU reset.

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	WDRF	BORF	EXTRF	PORF	MCUSR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	See Bit Description				

- **Bit 7..4 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 3 – WDRF: Watchdog System Reset Flag**

This bit is set if a watchdog system reset occurs. The bit is reset by a power-on reset, or by writing a logic zero to the flag.

- **Bit 2 – BORF: Brown-out Reset Flag**

This bit is set if a brown-out reset occurs. The bit is reset by a power-on reset, or by writing a logic zero to the flag.

- **Bit 1 – EXTRF: External Reset Flag**

This bit is set if an external reset occurs. The bit is reset by a power-on reset, or by writing a logic zero to the flag.

- **Bit 0 – PORF: Power-on Reset Flag**

This bit is set if a power-on reset occurs. The bit is reset only by writing a logic zero to the flag.

To make use of the reset flags to identify a reset condition, the user should read and then reset the MCUSR as early as possible in the program. If the register is cleared before another reset occurs, the source of the reset can be found by examining the reset flags.

8.5.2 WDTCSR – Watchdog Timer Control Register

Bit	7	6	5	4	3	2	1	0	
	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	WDTCSR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	0	0	0	

- **Bit 7 – WDIF: Watchdog Interrupt Flag**

This bit is set when a time-out occurs in the watchdog timer and the watchdog timer is configured for interrupt. WDIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, WDIF is cleared by writing a logic one to the flag. When the I-bit in SREG and WDIE are set, the watchdog time-out interrupt is executed.

- **Bit 6 – WDIE: Watchdog Interrupt Enable**

When this bit is written to one and the I-bit in the status register is set, the watchdog interrupt is enabled. If WDE is cleared in combination with this setting, the watchdog timer is in interrupt mode, and the corresponding interrupt is executed if time-out in the watchdog timer occurs.

If WDE is set, the watchdog timer is in interrupt and system reset mode. The first time-out in the watchdog timer will set WDIF. Executing the corresponding interrupt vector will clear WDIE and WDIF automatically by hardware (the watchdog goes to system reset mode). This is useful for keeping the watchdog timer security while using the interrupt. To stay in interrupt and system reset mode, WDIE must be set after each interrupt. This should however not be done within the interrupt service routine itself, as this might compromise the safety-function of the watchdog system reset mode. If the interrupt is not executed before the next time-out, a system reset will be applied.

Table 8-2. Watchdog Timer Configuration

WDTON	WDE	WDIE	Mode	Action on Time-out
0	0	0	Stopped	None
0	0	1	Interrupt Mode	Interrupt
0	1	0	System Reset Mode	Reset
0	1	1	Interrupt & System Reset Mode	Interrupt, then go to System Reset Mode
1	X	X	System Reset Mode	Reset

- **Bit 4 – WDCE: Watchdog Change Enable**

This bit is used in timed sequences for changing WDE and prescaler bits. To clear the WDE bit, and/or change the prescaler bits, WDCE must be set.

Once written to one, hardware will clear WDCE after four clock cycles.

- **Bit 3 – WDE: Watchdog System Reset Enable**

WDE is overridden by WDRF in MCUSR. This means that WDE is always set when WDRF is set. To clear WDE, WDRF must be cleared first. This feature ensures multiple resets during conditions causing failure, and a safe start-up after the failure.

- **Bits 5, 2..0 – WDP3..0: Watchdog Timer Prescaler Bits 3, 2, 1 and 0**

The WDP3..0 bits determine the watchdog timer prescaling when the watchdog timer is running. The different prescaling values and their corresponding time-out periods are shown in Table 8-3 on page 43.

Table 8-3. Watchdog Timer Prescale Select

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 5.0V
0	0	0	0	2K (2048) cycles	16 ms
0	0	0	1	4K (4096) cycles	32 ms
0	0	1	0	8K (8192) cycles	64 ms
0	0	1	1	16K (16384) cycles	0.125 s
0	1	0	0	32K (32768) cycles	0.25 s
0	1	0	1	64K (65536) cycles	0.5 s
0	1	1	0	128K (131072) cycles	1.0 s
0	1	1	1	256K (262144) cycles	2.0 s
1	0	0	0	512K (524288) cycles	4.0 s
1	0	0	1	1024K (1048576) cycles	8.0 s
1	0	1	0	Reserved (1)	
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

Note: 1. If selected, one of the valid settings below 0b1010 will be used.

9. Interrupts

This section describes the specifics of interrupt handling in Atmel® ATtiny88. For a general explanation of the AVR® interrupt handling, refer to Section 4.8 “Reset and Interrupt Handling” on page 14.

9.1 Interrupt Vectors

Table 9-1. Reset and Interrupt Vectors in ATtiny88

Vector No.	Program Address	Source	Interrupt Definition
1	0x000	RESET	External/power-on/brown-out/watchdog reset
2	0x001	INT0	External interrupt request 0
3	0x002	INT1	External interrupt request 1
4	0x003	PCINT0	Pin change interrupt request 0
5	0x004	PCINT1	Pin change interrupt request 1
6	0x005	PCINT2	Pin change interrupt request 2
7	0x006	PCINT3	Pin Change Interrupt Request 3
8	0x007	WDT	Watchdog time-out interrupt
9	0x008	TIMER1_CAPT	Timer/Counter1 capture event
10	0x009	TIMER1_COMPA	Timer/Counter1 compare match A
11	0x00A	TIMER1_COMPB	Timer/Counter1 compare match B
12	0x00B	TIMER1_OVF	Timer/Counter1 overflow
13	0x00C	TIMER0_COMPA	Timer/Counter0 compare match A
14	0x00D	TIMER0_COMPB	Timer/Counter0 compare match B
15	0x00E	TIMER0_OVF	Timer/Counter0 overflow
16	0x00F	SPI_STC	SPI serial transfer complete
17	0x010	ADC	ADC conversion complete
18	0x011	EE_RDY	EEPROM ready
19	0x012	ANA_COMP	Analog comparator
20	0x013	TWI	2-wire serial interface

The most typical and general program setup for the Reset and Interrupt Vector Addresses in ATtiny88 is:

```
Address      Labels Code                Comments
0x000               rjmp   RESET        ; Reset Handler
0x001               rjmp   INT0         ; IRQ0 Handler
0x002               rjmp   INT1         ; IRQ1 Handler
0x003               rjmp   PCINT0       ; PCINT0 Handler
0x004               rjmp   PCINT1       ; PCINT1 Handler
0x005               rjmp   PCINT2       ; PCINT2 Handler
0x006               rjmp   PCINT3       ; PCINT3 Handler
0x007               rjmp   WDT          ; Watchdog Timer Handler
0x008               rjmp   TIMER1_CAPT  ; Timer1 Capture Handler
0x009               rjmp   TIMER1_COMPA ; Timer1 Compare A Handler
0x00A               rjmp   TIMER1_COMPB ; Timer1 Compare B Handler
0x00B               rjmp   TIMER1_OVF   ; Timer1 Overflow Handler
0x00C               rjmp   TIMER0_COMPA ; Timer0 Compare A Handler
0x00D               rjmp   TIMER0_COMPB ; Timer0 Compare B Handler
0x00E               rjmp   TIMER0_OVF   ; Timer0 Overflow Handler
0x00F               rjmp   SPI_STC      ; SPI Transfer Complete Handler
0x010               rjmp   ADC          ; ADC Conversion Complete Handler
0x011               rjmp   EE_RDY       ; EEPROM Ready Handler
0x012               rjmp   ANA_COMP     ; Analog Comparator Handler
0x013               rjmp   TWI          ; 2-wire Serial Interface Handler;
0x014        RESET: ldi    r16, high(RAMEND); Main program start
0x015               out    SPH,r16      ; Set Stack Pointer to top of RAM
0x016               ldi    r16, low(RAMEND)
0x017               out    SPL,r16
0x018               sei                 ; Enable interrupts
0x019               <inst> xxx
                ...    ...    ...          ...
```

9.2 External Interrupts

The external interrupts are triggered by the INT0 and INT1 pins or any of the PCINT27..0 pins. Observe that, if enabled, the interrupts will trigger even if the INT0 and INT1 or PCINT27..0 pins are configured as outputs. This feature provides a way of generating a software interrupt, as follows.

- Pin change interrupt PCI3 triggers if a pin in PCINT27..24 is toggled while enabled
- Pin change interrupt PCI2 triggers if a pin in PCINT23..16 is toggled while enabled
- Pin change interrupt PCI1 triggers if a pin in PCINT15..8 is toggled while enabled
- Pin change interrupt PCI0 triggers if a pin in PCINT7..0 is toggled while enabled

The PCMSK3, PCMSK2, PCMSK1 and PCMSK0 registers control which pins contribute to the pin change interrupts. Pin change interrupts on PCINT27..0 are detected asynchronously. This means that these interrupts can be used for waking the part also from sleep modes other than idle mode.

The INT0 and INT1 interrupts can be triggered by a falling or rising edge, or a low level. This is configured as described in Section 9.3.1 “EICRA – External Interrupt Control Register A” on page 47. When INT0 or INT1 interrupts are enabled and are configured as level triggered, the interrupts will trigger as long as the corresponding pin is held low. Note that recognition of falling or rising edge interrupts on INT0 or INT1 requires the presence of an I/O clock, described in Section 6.1 “Clock Systems and their Distribution” on page 25.

9.2.1 Low Level Interrupt

Low level interrupts on INT0 and INT1 are detected asynchronously. This means that the interrupt sources can be used for waking the part also from sleep modes other than Idle (the I/O clock is halted in all sleep modes except idle mode).

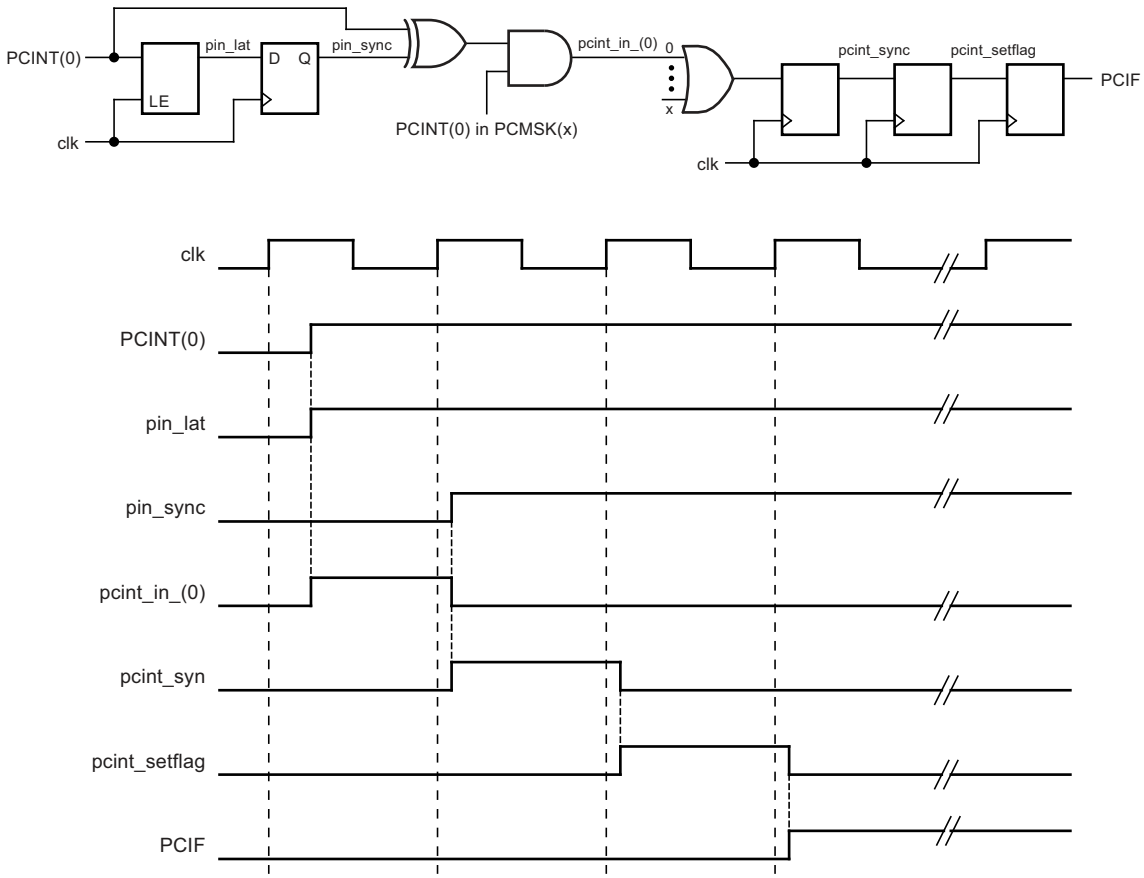
Note that if a level triggered interrupt is used for wake-up from power-down, the required level must be held long enough for the MCU to complete the wake-up to trigger the level interrupt. If the level disappears before the end of the start-up time, the MCU will still wake up, but no interrupt will be generated. The start-up time is defined by the SUT and CKSEL fuses as described in Section 6. “System Clock and Clock Options” on page 25.

If the low level on the interrupt pin is removed before the device has woken up then program execution will not be diverted to the interrupt service routine but continue from the instruction following the SLEEP command.

9.2.2 Pin Change Interrupt Timing

An example of timing of a pin change interrupt is shown in Figure 9-1.

Figure 9-1. Timing of Pin Change Interrupts



9.3 Register Description

9.3.1 EICRA – External Interrupt Control Register A

The external interrupt control register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..4 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bits 3, 2 – ISC11, ISC10: Interrupt Sense Control 1 Bit 1 and Bit 0**

The external interrupt 1 is activated by the external pin INT1 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT1 pin that activate the interrupt are defined in Table 9-2. The value on the INT1 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

Table 9-2. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

- **Bits 1, 0 – ISC01, ISC00: Interrupt Sense Control 0 Bit 1 and Bit 0**

The external interrupt 0 is activated by the external pin INT0 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT0 pin that activate the interrupt are defined in Table 9-3. The value on the INT0 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

Table 9-3. Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

9.3.2 EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..2 – Res: Reserved Bits**

These bits are unused in Atmel® ATtiny88, and will always read as zero.

- **Bit 1 – INT1: External Interrupt Request 1 Enable**

When the INT1 bit is set (one) and the I-bit in the status register (SREG) is set (one), the external pin interrupt is enabled. The interrupt sense control bits (ISC11 and ISC10) in the external interrupt control register A (EICRA) define whether the external interrupt is activated on rising and/or falling edge of the INT1 pin or level sensed. Activity on the pin will cause an interrupt request even if INT1 is configured as an output. The corresponding interrupt of external interrupt request 1 is executed from the INT1 interrupt vector.

- **Bit 0 – INT0: External Interrupt Request 0 Enable**

When the INT0 bit is set (one) and the I-bit in the status register (SREG) is set (one), the external pin interrupt is enabled. The interrupt sense control bits (ISC01 and ISC00) in the external interrupt control register A (EICRA) define whether the external interrupt is activated on rising and/or falling edge of the INT0 pin or level sensed. Activity on the pin will cause an interrupt request even if INT0 is configured as an output. The corresponding interrupt of external interrupt request 0 is executed from the INT0 interrupt vector.

9.3.3 EIFR – External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	INTF1	INTF0	EIFR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7..2 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 1 – INTF1: External Interrupt Flag 1**

When an edge or logic change on the INT1 pin triggers an interrupt request, INTF1 becomes set (one). If the I-bit in SREG and the INT1 bit in EIMSK are set (one), the MCU will jump to the corresponding interrupt vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. This flag is always cleared when INT1 is configured as a level interrupt.

- **Bit 0 – INTF0: External Interrupt Flag 0**

When an edge or logic change on the INT0 pin triggers an interrupt request, INTF0 becomes set (one). If the I-bit in SREG and the INT0 bit in EIMSK are set (one), the MCU will jump to the corresponding interrupt vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. This flag is always cleared when INT0 is configured as a level interrupt.

9.3.4 PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	PCIE3	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7..4 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 3 – PCIE3: Pin Change Interrupt Enable 3**

When the PCIE3 bit is set (one) and the I-bit in the status register (SREG) is set (one), pin change interrupt 3 is enabled. Any change on any enabled PCINT27..24 pin will cause an interrupt. The corresponding interrupt of pin change interrupt request is executed from the PCI3 interrupt vector. PCINT27..24 pins are enabled individually by the PCMSK3 register.

- **Bit 2 – PCIE2: Pin Change Interrupt Enable 2**

When the PCIE2 bit is set (one) and the I-bit in the status register (SREG) is set (one), pin change interrupt 2 is enabled. Any change on any enabled PCINT23..16 pin will cause an interrupt. The corresponding interrupt of pin change interrupt request is executed from the PCI2 interrupt vector. PCINT23..16 pins are enabled individually by the PCMSK2 register.

- **Bit 1 – PCIE1: Pin Change Interrupt Enable 1**

When the PCIE1 bit is set (one) and the I-bit in the status register (SREG) is set (one), pin change interrupt 1 is enabled. Any change on any enabled PCINT15..8 pin will cause an interrupt. The corresponding interrupt of pin change interrupt request is executed from the PCI1 interrupt vector. PCINT15..8 pins are enabled individually by the PCMSK1 register.

- **Bit 0 – PCIE0: Pin Change Interrupt Enable 0**

When the PCIE0 bit is set (one) and the I-bit in the status register (SREG) is set (one), pin change interrupt 0 is enabled. Any change on any enabled PCINT7..0 pin will cause an interrupt. The corresponding interrupt of pin change interrupt request is executed from the PCI0 interrupt vector. PCINT7..0 pins are enabled individually by the PCMSK0 register.

9.3.5 PCIFR – Pin Change Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	**PCIF3**	**PCIF2**	**PCIF1**	**PCIF0**	**PCIFR**
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7..4 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 3 – PCIF3: Pin Change Interrupt Flag 3**

When a logic change on any PCINT27..24 pin triggers an interrupt request, PCIF3 becomes set (one). If the I-bit in SREG and the PCIE3 bit in PCICR are set (one), the MCU will jump to the corresponding interrupt vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

- **Bit 2 – PCIF2: Pin Change Interrupt Flag 2**

When a logic change on any PCINT23..16 pin triggers an interrupt request, PCIF2 becomes set (one). If the I-bit in SREG and the PCIE2 bit in PCICR are set (one), the MCU will jump to the corresponding interrupt vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

- **Bit 1 – PCIF1: Pin Change Interrupt Flag 1**

When a logic change on any PCINT15..8 pin triggers an interrupt request, PCIF1 becomes set (one). If the I-bit in SREG and the PCIE1 bit in PCICR are set (one), the MCU will jump to the corresponding interrupt vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

- **Bit 0 – PCIF0: Pin Change Interrupt Flag 0**

When a logic change on any PCINT7..0 pin triggers an interrupt request, PCIF0 becomes set (one). If the I-bit in SREG and the PCIE0 bit in PCICR are set (one), the MCU will jump to the corresponding interrupt vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

9.3.6 PCMSK3 – Pin Change Mask Register 3

Bit	7	6	5	4	3	2	1	0	
	-	-	-	-	PCINT27	PCINT26	PCINT25	PCINT24	PCMSK3
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7..4 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 3..0 – PCINT27..24: Pin Change Enable Mask 27..24**

Each PCINT27..24-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT27..24 is set and the PCIE3 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT27..24 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

9.3.7 PCMSK2 – Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7..0 – PCINT23..16: Pin Change Enable Mask 23..16**

Each PCINT23..16-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT23..16 is set and the PCIE2 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT23..16 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

9.3.8 PCMSK1 – Pin Change Mask Register 1

Bit	7	6	5	4	3	2	1	0	
	PCINT15	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7..0 – PCINT15..8: Pin Change Enable Mask 15..8**

Each PCINT15..8-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT15..8 is set and the PCIE1 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT15..8 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

9.3.9 PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7..0 – PCINT7..0: Pin Change Enable Mask 7..0**

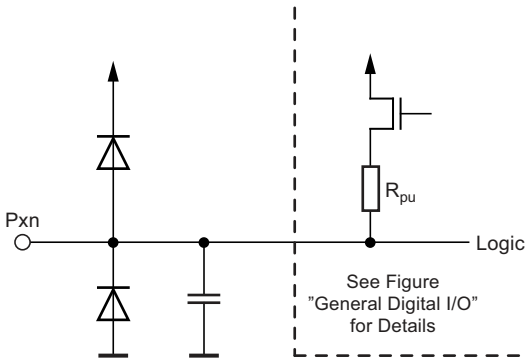
Each PCINT7..0 bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is set and the PCIE0 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

10. I/O-Ports

10.1 Introduction

All AVR® ports have true read-modify-write functionality when used as general digital I/O ports. This means that the direction of one port pin can be changed without unintentionally changing the direction of any other pin with the SBI and CBI instructions. The same applies when changing drive value (if configured as output) or enabling/disabling of pull-up resistors (if configured as input). The pin driver is strong enough to drive LED displays directly. All port pins have individually selectable pull-up resistors with a supply-voltage invariant resistance. All I/O pins have protection diodes to both V_{CC} and ground as indicated in Figure 10-1. Refer to Section 21. “Electrical Characteristics” on page 183 for a complete list of parameters.

Figure 10-1. I/O Pin Equivalent Schematic



All registers and bit references in this section are written in general form. A lower case “x” represents the numbering letter for the port, and a lower case “n” represents the bit number. However, when using the register or bit defines in a program, the precise form must be used. For example, PORTB3 for bit no. 3 in Port B, here documented generally as PORTxn. The physical I/O registers and bit locations are listed in Section 10.4 “Register Description” on page 65.

Three I/O memory address locations are allocated for each port, one each for the data register – PORTx, data direction register – DDRx, and the port input pins – PINx. The port Input pins I/O location is read only, while the data register and the data direction register are read/write. However, writing a logic one to a bit in the PINx register, will result in a toggle in the corresponding bit in the data register. In addition, the pull-up disable – PUD bit in MCUCR disables the pull-up function for all pins in all ports when set.

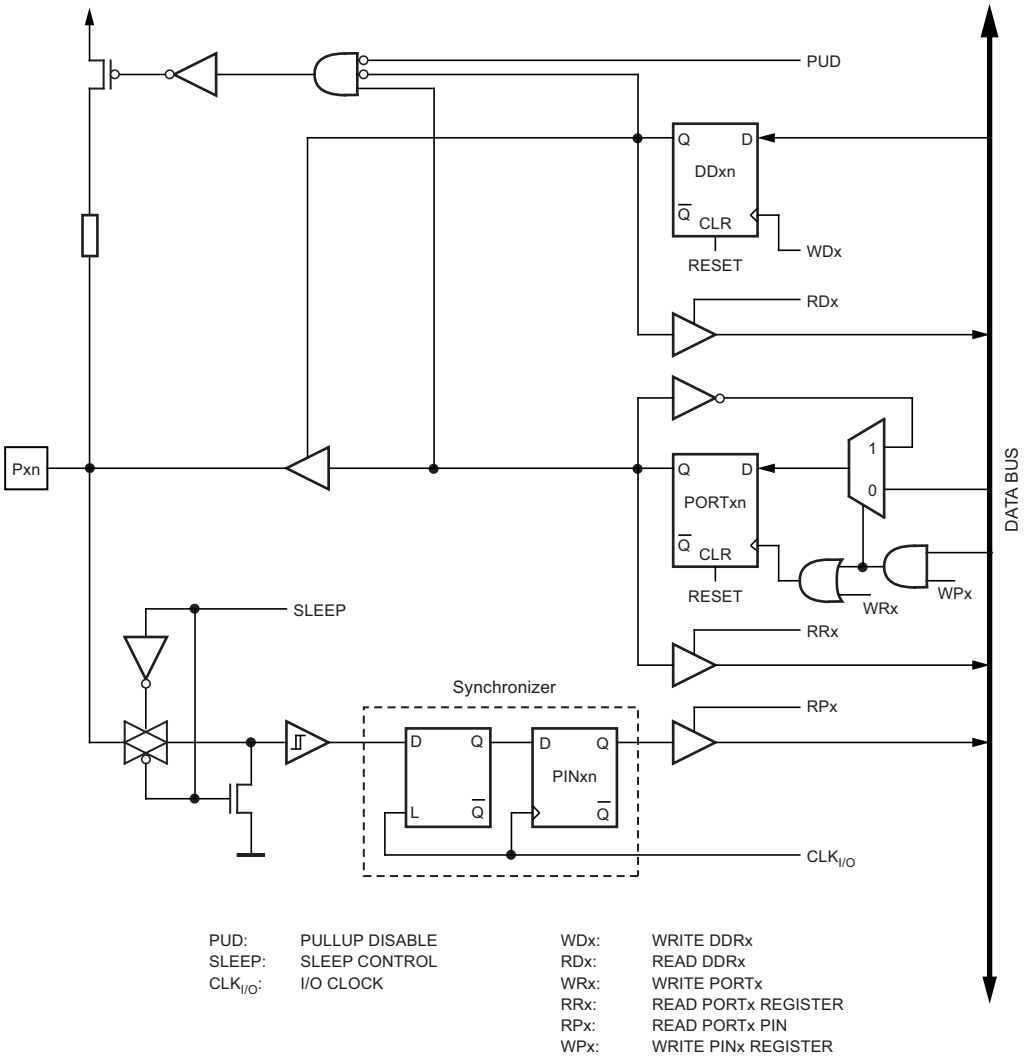
Using the I/O port as general digital I/O is described in Section 10.2 “Ports as General Digital I/O” on page 52. Most port pins are multiplexed with alternate functions for the peripheral features on the device. How each alternate function interferes with the port pin is described in Section 10.3 “Alternate Port Functions” on page 56. Refer to the individual module sections for a full description of the alternate functions.

Note that enabling the alternate function of some of the port pins does not affect the use of the other pins in the port as general digital I/O.

10.2 Ports as General Digital I/O

The ports are bi-directional I/O ports with optional internal pull-ups. Figure 10-2 shows a functional description of one I/O-port pin, here generically called Pxn.

Figure 10-2. General Digital I/O(1)



Note: 1. WRx, WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

10.2.1 Configuring the Pin

Each port pin consists of three register bits: DDxn, PORTxn, and PINxn. As shown in Section 10.4 “Register Description” on page 65, the DDxn bits are accessed at the DDRx I/O address, the PORTxn bits at the PORTx I/O address, and the PINxn bits at the PINx I/O address.

The DDxn bit in the DDRx register selects the direction of this pin. If DDxn is written logic one, Pxn is configured as an output pin. If DDxn is written logic zero, Pxn is configured as an input pin.

If PORTxn is written logic one when the pin is configured as an input pin, the pull-up resistor is activated. To switch the pull-up resistor off, PORTxn has to be written logic zero or the pin has to be configured as an output pin. The port pins are tri-stated when reset condition becomes active, even if no clocks are running.

If PORTxn is written logic one when the pin is configured as an output pin, the port pin is driven high (one). If PORTxn is written logic zero when the pin is configured as an output pin, the port pin is driven low (zero).

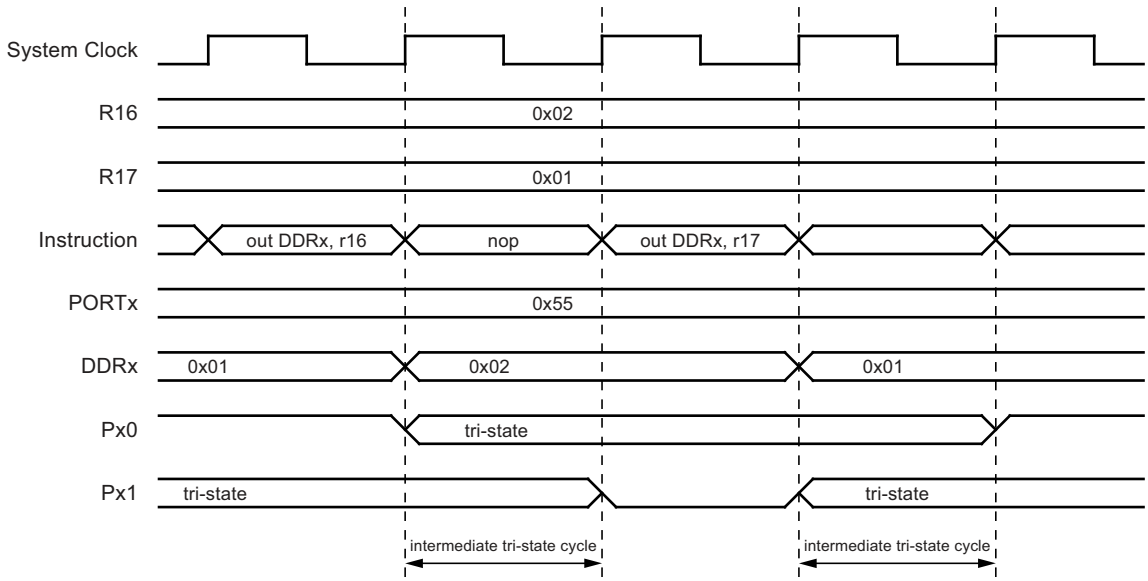
10.2.2 Toggling the Pin

Writing a logic one to PINxn toggles the value of PORTxn, independent on the value of DDRxn. Note that the SBI instruction can be used to toggle one single bit in a port.

10.2.3 Break-Before-Make Switching

In the break-before-make mode when switching the DDRxn bit from input to output an immediate tri-state period lasting one system clock cycle is introduced as indicated in Figure 10-3. For example, if the system clock is 4MHz and the DDRxn is written to make an output, the immediate tri-state period of 250 ns is introduced, before the value of PORTxn is seen on the port pin. To avoid glitches it is recommended that the maximum DDRxn toggle frequency is two system clock cycles. The break-before-make is a port-wise mode and it is activated by the port-wise BBMx enable bits. For details on BBMx bits, see Section 10.4.2 “PORTCR – Port Control Register” on page 65. When switching the DDRxn bit from output to input there is no immediate tri-state period introduced.

Figure 10-3. Break Before Make, Switching between Input and Output



10.2.4 Switching Between Input and Output

When switching between tri-state ({DDxn, PORTxn} = 0b00) and output high ({DDxn, PORTxn} = 0b11), an intermediate state with either pull-up enabled {DDxn, PORTxn} = 0b01) or output low ({DDxn, PORTxn} = 0b10) must occur. Normally, the pull-up enabled state is fully acceptable, as a high-impedant environment will not notice the difference between a strong high driver and a pull-up. If this is not the case, the PUD bit in the MCUCR register can be set to disable all pull-ups in all ports.

Switching between input with pull-up and output low generates the same problem. The user must use either the tri-state ({DDxn, PORTxn} = 0b00) or the output high state ({DDxn, PORTxn} = 0b11) as an intermediate step.

Table 10-1 summarizes the control signals for the pin value.

Table 10-1. Port Pin Configurations

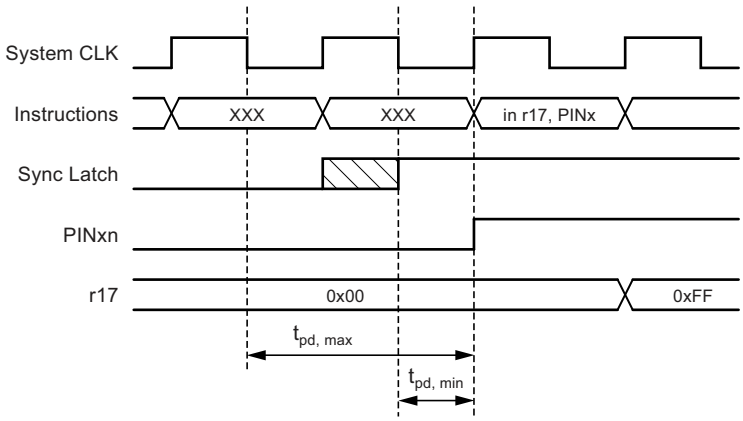
DDxn	PORTxn	PUD (in MCUCR) [1]	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output low (sink)
1	1	X	Output	No	Output high (source)

Note: 1. Or port-wise PUDx bit in PORTCR register.

10.2.5 Reading the Pin Value

Independent of the setting of data direction bit DDxn, the port pin can be read through the PINxn register bit. As shown in Figure 10-2 on page 52, the PINxn register bit and the preceding latch constitute a synchronizer. This is needed to avoid metastability if the physical pin changes value near the edge of the internal clock, but it also introduces a delay. Figure 10-4 shows a timing diagram of the synchronization when reading an externally applied pin value. The maximum and minimum propagation delays are denoted $t_{pd,max}$ and $t_{pd,min}$ respectively.

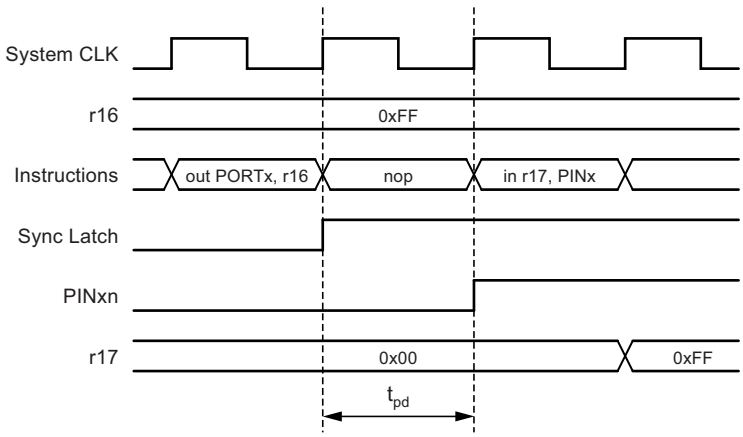
Figure 10-4. Synchronization when Reading an Externally Applied Pin Value



Consider the clock period starting shortly after the first falling edge of the system clock. The latch is closed when the clock is low, and goes transparent when the clock is high, as indicated by the shaded region of the “SYNC LATCH” signal. The signal value is latched when the system clock goes low. It is clocked into the PINxn register at the succeeding positive clock edge. As indicated by the two arrows tpd,max and tpd,min, a single signal transition on the pin will be delayed between ½ and 1½ system clock period depending upon the time of assertion.

When reading back a software assigned pin value, a nop instruction must be inserted as indicated in Figure 10-5. The out instruction sets the “SYNC LATCH” signal at the positive edge of the clock. In this case, the delay tpd through the synchronizer is 1 system clock period.

Figure 10-5. Synchronization when Reading a Software Assigned Pin Value



The following code example shows how to set port B pins 0 and 1 high, 2 and 3 low, and define the port pins from 4 to 7 as input with pull-ups assigned to port pins 6 and 7. The resulting pin values are read back again, but as previously discussed, a nop instruction is included to be able to read back the value recently assigned to some of the pins.

Assembly Code Example[1]

```
...
; Define pull-ups and set outputs high
; Define directions for port pins
ldi     r16,(1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0)
ldi     r17,(1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0)
out     PORTB,r16
out     DDRB,r17
; Insert nop for synchronization
nop
; Read port pins
in      r16,PINB
...
```

Note: 1. For the assembly program, two temporary registers are used to minimize the time from pull-ups are set on pins 0, 1, 6, and 7, until the direction bits are correctly set, defining bit 2 and 3 as low and redefining bits 0 and 1 as strong high drivers.

C Code Example

```
unsigned char i;
        ...
        /* Define pull-ups and set outputs high */
        /* Define directions for port pins */
        PORTB = (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0);
        DDRB = (1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0);
        /* Insert nop for synchronization*/
        __no_operation();
        /* Read port pins */
        i = PINB;
        ...
```

10.2.6 Digital Input Enable and Sleep Modes

As shown in Figure 10-2 on page 52, the digital input signal can be clamped to ground at the input of the schmitt trigger. The signal denoted SLEEP in the figure, is set by the MCU sleep controller in power-down mode to avoid high power consumption if some input signals are left floating, or have an analog signal level close to $V_{CC}/2$.

SLEEP is overridden for port pins enabled as external interrupt pins. If the external interrupt request is not enabled, SLEEP is active also for these pins. SLEEP is also overridden by various other alternate functions as described in Section 10.3 “Alternate Port Functions” on page 56.

If a logic high level (“one”) is present on an asynchronous external interrupt pin configured as “interrupt on rising edge, falling edge, or any logic change on pin” while the external interrupt is *not* enabled, the corresponding external interrupt flag will be set when resuming from the above mentioned Sleep mode, as the clamping in these sleep mode produces the requested logic change.

10.2.7 Unconnected Pins

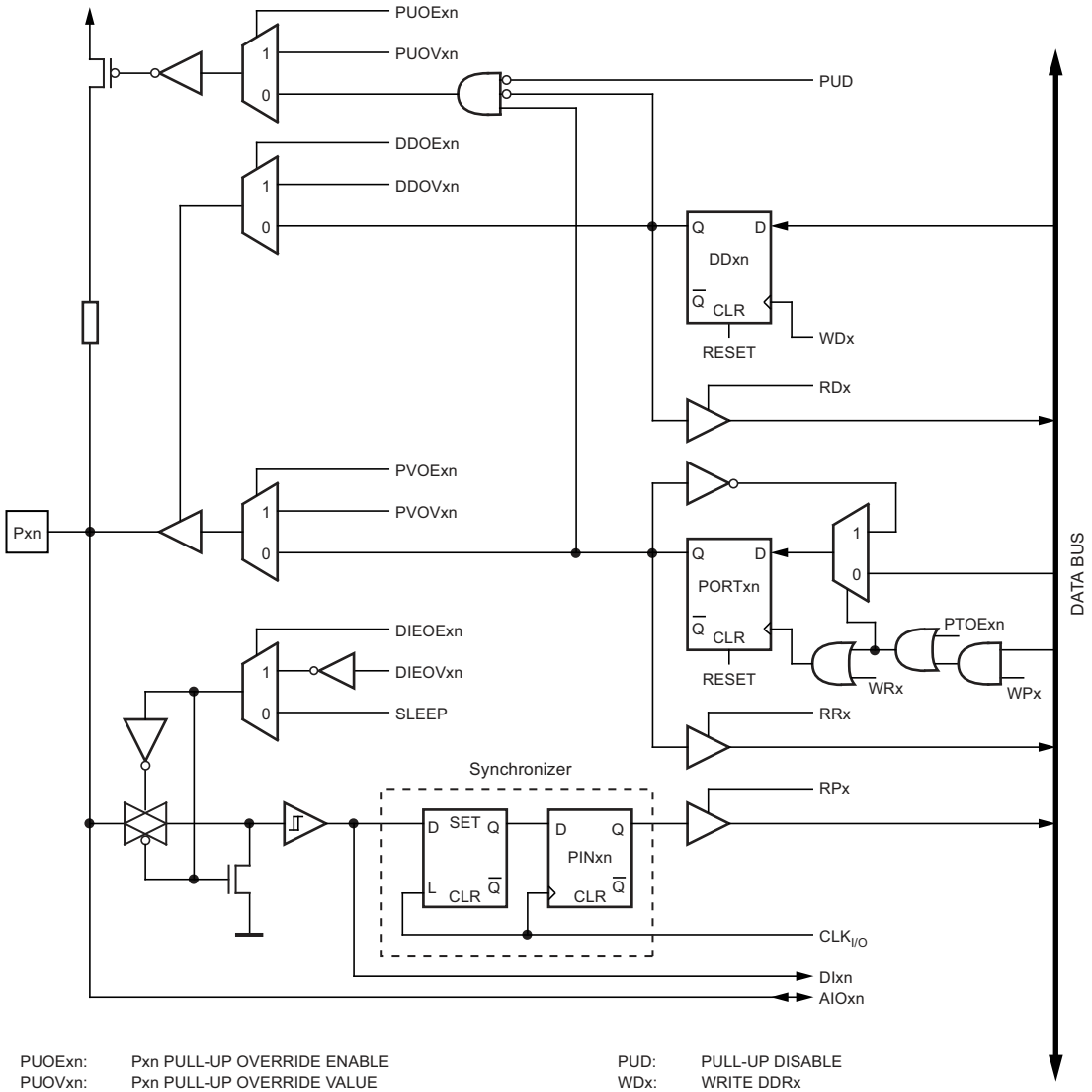
If some pins are unused, it is recommended to ensure that these pins have a defined level. Even though most of the digital inputs are disabled in the deep sleep modes as described above, floating inputs should be avoided to reduce current consumption in all other modes where the digital inputs are enabled (reset, active mode and idle mode).

The simplest method to ensure a defined level of an unused pin, is to enable the internal pull-up. In this case, the pull-up will be disabled during reset. If low power consumption during reset is important, it is recommended to use an external pull-up or pull-down. Connecting unused pins directly to V_{CC} or GND is not recommended, since this may cause excessive currents if the pin is accidentally configured as an output.

10.3 Alternate Port Functions

Most port pins have alternate functions in addition to being general digital I/Os. Figure 10-6 shows how the port pin control signals from the simplified Figure 10-2 on page 52 can be overridden by alternate functions. The overriding signals may not be present in all port pins, but the figure serves as a generic description applicable to all port pins in the AVR® microcontroller family.

Figure 10-6. Alternate Port Functions(1)



PUOExn:	Pxn PULL-UP OVERRIDE ENABLE	PUD:	PULL-UP DISABLE
PUOVxn:	Pxn PULL-UP OVERRIDE VALUE	WDx:	WRITE DDRx
DDOExn:	Pxn DATA DIRECTION OVERRIDE ENABLE	RDx:	READ DDRx
DDOVxn:	Pxn DATA DIRECTION OVERRIDE VALUE	RRx:	READ PORTx REGISTER
PVOExn:	Pxn PORT VALUE OVERRIDE ENABLE	WRx:	WRITE PORTx
PVOVxn:	Pxn PORT VALUE OVERRIDE VALUE	RPx:	READ PORTx PIN
DIEOExn:	Pxn DIGITAL INPUT ENABLE OVERRIDE ENABLE	WPx:	WRITE PINx
DIEOVxn:	Pxn DIGITAL INPUT ENABLE OVERRIDE VALUE	$CLK_{I/O}$:	I/O CLOCK
SLEEP:	SLEEP CONTROL	DIxn:	DIGITAL INPUT PIN n ON PORTx
PTOExn:	Pxn, PORT TOGGLE OVERRIDE ENABLE	AIOxn:	ANALOG INPUT/OUTPUT PIN n ON PORTx

Note: 1. WRx, WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports. All other signals are unique for each pin.

Table 10-2 summarizes the function of the overriding signals. The pin and port indexes from Figure 10-6 on page 56 are not shown in the succeeding tables. The overriding signals are generated internally in the modules having the alternate function.

Table 10-2. Generic Description of Overriding Signals for Alternate Functions

Signal Name	Full Name	Description
PUOE	Pull-up override enable	If this signal is set, the pull-up enable is controlled by the PUOV signal. If this signal is cleared, the pull-up is enabled when {DDxn, PORTxn, PUD} = 0b010.
PUOV	Pull-up override value	If PUOE is set, the pull-up is enabled/disabled when PUOV is set/cleared, regardless of the setting of the DDxn, PORTxn, and PUD register bits.
DDOE	Data direction override enable	If this signal is set, the output driver enable is controlled by the DDOV signal. If this signal is cleared, the output driver is enabled by the DDxn register bit.
DDOV	Data direction override value	If DDOE is set, the output driver is enabled/disabled when DDOV is set/cleared, regardless of the setting of the DDxn register bit.
PVOE	Port value override enable	If this signal is set and the output driver is enabled, the port value is controlled by the PVOV signal. If PVOE is cleared, and the output driver is enabled, the port value is controlled by the PORTxn register bit.
PVOV	Port value override value	If PVOE is set, the port value is set to PVOV, regardless of the setting of the PORTxn register bit.
PTOE	Port toggle override enable	If PTOE is set, the PORTxn register bit is inverted.
DIEOE	Digital input enable override enable	If this bit is set, the digital input enable is controlled by the DIEOV signal. If this signal is cleared, the digital input enable is determined by MCU state (normal mode, sleep mode).
DIEOV	Digital input enable override value	If DIEOE is set, the digital input is enabled/disabled when DIEOV is set/cleared, regardless of the MCU state (normal mode, sleep mode).
DI	Digital input	This is the digital input to alternate functions. In the figure, the signal is connected to the output of the schmitt trigger but before the synchronizer. Unless the digital input is used as a clock source, the module with the alternate function will use its own synchronizer.
AIO	Analog input/output	This is the analog input/output to/from alternate functions. The signal is connected directly to the pad, and can be used bi-directionally.

The following subsections shortly describe the alternate functions for each port, and relate the overriding signals to the alternate function. Refer to the alternate function description for further details.

10.3.1 Alternate Functions of Port A

The port A pins with alternate functions are shown in Table 10-3.

Table 10-3. Port A Pins Alternate Functions

Port Pin	Alternate Function
PA3	PCINT27 (pin change interrupt 27)
PA2	PCINT26 (pin change interrupt 26)
PA1	ADC7 (ADC input channel 7) PCINT25 (pin change interrupt 25)
PA0	ADC6 (ADC input channel 6) PCINT24 (pin change interrupt 24)

The alternate pin configuration is as follows:

- **PCINT27 – Port A, Bit 3**

PCINT27: pin change interrupt source 27.

- **PCINT26 – Port A, Bit 2**

PCINT26: pin change interrupt source 26.

- **ADC7/PCINT25 – Port A, Bit 1**

ADC7: PA1 can be used as ADC input channel 7.

PCINT25: pin change interrupt source 25.

- **ADC6/PCINT24 – Port A, Bit 0**

ADC6: PA0 can be used as ADC input channel 6.

PCINT24: Pin change interrupt source 24.

Table 10-4 relate the alternate functions of port A to the overriding signals shown in Figure 10-6 on page 56.

Table 10-4. Overriding Signals for Alternate Functions in PA3..PA0

Signal Name	PA3/PCINT27	PA2/PCINT26	PA1/ADC7/PCINT25	PA0/ADC6/PCINT24
PUOE	0	0	0	0
PUO	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
DIEOE	PCINT27 × PCIE3	PCINT26 × PCIE3	PCINT25 × PCIE3 + ADC7D	PCINT24 × PCIE3 + ADC6D
DIEOV	1	1	1	1
DI	PCINT27 INPUT	PCINT26 INPUT	PCINT25 INPUT	PCINT24 INPUT
AIO	–	–	ADC7 INPUT	ADC6 INPUT

10.3.2 Alternate Functions of Port B

The port B pins with alternate functions are shown in Table 10-5.

Table 10-5. Port B Pins Alternate Functions

Port Pin	Alternate Functions
PB7	PCINT7 (pin change interrupt 7)
PB6	CLKI (external clock input) PCINT6 (pin change interrupt 6)
PB5	SCK (SPI bus master clock input) PCINT5 (pin change interrupt 5)
PB4	MISO (SPI bus master input/slave output) PCINT4 (pin change interrupt 4)
PB3	MOSI (SPI bus master output/slave input) PCINT3 (pin change interrupt 3)
PB2	$\overline{SS}$ (SPI bus master slave select) OC1B (Timer/Counter1 output compare match B output) PCINT2 (pin change interrupt 2)
PB1	OC1A (Timer/Counter1 output compare match A output) PCINT1 (pin change interrupt 1)
PB0	ICP1 (Timer/Counter1 input capture input) CLKO (divided system clock output) PCINT0 (pin change interrupt 0)

The alternate pin configuration is as follows:

- **PCINT7 – Port B, Bit 7**

PCINT7: Pin change interrupt source 7. The PB7 pin can serve as an external interrupt source.

If PB7 is used as a clock pin, DDB7, PORTB7 and PINB7 will all read 0.

- **CLKI/PCINT6 – Port B, Bit 6**

CLKI: External clock input. When used as a clock pin, the pin can not be used as an I/O pin.

PCINT6: Pin change interrupt source 6. The PB6 pin can serve as an external interrupt source.

If PB6 is used as a clock pin, DDB6, PORTB6 and PINB6 will all read 0.

- **SCK/PCINT5 – Port B, Bit 5**

SCK: master clock output, slave clock input pin for SPI channel. When the SPI is enabled as a slave, this pin is configured as an input regardless of the setting of DDB5. When the SPI is enabled as a master, the data direction of this pin is controlled by DDB5. When the pin is forced by the SPI to be an input, the pull-up can still be controlled by the PORTB5 bit.

PCINT5: Pin change interrupt source 5. The PB5 pin can serve as an external interrupt source.

- **MISO/PCINT4 – Port B, Bit 4**

MISO: Master data input, slave data output pin for SPI channel. When the SPI is enabled as a master, this pin is configured as an input regardless of the setting of DDB4. When the SPI is enabled as a slave, the data direction of this pin is controlled by DDB4. When the pin is forced by the SPI to be an input, the pull-up can still be controlled by the PORTB4 bit.
PCINT4: Pin change interrupt source 4. The PB4 pin can serve as an external interrupt source.

- **MOSI/PCINT3 – Port B, Bit 3**

MOSI: SPI master data output, slave data input for SPI channel. When the SPI is enabled as a slave, this pin is configured as an input regardless of the setting of DDB3. When the SPI is enabled as a master, the data direction of this pin is controlled by DDB3. When the pin is forced by the SPI to be an input, the pull-up can still be controlled by the PORTB3 bit.

PCINT3: Pin change interrupt source 3. The PB3 pin can serve as an external interrupt source.

- **$\overline{SS}$/OC1B/PCINT2 – Port B, Bit 2**

$\overline{SS}$: Slave select input. When the SPI is enabled as a slave, this pin is configured as an input regardless of the setting of DDB2. As a slave, the SPI is activated when this pin is driven low. When the SPI is enabled as a master, the data direction of this pin is controlled by DDB2. When the pin is forced by the SPI to be an input, the pull-up can still be controlled by the PORTB2 bit.

OC1B, output compare match output: The PB2 pin can serve as an external output for the Timer/Counter1 compare match B. The PB2 pin has to be configured as an output (DDB2 set (one)) to serve this function. The OC1B pin is also the output pin for the PWM mode timer function.

PCINT2: Pin change interrupt source 2. The PB2 pin can serve as an external interrupt source.

- **OC1A/PCINT1 – Port B, Bit 1**

OC1A, output compare match output: The PB1 pin can serve as an external output for the Timer/Counter1 compare match A. The PB1 pin has to be configured as an output (DDB1 set (one)) to serve this function. The OC1A pin is also the output pin for the PWM mode timer function.

PCINT1: Pin change interrupt source 1. The PB1 pin can serve as an external interrupt source.

- **ICP1/CLKO/PCINT0 – Port B, Bit 0**

ICP1, input capture pin: The PB0 pin can act as an input capture pin for Timer/Counter1.

CLKO, divided system clock: The divided system clock can be output on the PB0 pin. The divided system clock will be output if the CKOUT fuse is programmed, regardless of the PORTB0 and DDB0 settings. It will also be output during reset.

PCINT0: Pin change interrupt source 0. The PB0 pin can serve as an external interrupt source.

Table 10-6 and Table 10-7 relate the alternate functions of Port B to the overriding signals shown in Figure 10-6 on page 56. SPI MSTR INPUT and SPI SLAVE OUTPUT constitute the MISO signal, while MOSI is divided into SPI MSTR OUTPUT and SPI SLAVE INPUT.

Table 10-6. Overriding Signals for Alternate Functions in PB7..PB4

Signal Name	PB7/PCINT7[1]	PB6/CLKI/PCINT6[1]	PB5/SCK/PCINT5	PB4/MISO/PCINT4
PUOE	0	INTOSC	SPE × $\overline{MSTR}$	SPE × MSTR
PUOV	0	0	PORTB5 × $\overline{PUD}$	PORTB4 × $\overline{PUD}$
DDOE	0	INTOSC	SPE × $\overline{MSTR}$	SPE × MSTR
DDOV	0	0	0	0
PVOE	0	0	SPE × MSTR	SPE × $\overline{MSTR}$
PVOV	0	0	SCK OUTPUT	SPI SLAVE OUTPUT
DIEOE	PCINT7 × PCIE0	$\overline{INTOSC}$ + PCINT6 × PCIE0	PCINT5 × PCIE0	PCINT4 × PCIE0
DIEOV	1	INTOSC	1	1
DI	PCINT7 INPUT	PCINT6 INPUT	PCINT5 INPUT SCK INPUT	PCINT4 INPUT SPI MSTR INPUT
AIO	–	Clock input	–	–

Note: 1. INTOSC means that one of the internal oscillators are selected (by the CKSEL fuses), EXTCK means that external clock is selected (by the CKSEL fuses).

Table 10-7. Overriding Signals for Alternate Functions in PB3..PB0

Signal Name	PB3/MOSI/PCINT3	PB2/SS/OC1B/PCINT2	PB1/OC1A/PCINT1	PB0/ICP1/PCINT0
PUOE	SPE × MSTR	SPE × MSTR	0	0
PUOV	PORTB3 × PUD	PORTB2 × PUD	0	0
DDOE	SPE × MSTR	SPE × MSTR	0	0
DDOV	0	0	0	0
PVOE	SPE × MSTR	OC1B ENABLE	OC1A ENABLE	0
PVOV	SPI MSTR OUTPUT	OC1B	OC1A	0
DIEOE	PCINT3 × PCIE0	PCINT2 × PCIE0	PCINT1 × PCIE0	PCINT0 × PCIE0
DIEOV	1	1	1	1
DI	PCINT3 INPUT SPI SLAVE INPUT	PCINT2 INPUT SPI SS	PCINT1 INPUT	PCINT0 INPUT ICP1 INPUT
AIO	–	–	–	–

10.3.3 Alternate Functions of Port C

The port C pins with alternate functions are shown in Table 10-8.

Table 10-8. Port C Pins Alternate Functions

Port Pin	Alternate Function
PC7	PCINT15 (pin change interrupt 15)
PC6	RESET (reset pin) PCINT14 (pin change interrupt 14)
PC5	ADC5 (ADC input channel 5) SCL (2-wire serial bus clock line) PCINT13 (pin change interrupt 13)
PC4	ADC4 (ADC input channel 4) SDA (2-wire serial bus data input/output line) PCINT12 (pin change interrupt 12)
PC3	ADC3 (ADC input channel 3) PCINT11 (pin change interrupt 11)
PC2	ADC2 (ADC input channel 2) PCINT10 (pin change interrupt 10)
PC1	ADC1 (ADC input channel 1) PCINT9 (pin change interrupt 9)
PC0	ADC0 (ADC input channel 0) PCINT8 (pin change interrupt 8)

The alternate pin configuration is as follows:

- **PCINT15 – Port C, Bit 7**

PCINT15: Pin change interrupt source 15. The PC7 pin can serve as an external interrupt source.

- **RESET/PCINT14 – Port C, Bit 6**

RESET, reset pin: When the RSTDISBL fuse is programmed, this pin functions as a normal Input pin, and the part will have to rely on power-on reset and brown-out reset as its reset sources. When the RSTDISBL fuse is unprogrammed, the reset circuitry is connected to the pin, and the pin can not be used as an input pin.

If PC6 is used as a reset pin, DDC6, PORTC6 and PINC6 will all read 0.

PCINT14: Pin change interrupt source 14. The PC6 pin can serve as an external interrupt source.

- **SCL/ADC5/PCINT13 – Port C, Bit 5**

SCL, 2-wire serial interface clock: When the TWEN bit in TWCR is set (one) to enable the 2-wire serial interface, pin PC5 is disconnected from the port and becomes the serial clock I/O pin for the 2-wire serial interface. In this mode, there is a spike filter on the pin to suppress spikes shorter than 50 ns on the input signal, and the pin is driven by an open drain driver with slew-rate limitation. PC5 can also be used as ADC input channel 5. Note that ADC input channel 5 uses digital power.

PCINT13: Pin change interrupt source 13. The PC5 pin can serve as an external interrupt source.

- **SDA/ADC4/PCINT12 – Port C, Bit 4**

SDA, 2-wire serial interface data: When the TWEN bit in TWCR is set (one) to enable the 2-wire serial interface, pin PC4 is disconnected from the port and becomes the serial data I/O pin for the 2-wire serial interface. In this mode, there is a spike filter on the pin to suppress spikes shorter than 50ns on the input signal, and the pin is driven by an open drain driver with slew-rate limitation.

PC4 can also be used as ADC input channel 4. Note that ADC input channel 4 uses digital power.

PCINT12: Pin change interrupt source 12. The PC4 pin can serve as an external interrupt source.

- **ADC3/PCINT11 – Port C, Bit 3**

PC3 can also be used as ADC input channel 3. Note that ADC input channel 3 uses analog power.

PCINT11: Pin change interrupt source 11. The PC3 pin can serve as an external interrupt source.

- **ADC2/PCINT10 – Port C, Bit 2**

PC2 can also be used as ADC input channel 2. Note that ADC input channel 2 uses analog power.

PCINT10: Pin change Interrupt source 10. The PC2 pin can serve as an external interrupt source.

- **ADC1/PCINT9 – Port C, Bit 1**

PC1 can also be used as ADC input channel 1. Note that ADC input channel 1 uses analog power.

PCINT9: Pin change Interrupt source 9. The PC1 pin can serve as an external interrupt source.

- **ADC0/PCINT8 – Port C, Bit 0**

PC0 can also be used as ADC input Channel 0. Note that ADC input channel 0 uses analog power.

PCINT8: Pin change interrupt source 8. The PC0 pin can serve as an external interrupt source.

Table 10-9 and Table 10-10 relate the alternate functions of port C to the overriding signals shown in Figure 10-6 on page 56.

Table 10-9. Overriding Signals for Alternate Functions in PC6..PC4[1]

Signal Name	PC7/PCINT15	PC6/$\overline{\text{RESET}}$/PCINT14	PC5/SCL/ADC5/PCINT13	PC4/SDA/ADC4/PCINT12
PUOE	0	RSTDISBL	TWEN	TWEN
PUOV	0	1	PORTC5 × $\overline{\text{PUD}}$	PORTC4 × $\overline{\text{PUD}}$
DDOE	0	RSTDISBL	TWEN	TWEN
DDOV	0	0	SCL_OUT	SDA_OUT
PVOE	0	0	TWEN	TWEN
PVOV	0	0	0	0
DIEOE	PCINT15 × PCIE1	RSTDISBL + PCINT14 × PCIE1	PCINT13 × PCIE1 + ADC5D	PCINT12 × PCIE1 + ADC4D
DIEOV	1	RSTDISBL	PCINT13 × PCIE1	PCINT12 × PCIE1
DI	PCINT15 INPUT	PCINT14 INPUT	PCINT13 INPUT	PCINT12 INPUT
AIO	-	RESET INPUT	ADC5 INPUT / SCL INPUT	ADC4 INPUT / SDA INPUT

Note: 1. When enabled, the 2-wire serial interface enables slew-rate controls on the output pins PC4 and PC5. This is not shown in the figure. In addition, spike filters are connected between the AIO outputs shown in the port figure and the digital logic of the TWI module

Table 10-10. Overriding Signals for Alternate Functions in PC3..PC0

Signal Name	PC3/ADC3/PCINT11	PC2/ADC2/PCINT10	PC1/ADC1/PCINT9	PC0/ADC0/PCINT8
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
DIEOE	PCINT11 × PCIE1 + ADC3D	PCINT10 × PCIE1 + ADC2D	PCINT9 × PCIE1 + ADC1D	PCINT8 × PCIE1 + ADC0D
DIEOV	PCINT11 × PCIE1	PCINT10 × PCIE1	PCINT9 × PCIE1	PCINT8 × PCIE1
DI	PCINT11 INPUT	PCINT10 INPUT	PCINT9 INPUT	PCINT8 INPUT
AIO	ADC3 INPUT	ADC2 INPUT	ADC1 INPUT	ADC0 INPUT

10.3.4 Alternate Functions of Port D

The port D pins with alternate functions are shown in Table 10-11.

Table 10-11. Port D Pins Alternate Functions

Port Pin	Alternate Function
PD7	AIN1 (analog comparator negative input) PCINT23 (pin change interrupt 23)
PD6	AIN0 (analog comparator positive input) PCINT22 (pin change interrupt 22)
PD5	T1 (Timer/Counter 1 external counter input) PCINT21 (pin change interrupt 21)
PD4	T0 (Timer/Counter 0 external counter input) PCINT20 (pin change interrupt 20)
PD3	INT1 (external interrupt 1 input) PCINT19 (pin change interrupt 19)
PD2	INT0 (external interrupt 0 input) PCINT18 (pin change interrupt 18)
PD1	PCINT17 (pin change interrupt 17)
PD0	PCINT16 (pin change interrupt 16)

The alternate pin configuration is as follows:

- **AIN1/PCINT23 – Port D, Bit 7**

AIN1: Analog comparator negative input. Configure the port pin as input with the internal pull-up switched off to avoid the digital port function from interfering with the function of the analog comparator.

PCINT23: Pin hange interrupt source 23. The PD7 pin can serve as an external interrupt source.

- **AIN0/PCINT22 – Port D, Bit 6**

AIN0: Analog comparator positive input. Configure the port pin as input with the internal pull-up switched off to avoid the digital port function from interfering with the function of the analog comparator.

PCINT22: Pin change interrupt source 22. The PD6 pin can serve as an external interrupt source.

- **T1/PCINT21 – Port D, Bit 5**

T1: Timer/Counter1 counter source.

PCINT21: Pin change interrupt source 21. The PD5 pin can serve as an external interrupt source.

- **T0/PCINT20 – Port D, Bit 4**

T0: Timer/Counter0 counter source.

PCINT20: Pin change interrupt source 20. The PD4 pin can serve as an external interrupt source.

- **INT1/PCINT19 – Port D, Bit 3**

INT1, external interrupt source 1: The PD3 pin can serve as an external interrupt source.

PCINT19: Pin change interrupt source 19. The PD3 pin can serve as an external interrupt source.

- **INT0/PCINT18 – Port D, Bit 2**

INT0, external interrupt source 0: The PD2 pin can serve as an external interrupt source.

PCINT18: Pin change interrupt source 18. The PD2 pin can serve as an external interrupt source.

- **PCINT17 – Port D, Bit 1**

PCINT17: Pin change interrupt source 17. The PD1 pin can serve as an external interrupt source.

- **PCINT16 – Port D, Bit 0**

PCINT16: Pin change interrupt source 16. The PD0 pin can serve as an external interrupt source.

Table 10-12 and Table 10-13 on page 65 relate the alternate functions of port D to the overriding signals shown in Figure 10-6 on page 56.

Table 10-12. Overriding Signals for Alternate Functions PD7..PD4

Signal Name	PD7/AIN1/PCINT23	PD6/AIN0/PCINT22	PD5/T1/PCINT21	PD4/T0/PCINT20
PUOE	0	0	0	0
PUO	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
DIEOE	PCINT23 × PCIE2	PCINT22 × PCIE2	PCINT21 × PCIE2	PCINT20 × PCIE2
DIEOV	1	1	1	1
DI	PCINT23 INPUT	PCINT22 INPUT	PCINT21 INPUT T1 INPUT	PCINT20 INPUT T0 INPUT
AIO	AIN1 INPUT	AIN0 INPUT	–	–

Table 10-13. Overriding Signals for Alternate Functions in PD3..PD0

Signal Name	PD3/INT1/PCINT19	PD2/INT0/PCINT18	PD1/PCINT17	PD0/PCINT16
PUOE	0	0	0	0
PUO	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
DIEOE	INT1 ENABLE + PCINT19 × PCIE2	INT0 ENABLE + PCINT18 × PCIE1	PCINT17 × PCIE2	PCINT16 × PCIE2
DIEOV	1	1	1	1
DI	PCINT19 INPUT INT1 INPUT	PCINT18 INPUT INT0 INPUT	PCINT17 INPUT	PCINT16 INPUT
AIO	–	–	–	–

10.4 Register Description

10.4.1 MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
	–	BPDS	BPDSE	PUD	–	–	–	–	MCUCR
Read/Write	R	R/W	R/W	R/W	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 4 – PUD: Pull-up Disable**

When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn registers are configured to enable the pull-ups ({DDxn, PORTxn} = 0b01). See Section 10.2.1 “Configuring the Pin” on page 52 for more details about this feature.

10.4.2 PORTCR – Port Control Register

Bit	7	6	5	4	3	2	1	0	
	BBMD	**BBMC**	**BBMB**	**BBMA**	PUDD	PUDC	PUDB	PUDA	PORTR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..4 – BBMx: Break-Before-Make Mode Enable**

When these bits are written to one, the port-wise break-before-make mode is activated. The intermediate tri-state cycle is then inserted when writing DDRxn to make an output. For further information, see Section 10.2.3 “Break-Before-Make Switching” on page 53.

- **Bits 3..0 – PUDx: Port-Wise Pull-up Disable**

When these bits are written to one, the port-wise pull-ups in the defined I/O ports are disabled even if the DDxn and PORTxn registers are configured to enable the pull-ups ({DDxn, PORTxn} = 0b01). The port-wise pull-up disable bits are ORed with the global pull-up disable bit (PUD) from the MCUCR register. See Section 10.2.1 “Configuring the Pin” on page 52 for more details about this feature.

10.4.3 PORTA – The Port A Data Register

Bit	7	6	5	4	3	2	1	0	
	-	-	-	-	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10.4.4 DDRA – The Port A Data Direction Register

Bit	7	6	5	4	3	2	1	0	
	-	-	-	-	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10.4.5 PINA – The Port A Input Pins

Bit	7	6	5	4	3	2	1	0	
	-	-	-	-	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	N/A	N/A	N/A	N/A	

10.4.6 PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10.4.7 DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10.4.8 PINB – The Port B Input Pins

Bit	7	6	5	4	3	2	1	0	
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

10.4.9 PORTC – The Port C Data Register

Bit	7	6	5	4	3	2	1	0	
	PORTC6	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10.4.10 DDRC – The Port C Data Direction Register

Bit	7	6	5	4	3	2	1	0	
	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10.4.11 PINC – The Port C Input Pins

Bit	7	6	5	4	3	2	1	0	
	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

10.4.12 PORTD – The Port D Data Register

Bit	7	6	5	4	3	2	1	0	
	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10.4.13 DDRD – The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10.4.14 PIND – The Port D Input Pins

Bit	7	6	5	4	3	2	1	0	
	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

11. 8-bit Timer/Counter0

11.1 Features

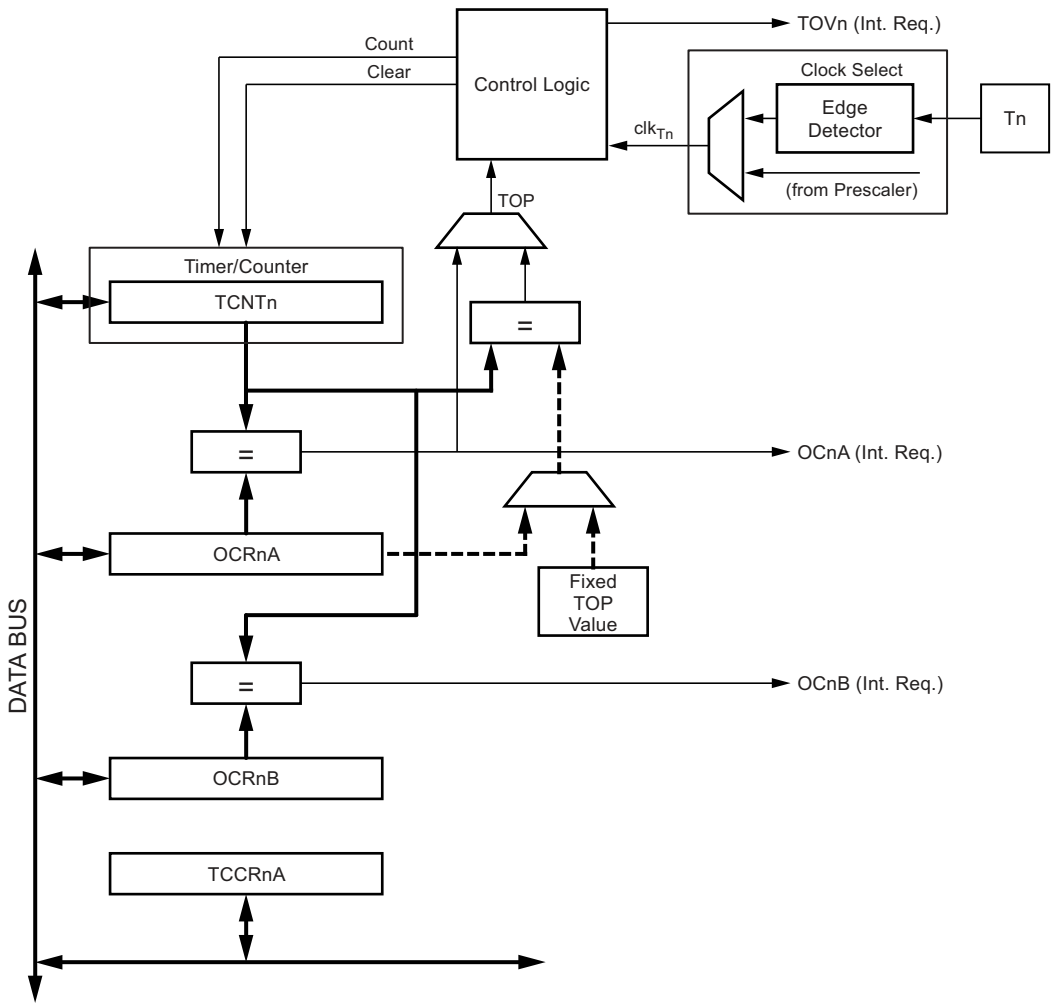
- Two independent output compare units
- Clear timer on compare match (auto reload)
- Three independent interrupt sources (TOV0, OCF0A, and OCF0B)

11.2 Overview

Timer/Counter0 is a general purpose 8-bit Timer/Counter module, with two independent output compare units. It allows accurate program execution timing (event management). A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 11-1. For the actual placement of I/O pins, refer to Section 1-1 “Pinout of ATtiny88” on page 3. CPU accessible I/O registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O register and bit locations are listed in the Section 11.8 “8-bit Timer/Counter Register Description” on page 73.

The PRTIM0 bit in Section 7.4.3 “PRR – Power Reduction Register” on page 36 must be written to zero to enable Timer/Counter0 module.

Figure 11-1. 8-bit Timer/Counter Block Diagram



11.2.1 Definitions

Many register and bit references in this section are written in general form, where a lower case “n” replaces the Timer/Counter number (in this case 0) and a lower case “x” replaces the output compare Unit (in this case compare unit A or compare unit B). However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT0 for accessing Timer/Counter0 counter value and so on.

The definitions in Table 11-1 are used extensively throughout the document.

Table 11-1. Definitions

Parameter	Definition
MAX	The counter reaches its MAXimum when it becomes 0xFF (decimal 255).
TOP	The counter reaches the TOP when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be the fixed value 0xFF (MAX) or the value stored in the OCR0A register. The assignment is dependent on the mode of operation.

11.2.2 Registers

The Timer/Counter (TCNT0) and output compare registers (OCR0A and OCR0B) are 8-bit registers. Interrupt request (abbreviated to Int.Req. in the figure) signals are all visible in the timer interrupt flag register (TIFR0). All interrupts are individually masked with the timer interrupt mask register (TIMSK0). TIFR0 and TIMSK0 are not shown in the figure.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the T0 pin. The clock select logic block controls which clock source and edge the Timer/Counter uses to increment its value. The Timer/Counter is inactive when no clock source is selected. The output from the clock select logic is referred to as the timer clock (clk_{T0}).

The output compare registers (OCR0A and OCR0B) are compared with the Timer/Counter value at all times. The compare match event will also set the compare flag (OCF0A or OCF0B) which can be used to generate an output compare interrupt request.

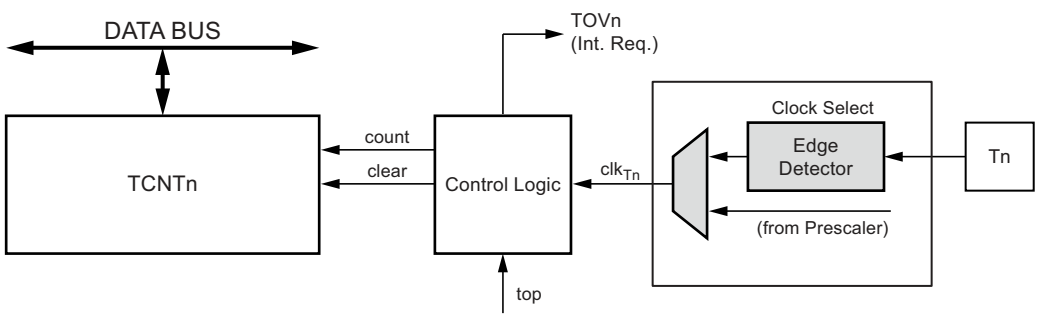
11.3 Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the clock select logic which is controlled by the clock select (CS02:0) bits located in the Timer/Counter control register (TCCR0A). For details on clock sources and prescaler, see Section 13. “Timer/Counter0 and Timer/Counter1 Prescalers” on page 102.

11.4 Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Figure 11-2 shows a block diagram of the counter and its surroundings.

Figure 11-2. Counter Unit Block Diagram



Signal description (internal signals):

count	Increment or decrement TCNT0 by 1.
clear	Clear TCNT0 (set all bits to zero).
clk_{Tn}	Timer/Counter clock, referred to as clk_{T0} in the following.
top	Signalize that TCNT0 has reached maximum value.

Depending of the mode of operation used, the counter is cleared or incremented at each timer clock (clk_{T0}). clk_{T0} can be generated from an external or internal clock source, selected by the clock select bits (CS02:0). When no clock source is selected (CS02:0 = 0) the timer is stopped. However, the TCNT0 value can be accessed by the CPU, regardless of whether clk_{T0} is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the clear timer on compare match bit (CTC0) located in the Timer/Counter control register (TCCR0A). For more details about advanced counting sequences, see Section 11.6 “Modes of Operation” on page 71.

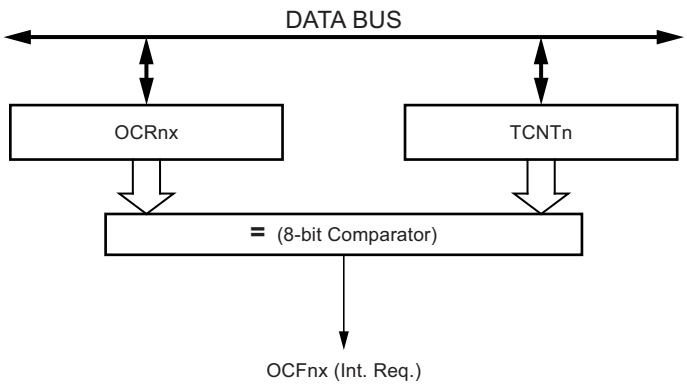
The Timer/Counter overflow flag (TOV0) is set according to the mode of operation selected by the CTC0 bit. TOV0 can be used for generating a CPU interrupt.

11.5 Output Compare Unit

The 8-bit comparator continuously compares TCNT0 with the output compare registers (OCR0A and OCR0B). Whenever TCNT0 equals OCR0A or OCR0B, the comparator signals a match. A match will set the output compare flag (OCF0A or OCF0B) at the next timer clock cycle. If the corresponding interrupt is enabled, the output compare flag generates an output compare interrupt. The output compare flag is automatically cleared when the interrupt is executed. Alternatively, the flag can be cleared by software by writing a logical one to its I/O bit location.

Figure 11-3 shows a block diagram of the output compare unit.

Figure 11-3. Output Compare Unit, Block Diagram



11.5.1 Compare Match Blocking by TCNT0 Write

All CPU write operations to the TCNT0 register will block any compare match that occur in the next timer clock cycle, even when the timer is stopped. This feature allows OCR0x to be initialized to the same value as TCNT0 without triggering an interrupt when the Timer/Counter clock is enabled.

11.5.2 Using the Output Compare Unit

Since writing TCNT0 in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNT0 when using the output compare unit, independently of whether the Timer/Counter is running or not. If the value written to TCNT0 equals the OCR0x value, the compare match will be missed, resulting in incorrect waveform generation.

11.6 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the output compare pins, is defined by the CTC0 bit.

For detailed timing information refer to Section 11.7 “Timer/Counter Timing Diagrams” on page 72.

11.6.1 Normal Mode

The simplest mode of operation is the normal mode (CTC0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 8-bit value (TOP = 0xFF) and then restarts from the bottom (0x00). In normal operation the Timer/Counter overflow flag (TOV0) will be set in the same timer clock cycle as the TCNT0 becomes zero. The TOV0 flag in this case behaves like a ninth bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV0 flag, the timer resolution can be increased by software. There are no special cases to consider in the normal mode, a new counter value can be written anytime.

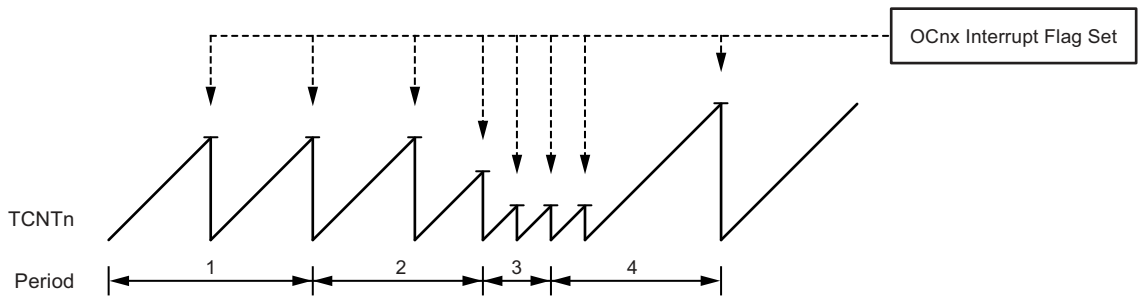
The output compare unit can be used to generate interrupts at some given time.

11.6.2 Clear Timer on Compare Match (CTC) Mode

In clear timer on compare or CTC mode (CTC0 = 1), the OCR0A register is used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT0) matches the OCR0A. The OCR0A defines the top value for the counter, hence also its resolution. This mode allows greater control of the compare match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 11-4. The counter value (TCNT0) increases until a compare match occurs between TCNT0 and OCR0A, and then counter (TCNT0) is cleared.

Figure 11-4. CTC Mode, Timing Diagram



An interrupt can be generated each time the counter value reaches the TOP value by using the OCF0A flag. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing TOP to a value close to BOTTOM when the counter is running with none or a low prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written to OCR0A is lower than the current value of TCNT0, the counter will miss the compare match. The counter will then have to count to its maximum value (0xFF) and wrap around starting at 0x00 before the compare match can occur.

As for the normal mode of operation, the TOV0 flag is set in the same timer clock cycle that the counter counts from MAX to 0x00.

11.7 Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock (clk_{T0}) is therefore shown as a clock enable signal in the following figures. The figures include information on when interrupt flags are set. Figure 11-5 contains timing data for basic Timer/Counter operation. The figure shows the count sequence close to the MAX value in all modes.

Figure 11-5. Timer/Counter Timing Diagram, no Prescaling

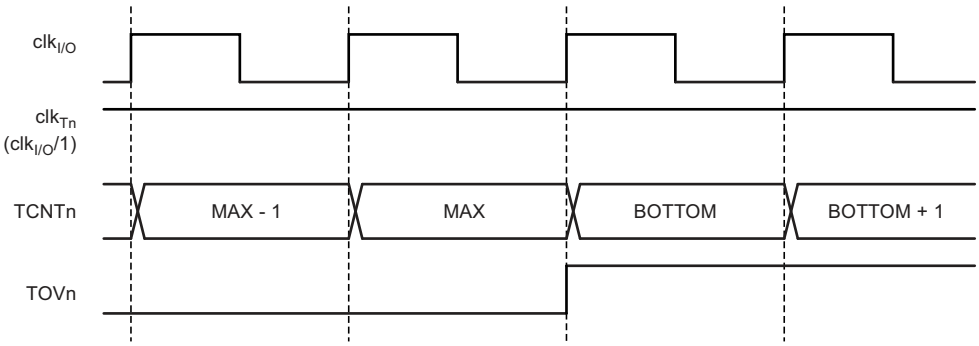


Figure 11-6 shows the same timing data, but with the prescaler enabled.

Figure 11-6. Timer/Counter Timing Diagram, with Prescaler ($f_{clk\_I/O}/8$)

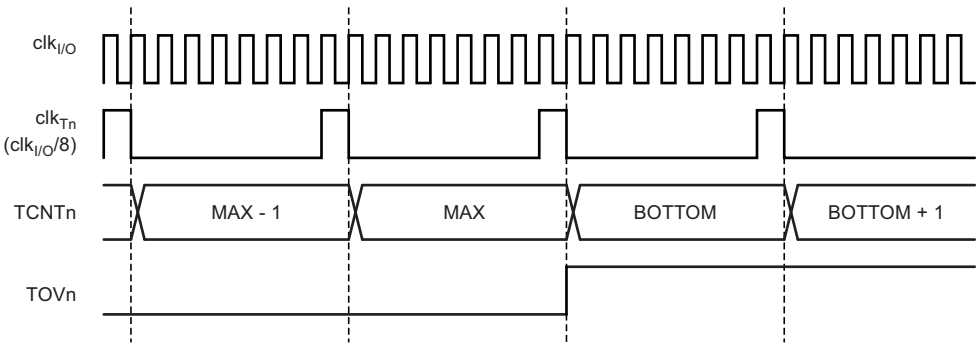


Figure 11-7 shows the setting of OCF0B in all modes and OCF0A in all modes except CTC mode.

Figure 11-7. Timer/Counter Timing Diagram, Setting of OCF0x, with Prescaler ($f_{clk\_I/O}/8$)

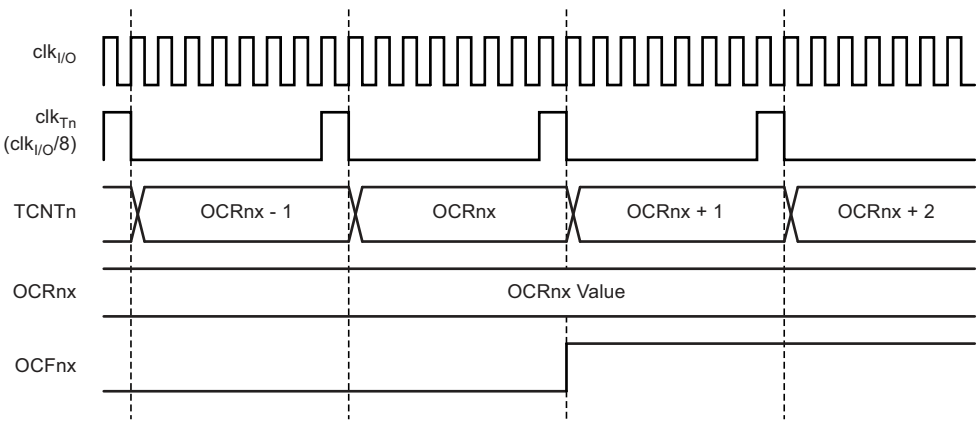
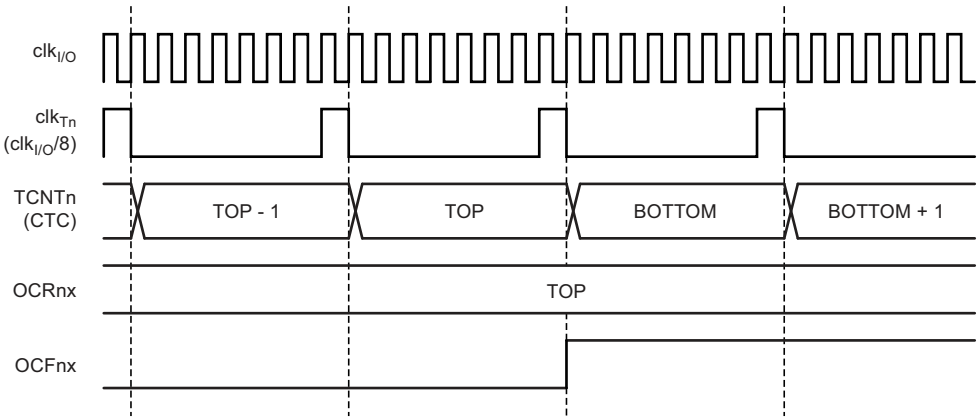


Figure 11-8 shows the setting of OCF0A and the clearing of TCNT0 in CTC mode where OCR0A is TOP.

Figure 11-8. Timer/Counter Timing Diagram, Clear Timer on Compare Match mode, with Prescaler ($f_{clk\_I/O}$/8)



11.8 8-bit Timer/Counter Register Description

11.8.1 TCCR0A – Timer/Counter Control Register A–

Bit	7	6	5	4	3	2	1	0	
	-	-	–	–	**CTC0**	**CS02**	**CS01**	**CS00**	TCCR0A
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:4 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 3 – CTC0: Clear Timer on Compare Match Mode**

This bit control the counting sequence of the counter, the source for maximum (TOP) counter value, see Table 11-2. Modes of operation supported by the Timer/Counter unit are: normal mode (counter), clear timer on compare match (CTC) mode (see Section 11.6 “Modes of Operation” on page 71).

Table 11-2. CTC Mode Bit Description

Mode	CTC0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)
0	0	Normal	0xFF	Immediate	MAX
1	1	CTC	OCRA	Immediate	MAX

Note: 1. MAX = 0xFF

- **Bits 2:0 – CS02:0: Clock Select**

The three clock select bits select the clock source to be used by the Timer/Counter.

Table 11-3. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$ (no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

If external pin modes are used for the Timer/Counter0, transitions on the T0 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

11.8.2 TCNT0 – Timer/Counter Register

Bit	7	6	5	4	3	2	1	0	
	TCNT0**[7:0]**								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Timer/Counter register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter. Writing to the TCNT0 register blocks (removes) the compare match on the following timer clock. Modifying the counter (TCNT0) while the counter is running, introduces a risk of missing a compare match between TCNT0 and the OCR0x registers.

11.8.3 OCR0A – Output Compare Register A

Bit	7	6	5	4	3	2	1	0	
	OCR0A**[7:0]**								OCR0A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The output compare register A contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an output compare interrupt.

11.8.4 OCR0B – Output Compare Register B

Bit	7	6	5	4	3	2	1	0	
	OCR0B**[7:0]**								OCR0B
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The output compare register B contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an output compare interrupt.

11.8.5 TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..3 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 2 – OCIE0B: Timer/Counter Output Compare Match B Interrupt Enable**

When the OCIE0B bit is written to one, and the I-bit in the status register is set, the Timer/Counter compare match B interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter occurs, i.e., when the OCF0B bit is set in the Timer/Counter interrupt flag register – TIFR0.

- **Bit 1 – OCIE0A: Timer/Counter0 Output Compare Match A Interrupt Enable**

When the OCIE0A bit is written to one, and the I-bit in the status register is set, the Timer/Counter0 compare match A interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter0 occurs, i.e., when the OCF0A bit is set in the Timer/Counter 0 interrupt flag register – TIFR0.

- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

When the TOIE0 bit is written to one, and the I-bit in the status register is set, the Timer/Counter0 overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in the Timer/Counter 0 interrupt flag register – TIFR0.

11.8.6 TIFR0 – Timer/Counter 0 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..3 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 2 – OCF0B: Timer/Counter 0 Output Compare B Match Flag**

The OCF0B bit is set when a compare match occurs between the Timer/Counter and the data in OCR0B – output compare register0 B. OCF0B is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF0B is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE0B (Timer/Counter compare B match interrupt enable), and OCF0B are set, the Timer/Counter compare match interrupt is executed.

- **Bit 1 – OCF0A: Timer/Counter 0 Output Compare A Match Flag**

The OCF0A bit is set when a compare match occurs between the Timer/Counter0 and the data in OCR0A – output compare register0. OCF0A is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF0A is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE0A (Timer/Counter0 compare match interrupt enable), and OCF0A are set, the Timer/Counter0 compare match interrupt is executed.

- **Bit 0 – TOV0: Timer/Counter0 Overflow Flag**

The bit TOV0 is set when an overflow occurs in Timer/Counter0. TOV0 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, TOV0 is cleared by writing a logic one to the flag. When the SREG I-bit, TOIE0 (Timer/Counter0 overflow interrupt enable), and TOV0 are set, the Timer/Counter0 overflow interrupt is executed.

See Table 11-2 on page 73, Section 11-2 “CTC Mode Bit Description” on page 73.

12. 16-bit Timer/Counter1 with PWM

12.1 Features

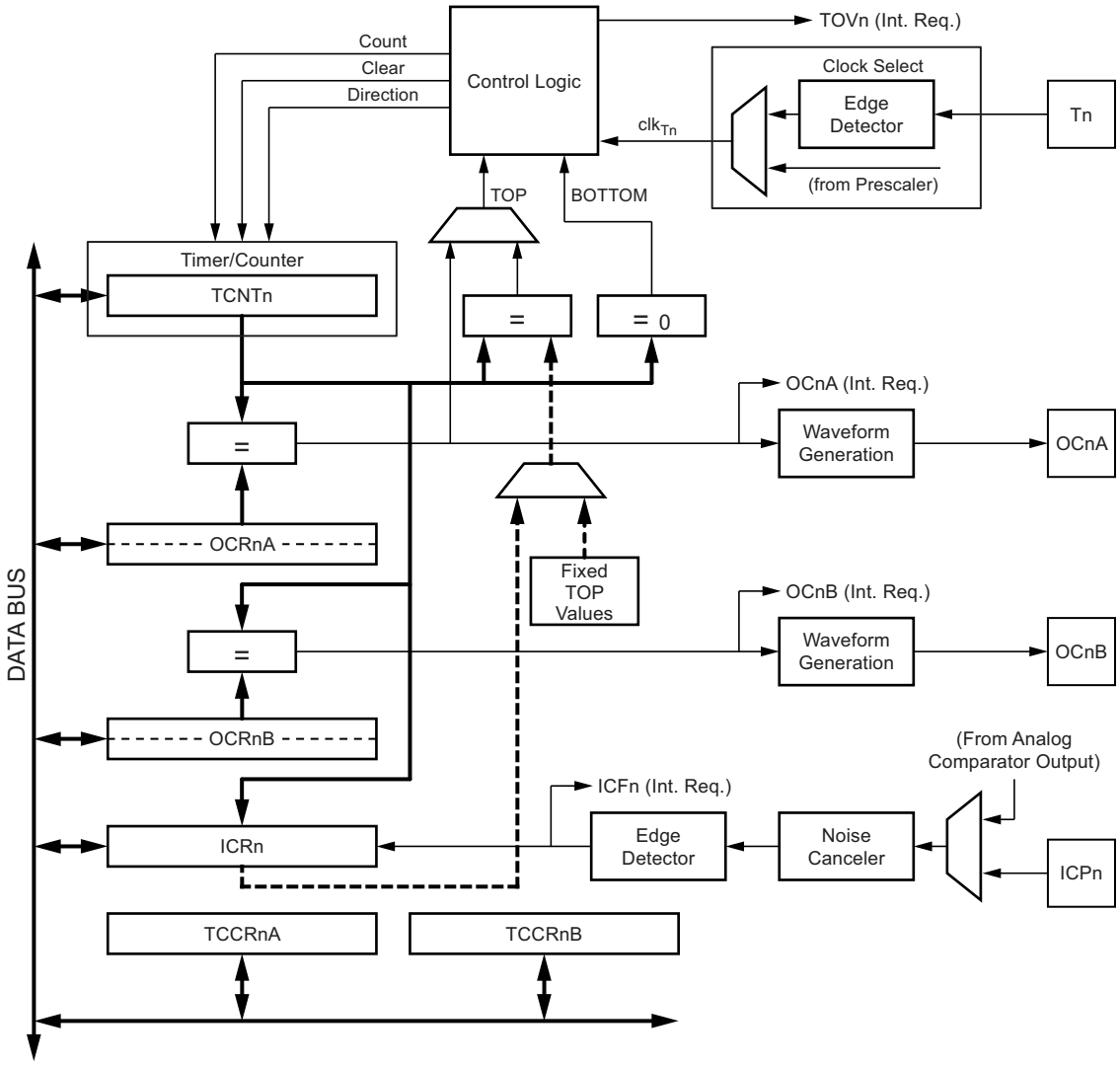
- True 16-bit design (i.e., Allows 16-bit PWM)
- Two independent output compare units
- Double buffered output compare registers
- One input capture unit
- Input capture noise canceler
- Clear timer on compare match (auto reload)
- Glitch-free, phase correct pulse width modulator (PWM)
- Variable PWM period
- Frequency generator
- External event counter
- Four independent interrupt sources (TOV1, OCF1A, OCF1B, and ICF1)

12.2 Overview

Most register and bit references in this section are written in general form, where a lower case “n” replaces the Timer/Counter number, and a lower case “x” replaces the output compare unit channel. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT1 for accessing Timer/Counter1 counter value and so on.

The 16-bit Timer/Counter unit allows accurate program execution timing (event management), wave generation, and signal timing measurement. A simplified block diagram of the 16-bit Timer/Counter is shown in Figure 12-1.

Figure 12-1. 16-bit Timer/Counter Block Diagram(1)



Note: 1. Refer to Figure 1-1 on page 3, Table 10-5 on page 59 and Table 10-11 on page 63 for Timer/Counter1 pin placement and description.

For actual placement of I/O pins, refer to Section 1-1 “Pinout of ATtiny88” on page 3. CPU accessible I/O registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O register and bit locations are listed in the Section 12.11 “Register Description” on page 95.

The PRTIM1 bit in Section 7.4.3 “PRR – Power Reduction Register” on page 36 must be written to zero to enable Timer/Counter1 module.

12.2.1 Registers

The Timer/Counter (TCNT1), output compare registers (OCR1A/B), and input capture register (ICR1) are all 16-bit registers. Special procedures must be followed when accessing the 16-bit registers. These procedures are described in the section Section 12.3 “Accessing 16-bit Registers” on page 78. The Timer/Counter control registers (TCCR1A/B) are 8-bit registers and have no CPU access restrictions. Interrupt requests (abbreviated to Int.Req. in the figure) signals are all visible in the timer interrupt flag register (TIFR1). All interrupts are individually masked with the timer interrupt mask register (TIMSK1). TIFR1 and TIMSK1 are not shown in the figure.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the T1 pin. The clock select logic block controls which clock source and edge the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the clock select logic is referred to as the timer clock (clk_{T1}).

The double buffered output compare registers (OCR1A/B) are compared with the Timer/Counter value at all time. The result of the compare can be used by the waveform generator to generate a PWM or variable frequency output on the output compare pin (OC1A/B). See Section 12.7 “Output Compare Units” on page 84. The compare match event will also set the compare match flag (OCF1A/B) which can be used to generate an output compare interrupt request.

The input capture register can capture the Timer/Counter value at a given external (edge triggered) event on either the input capture pin (ICP1) or on the analog comparator pins (See Section 16. “Analog Comparator” on page 142). The input capture unit includes a digital filtering unit (noise canceler) for reducing the chance of capturing noise spikes.

The TOP value, or maximum Timer/Counter value, can in some modes of operation be defined by either the OCR1A register, the ICR1 register, or by a set of fixed values. When using OCR1A as TOP value in a PWM mode, the OCR1A register can not be used for generating a PWM output. However, the TOP value will in this case be double buffered allowing the TOP value to be changed in run time. If a fixed TOP value is required, the ICR1 register can be used as an alternative, freeing the OCR1A to be used as PWM output.

12.2.2 Definitions

The following definitions are used extensively throughout the section:

Table 12-1. Definitions

Parameter	Definition
BOTTOM	The counter reaches the BOTTOM when it becomes 0x0000.
MAX	The counter reaches its MAXimum when it becomes 0xFFFF (decimal 65535).
TOP	The counter reaches the TOP when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be one of the fixed values: 0x00FF, 0x01FF, or 0x03FF, or to the value stored in the OCR1A or ICR1 register. The assignment is dependent of the mode of operation.

12.3 Accessing 16-bit Registers

The TCNT1, OCR1A/B, and ICR1 are 16-bit registers that can be accessed by the AVR CPU via the 8-bit data bus. The 16-bit register must be byte accessed using two read or write operations. Each 16-bit timer has a single 8-bit register for temporary storing of the high byte of the 16-bit access. The same temporary register is shared between all 16-bit registers within each 16-bit timer. Accessing the low byte triggers the 16-bit read or write operation. When the low byte of a 16-bit register is written by the CPU, the high byte stored in the temporary register, and the low byte written are both copied into the 16-bit register in the same clock cycle. When the low byte of a 16-bit register is read by the CPU, the high byte of the 16-bit register is copied into the temporary register in the same clock cycle as the low byte is read.

Not all 16-bit accesses uses the temporary register for the high byte. Reading the OCR1A/B 16-bit registers does not involve using the temporary register.

To do a 16-bit write, the high byte must be written before the low byte. For a 16-bit read, the low byte must be read before the high byte.

The following code examples show how to access the 16-bit timer registers assuming that no interrupts updates the temporary register. The same principle can be used directly for accessing the OCR1A/B and ICR1 registers. Note that when using “C”, the compiler handles the 16-bit access.

Assembly Code Examples[1]

```
...
; Set TCNT1 to 0x01FF
ldi     r17,0x01
ldi     r16,0xFF
out     TCNT1H,r17
out     TCNT1L,r16
; Read TCNT1 into r17:r16
in      r16,TCNT1L
in      r17,TCNT1H
...
```

C Code Examples[1]

```
unsigned int i;
...
/* Set TCNT1 to 0x01FF */
TCNT1 = 0x1FF;
/* Read TCNT1 into i */
i = TCNT1;
...
```

Note: 1. Section 3.2 “About Code Examples” on page 8
For I/O registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

The assembly code example returns the TCNT1 value in the r17:r16 register pair.

It is important to notice that accessing 16-bit registers are atomic operations. If an interrupt occurs between the two instructions accessing the 16-bit register, and the interrupt code updates the temporary register by accessing the same or any other of the 16-bit timer registers, then the result of the access outside the interrupt will be corrupted. Therefore, when both the main code and the interrupt code update the temporary register, the main code must disable the interrupts during the 16-bit access.

The following code examples show how to do an atomic read of the TCNT1 register contents. Reading any of the OCR1A/B or ICR1 registers can be done by using the same principle.

Assembly Code Example[1]

```
TIM16_ReadTCNT1:
        ; Save global interrupt flag
        in      r18,SREG
        ; Disable interrupts
        cli
        ; Read TCNT1 into r17:r16
        in      r16,TCNT1L
        in      r17,TCNT1H
        ; Restore global interrupt flag
        out     SREG,r18
        ret
```

C Code Example[1]

```
unsigned int TIM16_ReadTCNT1( void )
{
        unsigned char sreg;
        unsigned int i;
        /* Save global interrupt flag */
        sreg = SREG;
        /* Disable interrupts */
        _CLI();
        /* Read TCNT1 into i */
        i = TCNT1;
        /* Restore global interrupt flag */
        SREG = sreg;
        return i;
}
```

Note: 1. Section 3.2 “About Code Examples” on page 8
For I/O registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

The assembly code example returns the TCNT1 value in the r17:r16 register pair.

The following code examples show how to do an atomic write of the TCNT1 register contents. Writing any of the OCR1A/B or ICR1 registers can be done by using the same principle.

Assembly Code Example[1]

```
TIM16_WriteTCNT1:
        ; Save global interrupt flag
        in      r18,SREG
        ; Disable interrupts
        cli
        ; Set TCNT1 to r17:r16
        out     TCNT1H,r17
        out     TCNT1L,r16
        ; Restore global interrupt flag
        out     SREG,r18
        ret
```

C Code Example[1]

```
void TIM16_WriteTCNT1( unsigned int i )
{
        unsigned char sreg;
        unsigned int i;
        /* Save global interrupt flag */
        sreg = SREG;
        /* Disable interrupts */
        _CLI();
        /* Set TCNT1 to i */
        TCNT1 = i;
        /* Restore global interrupt flag */
        SREG = sreg;
}
```

Note: 1. Section 3.2 “About Code Examples” on page 8
For I/O registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

The assembly code example requires that the r17:r16 register pair contains the value to be written to TCNT1.

12.3.1 Reusing the Temporary High Byte Register

If writing to more than one 16-bit register where the high byte is the same for all registers written, then the high byte only needs to be written once. However, note that the same rule of atomic operation described previously also applies in this case.

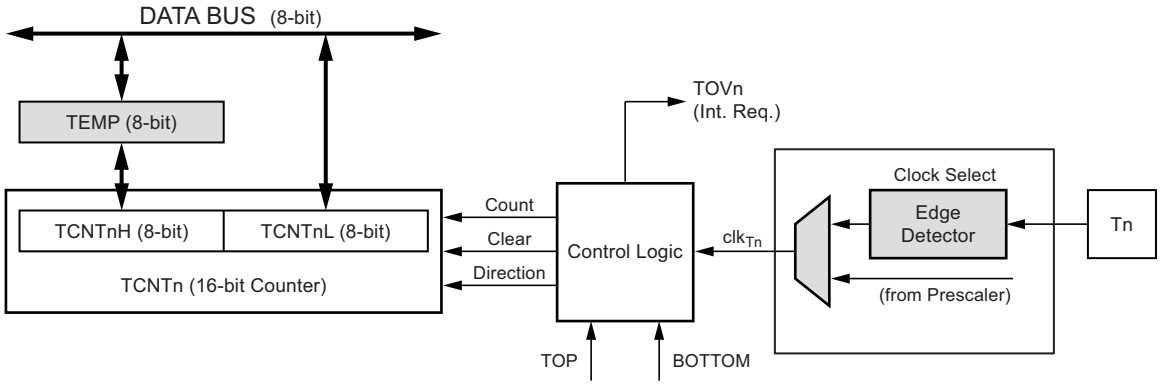
12.4 Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the clock select logic which is controlled by the clock select (CS12:0) bits located in the Timer/Counter control register B (TCCR1B). For details on clock sources and prescaler, see Section 13. “Timer/Counter0 and Timer/Counter1 Prescalers” on page 102.

12.5 Counter Unit

The main part of the 16-bit Timer/Counter is the programmable 16-bit bi-directional counter unit. Figure 12-2 shows a block diagram of the counter and its surroundings.

Figure 12-2. Counter Unit Block Diagram



Signal description (internal signals):

Count	Increment or decrement TCNT1 by 1.
Direction	Select between increment and decrement.
Clear	Clear TCNT1 (set all bits to zero).
clk_{T1}	Timer/Counter clock.
TOP	Signalize that TCNT1 has reached maximum value.
BOTTOM	Signalize that TCNT1 has reached minimum value (zero).

The 16-bit counter is mapped into two 8-bit I/O memory locations: counter high (TCNT1H) containing the upper eight bits of the counter, and counter low (TCNT1L) containing the lower eight bits. The TCNT1H register can only be indirectly accessed by the CPU. When the CPU does an access to the TCNT1H I/O location, the CPU accesses the high byte temporary register (TEMP). The temporary register is updated with the TCNT1H value when the TCNT1L is read, and TCNT1H is updated with the temporary register value when TCNT1L is written. This allows the CPU to read or write the entire 16-bit counter value within one clock cycle via the 8-bit data bus. It is important to notice that there are special cases of writing to the TCNT1 register when the counter is counting that will give unpredictable results. The special cases are described in the sections where they are of importance.

Depending on the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock (clk_{T1}). The clk_{T1} can be generated from an external or internal clock source, selected by the clock select bits (CS12:0). When no clock source is selected (CS12:0 = 0) the timer is stopped. However, the TCNT1 value can be accessed by the CPU, independent of whether clk_{T1} is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the waveform generation mode bits (WGM13:0) located in the Timer/Counter control registers A and B (TCCR1A and TCCR1B). There are close connections between how the counter behaves (counts) and how waveforms are generated on the output compare outputs OC1x. For more details about advanced counting sequences and waveform generation, see Section 12.9 “Modes of Operation” on page 87.

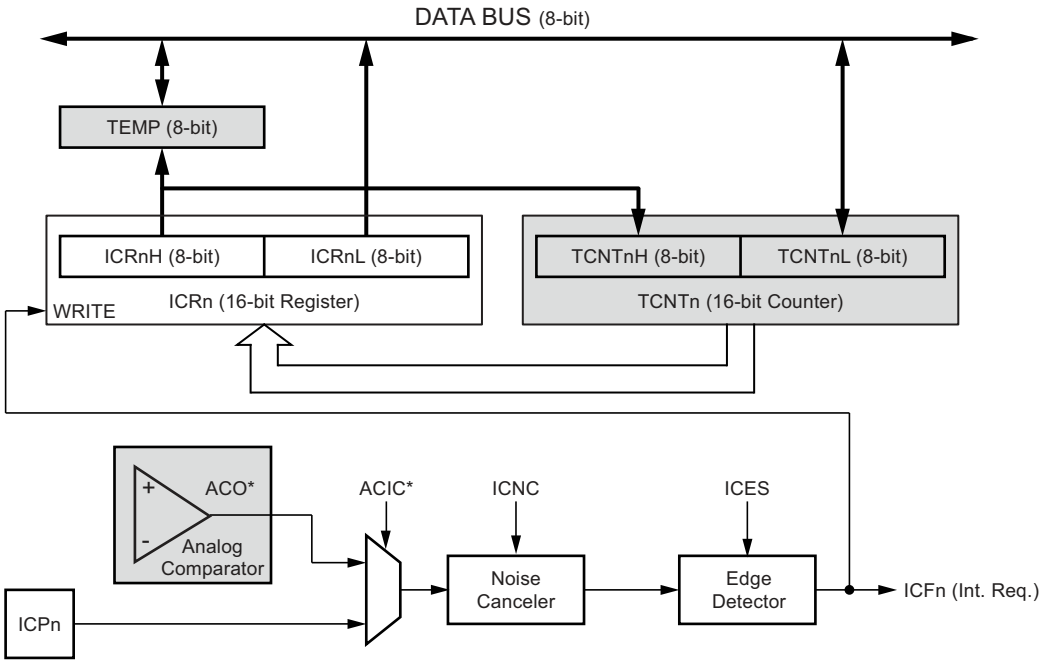
The Timer/Counter overflow flag (TOV1) is set according to the mode of operation selected by the WGM13:0 bits. TOV1 can be used for generating a CPU interrupt.

12.6 Input Capture Unit

The Timer/Counter incorporates an input capture unit that can capture external events and give them a time-stamp indicating time of occurrence. The external signal indicating an event, or multiple events, can be applied via the ICP1 pin or alternatively, via the analog-comparator unit. The time-stamps can then be used to calculate frequency, duty-cycle, and other features of the signal applied. Alternatively the time-stamps can be used for creating a log of the events.

The input capture unit is illustrated by the block diagram shown in Figure 12-3. The elements of the block diagram that are not directly a part of the input capture unit are gray shaded. The small “n” in register and bit names indicates the Timer/Counter number.

Figure 12-3. Input Capture Unit Block Diagram



When a change of the logic level (an event) occurs on the input capture pin (ICP1), alternatively on the analog comparator output (ACO), and this change confirms to the setting of the edge detector, a capture will be triggered. When a capture is triggered, the 16-bit value of the counter (TCNT1) is written to the input capture register (ICR1). The input capture flag (ICF1) is set at the same system clock as the TCNT1 value is copied into ICR1 register.

If enabled (ICIE1 = 1), the input capture flag generates an input capture interrupt. The ICF1 flag is automatically cleared when the interrupt is executed. Alternatively the ICF1 flag can be cleared by software by writing a logical one to its I/O bit location.

Reading the 16-bit value in the input capture register (ICR1) is done by first reading the low byte (ICR1L) and then the high byte (ICR1H). When the low byte is read the high byte is copied into the high byte temporary register (TEMP). When the CPU reads the ICR1H I/O location it will access the TEMP register.

The ICR1 register can only be written when using a waveform generation mode that utilizes the ICR1 register for defining the counter’s TOP value. In these cases the waveform generation mode (WGM13:0) bits must be set before the TOP value can be written to the ICR1 register. When writing the ICR1 register the high byte must be written to the ICR1H I/O location before the low byte is written to ICR1L.

For more information on how to access the 16-bit registers refer to Section 12.3 “Accessing 16-bit Registers” on page 78.

12.6.1 Input Capture Trigger Source

The main trigger source for the input capture unit is the input capture pin (ICP1). Timer/Counter1 can alternatively use the analog comparator output as trigger source for the input capture unit. The analog comparator is selected as trigger source by setting the analog comparator input capture (ACIC) bit in the analog comparator control and status register (ACSR). Be aware that changing trigger source can trigger a capture. The input capture flag must therefore be cleared after the change.

Both the input capture pin (ICP1) and the analog comparator output (ACO) inputs are sampled using the same technique as for the T1 pin (Figure 13-1 on page 102). The edge detector is also identical. However, when the noise canceler is enabled, additional logic is inserted before the edge detector, which increases the delay by four system clock cycles. Note that the input of the noise canceler and edge detector is always enabled unless the Timer/Counter is set in a waveform generation mode that uses ICR1 to define TOP.

An input capture can be triggered by software by controlling the port of the ICP1 pin.

12.6.2 Noise Canceler

The noise canceler improves noise immunity by using a simple digital filtering scheme. The noise canceler input is monitored over four samples, and all four must be equal for changing the output that in turn is used by the edge detector.

The noise canceler is enabled by setting the input capture noise canceler (ICNC1) bit in Timer/Counter control register B (TCCR1B). When enabled the noise canceler introduces additional four system clock cycles of delay from a change applied to the input, to the update of the ICR1 register. The noise canceler uses the system clock and is therefore not affected by the prescaler.

12.6.3 Using the Input Capture Unit

The main challenge when using the input capture unit is to assign enough processor capacity for handling the incoming events. The time between two events is critical. If the processor has not read the captured value in the ICR1 register before the next event occurs, the ICR1 will be overwritten with a new value. In this case the result of the capture will be incorrect.

When using the input capture interrupt, the ICR1 register should be read as early in the interrupt handler routine as possible. Even though the input capture interrupt has relatively high priority, the maximum interrupt response time is dependent on the maximum number of clock cycles it takes to handle any of the other interrupt requests.

Using the input capture unit in any mode of operation when the TOP value (resolution) is actively changed during operation, is not recommended.

Measurement of an external signal's duty cycle requires that the trigger edge is changed after each capture. Changing the edge sensing must be done as early as possible after the ICR1 register has been read. After a change of the edge, the input capture flag (ICF1) must be cleared by software (writing a logical one to the I/O bit location). For measuring frequency only, the clearing of the ICF1 flag is not required (if an interrupt handler is used).

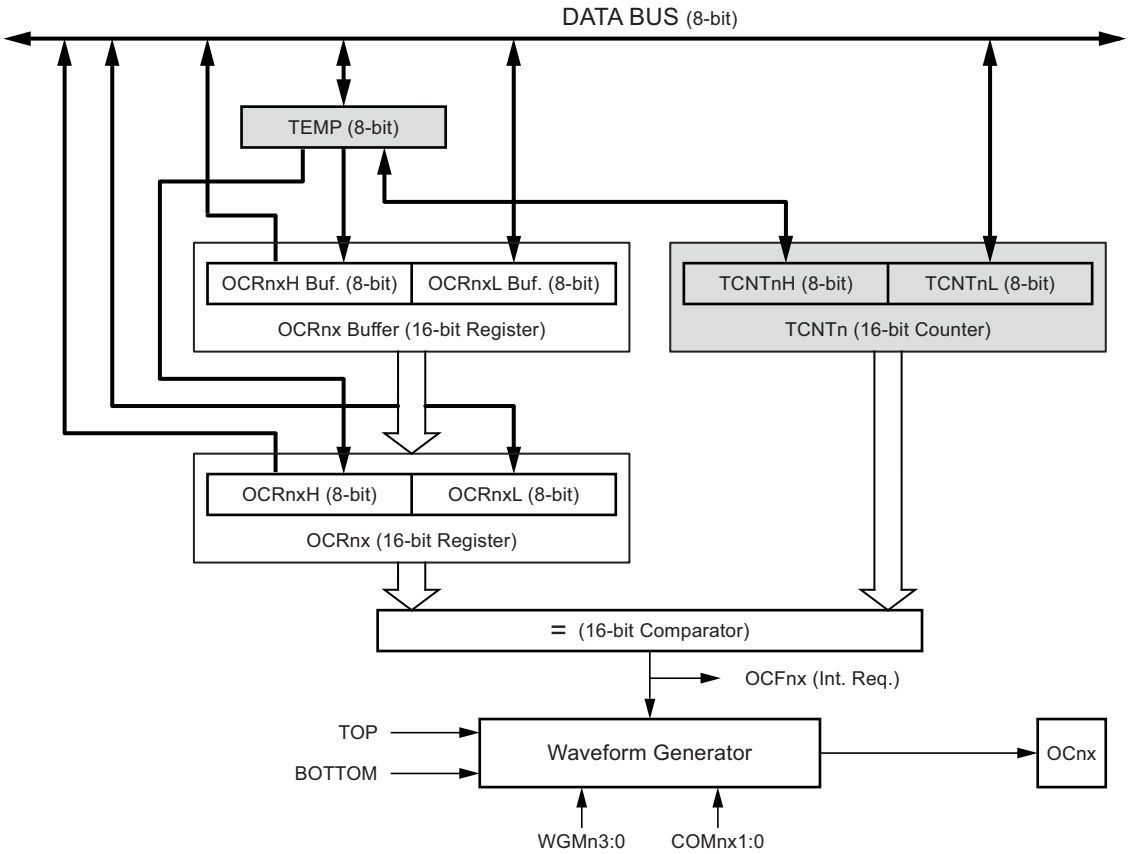
12.7 Output Compare Units

The 16-bit comparator continuously compares TCNT1 with the output compare register (OCR1x). If TCNT equals OCR1x the comparator signals a match. A match will set the output compare flag (OCF1x) at the next timer clock cycle. If enabled (OCIE1x = 1), the output compare flag generates an output compare interrupt. The OCF1x flag is automatically cleared when the interrupt is executed. Alternatively the OCF1x flag can be cleared by software by writing a logical one to its I/O bit location. The waveform generator uses the match signal to generate an output according to operating mode set by the waveform generation mode (WGM13:0) bits and compare output mode (COM1x1:0) bits. The TOP and BOTTOM signals are used by the waveform generator for handling the special cases of the extreme values in some modes of operation (See Section 12.9 "Modes of Operation" on page 87)

A special feature of output compare unit A allows it to define the Timer/Counter TOP value (i.e., counter resolution). In addition to the counter resolution, the TOP value defines the period time for waveforms generated by the waveform generator.

Figure 12-4 shows a block diagram of the output compare unit. The small “n” in the register and bit names indicates the device number (n = 1 for Timer/Counter 1), and the “x” indicates output compare unit (A/B). The elements of the block diagram that are not directly a part of the output compare unit are gray shaded.

Figure 12-4. Output Compare Unit, Block Diagram



The OCR1x register is double buffered when using any of the twelve pulse width modulation (PWM) modes. For the normal and clear timer on compare (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR1x compare register to either TOP or BOTTOM of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCR1x register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR1x buffer register, and if double buffering is disabled the CPU will access the OCR1x directly. The content of the OCR1x (buffer or compare) register is only changed by a write operation (the Timer/Counter does not update this register automatically as the TCNT1 and ICR1 register). Therefore OCR1x is not read via the high byte temporary register (TEMP). However, it is a good practice to read the low byte first as when accessing other 16-bit registers. Writing the OCR1x registers must be done via the TEMP register since the compare of all 16 bits is done continuously. The high byte (OCR1xH) has to be written first. When the high byte I/O location is written by the CPU, the TEMP register will be updated by the value written. Then when the low byte (OCR1xL) is written to the lower eight bits, the high byte will be copied into the upper 8-bits of either the OCR1x buffer or OCR1x compare register in the same system clock cycle.

For more information of how to access the 16-bit registers refer to Section 12.3 “Accessing 16-bit Registers” on page 78.

12.7.1 Force Output Compare

In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a one to the force output compare (FOC1x) bit. Forcing compare match will not set the OCF1x flag or reload/clear the timer, but the OC1x pin will be updated as if a real compare match had occurred (the COM11:0 bits settings define whether the OC1x pin is set, cleared or toggled).

12.7.2 Compare Match Blocking by TCNT1 Write

All CPU writes to the TCNT1 register will block any compare match that occurs in the next timer clock cycle, even when the timer is stopped. This feature allows OCR1x to be initialized to the same value as TCNT1 without triggering an interrupt when the Timer/Counter clock is enabled.

12.7.3 Using the Output Compare Unit

Since writing TCNT1 in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNT1 when using any of the output compare channels, independent of whether the Timer/Counter is running or not. If the value written to TCNT1 equals the OCR1x value, the compare match will be missed, resulting in incorrect waveform generation. Do not write the TCNT1 equal to TOP in PWM modes with variable TOP values. The compare match for the TOP will be ignored and the counter will continue to 0xFFFF. Similarly, do not write the TCNT1 value equal to BOTTOM when the counter is downcounting.

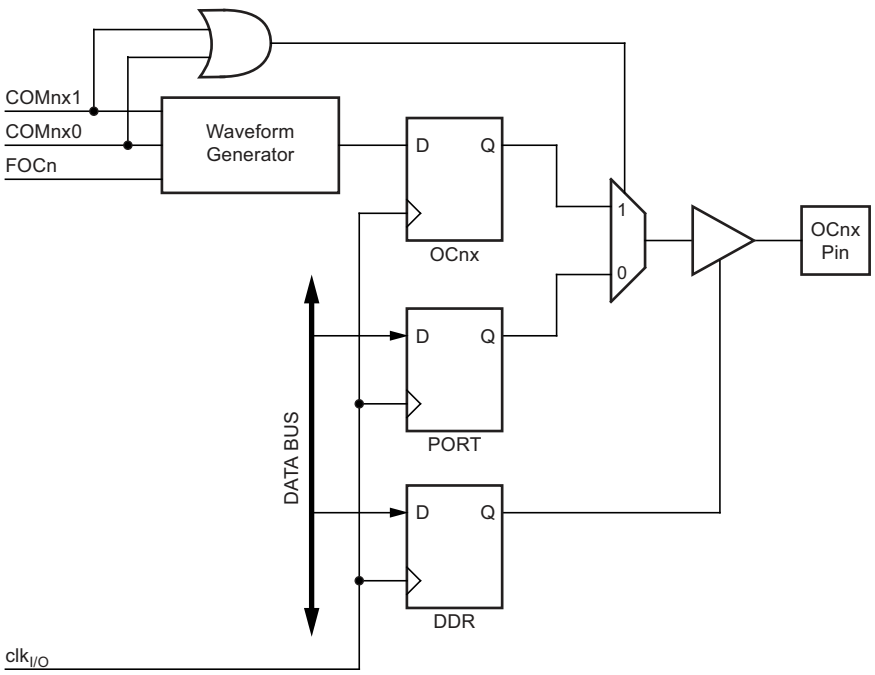
The setup of the OC1x should be performed before setting the data direction register for the port pin to output. The easiest way of setting the OC1x value is to use the force output compare (FOC1x) strobe bits in normal mode. The OC1x register keeps its value even when changing between waveform generation modes.

Be aware that the COM1x1:0 bits are not double buffered together with the compare value. Changing the COM1x1:0 bits will take effect immediately.

12.8 Compare Match Output Unit

The compare output mode (COM1x1:0) bits have two functions. The waveform generator uses the COM1x1:0 bits for defining the output compare (OC1x) state at the next compare match. Secondly the COM1x1:0 bits control the OC1x pin output source. Figure 12-5 shows a simplified schematic of the logic affected by the COM1x1:0 bit setting. The I/O registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O port control registers (DDR and PORT) that are affected by the COM1x1:0 bits are shown. When referring to the OC1x state, the reference is for the internal OC1x register, not the OC1x pin. If a system reset occur, the OC1x register is reset to “0”.

Figure 12-5. Compare Match Output Unit (non-PWM Mode), Schematic



The general I/O port function is overridden by the output compare (OC1x) from the waveform generator if either of the COM1x1:0 bits are set. However, the OC1x pin direction (input or output) is still controlled by the data direction register (DDR) for the port pin. The data direction register bit for the OC1x pin (DDR_OC1x) must be set as output before the OC1x value is visible on the pin. The port override function is generally independent of the waveform generation mode, but there are some exceptions. Refer to Table 12-2 on page 95, Table 12-3 on page 96 and Table 12-4 on page 96 for details.

The design of the output compare pin logic allows initialization of the OC1x state before the output is enabled. Note that some COM1x1:0 bit settings are reserved for certain modes of operation.
See Section 12.11 “Register Description” on page 95

The COM1x1:0 bits have no effect on the input capture unit.

12.8.1 Compare Output Mode and Waveform Generation

The waveform generator uses the COM1x1:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COM1x1:0 = 0 tells the waveform generator that no action on the OC1x register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 12-2 on page 95. For fast PWM mode refer to Table 12-3 on page 96, and for phase correct and phase and frequency correct PWM refer to Table 12-4 on page 96.

A change of the COM1x1:0 bits state will have effect at the first compare match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOC1x strobe bits.

12.9 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the output compare pins, is defined by the combination of the waveform generation mode (WGM13:0) and compare output mode (COM1x1:0) bits. The compare output mode bits do not affect the counting sequence, while the waveform generation mode bits do. The COM1x1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM1x1:0 bits control whether the output should be set, cleared or toggle at a compare match
(See Section 12.8 “Compare Match Output Unit” on page 86).

For detailed timing information refer to Section 12.10 “Timer/Counter Timing Diagrams” on page 93.

12.9.1 Normal Mode

The simplest mode of operation is the normal mode (WGM13:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 16-bit value (MAX = 0xFFFF) and then restarts from the BOTTOM (0x0000). In normal operation the Timer/Counter overflow flag (TOV1) will be set in the same timer clock cycle as the TCNT1 becomes zero. The TOV1 flag in this case behaves like a 17th bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV1 flag, the timer resolution can be increased by software. There are no special cases to consider in the normal mode, a new counter value can be written anytime.

The input capture unit is easy to use in normal mode. However, observe that the maximum interval between the external events must not exceed the resolution of the counter. If the interval between events are too long, the timer overflow interrupt or the prescaler must be used to extend the resolution for the capture unit.

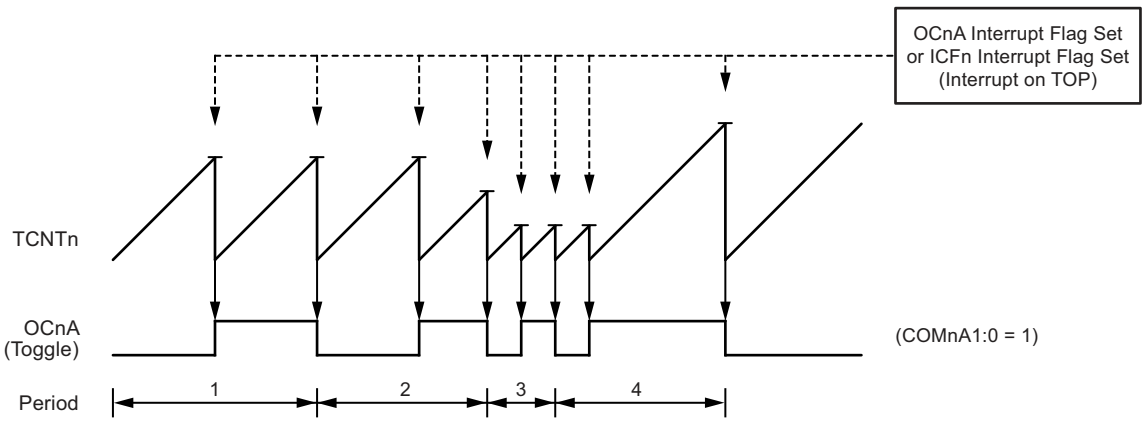
The output compare units can be used to generate interrupts at some given time. Using the output compare to generate waveforms in normal mode is not recommended, since this will occupy too much of the CPU time.

12.9.2 Clear Timer on Compare Match (CTC) Mode

In clear timer on compare or CTC mode (WGM13:0 = 4 or 12), the OCR1A or ICR1 register are used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT1) matches either the OCR1A (WGM13:0 = 4) or the ICR1 (WGM13:0 = 12). The OCR1A or ICR1 define the top value for the counter, hence also its resolution. This mode allows greater control of the compare match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 12-6. The counter value (TCNT1) increases until a compare match occurs with either OCR1A or ICR1, and then counter (TCNT1) is cleared.

Figure 12-6. CTC Mode, Timing Diagram



An interrupt can be generated at each time the counter value reaches the TOP value by either using the OCF1A or ICF1 Flag according to the register used to define the TOP value. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing the TOP to a value close to BOTTOM when the counter is running with none or a low prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written to OCR1A or ICR1 is lower than the current value of TCNT1, the counter will miss the compare match. The counter will then have to count to its maximum value (0xFFFF) and wrap around starting at 0x0000 before the compare match can occur. In many cases this feature is not desirable. An alternative will then be to use the fast PWM mode using OCR1A for defining TOP (WGM13:0 = 15) since the OCR1A then will be double buffered.

For generating a waveform output in CTC mode, the OC1A output can be set to toggle its logical level on each compare match by setting the compare output mode bits to toggle mode (COM1A1:0 = 1). The OC1A value will not be visible on the port pin unless the data direction for the pin is set to output (DDR_OC1A = 1). The waveform generated will have a maximum frequency of $f_{OC1A} = f_{clk\_I/O}/2$ when OCR1A is set to zero (0x0000). The waveform frequency is defined by the following equation:

$$f_{OCnA} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCRnA)}$$

The N variable represents the prescaler factor (1, 8, 64, 256, or 1024).

As for the normal mode of operation, the TOV1 flag is set in the same timer clock cycle that the counter counts from MAX to 0x0000.

12.9.3 Fast PWM Mode

The fast pulse width modulation or fast PWM mode (WGM13:0 = 5, 6, 7, 14, or 15) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. In non-inverting compare output mode, the output compare (OC1x) is set on the compare match between TCNT1 and OCR1x, and cleared at TOP. In inverting compare output mode output is cleared on compare match and set at TOP. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct and phase and frequency correct PWM modes that use dual-slope operation.

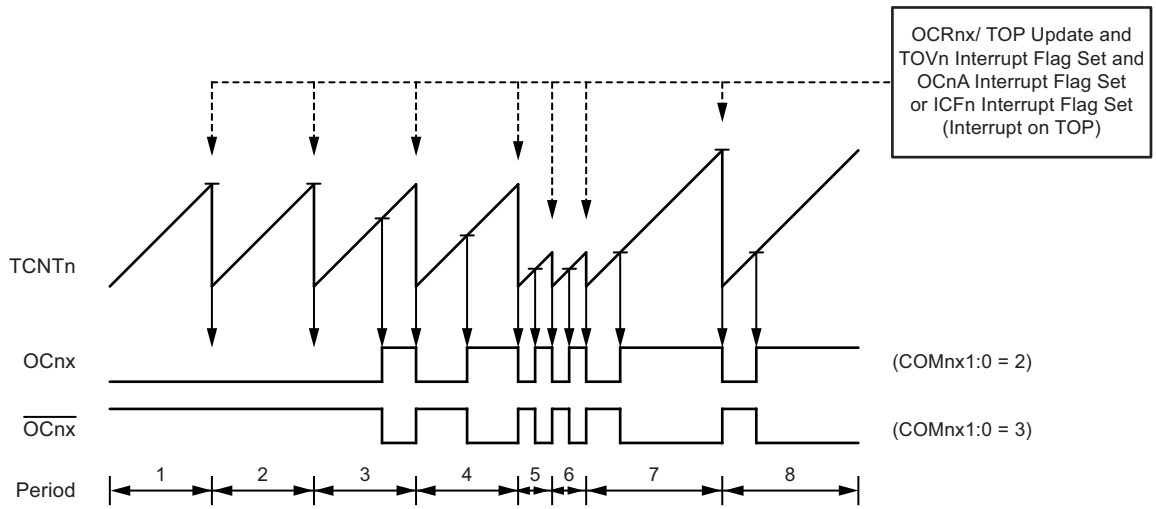
This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), hence reduces total system cost.

The PWM resolution for fast PWM can be fixed to 8-, 9-, or 10-bit, or defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or OCR1A set to 0x0003), and the maximum resolution is 16-bit (ICR1 or OCR1A set to MAX). The PWM resolution in bits can be calculated by using the following equation:

$$R_{FPWM} = \frac{\log(TOP+1)}{\log(2)}$$

In fast PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGM13:0 = 5, 6, or 7), the value in ICR1 (WGM13:0 = 14), or the value in OCR1A (WGM13:0 = 15). The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 12-7. The figure shows fast PWM mode when OCR1A or ICR1 is used to define TOP. The TCNT1 value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT1 slopes represent compare matches between OCR1x and TCNT1. The OC1x interrupt flag will be set when a compare match occurs.

Figure 12-7. Fast PWM Mode, Timing Diagram



The Timer/Counter overflow flag (TOV1) is set each time the counter reaches TOP. In addition the OC1A or ICF1 flag is set at the same timer clock cycle as TOV1 is set when either OCR1A or ICR1 is used for defining the TOP value. If one of the interrupts are enabled, the interrupt handler routine can be used for updating the TOP and compare values.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the compare registers. If the TOP value is lower than any of the compare registers, a compare match will never occur between the TCNT1 and the OCR1x. Note that when using fixed TOP values the unused bits are masked to zero when any of the OCR1x registers are written.

The procedure for updating ICR1 differs from updating OCR1A when used for defining the TOP value. The ICR1 register is not double buffered. This means that if ICR1 is changed to a low value when the counter is running with none or a low prescaler value, there is a risk that the new ICR1 value written is lower than the current value of TCNT1. The result will then be that the counter will miss the compare match at the TOP value. The counter will then have to count to the MAX value (0xFFFF) and wrap around starting at 0x0000 before the compare match can occur. The OCR1A register however, is double buffered. This feature allows the OCR1A I/O location to be written anytime. When the OCR1A I/O location is written the value written will be put into the OCR1A buffer register. The OCR1A compare register will then be updated with the value in the buffer register at the next timer clock cycle the TCNT1 matches TOP. The update is done at the same timer clock cycle as the TCNT1 is cleared and the TOV1 flag is set.

Using the ICR1 register for defining TOP works well when using fixed TOP values. By using ICR1, the OCR1A register is free to be used for generating a PWM output on OC1A. However, if the base PWM frequency is actively changed (by changing the TOP value), using the OCR1A as TOP is clearly a better choice due to its double buffer feature.

In fast PWM mode, the compare units allow generation of PWM waveforms on the OC1x pins. Setting the COM1x1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x1:0 to three (see Table on page 96). The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OC1x). The PWM waveform is generated by setting (or clearing) the OC1x register at the compare match between OCR1x and TCNT1, and clearing (or setting) the OC1x register at the timer clock cycle the counter is cleared (changes from TOP to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot (1 + TOP)}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCR1x register represents special cases when generating a PWM waveform output in the fast PWM mode. If the OCR1x is set equal to BOTTOM (0x0000) the output will be a narrow spike for each TOP+1 timer clock cycle. Setting the OCR1x equal to TOP will result in a constant high or low output (depending on the polarity of the output set by the COM1x1:0 bits.)

A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC1A to toggle its logical level on each compare match (COM1A1:0 = 1). This applies only if OCR1A is used to define the TOP value (WGM13:0 = 15). The waveform generated will have a maximum frequency of $f_{OC1A} = f_{clk\_I/O}/2$ when OCR1A is set to zero (0x0000). This feature is similar to the OC1A toggle in CTC mode, except the double buffer feature of the output compare unit is enabled in the fast PWM mode.

12.9.4 Phase Correct PWM Mode

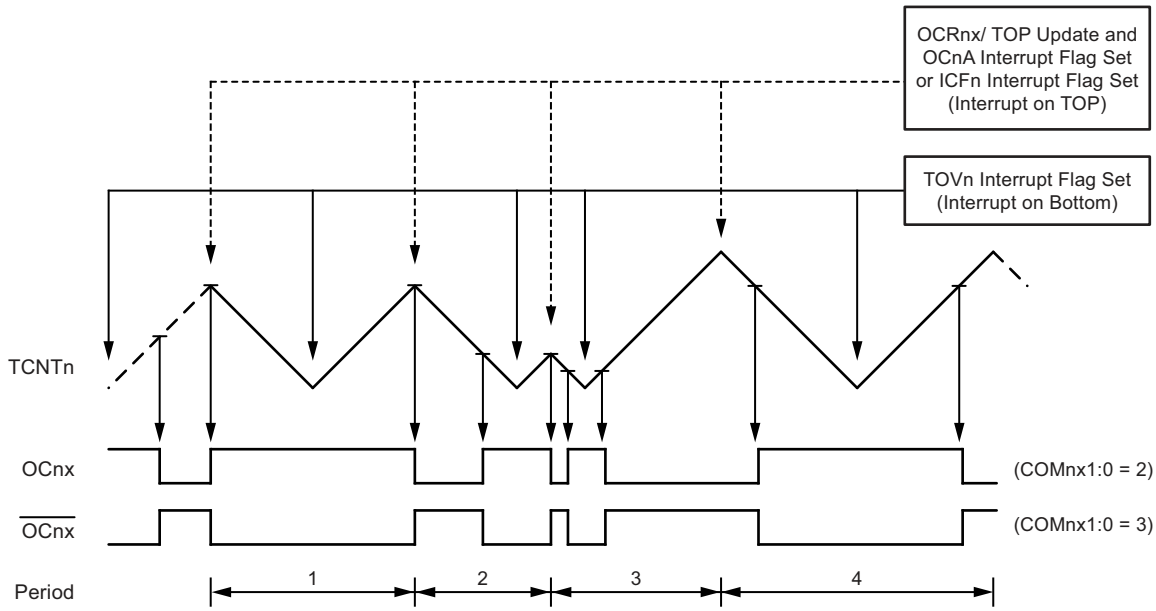
The phase correct pulse width modulation or phase correct PWM mode (WGM13:0 = 1, 2, 3, 10, or 11) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is, like the phase and frequency correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting compare output mode, the output compare (OC1x) is cleared on the compare match between TCNT1 and OCR1x while upcounting, and set on the compare match while downcounting. In inverting output compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

The PWM resolution for the phase correct PWM mode can be fixed to 8-, 9-, or 10-bit, or defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or OCR1A set to 0x0003), and the maximum resolution is 16-bit (ICR1 or OCR1A set to MAX). The PWM resolution in bits can be calculated by using the following equation:

$$R_{PCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In phase correct PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGM13:0 = 1, 2, or 3), the value in ICR1 (WGM13:0 = 10), or the value in OCR1A (WGM13:0 = 11). The counter has then reached the TOP and changes the count direction. The TCNT1 value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 12-8 on page 91. The figure shows phase correct PWM mode when OCR1A or ICR1 is used to define TOP. The TCNT1 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT1 slopes represent compare matches between OCR1x and TCNT1. The OC1x interrupt flag will be set when a compare match occurs.

Figure 12-8. Phase Correct PWM Mode, Timing Diagram



The Timer/Counter overflow flag (TOV1) is set each time the counter reaches BOTTOM. When either OCR1A or ICR1 is used for defining the TOP value, the OC1A or ICF1 flag is set accordingly at the same timer clock cycle as the OCR1x registers are updated with the double buffer value (at TOP). The interrupt flags can be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the compare registers. If the TOP value is lower than any of the compare registers, a compare match will never occur between the TCNT1 and the OCR1x. Note that when using fixed TOP values, the unused bits are masked to zero when any of the OCR1x registers are written. As the third period shown in Figure 12-8 illustrates, changing the TOP actively while the Timer/Counter is running in the phase correct mode can result in an unsymmetrical output. The reason for this can be found in the time of update of the OCR1x register. Since the OCR1x update occurs at TOP, the PWM period starts and ends at TOP. This implies that the length of the falling slope is determined by the previous TOP value, while the length of the rising slope is determined by the new TOP value. When these two values differ the two slopes of the period will differ in length. The difference in length gives the unsymmetrical result on the output.

It is recommended to use the phase and frequency correct mode instead of the phase correct mode when changing the TOP value while the Timer/Counter is running. When using a static TOP value there are practically no differences between the two modes of operation.

In phase correct PWM mode, the compare units allow generation of PWM waveforms on the OC1x pins. Setting the COM1x1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x1:0 to three (See Table 12-4 on page 96). The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OC1x). The PWM waveform is generated by setting (or clearing) the OC1x register at the compare match between OCR1x and TCNT1 when the counter increments, and clearing (or setting) the OC1x register at compare match between OCR1x and TCNT1 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnxPCPWM} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot TOP}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCR1x register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR1x is set equal to BOTTOM the output will be continuously low and if set equal to TOP the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values. If OCR1A is used to define the TOP value (WGM13:0 = 11) and COM1A1:0 = 1, the OC1A output will toggle with a 50% duty cycle.

12.9.5 Phase and Frequency Correct PWM Mode

The phase and frequency correct pulse width modulation, or phase and frequency correct PWM mode (WGM13:0 = 8 or 9) provides a high resolution phase and frequency correct PWM waveform generation option. The phase and frequency correct PWM mode is, like the phase correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting compare output mode, the output compare (OC1x) is cleared on the compare match between TCNT1 and OCR1x while upcounting, and set on the compare match while downcounting. In inverting compare output mode, the operation is inverted. The dual-slope operation gives a lower maximum operation frequency compared to the single-slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

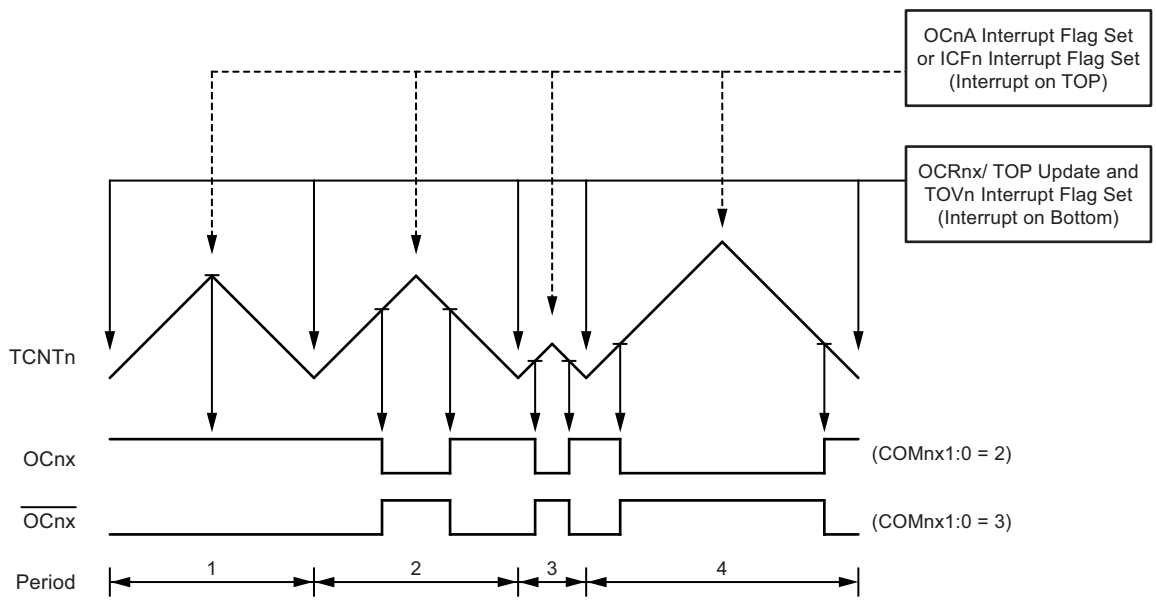
The main difference between the phase correct, and the phase and frequency correct PWM mode is the time the OCR1x register is updated by the OCR1x buffer register, (see Figure 12-8 on page 91 and Figure 12-9).

The PWM resolution for the phase and frequency correct PWM mode can be defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or OCR1A set to 0x0003), and the maximum resolution is 16-bit (ICR1 or OCR1A set to MAX). The PWM resolution in bits can be calculated using the following equation:

$$R_{PFCPWM} = \frac{\log(TOP+1)}{\log(2)}$$

In phase and frequency correct PWM mode the counter is incremented until the counter value matches either the value in ICR1 (WGM13:0 = 8), or the value in OCR1A (WGM13:0 = 9). The counter has then reached the TOP and changes the count direction. The TCNT1 value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct and frequency correct PWM mode is shown on Figure 12-9. The figure shows phase and frequency correct PWM mode when OCR1A or ICR1 is used to define TOP. The TCNT1 value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT1 slopes represent compare matches between OCR1x and TCNT1. The OC1x interrupt flag will be set when a compare match occurs.

Figure 12-9. Phase and Frequency Correct PWM Mode, Timing Diagram



The Timer/Counter overflow flag (TOV1) is set at the same timer clock cycle as the OCR1x registers are updated with the double buffer value (at BOTTOM). When either OCR1A or ICR1 is used for defining the TOP value, the OC1A or ICF1 flag set when TCNT1 has reached TOP. The interrupt flags can then be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the compare registers. If the TOP value is lower than any of the compare registers, a compare match will never occur between the TCNT1 and the OCR1x.

As Figure 12-9 on page 92 shows the output generated is, in contrast to the phase correct mode, symmetrical in all periods. Since the OCR1x registers are updated at BOTTOM, the length of the rising and the falling slopes will always be equal. This gives symmetrical output pulses and is therefore frequency correct.

Using the ICR1 register for defining TOP works well when using fixed TOP values. By using ICR1, the OCR1A register is free to be used for generating a PWM output on OC1A. However, if the base PWM frequency is actively changed by changing the TOP value, using the OCR1A as TOP is clearly a better choice due to its double buffer feature.

In phase and frequency correct PWM mode, the compare units allow generation of PWM waveforms on the OC1x pins. Setting the COM1x1:0 bits to 0b10 will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x1:0 to 0b11 (See Table 12-4 on page 96). The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OC1x). The PWM waveform is generated by setting (or clearing) the OC1x register at the compare match between OCR1x and TCNT1 when the counter increments, and clearing (or setting) the OC1x register at compare match between OCR1x and TCNT1 when the counter decrements.

The PWM frequency for the output when using phase and frequency correct PWM can be calculated by the following equation:

$$f_{OCnxPFCPWM} = \frac{f_{\text{clk\_I/O}}}{2 \cdot N \cdot TOP}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCR1x register represents special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR1x is set equal to BOTTOM the output will be continuously low and if set equal to TOP the output will be set to high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values. If OCR1A is used to define the TOP value and COM1A1:0 = 1, the OC1A output will toggle with a 50% duty cycle.

12.10 Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock (clk_{T1}) is therefore shown as a clock enable signal in the following figures. The figures include information on when interrupt flags are set, and when the OCR1x register is updated with the OCR1x buffer value (only for modes utilizing double buffering). Figure 12-10 shows a timing diagram for the setting of OCF1x.

Figure 12-10.Timer/Counter Timing Diagram, Setting of OCF1x, no Prescaling

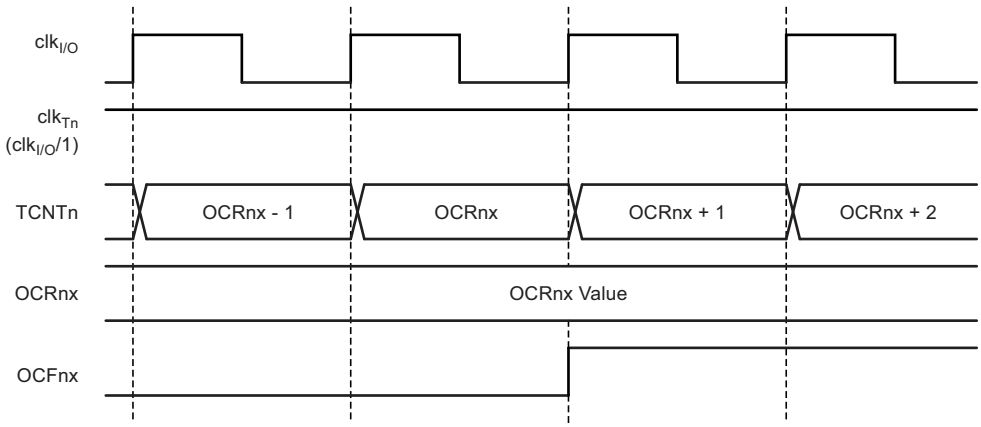


Figure 12-11 shows the same timing data, but with the prescaler enabled.

Figure 12-11.Timer/Counter Timing Diagram, Setting of OCF1x, with Prescaler ($f_{clk\_I/O}/8$)

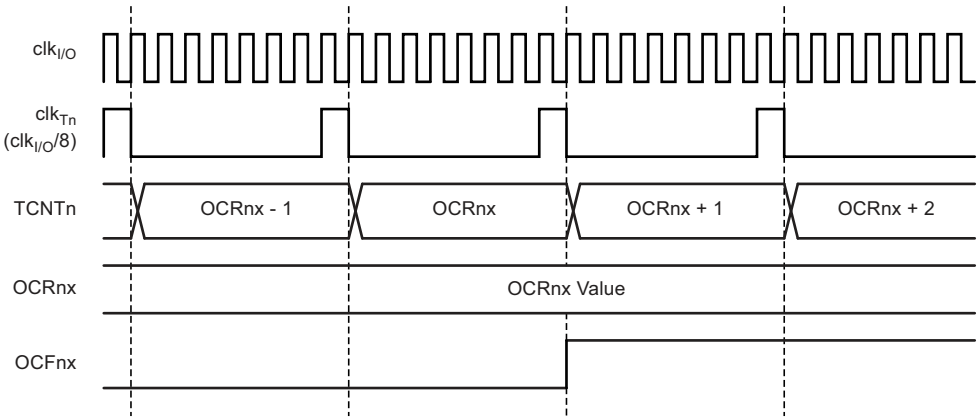


Figure 12-12 shows the count sequence close to TOP in various modes. When using phase and frequency correct PWM mode the OCR1x register is updated at BOTTOM. The timing diagrams will be the same, but TOP should be replaced by BOTTOM, TOP-1 by BOTTOM+1 and so on. The same renaming applies for modes that set the TOV1 flag at BOTTOM.

Figure 12-12.Timer/Counter Timing Diagram, no Prescaling

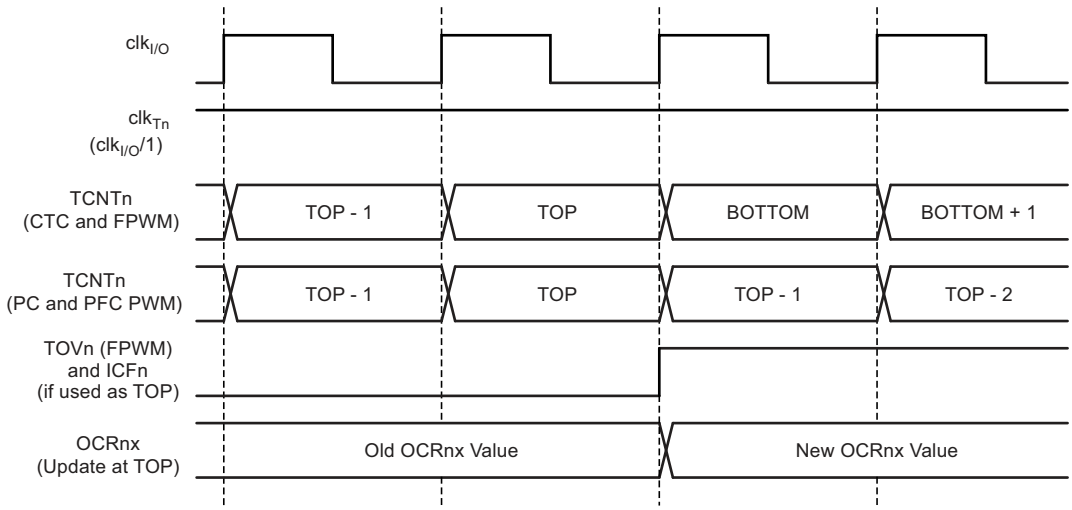
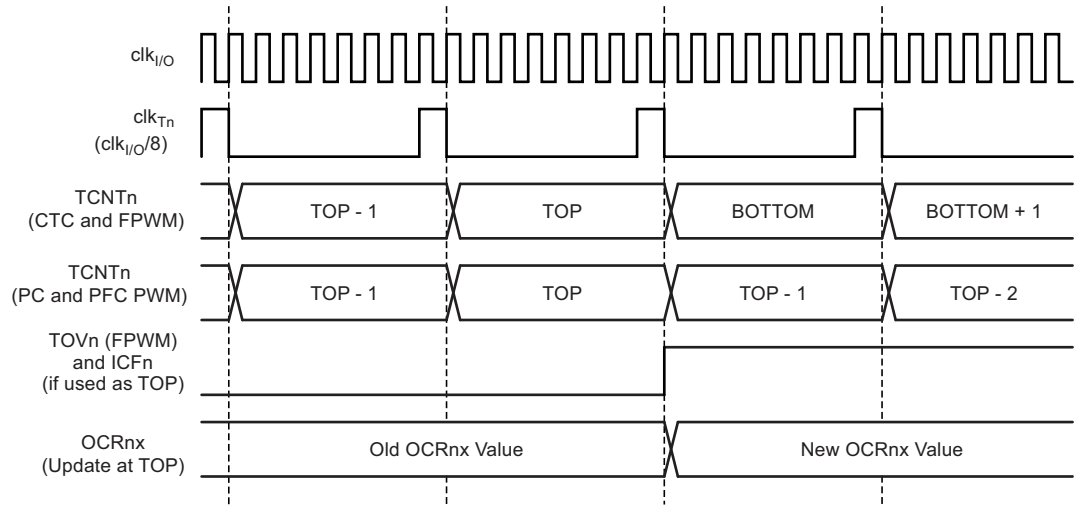


Figure 12-13 shows the same timing data, but with the prescaler enabled.

Figure 12-13.Timer/Counter Timing Diagram, with Prescaler ($f_{clk\_I/O}$/8)



12.11 Register Description

12.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – COM1A1:0: Compare Output Mode for Channel A**

- **Bit 5:4 – COM1B1:0: Compare Output Mode for Channel B**

The COM1A1:0 and COM1B1:0 control the output compare pins (OC1A and OC1B respectively) behavior. If one or both of the COM1A1:0 bits are written to one, the OC1A output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COM1B1:0 bit are written to one, the OC1B output overrides the normal port functionality of the I/O pin it is connected to. However, note that the data direction register (DDR) bit corresponding to the OC1A or OC1B pin must be set in order to enable the output driver.

When the OC1A or OC1B is connected to the pin, the function of the COM1x1:0 bits is dependent of the WGM13:0 bits setting. Table 12-2 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to a normal or a CTC mode (non-PWM).

Table 12-2. Compare Output Mode, non-PWM

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on compare match.
1	0	Clear OC1A/OC1B on compare match (set output to low level).
1	1	Set OC1A/OC1B on compare match (set output to high level).

Table 12-3 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to the fast PWM mode.

Table 12-3. Compare Output Mode, Fast PWM(1)

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 14 or 15: Toggle OC1A on compare match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on compare match, set OC1A/OC1B at TOP
1	1	Set OC1A/OC1B on compare match, clear OC1A/OC1B at TOP

Note: 1. A special case occurs when OCR1A/OCR1B equals TOP and COM1A1/COM1B1 is set. In this case the compare match is ignored, but the set or clear is done at TOP. See Section 12.9.3 “Fast PWM Mode” on page 88 for more details.

Table 12-4 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to the phase correct or the phase and frequency correct, PWM mode.

Table 12-4. Compare Output Mode, Phase Correct and Phase & Frequency Correct PWM(1)

COM1A1 COM1B1	COM1A0 COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 8, 9, 10 or 11: Toggle OC1A on compare match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on compare match when up-counting. Set OC1A/OC1B on compare match when downcounting.
1	1	Set OC1A/OC1B on compare match when up-counting. Clear OC1A/OC1B on compare match when downcounting.

Note: 1. A special case occurs when OCR1A/OCR1B equals TOP and COM1A1/COM1B1 is set. See Section 12.9.4 “Phase Correct PWM Mode” on page 90 for more details.

- **Bits 3,2 – Reserved**

These bits are reserved and will always read zero.

- **Bit 1:0 – WGM11:0: Waveform Generation Mode**

Combined with the WGM13:2 bits found in the TCCR1B register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see Table 12-5.

Table 12-5. Waveform Generation Mode Bit Description

Mode	WGM 13	WGM 12	WGM 11	WGM 10	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, phase correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, phase correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, phase correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, phase & frequency correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, phase & frequency correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, phase correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, phase correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

Modes of operation supported by the Timer/Counter unit are: normal mode (counter), clear timer on compare match (CTC) mode, and three types of pulse width modulation (PWM) modes.(See Section 12.9 “Modes of Operation” on page 87).

12.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ICNC1: Input Capture Noise Canceler**

Setting this bit (to one) activates the input capture noise canceler. When the noise canceler is activated, the input from the input capture pin (ICP1) is filtered. The filter function requires four successive equal valued samples of the ICP1 pin for changing its output. The input capture is therefore delayed by four oscillator cycles when the noise canceler is enabled.

- **Bit 6 – ICES1: Input Capture Edge Select**

This bit selects which edge on the input capture pin (ICP1) that is used to trigger a capture event. When the ICES1 bit is written to zero, a falling (negative) edge is used as trigger, and when the ICES1 bit is written to one, a rising (positive) edge will trigger the capture.

When a capture is triggered according to the ICES1 setting, the counter value is copied into the input capture register (ICR1). The event will also set the input capture flag (ICF1), and this can be used to cause an input capture interrupt, if this interrupt is enabled.

When the ICR1 is used as TOP value (see description of the WGM13:0 bits located in the TCCR1A and the TCCR1B register), the ICP1 is disconnected and consequently the input capture function is disabled.

- **Bit 5 – Reserved Bit**

This bit is reserved for future use. For ensuring compatibility with future devices, this bit must be written to zero when TCCR1B is written.

- **Bit 4:3 – WGM13:2: Waveform Generation Mode**

See TCCR1A register description.

- **Bit 2:0 – CS12:0: Clock Select**

The three clock select bits select the clock source to be used by the Timer/Counter, see Figure 12-10 on page 93 and Figure 12-11 on page 94.

Table 12-6. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

If external pin modes are used for the Timer/Counter1, transitions on the T1 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

12.11.3 TCCR1C – Timer/Counter1 Control Register C

Bit	7	6	5	4	3	2	1	0	
	FOC1A	**FOC1B**	–	–	–	–	–	–	**TCCR1C**
Read/Write	R/W	R/W	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC1A: Force Output Compare for Channel A**

- **Bit 6 – FOC1B: Force Output Compare for Channel B**

The FOC1A/FOC1B bits are only active when the WGM13:0 bits specifies a non-PWM mode. However, for ensuring compatibility with future devices, these bits must be set to zero when TCCR1A is written when operating in a PWM mode. When writing a logical one to the FOC1A/FOC1B bit, an immediate compare match is forced on the waveform generation unit. The OC1A/OC1B output is changed according to its COM1x1:0 bits setting. Note that the FOC1A/FOC1B bits are implemented as strobes. Therefore it is the value present in the COM1x1:0 bits that determine the effect of the forced compare.

A FOC1A/FOC1B strobe will not generate any interrupt nor will it clear the timer in clear timer on compare match (CTC) mode using OCR1A as TOP.

The FOC1A/FOC1B bits are always read as zero.

- **Bits 5..0 – Reserved**

These bits are reserved and will always read zero.

12.11.4 TCNT1H and TCNT1L – Timer/Counter1

Bit	7	6	5	4	3	2	1	0	
	TCNT1[15:8]								TCNT1H
	TCNT1[7:0]								TCNT1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The two Timer/Counter I/O locations (TCNT1H and TCNT1L, combined TCNT1) give direct access, both for read and for write operations, to the Timer/Counter unit 16-bit counter. To ensure that both the high and low bytes are read and written simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers.
See Section 12.3 “Accessing 16-bit Registers” on page 78

Modifying the counter (TCNT1) while the counter is running introduces a risk of missing a compare match between TCNT1 and one of the OCR1x registers.

Writing to the TCNT1 register blocks (removes) the compare match on the following timer clock for all compare units.

12.11.5 OCR1AH and OCR1AL – Output Compare Register 1 A

Bit	7	6	5	4	3	2	1	0	
	OCR1A[15:8]								OCR1AH
	OCR1A[7:0]								OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

12.11.6 OCR1BH and OCR1BL – Output Compare Register 1 B

Bit	7	6	5	4	3	2	1	0	
	OCR1B[15:8]								OCR1BH
	OCR1B[7:0]								OCR1BL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The output compare registers contain a 16-bit value that is continuously compared with the counter value (TCNT1). A match can be used to generate an output compare interrupt, or to generate a waveform output on the OC1x pin.

The output compare registers are 16-bit in size. To ensure that both the high and low bytes are written simultaneously when the CPU writes to these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. See Section 12.3 “Accessing 16-bit Registers” on page 78

12.11.7 ICR1H and ICR1L – Input Capture Register 1

Bit	7	6	5	4	3	2	1	0	
	ICR1[15:8]								ICR1H
	ICR1[7:0]								ICR1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The input capture is updated with the counter (TCNT1) value each time an event occurs on the ICP1 pin (or optionally on the analog comparator output for Timer/Counter1). The input capture can be used for defining the counter TOP value.

The input capture register is 16-bit in size. To ensure that both the high and low bytes are read simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. See Section 12.3 “Accessing 16-bit Registers” on page 78

12.11.8 TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7, 6 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 5 – ICIE1: Timer/Counter1, Input Capture Interrupt Enable**

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 input capture interrupt is enabled. The corresponding interrupt vector (See Section 9. “Interrupts” on page 44) is executed when the ICF1 flag, located in TIFR1, is set.

- **Bit 4, 3 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 2 – OCIE1B: Timer/Counter1, Output Compare B Match Interrupt Enable**

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 output compare B match interrupt is enabled. The corresponding interrupt vector (See Section 9. “Interrupts” on page 44) is executed when the OCF1B flag, located in TIFR1, is set.

- **Bit 1 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable**

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 output compare A match interrupt is enabled. The corresponding interrupt vector (See Section 9. “Interrupts” on page 44) is executed when the OCF1A flag, located in TIFR1, is set.

- **Bit 0 – TOIE1: Timer/Counter1, Overflow Interrupt Enable**

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 overflow interrupt is enabled. The corresponding interrupt vector (See Section 8.4 “Watchdog Timer” on page 40) is executed when the TOV1 flag, located in TIFR1, is set.

12.11.9 TIFR1 – Timer/Counter1 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7, 6 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 5 – ICF1: Timer/Counter1, Input Capture Flag**

This flag is set when a capture event occurs on the ICP1 pin. When the input capture register (ICR1) is set by the WGM13:0 to be used as the TOP value, the ICF1 flag is set when the counter reaches the TOP value.

ICF1 is automatically cleared when the input capture interrupt vector is executed. Alternatively, ICF1 can be cleared by writing a logic one to its bit location.

- **Bit 4, 3 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 2 – OCF1B: Timer/Counter1, Output Compare B Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the output compare register B (OCR1B).

Note that a forced output compare (FOC1B) strobe will not set the OCF1B flag.

OCF1B is automatically cleared when the output compare match B interrupt vector is executed. Alternatively, OCF1B can be cleared by writing a logic one to its bit location.

- **Bit 1 – OCF1A: Timer/Counter1, Output Compare A Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the output compare register A (OCR1A).

Note that a forced output compare (FOC1A) strobe will not set the OCF1A flag.

OCF1A is automatically cleared when the output compare match A interrupt vector is executed. Alternatively, OCF1A can be cleared by writing a logic one to its bit location.

- **Bit 0 – TOV1: Timer/Counter1, Overflow Flag**

The setting of this flag is dependent of the WGM13:0 bits setting. In normal and CTC modes, the TOV1 flag is set when the timer overflows. Refer to Table 12-5 on page 97 for the TOV1 flag behavior when using another WGM13:0 bit setting.

TOV1 is automatically cleared when the Timer/Counter1 overflow interrupt vector is executed. Alternatively, TOV1 can be cleared by writing a logic one to its bit location.

13. Timer/Counter0 and Timer/Counter1 Prescalers

Section 11. “8-bit Timer/Counter0” on page 68 and Section 12. “16-bit Timer/Counter1 with PWM” on page 76 share the same prescaler module, but the Timer/Counters can have different prescaler settings. The description below applies to both Timer/Counter1 and Timer/Counter0.

13.1 Internal Clock Source

The Timer/Counter can be clocked directly by the system clock (by setting the CSn2:0 = 1). This provides the fastest operation, with a maximum Timer/Counter clock frequency equal to system clock frequency ($f_{CLK\_I/O}$). Alternatively, one of four taps from the prescaler can be used as a clock source. The prescaled clock has a frequency of either $f_{CLK\_I/O}/8$, $f_{CLK\_I/O}/64$, $f_{CLK\_I/O}/256$, or $f_{CLK\_I/O}/1024$.

13.2 Prescaler Reset

The prescaler is free running, i.e., operates independently of the clock select logic of the Timer/Counter, and it is shared by Timer/Counter1 and Timer/Counter0. Since the prescaler is not affected by the Timer/Counter’s clock select, the state of the prescaler will have implications for situations where a prescaled clock is used. One example of prescaling artifacts occurs when the timer is enabled and clocked by the prescaler (CSn2:0 = 0b010, 0b011, 0b100, or 0b101). The number of system clock cycles from when the timer is enabled to the first count occurs can be from 1 to N+1 system clock cycles, where N equals the prescaler divisor (8, 64, 256, or 1024).

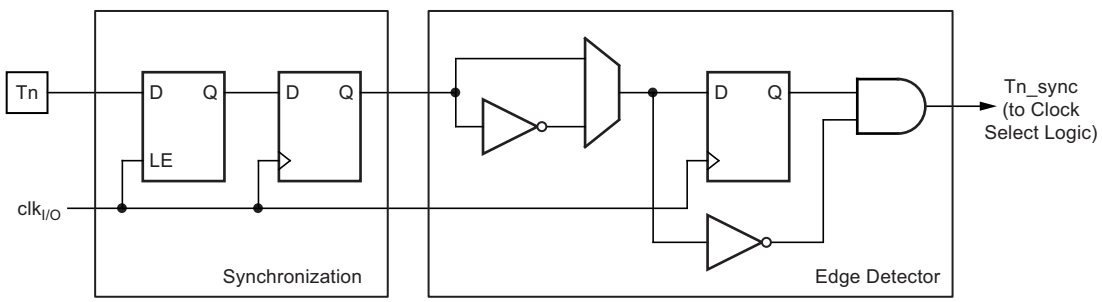
It is possible to use the prescaler reset for synchronizing the Timer/Counter to program execution. However, care must be taken if the other Timer/Counter that shares the same prescaler also uses prescaling. A prescaler reset will affect the prescaler period for all Timer/Counters it is connected to.

13.3 External Clock Source

An external clock source applied to the T1/T0 pin can be used as Timer/Counter clock (clk_{T1}/clk_{T0}). The T1/T0 pin is sampled once every system clock cycle by the pin synchronization logic. The synchronized (sampled) signal is then passed through the edge detector. Figure 13-1 shows a functional equivalent block diagram of the T1/T0 synchronization and edge detector logic. The registers are clocked at the positive edge of the internal system clock ($clk_{I/O}$). The latch is transparent in the high period of the internal system clock.

The edge detector generates one clk_{T1}/clk_{T0} pulse for each positive (CSn2:0 = 7) or negative (CSn2:0 = 6) edge it detects.

Figure 13-1. T1/T0 Pin Sampling



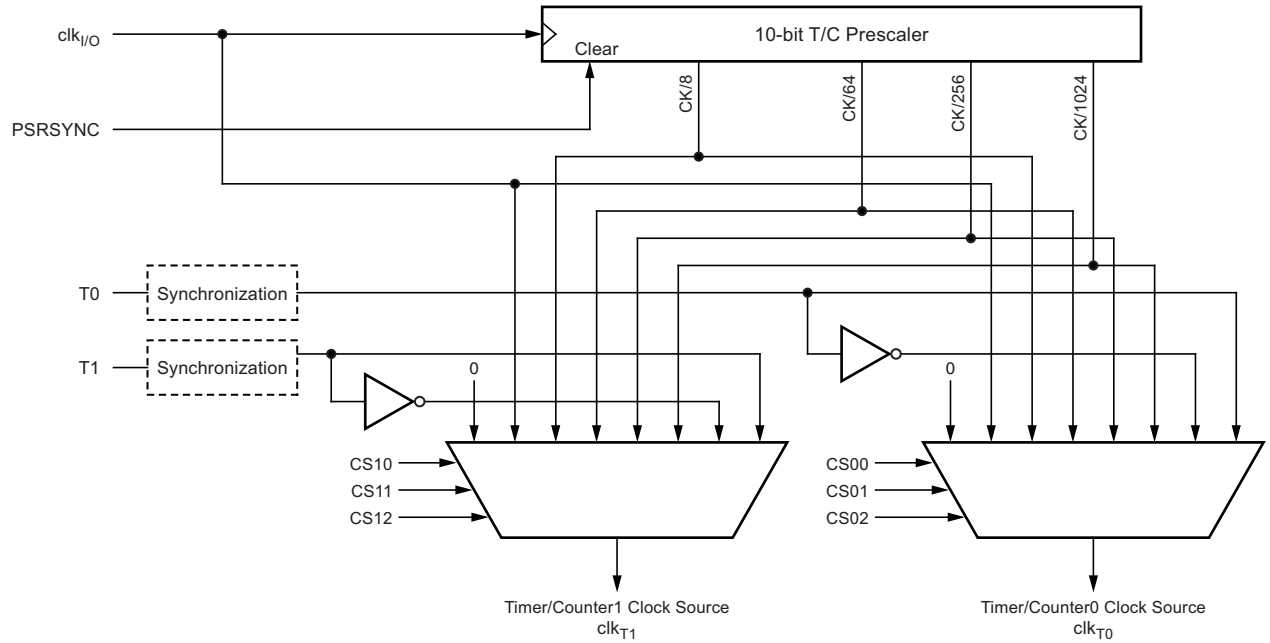
The synchronization and edge detector logic introduces a delay of 2.5 to 3.5 system clock cycles from an edge has been applied to the T1/T0 pin to the counter is updated.

Enabling and disabling of the clock input must be done when T1/T0 has been stable for at least one system clock cycle, otherwise it is a risk that a false Timer/Counter clock pulse is generated.

Each half period of the external clock applied must be longer than one system clock cycle to ensure correct sampling. The external clock must be guaranteed to have less than half the system clock frequency ($f_{ExtClk} < f_{clk\_I/O}/2$) given a 50% duty cycle. Since the edge detector uses sampling, the maximum frequency of an external clock it can detect is half the sampling frequency (nyquist sampling theorem). However, due to variation of the system clock frequency and duty cycle caused by oscillator source tolerances, it is recommended that maximum frequency of an external clock source is less than $f_{clk\_I/O}/2.5$.

An external clock source can not be prescaled.

Figure 13-2. Prescaler for Timer/Counter0 and Timer/Counter1(1)



Note: 1. The synchronization logic on the input pins **(T1/T0)** is shown in Figure 13-1.

13.4 Register Description

13.4.1 GTCCR – General Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0	
	TSM	–	–	–	–	–	–	PSRSYNC	GTCCR
Read/Write	R/W	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – TSM: Timer/Counter Synchronization Mode**

Writing the TSM bit to one activates the Timer/Counter synchronization mode. In this mode, the value that is written to the PSRSYNC bit is kept, hence keeping the corresponding prescaler reset signals asserted. This ensures that the corresponding Timer/Counters are halted and can be configured to the same value without the risk of one of them advancing during configuration. When the TSM bit is written to zero, the PSRSYNC bit are cleared by hardware, and the Timer/Counters start counting simultaneously.

- **Bits 6..1 – Reserved**

These bits are reserved and will always read zero.

- **Bit 0 – PSRSYNC: Prescaler Reset**

When this bit is one, Timer/Counter1 and Timer/Counter0 prescaler will be reset. This bit is normally cleared immediately by hardware, except if the TSM bit is set. Note that Timer/Counter1 and Timer/Counter0 share the same prescaler and a reset of this prescaler will affect both timers.

14. Serial Peripheral Interface – SPI

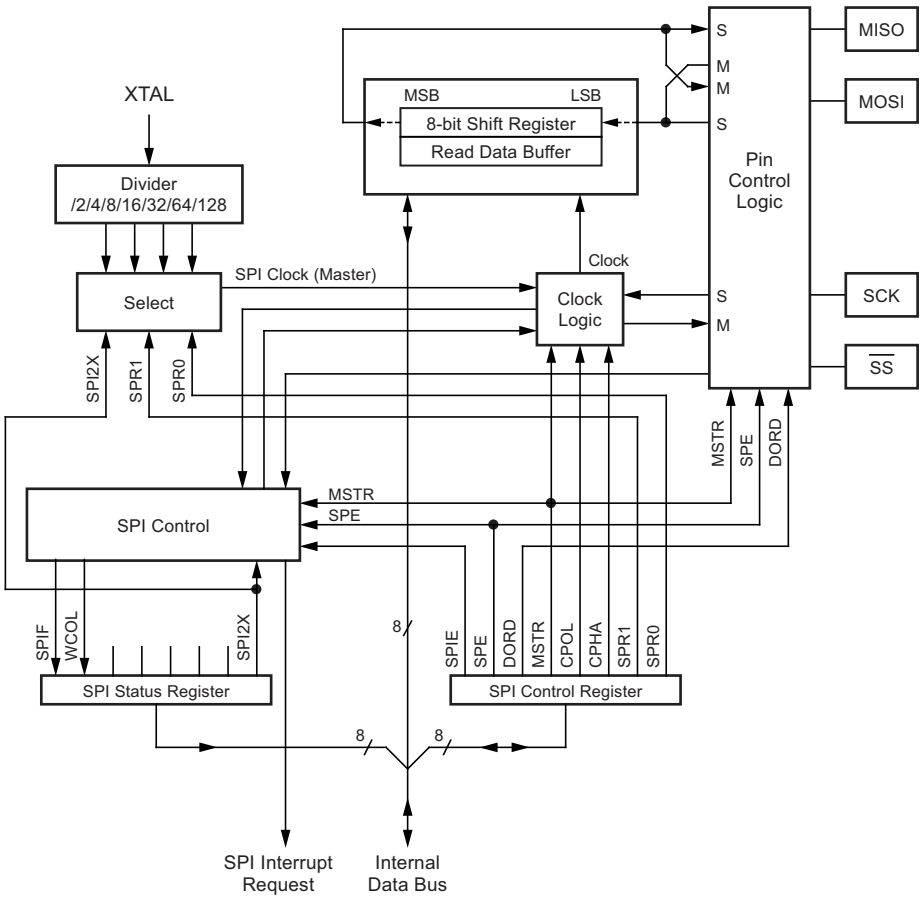
14.1 Features

- Full-duplex, three-wire synchronous data transfer
- Master or slave operation
- LSB first or MSB first data transfer
- Seven programmable bit rates
- End of transmission interrupt flag
- Write collision flag protection
- Wake-up from idle mode
- Double speed (CK/2) master SPI mode

14.2 Overview

The serial peripheral interface (SPI) allows high-speed synchronous data transfer between the Atmel® ATtiny88 and peripheral devices or between several AVR® devices.

Figure 14-1. SPI Block Diagram(1)



Note: 1. Refer to Figure 1-1 on page 3, and Table 10-5 on page 59 for SPI pin placement.

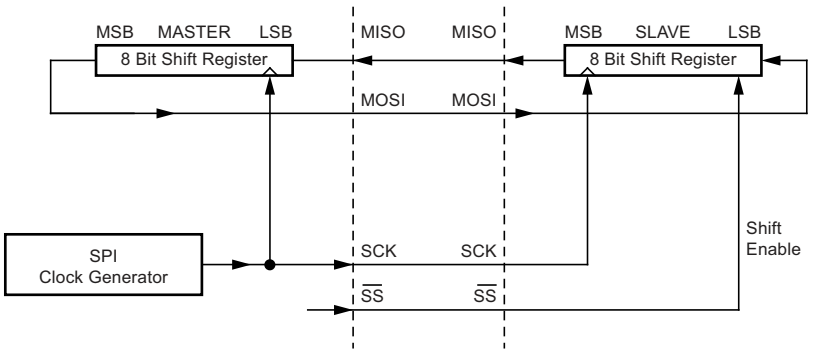
The PRSPI bit in Section 7.4.3 “PRR – Power Reduction Register” on page 36 must be written to zero to enable the SPI module.

The interconnection between master and slave CPUs with SPI is shown in Figure 14-2. The system consists of two shift registers, and a master clock generator. The SPI master initiates the communication cycle when pulling low the slave select $\overline{SS}$ pin of the desired slave. Master and slave prepare the data to be sent in their respective shift registers, and the master generates the required clock pulses on the SCK line to interchange data. Data is always shifted from master to slave on the master out – slave In, MOSI, line, and from slave to master on the master In – slave out, MISO, line. After each data packet, the master will synchronize the slave by pulling high the slave select, $\overline{SS}$, line.

When configured as a master, the SPI interface has no automatic control of the $\overline{SS}$ line. This must be handled by user software before communication can start. When this is done, writing a byte to the SPI data register starts the SPI clock generator, and the hardware shifts the eight bits into the slave. After shifting one byte, the SPI clock generator stops, setting the end of transmission flag (SPIF). If the SPI interrupt enable bit (SPIE) in the SPCR register is set, an interrupt is requested. The master may continue to shift the next byte by writing it into SPDR, or signal the end of packet by pulling high the slave select, $\overline{SS}$ line. The last incoming byte will be kept in the buffer register for later use.

When configured as a slave, the SPI interface will remain sleeping with MISO tri-stated as long as the $\overline{SS}$ pin is driven high. In this state, software may update the contents of the SPI data register, SPDR, but the data will not be shifted out by incoming clock pulses on the SCK pin until the $\overline{SS}$ pin is driven low. As one byte has been completely shifted, the end of transmission flag, SPIF is set. If the SPI interrupt enable bit, SPIE, in the SPCR register is set, an interrupt is requested. The slave may continue to place new data to be sent into SPDR before reading the incoming data. The last incoming byte will be kept in the buffer register for later use.

Figure 14-2. SPI Master-slave Interconnection



The system is single buffered in the transmit direction and double buffered in the receive direction. This means that bytes to be transmitted cannot be written to the SPI data register before the entire shift cycle is completed. When receiving data, however, a received character must be read from the SPI data register before the next character has been completely shifted in. Otherwise, the first byte is lost.

In SPI slave mode, the control logic will sample the incoming signal of the SCK pin. To ensure correct sampling of the clock signal, the frequency of the SPI clock should never exceed $f_{osc}/4$.

When the SPI is enabled, the data direction of the MOSI, MISO, SCK, and $\overline{SS}$ pins is overridden according to Table 14-1. For more details on automatic port overrides, refer to Section 10.3 “Alternate Port Functions” on page 56.

Table 14-1. SPI Pin Overrides(1)

Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User defined	Input
MISO	Input	User defined
SCK	User defined	Input
$\overline{SS}$	User defined	Input

Note: 1. See Section 10.3.2 “Alternate Functions of Port B” on page 59 for a detailed description of how to define the direction of the user defined SPI pins.

The following code examples show how to initialize the SPI as a master and how to perform a simple transmission. DDR_SPI in the examples must be replaced by the actual data direction register controlling the SPI pins. DD_MOSI, DD_MISO and DD_SCK must be replaced by the actual data direction bits for these pins. E.g. if MOSI is placed on pin PB5, replace DD_MOSI with DDB5 and DDR_SPI with DDRB.

Assembly Code Example[1]

```
SPI_MasterInit:
        ; Set MOSI and SCK output, all others input
        ldi     r17,(1<<DD_MOSI)|(1<<DD_SCK)
        out     DDR_SPI,r17
        ; Enable SPI, Master, set clock rate fck/16
        ldi     r17,(1<<SPE)|(1<<MSTR)|(1<<SPR0)
        out     SPCR,r17
        ret

SPI_MasterTransmit:
        ; Start transmission of data (r16)
        out     SPDR,r16
Wait_Transmit:
        ; Wait for transmission complete
        sbis    SPSR,SPIF
        rjmp    Wait_Transmit
        ret
```

Note: 1. See Section 3.2 “About Code Examples” on page 8.

C Code Example[1]

```
void SPI_MasterInit(void)
{
        /* Set MOSI and SCK output, all others input */
        DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK);
        /* Enable SPI, Master, set clock rate fck/16 */
        SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}

void SPI_MasterTransmit(char cData)
{
        /* Start transmission */
        SPDR = cData;
        /* Wait for transmission complete */
        while(!(SPSR & (1<<SPIF)))
                ;
}
```

Note: 1. See Section 3.2 “About Code Examples” on page 8.

The following code examples show how to initialize the SPI as a slave and how to perform a simple reception.

Assembly Code Example[1]

```
SPI_SlaveInit:
        ; Set MISO output, all others input
        ldi     r17,(1<<DD_MISO)
        out     DDR_SPI,r17
        ; Enable SPI
        ldi     r17,(1<<SPE)
        out     SPCR,r17
        ret

SPI_SlaveReceive:
        ; Wait for reception complete
        sbis    SPSR,SPIF
        rjmp    SPI_SlaveReceive
        ; Read received data and return
        in      r16,SPDR
        ret
```

Note: 1. See Section 3.2 “About Code Examples” on page 8.

C Code Example[1]

```
void SPI_SlaveInit(void)
{
        /* Set MISO output, all others input */
        DDR_SPI = (1<<DD_MISO);
        /* Enable SPI */
        SPCR = (1<<SPE);
}

char SPI_SlaveReceive(void)
{
        /* Wait for reception complete */
        while(!(SPSR & (1<<SPIF)))
                ;
        /* Return Data Register */
        return SPDR;
}
```

14.3 $\overline{SS}$ Pin Functionality

14.3.1 Slave Mode

When the SPI is configured as a slave, the slave select ($\overline{SS}$) pin is always input. When $\overline{SS}$ is held low, the SPI is activated, and MISO becomes an output if configured so by the user. All other pins are inputs. When $\overline{SS}$ is driven high, all pins are inputs, and the SPI is passive, which means that it will not receive incoming data. Note that the SPI logic will be reset once the $\overline{SS}$ pin is driven high.

The $\overline{SS}$ pin is useful for packet/byte synchronization to keep the slave bit counter synchronous with the master clock generator. When the $\overline{SS}$ pin is driven high, the SPI slave will immediately reset the send and receive logic, and drop any partially received data in the shift register.

14.3.2 Master Mode

When the SPI is configured as a master (MSTR in SPCR is set), the user can determine the direction of the $\overline{SS}$ pin.

If $\overline{SS}$ is configured as an output, the pin is a general output pin which does not affect the SPI system. Typically, the pin will be driving the $\overline{SS}$ pin of the SPI slave.

If $\overline{SS}$ is configured as an input, it must be held high to ensure master SPI operation. If the $\overline{SS}$ pin is driven low by peripheral circuitry when the SPI is configured as a master with the $\overline{SS}$ pin defined as an input, the SPI system interprets this as another master selecting the SPI as a slave and starting to send data to it. To avoid bus contention, the SPI system takes the following actions:

1. The MSTR bit in SPCR is cleared and the SPI system becomes a slave. As a result of the SPI becoming a slave, the MOSI and SCK pins become inputs.
2. The SPIF flag in SPSR is set, and if the SPI interrupt is enabled, and the I-bit in SREG is set, the interrupt routine will be executed.

Thus, when interrupt-driven SPI transmission is used in master mode, and there exists a possibility that $\overline{SS}$ is driven low, the interrupt should always check that the MSTR bit is still set. If the MSTR bit has been cleared by a slave select, it must be set by the user to re-enable SPI master mode.

14.4 Data Modes

There are four combinations of SCK phase and polarity with respect to serial data, which are determined by control bits CPHA and CPOL. The SPI data transfer formats are shown in Figure 14-3 and Figure 14-4.

Figure 14-3. SPI Transfer Format with CPHA = 0

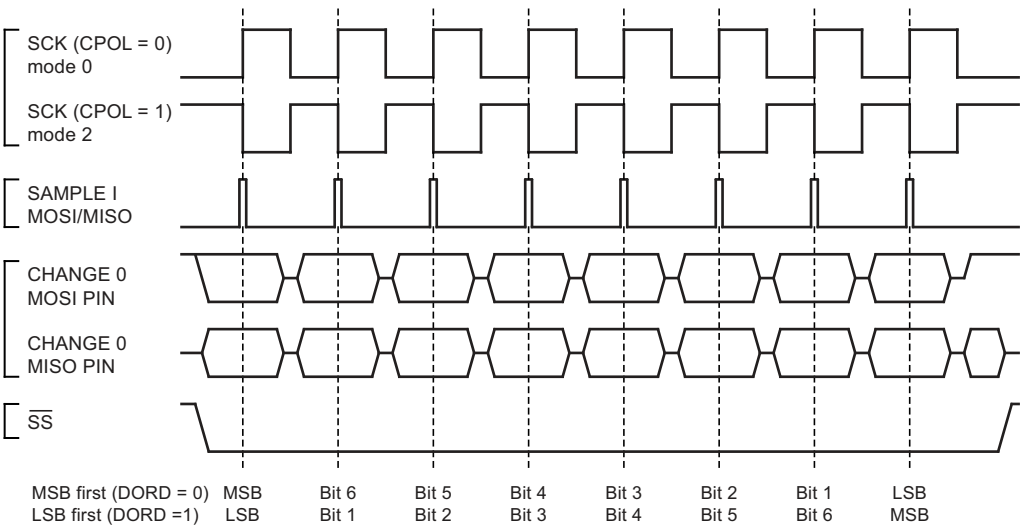
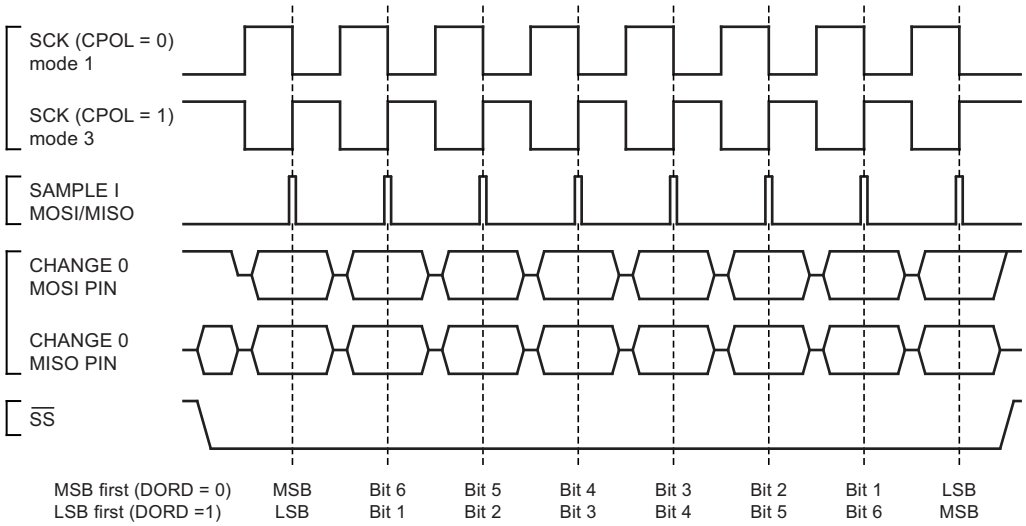


Figure 14-4. SPI Transfer Format with CPHA = 1



Data bits are shifted out and latched in on opposite edges of the SCK signal, ensuring sufficient time for data signals to stabilize. This is clearly seen by summarizing Table 14-3 on page 110 and Table 14-4 on page 110, as done in Table 14-2 below.

Table 14-2. Setting SPI Mode using Control Bits CPOL and CPHA

CPOL	CPHA	SPI Mode	Leading Edge	Trailing eDge
0	0	0	Sample (rising)	Setup (falling)
0	1	1	Setup (rising)	Sample (falling)
1	0	2	Sample (falling)	Setup (rising)
1	1	3	Setup (falling)	Sample (rising)

14.5 Register Description

14.5.1 SPCR – SPI Control Register

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – SPIE: SPI Interrupt Enable**

This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR register is set and the if the global interrupt enable bit in SREG is set.

- **Bit 6 – SPE: SPI Enable**

When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

- **Bit 5 – DORD: Data Order**

When the DORD bit is written to one, the LSB of the data word is transmitted first.

When the DORD bit is written to zero, the MSB of the data word is transmitted first.

- **Bit 4 – MSTR: Master/Slave Select**

This bit selects master SPI mode when written to one, and slave SPI mode when written logic zero. If $\overline{SS}$ is configured as an input and is driven low while MSTR is set, MSTR will be cleared, and SPIF in SPSR will become set. The user will then have to set MSTR to re-enable SPI master mode.

- **Bit 3 – CPOL: Clock Polarity**

When this bit is written to one, SCK is high when idle. When CPOL is written to zero, SCK is low when idle. Refer to Figure 14-3 and Figure 14-4 on page 109 for an example. The CPOL functionality is summarized below:

Table 14-3. CPOL Functionality

CPOL	Leading Edge	Trailing Edge
0	Rising	Falling
1	Falling	Rising

- **Bit 2 – CPHA: Clock Phase**

The settings of the Clock Phase bit (CPHA) determine if data is sampled on the leading (first) or trailing (last) edge of SCK. Refer to Figure 14-3 on page 108 and Figure 14-4 for an example. The CPOL functionality is summarized below:

Table 14-4. CPHA Functionality

CPHA	Leading Edge	Trailing Edge
0	Sample	Setup
1	Setup	Sample

- **Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0**

These two bits control the SCK rate of the device configured as a master. SPR1 and SPR0 have no effect on the slave. The relationship between SCK and the oscillator clock frequency f_{osc} is shown in the following table:

Table 14-5. Relationship Between SCK and the Oscillator Frequency

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

14.5.2 SPSR – SPI Status Register

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – SPIF: SPI Interrupt Flag**

When a serial transfer is complete, the SPIF flag is set. An interrupt is generated if SPIE in SPCR is set and global interrupts are enabled. If $\overline{SS}$ is an input and is driven low when the SPI is in master mode, this will also set the SPIF flag. SPIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, the SPIF bit is cleared by first reading the SPI status register with SPIF set, then accessing the SPI data register (SPDR).

- **Bit 6 – WCOL: Write COLlision Flag**

The WCOL bit is set if the SPI data register (SPDR) is written during a data transfer. The WCOL bit (and the SPIF bit) are cleared by first reading the SPI status register with WCOL set, and then accessing the SPI data register.

- **Bit 5..1 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 0 – SPI2X: Double SPI Speed Bit**

When this bit is written logic one the SPI speed (SCK frequency) will be doubled when the SPI is in master mode (see Table 14-5 on page 110). This means that the minimum SCK period will be two CPU clock periods. When the SPI is configured as slave, the SPI is only guaranteed to work at $f_{osc}/4$ or lower.

The SPI interface on the Atmel® ATtiny88 is also used for program memory and EEPROM downloading or uploading. See Section 20.8 “Serial Downloading” on page 179 for serial programming and verification.

14.5.3 SPDR – SPI Data Register

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

The SPI data register is a read/write register used for data transfer between the register file and the SPI shift register. Writing to the register initiates data transmission. Reading the register causes the shift register receive buffer to be read.

15. 2-Wire Serial Interface

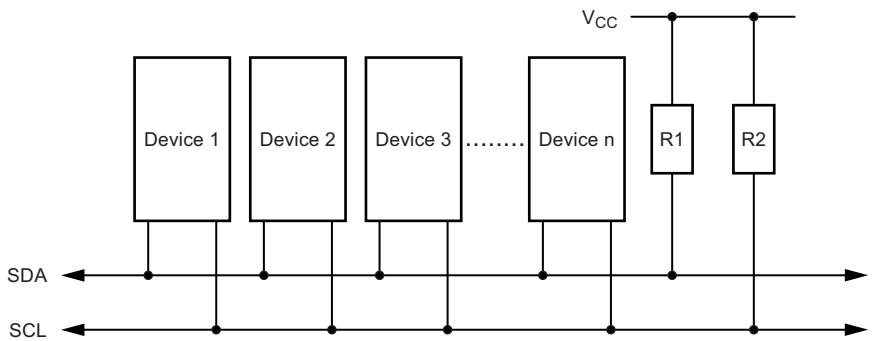
15.1 Features

- Simple yet powerful and flexible communication interface, only two bus lines needed
- Both master and slave operation supported
- Device can operate as transmitter or receiver
- 7-bit address space allows up to 128 different slave addresses
- Multi-master arbitration support
- Data transfer speed up to 400kHz in slave mode
- Slew-rate limited output drivers
- Noise suppression circuitry rejects spikes on bus lines
- Fully programmable slave address with general call support
- Address recognition causes wake-up when AVR® is in Sleep mode
- Compatible with philips I^2C protocol

15.2 2-wire Serial Interface Bus Definition

The 2-wire serial interface (TWI) is ideally suited for typical microcontroller applications. The TWI protocol allows the systems designer to interconnect up to 128 different devices using only two bi-directional bus lines, one for clock (SCL) and one for data (SDA). The only external hardware needed to implement the bus is a single pull-up resistor for each of the TWI bus lines. All devices connected to the bus have individual addresses, and mechanisms for resolving bus contention are inherent in the TWI protocol.

Figure 15-1. TWI Bus Interconnection



15.2.1 TWI Terminology

The following definitions are frequently encountered in this section.

Table 15-1. TWI Terminology

Term	Description
Master	The device that initiates and terminates a transmission and generates the SCL clock.
Slave	The device addressed by a master.
Transmitter	The device placing data on the bus.
Receiver	The device reading data from the bus.

The PRTWI bit in Section 7.4.3 “PRR – Power Reduction Register” on page 36 must be written to zero to enable the 2-wire serial interface.

15.2.2 Electrical Interconnection

As depicted in Figure 15-1 on page 112, both bus lines are connected to the positive supply voltage through pull-up resistors. The bus drivers of all TWI-compliant devices are open-drain or open-collector. This implements a wired-AND function which is essential to the operation of the interface. A low level on a TWI bus line is generated when one or more TWI devices output a zero. A high level is output when all TWI devices tri-state their outputs, allowing the pull-up resistors to pull the line high. Note that all AVR® devices connected to the TWI bus must be powered in order to allow any bus operation.

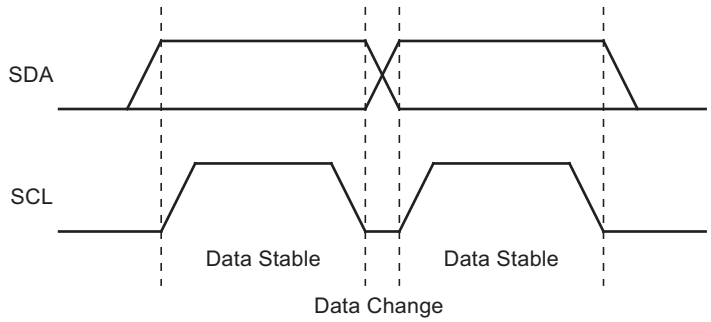
The number of devices that can be connected to the bus is only limited by the bus capacitance limit of 400 pF and the 7-bit slave address space. A detailed specification of the electrical characteristics of the TWI is given in Section 21.7 “2-wire Serial Interface Characteristics” on page 188. Two different sets of specifications are presented there, one relevant for bus speeds below 100kHz, and one valid for bus speeds up to 400kHz.

15.3 Data Transfer and Frame Format

15.3.1 Transferring Bits

Each data bit transferred on the TWI bus is accompanied by a pulse on the clock line. The level of the data line must be stable when the clock line is high. The only exception to this rule is for generating start and stop conditions.

Figure 15-2. Data Validity

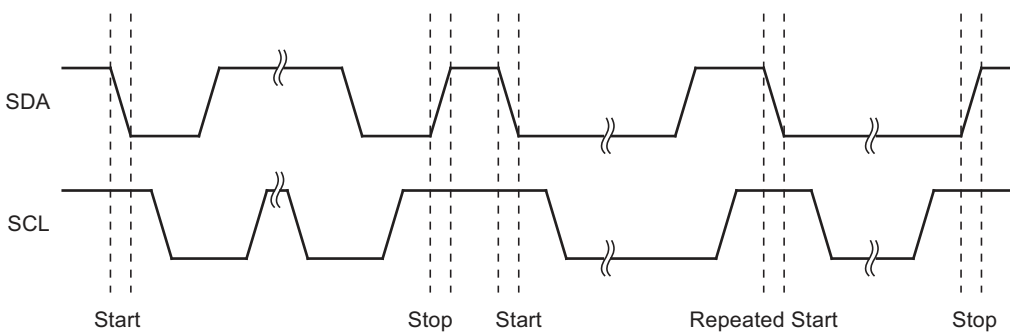


15.3.2 START and STOP Conditions

The master initiates and terminates a data transmission. The transmission is initiated when the master issues a START condition on the bus, and it is terminated when the master issues a STOP condition. Between a START and a STOP condition, the bus is considered busy, and no other master should try to seize control of the bus. A special case occurs when a new START condition is issued between a START and STOP condition. This is referred to as a REPEATED START condition, and is used when the master wishes to initiate a new transfer without relinquishing control of the bus. After a REPEATED START, the bus is considered busy until the next STOP. This is identical to the START behavior, and therefore START is used to describe both START and REPEATED START for the remainder of this datasheet, unless otherwise noted.

As depicted below, START and STOP conditions are signalled by changing the level of the SDA line when the SCL line is high.

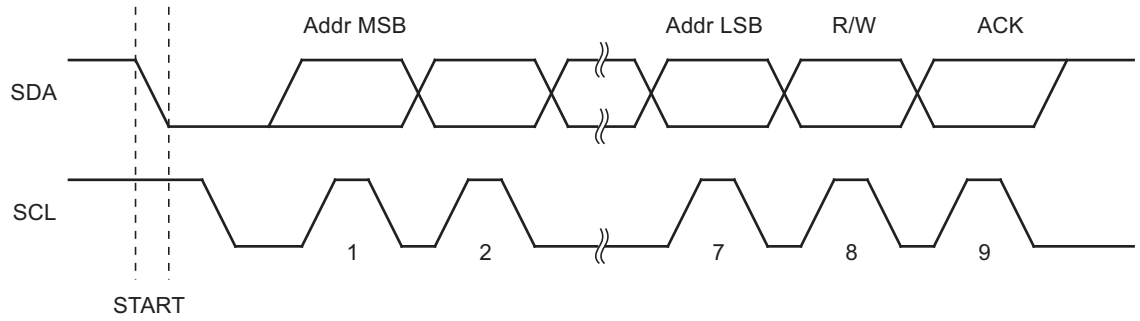
Figure 15-3. START, REPEATED START and STOP conditions



15.3.3 Address Packet Format

All address packets transmitted on the TWI bus are 9 bits long, consisting of 7 address bits, one READ/WRITE control bit and an acknowledge bit. If the READ/WRITE bit is set, a read operation is to be performed, otherwise a write operation should be performed. When a slave recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle. If the addressed slave is busy, or for some other reason can not service the master's request, the SDA line should be left high in the ACK clock cycle. The master can then transmit a STOP condition, or a REPEATED START condition to initiate a new transmission. An address packet consisting of a slave address and a READ or a WRITE bit is called SLA+R or SLA+W, respectively.

Figure 15-4. Address Packet Format



The MSB of the address byte is transmitted first. Slave addresses can freely be allocated by the designer, but the address 0000 000 is reserved for a general call.

When a general call is issued, all slaves should respond by pulling the SDA line low in the ACK cycle. A general call is used when a master wishes to transmit the same message to several slaves in the system. When the general call address followed by a Write bit is transmitted on the bus, all slaves set up to acknowledge the general call will pull the SDA line low in the ack cycle.

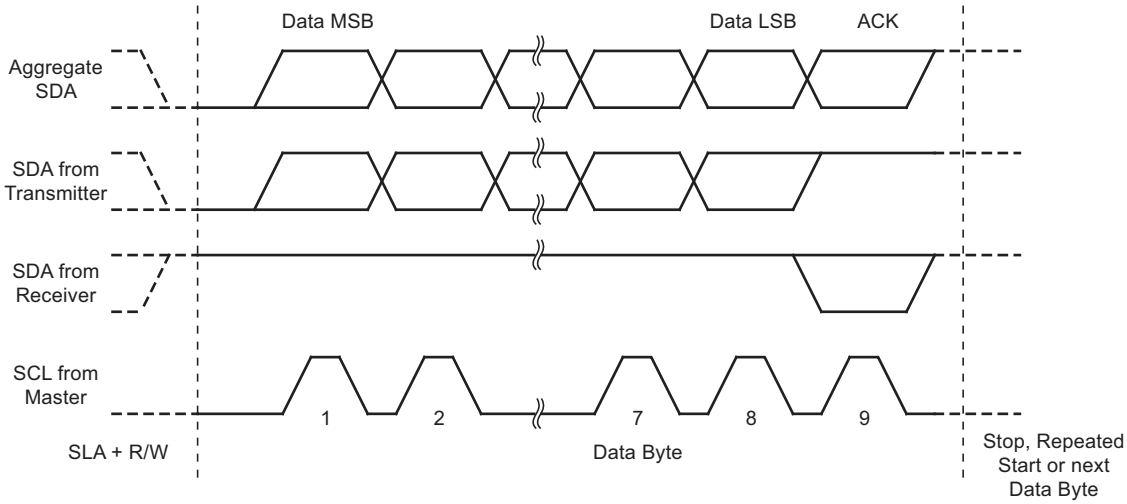
The following data packets will then be received by all the slaves that acknowledged the general call. Note that transmitting the general call address followed by a read bit is meaningless, as this would cause contention if several slaves started transmitting different data.

All addresses of the format 1111 xxx should be reserved for future purposes.

15.3.4 Data Packet Format

All data packets transmitted on the TWI bus are nine bits long, consisting of one data byte and an acknowledge bit. During a data transfer, the master generates the clock and the START and STOP conditions, while the receiver is responsible for acknowledging the reception. An acknowledge (ACK) is signalled by the receiver pulling the SDA line low during the ninth SCL cycle. If the receiver leaves the SDA line high, a NACK is signalled. When the receiver has received the last byte, or for some reason cannot receive any more bytes, it should inform the transmitter by sending a NACK after the final byte. The MSB of the data byte is transmitted first.

Figure 15-5. Data Packet Format

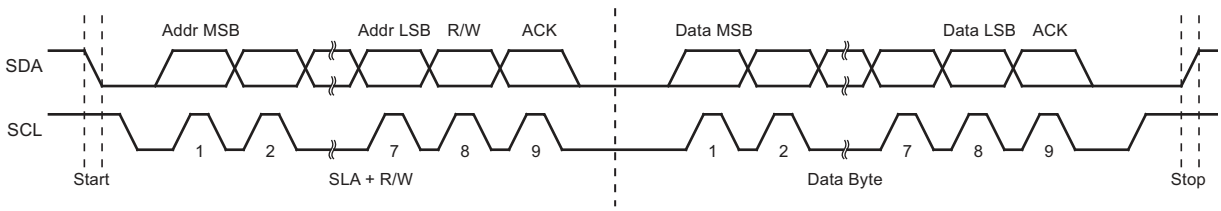


15.3.5 Combining Address and Data Packets into a Transmission

A transmission basically consists of a START condition, a SLA+R/W, one or more data packets and a STOP condition. An empty message, consisting of a START followed by a STOP condition, is illegal. Note that the Wired-ANDing of the SCL line can be used to implement handshaking between the master and the slave. The slave can extend the SCL low period by pulling the SCL line low. This is useful if the clock speed set up by the master is too fast for the slave, or the slave needs extra time for processing between the data transmissions. The slave extending the SCL low period will not affect the SCL high period, which is determined by the master. As a consequence, the slave can reduce the TWI data transfer speed by prolonging the SCL duty cycle.

Figure 15-6 shows a typical data transmission. Note that several data bytes can be transmitted between the SLA+R/W and the STOP condition, depending on the software protocol implemented by the application software.

Figure 15-6. Typical Data Transmission

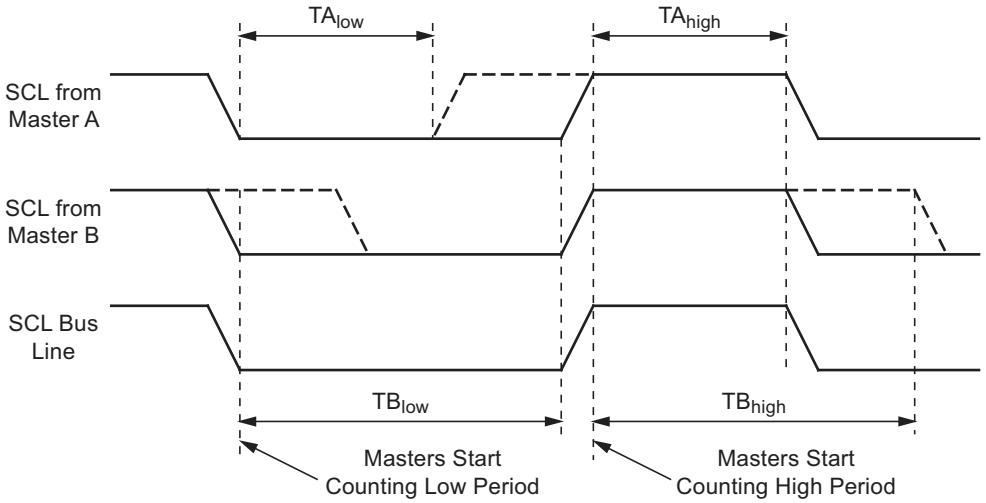


15.4 Multi-master Bus Systems, Arbitration and Synchronization

The TWI protocol allows bus systems with several masters. Special concerns have been taken in order to ensure that transmissions will proceed as normal, even if two or more masters initiate a transmission at the same time. Two problems arise in multi-master systems:

- An algorithm must be implemented allowing only one of the masters to complete the transmission. All other masters should cease transmission when they discover that they have lost the selection process. This selection process is called arbitration. When a contending master discovers that it has lost the arbitration process, it should immediately switch to slave mode to check whether it is being addressed by the winning master. The fact that multiple masters have started transmission at the same time should not be detectable to the slaves, i.e. the data being transferred on the bus must not be corrupted.
- Different masters may use different SCL frequencies. A scheme must be devised to synchronize the serial clocks from all masters, in order to let the transmission proceed in a lockstep fashion. This will facilitate the arbitration process.

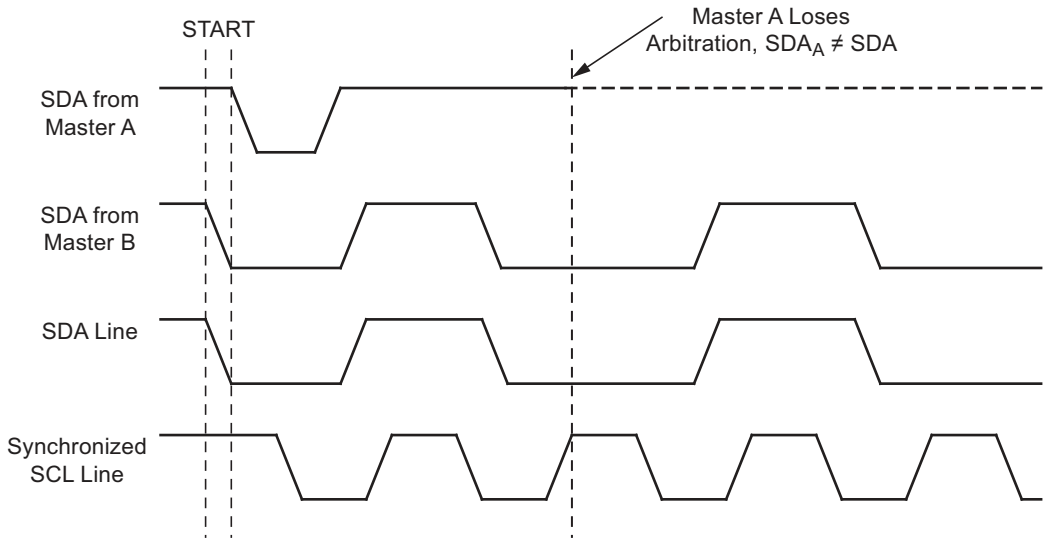
Figure 15-7. SCL Synchronization Between Multiple Masters



The wired-ANDing of the bus lines is used to solve both these problems. The serial clocks from all masters will be wired-ANDed, yielding a combined clock with a high period equal to the one from the master with the shortest high period. The low period of the combined clock is equal to the low period of the master with the longest low period. Note that all masters listen to the SCL line, effectively starting to count their SCL high and low time-out periods when the combined SCL line goes high or low, respectively.

Arbitration is carried out by all masters continuously monitoring the SDA line after outputting data. If the value read from the SDA line does not match the value the master had output, it has lost the arbitration. Note that a master can only lose arbitration when it outputs a high SDA value while another master outputs a low value. The losing master should immediately go to slave mode, checking if it is being addressed by the winning master. The SDA line should be left high, but losing masters are allowed to generate a clock signal until the end of the current data or address packet. Arbitration will continue until only one master remains, and this may take many bits. If several masters are trying to address the same slave, arbitration will continue into the data packet.

Figure 15-8. Arbitration Between Two Masters



Note that arbitration is not allowed between:

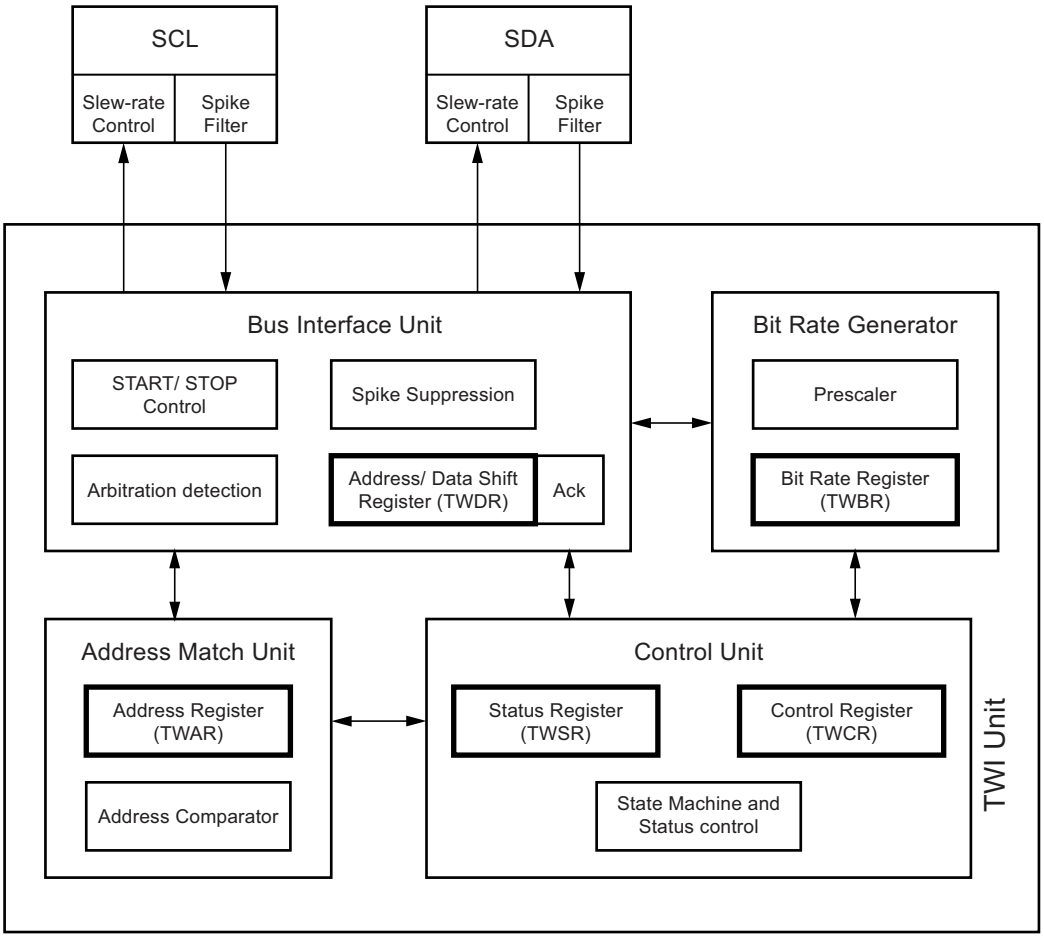
- A REPEATED START condition and a data bit.
- A STOP condition and a data bit.
- A REPEATED START and a STOP condition.

It is the user software's responsibility to ensure that these illegal arbitration conditions never occur. This implies that in multi-master systems, all data transfers must use the same composition of SLA+R/W and data packets. In other words: All transmissions must contain the same number of data packets, otherwise the result of the arbitration is undefined.

15.5 Overview of the TWI Module

The TWI module is comprised of several submodules, as shown in Figure 15-9. All registers drawn in a thick line are accessible through the AVR® data bus.

Figure 15-9. Overview of the TWI Module



15.5.1 SCL and SDA Pins

These pins interface the AVR TWI with the rest of the MCU system. The output drivers contain a slew-rate limiter in order to conform to the TWI specification. The input stages contain a spike suppression unit removing spikes shorter than 50ns. Note that the internal pull-ups in the AVR pads can be enabled by setting the PORT bits corresponding to the SCL and SDA pins, as explained in the I/O port section. The internal pull-ups can in some systems eliminate the need for external ones.

15.5.2 Bit Rate Generator Unit

When operating in a master mode this unit controls the period of SCL. The SCL period is controlled by settings in the TWI Bit rate register (TWBR) and the prescaler bits in the TWI status register (TWSR). Slave operation does not depend on bit rate or prescaler settings, but the clock frequency in the slave must be at least 16 times higher than the SCL frequency. Note that slaves may prolong the SCL low period, thereby reducing the average TWI bus clock period.

The TWI can be set to operate in high-speed mode, as described in Section 15.9.7 “TWHSR – TWI High Speed Register” on page 141. In high-speed mode the TWI uses the system clock, whereas in normal mode it relies on a prescaled version of the same. Depending on the clock signal used, the SCL frequency is generated according to one of the following equations.

In normal mode:

$$f_{SCL} = \frac{clk_{I/O}}{16 + (2 \times TWBR \times TWPS)}$$

In high-speed mode:

$$f_{SCL} = \frac{clk_{TWIHS}}{16 + (2 \times TWBR \times TWPS)}$$

...where:

- $clk_{I/O}$ = prescaled system clock, see Figure 6-1 on page 25
- clk_{TWIHS} = system clock, see Figure 6-1 on page 25
- TWBR = value of TWI Bit Rate Register, see Section 15.9.1 “TWBR – TWI Bit Rate Register” on page 137
- TWPS = value of TWI prescaler, see Table 15-8 on page 139

Note: In TWI master mode TWBR must be 10, or higher .

15.5.3 Bus Interface Unit

This unit contains the data and address shift register (TWDR), a START/STOP controller and arbitration detection hardware. The TWDR contains the address or data bytes to be transmitted, or the address or data bytes received. In addition to the 8-bit TWDR, the bus interface unit also contains a register containing the (N)ACK bit to be transmitted or received. This (N)ACK register is not directly accessible by the application software. However, when receiving, it can be set or cleared by manipulating the TWI control register (TWCR). When in transmitter mode, the value of the received (N)ACK bit can be determined by the value in the TWSR.

The START/STOP controller is responsible for generation and detection of START, REPEATED START, and STOP conditions. The START/STOP controller is able to detect START and STOP conditions even when the AVR® MCU is in one of the sleep modes, enabling the MCU to wake up if addressed by a master.

If the TWI has initiated a transmission as master, the arbitration detection hardware continuously monitors the transmission trying to determine if arbitration is in process. If the TWI has lost an arbitration, the control unit is informed. correct action can then be taken and appropriate status codes generated.

15.5.4 Address Match Unit

The address match unit checks if received address bytes match the seven-bit address in the TWI address register (TWAR). If the TWI general call recognition enable (TWGCE) bit in the TWAR is written to one, all incoming address bits will also be compared against the general call address. Upon an address match, the control unit is informed, allowing correct action to be taken. The TWI may or may not acknowledge its address, depending on settings in the TWCR. The address match unit is able to compare addresses even when the AVR MCU is in sleep mode, enabling the MCU to wake up if addressed by a master. If another interrupt (e.g., INT0) occurs during TWI power-down address match and wakes up the CPU, the TWI aborts operation and return to it’s idle state. If this cause any problems, ensure that TWI address match is the only enabled interrupt when entering power-down.

15.5.5 Control Unit

The control unit monitors the TWI bus and generates responses corresponding to settings in the TWI control register (TWCR). When an event requiring the attention of the application occurs on the TWI bus, the TWI interrupt flag (TWINT) is asserted. In the next clock cycle, the TWI status register (TWSR) is updated with a status code identifying the event. The TWSR only contains relevant status information when the TWI interrupt flag is asserted. At all other times, the TWSR contains a special status code indicating that no relevant status information is available. As long as the TWINT flag is set, the SCL line is held low. This allows the application software to complete its tasks before allowing the TWI transmission to continue.

The TWINT flag is set in the following situations:

- After the TWI has transmitted a START/REPEATED START condition.
- After the TWI has transmitted SLA+R/W.
- After the TWI has transmitted an address byte.
- After the TWI has lost arbitration.
- After the TWI has been addressed by own slave address or general call.
- After the TWI has received a data byte.
- After a STOP or REPEATED START has been received while still addressed as a slave.
- When a bus error has occurred due to an illegal START or STOP condition.

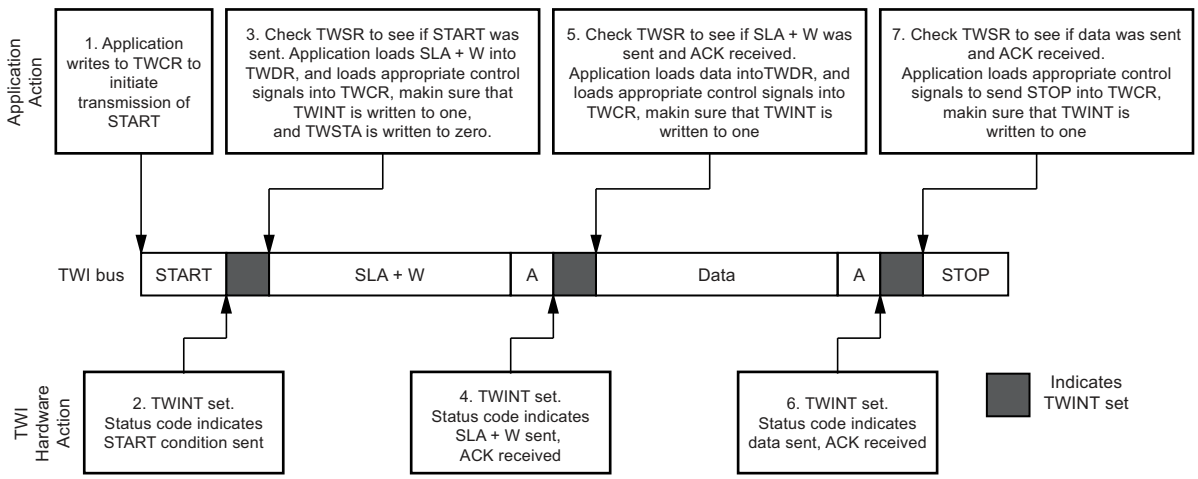
15.6 Using the TWI

The AVR® TWI is byte-oriented and interrupt based. Interrupts are issued after all bus events, like reception of a byte or transmission of a START condition. Because the TWI is interrupt-based, the application software is free to carry on other operations during a TWI byte transfer. Note that the TWI interrupt enable (TWIE) bit in TWCR together with the global interrupt enable bit in SREG allow the application to decide whether or not assertion of the TWINT flag should generate an interrupt request. If the TWIE bit is cleared, the application must poll the TWINT flag in order to detect actions on the TWI bus.

When the TWINT flag is asserted, the TWI has finished an operation and awaits application response. In this case, the TWI status register (TWSR) contains a value indicating the current state of the TWI bus. The application software can then decide how the TWI should behave in the next TWI bus cycle by manipulating the TWCR and TWDR registers.

Figure 15-10 is a simple example of how the application can interface to the TWI hardware. In this example, a master wishes to transmit a single data byte to a Slave. This description is quite abstract, a more detailed explanation follows later in this section. A simple code example implementing the desired behavior is also presented.

Figure 15-10.Interfacing the Application to the TWI in a Typical Transmission



1. The first step in a TWI transmission is to transmit a START condition. This is done by writing a specific value into TWCR, instructing the TWI hardware to transmit a START condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the START condition.
2. When the START condition has been transmitted, the TWINT flag in TWCR is set, and TWSR is updated with a status code indicating that the START condition has successfully been sent.
3. The application software should now examine the value of TWSR, to make sure that the START condition was successfully transmitted. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load SLA+W into TWDR. Remember that TWDR is used both for address and data. After TWDR has been loaded with the desired SLA+W, a specific value must be written to TWCR, instructing the TWI hardware to transmit the SLA+W present in TWDR. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the address packet.
4. When the address packet has been transmitted, the TWINT flag in TWCR is set, and TWSR is updated with a status code indicating that the address packet has successfully been sent. The status code will also reflect whether a slave acknowledged the packet or not.
5. The application software should now examine the value of TWSR, to make sure that the address packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load a data packet into TWDR. Subsequently, a specific value must be written to TWCR, instructing the TWI hardware to transmit the data packet present in TWDR. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the data packet.
6. When the data packet has been transmitted, the TWINT flag in TWCR is set, and TWSR is updated with a status code indicating that the data packet has successfully been sent. The status code will also reflect whether a slave acknowledged the packet or not.
7. The application software should now examine the value of TWSR, to make sure that the data packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must write a specific value to TWCR, instructing the TWI hardware to transmit a STOP condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the STOP condition. Note that TWINT is NOT set after a STOP condition has been sent.

Even though this example is simple, it shows the principles involved in all TWI transmissions. These can be summarized as follows:

- When the TWI has finished an operation and expects application response, the TWINT flag is set. The SCL line is pulled low until TWINT is cleared.
- When the TWINT flag is set, the user must update all TWI registers with the value relevant for the next TWI bus cycle. As an example, TWDR must be loaded with the value to be transmitted in the next bus cycle.
- After all TWI register updates and other pending application software tasks have been completed, TWCR is written. When writing TWCR, the TWINT bit should be set. Writing a one to TWINT clears the flag. The TWI will then commence executing whatever operation was specified by the TWCR setting.

In the following an assembly and C implementation of the example is given. Note that the code below assumes that several definitions have been made, for example by using include-files.

Table 15-2. Assembly Code Example

	Assembly Code Example	C Example	Comments
1	`ldi r16, (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)` `out TWCR, r16`	`TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)`	Send START condition

2	`wait1:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait1`
		`while (!(TWCR & (1<<TWINT)))`
		`;`
			Wait for TWINT flag set. This indicates that the START condition has been transmitted

3	`in r16,TWSR`
	`andi r16, 0xF8`
	`cpi r16, START`
	`brne ERROR`
		`if ((TWSR & 0xF8) != START)`
		`ERROR();`
			Check value of TWI status register. Mask prescaler bits. If status different from START go to ERROR

	`ldi r16, SLA_W`
	`out TWDR, r16`
	`ldi r16, (1<<TWINT)|(1<<TWEN)`
	`out TWCR, r16`
		`TWDR = SLA_W;`
		`TWCR = (1<<TWINT)|(1<<TWEN);`
			Load SLA_W into TWDR register. Clear TWINT bit in TWCR to start transmission of address

4	`wait2:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait2`
		`while (!(TWCR & (1<<TWINT)))`
		`;`
			Wait for TWINT flag set. This indicates that the SLA+W has been transmitted, and ACK/NACK has been received.

5	`in r16,TWSR`
	`andi r16, 0xF8`
	`cpi r16, MT_SLA_ACK`
	`brne ERROR`
		`if ((TWSR & 0xF8) != MT_SLA_ACK)`
		`ERROR();`
			Check value of TWI status register. Mask prescaler bits. If status different from MT_SLA_ACK go to ERROR

	`ldi r16, DATA`
	`out TWDR, r16`
	`ldi r16, (1<<TWINT)|(1<<TWEN)`
	`out TWCR, r16`
		`TWDR = DATA;`
		`TWCR = (1<<TWINT)|(1<<TWEN);`
			Load DATA into TWDR register. clear TWINT bit in TWCR to start transmission of data

6	`wait3:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait3`
		`while (!(TWCR & (1<<TWINT)))`
		`;`
			Wait for TWINT flag set. This indicates that the DATA has been transmitted, and ACK/NACK has been received.

7	`in r16,TWSR`
	`andi r16, 0xF8`
	`cpi r16, MT_DATA_ACK`
	`brne ERROR`
		`if ((TWSR & 0xF8) != MT_DATA_ACK)`
		`ERROR();`
			Check value of TWI status register. Mask prescaler bits. If status different from MT_DATA_ACK go to ERROR

	`ldi r16, (1<<TWINT)|(1<<TWEN)|1<<TWSTO)` `out TWCR, r16`	`TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);`	Transmit STOP condition

15.7 Transmission Modes

The TWI can operate in one of four major modes. These are named master transmitter (MT), master receiver (MR), slave transmitter (ST) and slave receiver (SR). Several of these modes can be used in the same application. As an example, the TWI can use MT mode to write data into a TWI EEPROM, MR mode to read the data back from the EEPROM. If other masters are present in the system, some of these might transmit data to the TWI, and then SR mode would be used. It is the application software that decides which modes are legal.

The following sections describe each of these modes. Possible status codes are described along with figures detailing data transmission in each of the modes. These figures contain the following abbreviations:

S: START condition

Rs: REPEATED START condition

R: Read bit (high level at SDA)

W: Write bit (low level at SDA)

A: Acknowledge bit (low level at SDA)

$\overline{A}$: Not acknowledge bit (high level at SDA)

Data: 8-bit data byte

P: STOP condition

SLA: Slave address

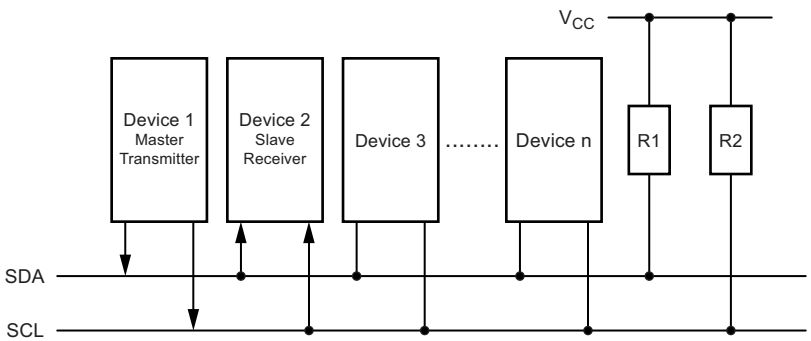
In Figure 15-12 on page 125 to Figure 15-18 on page 135, circles are used to indicate that the TWINT flag is set. The numbers in the circles show the status code held in TWSR, with the prescaler bits masked to zero. At these points, actions must be taken by the application to continue or complete the TWI transfer. The TWI transfer is suspended until the TWINT flag is cleared by software.

When the TWINT flag is set, the status code in TWSR is used to determine the appropriate software action. For each status code, the required software action and details of the following serial transfer are given in Table 15-3 on page 124 to Table 15-6 on page 134. Note that the prescaler bits are masked to zero in these tables.

15.7.1 Master Transmitter Mode

In the master transmitter mode, a number of data bytes are transmitted to a slave receiver (see Figure 15-11). In order to enter a master mode, a START condition must be transmitted. The format of the following address packet determines whether master transmitter or master receiver mode is to be entered. If SLA+W is transmitted, MT mode is entered, if SLA+R is transmitted, MR mode is entered. All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 15-11.Data Transfer in Master Transmitter Mode



A START condition is sent by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	1	0	X	1	0	X

TWEN must be set to enable the 2-wire serial interface, TWSTA must be written to one to transmit a START condition and TWINT must be written to one to clear the TWINT flag. The TWI will then test the 2-wire serial bus and generate a START condition as soon as the bus becomes free. After a START condition has been transmitted, the TWINT flag is set by hardware, and the status code in TWSR will be 0x08 (see Table 15-3 on page 124). In order to enter MT mode, SLA+W must be transmitted. This is done by writing SLA+W to TWDR. Thereafter the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	0	0	X	1	0	X

When SLA+W have been transmitted and an acknowledgement bit has been received, TWINT is set again and a number of status codes in TWSR are possible. Possible status codes in master mode are 0x18, 0x20, or 0x38. The appropriate action to be taken for each of these status codes is detailed in Table 15-3 on page 124.

When SLA+W has been successfully transmitted, a data packet should be transmitted. This is done by writing the data byte to TWDR. TWDR must only be written when TWINT is high. If not, the access will be discarded, and the write collision bit (TWWC) will be set in the TWCR register. After updating TWDR, the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	0	0	X	1	0	X

This scheme is repeated until the last byte has been sent and the transfer is ended by generating a STOP condition or a repeated START condition. A STOP condition is generated by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	0	1	X	1	0	X

A REPEATED START condition is generated by writing the following value to TWCR:

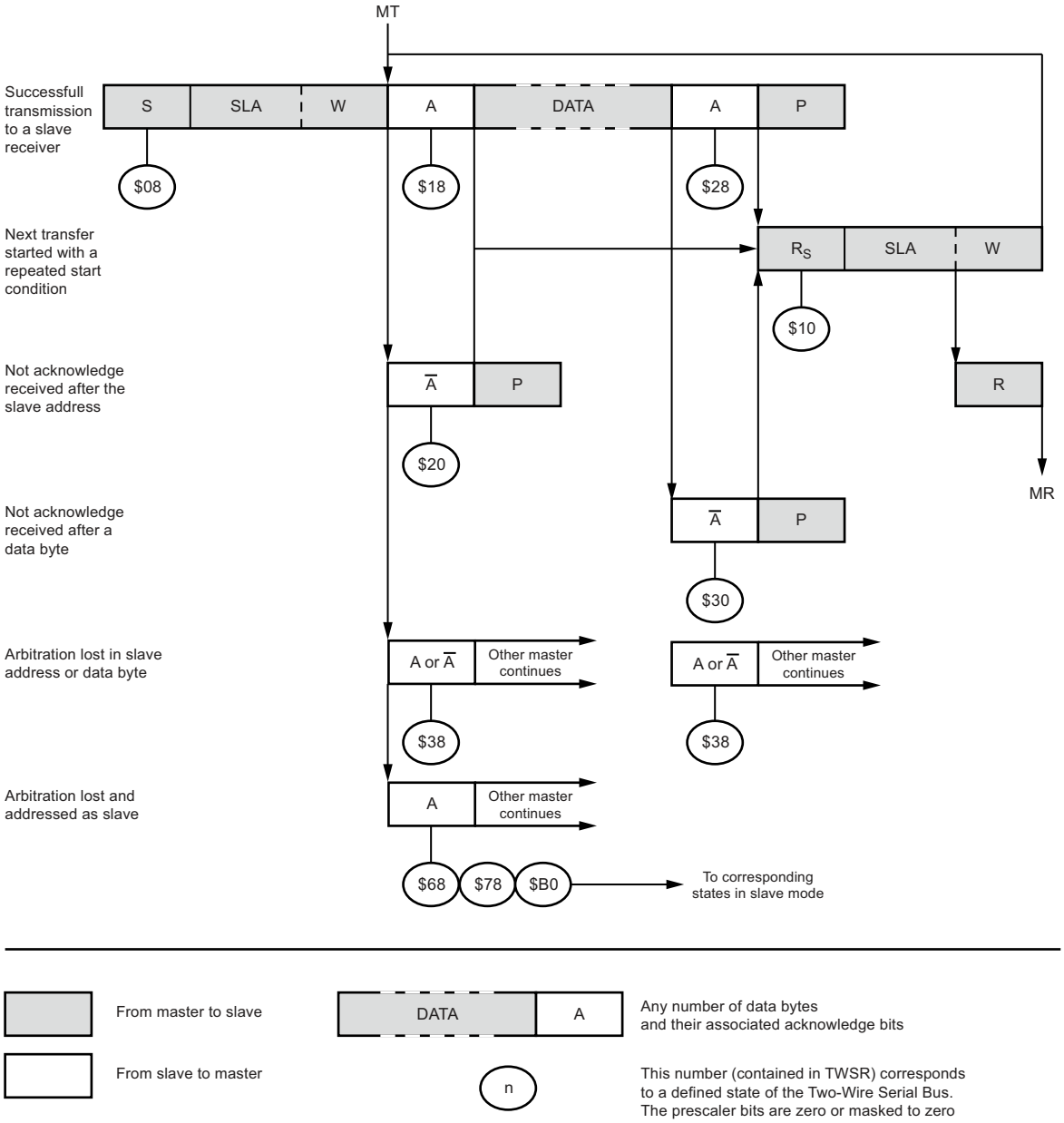
TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	1	0	X	1	0	X

After a repeated START condition (state 0x10) the 2-wire serial interface can access the same slave again, or a new slave without transmitting a STOP condition. Repeated START enables the master to switch between slaves, master transmitter mode and master receiver mode without losing control of the bus.

Table 15-3. Status codes for Master Transmitter Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x08	A START condition has been transmitted	Load SLA+W	0	0	1	X	SLA+W will be transmitted; ACK or NOT ACK will be received
0x10	A repeated START condition has been transmitted	Load SLA+W or Load SLA+R	0 0	0 0	1 1	X X	SLA+W will be transmitted; ACK or NOT ACK will be received SLA+R will be transmitted; Logic will switch to Master Receiver mode
0x18	SLA+W has been transmitted; ACK has been received	Load data byte or No TWDR action or No TWDR action or No TWDR action	0 1 0 1	0 0 1 1	1 1 1 1	X X X X	Data byte will be transmitted and ACK or NOT ACK will be received Repeated START will be transmitted STOP condition will be transmitted and TWSTO Flag will be reset STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
0x20	SLA+W has been transmitted; NOT ACK has been received	Load data byte or No TWDR action or No TWDR action or No TWDR action	0 1 0 1	0 0 1 1	1 1 1 1	X X X X	Data byte will be transmitted and ACK or NOT ACK will be received Repeated START will be transmitted STOP condition will be transmitted and TWSTO Flag will be reset STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
0x28	Data byte has been transmitted; ACK has been received	Load data byte or No TWDR action or No TWDR action or No TWDR action	0 1 0 1	0 0 1 1	1 1 1 1	X X X X	Data byte will be transmitted and ACK or NOT ACK will be received Repeated START will be transmitted STOP condition will be transmitted and TWSTO Flag will be reset STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
0x30	Data byte has been transmitted; NOT ACK has been received	Load data byte or No TWDR action or No TWDR action or No TWDR action	0 1 0 1	0 0 1 1	1 1 1 1	X X X X	Data byte will be transmitted and ACK or NOT ACK will be received Repeated START will be transmitted STOP condition will be transmitted and TWSTO Flag will be reset STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
0x38	Arbitration lost in SLA+W or data bytes	No TWDR action or No TWDR action	0 1	0 0	1 1	X X	2-wire Serial Bus will be released and not addressed Slave mode entered A START condition will be transmitted when the bus becomes free

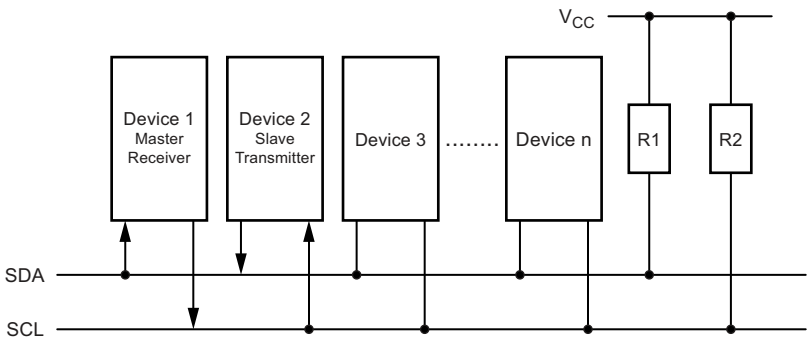
Figure 15-12.Formats and States in the Master Transmitter Mode



15.7.2 Master Receiver Mode

In the master receiver mode, a number of data bytes are received from a slave transmitter (slave see Figure 15-13 on page 126). In order to enter a master mode, a START condition must be transmitted. The format of the following address packet determines whether master transmitter or master receiver mode is to be entered. If SLA+W is transmitted, MT mode is entered, if SLA+R is transmitted, MR mode is entered. All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 15-13.Data Transfer in Master Receiver Mode



A START condition is sent by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	1	0	X	1	0	X

TWEN must be written to one to enable the 2-wire serial interface, TWSTA must be written to one to transmit a START condition and TWINT must be set to clear the TWINT flag. The TWI will then test the 2-wire serial bus and generate a START condition as soon as the bus becomes free. After a START condition has been transmitted, the TWINT flag is set by hardware, and the status code in TWSR will be 0x08 (See Table 15-3 on page 124). In order to enter MR mode, SLA+R must be transmitted. This is done by writing SLA+R to TWDR. Thereafter the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	0	0	X	1	0	X

When SLA+R have been transmitted and an acknowledgement bit has been received, TWINT is set again and a number of status codes in TWSR are possible. Possible status codes in master mode are 0x38, 0x40, or 0x48. The appropriate action to be taken for each of these status codes is detailed in Table 15-4 on page 127. Received data can be read from the TWDR register when the TWINT flag is set high by hardware. This scheme is repeated until the last byte has been received. After the last byte has been received, the MR should inform the ST by sending a NACK after the last received data byte. The transfer is ended by generating a STOP condition or a repeated START condition. A STOP condition is generated by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	0	1	X	1	0	X

A REPEATED START condition is generated by writing the following value to TWCR:

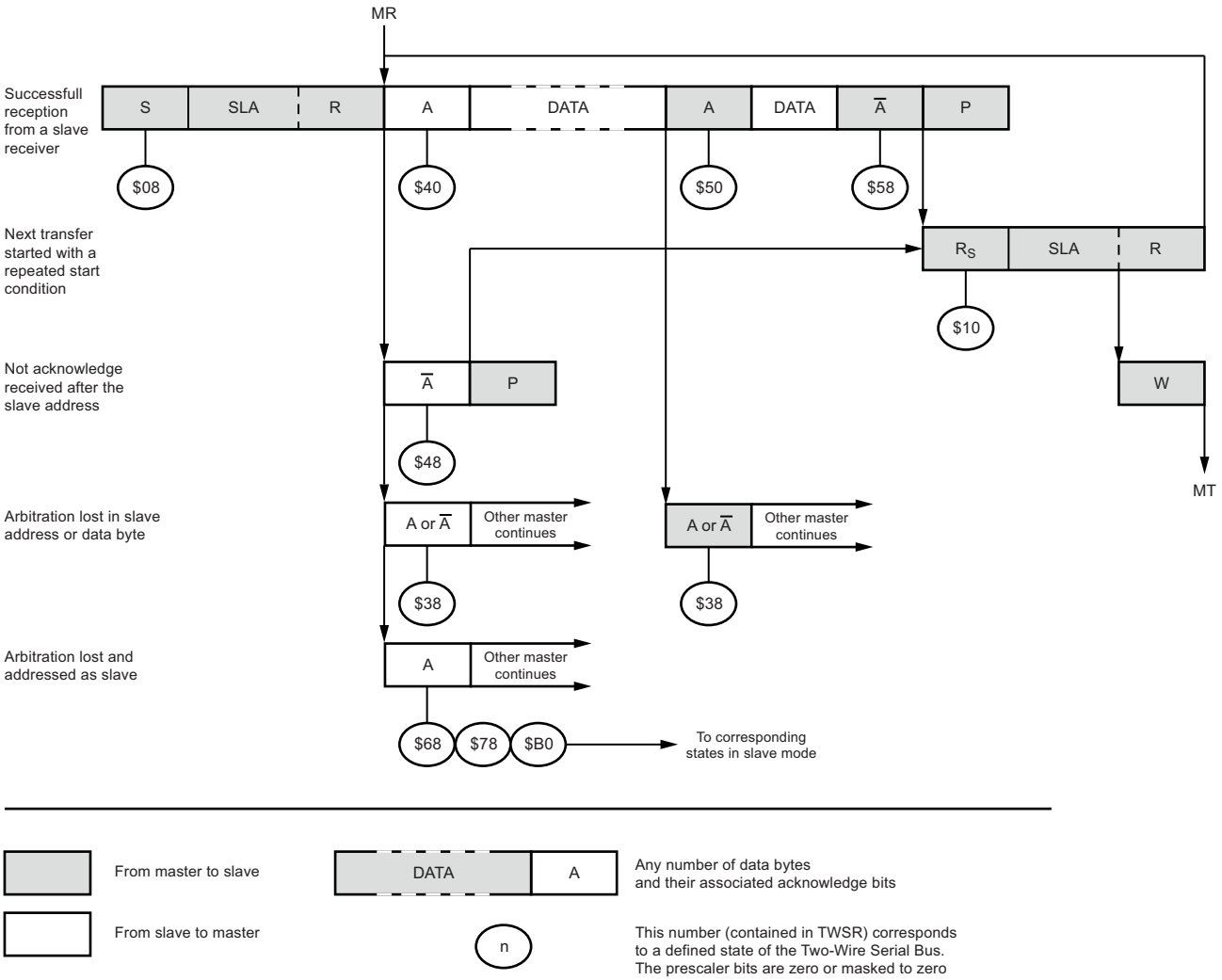
TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	1	0	X	1	0	X

After a repeated START condition (state 0x10) the 2-wire serial interface can access the same slave again, or a new slave without transmitting a STOP condition. Repeated START enables the master to switch between slaves, master transmitter mode and master receiver mode without losing control over the bus.

Table 15-4. Status codes for Master Receiver Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x08	A START condition has been transmitted	Load SLA+R	0	0	1	X	SLA+R will be transmitted ACK or NOT ACK will be received
0x10	A repeated START condition has been transmitted	Load SLA+R or Load SLA+W	0 0	0 0	1 1	X X	SLA+R will be transmitted ACK or NOT ACK will be received SLA+W will be transmitted Logic will switch to Master Transmitter mode
0x38	Arbitration lost in SLA+R or NOT ACK bit	No TWDR action or No TWDR action	0 1	0 0	1 1	X X	2-wire Serial Bus will be released and not addressed Slave mode will be entered A START condition will be transmitted when the bus becomes free
0x40	SLA+R has been transmitted; ACK has been received	No TWDR action or No TWDR action	0 0	0 0	1 1	0 1	Data byte will be received and NOT ACK will be returned Data byte will be received and ACK will be returned
0x48	SLA+R has been transmitted; NOT ACK has been received	No TWDR action or No TWDR action or No TWDR action	1 0 1	0 1 1	1 1 1	X X X	Repeated START will be transmitted STOP condition will be transmitted and TWSTO Flag will be reset STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset
0x50	Data byte has been received; ACK has been returned	Read data byte or Read data byte	0 0	0 0	1 1	0 1	Data byte will be received and NOT ACK will be returned Data byte will be received and ACK will be returned
0x58	Data byte has been received; NOT ACK has been returned	Read data byte or Read data byte or Read data byte	1 0 1	0 1 1	1 1 1	X X X	Repeated START will be transmitted STOP condition will be transmitted and TWSTO Flag will be reset STOP condition followed by a START condition will be transmitted and TWSTO Flag will be reset

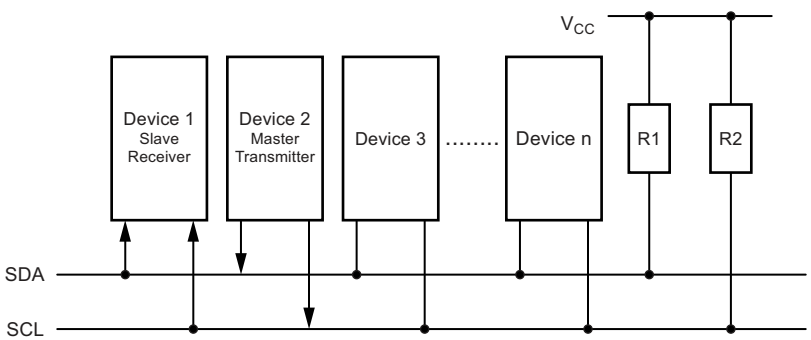
Figure 15-14.Formats and States in the Master Receiver Mode



15.7.3 Slave Receiver Mode

In the slave receiver mode, a number of data bytes are received from a master transmitter (see Figure 15-15). All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 15-15.Data transfer in Slave Receiver mode



To initiate the slave receiver mode, TWAR and TWCR must be initialized as follows:

TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
value	Device's Own Slave Address							

The upper 7 bits are the address to which the 2-wire serial interface will respond when addressed by a master. If the LSB is set, the TWI will respond to the general call address (0x00), otherwise it will ignore the general call address.

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	0	1	0	0	0	1	0	X

TWEN must be written to one to enable the TWI. The TWEA bit must be written to one to enable the acknowledgement of the device's own slave address or the general call address. TWSTA and TWSTO must be written to zero.

When TWAR and TWCR have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address if enabled) followed by the data direction bit. If the direction bit is "0" (write), the TWI will operate in SR mode, otherwise ST mode is entered. After its own slave address and the write bit have been received, the TWINT flag is set and a valid status code can be read from TWSR. The status code is used to determine the appropriate software action. The appropriate action to be taken for each status code is detailed in Table 15-5 on page 130. The slave receiver mode may also be entered if arbitration is lost while the TWI is in the master mode (see states 0x68 and 0x78).

If the TWEA bit is reset during a transfer, the TWI will return a "not acknowledge" ("1") to SDA after the next received data byte. This can be used to indicate that the slave is not able to receive any more bytes. While TWEA is zero, the TWI does not acknowledge its own slave address. However, the 2-wire serial bus is still monitored and address recognition may resume at any time by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the 2-wire serial bus.

In all sleep modes other than Idle mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the 2-wire serial bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock low during the wake up and until the TWINT flag is cleared (by writing it to one). Further data reception will be carried out as normal, with the AVR® clocks running as normal. Observe that if the AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions.

Note that the 2-wire serial interface data register – TWDR does not reflect the last byte present on the bus when waking up from these Sleep modes.

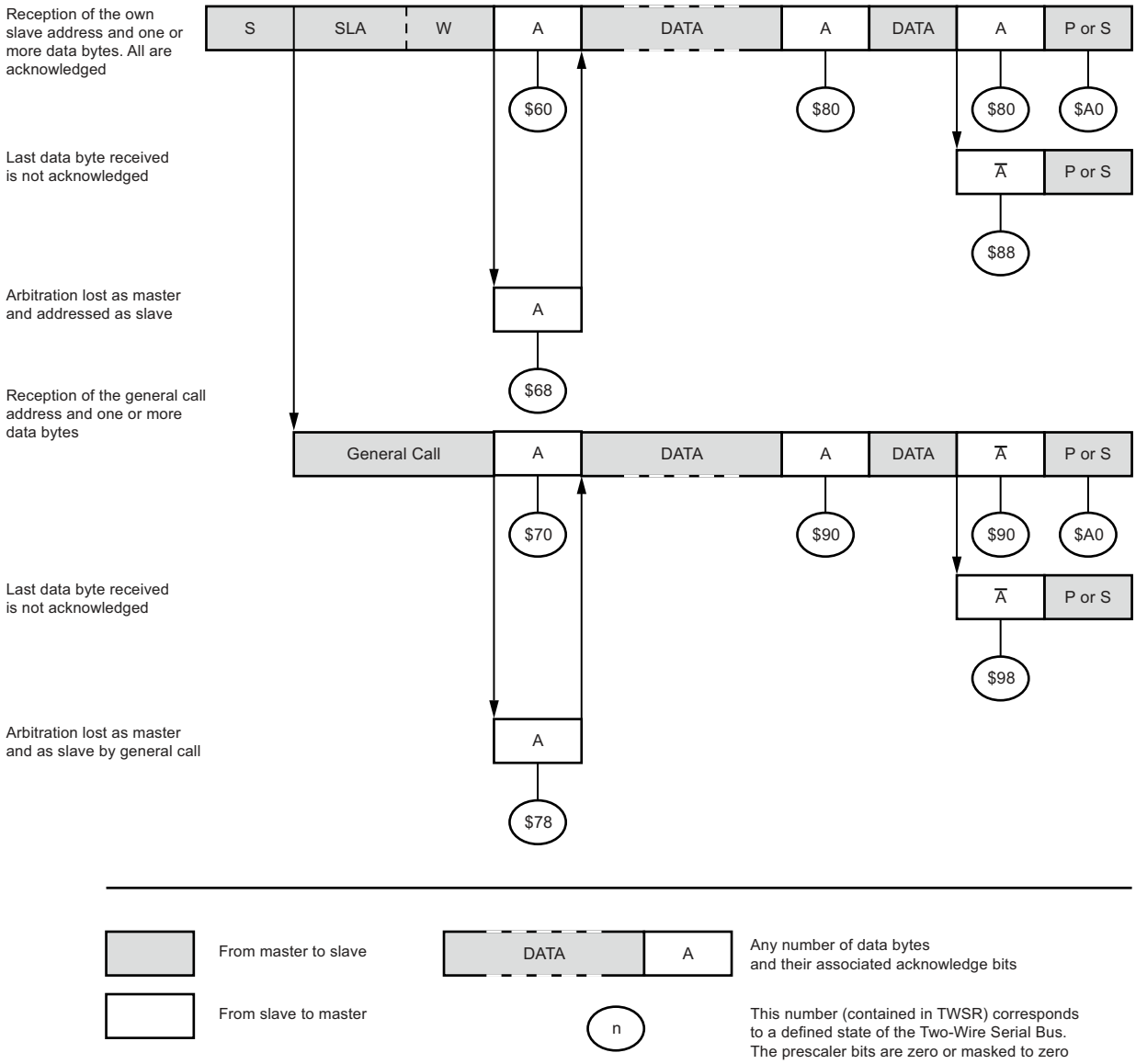
Table 15-5. Status Codes for Slave Receiver Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x60	Own SLA+W has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x68	Arbitration lost in SLA+R/W as Master; own SLA+W has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x70	General call address has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x78	Arbitration lost in SLA+R/W as Master; General call address has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x80	Previously addressed with own SLA+W; data has been received; ACK has been returned	Read data byte or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		Read data byte	X	0	1	1	Data byte will be received and ACK will be returned
0x88	Previously addressed with own SLA+W; data has been received; NOT ACK has been returned	Read data byte or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
			0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
		Read data byte or	1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA;
						1	a START condition will be transmitted when the bus becomes free
		Read data byte or	1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free
		Read data byte					

Table 15-5. Status Codes for Slave Receiver Mode (Continued)

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x90	Previously addressed with general call; data has been received; ACK has been returned	Read data byte or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		Read data byte	X	0	1	1	Data byte will be received and ACK will be returned
0x98	Previously addressed with general call; data has been received; NOT ACK has been returned	Read data byte or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		Read data byte or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
		Read data byte or	1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		Read data byte	1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free
0xA0	A STOP condition or repeated START condition has been received while still addressed as Slave	No action	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
			0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
			1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
			1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free

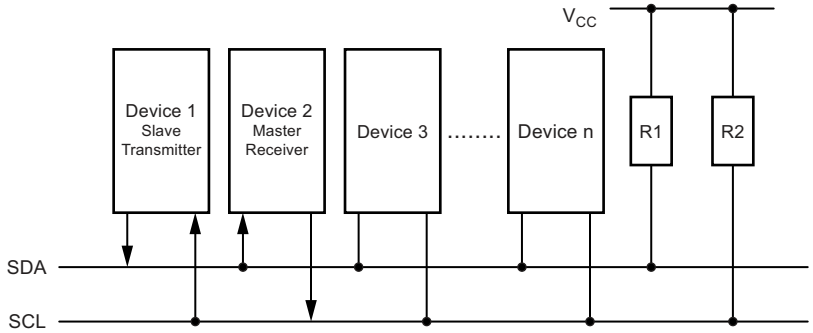
Figure 15-16.Formats and States in the Slave Receiver Mode



15.7.4 Slave Transmitter Mode

In the slave transmitter mode, a number of data bytes are transmitted to a master receiver (see Figure 15-17). All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 15-17.Data Transfer in Slave Transmitter Mode



To initiate the slave transmitter mode, TWAR and TWCR must be initialized as follows:

TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
value	Device's Own Slave Address							

The upper seven bits are the address to which the 2-wire serial interface will respond when addressed by a master. If the LSB is set, the TWI will respond to the general call address (0x00), otherwise it will ignore the general call address.

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	0	1	0	0	0	1	0	X

TWEN must be written to one to enable the TWI. The TWEA bit must be written to one to enable the acknowledgement of the device's own slave address or the general call address. TWSTA and TWSTO must be written to zero.

When TWAR and TWCR have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address if enabled) followed by the data direction bit. If the direction bit is "1" (read), the TWI will operate in ST mode, otherwise SR mode is entered. After its own slave address and the write bit have been received, the TWINT flag is set and a valid status code can be read from TWSR. The status code is used to determine the appropriate software action. The appropriate action to be taken for each status code is detailed in Table 15-6 on page 134. The slave transmitter mode may also be entered if arbitration is lost while the TWI is in the master mode (see state 0xB0).

If the TWEA bit is written to zero during a transfer, the TWI will transmit the last byte of the transfer. State 0xC0 or state 0xC8 will be entered, depending on whether the master receiver transmits a NACK or ACK after the final byte. The TWI is switched to the not addressed Slave mode, and will ignore the master if it continues the transfer. Thus the master receiver receives all "1" as serial data. State 0xC8 is entered if the master demands additional data bytes (by transmitting ACK), even though the slave has transmitted the last byte (TWEA zero and expecting NACK from the master).

While TWEA is zero, the TWI does not respond to its own slave address. However, the 2-wire serial bus is still monitored and address recognition may resume at any time by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the 2-wire serial bus.

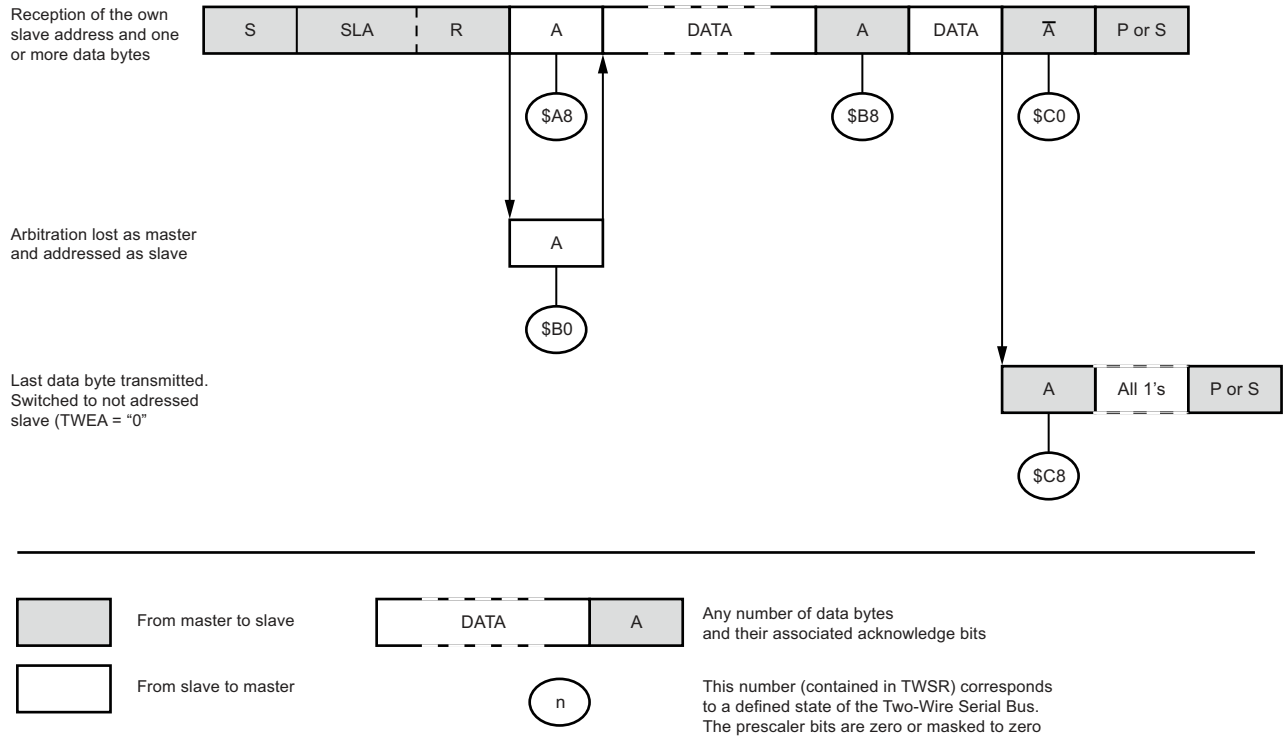
In all sleep modes other than Idle mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the 2-wire serial bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock will low during the wake up and until the TWINT flag is cleared (by writing it to one). Further data transmission will be carried out as normal, with the AVR® clocks running as normal. Observe that if the AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions.

Note that the 2-wire serial interface data register – TWDR does not reflect the last byte present on the bus when waking up from these sleep modes.

Table 15-6. Status Codes for Slave Transmitter Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0xA8	Own SLA+R has been received; ACK has been returned	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received
		Load data byte	X	0	1	1	Data byte will be transmitted and ACK should be received
0xB0	Arbitration lost in SLA+R/W as Master; own SLA+R has been received; ACK has been returned	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received
		Load data byte	X	0	1	1	Data byte will be transmitted and ACK should be received
0xB8	Data byte in TWDR has been transmitted; ACK has been received	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received
		Load data byte	X	0	1	1	Data byte will be transmitted and ACK should be received
0xC0	Data byte in TWDR has been transmitted; NOT ACK has been received	No TWDR action or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		No TWDR action or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
		No TWDR action or	1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		No TWDR action	1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free
0xC8	Last data byte in TWDR has been transmitted (TWEA = “0”); ACK has been received	No TWDR action or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		No TWDR action or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
		No TWDR action or	1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		No TWDR action	1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free

Figure 15-18.Formats and States in the Slave Transmitter Mode



15.7.5 Miscellaneous States

There are two status codes that do not correspond to a defined TWI state, see Table 15-7.

Status 0xF8 indicates that no relevant information is available because the TWINT flag is not set. This occurs between other states, and when the TWI is not involved in a serial transfer.

Status 0x00 indicates that a bus error has occurred during a 2-wire serial bus transfer. A bus error occurs when a START or STOP condition occurs at an illegal position in the format frame. Examples of such illegal positions are during the serial transfer of an address byte, a data byte, or an acknowledge bit. When a bus error occurs, TWINT is set. To recover from a bus error, the TWSTO flag must set and TWINT must be cleared by writing a logic one to it. This causes the TWI to enter the not addressed Slave mode and to clear the TWSTO flag (no other bits in TWCR are affected). The SDA and SCL lines are released, and no STOP condition is transmitted.

Table 15-7. Miscellaneous States

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0xF8	No relevant state information available; TWINT = “0”	No TWDR action	No TWCR action				Wait or proceed current transfer
0x00	Bus error due to an illegal START or STOP condition	No TWDR action	0	1	1	X	Only the internal hardware is affected, no STOP condition is sent on the bus. In all cases, the bus is released and TWSTO is cleared.

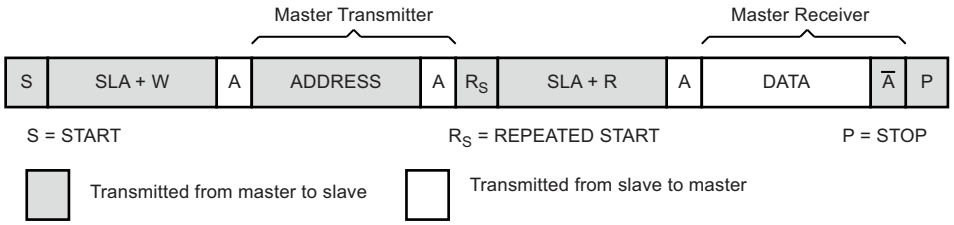
15.7.6 Combining Several TWI Modes

In some cases, several TWI modes must be combined in order to complete the desired action. Consider for example reading data from a serial EEPROM. Typically, such a transfer involves the following steps:

1. The transfer must be initiated.
2. The EEPROM must be instructed what location should be read.
3. The reading must be performed.
4. The transfer must be finished.

Note that data is transmitted both from master to slave and vice versa. The master must instruct the slave what location it wants to read, requiring the use of the MT mode. Subsequently, data must be read from the slave, implying the use of the MR mode. Thus, the transfer direction must be changed. The master must keep control of the bus during all these steps, and the steps should be carried out as an atomical operation. If this principle is violated in a multi master system, another master can alter the data pointer in the EEPROM between steps 2 and 3, and the master will read the wrong data location. Such a change in transfer direction is accomplished by transmitting a REPEATED START between the transmission of the address byte and reception of the data. After a REPEATED START, the master keeps ownership of the bus. The following figure shows the flow in this transfer.

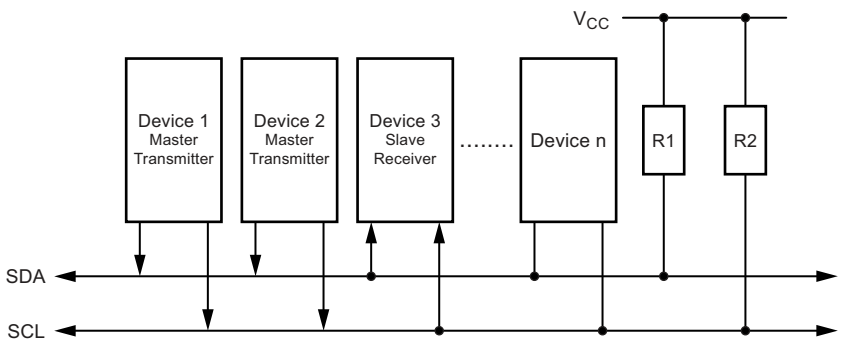
Figure 15-19.Combining Several TWI Modes to Access a Serial EEPROM



15.8 Multi-master Systems and Arbitration

If multiple masters are connected to the same bus, transmissions may be initiated simultaneously by one or more of them. The TWI standard ensures that such situations are handled in such a way that one of the masters will be allowed to proceed with the transfer, and that no data will be lost in the process. An example of an arbitration situation is depicted below, where two masters are trying to transmit data to a slave receiver.

Figure 15-20.An Arbitration Example

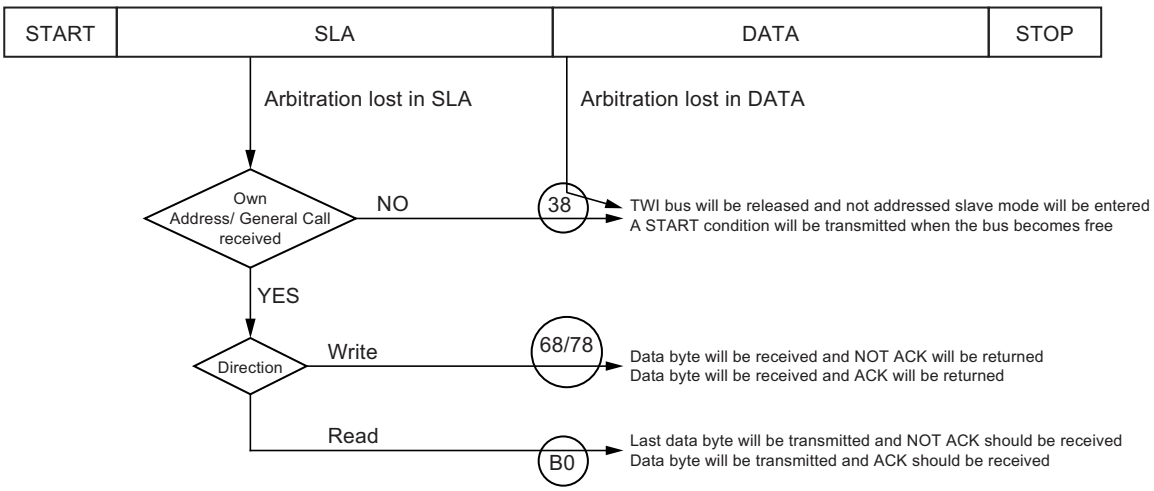


Several different scenarios may arise during arbitration, as described below:

- Two or more masters are performing identical communication with the same slave. In this case, neither the slave nor any of the masters will know about the bus contention.
- Two or more masters are accessing the same slave with different data or direction bit. In this case, arbitration will occur, either in the READ/WRITE bit or in the data bits. The masters trying to output a one on SDA while another master outputs a zero will lose the arbitration. Losing masters will switch to not addressed Slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.
- Two or more masters are accessing different slaves. In this case, arbitration will occur in the SLA bits. Masters trying to output a one on SDA while another master outputs a zero will lose the arbitration. Masters losing arbitration in SLA will switch to slave mode to check if they are being addressed by the winning master. If addressed, they will switch to SR or ST mode, depending on the value of the READ/WRITE bit. If they are not being addressed, they will switch to not addressed slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.

This is summarized in Figure 15-21. Possible status values are given in circles.

Figure 15-21.Possible Status Codes Caused by Arbitration



15.9 Register Description

15.9.1 TWBR – TWI Bit Rate Register

Bit	7	6	5	4	3	2	1	0	
	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..0 – TWI Bit Rate Register**

TWBR selects the division factor for the bit rate generator. The bit rate generator is a frequency divider which generates the SCL clock frequency in the Master modes. See Section 15.5.2 “Bit Rate Generator Unit” on page 117 for calculating bit rates.

If the TWI operates in master mode TWBR must be set to 10, or higher.

15.9.2 TWCR – TWI Control Register

Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The TWCR is used to control the operation of the TWI. It is used to enable the TWI, to initiate a master access by applying a START condition to the bus, to generate a receiver acknowledge, to generate a stop condition, and to control halting of the bus while the data to be written to the bus are written to the TWDR. It also indicates a write collision if data is attempted written to TWDR while the register is inaccessible.

- **Bit 7 – TWINT: TWI Interrupt Flag**

This bit is set by hardware when the TWI has finished its current job and expects application software response. If the I-bit in SREG and TWIE in TWCR are set, the MCU will jump to the TWI interrupt vector. While the TWINT flag is set, the SCL low period is stretched. The TWINT flag must be cleared by software by writing a logic one to it. Note that this flag is not automatically cleared by hardware when executing the interrupt routine. Also note that clearing this flag starts the operation of the TWI, so all accesses to the TWI address register (TWAR), TWI status register (TWSR), and TWI data register (TWDR) must be complete before clearing this flag.

- **Bit 6 – TWEA: TWI Enable Acknowledge Bit**

The TWEA bit controls the generation of the acknowledge pulse. If the TWEA bit is written to one, the ACK pulse is generated on the TWI bus if the following conditions are met:

1. The device's own slave address has been received.
2. A general call has been received, while the TWGCE bit in the TWAR is set.
3. A data byte has been received in master receiver or slave receiver mode.

By writing the TWEA bit to zero, the device can be virtually disconnected from the 2-wire serial bus temporarily. Address recognition can then be resumed by writing the TWEA bit to one again.

- **Bit 5 – TWSTA: TWI START Condition Bit**

The application writes the TWSTA bit to one when it desires to become a master on the 2-wire serial bus. The TWI hardware checks if the bus is available, and generates a START condition on the bus if it is free. However, if the bus is not free, the TWI waits until a STOP condition is detected, and then generates a new START condition to claim the bus master status. TWSTA must be cleared by software when the START condition has been transmitted.

- **Bit 4 – TWSTO: TWI STOP Condition Bit**

Writing the TWSTO bit to one in master mode will generate a STOP condition on the 2-wire serial bus. When the STOP condition is executed on the bus, the TWSTO bit is cleared automatically. In slave mode, setting the TWSTO bit can be used to recover from an error condition. This will not generate a STOP condition, but the TWI returns to a well-defined unaddressed slave mode and releases the SCL and SDA lines to a high impedance state.

- **Bit 3 – TWWC: TWI Write Collision Flag**

The TWWC bit is set when attempting to write to the TWI data register – TWDR when TWINT is low. This flag is cleared by writing the TWDR register when TWINT is high.

- **Bit 2 – TWEN: TWI Enable Bit**

The TWEN bit enables TWI operation and activates the TWI interface. When TWEN is written to one, the TWI takes control over the I/O pins connected to the SCL and SDA pins, enabling the slew-rate limiters and spike filters. If this bit is written to zero, the TWI is switched off and all TWI transmissions are terminated, regardless of any ongoing operation.

- **Bit 1 – Res: Reserved Bit**

This bit is a reserved bit and will always read as zero.

- **Bit 0 – TWIE: TWI Interrupt Enable**

When this bit is written to one, and the I-bit in SREG is set, the TWI interrupt request will be activated for as long as the TWINT flag is high.

15.9.3 TWSR – TWI Status Register

Bit	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

- **Bits 7..3 – TWS: TWI Status**

These 5 bits reflect the status of the TWI logic and the 2-wire serial bus. The different status codes are described later in this section. Note that the value read from TWSR contains both the 5-bit status value and the 2-bit prescaler value. The application designer should mask the prescaler bits to zero when checking the status bits. This makes status checking independent of prescaler setting. This approach is used in this datasheet, unless otherwise noted.

- **Bit 2 – Res: Reserved Bit**

This bit is reserved and will always read as zero.

- **Bits 1..0 – TWPS: TWI Prescaler Bits**

These bits can be read and written, and control the bit rate prescaler.

Table 15-8. TWI Bit Rate Prescaler

TWPS1	TWPS0	Prescaler Value
0	0	1
0	1	4
1	0	16
1	1	64

To calculate bit rates, see Section 15.5.2 “Bit Rate Generator Unit” on page 117. The value of TWPS1..0 is used in the equation.

15.9.4 TWDR – TWI Data Register

Bit	7	6	5	4	3	2	1	0	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

In transmit mode, TWDR contains the next byte to be transmitted. In receive mode, the TWDR contains the last byte received. It is writable while the TWI is not in the process of shifting a byte. This occurs when the TWI interrupt flag (TWINT) is set by hardware. Note that the data register cannot be initialized by the user before the first interrupt occurs. The data in TWDR remains stable as long as TWINT is set. While data is shifted out, data on the bus is simultaneously shifted in. TWDR always contains the last byte present on the bus, except after a wake up from a sleep mode by the TWI interrupt. In this case, the contents of TWDR is undefined. In the case of a lost bus arbitration, no data is lost in the transition from master to slave. Handling of the ACK bit is controlled automatically by the TWI logic, the CPU cannot access the ACK bit directly.

- **Bits 7..0 – TWD: TWI Data Register**

These eight bits constitute the next data byte to be transmitted, or the latest data byte received on the 2-wire serial bus.

15.9.5 TWAR – TWI (Slave) Address Register

Bit	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

The TWAR should be loaded with the 7-bit slave address (in the seven most significant bits of TWAR) to which the TWI will respond when programmed as a slave transmitter or receiver, and not needed in the master modes. In multi master systems, TWAR must be set in masters which can be addressed as slaves by other masters.

The LSB of TWAR is used to enable recognition of the general call address (0x00). There is an associated address comparator that looks for the slave address (or general call address if enabled) in the received serial address. If a match is found, an interrupt request is generated.

- **Bits 7..1 – TWA: TWI (Slave) Address Register**

These seven bits constitute the slave address of the TWI unit.

- **Bit 0 – TWGCE: TWI General Call Recognition Enable Bit**

If set, this bit enables the recognition of a general call given over the 2-wire serial bus.

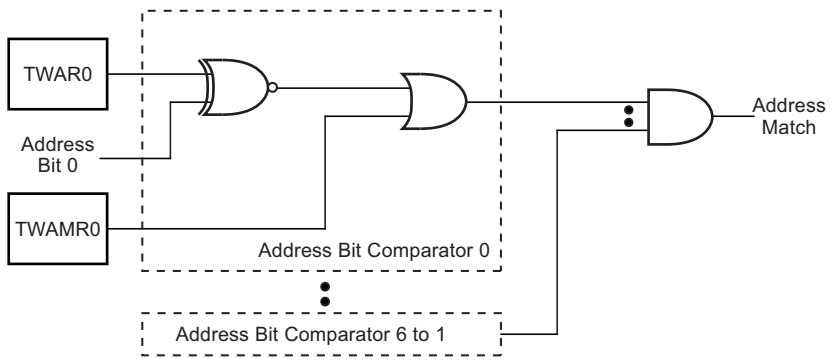
15.9.6 TWAMR – TWI (Slave) Address Mask Register–

Bit	7	6	5	4	3	2	1	0	
	TWAM[6:0]							–	TWAMR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..1 – TWAM: TWI Address Mask**

The TWAMR can be loaded with a 7-bit salve address mask. Each of the bits in TWAMR can mask (disable) the corresponding address bits in the TWI address register (TWAR). If the mask bit is set to one then the address match logic ignores the compare between the incoming address bit and the corresponding bit in TWAR. Figure 15-22 shown the address match logic in detail.

Figure 15-22.TWI Address Match Logic, Block Diagram



- **Bit 0 – Res: Reserved Bit**

These bits are reserved and will always read zero.

15.9.7 TWHSR – TWI High Speed Register

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	–	TWHS	TWHSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..1 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 0 – TWHS: TWI High Speed Enable**

TWI high speed mode is enabled by writing this bit to one. In this mode the undivided system clock is selected as TWI clock. See Figure 6-1 on page 25.

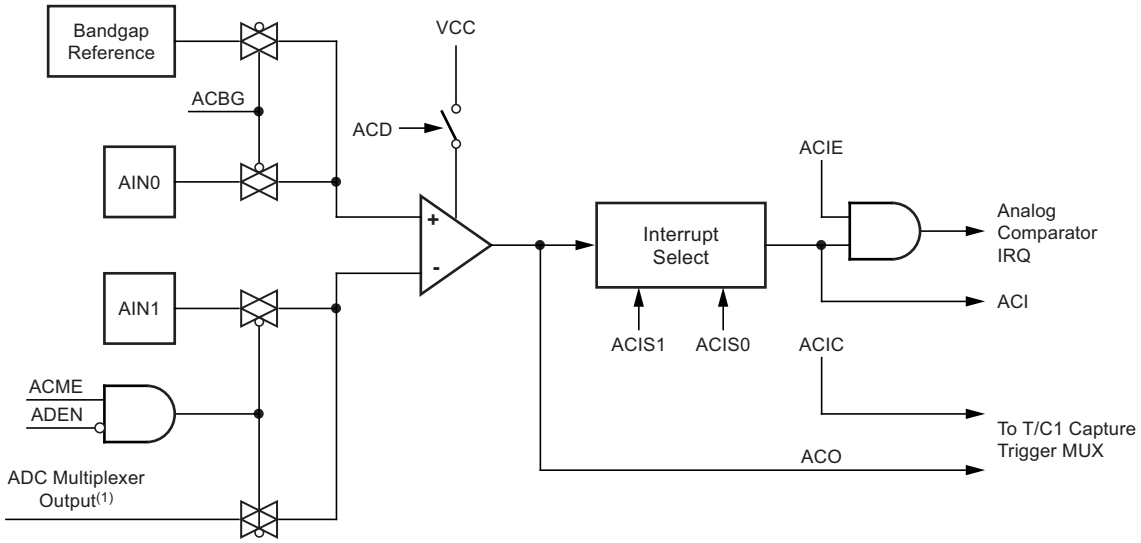
The TWI high speed mode requires that the high-speed clock, clk_{TWIHS}, is exactly two times higher than the I/O clock frequency, $clk_{I/O}$. This means the user must make sure the I/O clock frequency $clk_{I/O}$ is scaled down by a factor of 2. For example, if the internal 8MHz oscillator has been selected as source clock, the user must set the prescaler to scale the system clock (and, hence, the I/O clock) down to 4MHz. For more information about clock systems, see Section 6.1 “Clock Systems and their Distribution” on page 25.

16. Analog Comparator

The analog comparator compares the input values on the positive pin AIN0 and negative pin AIN1. When the voltage on the positive pin AIN0 is higher than the voltage on the negative pin AIN1, the analog comparator output, ACO, is set. The comparator's output can be set to trigger the Timer/Counter1 input capture function. In addition, the comparator can trigger a separate interrupt, exclusive to the analog comparator. The user can select interrupt triggering on comparator output rise, fall or toggle. A block diagram of the comparator and its surrounding logic is shown in Figure 16-1.

The ADC power reduction bit, PRADC, must be disabled in order to use the ADC input multiplexer. This is done by clearing the PRADC bit in the power reduction register, PRR. See Section 7.4.3 "PRR – Power Reduction Register" on page 36 for more details.

Figure 16-1. Analog Comparator Block Diagram(2)



Notes: 1. See Table 16-1.

2. Refer to Figure 1-1 on page 3 and Table 10-11 on page 63 for analog comparator pin placement.

16.1 Analog Comparator Multiplexed Input

It is possible to select any of the ADC7..0 pins to replace the negative input to the analog comparator. The ADC multiplexer is used to select this input, and consequently, the ADC must be switched off to utilize this feature. If the analog comparator multiplexer enable bit (ACME in ADCSRB) is set and the ADC is switched off (ADEN in ADCSRA is zero), MUX2..0 in ADMUX select the input pin to replace the negative input to the analog comparator, as shown in Table 16-1. If ACME is cleared or ADEN is set, AIN1 is applied to the negative input to the analog comparator.

Table 16-1. Analog Comparator Multiplexed Input

ACME	ADEN	MUX2..0	Analog Comparator Negative Input
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6
1	0	111	ADC7

16.2 Register Description

16.2.1 ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 6 – ACME: Analog Comparator Multiplexer Enable**

When this bit is written logic one and the ADC is switched off (ADEN in ADCSRA is zero), the ADC multiplexer selects the negative input to the analog comparator. When this bit is written logic zero, AIN1 is applied to the negative input of the analog comparator. For a detailed description of this bit, see
Section 16.1 “Analog Comparator Multiplexed Input” on page 142.

16.2.2 ACSR – Analog Comparator Control and Status Register

Bit	7	6	5	4	3	2	1	0	
	ACD	**ACBG**	**ACO**	**ACI**	**ACIE**	**ACIC**	**ACIS1**	**ACIS0**	**ACSR**
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

- **Bit 7 – ACD: Analog Comparator Disable**

When this bit is written logic one, the power to the analog comparator is switched off. This bit can be set at any time to turn off the analog comparator. This will reduce power consumption in active and idle mode. When changing the ACD bit, the analog comparator interrupt must be disabled by clearing the ACIE bit in ACSR. Otherwise an interrupt can occur when the bit is changed.

- **Bit 6 – ACBG: Analog Comparator Bandgap Select**

When this bit is set, a fixed internal bandgap reference voltage replaces the positive input to the analog comparator. When this bit is cleared, AIN0 is applied to the positive input of the analog comparator. See
Section 8.3 “Internal Voltage Reference” on page 40

- **Bit 5 – ACO: Analog Comparator Output**

The output of the analog comparator is synchronized and then directly connected to ACO. The synchronization introduces a delay of 1 – 2 clock cycles.

- **Bit 4 – ACI: Analog Comparator Interrupt Flag**

This bit is set by hardware when a comparator output event triggers the interrupt mode defined by ACIS1 and ACIS0. The analog comparator interrupt routine is executed if the ACIE bit is set and the I-bit in SREG is set. ACI is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ACI is cleared by writing a logic one to the flag.

- **Bit 3 – ACIE: Analog Comparator Interrupt Enable**

When the ACIE bit is written logic one and the I-bit in the status register is set, the analog comparator interrupt is activated. When written logic zero, the interrupt is disabled.

- **Bit 2 – ACIC: Analog Comparator Input Capture Enable**

When written logic one, this bit enables the input capture function in Timer/Counter1 to be triggered by the analog comparator. The comparator output is in this case directly connected to the input capture front-end logic, making the comparator utilize the noise canceler and edge select features of the Timer/Counter1 input capture interrupt. When written logic zero, no connection between the analog comparator and the input capture function exists. To make the comparator trigger the Timer/Counter1 input capture interrupt, the ICIE1 bit in the timer interrupt mask register (TIMSK1) must be set.

- **Bits 1, 0 – ACIS1, ACIS0: Analog Comparator Interrupt Mode Select**

These bits determine which comparator events that trigger the analog comparator interrupt. The different settings are shown in Table 16-2.

Table 16-2. ACIS1/ACIS0 Settings

ACIS1	ACIS0	Interrupt Mode
0	0	Comparator interrupt on output toggle.
0	1	Reserved
1	0	Comparator interrupt on falling output edge.
1	1	Comparator interrupt on rising output edge.

When changing the ACIS1/ACIS0 bits, the analog comparator interrupt must be disabled by clearing its interrupt enable bit in the ACSR register. Otherwise an interrupt can occur when the bits are changed.

16.2.3 DIDR1 – Digital Input Disable Register 1

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	AIN1D	AIN0D	DIDR1
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7..2 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 1, 0 – AIN1D, AIN0D: AIN1, AIN0 Digital Input Disable**

When this bit is written logic one, the digital input buffer on the AIN1/0 pin is disabled. The corresponding PIN register bit will always read as zero when this bit is set. When an analog signal is applied to the AIN1/0 pin and the digital input from this pin is not needed, this bit should be written logic one to reduce power consumption in the digital input buffer.

17. Analog-to-Digital Converter

17.1 Features

- 10-bit resolution
- 1 LSB integral non-linearity
- ±2 LSB absolute accuracy
- 65µs conversion time
- 15kSPS at maximum resolution
- Six multiplexed single ended input channels
- Two additional multiplexed single ended input channels (TQFP and QFN packages, only)
- Temperature sensor input channel
- Optional left adjustment for ADC result readout
- 0 – V_{CC} ADC input voltage range
- Selectable 1.1V ADC reference voltage
- Free running or single conversion mode
- Interrupt on ADC conversion complete
- Sleep mode noise canceler

17.2 Overview

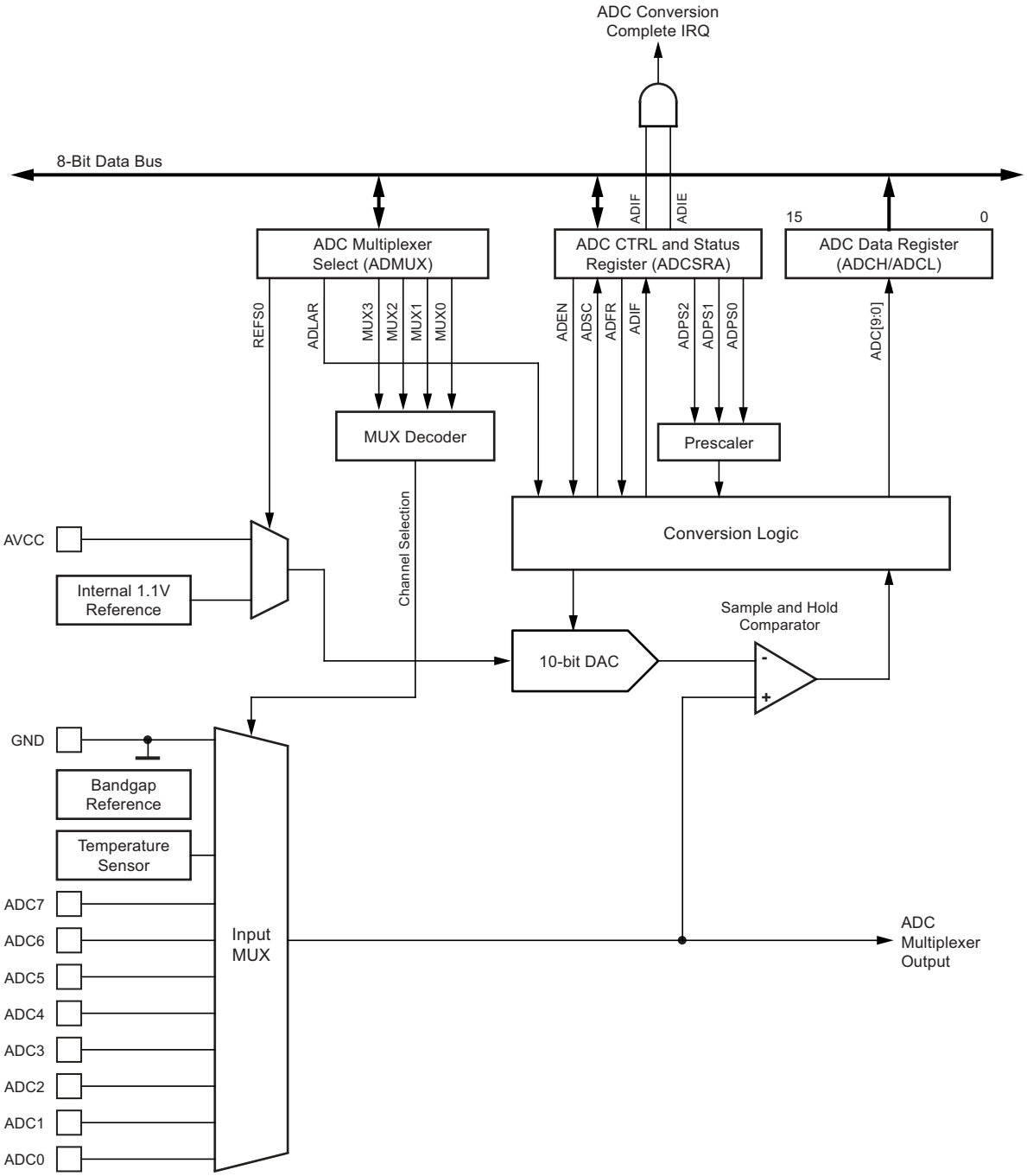
Atmel® ATtiny88 features a 10-bit, successive approximation analog-to-digital converter (ADC). The ADC is wired to a nine-channel analog multiplexer, which allows the ADC to measure the voltage at six (or eight, in 32-lead packages) single-ended input pins and from one internal, single-ended voltage channel coming from the internal temperature sensor. Single-ended voltage inputs are referred to 0V (GND).

The ADC contains a sample and hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion. A block diagram of the ADC is shown in Figure 17-1 on page 146.

There is a separate analog supply voltage pin for the ADC, AV_{CC}. Analog supply voltage is connected to the ADC via a passive switch. The voltage difference between supply voltage pins V_{CC} and AV_{CC} may not exceed. See section Section 17.7 “ADC Noise Canceler” on page 151 on how to connect the analog suplly voltage pin.

Internal reference voltage of nominally 1.1V is provided on-chip. Alternatively, V_{CC} can be used as reference voltage.

Figure 17-1. Analog to Digital Converter Block Schematic Operation



17.3 Operation

In order to be able to use the ADC the power reduction bit, PRADC, in the power reduction register must be disabled. This is done by clearing the PRADC bit. See Section 7.4.3 “PRR – Power Reduction Register” on page 36 for more details.

The ADC converts an analog input voltage to a 10-bit digital value through successive approximation. The minimum value represents GND and the maximum value represents the reference voltage. The ADC voltage reference may be selected by writing the REFS0 bit in the ADMUX register. Alternatives are the V_{CC} supply pin and the internal 1.1V voltage reference.

The analog input channel is selected by writing to the MUX bits in ADMUX. Any of the ADC input pins, as well as GND and a fixed bandgap voltage reference, can be selected as single ended inputs to the ADC.

The ADC is enabled by setting the ADC enable bit, ADEN in ADCSRA. Voltage reference and input channel selections will not go into effect until ADEN is set. The ADC does not consume power when ADEN is cleared, so it is recommended to switch off the ADC before entering power saving sleep modes.

The ADC generates a 10-bit result which is presented in the ADC data registers, ADCH and ADCL. By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the ADLAR bit in ADMUX.

If the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH, only. Otherwise, ADCL must be read first, then ADCH, to ensure that the content of the data registers belongs to the same conversion. Once ADCL is read, ADC access to data registers is blocked. This means that if ADCL has been read, and a conversion completes before ADCH is read, neither register is updated and the result from the conversion is lost. When ADCH is read, ADC access to the ADCH and ADCL registers is re-enabled.

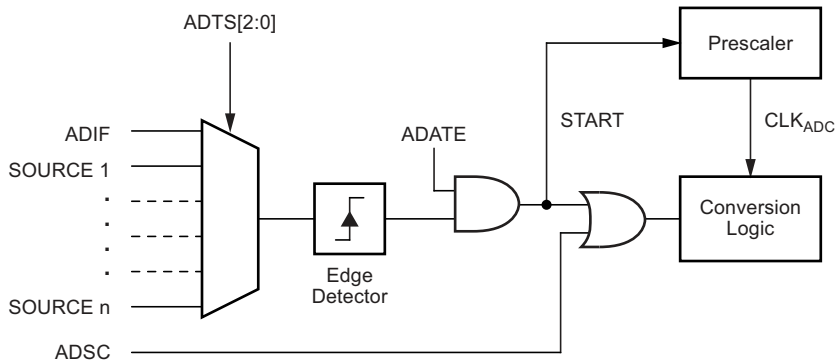
The ADC has its own interrupt which can be triggered when a conversion completes. When ADC access to the data registers is prohibited between reading of ADCH and ADCL, the interrupt will trigger even if the result is lost.

17.4 Starting a Conversion

Make sure the ADC is powered by clearing the ADC power reduction bit, PRADC, in the power reduction register, PRR (see Section 7.4.3 “PRR – Power Reduction Register” on page 36). A single conversion is started by writing a logical one to the ADC start conversion bit, ADSC. This bit stays high as long as the conversion is in progress and will be cleared by hardware when the conversion is completed. If a different data channel is selected while a conversion is in progress, the ADC will finish the current conversion before performing the channel change.

Alternatively, a conversion can be triggered automatically by various sources. Auto triggering is enabled by setting the ADC auto trigger enable bit, ADATE in ADCSRA. The trigger source is selected by setting the ADC trigger select bits, ADTS in ADCSRB (See description of the ADTS bits for a list of the trigger sources). When a positive edge occurs on the selected trigger signal, the ADC prescaler is reset and a conversion is started. This provides a method of starting conversions at fixed intervals. If the trigger signal still is set when the conversion completes, a new conversion will not be started. If another positive edge occurs on the trigger signal during conversion, the edge will be ignored. Note that an interrupt flag will be set even if the specific interrupt is disabled or the global interrupt enable bit in SREG is cleared. A conversion can thus be triggered without causing an interrupt. However, the interrupt flag must be cleared in order to trigger a new conversion at the next interrupt event.

Figure 17-2. ADC Auto Trigger Logic



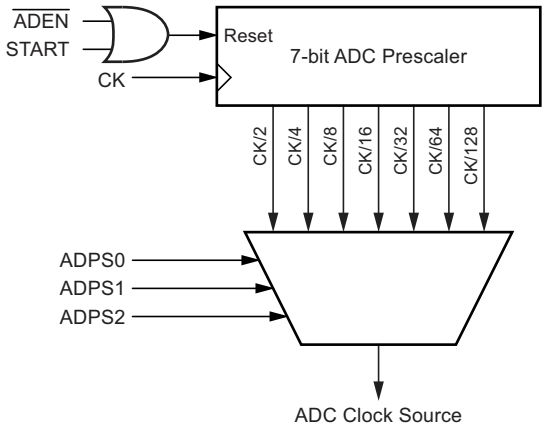
Using the ADC interrupt flag as a trigger source makes the ADC start a new conversion as soon as the ongoing conversion has finished. The ADC then operates in free running mode, constantly sampling and updating the ADC data register. The first conversion must be started by writing a logical one to the ADSC bit in ADCSRA. In this mode the ADC will perform successive conversions independently of whether the ADC interrupt flag, ADIF is cleared or not.

If Auto triggering is enabled, single conversions can be started by writing ADSC in ADCSRA to one. ADSC can also be used to determine if a conversion is in progress. The ADSC bit will be read as one during a conversion, independently of how the conversion was started.

17.5 Prescaling and Conversion Timing

The successive approximation circuitry requires an input clock frequency between 50kHz and 200kHz to get maximum resolution.

Figure 17-3. ADC Prescaler

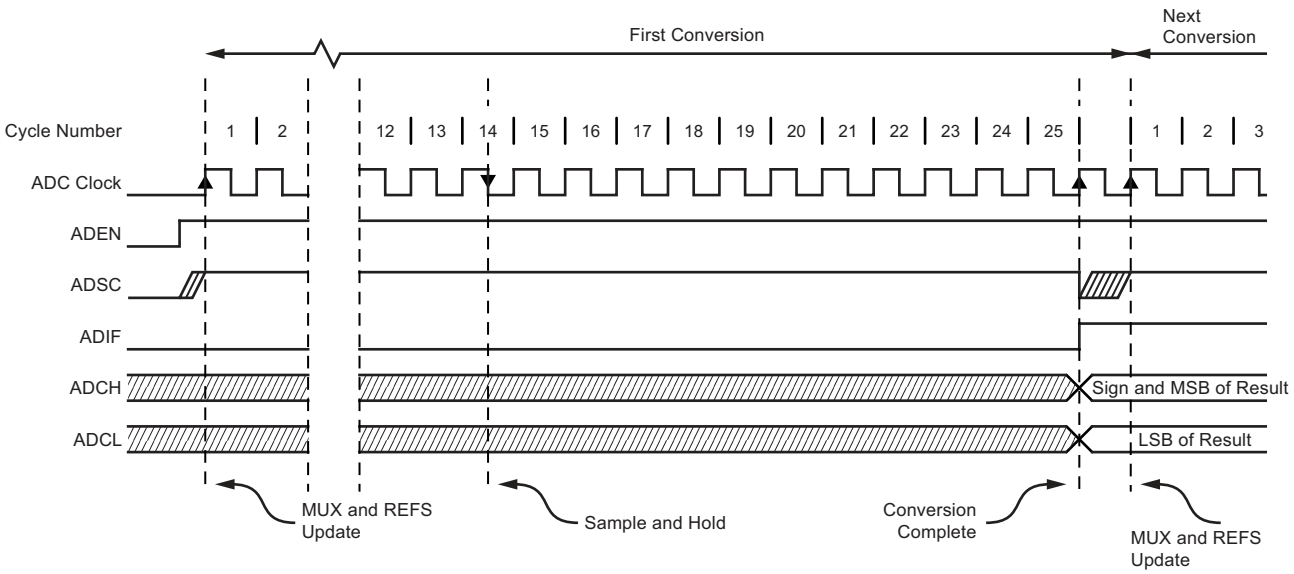


The ADC module contains a prescaler, as illustrated in Figure 17-3, which generates an acceptable ADC clock frequency from any CPU frequency above 100kHz. The prescaling is set by the ADPS bits in ADCSRA. The prescaler starts counting from the moment the ADC is switched on by setting the ADEN bit in ADCSRA. The prescaler keeps running for as long as the ADEN bit is set, and is continuously reset when ADEN is low.

When initiating a single ended conversion by setting the ADSC bit in ADCSRA, the conversion starts at the following rising edge of the ADC clock cycle.

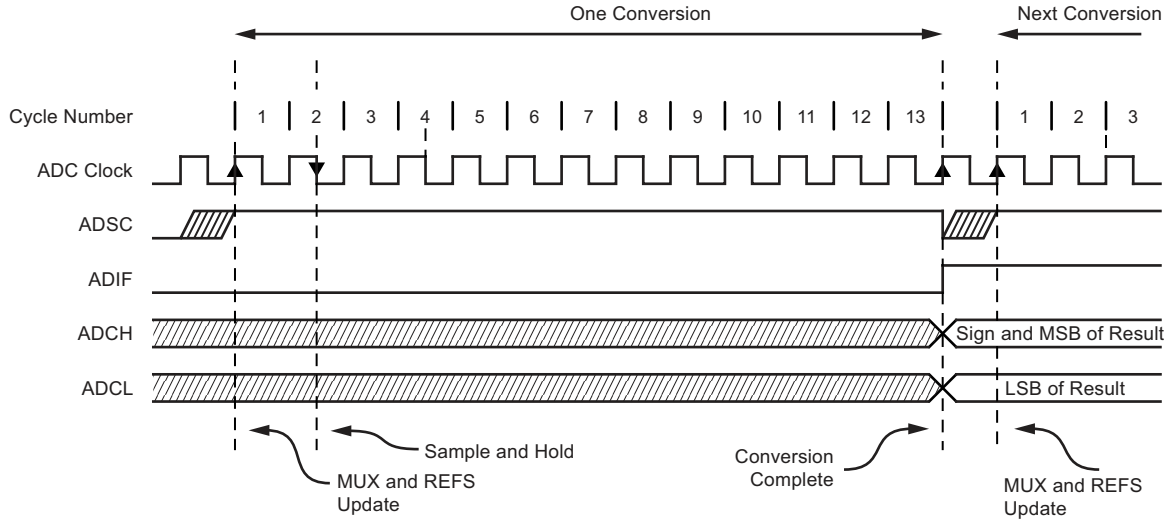
A normal conversion takes 13 ADC clock cycles. The first conversion after the ADC is switched on (ADEN in ADCSRA is set) takes 25 ADC clock cycles in order to initialize the analog circuitry, as shown in Figure 17-4 below.

Figure 17-4. ADC Timing Diagram, First Conversion (Single Conversion Mode)



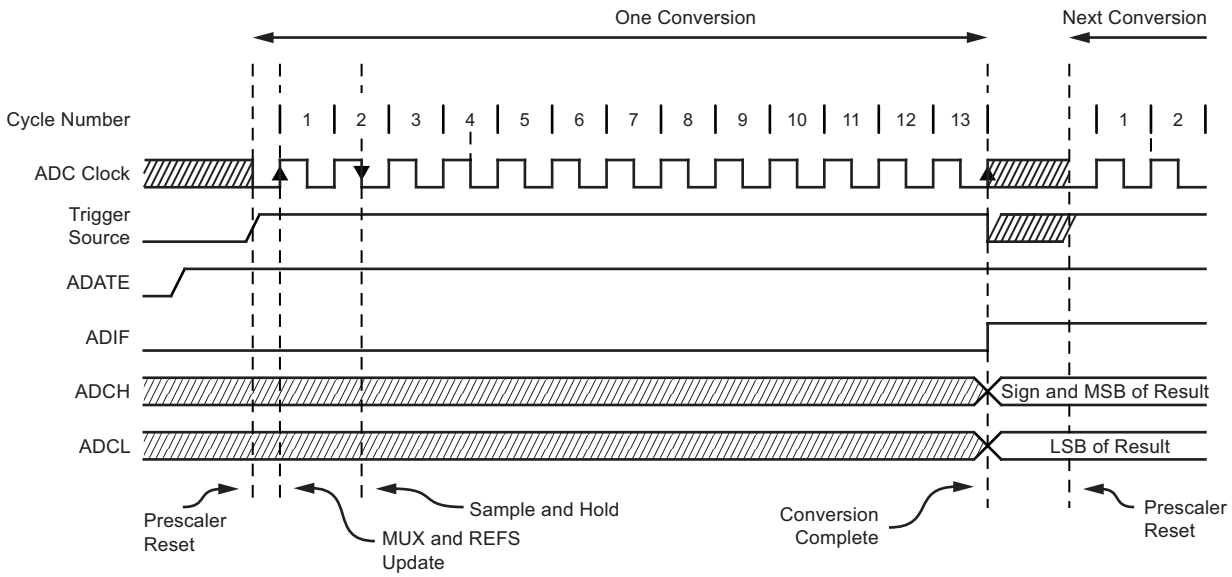
The actual sample-and-hold takes place 1.5 ADC clock cycles after the start of a normal conversion and 13.5 ADC clock cycles after the start of an first conversion. When a conversion is complete, the result is written to the ADC data registers, and ADIF is set. In single conversion mode, ADSC is cleared simultaneously. The software may then set ADSC again, and a new conversion will be initiated on the first rising ADC clock edge.

Figure 17-5. ADC Timing Diagram, Single Conversion



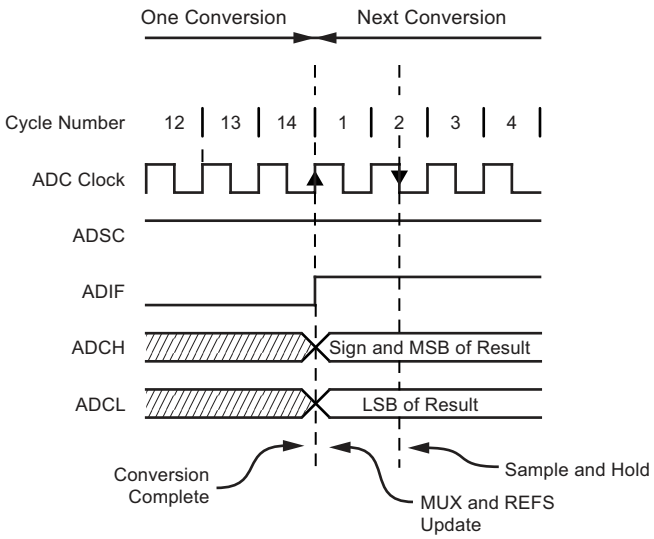
When auto triggering is used, the prescaler is reset when the trigger event occurs, as shown in Figure 17-6. This assures a fixed delay from the trigger event to the start of conversion. In this mode, the sample-and-hold takes place two ADC clock cycles after the rising edge on the trigger source signal. Three additional CPU clock cycles are used for synchronization logic.

Figure 17-6. ADC Timing Diagram, Auto Triggered Conversion



In free running mode, a new conversion will be started immediately after the conversion completes, while ADSC remains high. See Figure 17-7.

Figure 17-7. ADC Timing Diagram, Free Running Conversion



For a summary of conversion times, see Table 17-1.

Table 17-1. ADC Conversion Time

Condition	Sample & Hold (Cycles from Start of Conversion)	Conversion Time (Cycles)
First conversion	13.5	25
Normal conversions, single ended	1.5	13
Auto triggered conversions	2	13.5
Free running conversions	2.5	14

17.6 Changing Channel or Reference Selection

Bits MUXn and REFS0 in the ADMUX register are single buffered through a temporary register to which the CPU has random access. This ensures that the channels and reference selection only takes place at a safe point during the conversion. The channel and reference selection is continuously updated until a conversion is started. Once the conversion starts, the channel and reference selection is locked to ensure a sufficient sampling time for the ADC. Continuous updating resumes in the last ADC clock cycle before the conversion completes (ADIF in ADCSRA is set). Note that the conversion starts on the following rising ADC clock edge after ADSC is written. The user is thus advised not to write new channel or reference selection values to ADMUX until one ADC clock cycle after ADSC is written.

If auto triggering is used, the exact time of the triggering event can be indeterministic. Special care must be taken when updating the ADMUX register, in order to control which conversion will be affected by the new settings.

If both ADATE and ADEN is written to one, an interrupt event can occur at any time. If the ADMUX register is changed in this period, the user cannot tell if the next conversion is based on the old or the new settings. ADMUX can be safely updated in the following ways:

- When ADATE or ADEN is cleared.
- During conversion, minimum one ADC clock cycle after the trigger event.
- After a conversion, before the interrupt flag used as trigger source is cleared.

When updating ADMUX in one of these conditions, the new settings will affect the next ADC conversion.

17.6.1 ADC Input Channels

When changing channel selections, the user should observe the following guidelines to ensure that the correct channel is selected:

In single conversion mode, always select the channel before starting the conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the conversion to complete before changing the channel selection.

In free running mode, always select the channel before starting the first conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the first conversion to complete, and then change the channel selection. Since the next conversion has already started automatically, the next result will reflect the previous channel selection. Subsequent conversions will reflect the new channel selection.

17.6.2 ADC Voltage Reference

The ADC reference voltage (V_{REF}) indicates the conversion range for the ADC. Single ended channels that exceed V_{REF} will result in codes close to 0x3FF. V_{REF} can be selected as either AV_{CC}, or internal 1.1V reference. The internal 1.1V reference is generated from the internal bandgap reference (V_{BG}) through an internal amplifier.

The first ADC conversion result after switching reference voltage source may be inaccurate, and the user is advised to discard this result.

17.7 ADC Noise Canceler

The ADC features a noise canceler that enables conversion during sleep mode. This reduces noise induced from the CPU core and other I/O peripherals. The noise canceler can be used with ADC noise reduction and idle mode. To make use of this feature, the following procedure should be used:

- Make sure that the ADC is enabled and is not busy converting. Single conversion mode must be selected and the ADC conversion complete interrupt must be enabled.
- Enter ADC noise reduction mode (or Idle mode). The ADC will start a conversion once the CPU has been halted.
- If no other interrupts occur before the ADC conversion completes, the ADC interrupt will wake up the CPU and execute the ADC conversion complete interrupt routine. If another interrupt wakes up the CPU before the ADC conversion is complete, that interrupt will be executed, and an ADC conversion complete interrupt request will be generated when the ADC conversion completes. The CPU will remain in active mode until a new sleep command is executed.

Note that the ADC will not automatically be turned off when entering other sleep modes than Idle mode and ADC noise reduction mode. The user is advised to write zero to ADEN before entering such sleep modes to avoid excessive power consumption.

17.8 Analog Input Circuitry

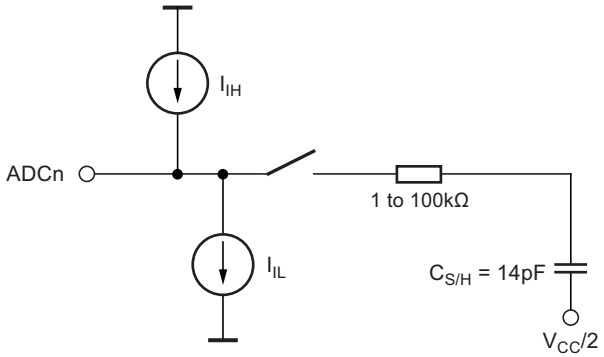
The analog input circuitry for single ended channels is illustrated in Figure 17-8 on page 152 An analog source applied to ADCn is subjected to the pin capacitance and input leakage of that pin, regardless of whether that channel is selected as input for the ADC. When the channel is selected, the source must drive the S/H capacitor through the series resistance (combined resistance in the input path).

The ADC is optimized for analog signals with an output impedance of approximately 10kΩ or less. If such a source is used, the sampling time will be negligible. If a source with higher impedance is used, the sampling time will depend on how long time the source needs to charge the S/H capacitor, which can vary widely.

With slowly varying signals the user is recommended to use sources with low impedance, only, since this minimizes the required charge transfer to the S/H capacitor.

In order to avoid distortion from unpredictable signal convolution, signal components higher than the nyquist frequency ($f_{ADC}/2$) should not be present. The user is advised to remove high frequency components with a low-pass filter before applying the signals as inputs to the ADC.

Figure 17-8. Analog Input Circuitry



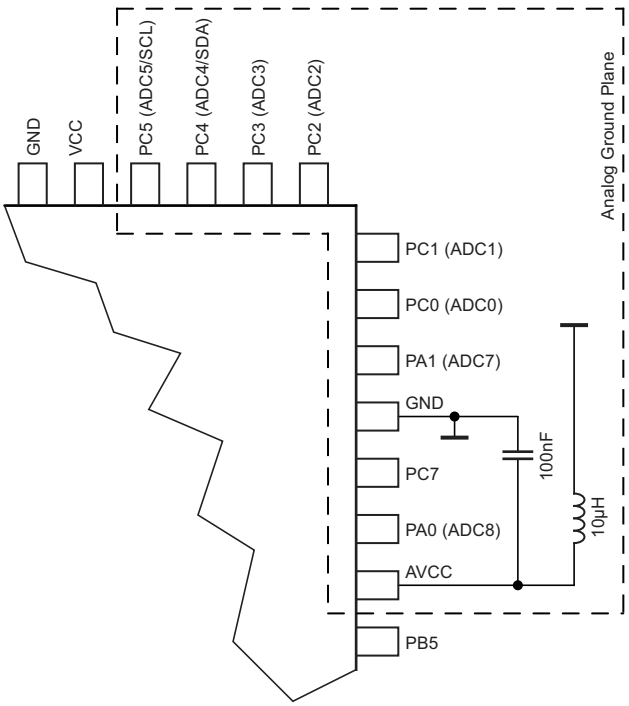
Note: The capacitor in the figure depicts the total capacitance, including the sample/hold capacitor and any stray or parasitic capacitance inside the device. The value given is worst case.

17.9 Analog Noise Canceling Techniques

Digital circuitry inside and outside the device generates EMI which might affect the accuracy of analog measurements. When conversion accuracy is critical, the noise level can be reduced by applying the following techniques:

- Keep analog signal paths as short as possible.
- Make sure analog tracks run over the analog ground plane.
- Keep analog tracks well away from high-speed switching digital tracks.
- If any port pin is used as a digital output, it mustn't switch while a conversion is in progress.
- The analog supply voltage pin (AV_{CC}) should be connected to the digital supply voltage pin (V_{CC}) via an LC network as shown in Figure 17-9.

Figure 17-9. ADC Power Connections



Where high ADC accuracy is required it is recommended to use ADC noise reduction mode, as described in Section 17.7 “ADC Noise Canceler” on page 151. This is especially the case when system clock frequency is above 1MHz, or when the ADC is used for reading the internal temperature sensor, as described in Section 17.12 “Temperature Measurement” on page 155. A good system design with properly placed, external bypass capacitors does reduce the need for using ADC noise reduction mode

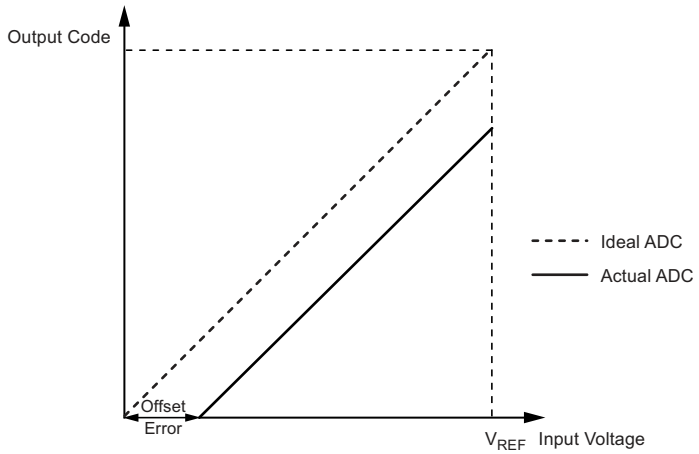
17.10 ADC Accuracy Definitions

An n-bit single-ended ADC converts a voltage linearly between GND and V_{REF} in 2^n steps (LSBs). The lowest code is read as 0, and the highest code is read as 2^n-1.

Several parameters describe the deviation from the ideal behavior:

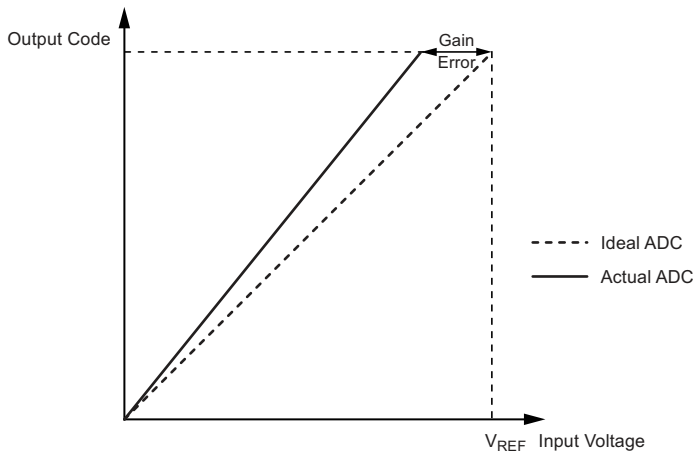
- Offset: The deviation of the first transition (0x000 to 0x001) compared to the ideal transition (at 0.5 LSB). Ideal value: 0 LSB.

Figure 17-10. Offset Error



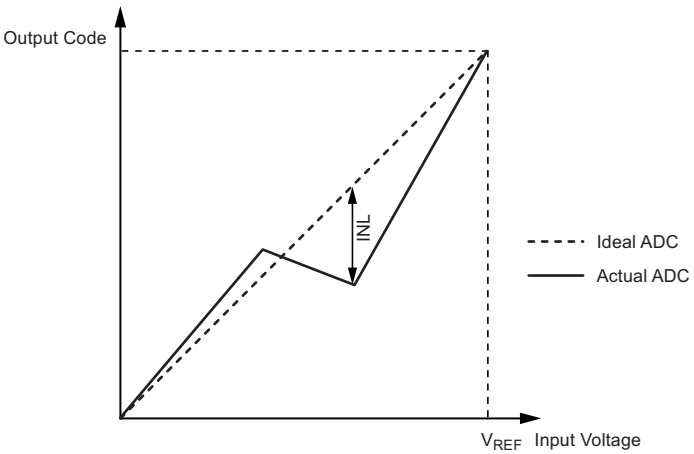
- Gain error: After adjusting for offset, the gain error is found as the deviation of the last transition (0x3FE to 0x3FF) compared to the ideal transition (at 1.5 LSB below maximum). Ideal value: 0 LSB

Figure 17-11. Gain Error



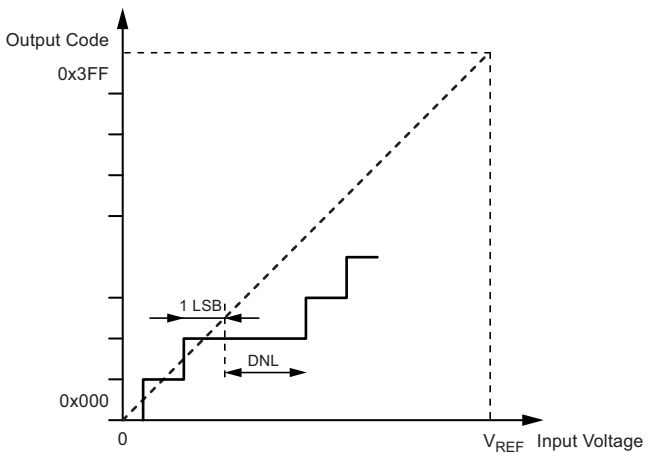
- Integral non-linearity (INL): After adjusting for offset and gain error, the INL is the maximum deviation of an actual transition compared to an ideal transition for any code. Ideal value: 0 LSB.

Figure 17-12.Integral Non-linearity (INL)



- Differential non-linearity (DNL): The maximum deviation of the actual code width (the interval between two adjacent transitions) from the ideal code width (1 LSB). Ideal value: 0 LSB.

Figure 17-13.Differential Non-linearity (DNL)



- Quantization error: Due to the quantization of the input voltage into a finite number of codes, a range of input voltages (1 LSB wide) will code to the same value. Always ±0.5 LSB.
- Absolute accuracy: The maximum deviation of an actual (unadjusted) transition compared to an ideal transition for any code. This is the compound effect of offset, gain error, differential error, non-linearity, and quantization error. Ideal value: ±0.5 LSB.

17.11 ADC Conversion Result

After the conversion is complete (ADIF is high), the conversion result can be found in the ADC result registers (ADCL, ADCH).

For single ended conversion, the result is

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

where V_{IN} is the voltage on the selected input pin and V_{REF} the selected voltage reference (see Table 17-3 on page 156 and Table 17-4 on page 156). 0x000 represents analog ground, and 0x3FF represents the selected reference voltage minus one LSB.

17.12 Temperature Measurement

The temperature measurement is based on an on-chip temperature sensor that is coupled to a single-ended ADC8 channel. Selecting the ADC8 channel by writing the MUX3..0 bits in ADMUX register to “1000” enables the temperature sensor. The internal 1.1V voltage reference must also be selected for the ADC voltage reference source in the temperature sensor measurement. When the temperature sensor is enabled, the ADC converter can be used in single conversion mode to measure the voltage over the temperature sensor.

The measured voltage has a linear relationship to the temperature as described in Table 17-2 The sensitivity is approximately 1 LSB / °C and the accuracy depends on the method of user calibration. Typically, the measurement accuracy after a single temperature calibration is ±10°C, assuming calibration at room temperature. Better accuracies are achieved by using two temperature points for calibration.

Table 17-2. Temperature versus Sensor Output Voltage (Typical Case)

Temperature	–40°C	+25°C	+125°C
ADC	230 LSB	300 LSB	370 LSB

The values described in Table 17-2 are typical values. However, due to process variation the temperature sensor output voltage varies from one chip to another. To be capable of achieving more accurate results the temperature measurement can be calibrated in the application software. The sofware calibration can be done using the formula:

```
T = k * [(ADCH << 8) | ADCL] + TOS
```

where ADCH and ADCL are the ADC data registers, k is the fixed slope coefficient and T_{OS} is the temperature sensor offset. Typically, k is very close to 1.0 and in single-point calibration the coefficient may be omitted. Where higher accuracy is required the slope coefficient should be evaluated based on measurements at two temperatures.

17.13 Register Description

17.13.1 ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
	–	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7,4 – Res: Reserved Bits**

These bits are reserved and will always read zero.

- **Bit 6 – REFS0: Reference Selection Bits**

This bit select the voltage reference for the ADC, as shown in Table 17-3. If this bit is changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set).

Table 17-3. Voltage Reference Selections for ADC

REFS0	Voltage Reference Selection
0	Internal 1.1V voltage reference
1	AV_{CC} reference

- **Bit 5 – ADLAR: ADC Left Adjust Result**

The ADLAR bit affects the presentation of the ADC conversion result in the ADC data register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC data register immediately, regardless of any ongoing conversions. For a complete description of this bit, see Section 17.13.3 “ADCL and ADCH – The ADC Data Register” on page 158.

- **Bits 3:0 – MUX3:0: Analog Channel Selection Bits**

The value of these bits selects which analog inputs are connected to the ADC. Selecting the single-ended channel ADC8 enables the temperature measurement. See Table 17-4 for details. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set).

Table 17-4. Input Channel Selections

MUX3..0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8(1)
1001	(reserved)
1010	(reserved)
1011	(reserved)
1100	(reserved)
1101	(reserved)
1110	1.1V (V_{BG})
1111	0V (GND)

Note: 1. Section 17.12 “Temperature Measurement” on page 155

17.13.2 ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ADEN: ADC Enable**

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

- **Bit 6 – ADSC: ADC Start Conversion**

In Single Conversion mode, write this bit to one to start each conversion. In free running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

- **Bit 5 – ADATE: ADC Auto Trigger Enable**

When this bit is written to one, auto triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC trigger select bits, ADTS in ADCSRB.

- **Bit 4 – ADIF: ADC Interrupt Flag**

This bit is set when an ADC conversion completes and the data registers are updated. The ADC conversion complete interrupt is executed if the ADIE bit and the I-bit in SREG are set. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag. Beware that if doing a read-modify-write on ADCSRA, a pending interrupt can be disabled. This also applies if the SBI and CBI instructions are used.

- **Bit 3 – ADIE: ADC Interrupt Enable**

When this bit is written to one and the I-bit in SREG is set, the ADC conversion complete interrupt is activated.

- **Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits**

These bits determine the division factor between the system clock frequency and the input clock to the ADC.

Table 17-5. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

17.13.3 ADCL and ADCH – The ADC Data Register

17.13.3.1 ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

17.13.3.2 ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

- **ADC9:0: ADC Conversion Result**

These bits represent the result from the conversion, as detailed in Section 17.11 “ADC Conversion Result” on page 155.

When an ADC conversion is complete, the result is found in these two registers.

When ADCL is read, the ADC data register is not updated until ADCH is read. Consequently, if the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH.

The ADLAR bit in ADMUX, and the MUXn bits in ADMUX affect the way the result is read from the registers. If ADLAR is set, the result is left adjusted. If ADLAR is cleared (default), the result is right adjusted.

17.13.4 ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7, 5:3 – Res: Reserved Bits**

These bits are reserved for future use. To ensure compatibility with future devices, these bits must be written to zero when ADCSRB is written.

- **Bit 2:0 – ADTS2:0: ADC Auto Trigger Source**

If ADATE in ADCSRA is written to one, the value of these bits selects which source will trigger an ADC conversion. If ADATE is cleared, the ADTS2:0 settings will have no effect. A conversion will be triggered by the rising edge of the selected interrupt flag. Note that switching from a trigger source that is cleared to a trigger source that is set, will generate a positive edge on the trigger signal. If ADEN in ADCSRA is set, this will start a conversion. Switching to free running mode (ADTS[2:0]=0) will not cause a trigger event, even if the ADC interrupt flag is set.

Table 17-6. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free running mode
0	0	1	Analog comparator
0	1	0	External interrupt request 0
0	1	1	Timer/Counter0 compare match A
1	0	0	Timer/Counter0 overflow
1	0	1	Timer/Counter1 compare match B
1	1	0	Timer/Counter1 overflow
1	1	1	Timer/Counter1 capture event

17.13.5 DIDR0 – Digital Input Disable Register 0

Bit	7	6	5	4	3	2	1	0	
	ADC7D	ADC6D	**ADC5D**	**ADC4D**	**ADC3D**	**ADC2D**	**ADC1D**	**ADC0D**	**DIDR0**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:0 – ADC7D..ADC0D: ADC7..0 Digital Input Disable**

When this bit is written logic one, the digital input buffer on the corresponding ADC pin is disabled. The corresponding PIN register bit will always read as zero when this bit is set. When an analog signal is applied to the ADC7..0 pin and the digital input from this pin is not needed, this bit should be written logic one to reduce power consumption in the digital input buffer.

18. debugWIRE On-chip Debug System

18.1 Features

- Complete program flow control
- Emulates all on-chip functions, both digital and analog, except RESET pin
- Real-time operation
- Symbolic debugging support (both at C and assembler source level, or for other HLLs)
- Unlimited number of program break points (using software break points)
- Non-intrusive operation
- Electrical characteristics identical to real device
- Automatic configuration system
- High-speed operation
- Programming of non-volatile memories

18.2 Overview

The debugWIRE on-chip debug system uses a one-wire, bi-directional interface to control the program flow, execute AVR® instructions in the CPU and to program the different non-volatile memories.

18.3 Physical Interface

When the debugWIRE enable (DWEN) fuse is programmed and lock bits are unprogrammed, the debugWIRE system within the target device is activated. The RESET port pin is configured as a wire-AND (open-drain) bi-directional I/O pin with pull-up enabled and becomes the communication gateway between target and emulator.

Figure 18-1. The debugWIRE Setup

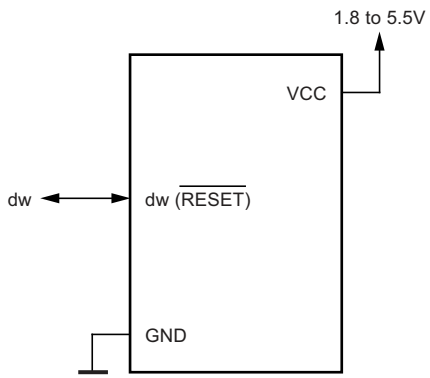


Figure 18-1 shows the schematic of a target MCU, with debugWIRE enabled, and the emulator connector. The system clock is not affected by debugWIRE and will always be the clock source selected by the CKSEL fuses.

When designing a system where debugWIRE will be used, the following observations must be made for correct operation:

- Pull-up resistors on the dW/(RESET) line must not be smaller than 10kΩ. The pull-up resistor is not required for debugWIRE functionality.
- Connecting the RESET pin directly to V_{CC} will not work.
- Capacitors connected to the RESET pin must be disconnected when using debugWire.
- All external reset sources must be disconnected.

18.4 Software Break Points

debugWIRE supports program memory break points by the AVR® break instruction. Setting a break point in AVR Studio® will insert a BREAK instruction in the program memory. The instruction replaced by the BREAK instruction will be stored. When program execution is continued, the stored instruction will be executed before continuing from the program memory. A break can be inserted manually by putting the BREAK instruction in the program.

The flash must be re-programmed each time a break point is changed. This is automatically handled by AVR Studio through the debugWIRE interface. The use of break points will therefore reduce the flash data retention. Devices used for debugging purposes should not be shipped to end customers.

18.5 Limitations of debugWIRE

The debugWIRE communication pin (dW) is physically located on the same pin as external reset (RESET). An external reset source is therefore not supported when the debugWIRE is enabled.

The debugWIRE system accurately emulates all I/O functions when running at full speed, i.e., when the program in the CPU is running. When the CPU is stopped, care must be taken while accessing some of the I/O registers via the debugger (AVR Studio).

The debugWIRE interface is asynchronous, which means that the debugger needs to synchro-nize to the system clock. If the system clock is changed by software (e.g. by writing CLKPS bits) communication via debugWIRE may fail. Also, clock frequencies below 100kHz may cause communication problems.

A programmed DWEN fuse enables some parts of the clock system to be running in all sleep modes. This will increase the power consumption while in sleep. Thus, the DWEN Fuse should be disabled when debugWire is not used.

18.6 Register Description

18.6.1 DWDR – debugWire Data Register

Bit	7	6	5	4	3	2	1	0	
	DWDR[7:0]								DWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The DWDR register provides a communication channel from the running program in the MCU to the debugger. This register is only accessible by the debugWIRE and can therefore not be used as a general purpose register in the normal operations.

19. Self-Programming the Flash

The device provides a self-programming mechanism for downloading and uploading program code by the MCU itself. The self-programming can use any available data interface and associated protocol to read code and write (program) that code into the program memory. The SPM instruction is disabled by default but it can be enabled by programming the SELFPRGEN fuse (to “0”).

The program memory is updated in a page by page fashion. Before programming a page with the data stored in the temporary page buffer, the page must be erased. The temporary page buffer is filled one word at a time using SPM and the buffer can be filled either before the page erase command or between a page erase and a page write operation:

Alternative 1, fill the buffer before a page erase:

- Fill temporary page buffer
- Perform a page erase
- Perform a page write

Alternative 2, fill the buffer after page erase:

- Perform a page erase
- Fill temporary page buffer
- Perform a page write

If only a part of the page needs to be changed, the rest of the page must be stored (for example in the temporary page buffer) before the erase, and then be re-written. Alternative 1 provides an effective read-modify-write feature which allows the user software to first read the page, do the necessary changes, and then write back the modified data. If alternative 2 is used, it is not possible to read the old data while loading since the page is already erased. The temporary page buffer can be accessed in a random sequence. It is essential that the page address used in both the page erase and page write operation is addressing the same page.

19.1 Performing Page Erase by SPM

To execute page erase, set up the address in the Z-pointer, write “00000011” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The data in R1 and R0 is ignored. The page address must be written to PCPAGE in the Z-register. Other bits in the Z-pointer will be ignored during this operation.

The CPU is halted during the page erase operation.

19.2 Filling the Temporary Buffer (Page Loading)

To write an instruction word, set up the address in the Z-pointer and data in R1:R0, write “00000001” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The content of PCWORD in the Z-register is used to address the data in the temporary buffer. The temporary buffer will auto-erase after a page write operation or by writing the CTPB bit in SPMCSR. It is also erased after a system reset. Note that it is not possible to write more than one time to each address without erasing the temporary buffer.

If the EEPROM is written in the middle of an SPM page load operation, all data loaded will be lost.

19.3 Performing a Page Write

To execute page write, set up the address in the Z-pointer, write “00000101” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The data in R1 and R0 is ignored. The page address must be written to PCPAGE. Other bits in the Z-pointer must be written to zero during this operation.

The CPU is halted during the Page Write operation.

19.4 Addressing the Flash During Self-Programming

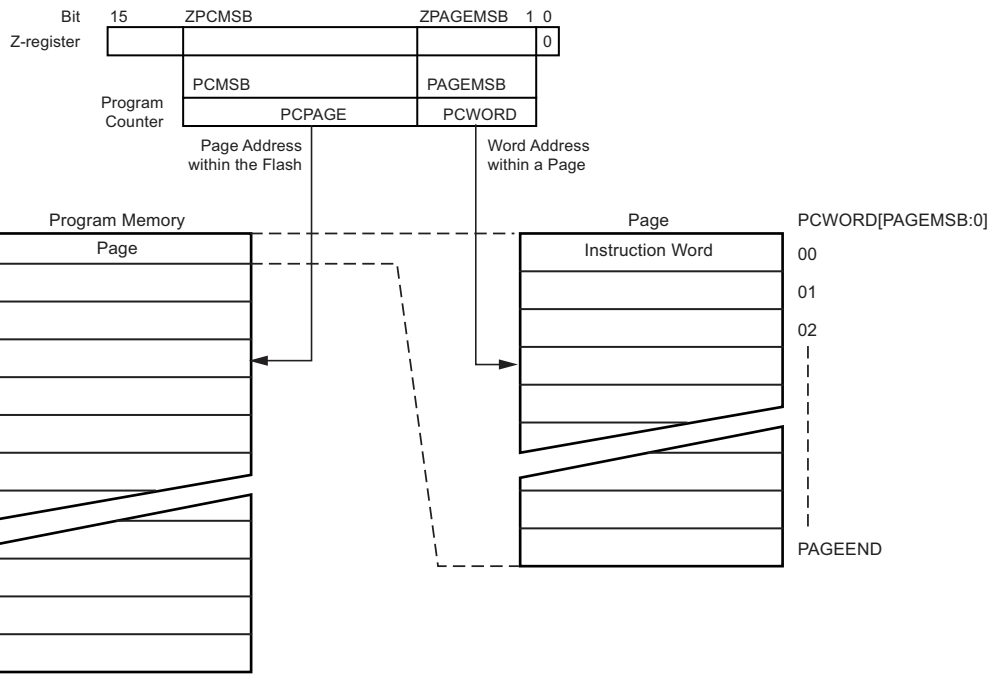
The Z-pointer is used to address the SPM commands.

Bit	15	14	13	12	11	10	9	8
ZH (R31)	Z15	Z14	Z13	Z12	Z11	Z10	Z9	Z8
ZL (R30)	Z7	Z6	Z5	Z4	Z3	Z2	Z1	Z0
	7	6	5	4	3	2	1	0

Since the flash is organized in pages (see Table 20-7 on page 170), the program counter can be treated as having two different sections. One section, consisting of the least significant bits, is addressing the words within a page, while the most significant bits are addressing the pages. This is shown in Figure 19-1. Note that the page erase and page write operations are addressed independently. Therefore it is of major importance that the software addresses the same page in both the page erase and page write operation.

The LPM instruction uses the Z-pointer to store the address. Since this instruction addresses the flash byte-by-byte, also the LSB (bit Z0) of the Z-pointer is used.

Figure 19-1. Addressing the Flash During SPM(1)



Note: 1. The different variables used in Figure 19-1 are listed in Table 20-7 on page 170.

19.4.1 EEPROM Write Prevents Writing to SPMCSR

Note that an EEPROM write operation will block all software programming to flash. Reading the fuses and lock bits from software will also be prevented during the EEPROM write operation. It is recommended that the user checks the status bit (EEPE) in the EECR register and verifies that the bit is cleared before writing to the SPMCSR register.

19.4.2 Reading the Fuse and Lock Bits from Software

It is possible to read both the fuse and lock bits from software. To read the lock bits, load the Z-pointer with 0x0001 and set the RFLB and SELFPRGEN bits in SPMCSR. When an LPM instruction is executed within three CPU cycles after the RFLB and SELFPRGEN bits are set in SPMCSR, the value of the lock bits will be loaded in the destination register. The RFLB and SELFPRGEN bits will auto-clear upon completion of reading the lock bits or if no LPM instruction is executed within three CPU cycles or no SPM instruction is executed within four CPU cycles. When RFLB and SELFPRGEN are cleared, LPM will work as described in the instruction set manual

.

Bit	7	6	5	4	3	2	1	0
Rd	–	–	–	–	–	–	LB2	LB1

The algorithm for reading the fuse low byte is similar to the one described above for reading the Lock bits. To read the fuse low byte, load the Z-pointer with 0x0000 and set the RFLB and SELFPRGEN bits in SPMCSR. When an LPM instruction is executed within three cycles after the RFLB and SELFPRGEN bits are set in the SPMCSR, the value of the fuse low byte (FLB) will be loaded in the destination register as shown below.See Table 20-5 on page 169 for a detailed description and mapping of the fuse low byte.

Bit	7	6	5	4	3	2	1	0
Rd	FLB7	FLB6	FLB5	FLB4	FLB3	FLB2	FLB1	FLB0

Similarly, when reading the fuse high byte (FHB), load 0x0003 in the Z-pointer. When an LPM instruction is executed within three cycles after the RFLB and SELFPRGEN bits are set in the SPMCSR, the value of the fuse high byte will be loaded in the destination register as shown below. See Table 20-4 on page 169 for detailed description and mapping of the fuse high byte.

Bit	7	6	5	4	3	2	1	0
Rd	FHB7	FHB6	FHB5	FHB4	FHB3	FHB2	FHB1	FHB0

Fuse extended byte can be read by loading the Z-pointer with 0x0002. When an LPM instruction is executed within three cycles after the RFLB and SPMEN bits are set in the SPMCSR, the value of the fuse extended byte (FEB) will be loaded in the destination register as shown below. See Table 20-3 on page 168 for detailed description and mapping of the fuse extended byte.

Bit	7	6	5	4	3	2	1	0
Rd	FEB7	FEB6	FEB5	FEB4	FEB3	FEB2	FEB1	FEB0

Fuse and lock bits that are programmed, will be read as zero. Fuse and lock bits that are unprogrammed, will be read as one.

19.4.3 Preventing Flash Corruption

During periods of low V_{CC}, the flash program can be corrupted because the supply voltage is too low for the CPU and the flash to operate properly. These issues are the same as for board level systems using the flash, and the same design solutions should be applied.

A flash program corruption can be caused by two situations when the voltage is too low. First, a regular write sequence to the flash requires a minimum voltage to operate correctly. Secondly, the CPU itself can execute instructions incorrectly, if the supply voltage for executing instructions is too low.

Flash corruption can easily be avoided by following these design recommendations (one is sufficient):

1. Keep the AVR® RESET active (low) during periods of insufficient power supply voltage. This can be done by enabling the internal brown-out detector (BOD) if the operating voltage matches the detection level. If not, an external low V_{CC} reset protection circuit can be used. If a reset occurs while a write operation is in progress, the write operation will be completed provided that the power supply voltage is sufficient.
2. Keep the AVR core in power-down sleep mode during periods of low V_{CC}. This will prevent the CPU from attempting to decode and execute instructions, effectively protecting the SPMCSR register and thus the flash from unintentional writes.

19.4.4 Programming Time for Flash when Using SPM

The calibrated oscillator is used to time flash accesses. Table 19-1 shows the typical programming time for flash accesses from the CPU.

Table 19-1. SPM Programming Time

Symbol	Min Programming Time	Max Programming Time
Flash write (page erase, page write, and write lock bits by SPM)	3.7ms	4.5ms

19.4.5 Simple Assembly Code Example for a Boot Loader

Note that the RWWSB bit will always be read as zero in Atmel® ATtiny88. Nevertheless, to ensure compatibility with devices supporting read-while-write it is recommended to check this bit as shown in the code example.

```
;-the routine writes one page of data from RAM to Flash
; the first data location in RAM is pointed to by the Y pointer
; the first data location in Flash is pointed to by the Z-pointer
;-error handling is not included
;-the routine must be placed inside the Boot space
; (at least the Do_spm sub routine). Only code inside NRWW section can
; be read during Self-Programming (Page Erase and Page Write).
;-registers used: r0, r1, temp1 (r16), temp2 (r17), looplo (r24),
; loophi (r25), spmcrval (r20)
; storing and restoring of registers is not included in the routine
; register usage can be optimized at the expense of code size
;-It is assumed that either the interrupt table is moved to the Boot
; loader section or that the interrupts are disabled.
.equ PAGESIZEB = PAGESIZE*2      ;PAGESIZEB is page size in BYTES, not words
.org SMALLBOOTSTART
Write_page:
   ;    Page Erase
   ldi         spmcrval, (1<<PGERS) | (1<<SELFPRGEN)
   rcall       Do_spm

   ;    re-enable the RWW section
   ldi         spmcrval, (1<<CTPB) | (1<<SELFPRGEN)
   rcall       Do_spm

   ;    transfer data from RAM to Flash page buffer
   ldi         looplo, low(PAGESIZEB)     ;init loop variable
   ldi         loophi, high(PAGESIZEB)    ;not required for PAGESIZEB<=256
Wrloop:
   ld          r0, Y+
   ld          r1, Y+
   ldi         spmcrval, (1<<SELFPRGEN)
   rcall       Do_spm
   adiw        ZH:ZL, 2
   sbiw        loophi:looplo, 2           ;use subi for PAGESIZEB<=256
   brne        Wrloop

   ;    execute Page Write
   subi        ZL, low(PAGESIZEB)         ;restore pointer
   sbci        ZH, high(PAGESIZEB)        ;not required for PAGESIZEB<=256
   ldi         spmcrval, (1<<PGWRT) | (1<<SELFPRGEN)
   rcall       Do_spm

   ;    re-enable the RWW section
```

```
   ldi         spmcrval, (1<<CTPB) | (1<<SELFPRGEN)
   rcall       Do_spm

   ;    read back and check, optional
   ldi         looplo, low(PAGESIZEB)     ;init loop variable
   ldi         loophi, high(PAGESIZEB)    ;not required for PAGESIZEB<=256
   subi        YL, low(PAGESIZEB)         ;restore pointer
   sbci        YH, high(PAGESIZEB)
Rdloop:
   lpm         r0, Z+
   ld          r1, Y+
   cpse        r0, r1
   rjmp        Error
   sbiw        loophi:looplo, 1           ;use subi for PAGESIZEB<=256
   brne        Rdloop

   ;    return to RWW section
   ;    verify that RWW section is safe to read
Return:
   in          temp1, SPMCSR
   sbrs        temp1, RWWSB               ; If RWWSB is set, the RWW section is not
                                          ready yet
   ret
   ;    re-enable the RWW section
   ldi         spmcrval, (1<<CTPB) | (1<<SELFPRGEN)
   rcall       Do_spm
   rjmp        Return

Do_spm:
   ;    check for previous SPM complete
Wait_spm:
   in          temp1, SPMCSR
   sbrc        temp1, SELFPRGEN
   rjmp        Wait_spm
   ;    input: spmcrval determines SPM action
   ;    disable interrupts if enabled, store status
   in  temp2, SREG
   cli
   ;    check that no EEPROM write access is present
Wait_ee:
   sbic        EECR, EEPE
   rjmp        Wait_ee
   ;    SPM timed sequence
   out         SPMCSR, spmcrval
   spm
   ;    restore SREG (to enable interrupts if originally enabled)
   out         SREG, temp2
   ret
```

19.5 Register Description

19.5.1 SPMCSR – Store Program Memory Control and Status Register

The store program memory control and status register contains the control bits needed to control the program memory operations.

Bit	7	6	5	4	3	2	1	0	
	-	RWWSB	–	CTPB	RFLB	PGWRT	PGERS	SELFPRGEN	SPMCSR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – Res: Reserved Bit**

These bits are reserved and will always read zero.

- **Bit 6 – RWWSB: Read-While-Write Section Busy**

This bit is for compatibility with devices supporting read-while-write. It will always read as zero in Atmel® ATtiny88.

- **Bit 5 – Res: Reserved Bit**

These bits are reserved and will always read zero.

- **Bit 4 – CTPB: Clear Temporary Page Buffer**

If the CTPB bit is written while filling the temporary page buffer, the temporary page buffer will be cleared and the data will be lost.

- **Bit 3 – RFLB: Read Fuse and Lock Bits**

An LPM instruction within three cycles after RFLB and SELFPRGEN are set in the SPMCSR register, will read either the lock bits or the fuse bits (depending on Z0 in the Z-pointer) into the destination register. See Section 19.4.2 “Reading the Fuse and Lock Bits from Software” on page 164 for details.

- **Bit 2 – PGWRT: Page Write**

If this bit is written to one at the same time as SELFPRGEN, the next SPM instruction within four clock cycles executes page write, with the data stored in the temporary buffer. The page address is taken from the high part of the Z-pointer. The data in R1 and R0 are ignored. The PGWRT bit will auto-clear upon completion of a page write, or if no SPM instruction is executed within four clock cycles. The CPU is halted during the entire page write operation.

- **Bit 1 – PGERS: Page Erase**

If this bit is written to one at the same time as SELFPRGEN, the next SPM instruction within four clock cycles executes page Erase. The page address is taken from the high part of the Z-pointer. The data in R1 and R0 are ignored. The PGERS bit will auto-clear upon completion of a page erase, or if no SPM instruction is executed within four clock cycles. The CPU is halted during the entire page write operation.

- **Bit 0 – SELFPRGEN: Self Programming Enable**

This bit enables the SPM instruction for the next four clock cycles. If written to one together with either CTPB, RFLB, PGWRT, or PGERS, the following SPM instruction will have a special meaning, see description above. If only SELFPRGEN is written, the following SPM instruction will store the value in R1:R0 in the temporary page buffer addressed by the Z-pointer. The LSB of the Z-pointer is ignored. The SELFPRGEN bit will auto-clear upon completion of an SPM instruction, or if no SPM instruction is executed within four clock cycles. During page erase and page write, the SELFPRGEN bit remains high until the operation is completed.

Writing any other combination than “10001”, “01001”, “00101”, “00011” or “00001” in the lower five bits will have no effect.

20. Memory Programming

20.1 Program And Data Memory Lock Bits

The Atmel® ATtiny88 provides two lock bits which can be left unprogrammed (“1”) or can be programmed (“0”) to obtain the additional features listed in Table 20-2. The lock bits can only be erased to “1” with the chip erase command. The Atmel ATtiny88 has no separate boot loader section. The SPM instruction is enabled for the whole flash if the SELFPRGEN fuse is programmed (“0”), otherwise it is disabled.

Table 20-1. Lock Bit Byte(1)

Lock Bit Byte	Bit No	Description	Default Value
	7	–	1 (unprogrammed)
	6	–	1 (unprogrammed)
	5	–	1 (unprogrammed)
	4	–	1 (unprogrammed)
	3	–	1 (unprogrammed)
	2	–	1 (unprogrammed)
LB2	1	Lock bit	1 (unprogrammed)
LB1	0	Lock bit	1 (unprogrammed)

Note: 1. “1” means unprogrammed, “0” means programmed.

Table 20-2. Lock Bit Protection Modes(1)(2)

Memory Lock Bits			Protection Type
LB Mode	LB2	LB1	
1	1	1	No memory lock features enabled.
2	1	0	Further programming of the flash and EEPROM is disabled in parallel and serial programming mode. The fuse bits are locked in both serial and parallel programming mode.(1)
3	0	0	Further programming and verification of the flash and EEPROM is disabled in parallel and serial programming mode. The fuse bits are locked in both serial and parallel programming mode.(1)

Notes: 1. Program the fuse bits before programming the LB1 and LB2.
2. “1” means unprogrammed, “0” means programmed

20.2 Fuse Bits

The Atmel ATtiny88 has three fuse bytes. Table 20-3 – Table 20-5 on page 169 describe briefly the functionality of all the fuses and how they are mapped into the fuse bytes. Note that the fuses are read as logical zero, “0”, if they are programmed.

Table 20-3. Fuse Extended Byte

Fuse Extended Byte	Bit No	Description	Default Value
–	7	–	1
–	6	–	1
–	5	–	1
–	4	–	1
–	3	–	1
–	2	–	1
–	1	–	1
SELFPRGEN (1)	0	Self Programming Enable	1 (unprogrammed)

Note: 1. Enables SPM instruction. See Section 19. “Self-Programming the Flash” on page 162.

Table 20-4. Fuse High Byte

Fuse High Byte	Bit No	Description	Default Value
RSTDISBL [1] [2]	7	External reset disable	1 (unprogrammed)
DWEN[2]	6	debugWIRE enable	1 (unprogrammed)
SPIEN[3]	5	Enable serial program and data downloading	0 (programmed) (SPI programming enabled)
WDTON[4]	4	Watchdog timer always on	1 (unprogrammed)
EESAVE	3	EEPROM memory preserved through chip erase	1 (unprogrammed) (EEPROM not preserved)
BODLEVEL2[5]	2	Brown-out detector trigger level	1 (unprogrammed)
BODLEVEL1[5]	1	Brown-out detector trigger level	1 (unprogrammed)
BODLEVEL0[5]	0	Brown-out detector trigger level	1 (unprogrammed)

Notes:
1. See Section 10.3.3 “Alternate Functions of Port C” on page 61 for description of RSTDISBL fuse.
2. Programming this fuse bit will change the functionality of the RESET pin and render further programming via the serial interface impossible. The fuse bit can be unprogrammed using the parallel programming algorithm (see Section 20.7 “Parallel Programming” on page 172).
3. The SPIEN fuse is not accessible in serial programming mode.
4. See Section 8.5.2 “WDTCSR – Watchdog Timer Control Register” on page 42 for details.
5. See Table 21-5 on page 187 for BODLEVEL fuse decoding.

Table 20-5. Fuse Low Byte

Fuse Low Byte	Bit No	Description	Default Value
CKDIV8[4]	7	Divide clock by 8	0 (programmed)
CKOUT[3]	6	Clock output	1 (unprogrammed)
SUT1	5	Select start-up time	1 (unprogrammed)[1]
SUT0	4	Select start-up time	0 (programmed)[1]
-	3	-	1 (unprogrammed)[2]
-	2	-	1 (unprogrammed)[2]
CKSEL1	1	Select clock source	1 (unprogrammed)[2]
CKSEL0	0	Select clock source	0 (programmed)[2]

Note:
1. The default value of SUT1..0 results in maximum start-up time for the default clock source. See Table 6-4 on page 27 for details.
2. The default setting of CKSEL1..0 results in internal oscillator at 8MHz. See Table 6-3 on page 27 for details.
3. The CKOUT fuse allows the system clock to be output on PORTB0. See Section 6.6 “Clock Output Buffer” on page 29 for details.
4. See Section 6.7 “System Clock Prescaler” on page 29 for details.

The status of the fuse bits is not affected by chip erase. Note that the fuse bits are locked if Lock bit1 (LB1) is programmed. program the fuse bits before programming the lock bits.

20.2.1 Latching of Fuses

The fuse values are latched when the device enters programming mode and changes of the fuse values will have no effect until the part leaves programming mode. This does not apply to the EESAVE fuse which will take effect once it is programmed. The fuses are also latched on power-up in normal mode.

20.3 Signature Bytes

All Atmel® microcontrollers have a three-byte signature code which identifies the device. This code can be read in both serial and parallel mode, also when the device is locked. The three bytes reside in a separate address space. For the Atmel ATtiny88 the signature bytes are given in Table 20-6.

Table 20-6. Device ID

Part	Signature Bytes Address		
	0x000	0x001	0x002
ATtiny88	0x1E	0x93	0x11

20.4 Calibration Byte

The Atmel ATtiny88 has a byte calibration value for the internal oscillator. This byte resides in the high byte of address 0x000 in the signature address space. During reset, this byte is automatically written into the OSCCAL register to ensure correct frequency of the calibrated oscillator.

20.5 Page Size

Table 20-7. No. of Words in a Page and No. of Pages in the Flash

Device	Flash Size	Page Size	PCWORD	No. of Pages	PCPAGE	PCMSB
ATtiny88	4Kwords (8Kbytes)	32 words	PC[4:0]	128	PC[11:5]	11

Table 20-8. No. of Words in a Page and No. of Pages in the EEPROM

Device	EEPROM Size	Page Size	PCWORD	No. of Pages	PCPAGE	EEAMSB
ATtiny88	64 bytes	4 bytes	EEA[1:0]	16	EEA[5:2]	5

20.6 Parallel Programming Parameters, Pin Mapping, and Commands

This section describes how to parallel program and verify flash program memory, EEPROM data memory, memory lock bits, and fuse bits in the Atmel ATtiny88. Pulses are assumed to be at least 250 ns unless otherwise noted.

20.6.1 Signal Names

In this section, some pins of the Atmel ATtiny88 are referenced by signal names describing their functionality during parallel programming, see Figure 20-1 on page 171 and Table 20-9 on page 171. Pins not described in the following table are referenced by pin names.

Figure 20-1. Parallel Programming

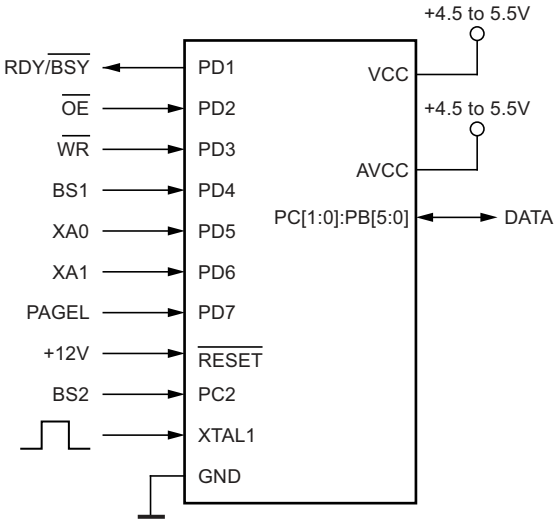


Table 20-9. Pin Name Mapping

Signal Name in Programming Mode	Pin Name	I/O	Function
RDY/$\overline{BSY}$	PD1	O	0: Device is busy programming, 1: Device is ready for new command
$\overline{OE}$	PD2	I	Output enable (active low)
$\overline{WR}$	PD3	I	Write pulse (active low)
BS1	PD4	I	Byte select 1 (“0” selects low byte, “1” selects high byte)
XA0	PD5	I	XTAL action bit 0
XA1	PD6	I	XTAL action bit 1
PAGEL	PD7	I	Program memory and EEPROM data page load
BS2	PC2	I	Byte select 2 (“0” selects low byte, “1” selects 2’nd high byte)
DATA	{PC[1:0]: PB[5:0]}	I/O	Bi-directional data bus (output when $\overline{OE}$ is low)

Note: $V_{CC} - 0.3V < AV_{CC} < V_{CC} + 0.3V$, however, AV_{CC} should always be within 4.5 – 5.5V

Table 20-10. Pin Values Used to Enter Programming Mode

Pin	Symbol	Value
PAGEL	Prog_enable[3]	0
XA1	Prog_enable[2]	0
XA0	Prog_enable[1]	0
BS1	Prog_enable[0]	0

The XA1/XA0 pins determine the action executed when the CLKI pin is given a positive pulse. The bit coding is shown in Table 20-11.

Table 20-11. XA1 and XA0 Coding

XA1	XA0	Action when CLKI is Pulsed
0	0	Load flash or EEPROM address (high or low address byte determined by BS1).
0	1	Load data (high or low data byte for flash determined by BS1).
1	0	Load command
1	1	No action, idle

When pulsing $\overline{WR}$ or $\overline{OE}$, the command loaded determines the action executed. The different commands are shown in Table 20-12.

Table 20-12. Command Byte Bit Coding

Command Byte	Command Executed
1000 0000	Chip erase
0100 0000	Write fuse bits
0010 0000	Write lock bits
0001 0000	Write flash
0001 0001	Write EEPROM
0000 1000	Read signature bytes and calibration byte
0000 0100	Read fuse and lock bits
0000 0010	Read flash
0000 0011	Read EEPROM

20.7 Parallel Programming

20.7.1 Enter Programming Mode

The following algorithm puts the device in parallel (high-voltage) programming mode:

1. Set prog_enable pins listed in Table 20-10 on page 171 to “0000”, RESET pin to 0V and V_{CC} to 0V.
2. Apply 4.5 – 5.5V between V_{CC} and GND.

Ensure that V_{CC} reaches at least 2.7V within the next 20µs.

3. Wait 20 – 60µs, and apply 11.5 – 12.5V to RESET.
4. Keep the prog_enable pins unchanged for at least 10µs after the high-voltage has been applied to ensure the prog_enable signature has been latched.
5. Wait at least 300µs before giving any parallel programming commands.
6. Exit programming mode by power the device down or by bringing RESET pin to 0V.

If the rise time of the V_{CC} is unable to fulfill the requirements listed above, the following alternative algorithm can be used.

1. Set prog_enable pins listed in Table 20-10 on page 171 to “0000”, RESET pin to 0V and V_{CC} to 0V.
2. Apply 4.5 – 5.5V between V_{CC} and GND.
3. Monitor V_{CC}, and as soon as V_{CC} reaches 0.9 – 1.1V, apply 11.5 – 12.5V to RESET.
4. Keep the prog_enable pins unchanged for at least 10µs after the high-voltage has been applied to ensure the prog_enable signature has been latched.
5. Wait until V_{CC} actually reaches 4.5 -5.5V before giving any parallel programming commands.
6. Exit programming mode by power the device down or by bringing RESET pin to 0V.

20.7.2 Considerations for Efficient Programming

The loaded command and address are retained in the device during programming. For efficient programming, the following should be considered.

- The command needs only be loaded once when writing or reading multiple memory locations.
- Skip writing the data value 0xFF, that is the contents of the entire EEPROM (unless the EESAVE fuse is programmed) and flash after a chip erase.
- Address high byte needs only be loaded before programming or reading a new 256 word window in flash or 256 byte EEPROM. This consideration also applies to signature bytes reading.

20.7.3 Chip Erase

The chip erase will erase the Flash and EEPROM[1] memories plus lock bits. The lock bits are not reset until the program memory has been completely erased. The fuse bits are not changed. A chip erase must be performed before the flash and/or EEPROM are reprogrammed.

Note: 1. The EEPROM memory is preserved during chip erase if the EESAVE fuse is programmed.

Load command “chip erase”:

1. Set XA1, XA0 to “10”. This enables command loading.
2. Set BS1 to “0”.
3. Set DATA to “1000 0000”. This is the command for chip erase.
4. Give CLKI a positive pulse. This loads the command.
5. Give $\overline{WR}$ a negative pulse. This starts the chip erase. RDY/$\overline{BSY}$ goes low.
6. Wait until RDY/$\overline{BSY}$ goes high before loading a new command.

20.7.4 Programming the Flash

The flash is organized in pages, see Table 20-7 on page 170. When programming the flash, the program data is latched into a page buffer. This allows one page of program data to be programmed simultaneously. The following procedure describes how to program the entire flash memory:

A. Load command “write flash”

1. Set XA1, XA0 to “10”. This enables command loading.
2. Set BS1 to “0”.
3. Set DATA to “0001 0000”. This is the command for Write Flash.
4. Give CLKI a positive pulse. This loads the command.

B. Load address low byte

1. Set XA1, XA0 to “00”. This enables address loading.
2. Set BS1 to “0”. This selects low address.
3. Set DATA = address low byte (0x00 – 0xFF).
4. Give CLKI a positive pulse. This loads the address low byte.

C. Load data low byte

1. Set XA1, XA0 to “01”. This enables data loading.
2. Set DATA = data low byte (0x00 – 0xFF).
3. Give CLKI a positive pulse. This loads the data byte.

D. Load data high byte

1. Set BS1 to “1”. This selects high data byte.
2. Set XA1, XA0 to “01”. This enables data loading.
3. Set DATA = data high byte (0x00 – 0xFF).
4. Give CLKI a positive pulse. This loads the data byte.

E. Latch data

1. Set BS1 to “1”. This selects high data byte.
2. Give PAGEL a positive pulse. This latches the data bytes. (See Figure 20-3 on page 175 for signal waveforms)

F. Repeat B through E until the entire buffer is filled or until all data within the page is loaded.

While the lower bits in the address are mapped to words within the page, the higher bits address the pages within the FLASH. This is illustrated in Figure 20-2 on page 174. Note that if less than eight bits are required to address words in the page (page size < 256), the most significant bit(s) in the address low byte are used to address the page when performing a page write.

G. Load address high byte

1. Set XA1, XA0 to “00”. This enables address loading.
2. Set BS1 to “1”. This selects high address.
3. Set DATA = address high byte (0x00 – 0xFF).
4. Give CLKI a positive pulse. This loads the address high byte.

H. Program page

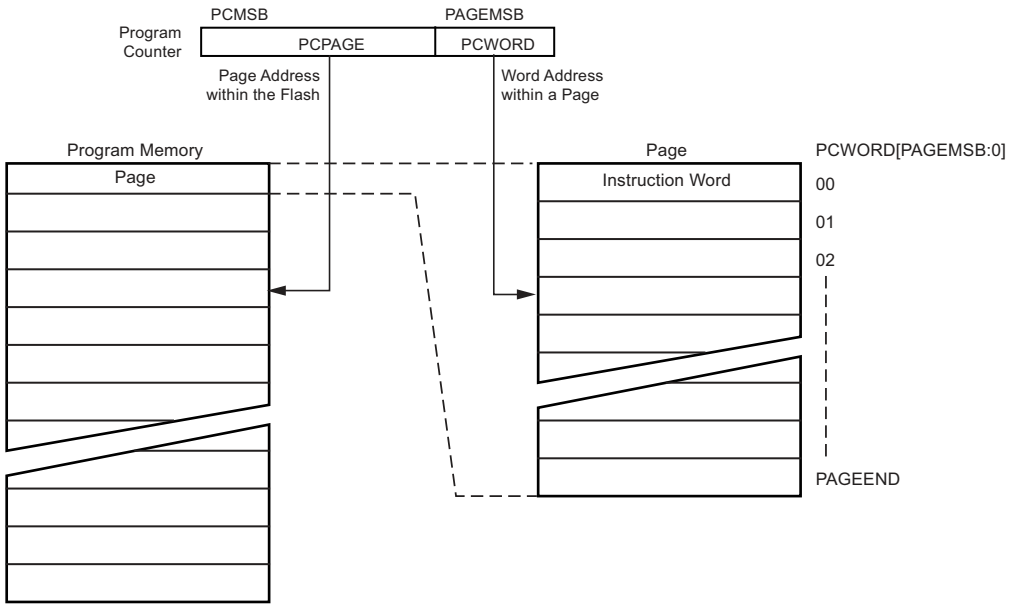
1. Give $\overline{WR}$ a negative pulse. This starts programming of the entire page of data. RDY/$\overline{BSY}$ goes low.
2. Wait until RDY/$\overline{BSY}$ goes high (See Figure 20-3 on page 175 for signal waveforms).

I. Repeat B through H until the entire flash is programmed or until all data has been programmed.

J. End page programming

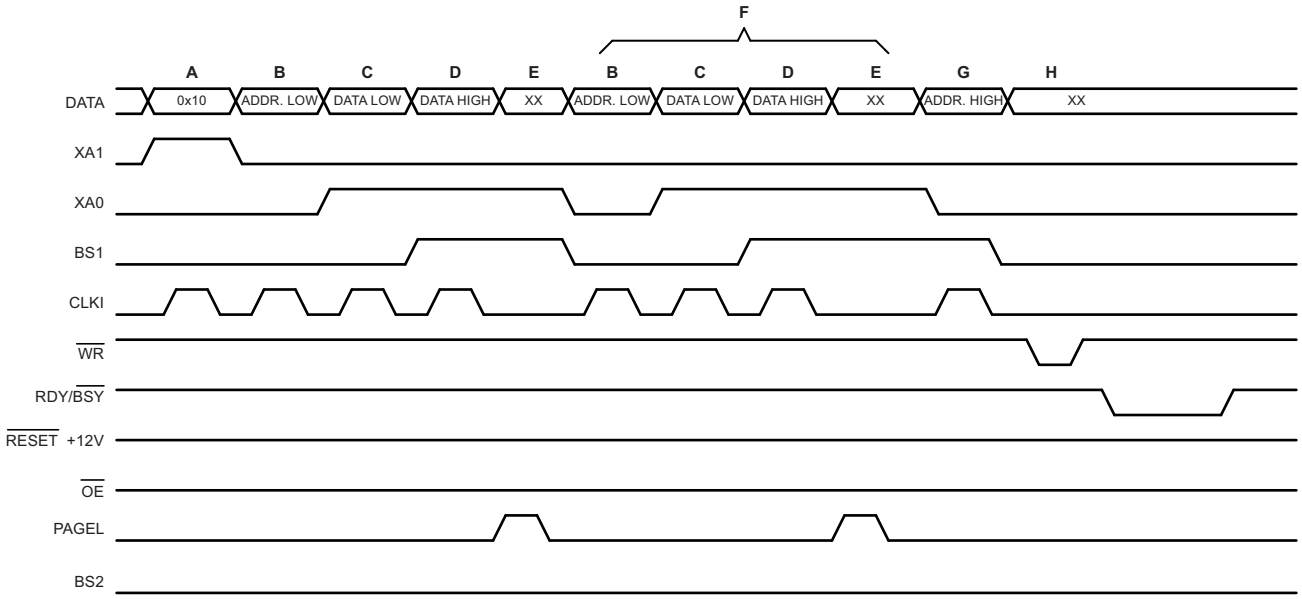
1. 1. Set XA1, XA0 to “10”. This enables command loading.
2. Set DATA to “0000 0000”. This is the command for No operation.
3. Give CLKI a positive pulse. This loads the command, and the internal write signals are reset.

Figure 20-2. Addressing the Flash Which is Organized in Pages(1)



Note: 1. PCPAGE and PCWORD are listed in Table 20-7 on page 170.

Figure 20-3. Programming the Flash Waveforms(1)



Note: 1. “XX” is don’t care. The letters refer to the programming description above.

20.7.5 Programming the EEPROM

The EEPROM is organized in pages, see Table 20-8 on page 170. When programming the EEPROM, the program data is latched into a page buffer. This allows one page of data to be programmed simultaneously. The programming algorithm for the EEPROM data memory is as follows (refer to Section 20.7.4 “Programming the Flash” on page 173 for details on command, address and data loading):

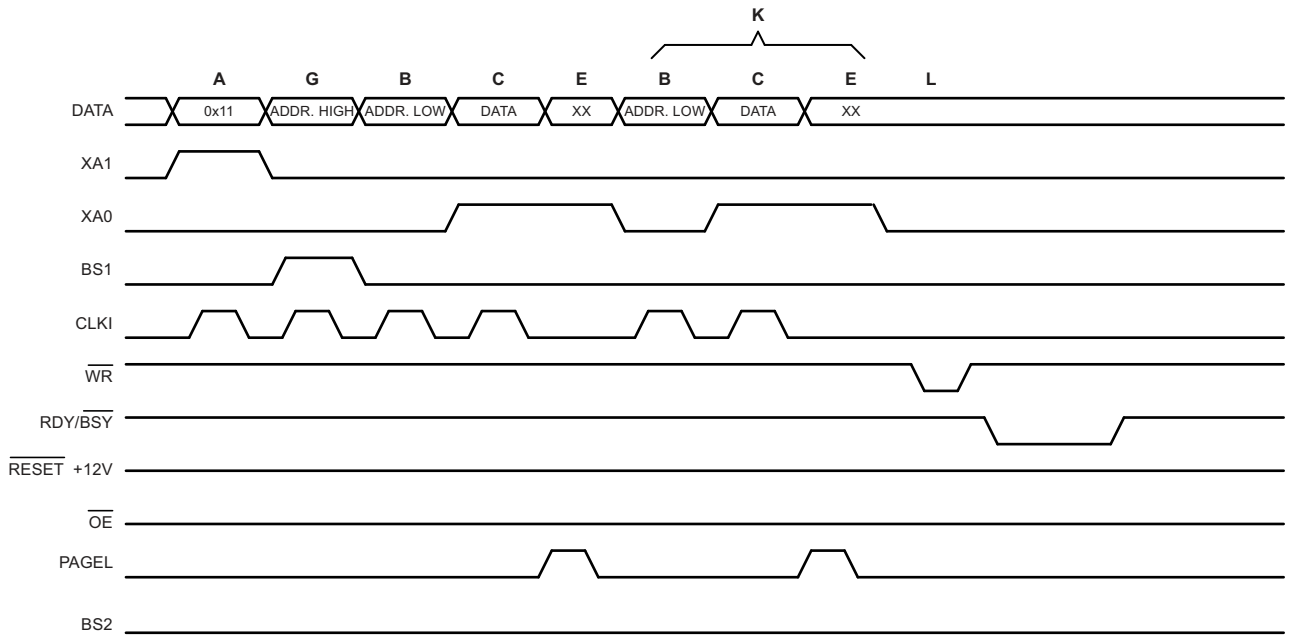
1. A: Load Command “0001 0001”.
2. G: Load address high byte (0x00 – 0xFF).
3. B: Load address low byte (0x00 – 0xFF).
4. C: Load data (0x00 – 0xFF).
5. E: Latch data (give PAGEL a positive pulse).

K: Repeat 3 through 5 until the entire buffer is filled.

L: Program EEPROM page

1. Set BS1 to “0”.
2. Give $\overline{WR}$ a negative pulse. This starts programming of the EEPROM page. RDY/$\overline{BSY}$ goes low.
3. Wait until to RDY/$\overline{BSY}$ goes high before programming the next page (See Figure 20-4 on page 176 for signal waveforms).

Figure 20-4. Programming the EEPROM Waveforms



20.7.6 Reading the Flash

The algorithm for reading the flash memory is as follows (refer to Section 20.7.4 “Programming the Flash” on page 173 for details on command and address loading):

1. A: Load command “0000 0010”.
2. G: Load address high byte (0x00 – 0xFF).
3. B: Load address low byte (0x00 – 0xFF).
4. Set $\overline{OE}$ to “0”, and BS1 to “0”. The flash word low byte can now be read at DATA.
5. Set BS1 to “1”. The flash word high byte can now be read at DATA.
6. Set $\overline{OE}$ to “1”.

20.7.7 Reading the EEPROM

The algorithm for reading the EEPROM memory is as follows (refer to Section 20.7.4 “Programming the Flash” on page 173 for details on command and address loading):

1. A: Load command “0000 0011”.
2. G: Load address high byte (0x00 – 0xFF).
3. B: Load address low byte (0x00 – 0xFF).
4. Set $\overline{OE}$ to “0”, and BS1 to “0”. The EEPROM data byte can now be read at DATA.
5. Set $\overline{OE}$ to “1”.

20.7.8 Programming the Fuse Low Bits

The algorithm for programming the fuse low bits is as follows (refer to Section 20.7.4 “Programming the Flash” on page 173 for details on command and data loading):

1. A: Load command “0100 0000”.
2. C: Load data low byte. Bit n = “0” programs and bit n = “1” erases the fuse bit.
3. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.

20.7.9 Programming the Fuse High Bits

The algorithm for programming the fuse high bits is as follows (refer to Section 20.7.4 “Programming the Flash” on page 173 for details on command and data loading):

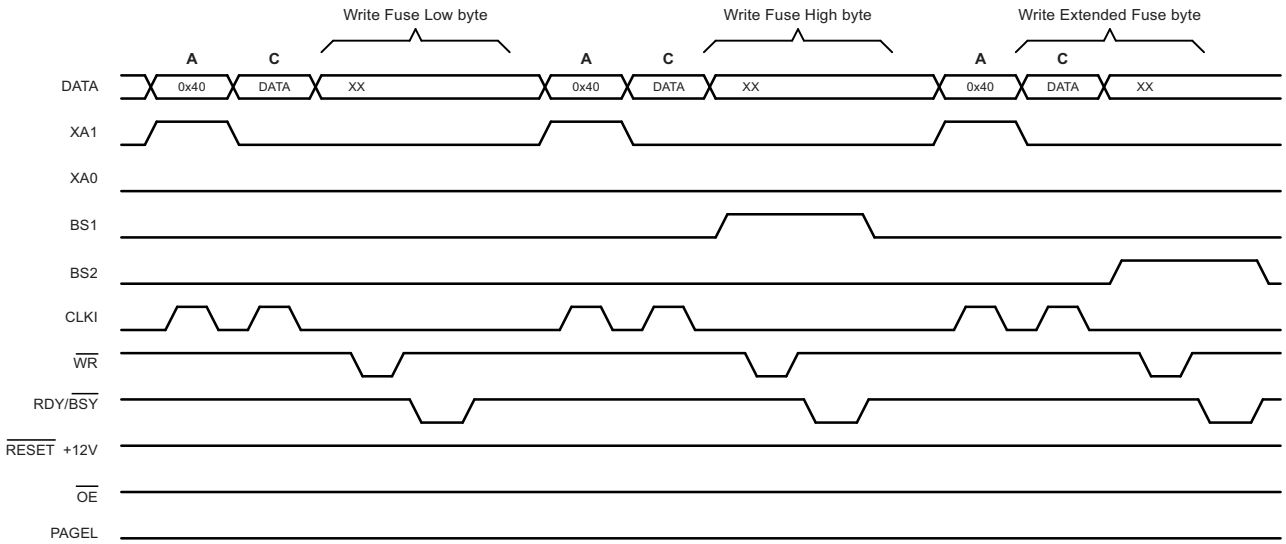
1. A: Load command “0100 0000”.
2. C: Load data low byte. Bit n = “0” programs and bit n = “1” erases the fuse bit.
3. Set BS1 to “1” and BS2 to “0”. This selects high data byte.
4. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.
5. Set BS1 to “0”. This selects low data byte.

20.7.10 Programming the Fuse Extended Bits

The algorithm for programming the fuse extended bits is as follows (refer to Section 20.7.4 “Programming the Flash” on page 173 for details on command and data loading):

1. 1. A: Load command “0100 0000”.
2. 2. C: Load data low byte. Bit n = “0” programs and bit n = “1” erases the fuse bit.
3. 3. Set BS1 to “0” and BS2 to “1”. This selects extended data byte.
4. 4. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.
5. 5. Set BS2 to “0”. This selects low data byte.

Figure 20-5. Programming the Fuses Waveforms



20.7.11 Programming the Lock Bits

The algorithm for programming the lock bits is as follows (refer to Section 20.7.4 “Programming the Flash” on page 173 for details on command and data loading):

1. A: Load command “0010 0000”.
2. C: Load data low byte. Bit n = “0” programs the lock bit. If LB mode 3 is programmed (LB1 and LB2 is programmed), it is not possible to program the lock bits by any external programming mode.
3. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.

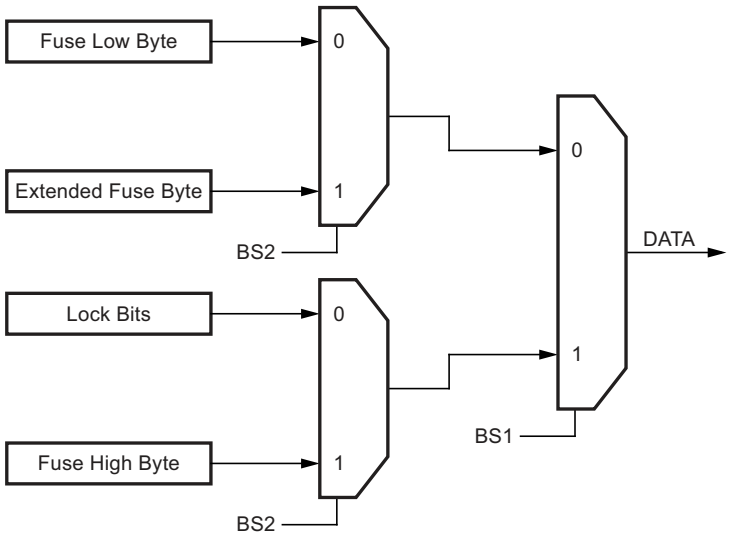
The lock bits can only be cleared by executing chip erase.

20.7.12 Reading the Fuse and Lock Bits

The algorithm for reading the fuse and lock bits is as follows (refer to Section 20.7.4 “Programming the Flash” on page 173 for details on command loading):

1. A: Load command “0000 0100”.
2. Set $\overline{OE}$ to “0”, BS2 to “0” and BS1 to “0”. The status of the fuse low bits can now be read at DATA (“0” means programmed).
3. Set $\overline{OE}$ to “0”, BS2 to “1” and BS1 to “1”. The status of the fuse high bits can now be read at DATA (“0” means programmed).
4. Set OE to “0”, BS2 to “1”, and BS1 to “0”. The status of the fuse extended bits can now be read at DATA (“0” means programmed).
5. Set $\overline{OE}$ to “0”, BS2 to “0” and BS1 to “1”. The status of the lock bits can now be read at DATA (“0” means programmed).
6. Set $\overline{OE}$ to “1”.

Figure 20-6. Mapping Between BS1, BS2 and the Fuse and Lock Bits During Read



20.7.13 Reading the Signature Bytes

The algorithm for reading the signature bytes is as follows (refer to Section 20.7.4 “Programming the Flash” on page 173 for details on command and address loading):

1. A: Load command “0000 1000”.
2. B: Load address low byte (0x00 – 0x02).
3. Set $\overline{OE}$ to “0”, and BS1 to “0”. The selected signature byte can now be read at DATA.
4. Set $\overline{OE}$ to “1”.

20.7.14 Reading the Calibration Byte

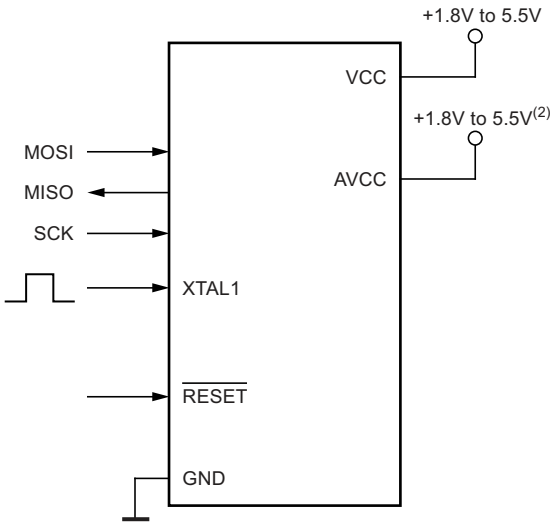
The algorithm for reading the calibration byte is as follows (refer to Section 20.7.4 “Programming the Flash” on page 173 for details on command and address loading):

1. A: Load command “0000 1000”.
2. B: Load address low byte, 0x00.
3. Set $\overline{OE}$ to “0”, and BS1 to “1”. The calibration byte can now be read at DATA.
4. Set $\overline{OE}$ to “1”.

20.8 Serial Downloading

Both the flash and EEPROM memory arrays can be programmed using the serial SPI bus while $\overline{\text{RESET}}$ is pulled to GND. The serial interface consists of pins SCK, MOSI (input) and MISO (output). After $\overline{\text{RESET}}$ is set low, the programming enable instruction needs to be executed first before program/erase operations can be executed. NOTE, in Table 20-13, the pin mapping for SPI programming is listed. Not all parts use the SPI pins dedicated for the internal SPI interface.

Figure 20-7. Serial Programming and Verify(1)



Notes: 1. If the device is clocked by the internal oscillator, it is no need to connect a clock source to the CLKI pin.

2. $V_{CC} - 0.3V < AV_{CC} < V_{CC} + 0.3V$, however, AV_{CC} should always be within 2.7 – 5.5V

When programming the EEPROM, an auto-erase cycle is built into the self-timed programming operation (in the serial mode ONLY) and there is no need to first execute the chip erase instruction. The chip erase operation turns the content of every memory location in both the program and EEPROM arrays into 0xFF.

Depending on CKSEL fuses, a valid clock must be present. The minimum low and high periods for the serial clock (SCK) input are defined as follows:

Low: > 2 CPU clock cycles for f_{ck} < 16MHz, 3 CPU clock cycles for $f_{ck} \geq$ 16MHz

High: > 2 CPU clock cycles for f_{ck} < 16MHz, 3 CPU clock cycles for $f_{ck} \geq$ 16MHz

20.8.1 Serial Programming Pin Mapping

Table 20-13. Pin Mapping Serial Programming

Symbol	Pins	I/O	Description
MOSI	PB3	I	Serial data in
MISO	PB4	O	Serial data out
SCK	PB5	I	Serial clock

20.8.2 Serial Programming Algorithm

When writing serial data to the Atmel® ATtiny88, data is clocked on the rising edge of SCK.

When reading data from the Atmel ATtiny88, data is clocked on the falling edge of SCK. See Figure 21-9 on page 194 and Figure 21-10 on page 194 for timing details.

To program and verify the Atmel ATtiny88 in the serial programming mode, the following sequence is recommended (see serial programming instruction set in Table 20-15 on page 181):

1. Power-up sequence:
   Apply power between V_{CC} and GND while $\overline{\text{RESET}}$ and SCK are set to “0”. In some systems, the programmer can not guarantee that SCK is held low during power-up. In this case, $\overline{\text{RESET}}$ must be given a positive pulse of at least two CPU clock cycles duration after SCK has been set to “0”.
2. Wait for at least 20 ms and enable serial programming by sending the programming enable serial instruction to pin MOSI.
3. The serial programming instructions will not work if the communication is out of synchronization. When in sync. the second byte (0x53), will echo back when issuing the third byte of the programming enable instruction. Whether the echo is correct or not, all four bytes of the instruction must be transmitted. If the 0x53 did not echo back, give $\overline{\text{RESET}}$ a positive pulse and issue a new programming enable command.
4. The flash is programmed one page at a time. The memory page is loaded one byte at a time by supplying the 6 LSB of the address and data together with the load program memory page instruction. To ensure correct loading of the page, the data low byte must be loaded before data high byte is applied for a given address. The program memory page is stored by loading the write program memory page instruction with the 7 MSB of the address. If polling (RDY/$\overline{\text{BSY}}$) is not used, the user must wait at least t_{WD_FLASH} before issuing the next page (seeTable 20-14). Accessing the serial programming interface before the flash write operation completes can result in incorrect programming.
5. **A**: The EEPROM array is programmed one byte at a time by supplying the address and data together with the appropriate Write instruction. An EEPROM memory location is first automatically erased before new data is written. If polling (RDY/$\overline{\text{BSY}}$) is not used, the user must wait at least t_{WD_EEPROM} before issuing the next byte (see Table 20-14). In a chip erased device, no 0xFFs in the data file(s) need to be programmed.
   B: The EEPROM array is programmed one page at a time. The memory page is loaded one byte at a time by supplying the 6 LSB of the address and data together with the load EEPROM memory page instruction. The EEPROM memory page is stored by loading the write EEPROM memory page instruction with the 7 MSB of the address. When using EEPROM page access only byte locations loaded with the load EEPROM memory page instruction is altered. The remaining locations remain unchanged. If polling (RDY/$\overline{\text{BSY}}$) is not used, the used must wait at least t_{WD_EEPROM} before issuing the next byte (see Table 20-14). In a chip erased device, no 0xFF in the data file(s) need to be programmed.
6. Any memory location can be verified by using the read instruction which returns the content at the selected address at serial output MISO.
7. At the end of the programming session, $\overline{\text{RESET}}$ can be set high to commence normal operation.
8. Power-off sequence (if needed):
   Set $\overline{\text{RESET}}$ to “1”.
   Turn V_{CC} power off.

Table 20-14. Typical Wait Delay Before Writing the Next Flash or EEPROM Location

Symbol	Minimum Wait Delay
t_{WD_FLASH}	4.5ms
t_{WD_EEPROM}	3.6ms
t_{WD_ERASE}	9.0ms

20.8.3 Serial Programming Instruction set

Table 20-15 and Figure 20-8 on page 182 describes the Instruction set.

Table 20-15. Serial Programming Instruction Set (Hexadecimal values)

Instruction/Operation	Instruction Format			
	Byte 1	Byte 2	Byte 3	Byte4
Programming enable	$AC	$53	$00	$00
Chip erase (program memory/EEPROM)	$AC	$80	$00	$00
Poll RDY/$\overline{\text{BSY}}$	$F0	$00	$00	data byte out
Load instructions				
Load extended address byte(1)	$4D	$00	Extended adr	$00
Load program memory page, high byte	$48	$00	adr LSB	high data byte in
Load program memory page, low byte	$40	$00	adr LSB	low data byte in
Load EEPROM memory page (page access)	$C1	$00	0000 000aa	data byte in
Read instructions				
Read program memory, high byte	$28	adr MSB	adr LSB	high data byte out
Read program memory, low byte	$20	adr MSB	adr LSB	low data byte out
Read EEPROM memory	$A0	0000 00aa	aaaa aaaa	data byte out
Read lock bits	$58	$00	$00	data byte out
Read signature byte	$30	$00	0000 000aa	data byte out
Read fuse bits	$50	$00	$00	data byte out
Read fuse high bits	$58	$08	$00	data byte out
Read fuse extended bits	$50	$08	$00	data byte out
Read calibration byte	$38	$00	$00	data byte out
Write instructions				
Write program memory page	$4C	adr MSB	adr LSB	$00
Write EEPROM memory	$C0	0000 00aa	aaaa aaaa	data byte in
Write EEPROM memory page (page access)	$C2	0000 00aa	aaaa aa00	$00
Write lock bits	$AC	$E0	$00	data byte in
Write fuse bits	$AC	$A0	$00	data byte in
Write fuse high bits	$AC	$A8	$00	data byte in
Write fuse extended bits	$AC	$A4	$00	data byte in

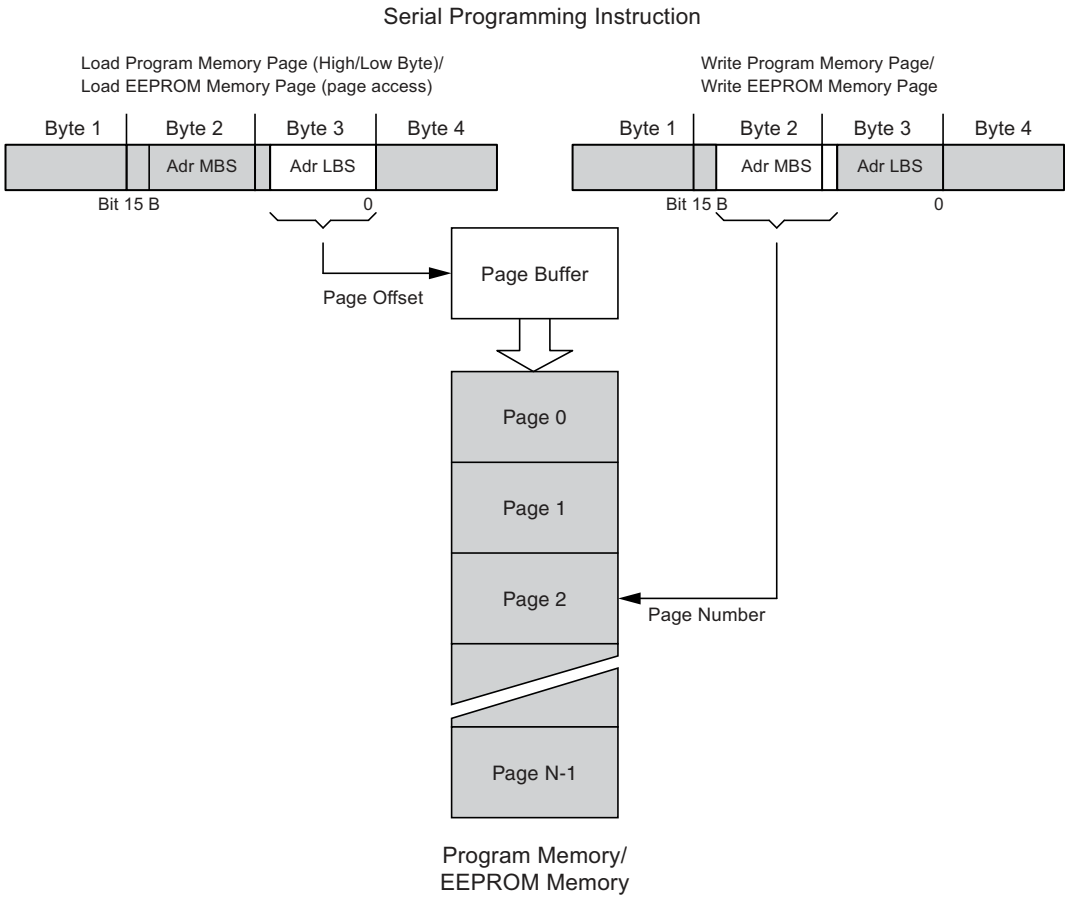
Notes:
1. Not all instructions are applicable for all parts.
2. a = address.
3. Bits are programmed ‘0’, unprogrammed ‘1’.
4. To ensure future compatibility, unused fuses and lock bits should be unprogrammed (‘1’).
5. Refer to the corresponding section for fuse and lock bits, calibration and signature bytes and page size.
6. See htt://www.atmel.com/avr for application notes regarding programming and programmers.

If the LSB in RDY/BSY data byte out is ‘1’, a programming operation is still pending. Wait until this bit returns ‘0’ before the next instruction is carried out.

Within the same page, the low data byte must be loaded prior to the high data byte.

After data is loaded to the page buffer, program the EEPROM page, see Figure 20-8.

Figure 20-8. Serial Programming Instruction Example



21. Electrical Characteristics

21.1 Absolute Maximum Ratings

Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Parameters	Min.	Max.	Unit
Operating temperature	–55	+125	°C
Storage temperature	–65	+150	°C
Voltage on any pin except $\overline{\text{RESET}}$ with respect to ground	–0.5	V_{CC} + 0.5	V
Voltage on $\overline{\text{RESET}}$ with respect to ground	–0.5	+13.0	V
Maximum operating voltage		6.0	V
DC current per I/O pin DC current V_{CC} and GND pins		40 200	mA

21.2 DC Characteristics

T_A = –40°C to +125°C, V_{CC} = 2.7V to 5.5V (unless otherwise noted)(1)

Parameter	Condition	Symbol	Min	Typ	Max	Unit
Input low voltage, except $\overline{\text{RESET}}$ pin	V_{CC} = 2.7V to 5.5V	V_{IL}	–0.5		0.3 V_{CC}	V
Input low voltage, $\overline{\text{RESET}}$ pin as reset	V_{CC} = 2.7V to 5.5V	V_{IL}	–0.5		0.2 V_{CC}	V
Input high voltage, except $\overline{\text{RESET}}$ pin	V_{CC} = 2.7V to 5.5V	V_{IH}	0.6 V_{CC}		V_{CC} + 0.5	V
Input high voltage, $\overline{\text{RESET}}$ pin as reset	V_{CC} = 2.7V to 5.5V		0.9 V_{CC}		V_{CC} + 0.5	V
Output low voltage(2), except high sink I/O pins	I_{OL} = 10mA, V_{CC} = 5V	V_{OL}			0.7	V
	I_{OL} = 5mA, V_{CC} = 3V				0.5	V
Output low voltage High sink I/O pins(3)	I_{OL} = 20mA, V_{CC} = 5V	V_{OL}			0.7	V
	I_{OL} = 10mA, V_{CC} = 3V				0.5	V

Notes:
1. All DC characteristics contained in this data sheet are based on actual silicon characterization of ATtiny88 AVR microcontrollers manufactured in corner run process technology.
2. Although each I/O port can sink more than the test conditions (10mA at V_{CC} = 5V, 5mA at V_{CC} = 3V, 2mA at V_{CC} = 2.7V) under steady state conditions (non-transient), the following must be observed:
   - The sum of all I_{OL}, for ports A0, A3, B0 – B7, C7, D5 – D7 should not exceed 100mA.
   - The sum of all I_{OL}, for ports A1 – A2, C0 – C6, D0 – D4 should not exceed 100mA.

   If I_{OL} exceeds the test condition, V_{OL} may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test condition.
3. High sink I/O pins are PD0, PD1, PD2 and PD3.
4. Although each I/O port can source more than the test conditions (10mA at V_{CC} = 5V, 5mA at V_{CC} = 3V, 2mA at V_{CC} = 2.7V) under steady state conditions (non-transient), the following must be observed:
   - The sum of all I_{OH}, for ports A2 – A3, B0 – B7, C6, D0 – D7 should not exceed 100mA.
   - The sum of all I_{OH}, for ports A0 – A1, C0 – C5, C7 should not exceed 100mA.

   If II_{OH} exceeds the test condition, V_{OH} may exceed the related specification. Pins are not guaranteed to source current greater than the listed test condition.
5. Measured with all I/O modules turned off (PRR = 0xFF).
6. Measured with brown-out detection (BOD) disabled.

21.2 DC Characteristics (Continued)

T_A = –40°C to +125°C, V_{CC} = 2.7V to 5.5V (unless otherwise noted)[1]

Parameter	Condition	Symbol	Min	Typ	Max	Unit
Output high voltage[4], except high sink I/O pins	I_{OH} = –15 mA, V_{CC} = 5V	V_{OH}	4.1			V
	I_{OH} = –5 mA, V_{CC} = 3V		2.3			V
Output high voltage High sink I/O pins[3]	I_{OH} = –15 mA, V_{CC} = 5V	V_{OH}	4.1			V
	I_{OH} = –5 mA, V_{CC} = 3V		2.3			V
Input leakage current I/O pin	V_{CC} = 5.5V, pin low (absolute value)	I_{LIL}			1	µA
Input leakage current I/O pin	V_{CC} = 5.5V, pin high (absolute value)	I_{LIH}			1	µA
Pull-up resistor, I/O pin	V_{CC} = 5.5V, input low	R_{PU}	20		50	kΩ
Pull-up resistor, reset pin	V_{CC} = 5.5V, input low		30		60	kΩ
Supply current, active mode[5]	8MHz, V_{CC} = 3.0V	I_{CC}		1.2	4	mA
	8MHz, V_{CC} = 5.0V			4.4	8	mA
	16MHz, V_{CC} = 5.0V			8	12	mA
Supply current, idle mode[5]	8MHz, V_{CC} = 3.0V			0.5	0.7	mA
	8MHz, V_{CC} = 5.0V			1.0	1.5	mA
	16MHz, V_{CC} = 5.0V			2.0	3.0	mA
Supply current, power-down mode[6]	WDT enabled, V_{CC} = 3V			6	44	µA
	WDT enabled, V_{CC} = 5V			12	66	µA
	WDT disabled, V_{CC} = 3V			4	20	µA
	WDT disabled, V_{CC} = 5V			6	30	µA
Analog comparator input offset voltage (absolute value)	0.4V < Vin < V_{CC} – 0.5	Vacio		10	40	mV
Analog comparator input leakage current	V_{CC} = 5V Vin = V_{CC}/2	Iaclk	–50		+50	nA

Notes:
1. All DC characteristics contained in this data sheet are based on actual silicon characterization of ATtiny88 AVR microcontrollers manufactured in corner run process technology.
2. Although each I/O port can sink more than the test conditions (10mA at V_{CC} = 5V, 5mA at V_{CC} = 3V, 2mA at V_{CC} = 2.7V) under steady state conditions (non-transient), the following must be observed:
   - The sum of all I_{OL}, for ports A0, A3, B0 – B7, C7, D5 – D7 should not exceed 100mA.
   - The sum of all I_{OL}, for ports A1 – A2, C0 – C6, D0 – D4 should not exceed 100mA.

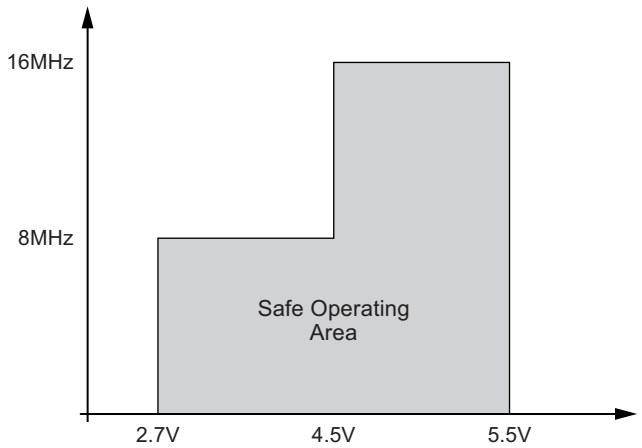
   If I_{OL} exceeds the test condition, V_{OL} may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test condition.
3. High sink I/O pins are PD0, PD1, PD2 and PD3.
4. Although each I/O port can source more than the test conditions (10mA at V_{CC} = 5V, 5mA at V_{CC} = 3V, 2mA at V_{CC} = 2.7V) under steady state conditions (non-transient), the following must be observed:
   - The sum of all I_{OH}, for ports A2 – A3, B0 – B7, C6, D0 – D7 should not exceed 100mA.
   - The sum of all I_{OH}, for ports A0 – A1, C0 – C5, C7 should not exceed 100mA.

   If I_{OH} exceeds the test condition, V_{OH} may exceed the related specification. Pins are not guaranteed to source current greater than the listed test condition.
5. Measured with all I/O modules turned off (PRR = 0xFF).
6. Measured with brown-out detection (BOD) disabled.

21.3 Speed Grades

Maximum frequency is dependent on $V_{CC.}$ as shown below.

Figure 21-1. , Maximum Frequency versus V_{CC}



21.4 Clock Characterizations

21.4.1 Calibrated Internal Oscillator Accuracy

It is possible to manually calibrate the internal oscillator to be more accurate than default factory calibration. Please note that the oscillator frequency depends on temperature and voltage.

Table 21-1. Calibration Accuracy of Internal Oscillator

Calibration Method	Target Frequency	V_{CC}	Temperature	Accuracy at given Voltage and Temperature[1]
Factory Calibration	8.0MHz	3V	25°C	±2%
		2.7V to 5.5V	–40°C to +125°C	±14%

Notes: 1. Accuracy of oscillator frequency at calibration point (fixed temperature and fixed voltage).

21.4.2 Watchdog Oscillator Accuracy

Table 21-2. Accuracy of Watchdog Oscillator

Parameter	Condition	Symbol	Min	Typ.	Max	Unit
Watchdog oscillator frequency	V_{CC} = 2.7 to 5.5V	Fwdt	76	128	180	kHz

21.4.3 External Clock Drive

Figure 21-2. External Clock Drive Waveforms

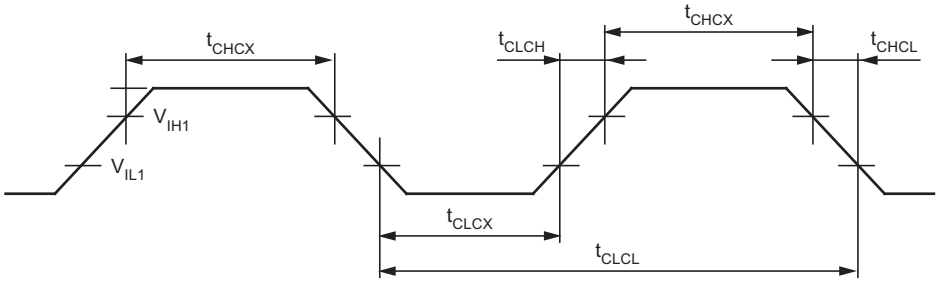


Table 21-3. External Clock Drive

Parameter	V_{CC} = 2.7V to 5.5V		V_{CC} = 4.5V to 5.5V		Unit
	Min.	Max.	Min.	Max.	
Oscillator frequency	0	8	0	16	MHz
Clock period	125		62.5		ns
High time	40		20		ns
Low time	40		20		ns
Rise time		1.6		0.5	µs
Fall time		1.6		0.5	µs
Change in period from one clock cycle to the next		2		2	%

21.5 System and Reset Characterizations

Table 21-4. Reset, Brown-out, and Internal Voltage Characteristics(1)

Parameter	Condition	Symbol	Min	Typ	Max	Unit
Power-on reset threshold voltage (rising)	T_A = –40 to +125°C	V_{POT}		1.5		V
Power-on reset threshold voltage (falling)(2)	T_A = –40 to +125°C	V_{POT}		1.2		V
$\overline{\text{RESET}}$ pin threshold voltage		V_{RST}	0.2 × V_{CC}		0.9 × V_{CC}	V
Vcc max start voltage to ensure internal power-on reset signal		Vpormax			0.4	V
Vcc min start voltage to ensure internal power-on reset signal		Vpormin	–0.1			V
Minimum pulse width on $\overline{\text{RESET}}$ pin	V_{CC} = 3V V_{CC} = 5V	t_{RST}	1100 600			ns
Brown-out detector hysteresis		V_{HYST}		80		mV
Internal bandgap reference voltage	V_{CC} = 5V	V_{BG}	1.0	1.1	1.2	V

Notes: 1. Values are guidelines, only
2. The Power-on Reset will not work unless the supply voltage has been below V_{POT} (falling)

Table 21-5. V_{BOT} versus BODLEVEL Fuse Coding(1)

BODLEVEL [2..0] Fuses	Min	Typ	Max	Unit
111	BOD disabled			
110	Reserved			V
101	2.4	2.7	2.9	
100	3.9	4.3	4.6	
0XX	Reserved			

Note: 1. V_{BOT} may be below nominal minimum operating voltage for some devices. For devices where this is the case, the device is tested down to $V_{CC} = V_{BOT}$ during the production test. This guarantees that a Brown-out Reset will occur before V_{CC} drops to a voltage where correct operation of the microcontroller is no longer guaranteed.

21.6 ADC Characteristics

Table 21-6. T_A = –40°C to +125°C, V_{CC} = 4V (Unless Otherwise Noted)

Parameter	Condition	Symbol	Min	Typ	Max	Unit
Resolution	–40°C to 125°C - 2.70 to 5.50V ADC clock = 200kHz			10		bits
Absolute accuracy	V_{CC} = 4.0v, V_{Ref} = AV_{CC} = V_{CC}	TUE		3.0	4.5	LSB
Integral non linearity	V_{CC} = 4.0v, V_{Ref} = AV_{CC} = V_{CC}	INL		0.6	1.5	LSB
Differential non linearity	V_{CC} = 4.0v, V_{Ref} = AV_{CC} = V_{CC}	DNL		0.3	1.0	LSB
Gain error	V_{CC} = 4.0v, V_{Ref} = AV_{CC} = V_{CC}		–6.0	–3.0	0.0	LSB
Offset error	V_{CC} = 4.0v, V_{Ref} = AV_{CC} = V_{CC}		0.0	3.0	5.0	LSB
Clock frequency			50		200	kHz
Analog supply voltage		AV_{CC}	V_{CC} – 0.3		V_{CC} + 0.3	V
Input voltage		V_{in}	GND		V_{Ref}	V

21.7 2-wire Serial Interface Characteristics

Table 21-7 describes the requirements for devices connected to the 2-wire serial bus. The ATtiny88 2-wire serial interface meets or exceeds these requirements under the noted conditions.

Timing symbols refer to Figure 21-3.

Table 21-7. 2-wire Serial Bus Requirements

Parameter	Condition	Symbol	Min	Max	Unit
Input low-voltage		V_{IL}	–0.5	$0.3 \times V_{CC}$	V
Input high-voltage		V_{IH}	$0.7 \times V_{CC}$	$V_{CC} + 0.5$	V
Hysteresis of Schmitt trigger inputs		V_{hys}(1)	$0.05 \times V_{CC}$(2)	–	V
Output low-voltage	3mA sink current	V_{OL}(1)	0	0.4	V
Rise time for both SDA and SCL		t_r(1)	$20 + 0.1C_b$(3)(2)	300	ns
Output fall time from V_{IHmin} to V_{ILmax}	10pF < C_b < 400pF(3)	t_{of}(1)	$20 + 0.1C_b$(3)(2)	250	ns
Spikes suppressed by input filter		t_{SP}(1)	0	50(2)	ns
Input current each I/O pin	$0.1V_{CC} < V_i < 0.9V_{CC}$	I_i	–10	+10	µA
Capacitance for each I/O pin		C_i(1)	–	10	pF
SCL clock frequency	f_{CK}(4) > max(16f_{SCL}, 250kHz)(5)	f_{SCL}	0	400	kHz
Value of pull-up resistor	$f_{SCL} \le 100kHz$	Rp	$\frac{V_{CC} - 0,4V}{3mA}$	$\frac{1000ns}{C_b}$	Ω
	$f_{SCL} > 100kHz$		$\frac{V_{CC} - 0,4V}{3mA}$	$\frac{300ns}{C_b}$	Ω
Hold time (repeated) START condition	$f_{SCL} \le 100kHz$	$t_{HD;STA}$	4.0	–	µs
	$f_{SCL} > 100kHz$		0.6	–	µs
Low period of the SCL clock	$f_{SCL} \le 100kHz$(6)	t_{LOW}	4.7	–	µs
	$f_{SCL} > 100kHz$(7)		1.3	–	µs
High period of the SCL clock	$f_{SCL} \le 100kHz$	t_{HIGH}	4.0	–	µs
	$f_{SCL} > 100kHz$		0.6	–	µs
Set-up time for a repeated START condition	$f_{SCL} \le 100kHz$	$t_{SU;STA}$	4.7	–	µs
	$f_{SCL} > 100kHz$		0.6	–	µs
Data hold time	$f_{SCL} \le 100kHz$	$t_{HD;DAT}$	0	3.45	µs
	$f_{SCL} > 100kHz$		0	0.9	µs
Data setup time	$f_{SCL} \le 100kHz$	$t_{SU;DAT}$	250	–	ns
	$f_{SCL} > 100kHz$		100	–	ns

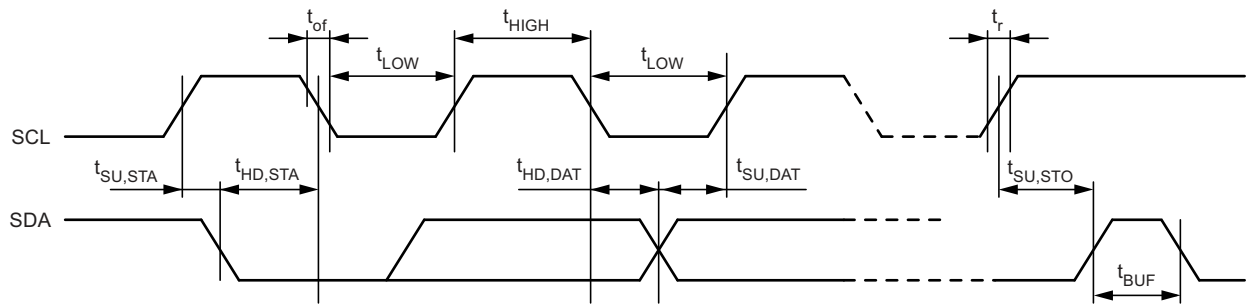
Notes:
1. This parameter is characterized, only, and not fully tested.
2. Required only for f_{SCL} > 100kHz.
3. C_b = capacitance of one bus line in pF.
4. f_{CK} = CPU clock frequency
5. This requirement applies to all 2-wire serial interface operation in ATtiny88. Other devices connected to the 2-wire serial bus need only obey the general f_{SCL} requirement.
6. The actual low period generated by the 2-wire serial interface of ATtiny88 is ($1/f_{SCL} - 2/f_{CK}$), thus f_{CK} must be greater than 6MHz for the low time requirement to be strictly met at f_{SCL} = 100kHz.
7. The actual low period generated by the 2-wire serial interface of ATtiny88 is ($1/f_{SCL} - 2/f_{CK}$), thus the low time requirement will not be strictly met for f_{SCL} > 308kHz when f_{CK} = 8MHz. Still, ATtiny88 devices connected to the bus may communicate at full speed (400kHz) with other ATtiny88 devices, as well as any other device with a proper t_{LOW} acceptance margin.

Table 21-7. 2-wire Serial Bus Requirements (Continued)

Parameter	Condition	Symbol	Min	Max	Unit
Setup time for STOP condition	$f_{SCL} \leq 100kHz$	$t_{SU;STO}$	4.0	–	µs
	$f_{SCL} > 100kHz$		0.6	–	µs
Bus free time between a STOP and START condition	$f_{SCL} \leq 100kHz$	t_{BUF}	4.7	–	µs
	$f_{SCL} > 100kHz$		1.3	–	µs

Notes:
1. This parameter is characterized, only, and not fully tested.
2. Required only for $f_{SCL} > 100kHz$.
3. C_b = capacitance of one bus line in pF.
4. f_{CK} = CPU clock frequency
5. This requirement applies to all 2-wire serial interface operation in ATtiny88. Other devices connected to the 2-wire serial bus need only obey the general f_{SCL} requirement.
6. The actual low period generated by the 2-wire serial interface of ATtiny88 is $(1/f_{SCL} – 2/f_{CK})$, thus f_{CK} must be greater than 6MHz for the low time requirement to be strictly met at f_{SCL} = 100kHz.
7. The actual low period generated by the 2-wire serial interface of ATtiny88 is $(1/f_{SCL} – 2/f_{CK})$, thus the low time requirement will not be strictly met for $f_{SCL} > 308kHz$ when f_{CK} = 8MHz. Still, ATtiny88 devices connected to the bus may communicate at full speed (400kHz) with other ATtiny88 devices, as well as any other device with a proper t_{LOW} acceptance margin.

Figure 21-3. 2-wire Serial Bus Timing



21.8 SPI Characteristics

See Figure 21-4 and Figure 21-5 for details.

Table 21-8. SPI Timing Parameters

No.	Description	Mode	Min	Typ	Max	Unit
1	SCK period	Master		See Table 14-5		ns
2	SCK high/low	Master		50% duty cycle		
3	Rise/fall time	Master		3.6		
4	Setup	Master		10		
5	Hold	Master		10		
6	Out to SCK	Master		$0.5 \times t_{sck}$		
7	SCK to out	Master		10		
8	SCK to out high	Master		10		
9	$\overline{SS}$ low to out	Slave		15		
10	SCK period	Slave	$4 \times t_{ck}$			
11	SCK high/low(1)	Slave	$2 \times t_{ck}$			
12	Rise/fall time	Slave			1600	
13	Setup	Slave	10			
14	Hold	Slave	t_{ck}			
15	SCK to out	Slave		15		
16	SCK to $\overline{SS}$ high	Slave	20			
17	$\overline{SS}$ high to tri-state	Slave		10		
18	$\overline{SS}$ low to SCK	Slave	20			

Notes: 1. In SPI Programming mode the minimum SCK high/low period is:
- 2 t_{CLCL} for f_{CK} < 12MHz
- 3 t_{CLCL} for f_{CK} > 12MHz

2. All DC characteristics contained in this datasheet are based on simulation and characterization of other AVR microcontrollers manufactured in the same process technology. These values are preliminary values representing design targets, and will be updated after characterization of actual silicon.

Figure 21-4. SPI Interface Timing Requirements (Master Mode)

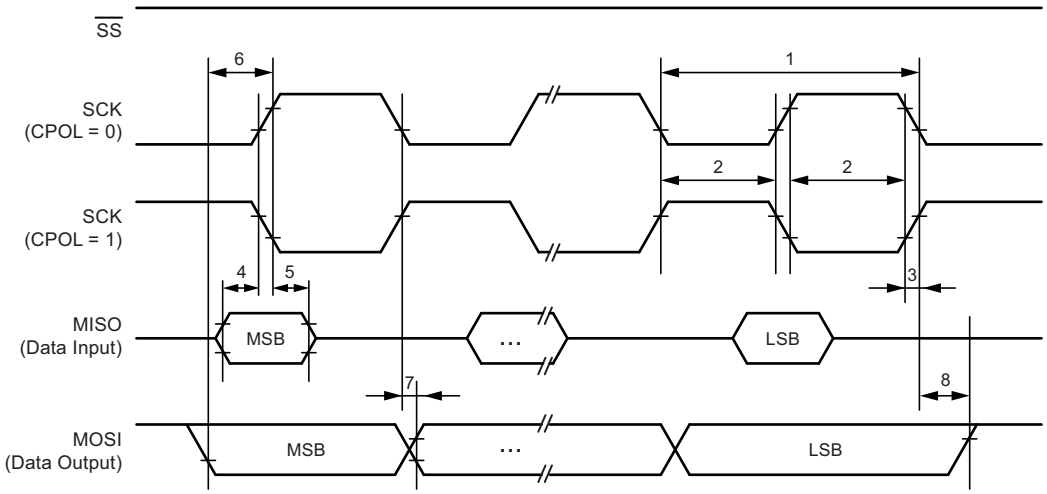
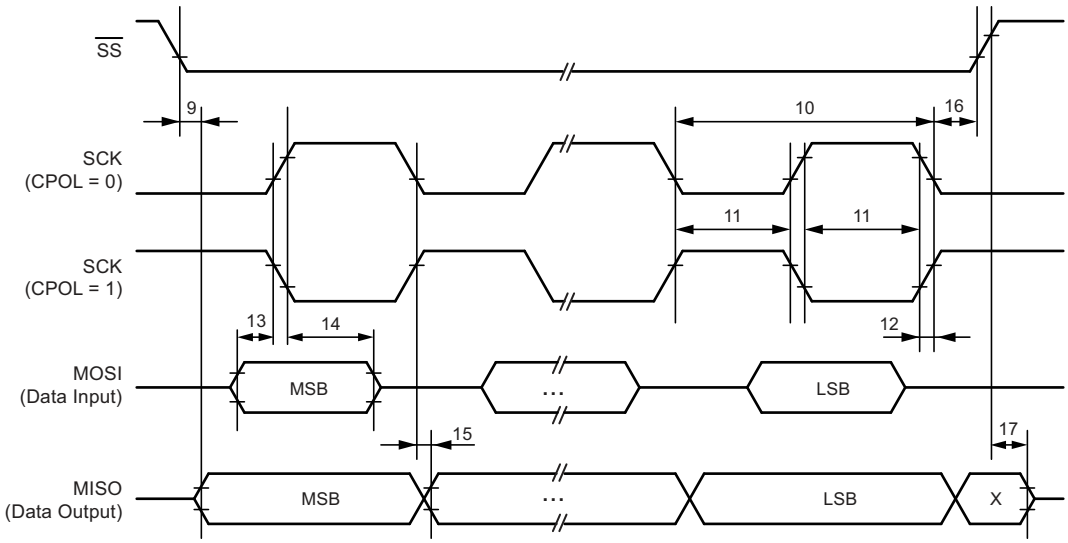


Figure 21-5. SPI Interface Timing Requirements (Slave Mode)



21.9 Parallel Programming Characteristics

Figure 21-6. Parallel Programming Timing, Including some General Timing Requirements

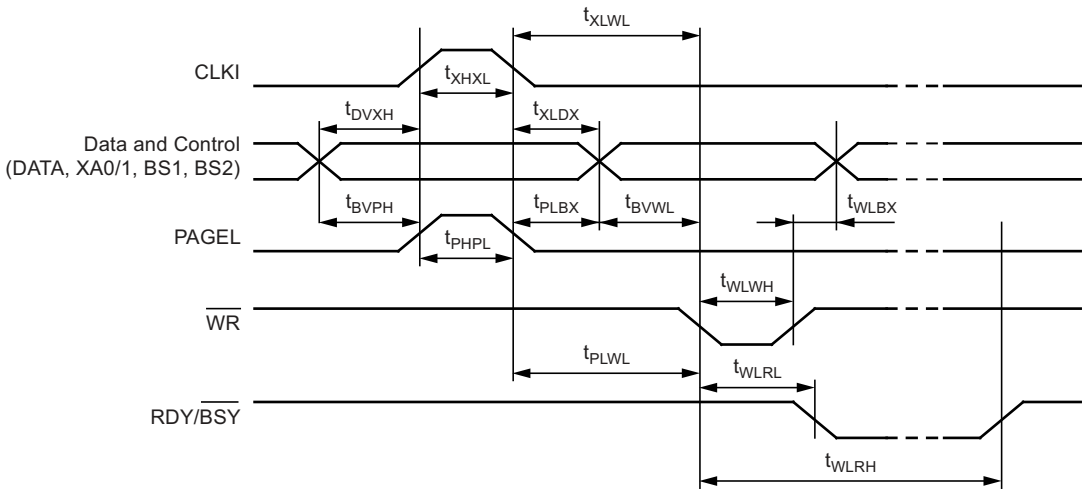
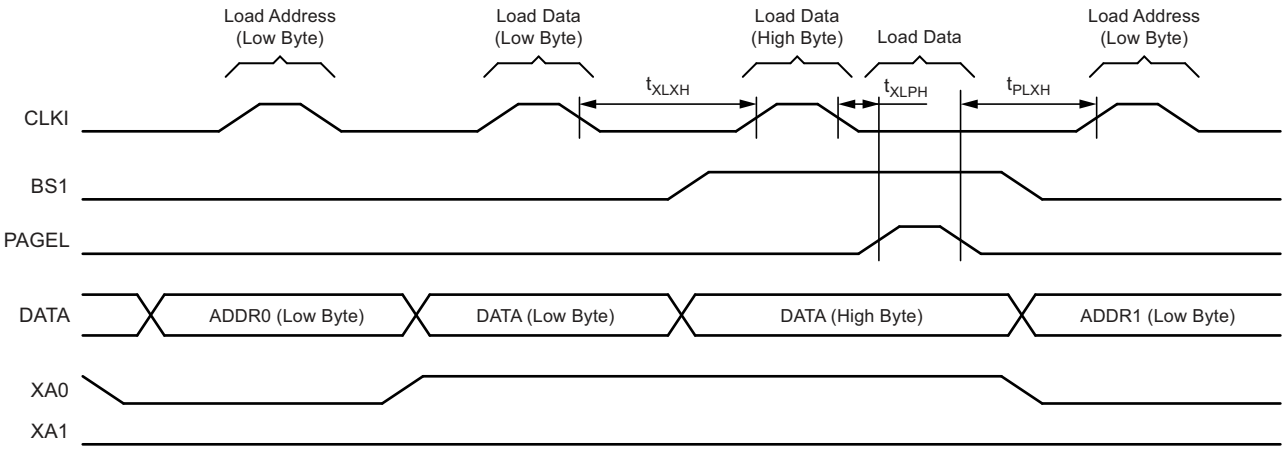
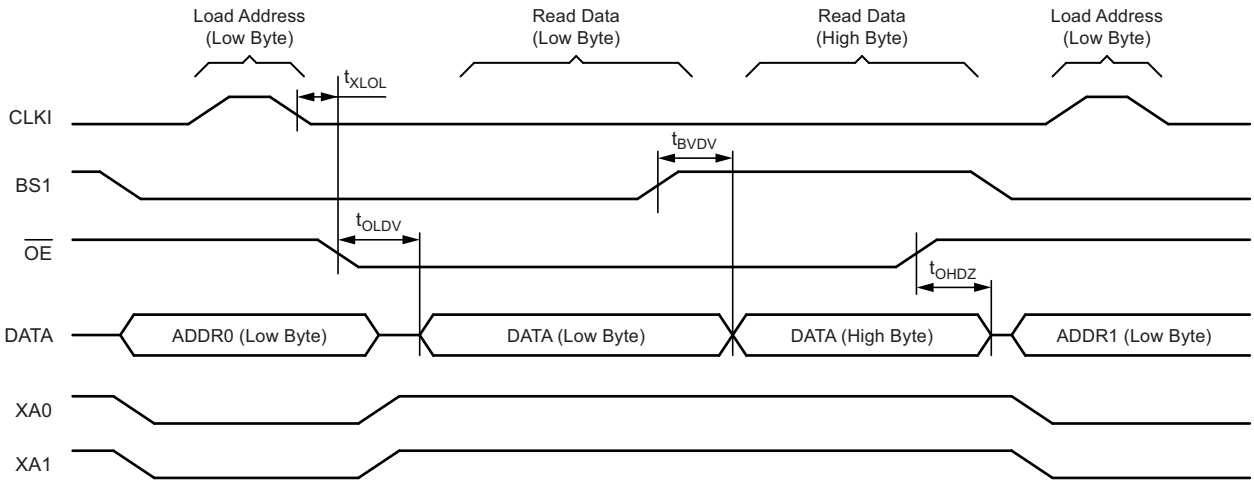


Figure 21-7. Parallel Programming Timing, Loading Sequence with Timing Requirements(1)



Note: 1. The timing requirements shown in Figure 21-6 (i.e., t_{DVXH}, t_{XHXL}, and t_{XLDX}) also apply to loading operation.

Figure 21-8. Parallel Programming Timing, Reading Sequence (within the Same Page) with Timing Requirements(1)



Note: 1. The timing requirements shown in Figure 21-6 (i.e., t_{DVXH}, t_{XHXL}, and t_{XLDX}) also apply to reading operation.

Table 21-9. Parallel Programming Characteristics, T_A = 25°C, V_{CC} = 5V

Parameter	Symbol	Min	Typ	Max	Unit
Programming enable voltage	V_{PP}	11.5		12.5	V
Programming enable current	I_{PP}			250	mA
Data and control valid before CLKI high	t_{DVXH}	67			ns
CLKI low to CLKI high	t_{XLXH}	200			ns
CLKI pulse width high	t_{XHXL}	150			ns
Data and control hold after CLKI low	t_{XLDX}	67			ns
CLKI low to $\overline{WR}$ low	t_{XLWL}	0			ns
CLKI low to PAGEL high	t_{XLPH}	0			ns
PAGEL low to CLKI high	t_{PLXH}	150			ns
BS1 valid before PAGEL high	t_{BVPH}	67			ns
PAGEL pulse width high	t_{PHPL}	150			ns
BS1 hold after PAGEL low	t_{PLBX}	67			ns
BS2/1 hold after $\overline{WR}$ low	t_{WLBX}	67			ns
PAGEL low to $\overline{WR}$ low	t_{PLWL}	67			ns
BS1 valid to $\overline{WR}$ low	t_{BVWL}	67			ns
$\overline{WR}$ pulse width low	t_{WLWH}	150			ns
$\overline{WR}$ low to RDY/$\overline{BSY}$ low	t_{WLRL}	0		1	ms
$\overline{WR}$ low to RDY/$\overline{BSY}$ high(1)	t_{WLRH}	3.7		4.5	ms
$\overline{WR}$ low to RDY/$\overline{BSY}$ high for chip erase(2)	t_{WLRH_CE}	7.5		9	ms
CLKI low to $\overline{OE}$ low	t_{XLOL}	0			ns
BS1 valid to DATA valid	t_{BVDV}	0		250	ns
$\overline{OE}$ low to DATA valid	t_{OLDV}			250	ns
$\overline{OE}$ high to DATA tri-stated	t_{OHDZ}			250	ns

Notes: 1. t_{WLRH} is valid for the write flash, write EEPROM, write fuse bits and write lock bits commands.
2. t_{WLRH_CE} is valid for the chip erase command.

21.10 Serial Programming Characteristics

Figure 21-9. Serial Programming Timing

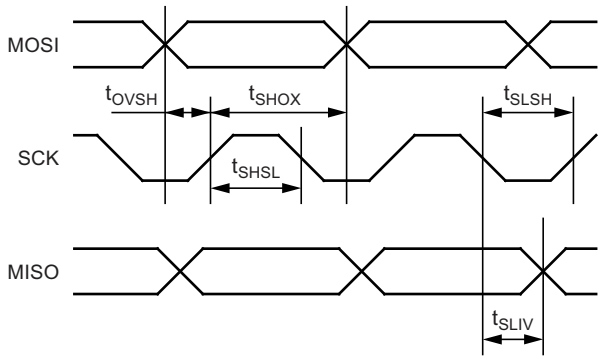


Figure 21-10. Serial Programming Waveforms

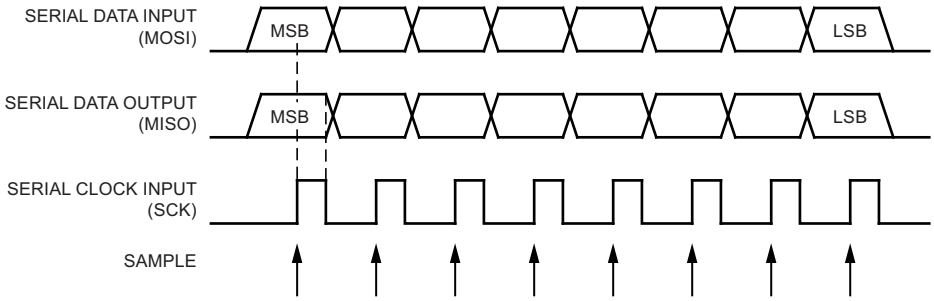


Table 21-10. Serial Programming Characteristics, T_A = –40°C to +125°C, V_{CC} = 2.7 to 5.5V (Unless Otherwise Noted)

Parameter	Symbol	Min	Typ	Max	Unit
Oscillator frequency (V_{CC} = 2.7V to 5.5V)	$1/t_{CLCL}$	0		8	MHz
Oscillator period (V_{CC} = 2.7V to 5.5V)	t_{CLCL}	125			ns
Oscillator frequency (V_{CC} = 4.5V to 5.5V)	$1/t_{CLCL}$	0		16	MHz
Oscillator period (V_{CC} = 4.5V to 5.5V)	t_{CLCL}	62.5			ns
SCK pulse width high	t_{SHSL}	2 t_{CLCL}*			ns
SCK pulse width low	t_{SLSH}	2 t_{CLCL}*			ns
MOSI setup to SCK high	t_{OVSH}	t_{CLCL}			ns
MOSI hold after SCK high	t_{SHOX}	2 t_{CLCL}			ns

22. Typical Charateristics

The data contained in this section is largely based on simulations and characterization of similar devices in the same process and design methods. Thus, the data should be treated as indications of how the part will behave.

The following charts show typical behavior. These figures are not tested during manufacturing. All current consumption measurements are performed with all I/O pins configured as inputs and with internal pull-ups enabled. A sine wave generator with rail-to-rail output is used as clock source.

The power consumption in power-down mode is independent of clock selection.

The current consumption is a function of several factors such as: operating voltage, operating frequency, loading of I/O pins, switching rate of I/O pins, code executed and ambient temperature. The dominating factors are operating voltage and frequency.

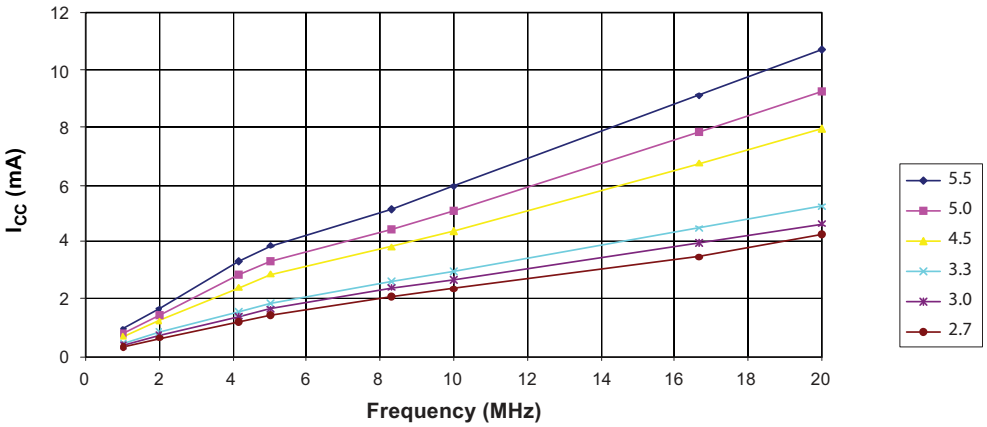
The current drawn from capacitive loaded pins may be estimated (for one pin) as $C_L \times V_{CC} \times f$ where C_L = load capacitance, V_{CC} = operating voltage and f = average switching frequency of I/O pin.

The parts are characterized at frequencies higher than test limits. Parts are not guaranteed to function properly at frequencies higher than the ordering code indicates.

The difference between current consumption in power-down mode with watchdog timer enabled and power-down mode with watchdog timer disabled represents the differential current drawn by the watchdog timer.

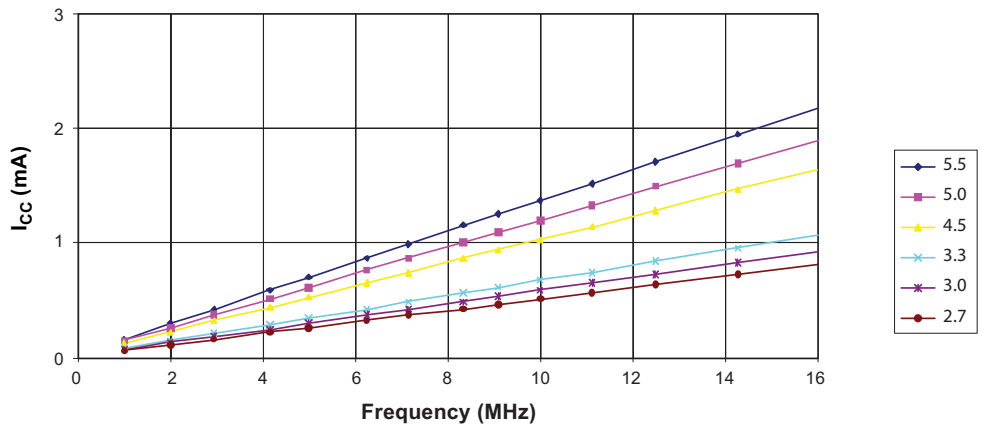
22.1 Active Supply Current

Figure 22-1. Active Supply Current versus Frequency (0 - 16 MHz)



22.2 Idle Supply Current

Figure 22-2. Idle Supply Current versus Frequency (0 - 16 MHz)



22.3 Power-down Supply Current

Figure 22-3. Power-down Supply Current versus V_{CC} (Watchdog Timer Disabled)

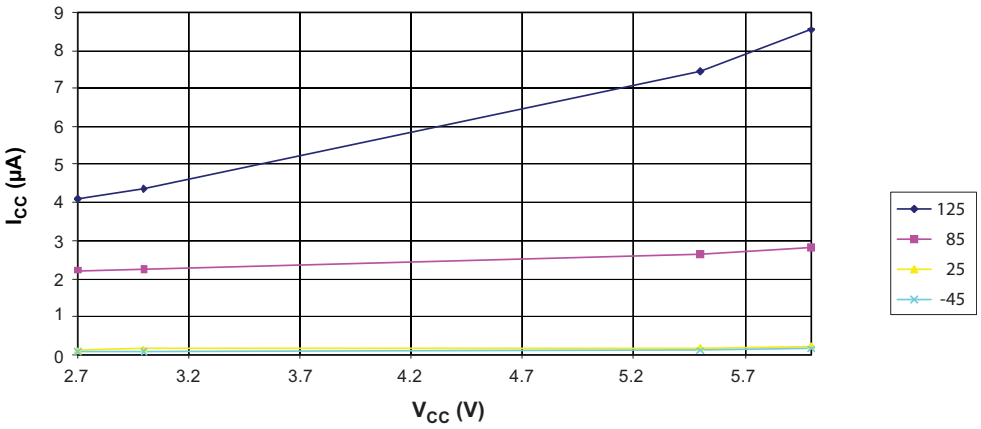
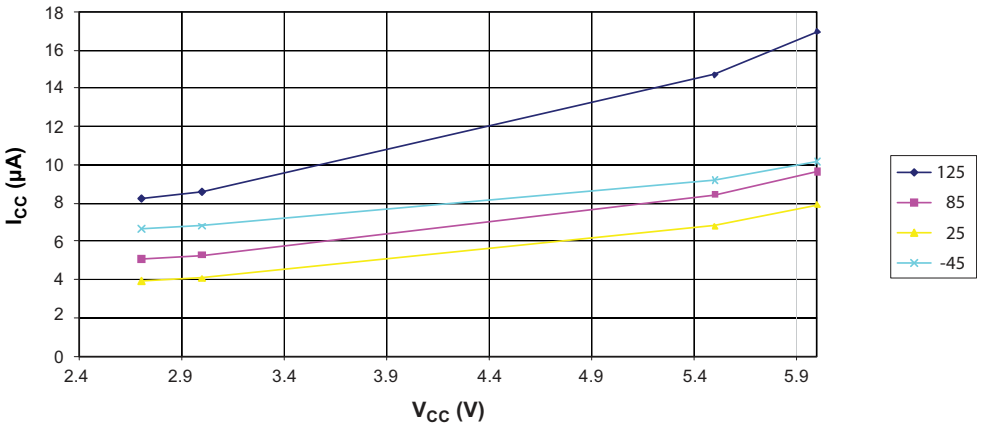


Figure 22-4. Power-down Supply Current versus V_{CC} (Watchdog Timer Enabled)



22.4 Pin Pull-up

Figure 22-5. I/O Pin Pull-up Resistor Current versus Input Voltage

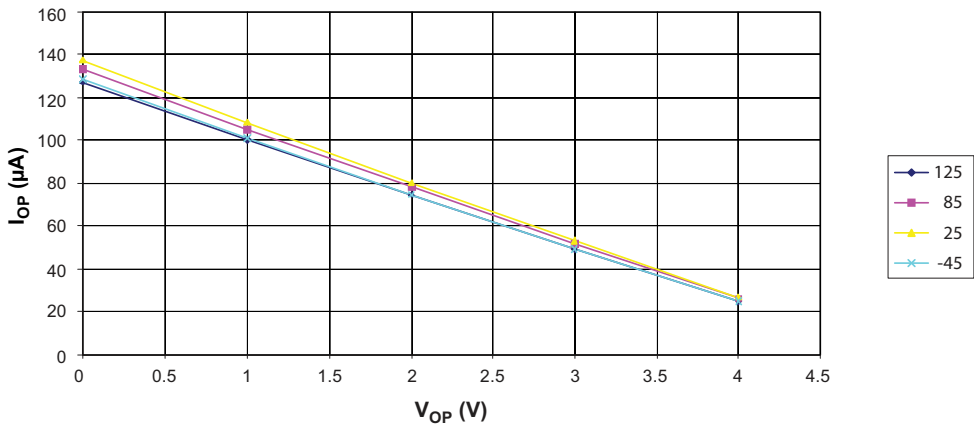
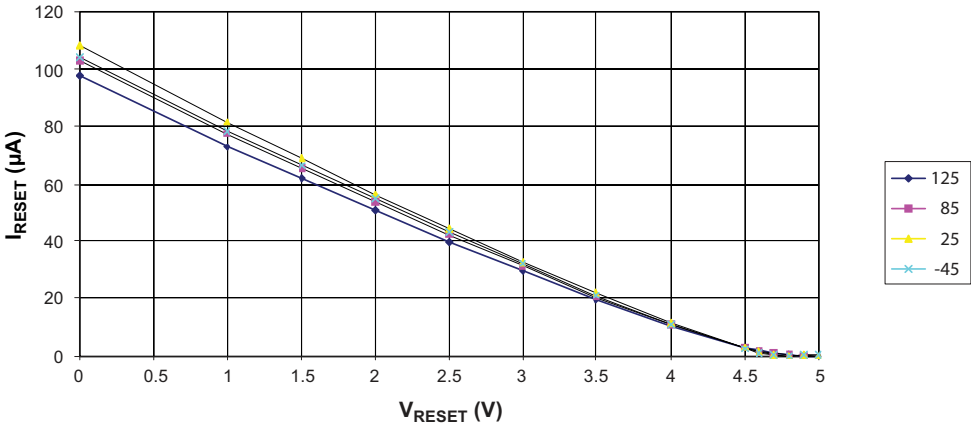


Figure 22-6. Reset Pull-up Resistor Current versus Reset Pin Voltage



22.5 Pin Driver Strength

Figure 22-7. High Sink I/O Pin Output Voltage versus Sink Current (V_{CC} = 3V / Iol = 5mA)

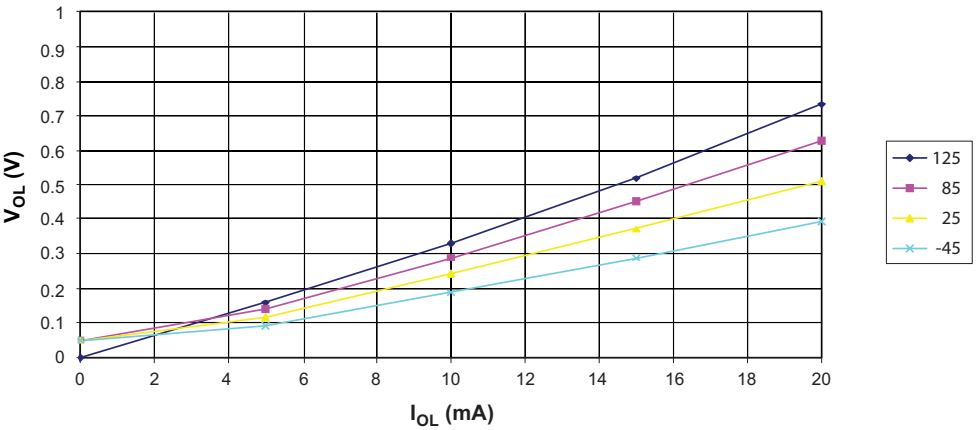


Figure 22-8. High Sink I/O Pin Output Voltage versus Sink Current (V_{CC} = 5V / Iol = 20mA)

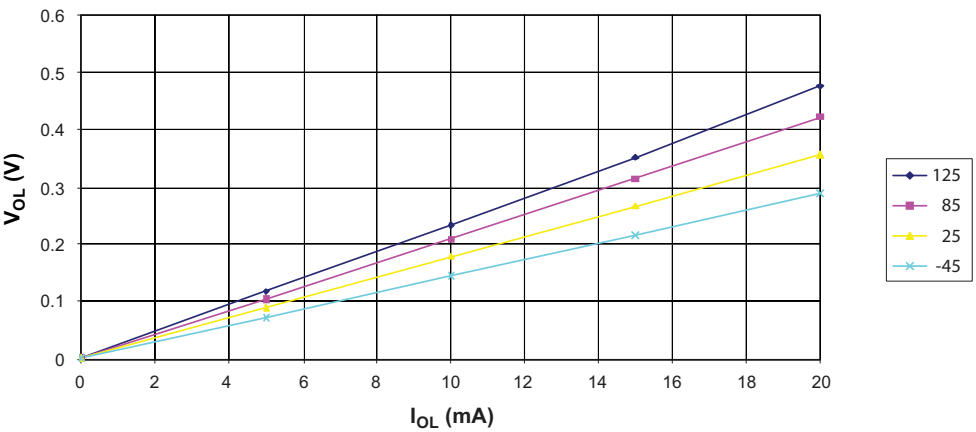


Figure 22-9. Standard I/O Pin Output Voltage versus Source Current (V_{CC} = 3V / Ioh = –5mA)

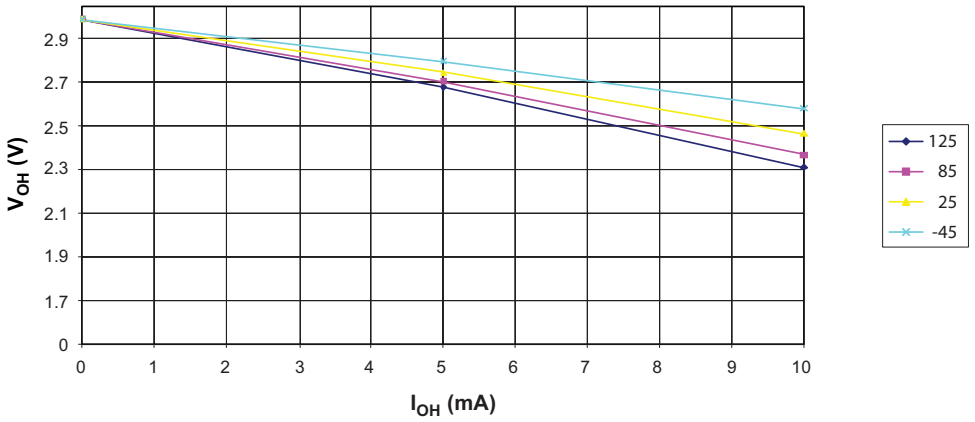
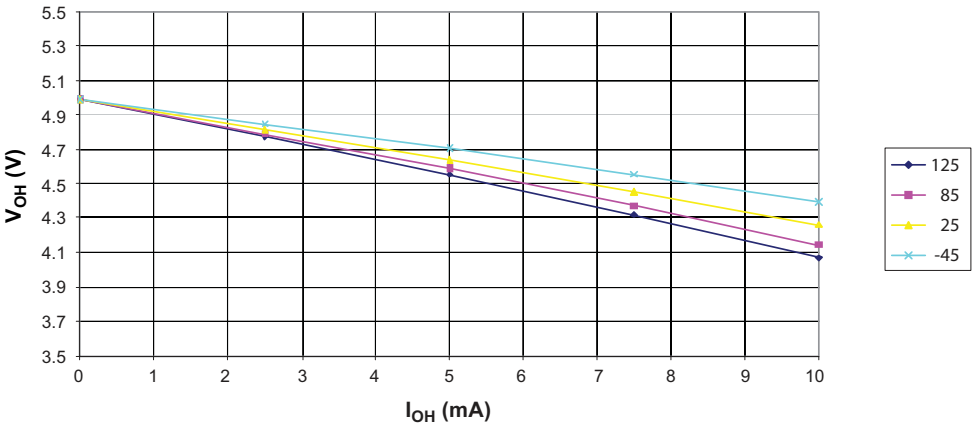


Figure 22-10. Standard I/O Pin Output Voltage versus Source Current (V_{CC} = 5V / Ioh = –15mA)



22.6 Pin Threshold and Hysteresis

Figure 22-11. I/O Pin Input Threshold Voltage versus V_{CC} (V_{IH}, IO Pin Read as '1')

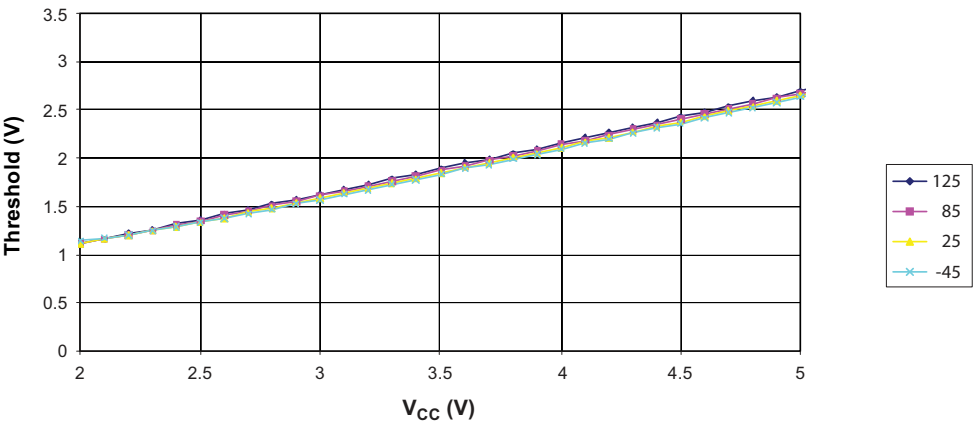


Figure 22-12. I/O Pin Input Threshold Voltage versus V_{CC} (V_{IL}, IO Pin Read as '0')

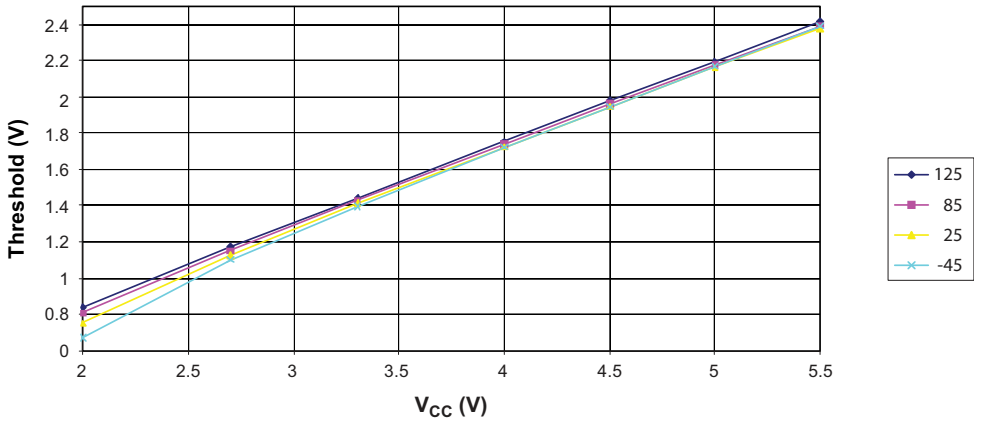


Figure 22-13. Reset Input Threshold Voltage versus V_{CC} (V_{IH}, IO Pin Read as '1')

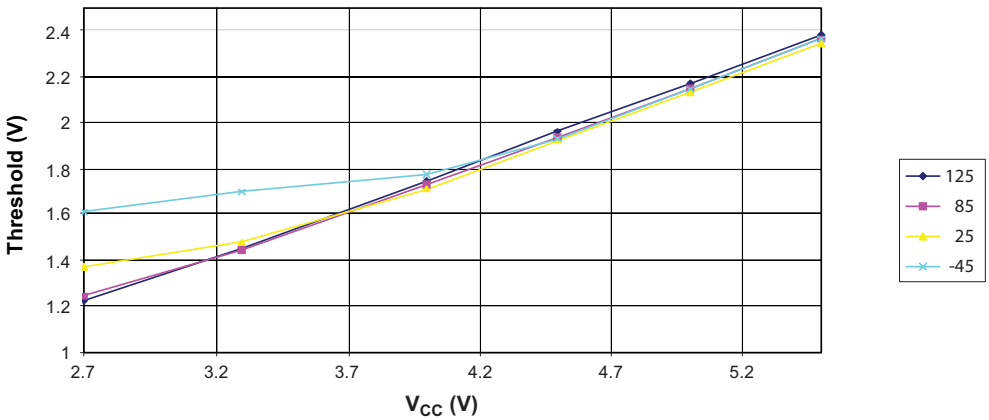
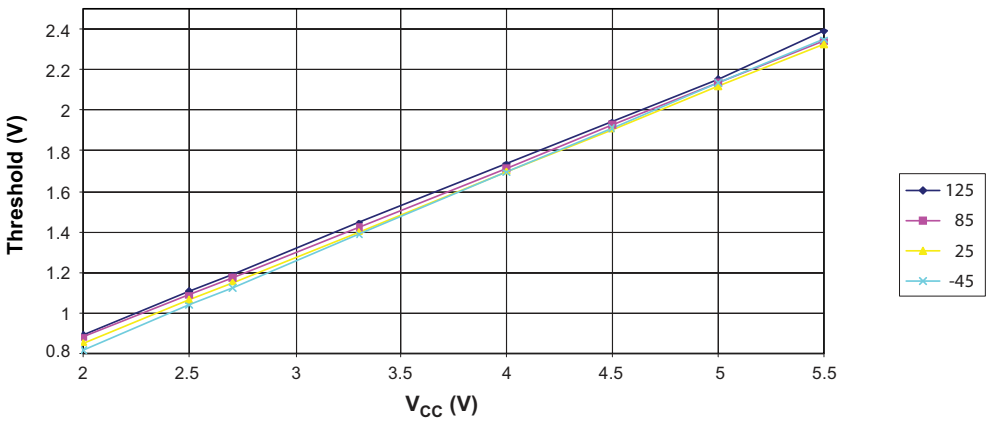


Figure 22-14. Reset Input Threshold Voltage versus V_{CC} (V_{IL}, IO Pin Read as '0')



22.7 BOD Threshold

Figure 22-15. BOD Threshold versus Temperature (BOD Level is 4.3V)

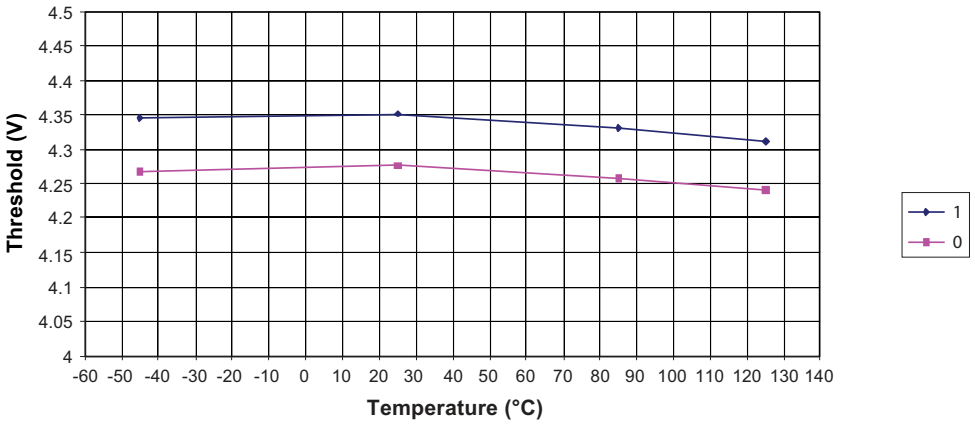
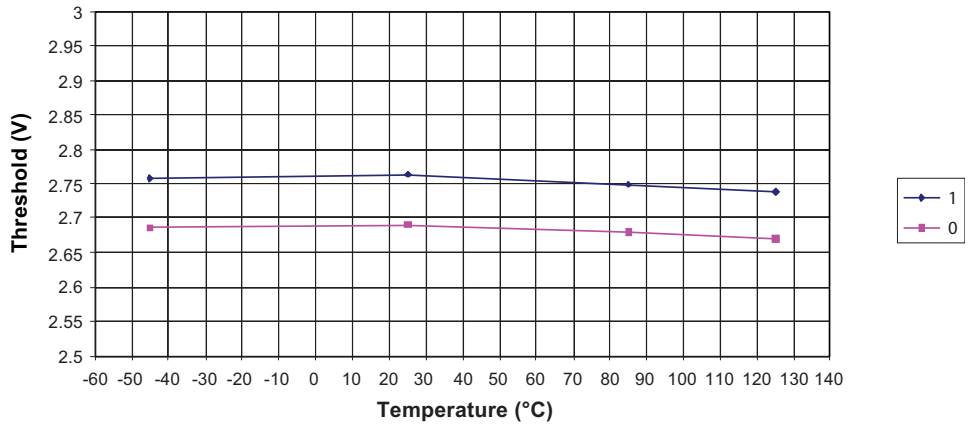


Figure 22-16. BOD Threshold versus Temperature (BOD Level is 2.7V)



22.8 Internal Oscillator Speed

Figure 22-17. Watchdog Oscillator Frequency versus Temperature

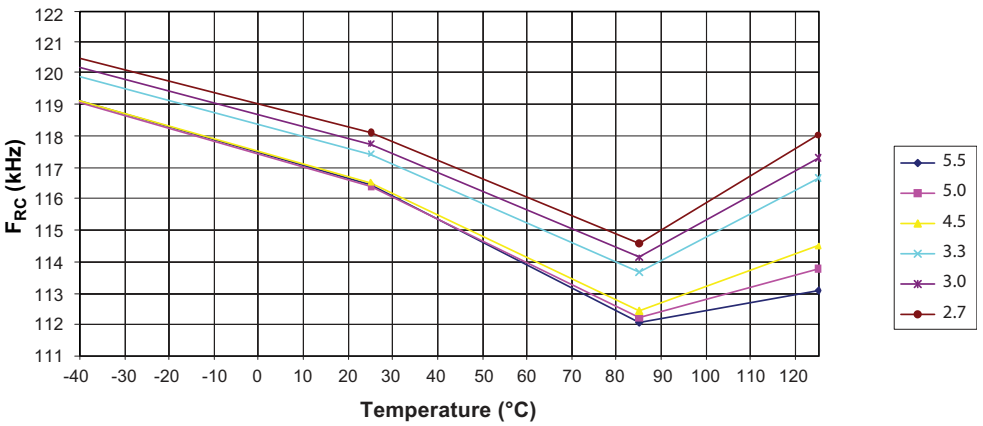


Figure 22-18. Calibrated 8MHz Oscillator Frequency versus Temperature

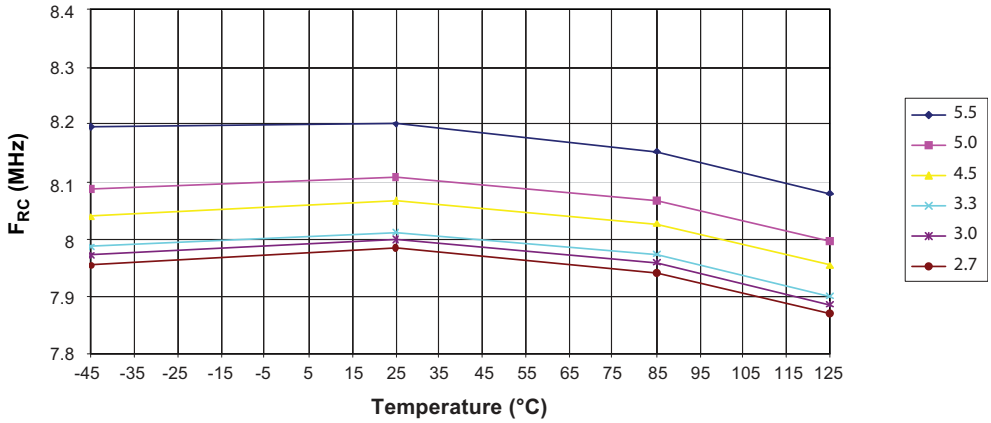
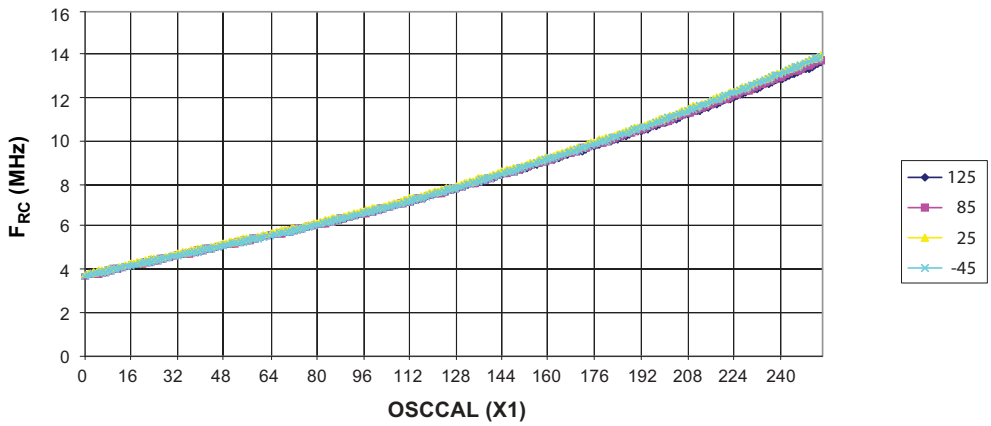


Figure 22-19. Calibrated 8MHz Oscillator Frequency versus OSCCAL Value



23. Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xFF)	Reserved	–	–	–	–	–	–	–	–	
(0xFE)	Reserved	–	–	–	–	–	–	–	–	
(0xFD)	Reserved	–	–	–	–	–	–	–	–	
(0xFC)	Reserved	–	–	–	–	–	–	–	–	
(0xFB)	Reserved	–	–	–	–	–	–	–	–	
(0xFA)	Reserved	–	–	–	–	–	–	–	–	
(0xF9)	Reserved	–	–	–	–	–	–	–	–	
(0xF8)	Reserved	–	–	–	–	–	–	–	–	
(0xF7)	Reserved	–	–	–	–	–	–	–	–	
(0xF6)	Reserved	–	–	–	–	–	–	–	–	
(0xF5)	Reserved	–	–	–	–	–	–	–	–	
(0xF4)	Reserved	–	–	–	–	–	–	–	–	
(0xF3)	Reserved	–	–	–	–	–	–	–	–	
(0xF2)	Reserved	–	–	–	–	–	–	–	–	
(0xF1)	Reserved	–	–	–	–	–	–	–	–	
(0xF0)	Reserved	–	–	–	–	–	–	–	–	
(0xEF)	Reserved	–	–	–	–	–	–	–	–	
(0xEE)	Reserved	–	–	–	–	–	–	–	–	
(0xED)	Reserved	–	–	–	–	–	–	–	–	
(0xEC)	Reserved	–	–	–	–	–	–	–	–	
(0xEB)	Reserved	–	–	–	–	–	–	–	–	
(0xEA)	Reserved	–	–	–	–	–	–	–	–	
(0xE9)	Reserved	–	–	–	–	–	–	–	–	
(0xE8)	Reserved	–	–	–	–	–	–	–	–	
(0xE7)	Reserved	–	–	–	–	–	–	–	–	
(0xE6)	Reserved	–	–	–	–	–	–	–	–	
(0xE5)	Reserved	–	–	–	–	–	–	–	–	
(0xE4)	Reserved	–	–	–	–	–	–	–	–	
(0xE3)	Reserved	–	–	–	–	–	–	–	–	
(0xE2)	Reserved	–	–	–	–	–	–	–	–	
(0xE1)	Reserved	–	–	–	–	–	–	–	–	
(0xE0)	Reserved	–	–	–	–	–	–	–	–	
(0xDF)	Reserved	–	–	–	–	–	–	–	–	

Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O Registers within the address range 0x00 – 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the status flags are cleared by writing a logical one to them. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such status flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 – 0x3F must be used. When addressing I/O registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The ATtiny88 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the extended I/O space from 0x60 – 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

23. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xDE)	Reserved	–	–	–	–	–	–	–	–	
(0xDD)	Reserved	–	–	–	–	–	–	–	–	
(0xDC)	Reserved	–	–	–	–	–	–	–	–	
(0xDB)	Reserved	–	–	–	–	–	–	–	–	
(0xDA)	Reserved	–	–	–	–	–	–	–	–	
(0xD9)	Reserved	–	–	–	–	–	–	–	–	
(0xD8)	Reserved	–	–	–	–	–	–	–	–	
(0xD7)	Reserved	–	–	–	–	–	–	–	–	
(0xD6)	Reserved	–	–	–	–	–	–	–	–	
(0xD5)	Reserved	–	–	–	–	–	–	–	–	
(0xD4)	Reserved	–	–	–	–	–	–	–	–	
(0xD3)	Reserved	–	–	–	–	–	–	–	–	
(0xD2)	Reserved	–	–	–	–	–	–	–	–	
(0xD1)	Reserved	–	–	–	–	–	–	–	–	
(0xD0)	Reserved	–	–	–	–	–	–	–	–	
(0xCF)	Reserved	–	–	–	–	–	–	–	–	
(0xCE)	Reserved	–	–	–	–	–	–	–	–	
(0xCD)	Reserved	–	–	–	–	–	–	–	–	
(0xCC)	Reserved	–	–	–	–	–	–	–	–	
(0xCB)	Reserved	–	–	–	–	–	–	–	–	
(0xCA)	Reserved	–	–	–	–	–	–	–	–	
(0xC9)	Reserved	–	–	–	–	–	–	–	–	
(0xC8)	Reserved	–	–	–	–	–	–	–	–	
(0xC7)	Reserved	–	–	–	–	–	–	–	–	
(0xC6)	Reserved	–	–	–	–	–	–	–	–	
(0xC5)	Reserved	–	–	–	–	–	–	–	–	
(0xC4)	Reserved	–	–	–	–	–	–	–	–	
(0xC3)	Reserved	–	–	–	–	–	–	–	–	
(0xC2)	Reserved	–	–	–	–	–	–	–	–	
(0xC1)	Reserved	–	–	–	–	–	–	–	–	
(0xC0)	Reserved	–	–	–	–	–	–	–	–	
(0xBF)	Reserved	–	–	–	–	–	–	–	–	
(0xBE)	TWHSR	–	–	–	–	–	–	–	TWHS	141

Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O Registers within the address range 0x00 – 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the status flags are cleared by writing a logical one to them. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such status flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 – 0x3F must be used. When addressing I/O registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The ATtiny88 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the extended I/O space from 0x60 – 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

23. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xBD)	TWAMR	TWAM6	TWAM5	TWAM4	TWAM3	TWAM2	TWAM1	TWAM0	–	140
(0xBC)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	138
(0xBB)	TWDR	2-wire serial interface data register								139
(0xBA)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	140
(0xB9)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	139
(0xB8)	TWBR	2-wire serial interface bit rate register								137
(0xB7)	Reserved	–	–	–	–	–	–	–	–	
(0xB6)	Reserved	–	–	–	–	–	–	–	–	
(0xB5)	Reserved	–	–	–	–	–	–	–	–	
(0xB4)	Reserved	–	–	–	–	–	–	–	–	
(0xB3)	Reserved	–	–	–	–	–	–	–	–	
(0xB2)	Reserved	–	–	–	–	–	–	–	–	
(0xB1)	Reserved	–	–	–	–	–	–	–	–	
(0xB0)	Reserved	–	–	–	–	–	–	–	–	
(0xAF)	Reserved	–	–	–	–	–	–	–	–	
(0xAE)	Reserved	–	–	–	–	–	–	–	–	
(0xAD)	Reserved	–	–	–	–	–	–	–	–	
(0xAC)	Reserved	–	–	–	–	–	–	–	–	
(0xAB)	Reserved	–	–	–	–	–	–	–	–	
(0xAA)	Reserved	–	–	–	–	–	–	–	–	
(0xA9)	Reserved	–	–	–	–	–	–	–	–	
(0xA8)	Reserved	–	–	–	–	–	–	–	–	
(0xA7)	Reserved	–	–	–	–	–	–	–	–	
(0xA6)	Reserved	–	–	–	–	–	–	–	–	
(0xA5)	Reserved	–	–	–	–	–	–	–	–	
(0xA4)	Reserved	–	–	–	–	–	–	–	–	
(0xA3)	Reserved	–	–	–	–	–	–	–	–	
(0xA2)	Reserved	–	–	–	–	–	–	–	–	
(0xA1)	Reserved	–	–	–	–	–	–	–	–	
(0xA0)	Reserved	–	–	–	–	–	–	–	–	
(0x9F)	Reserved	–	–	–	–	–	–	–	–	
(0x9E)	Reserved	–	–	–	–	–	–	–	–	
(0x9D)	Reserved	–	–	–	–	–	–	–	–	

Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O Registers within the address range 0x00 – 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the status flags are cleared by writing a logical one to them. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such status flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 – 0x3F must be used. When addressing I/O registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The ATtiny88 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the extended I/O space from 0x60 – 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

23. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0x9C)	Reserved	–	–	–	–	–	–	–	–	
(0x9B)	Reserved	–	–	–	–	–	–	–	–	
(0x9A)	Reserved	–	–	–	–	–	–	–	–	
(0x99)	Reserved	–	–	–	–	–	–	–	–	
(0x98)	Reserved	–	–	–	–	–	–	–	–	
(0x97)	Reserved	–	–	–	–	–	–	–	–	
(0x96)	Reserved	–	–	–	–	–	–	–	–	
(0x95)	Reserved	–	–	–	–	–	–	–	–	
(0x94)	Reserved	–	–	–	–	–	–	–	–	
(0x93)	Reserved	–	–	–	–	–	–	–	–	
(0x92)	Reserved	–	–	–	–	–	–	–	–	
(0x91)	Reserved	–	–	–	–	–	–	–	–	
(0x90)	Reserved	–	–	–	–	–	–	–	–	
(0x8F)	Reserved	–	–	–	–	–	–	–	–	
(0x8E)	Reserved	–	–	–	–	–	–	–	–	
(0x8D)	Reserved	–	–	–	–	–	–	–	–	
(0x8C)	Reserved	–	–	–	–	–	–	–	–	
(0x8B)	OCR1BH	Timer/Counter1 – Output compare register B high byte								99
(0x8A)	OCR1BL	Timer/Counter1 – Output compare register B low byte								99
(0x89)	OCR1AH	Timer/Counter1 – Output compare register A high byte								99
(0x88)	OCR1AL	Timer/Counter1 – Output compare register A low byte								99
(0x87)	ICR1H	Timer/Counter1 – Input capture register high byte								100
(0x86)	ICR1L	Timer/Counter1 – Input capture register low byte								100
(0x85)	TCNT1H	Timer/Counter1 – Counter register high byte								99
(0x84)	TCNT1L	Timer/Counter1 – Counter register low byte								99
(0x83)	Reserved	–	–	–	–	–	–	–	–	
(0x82)	TCCR1C	FOC1A	FOC1B	–	–	–	–	–	–	98
(0x81)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	97
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	95
(0x7F)	DIDR1	–	–	–	–	–	–	AIN1D	AIN0D	144
(0x7E)	DIDR0	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	159
(0x7D)	Reserved	–	–	–	–	–	–	–	–	
(0x7C)	ADMUX	–	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	155

Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O Registers within the address range 0x00 – 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the status flags are cleared by writing a logical one to them. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such status flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 – 0x3F must be used. When addressing I/O registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The ATtiny88 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the extended I/O space from 0x60 – 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

23. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0x7B)	ADCSRB	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	158
(0x7A)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	157
(0x79)	ADCH	ADC data register high byte								158
(0x78)	ADCL	ADC data register low byte								158
(0x77)	Reserved	–	–	–	–	–	–	–	–	
(0x76)	Reserved	–	–	–	–	–	–	–	–	
(0x75)	Reserved	–	–	–	–	–	–	–	–	
(0x74)	Reserved	–	–	–	–	–	–	–	–	
(0x73)	Reserved	–	–	–	–	–	–	–	–	
(0x72)	Reserved	–	–	–	–	–	–	–	–	
(0x71)	Reserved	–	–	–	–	–	–	–	–	
(0x70)	Reserved	–	–	–	–	–	–	–	–	
(0x6F)	TIMSK1	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	100
(0x6E)	TIMSK0	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	75
(0x6D)	PCMSK2	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	50
(0x6C)	PCMSK1	PCINT15	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	50
(0x6B)	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	50
(0x6A)	PCMSK3	–	–	-	-	PCINT27	PCINT26	PCINT25	PCINT24	50
(0x69)	EICRA	–	–	–	–	ISC11	ISC10	ISC01	ISC00	47
(0x68)	PCICR	–	–	–	–	PCIE3	PCIE2	PCIE1	PCIE0	48
(0x67)	Reserved	–	–	–	–	–	–	–	–	
(0x66)	OSCCAL	Oscillator calibration register								30
(0x65)	Reserved	–	–	–	–	–	–	–	–	
(0x64)	PRR	PRTWI	–	PRTIM0	–	PRTIM1	PRSPI	–	PRADC	36
(0x63)	Reserved	–	–	–	–	–	–	–	–	
(0x62)	Reserved	–	–	–	–	–	–	–	–	
(0x61)	CLKPR	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0	30
(0x60)	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	42
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	11
0x3E (0x5E)	Reserved	–	–	–	–	–	–	–	–	
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	13
0x3C (0x5C)	Reserved	–	–	–	–	–	–	–	–	
0x3B (0x5B)	Reserved	–	–	–	–	–	–	–	–	

Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O Registers within the address range 0x00 – 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the status flags are cleared by writing a logical one to them. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such status flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 – 0x3F must be used. When addressing I/O registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The ATtiny88 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the extended I/O space from 0x60 – 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

23. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x3A (0x5A)	Reserved	–	–	–	–	–	–	–	–	
0x39 (0x59)	Reserved	–	–	–	–	–	–	–	–	
0x38 (0x58)	Reserved	–	–	–	–	–	–	–	–	
0x37 (0x57)	SPMCSR	–	RWWSB	–	CTPB	RFLB	PGWRT	PGERS	SELFPRGEN	167
0x36 (0x56)	Reserved	–	–	–	–	–	–	–	–	
0x35 (0x55)	MCUCR	–	BPDS	BPDSE	PUD	–	–	–	–	
0x34 (0x54)	MCUSR	–	–	–	–	WDRF	BORF	EXTRF	PORF	42
0x33 (0x53)	SMCR	–	–	–	–	–	SM1	SM0	SE	35
0x32 (0x52)	Reserved	–	–	–	–	–	–	–	–	
0x31 (0x51)	DWDR	debugWire data register								161
0x30 (0x50)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	143
0x2F (0x4F)	Reserved	–	–	–	–	–	–	–	–	
0x2E (0x4E)	SPDR	SPI data register								111
0x2D (0x4D)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	111
0x2C (0x4C)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	109
0x2B (0x4B)	GPIOR2	General purpose I/O register 2								23
0x2A (0x4A)	GPIOR1	General purpose I/O register 1								24
0x29 (0x49)	Reserved	–	–	–	–	–	–	–	–	
0x28 (0x48)	OCR0B	Timer/Counter0 output compare register B								74
0x27 (0x47)	OCR0A	Timer/Counter0 output compare register A								74
0x26 (0x46)	TCNT0	Timer/Counter0 (8-bit)								74
0x25 (0x45)	TCCR0A	–	–	–	–	CTC0	CS02	CS01	CS00	73
0x24 (0x44)	Reserved	–	–	–	–	–	–	–	–	
0x23 (0x43)	GTCCR	TSM	–	–	–	–	–	–	PSRSYNC	103
0x22 (0x42)	Reserved	–	–	–	–	–	–	–	–	
0x21 (0x41)	EEARL	EEPROM address register low byte								21
0x20 (0x40)	EEDR	EEPROM data register								22
0x1F (0x3F)	EECR	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	22
0x1E (0x3E)	GPIOR0	General purpose I/O register 0								24
0x1D (0x3D)	EIMSK	–	–	–	–	–	–	INT1	INT0	48
0x1C (0x3C)	EIFR	–	–	–	–	–	–	INTF1	INTF0	48
0x1B (0x3B)	PCIFR	–	–	–	–	PCIF3	PCIF2	PCIF1	PCIF0	49
0x1A (0x3A)	Reserved	–	–	–	–	–	–	–	–	

Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O Registers within the address range 0x00 – 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the status flags are cleared by writing a logical one to them. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such status flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 – 0x3F must be used. When addressing I/O registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The ATtiny88 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the extended I/O space from 0x60 – 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

23. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x19 (0x39)	Reserved	–	–	–	–	–	–	–	–	
0x18 (0x38)	Reserved	–	–	–	–	–	–	–	–	
0x17 (0x37)	Reserved	–	–	–	–	–	–	–	–	
0x16 (0x36)	TIFR1	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	101
0x15 (0x35)	TIFR0	–	–	–	–	–	OCF0B	OCF0A	TOV0	75
0x14 (0x34)	Reserved	–	–	–	–	–	–	–	–	
0x13 (0x33)	Reserved	–	–	–	–	–	–	–	–	
0x12 (0x32)	PORTCR	BBMD	BBMC	BBMB	BBMA	PUDD	PUDC	PUDB	PUDA	65
0x11 (0x31)	Reserved	–	–	–	–	–	–	–	–	
0x10 (0x30)	Reserved	–	–	–	–	–	–	–	–	
0x0F (0x2F)	Reserved	–	–	–	–	–	–	–	–	
0x0E (0x2E)	PORTA	–	–	–	–	PORTA3	PORTA2	PORTA1	PORTA0	67
0x0D (0x2D)	DDRA	–	–	–	–	DDA3	DDA2	DDA1	DDA0	67
0x0C (0x2C)	PINA	–	–	–	–	PINA3	PINA2	PINA1	PINA0	67
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	67
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	67
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	67
0x08 (0x28)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	67
0x07 (0x27)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	67
0x06 (0x26)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	67
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	66
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	66
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	66
0x02 (0x22)	Reserved	–	–	–	–	–	–	–	–	
0x01 (0x21)	Reserved	–	–	–	–	–	–	–	–	
0x00 (0x20)	Reserved	–	–	–	–	–	–	–	–	

Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O Registers within the address range 0x00 – 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the status flags are cleared by writing a logical one to them. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such status flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 – 0x3F must be used. When addressing I/O registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The ATtiny88 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the extended I/O space from 0x60 – 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

24. Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
Arithmetic and Logic Instructions					
ADD	Rd, Rr	Add two registers	Rd ← Rd + Rr	Z, C, N, V, H	1
ADC	Rd, Rr	Add with carry two registers	Rd ← Rd + Rr + C	Z, C, N, V, H	1
ADIW	Rdl, K	Add immediate to word	Rdh:Rdl ← Rdh:Rdl + K	Z, C, N, V, S	2
SUB	Rd, Rr	Subtract two registers	Rd ← Rd – Rr	Z, C, N, V, H	1
SUBI	Rd, K	Subtract constant from register	Rd ← Rd – K	Z, C, N, V, H	1
SBC	Rd, Rr	Subtract with carry two registers	Rd ← Rd – Rr – C	Z, C, N, V, H	1
SBCI	Rd, K	Subtract with carry constant from reg.	Rd ← Rd – K – C	Z, C, N, V, H	1
SBIW	Rdl, K	Subtract immediate from word	Rdh:Rdl ← Rdh:Rdl – K	Z, C, N, V, S	2
AND	Rd, Rr	Logical AND registers	Rd ← Rd × Rr	Z, N, V	1
ANDI	Rd, K	Logical AND register and constant	Rd ← Rd × K	Z, N, V	1
OR	Rd, Rr	Logical OR registers	Rd ← Rd v Rr	Z, N, V	1
ORI	Rd, K	Logical OR register and constant	Rd ← Rd v K	Z, N, V	1
EOR	Rd, Rr	Exclusive OR registers	Rd ← Rd ⊕ Rr	Z, N, V	1
COM	Rd	One’s complement	Rd ← 0xFF – Rd	Z, C, N, V	1
NEG	Rd	Two’s complement	Rd ← 0x00 – Rd	Z, C, N, V, H	1
SBR	Rd, K	Set bit(s) in register	Rd ← Rd v K	Z, N, V	1
CBR	Rd, K	Clear bit(s) in register	Rd ← Rd × (0xFF – K)	Z, N, V	1
INC	Rd	Increment	Rd ← Rd + 1	Z, N, V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z, N, V	1
TST	Rd	Test for zero or minus	Rd ← Rd × Rd	Z, N, V	1
CLR	Rd	Clear register	Rd ← Rd ⊕ Rd	Z, N, V	1
SER	Rd	Set register	Rd ← 0xFF	None	1
Branch Instructions					
RJMP	k	Relative jump	PC ← PC + k + 1	None	2
IJMP		Indirect jump to (Z)	PC ← Z	None	2
RCALL	k	Relative subroutine call	PC ← PC + k + 1	None	3
ICALL		Indirect call to (Z)	PC ← Z	None	3
RET		Subroutine return	PC ← STACK	None	4
RETI		Interrupt return	PC ← STACK	I	4
CPSE	Rd, Rr	Compare, skip if equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
CP	Rd, Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd, Rr	Compare with carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd, K	Compare register with immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if bit in register cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip if bit in register is set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if bit in I/O register cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if bit in I/O register is set	if (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if status flag set	if (SREG(s) = 1) then PC ← PC + k + 1	None	1/2
BRBC	s, k	Branch if status flag cleared	if (SREG(s) = 0) then PC ← PC + k + 1	None	1/2

24. Instruction Set Summary (Continued)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BREQ	k	Branch if equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if lower	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if minus	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if plus	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if greater or equal, signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	Branch if less than zero, signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag is set	if (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag is cleared	if (V = 0) then PC ← PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if (I = 0) then PC ← PC + k + 1	None	1/2
Bit and Bit-test Instructions					
SBI	P,b	Set bit in I/O register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical shift left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical shift right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate left through carry	Rd(0) ← C,Rd(n+1) ← Rd(n),C ← Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate right through carry	Rd(7) ← C,Rd(n) ← Rd(n+1),C ← Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic shift right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap nibbles	Rd(3..0) ← Rd(7..4),Rd(7..4) ← Rd(3..0)	None	1
BSET	s	Flag set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit store from register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to register	Rd(b) ← T	None	1
SEC		Set carry	C ← 1	C	1
CLC		Clear carry	C ← 0	C	1
SEN		Set negative flag	N ← 1	N	1
CLN		Clear negative flag	N ← 0	N	1
SEZ		Set zero flag	Z ← 1	Z	1
CLZ		Clear zero flag	Z ← 0	Z	1
SEI		Global interrupt enable	I ← 1	I	1

24. Instruction Set Summary (Continued)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
CLI		Global interrupt disable	I ← 0	I	1
SES		Set signed test flag	S ← 1	S	1
CLS		Clear signed test flag	S ← 0	S	1
SEV		Set twos complement overflow	V ← 1	V	1
CLV		Clear twos complement overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set half carry flag in SREG	H ← 1	H	1
CLH		Clear half carry flag in SREG	H ← 0	H	1
Data Transfer Instructions					
MOV	Rd, Rr	Move between registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy register word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load immediate	Rd ← K	None	1
LD	Rd, X	Load indirect	Rd ← (X)	None	2
LD	Rd, X+	Load indirect and post-inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load indirect and pre-dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load indirect and post-inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load indirect and pre-dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd, Y+q	Load indirect with displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load indirect and post-inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load indirect and pre-dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load indirect with displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store indirect	(X) ← Rr	None	2
ST	X+, Rr	Store indirect and post-inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store indirect and pre-dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store indirect and post-inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store indirect and pre-dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q, Rr	Store indirect with displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store indirect and post-inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store indirect and pre-dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q, Rr	Store indirect with displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store direct to SRAM	(k) ← Rr	None	2
LPM		Load program memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load program memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load program memory and post-inc	Rd ← (Z), Z ← Z+1	None	3

24. Instruction Set Summary (Continued)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
SPM		Store program memory	(Z) ← R1:R0	None	-
IN	Rd, P	In port	Rd ← P	None	1
OUT	P, Rr	Out port	P ← Rr	None	1
PUSH	Rr	Push register on stack	STACK ← Rr	None	2
POP	Rd	Pop register from stack	Rd ← STACK	None	2
MCU Control Instructions					
NOP		No operation		None	1
SLEEP		Sleep	(see specific descr. for sleep function)	None	1
WDR		Watchdog reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For on-chip debug only	None	N/A

25. Ordering Information

25.1 ATtiny88

Speed (MHz)	Power Supply	Ordering Code	Package[1]	Operational Range
16	2.7 – 5.5	ATtiny88-15AZ	TQFP32 - MA	Automotive (–40°C to +125°C)
		ATtiny88-15MZ	QFN32 - PN	Automotive (–40°C to +125°C)

Note: 1. Pb-free packaging alternative, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive).Also halide free and fully green.

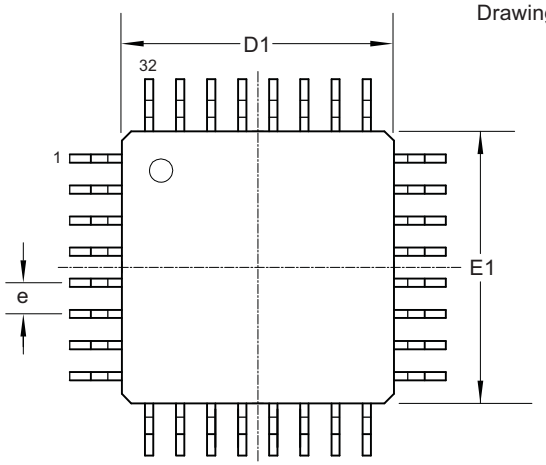
25.2 Package Types

	Package Type
MA	32-lead, 7 × 7mm body size, 1.0mm body thickness, 0.8mm lead pitch, thin profile plastic quad flat package (TQFP)
PN	32-pad, 5 × 5 × 1.0mm body, lead pitch 0.50mm, quad flat no-lead (QFN)

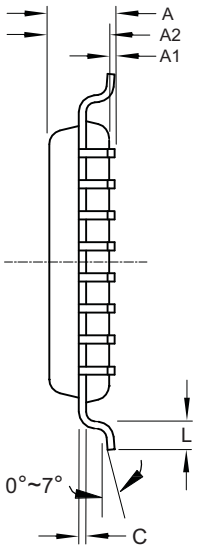
26. Packaging Information

26.1 MA

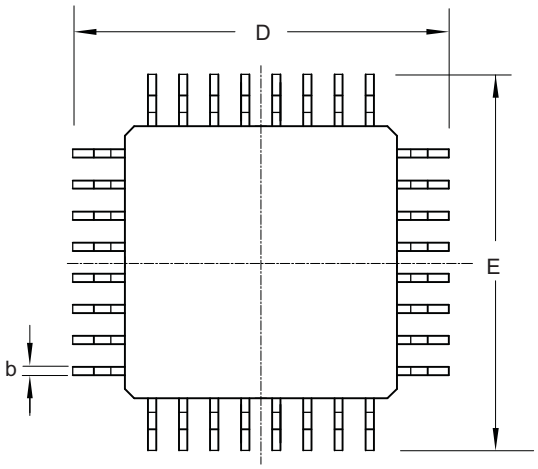
Drawings not scaled



Top View



Side View



Bottom View

COMMON DIMENSIONS (Unit of Measure = mm)				
Symbol	**MIN**	**NOM**	**MAX**	**NOTE**
A	----	----	1.20	
A1	0.05	----	0.15	
A2	0.95	1.00	1.05	
D/E	8.75	9.00	9.25	
D1/E1	6.90	7.00	7.10	2
C	0.09	----	0.20	
L	0.45	----	0.75	
b	0.30	----	0.45	
e	0.80 TYP.			
n	32			

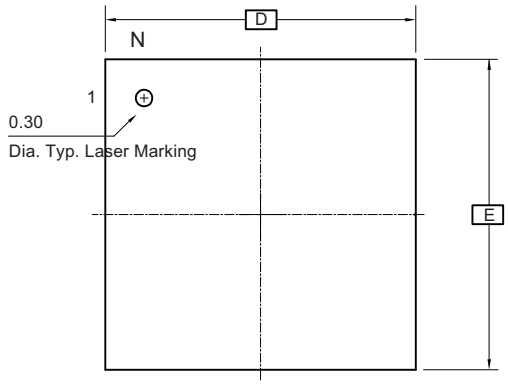
Notes: 1. This drawing is for general information only. Refer to JEDEC Drawing MS-026, Variation ABA.
2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25mm per side. Dimensions D1 and E1 are maximum plastic body size dimensions including mold mismatch.
3. Lead coplanarity is 0.10mm maximum.

02/29/12

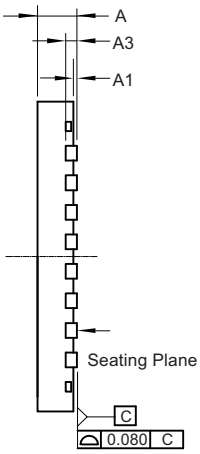
Atmel **Package Drawing Contact:** packagedrawings@atmel.com	TITLE	GPC	DRAWING NO.	REV.
	MA, 32 Lds - 0.80mm Pitch, 7x7x1.00mm Body size Thin Profile Plastic Quad Flat Package (TQFP)	AUT	MA	C

26.2 PN

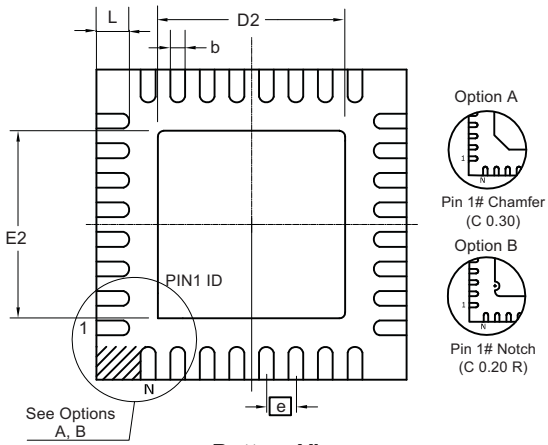
Drawings not scaled



Top View



Side View



Bottom View

COMMON DIMENSIONS (Unit of Measure = mm)				
Symbol	**MIN**	**NOM**	**MAX**	**NOTE**
A	0.80	0.85	0.90	
A1	0.00	----	0.05	
A3		0.20 REF		
D/E		5.00 BSC		
D2/E2	3.00	3.10	3.20	
L	0.30	0.40	0.50	
b	0.18	0.25	0.30	2
e		0.50 BSC		
n		32		

Notes: 1. This drawing is for general information only. Refer to JEDEC Drawing MO-220, Variation VHHD-2, for proper dimensions, tolerances, datums, etc.
2. Dimensions b applies to metallized terminal and is measured between 0.15mm and 0.30mm from the terminal tip. If the terminal has the optical radius on the other end of the terminal, the dimensions should not be measured in that radius area.

01/31/12

Package Drawing Contact: packagedrawings@atmel.com	TITLE	GPC	DRAWING NO.	REV.
	PN, 32 Leads - 0.50mm Pitch, 5x5mm Very Thin Quad Flat no Lead Package (VQFN) Sawn	ZMF	PN	I

27. Errata

27.1 Rev. A

No known errata.

28. Revision History

Please note that the following page numbers referred to in this section refer to the specific revision mentioned, not to this document.

Revision No.	History
9157E-AVR-07/14	• Section 25.2 “Package Types” on page 213 updated
9157D-AVR-07/14	• Put datasheet in the latest template
9157C-AVR-03/12	• MA package updated • PN package updated
9157B-AVR-01/10	• External clock drive updated • Serial programming characteristics updated
9157A-AVR-10/09	• External clock drive updated

29. Table of Contents

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**