## Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 130 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-Chip 2-cycle Multiplier
- Non-volatile Program and Data Memories
  - In-System Self-Programmable Flash, Endurance: 10,000 Write/Erase Cycles
    - 32K bytes (ATmega325/ATmega3250)
    - 64K bytes (ATmega645/ATmega6450)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - EEPROM, Endurance: 100,000 Write/Erase Cycles
    - 1K bytes (ATmega325/ATmega3250)
    - 2K bytes (ATmega645/ATmega6450)
  - Internal SRAM
    - 2K bytes (ATmega325/ATmega3250)
    - 4K bytes (ATmega645/ATmega6450)
  - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four PWM Channels
  - 8-channel, 10-bit ADC
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Universal Serial Interface with Start Condition Detector
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Five Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, and Standby
- I/O and Packages
  - 53/68 Programmable I/O Lines
  - 64-lead TQFP, 64-pad MLF, and 100-lead TQFP
- Speed Grade:
  - ATmega325V/ATmega3250V/ATmega645V/ATmega6450V:
    - 0 - 4 MHz @ 1.8 - 5.5V, 0 - 8 MHz @ 2.7 - 5.5V
  - ATmega325/3250/645/6450:
    - 0 - 8 MHz @ 2.7 - 5.5V, 0 - 16 MHz @ 4.5 - 5.5V
- Temperature range:
  - -40°C to 85°C Industrial

# 8-bit AVR® Microcontroller with In-System Programmable Flash

# ATmega325/V
# ATmega3250/V
# ATmega645/V
# ATmega6450/V

## Preliminary

2570C–AVR–11/04

## Features (Continued)

- **Ultra-Low Power Consumption**
  - **Active Mode:**
    **1 MHz, 1.8V: 350 µA**
    **32 kHz, 1.8V: 20 µA (including Oscillator)**
  - **Power-down Mode:**
    **100 nA at 1.8V**

## Pin Configurations

**Figure 1.** Pinout ATmega3250/6450

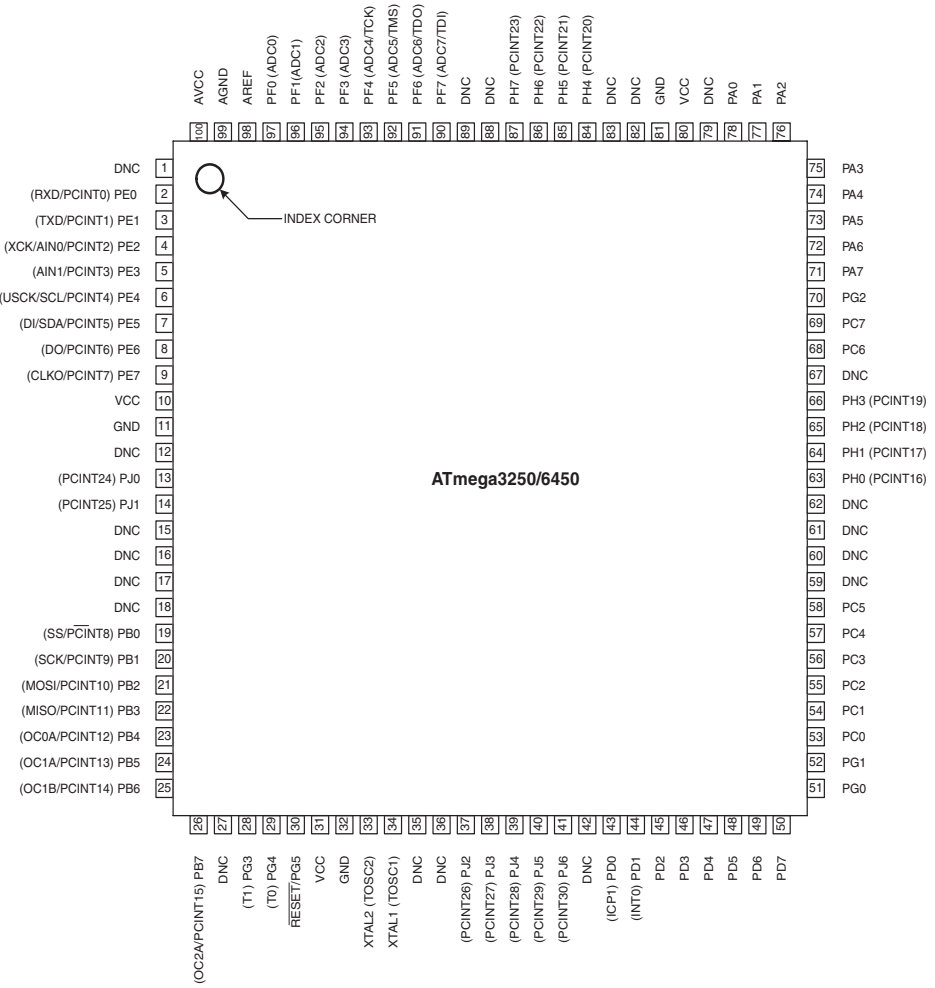
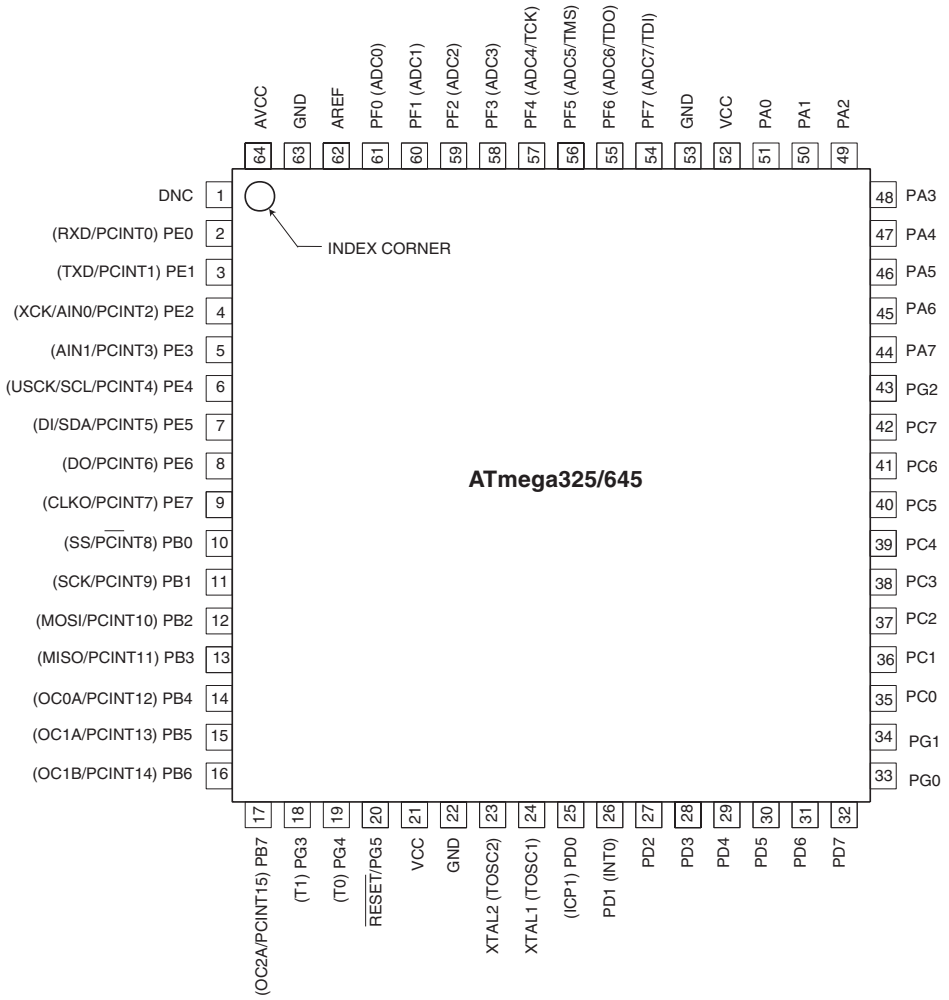


**Figure 2.** Pinout ATmega325/645




Note: The large center pad underneath the MLF packages is made of metal and internally connected to GND. It should be soldered or glued to the board to ensure good mechanical stability. If the center pad is left unconnected, the package might loosen from the board.

## Disclaimer

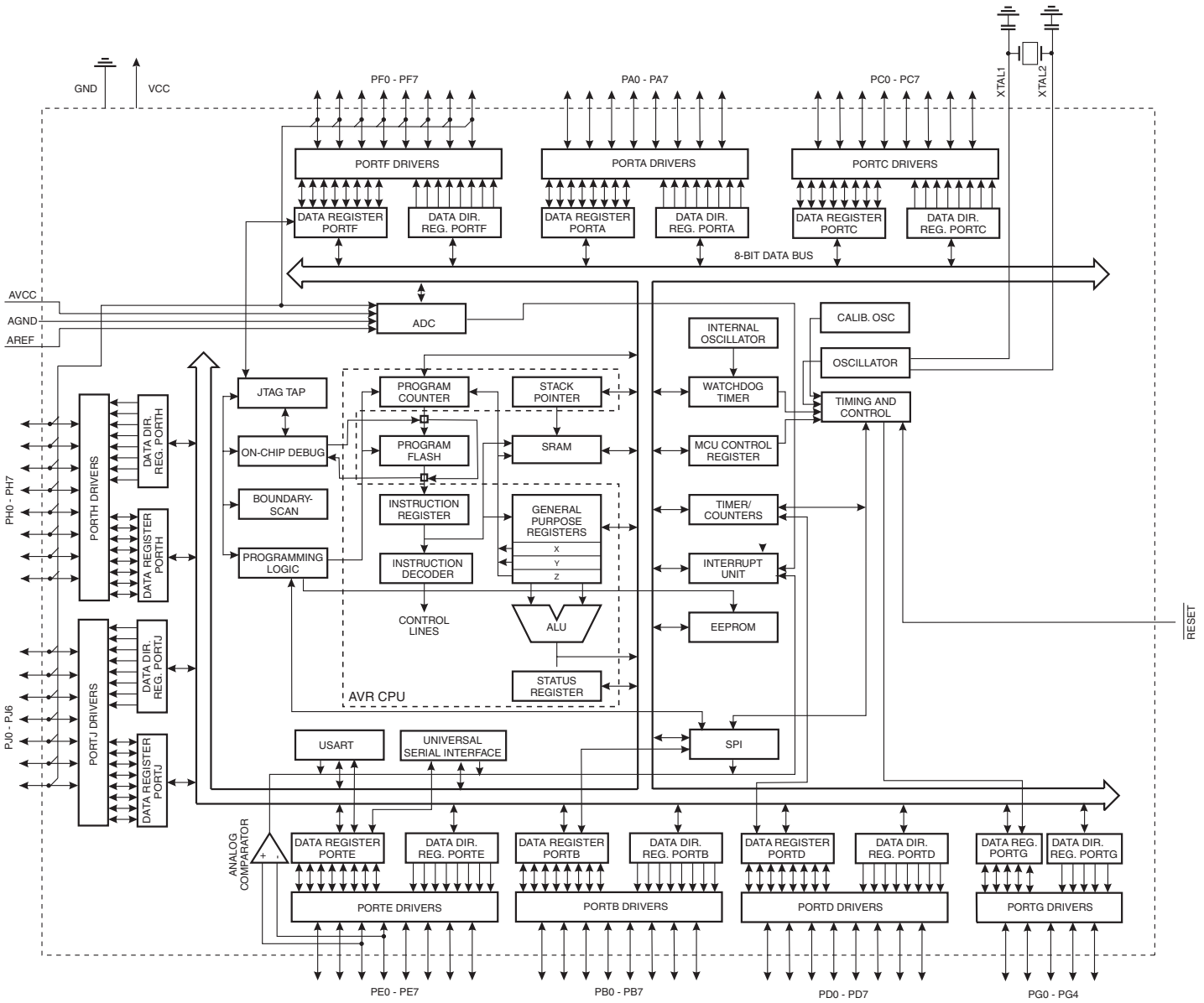
Typical values contained in this datasheet are based on simulations and characterization of other AVR microcontrollers manufactured on the same process technology. Min and Max values will be available after the device is characterized.

# Overview

The ATmega325/3250/645/6450 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 ATmega325/3250/645/6450 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

## Block Diagram

**Figure 3.** 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 ATmega325/3250/645/6450 provides the following features: 32/64K bytes of In-System Programmable Flash with Read-While-Write capabilities, 1/2K bytes EEPROM, 2/4K byte SRAM, 53/68 general purpose I/O lines, 32 general purpose working registers, a JTAG interface for Boundary-scan, On-chip Debugging support and programming, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, Universal Serial Interface with Start Condition Detector, an 8-channel, 10-bit ADC, a programmable Watchdog Timer with internal Oscillator, an SPI serial port, and five software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset. In Power-save mode, the asynchronous timer will continue to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low-power consumption.

The device is manufactured using Atmel's high density non-volatile memory technology. The On-chip In-System re-Programmable (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 Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega325/3250/645/6450 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega325/3250/645/6450 AVR is supported with a full suite of program and system development tools including: C Compilers, Macro Assemblers, Program Debugger/Simulators, In-Circuit Emulators, and Evaluation kits.

## Comparison between ATmega325, ATmega3250, ATmega645 and ATmega6450

The ATmega325, ATmega3250, ATmega645, and ATmega6450 differs only in memory sizes, pin count and pinout. Table 1 on page 6 summarizes the different configurations for the four devices.

**Table 1.** Configuration Summary

| Device | Flash | EEPROM | RAM | General Purpose I/O Pins |
|---|---|---|---|---|
| ATmega325 | 32K bytes | 1K bytes | 2K bytes | 53 |
| ATmega3250 | 32K bytes | 1K bytes | 2K bytes | 68 |
| ATmega645 | 64K bytes | 2K bytes | 4K bytes | 53 |
| ATmega6450 | 64K bytes | 2K bytes | 4K bytes | 68 |

## Pin Descriptions

The following section describes the I/O-pin special functions.

**$V_{CC}$**

Digital supply voltage.

**GND**

Ground.

**Port A (PA7..PA0)**

Port A is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port A 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.

**Port B (PB7..PB0)**

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.

Port B has better driving capabilities than the other ports.

Port B also serves the functions of various special features of the ATmega325/3250/645/6450 as listed on page 59.

**Port C (PC7..PC0)**

Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C 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.

**Port D (PD7..PD0)**

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. 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.

Port D also serves the functions of various special features of the ATmega325/3250/645/6450 as listed on page 62.

**Port E (PE7..PE0)**

Port E is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port E output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port E pins that are externally pulled low will source

current if the pull-up resistors are activated. The Port E pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port E also serves the functions of various special features of the ATmega325/3250/645/6450 as listed on page 63.

**Port F (PF7..PF0)**

Port F serves as the analog inputs to the A/D Converter.

Port F also serves as an 8-bit bi-directional I/O port, if the A/D Converter is not used. Port pins can provide internal pull-up resistors (selected for each bit). The Port F output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port F pins that are externally pulled low will source current if the pull-up resistors are activated. The Port F pins are tri-stated when a reset condition becomes active, even if the clock is not running. If the JTAG interface is enabled, the pull-up resistors on pins PF7(TDI), PF5(TMS), and PF4(TCK) will be activated even if a reset occurs.

Port F also serves the functions of the JTAG interface.

**Port G (PG4..PG0)**

Port G is a 5-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port G output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port G pins that are externally pulled low will source current if the pull-up resistors are activated. The Port G pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port G also serves the functions of various special features of the ATmega325/3250/645/6450 as listed on page 63.

**Port H (PH7..PH0)**

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

Port H also serves the functions of various special features of the ATmega3250/6450 as listed on page 63.

**Port J (PJ6..PJ0)**

Port J is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port J output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port J pins that are externally pulled low will source current if the pull-up resistors are activated. The Port J pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port J also serves the functions of various special features of the ATmega3250/6450 as listed on page 63.

**$\overline{\text{RESET}}$**

Reset input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running. The minimum pulse length is given in Table 16 on page 39. Shorter pulses are not guaranteed to generate a reset.

**XTAL1**

Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

**XTAL2**

Output from the inverting Oscillator amplifier.

**AVCC**

AVCC is the supply voltage pin for Port F and the A/D Converter. It should be externally connected to $V_{CC}$, even if the ADC is not used. If the ADC is used, it should be connected to $V_{CC}$ through a low-pass filter.

**AREF** This is the analog reference pin for the A/D Converter.

## 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”.

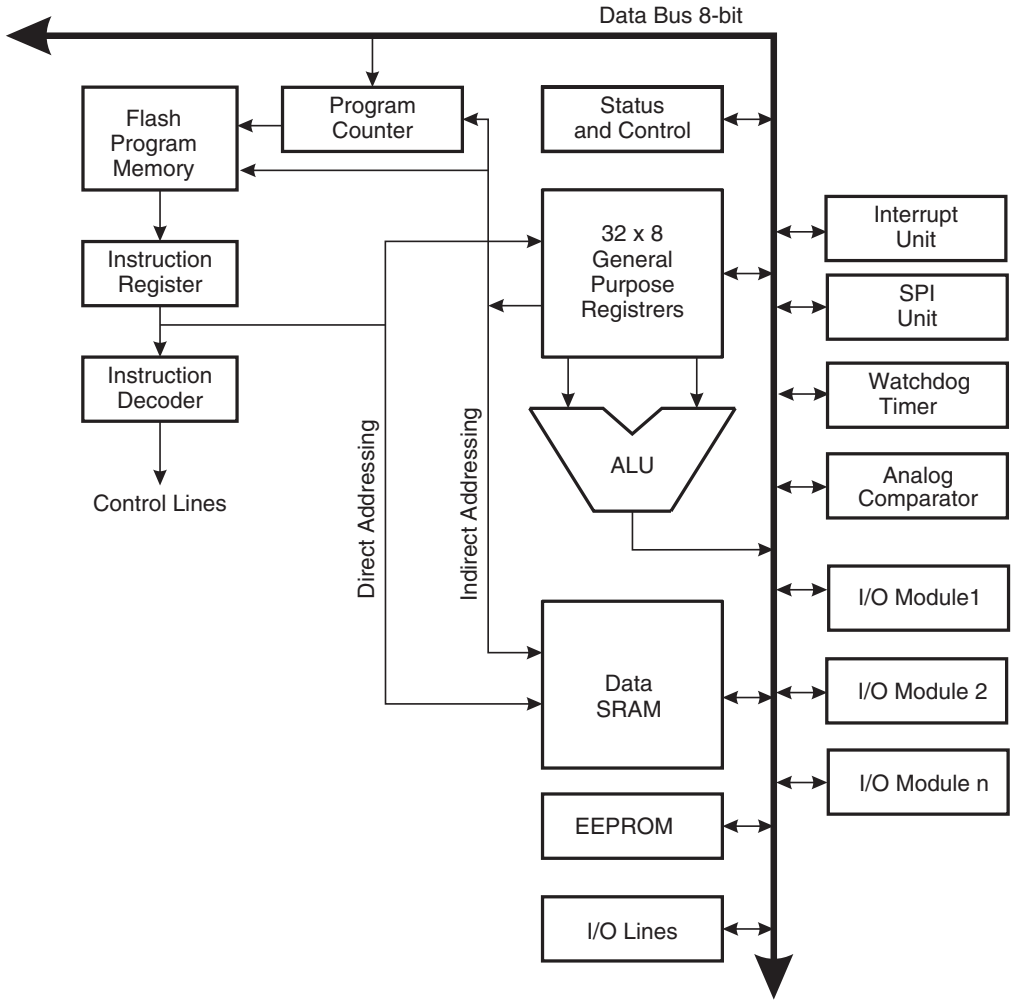
# AVR CPU Core

## 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.

## Architectural Overview

**Figure 4.** 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. Every program memory address contains a 16- or 32-bit instruction.

Program Flash memory space is divided in two sections, the Boot Program section and the Application Program section. Both sections have dedicated Lock bits for write and read/write protection. The SPM instruction that writes into the Application Flash memory section must reside in the Boot Program section.

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 ATmega325/3250/645/6450 has Extended I/O space from 0x60 - 0xFF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

## 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 the “Instruction Set” section for a detailed description.

## 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 $\oplus$ 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.

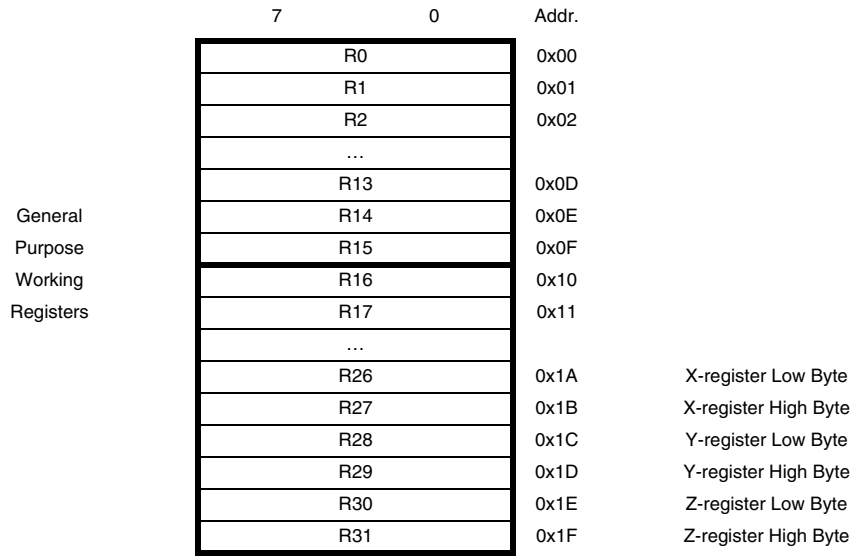
## General Purpose Register File

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 5 shows the structure of the 32 general purpose working registers in the CPU.

**Figure 5.** AVR CPU General Purpose Working Registers



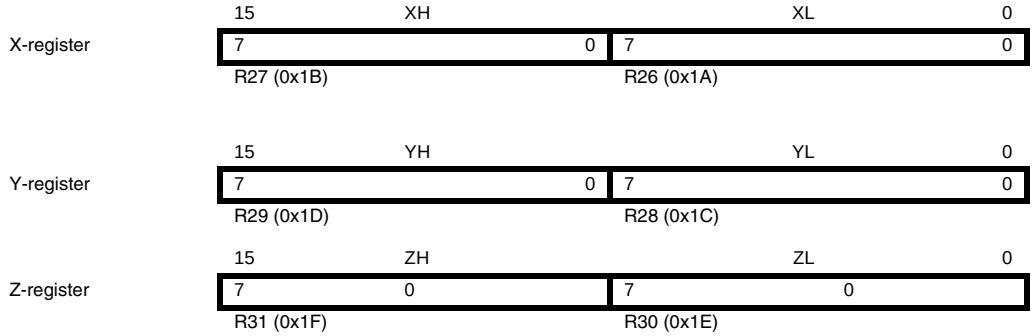

Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions.

As shown in Figure 5, 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.

### 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 6.

**Figure 6.** 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).

## 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 must be set to point above 0x60. 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 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

## 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 7 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 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

**Figure 7.** The Parallel Instruction Fetches and Instruction Executions

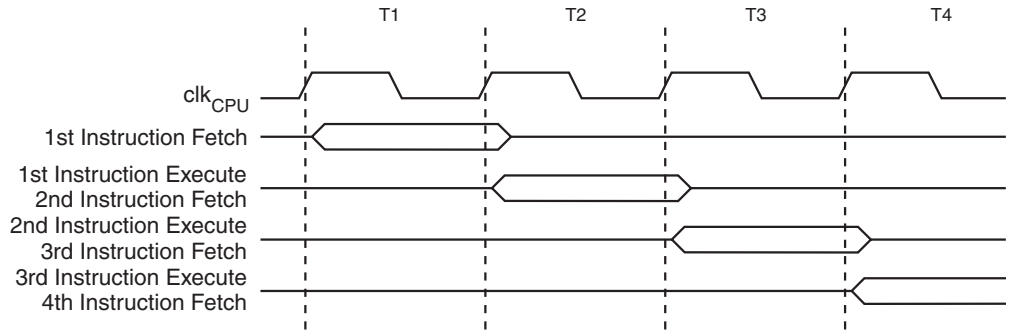
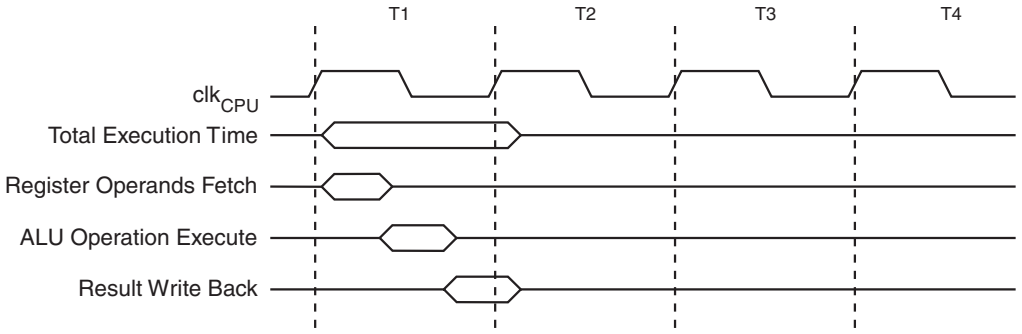



Figure 8 shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

**Figure 8.** Single Cycle ALU Operation




## 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 Boot Lock bits BLB02 or BLB12 are programmed. This feature improves software security. See the section “Memory Programming” on page 257 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 “Interrupts” on page 47. 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. The Interrupt Vectors can be moved to the start of the Boot Flash section by setting the IVSEL bit in the MCU Control Register (MCUCR). Refer to “Interrupts” on page 47 for more information. The Reset Vector can also be

moved to the start of the Boot Flash section by programming the BOOTRST Fuse, see “Boot Loader Support – Read-While-Write Self-Programming” on page 243.

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, EEMWE   ; start EEPROM write
sbi EECR, EEWE
out SREG, r16     ; restore SREG value (I-bit)
```

C Code Example

```
char cSREG;
cSREG = SREG; /* store SREG value */
/* disable interrupts during timed sequence */
__disable_interrupt();
EECR |= (1<<EEMWE); /* start EEPROM write */
EECR |= (1<<EEWE);
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) */
```

**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.

## AVR ATmega325/3250/645/6450 Memories

This section describes the different memories in the ATmega325/3250/645/6450. The AVR architecture has two main memory spaces, the Data Memory and the Program Memory space. In addition, the ATmega325/3250/645/6450 features an EEPROM Memory for data storage. All three memory spaces are linear.

### In-System Reprogrammable Flash Program Memory

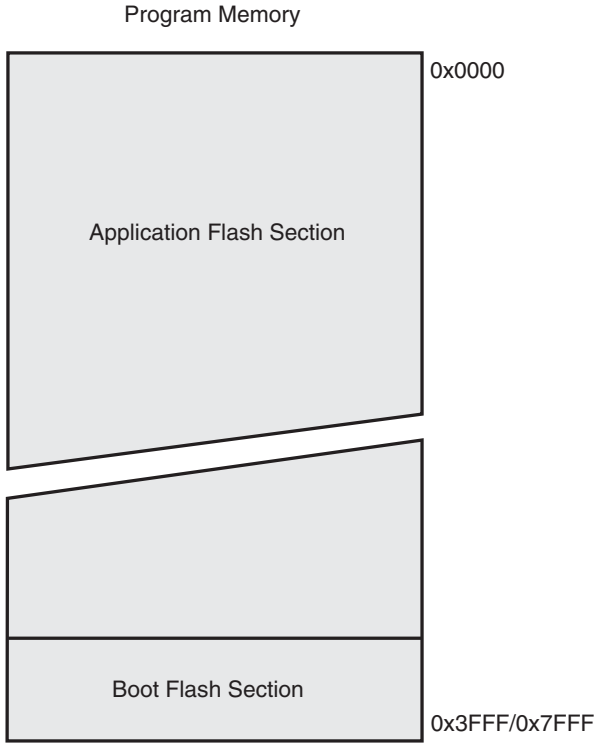
The ATmega325/3250/645/6450 contains 32/64K bytes On-chip In-System Reprogrammable Flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 16/32K x 16. For software security, the Flash Program memory space is divided into two sections, Boot Program section and Application Program section.

The Flash memory has an endurance of at least 10,000 write/erase cycles. The ATmega325/3250/645/6450 Program Counter (PC) is 14/15 bits wide, thus addressing the 16/32K program memory locations. The operation of Boot Program section and associated Boot Lock bits for software protection are described in detail in “Boot Loader Support – Read-While-Write Self-Programming” on page 243. “Memory Programming” on page 257 contains a detailed description on Flash data serial downloading using the SPI pins or the JTAG interface.

Constant tables can be allocated within the entire program memory address space (see the LPM – Load Program Memory instruction description).

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

**Figure 9.** Program Memory Map



## SRAM Data Memory

Figure 10 shows how the ATmega325/3250/645/6450 SRAM Memory is organized.

The ATmega325/3250/645/6450 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 2304/4352 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 2048/4096 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.

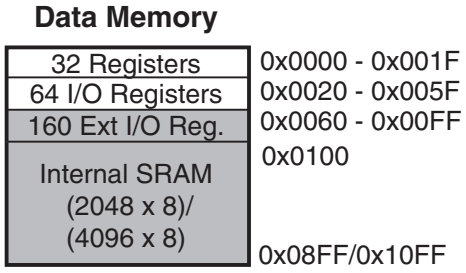
The direct addressing reaches the entire data space.

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 2,048 bytes of internal data SRAM in the ATmega325/3250/645/6450 are all accessible through all these addressing modes. The Register File is described in “General Purpose Register File” on page 12.

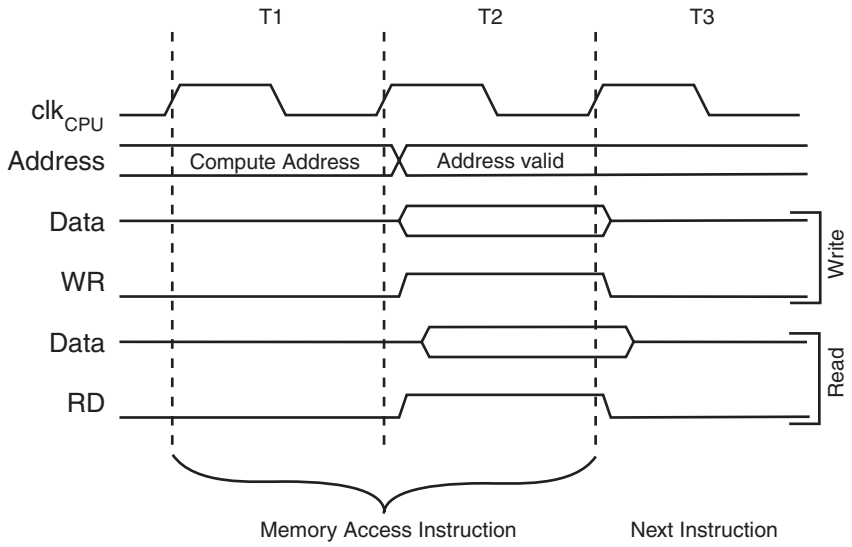
**Figure 10.** Data Memory Map




## 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 11.

**Figure 11.** On-chip Data SRAM Access Cycles




## EEPROM Data Memory

The ATmega325/3250/645/6450 contains 1/2K bytes of data EEPROM memory. It is 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.

For a detailed description of SPI, JTAG and Parallel data downloading to the EEPROM, see page 271, page 275, and page 260 respectively.

### EEPROM Read/Write Access

The EEPROM Access Registers are accessible in the I/O space.

The write access time for the EEPROM is given in Table 2. 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 “Preventing EEPROM Corruption” on page 23. 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 the description of the EEPROM Control Register 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.

### The EEPROM Address Register – EEARH and EEARL

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | – | EEAR10 | EEAR9 | EEAR8 | EEARH |
| | EEAR7 | EEAR6 | EEAR5 | EEAR4 | EEAR3 | EEAR2 | EEAR1 | EEAR0 | EEARL |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Read/Write | R | R | R | R | R | R/W | R/W | R/W | |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | X | X | X | |
| | X | X | X | X | X | X | X | X | |

- **Bits 15..11 – Res: Reserved Bits**

These bits are reserved bits in the ATmega325/3250/645/6450 and will always read as zero.

- **Bits 10..0 – EEAR10..0: EEPROM Address**

The EEPROM Address Registers – EEARH and EEARL specify the EEPROM address in the 1/2K bytes EEPROM space. The EEPROM data bytes are addressed linearly between 0 and 1023/2047. The initial value of EEAR is undefined. A proper value must be written before the EEPROM may be accessed.

Note: EEAR10 is only valid for ATmega645 and ATmega6450.

### The EEPROM Data Register – EEDR

| 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.

### The EEPROM Control Register – EECR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | EERIE | EEMWE | EEWE | EERE | EECR |
| Read/Write | R | R | R | R | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | X | 0 | |

- **Bits 7..4 – Res: Reserved Bits**

These bits are reserved bits in the ATmega325/3250/645/6450 and will always read as zero.

- **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 EEWE is cleared.

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

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

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

The EEPROM Write Enable Signal EEWE is the write strobe to the EEPROM. When address and data are correctly set up, the EEWE bit must be written to one to write the value into the EEPROM. The EEMWE bit must be written to one before a logical one is written to EEWE, 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 EEWE becomes zero.
2. Wait until SPMEN 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 EEMWE bit while writing a zero to EEWE in EECR.
6. Within four clock cycles after setting EEMWE, write a logical one to EEWE.

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. Step 2 is only relevant if the software contains a Boot Loader allowing the CPU to program the Flash. If the Flash is never being updated by the CPU, step 2 can be omitted. See “Boot Loader Support – Read-While-Write Self-Programming” on page 243 for details about Boot programming.

**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 EEWE bit is cleared by hardware. The user software can poll this bit and wait for a zero before writing the next byte. When EEWE 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 EEWE 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 2 lists the typical programming time for EEPROM access from the CPU.

**Table 2.** EEPROM Programming Time

| Symbol | Number of Calibrated RC Oscillator Cycles | Typical Programming Time |
|---|---|---|
| EEPROM write (from CPU) | 27,072 | 3.4 ms |

The following code examples show one assembly and one C function for writing to 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. The examples also assume that no Flash Boot Loader is present in the software. If such code is present, the EEPROM write function must also wait for any ongoing SPM command to finish.

Assembly Code Example

```
EEPROM_write:
  ; Wait for completion of previous write
  sbic EECR,EEWE
  rjmp EEPROM_write
  ; Set up address (r18:r17) in address register
  out  EEARH, r18
  out  EEARL, r17
  ; Write data (r16) to Data Register
  out  EEDR,r16
  ; Write logical one to EEMWE
  sbi  EECR,EEMWE
  ; Start eeprom write by setting EEWE
  sbi  EECR,EEWE
  ret
```

C Code Example

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

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,EEWE
  rjmp EEPROM_read
  ; Set up address (r18:r17) in address register
  out  EEARH, r18
  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<<EEWE))
    ;
  /* Set up address register */
  EEAR = uiAddress;
  /* Start eeprom read by writing EERE */
  EECR |= (1<<EERE);
  /* Return data from Data Register */
  return EEDR;
}
```

**EEPROM Write During Power-down Sleep Mode**

When entering Power-down sleep mode while an EEPROM write operation is active, the EEPROM write operation will continue, and will complete before the Write Access time has passed. However, when the write operation is completed, the clock continues running, and as a consequence, the device does not enter Power-down entirely. It is therefore recommended to verify that the EEPROM write operation is completed before entering Power-down.

**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 following this design recommendation:

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 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.

## I/O Memory

The I/O space definition of the ATmega325/3250/645/6450 is shown in "Register Summary" on page 323.

All ATmega325/3250/645/6450 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 ATmega325/3250/645/6450 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, 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.

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

### General Purpose I/O Registers

The ATmega325/3250/645/6450 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.

### General Purpose I/O Register 2 – GPIOR2

| 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 | |

### General Purpose I/O Register 1 – GPIOR1

| 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 | |

### General Purpose I/O Register 0 – GPIOR0

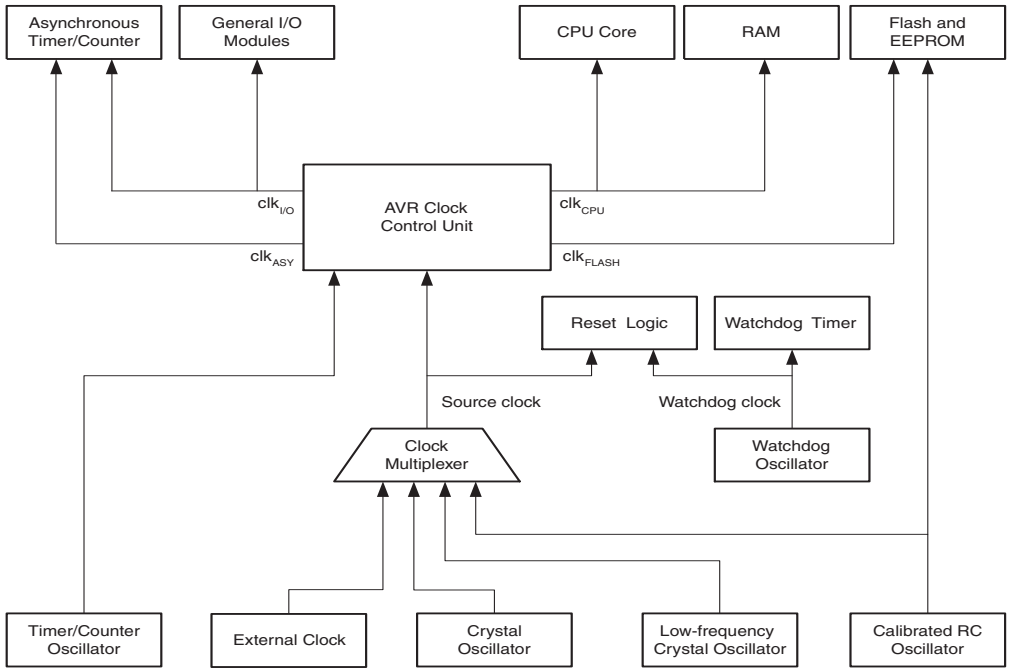
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | MSB | | | | | | | LSB | GPIOR0 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

# System Clock and Clock Options

## Clock Systems and their Distribution

Figure 12 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 “Power Management and Sleep Modes” on page 33. The clock systems are detailed below.

**Figure 12.** Clock Distribution




**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.

**I/O Clock – $clk_{I/O}$**

The I/O clock is used by the majority of the I/O modules, like Timer/Counters, SPI, and USART. The I/O clock is also used by the External Interrupt module, but note that some external interrupts are detected by asynchronous logic, allowing such interrupts to be detected even if the I/O clock is halted. Also note that start condition detection in the USI module is carried out asynchronously when $clk_{I/O}$ is halted, enabling USI start condition detection in all sleep modes.

**Flash Clock – $clk_{FLASH}$**

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

**Asynchronous Timer Clock – $clk_{ASY}$**

The Asynchronous Timer clock allows the Asynchronous Timer/Counter to be clocked directly from an external clock or an external 32 kHz clock crystal. The dedicated clock domain allows using this Timer/Counter as a real-time counter even when the device is in sleep mode.

### ADC 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.

## 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 3.** Device Clocking Options Select(1)

| Device Clocking Option | CKSEL3..0 |
|---|---|
| External Crystal/Ceramic Resonator | 1111 - 1000 |
| External Low-frequency Crystal | 0111 - 0110 |
| Calibrated Internal RC Oscillator | 0010 |
| External Clock | 0000 |
| Reserved | 0011, 0001, 0101, 0100 |

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

The various choices for each clocking option is given in the following sections. When the CPU wakes up from Power-down or Power-save, the selected clock source is used to time the start-up, ensuring stable Oscillator operation before instruction execution starts. When the CPU starts from reset, there is an additional delay allowing the power to reach a stable level before commencing normal operation. The Watchdog Oscillator is used for timing this real-time part of the start-up time. The number of WDT Oscillator cycles used for each time-out is shown in Table 4. The frequency of the Watchdog Oscillator is voltage dependent as shown in “ATmega325/3250/645/6450 Typical Characteristics – Preliminary Data” on page 293.

**Table 4.** Number of Watchdog Oscillator Cycles

| Typ Time-out ($V_{CC}$ = 5.0V) | Typ Time-out ($V_{CC}$ = 3.0V) | Number of Cycles |
|---|---|---|
| 4.1 ms | 4.3 ms | 4K (4,096) |
| 65 ms | 69 ms | 64K (65,536) |

## Default Clock Source

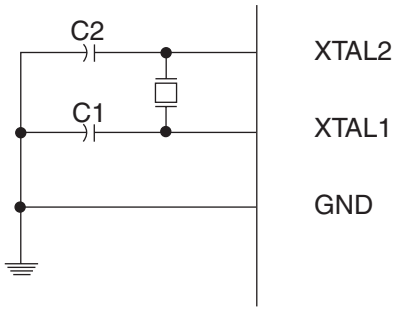
The device is shipped with CKSEL = “0010”, SUT = “10”, and CKDIV8 programmed. The default clock source setting is the Internal RC Oscillator with longest start-up time and an initial system clock prescaling of 8. This default setting ensures that all users can make their desired clock source setting using an In-System or Parallel programmer.

## Crystal Oscillator

XTAL1 and XTAL2 are input and output, respectively, of an inverting amplifier which can be configured for use as an On-chip Oscillator, as shown in Figure 13. Either a quartz crystal or a ceramic resonator may be used.

C1 and C2 should always be equal for both crystals and resonators. The optimal value of the capacitors depends on the crystal or resonator in use, the amount of stray capacitance, and the electromagnetic noise of the environment. Some initial guidelines for choosing capacitors for use with crystals are given in Table 5. For ceramic resonators, the capacitor values given by the manufacturer should be used.

**Figure 13.** Crystal Oscillator Connections




The Oscillator can operate in three different modes, each optimized for a specific frequency range. The operating mode is selected by the fuses CKSEL3..1 as shown in Table 5.

**Table 5.** Crystal Oscillator Operating Modes

| CKSEL3..1 | Frequency Range (MHz) | Recommended Range for Capacitors C1 and C2 for Use with Crystals (pF) |
|---|---|---|
| 100[1] | 0.4 - 0.9 | – |
| 101 | 0.9 - 3.0 | 12 - 22 |
| 110 | 3.0 - 8.0 | 12 - 22 |
| 111 | 8.0 - | 12 - 22 |

Notes: 1. This option should not be used with crystals, only with ceramic resonators.

The CKSEL0 Fuse together with the SUT1..0 Fuses select the start-up times as shown in Table 6.

**Table 6.** Start-up Times for the Crystal Oscillator Clock Selection

| CKSEL0 | SUT1..0 | Start-up Time from Power-down and Power-save | Additional Delay from Reset ($V_{CC}$ = 5.0V) | Recommended Usage |
|---|---|---|---|---|
| 0 | 00 | 258 CK[1] | 14CK + 4.1 ms | Ceramic resonator, fast rising power |
| 0 | 01 | 258 CK[1] | 14CK + 65 ms | Ceramic resonator, slowly rising power |
| 0 | 10 | 1K CK[2] | 14CK | Ceramic resonator, BOD enabled |
| 0 | 11 | 1K CK[2] | 14CK + 4.1 ms | Ceramic resonator, fast rising power |
| 1 | 00 | 1K CK[2] | 14CK + 65 ms | Ceramic resonator, slowly rising power |
| 1 | 01 | 16K CK | 14CK | Crystal Oscillator, BOD enabled |
| 1 | 10 | 16K CK | 14CK + 4.1 ms | Crystal Oscillator, fast rising power |
| 1 | 11 | 16K CK | 14CK + 65 ms | Crystal Oscillator, slowly rising power |

Notes: 1. These options should only be used when not operating close to the maximum frequency of the device, and only if frequency stability at start-up is not important for the application. These options are not suitable for crystals.
2. These options are intended for use with ceramic resonators and will ensure frequency stability at start-up. They can also be used with crystals when not operating close to the maximum frequency of the device, and if frequency stability at start-up is not important for the application.

## Low-frequency Crystal Oscillator

To use a 32.768 kHz watch crystal as the clock source for the device, the low-frequency crystal Oscillator must be selected by setting the CKSEL Fuses to “0110” or “0111”. The crystal should be connected as shown in Figure 13. When this Oscillator is selected, start-up times are determined by the SUT Fuses as shown in Table 7 and CKSEL1..0 as shown in Table 8.

**Table 7.** Start-up Times for the Low-frequency Crystal Oscillator Clock Selection

| SUT1..0 | Additional Delay from Reset ($V_{CC}$ = 5.0V) | Recommended Usage |
|---|---|---|
| 00 | 14CK | Fast rising power or BOD enabled |
| 01 | 14CK + 4.1 ms | Slowly rising power |
| 10 | 14CK + 65 ms | Stable frequency at start-up |
| 11 | Reserved | |

**Table 8.** Start-up Times for the Low-frequency Crystal Oscillator Clock Selection

| CKSEL3..0 | Start-up Time from Power-down and Power-save | Recommended Usage |
|---|---|---|
| 0110(1) | 1K CK | |
| 0111 | 32K CK | Stable frequency at start-up |

Note: 1. This option should only be used if frequency stability at start-up is not important for the application

## Calibrated Internal RC Oscillator

The calibrated internal RC Oscillator by default provides a 8.0 MHz clock. The frequency is nominal value at 3V and 25°C. The device is shipped with the CKDIV8 Fuse programmed. See “System Clock Prescaler” on page 31 for more details. This clock may be selected as the system clock by programming the CKSEL Fuses as shown in Table 9. If selected, it will operate with no external components. During reset, hardware loads the calibration byte into the OSCCAL Register and thereby automatically calibrates the RC Oscillator. At 3V and 25°C, this calibration gives a frequency of 8 MHz ± 1%. The oscillator can be calibrated to any frequency in the range 7.3 - 8.1 MHz within ±1% accuracy, by changing the OSCCAL register. 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 “Calibration Byte” on page 260.

**Table 9.** Internal Calibrated RC Oscillator Operating Modes(1)(3)

| Frequency Range(2) (MHz) | CKSEL3..0 |
|---|---|
| 7.3 - 8.1 | 0010 |

Notes: 1. The device is shipped with this option selected.
2. The frequency ranges are preliminary values. Actual values are TBD.
3. If 8 MHz 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 Table 10 on page 29.

**Table 10.** Start-up times for the internal calibrated RC Oscillator clock selection

| Power Conditions | Start-up Time from Power-down and Power-save | Additional Delay from Reset ($V_{CC}$ = 5.0V) | SUT1..0 |
|---|---|---|---|
| BOD enabled | 6 CK | 14CK | 00 |
| Fast rising power | 6 CK | 14CK + 4.1 ms | 01 |
| Slowly rising power | 6 CK | 14CK + 65 ms(1) | 10 |
| Reserved | | | 11 |

Note: 1. The device is shipped with this option selected.

### Oscillator Calibration Register – OSCCAL

| 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 | | | | | | | | |

- **Bits 7..0 – CAL7..0: Oscillator Calibration Value**

The Oscillator Calibration Register is used to trim the Calibrated Internal RC Oscillator to remove process variations from the oscillator frequency. The factory-calibrated value is automatically written to this register during chip reset, giving an oscillator frequency of 8.0 MHz at 25°C. The application software can write this register to change the oscillator frequency. The oscillator can be calibrated to any frequency in the range 7.3 - 8.1 MHz within ±1% accuracy. Calibration outside that range is not guaranteed.

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

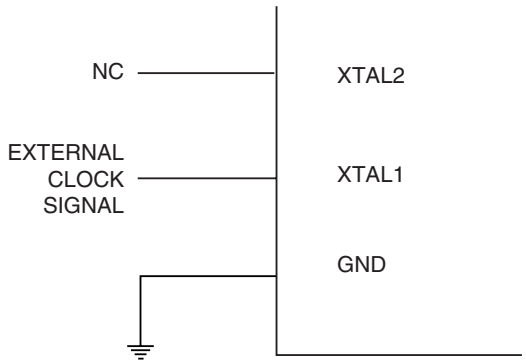
The CAL7 bit determines the range of operation for the oscillator. Setting this bit to 0 gives the lowest frequency range, setting this bit to 1 gives the highest frequency range. The two frequency ranges are overlapping, in other words a setting of OSCCAL = 0x7F gives a higher frequency than OSCCAL = 0x80.

The CAL6..0 bits are used to tune the frequency within the selected range. A setting of 0x00 gives the lowest frequency in that range, and a setting of 0x7F gives the highest frequency in the range. Incrementing CAL6..0 by 1 will give a frequency increment of less than 2% in the frequency range 7.3 - 8.1 MHz.

## External Clock

To drive the device from an external clock source, XTAL1 should be driven as shown in Figure 14. To run the device on an external clock, the CKSEL Fuses must be programmed to “0000”.

**Figure 14.** External Clock Drive Configuration




When this clock source is selected, start-up times are determined by the SUT Fuses as shown in Table 12.

**Table 11.** Crystal Oscillator Clock Frequency

| CKSEL3..0 | Frequency Range |
|---|---|
| 0000 | 0 - 16 MHz |

**Table 12.** Start-up Times for the External Clock Selection

| SUT1..0 | Start-up Time from Power-down and Power-save | Additional Delay from Reset ($V_{CC}$ = 5.0V) | Recommended Usage |
|---|---|---|---|
| 00 | 6 CK | 14CK | BOD enabled |
| 01 | 6 CK | 14CK + 4.1 ms | Fast rising power |
| 10 | 6 CK | 14CK + 65 ms | Slowly rising power |
| 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. It is required to ensure that the MCU is kept in Reset during such changes in the clock frequency.

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 “System Clock Prescaler” on page 31 for details.

## Clock Output Buffer

When the CKOUT Fuse is programmed, the system Clock will be output on CLKO. This mode is suitable when the chip clock is used to drive other circuits on the system. The clock will be output also during reset and the normal operation of I/O pin will be overridden when the fuse is programmed. Any clock source, including internal RC Oscillator, can be selected when CLKO serves as clock output. If the System Clock Prescaler is used, it is the divided system clock that is output when the CKOUT Fuse is programmed.

## Timer/Counter Oscillator

ATmega325/3250/645/6450 share the Timer/Counter Oscillator Pins (TOSC1 and TOSC2) with XTAL1 and XTAL2. This means that the Timer/Counter Oscillator can only be used when the calibrated internal RC Oscillator is selected as system clock source.

The Oscillator is optimized for use with a 32.768 kHz watch crystal. See Figure 13 on page 27 for crystal connection.

Applying an external clock source to TOSC1 can be done if EXTCLK in the ASSR Register is written to logic one. See “Asynchronous operation of the Timer/Counter” on page 138 for further description on selecting external clock as input instead of a 32 kHz crystal.

## System Clock Prescaler

The ATmega325/3250/645/6450 system clock can be divided by setting the Clock Prescale Register – CLKPR. This feature can be used to decrease 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 13.

### Clock Prescale Register – CLKPR

| 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 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 13.

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

1. Write the Clock Prescaler Change Enable (CLKPCE) bit to one and all other bits in 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.

The CKDIV8 Fuse determines the initial value of the CLKPS bits. If CKDIV8 is unprogrammed, the CLKPS bits will be reset to “0000”. If CKDIV8 is programmed, CLKPS bits are reset to “0011”, 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 13.** 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 |

### Switching Time

When switching between prescaler settings, the System Clock Prescaler ensures that no glitches occur in the clock system and 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 another 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.

## 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.

To enter any of the five sleep modes, the SE bit in SMCR must be written to logic one and a SLEEP instruction must be executed. The SM2, SM1, and SM0 bits in the SMCR Register select which sleep mode (Idle, ADC Noise Reduction, Power-down, Power-save, or Standby) will be activated by the SLEEP instruction. See Table 14 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.

Figure 12 on page 25 presents the different clock systems in the ATmega325/3250/645/6450, and their distribution. The figure is helpful in selecting an appropriate sleep mode.

### Sleep Mode Control Register – SMCR

The Sleep Mode Control Register contains control bits for power management.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | SM2 | SM1 | SM0 | SE | SMCR |
| 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 3, 2, 1 – SM2..0: Sleep Mode Select Bits 2, 1, and 0**

These bits select between the five available sleep modes as shown in Table 14.

**Table 14.** Sleep Mode Select

| SM2 | SM1 | SM0 | Sleep Mode |
|---|---|---|---|
| 0 | 0 | 0 | Idle |
| 0 | 0 | 1 | ADC Noise Reduction |
| 0 | 1 | 0 | Power-down |
| 0 | 1 | 1 | Power-save |
| 1 | 0 | 0 | Reserved |
| 1 | 0 | 1 | Reserved |
| 1 | 1 | 0 | Standby(1) |
| 1 | 1 | 1 | Reserved |

Note: 1. Standby mode is only recommended for use with external crystals or resonators.

- **Bit 1 – 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.

### Idle Mode

When the SM2..0 bits are written to 000, the SLEEP instruction makes the MCU enter Idle mode, stopping the CPU but allowing the SPI, USART, Analog Comparator, ADC, USI, 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 Timer Overflow and USART Transmit Complete 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.

### ADC Noise Reduction Mode

When the SM2..0 bits are written to 001, the SLEEP instruction makes the MCU enter ADC Noise Reduction mode, stopping the CPU but allowing the ADC, the external interrupts, the USI start condition detection, Timer/Counter2, 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 form the ADC Conversion Complete interrupt, only an External Reset, a Watchdog Reset, a Brown-out Reset, a USI start condition interrupt, a Timer/Counter2 interrupt, an SPM/EEPROM ready interrupt, an external level interrupt on INT0 or a pin change interrupt can wake up the MCU from ADC Noise Reduction mode.

### Power-down Mode

When the SM2..0 bits are written to 010, the SLEEP instruction makes the MCU enter Power-down mode. In this mode, the external Oscillator is stopped, while the external interrupts, the USI start condition detection, and the Watchdog continue operating (if enabled). Only an External Reset, a Watchdog Reset, a Brown-out Reset, USI start condition interrupt, an external level interrupt on INT0, 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 “External Interrupts” on page 76 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 “Clock Sources” on page 26.

### Power-save Mode

When the SM2..0 bits are written to 011, the SLEEP instruction makes the MCU enter Power-save mode. This mode is identical to Power-down, with one exception:

If Timer/Counter2 controller is enabled, it will keep running during sleep. The device can wake up from either Timer Overflow or Output Compare event from Timer/Counter2 if the corresponding Timer/Counter2 interrupt enable bits are set in TIMSK2, and the Global Interrupt Enable bit in SREG is set.

If the Timer/Counter2 is not running, Power-down mode is recommended instead of Power-save mode.

The Timer/Counter2 can be clocked both synchronously and asynchronously in Power-save mode. If Timer/Counter2 is not using the asynchronous clock, the Timer/Counter

Oscillator is stopped during sleep. If Timer/Counter2 is not using the synchronous clock, the clock source is stopped during sleep. Note that even if the synchronous clock is running in Power-save, this clock is only available for Timer/Counter2.

## Standby Mode

When the SM2..0 bits are 110 and an external crystal/resonator clock option is selected, the SLEEP instruction makes the MCU enter Standby mode. This mode is identical to Power-down with the exception that the Oscillator is kept running. From Standby mode, the device wakes up in six clock cycles.

**Table 15.** Active Clock Domains and Wake-up Sources in the Different Sleep Modes.

| | Active Clock Domains | | | | | Oscillators | | Wake-up Sources | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Sleep Mode** | **$clk_{CPU}$** | **$clk_{FLASH}$** | **$clk_{IO}$** | **$clk_{ADC}$** | **$clk_{ASY}$** | **Main Clock Source Enabled** | **Timer Osc Enabled** | **INT0 and Pin Change** | **USI Start Condition** | **Timer2** | **SPM/EEPROM Ready** | **ADC** | **Other I/O** |
| Idle | | | X | X | X | X | X(2) | X | X | X | X | X | X |
| ADC Noise Reduction | | | | X | X | X | X(2) | X(3) | X | X(2) | X | X | |
| Power-down | | | | | | | | X(3) | X | | | | |
| Power-save | | | | | X | | X(2) | X(3) | X | X | | | |
| Standby(1) | | | | | | X | | X(3) | X | | | | |

Notes: 1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is not running in asynchronous mode.
3. For INT0, only level interrupt.

## 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 inaccessible. Resources used by the peripheral when stopping the clock will remain occupied so the peripheral should 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 reduce the overall power consumption. In all other sleep modes, the clock is already stopped.

### Power Reduction Register - PRR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | PRTIM1 | PRSPI | PRUSART0 | PRADC | PRR |
| 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 bits in ATmega325/3250/645/6450 and will always read as zero.

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

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

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

Writing 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 1 - PRUSART: Power Reduction USART**

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

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

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

Note: The Analog Comparator is disabled using the ACD-bit in the "Analog Comparator Control and Status Register – ACSR" on page 190.

## Minimizing Power Consumption

There are several possibilities 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.

**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 "Analog to Digital Converter" on page 194 for details on ADC operation.

**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 "Analog Comparator" on page 190 for details on how to configure the Analog Comparator.

**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 "Brown-out Detection" on page 40 for details on how to configure the Brown-out Detector.

**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 "Internal Voltage Reference" on page 43 for details on the start-up time.

**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 “Watchdog Timer” on page 43 for details on how to configure the Watchdog Timer.

**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 “Digital Input Enable and Sleep Modes” on page 56 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 “Digital Input Disable Register 1 – DIDR1” on page 193 and “Digital Input Disable Register 0 – DIDR0” on page 210 for details.

**JTAG Interface and On-chip Debug System**

If the On-chip debug system is enabled by the OCDEN Fuse and the chip enter Power down or Power save sleep mode, the main clock source remains enabled. In these sleep modes, this will contribute significantly to the total current consumption. There are three alternative ways to avoid this:

- Disable OCDEN Fuse.
- Disable JTAGEN Fuse.
- Write one to the JTD bit in MCUCSR.

The TDO pin is left floating when the JTAG interface is enabled while the JTAG TAP controller is not shifting data. If the hardware connected to the TDO pin does not pull up the logic level, power consumption will increase. Note that the TDI pin for the next device in the scan chain contains a pull-up that avoids this problem. Writing the JTD bit in the MCUCSR register to one or leaving the JTAG fuse unprogrammed disables the JTAG interface.

# System Control and Reset

## 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 a JMP – Absolute 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. This is also the case if the Reset Vector is in the Application section while the Interrupt Vectors are in the Boot section or vice versa. The circuit diagram in Figure 15 shows the reset logic. Table 16 defines the electrical parameters of the reset circuitry.

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 “Clock Sources” on page 26.

## Reset Sources

The ATmega325/3250/645/6450 has five sources of reset:

- Power-on Reset. The MCU is reset when the supply voltage is below the Power-on Reset threshold ($V_{POT}$).
- External Reset. The MCU is reset when a low level is present on the $\overline{\text{RESET}}$ pin for longer than the minimum pulse length.
- Watchdog Reset. The MCU is reset when the Watchdog Timer period expires and the Watchdog 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.
- JTAG AVR Reset. The MCU is reset as long as there is a logic one in the Reset Register, one of the scan chains of the JTAG system. Refer to the section “IEEE 1149.1 (JTAG) Boundary-scan” on page 217 for details.

**Figure 15.** Reset Logic

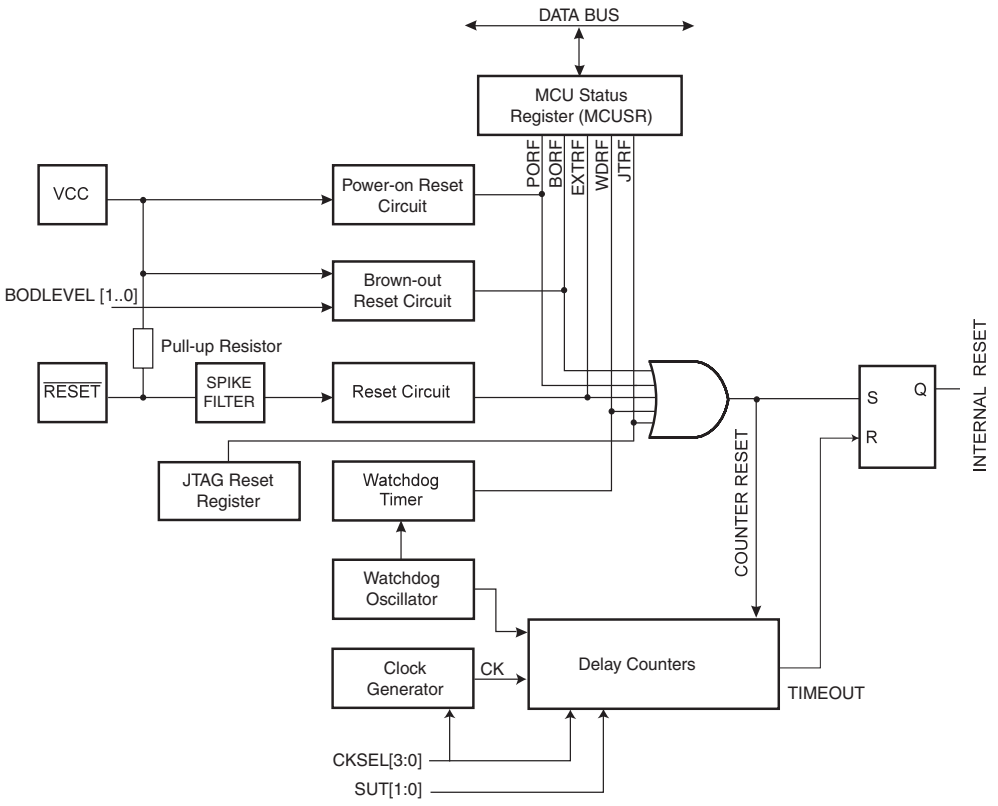



**Table 16.** Reset Characteristics

| Symbol | Parameter | Condition | Min | Typ | Max | Units |
|---|---|---|---|---|---|---|
| $V_{POT}$ | Power-on Reset Threshold Voltage (rising) | $T_A$ = -40°C to 85°C | 0.7 | 1.0 | 1.4 | V |
| | Power-on Reset Threshold Voltage (falling)(1) | $T_A$ = -40°C to 85°C | 0.6 | 0.9 | 1.3 | V |
| $V_{RST}$ | $\overline{\text{RESET}}$ Pin Threshold Voltage | $V_{CC}$ = 3V | $0.2V_{CC}$ | | $0.85V_{CC}$ | V |
| $t_{RST}$ | Minimum pulse width on $\overline{\text{RESET}}$ Pin | $V_{CC}$ = 3V | | 800 | | ns |

Notes: 1. The Power-on Reset will not work unless the supply voltage has been below $V_{POT}$ (falling)

**Power-on Reset**

A Power-on Reset (POR) pulse is generated by an On-chip detection circuit. The detection level is defined in Table 16. 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 16.** MCU Start-up, $\overline{\text{RESET}}$ Tied to $V_{CC}$

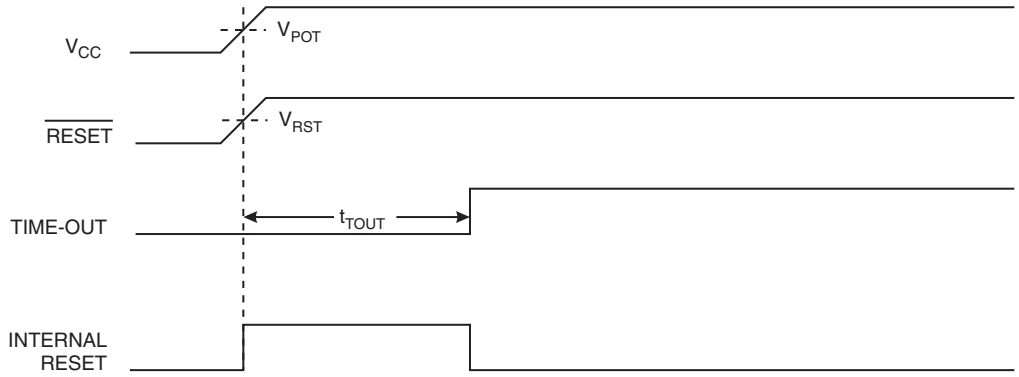
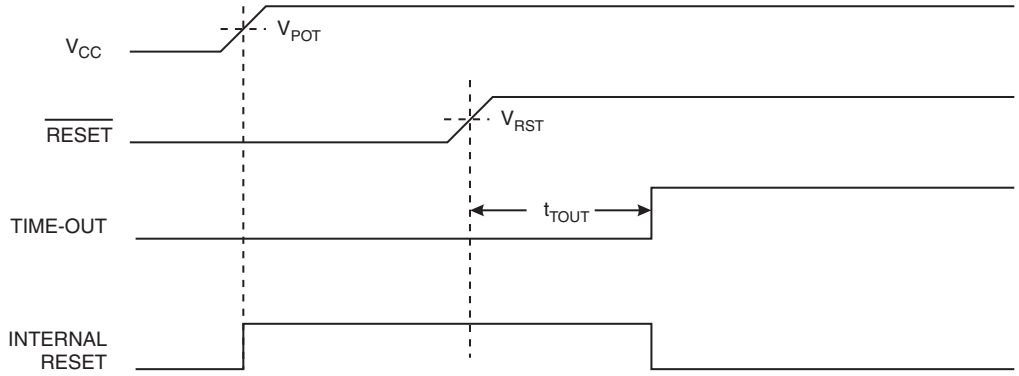


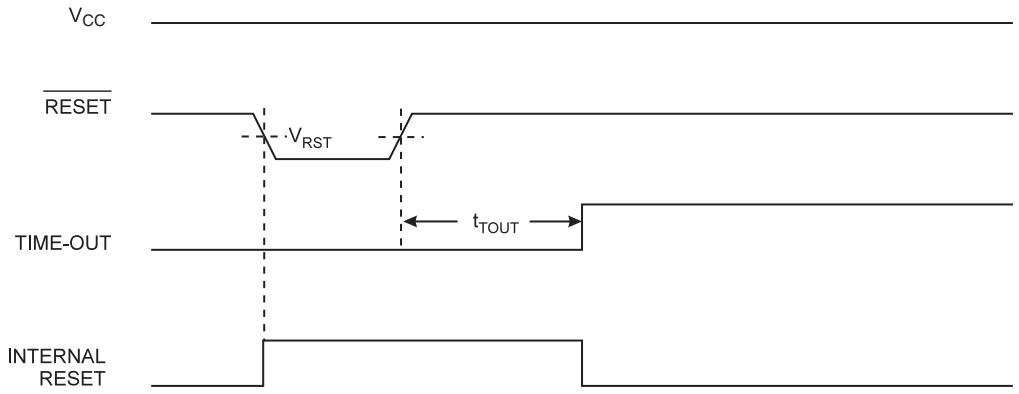

**Figure 17.** MCU Start-up, $\overline{\text{RESET}}$ Extended Externally




**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 16) 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.

**Figure 18.** External Reset During Operation




**Brown-out Detection**

ATmega325/3250/645/6450 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 hys-

teresis 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 $V_{BOT-} = V_{BOT} - V_{HYST}/2$.

**Table 17.** BODLEVEL Fuse Coding(1)

| BODLEVEL 2..0 Fuses | Min $V_{BOT}$ | Typ $V_{BOT}$ | Max $V_{BOT}$ | Units |
|---|---|---|---|---|
| 11 | BOD Disabled | | | |
| 10 | | 1.8 | | V |
| 01 | | 2.7 | | |
| 00 | | 4.3 | | |

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. The test is performed using BODLEVEL = 10 for ATmega325/3250/645/6450V and BODLEVEL = 01 for ATmega325/3250/645/6450L.

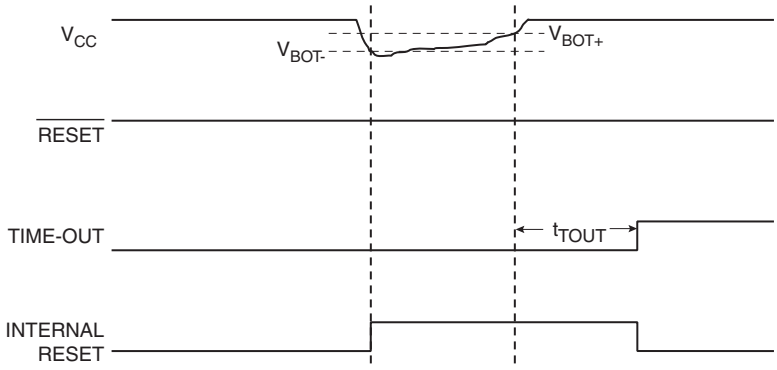
**Table 18.** Brown-out Characteristics

| Symbol | Parameter | Min | Typ | Max | Units |
|---|---|---|---|---|---|
| $V_{HYST}$ | Brown-out Detector Hysteresis | | 50 | | mV |
| $t_{BOD}$ | Min Pulse Width on Brown-out Reset | | 2 | | µs |

When the BOD is enabled, and $V_{CC}$ decreases to a value below the trigger level ($V_{BOT-}$ in Figure 19), the Brown-out Reset is immediately activated. When $V_{CC}$ increases above the trigger level ($V_{BOT+}$ in Figure 19), 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 Table 16.

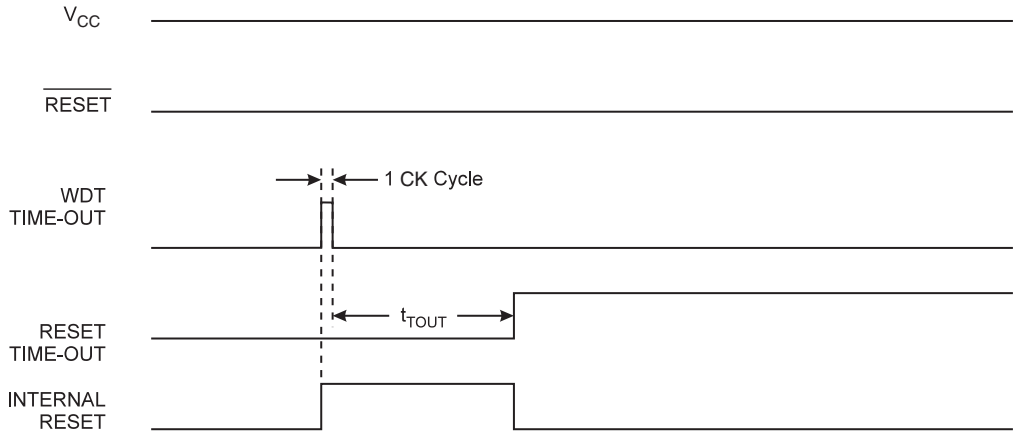
**Figure 19.** Brown-out Reset During Operation




**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 page 43 for details on operation of the Watchdog Timer.

**Figure 20.** Watchdog Reset During Operation




### MCU Status Register – MCUSR

The MCU Status Register provides information on which reset source caused an MCU reset.

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

- **Bit 4 – JTRF: JTAG Reset Flag**

This bit is set if a reset is being caused by a logic one in the JTAG Reset Register selected by the JTAG instruction AVR_RESET. This bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

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

This bit is set if a Watchdog 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.

## Internal Voltage Reference

ATmega325/3250/645/6450 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.

### 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 Table 19. 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 [1..0] Fuse).
2. When the bandgap 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.

**Table 19.** Internal Voltage Reference Characteristics(1)

| Symbol | Parameter | Condition | Min | Typ | Max | Units |
|---|---|---|---|---|---|---|
| $V_{BG}$ | Bandgap reference voltage | $V_{CC}$ = 2.7V, $T_A$ = 25°C | 1.0 | 1.1 | 1.2 | V |
| $t_{BG}$ | Bandgap reference start-up time | $V_{CC}$ = 2.7V, $T_A$ = 25°C | | 40 | 70 | µs |
| $I_{BG}$ | Bandgap reference current consumption | $V_{CC}$ = 2.7V, $T_A$ = 25°C | | 15 | | µA |

Note: 1. Values are guidelines only. Actual values are TBD.

## Watchdog Timer

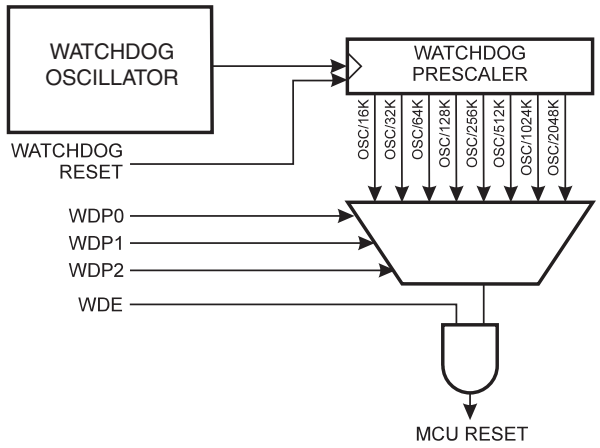
The Watchdog Timer is clocked from a separate On-chip Oscillator which runs at 1 MHz. This is the typical value at $V_{CC}$ = 5V. See characterization data for typical values at other $V_{CC}$ levels. By controlling the Watchdog Timer prescaler, the Watchdog Reset interval can be adjusted as shown in Table 21 on page 45. 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. Eight different clock cycle periods can be selected to determine the reset period. If the reset period expires without another Watchdog Reset, the ATmega325/3250/645/6450 resets and executes from the Reset Vector. For timing details on the Watchdog Reset, refer to Table 21 on page 45.

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 20. Refer to “Timed Sequences for Changing the Configuration of the Watchdog Timer” on page 46 for details.

**Table 20.** WDT Configuration as a Function of the Fuse Settings of WDTON

| WDTON | Safety Level | WDT Initial State | How to Disable the WDT | How to Change Time-out |
|---|---|---|---|---|
| Unprogrammed | 1 | Disabled | Timed sequence | Timed sequence |
| Programmed | 2 | Enabled | Always enabled | Timed sequence |

**Figure 21.** Watchdog Timer




### Watchdog Timer Control Register – WDTCR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | WDCE | WDE | WDP2 | WDP1 | WDP0 | WDTCR |
| 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 | |

- **Bits 7..5 – Res: Reserved Bits**

These bits are reserved bits in the ATmega325/3250/645/6450 and will always read as zero.

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

This bit must be set when the WDE bit is written to logic zero. Otherwise, the Watchdog will not be disabled. Once written to one, hardware will clear this bit after four clock cycles. Refer to the description of the WDE bit for a Watchdog disable procedure. This bit must also be set when changing the prescaler bits. See “Timed Sequences for Changing the Configuration of the Watchdog Timer” on page 46.

- **Bit 3 – WDE: Watchdog Enable**

When the WDE is written to logic one, the Watchdog Timer is enabled, and if the WDE is written to logic zero, the Watchdog Timer function is disabled. WDE can only be cleared if the WDCE bit has logic level one. 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 even though it is set to one before the disable operation starts.
2. Within the next four clock cycles, write a logic 0 to WDE. This disables the Watchdog.

In safety level 2, it is not possible to disable the Watchdog Timer, even with the algorithm described above. See “Timed Sequences for Changing the Configuration of the Watchdog Timer” on page 46.

- **Bits 2..0 – WDP2, WDP1, WDP0: Watchdog Timer Prescaler 2, 1, and 0**

The WDP2, WDP1, and WDP0 bits determine the Watchdog Timer prescaling when the Watchdog Timer is enabled. The different prescaling values and their corresponding Time-out Periods are shown in Table 21 on page 45.

**Table 21.** Watchdog Timer Prescale Select

| WDP2 | WDP1 | WDP0 | Number of WDT Oscillator Cycles | Typical Time-out at $V_{CC}$ = 3.0V | Typical Time-out at $V_{CC}$ = 5.0V |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 16K cycles | 17.1 ms | 16.3 ms |
| 0 | 0 | 1 | 32K cycles | 34.3 ms | 32.5 ms |
| 0 | 1 | 0 | 64K cycles | 68.5 ms | 65 ms |
| 0 | 1 | 1 | 128K cycles | 0.14 s | 0.13 s |
| 1 | 0 | 0 | 256K cycles | 0.27 s | 0.26 s |
| 1 | 0 | 1 | 512K cycles | 0.55 s | 0.52 s |
| 1 | 1 | 0 | 1,024K cycles | 1.1 s | 1.0 s |
| 1 | 1 | 1 | 2,048K cycles | 2.2 s | 2.1 s |

The following code example shows one assembly and one C function for turning off the WDT. The example assumes that interrupts are controlled (e.g. by disabling interrupts globally) so that no interrupts will occur during execution of these functions.

Assembly Code Example[1]

```
WDT_off:
  ; Reset WDT
  wdr
  ; Write logical one to WDCE and WDE
  in   r16, WDTCR
  ori  r16, (1<<WDCE)|(1<<WDE)
  out  WDTCR, r16
  ; Turn off WDT
  ldi  r16, (0<<WDE)
  out  WDTCR, r16
  ret
```

C Code Example[1]

```
void WDT_off(void)
{
  /* Reset WDT */
  __watchdog_reset();
  /* Write logical one to WDCE and WDE */
  WDTCR |= (1<<WDCE) | (1<<WDE);
  /* Turn off WDT */
  WDTCR = 0x00;
}
```

Note: 1. See “About Code Examples” on page 8.

## 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.

### Safety Level 1

In this mode, the Watchdog Timer is initially disabled, but can be enabled by writing the WDE bit to 1 without any restriction. A timed sequence is needed when changing the Watchdog Time-out period or disabling an enabled Watchdog Timer. To disable an enabled Watchdog Timer, and/or changing the Watchdog Time-out, 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.

### 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.

Within the next four clock cycles, in the same operation, write the WDP bits as desired, but with the WDCE bit cleared. The value written to the WDE bit is irrelevant.

## Interrupts

This section describes the specifics of the interrupt handling as performed in ATmega325/3250/645/6450. For a general explanation of the AVR interrupt handling, refer to “Reset and Interrupt Handling” on page 14.

### Interrupt Vectors in ATmega325/3250/645/6450

**Table 22.** Reset and Interrupt Vectors

| Vector No. | Program Address(2) | Source | Interrupt Definition |
|---|---|---|---|
| 1 | 0x0000(1) | RESET | External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset |
| 2 | 0x0002 | INT0 | External Interrupt Request 0 |
| 3 | 0x0004 | PCINT0 | Pin Change Interrupt Request 0 |
| 4 | 0x0006 | PCINT1 | Pin Change Interrupt Request 1 |
| 5 | 0x0008 | TIMER2 COMP | Timer/Counter2 Compare Match |
| 6 | 0x000A | TIMER2 OVF | Timer/Counter2 Overflow |
| 7 | 0x000C | TIMER1 CAPT | Timer/Counter1 Capture Event |
| 8 | 0x000E | TIMER1 COMPA | Timer/Counter1 Compare Match A |
| 9 | 0x0010 | TIMER1 COMPB | Timer/Counter1 Compare Match B |
| 10 | 0x0012 | TIMER1 OVF | Timer/Counter1 Overflow |
| 11 | 0x0014 | TIMER0 COMP | Timer/Counter0 Compare Match |
| 12 | 0x0016 | TIMER0 OVF | Timer/Counter0 Overflow |
| 13 | 0x0018 | SPI, STC | SPI Serial Transfer Complete |
| 14 | 0x001A | USART, RX | USART, Rx Complete |
| 15 | 0x001C | USART, UDRE | USART Data Register Empty |
| 16 | 0x001E | USART, TX | USART, Tx Complete |
| 17 | 0x0020 | USI START | USI Start Condition |
| 18 | 0x0022 | USI OVERFLOW | USI Overflow |
| 19 | 0x0024 | ANALOG COMP | Analog Comparator |
| 20 | 0x0026 | ADC | ADC Conversion Complete |
| 21 | 0x0028 | EE READY | EEPROM Ready |
| 22 | 0x002A | SPM READY | Store Program Memory Ready |
| 23 | 0x002C | NOT_USED | RESERVED |
| 24(3) | 0x002E | PCINT2 | Pin Change Interrupt Request 2 |
| 25(3) | 0x0030 | PCINT3 | Pin Change Interrupt Request 3 |

Notes:
1. When the BOOTRST Fuse is programmed, the device will jump to the Boot Loader address at reset, see “Boot Loader Support – Read-While-Write Self-Programming” on page 243.
2. When the IVSEL bit in MCUCR is set, Interrupt Vectors will be moved to the start of the Boot Flash Section. The address of each Interrupt Vector will then be the address in this table added to the start address of the Boot Flash Section.
3. PCINT2 and PCINT3 are only present in ATmega3250 and ATmega6450.

Table 23 shows reset and Interrupt Vectors placement for the various combinations of BOOTRST and IVSEL settings. If the program never enables an interrupt source, the Interrupt Vectors are not used, and regular program code can be placed at these locations. This is also the case if the Reset Vector is in the Application section while the Interrupt Vectors are in the Boot section or vice versa.

**Table 23.** Reset and Interrupt Vectors Placement[1]

| BOOTRST | IVSEL | Reset Address | Interrupt Vectors Start Address |
|---|---|---|---|
| 1 | 0 | 0x0000 | 0x0002 |
| 1 | 1 | 0x0000 | Boot Reset Address + 0x0002 |
| 0 | 0 | Boot Reset Address | 0x0002 |
| 0 | 1 | Boot Reset Address | Boot Reset Address + 0x0002 |

Note: 1. The Boot Reset Address is shown in Table 104 on page 255. For the BOOTRST Fuse “1” means unprogrammed while “0” means programmed.

The most typical and general program setup for the Reset and Interrupt Vector Addresses in ATmega325/3250/645/6450 is:

```
Address   Labels   Code                        Comments
0x0000             jmp     RESET               ; Reset Handler
0x0002             jmp     EXT_INT0            ; IRQ0 Handler
0x0004             jmp     PCINT0              ; PCINT0 Handler
0x0006             jmp     PCINT1              ; PCINT1 Handler
0x0008             jmp     TIM2_COMP           ; Timer2 Compare Handler
0x000A             jmp     TIM2_OVF            ; Timer2 Overflow Handler
0x000C             jmp     TIM1_CAPT           ; Timer1 Capture Handler
0x000E             jmp     TIM1_COMPA          ; Timer1 CompareA Handler
0x0010             jmp     TIM1_COMPB          ; Timer1 CompareB Handler
0x0012             jmp     TIM1_OVF            ; Timer1 Overflow Handler
0x0014             jmp     TIM0_COMP           ; Timer0 Compare Handler
0x0016             jmp     TIM0_OVF            ; Timer0 Overflow Handler
0X0018             jmp     SPI_STC             ; SPI Transfer Complete Handler
0x001A             jmp     USART_RXC           ; USART RX Complete Handler
0x001C             jmp     USART_UDRE          ; USART,UDR Empty Handler
0x001E             jmp     USART_TXC           ; USART TX Complete Handler
0x0020             jmp     USI_STRT            ; USI Start Condition Handler
0x0022             jmp     USI_OVF             ; USI Overflow Handler
0x0024             jmp     ANA_COMP            ; Analog Comparator Handler
0x0026             jmp     ADC                 ; ADC Conversion Complete Handler
0x0028             jmp     EE_RDY              ; EEPROM Ready Handler
0x002A             jmp     SPM_RDY             ; SPM Ready Handler
0x002C                     ;NOT_USED           ; RESERVED
0x002E             jmp     PCINT2              ; PCINT2 Handler
0x0030             jmp     PCINT3              ; PCINT3 Handler
;
0x0032    RESET:   ldi     r16, high(RAMEND)   ; Main program start
0x0033             out     SPH,r16             ; Set Stack Pointer to top of RAM
0x0034             ldi     r16, low(RAMEND)
0x0035             out     SPL,r16
0x0036             sei                         ; Enable interrupts
0x0037             <instr> xxx
            ...      ...     ...
```

When the BOOTRST Fuse is unprogrammed, the Boot section size set to 4K bytes and the IVSEL bit in the MCUCR Register is set before any interrupts are enabled, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address  Labels Code                 Comments
0x0000   RESET: ldi    r16,high(RAMEND); Main program start
0x0001          out    SPH,r16        ; Set Stack Pointer to top of
RAM
0x0002          ldi    r16,low(RAMEND)
0x0003          out    SPL,r16
0x0004          sei                   ; Enable interrupts
0x0005          <instr> xxx
;
.org 0x3802/0x7802
0x3804/0x7804   jmp    EXT_INT0       ; IRQ0 Handler
0x3806/0x7806   jmp    PCINT0         ; PCINT0 Handler
...             ...    ...            ;
0x1C2C          jmp    SPM_RDY        ; Store Program Memory Ready
Handler
```

When the BOOTRST Fuse is programmed and the Boot section size set to 4K bytes, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address  Labels Code             Comments
.org 0x0002
0x0002          jmp    EXT_INT0       ; IRQ0 Handler
0x0004          jmp    PCINT0         ; PCINT0 Handler
...             ...    ...            ;
0x002C          jmp    SPM_RDY        ; Store Program Memory Ready
Handler
;
.org 0x3800/0x7800
0x3800/0x7801RESET:ldir16,high(RAMEND); Main program start
0x3801/0x7801   out    SPH,r16      ; Set Stack Pointer to top of RAM
0x3802/0x7802   ldi    r16,low(RAMEND)
0x3803/0x7803   out    SPL,r16
0x3804/0x7804   sei                   ; Enable interrupts
0x3805/0x7805   <instr>  xxx
```

When the BOOTRST Fuse is programmed, the Boot section size set to 4K bytes and the IVSEL bit in the MCUCR Register is set before any interrupts are enabled, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address  Labels Code                 Comments
;
.org 0x3800/0x7800
0x3800/0x7800   jmp    RESET          ; Reset handler
0x3802/0x7802   jmp    EXT_INT0       ; IRQ0 Handler
0x3804/0x7804   jmp    PCINT0         ; PCINT0 Handler
...             ...    ...            ;
0x382C/0x782C   jmp    SPM_RDY   ; Store Program Memory Ready Handler
;
```

```
0x382E/0x782ERESET:ldir16,high(RAMEND); Main program start
0x382F/0x782F  out    SPH,r16      ; Set Stack Pointer to top of RAM
0x3830/0x7830  ldi    r16,low(RAMEND)
0x3831/0x7831  out    SPL,r16
0x3832/0x7832  sei                 ; Enable interrupts
0x3833/0x7833  <instr>  xxx
```

### Moving Interrupts Between Application and Boot Space

The MCU Control Register controls the placement of the Interrupt Vector table.

### MCU Control Register – MCUCR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | JTD | – | – | PUD | – | – | IVSEL | IVCE | MCUCR |
| Read/Write | R/W | R | R | R/W | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 1 – IVSEL: Interrupt Vector Select**

When the IVSEL bit is cleared (zero), the Interrupt Vectors are placed at the start of the Flash memory. When this bit is set (one), the Interrupt Vectors are moved to the beginning of the Boot Loader section of the Flash. The actual address of the start of the Boot Flash Section is determined by the BOOTSZ Fuses. Refer to the section “Boot Loader Support – Read-While-Write Self-Programming” on page 243 for details. To avoid unintentional changes of Interrupt Vector tables, a special write procedure must be followed to change the IVSEL bit:

1. Write the Interrupt Vector Change Enable (IVCE) bit to one.
2. Within four cycles, write the desired value to IVSEL while writing a zero to IVCE.

Interrupts will automatically be disabled while this sequence is executed. Interrupts are disabled in the cycle IVCE is set, and they remain disabled until after the instruction following the write to IVSEL. If IVSEL is not written, interrupts remain disabled for four cycles. The I-bit in the Status Register is unaffected by the automatic disabling.

Note: If Interrupt Vectors are placed in the Boot Loader section and Boot Lock bit BLB02 is programmed, interrupts are disabled while executing from the Application section. If Interrupt Vectors are placed in the Application section and Boot Lock bit BLB12 is programed, interrupts are disabled while executing from the Boot Loader section. Refer to the section “Boot Loader Support – Read-While-Write Self-Programming” on page 243 for details on Boot Lock bits.

- **Bit 0 – IVCE: Interrupt Vector Change Enable**

The IVCE bit must be written to logic one to enable change of the IVSEL bit. IVCE is cleared by hardware four cycles after it is written or when IVSEL is written. Setting the IVCE bit will disable interrupts, as explained in the IVSEL description above. See Code Example below.

Assembly Code Example

```
Move_interrupts:
  ; Enable change of Interrupt Vectors
  ldi  r16, (1<<IVCE)
  out  MCUCR, r16
  ; Move interrupts to Boot Flash section
  ldi  r16, (1<<IVSEL)
  out  MCUCR, r16
  ret
```

C Code Example

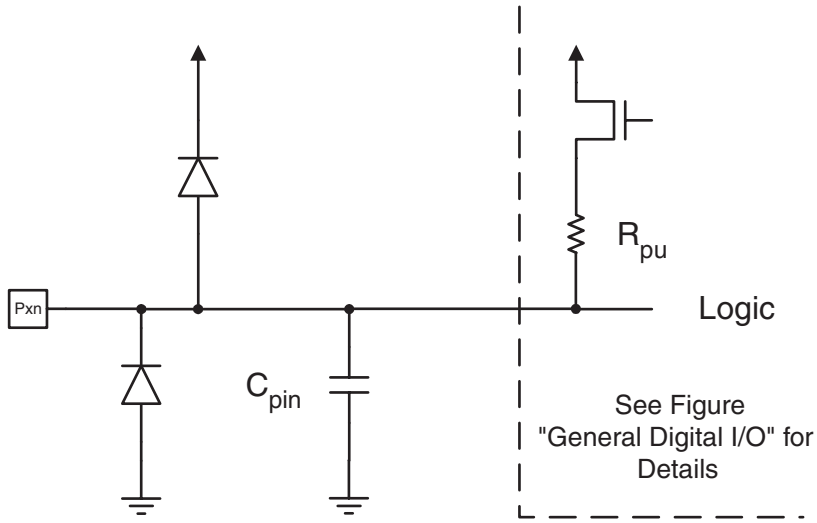
```
void Move_interrupts(void)
{
  /* Enable change of Interrupt Vectors */
  MCUCR = (1<<IVCE);
  /* Move interrupts to Boot Flash section */
  MCUCR = (1<<IVSEL);
}
```

# I/O-Ports

## 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). Each output buffer has symmetrical drive characteristics with both high sink and source capability. 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 22. Refer to “Electrical Characteristics” on page 287 for a complete list of parameters.

**Figure 22.** 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 “Register Description for I/O-Ports” on page 73.

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.

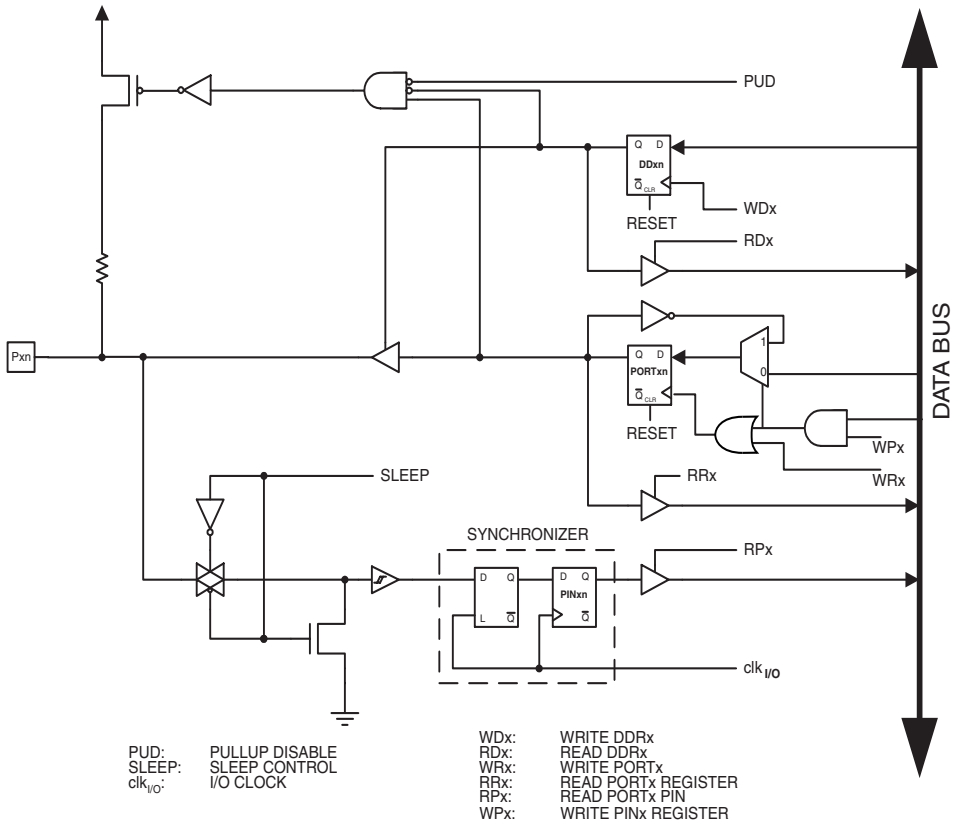
Using the I/O port as General Digital I/O is described in “Ports as General Digital I/O” on page 53. Most port pins are multiplexed with alternate functions for the peripheral features on the device. How each alternate function interferes with the port pin is described in “Alternate Port Functions” on page 57. 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.

## Ports as General Digital I/O

The ports are bi-directional I/O ports with optional internal pull-ups. Figure 23 shows a functional description of one I/O-port pin, here generically called Pxn.

**Figure 23.** 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.

### Configuring the Pin

Each port pin consists of three register bits: DDxn, PORTxn, and PINxn. As shown in "Register Description for I/O-Ports" on page 73, 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).

### 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.

### 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 24 summarizes the control signals for the pin value.

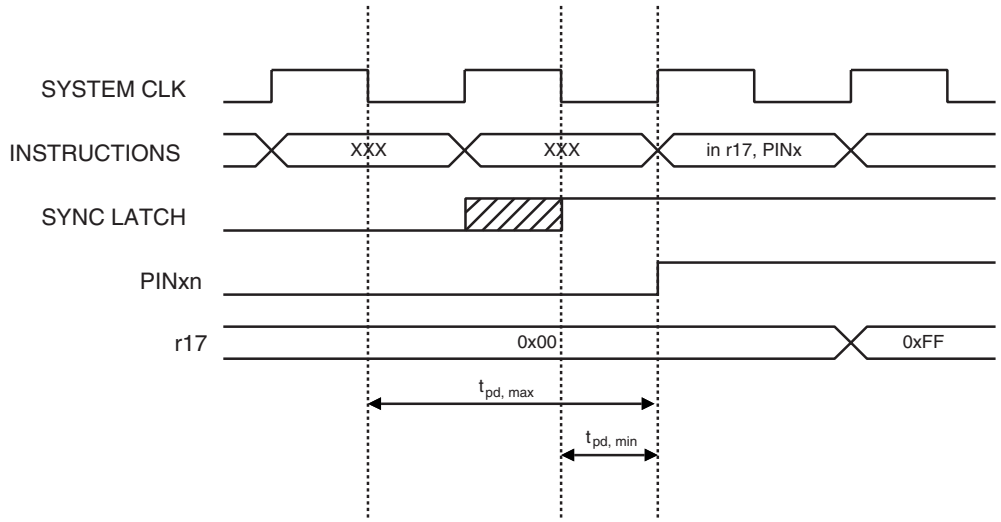
**Table 24.** Port Pin Configurations

| DDxn | PORTxn | PUD (in MCUCR) | 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) |

### 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 23, 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 24 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.

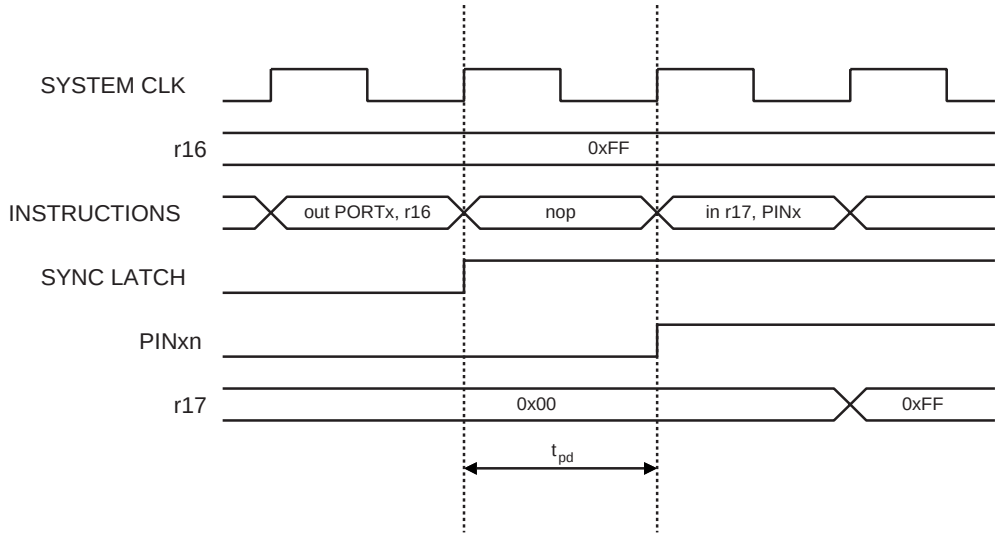
**Figure 24.** Synchronization when Reading an Externally Applied Pin value



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 25. 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 25.** 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
...
```

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;
...
```

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.

**Digital Input Enable and Sleep Modes**

As shown in Figure 23, 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, Power-save mode, and Standby 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 “Alternate Port Functions” on page 57.

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.

**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

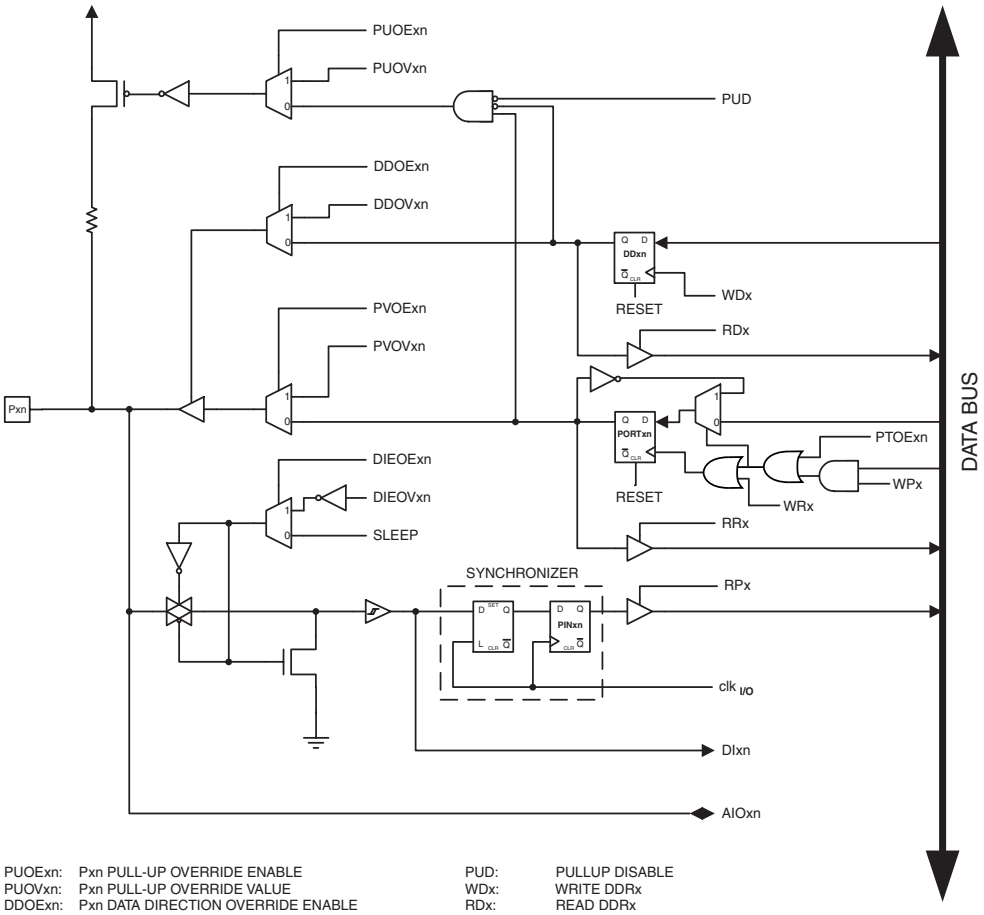
described above, floating inputs should be avoided to reduce current consumption in all other modes where the digital inputs are enabled (Reset, Active mode and Idle mode).

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.

## Alternate Port Functions

Most port pins have alternate functions in addition to being general digital I/Os. Figure 26 shows how the port pin control signals from the simplified Figure 23 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 26.** Alternate Port Functions(1)




| | | | |
|---|---|---|---|
| PUOExn: | Pxn PULL-UP OVERRIDE ENABLE | PUD: | PULLUP 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 25 summarizes the function of the overriding signals. The pin and port indexes from Figure 26 are not shown in the succeeding tables. The overriding signals are generated internally in the modules having the alternate function.

**Table 25.** 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.

**MCU Control Register – MCUCR**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | JTD | – | – | PUD | – | – | IVSEL | IVCE | MCUCR |
| Read/Write | R/W | R | R | R/W | R | R | R/W | R/W | |
| 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 “Configuring the Pin” on page 53 for more details about this feature.

**Alternate Functions of Port B**

The Port B pins with alternate functions are shown in Table 26.

**Table 26.** Port B Pins Alternate Functions

| Port Pin | Alternate Functions |
|---|---|
| PB7 | OC2A/$\overline{\text{PCINT15}}$ (Output Compare and PWM Output A for Timer/Counter2 or Pin Change Interrupt15). |
| PB6 | OC1B/$\overline{\text{PCINT}}$14 (Output Compare and PWM Output B for Timer/Counter1 or Pin Change Interrupt14). |
| PB5 | OC1A/$\overline{\text{PCINT13}}$ (Output Compare and PWM Output A for Timer/Counter1 or Pin Change Interrupt13). |
| PB4 | OC0A/$\overline{\text{PCINT12}}$ (Output Compare and PWM Output A for Timer/Counter0 or Pin Change Interrupt12). |
| PB3 | MISO/$\overline{\text{PCINT11}}$ (SPI Bus Master Input/Slave Output or Pin Change Interrupt11). |
| PB2 | MOSI/$\overline{\text{PCINT10}}$ (SPI Bus Master Output/Slave Input or Pin Change Interrupt10). |
| PB1 | SCK/$\overline{\text{PCINT9}}$ (SPI Bus Serial Clock or Pin Change Interrupt9). |
| PB0 | $\overline{\text{SS}}$/$\overline{\text{PCINT8}}$ (SPI Slave Select input or Pin Change Interrupt8). |

The alternate pin configuration is as follows:

- **OC2A/PCINT15, Bit 7**

OC2, Output Compare Match A output: The PB7 pin can serve as an external output for the Timer/Counter2 Output Compare A. The pin has to be configured as an output (DDB7 set (one)) to serve this function. The OC2A pin is also the output pin for the PWM mode timer function.

PCINT15, Pin Change Interrupt source 15: The PB7 pin can serve as an external interrupt source.

- **OC1B/PCINT14, Bit 6**

OC1B, Output Compare Match B output: The PB6 pin can serve as an external output for the Timer/Counter1 Output Compare B. The pin has to be configured as an output (DDB6 set (one)) to serve this function. The OC1B pin is also the output pin for the PWM mode timer function.

PCINT14, Pin Change Interrupt Source 14: The PB6 pin can serve as an external interrupt source.

- **OC1A/PCINT13, Bit 5**

OC1A, Output Compare Match A output: The PB5 pin can serve as an external output for the Timer/Counter1 Output Compare A. The pin has to be configured as an output (DDB5 set (one)) to serve this function. The OC1A pin is also the output pin for the PWM mode timer function.

PCINT13, Pin Change Interrupt Source 13: The PB5 pin can serve as an external interrupt source.

- **OC0A/PCINT12, Bit 4**

OC0A, Output Compare Match A output: The PB4 pin can serve as an external output for the Timer/Counter0 Output Compare A. The pin has to be configured as an output (DDB4 set (one)) to serve this function. The OC0A pin is also the output pin for the PWM mode timer function.

PCINT12, Pin Change Interrupt Source 12: The PB4 pin can serve as an external interrupt source.

- **MISO/PCINT11 – Port B, Bit 3**

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

PCINT11, Pin Change Interrupt Source 11: The PB3 pin can serve as an external interrupt source.

- **MOSI/PCINT10 – Port B, Bit 2**

MOSI: SPI Master Data output, Slave Data input for SPI. When the SPI is enabled as a Slave, this pin is configured as an input regardless of the setting of DDB2. When the SPI is enabled as a Master, the data direction of this pin is controlled by DDB2. When the pin is forced to be an input, the pull-up can still be controlled by the PORTB2 bit.

PCINT10, Pin Change Interrupt Source 10: The PB2 pin can serve as an external interrupt source.

- **SCK/PCINT9 – Port B, Bit 1**

SCK: Master Clock output, Slave Clock input pin for SPI. When the SPI is enabled as a Slave, this pin is configured as an input regardless of the setting of DDB1. When the SPI is enabled as a Master, the data direction of this pin is controlled by DDB1. When the pin is forced to be an input, the pull-up can still be controlled by the PORTB1 bit.

PCINT9, Pin Change Interrupt Source 9: The PB1 pin can serve as an external interrupt source.

- **$\overline{\text{SS}}$/PCINT8 – Port B, Bit 0**

$\overline{\text{SS}}$: Slave Port Select input. When the SPI is enabled as a Slave, this pin is configured as an input regardless of the setting of DDB0. 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 DDB0. When the pin is forced to be an input, the pull-up can still be controlled by the PORTB0 bit

PCINT8, Pin Change Interrupt Source 8: The PB0 pin can serve as an external interrupt source.

Table 27 and Table 28 relate the alternate functions of Port B to the overriding signals shown in Figure 26 on page 57. SPI MSTR INPUT and SPI SLAVE OUTPUT constitute

the MISO signal, while MOSI is divided into SPI MSTR OUTPUT and SPI SLAVE INPUT.

**Table 27.** Overriding Signals for Alternate Functions in PB7..PB4

| Signal Name | PB7/OC2A/ PCINT15 | PB6/OC1B/ PCINT14 | PB5/OC1A/ PCINT13 | PB4/OC0A/ PCINT12 |
|---|---|---|---|---|
| PUOE | 0 | 0 | 0 | 0 |
| PUOV | 0 | 0 | 0 | 0 |
| DDOE | 0 | 0 | 0 | 0 |
| DDOV | 0 | 0 | 0 | 0 |
| PVOE | OC2A ENABLE | OC1B ENABLE | OC1A ENABLE | OC0A ENABLE |
| PVOV | OC2A | OC1B | OC1A | OC0A |
| PTOE | – | – | – | – |
| DIEOE | PCINT15 • PCIE1 | PCINT14 • PCIE1 | PCINT13 • PCIE1 | PCINT12 • PCIE1 |
| DIEOV | 1 | 1 | 1 | 1 |
| DI | PCINT15 INPUT | PCINT14 INPUT | PCINT13 INPUT | PCINT12 INPUT |
| AIO | – | – | – | – |

**Table 28.** Overriding Signals for Alternate Functions in PB3..PB0

| Signal Name | PB3/MISO/ PCINT11 | PB2/MOSI/ PCINT10 | PB1/SCK/ PCINT9 | PB0/$\overline{SS}$/ PCINT8 |
|---|---|---|---|---|
| PUOE | SPE • MSTR | SPE • $\overline{MSTR}$ | SPE • $\overline{MSTR}$ | SPE • $\overline{MSTR}$ |
| PUOV | PORTB3 • $\overline{PUD}$ | PORTB2 • $\overline{PUD}$ | PORTB1 • $\overline{PUD}$ | PORTB0 • $\overline{PUD}$ |
| DDOE | SPE • MSTR | SPE • $\overline{MSTR}$ | SPE • $\overline{MSTR}$ | SPE • $\overline{MSTR}$ |
| DDOV | 0 | 0 | 0 | 0 |
| PVOE | SPE • $\overline{MSTR}$ | SPE • MSTR | SPE • MSTR | 0 |
| PVOV | SPI SLAVE OUTPUT | SPI MSTR OUTPUT | SCK OUTPUT | 0 |
| PTOE | – | – | – | – |
| DIEOE | PCINT11 • PCIE1 | PCINT10 • PCIE1 | PCINT9 • PCIE1 | PCINT8 • PCIE1 |
| DIEOV | 1 | 1 | 1 | 1 |
| DI | PCINT11 INPUT<br>SPI MSTR INPUT | PCINT10 INPUT<br>SPI SLAVE INPUT | PCINT9 INPUT<br>SCK INPUT | PCINT8 INPUT<br>SPI $\overline{SS}$ |
| AIO | – | – | – | – |

**Alternate Functions of Port D**

The Port D pins with alternate functions are shown in Table 29.

**Table 29.** Port D Pins Alternate Functions

| Port Pin | Alternate Function |
|---|---|
| PD7 | - |
| PD6 | - |
| PD5 | - |
| PD4 | - |
| PD3 | - |
| PD2 | - |
| PD1 | $\overline{\text{INT0}}$ (External Interrupt0 Input) |
| PD0 | ICP1 (Timer/Counter1 Input Capture pin) |

The alternate pin configuration is as follows:

- **$\overline{\text{INT0}}$ – Port D, Bit 1**

INT0, External Interrupt Source 0. The PD1 pin can serve as an external interrupt source to the MCU.

- **ICP1 – Port D, Bit 0**

ICP1 – Input Capture pin1: The PD0 pin can act as an Input Capture pin for Timer/Counter1.

Table 30 relates the alternate functions of Port D to the overriding signals shown in Figure 26 on page 57.

**Table 30.** Overriding Signals for Alternate Functions in PD3..PD0

| Signal Name | PD3 | PD2 | PD1/INT0 | PD0/ICP1 |
|---|---|---|---|---|
| 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 |
| PTOE | – | – | – | – |
| DIEOE | 0 | 0 | INT0 ENABLE | 0 |
| DIEOV | 0 | 0 | INT0 ENABLE | 0 |
| DI | – | – | INT0 INPUT | ICP1 INPUT |
| AIO | – | – | – | – |

**Alternate Functions of Port E**

The Port E pins with alternate functions are shown in Table 31.

**Table 31.** Port E Pins Alternate Functions

| Port Pin | Alternate Function |
|---|---|
| PE7 | PCINT7 (Pin Change Interrupt7)<br>CLKO (Divided System Clock) |
| PE6 | DO/PCINT6 (USI Data Output or Pin Change Interrupt6) |
| PE5 | DI/SDA/PCINT5 (USI Data Input or TWI Serial DAta or Pin Change Interrupt5) |
| PE4 | USCK/SCL/PCINT4 (USART External Clock Input/Output or TWI Serial Clock or Pin Change Interrupt4) |
| PE3 | AIN1/PCINT3 (Analog Comparator Negative Input or Pin Change Interrupt3) |
| PE2 | XCK/AIN0/ PCINT2 (USART External Clock or Analog Comparator Positive Input or Pin Change Interrupt2) |
| PE1 | TXD/PCINT1 (USART Transmit Pin or Pin Change Interrupt1) |
| PE0 | RXD/PCINT0 (USART Receive Pin or Pin Change Interrupt0) |

- **PCINT7 – Port E, Bit 7**

PCINT7, Pin Change Interrupt Source 7: The PE7 pin can serve as an external interrupt source.

CLKO, Divided System Clock: The divided system clock can be output on the PE7 pin. The divided system clock will be output if the CKOUT Fuse is programmed, regardless of the PORTE7 and DDE7 settings. It will also be output during reset.

- **DO/PCINT6 – Port E, Bit 6**

DO, Universal Serial Interface Data output.

PCINT6, Pin Change Interrupt Source 6: The PE6 pin can serve as an external interrupt source.

- **DI/SDA/PCINT5 – Port E, Bit 5**

DI, Universal Serial Interface Data input.

SDA, Two-wire Serial Interface Data:

PCINT5, Pin Change Interrupt Source 5: The PE5 pin can serve as an external interrupt source.

- **USCK/SCL/PCINT4 – Port E, Bit 4**

USCK, Universal Serial Interface Clock.

SCL, Two-wire Serial Interface Clock.

PCINT4, Pin Change Interrupt Source 4: The PE4 pin can serve as an external interrupt source.

- **AIN1/PCINT3 – Port E, Bit 3**

AIN1 – Analog Comparator Negative input. This pin is directly connected to the negative input of the Analog Comparator.

PCINT3, Pin Change Interrupt Source 3: The PE3 pin can serve as an external interrupt source.

- **XCK/AIN0/PCINT2 – Port E, Bit 2**

XCK, USART External Clock. The Data Direction Register (DDE2) controls whether the clock is output (DDE2 set) or input (DDE2 cleared). The XCK pin is active only when the USART operates in synchronous mode.

AIN0 – Analog Comparator Positive input. This pin is directly connected to the positive input of the Analog Comparator.

PCINT2, Pin Change Interrupt Source 2: The PE2 pin can serve as an external interrupt source.

- **TXD/PCINT1 – Port E, Bit 1**

TXD0, UART0 Transmit pin.

PCINT1, Pin Change Interrupt Source 1: The PE1 pin can serve as an external interrupt source.

- **RXD/PCINT0 – Port E, Bit 0**

RXD, USART Receive pin. Receive Data (Data input pin for the USART). When the USART Receiver is enabled this pin is configured as an input regardless of the value of DDE0. When the USART forces this pin to be an input, a logical one in PORTE0 will turn on the internal pull-up.

PCINT0, Pin Change Interrupt Source 0: The PE0 pin can serve as an external interrupt source.

Table 32 and Table 33 relates the alternate functions of Port E to the overriding signals shown in Figure 26 on page 57.

**Table 32.** Overriding Signals for Alternate Functions PE7..PE4

| Signal Name | PE7/PCINT7 | PE6/DO/ PCINT6 | PE5/DI/SDA/ PCINT5 | PE4/USCK/SCL/ PCINT4 |
|---|---|---|---|---|
| PUOE | 0 | 0 | USI_TWO-WIRE | 0 |
| PUOV | 0 | 0 | 0 | 0 |
| DDOE | CKOUT(1) | 0 | USI_TWO-WIRE | USI_TWO-WIRE |
| DDOV | 1 | 0 | $(\overline{SDA} + \overline{PORTE5}) \bullet$ DDE5 | (USI_SCL_HOLD + $\overline{PORTE4}$) + DDE4 |
| PVOE | CKOUT(1) | USI_THREE-WIRE | USI_TWO-WIRE • DDE5 | USI_TWO-WIRE • DDE4 |
| PVOV | $clk_{I/O}$ | DO | 0 | 0 |
| PTOE | – | – | – | USITC |
| DIEOE | PCINT7 • PCIE0 | PCINT6 • PCIE0 | (PCINT5 • PCIE0) + USISIE | (PCINT4 • PCIE0) + USISIE |
| DIEOV | 1 | 1 | 1 | 1 |
| DI | PCINT7 INPUT | PCINT6 INPUT | DI/SDA INPUT PCINT5 INPUT | USCKL/SCL INPUT PCINT4 INPUT |
| AIO | – | – | – | – |

Note: 1. CKOUT is one if the CKOUT Fuse is programmed

**Table 33.** Overriding Signals for Alternate Functions in PE3..PE0

| Signal Name | PE3/AIN1/ PCINT3 | PE2/XCK/AIN0/ PCINT2 | PE1/TXD/ PCINT1 | PE0/RXD/PCINT 0 |
|---|---|---|---|---|
| PUOE | 0 | XCK OUTPUT ENABLE | TXEN | RXEN |
| PUOV | 0 | XCK | 0 | PORTE0 • $\overline{PUD}$ |
| DDOE | 0 | 0 | TXEN | RXEN |
| DDOV | 0 | 0 | 1 | 0 |
| PVOE | 0 | 0 | TXEN | 0 |
| PVOV | 0 | 0 | TXD | 0 |
| PTOE | – | – | – | – |
| DIEOE | (PCINT3 • PCIE0) + AIN1D(1) | (PCINT2 • PCIE0) + AIN0D(1) | PCINT1 • PCIE0 | PCINT0 • PCIE0 |
| DIEOV | PCINT3 • PCIE0 | PCINT2 • PCIE0 | 1 | 1 |
| DI | PCINT3 INPUT | XCK/PCINT2 INPUT | PCINT1 INPUT | RXD/PCINT0 INPUT |
| AIO | AIN1 INPUT | AIN0 INPUT | – | – |

Note: 1. AIN0D and AIN1D is described in “Digital Input Disable Register 1 – DIDR1” on page 193.

**Alternate Functions of Port F**

The Port F has an alternate function as analog input for the ADC as shown in Table 34. If some Port F pins are configured as outputs, it is essential that these do not switch when a conversion is in progress. This might corrupt the result of the conversion. If the JTAG interface is enabled, the pull-up resistors on pins PF7(TDI), PF5(TMS) and PF4(TCK) will be activated even if a reset occurs.

**Table 34.** Port F Pins Alternate Functions

| Port Pin | Alternate Function |
|---|---|
| PF7 | ADC7/TDI (ADC input channel 7 or JTAG Test Data Input) |
| PF6 | ADC6/TDO (ADC input channel 6 or JTAG Test Data Output) |
| PF5 | ADC5/TMS (ADC input channel 5 or JTAG Test mode Select) |
| PF4 | ADC4/TCK (ADC input channel 4 or JTAG Test ClocK) |
| PF3 | ADC3 (ADC input channel 3) |
| PF2 | ADC2 (ADC input channel 2) |
| PF1 | ADC1 (ADC input channel 1) |
| PF0 | ADC0 (ADC input channel 0) |

- **TDI, ADC7 – Port F, Bit 7**

ADC7, Analog to Digital Converter, Channel 7.

TDI, JTAG Test Data In: Serial input data to be shifted in to the Instruction Register or Data Register (scan chains). When the JTAG interface is enabled, this pin can not be used as an I/O pin.

- **TDO, ADC6 – Port F, Bit 6**

ADC6, Analog to Digital Converter, Channel 6.

TDO, JTAG Test Data Out: Serial output data from Instruction Register or Data Register. When the JTAG interface is enabled, this pin can not be used as an I/O pin. In TAP states that shift out data, the TDO pin drives actively. In other states the pin is pulled high.

- **TMS, ADC5 – Port F, Bit 5**

ADC5, Analog to Digital Converter, Channel 5.

TMS, JTAG Test mode Select: This pin is used for navigating through the TAP-controller state machine. When the JTAG interface is enabled, this pin can not be used as an I/O pin.

- **TCK, ADC4 – Port F, Bit 4**

ADC4, Analog to Digital Converter, Channel 4.

TCK, JTAG Test Clock: JTAG operation is synchronous to TCK. When the JTAG interface is enabled, this pin can not be used as an I/O pin.

- **ADC3 - ADC0 – Port F, Bit 3:0**

Analog to Digital Converter, Channel 3-0.

**Table 35.** Overriding Signals for Alternate Functions in PF7..PF4

| Signal Name | PF7/ADC7/TDI | PF6/ADC6/TDO | PF5/ADC5/TMS | PF4/ADC4/TCK |
|---|---|---|---|---|
| PUOE | JTAGEN | JTAGEN | JTAGEN | JTAGEN |
| PUOV | 1 | 1 | 1 | 1 |
| DDOE | JTAGEN | JTAGEN | JTAGEN | JTAGEN |
| DDOV | 0 | SHIFT_IR + SHIFT_DR | 0 | 0 |
| PVOE | 0 | JTAGEN | 0 | 0 |
| PVOV | 0 | TDO | 0 | 0 |
| PTOE | – | – | – | – |
| DIEOE | JTAGEN | JTAGEN | JTAGEN | JTAGEN |
| DIEOV | 0 | 0 | 0 | 0 |
| DI | – | – | – | – |
| AIO | TDI<br>ADC7 INPUT | ADC6 INPUT | TMS<br>ADC5 INPUT | TCK<br>ADC4 INPUT |

**Table 36.** Overriding Signals for Alternate Functions in PF3..PF0

| Signal Name | PF3/ADC3 | PF2/ADC2 | PF1/ADC1 | PF0/ADC0 |
|---|---|---|---|---|
| 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 |
| PTOE | – | – | – | – |
| DIEOE | 0 | 0 | 0 | 0 |
| DIEOV | 0 | 0 | 0 | 0 |
| DI | – | – | – | – |
| AIO | ADC3 INPUT | ADC2 INPUT | ADC1 INPUT | ADC0 INPUT |

**Alternate Functions of Port G**

The alternate pin configuration is as follows:

**Table 37.** Port G Pins Alternate Functions

| Port Pin | Alternate Function |
|---|---|
| PG5 | $\overline{\text{RESET}}$(1) |
| PG4 | T0 (Timer/Counter0 Clock Inpu) |
| PG3 | T1 (Timer/Counter1 Clock Input) |
| PG2 | - |
| PG1 | - |
| PG0 | - |

Note: 1. Port G, PG5 is input only. Pull-up is always on.
See Table 109 on page 258 for RSTDISBL fuse.

The alternate pin configuration is as follows:

- **$\overline{\text{RESET}}$ – Port G, Bit 5**

$\overline{\text{RESET}}$: External Reset input. When the RSTDISBL Fuse is programmed ('0'), PG5 will function as input with pull-up always on.

- **T0 – Port G, Bit 4**

T0, Timer/Counter0 Counter Source.

- **T1 – Port G, Bit 3**

T1, Timer/Counter1 Counter Source.

Table 37 and Table 38 relates the alternate functions of Port G to the overriding signals shown in Figure 26 on page 57.

**Table 38.** Overriding Signals for Alternate Functions in PG4 and PG3

| Signal Name | PG4/T0 | PG3/T1 |
|---|---|---|
| PUOE | 0 | 0 |
| PUOV | 0 | 0 |
| DDOE | 0 | 0 |
| DDOV | 0 | 0 |
| PVOE | 0 | 0 |
| PVOV | 0 | 0 |
| PTOE | – | – |
| DIEOE | 0 | 0 |
| DIEOV | 0 | 0 |
| DI | T0 INPUT | T1 INPUT |
| AIO | - | - |

**Alternate Functions of Port H**

Port H is only present in ATmega3250/6450. The alternate pin configuration is as follows:

**Table 39.** Port H Pins Alternate Functions

| Port Pin | Alternate Function |
|---|---|
| PH7 | PCINT23 (Pin Change Interrupt23) |
| PH6 | PCINT22 (Pin Change Interrupt22) |
| PH5 | PCINT21 (Pin Change Interrupt21) |
| PH4 | PCINT20 (Pin Change Interrupt20) |
| PH3 | PCINT19 (Pin Change Interrupt19) |
| PH2 | PCINT18 (Pin Change Interrupt18) |
| PH1 | PCINT17 (Pin Change Interrupt17) |
| PH0 | PCINT16 (Pin Change Interrupt16) |

The alternate pin configuration is as follows:

- **PCINT23 – Port H, Bit 7**

PCINT23, Pin Change Interrupt Source 23: The PH7 pin can serve as an external interrupt source.

- **PCINT22 – Port H, Bit 6**

PCINT22, Pin Change Interrupt Source 22: The PH6 pin can serve as an external interrupt source.

- **PCINT21 – Port H, Bit 5**

PCINT21, Pin Change Interrupt Source 21: The PH5 pin can serve as an external interrupt source.

- **PCINT20 – Port H, Bit 4**

PCINT20, Pin Change Interrupt Source 20: The PH4 pin can serve as an external interrupt source.

- **PCINT19 – Port H, Bit 3**

PCINT19, Pin Change Interrupt Source 19: The PH3 pin can serve as an external interrupt source.

- **PCINT18 – Port H, Bit 2**

PCINT18, Pin Change Interrupt Source 18: The PH2 pin can serve as an external interrupt source.

- **PCINT17 – Port H, Bit 1**

PCINT17, Pin Change Interrupt Source 17: The P1 pin can serve as an external interrupt source.

- **PCINT16 – Port H, Bit 0**

PCINT16, Pin Change Interrupt Source 16: The PH0 pin can serve as an external interrupt source.

Table 40 and Table 41 relates the alternate functions of Port H to the overriding signals shown in Figure 26 on page 57.

**Table 40.** Overriding Signals for Alternate Functions in PH7:4

| Signal Name | PH7/PCINT23 | PH6/PCINT22 | PH5/PCINT21 | PH4/PCINT20 |
|---|---|---|---|---|
| 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 |
| PTOE | – | – | – | – |
| DIEOE | PCINT23 • PCIE0 | PCINT22 • PCIE0 | PCINT21 • PCIE0 | PCINT20 • PCIE0 |
| DIEOV | 0 | 0 | 0 | 0 |
| DI | PCINT23 INPUT | PCINT22 INPUT | PCINT21 INPUT | PCINT20 INPUT |
| AIO | – | – | – | – |

**Table 41.** Overriding Signals for Alternate Functions in PH3:0

| Signal Name | PH3/PCINT19 | PH2/PCINT18 | PH1/PCINT17 | PH0/PCINT16 |
|---|---|---|---|---|
| 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 |
| PTOE | – | – | – | – |
| DIEOE | PCINT19 • PCIE0 | PCINT18 • PCIE0 | PCINT17 • PCIE0 | PCINT16 • PCIE0 |
| DIEOV | 0 | 0 | 0 | 0 |
| DI | PCINT19 INPUT | PCINT18 INPUT | PCINT17 INPUT | PCINT16 INPUT |
| AIO | – | – | – | – |

**Alternate Functions of Port J**

Port J is only present in ATmega3250/6450. The alternate pin configuration is as follows:

**Table 42.** Port J Pins Alternate Functions

| Port Pin | Alternate Function |
|---|---|
| PJ6 | PCINT30 (Pin Change Interrupt30) |
| PJ5 | PCINT29 (Pin Change Interrupt29) |
| PJ4 | PCINT28 (Pin Change Interrupt28) |
| PJ3 | PCINT27 (Pin Change Interrupt27) |
| PJ2 | PCINT26(Pin Change Interrupt26) |
| PJ1 | PCINT25(Pin Change Interrupt25) |
| PJ0 | PCINT24 (Pin Change Interrupt26) |

The alternate pin configuration is as follows:

- **PCINT30 – Port J, Bit 6**

PCINT30, Pin Change Interrupt Source 30: The PE30 pin can serve as an external interrupt source.

- **PCINT29 – Port J, Bit 5**

PCINT29, Pin Change Interrupt Source 29: The PE29 pin can serve as an external interrupt source.

- **PCINT28 – Port J, Bit 4**

PCINT28, Pin Change Interrupt Source 28: The PE28 pin can serve as an external interrupt source.

- **PCINT27 – Port J, Bit 3**

PCINT27, Pin Change Interrupt Source 27: The PE27 pin can serve as an external interrupt source.

- **PCINT26 – Port J, Bit 2**

PCINT26, Pin Change Interrupt Source 26: The PE26 pin can serve as an external interrupt source.

- **PCINT25 – Port J, Bit 1**

PCINT25, Pin Change Interrupt Source 25: The PE25 pin can serve as an external interrupt source.

- **PCINT24 – Port J, Bit 0**

PCINT24, Pin Change Interrupt Source 24: The PE24 pin can serve as an external interrupt source.

Table 43 and Table 44 relates the alternate functions of Port J to the overriding signals shown in Figure 26 on page 57.

**Table 43.** Overriding Signals for Alternate Functions in PJ7:4

| Signal Name | | PJ6/PCINT30 | PJ5/PCINT29 | PJ4/PCINT28 |
|---|---|---|---|---|
| PUOE | | 0 | 0 | 0 |
| PUOV | | 0 | 0 | 0 |
| DDOE | | 0 | 0 | 0 |
| DDOV | | 0 | 0 | 0 |
| PVOE | | 0 | 0 | 0 |
| PVOV | | 0 | 0 | 0 |
| PTOE | | – | – | – |
| DIEOE | | PCINT30 • PCIE0 | PCINT29 • PCIE0 | PCINT28 • PCIE0 |
| DIEOV | | 0 | 0 | 0 |
| DI | | – | – | – |
| AIO | | – | – | – |

**Table 44.** Overriding Signals for Alternate Functions in PH3:0

| Signal Name | PJ3/PCINT27 | PJ2/PCINT26 | PJ1/PCINT25 | PJ0/PCINT24 |
|---|---|---|---|---|
| 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 |
| PTOE | – | – | – | – |
| DIEOE | PCINT27 • PCIE0 | PCINT26 • PCIE0 | PCINT25 • PCIE0 | PCINT24 • PCIE0 |
| DIEOV | 0 | 0 | 0 | 0 |
| DI | – | – | – | – |
| AIO | – | – | – | – |

## Register Description for I/O-Ports

### Port A Data Register – PORTA

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PORTA7 | PORTA6 | PORTA5 | PORTA4 | PORTA3 | PORTA2 | PORTA1 | PORTA0 | PORTA |
| 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 | |

### Port A Data Direction Register – DDRA

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | DDA7 | DDA6 | DDA5 | DDA4 | DDA3 | DDA2 | DDA1 | DDA0 | DDRA |
| 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 | |

### Port A Input Pins Address – PINA

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PINA7 | PINA6 | PINA5 | PINA4 | PINA3 | PINA2 | PINA1 | PINA0 | PINA |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | |

### Port B Data Register – PORTB

| 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 | |

### Port B Data Direction Register – DDRB

| 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 | |

### Port B Input Pins Address – PINB

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

### Port C Data Register – PORTC

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PORTC7 | 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 | |

### Port C Data Direction Register – DDRC

| 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 | |

### Port C Input Pins Address – PINC

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

### Port D Data Register – PORTD

| 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 | |

### Port D Data Direction Register – DDRD

| 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 | |

### Port D Input Pins Address – PIND

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

### Port E Data Register – PORTE

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PORTE7 | PORTE6 | PORTE5 | PORTE4 | PORTE3 | PORTE2 | PORTE1 | PORTE0 | PORTE |
| 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 | |

### Port E Data Direction Register – DDRE

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | DDE7 | DDE6 | DDE5 | DDE4 | DDE3 | DDE2 | DDE1 | DDE0 | DDRE |
| 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 | |

### Port E Input Pins Address – PINE

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PINE7 | PINE6 | PINE5 | PINE4 | PINE3 | PINE2 | PINE1 | PINE0 | PINE |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | |

### Port F Data Register – PORTF

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PORTF7 | PORTF6 | PORTF5 | PORTF4 | PORTF3 | PORTF2 | PORTF1 | PORTF0 | PORTF |
| 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 | |

### Port F Data Direction Register – DDRF

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | DDF7 | DDF6 | DDF5 | DDF4 | DDF3 | DDF2 | DDF1 | DDF0 | DDRF |
| 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 | |

### Port F Input Pins Address – PINF

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PINF7 | PINF6 | PINF5 | PINF4 | PINF3 | PINF2 | PINF1 | PINF0 | PINF |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | |

### Port G Data Register – PORTG

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | PORTG4 | PORTG3 | PORTG2 | PORTG1 | PORTG0 | PORTG |
| 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 | |

### Port G Data Direction Register – DDRG

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | DDG4 | DDG3 | DDG2 | DDG1 | DDG0 | DDRG |
| 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 | |

### Port G Input Pins Address – PING

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | PING5 | PING4 | PING3 | PING2 | PING1 | PING0 | PING |
| Read/Write | R | R | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | N/A | N/A | N/A | N/A | N/A | |

### Port H Data Register – PORTH(1)

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PORTH7 | PORTH6 | PORTH5 | PORTH4 | PORTH3 | PORTH2 | PORTH1 | PORTH0 | PORTH |
| 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 | |

### Port H Data Direction Register – DDRH(1)

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | DDH7 | DDH6 | DDH5 | DDH4 | DDH3 | DDH2 | DDH1 | DDH0 | DDRH |
| 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 | |

### Port H Input Pins Address – PINH(1)

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PINH7 | PINH6 | PINH5 | PINH4 | PINH3 | PINH2 | PINH1 | PINH0 | PINH |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | |

### Port J Data Register – PORTJ(1)

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | PORTJ6 | PORTJ5 | PORTJ4 | PORTJ3 | PORTJ2 | PORTJ1 | PORTJ0 | PORTJ |
| Read/Write | R | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Port J Data Direction Register – DDRJ(1)

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | DDJ6 | DDJ5 | DDJ4 | DDJ3 | DDJ2 | DDJ1 | DDJ0 | DDRJ |
| Read/Write | R | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Port J Input Pins Address – PINJ(1)

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | PINJ6 | PINJ5 | PINJ4 | PINJ3 | PINJ2 | PINJ1 | PINJ0 | PINJ |
| Read/Write | R | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | N/A | N/A | N/A | N/A | N/A | N/A | N/A | |

Note: 1. Register only available in ATmega3250/6450.

## External Interrupts

The External Interrupts are triggered by the INT0 pin or any of the PCINT30..0 pins. Observe that, if enabled, the interrupts will trigger even if the INT0 or PCINT30..0 pins are configured as outputs. This feature provides a way of generating a software interrupt. The pin change interrupt PCI1 will trigger if any enabled PCINT15..8 pin toggles. Pin change interrupts PCI0 will trigger if any enabled PCINT7..0 pin toggles. The PCMSK3, PCMSK2, PCMSK1, and PCMSK0 Registers control which pins contribute to the pin change interrupts. Pin change interrupts on PCINT30..0 are detected asynchronously. This implies that these interrupts can be used for waking the part also from sleep modes other than Idle mode.

The INT0 interrupts can be triggered by a falling or rising edge or a low level. This is set up as indicated in the specification for the External Interrupt Control Register A – EICRA. When the INT0 interrupt is enabled and is configured as level triggered, the interrupt will trigger as long as the pin is held low. Note that recognition of falling or rising edge interrupts on INT0 requires the presence of an I/O clock, described in “Clock Systems and their Distribution” on page 25. Low level interrupt on INT0 is detected asynchronously. This implies that this interrupt can be used for waking the part also from sleep modes other than Idle mode. 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 “System Clock and Clock Options” on page 25.

### External Interrupt Control Register A – EICRA

The External Interrupt Control Register A contains control bits for interrupt sense control.

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

- **Bit 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 45. 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 45.** 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. |

**External Interrupt Mask Register – EIMSK**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PCIE3 | PCIE2 | PCIE1 | PCIE0 | – | – | – | INT0 | EIMSK |
| Read/Write | R/W | R/W | R/W | R/W | R | R | R | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – 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 PCINT30..24 pin will cause an interrupt. The corresponding interrupt of Pin Change Interrupt Request is executed from the PCINT3 Interrupt Vector. PCINT30..24 pins are enabled individually by the PCMSK3 Register.

This bit is reserved bit in ATmega325/645 and should always be written to zero.

- **Bit 6 – 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 PCINT2 Interrupt Vector. PCINT23..16 pins are enabled individually by the PCMSK2 Register.

This bit is reserved bit in ATmega325/645 and should always be written to zero.

- **Bit 5 – 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 PCINT1 Interrupt Vector. PCINT15..8 pins are enabled individually by the PCMSK1 Register.

- **Bit 4 – 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 PCINT0 Interrupt Vector. PCINT7..0 pins are enabled individually by the PCMSK0 Register.

- **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 Control0 bits 1/0 (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.

### External Interrupt Flag Register – EIFR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | PCIF3 | PCIF2 | PCIF1 | PCIF0 | – | – | – | INTF0 | EIFR |
| Read/Write | R/W | R/W | R/W | R/W | R | R | R | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7– PCIF3: Pin Change Interrupt Flag 3**

When a logic change on any PCINT30..24 pin triggers an interrupt request, PCIF3 becomes set (one). If the I-bit in SREG and the PCIE3 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 bit is reserved bit in ATmega325/645 and will always be read as zero.

- **Bit 6– PCIF2: Pin Change Interrupt Flag 2**

When a logic change on any PCINT24..16 pin triggers an interrupt request, PCIF2 becomes set (one). If the I-bit in SREG and the PCIE2 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 bit is reserved bit in ATmega325/645 and will always be read as zero.

- **Bit 5– 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 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.

- **Bit 4– 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 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.

- **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.

### Pin Change Mask Register 3 – PCMSK3[1]

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | PCINT30 | PCINT29 | PCINT28 | PCINT27 | PCINT26 | PCINT25 | PCINT24 | PCMSK3 |
| Read/Write | R | 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 6..0 – PCINT30..24: Pin Change Enable Mask 30..24**

Each PCINT30..24-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT30..24 is set and the PCIE3 bit in EIMSK is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT30..24 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

### Pin Change Mask Register 2 – PCMSK2[1]

| 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 EIMSK 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.

Note: 1. PCMSK3 and PCMSK2 are only present in ATmega3250/6450.

### Pin Change Mask Register 1 – PCMSK1

| 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 EIMSK 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.

### Pin Change Mask Register 0 – PCMSK0

| 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 EIMSK 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.

## 8-bit Timer/Counter0 with PWM

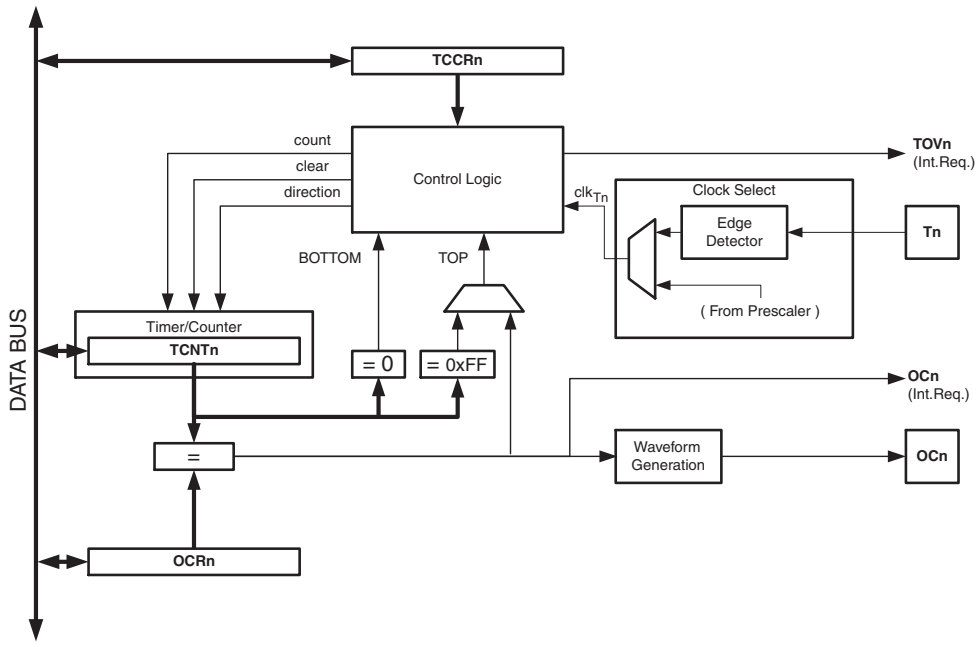
Timer/Counter0 is a general purpose, single compare unit, 8-bit Timer/Counter module. The main features are:

- **Single Compare Unit Counter**
- **Clear Timer on Compare Match (Auto Reload)**
- **Glitch-free, Phase Correct Pulse Width Modulator (PWM)**
- **Frequency Generator**
- **External Event Counter**
- **10-bit Clock Prescaler**
- **Overflow and Compare Match Interrupt Sources (TOV0 and OCF0A)**

### Overview

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 27. For the actual placement of I/O pins, refer to “Pinout ATmega3250/6450” on page 2 and “Pinout ATmega325/645” 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 “8-bit Timer/Counter Register Description” on page 90.

**Figure 27.** 8-bit Timer/Counter Block Diagram




### Registers

The Timer/Counter (TCNT0) and Output Compare Register (OCR0A) 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 (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 ($clk_{T0}$).

The double buffered Output Compare Register (OCR0A) is compared with the Timer/Counter value at all times. 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 (OC0A). See “Output Compare Unit” on page 82. for details. The compare match

event will also set the Compare Flag (OCF0A) which can be used to generate an Output Compare interrupt request.

### Definitions

Many register and bit references in this section are written in general form. A lower case “n” replaces the Timer/Counter number, in this case 0. A lower case “x” replaces the Output Compare unit number, in this case unit A. 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 46 are also used extensively throughout the document.

**Table 46.** Definitions of Timer/Counter values.

| | |
|---|---|
| BOTTOM | The counter reaches the BOTTOM when it becomes 0x00. |
| 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. |

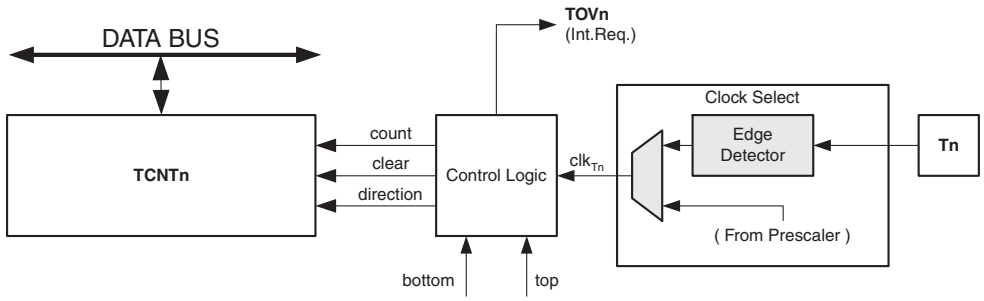
## Timer/Counter Clock Sources

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 “Timer/Counter0 and Timer/Counter1 Prescalers” on page 94.

## Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Figure 28 shows a block diagram of the counter and its surroundings.

**Figure 28.** Counter Unit Block Diagram




Signal description (internal signals):

**count** Increment or decrement TCNT0 by 1.

**direction** Select between increment and decrement.

**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.

**bottom** Signalize that TCNT0 has reached minimum value (zero).

Depending of the mode of operation used, the counter is cleared, incremented, or decremented 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 WGM01 and WGM00 bits located in the Timer/Counter Control Register (TCCR0A). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare output OC0A. For more details about advanced counting sequences and waveform generation, see “Modes of Operation” on page 85.

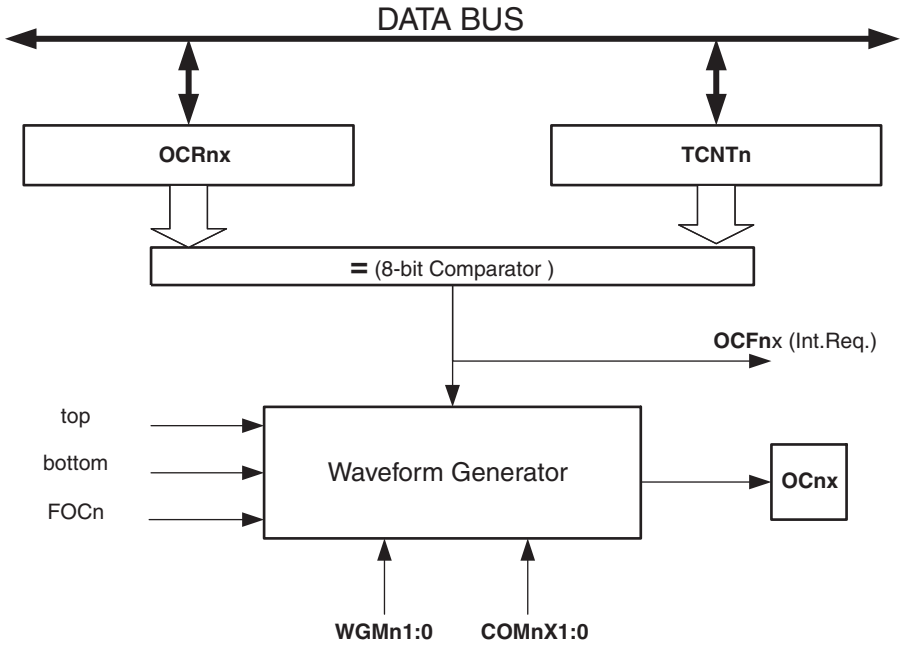
The Timer/Counter Overflow Flag (TOV0) is set according to the mode of operation selected by the WGM01:0 bits. TOV0 can be used for generating a CPU interrupt.

## Output Compare Unit

The 8-bit comparator continuously compares TCNT0 with the Output Compare Register (OCR0A). Whenever TCNT0 equals OCR0A, the comparator signals a match. A match will set the Output Compare Flag (OCF0A) at the next timer clock cycle. If enabled (OCIE0A = 1 and Global Interrupt Flag in SREG is set), the Output Compare Flag generates an Output Compare interrupt. The OCF0A Flag is automatically cleared when the interrupt is executed. Alternatively, the OCF0A 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 WGM01:0 bits and Compare Output mode (COM0A1:0) bits. The max and bottom signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation (See “Modes of Operation” on page 85.).

Figure 29 shows a block diagram of the Output Compare unit.

**Figure 29.** Output Compare Unit, Block Diagram



The OCR0A Register is double buffered when using any of the 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 OCR0 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 OCR0A Register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR0A Buffer Register, and if double buffering is disabled the CPU will access the OCR0A directly.

**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 (FOC0A) bit. Forcing compare match will not set the OCF0A Flag or reload/clear the timer, but the OC0A pin will be updated as if a real compare match had occurred (the COM0A1:0 bits settings define whether the OC0A pin is set, cleared or toggled).

**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 OCR0A to be initialized to the same value as TCNT0 without triggering an interrupt when the Timer/Counter clock is enabled.

**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 OCR0A value, the compare match will be missed, resulting in incorrect waveform generation. Similarly, do not write the TCNT0 value equal to BOTTOM when the counter is counting down.

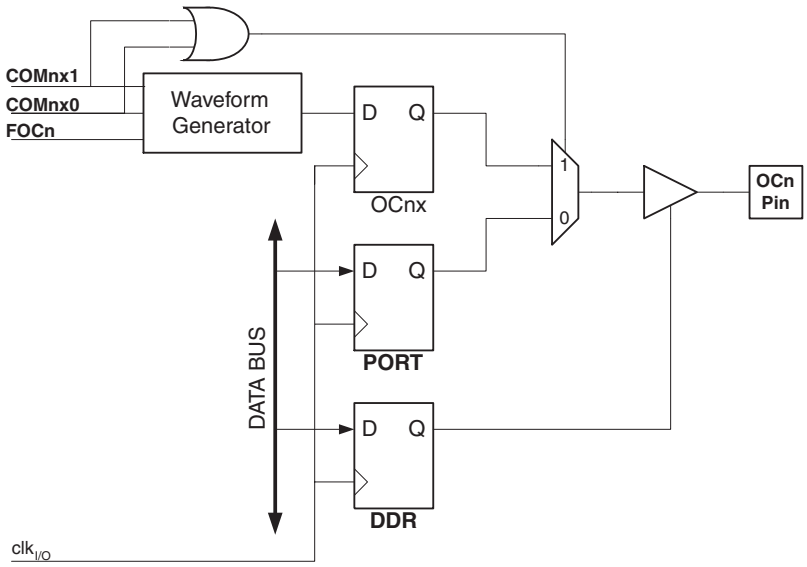
The setup of the OC0A should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC0A value is to use the Force Output Compare (FOC0A) strobe bits in Normal mode. The OC0A Register keeps its value even when changing between Waveform Generation modes.

Be aware that the COM0A1:0 bits are not double buffered together with the compare value. Changing the COM0A1:0 bits will take effect immediately.

## Compare Match Output Unit

The Compare Output mode (COM0A1:0) bits have two functions. The Waveform Generator uses the COM0A1:0 bits for defining the Output Compare (OC0A) state at the next compare match. Also, the COM0A1:0 bits control the OC0A pin output source. Figure 30 shows a simplified schematic of the logic affected by the COM0A1:0 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 COM0A1:0 bits are shown. When referring to the OC0A state, the reference is for the internal OC0A Register, not the OC0A pin. If a System Reset occur, the OC0A Register is reset to “0”.

**Figure 30.** Compare Match Output Unit, Schematic




The general I/O port function is overridden by the Output Compare (OC0A) from the Waveform Generator if either of the COM0A1:0 bits are set. However, the OC0A 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 OC0A pin (DDR_OC0A) must be set as output before the OC0A value is visible on the pin. The port override function is independent of the Waveform Generation mode.

The design of the Output Compare pin logic allows initialization of the OC0A state before the output is enabled. Note that some COM0A1:0 bit settings are reserved for certain modes of operation. See “8-bit Timer/Counter Register Description” on page 90.

### Compare Output Mode and Waveform Generation

The Waveform Generator uses the COM0A1:0 bits differently in Normal, CTC, and PWM modes. For all modes, setting the COM0A1:0 = 0 tells the Waveform Generator that no action on the OC0A Register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 48 on page 91. For fast PWM mode, refer to Table 49 on page 91, and for phase correct PWM refer to Table 50 on page 92.

A change of the COM0A1:0 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 FOC0A strobe bits.

## 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 (WGM01:0) and Compare Output mode (COM0A1:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM0A1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM0A1:0 bits control whether the output should be set, cleared, or toggled at a compare match (See “Compare Match Output Unit” on page 83.).

For detailed timing information refer to Figure 34, Figure 35, Figure 36 and Figure 37 in “Timer/Counter Timing Diagrams” on page 89.

### Normal Mode

The simplest mode of operation is the Normal mode (WGM01:0 = 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 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.

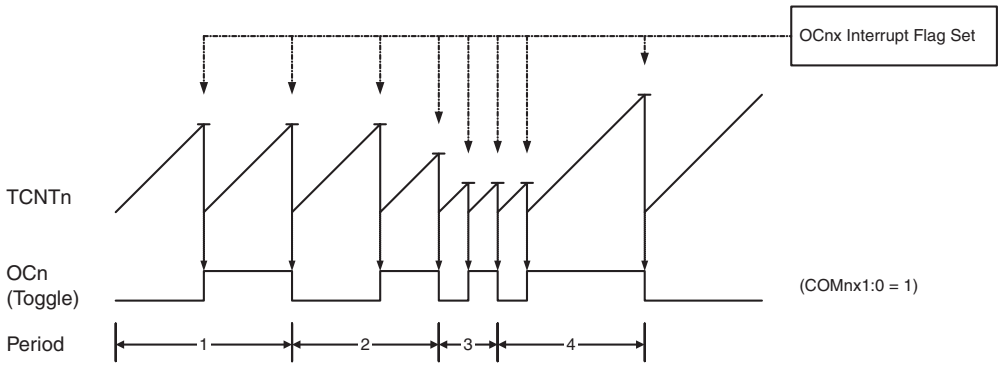
The Output Compare unit can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

### Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC mode (WGM01:0 = 2), 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 31. The counter value (TCNT0) increases until a compare match occurs between TCNT0 and OCR0A, and then counter (TCNT0) is cleared.

**Figure 31.** 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.

For generating a waveform output in CTC mode, the OC0A output can be set to toggle its logical level on each compare match by setting the Compare Output mode bits to toggle mode (COM0A1:0 = 1). The OC0A value will not be visible on the port pin unless the data direction for the pin is set to output. The waveform generated will have a maximum frequency of $f_{OC0} = f_{clk_I/O}/2$ when OCR0A is set to zero (0x00). The waveform frequency is defined by the following equation:

$$f_{OCnx} = \frac{f_{clk_I/O}}{2 \cdot N \cdot (1 + OCRnx)}$$

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

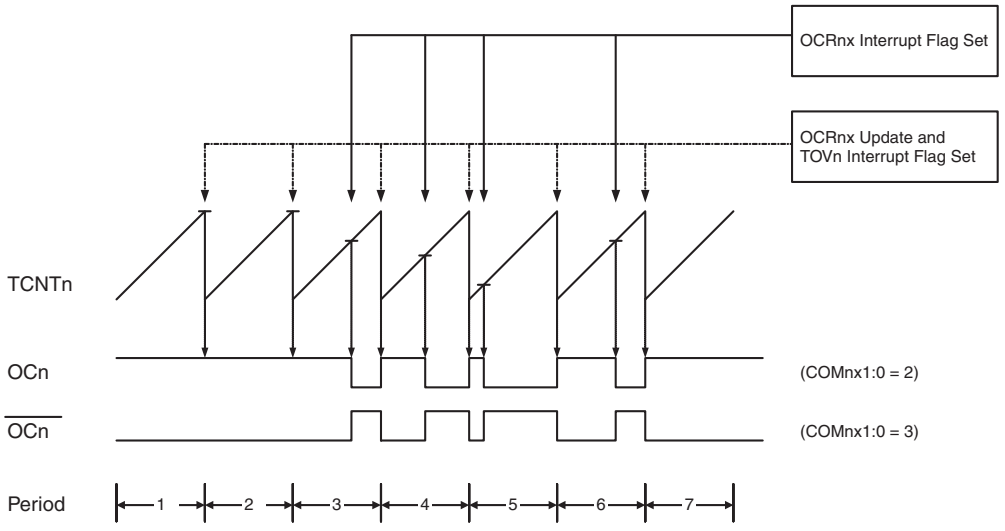
As for the Normal mode of operation, the TOV0 Flag is set in the same timer clock cycle that the counter counts from MAX to 0x00.

### Fast PWM Mode

The fast Pulse Width Modulation or fast PWM mode (WGM01:0 = 3) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM option by its single-slope operation. The counter counts from BOTTOM to MAX then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC0A) is cleared on the compare match between TCNT0 and OCR0A, and set at BOTTOM. In inverting Compare Output mode, the output is set on compare match and cleared at BOTTOM. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct PWM mode 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), and therefore reduces total system cost.

In fast PWM mode, the counter is incremented until the counter value matches the MAX value. The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 32. The TCNT0 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 TCNT0 slopes represent compare matches between OCR0A and TCNT0.

**Figure 32.** Fast PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV0) is set each time the counter reaches MAX. If the interrupt is enabled, the interrupt handler routine can be used for updating the compare value.

In fast PWM mode, the compare unit allows generation of PWM waveforms on the OC0A pin. Setting the COM0A1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM0A1:0 to three (See Table 49 on page 91). The actual OC0A value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by setting (or clearing) the OC0A Register at the compare match between OCR0A and TCNT0, and clearing (or setting) the OC0A Register at the timer clock cycle the counter is cleared (changes from MAX to BOTTOM).

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

$$f_{OCnxPWM} = \frac{f_{clk_I/O}}{N \cdot 256}$$

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

The extreme values for the OCR0A Register represents special cases when generating a PWM waveform output in the fast PWM mode. If the OCR0A is set equal to BOTTOM, the output will be a narrow spike for each MAX+1 timer clock cycle. Setting the OCR0A equal to MAX will result in a constantly high or low output (depending on the polarity of the output set by the COM0A1:0 bits.)

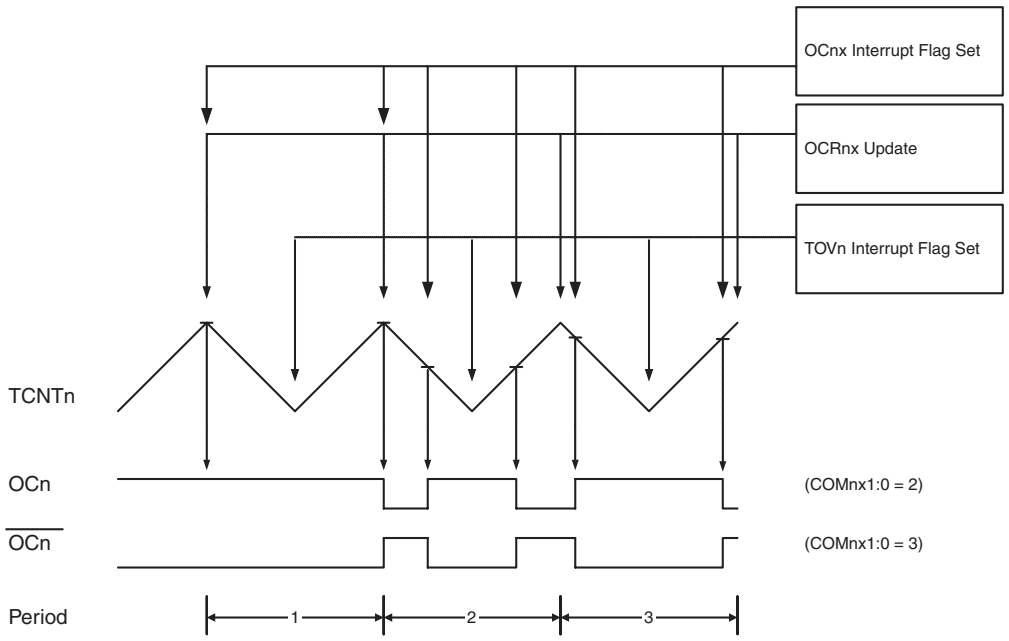
A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC0A to toggle its logical level on each compare match (COM0A1:0 = 1). The waveform generated will have a maximum frequency of $f_{OC0} = f_{clk_I/O}/2$ when OCR0A is set to zero. This feature is similar to the OC0A toggle in CTC mode, except the double buffer feature of the Output Compare unit is enabled in the fast PWM mode.

**Phase Correct PWM Mode**

The phase correct PWM mode (WGM01:0 = 1) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is based on a dual-slope operation. The counter counts repeatedly from BOTTOM to MAX and then from MAX to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC0A) is cleared on the compare match between TCNT0 and OCR0A while counting up, and set on the compare match while counting down. 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 is fixed to eight bits. In phase correct PWM mode the counter is incremented until the counter value matches MAX. When the counter reaches MAX, it changes the count direction. The TCNT0 value will be equal to MAX for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 33. The TCNT0 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 TCNT0 slopes represent compare matches between OCR0A and TCNT0.

**Figure 33.** Phase Correct PWM Mode, Timing Diagram




The Timer/Counter Overflow Flag (TOV0) is set each time the counter reaches BOTTOM. The Interrupt Flag can be used to generate an interrupt each time the counter reaches the BOTTOM value.

In phase correct PWM mode, the compare unit allows generation of PWM waveforms on the OC0A pin. Setting the COM0A1:0 bits to two will produce a non-inverted PWM. An inverted PWM output can be generated by setting the COM0A1:0 to three (See Table 50 on page 92). The actual OC0A value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by clearing (or setting) the OC0A Register at the compare match between OCR0A and TCNT0 when the counter increments, and setting (or clearing) the OC0A Register at compare match between OCR0A and TCNT0 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnxPCPWM} = \frac{f_{clk_I/O}}{N \cdot 510}$$

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

The extreme values for the OCR0A Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR0A is set equal to BOTTOM, the output will be continuously low and if set equal to MAX the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

At the very start of period 2 in Figure 33 OCn has a transition from high to low even though there is no Compare Match. The point of this transition is to guarantee symmetry around BOTTOM. There are two cases that give a transition without Compare Match.

- OCR0A changes its value from MAX, like in Figure 33. When the OCR0A value is MAX the OCn pin value is the same as the result of a down-counting Compare Match. To ensure symmetry around BOTTOM the OCn value at MAX must correspond to the result of an up-counting Compare Match.

- The timer starts counting from a value higher than the one in OCR0A, and for that reason misses the Compare Match and hence the OCn change that would have happened on the way up.

## 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 34 contains timing data for basic Timer/Counter operation. The figure shows the count sequence close to the MAX value in all modes other than phase correct PWM mode.

**Figure 34.** Timer/Counter Timing Diagram, no Prescaling

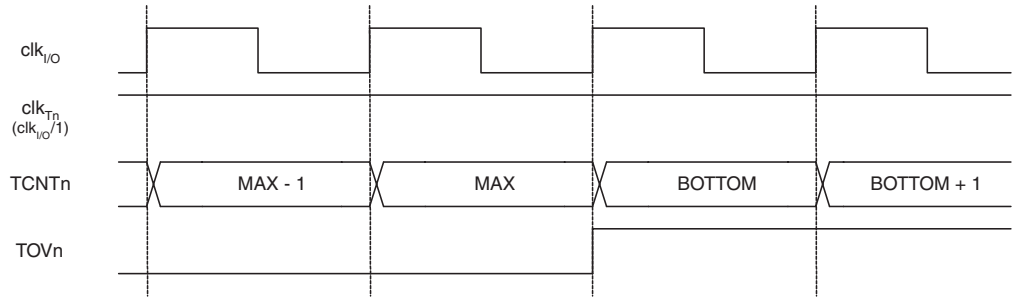



Figure 35 shows the same timing data, but with the prescaler enabled.

**Figure 35.** Timer/Counter Timing Diagram, with Prescaler ($f_{clk_I/O}/8$)

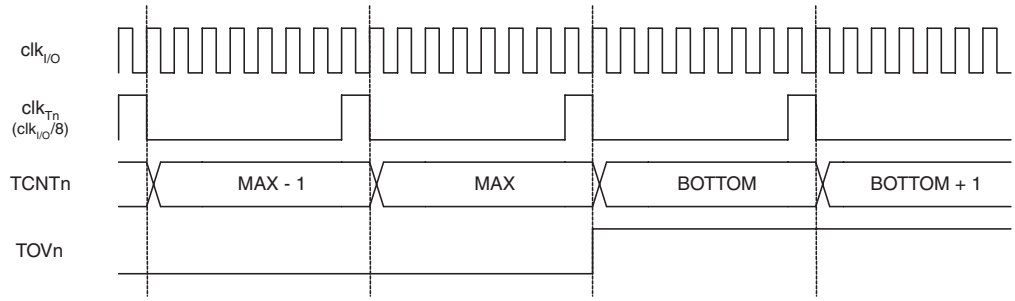



Figure 36 shows the setting of OCF0A in all modes except CTC mode.

**Figure 36.** Timer/Counter Timing Diagram, Setting of OCF0A, with Prescaler ($f_{clk_I/O}/8$)

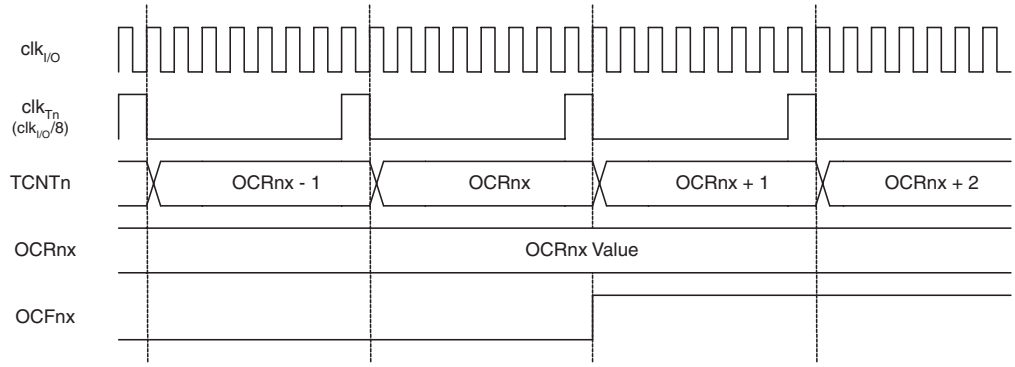
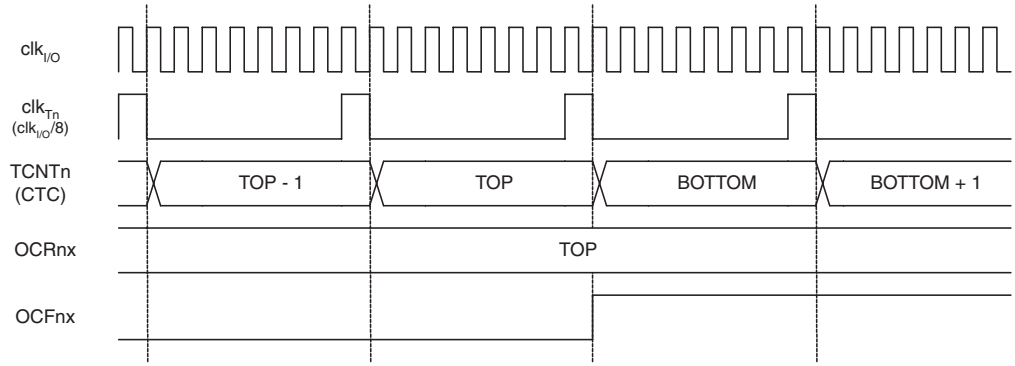


Figure 37 shows the setting of OCF0A and the clearing of TCNT0 in CTC mode.

**Figure 37.** Timer/Counter Timing Diagram, Clear Timer on Compare Match mode, with Prescaler ($f_{clk_I/O}/8$)




## 8-bit Timer/Counter Register Description

### Timer/Counter Control Register A – TCCR0A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | FOC0A | WGM00 | COM0A1 | COM0A0 | WGM01 | CS02 | CS01 | CS00 | TCCR0 |
| Read/Write | 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 – FOC0A: Force Output Compare A**

The FOC0A bit is only active when the WGM00 bit specifies a non-PWM mode. However, for ensuring compatibility with future devices, this bit must be set to zero when TCCR0 is written when operating in PWM mode. When writing a logical one to the FOC0A bit, an immediate compare match is forced on the Waveform Generation unit. The OC0A output is changed according to its COM0A1:0 bits setting. Note that the FOC0A bit is implemented as a strobe. Therefore it is the value present in the COM0A1:0 bits that determines the effect of the forced compare.

A FOC0A strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR0A as TOP.

The FOC0A bit is always read as zero.

- **Bit 6, 3 – WGM01:0: Waveform Generation Mode**

These bits control the counting sequence of the counter, the source for the maximum (TOP) counter value, and what type of waveform generation to be used. Modes of operation supported by the Timer/Counter unit are: Normal mode, Clear Timer on Compare match (CTC) mode, and two types of Pulse Width Modulation (PWM) modes. See Table 47 and “Modes of Operation” on page 85.

**Table 47.** Waveform Generation Mode Bit Description[1]

| Mode | WGM01 (CTC0) | WGM00 (PWM0) | Timer/Counter Mode of Operation | TOP | Update of OCR0A at | TOV0 Flag Set on |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | Normal | 0xFF | Immediate | MAX |
| 1 | 0 | 1 | PWM, Phase Correct | 0xFF | TOP | BOTTOM |
| 2 | 1 | 0 | CTC | OCR0A | Immediate | MAX |
| 3 | 1 | 1 | Fast PWM | 0xFF | TOP | MAX |

Note: 1. The CTC0 and PWM0 bit definition names are now obsolete. Use the WGM01:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

- **Bit 5:4 – COM01:0: Compare Match Output Mode**

These bits control the Output Compare pin (OC0A) behavior. If one or both of the COM0A1:0 bits are set, the OC0A 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 OC0A pin must be set in order to enable the output driver.

When OC0A is connected to the pin, the function of the COM0A1:0 bits depends on the WGM01:0 bit setting. Table 48 shows the COM0A1:0 bit functionality when the WGM01:0 bits are set to a normal or CTC mode (non-PWM).

**Table 48.** Compare Output Mode, non-PWM Mode

| COM0A1 | COM0A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC0A disconnected. |
| 0 | 1 | Toggle OC0A on compare match |
| 1 | 0 | Clear OC0A on compare match |
| 1 | 1 | Set OC0A on compare match |

Table 49 shows the COM0A1:0 bit functionality when the WGM01:0 bits are set to fast PWM mode.

**Table 49.** Compare Output Mode, Fast PWM Mode[1]

| COM0A1 | COM0A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC0A disconnected. |
| 0 | 1 | Reserved |
| 1 | 0 | Clear OC0A on compare match, set OC0A at TOP |
| 1 | 1 | Set OC0A on compare match, clear OC0A at TOP |

Note: 1. A special case occurs when OCR0A equals TOP and COM0A1 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See “Fast PWM Mode” on page 86 for more details.

Table 50 shows the COM0A1:0 bit functionality when the WGM01:0 bits are set to phase correct PWM mode.

**Table 50.** Compare Output Mode, Phase Correct PWM Mode[1]

| COM0A1 | COM0A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC0A disconnected. |
| 0 | 1 | Reserved |
| 1 | 0 | Clear OC0A on compare match when up-counting. Set OC0A on compare match when counting down. |
| 1 | 1 | Set OC0A on compare match when up-counting. Clear OC0A on compare match when counting down. |

Note: 1. A special case occurs when OCR0A equals TOP and COM0A1 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See “Phase Correct PWM Mode” on page 87 for more details.

- **Bit 2:0 – CS02:0: Clock Select**

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

**Table 51.** 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.

### Timer/Counter Register – TCNT0

| 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 OCR0A Register.

### Output Compare Register A – OCR0A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | OCR0A[7:0] | | | | | | | | OCR0 |
| 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, or to generate a waveform output on the OC0A pin.

### Timer/Counter 0 Interrupt Mask Register – TIMSK0

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

- **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 (one), 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 (one), 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.

### Timer/Counter 0 Interrupt Flag Register – TIFR0

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

- **Bit 1 – OCF0A: Output Compare Flag 0 A**

The OCF0A bit is set (one) 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 (one), the Timer/Counter0 Compare match Interrupt is executed.

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

The bit TOV0 is set (one) 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 (one), the Timer/Counter0 Overflow interrupt is executed. In phase correct PWM mode, this bit is set when Timer/Counter0 changes counting direction at 0x00.

## Timer/Counter0 and Timer/Counter1 Prescalers

Timer/Counter1 and Timer/Counter0 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.

### 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 $f_{CLK_I/O}/1024$.

### 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 (6 > CSn2:0 > 1). 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).

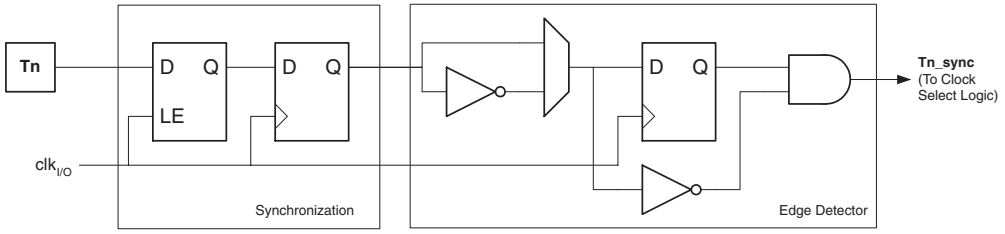
It is possible to use the prescaler reset for synchronizing the Timer/Counter to program execution. However, care must be taken if the other Timer/Counter that shares the same prescaler also uses prescaling. A prescaler reset will affect the prescaler period for all Timer/Counters it is connected to.

### 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 38 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 38.** 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.

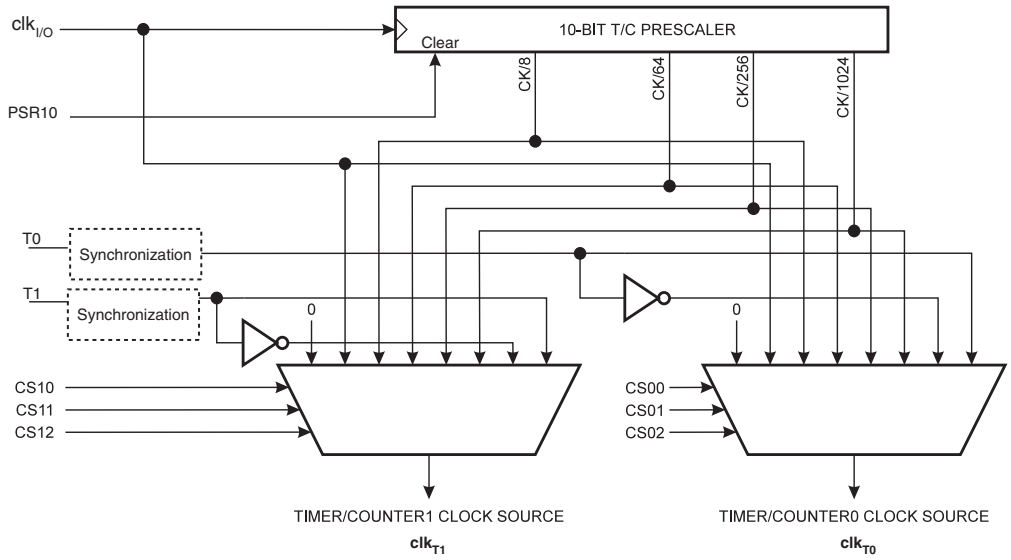
Enabling and disabling of the clock input must be done when T1/T0 has been stable for at least one system clock cycle, otherwise it is a risk that a false Timer/Counter clock pulse is generated.

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 50/50% 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 (crystal, resonator, and capacitors) 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 39.** Prescaler for Timer/Counter0 and Timer/Counter1[1]




Note: 1. The synchronization logic on the input pins (T1/T0) is shown in Figure 38.

## General Timer/Counter Control Register – GTCCR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | TSM | – | – | – | – | – | PSR2 | PSR10 | GTCCR |
| Read/Write | R/W | R | R | R | R | R | R/W | 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 PSR2 and PSR10 bits 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 PSR2 and PSR10 bits are cleared by hardware, and the Timer/Counters start counting simultaneously.

- **Bit 0 – PSR10: Prescaler Reset Timer/Counter1 and Timer/Counter0**

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.

## 16-bit Timer/Counter1

The 16-bit Timer/Counter unit allows accurate program execution timing (event management), wave generation, and signal timing measurement. The main features are:

- **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)**

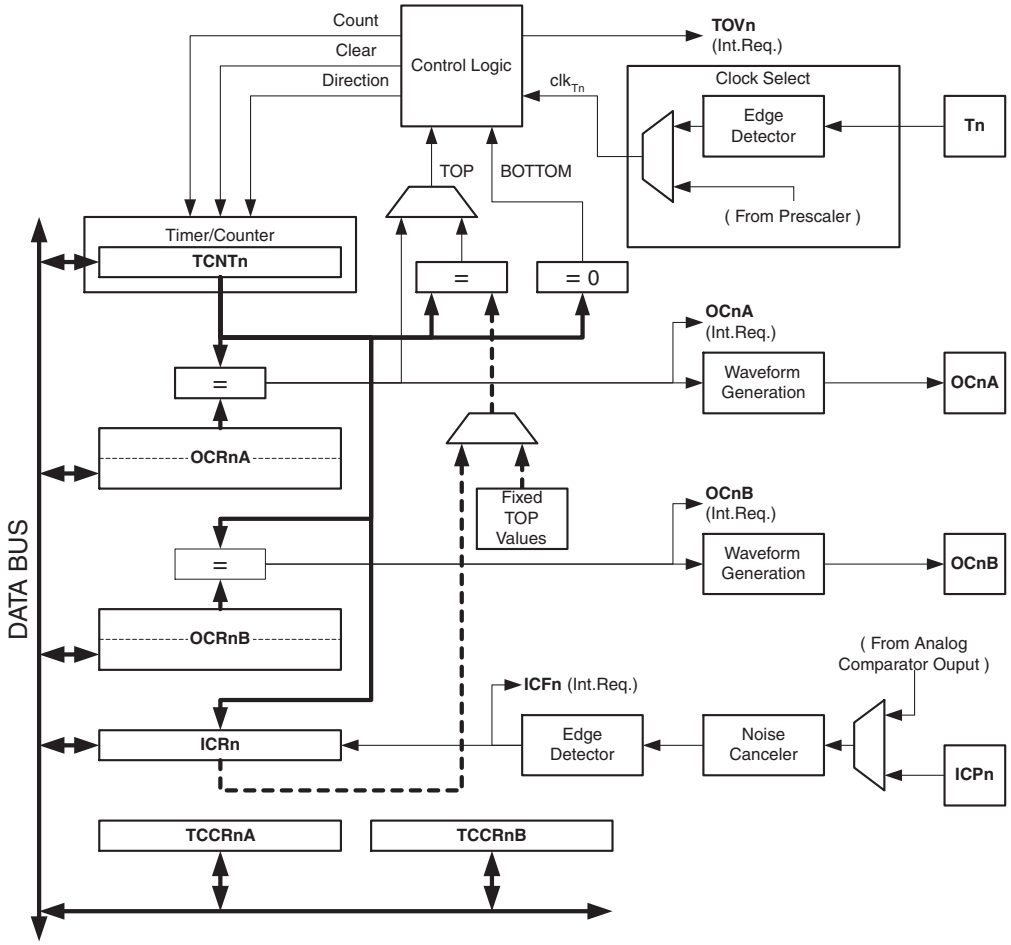
## Overview

Most register and bit references in this section are written in general form. A lower case “n” replaces the Timer/Counter number, and a lower case “x” replaces the Output Compare unit. 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.

A simplified block diagram of the 16-bit Timer/Counter is shown in Figure 40. For the actual placement of I/O pins, refer to “Pinout ATmega3250/6450” on page 2. 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 “16-bit Timer/Counter Register Description” on page 117.

The PRTIM1 bit in “Power Reduction Register” on page 35 must be written to zero to enable the Timer/Counter1 module.

**Figure 40.** 16-bit Timer/Counter Block Diagram[1]




Note: 1. Refer to Figure 1 on page 2, “Alternate Functions of Port D” on page 62, and “Alternate Functions of Port G” on page 67for Timer/Counter1 pin placement and description.

**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 “Accessing 16-bit Registers” on page 99. 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 “Output Compare Units” on page 105.. 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 “Analog Comparator” on page 190.) 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.

**Definitions**

The following definitions are used extensively throughout the section:

**Table 52.** Definitions of Timer/Counter values.

| | |
|---|---|
| BOTTOM | The counter reaches the *BOTTOM* when it becomes 0x0000. |
| MAX | The counter reaches its *MAX*imum 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. |

**Compatibility**

The 16-bit Timer/Counter has been updated and improved from previous versions of the 16-bit AVR Timer/Counter. This 16-bit Timer/Counter is fully compatible with the earlier version regarding:

- All 16-bit Timer/Counter related I/O Register address locations, including Timer Interrupt Registers.
- Bit locations inside all 16-bit Timer/Counter Registers, including Timer Interrupt Registers.
- Interrupt Vectors.

The following control bits have changed name, but have same functionality and register location:

- PWM10 is changed to WGM10.
- PWM11 is changed to WGM11.
- CTC1 is changed to WGM12.

The following bits are added to the 16-bit Timer/Counter Control Registers:

- FOC1A and FOC1B are added to TCCR1C.
- WGM13 is added to TCCR1B.

The 16-bit Timer/Counter has improvements that will affect the compatibility in some special cases.

## 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. See “About Code Examples” on page 8.

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 */
  __disable_interrupt();
  /* Read TCNT1 into i */
  i = TCNT1;
  /* Restore global interrupt flag */
  SREG = sreg;
  return i;
}
```

Note: 1. See “About Code Examples” on page 8.

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 */
  __disable_interrupt();
  /* Set TCNT1 to i */
  TCNT1 = i;
  /* Restore global interrupt flag */
  SREG = sreg;
}
```

Note: 1. See “About Code Examples” on page 8.

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

### 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.

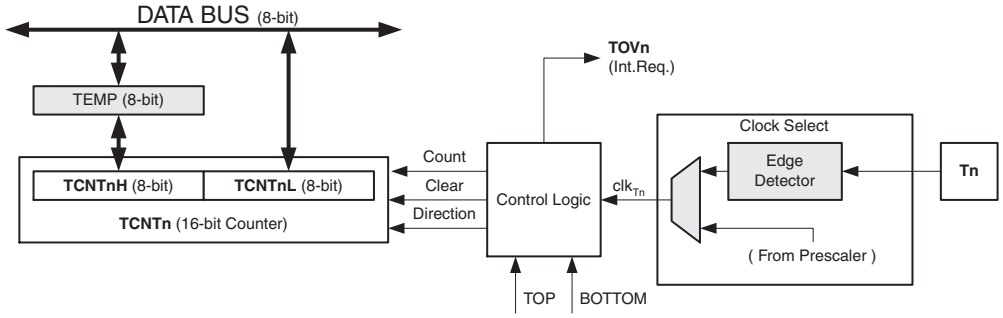
## Timer/Counter Clock Sources

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 “Timer/Counter0 and Timer/Counter1 Prescalers” on page 94.

## Counter Unit

The main part of the 16-bit Timer/Counter is the programmable 16-bit bi-directional counter unit. Figure 41 shows a block diagram of the counter and its surroundings.

**Figure 41.** 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 “Modes of Operation” on page 108.

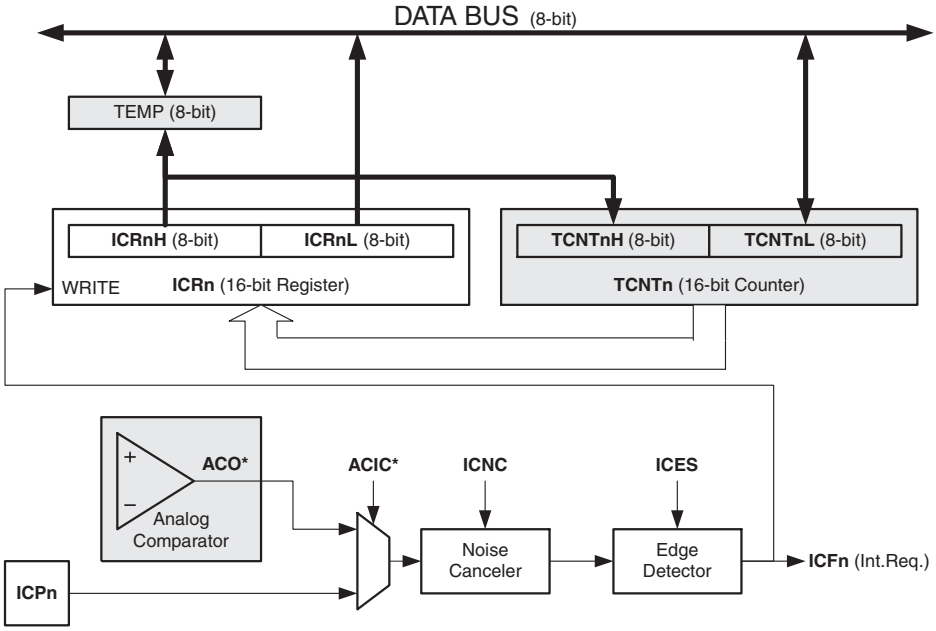
The Timer/Counter Overflow Flag (TOV1) is set according to the mode of operation selected by the WGM13:0 bits. TOV1 can be used for generating a CPU interrupt.

## 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 42. 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 42.** 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 "Accessing 16-bit Registers" on page 99.

**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 38 on page 94). 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.

**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.

**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).

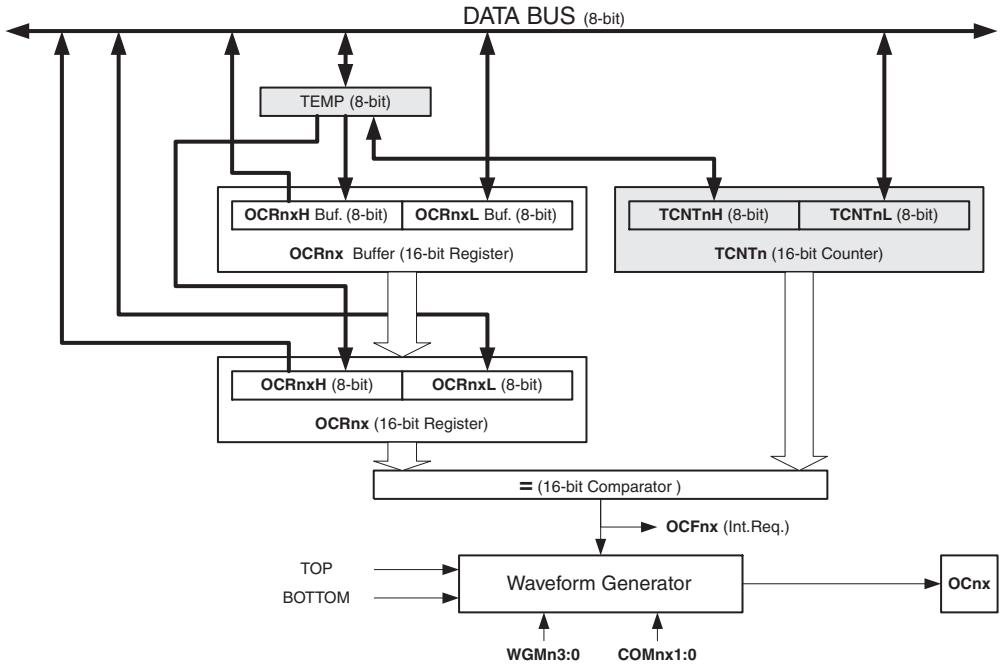
## Output Compare Units

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 “Modes of Operation” on page 108.)

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 43 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 43.** 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 “Accessing 16-bit Registers” on page 99.

**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).

**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.

**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 units, 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 counting down.

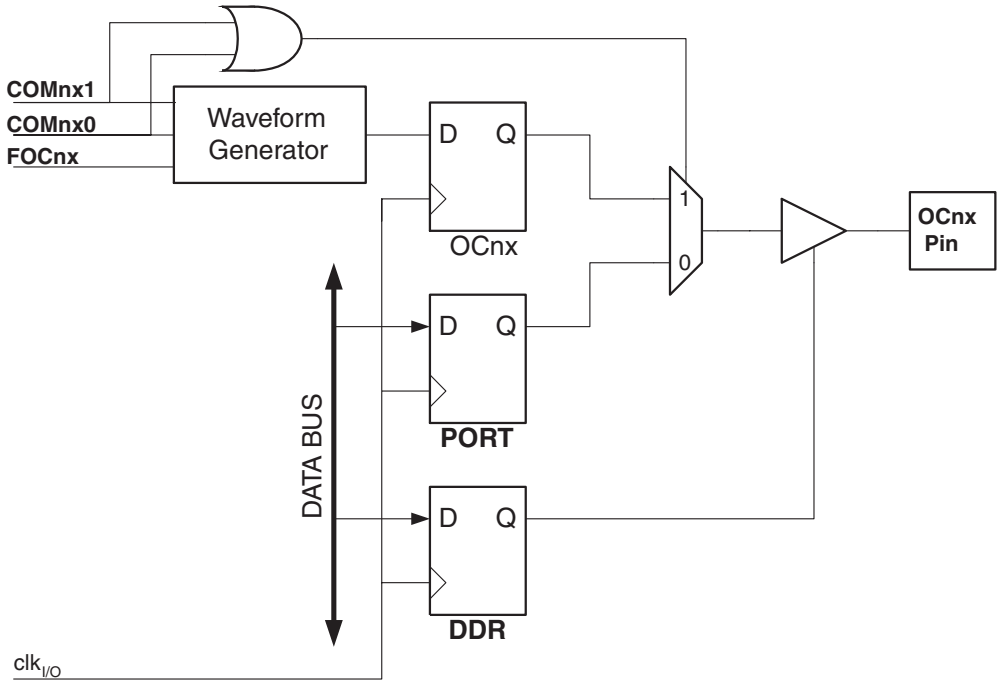
The setup of the OC1x should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC1x value is to use the Force Output Compare (FOC1x) strobe bits in Normal mode. The OC1x Register keeps its value even when changing between Waveform Generation 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.

## 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 44 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 44.** Compare Match Output Unit, 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 53, Table 54 and Table 55 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 “16-bit Timer/Counter Register Description” on page 117.

The COM1x1:0 bits have no effect on the Input Capture unit.

### 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 53 on page 117. For fast PWM mode refer to Table 54 on page 118, and for phase correct and phase and frequency correct PWM refer to Table 55 on page 118.

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.

## 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 “Compare Match Output Unit” on page 107.)

For detailed timing information refer to “Timer/Counter Timing Diagrams” on page 115.

### 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.

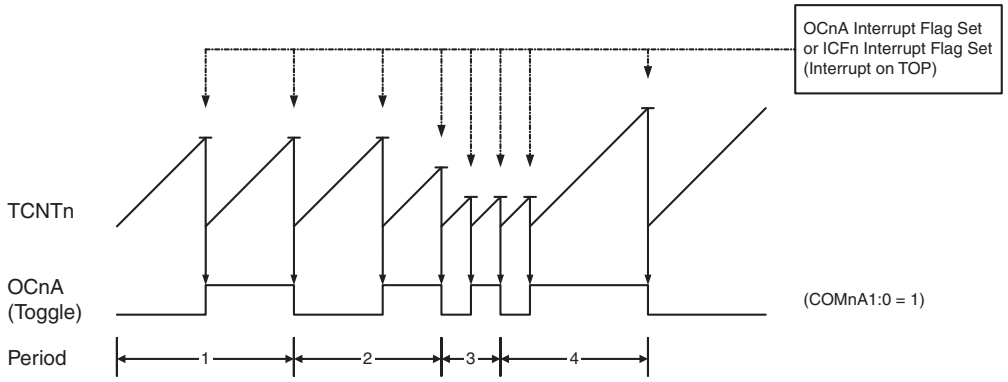
The Output Compare units can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

### 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 45. The counter value (TCNT1) increases until a compare match occurs with either OCR1A or ICR1, and then counter (TCNT1) is cleared.

**Figure 45.** 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.

**Fast PWM Mode**

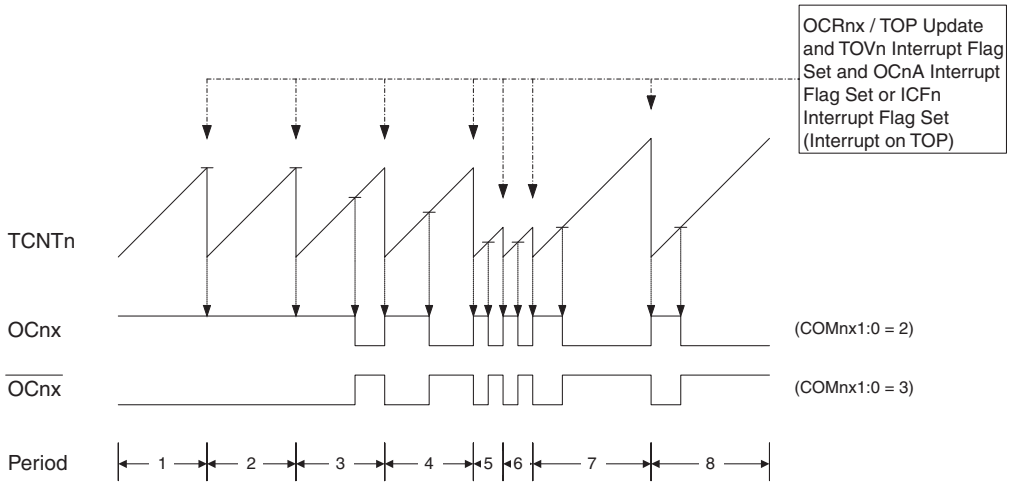
The *fast Pulse Width Modulation* or fast PWM mode (WGM13:0 = 5, 6, 7, 14, or 15) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. In non-inverting 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 46. 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 46.** 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 54 on page 118). 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.

**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 counting up, and set on the compare match while counting down. 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.

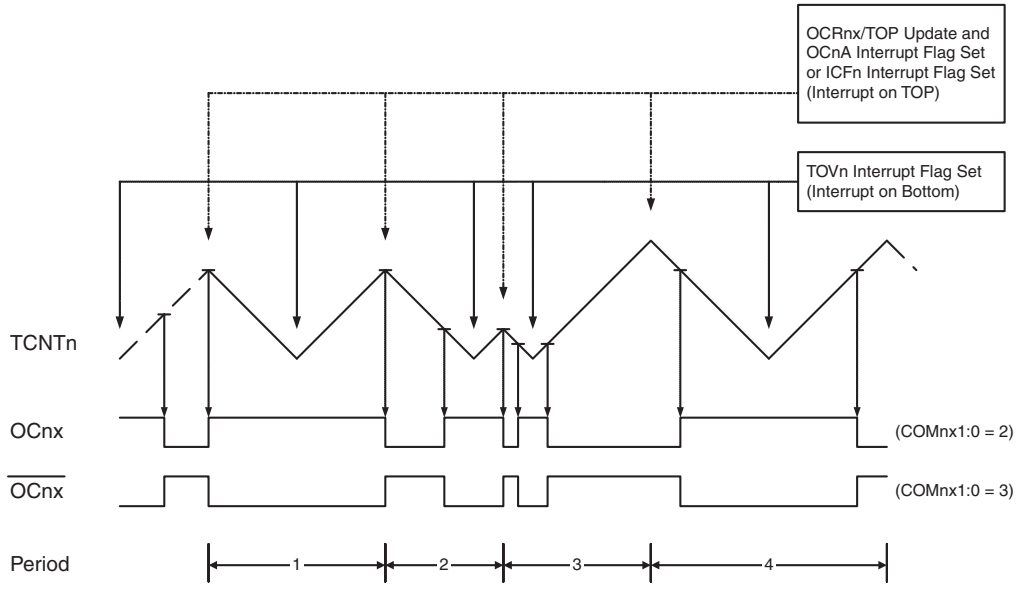
The PWM resolution for the phase correct PWM mode can be fixed to 8-, 9-, or 10-bit, or defined by either ICR1 or OCR1A. The minimum resolution allowed is 2-bit (ICR1 or

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 47. 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 47.** 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 47 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 fall-

ing 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 1 on page 118). 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.

**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 counting up, and set on the compare match while counting down. 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.

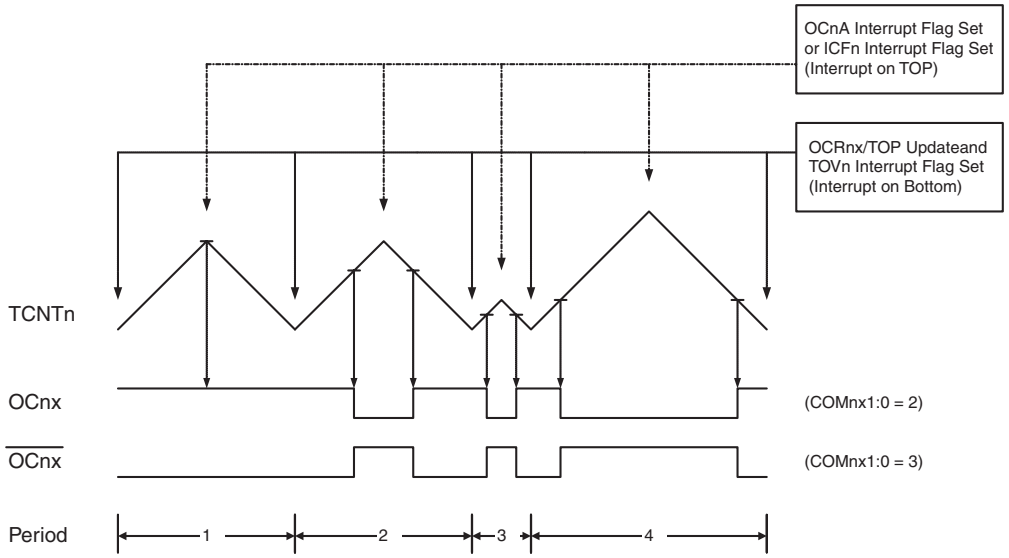
The main difference between the phase correct, and the phase and frequency correct PWM mode is the time the OCR1x Register is updated by the OCR1x Buffer Register, (see Figure 47 and Figure 48).

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 48. 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 48.** 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 48 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 two will produce a

non-inverted PWM and an inverted PWM output can be generated by setting the COM1x1:0 to three (See Table 1 on page 118). 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:

$$f_{OCnxPFCPWM} = \frac{f_{clk_I/O}}{2 \cdot N \cdot TOP}$$

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 (WGM13:0 = 9) and COM1A1:0 = 1, the OC1A output will toggle with a 50% duty cycle.

## 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 49 shows a timing diagram for the setting of OCF1x.

**Figure 49.** Timer/Counter Timing Diagram, Setting of OCF1x, no Prescaling

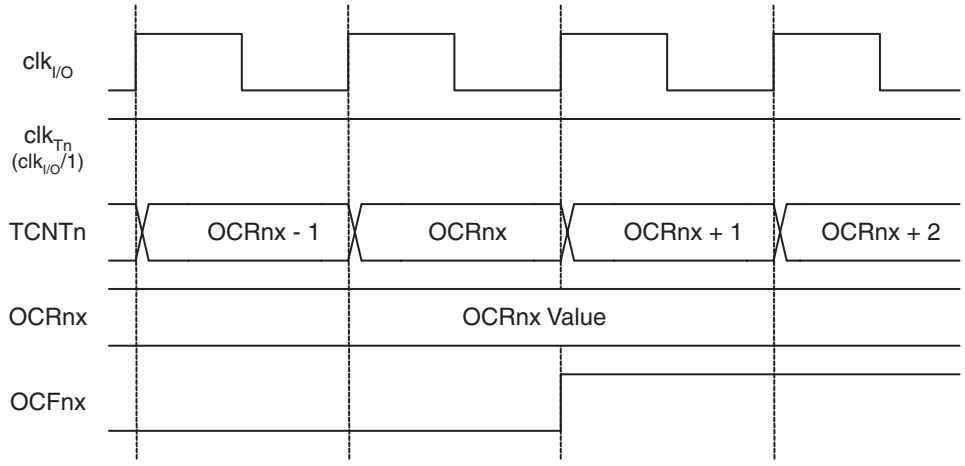



Figure 50 shows the same timing data, but with the prescaler enabled.

**Figure 50.** Timer/Counter Timing Diagram, Setting of OCF1x, with Prescaler ($f_{clk_I/O}/8$)

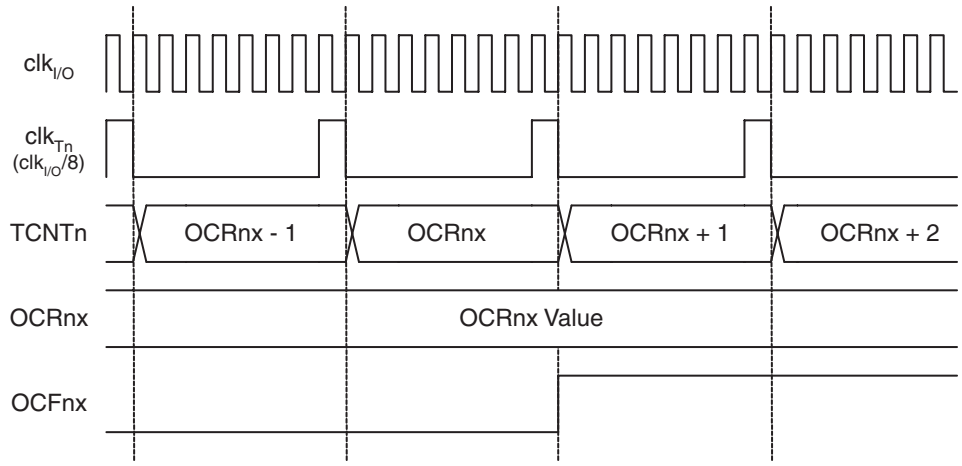



Figure 51 shows the count sequence close to TOP in various modes. When using phase and frequency correct PWM mode the OCR1x Register is updated at BOTTOM. The timing diagrams will be the same, but TOP should be replaced by BOTTOM, TOP-1 by BOTTOM+1 and so on. The same renaming applies for modes that set the TOV1 Flag at BOTTOM.

**Figure 51.** Timer/Counter Timing Diagram, no Prescaling

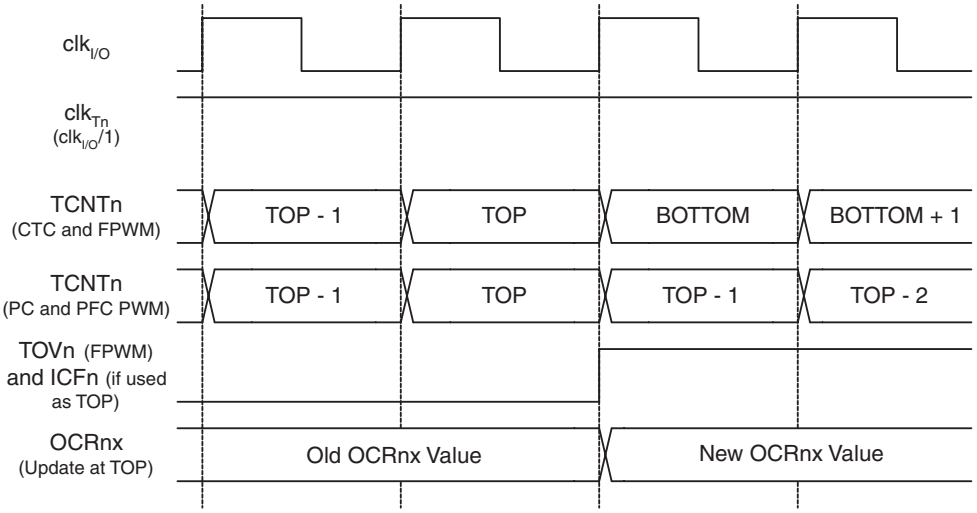
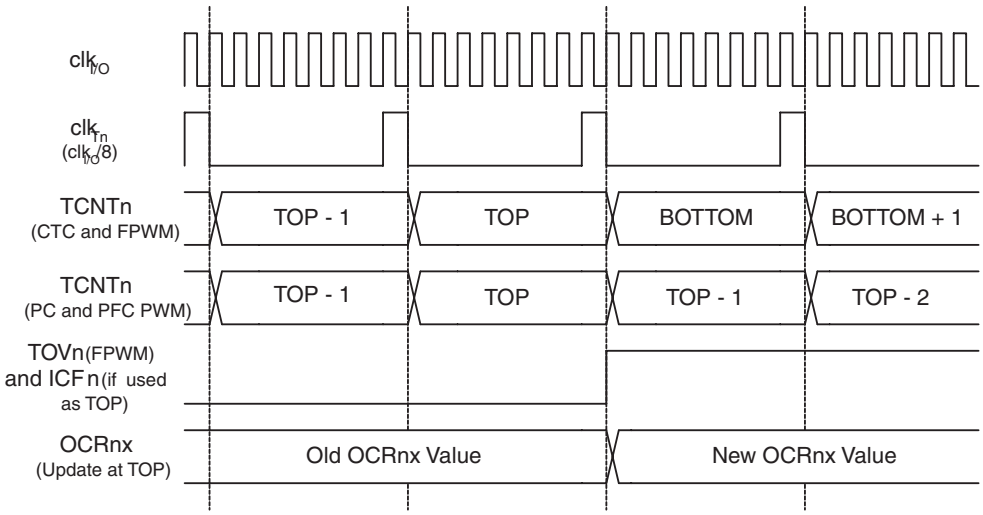



Figure 52 shows the same timing data, but with the prescaler enabled.

**Figure 52.** Timer/Counter Timing Diagram, with Prescaler ($f_{clk_I/O}/8$)




## 16-bit Timer/Counter Register Description

### Timer/Counter1 Control Register A – TCCR1A

| 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 Unit A**
- **Bit 5:4 – COM1B1:0: Compare Output Mode for Unit 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 53 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to a Normal or a CTC mode (non-PWM).

**Table 53.** 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 54 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to the fast PWM mode.

**Table 54.** 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 “Fast PWM Mode” on page 109. for more details.

Table 55 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 55.** Compare Output Mode, Phase Correct and Phase and 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 counting down. |
| 1 | 1 | Set OC1A/OC1B on Compare Match when up-counting. Clear OC1A/OC1B on Compare Match when counting down. |

Note: 1. A special case occurs when OCR1A/OCR1B equals TOP and COM1A1/COM1B1 is set. See “Phase Correct PWM Mode” on page 111. for more details.

- **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 56. 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 “Modes of Operation” on page 108.).

**Table 56.** Waveform Generation Mode Bit Description[1]

| Mode | WGM13 | WGM12 (CTC1) | WGM11 (PWM11) | WGM10 (PWM10) | 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 and Frequency Correct | ICR1 | BOTTOM | BOTTOM |
| 9 | 1 | 0 | 0 | 1 | PWM, Phase and 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 |

Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

### Timer/Counter1 Control Register B – TCCR1B

| 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 49 and Figure 50.

**Table 57.** 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.

### Timer/Counter1 Control Register C – TCCR1C

| 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 Unit A**
- **Bit 6 – FOC1B: Force Output Compare for Unit 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.

### Timer/Counter1 – TCNT1H and TCNT1L

| 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 “Accessing 16-bit Registers” on page 99.

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.

### Output Compare Register 1 A – OCR1AH and OCR1AL

| 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 | |

### Output Compare Register 1 B – OCR1BH and OCR1BL

| 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 “Accessing 16-bit Registers” on page 99.

### Input Capture Register 1 – ICR1H and ICR1L

| 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 “Accessing 16-bit Registers” on page 99.

### Timer/Counter1 Interrupt Mask Register – TIMSK1

| 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 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 “Interrupts” on page 47.) is executed when the ICF1 Flag, located in TIFR1, is set.

- **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 “Interrupts” on page 47.) 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 “Interrupts” on page 47.) 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 “Interrupts” on page 47.) is executed when the TOV1 Flag, located in TIFR1, is set.

### Timer/Counter1 Interrupt Flag Register – TIFR1

| 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 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 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 56 on page 119 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.

## 8-bit Timer/Counter2 with PWM and Asynchronous Operation

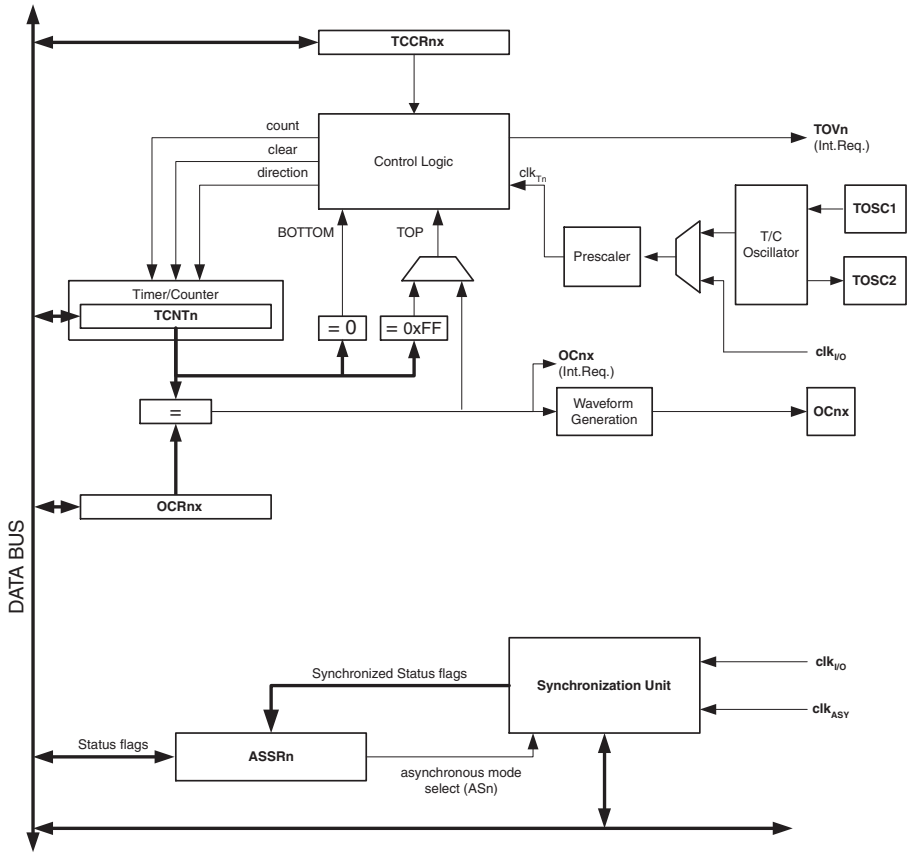
Timer/Counter2 is a general purpose, single compare unit, 8-bit Timer/Counter module. The main features are:

- **Single Compare Unit Counter**
- **Clear Timer on Compare Match (Auto Reload)**
- **Glitch-free, Phase Correct Pulse Width Modulator (PWM)**
- **Frequency Generator**
- **10-bit Clock Prescaler**
- **Overflow and Compare Match Interrupt Sources (TOV2 and OCF2A)**
- **Allows Clocking from External 32 kHz Watch Crystal Independent of the I/O Clock**

### Overview

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 53. For the actual placement of I/O pins, refer to “Pinout ATmega3250/6450” on page 2. 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 “8-bit Timer/Counter Register Description” on page 135.

**Figure 53.** 8-bit Timer/Counter Block Diagram




### Registers

The Timer/Counter (TCNT2) and Output Compare Register (OCR2A) are 8-bit registers. Interrupt request (shorten as Int.Req.) signals are all visible in the Timer Interrupt Flag Register (TIFR2). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK2). TIFR2 and TIMSK2 are not shown in the figure.

The Timer/Counter can be clocked internally, via the prescaler, or asynchronously clocked from the TOSC1/2 pins, as detailed later in this section. The asynchronous operation is controlled by the Asynchronous Status Register (ASSR). The Clock Select logic block controls which clock source the Timer/Counter uses to increment (or decre-

ment) 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 ($clk_{T2}$).

The double buffered Output Compare Register (OCR2A) is compared with the Timer/Counter value at all times. 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 (OC2A). See “Output Compare Unit” on page 126. for details. The compare match event will also set the Compare Flag (OCF2A) which can be used to generate an Output Compare interrupt request.

### Definitions

Many register and bit references in this document are written in general form. A lower case “n” replaces the Timer/Counter number, in this case 2. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT2 for accessing Timer/Counter2 counter value and so on.

The definitions in Table 58 are also used extensively throughout the section.

**Table 58.** Definitions of Timer/Counter values.

| | |
|---|---|
| BOTTOM | The counter reaches the BOTTOM when it becomes zero (0x00). |
| 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 OCR2A Register. The assignment is dependent on the mode of operation. |

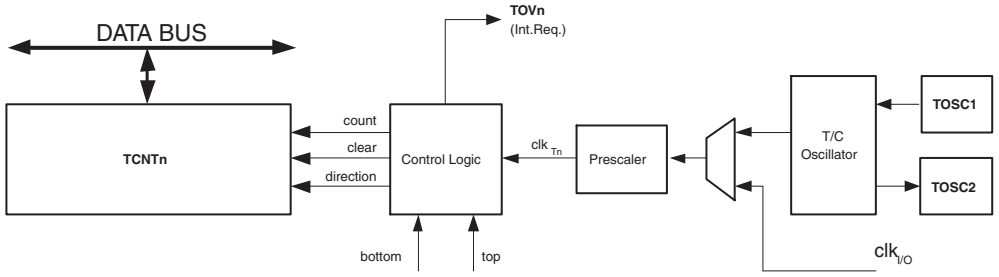
## Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal synchronous or an external asynchronous clock source. The clock source $clk_{T2}$ is by default equal to the MCU clock, $clk_{I/O}$. When the AS2 bit in the ASSR Register is written to logic one, the clock source is taken from the Timer/Counter Oscillator connected to TOSC1 and TOSC2. For details on asynchronous operation, see “Asynchronous Status Register – ASSR” on page 138. For details on clock sources and prescaler, see “Timer/Counter Prescaler” on page 141.

## Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Figure 54 shows a block diagram of the counter and its surrounding environment.

**Figure 54.** Counter Unit Block Diagram




Signal description (internal signals):

**count** Increment or decrement TCNT2 by 1.

**direction** Selects between increment and decrement.

**clear** Clear TCNT2 (set all bits to zero).

**$clk_{T2}$** Timer/Counter clock.

**top** Signalizes that TCNT2 has reached maximum value.

**bottom** Signalizes that TCNT2 has reached minimum value (zero).

Depending on the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock ($clk_{T2}$). $clk_{T2}$ can be generated from an external or internal clock source, selected by the Clock Select bits (CS22:0). When no clock source is selected (CS22:0 = 0) the timer is stopped. However, the TCNT2 value can be accessed by the CPU, regardless of whether $clk_{T2}$ 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 WGM21 and WGM20 bits located in the Timer/Counter Control Register (TCCR2A). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare output OC2A. For more details about advanced counting sequences and waveform generation, see “Modes of Operation” on page 129.

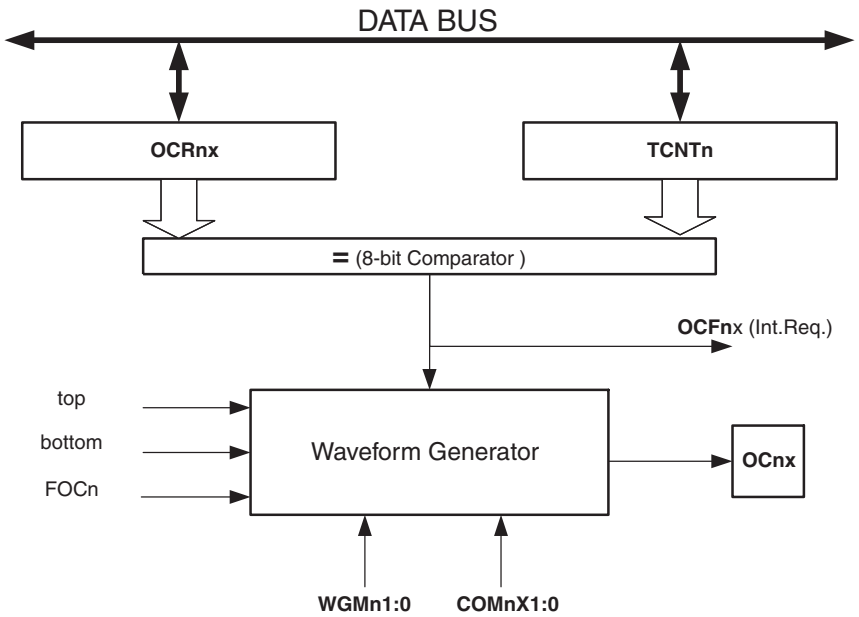
The Timer/Counter Overflow Flag (TOV2) is set according to the mode of operation selected by the WGM21:0 bits. TOV2 can be used for generating a CPU interrupt.

## Output Compare Unit

The 8-bit comparator continuously compares TCNT2 with the Output Compare Register (OCR2A). Whenever TCNT2 equals OCR2A, the comparator signals a match. A match will set the Output Compare Flag (OCF2A) at the next timer clock cycle. If enabled (OCIE2A = 1), the Output Compare Flag generates an Output Compare interrupt. The OCF2A Flag is automatically cleared when the interrupt is executed. Alternatively, the OCF2A 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 WGM21:0 bits and Compare Output mode (COM2A1:0) bits. The max and bottom signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation (“Modes of Operation” on page 129).

Figure 55 shows a block diagram of the Output Compare unit.

**Figure 55.** Output Compare Unit, Block Diagram



The OCR2A Register is double buffered when using any of the 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 OCR2A 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 OCR2A Register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR2A Buffer Register, and if double buffering is disabled the CPU will access the OCR2A directly.

**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 (FOC2A) bit. Forcing compare match will not set the OCF2A Flag or reload/clear the timer, but the OC2A pin will be updated as if a real compare match had occurred (the COM2A1:0 bits settings define whether the OC2A pin is set, cleared or toggled).

**Compare Match Blocking by TCNT2 Write**

All CPU write operations to the TCNT2 Register will block any compare match that occurs in the next timer clock cycle, even when the timer is stopped. This feature allows OCR2A to be initialized to the same value as TCNT2 without triggering an interrupt when the Timer/Counter clock is enabled.

**Using the Output Compare Unit**

Since writing TCNT2 in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNT2 when using the Output Compare unit, independently of whether the Timer/Counter is running or not. If the value written to TCNT2 equals the OCR2A value, the compare match will be missed, resulting in incorrect waveform generation. Similarly, do not write the TCNT2 value equal to BOTTOM when the counter is counting down.

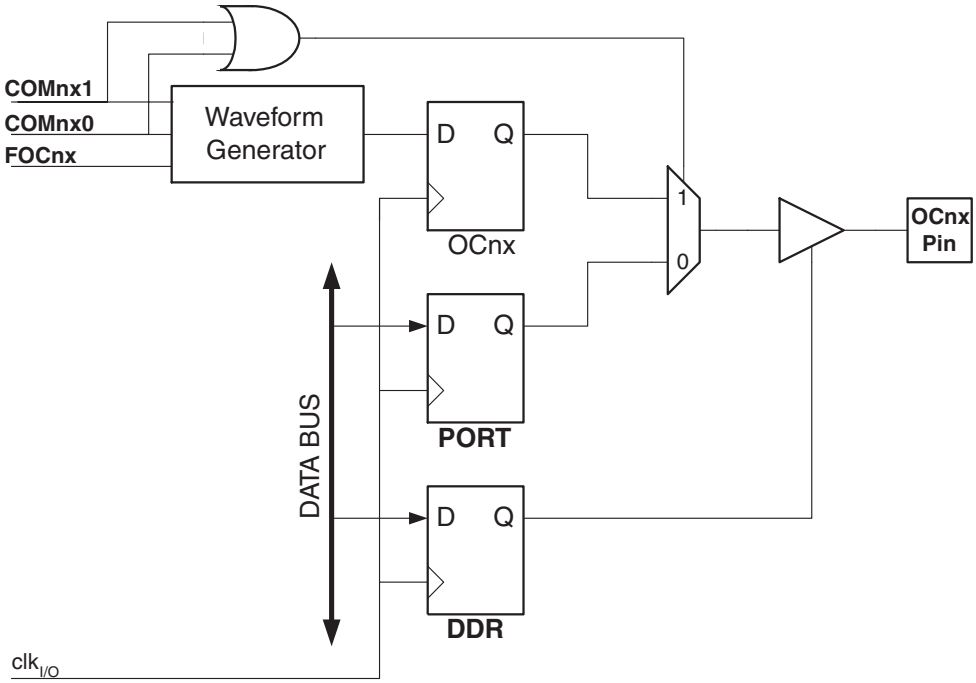
The setup of the OC2A should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC2A value is to use the Force Output Compare (FOC2A) strobe bit in Normal mode. The OC2A Register keeps its value even when changing between Waveform Generation modes.

Be aware that the COM2A1:0 bits are not double buffered together with the compare value. Changing the COM2A1:0 bits will take effect immediately.

## Compare Match Output Unit

The Compare Output mode (COM2A1:0) bits have two functions. The Waveform Generator uses the COM2A1:0 bits for defining the Output Compare (OC2A) state at the next compare match. Also, the COM2A1:0 bits control the OC2A pin output source. Figure 56 shows a simplified schematic of the logic affected by the COM2A1:0 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 COM2A1:0 bits are shown. When referring to the OC2A state, the reference is for the internal OC2A Register, not the OC2A pin.

**Figure 56.** Compare Match Output Unit, Schematic




The general I/O port function is overridden by the Output Compare (OC2A) from the Waveform Generator if either of the COM2A1:0 bits are set. However, the OC2A 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 OC2A pin (DDR_OC2A) must be set as output before the OC2A value is visible on the pin. The port override function is independent of the Waveform Generation mode.

The design of the Output Compare pin logic allows initialization of the OC2A state before the output is enabled. Note that some COM2A1:0 bit settings are reserved for certain modes of operation. See “8-bit Timer/Counter Register Description” on page 135.

### Compare Output Mode and Waveform Generation

The Waveform Generator uses the COM2A1:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COM2A1:0 = 0 tells the Waveform Generator that no action on the OC2A Register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 60 on page 136. For fast PWM mode, refer to Table 61 on page 136, and for phase correct PWM refer to Table 62 on page 136.

A change of the COM2A1:0 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 FOC2A strobe bits.

## 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 (WGM21:0) and Compare Output mode (COM2A1:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM2A1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM2A1:0 bits control whether the output should be set, cleared, or toggled at a compare match (See “Compare Match Output Unit” on page 128.).

For detailed timing information refer to “Timer/Counter Timing Diagrams” on page 133.

### Normal Mode

The simplest mode of operation is the Normal mode (WGM21:0 = 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 8-bit value (TOP = 0xFF) and then restarts from the bottom (0x00). In normal operation the Timer/Counter Overflow Flag (TOV2) will be set in the same timer clock cycle as the TCNT2 becomes zero. The TOV2 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 TOV2 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.

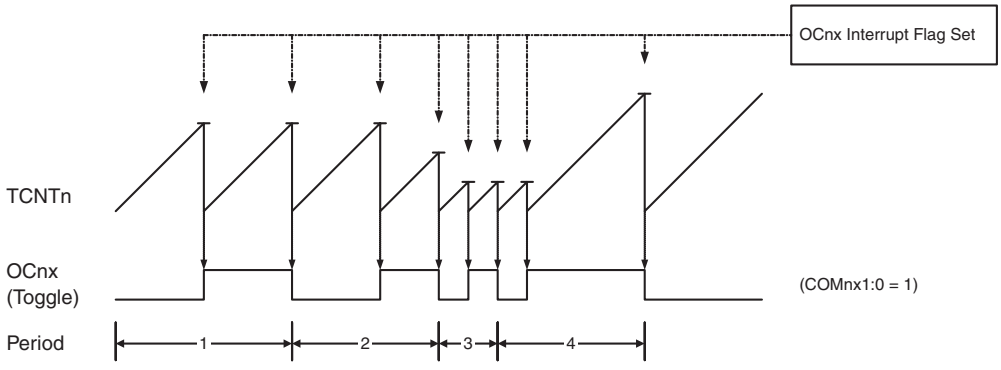
The Output Compare unit can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

### Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC mode (WGM21:0 = 2), the OCR2A Register is used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT2) matches the OCR2A. The OCR2A 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 57. The counter value (TCNT2) increases until a compare match occurs between TCNT2 and OCR2A, and then counter (TCNT2) is cleared.

**Figure 57.** CTC Mode, Timing Diagram




An interrupt can be generated each time the counter value reaches the TOP value by using the OCF2A Flag. 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 OCR2A is lower than the current value of TCNT2, 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.

For generating a waveform output in CTC mode, the OC2A output can be set to toggle its logical level on each compare match by setting the Compare Output mode bits to toggle mode (COM2A1:0 = 1). The OC2A value will not be visible on the port pin unless the data direction for the pin is set to output. The waveform generated will have a maximum frequency of $f_{OC2A} = f_{clk_I/O}/2$ when OCR2A is set to zero (0x00). The waveform frequency is defined by the following equation:

$$f_{OCnx} = \frac{f_{clk_I/O}}{2 \cdot N \cdot (1 + OCRnx)}$$

The *N* variable represents the prescale factor (1, 8, 32, 64, 128, 256, or 1024).

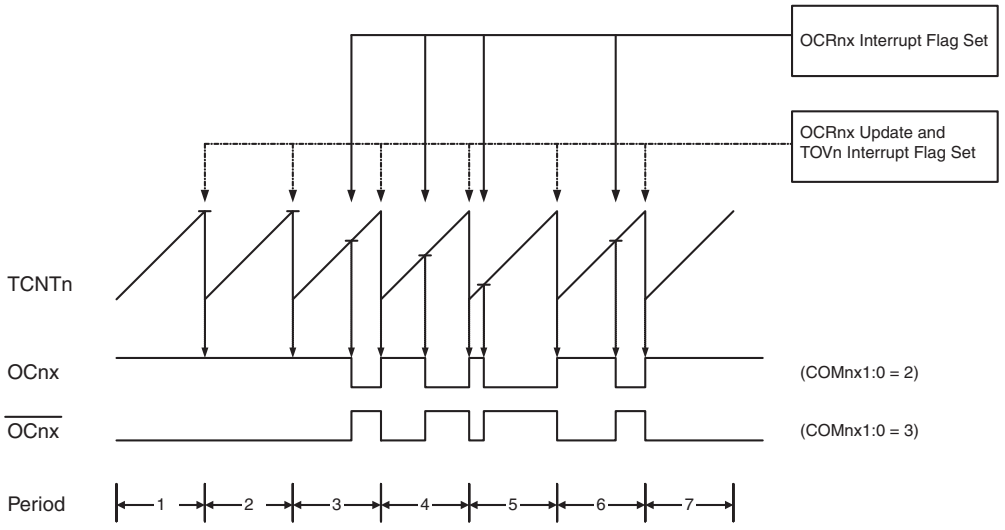
As for the Normal mode of operation, the TOV2 Flag is set in the same timer clock cycle that the counter counts from MAX to 0x00.

**Fast PWM Mode**

The fast Pulse Width Modulation or fast PWM mode (WGM21:0 = 3) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM option by its single-slope operation. The counter counts from BOTTOM to MAX then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC2A) is cleared on the compare match between TCNT2 and OCR2A, and set at BOTTOM. In inverting Compare Output mode, the output is set on compare match and cleared at BOTTOM. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct PWM mode that uses 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), and therefore reduces total system cost.

In fast PWM mode, the counter is incremented until the counter value matches the MAX value. The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 58. The TCNT2 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 TCNT2 slopes represent compare matches between OCR2A and TCNT2.

**Figure 58.** Fast PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOV2) is set each time the counter reaches MAX. If the interrupt is enabled, the interrupt handler routine can be used for updating the compare value.

In fast PWM mode, the compare unit allows generation of PWM waveforms on the OC2A pin. Setting the COM2A1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM2A1:0 to three (See Table 61 on page 136). The actual OC2A value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by setting (or clearing) the OC2A Register at the compare match between OCR2A and TCNT2, and clearing (or setting) the OC2A Register at the timer clock cycle the counter is cleared (changes from MAX to BOTTOM).

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

$$f_{OCnxPWM} = \frac{f_{clk_I/O}}{N \cdot 256}$$

The *N* variable represents the prescale factor (1, 8, 32, 64, 128, 256, or 1024).

The extreme values for the OCR2A Register represent special cases when generating a PWM waveform output in the fast PWM mode. If the OCR2A is set equal to BOTTOM, the output will be a narrow spike for each MAX+1 timer clock cycle. Setting the OCR2A equal to MAX will result in a constantly high or low output (depending on the polarity of the output set by the COM2A1:0 bits.)

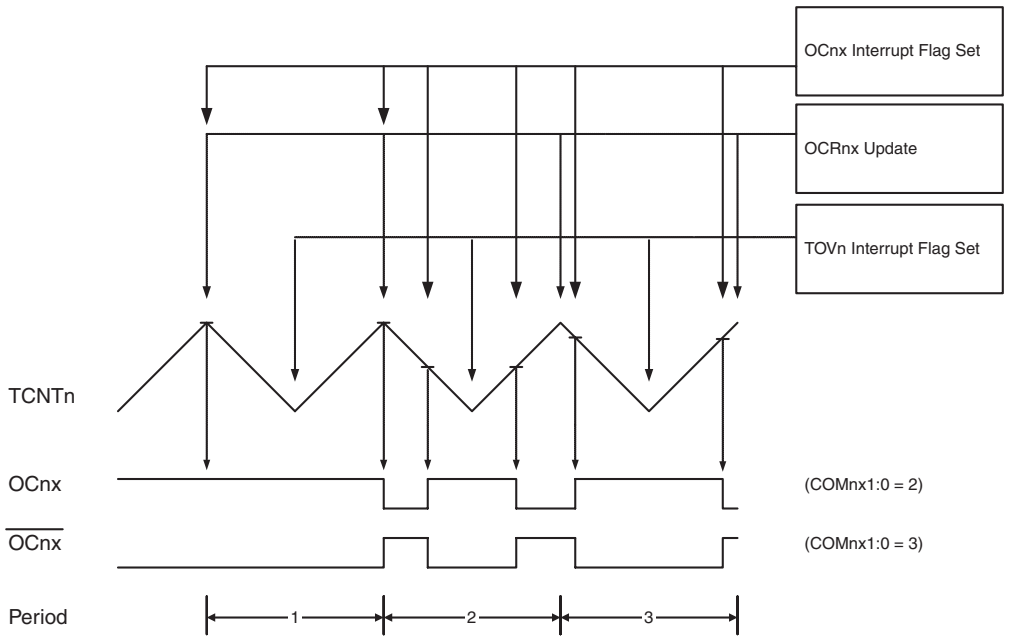
A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC2A to toggle its logical level on each compare match (COM2A1:0 = 1). The waveform generated will have a maximum frequency of $f_{oc2} = f_{clk_I/O}/2$ when OCR2A is set to zero. This feature is similar to the OC2A toggle in CTC mode, except the double buffer feature of the Output Compare unit is enabled in the fast PWM mode.

**Phase Correct PWM Mode**

The phase correct PWM mode (WGM21:0 = 1) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is based on a dual-slope operation. The counter counts repeatedly from BOTTOM to MAX and then from MAX to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC2A) is cleared on the compare match between TCNT2 and OCR2A while counting up, and set on the compare match while counting down. 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 is fixed to eight bits. In phase correct PWM mode the counter is incremented until the counter value matches MAX. When the counter reaches MAX, it changes the count direction. The TCNT2 value will be equal to MAX for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 59. The TCNT2 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 TCNT2 slopes represent compare matches between OCR2A and TCNT2.

**Figure 59.** Phase Correct PWM Mode, Timing Diagram




The Timer/Counter Overflow Flag (TOV2) is set each time the counter reaches BOTTOM. The Interrupt Flag can be used to generate an interrupt each time the counter reaches the BOTTOM value.

In phase correct PWM mode, the compare unit allows generation of PWM waveforms on the OC2A pin. Setting the COM2A1:0 bits to two will produce a non-inverted PWM. An inverted PWM output can be generated by setting the COM2A1:0 to three (See Table 62 on page 136). The actual OC2A value will only be visible on the port pin if the data direction for the port pin is set as output. The PWM waveform is generated by clearing (or setting) the OC2A Register at the compare match between OCR2A and TCNT2 when the counter increments, and setting (or clearing) the OC2A Register at compare match between OCR2A and TCNT2 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnxPCPWM} = \frac{f_{clk_I/O}}{N \cdot 510}$$

The *N* variable represents the prescale factor (1, 8, 32, 64, 128, 256, or 1024).

The extreme values for the OCR2A Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR2A is set equal to BOTTOM, the output will be continuously low and if set equal to MAX the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values.

At the very start of period 2 in Figure 59 OCn has a transition from high to low even though there is no Compare Match. The point of this transition is to guarantee symmetry around BOTTOM. There are two cases that give a transition without Compare Match.

- OCR2A changes its value from MAX, like in Figure 59. When the OCR2A value is MAX the OCn pin value is the same as the result of a down-counting compare match. To ensure symmetry around BOTTOM the OCn value at MAX must correspond to the result of an up-counting Compare Match.

- The timer starts counting from a value higher than the one in OCR2A, and for that reason misses the Compare Match and hence the OCn change that would have happened on the way up.

## Timer/Counter Timing Diagrams

The following figures show the Timer/Counter in synchronous mode, and the timer clock ($clk_{T2}$) is therefore shown as a clock enable signal. In asynchronous mode, $clk_{I/O}$ should be replaced by the Timer/Counter Oscillator clock. The figures include information on when Interrupt Flags are set. Figure 60 contains timing data for basic Timer/Counter operation. The figure shows the count sequence close to the MAX value in all modes other than phase correct PWM mode.

**Figure 60.** Timer/Counter Timing Diagram, no Prescaling

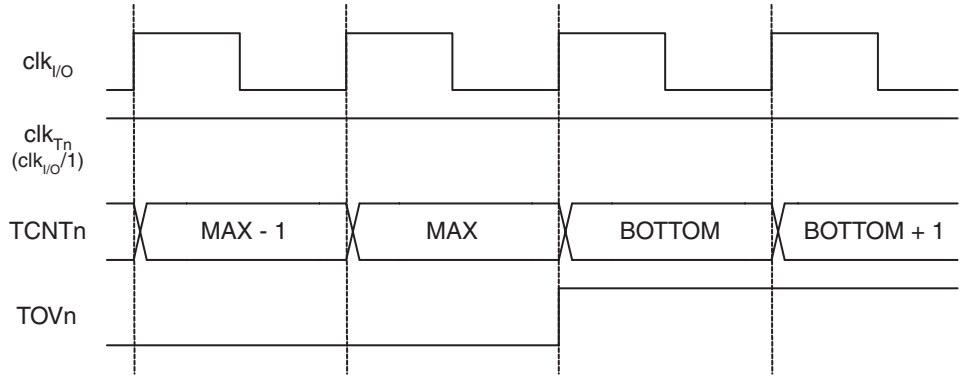



Figure 61 shows the same timing data, but with the prescaler enabled.

**Figure 61.** Timer/Counter Timing Diagram, with Prescaler ($f_{clk_I/O}/8$)

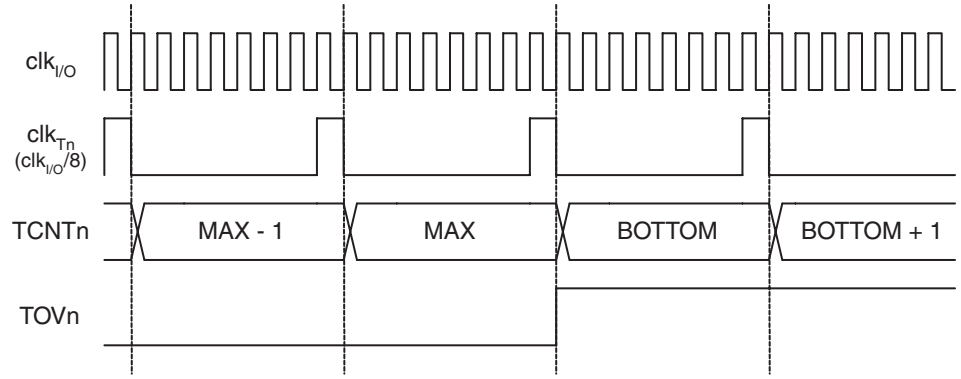



Figure 62 shows the setting of OCF2A in all modes except CTC mode.

**Figure 62.** Timer/Counter Timing Diagram, Setting of OCF2A, with Prescaler ($f_{clk_I/O}/8$)

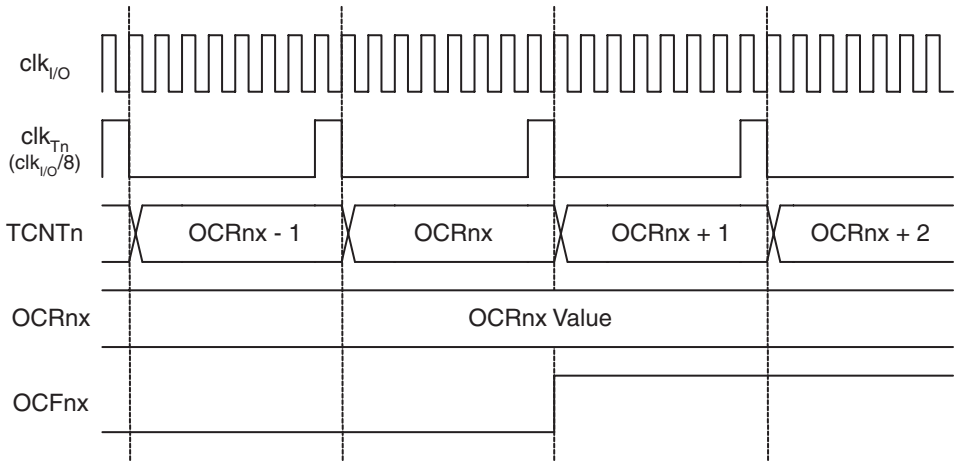
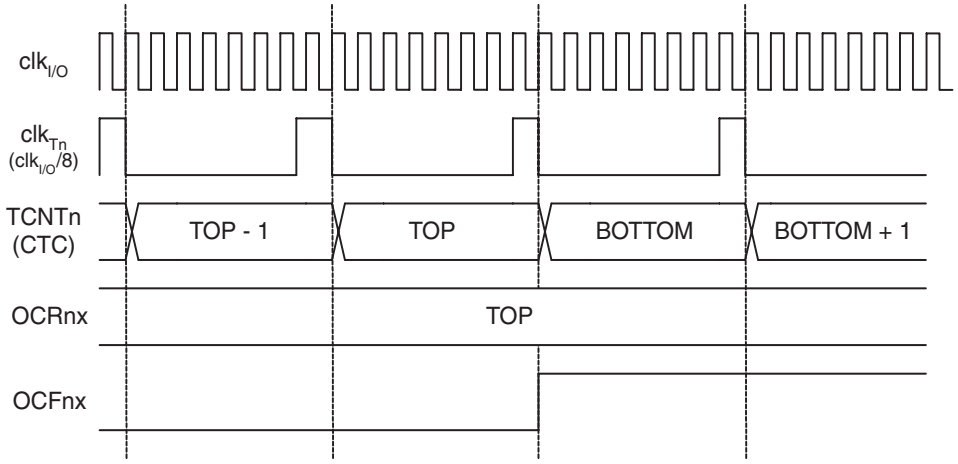



Figure 63 shows the setting of OCF2A and the clearing of TCNT2 in CTC mode.

**Figure 63.** Timer/Counter Timing Diagram, Clear Timer on Compare Match mode, with Prescaler ($f_{clk_I/O}/8$)



## 8-bit Timer/Counter Register Description

### Timer/Counter Control Register A– TCCR2A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | FOC2A | WGM20 | COM2A1 | COM2A0 | WGM21 | CS22 | CS21 | CS20 | TCCR2A |
| Read/Write | 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 – FOC2A: Force Output Compare A**

The FOC2A bit is only active when the WGM bits specify a non-PWM mode. However, for ensuring compatibility with future devices, this bit must be set to zero when TCCR2A is written when operating in PWM mode. When writing a logical one to the FOC2A bit, an immediate compare match is forced on the Waveform Generation unit. The OC2A output is changed according to its COM2A1:0 bits setting. Note that the FOC2A bit is implemented as a strobe. Therefore it is the value present in the COM2A1:0 bits that determines the effect of the forced compare.

A FOC2A strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR2A as TOP.

The FOC2A bit is always read as zero.

- **Bit 6, 3 – WGM21:0: Waveform Generation Mode**

These bits control the counting sequence of the counter, the source for the maximum (TOP) counter value, and what type of waveform generation to be used. Modes of operation supported by the Timer/Counter unit are: Normal mode, Clear Timer on Compare match (CTC) mode, and two types of Pulse Width Modulation (PWM) modes. See Table 59 and “Modes of Operation” on page 129.

**Table 59.** Waveform Generation Mode Bit Description[1]

| Mode | WGM21 (CTC2) | WGM20 (PWM2) | Timer/Counter Mode of Operation | TOP | Update of OCR2A at | TOV2 Flag Set on |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | Normal | 0xFF | Immediate | MAX |
| 1 | 0 | 1 | PWM, Phase Correct | 0xFF | TOP | BOTTOM |
| 2 | 1 | 0 | CTC | OCR2A | Immediate | MAX |
| 3 | 1 | 1 | Fast PWM | 0xFF | TOP | MAX |

Note: 1. The CTC2 and PWM2 bit definition names are now obsolete. Use the WGM21:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

- **Bit 5:4 – COM2A1:0: Compare Match Output Mode A**

These bits control the Output Compare pin (OC2A) behavior. If one or both of the COM2A1:0 bits are set, the OC2A 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 OC2A pin must be set in order to enable the output driver.

When OC2A is connected to the pin, the function of the COM2A1:0 bits depends on the WGM21:0 bit setting. Table 60 shows the COM2A1:0 bit functionality when the WGM21:0 bits are set to a normal or CTC mode (non-PWM).

**Table 60.** Compare Output Mode, non-PWM Mode

| COM2A1 | COM2A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC2A disconnected. |
| 0 | 1 | Toggle OC2A on compare match. |
| 1 | 0 | Clear OC2A on compare match. |
| 1 | 1 | Set OC2A on compare match. |

Table 61 shows the COM2A1:0 bit functionality when the WGM21:0 bits are set to fast PWM mode.

**Table 61.** Compare Output Mode, Fast PWM Mode(1)

| COM2A1 | COM2A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC2A disconnected. |
| 0 | 1 | Reserved |
| 1 | 0 | Clear OC2A on compare match, set OC2A at TOP. |
| 1 | 1 | Set OC2A on compare match, clear OC2A at TOP. |

Note: 1. A special case occurs when OCR2A equals TOP and COM2A1 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See “Fast PWM Mode” on page 130 for more details.

Table 62 shows the COM21:0 bit functionality when the WGM21:0 bits are set to phase correct PWM mode.

**Table 62.** Compare Output Mode, Phase Correct PWM Mode(1)

| COM2A1 | COM2A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC2A disconnected. |
| 0 | 1 | Reserved |
| 1 | 0 | Clear OC2A on compare match when up-counting. Set OC2A on compare match when counting down. |
| 1 | 1 | Set OC2A on compare match when up-counting. Clear OC2A on compare match when counting down. |

Note: 1. A special case occurs when OCR2A equals TOP and COM2A1 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See “Phase Correct PWM Mode” on page 131 for more details.

- **Bit 2:0 – CS22:0: Clock Select**

The three Clock Select bits select the clock source to be used by the Timer/Counter, see Table 63.

**Table 63.** Clock Select Bit Description

| CS22 | CS21 | CS20 | Description |
|---|---|---|---|
| 0 | 0 | 0 | No clock source (Timer/Counter stopped). |
| 0 | 0 | 1 | $clk_{T2S}$/(No prescaling) |
| 0 | 1 | 0 | $clk_{T2S}$/8 (From prescaler) |
| 0 | 1 | 1 | $clk_{T2S}$/32 (From prescaler) |
| 1 | 0 | 0 | $clk_{T2S}$/64 (From prescaler) |
| 1 | 0 | 1 | $clk_{T2S}$/128 (From prescaler) |
| 1 | 1 | 0 | $clk_{T2S}$/256 (From prescaler) |
| 1 | 1 | 1 | $clk_{T2S}$/1024 (From prescaler) |

### Timer/Counter Register – TCNT2

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | TCNT2[7:0] | | | | | | | | TCNT2 |
| 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 TCNT2 Register blocks (removes) the compare match on the following timer clock. Modifying the counter (TCNT2) while the counter is running, introduces a risk of missing a compare match between TCNT2 and the OCR2A Register.

### Output Compare Register A - OCR2A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | OCR2A[7:0] | | | | | | | | OCR2A |
| 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 (TCNT2). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC2A pin.

## Asynchronous operation of the Timer/Counter

### Asynchronous Status Register – ASSR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | EXCLK | AS2 | TCN2UB | OCR2UB | TCR2UB | ASSR |
| Read/Write | R | R | R | R/W | R/W | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 4 – EXCLK: Enable External Clock Input**

When EXCLK is written to one, and asynchronous clock is selected, the external clock input buffer is enabled and an external clock can be input on Timer Oscillator 1 (TOSC1) pin instead of a 32 kHz crystal. Writing to EXCLK should be done before asynchronous operation is selected. Note that the crystal Oscillator will only run when this bit is zero.

- **Bit 3 – AS2: Asynchronous Timer/Counter2**

When AS2 is written to zero, Timer/Counter2 is clocked from the I/O clock, $clk_{I/O}$. When AS2 is written to one, Timer/Counter2 is clocked from a crystal Oscillator connected to the Timer Oscillator 1 (TOSC1) pin. When the value of AS2 is changed, the contents of TCNT2, OCR2A, and TCCR2A might be corrupted.

- **Bit 2 – TCN2UB: Timer/Counter2 Update Busy**

When Timer/Counter2 operates asynchronously and TCNT2 is written, this bit becomes set. When TCNT2 has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that TCNT2 is ready to be updated with a new value.

- **Bit 1 – OCR2UB: Output Compare Register2 Update Busy**

When Timer/Counter2 operates asynchronously and OCR2A is written, this bit becomes set. When OCR2A has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that OCR2A is ready to be updated with a new value.

- **Bit 0 – TCR2UB: Timer/Counter Control Register2 Update Busy**

When Timer/Counter2 operates asynchronously and TCCR2A is written, this bit becomes set. When TCCR2A has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that TCCR2A is ready to be updated with a new value.

If a write is performed to any of the three Timer/Counter2 Registers while its update busy flag is set, the updated value might get corrupted and cause an unintentional interrupt to occur.

The mechanisms for reading TCNT2, OCR2A, and TCCR2A are different. When reading TCNT2, the actual timer value is read. When reading OCR2A or TCCR2A, the value in the temporary storage register is read.

**Asynchronous Operation of Timer/Counter2**

When Timer/Counter2 operates asynchronously, some considerations must be taken.

- Warning: When switching between asynchronous and synchronous clocking of Timer/Counter2, the Timer Registers TCNT2, OCR2A, and TCCR2A might be corrupted. A safe procedure for switching clock source is:
  1. Disable the Timer/Counter2 interrupts by clearing OCIE2A and TOIE2.
  2. Select clock source by setting AS2 as appropriate.
  3. Write new values to TCNT2, OCR2A, and TCCR2A.
  4. To switch to asynchronous operation: Wait for TCN2UB, OCR2UB, and TCR2UB.
  5. Clear the Timer/Counter2 Interrupt Flags.
  6. Enable interrupts, if needed.
- The CPU main clock frequency must be more than four times the Oscillator frequency.
- When writing to one of the registers TCNT2, OCR2A, or TCCR2A, the value is transferred to a temporary register, and latched after two positive edges on TOSC1. The user should not write a new value before the contents of the temporary register have been transferred to its destination. Each of the three mentioned registers have their individual temporary register, which means that e.g. writing to TCNT2 does not disturb an OCR2A write in progress. To detect that a transfer to the destination register has taken place, the Asynchronous Status Register – ASSR has been implemented.
- When entering Power-save or ADC Noise Reduction mode after having written to TCNT2, OCR2A, or TCCR2A, the user must wait until the written register has been updated if Timer/Counter2 is used to wake up the device. Otherwise, the MCU will enter sleep mode before the changes are effective. This is particularly important if the Output Compare2 interrupt is used to wake up the device, since the Output Compare function is disabled during writing to OCR2A or TCNT2. If the write cycle is not finished, and the MCU enters sleep mode before the OCR2UB bit returns to zero, the device will never receive a compare match interrupt, and the MCU will not wake up.
- If Timer/Counter2 is used to wake the device up from Power-save or ADC Noise Reduction mode, precautions must be taken if the user wants to re-enter one of these modes: The interrupt logic needs one TOSC1 cycle to be reset. If the time between wake-up and re-entering sleep mode is less than one TOSC1 cycle, the interrupt will not occur, and the device will fail to wake up. If the user is in doubt whether the time before re-entering Power-save or ADC Noise Reduction mode is sufficient, the following algorithm can be used to ensure that one TOSC1 cycle has elapsed:
  1. Write a value to TCCR2A, TCNT2, or OCR2A.
  2. Wait until the corresponding Update Busy Flag in ASSR returns to zero.
  3. Enter Power-save or ADC Noise Reduction mode.
- When the asynchronous operation is selected, the 32.768 kHz Oscillator for Timer/Counter2 is always running, except in Power-down and Standby modes. After a Power-up Reset or wake-up from Power-down or Standby mode, the user should be aware of the fact that this Oscillator might take as long as one second to stabilize. The user is advised to wait for at least one second before using Timer/Counter2 after power-up or wake-up from Power-down or Standby mode. The contents of all Timer/Counter2 Registers must be considered lost after a wake-up from Power-down or Standby mode due to unstable clock signal upon start-up, no matter whether the Oscillator is in use or a clock signal is applied to the TOSC1 pin.

- Description of wake up from Power-save or ADC Noise Reduction mode when the timer is clocked asynchronously: When the interrupt condition is met, the wake up process is started on the following cycle of the timer clock, that is, the timer is always advanced by at least one before the processor can read the counter value. After wake-up, the MCU is halted for four cycles, it executes the interrupt routine, and resumes execution from the instruction following SLEEP.
- Reading of the TCNT2 Register shortly after wake-up from Power-save may give an incorrect result. Since TCNT2 is clocked on the asynchronous TOSC clock, reading TCNT2 must be done through a register synchronized to the internal I/O clock domain. Synchronization takes place for every rising TOSC1 edge. When waking up from Power-save mode, and the I/O clock ($clk_{I/O}$) again becomes active, TCNT2 will read as the previous value (before entering sleep) until the next rising TOSC1 edge. The phase of the TOSC clock after waking up from Power-save mode is essentially unpredictable, as it depends on the wake-up time. The recommended procedure for reading TCNT2 is thus as follows:
  1. Write any value to either of the registers OCR2A or TCCR2A.
  2. Wait for the corresponding Update Busy Flag to be cleared.
  3. Read TCNT2.
- During asynchronous operation, the synchronization of the Interrupt Flags for the asynchronous timer takes 3 processor cycles plus one timer cycle. The timer is therefore advanced by at least one before the processor can read the timer value causing the setting of the Interrupt Flag. The Output Compare pin is changed on the timer clock and is not synchronized to the processor clock.

### Timer/Counter2 Interrupt Mask Register – TIMSK2

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | – | – | OCIE2A | TOIE2 | TIMSK2 |
| Read/Write | R | R | R | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 1 – OCIE2A: Timer/Counter2 Output Compare Match A Interrupt Enable**

When the OCIE2A bit is written to one and the I-bit in the Status Register is set (one), the Timer/Counter2 Compare Match A interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter2 occurs, i.e., when the OCF2A bit is set in the Timer/Counter 2 Interrupt Flag Register – TIFR2.

- **Bit 0 – TOIE2: Timer/Counter2 Overflow Interrupt Enable**

When the TOIE2 bit is written to one and the I-bit in the Status Register is set (one), the Timer/Counter2 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter2 occurs, i.e., when the TOV2 bit is set in the Timer/Counter2 Interrupt Flag Register – TIFR2.

### Timer/Counter2 Interrupt Flag Register – TIFR2

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | – | – | OCF2A | TOV2 | TIFR2 |
| Read/Write | R | R | R | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 1 – OCF2A: Output Compare Flag 2 A**

The OCF2A bit is set (one) when a compare match occurs between the Timer/Counter2 and the data in OCR2A – Output Compare Register2. OCF2A is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF2A is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE2A

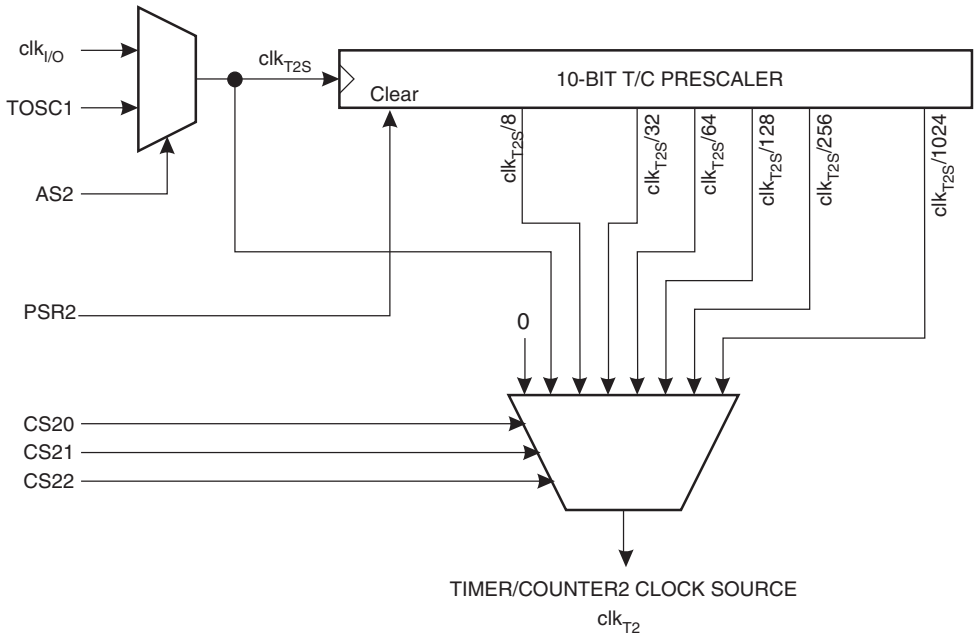
(Timer/Counter2 Compare match Interrupt Enable), and OCF2A are set (one), the Timer/Counter2 Compare match Interrupt is executed.

- **Bit 0 – TOV2: Timer/Counter2 Overflow Flag**

The TOV2 bit is set (one) when an overflow occurs in Timer/Counter2. TOV2 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, TOV2 is cleared by writing a logic one to the flag. When the SREG I-bit, TOIE2A (Timer/Counter2 Overflow Interrupt Enable), and TOV2 are set (one), the Timer/Counter2 Overflow interrupt is executed. In PWM mode, this bit is set when Timer/Counter2 changes counting direction at 0x00.

## Timer/Counter Prescaler

**Figure 64.** Prescaler for Timer/Counter2




The clock source for Timer/Counter2 is named $clk_{T2S}$. $clk_{T2S}$ is by default connected to the main system I/O clock $clk_{I/O}$. By setting the AS2 bit in ASSR, Timer/Counter2 is asynchronously clocked from the TOSC1 pin. This enables use of Timer/Counter2 as a Real Time Counter (RTC). When AS2 is set, pins TOSC1 and TOSC2 are disconnected from Port C. A crystal can then be connected between the TOSC1 and TOSC2 pins to serve as an independent clock source for Timer/Counter2. The Oscillator is optimized for use with a 32.768 kHz crystal. If applying an external clock on TOSC1, the EXCLK bit in ASSR must be set.

For Timer/Counter2, the possible prescaled selections are: $clk_{T2S}/8$, $clk_{T2S}/32$, $clk_{T2S}/64$, $clk_{T2S}/128$, $clk_{T2S}/256$, and $clk_{T2S}/1024$. Additionally, $clk_{T2S}$ as well as 0 (stop) may be selected. Setting the PSR2 bit in GTCCR resets the prescaler. This allows the user to operate with a predictable prescaler.

### General Timer/Counter Control Register – GTCCR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | TSM | – | – | – | – | – | PSR2 | PSR10 | GTCCR |
| Read/Write | R/W | R | R | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 1 – PSR2: Prescaler Reset Timer/Counter2**

When this bit is one, the Timer/Counter2 prescaler will be reset. This bit is normally cleared immediately by hardware. If the bit is written when Timer/Counter2 is operating in asynchronous mode, the bit will remain one until the prescaler has been reset. The bit will not be cleared by hardware if the TSM bit is set. Refer to the description of the “Bit 7 – TSM: Timer/Counter Synchronization Mode” on page 95 for a description of the Timer/Counter Synchronization mode.

## Serial Peripheral Interface – SPI

The ATmega325/3250/645/6450 SPI includes the following features:

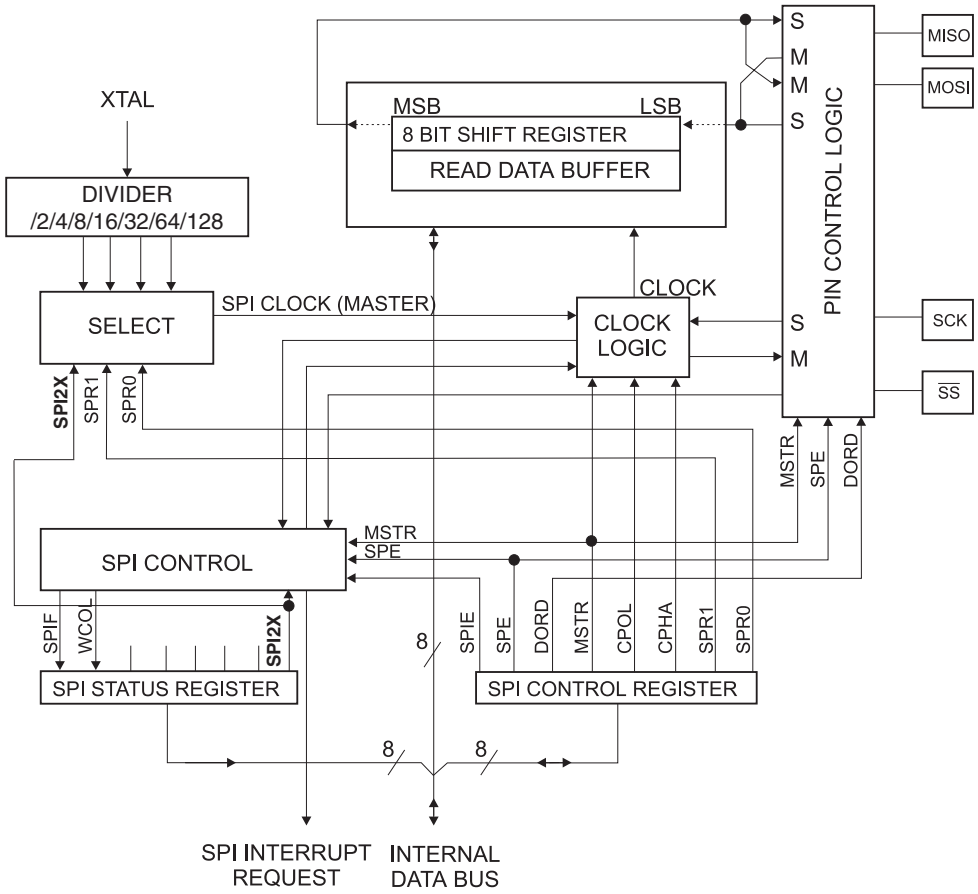
- **Full-duplex, Three-wire Synchronous Data Transfer**
- **Master or Slave Operation**
- **LSB First or MSB First Data Transfer**
- **Seven Programmable Bit Rates**
- **End of Transmission Interrupt Flag**
- **Write Collision Flag Protection**
- **Wake-up from Idle Mode**
- **Double Speed (CK/2) Master SPI Mode**

## Overview

The Serial Peripheral Interface (SPI) allows high-speed synchronous data transfer between the ATmega325/3250/645/6450 and peripheral devices or between several AVR devices.

The PRSPI bit in “Power Reduction Register” on page 35 must be written to zero to enable the SPI module.

**Figure 65.** SPI Block Diagram[1]




Note: 1. Refer to Figure 1 on page 2, and Table 26 on page 59 for SPI pin placement.

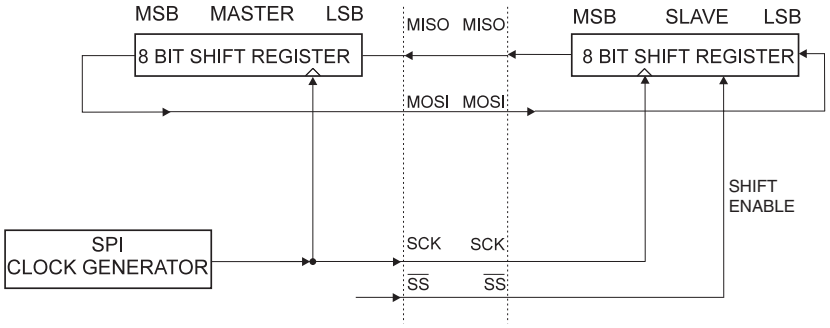
The interconnection between Master and Slave CPUs with SPI is shown in Figure 66. The system consists of two shift Registers, and a Master clock generator. The SPI Master initiates the communication cycle when pulling low the Slave Select $\overline{SS}$ pin of the desired Slave. Master and Slave prepare 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 66.** 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 64. For more details on automatic port overrides, refer to "Alternate Port Functions" on page 57.

**Table 64.** 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 "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
```

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 “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
```

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;
}
```

Note: 1. See “About Code Examples” on page 8.

## $\overline{SS}$ Pin Functionality

### 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.

### 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.

### SPI Control Register – SPCR

| 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 67 and Figure 68 for an example. The CPOL functionality is summarized below:

**Table 65.** 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 67 and Figure 68 for an example. The CPOL functionality is summarized below:

**Table 66.** 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 67.** 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 | $f_{osc}/64$ |

### SPI Status Register – SPSR

| 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 bits in the ATmega325/3250/645/6450 and will always read as 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 67). 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 ATmega325/3250/645/6450 is also used for program memory and EEPROM downloading or uploading. See page 271 for serial programming and verification.

### SPI Data Register – SPDR

| 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.

## 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 67 and Figure 68. 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 65 and Table 66, as done below:

**Table 68.** CPOL Functionality

| | Leading Edge | Trailing eDge | SPI Mode |
|---|---|---|---|
| CPOL=0, CPHA=0 | Sample (Rising) | Setup (Falling) | 0 |
| CPOL=0, CPHA=1 | Setup (Rising) | Sample (Falling) | 1 |
| CPOL=1, CPHA=0 | Sample (Falling) | Setup (Rising) | 2 |
| CPOL=1, CPHA=1 | Setup (Falling) | Sample (Rising) | 3 |

**Figure 67.** SPI Transfer Format with CPHA = 0

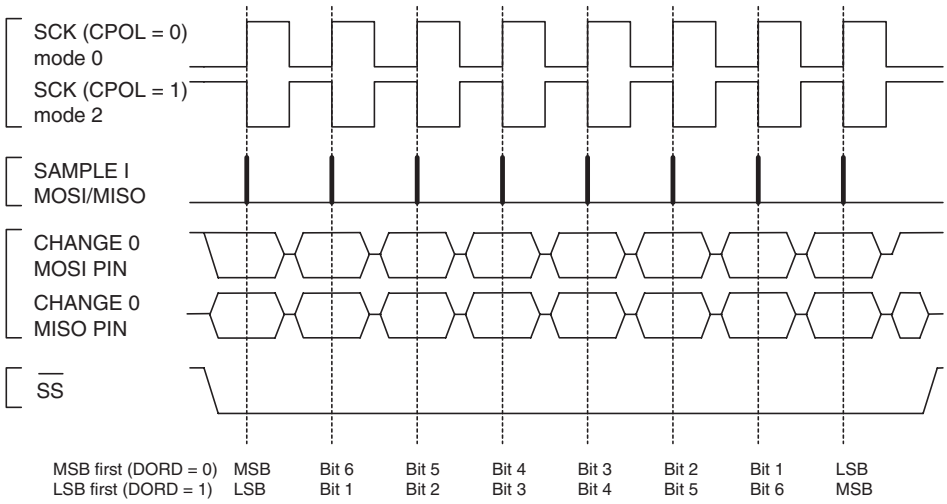
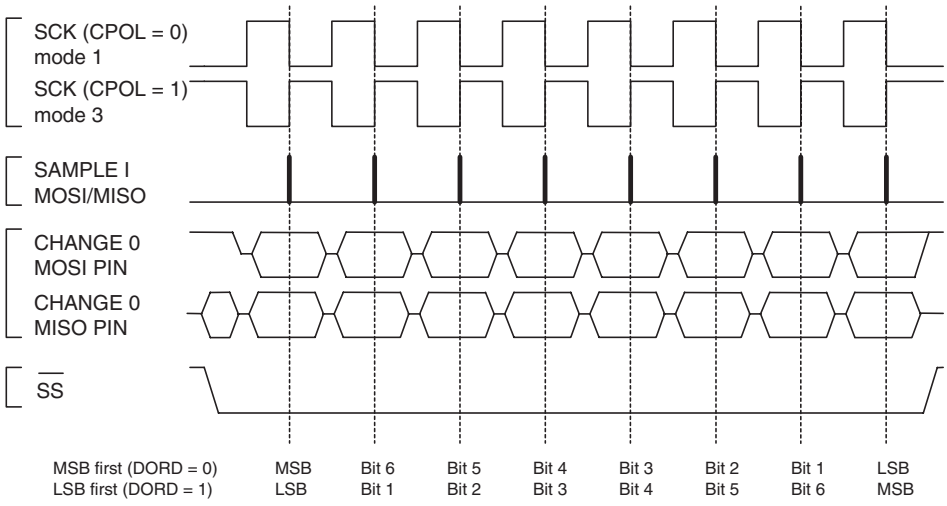



**Figure 68.** SPI Transfer Format with CPHA = 1



## USART0

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device. The main features are:

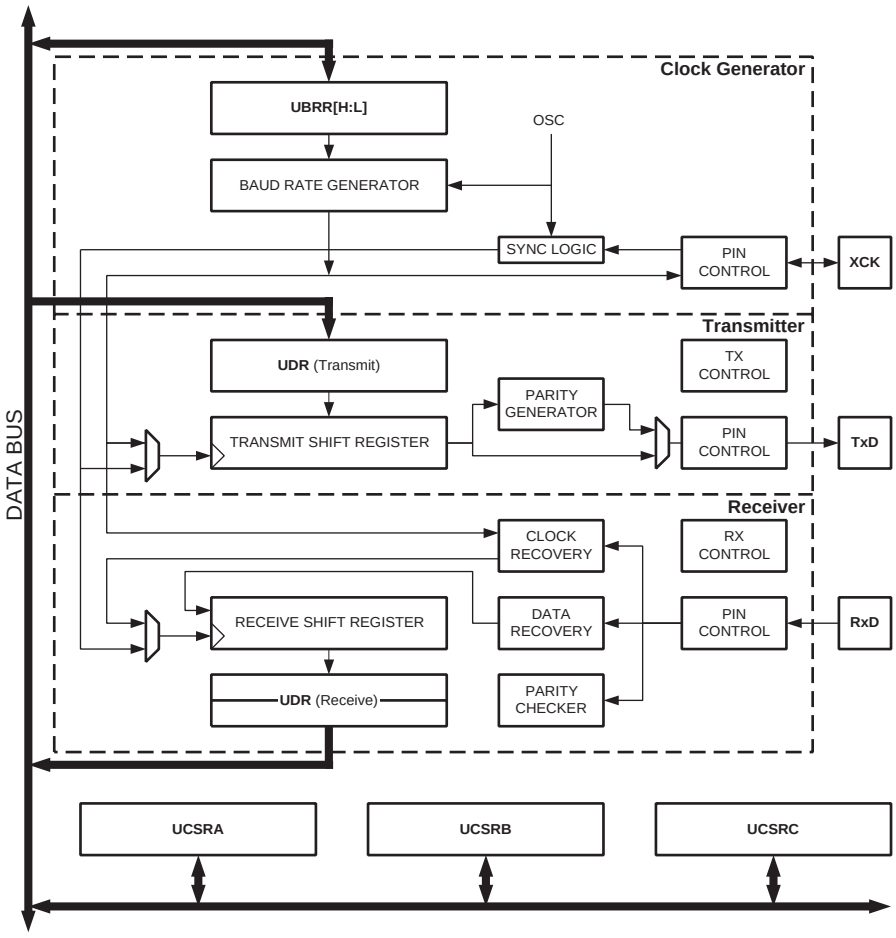
- **Full Duplex Operation (Independent Serial Receive and Transmit Registers)**
- **Asynchronous or Synchronous Operation**
- **Master or Slave Clocked Synchronous Operation**
- **High Resolution Baud Rate Generator**
- **Supports Serial Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Stop Bits**
- **Odd or Even Parity Generation and Parity Check Supported by Hardware**
- **Data OverRun Detection**
- **Framing Error Detection**
- **Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter**
- **Three Separate Interrupts on TX Complete, TX Data Register Empty and RX Complete**
- **Multi-processor Communication Mode**
- **Double Speed Asynchronous Communication Mode**

## Overview

A simplified block diagram of the USART Transmitter is shown in Figure 69. CPU accessible I/O Registers and I/O pins are shown in bold.

The Power Reduction USART bit, PRUSART0, in “Power Reduction Register” on page 35 must be written to zero to enable the USART0 module.

**Figure 69.** USART Block Diagram[1]




Note: 1. Refer to Figure 1 on page 2, Figure 2 on page 3 and “Alternate Functions of Port E” on page 63 for USART pin placement.

The dashed boxes in the block diagram separate the three main parts of the USART (listed from the top): Clock Generator, Transmitter and Receiver. Control Registers are shared by all units. The Clock Generation logic consists of synchronization logic for external clock input used by synchronous slave operation, and the baud rate generator. The XCK (Transfer Clock) pin is only used by synchronous transfer mode. The Transmitter consists of a single write buffer, a serial Shift Register, Parity Generator and Control logic for handling different serial frame formats. The write buffer allows a continuous transfer of data without any delay between frames. The Receiver is the most complex part of the USART module due to its clock and data recovery units. The recovery units are used for asynchronous data reception. In addition to the recovery units, the Receiver includes a Parity Checker, Control logic, a Shift Register and a two level receive buffer (UDRn). The Receiver supports the same frame formats as the Transmitter, and can detect Frame Error, Data OverRun and Parity Errors.

**AVR USART vs. AVR UART – Compatibility**

The USART is fully compatible with the AVR UART regarding:

- Bit locations inside all USART Registers.
- Baud Rate Generation.
- Transmitter Operation.
- Transmit Buffer Functionality.
- Receiver Operation.

However, the receive buffering has two improvements that will affect the compatibility in some special cases:

- A second Buffer Register has been added. The two Buffer Registers operate as a circular FIFO buffer. Therefore the UDRn must only be read once for each incoming data! More important is the fact that the Error Flags (FEn and DORn) and the ninth data bit (RXB8n) are buffered with the data in the receive buffer. Therefore the status bits must always be read before the UDRn Register is read. Otherwise the error status will be lost since the buffer state is lost.
- The Receiver Shift Register can now act as a third buffer level. This is done by allowing the received data to remain in the serial Shift Register (see Figure 69) if the Buffer Registers are full, until a new start bit is detected. The USART is therefore more resistant to Data OverRun (DORn) error conditions.

The following control bits have changed name, but have same functionality and register location:

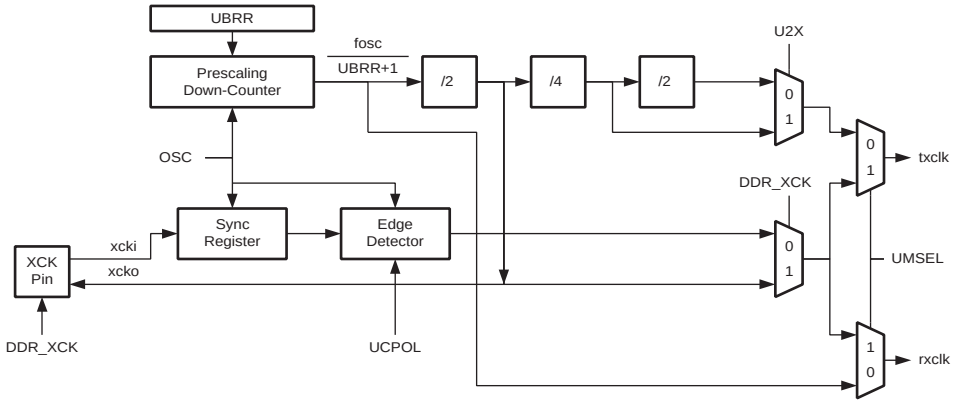
- CHR9 is changed to UCSZn2.
- OR is changed to DORn.

## Clock Generation

The Clock Generation logic generates the base clock for the Transmitter and Receiver. The USART supports four modes of clock operation: Normal asynchronous, Double Speed asynchronous, Master synchronous and Slave synchronous mode. The UMSELn bit in USART Control and Status Register C (UCSRnC) selects between asynchronous and synchronous operation. Double Speed (asynchronous mode only) is controlled by the U2Xn found in the UCSRnA Register. When using synchronous mode (UMSELn = 1), the Data Direction Register for the XCK pin (DDR_XCK) controls whether the clock source is internal (Master mode) or external (Slave mode). The XCK pin is only active when using synchronous mode.

Figure 70 shows a block diagram of the clock generation logic.

**Figure 70.** Clock Generation Logic, Block Diagram




Signal description:

**txclk** Transmitter clock (Internal Signal).

**rxclk** Receiver base clock (Internal Signal).

**xcki** Input from XCK pin (internal Signal). Used for synchronous slave operation.

**xcko** Clock output to XCK pin (Internal Signal). Used for synchronous master operation.

**fosc** XTAL pin frequency (System Clock).

### Internal Clock Generation – The Baud Rate Generator

Internal clock generation is used for the asynchronous and the synchronous master modes of operation. The description in this section refers to Figure 70.

The USART Baud Rate Register (UBRR) and the down-counter connected to it function as a programmable prescaler or baud rate generator. The down-counter, running at system clock ($f_{osc}$), is loaded with the UBRR value each time the counter has counted down to zero or when the UBRRL Register is written. A clock is generated each time the counter reaches zero. This clock is the baud rate generator clock output (= $f_{osc}$/(UBRR+1)). The Transmitter divides the baud rate generator clock output by 2, 8 or 16 depending on mode. The baud rate generator output is used directly by the Receiver's clock and data recovery units. However, the recovery units use a state machine that uses 2, 8 or 16 states depending on mode set by the state of the UMSELn, U2Xn and DDR_XCK bits.

Table 69 contains equations for calculating the baud rate (in bits per second) and for calculating the UBRR value for each mode of operation using an internally generated clock source.

**Table 69.** Equations for Calculating Baud Rate Register Setting

| Operating Mode | Equation for Calculating Baud Rate[1] | Equation for Calculating UBRR Value |
|---|---|---|
| Asynchronous Normal mode (U2Xn = 0) | $BAUD = \frac{f_{OSC}}{16(\boldsymbol{UBRR}+1)}$ | $\boldsymbol{UBRR} = \frac{f_{OSC}}{16BAUD} - 1$ |
| Asynchronous Double Speed mode (U2Xn = 1) | $BAUD = \frac{f_{OSC}}{8(\boldsymbol{UBRR}+1)}$ | $\boldsymbol{UBRR} = \frac{f_{OSC}}{8BAUD} - 1$ |
| Synchronous Master mode | $BAUD = \frac{f_{OSC}}{2(\boldsymbol{UBRR}+1)}$ | $\boldsymbol{UBRR} = \frac{f_{OSC}}{2BAUD} - 1$ |

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps)

**BAUD** Baud rate (in bits per second, bps)

**$f_{osc}$** System Oscillator clock frequency

**UBRR** Contents of the UBRRH and UBRRL Registers, (0-4095)

Some examples of UBRR values for some system clock frequencies are found in Table 77 (see page 174).

### Double Speed Operation (U2Xn)

The transfer rate can be doubled by setting the U2Xn bit in UCSRnA. Setting this bit only has effect for the asynchronous operation. Set this bit to zero when using synchronous operation.

Setting this bit will reduce the divisor of the baud rate divider from 16 to 8, effectively doubling the transfer rate for asynchronous communication. Note however that the Receiver will in this case only use half the number of samples (reduced from 16 to 8) for data sampling and clock recovery, and therefore a more accurate baud rate setting and system clock are required when this mode is used. For the Transmitter, there are no downsides.

### External Clock

External clocking is used by the synchronous slave modes of operation. The description in this section refers to Figure 70 for details.

External clock input from the XCK pin is sampled by a synchronization register to minimize the chance of meta-stability. The output from the synchronization register must then pass through an edge detector before it can be used by the Transmitter and Receiver. This process introduces a two CPU clock period delay and therefore the maximum external XCK clock frequency is limited by the following equation:

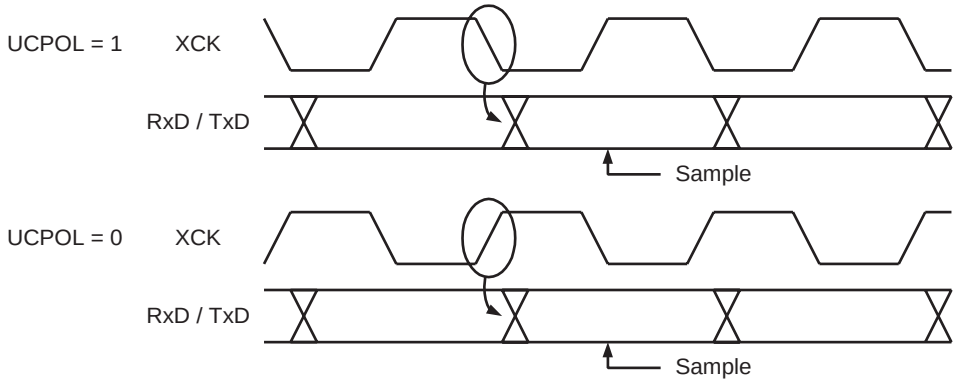
$$f_{XCK} < \frac{f_{OSC}}{4}$$

Note that $f_{osc}$ depends on the stability of the system clock source. It is therefore recommended to add some margin to avoid possible loss of data due to frequency variations.

### Synchronous Clock Operation

When synchronous mode is used (UMSELn = 1), the XCK pin will be used as either clock input (Slave) or clock output (Master). The dependency between the clock edges and data sampling or data change is the same. The basic principle is that data input (on RxD) is sampled at the opposite XCK clock edge of the edge the data output (TxD) is changed.

**Figure 71.** Synchronous Mode XCK Timing.




The UCPOLn bit in UCSRnC selects which XCK clock edge is used for data sampling and which is used for data change. As Figure 71 shows, when UCPOLn is zero the data will be changed at rising XCK edge and sampled at falling XCK edge. If UCPOLn is set, the data will be changed at falling XCK edge and sampled at rising XCK edge.

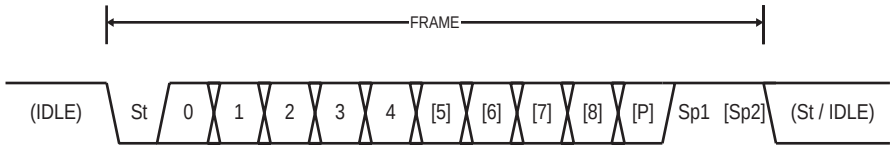
## Frame Formats

A serial frame is defined to be one character of data bits with synchronization bits (start and stop bits), and optionally a parity bit for error checking. The USART accepts all 30 combinations of the following as valid frame formats:

- 1 start bit
- 5, 6, 7, 8, or 9 data bits
- no, even or odd parity bit
- 1 or 2 stop bits

A frame starts with the start bit followed by the least significant data bit. Then the next data bits, up to a total of nine, are succeeding, ending with the most significant bit. If enabled, the parity bit is inserted after the data bits, before the stop bits. When a complete frame is transmitted, it can be directly followed by a new frame, or the communication line can be set to an idle (high) state. Figure 72 illustrates the possible combinations of the frame formats. Bits inside brackets are optional.

**Figure 72.** Frame Formats




**St** Start bit, always low.

**(n)** Data bits (0 to 8).

**P** Parity bit. Can be odd or even.

**Sp** Stop bit, always high.

**IDLE** No transfers on the communication line (RxD or TxD). An IDLE line must be high.

The frame format used by the USART is set by the UCSZn2:0, UPMn1:0 and USBSn bits in UCSRnB and UCSRnC. The Receiver and Transmitter use the same setting. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the Receiver and Transmitter.

The USART Character SiZe (UCSZn2:0) bits select the number of data bits in the frame. The USART Parity mode (UPMn1:0) bits enable and set the type of parity bit. The selection between one or two stop bits is done by the USART Stop Bit Select (USBSn) bit. The Receiver ignores the second stop bit. An FEn (Frame Error) will therefore only be detected in the cases where the first stop bit is zero.

### Parity Bit Calculation

The parity bit is calculated by doing an exclusive-or of all the data bits. If odd parity is used, the result of the exclusive or is inverted. The relation between the parity bit and data bits is as follows:

$$P_{even} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

**$P_{even}$** Parity bit using even parity

**$P^{odd}$** Parity bit using odd parity

**$d_n$** Data bit n of the character

If used, the parity bit is located between the last data bit and first stop bit of a serial frame.

## USART Initialization

The USART has to be initialized before any communication can take place. The initialization process normally consists of setting the baud rate, setting frame format and enabling the Transmitter or the Receiver depending on the usage. For interrupt driven USART operation, the Global Interrupt Flag should be cleared (and interrupts globally disabled) when doing the initialization.

Before doing a re-initialization with changed baud rate or frame format, be sure that there are no ongoing transmissions during the period the registers are changed. The TXCn Flag can be used to check that the Transmitter has completed all transfers, and the RXCn Flag can be used to check that there are no unread data in the receive buffer. Note that the TXCn Flag must be cleared before each transmission (before UDRn is written) if it is used for this purpose.

The following simple USART initialization code examples show one assembly and one C function that are equal in functionality. The examples assume asynchronous operation using polling (no interrupts enabled) and a fixed frame format. The baud rate is

given as a function parameter. For the assembly code, the baud rate parameter is assumed to be stored in the r17:r16 Registers.

Assembly Code Example[1]

```
USART_Init:
  ; Set baud rate
  out  UBRRH, r17
  out  UBRRL, r16
  ; Enable receiver and transmitter
  ldi  r16, (1<<RXEN0)|(1<<TXEN0)
  out  UCSR0B,r16
  ; Set frame format: 8data, 2stop bit
  ldi  r16, (1<<USBS0)|(3<<UCSZ0)
  out  UCSR0C,r16
  ret
```

C Code Example[1]

```
#define FOSC 1843200 // Clock Speed
#define BAUD 9600
#define MYUBRR FOSC/16/BAUD-1
void main( void )
{
  ...
  USART_Init(MYUBRR)
  ...
}

void USART_Init( unsigned int ubrr)
{
  /* Set baud rate */
  UBRRH = (unsigned char)(ubrr>>8);
  UBRRL = (unsigned char)ubrr;
  /* Enable receiver and transmitter */
  UCSR0B = (1<<RXEN0)|(1<<TXEN0);
  /* Set frame format: 8data, 2stop bit */
  UCSR0C = (1<<USBS0)|(3<<UCSZ0);
}
```

Note: 1. See “About Code Examples” on page 8.

More advanced initialization routines can be made that include frame format as parameters, disable interrupts and so on. However, many applications use a fixed setting of the baud and control registers, and for these types of applications the initialization code can be placed directly in the main routine, or be combined with initialization code for other I/O modules.

## Data Transmission – The USART Transmitter

The USART Transmitter is enabled by setting the *Transmit Enable* (TXENn) bit in the UCSRnB Register. When the Transmitter is enabled, the normal port operation of the TxD pin is overridden by the USART and given the function as the Transmitter's serial output. The baud rate, mode of operation and frame format must be set up once before doing any transmissions. If synchronous operation is used, the clock on the XCK pin will be overridden and used as transmission clock.

### Sending Frames with 5 to 8 Data Bit

A data transmission is initiated by loading the transmit buffer with the data to be transmitted. The CPU can load the transmit buffer by writing to the UDRn I/O location. The buffered data in the transmit buffer will be moved to the Shift Register when the Shift Register is ready to send a new frame. The Shift Register is loaded with new data if it is in idle state (no ongoing transmission) or immediately after the last stop bit of the previous frame is transmitted. When the Shift Register is loaded with new data, it will transfer one complete frame at the rate given by the Baud Register, U2Xn bit or by XCK depending on mode of operation.

The following code examples show a simple USART transmit function based on polling of the *Data Register Empty* (UDREn) Flag. When using frames with less than eight bits, the most significant bits written to the UDRn are ignored. The USART has to be initialized before the function can be used. For the assembly code, the data to be sent is assumed to be stored in Register R16.

Assembly Code Example[1]

```
USART_Transmit:
  ; Wait for empty transmit buffer
  sbis UCSR0A,UDRE0
  rjmp USART_Transmit
  ; Put data (r16) into buffer, sends the data
  out  UDR0,r16
  ret
```

C Code Example[1]

```
void USART_Transmit( unsigned char data )
{
  /* Wait for empty transmit buffer */
  while ( !( UCSR0A & (1<<UDRE0)) )
       ;
  /* Put data into buffer, sends the data */
  UDR0 = data;
}
```

Note: 1. See "About Code Examples" on page 8.

The function simply waits for the transmit buffer to be empty by checking the UDREn Flag, before loading it with new data to be transmitted. If the Data Register Empty interrupt is utilized, the interrupt routine writes the data into the buffer.

### Sending Frames with 9 Data Bit

If 9-bit characters are used (UCSZ = 7), the ninth bit must be written to the TXB8n bit in UCSRnB before the low byte of the character is written to UDRn. The following code examples show a transmit function that handles 9-bit characters. For the assembly code, the data to be sent is assumed to be stored in registers R17:R16.

Assembly Code Example[1][2]

```
USART_Transmit:
  ; Wait for empty transmit buffer
  sbis UCSR0A,UDRE0
  rjmp USART_Transmit
  ; Copy 9th bit from r17 to TXB80
  cbi  UCSR0B,TXB80
  sbrc r17,0
  sbi  UCSR0B,TXB80
  ; Put LSB data (r16) into buffer, sends the data
  out  UDR0,r16
  ret
```

C Code Example[1][2]

```
void USART_Transmit( unsigned int data )
{
  /* Wait for empty transmit buffer */
  while ( !( UCSR0A & (1<<UDRE0)) )
       ;
  /* Copy 9th bit to TXB80 */
  UCSR0B &= ~(1<<TXB80);
  if ( data & 0x0100 )
    UCSR0B |= (1<<TXB80);
  /* Put data into buffer, sends the data */
  UDR0 = data;
}
```

Notes:
1. These transmit functions are written to be general functions. They can be optimized if the contents of the UCSRnB is static. For example, only the TXB8n bit of the UCSRnB Register is used after initialization.
2. See “About Code Examples” on page 8.

The ninth bit can be used for indicating an address frame when using multi processor communication mode or for other protocol handling as for example synchronization.

### Transmitter Flags and Interrupts

The USART Transmitter has two flags that indicate its state: USART Data Register Empty (UDREn) and Transmit Complete (TXCn). Both flags can be used for generating interrupts.

The Data Register Empty (UDREn) Flag indicates whether the transmit buffer is ready to receive new data. This bit is set when the transmit buffer is empty, and cleared when the transmit buffer contains data to be transmitted that has not yet been moved into the Shift Register. For compatibility with future devices, always write this bit to zero when writing the UCSRnA Register.

When the Data Register Empty Interrupt Enable (UDRIEn) bit in UCSRnB is written to one, the USART Data Register Empty Interrupt will be executed as long as UDREn is set (provided that global interrupts are enabled). UDREn is cleared by writing UDRn. When interrupt-driven data transmission is used, the Data Register Empty interrupt routine must either write new data to UDRn in order to clear UDREn or disable the Data Register Empty interrupt, otherwise a new interrupt will occur once the interrupt routine terminates.

The Transmit Complete (TXCn) Flag bit is set one when the entire frame in the Transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer. The TXCn Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXCn Flag is useful in half-duplex communication interfaces (like the RS-485 standard), where a transmitting application must enter receive mode and free the communication bus immediately after completing the transmission.

When the Transmit Compete Interrupt Enable (TXCIEn) bit in UCSRnB is set, the USART Transmit Complete Interrupt will be executed when the TXCn Flag becomes set (provided that global interrupts are enabled). When the transmit complete interrupt is used, the interrupt handling routine does not have to clear the TXCn Flag, this is done automatically when the interrupt is executed.

### Parity Generator

The Parity Generator calculates the parity bit for the serial frame data. When parity bit is enabled (UPMn1 = 1), the transmitter control logic inserts the parity bit between the last data bit and the first stop bit of the frame that is sent.

### Disabling the Transmitter

The disabling of the Transmitter (setting the TXENn to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted. When disabled, the Transmitter will no longer override the TxD pin.

## Data Reception – The USART Receiver

The USART Receiver is enabled by writing the Receive Enable (RXENn) bit in the UCSRnB Register to one. When the Receiver is enabled, the normal pin operation of the RxD pin is overridden by the USART and given the function as the Receiver's serial input. The baud rate, mode of operation and frame format must be set up once before any serial reception can be done. If synchronous operation is used, the clock on the XCK pin will be used as transfer clock.

### Receiving Frames with 5 to 8 Data Bits

The Receiver starts data reception when it detects a valid start bit. Each bit that follows the start bit will be sampled at the baud rate or XCK clock, and shifted into the Receive Shift Register until the first stop bit of a frame is received. A second stop bit will be ignored by the Receiver. When the first stop bit is received, i.e., a complete serial frame is present in the Receive Shift Register, the contents of the Shift Register will be moved into the receive buffer. The receive buffer can then be read by reading the UDRn I/O location.

The following code example shows a simple USART receive function based on polling of the Receive Complete (RXCn) Flag. When using frames with less than eight bits the most significant bits of the data read from the UDRn will be masked to zero. The USART has to be initialized before the function can be used.

Assembly Code Example[1]

```
USART_Receive:
  ; Wait for data to be received
  sbis UCSR0A, RXC0
  rjmp USART_Receive
  ; Get and return received data from buffer
  in   r16, UDR0
  ret
```

C Code Example[1]

```
unsigned char USART_Receive( void )
{
  /* Wait for data to be received */
  while ( !(UCSR0A & (1<<RXC0)) )
       ;
  /* Get and return received data from buffer */
  return UDR0;
}
```

Note: 1. See "About Code Examples" on page 8.

The function simply waits for data to be present in the receive buffer by checking the RXCn Flag, before reading the buffer and returning the value.

### Receiving Frames with 9 Data Bits

If 9-bit characters are used (UCSZ=7) the ninth bit must be read from the RXB8n bit in UCSRnB **before** reading the low bits from the UDRn. This rule applies to the FEn, DORn and UPEn Status Flags as well. Read status from UCSRnA, then data from UDRn. Reading the UDRn I/O location will change the state of the receive buffer FIFO and consequently the TXB8n, FEn, DORn and UPEn bits, which all are stored in the FIFO, will change.

The following code example shows a simple USART receive function that handles both nine bit characters and the status bits.

Assembly Code Example[1]

```
USART_Receive:
  ; Wait for data to be received
  sbis UCSR0A, RXC0
  rjmp USART_Receive
  ; Get status and 9th bit, then data from buffer
  in   r18, UCSR0A
  in   r17, UCSR0B
  in   r16, UDR0
  ; If error, return -1
  andi r18,(1<<FE0)|(1<<DOR0)|(1<<UPE0)
  breq USART_ReceiveNoError
  ldi  r17, HIGH(-1)
  ldi  r16, LOW(-1)
USART_ReceiveNoError:
  ; Filter the 9th bit, then return
  lsr  r17
  andi r17, 0x01
  ret
```

C Code Example[1]

```
unsigned int USART_Receive( void )
{
  unsigned char status, resh, resl;
  /* Wait for data to be received */
  while ( !(UCSR0A & (1<<RXC0)) )
       ;
  /* Get status and 9th bit, then data */
  /* from buffer */
  status = UCSR0A;
  resh = UCSR0B;
  resl = UDR0;
  /* If error, return -1 */
  if ( status & (1<<FE0)|(1<<DOR0)|(1<<UPE0) )
    return -1;
  /* Filter the 9th bit, then return */
  resh = (resh >> 1) & 0x01;
  return ((resh << 8) | resl);
}
```

Note: 1. See “About Code Examples” on page 8.

The receive function example reads all the I/O Registers into the Register File before any computation is done. This gives an optimal receive buffer utilization since the buffer location read will be free to accept new data as early as possible.

### Receive Compete Flag and Interrupt

The USART Receiver has one flag that indicates the Receiver state.

The Receive Complete (RXCn) Flag indicates if there are unread data present in the receive buffer. This flag is one when unread data exist in the receive buffer, and zero when the receive buffer is empty (i.e., does not contain any unread data). If the Receiver is disabled (RXENn = 0), the receive buffer will be flushed and consequently the RXCn bit will become zero.

When the Receive Complete Interrupt Enable (RXCIEn) in UCSRnB is set, the USART Receive Complete interrupt will be executed as long as the RXCn Flag is set (provided that global interrupts are enabled). When interrupt-driven data reception is used, the receive complete routine must read the received data from UDRn in order to clear the RXCn Flag, otherwise a new interrupt will occur once the interrupt routine terminates.

### Receiver Error Flags

The USART Receiver has three Error Flags: Frame Error (FEn), Data OverRun (DORn) and Parity Error (UPEn). All can be accessed by reading UCSRnA. Common for the Error Flags is that they are located in the receive buffer together with the frame for which they indicate the error status. Due to the buffering of the Error Flags, the UCSRnA must be read before the receive buffer (UDRn), since reading the UDRn I/O location changes the buffer read location. Another equality for the Error Flags is that they can not be altered by software doing a write to the flag location. However, all flags must be set to zero when the UCSRnA is written for upward compatibility of future USART implementations. None of the Error Flags can generate interrupts.

The Frame Error (FEn) Flag indicates the state of the first stop bit of the next readable frame stored in the receive buffer. The FEn Flag is zero when the stop bit was correctly read (as one), and the FEn Flag will be one when the stop bit was incorrect (zero). This flag can be used for detecting out-of-sync conditions, detecting break conditions and protocol handling. The FEn Flag is not affected by the setting of the USBSn bit in UCSRnC since the Receiver ignores all, except for the first, stop bits. For compatibility with future devices, always set this bit to zero when writing to UCSRnA.

The Data OverRun (DORn) Flag indicates data loss due to a receiver buffer full condition. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the Receive Shift Register, and a new start bit is detected. If the DORn Flag is set there was one or more serial frame lost between the frame last read from UDRn, and the next frame read from UDRn. For compatibility with future devices, always write this bit to zero when writing to UCSRnA. The DORn Flag is cleared when the frame received was successfully moved from the Shift Register to the receive buffer.

The Parity Error (UPEn) Flag indicates that the next frame in the receive buffer had a Parity Error when received. If Parity Check is not enabled the UPEn bit will always be read zero. For compatibility with future devices, always set this bit to zero when writing to UCSRnA. For more details see “Parity Bit Calculation” on page 157 and “Parity Checker” on page 164.

### Parity Checker

The Parity Checker is active when the high USART Parity mode (UPMn1) bit is set. Type of Parity Check to be performed (odd or even) is selected by the UPMn0 bit. When enabled, the Parity Checker calculates the parity of the data bits in incoming frames and compares the result with the parity bit from the serial frame. The result of the check is stored in the receive buffer together with the received data and stop bits. The Parity Error (UPEn) Flag can then be read by software to check if the frame had a Parity Error.

The UPEn bit is set if the next character that can be read from the receive buffer had a Parity Error when received and the Parity Checking was enabled at that point (UPMn1 = 1). This bit is valid until the receive buffer (UDRn) is read.

### Disabling the Receiver

In contrast to the Transmitter, disabling of the Receiver will be immediate. Data from ongoing receptions will therefore be lost. When disabled (i.e., the RXENn is set to zero) the Receiver will no longer override the normal function of the RxD port pin. The Receiver buffer FIFO will be flushed when the Receiver is disabled. Remaining data in the buffer will be lost

### Flushing the Receive Buffer

The receiver buffer FIFO will be flushed when the Receiver is disabled, i.e., the buffer will be emptied of its contents. Unread data will be lost. If the buffer has to be flushed during normal operation, due to for instance an error condition, read the UDRn I/O location until the RXCn Flag is cleared. The following code example shows how to flush the receive buffer.

Assembly Code Example[1]

```
USART_Flush:
  sbis UCSR0A, RXC0
  ret
  in   r16, UDR0
  rjmp USART_Flush
```

C Code Example[1]

```
void USART_Flush( void )
{
  unsigned char dummy;
  while ( UCSR0A & (1<<RXC0) ) dummy = UDR0;
}
```

Note: 1. See “About Code Examples” on page 8.

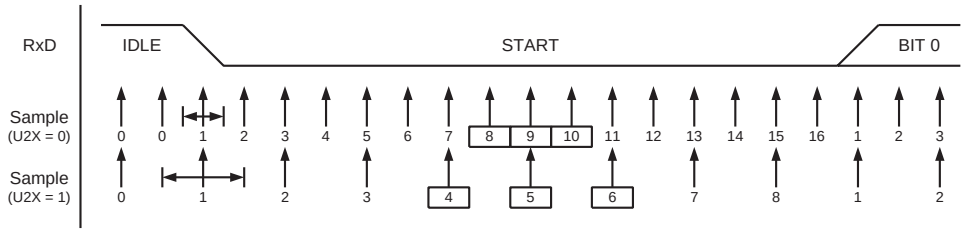
## Asynchronous Data Reception

The USART includes a clock recovery and a data recovery unit for handling asynchronous data reception. The clock recovery logic is used for synchronizing the internally generated baud rate clock to the incoming asynchronous serial frames at the RxD pin. The data recovery logic samples and low pass filters each incoming bit, thereby improving the noise immunity of the Receiver. The asynchronous reception operational range depends on the accuracy of the internal baud rate clock, the rate of the incoming frames, and the frame size in number of bits.

### Asynchronous Clock Recovery

The clock recovery logic synchronizes internal clock to the incoming serial frames. Figure 73 illustrates the sampling process of the start bit of an incoming frame. The sample rate is 16 times the baud rate for Normal mode, and eight times the baud rate for Double Speed mode. The horizontal arrows illustrate the synchronization variation due to the sampling process. Note the larger time variation when using the Double Speed mode (U2Xn = 1) of operation. Samples denoted zero are samples done when the RxD line is idle (i.e., no communication activity).

**Figure 73.** Start Bit Sampling

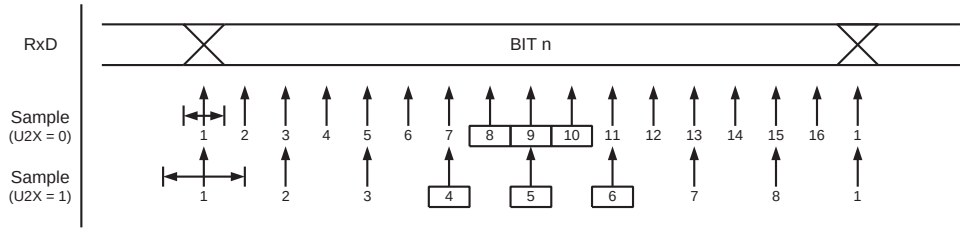



When the clock recovery logic detects a high (idle) to low (start) transition on the RxD line, the start bit detection sequence is initiated. Let sample 1 denote the first zero-sample as shown in the figure. The clock recovery logic then uses samples 8, 9, and 10 for Normal mode, and samples 4, 5, and 6 for Double Speed mode (indicated with sample numbers inside boxes on the figure), to decide if a valid start bit is received. If two or more of these three samples have logical high levels (the majority wins), the start bit is rejected as a noise spike and the Receiver starts looking for the next high to low-transition. If however, a valid start bit is detected, the clock recovery logic is synchronized and the data recovery can begin. The synchronization process is repeated for each start bit.

**Asynchronous Data Recovery**

When the receiver clock is synchronized to the start bit, the data recovery can begin. The data recovery unit uses a state machine that has 16 states for each bit in Normal mode and eight states for each bit in Double Speed mode. Figure 74 shows the sampling of the data bits and the parity bit. Each of the samples is given a number that is equal to the state of the recovery unit.

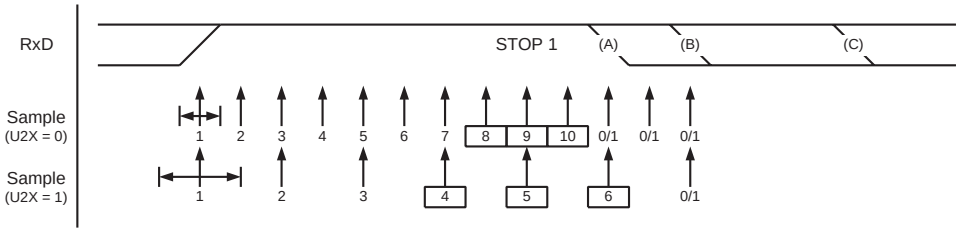
**Figure 74.** Sampling of Data and Parity Bit




The decision of the logic level of the received bit is taken by doing a majority voting of the logic value to the three samples in the center of the received bit. The center samples are emphasized on the figure by having the sample number inside boxes. The majority voting process is done as follows: If two or all three samples have high levels, the received bit is registered to be a logic 1. If two or all three samples have low levels, the received bit is registered to be a logic 0. This majority voting process acts as a low pass filter for the incoming signal on the RxD pin. The recovery process is then repeated until a complete frame is received. Including the first stop bit. Note that the Receiver only uses the first stop bit of a frame.

Figure 75 shows the sampling of the stop bit and the earliest possible beginning of the start bit of the next frame.

**Figure 75.** Stop Bit Sampling and Next Start Bit Sampling




The same majority voting is done to the stop bit as done for the other bits in the frame. If the stop bit is registered to have a logic 0 value, the Frame Error (FEn) Flag will be set.

A new high to low transition indicating the start bit of a new frame can come right after the last of the bits used for majority voting. For Normal Speed mode, the first low level sample can be at point marked (A) in Figure 75. For Double Speed mode the first low level must be delayed to (B). (C) marks a stop bit of full length. The early start bit detection influences the operational range of the Receiver.

### Asynchronous Operational Range

The operational range of the Receiver is dependent on the mismatch between the received bit rate and the internally generated baud rate. If the Transmitter is sending frames at too fast or too slow bit rates, or the internally generated baud rate of the Receiver does not have a similar (see Table 70) base frequency, the Receiver will not be able to synchronize the frames to the start bit.

The following equations can be used to calculate the ratio of the incoming data rate and internal receiver baud rate.

$$R_{slow} = \frac{(D+1)S}{S-1+D\cdot S+S_F} \qquad R_{fast} = \frac{(D+2)S}{(D+1)S+S_M}$$

**D** Sum of character size and parity size (D = 5 to 10 bit)

**S** Samples per bit. S = 16 for Normal Speed mode and S = 8 for Double Speed mode.

**$S_F$** First sample number used for majority voting. $S_F$ = 8 for normal speed and $S_F$ = 4 for Double Speed mode.

**$S_M$** Middle sample number used for majority voting. $S_M$ = 9 for normal speed and $S_M$ = 5 for Double Speed mode.

**$R_{slow}$** is the ratio of the slowest incoming data rate that can be accepted in relation to the receiver baud rate. $R_{fast}$ is the ratio of the fastest incoming data rate that can be accepted in relation to the receiver baud rate.

Table 70 and Table 71 list the maximum receiver baud rate error that can be tolerated. Note that Normal Speed mode has higher toleration of baud rate variations.

**Table 70.** Recommended Maximum Receiver Baud Rate Error for Normal Speed Mode (U2Xn = 0)

| D # (Data+Parity Bit) | $R_{slow}$ (%) | $R_{fast}$ (%) | Max Total Error (%) | Recommended Max Receiver Error (%) |
|---|---|---|---|---|
| 5 | 93.20 | 106.67 | +6.67/-6.8 | ± 3.0 |
| 6 | 94.12 | 105.79 | +5.79/-5.88 | ± 2.5 |
| 7 | 94.81 | 105.11 | +5.11/-5.19 | ± 2.0 |
| 8 | 95.36 | 104.58 | +4.58/-4.54 | ± 2.0 |
| 9 | 95.81 | 104.14 | +4.14/-4.19 | ± 1.5 |
| 10 | 96.17 | 103.78 | +3.78/-3.83 | ± 1.5 |

**Table 71.** Recommended Maximum Receiver Baud Rate Error for Double Speed Mode (U2Xn = 1)

| D # (Data+Parity Bit) | $R_{slow}$ (%) | $R_{fast}$ (%) | Max Total Error (%) | Recommended Max Receiver Error (%) |
|---|---|---|---|---|
| 5 | 94.12 | 105.66 | +5.66/-5.88 | ± 2.5 |
| 6 | 94.92 | 104.92 | +4.92/-5.08 | ± 2.0 |
| 7 | 95.52 | 104,35 | +4.35/-4.48 | ± 1.5 |
| 8 | 96.00 | 103.90 | +3.90/-4.00 | ± 1.5 |
| 9 | 96.39 | 103.53 | +3.53/-3.61 | ± 1.5 |
| 10 | 96.70 | 103.23 | +3.23/-3.30 | ± 1.0 |

The recommendations of the maximum receiver baud rate error was made under the assumption that the Receiver and Transmitter equally divides the maximum total error.

There are two possible sources for the receivers baud rate error. The Receiver's system clock (XTAL) will always have some minor instability over the supply voltage range and the temperature range. When using a crystal to generate the system clock, this is rarely a problem, but for a resonator the system clock may differ more than 2% depending of the resonators tolerance. The second source for the error is more controllable. The baud rate generator can not always do an exact division of the system frequency to get the baud rate wanted. In this case an UBRR value that gives an acceptable low error can be used if possible.

## Multi-processor Communication Mode

Setting the Multi-processor Communication mode (MPCMn) bit in UCSRnA enables a filtering function of incoming frames received by the USART Receiver. Frames that do not contain address information will be ignored and not put into the receive buffer. This effectively reduces the number of incoming frames that has to be handled by the CPU, in a system with multiple MCUs that communicate via the same serial bus. The Transmitter is unaffected by the MPCMn setting, but has to be used differently when it is a part of a system utilizing the Multi-processor Communication mode.

If the Receiver is set up to receive frames that contain 5 to 8 data bits, then the first stop bit indicates if the frame contains data or address information. If the Receiver is set up for frames with nine data bits, then the ninth bit (RXB8n) is used for identifying address and data frames. When the frame type bit (the first stop or the ninth bit) is one, the frame contains an address. When the frame type bit is zero the frame is a data frame.

The Multi-processor Communication mode enables several slave MCUs to receive data from a master MCU. This is done by first decoding an address frame to find out which MCU has been addressed. If a particular slave MCU has been addressed, it will receive the following data frames as normal, while the other slave MCUs will ignore the received frames until another address frame is received.

### Using MPCMn

For an MCU to act as a master MCU, it can use a 9-bit character frame format (UCSZ = 7). The ninth bit (TXB8n) must be set when an address frame (TXB8n = 1) or cleared when a data frame (TXB = 0) is being transmitted. The slave MCUs must in this case be set to use a 9-bit character frame format.

The following procedure should be used to exchange data in Multi-processor Communication mode:

1. All Slave MCUs are in Multi-processor Communication mode (MPCMn in UCSRnA is set).
2. The Master MCU sends an address frame, and all slaves receive and read this frame. In the Slave MCUs, the RXCn Flag in UCSRnA will be set as normal.
3. Each Slave MCU reads the UDRn Register and determines if it has been selected. If so, it clears the MPCMn bit in UCSRnA, otherwise it waits for the next address byte and keeps the MPCMn setting.
4. The addressed MCU will receive all data frames until a new address frame is received. The other Slave MCUs, which still have the MPCMn bit set, will ignore the data frames.
5. When the last data frame is received by the addressed MCU, the addressed MCU sets the MPCMn bit and waits for a new address frame from master. The process then repeats from 2.

Using any of the 5- to 8-bit character frame formats is possible, but impractical since the Receiver must change between using n and n+1 character frame formats. This makes full-duplex operation difficult since the Transmitter and Receiver uses the same character size setting. If 5- to 8-bit character frames are used, the Transmitter must be set to use two stop bit (USBSn = 1) since the first stop bit is used for indicating the frame type.

Do not use Read-Modify-Write instructions (SBI and CBI) to set or clear the MPCMn bit. The MPCMn bit shares the same I/O location as the TXCn Flag and this might accidentally be cleared when using SBI or CBI instructions.

## USART Register Description

### USART I/O Data Register n – UDRn

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | RXB[7:0] | | | | | | | | UDRn (Read) |
| | TXB[7:0] | | | | | | | | UDRn (Write) |
| 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 USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or UDRn. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDRn Register location. Reading the UDRn Register location will return the contents of the Receive Data Buffer Register (RXB).

For 5-, 6-, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver.

The transmit buffer can only be written when the UDREn Flag in the UCSRnA Register is set. Data written to UDRn when the UDREn Flag is not set, will be ignored by the USART Transmitter. When data is written to the transmit buffer, and the Transmitter is enabled, the Transmitter will load the data into the Transmit Shift Register when the Shift Register is empty. Then the data will be serially transmitted on the TxD pin.

The receive buffer consists of a two level FIFO. The FIFO will change its state whenever the receive buffer is accessed. Due to this behavior of the receive buffer, do not use Read-Modify-Write instructions (SBI and CBI) on this location. Be careful when using bit test instructions (SBIC and SBIS), since these also will change the state of the FIFO.

### USART Control and Status Register n A – UCSRnA

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | RXCn | TXCn | UDREn | FEn | DORn | UPEn | U2Xn | MPCMn | UCSRnA |
| Read/Write | R | R/W | R | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – RXCn: USART Receive Complete**

This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). If the Receiver is disabled, the receive buffer will be flushed and consequently the RXCn bit will become zero. The RXCn Flag can be used to generate a Receive Complete interrupt (see description of the RXCIEn bit).

- **Bit 6 – TXCn: USART Transmit Complete**

This flag bit is set when the entire frame in the Transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer (UDRn). The TXCn Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXCn Flag can generate a Transmit Complete interrupt (see description of the TXCIEn bit).

- **Bit 5 – UDREn: USART Data Register Empty**

The UDREn Flag indicates if the transmit buffer (UDRn) is ready to receive new data. If UDREn is one, the buffer is empty, and therefore ready to be written. The UDREn Flag can generate a Data Register Empty interrupt (see description of the UDRIEn bit).

UDREn is set after a reset to indicate that the Transmitter is ready.

- **Bit 4 – FEn: Frame Error**

This bit is set if the next character in the receive buffer had a Frame Error when received. I.e., when the first stop bit of the next character in the receive buffer is zero. This bit is valid until the receive buffer (UDRn) is read. The FEn bit is zero when the stop bit of received data is one. Always set this bit to zero when writing to UCSRnA.

- **Bit 3 – DORn: Data OverRun**

This bit is set if a Data OverRun condition is detected. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the Receive Shift Register, and a new start bit is detected. This bit is valid until the receive buffer (UDRn) is read. Always set this bit to zero when writing to UCSRnA.

- **Bit 2 – UPEn: USART Parity Error**

This bit is set if the next character in the receive buffer had a Parity Error when received and the Parity Checking was enabled at that point (UPMn1 = 1). This bit is valid until the receive buffer (UDRn) is read. Always set this bit to zero when writing to UCSRnA.

- **Bit 1 – U2Xn: Double the USART Transmission Speed**

This bit only has effect for the asynchronous operation. Write this bit to zero when using synchronous operation.

Writing this bit to one will reduce the divisor of the baud rate divider from 16 to 8 effectively doubling the transfer rate for asynchronous communication.

- **Bit 0 – MPCMn: Multi-processor Communication Mode**

This bit enables the Multi-processor Communication mode. When the MPCMn bit is written to one, all the incoming frames received by the USART Receiver that do not contain address information will be ignored. The Transmitter is unaffected by the MPCMn setting. For more detailed information see “Multi-processor Communication Mode” on page 169.

**USART Control and Status Register n B – UCSRnB**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | RXCIEn | TXCIEn | UDRIEn | RXENn | TXENn | UCSZn2 | RXB8n | TXB8n | UCSRnB |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – RXCIEn: RX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the RXCn Flag. A USART Receive Complete interrupt will be generated only if the RXCIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXCn bit in UCSRnA is set.

- **Bit 6 – TXCIEn: TX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the TXCn Flag. A USART Transmit Complete interrupt will be generated only if the TXCIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXCn bit in UCSRnA is set.

- **Bit 5 – UDRIEn: USART Data Register Empty Interrupt Enable**

Writing this bit to one enables interrupt on the UDREn Flag. A Data Register Empty interrupt will be generated only if the UDRIEn bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDREn bit in UCSRnA is set.

- **Bit 4 – RXENn: Receiver Enable**

Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxD pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FEn, DORn, and UPEn Flags.

- **Bit 3 – TXENn: Transmitter Enable**

Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxD pin when enabled. The disabling of the Transmitter (writing TXENn to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted. When disabled, the Transmitter will no longer override the TxD port.

- **Bit 2 – UCSZn2: Character Size**

The UCSZn2 bits combined with the UCSZn1:0 bit in UCSRnC sets the number of data bits (Character SiZe) in a frame the Receiver and Transmitter use.

- **Bit 1 – RXB8n: Receive Data Bit 8**

RXB8n is the ninth data bit of the received character when operating with serial frames with nine data bits. Must be read before reading the low bits from UDRn.

- **Bit 0 – TXB8n: Transmit Data Bit 8**

TXB8n is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. Must be written before writing the low bits to UDRn.

**USART Control and Status Register n C – UCSRnC**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | UMSELn | UPMn1 | UPMn0 | USBSn | UCSZn1 | UCSZn0 | UCPOLn | UCSRnC |
| Read/Write | R | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | |

- **Bit 6 – UMSELn: USART Mode Select**

This bit selects between asynchronous and synchronous mode of operation.

**Table 72.** UMSEL Bit Settings

| UMSELn | Mode |
|---|---|
| 0 | Asynchronous Operation |
| 1 | Synchronous Operation |

- **Bit 5:4 – UPMn1:0: Parity Mode**

These bits enable and set type of parity generation and check. If enabled, the Transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The Receiver will generate a parity value for the incoming data and compare it to the UPMn0 setting. If a mismatch is detected, the UPEn Flag in UCSRnA will be set.

**Table 73.** UPM Bits Settings

| UPMn1 | UPMn0 | Parity Mode |
|---|---|---|
| 0 | 0 | Disabled |
| 0 | 1 | Reserved |
| 1 | 0 | Enabled, Even Parity |
| 1 | 1 | Enabled, Odd Parity |

- **Bit 3 – USBSn: Stop Bit Select**

This bit selects the number of stop bits to be inserted by the Transmitter. The Receiver ignores this setting.

**Table 74.** USBS Bit Settings

| USBSn | Stop Bit(s) |
|---|---|
| 0 | 1-bit |
| 1 | 2-bit |

- **Bit 2:1 – UCSZn1:0: Character Size**

The UCSZn1:0 bits combined with the UCSZn2 bit in UCSRnB sets the number of data bits (Character SiZe) in a frame the Receiver and Transmitter use.

**Table 75.** UCSZ Bits Settings

| UCSZn2 | UCSZn1 | UCSZn0 | Character Size |
|---|---|---|---|
| 0 | 0 | 0 | 5-bit |
| 0 | 0 | 1 | 6-bit |
| 0 | 1 | 0 | 7-bit |
| 0 | 1 | 1 | 8-bit |
| 1 | 0 | 0 | Reserved |
| 1 | 0 | 1 | Reserved |
| 1 | 1 | 0 | Reserved |
| 1 | 1 | 1 | 9-bit |

- **Bit 0 – UCPOLn: Clock Polarity**

This bit is used for synchronous mode only. Write this bit to zero when asynchronous mode is used. The UCPOLn bit sets the relationship between data output change and data input sample, and the synchronous clock (XCK).

**Table 76.** UCPOL Bit Settings

| UCPOLn | Transmitted Data Changed (Output of TxD Pin) | Received Data Sampled (Input on RxD Pin) |
|---|---|---|
| 0 | Rising XCK Edge | Falling XCK Edge |
| 1 | Falling XCK Edge | Rising XCK Edge |

## USART Baud Rate Registers – UBRRnL and UBRRnH

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | UBRRn[11:8] | | | | UBRRnH |
| | UBRRn[7:0] | | | | | | | | UBRRnL |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Read/Write | R | R | R | R | 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 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 15:12 – Reserved Bits**

These bits are reserved for future use. For compatibility with future devices, these bit must be written to zero when UBRRH is written.

- **Bit 11:0 – UBRR11:0: USART Baud Rate Register**

This is a 12-bit register which contains the USART baud rate. The UBRRH contains the four most significant bits, and the UBRRL contains the eight least significant bits of the USART baud rate. Ongoing transmissions by the Transmitter and Receiver will be corrupted if the baud rate is changed. Writing UBRRL will trigger an immediate update of the baud rate prescaler.

## Examples of Baud Rate Setting

For standard crystal and resonator frequencies, the most commonly used baud rates for asynchronous operation can be generated by using the UBRR settings in Table 77. UBRR values which yield an actual baud rate differing less than 0.5% from the target baud rate, are bold in the table. Higher error ratings are acceptable, but the Receiver will have less noise resistance when the error ratings are high, especially for large serial frames (see “Asynchronous Operational Range” on page 167). The error values are calculated using the following equation:

$$\text{Error}[\%] = \left(\frac{\text{BaudRate}_{\text{Closest Match}}}{\text{BaudRate}} - 1\right) \bullet 100\%$$

**Table 77.** Examples of UBRR Settings for Commonly Used Oscillator Frequencies

| Baud Rate (bps) | $f_{osc}$ = 1.0000 MHz | | | | $f_{osc}$ = 1.8432 MHz | | | | $f_{osc}$ = 2.0000 MHz | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | |
| | UBRR | Error | UBRR | Error | UBRR | Error | UBRR | Error | UBRR | Error | UBRR | Error |
| 2400 | **25** | **0.2%** | **51** | **0.2%** | **47** | **0.0%** | **95** | **0.0%** | **51** | **0.2%** | **103** | **0.2%** |
| 4800 | **12** | **0.2%** | **25** | **0.2%** | **23** | **0.0%** | **47** | **0.0%** | **25** | **0.2%** | **51** | **0.2%** |
| 9600 | 6 | -7.0% | **12** | **0.2%** | **11** | **0.0%** | **23** | **0.0%** | **12** | **0.2%** | **25** | **0.2%** |
| 14.4k | 3 | 8.5% | 8 | -3.5% | **7** | **0.0%** | **15** | **0.0%** | 8 | -3.5% | 16 | 2.1% |
| 19.2k | 2 | 8.5% | 6 | -7.0% | **5** | **0.0%** | **11** | **0.0%** | 6 | -7.0% | **12** | **0.2%** |
| 28.8k | 1 | 8.5% | 3 | 8.5% | **3** | **0.0%** | **7** | **0.0%** | 3 | 8.5% | 8 | -3.5% |
| 38.4k | 1 | -18.6% | 2 | 8.5% | **2** | **0.0%** | **5** | **0.0%** | 2 | 8.5% | 6 | -7.0% |
| 57.6k | 0 | 8.5% | 1 | 8.5% | **1** | **0.0%** | **3** | **0.0%** | 1 | 8.5% | 3 | 8.5% |
| 76.8k | – | – | 1 | -18.6% | 1 | -25.0% | **2** | **0.0%** | 1 | -18.6% | 2 | 8.5% |
| 115.2k | – | – | 0 | 8.5% | **0** | **0.0%** | **1** | **0.0%** | 0 | 8.5% | 1 | 8.5% |
| 230.4k | – | – | – | – | – | – | **0** | **0.0%** | – | – | – | – |
| 250k | – | – | – | – | – | – | – | – | – | – | 0 | 0.0% |
| Max. (1) | 62.5 kbps | | 125 kbps | | 115.2 kbps | | 230.4 kbps | | 125 kbps | | 250 kbps | |

1. UBRR = 0, Error = 0.0%

**Table 78.** Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

| Baud Rate (bps) | $f_{osc}$ = 3.6864 MHz | | | | $f_{osc}$ = 4.0000 MHz | | | | $f_{osc}$ = 7.3728 MHz | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | |
| | UBRR | Error | UBRR | Error | UBRR | Error | UBRR | Error | UBRR | Error | UBRR | Error |
| 2400 | **95** | **0.0%** | **191** | **0.0%** | **103** | **0.2%** | **207** | **0.2%** | **191** | **0.0%** | **383** | **0.0%** |
| 4800 | **47** | **0.0%** | **95** | **0.0%** | **51** | **0.2%** | **103** | **0.2%** | **95** | **0.0%** | **191** | **0.0%** |
| 9600 | **23** | **0.0%** | **47** | **0.0%** | **25** | **0.2%** | **51** | **0.2%** | **47** | **0.0%** | **95** | **0.0%** |
| 14.4k | **15** | **0.0%** | **31** | **0.0%** | 16 | 2.1% | 34 | -0.8% | **31** | **0.0%** | **63** | **0.0%** |
| 19.2k | **11** | **0.0%** | **23** | **0.0%** | **12** | **0.2%** | **25** | **0.2%** | **23** | **0.0%** | **47** | **0.0%** |
| 28.8k | **7** | **0.0%** | **15** | **0.0%** | 8 | -3.5% | 16 | 2.1% | **15** | **0.0%** | **31** | **0.0%** |
| 38.4k | **5** | **0.0%** | **11** | **0.0%** | 6 | -7.0% | **12** | **0.2%** | **11** | **0.0%** | **23** | **0.0%** |
| 57.6k | **3** | **0.0%** | **7** | **0.0%** | 3 | 8.5% | 8 | -3.5% | **7** | **0.0%** | **15** | **0.0%** |
| 76.8k | **2** | **0.0%** | **5** | **0.0%** | 2 | 8.5% | 6 | -7.0% | **5** | **0.0%** | **11** | **0.0%** |
| 115.2k | **1** | **0.0%** | **3** | **0.0%** | 1 | 8.5% | 3 | 8.5% | **3** | **0.0%** | **7** | **0.0%** |
| 230.4k | **0** | **0.0%** | **1** | **0.0%** | 0 | 8.5% | 1 | 8.5% | **1** | **0.0%** | **3** | **0.0%** |
| 250k | 0 | -7.8% | 1 | -7.8% | **0** | **0.0%** | **1** | **0.0%** | 1 | -7.8% | 3 | -7.8% |
| 0.5M | – | – | 0 | -7.8% | – | – | **0** | **0.0%** | 0 | -7.8% | 1 | -7.8% |
| 1M | – | – | – | – | – | – | – | – | – | – | 0 | -7.8% |
| Max. (1) | 230.4 kbps | | 460.8 kbps | | 250 kbps | | 0.5 Mbps | | 460.8 kbps | | 921.6 kbps | |

1. UBRR = 0, Error = 0.0%

**Table 79.** Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

| Baud Rate (bps) | $f_{osc}$ = 8.0000 MHz | | | | $f_{osc}$ = 11.0592 MHz | | | | $f_{osc}$ = 14.7456 MHz | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | |
| | UBRR | Error | UBRR | Error | UBRR | Error | UBRR | Error | UBRR | Error | UBRR | Error |
| 2400 | **207** | **0.2%** | **416** | **-0.1%** | **287** | **0.0%** | **575** | **0.0%** | **383** | **0.0%** | **767** | **0.0%** |
| 4800 | **103** | **0.2%** | **207** | **0.2%** | **143** | **0.0%** | **287** | **0.0%** | **191** | **0.0%** | **383** | **0.0%** |
| 9600 | **51** | **0.2%** | **103** | **0.2%** | **71** | **0.0%** | **143** | **0.0%** | **95** | **0.0%** | **191** | **0.0%** |
| 14.4k | 34 | -0.8% | 68 | 0.6% | **47** | **0.0%** | **95** | **0.0%** | **63** | **0.0%** | **127** | **0.0%** |
| 19.2k | **25** | **0.2%** | **51** | **0.2%** | **35** | **0.0%** | **71** | **0.0%** | **47** | **0.0%** | **95** | **0.0%** |
| 28.8k | 16 | 2.1% | 34 | -0.8% | **23** | **0.0%** | **47** | **0.0%** | **31** | **0.0%** | **63** | **0.0%** |
| 38.4k | **12** | **0.2%** | **25** | **0.2%** | **17** | **0.0%** | **35** | **0.0%** | **23** | **0.0%** | **47** | **0.0%** |
| 57.6k | 8 | -3.5% | 16 | 2.1% | **11** | **0.0%** | **23** | **0.0%** | **15** | **0.0%** | **31** | **0.0%** |
| 76.8k | 6 | -7.0% | **12** | **0.2%** | **8** | **0.0%** | **17** | **0.0%** | **11** | **0.0%** | **23** | **0.0%** |
| 115.2k | 3 | 8.5% | 8 | -3.5% | **5** | **0.0%** | **11** | **0.0%** | **7** | **0.0%** | **15** | **0.0%** |
| 230.4k | 1 | 8.5% | 3 | 8.5% | **2** | **0.0%** | **5** | **0.0%** | **3** | **0.0%** | **7** | **0.0%** |
| 250k | **1** | **0.0%** | **3** | **0.0%** | 2 | -7.8% | 5 | -7.8% | 3 | -7.8% | 6 | 5.3% |
| 0.5M | **0** | **0.0%** | **1** | **0.0%** | – | – | 2 | -7.8% | 1 | -7.8% | 3 | -7.8% |
| 1M | – | – | **0** | **0.0%** | – | – | – | – | 0 | -7.8% | 1 | -7.8% |
| Max. (1) | 0.5 Mbps | | 1 Mbps | | 691.2 kbps | | 1.3824 Mbps | | 921.6 kbps | | 1.8432 Mbps | |

1. UBRR = 0, Error = 0.0%

**Table 80.** Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

| Baud Rate (bps) | $f_{osc}$ = 16.0000 MHz | | | | $f_{osc}$ = 18.4320 MHz | | | | $f_{osc}$ = 20.0000 MHz | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | |
| | UBRR | Error | UBRR | Error | UBRR | Error | UBRR | Error | UBRR | Error | UBRR | Error |
| 2400 | **416** | **-0.1%** | **832** | **0.0%** | **479** | **0.0%** | **959** | **0.0%** | **520** | **0.0%** | **1041** | **0.0%** |
| 4800 | **207** | **0.2%** | **416** | **-0.1%** | **239** | **0.0%** | **479** | **0.0%** | **259** | **0.2%** | **520** | **0.0%** |
| 9600 | **103** | **0.2%** | **207** | **0.2%** | **119** | **0.0%** | **239** | **0.0%** | **129** | **0.2%** | **259** | **0.2%** |
| 14.4k | 68 | 0.6% | **138** | **-0.1%** | **79** | **0.0%** | **159** | **0.0%** | **86** | **-0.2%** | **173** | **-0.2%** |
| 19.2k | **51** | **0.2%** | **103** | **0.2%** | **59** | **0.0%** | **119** | **0.0%** | **64** | **0.2%** | **129** | **0.2%** |
| 28.8k | 34 | -0.8% | 68 | 0.6% | **39** | **0.0%** | **79** | **0.0%** | 42 | 0.9% | **86** | **-0.2%** |
| 38.4k | **25** | **0.2%** | **51** | **0.2%** | **29** | **0.0%** | **59** | **0.0%** | 32 | -1.4% | **64** | **0.2%** |
| 57.6k | 16 | 2.1% | 34 | -0.8% | **19** | **0.0%** | **39** | **0.0%** | 21 | -1.4% | 42 | 0.9% |
| 76.8k | **12** | **0.2%** | **25** | **0.2%** | **14** | **0.0%** | **29** | **0.0%** | 15 | 1.7% | 32 | -1.4% |
| 115.2k | 8 | -3.5% | 16 | 2.1% | **9** | **0.0%** | **19** | **0.0%** | 10 | -1.4% | 21 | -1.4% |
| 230.4k | 3 | 8.5% | 8 | -3.5% | **4** | **0.0%** | **9** | **0.0%** | 4 | 8.5% | 10 | -1.4% |
| 250k | **3** | **0.0%** | **7** | **0.0%** | 4 | -7.8% | 8 | 2.4% | **4** | **0.0%** | **9** | **0.0%** |
| 0.5M | **1** | **0.0%** | **3** | **0.0%** | – | – | 4 | -7.8% | – | – | 4 | 0.0% |
| 1M | **0** | **0.0%** | **1** | **0.0%** | – | – | – | – | – | – | – | – |
| Max. (1) | 1 Mbps | | 2 Mbps | | 1.152 Mbps | | 2.304 Mbps | | 1.25 Mbps | | 2.5 Mbps | |

1. UBRR = 0, Error = 0.0%

## Universal Serial Interface – USI

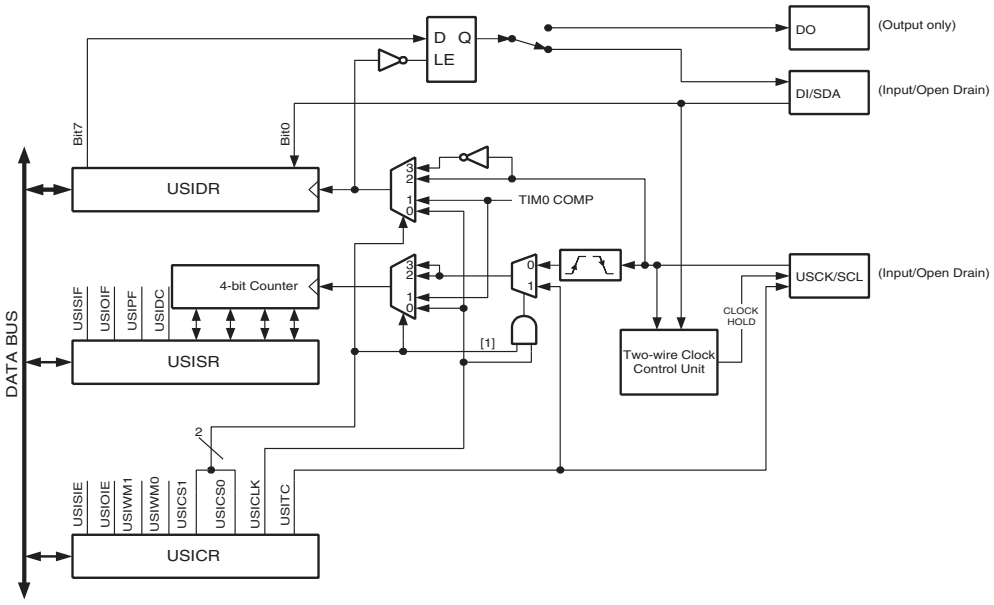
The Universal Serial Interface, or USI, provides the basic hardware resources needed for serial communication. Combined with a minimum of control software, the USI allows significantly higher transfer rates and uses less code space than solutions based on software only. Interrupts are included to minimize the processor load. The main features of the USI are:

- **Two-wire Synchronous Data Transfer (Master or Slave, $f_{SCLmax} = f_{CK}/16$)**
- **Three-wire Synchronous Data Transfer (Master or Slave $f_{SCKmax} = f_{CK}/4$)**
- **Data Received Interrupt**
- **Wake up from Idle Mode**
- **In Two-wire Mode: Wake-up from All Sleep Modes, Including Power-down Mode**
- **Two-wire Start Condition Detector with Interrupt Capability**

## Overview

A simplified block diagram of the USI is shown on Figure 76. For the actual placement of I/O pins, refer to “Pinout ATmega3250/6450” on page 2 and “Pinout ATmega325/645” 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 “USI Register Descriptions” on page 185.

**Figure 76.** Universal Serial Interface, Block Diagram




The 8-bit Shift Register is directly accessible via the data bus and contains the incoming and outgoing data. The register has no buffering so the data must be read as quickly as possible to ensure that no data is lost. The most significant bit is connected to one of two output pins depending of the wire mode configuration. A transparent latch is inserted between the Serial Register Output and output pin, which delays the change of data output to the opposite clock edge of the data input sampling. The serial input is always sampled from the Data Input (DI) pin independent of the configuration.

The 4-bit counter can be both read and written via the data bus, and can generate an overflow interrupt. Both the Serial Register and the counter are clocked simultaneously by the same clock source. This allows the counter to count the number of bits received or transmitted and generate an interrupt when the transfer is complete. Note that when an external clock source is selected the counter counts both clock edges. In this case the counter counts the number of edges, and not the number of bits. The clock can be

selected from three different sources: The USCK pin, Timer/Counter0 Compare Match or from software.

The Two-wire clock control unit can generate an interrupt when a start condition is detected on the Two-wire bus. It can also generate wait states by holding the clock pin low after a start condition is detected, or after the counter overflows.

## Functional Descriptions

### Three-wire Mode

The USI Three-wire mode is compliant to the Serial Peripheral Interface (SPI) mode 0 and 1, but does not have the slave select (SS) pin functionality. However, this feature can be implemented in software if necessary. Pin names used by this mode are: DI, DO, and USCK.

**Figure 77.** Three-wire Mode Operation, Simplified Diagram

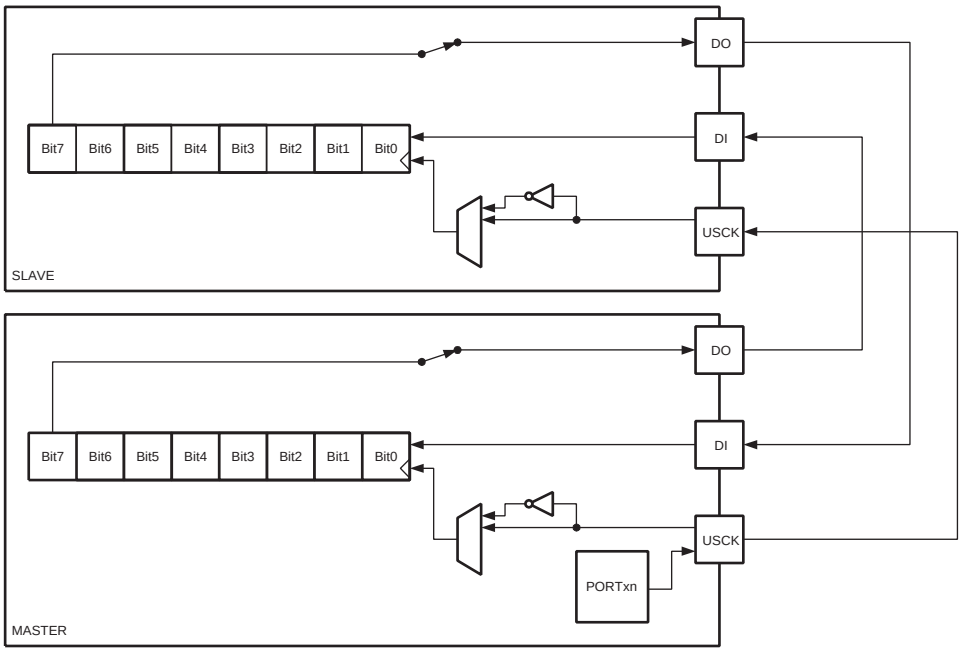
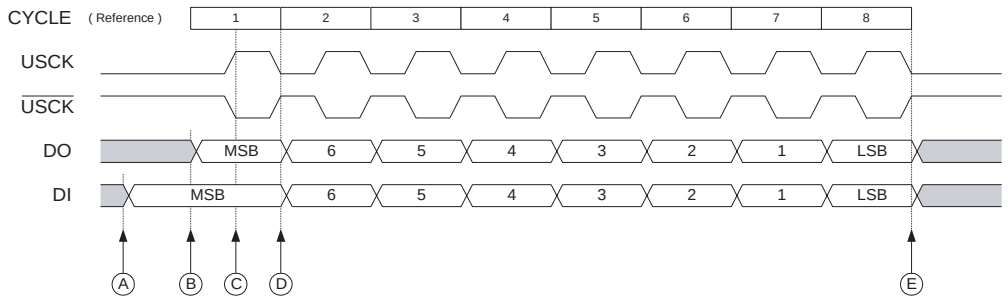



Figure 77 shows two USI units operating in Three-wire mode, one as Master and one as Slave. The two Shift Registers are interconnected in such way that after eight USCK clocks, the data in each register are interchanged. The same clock also increments the USI's 4-bit counter. The Counter Overflow (interrupt) Flag, or USIOIF, can therefore be used to determine when a transfer is completed. The clock is generated by the Master device software by toggling the USCK pin via the PORT Register or by writing a one to the USITC bit in USICR.

**Figure 78.** Three-wire Mode, Timing Diagram




The Three-wire mode timing is shown in Figure 78. At the top of the figure is a USCK cycle reference. One bit is shifted into the USI Shift Register (USIDR) for each of these cycles. The USCK timing is shown for both external clock modes. In External Clock mode 0 (USICS0 = 0), DI is sampled at positive edges, and DO is changed (Data Register is shifted by one) at negative edges. External Clock mode 1 (USICS0 = 1) uses the opposite edges versus mode 0, i.e., samples data at negative and changes the output at positive edges. The USI clock modes corresponds to the SPI data mode 0 and 1.

Referring to the timing diagram (Figure 78.), a bus transfer involves the following steps:

1. The Slave device and Master device sets up its data output and, depending on the protocol used, enables its output driver (mark A and B). The output is set up by writing the data to be transmitted to the Serial Data Register. Enabling of the output is done by setting the corresponding bit in the port Data Direction Register. Note that point A and B does not have any specific order, but both must be at least one half USCK cycle before point C where the data is sampled. This must be done to ensure that the data setup requirement is satisfied. The 4-bit counter is reset to zero.
2. The Master generates a clock pulse by software toggling the USCK line twice (C and D). The bit value on the slave and master's data input (DI) pin is sampled by the USI on the first edge (C), and the data output is changed on the opposite edge (D). The 4-bit counter will count both edges.
3. Step 2. is repeated eight times for a complete register (byte) transfer.
4. After eight clock pulses (i.e., 16 clock edges) the counter will overflow and indicate that the transfer is completed. The data bytes transferred must now be processed before a new transfer can be initiated. The overflow interrupt will wake up the processor if it is set to Idle mode. Depending of the protocol used the slave device can now set its output to high impedance.

**SPI Master Operation Example**

The following code demonstrates how to use the USI module as a SPI Master:

```
SPITransfer:
    sts    USIDR,r16
    ldi    r16,(1<<USIOIF)
    sts    USISR,r16
    ldi    r16,(1<<USIWM0)|(1<<USICS1)|(1<<USICLK)|(1<<USITC)
SPITransfer_loop:
    sts    USICR,r16
    lds    r16, USISR
    sbrs   r16, USIOIF
    rjmp   SPITransfer_loop
    lds    r16,USIDR
```

```
ret
```

The code is size optimized using only eight instructions (+ ret). The code example assumes that the DO and USCK pins are enabled as output in the DDRE Register. The value stored in register r16 prior to the function is called is transferred to the Slave device, and when the transfer is completed the data received from the Slave is stored back into the r16 Register.

The second and third instructions clears the USI Counter Overflow Flag and the USI counter value. The fourth and fifth instruction set Three-wire mode, positive edge Shift Register clock, count at USITC strobe, and toggle USCK. The loop is repeated 16 times.

The following code demonstrates how to use the USI module as a SPI Master with maximum speed (fsck = fck/4):

```
SPITransfer_Fast:

    sts    USIDR,r16
    ldi    r16,(1<<USIWM0)|(0<<USICS0)|(1<<USITC)
    ldi    r17,(1<<USIWM0)|(0<<USICS0)|(1<<USITC)|(1<<USICLK)

    sts    USICR,r16 ; MSB
    sts    USICR,r17
    sts    USICR,r16
    sts    USICR,r17
    sts    USICR,r16
    sts    USICR,r17
    sts    USICR,r16
    sts    USICR,r17
    sts    USICR,r16
    sts    USICR,r17
    sts    USICR,r16
    sts    USICR,r17
    sts    USICR,r16
    sts    USICR,r17
    sts    USICR,r16 ; LSB
    sts    USICR,r17

    lds    r16,USIDR
ret
```

**SPI Slave Operation Example**

The following code demonstrates how to use the USI module as a SPI Slave:

```
init:
    ldi    r16,(1<<USIWM0)|(1<<USICS1)
    sts    USICR,r16
...
SlaveSPITransfer:
    sts    USIDR,r16
    ldi    r16,(1<<USIOIF)
    sts    USISR,r16
SlaveSPITransfer_loop:
    lds    r16, USISR
    sbrs   r16, USIOIF
    rjmp   SlaveSPITransfer_loop
    lds    r16,USIDR
    ret
```

The code is size optimized using only eight instructions (+ ret). The code example assumes that the DO is configured as output and USCK pin is configured as input in the DDR Register. The value stored in register r16 prior to the function is called is trans-

ferred to the master device, and when the transfer is completed the data received from the Master is stored back into the r16 Register.

Note that the first two instructions is for initialization only and needs only to be executed once.These instructions sets Three-wire mode and positive edge Shift Register clock. The loop is repeated until the USI Counter Overflow Flag is set.

### Two-wire Mode

The USI Two-wire mode is compliant to the Inter IC (TWI) bus protocol, but without slew rate limiting on outputs and input noise filtering. Pin names used by this mode are SCL and SDA.

**Figure 79.** Two-wire Mode Operation, Simplified Diagram

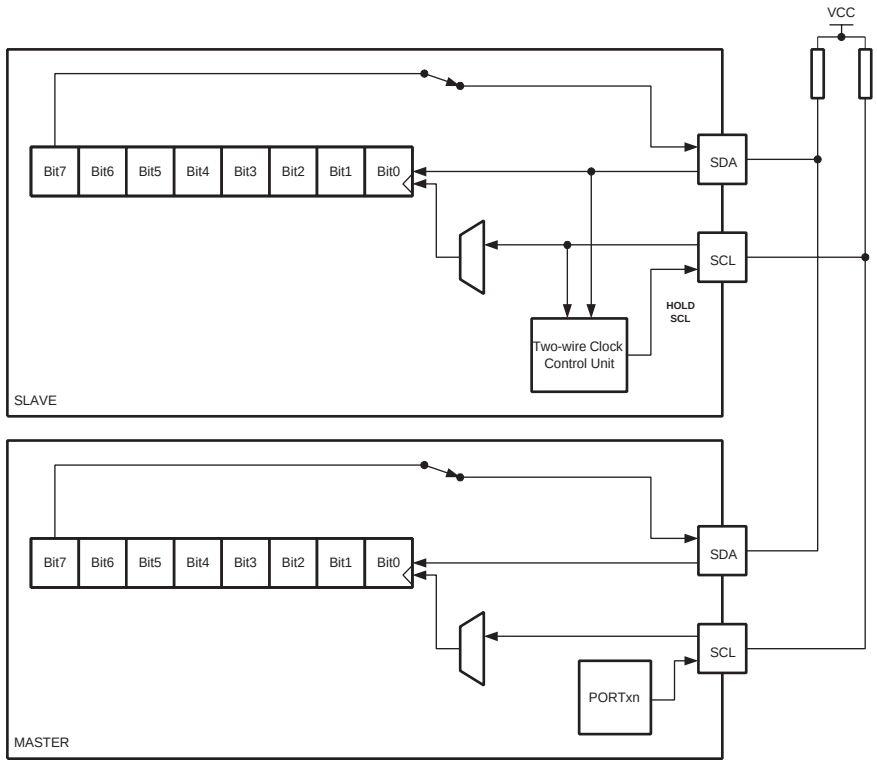
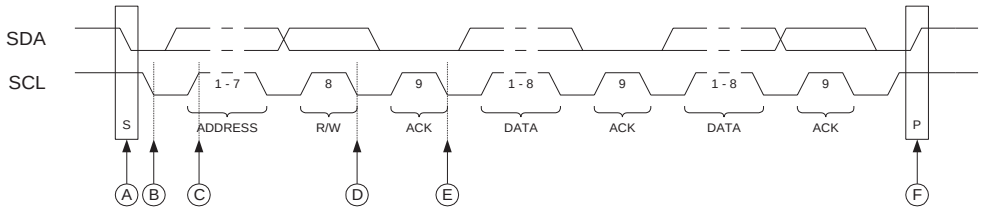



Figure 79 shows two USI units operating in Two-wire mode, one as Master and one as Slave. It is only the physical layer that is shown since the system operation is highly dependent of the communication scheme used. The main differences between the Master and Slave operation at this level, is the serial clock generation which is always done by the Master, and only the Slave uses the clock control unit. Clock generation must be implemented in software, but the shift operation is done automatically by both devices. Note that only clocking on negative edge for shifting data is of practical use in this mode. The slave can insert wait states at start or end of transfer by forcing the SCL clock low. This means that the Master must always check if the SCL line was actually released after it has generated a positive edge.

Since the clock also increments the counter, a counter overflow can be used to indicate that the transfer is completed. The clock is generated by the master by toggling the USCK pin via the PORT Register.

The data direction is not given by the physical layer. A protocol, like the one used by the TWI-bus, must be implemented to control the data flow.

**Figure 80.** Two-wire Mode, Typical Timing Diagram

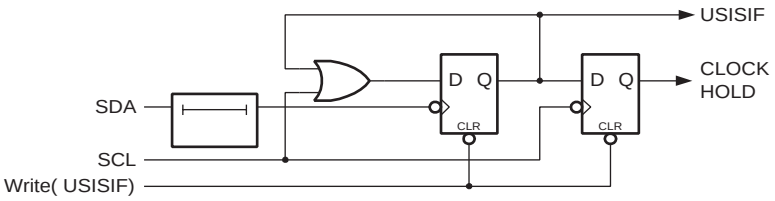



Referring to the timing diagram (Figure 80.), a bus transfer involves the following steps:

1. The a start condition is generated by the Master by forcing the SDA low line while the SCL line is high (A). SDA can be forced low either by writing a zero to bit 7 of the Shift Register, or by setting the corresponding bit in the PORT Register to zero. Note that the Data Direction Register bit must be set to one for the output to be enabled. The slave device's start detector logic (Figure 81.) detects the start condition and sets the USISIF Flag. The flag can generate an interrupt if necessary.
2. In addition, the start detector will hold the SCL line low after the Master has forced an negative edge on this line (B). This allows the Slave to wake up from sleep or complete its other tasks before setting up the Shift Register to receive the address. This is done by clearing the start condition flag and reset the counter.
3. The Master set the first bit to be transferred and releases the SCL line (C). The Slave samples the data and shift it into the Serial Register at the positive edge of the SCL clock.
4. After eight bits are transferred containing slave address and data direction (read or write), the Slave counter overflows and the SCL line is forced low (D). If the slave is not the one the Master has addressed, it releases the SCL line and waits for a new start condition.
5. If the Slave is addressed it holds the SDA line low during the acknowledgment cycle before holding the SCL line low again (i.e., the Counter Register must be set to 14 before releasing SCL at (D)). Depending of the R/W bit the Master or Slave enables its output. If the bit is set, a master read operation is in progress (i.e., the slave drives the SDA line) The slave can hold the SCL line low after the acknowledge (E).
6. Multiple bytes can now be transmitted, all in same direction, until a stop condition is given by the Master (F). Or a new start condition is given.

If the Slave is not able to receive more data it does not acknowledge the data byte it has last received. When the Master does a read operation it must terminate the operation by force the acknowledge bit low after the last byte transmitted.

**Figure 81.** Start Condition Detector, Logic Diagram



**Start Condition Detector**

The start condition detector is shown in Figure 81. The SDA line is delayed (in the range of 50 to 300 ns) to ensure valid sampling of the SCL line. The start condition detector is only enabled in Two-wire mode.

The start condition detector is working asynchronously and can therefore wake up the processor from the Power-down sleep mode. However, the protocol used might have restrictions on the SCL hold time. Therefore, when using this feature in this case the Oscillator start-up time set by the CKSEL Fuses (see “Clock Systems and their Distribution” on page 25) must also be taken into the consideration. Refer to the USISIF bit description on page 186 for further details.

## Alternative USI Usage

When the USI unit is not used for serial communication, it can be set up to do alternative tasks due to its flexible design.

**Half-duplex Asynchronous Data Transfer**

By utilizing the Shift Register in Three-wire mode, it is possible to implement a more compact and higher performance UART than by software only.

**4-bit Counter**

The 4-bit counter can be used as a stand-alone counter with overflow interrupt. Note that if the counter is clocked externally, both clock edges will generate an increment.

**12-bit Timer/Counter**

Combining the USI 4-bit counter and Timer/Counter0 allows them to be used as a 12-bit counter.

**Edge Triggered External Interrupt**

By setting the counter to maximum value (F) it can function as an additional external interrupt. The Overflow Flag and Interrupt Enable bit are then used for the external interrupt. This feature is selected by the USICS1 bit.

**Software Interrupt**

The counter overflow interrupt can be used as a software interrupt triggered by a clock strobe.

## USI Register Descriptions

**USI Data Register – USIDR**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | MSB | | | | | | | LSB | USIDR |
| 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 USI uses no buffering of the Serial Register, i.e., when accessing the Data Register (USIDR) the Serial Register is accessed directly. If a serial clock occurs at the same cycle the register is written, the register will contain the value written and no shift is performed. A (left) shift operation is performed depending of the USICS1..0 bits setting. The shift operation can be controlled by an external clock edge, by a Timer/Counter0 Compare Match, or directly by software using the USICLK strobe bit. Note that even when no wire mode is selected (USIWM1..0 = 0) both the external data input (DI/SDA) and the external clock input (USCK/SCL) can still be used by the Shift Register.

The output pin in use, DO or SDA depending on the wire mode, is connected via the output latch to the most significant bit (bit 7) of the Data Register. The output latch is open (transparent) during the first half of a serial clock cycle when an external clock source is selected (USICS1 = 1), and constantly open when an internal clock source is used (USICS1 = 0). The output will be changed immediately when a new MSB written as long as the latch is open. The latch ensures that data input is sampled and data output is changed on opposite clock edges.

Note that the corresponding Data Direction Register to the pin must be set to one for enabling data output from the Shift Register.

### USI Status Register – USISR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | USISIF | USIOIF | USIPF | USIDC | USICNT3 | USICNT2 | USICNT1 | USICNT0 | USISR |
| Read/Write | R/W | R/W | R/W | R | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

The Status Register contains Interrupt Flags, line Status Flags and the counter value.

- **Bit 7 – USISIF: Start Condition Interrupt Flag**

When Two-wire mode is selected, the USISIF Flag is set (to one) when a start condition is detected. When output disable mode or Three-wire mode is selected, the flag is set when the 4-bit counter is incremented.

An interrupt will be generated when the flag is set while the USISIE bit in USICR and the Global Interrupt Enable Flag are set. The flag will only be cleared by writing a logical one to the USISIF bit. Clearing this bit will release the start detection hold of USCL in Two-wire mode.

A start condition interrupt will wake up the processor from all sleep modes.

- **Bit 6 – USIOIF: Counter Overflow Interrupt Flag**

This flag is set (one) when the 4-bit counter overflows (i.e., at the transition from 15 to 0). An interrupt will be generated when the flag is set while the USIOIE bit in USICR and the Global Interrupt Enable Flag are set. The flag will only be cleared if a one is written to the USIOIF bit. Clearing this bit will release the counter overflow hold of SCL in Two-wire mode.

A counter overflow interrupt will wake up the processor from Idle sleep mode.

- **Bit 5 – USIPF: Stop Condition Flag**

When Two-wire mode is selected, the USIPF Flag is set (one) when a stop condition is detected. The flag is cleared by writing a one to this bit. Note that this is not an Interrupt Flag. This signal is useful when implementing Two-wire bus master arbitration.

- **Bit 4 – USIDC: Data Output Collision**

This bit is logical one when bit 7 in the Shift Register differs from the physical pin value. The flag is only valid when Two-wire mode is used. This signal is useful when implementing Two-wire bus master arbitration.

- **Bits 3..0 – USICNT3..0: Counter Value**

These bits reflect the current 4-bit counter value. The 4-bit counter value can directly be read or written by the CPU.

The 4-bit counter increments by one for each clock generated either by the external clock edge detector, by a Timer/Counter0 Compare Match, or by software using USICLK or USITC strobe bits. The clock source depends of the setting of the USICS1..0 bits. For external clock operation a special feature is added that allows the clock to be generated by writing to the USITC strobe bit. This feature is enabled by write a one to the USICLK bit while setting an external clock source (USICS1 = 1).

Note that even when no wire mode is selected (USIWM1..0 = 0) the external clock input (USCK/SCL) are can still be used by the counter.

### USI Control Register – USICR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|

| | USISIE | USIOIE | USIWM1 | USIWM0 | USICS1 | USICS0 | USICLK | USITC | USICR |
|---|---|---|---|---|---|---|---|---|---|
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | W | W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

The Control Register includes interrupt enable control, wire mode setting, Clock Select setting, and clock strobe.

- **Bit 7 – USISIE: Start Condition Interrupt Enable**

Setting this bit to one enables the Start Condition detector interrupt. If there is a pending interrupt when the USISIE and the Global Interrupt Enable Flag is set to one, this will immediately be executed. Refer to the USISIF bit description on page 186 for further details.

- **Bit 6 – USIOIE: Counter Overflow Interrupt Enable**

Setting this bit to one enables the Counter Overflow interrupt. If there is a pending interrupt when the USIOIE and the Global Interrupt Enable Flag is set to one, this will immediately be executed. Refer to the USIOIF bit description on page 186 for further details.

- **Bit 5..4 – USIWM1..0: Wire Mode**

These bits set the type of wire mode to be used. Basically only the function of the outputs are affected by these bits. Data and clock inputs are not affected by the mode selected and will always have the same function. The counter and Shift Register can therefore be clocked externally, and data input sampled, even when outputs are disabled. The relations between USIWM1..0 and the USI operation is summarized in Table 81.

**Table 81.** Relations between USIWM1..0 and the USI Operation

| USIWM1 | USIWM0 | Description |
|---|---|---|
| 0 | 0 | Outputs, clock hold, and start detector disabled. Port pins operates as normal. |
| 0 | 1 | Three-wire mode. Uses DO, DI, and USCK pins.<br>The *Data Output* (DO) pin overrides the corresponding bit in the PORT Register in this mode. However, the corresponding DDR bit still controls the data direction. When the port pin is set as input the pins pull-up is controlled by the PORT bit.<br>The *Data Input* (DI) and *Serial Clock* (USCK) pins do not affect the normal port operation. When operating as master, clock pulses are software generated by toggling the PORT Register, while the data direction is set to output. The USITC bit in the USICR Register can be used for this purpose. |
| 1 | 0 | Two-wire mode. Uses SDA (DI) and SCL (USCK) pins(1).<br>The *Serial Data* (SDA) and the *Serial Clock* (SCL) pins are bi-directional and uses open-collector output drives. The output drivers are enabled by setting the corresponding bit for SDA and SCL in the DDR Register.<br>When the output driver is enabled for the SDA pin, the output driver will force the line SDA low if the output of the Shift Register or the corresponding bit in the PORT Register is zero. Otherwise the SDA line will not be driven (i.e., it is released). When the SCL pin output driver is enabled the SCL line will be forced low if the corresponding bit in the PORT Register is zero, or by the start detector. Otherwise the SCL line will not be driven.<br>The SCL line is held low when a start detector detects a start condition and the output is enabled. Clearing the Start Condition Flag (USISIF) releases the line. The SDA and SCL pin inputs is not affected by enabling this mode. Pull-ups on the SDA and SCL port pin are disabled in Two-wire mode. |
| 1 | 1 | Two-wire mode. Uses SDA and SCL pins.<br>Same operation as for the Two-wire mode described above, except that the SCL line is also held low when a counter overflow occurs, and is held low until the Counter Overflow Flag (USIOIF) is cleared. |

Note: 1. The DI and USCK pins are renamed to *Serial Data* (SDA) and *Serial Clock* (SCL) respectively to avoid confusion between the modes of operation.

- **Bit 3..2 – USICS1..0: Clock Source Select**

These bits set the clock source for the Shift Register and counter. The data output latch ensures that the output is changed at the opposite edge of the sampling of the data input (DI/SDA) when using external clock source (USCK/SCL). When software strobe or Timer/Counter0 Compare Match clock option is selected, the output latch is transparent and therefore the output is changed immediately. Clearing the USICS1..0 bits enables software strobe option. When using this option, writing a one to the USICLK bit clocks both the Shift Register and the counter. For external clock source (USICS1 = 1), the USICLK bit is no longer used as a strobe, but selects between external clocking and software clocking by the USITC strobe bit.

Table 82 shows the relationship between the USICS1..0 and USICLK setting and clock source used for the Shift Register and the 4-bit counter.

**Table 82.** Relations between the USICS1..0 and USICLK Setting

| USICS1 | USICS0 | USICLK | Shift Register Clock Source | 4-bit Counter Clock Source |
|---|---|---|---|---|
| 0 | 0 | 0 | No Clock | No Clock |
| 0 | 0 | 1 | Software clock strobe (USICLK) | Software clock strobe (USICLK) |
| 0 | 1 | X | Timer/Counter0 Compare Match | Timer/Counter0 Compare Match |
| 1 | 0 | 0 | External, positive edge | External, both edges |
| 1 | 1 | 0 | External, negative edge | External, both edges |
| 1 | 0 | 1 | External, positive edge | Software clock strobe (USITC) |
| 1 | 1 | 1 | External, negative edge | Software clock strobe (USITC) |

- **Bit 1 – USICLK: Clock Strobe**

Writing a one to this bit location strobes the Shift Register to shift one step and the counter to increment by one, provided that the USICS1..0 bits are set to zero and by doing so the software clock strobe option is selected. The output will change immediately when the clock strobe is executed, i.e., in the same instruction cycle. The value shifted into the Shift Register is sampled the previous instruction cycle. The bit will be read as zero.

When an external clock source is selected (USICS1 = 1), the USICLK function is changed from a clock strobe to a Clock Select Register. Setting the USICLK bit in this case will select the USITC strobe bit as clock source for the 4-bit counter (see Table 82).

- **Bit 0 – USITC: Toggle Clock Port Pin**

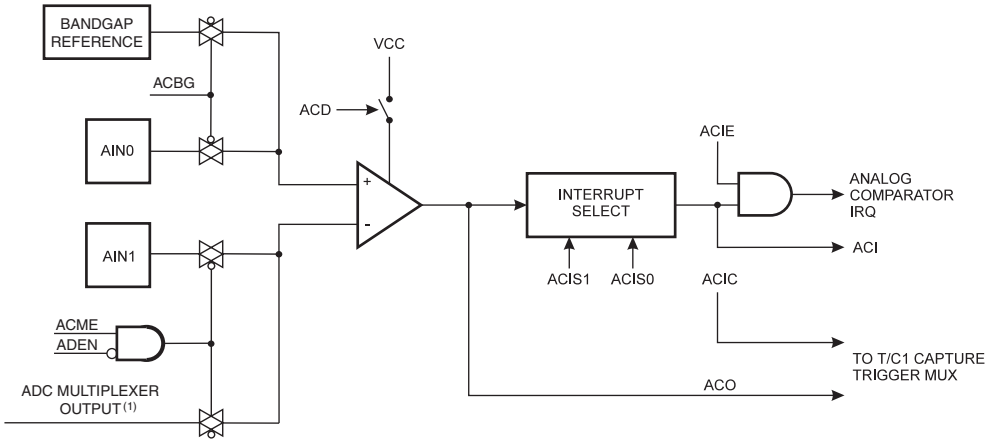
Writing a one to this bit location toggles the USCK/SCL value either from 0 to 1, or from 1 to 0. The toggling is independent of the setting in the Data Direction Register, but if the PORT value is to be shown on the pin the DDRE4 must be set as output (to one). This feature allows easy clock generation when implementing master devices. The bit will be read as zero.
When an external clock source is selected (USICS1 = 1) and the USICLK bit is set to one, writing to the USITC strobe bit will directly clock the 4-bit counter. This allows an early detection of when the transfer is done when operating as a master device.

## 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 82.

The Power Reduction ADC bit, PRADC, in "Power Reduction Register" on page 35 must be disabled by writing a logical zero to be able to use the ADC module.

**Figure 82.** Analog Comparator Block Diagram(2)




Notes: 1. See Table 84 on page 193.
2. Refer to Figure 1 on page 2, Figure 2 on page 3, and "Alternate Functions of Port E" on page 63 for Analog Comparator pin placement.

### ADC Control and Status Register B – ADCSRB

| 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 "Analog Comparator Multiplexed Input" on page 193.

### Analog Comparator Control and Status Register – ACSR

| 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 Compar-

ator 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 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 “Internal Voltage Reference” on page 43.

- **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 83.

**Table 83.** 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.

## 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 84. If ACME is cleared or ADEN is set, AIN1 is applied to the negative input to the Analog Comparator.

**Table 84.** 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 |

### Digital Input Disable Register 1 – DIDR1

| 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 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.

# Analog to Digital Converter

## Features

- **10-bit Resolution**
- **0.5 LSB Integral Non-linearity**
- **± 2 LSB Absolute Accuracy**
- **13 µs - 260 µs Conversion Time (50 kHz to 1 MHz ADC clock)**
- **Up to 15 kSPS at Maximum Resolution (200 kHz ADC clock)**
- **Eight Multiplexed Single Ended Input Channels**
- **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**
- **ADC Start Conversion by Auto Triggering on Interrupt Sources**
- **Interrupt on ADC Conversion Complete**
- **Sleep Mode Noise Canceler**

The ATmega325/3250/645/6450 features a 10-bit successive approximation ADC. The ADC is connected to an 8-channel Analog Multiplexer which allows eight single-ended voltage inputs constructed from the pins of Port F. The single-ended voltage inputs refer to 0V (GND).

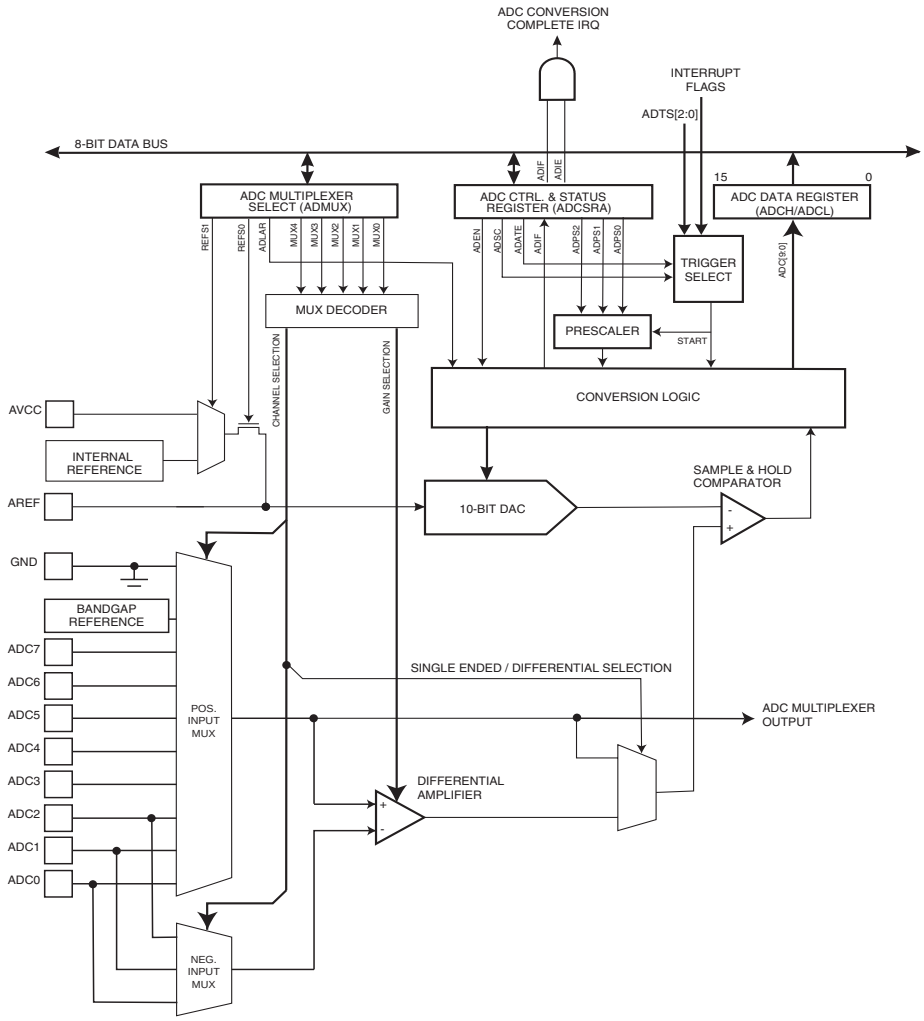
The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion. A block diagram of the ADC is shown in Figure 83.

The ADC has a separate analog supply voltage pin, AVCC. $AV_{CC}$ must not differ more than ± 0.3V from $V_{CC}$. See the paragraph “ADC Noise Canceler” on page 200 on how to connect this pin.

Internal reference voltages of nominally 1.1V or AVCC are provided On-chip. The voltage reference may be externally decoupled at the AREF pin by a capacitor for better noise performance.

The Power Reduction ADC bit, PRADC, in “Power Reduction Register” on page 35 must be disabled by writing a logical zero to be able to use the ADC module.

**Figure 83.** Analog to Digital Converter Block Schematic




## Operation

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 voltage on the AREF pin minus 1 LSB. Optionally, AVCC or an internal 1.1V reference voltage may be connected to the AREF pin by writing to the REFSn bits in the ADMUX Register. The internal voltage reference may thus be decoupled by an external capacitor at the AREF pin to improve noise immunity.

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. 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.

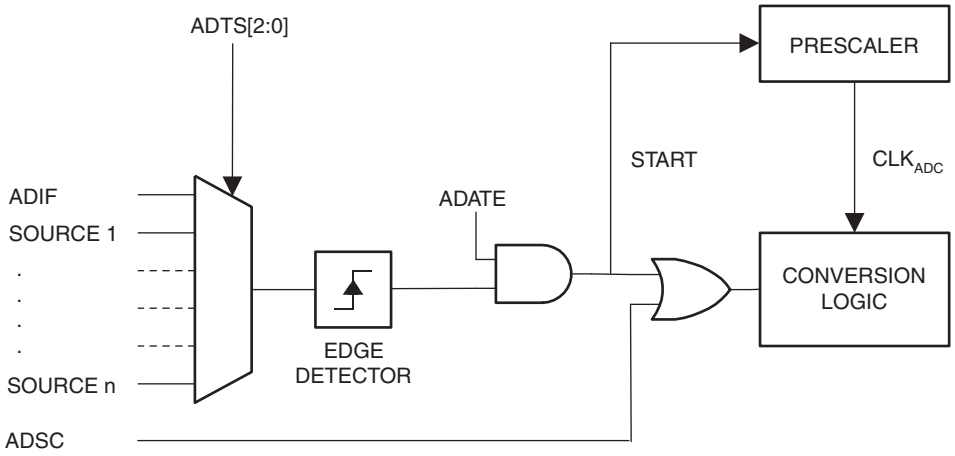
The ADC has its own interrupt which can be triggered when a conversion completes. When ADC access to the Data Registers is prohibited between reading of ADCH and ADCL, the interrupt will trigger even if the result is lost.

## Starting a Conversion

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 84.** ADC Auto Trigger Logic



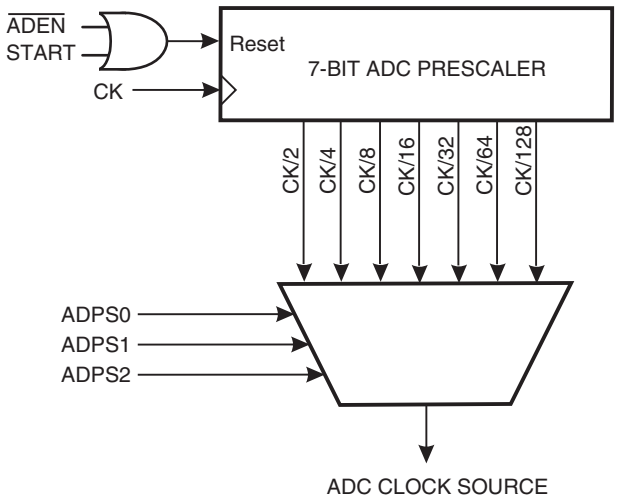

Using the ADC Interrupt Flag as a trigger source makes the ADC start a new conversion as soon as the ongoing conversion has finished. The ADC then operates in Free Running mode, constantly sampling and updating the ADC Data Register. The first conversion must be started by writing a logical one to the 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.

## Prescaling and Conversion Timing

**Figure 85.** ADC Prescaler




By default, the successive approximation circuitry requires an input clock frequency between 50 kHz and 200 kHz to get maximum resolution. If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200 kHz to get a higher sample rate.

The ADC module contains a prescaler, which generates an acceptable ADC clock frequency from any CPU frequency above 100 kHz. 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.

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.

When Auto Triggering is used, the prescaler is reset when the trigger event occurs. 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. When using Differential mode, along with Auto triggering from a source other than the ADC Conversion Complete, each conversion will require 25 ADC clocks. This is because the ADC must be disabled and re-enabled after every conversion.

In Free Running mode, a new conversion will be started immediately after the conversion completes, while ADSC remains high. For a summary of conversion times, see Table 85.

**Figure 86.** ADC Timing Diagram, First Conversion (Single Conversion Mode)

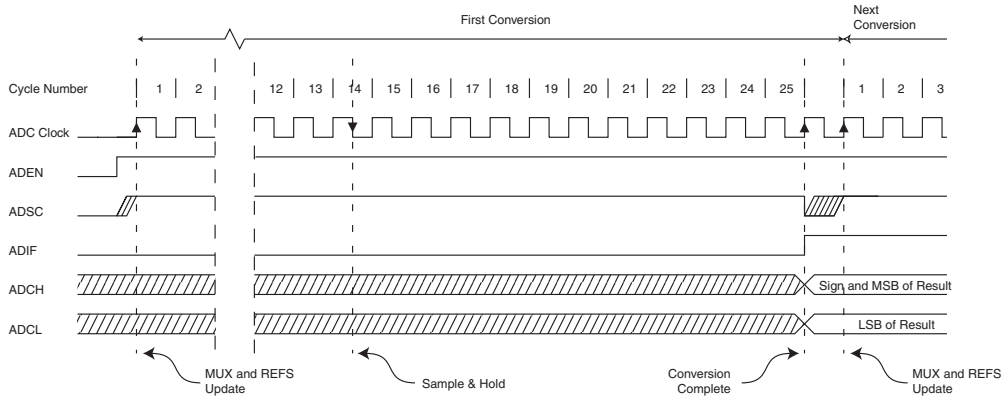



**Figure 87.** ADC Timing Diagram, Single Conversion

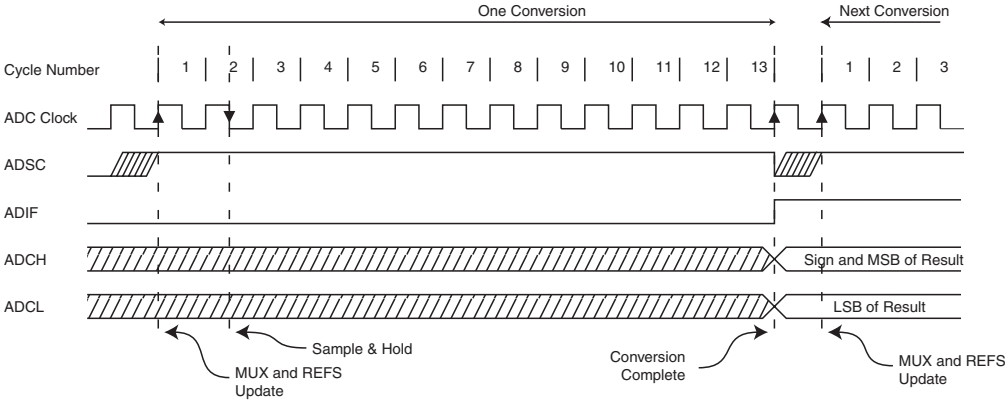



**Figure 88.** ADC Timing Diagram, Auto Triggered Conversion

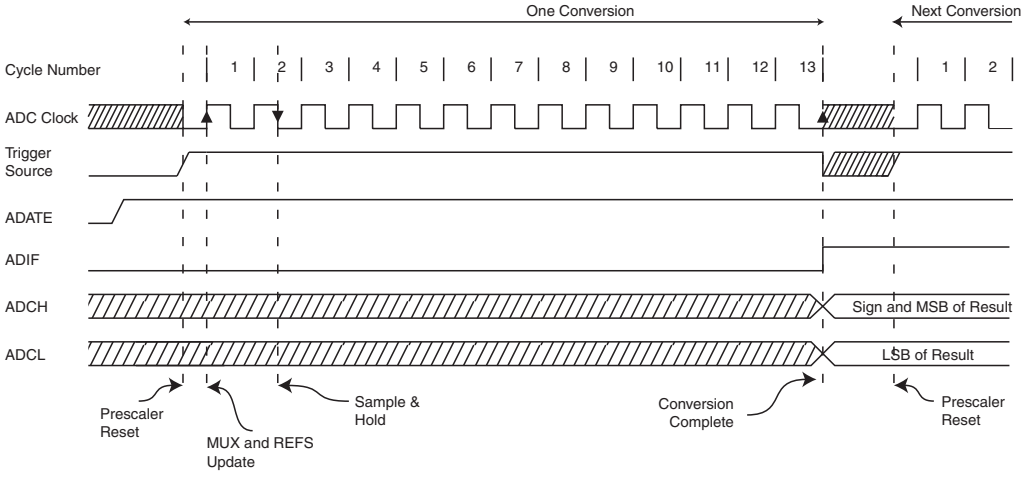


**Figure 89.** ADC Timing Diagram, Free Running Conversion

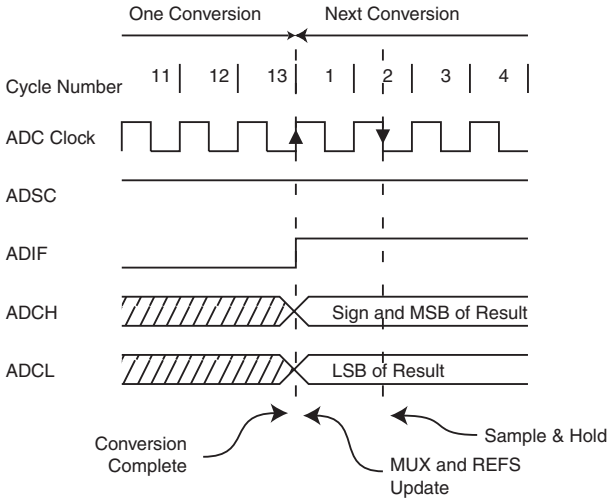



**Table 85.** ADC Conversion Time

| 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 |

## Changing Channel or Reference Selection

The MUXn and REFS1:0 bits 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:

1. When ADATE or ADEN is cleared.
2. During conversion, minimum one ADC clock cycle after the trigger event.
3. 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.

### 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.

### ADC Voltage Reference

The reference voltage for the ADC ($V_{REF}$) 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 AVCC, internal 1.1V reference, or external AREF pin.

AVCC is connected to the ADC through a passive switch. The internal 1.1V reference is generated from the internal bandgap reference ($V_{BG}$) through an internal buffer. In either case, the external AREF pin is directly connected to the ADC, and the reference voltage can be made more immune to noise by connecting a capacitor between the AREF pin and ground. $V_{REF}$ can also be measured at the AREF pin with a high impedant voltmeter. Note that $V_{REF}$ is a high impedant source, and only a capacitive load should be connected in a system.

If the user has a fixed voltage source connected to the AREF pin, the user may not use the other reference voltage options in the application, as they will be shorted to the external voltage. If no external voltage is applied to the AREF pin, the user may switch between AVCC and 1.1V as reference selection. The first ADC conversion result after switching reference voltage source may be inaccurate, and the user is advised to discard this result.

## ADC Noise Canceler

The ADC features a noise canceler that enables conversion during sleep mode to reduce 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:

1. 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.
2. Enter ADC Noise Reduction mode (or Idle mode). The ADC will start a conversion once the CPU has been halted.
3. 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 be automatically 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.

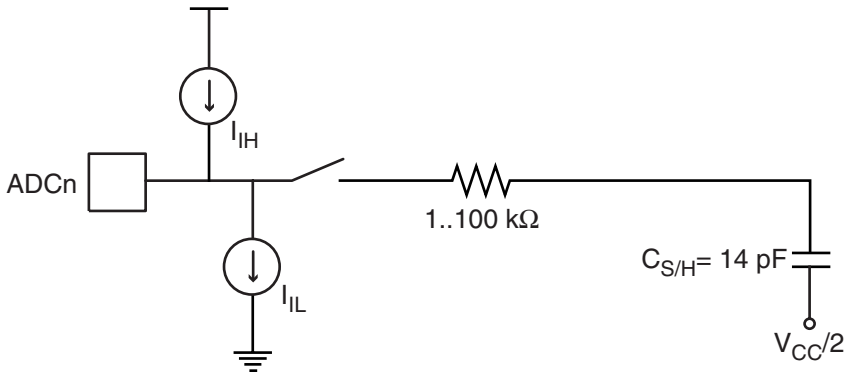
### Analog Input Circuitry

The analog input circuitry for single ended channels is illustrated in Figure 90. 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 10 kΩ 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, with can vary widely. The user is recommended to only use low impedant sources with slowly varying signals, since this minimizes the required charge transfer to the S/H capacitor.

Signal components higher than the Nyquist frequency ($f_{ADC}/2$) should not be present for either kind of channels, to avoid distortion from unpredictable signal convolution. The user is advised to remove high frequency components with a low-pass filter before applying the signals as inputs to the ADC.

**Figure 90.** Analog Input Circuitry

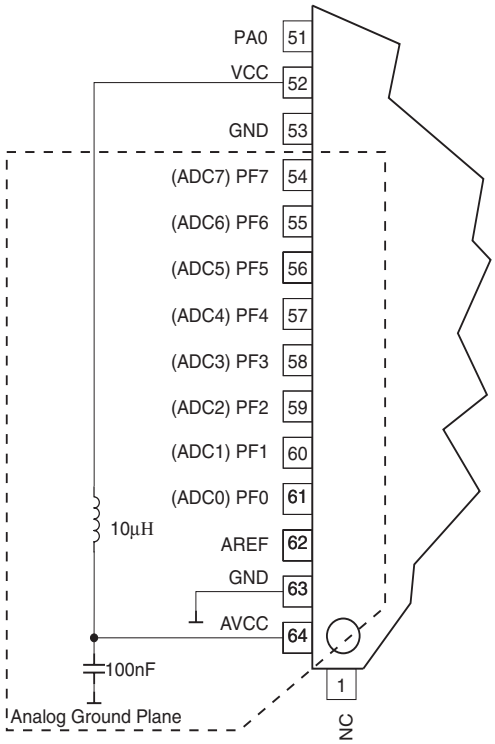



### Analog Noise Canceling Techniques

Digital circuitry inside and outside the device generates EMI which might affect the accuracy of analog measurements. If conversion accuracy is critical, the noise level can be reduced by applying the following techniques:

1. Keep analog signal paths as short as possible. Make sure analog tracks run over the analog ground plane, and keep them well away from high-speed switching digital tracks.
2. The AVCC pin on the device should be connected to the digital $V_{CC}$ supply voltage via an LC network as shown in Figure 91.
3. Use the ADC noise canceler function to reduce induced noise from the CPU.
4. If any ADC port pins are used as digital outputs, it is essential that these do not switch while a conversion is in progress.

**Figure 91.** ADC Power Connections



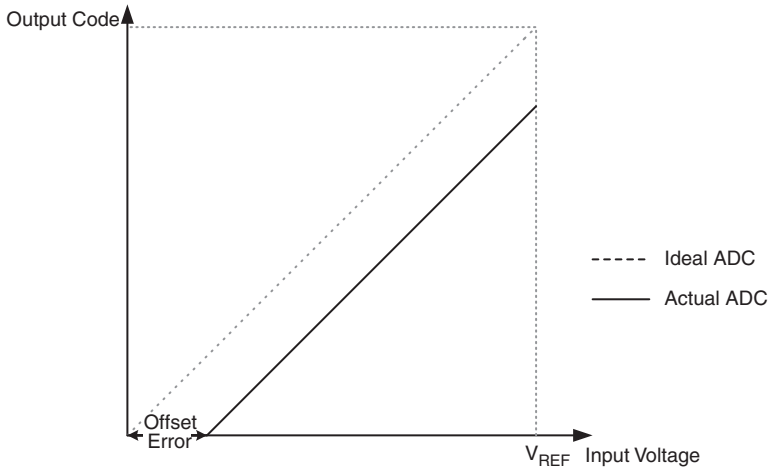

**ADC Accuracy Definitions**

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:

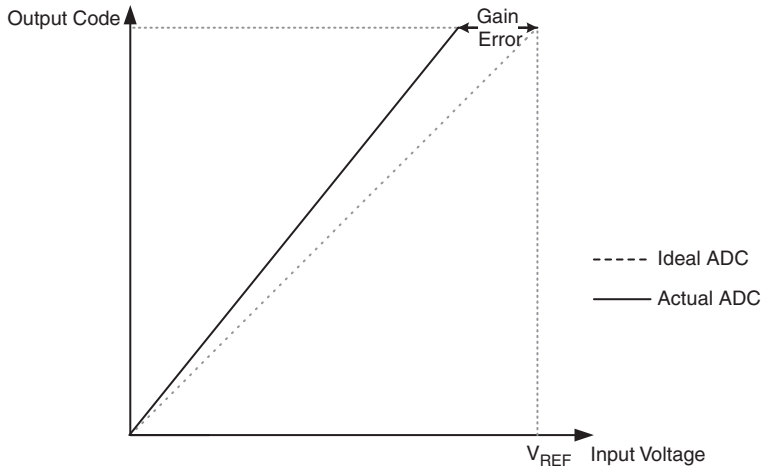
- Offset: The deviation of the first transition (0x000 to 0x001) compared to the ideal transition (at 0.5 LSB). Ideal value: 0 LSB.

**Figure 92.** Offset Error



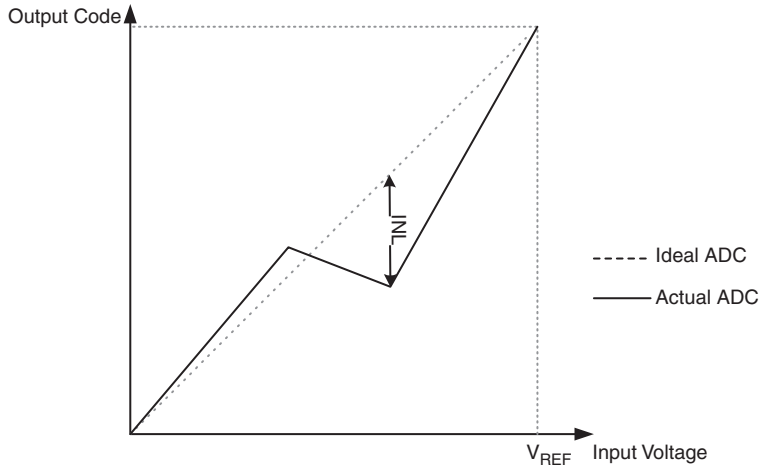
- Gain Error: After adjusting for offset, the Gain Error is found as the deviation of the last transition (0x3FE to 0x3FF) compared to the ideal transition (at 1.5 LSB below maximum). Ideal value: 0 LSB

**Figure 93.** Gain Error



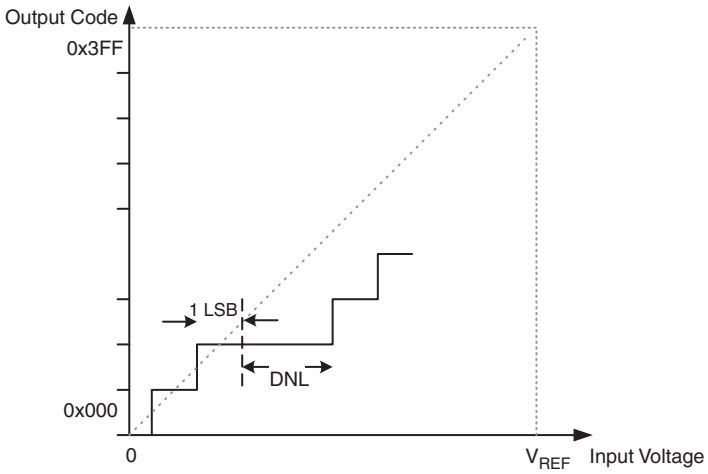

- Integral Non-linearity (INL): After adjusting for offset and gain error, the INL is the maximum deviation of an actual transition compared to an ideal transition for any code. Ideal value: 0 LSB.

**Figure 94.** 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 95.** 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.

## 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 87 on page 206 and Table 88 on page 207). 0x000 represents analog ground, and 0x3FF represents the selected reference voltage minus one LSB.

$$ADC = \frac{(V_{POS} - V_{NEG}) \cdot 512}{V_{REF}}$$

**Figure 96.** Differential Measurement Range

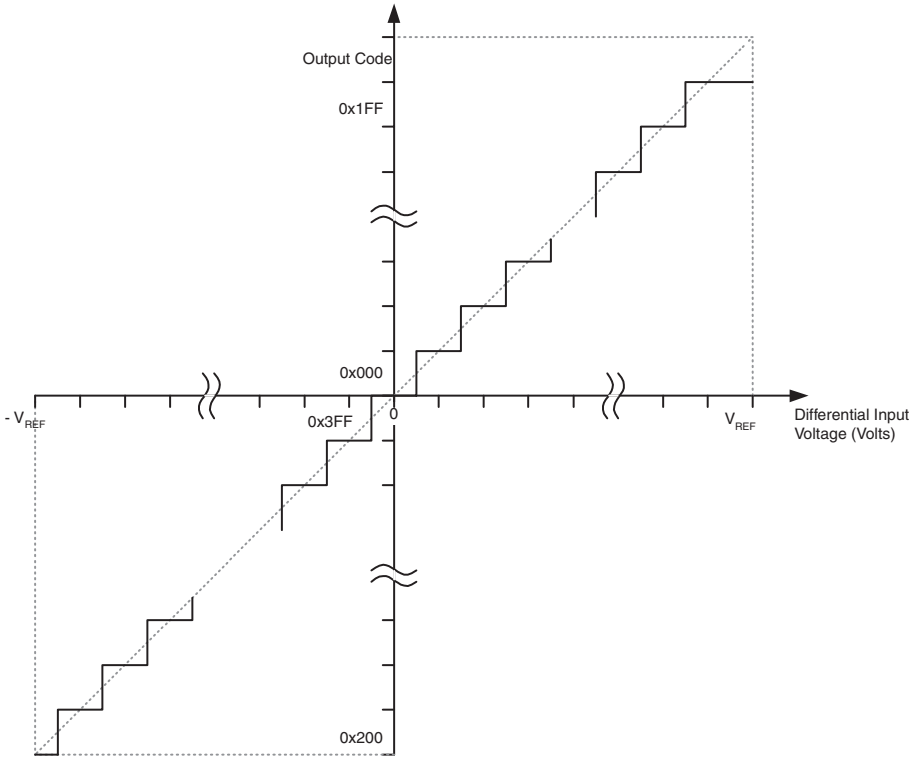



**Table 86.** Correlation Between Input Voltage and Output Codes

| $V_{ADCn}$ | Read Code | Corresponding Decimal Value |
|---|---|---|
| $V_{ADCm} + V_{REF}$ | 0x1FF | 511 |
| $V_{ADCm} + {}^{511}/_{512}\ V_{REF}$ | 0x1FF | 511 |
| $V_{ADCm} + {}^{510}/_{512}\ V_{REF}$ | 0x1FE | 510 |
| ... | ... | ... |
| $V_{ADCm} + {}^{1}/_{512}\ V_{REF}$ | 0x001 | 1 |
| $V_{ADCm}$ | 0x000 | 0 |
| $V_{ADCm} - {}^{1}/_{512}\ V_{REF}$ | 0x3FF | -1 |
| ... | ... | ... |
| $V_{ADCm} - {}^{511}/_{512}\ V_{REF}$ | 0x201 | -511 |
| $V_{ADCm} - V_{REF}$ | 0x200 | -512 |

ADMUX = 0xFB (ADC3 - ADC2, 1.1V reference, left adjusted result)

Voltage on ADC3 is 300 mV, voltage on ADC2 is 500 mV.

ADCR = 512 * (300 - 500) / 1100 = -93 = 0x3A3.

ADCL will thus read 0xC0, and ADCH will read 0xD8. Writing zero to ADLAR right adjusts the result: ADCL = 0xA3, ADCH = 0x03.

### ADC Multiplexer Selection Register – ADMUX

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | REFS1 | REFS0 | ADLAR | MUX4 | MUX3 | MUX2 | MUX1 | MUX0 | ADMUX |
| 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 7:6 – REFS1:0: Reference Selection Bits**

These bits select the voltage reference for the ADC, as shown in Table 87. 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). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

**Table 87.** Voltage Reference Selections for ADC

| REFS1 | REFS0 | Voltage Reference Selection |
|---|---|---|
| 0 | 0 | AREF, Internal Vref turned off |
| 0 | 1 | AVCC with external capacitor at AREF pin |
| 1 | 0 | Reserved |
| 1 | 1 | Internal 1.1V Voltage Reference with external capacitor at AREF pin |

- **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 “The ADC Data Register – ADCL and ADCH” on page 209.

- **Bits 4:0 – MUX4:0: Analog Channel Selection Bits**

The value of these bits selects which combination of analog inputs are connected to the ADC. See Table 88 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 88.** Input Channel Selections

| MUX4..0 | Single Ended Input | Positive Differential Input | Negative Differential Input |
|---|---|---|---|
| 00000 | ADC0 | N/A | |
| 00001 | ADC1 | | |
| 00010 | ADC2 | | |
| 00011 | ADC3 | | |
| 00100 | ADC4 | | |
| 00101 | ADC5 | | |
| 00110 | ADC6 | | |
| 00111 | ADC7 | | |
| 01000 | N/A | | |
| 01001 | | | |
| 01010 | | | |
| 01011 | | | |
| 01100 | | | |
| 01101 | | | |
| 01110 | | | |
| 01111 | | | |
| 10000 | | ADC0 | ADC1 |
| 10001 | | ADC1 | ADC1 |
| 10010 | | ADC2 | ADC1 |
| 10011 | | ADC3 | ADC1 |
| 10100 | | ADC4 | ADC1 |
| 10101 | | ADC5 | ADC1 |
| 10110 | | ADC6 | ADC1 |
| 10111 | | ADC7 | ADC1 |
| 11000 | | ADC0 | ADC2 |
| 11001 | | ADC1 | ADC2 |
| 11010 | | ADC2 | ADC2 |
| 11011 | | ADC3 | ADC2 |
| 11100 | | ADC4 | ADC2 |
| 11101 | | ADC5 | ADC2 |
| 11110 | 1.1V ($V_{BG}$) | N/A | |
| 11111 | 0V (GND) | | |

### ADC Control and Status Register A – ADCSRA

| 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 XTAL frequency and the input clock to the ADC.

**Table 89.** 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 |

### The ADC Data Register – ADCL and ADCH

*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 | |

*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 | |

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.

- **ADC9:0: ADC Conversion Result**

These bits represent the result from the conversion, as detailed in “ADC Conversion Result” on page 204.

### ADC Control and Status Register B – ADCSRB

| 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 – Res: Reserved Bit**

This bit is reserved for future use. To ensure compatibility with future devices, this bit 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 90.** 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 MatchA |
| 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 |

### Digital Input Disable Register 0 – DIDR0

| 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.

# JTAG Interface and On-chip Debug System

## Features

- **JTAG (IEEE std. 1149.1 Compliant) Interface**
- **Boundary-scan Capabilities According to the IEEE std. 1149.1 (JTAG) Standard**
- **Debugger Access to:**
  - **All Internal Peripheral Units**
  - **Internal and External RAM**
  - **The Internal Register File**
  - **Program Counter**
  - **EEPROM and Flash Memories**
- **Extensive On-chip Debug Support for Break Conditions, Including**
  - **AVR Break Instruction**
  - **Break on Change of Program Memory Flow**
  - **Single Step Break**
  - **Program Memory Break Points on Single Address or Address Range**
  - **Data Memory Break Points on Single Address or Address Range**
- **Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface**
- **On-chip Debugging Supported by AVR Studio®**

## Overview

The AVR IEEE std. 1149.1 compliant JTAG interface can be used for

- Testing PCBs by using the JTAG Boundary-scan capability
- Programming the non-volatile memories, Fuses and Lock bits
- On-chip debugging

A brief description is given in the following sections. Detailed descriptions for Programming via the JTAG interface, and using the Boundary-scan Chain can be found in the sections “Programming via the JTAG Interface” on page 275 and “IEEE 1149.1 (JTAG) Boundary-scan” on page 217**,** respectively. The On-chip Debug support is considered being private JTAG instructions, and distributed within ATMEL and to selected third party vendors only.

Figure 97 shows a block diagram of the JTAG interface and the On-chip Debug system. The TAP Controller is a state machine controlled by the TCK and TMS signals. The TAP Controller selects either the JTAG Instruction Register or one of several Data Registers as the scan chain (Shift Register) between the TDI – input and TDO – output. The Instruction Register holds JTAG instructions controlling the behavior of a Data Register.

The ID-Register, Bypass Register, and the Boundary-scan Chain are the Data Registers used for board-level testing. The JTAG Programming Interface (actually consisting of several physical and virtual Data Registers) is used for serial programming via the JTAG interface. The Internal Scan Chain and Break Point Scan Chain are used for On-chip debugging only.

## Test Access Port – TAP

The JTAG interface is accessed through four of the AVR’s pins. In JTAG terminology, these pins constitute the Test Access Port – TAP. These pins are:

- TMS: Test mode select. This pin is used for navigating through the TAP-controller state machine.
- TCK: Test Clock. JTAG operation is synchronous to TCK.
- TDI: Test Data In. Serial input data to be shifted in to the Instruction Register or Data Register (Scan Chains).
- TDO: Test Data Out. Serial output data from Instruction Register or Data Register.

The IEEE std. 1149.1 also specifies an optional TAP signal; TRST – Test ReSeT – which is not provided.

When the JTAGEN fuse is unprogrammed, these four TAP pins are normal port pins and the TAP controller is in reset. When programmed and the JTD bit in MCUCSR is cleared, the TAP pins are internally pulled high and the JTAG is enabled for Boundary-scan and programming. The device is shipped with this fuse programmed.

For the On-chip Debug system, in addition to the JTAG interface pins, the $\overline{\text{RESET}}$ pin is monitored by the debugger to be able to detect external reset sources. The debugger can also pull the $\overline{\text{RESET}}$ pin low to reset the whole system, assuming only open collectors on the reset line are used in the application.

**Figure 97.** Block Diagram

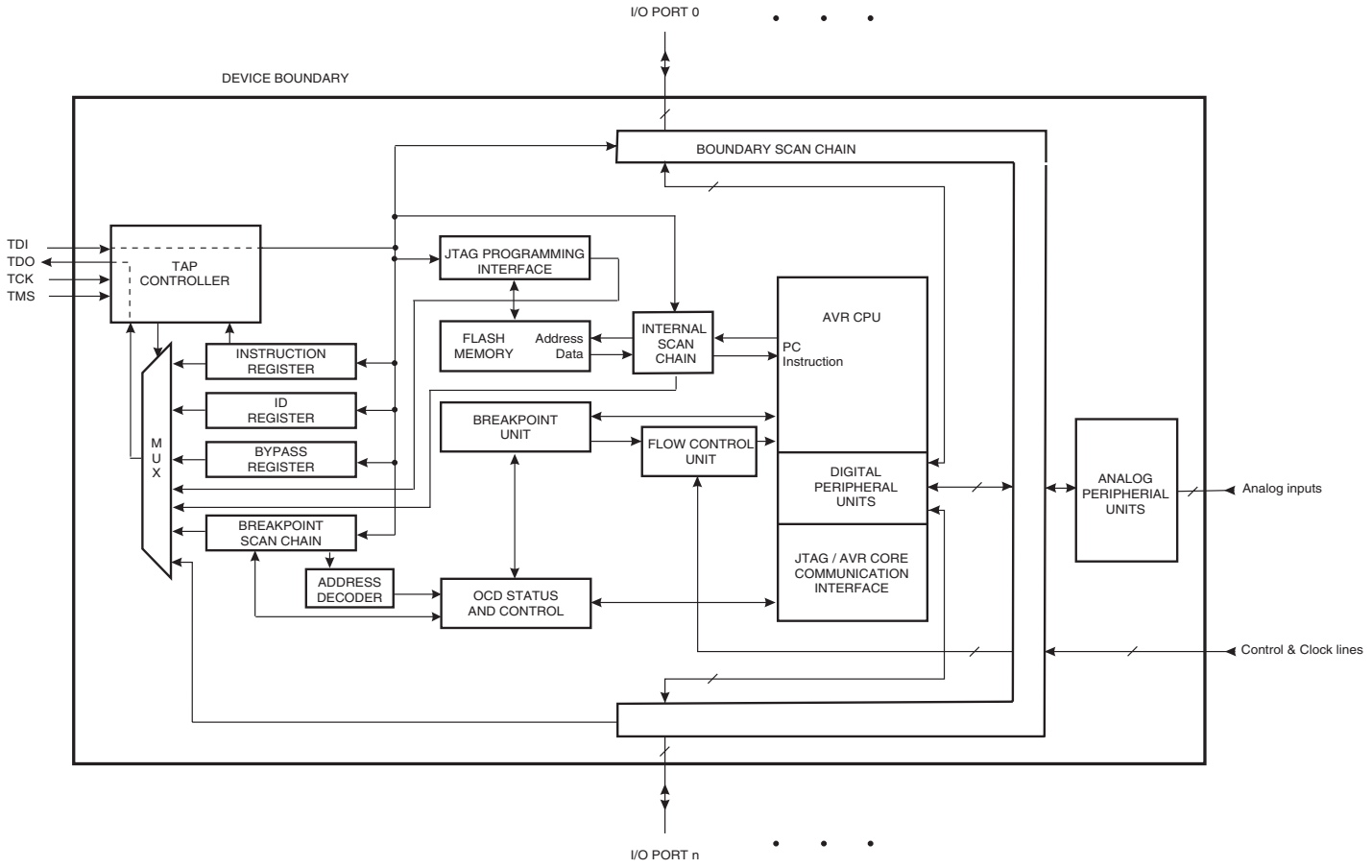
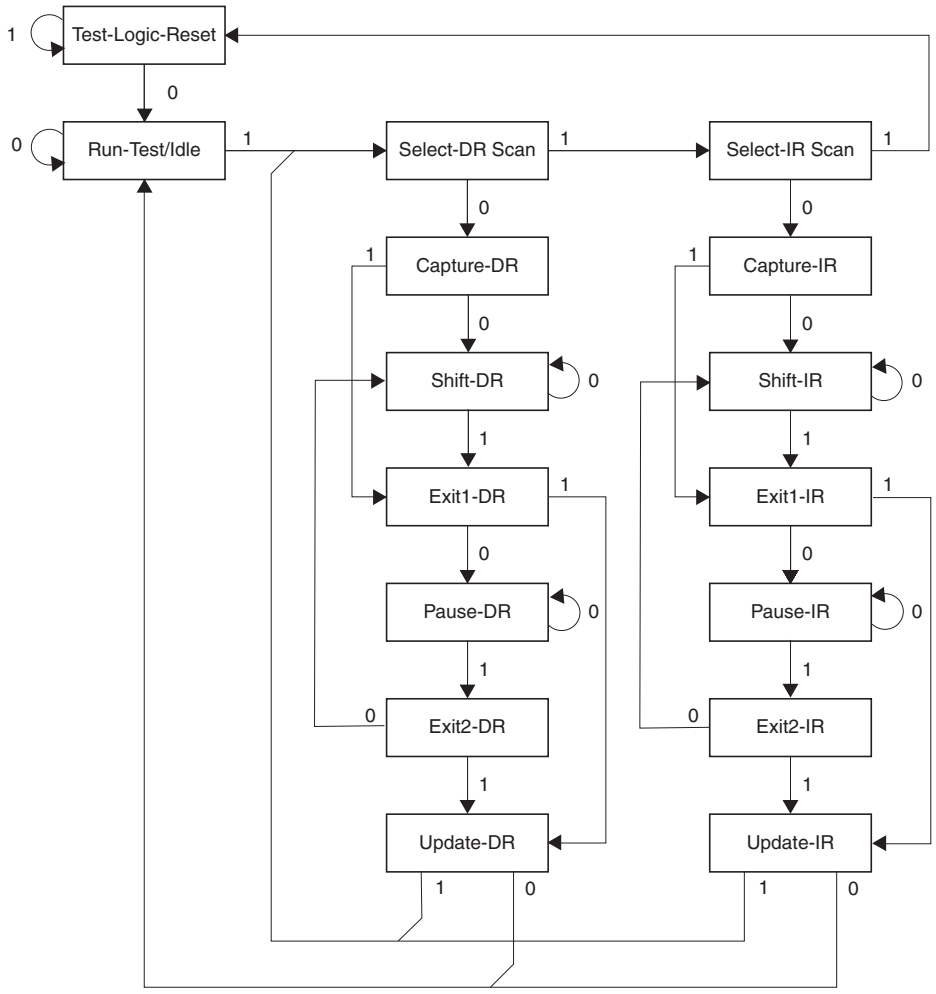


**Figure 98.** TAP Controller State Diagram




## TAP Controller

The TAP controller is a 16-state finite state machine that controls the operation of the Boundary-scan circuitry, JTAG programming circuitry, or On-chip Debug system. The state transitions depicted in Figure 98 depend on the signal present on TMS (shown adjacent to each state transition) at the time of the rising edge at TCK. The initial state after a Power-on Reset is Test-Logic-Reset.

As a definition in this document, the LSB is shifted in and out first for all Shift Registers.

Assuming Run-Test/Idle is the present state, a typical scenario for using the JTAG interface is:

- At the TMS input, apply the sequence 1, 1, 0, 0 at the rising edges of TCK to enter the Shift Instruction Register – Shift-IR state. While in this state, shift the four bits of the JTAG instructions into the JTAG Instruction Register from the TDI input at the rising edge of TCK. The TMS input must be held low during input of the 3 LSBs in order to remain in the Shift-IR state. The MSB of the instruction is shifted in when this state is left by setting TMS high. While the instruction is shifted in from the TDI pin, the captured IR-state 0x01 is shifted out on the TDO pin. The JTAG Instruction selects a particular Data Register as path between TDI and TDO and controls the circuitry surrounding the selected Data Register.
- Apply the TMS sequence 1, 1, 0 to re-enter the Run-Test/Idle state. The instruction is latched onto the parallel output from the Shift Register path in the Update-IR

state. The Exit-IR, Pause-IR, and Exit2-IR states are only used for navigating the state machine.

- At the TMS input, apply the sequence 1, 0, 0 at the rising edges of TCK to enter the Shift Data Register – Shift-DR state. While in this state, upload the selected Data Register (selected by the present JTAG instruction in the JTAG Instruction Register) from the TDI input at the rising edge of TCK. In order to remain in the Shift-DR state, the TMS input must be held low during input of all bits except the MSB. The MSB of the data is shifted in when this state is left by setting TMS high. While the Data Register is shifted in from the TDI pin, the parallel inputs to the Data Register captured in the Capture-DR state is shifted out on the TDO pin.
- Apply the TMS sequence 1, 1, 0 to re-enter the Run-Test/Idle state. If the selected Data Register has a latched parallel-output, the latching takes place in the Update-DR state. The Exit-DR, Pause-DR, and Exit2-DR states are only used for navigating the state machine.

As shown in the state diagram, the Run-Test/Idle state need not be entered between selecting JTAG instruction and using Data Registers, and some JTAG instructions may select certain functions to be performed in the Run-Test/Idle, making it unsuitable as an Idle state.

Note: Independent of the initial state of the TAP Controller, the Test-Logic-Reset state can always be entered by holding TMS high for five TCK clock periods.

For detailed information on the JTAG specification, refer to the literature listed in “Bibliography” on page 216.

## Using the Boundary-scan Chain

A complete description of the Boundary-scan capabilities are given in the section “IEEE 1149.1 (JTAG) Boundary-scan” on page 217.

## Using the On-chip Debug System

As shown in Figure 97, the hardware support for On-chip Debugging consists mainly of

- A scan chain on the interface between the internal AVR CPU and the internal peripheral units.
- Break Point unit.
- Communication interface between the CPU and JTAG system.

All read or modify/write operations needed for implementing the Debugger are done by applying AVR instructions via the internal AVR CPU Scan Chain. The CPU sends the result to an I/O memory mapped location which is part of the communication interface between the CPU and the JTAG system.

The Break Point Unit implements Break on Change of Program Flow, Single Step Break, two Program Memory Break Points, and two combined Break Points. Together, the four Break Points can be configured as either:

- 4 single Program Memory Break Points.
- 3 Single Program Memory Break Point + 1 single Data Memory Break Point.
- 2 single Program Memory Break Points + 2 single Data Memory Break Points.
- 2 single Program Memory Break Points + 1 Program Memory Break Point with mask (“range Break Point”).
- 2 single Program Memory Break Points + 1 Data Memory Break Point with mask (“range Break Point”).

A debugger, like the AVR Studio, may however use one or more of these resources for its internal purpose, leaving less flexibility to the end-user.

A list of the On-chip Debug specific JTAG instructions is given in “On-chip Debug Specific JTAG Instructions” on page 215.

The JTAGEN Fuse must be programmed to enable the JTAG Test Access Port. In addition, the OCDEN Fuse must be programmed and no Lock bits must be set for the On-chip debug system to work. As a security feature, the On-chip debug system is disabled when either of the LB1 or LB2 Lock bits are set. Otherwise, the On-chip debug system would have provided a back-door into a secured device.

The AVR Studio enables the user to fully control execution of programs on an AVR device with On-chip Debug capability, AVR In-Circuit Emulator, or the built-in AVR Instruction Set Simulator. AVR Studio® supports source level execution of Assembly programs assembled with Atmel Corporation’s AVR Assembler and C programs compiled with third party vendors’ compilers.

AVR Studio runs under Microsoft® Windows® 95/98/2000, Windows NT® and Windows XP®.

For a full description of the AVR Studio, please refer to the AVR Studio User Guide. Only highlights are presented in this document.

All necessary execution commands are available in AVR Studio, both on source level and on disassembly level. The user can execute the program, single step through the code either by tracing into or stepping over functions, step out of functions, place the cursor on a statement and execute until the statement is reached, stop the execution, and reset the execution target. In addition, the user can have an unlimited number of code Break Points (using the BREAK instruction) and up to two data memory Break Points, alternatively combined as a mask (range) Break Point.

## On-chip Debug Specific JTAG Instructions

The On-chip debug support is considered being private JTAG instructions, and distributed within ATMEL and to selected third party vendors only. Instruction opcodes are listed for reference.

**PRIVATE0; 0x8** Private JTAG instruction for accessing On-chip debug system.

**PRIVATE1; 0x9** Private JTAG instruction for accessing On-chip debug system.

**PRIVATE2; 0xA** Private JTAG instruction for accessing On-chip debug system.

**PRIVATE3; 0xB** Private JTAG instruction for accessing On-chip debug system.

## On-chip Debug Related Register in I/O Memory

### On-chip Debug Register – OCDR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | MSB/IDRD | | | | | | | LSB | OCDR |
| 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 OCDR Register provides a communication channel from the running program in the microcontroller to the debugger. The CPU can transfer a byte to the debugger by writing to this location. At the same time, an internal flag; I/O Debug Register Dirty – IDRD – is set to indicate to the debugger that the register has been written. When the CPU reads the OCDR Register the 7 LSB will be from the OCDR Register, while the MSB is the IDRD bit. The debugger clears the IDRD bit when it has read the information.

In some AVR devices, this register is shared with a standard I/O location. In this case, the OCDR Register can only be accessed if the OCDEN Fuse is programmed, and the debugger enables access to the OCDR Register. In all other cases, the standard I/O location is accessed.

Refer to the debugger documentation for further information on how to use this register.

## Using the JTAG Programming Capabilities

Programming of AVR parts via JTAG is performed via the 4-pin JTAG port, TCK, TMS, TDI, and TDO. These are the only pins that need to be controlled/observed to perform JTAG programming (in addition to power pins). It is not required to apply 12V externally. The JTAGEN Fuse must be programmed and the JTD bit in the MCUCR Register must be cleared to enable the JTAG Test Access Port.

The JTAG programming capability supports:

- Flash programming and verifying.
- EEPROM programming and verifying.
- Fuse programming and verifying.
- Lock bit programming and verifying.

The Lock bit security is exactly as in parallel programming mode. If the Lock bits LB1 or LB2 are programmed, the OCDEN Fuse cannot be programmed unless first doing a chip erase. This is a security feature that ensures no back-door exists for reading out the content of a secured device.

The details on programming through the JTAG interface and programming specific JTAG instructions are given in the section “Programming via the JTAG Interface” on page 275.

## Bibliography

For more information about general Boundary-scan, the following literature can be consulted:

- IEEE: IEEE Std. 1149.1-1990. IEEE Standard Test Access Port and Boundary-scan Architecture, IEEE, 1993.

Colin Maunder: The Board Designers Guide to Testable Logic Circuits, Addison-Wesley, 1992.

# IEEE 1149.1 (JTAG) Boundary-scan

## Features

- **JTAG (IEEE std. 1149.1 compliant) Interface**
- **Boundary-scan Capabilities According to the JTAG Standard**
- **Full Scan of all Port Functions as well as Analog Circuitry having Off-chip Connections**
- **Supports the Optional IDCODE Instruction**
- **Additional Public AVR_RESET Instruction to Reset the AVR**

## System Overview

The Boundary-scan chain has the capability of driving and observing the logic levels on the digital I/O pins, as well as the boundary between digital and analog logic for analog circuitry having off-chip connections. At system level, all ICs having JTAG capabilities are connected serially by the TDI/TDO signals to form a long Shift Register. An external controller sets up the devices to drive values at their output pins, and observe the input values received from other devices. The controller compares the received data with the expected result. In this way, Boundary-scan provides a mechanism for testing interconnections and integrity of components on Printed Circuits Boards by using the four TAP signals only.

The four IEEE 1149.1 defined mandatory JTAG instructions IDCODE, BYPASS, SAMPLE/PRELOAD, and EXTEST, as well as the AVR specific public JTAG instruction AVR_RESET can be used for testing the Printed Circuit Board. Initial scanning of the Data Register path will show the ID-Code of the device, since IDCODE is the default JTAG instruction. It may be desirable to have the AVR device in reset during test mode. If not reset, inputs to the device may be determined by the scan operations, and the internal software may be in an undetermined state when exiting the test mode. Entering reset, the outputs of any port pin will instantly enter the high impedance state, making the HIGHZ instruction redundant. If needed, the BYPASS instruction can be issued to make the shortest possible scan chain through the device. The device can be set in the reset state either by pulling the external $\overline{\text{RESET}}$ pin low, or issuing the AVR_RESET instruction with appropriate setting of the Reset Data Register.

The EXTEST instruction is used for sampling external pins and loading output pins with data. The data from the output latch will be driven out on the pins as soon as the EXTEST instruction is loaded into the JTAG IR-Register. Therefore, the SAMPLE/PRELOAD should also be used for setting initial values to the scan ring, to avoid damaging the board when issuing the EXTEST instruction for the first time. SAMPLE/PRELOAD can also be used for taking a snapshot of the external pins during normal operation of the part.

The JTAGEN Fuse must be programmed and the JTD bit in the I/O Register MCUCR must be cleared to enable the JTAG Test Access Port.

When using the JTAG interface for Boundary-scan, using a JTAG TCK clock frequency higher than the internal chip frequency is possible. The chip clock is not required to run.

## Data Registers

The Data Registers relevant for Boundary-scan operations are:

- Bypass Register
- Device Identification Register
- Reset Register
- Boundary-scan Chain

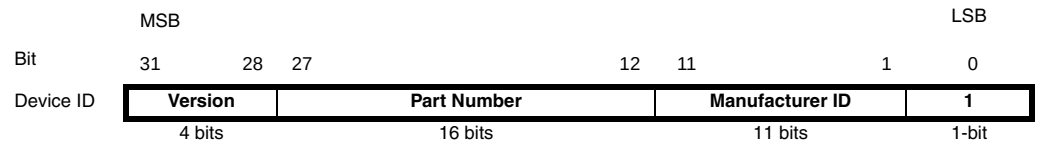
**Bypass Register**

The Bypass Register consists of a single Shift Register stage. When the Bypass Register is selected as path between TDI and TDO, the register is reset to 0 when leaving the Capture-DR controller state. The Bypass Register can be used to shorten the scan chain on a system when the other devices are to be tested.

**Device Identification Register**

Figure 99 shows the structure of the Device Identification Register.

**Figure 99.** The Format of the Device Identification Register




*Version*

Version is a 4-bit number identifying the revision of the component. The JTAG version number follows the revision of the device. Revision A is 0x0, revision B is 0x1 and so on.

*Part Number*

The part number is a 16-bit code identifying the component. The JTAG Part Number for ATmega325/3250/645/6450 is listed in Table 91.

**Table 91.** AVR JTAG Part Number

| Part Number | JTAG Part Number (Hex) |
|---|---|
| ATmega325 | 0x9505 |
| ATmega3250 | 0x9506 |
| ATmega645 | 0x9605 |
| ATmega6450 | 0x9606 |

*Manufacturer ID*

The Manufacturer ID is a 11-bit code identifying the manufacturer. The JTAG manufacturer ID for ATMEL is listed in Table 92.

**Table 92.** Manufacturer ID

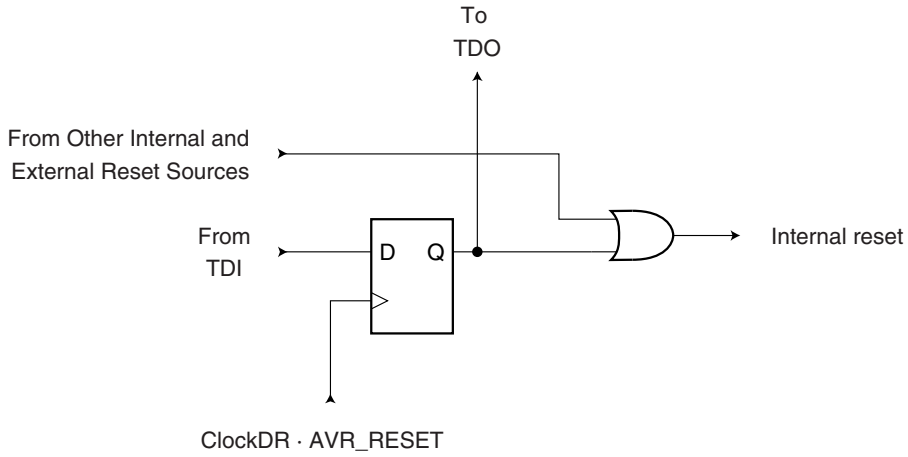
| Manufacturer | JTAG Manufacturer ID (Hex) |
|---|---|
| ATMEL | 0x01F |

**Reset Register**

The Reset Register is a test Data Register used to reset the part. Since the AVR tri-states Port Pins when reset, the Reset Register can also replace the function of the unimplemented optional JTAG instruction HIGHZ.

A high value in the Reset Register corresponds to pulling the external Reset low. The part is reset as long as there is a high value present in the Reset Register. Depending on the fuse settings for the clock options, the part will remain reset for a reset time-out period (refer to “Clock Sources” on page 26) after releasing the Reset Register. The output from this Data Register is not latched, so the reset will take place immediately, as shown in Figure 100.

**Figure 100.** Reset Register




**Boundary-scan Chain**

The Boundary-scan Chain has the capability of driving and observing the logic levels on the digital I/O pins, as well as the boundary between digital and analog logic for analog circuitry having off-chip connections.

See “Boundary-scan Chain” on page 221 for a complete description.

## Boundary-scan Specific JTAG Instructions

The Instruction Register is 4-bit wide, supporting up to 16 instructions. Listed below are the JTAG instructions useful for Boundary-scan operation. Note that the optional HIGHZ instruction is not implemented, but all outputs with tri-state capability can be set in high-impedant state by using the AVR_RESET instruction, since the initial state for all port pins is tri-state.

As a definition in this data sheet, the LSB is shifted in and out first for all Shift Registers.

The OPCODE for each instruction is shown behind the instruction name in hex format. The text describes which Data Register is selected as path between TDI and TDO for each instruction.

**EXTEST; 0x0**

Mandatory JTAG instruction for selecting the Boundary-scan Chain as Data Register for testing circuitry external to the AVR package. For port-pins, Pull-up Disable, Output Control, Output Data, and Input Data are all accessible in the scan chain. For Analog circuits having off-chip connections, the interface between the analog and the digital logic is in the scan chain. The contents of the latched outputs of the Boundary-scan chain is driven out as soon as the JTAG IR-Register is loaded with the EXTEST instruction.

The active states are:

- Capture-DR: Data on the external pins are sampled into the Boundary-scan Chain.
- Shift-DR: The Internal Scan Chain is shifted by the TCK input.
- Update-DR: Data from the scan chain is applied to output pins.

**IDCODE; 0x1**

Optional JTAG instruction selecting the 32 bit ID-Register as Data Register. The ID-Register consists of a version number, a device number and the manufacturer code chosen by JEDEC. This is the default instruction after power-up.

The active states are:

- Capture-DR: Data in the IDCODE Register is sampled into the Boundary-scan Chain.
- Shift-DR: The IDCODE scan chain is shifted by the TCK input.

**SAMPLE_PRELOAD; 0x2**

Mandatory JTAG instruction for pre-loading the output latches and taking a snap-shot of the input/output pins without affecting the system operation. However, the output latches are not connected to the pins. The Boundary-scan Chain is selected as Data Register.

The active states are:

- Capture-DR: Data on the external pins are sampled into the Boundary-scan Chain.
- Shift-DR: The Boundary-scan Chain is shifted by the TCK input.
- Update-DR: Data from the Boundary-scan chain is applied to the output latches. However, the output latches are not connected to the pins.

**AVR_RESET; 0xC**

The AVR specific public JTAG instruction for forcing the AVR device into the Reset mode or releasing the JTAG reset source. The TAP controller is not reset by this instruction. The one bit Reset Register is selected as Data Register. Note that the reset will be active as long as there is a logic “one” in the Reset Chain. The output from this chain is not latched.

The active states are:

- Shift-DR: The Reset Register is shifted by the TCK input.

**BYPASS; 0xF**

Mandatory JTAG instruction selecting the Bypass Register for Data Register.

The active states are:

- Capture-DR: Loads a logic “0” into the Bypass Register.
- Shift-DR: The Bypass Register cell between TDI and TDO is shifted.

## Boundary-scan Related Register in I/O Memory

**MCU Control Register – MCUCR**

The MCU Control Register contains control bits for general MCU functions.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | JTD | – | – | PUD | – | – | IVSEL | IVCE | MCUCR |
| Read/Write | R/W | R | R | R/W | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – JTD: JTAG Interface Disable**

When this bit is zero, the JTAG interface is enabled if the JTAGEN Fuse is programmed. If this bit is one, the JTAG interface is disabled. In order to avoid unintentional disabling or enabling of the JTAG interface, a timed sequence must be followed when changing this bit: The application software must write this bit to the desired value twice within four cycles to change its value. Note that this bit must not be altered when using the On-chip Debug system.

If the JTAG interface is left unconnected to other JTAG circuitry, the JTD bit should be set to one. The reason for this is to avoid static current at the TDO pin in the JTAG interface.

### MCU Status Register – MCUSR

The MCU Status Register provides information on which reset source caused an MCU reset.

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

- **Bit 4 – JTRF: JTAG Reset Flag**

This bit is set if a reset is being caused by a logic one in the JTAG Reset Register selected by the JTAG instruction AVR_RESET. This bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

## Boundary-scan Chain

The Boundary-scan chain has the capability of driving and observing the logic levels on the digital I/O pins, as well as the boundary between digital and analog logic for analog circuitry having off-chip connection.

### Scanning the Digital Port Pins

Figure 101 shows the Boundary-scan Cell for a bi-directional port pin with pull-up function. The cell consists of a standard Boundary-scan cell for the Pull-up Enable – PUExn – function, and a bi-directional pin cell that combines the three signals Output Control – OCxn, Output Data – ODxn, and Input Data – IDxn, into only a two-stage Shift Register. The port and pin indexes are not used in the following description

The Boundary-scan logic is not included in the figures in the Data Sheet. Figure 102 shows a simple digital port pin as described in the section “I/O-Ports” on page 52. The Boundary-scan details from Figure 101 replaces the dashed box in Figure 102.

When no alternate port function is present, the Input Data – ID – corresponds to the PINxn Register value (but ID has no synchronizer), Output Data corresponds to the PORT Register, Output Control corresponds to the Data Direction – DD Register, and the Pull-up Enable – PUExn – corresponds to logic expression $\overline{PUD} \cdot \overline{DDxn} \cdot PORTxn$.

Digital alternate port functions are connected outside the dotted box in Figure 102 to make the scan chain read the actual pin value. For Analog function, there is a direct connection from the external pin to the analog circuit, and a scan chain is inserted on the interface between the digital logic and the analog circuitry.

**Figure 101.** Boundary-scan Cell for Bi-directional Port Pin with Pull-up Function.

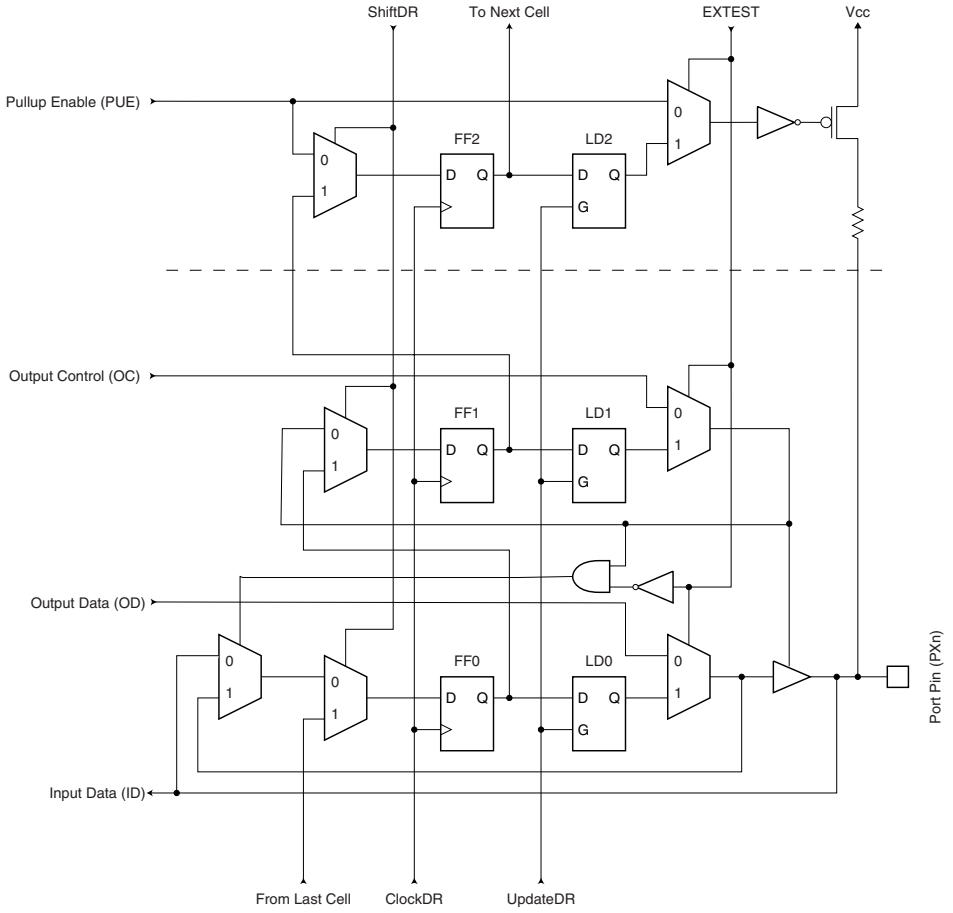
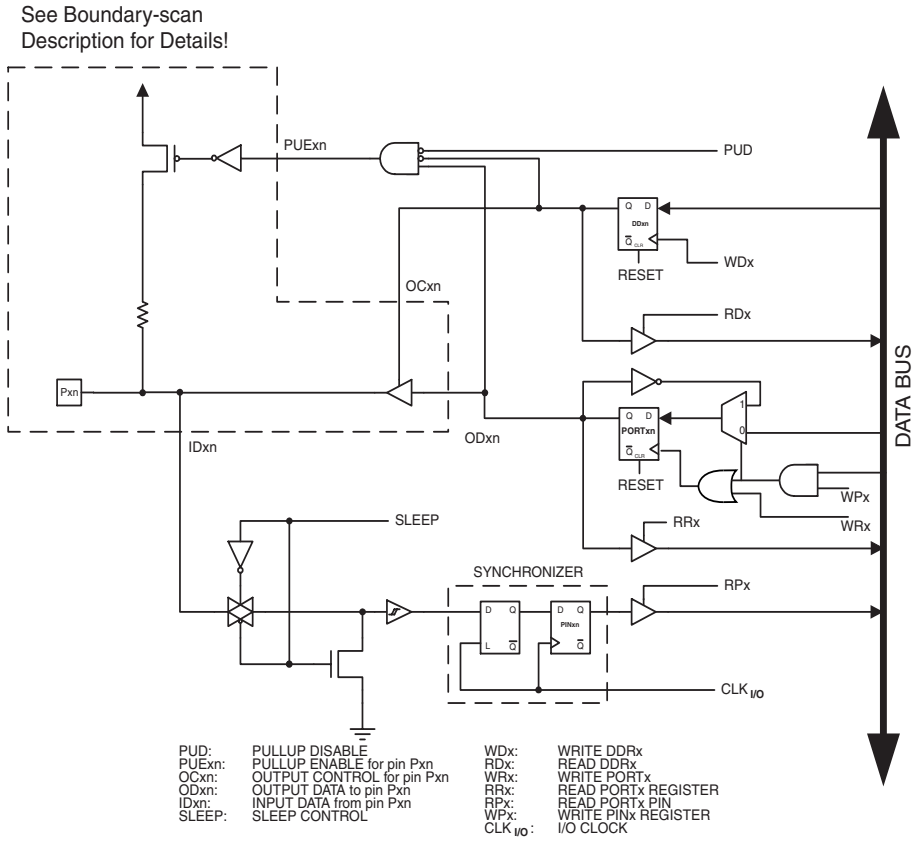


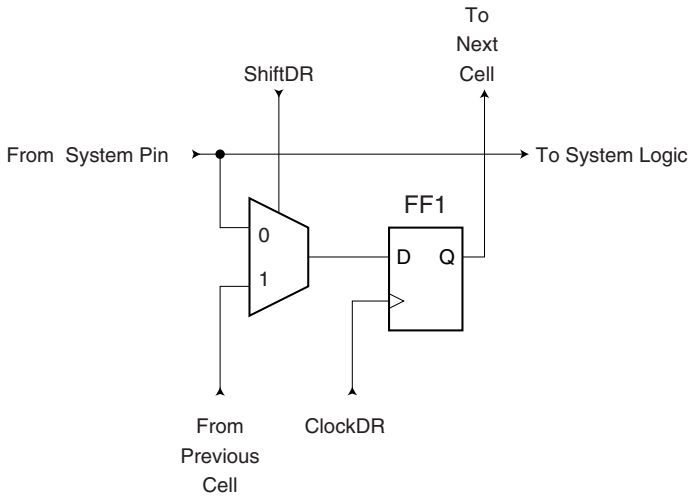
**Figure 102.** General Port Pin Schematic Diagram




**Scanning the RESET Pin**

The RESET pin accepts 5V active low logic for standard reset operation, and 12V active high logic for High Voltage Parallel programming. An observe-only cell as shown in Figure 103 is inserted both for the 5V reset signal; RSTT, and the 12V reset signal; RSTHV.

**Figure 103.** Observe-only Cell



## Scanning the Clock Pins

The AVR devices have many clock options selectable by fuses. These are: Internal RC Oscillator, External Clock, (High Frequency) Crystal Oscillator, Low-frequency Crystal Oscillator, and Ceramic Resonator.

Figure 104 shows how each Oscillator with external connection is supported in the scan chain. The Enable signal is supported with a general Boundary-scan cell, while the Oscillator/clock output is attached to an observe-only cell. In addition to the main clock, the timer Oscillator is scanned in the same way. The output from the internal RC Oscillator is not scanned, as this Oscillator does not have external connections.

**Figure 104.** Boundary-scan Cells for Oscillators and Clock Options

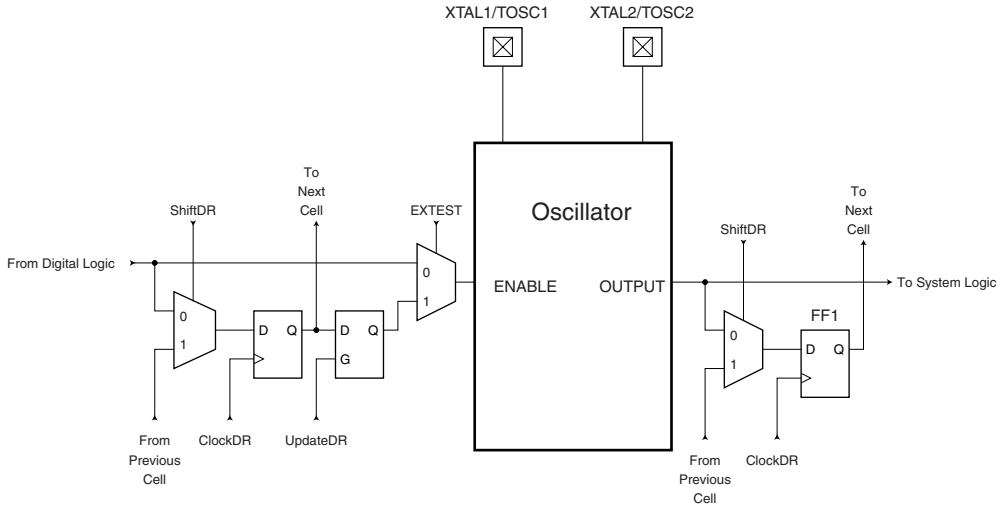



Table 93 summaries the scan registers for the external clock pin XTAL1, oscillators with XTAL1/XTAL2 connections as well as 32kHz Timer Oscillator.

**Table 93.** Scan Signals for the Oscillator(1)(2)(3)

| Enable Signal | Scanned Clock Line | Clock Option | Scanned Clock Line when not Used |
|---|---|---|---|
| EXTCLKEN | EXTCLK (XTAL1) | External Clock | 0 |
| OSCON | OSCCK | External Crystal<br>External Ceramic Resonator | 1 |
| OSC32EN | OSC32CK | Low Freq. External Crystal | 1 |

Notes:
1. Do not enable more than one clock source as main clock at a time.
2. Scanning an Oscillator output gives unpredictable results as there is a frequency drift between the internal Oscillator and the JTAG TCK clock. If possible, scanning an external clock is preferred.
3. The clock configuration is programmed by fuses. As a fuse is not changed run-time, the clock configuration is considered fixed for a given application. The user is advised to scan the same clock option as to be used in the final system. The enable signals are supported in the scan chain because the system logic can disable clock options in sleep modes, thereby disconnecting the Oscillator pins from the scan path if not provided.

### Scanning the Analog Comparator

The relevant Comparator signals regarding Boundary-scan are shown in Figure 105. The Boundary-scan cell from Figure 106 is attached to each of these signals. The signals are described in Table 94.

The Comparator need not be used for pure connectivity testing, since all analog inputs are shared with a digital port pin as well.

**Figure 105.** Analog Comparator

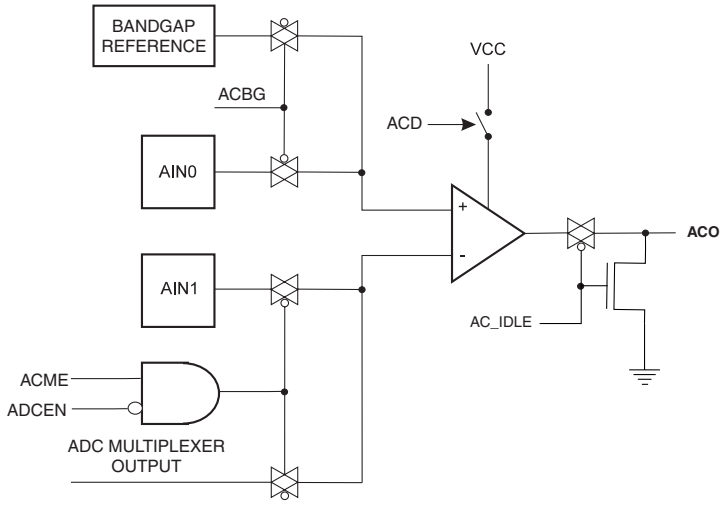



**Figure 106.** General Boundary-scan cell Used for Signals for Comparator and ADC

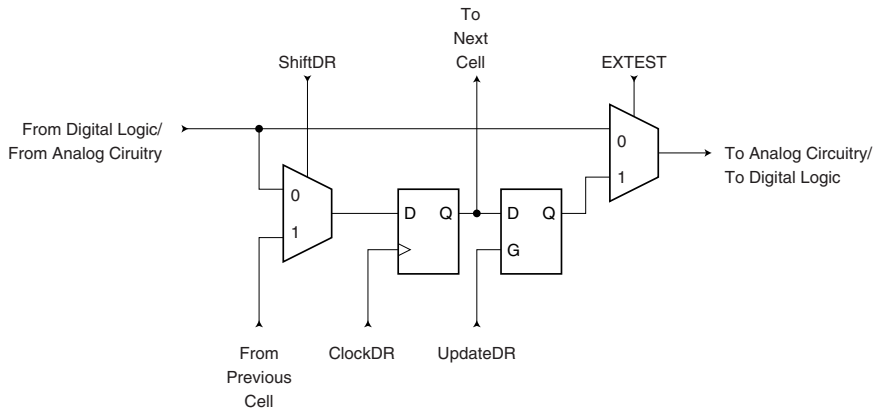


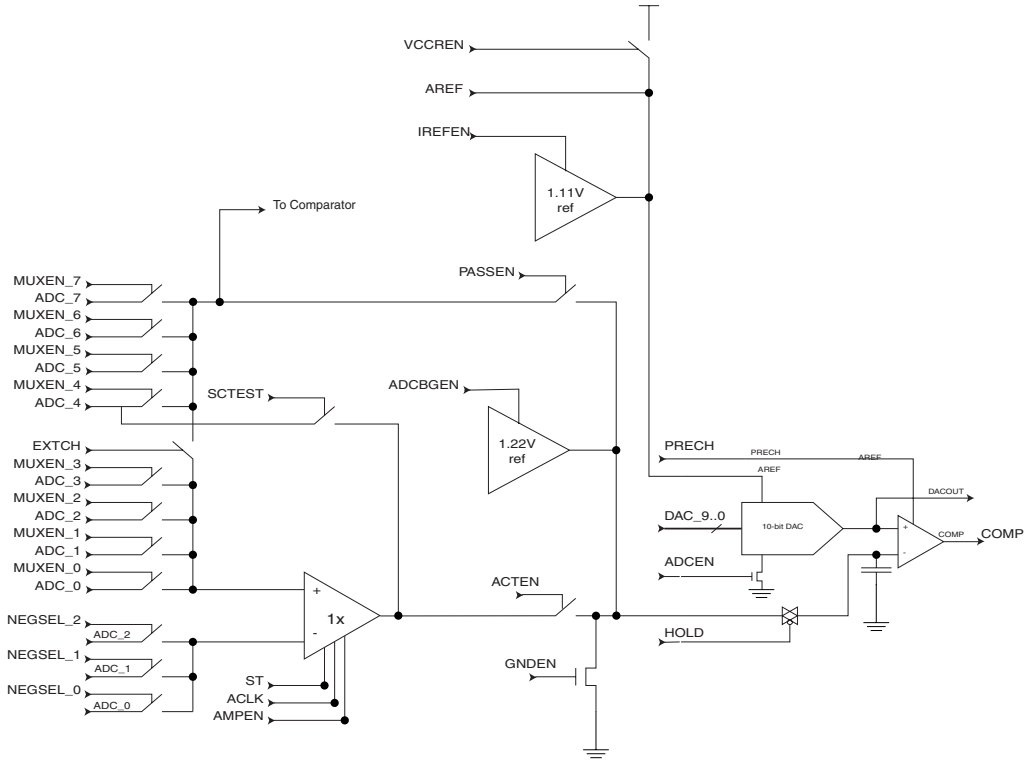
**Table 94.** Boundary-scan Signals for the Analog Comparator

| Signal Name | Direction as Seen from the Comparator | Description | Recommended Input when Not in Use | Output Values when Recommended Inputs are Used |
|---|---|---|---|---|
| AC_IDLE | input | Turns off Analog Comparator when true | 1 | Depends upon µC code being executed |
| ACO | output | Analog Comparator Output | Will become input to µC code being executed | 0 |
| ACME | input | Uses output signal from ADC mux when true | 0 | Depends upon µC code being executed |
| ACBG | input | Bandgap Reference enable | 0 | Depends upon µC code being executed |

## Scanning the ADC

Figure 107 shows a block diagram of the ADC with all relevant control and observe signals. The Boundary-scan cell from Figure 103 is attached to each of these signals. The ADC need not be used for pure connectivity testing, since all analog inputs are shared with a digital port pin as well.

**Figure 107.** Analog to Digital Converter




The signals are described briefly in Table 95.

**Table 95.** Boundary-scan Signals for the ADC[1]

| Signal Name | Direction as seen from the ADC | Description | Recommended Input when not in use | Output Values when recommended inputs are used, and CPU is not using the ADC |
|---|---|---|---|---|
| COMP | Output | Comparator Output | 0 | 0 |
| ACLK | Input | Clock signal to differential amplifier implemented as Switch-cap filters | 0 | 0 |
| ACTEN | Input | Enable path from differential amplifier to the comparator | 0 | 0 |
| ADCBGEN | Input | Enable Band-gap reference as negative input to comparator | 0 | 0 |
| ADCEN | Input | Power-on signal to the ADC | 0 | 0 |
| AMPEN | Input | Power-on signal to the differential amplifier | 0 | 0 |
| DAC_9 | Input | Bit 9 of digital value to DAC | 1 | 1 |
| DAC_8 | Input | Bit 8 of digital value to DAC | 0 | 0 |
| DAC_7 | Input | Bit 7 of digital value to DAC | 0 | 0 |
| DAC_6 | Input | Bit 6 of digital value to DAC | 0 | 0 |
| DAC_5 | Input | Bit 5 of digital value to DAC | 0 | 0 |
| DAC_4 | Input | Bit 4 of digital value to DAC | 0 | 0 |
| DAC_3 | Input | Bit 3 of digital value to DAC | 0 | 0 |
| DAC_2 | Input | Bit 2 of digital value to DAC | 0 | 0 |
| DAC_1 | Input | Bit 1 of digital value to DAC | 0 | 0 |
| DAC_0 | Input | Bit 0 of digital value to DAC | 0 | 0 |
| EXTCH | Input | Connect ADC channels 0 - 3 to by-pass path around differential amplifier | 1 | 1 |
| GNDEN | Input | Ground the negative input to comparator when true | 0 | 0 |
| HOLD | Input | Sample & Hold signal. Sample analog signal when low. Hold signal when high. If differential amplifier are used, this signal must go active when ACLK is high. | 1 | 1 |
| IREFEN | Input | Enables Band-gap reference as AREF signal to DAC | 0 | 0 |
| MUXEN_7 | Input | Input Mux bit 7 | 0 | 0 |
| MUXEN_6 | Input | Input Mux bit 6 | 0 | 0 |
| MUXEN_5 | Input | Input Mux bit 5 | 0 | 0 |
| MUXEN_4 | Input | Input Mux bit 4 | 0 | 0 |
| MUXEN_3 | Input | Input Mux bit 3 | 0 | 0 |
| MUXEN_2 | Input | Input Mux bit 2 | 0 | 0 |
| MUXEN_1 | Input | Input Mux bit 1 | 0 | 0 |
| MUXEN_0 | Input | Input Mux bit 0 | 1 | 1 |

**Table 95.** Boundary-scan Signals for the ADC[1] (Continued)

| Signal Name | Direction as seen from the ADC | Description | Recommended Input when not in use | Output Values when recommended inputs are used, and CPU is not using the ADC |
|---|---|---|---|---|
| NEGSEL_2 | Input | Input Mux for negative input for differential signal, bit 2 | 0 | 0 |
| NEGSEL_1 | Input | Input Mux for negative input for differential signal, bit 1 | 0 | 0 |
| NEGSEL_0 | Input | Input Mux for negative input for differential signal, bit 0 | 0 | 0 |
| PASSEN | Input | Enable pass-gate of differential amplifier. | 1 | 1 |
| PRECH | Input | Precharge output latch of comparator. (Active low) | 1 | 1 |
| SCTEST | Input | Switch-cap TEST enable. Output from differential amplifier send out to Port Pin having ADC_4 | 0 | 0 |
| ST | Input | Output of differential amplifier will settle faster if this signal is high first two ACLK periods after AMPEN goes high. | 0 | 0 |
| VCCREN | Input | Selects Vcc as the ACC reference voltage. | 0 | 0 |

Note: 1. Incorrect setting of the switches in Figure 107 will make signal contention and may damage the part. There are several input choices to the S&H circuitry on the negative input of the output comparator in Figure 107. Make sure only one path is selected from either one ADC pin, Bandgap reference source, or Ground.

If the ADC is not to be used during scan, the recommended input values from Table 95 should be used. The user is recommended **not** to use the differential amplifier during scan. Switch-Cap based differential amplifier require fast operation and accurate timing which is difficult to obtain when used in a scan chain. Details concerning operations of the differential amplifier is therefore not provided.

The AVR ADC is based on the analog circuitry shown in Figure 107 with a successive approximation algorithm implemented in the digital logic. When used in Boundary-scan, the problem is usually to ensure that an applied analog voltage is measured within some limits. This can easily be done without running a successive approximation algorithm: apply the lower limit on the digital DAC[9:0] lines, make sure the output from the comparator is low, then apply the upper limit on the digital DAC[9:0] lines, and verify the output from the comparator to be high.

The ADC need not be used for pure connectivity testing, since all analog inputs are shared with a digital port pin as well.

When using the ADC, remember the following

- The port pin for the ADC channel in use must be configured to be an input with pull-up disabled to avoid signal contention.
- In Normal mode, a dummy conversion (consisting of 10 comparisons) is performed when enabling the ADC. The user is advised to wait at least 200ns after enabling the ADC before controlling/observing any ADC signal, or perform a dummy conversion before using the first result.
- The DAC values must be stable at the midpoint value 0x200 when having the HOLD signal low (Sample mode).

As an example, consider the task of verifying a 1.5V ± 5% input signal at ADC channel 3 when the power supply is 5.0V and AREF is externally connected to $V_{CC}$.

The lower limit is: $\lceil 1024 \cdot 1.5V \cdot 0{,}95/5V \rceil = 291 = 0x123$
The upper limit is: $\lceil 1024 \cdot 1.5V \cdot 1.05/5V \rceil = 323 = 0x143$

The recommended values from Table 95 are used unless other values are given in the algorithm in Table 96. Only the DAC and port pin values of the Scan Chain are shown. The column “Actions” describes what JTAG instruction to be used before filling the Boundary-scan Register with the succeeding columns. The verification should be done on the data scanned out when scanning in the data on the same row in the table.

**Table 96.** Algorithm for Using the ADC

| Step | Actions | ADCEN | DAC | MUXEN | HOLD | PRECH | PA3. Data | PA3. Control | PA3. Pull-up_ Enable |
|---|---|---|---|---|---|---|---|---|---|
| 1 | SAMPLE_ PRELOAD | 1 | 0x200 | 0x08 | 1 | 1 | 0 | 0 | 0 |
| 2 | EXTEST | 1 | 0x200 | 0x08 | 0 | 1 | 0 | 0 | 0 |
| 3 | | 1 | 0x200 | 0x08 | 1 | 1 | 0 | 0 | 0 |
| 4 | | 1 | 0x123 | 0x08 | 1 | 1 | 0 | 0 | 0 |
| 5 | | 1 | 0x123 | 0x08 | 1 | 0 | 0 | 0 | 0 |
| 6 | Verify the COMP bit scanned out to be 0 | 1 | 0x200 | 0x08 | 1 | 1 | 0 | 0 | 0 |
| 7 | | 1 | 0x200 | 0x08 | 0 | 1 | 0 | 0 | 0 |
| 8 | | 1 | 0x200 | 0x08 | 1 | 1 | 0 | 0 | 0 |
| 9 | | 1 | 0x143 | 0x08 | 1 | 1 | 0 | 0 | 0 |
| 10 | | 1 | 0x143 | 0x08 | 1 | 0 | 0 | 0 | 0 |
| 11 | Verify the COMP bit scanned out to be 1 | 1 | 0x200 | 0x08 | 1 | 1 | 0 | 0 | 0 |

Using this algorithm, the timing constraint on the HOLD signal constrains the TCK clock frequency. As the algorithm keeps HOLD high for five steps, the TCK clock frequency has to be at least five times the number of scan bits divided by the maximum hold time, $t_{hold,max}$

## ATmega325/3250/645/6450 Boundary-scan Order

Table 97 and Table 98 shows the Scan order between TDI and TDO when the Boundary-scan chain is selected as data path. Bit 0 is the LSB; the first bit scanned in, and the first bit scanned out. The scan order follows the pin-out order as far as possible. Therefore, the bits of Port A is scanned in the opposite bit order of the other ports. Exceptions from the rules are the Scan chains for the analog circuits, which constitute the most significant bits of the scan chain regardless of which physical pin they are connected to. In Figure 101, PXn. Data corresponds to FF0, PXn. Control corresponds to FF1, and PXn. Pull-up_enable corresponds to FF2. Bit 4, 5, 6 and 7 of Port F is not in the scan chain, since these pins constitute the TAP pins when the JTAG is enabled.

**Table 97.** ATmega325/645 Boundary-scan Order, 64-pin

| Bit Number | Signal Name | Module |
|---|---|---|
| 197 | AC_IDLE | Comparator |
| 196 | ACO | |
| 195 | ACME | |
| 194 | AINBG | |
| 193 | COMP | ADC |
| 192 | ACLK | |
| 191 | ACTEN | |
| 190 | PRIVATE_SIGNAL1[1] | |
| 189 | ADCBGEN | |
| 188 | ADCEN | |
| 187 | AMPEN | |
| 186 | DAC_9 | |
| 185 | DAC_8 | |
| 184 | DAC_7 | |
| 183 | DAC_6 | |
| 182 | DAC_5 | |
| 181 | DAC_4 | |
| 180 | DAC_3 | |
| 179 | DAC_2 | |
| 178 | DAC_1 | |
| 177 | DAC_0 | |
| 176 | EXTCH | |
| 175 | GNDEN | |
| 174 | HOLD | |
| 173 | IREFEN | |
| 172 | MUXEN_7 | |
| 171 | MUXEN_6 | |
| 170 | MUXEN_5 | |

**Table 97.** ATmega325/645 Boundary-scan Order, 64-pin (Continued)

| Bit Number | Signal Name | Module |
|---|---|---|
| 169 | MUXEN_4 | |
| 168 | MUXEN_3 | |
| 167 | MUXEN_2 | |
| 166 | MUXEN_1 | |
| 165 | MUXEN_0 | |
| 164 | NEGSEL_2 | |
| 163 | NEGSEL_1 | |
| 162 | NEGSEL_0 | |
| 161 | PASSEN | |
| 160 | PRECH | |
| 159 | ST | |
| 158 | VCCREN | |
| 157 | PE0.Data | Port E |
| 156 | PE0.Control | |
| 155 | PE0.Pull-up_Enable | |
| 154 | PE1.Data | |
| 153 | PE1.Control | |
| 152 | PE1.Pull-up_Enable | |
| 151 | PE2.Data | |
| 150 | PE2.Control | |
| 149 | PE2.Pull-up_Enable | |
| 148 | PE3.Data | |
| 147 | PE3.Control | |
| 146 | PE3.Pull-up_Enable | |
| 145 | PE4.Data | |
| 144 | PE4.Control | |
| 143 | PE4.Pull-up_Enable | |
| 142 | PE5.Data | |
| 141 | PE5.Control | |
| 140 | PE5.Pull-up_Enable | |
| 139 | PE6.Data | |
| 138 | PE6.Control | |
| 137 | PE6.Pull-up_Enable | |
| 136 | PE7.Data | |
| 135 | PE7.Control | |
| 134 | PE7.Pull-up_Enable | |

**Table 97.** ATmega325/645 Boundary-scan Order, 64-pin (Continued)

| Bit Number | Signal Name | Module |
|---|---|---|
| 133 | PB0.Data | Port B |
| 132 | PB0.Control | |
| 131 | PB0.Pull-up_Enable | |
| 130 | PB1.Data | |
| 129 | PB1.Control | |
| 128 | PB1.Pull-up_Enable | |
| 127 | PB2.Data | |
| 126 | PB2.Control | |
| 125 | PB2.Pull-up_Enable | |
| 124 | PB3.Data | |
| 123 | PB3.Control | |
| 122 | PB3.Pull-up_Enable | |
| 121 | PB4.Data | |
| 120 | PB4.Control | |
| 119 | PB4.Pull-up_Enable | |
| 118 | PB5.Data | |
| 117 | PB5.Control | |
| 116 | PB5.Pull-up_Enable | |
| 115 | PB6.Data | |
| 114 | PB6.Control | |
| 113 | PB6.Pull-up_Enable | |
| 112 | PB7.Data | |
| 111 | PB7.Control | |
| 110 | PB7.Pull-up_Enable | |
| 109 | PG3.Data | Port G |
| 108 | PG3.Control | |
| 107 | PG3.Pull-up_Enable | |
| 106 | PG4.Data | |
| 105 | PG4.Control | |
| 104 | PG4.Pull-up_Enable | |
| 103 | PG5 | (Observe Only) |
| 102 | RSTT | Reset Logic (Observe-only) |
| 101 | RSTHV | |

**Table 97.** ATmega325/645 Boundary-scan Order, 64-pin (Continued)

| Bit Number | Signal Name | Module |
|---|---|---|
| 100 | EXTCLKEN | Enable signals for main Clock/Oscillators |
| 99 | OSCON | |
| 98 | RCOSCEN | |
| 97 | OSC32EN | |
| 96 | EXTCLK (XTAL1) | Clock input and Oscillators for the main clock (Observe-only) |
| 95 | OSCCK | |
| 94 | RCCK | |
| 93 | OSC32CK | |
| 92 | PD0.Data | Port D |
| 91 | PD0.Control | |
| 90 | PD0.Pull-up_Enable | |
| 89 | PD1.Data | |
| 88 | PD1.Control | |
| 87 | PD1.Pull-up_Enable | |
| 86 | PD2.Data | |
| 85 | PD2.Control | |
| 84 | PD2.Pull-up_Enable | |
| 83 | PD3.Data | |
| 82 | PD3.Control | |
| 81 | PD3.Pull-up_Enable | |
| 80 | PD4.Data | |
| 79 | PD4.Control | |
| 78 | PD4.Pull-up_Enable | |
| 77 | PD5.Data | |
| 76 | PD5.Control | |
| 75 | PD5.Pull-up_Enable | |
| 74 | PD6.Data | |
| 73 | PD6.Control | |
| 72 | PD6.Pull-up_Enable | |
| 71 | PD7.Data | |
| 70 | PD7.Control | |
| 69 | PD7.Pull-up_Enable | |
| 68 | PG0.Data | Port G |
| 67 | PG0.Control | |
| 66 | PG0.Pull-up_Enable | |
| 65 | PG1.Data | |

**Table 97.** ATmega325/645 Boundary-scan Order, 64-pin (Continued)

| Bit Number | Signal Name | Module |
|---|---|---|
| 64 | PG1.Control | |
| 63 | PG1.Pull-up_Enable | |
| 62 | PC0.Data | Port C |
| 61 | PC0.Control | |
| 60 | PC0.Pull-up_Enable | |
| 59 | PC1.Data | |
| 58 | PC1.Control | |
| 57 | PC1.Pull-up_Enable | |
| 56 | PC2.Data | |
| 55 | PC2.Control | |
| 54 | PC2.Pull-up_Enable | |
| 53 | PC3.Data | |
| 52 | PC3.Control | |
| 51 | PC3.Pull-up_Enable | |
| 50 | PC4.Data | |
| 49 | PC4.Control | |
| 48 | PC4.Pull-up_Enable | |
| 47 | PC5.Data | |
| 46 | PC5.Control | |
| 45 | PC5.Pull-up_Enable | |
| 44 | PC6.Data | |
| 43 | PC6.Control | |
| 42 | PC6.Pull-up_Enable | |
| 41 | PC7.Data | |
| 40 | PC7.Control | |
| 39 | PC7.Pull-up_Enable | |
| 38 | PG2.Data | Port G |
| 37 | PG2.Control | |
| 36 | PG2.Pull-up_Enable | |
| 35 | PA7.Data | Port A |
| 34 | PA7.Control | |
| 33 | PA7.Pull-up_Enable | |
| 32 | PA6.Data | |
| 31 | PA6.Control | |
| 30 | PA6.Pull-up_Enable | |
| 29 | PA5.Data | |

**Table 97.** ATmega325/645 Boundary-scan Order, 64-pin (Continued)

| Bit Number | Signal Name | Module |
|---|---|---|
| 28 | PA5.Control | |
| 27 | PA5.Pull-up_Enable | |
| 26 | PA4.Data | |
| 25 | PA4.Control | |
| 24 | PA4.Pull-up_Enable | |
| 23 | PA3.Data | |
| 22 | PA3.Control | |
| 21 | PA3.Pull-up_Enable | |
| 20 | PA2.Data | |
| 19 | PA2.Control | |
| 18 | PA2.Pull-up_Enable | |
| 17 | PA1.Data | |
| 16 | PA1.Control | |
| 15 | PA1.Pull-up_Enable | |
| 14 | PA0.Data | |
| 13 | PA0.Control | |
| 12 | PA0.Pull-up_Enable | |
| 11 | PF3.Data | Port F |
| 10 | PF3.Control | |
| 9 | PF3.Pull-up_Enable | |
| 8 | PF2.Data | |
| 7 | PF2.Control | |
| 6 | PF2.Pull-up_Enable | |
| 5 | PF1.Data | |
| 4 | PF1.Control | |
| 3 | PF1.Pull-up_Enable | |
| 2 | PF0.Data | |
| 1 | PF0.Control | |
| 0 | PF0.Pull-up_Enable | |

Note: 1. PRIVATE_SIGNAL1 should always be scanned in as zero.

**Table 98.** ATmega3250/6450 Boundary-scan Order, 100-pin

| Bit Number | Signal Name | Module |
|---|---|---|
| **242** | AC_IDLE | Comparator |
| 241 | ACO | |
| 240 | ACME | |
| 239 | AINBG | |
| 238 | COMP | ADC |
| 237 | ACLK | |
| 236 | ACTEN | |
| 235 | PRIVATE_SIGNAL1[1] | |
| 234 | ADCBGEN | |
| 233 | ADCEN | |
| 232 | AMPEN | |
| 231 | DAC_9 | |
| 230 | DAC_8 | |
| 229 | DAC_7 | |
| 228 | DAC_6 | |
| 227 | DAC_5 | |
| 226 | DAC_4 | |
| 225 | DAC_3 | |
| 224 | DAC_2 | |
| 223 | DAC_1 | |
| 222 | DAC_0 | |
| 221 | EXTCH | |
| 220 | GNDEN | |
| 219 | HOLD | |
| 218 | IREFEN | |
| 217 | MUXEN_7 | |
| 216 | MUXEN_6 | |
| 215 | MUXEN_5 | |
| 214 | MUXEN_4 | |
| 213 | MUXEN_3 | |
| 212 | MUXEN_2 | |
| 211 | MUXEN_1 | |
| 210 | MUXEN_0 | |
| 209 | NEGSEL_2 | |
| 208 | NEGSEL_1 | |

**Table 98.** ATmega3250/6450 Boundary-scan Order, 100-pin (Continued)

| Bit Number | Signal Name | Module |
|---|---|---|
| 207 | NEGSEL_0 | |
| 206 | PASSEN | |
| 205 | PRECH | |
| 204 | ST | |
| 203 | VCCREN | |
| 202 | PE0.Data | Port E |
| 201 | PE0.Control | |
| 200 | PE0.Pull-up_Enable | |
| 199 | PE1.Data | |
| 198 | PE1.Control | |
| 197 | PE1.Pull-up_Enable | |
| 196 | PE2.Data | |
| 195 | PE2.Control | |
| 194 | PE2.Pull-up_Enable | |
| 193 | PE3.Data | |
| 192 | PE3.Control | |
| 191 | PE3.Pull-up_Enable | |
| 190 | PE4.Data | |
| 189 | PE4.Control | |
| 188 | PE4.Pull-up_Enable | |
| 187 | PE5.Data | |
| 186 | PE5.Control | |
| 185 | PE5.Pull-up_Enable | |
| 184 | PE6.Data | |
| 183 | PE6.Control | |
| 182 | PE6.Pull-up_Enable | |
| 181 | PE7.Data | |
| 180 | PE7.Control | |
| 179 | PE7.Pull-up_Enable | |
| 178 | PJ0.Data | Port J |
| 177 | PJ0.Control | |
| 176 | PJ0.Pull-up_Enable | |
| 175 | PJ1.Data | |
| 174 | PJ1.Control | |
| 173 | PJ1.Pull-up_Enable | |
| 172 | PB0.Data | Port B |

**Table 98.** ATmega3250/6450 Boundary-scan Order, 100-pin (Continued)

| Bit Number | Signal Name | Module |
|---|---|---|
| 171 | PB0.Control | |
| 170 | PB0.Pull-up_Enable | |
| 169 | PB1.Data | |
| 168 | PB1.Control | |
| 167 | PB1.Pull-up_Enable | |
| 166 | PB2.Data | |
| 165 | PB2.Control | |
| 164 | PB2.Pull-up_Enable | |
| 163 | PB3.Data | |
| 162 | PB3.Control | |
| 161 | PB3.Pull-up_Enable | |
| 160 | PB4.Data | |
| 159 | PB4.Control | |
| 158 | PB4.Pull-up_Enable | |
| 157 | PB5.Data | |
| 156 | PB5.Control | |
| 155 | PB5.Pull-up_Enable | |
| 154 | PB6.Data | |
| 153 | PB6.Control | |
| 152 | PB6.Pull-up_Enable | |
| 151 | PB7.Data | |
| 150 | PB7.Control | |
| 149 | PB7.Pull-up_Enable | |
| 148 | PG3.Data | Port G |
| 147 | PG3.Control | |
| 146 | PG3.Pull-up_Enable | |
| 145 | PG4.Data | |
| 144 | PG4.Control | |
| 143 | PG4.Pull-up_Enable | |
| 142 | PG5 | (Observe Only) |
| 141 | RSTT | Reset Logic (Observe-only) |
| 140 | RSTHV | |
| 139 | EXTCLKEN | Enable signals for main Clock/Oscillators |
| 138 | OSCON | |
| 137 | RCOSCEN | |
| 136 | OSC32EN | |

**Table 98.** ATmega3250/6450 Boundary-scan Order, 100-pin (Continued)

| Bit Number | Signal Name | Module |
|---|---|---|
| 135 | EXTCLK (XTAL1) | Clock input and Oscillators for the main clock (Observe-only) |
| 134 | OSCCK | |
| 133 | RCCK | |
| 132 | OSC32CK | |
| 131 | PJ2.Data | Port J |
| 130 | PJ2.Control | |
| 129 | PJ2.Pull-up_Enable | |
| 128 | PJ3.Data | |
| 127 | PJ3.Control | |
| 126 | PJ3.Pull-up_Enable | |
| 125 | PJ4.Data | |
| 124 | PJ4.Control | |
| 123 | PJ4.Pull-up_Enable | |
| 122 | PJ5.Data | |
| 121 | PJ5.Control | |
| 120 | PJ5.Pull-up_Enable | |
| 119 | PJ6.Data | |
| 118 | PJ6.Control | |
| 117 | PJ6.Pull-up_Enable | |
| 116 | PD0.Data | Port D |
| 115 | PD0.Control | |
| 114 | PD0.Pull-up_Enable | |
| 113 | PD1.Data | |
| 112 | PD1.Control | |
| 111 | PD1.Pull-up_Enable | |
| 110 | PD2.Data | |
| 109 | PD2.Control | |
| 108 | PD2.Pull-up_Enable | |
| 107 | PD3.Data | |
| 106 | PD3.Control | |
| 105 | PD3.Pull-up_Enable | |
| 104 | PD4.Data | |
| 103 | PD4.Control | |
| 102 | PD4.Pull-up_Enable | |
| 101 | PD5.Data | |
| 100 | PD5.Control | |

**Table 98.** ATmega3250/6450 Boundary-scan Order, 100-pin (Continued)

| Bit Number | Signal Name | Module |
|---|---|---|
| 99 | PD5.Pull-up_Enable | |
| 98 | PD6.Data | |
| 97 | PD6.Control | |
| 96 | PD6.Pull-up_Enable | |
| 95 | PD7.Data | |
| 94 | PD7.Control | |
| 93 | PD7.Pull-up_Enable | |
| 92 | PG0.Data | Port G |
| 91 | PG0.Control | |
| 90 | PG0.Pull-up_Enable | |
| 89 | PG1.Data | |
| 88 | PG1.Control | |
| 87 | PG1.Pull-up_Enable | |
| 86 | PC0.Data | Port C |
| 85 | PC0.Control | |
| 84 | PC0.Pull-up_Enable | |
| 83 | PC1.Data | |
| 82 | PC1.Control | |
| 81 | PC1.Pull-up_Enable | |
| 80 | PC2.Data | |
| 79 | PC2.Control | |
| 78 | PC2.Pull-up_Enable | |
| 77 | PC3.Data | |
| 76 | PC3.Control | |
| 75 | PC3.Pull-up_Enable | |
| 74 | PC4.Data | |
| 73 | PC4.Control | |
| 72 | PC4.Pull-up_Enable | |
| 71 | PC5.Data | |
| 70 | PC5.Control | |
| 69 | PC5.Pull-up_Enable | |
| 68 | PH0.Data | Port H |
| 67 | PH0.Control | |
| 66 | PH0.Pull-up_Enable | |
| 65 | PH1.Data | |
| 64 | PH1.Control | |

**Table 98.** ATmega3250/6450 Boundary-scan Order, 100-pin (Continued)

| Bit Number | Signal Name | Module |
|---|---|---|
| 63 | PH1.Pull-up_Enable | |
| 62 | PH2.Data | |
| 61 | PH2.Control | |
| 60 | PH2.Pull-up_Enable | |
| 59 | PH3.Data | |
| 58 | PH3.Control | |
| 57 | PH3.Pull-up_Enable | |
| 56 | PC6.Data | Port C |
| 55 | PC6.Control | |
| 54 | PC6.Pull-up_Enable | |
| 53 | PC7.Data | |
| 52 | PC7.Control | |
| 51 | PC7.Pull-up_Enable | |
| 50 | PG2.Data | Port G |
| 49 | PG2.Control | |
| 48 | PG2.Pull-up_Enable | |
| 47 | PA7.Data | Port A |
| 46 | PA7.Control | |
| 45 | PA7.Pull-up_Enable | |
| 44 | PA6.Data | |
| 43 | PA6.Control | |
| 42 | PA6.Pull-up_Enable | |
| 41 | PA5.Data | |
| 40 | PA5.Control | |
| 39 | PA5.Pull-up_Enable | |
| 38 | PA4.Data | |
| 37 | PA4.Control | |
| 36 | PA4.Pull-up_Enable | |
| 35 | PA3.Data | |
| 34 | PA3.Control | |
| 33 | PA3.Pull-up_Enable | |
| 32 | PA2.Data | |
| 31 | PA2.Control | |
| 30 | PA2.Pull-up_Enable | |
| 29 | PA1.Data | |
| 28 | PA1.Control | |

**Table 98.** ATmega3250/6450 Boundary-scan Order, 100-pin (Continued)

| Bit Number | Signal Name | Module |
|---|---|---|
| 27 | PA1.Pull-up_Enable | |
| 26 | PA0.Data | |
| 25 | PA0.Control | |
| 24 | PA0.Pull-up_Enable | |
| 23 | PH4.Data | Port H |
| 22 | PH4.Control | |
| 21 | PH4.Pull-up_Enable | |
| 20 | PH5.Data | |
| 19 | PH5.Control | |
| 18 | PH5.Pull-up_Enable | |
| 17 | PH6.Data | |
| 16 | PH6.Control | |
| 15 | PH6.Pull-up_Enable | |
| 14 | PH7.Data | |
| 13 | PH7.Control | |
| 12 | PH7.Pull-up_Enable | |
| 11 | PF3.Data | Port F |
| 10 | PF3.Control | |
| 9 | PF3.Pull-up_Enable | |
| 8 | PF2.Data | |
| 7 | PF2.Control | |
| 6 | PF2.Pull-up_Enable | |
| 5 | PF1.Data | |
| 4 | PF1.Control | |
| 3 | PF1.Pull-up_Enable | |
| 2 | PF0.Data | |
| 1 | PF0.Control | |
| 0 | PF0.Pull-up_Enable | |

Note: 1. PRIVATE_SIGNAL1 should always be scanned in as zero.

## Boundary-scan Description Language Files

Boundary-scan Description Language (BSDL) files describe Boundary-scan capable devices in a standard format used by automated test-generation software. The order and function of bits in the Boundary-scan Data Register are included in this description. A BSDL file for ATmega325/3250/645/6450 is available.

## Boot Loader Support – Read-While-Write Self-Programming

The Boot Loader Support provides a real Read-While-Write Self-Programming mechanism for downloading and uploading program code by the MCU itself. This feature allows flexible application software updates controlled by the MCU using a Flash-resident Boot Loader program. The Boot Loader program can use any available data interface and associated protocol to read code and write (program) that code into the Flash memory, or read the code from the program memory. The program code within the Boot Loader section has the capability to write into the entire Flash, including the Boot Loader memory. The Boot Loader can thus even modify itself, and it can also erase itself from the code if the feature is not needed anymore. The size of the Boot Loader memory is configurable with fuses and the Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

## Boot Loader Features

- **Read-While-Write Self-Programming**
- **Flexible Boot Memory Size**
- **High Security (Separate Boot Lock Bits for a Flexible Protection)**
- **Separate Fuse to Select Reset Vector**
- **Optimized Page[1] Size**
- **Code Efficient Algorithm**
- **Efficient Read-Modify-Write Support**

Note: 1. A page is a section in the Flash consisting of several bytes (see Table 116 on page 262) used during programming. The page organization does not affect normal operation.

## Application and Boot Loader Flash Sections

The Flash memory is organized in two main sections, the Application section and the Boot Loader section (see Figure 109). The size of the different sections is configured by the BOOTSZ Fuses as shown in Table 104 on page 255 and Figure 109. These two sections can have different level of protection since they have different sets of Lock bits.

### Application Section

The Application section is the section of the Flash that is used for storing the application code. The protection level for the Application section can be selected by the application Boot Lock bits (Boot Lock bits 0), see Table 100 on page 246. The Application section can never store any Boot Loader code since the SPM instruction is disabled when executed from the Application section.

### BLS – Boot Loader Section

While the Application section is used for storing the application code, the The Boot Loader software must be located in the BLS since the SPM instruction can initiate a programming when executing from the BLS only. The SPM instruction can access the entire Flash, including the BLS itself. The protection level for the Boot Loader section can be selected by the Boot Loader Lock bits (Boot Lock bits 1), see Table 101 on page 246.

## Read-While-Write and No Read-While-Write Flash Sections

Whether the CPU supports Read-While-Write or if the CPU is halted during a Boot Loader software update is dependent on which address that is being programmed. In addition to the two sections that are configurable by the BOOTSZ Fuses as described above, the Flash is also divided into two fixed sections, the Read-While-Write (RWW) section and the No Read-While-Write (NRWW) section. The limit between the RWW- and NRWW sections is given in Table 105 on page 255 and Figure 109 on page 245. The main difference between the two sections is:

- When erasing or writing a page located inside the RWW section, the NRWW section can be read during the operation.
- When erasing or writing a page located inside the NRWW section, the CPU is halted during the entire operation.

Note that the user software can never read any code that is located inside the RWW section during a Boot Loader software operation. The syntax “Read-While-Write section” refers to which section that is being programmed (erased or written), not which section that actually is being read during a Boot Loader software update.

**RWW – Read-While-Write Section**

If a Boot Loader software update is programming a page inside the RWW section, it is possible to read code from the Flash, but only code that is located in the NRWW section. During an on-going programming, the software must ensure that the RWW section never is being read. If the user software is trying to read code that is located inside the RWW section (i.e., by a call/jmp/lpm or an interrupt) during programming, the software might end up in an unknown state. To avoid this, the interrupts should either be disabled or moved to the Boot Loader section. The Boot Loader section is always located in the NRWW section. The RWW Section Busy bit (RWWSB) in the Store Program Memory Control and Status Register (SPMCSR) will be read as logical one as long as the RWW section is blocked for reading. After a programming is completed, the RWWSB must be cleared by software before reading code located in the RWW section. See “Store Program Memory Control and Status Register – SPMCSR” on page 247. for details on how to clear RWWSB.

**NRWW – No Read-While-Write Section**

The code located in the NRWW section can be read when the Boot Loader software is updating a page in the RWW section. When the Boot Loader code updates the NRWW section, the CPU is halted during the entire Page Erase or Page Write operation.

**Table 99.** Read-While-Write Features

| Which Section does the Z-pointer Address During the Programming? | Which Section Can be Read During Programming? | Is the CPU Halted? | Read-While-Write Supported? |
|---|---|---|---|
| RWW Section | NRWW Section | No | Yes |
| NRWW Section | None | Yes | No |

**Figure 108.** Read-While-Write vs. No Read-While-Write

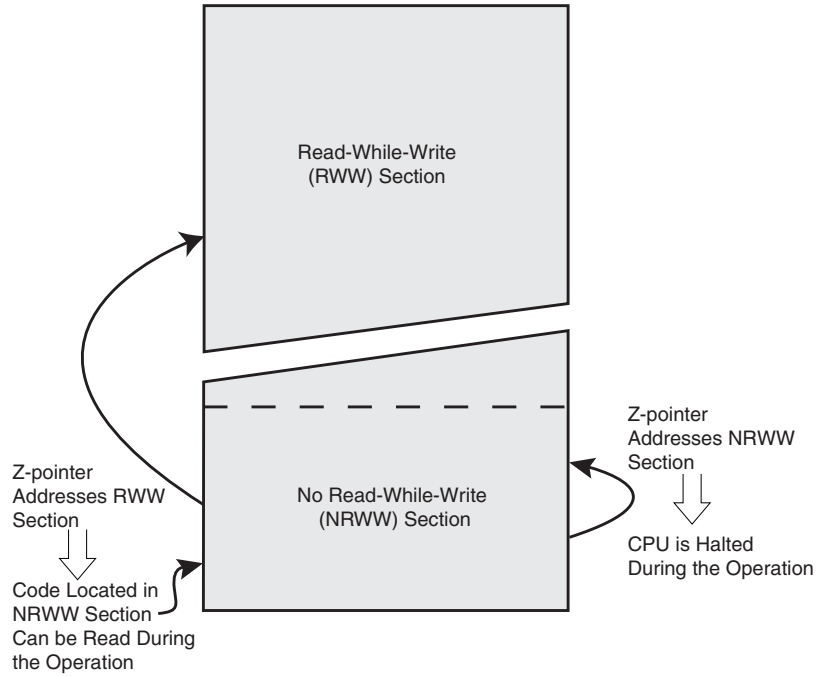
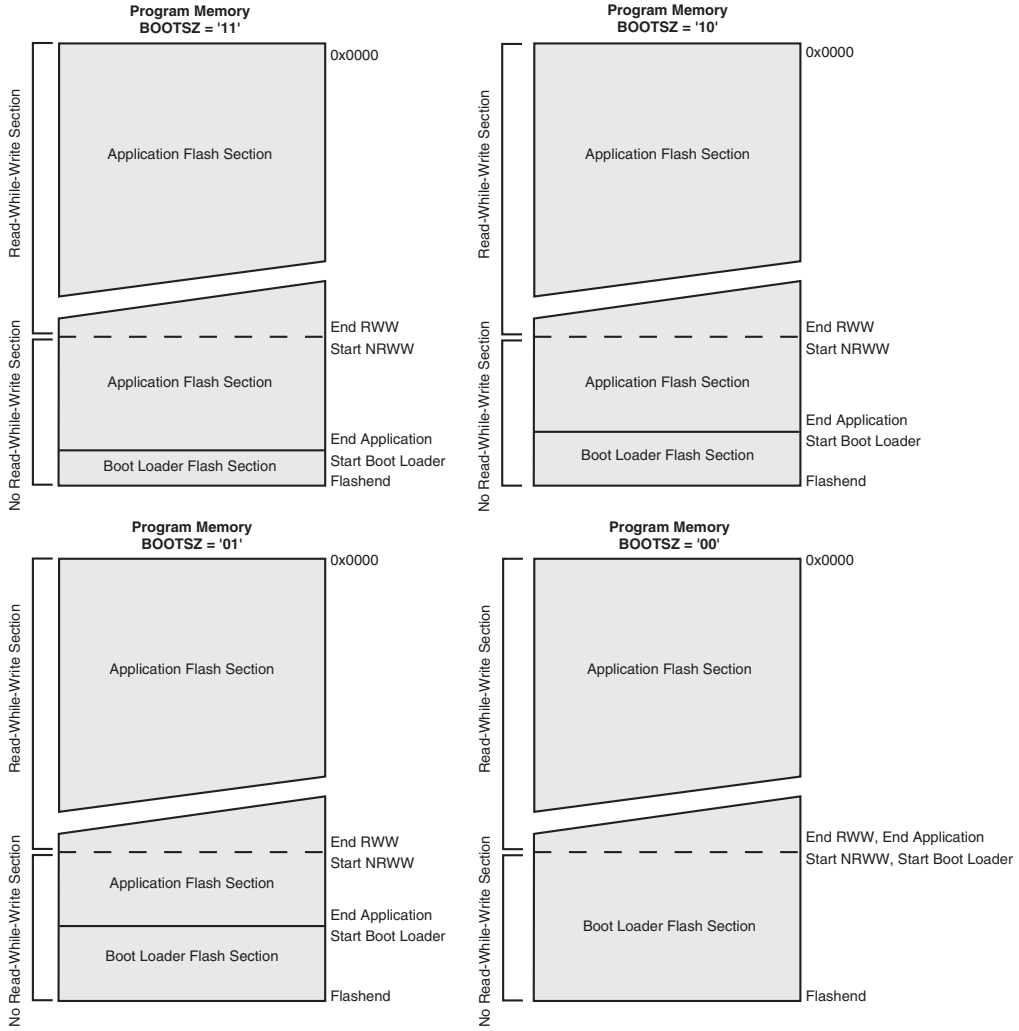


**Figure 109.** Memory Sections




Note: 1. The parameters in the figure above are given in Table 104 on page 255.

## Boot Loader Lock Bits

If no Boot Loader capability is needed, the entire Flash is available for application code. The Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

The user can select:

- To protect the entire Flash from a software update by the MCU.
- To protect only the Boot Loader Flash section from a software update by the MCU.
- To protect only the Application Flash section from a software update by the MCU.
- Allow software update in the entire Flash.

See Table 100 and Table 101 for further details. The Boot Lock bits and general Lock bits can be set in software and in Serial or Parallel Programming mode, but they can be cleared by a Chip Erase command only. The general Write Lock (Lock Bit mode 2) does not control the programming of the Flash memory by SPM instruction. Similarly, the general Read/Write Lock (Lock Bit mode 1) does not control reading nor writing by LPM/SPM, if it is attempted.

**Table 100.** Boot Lock Bit0 Protection Modes (Application Section)[1]

| BLB0 Mode | BLB02 | BLB01 | Protection |
|---|---|---|---|
| 1 | 1 | 1 | No restrictions for SPM or LPM accessing the Application section. |
| 2 | 1 | 0 | SPM is not allowed to write to the Application section. |
| 3 | 0 | 0 | SPM is not allowed to write to the Application section, and LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section. |
| 4 | 0 | 1 | LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section. |

Note: 1. “1” means unprogrammed, “0” means programmed

**Table 101.** Boot Lock Bit1 Protection Modes (Boot Loader Section)[1]

| BLB1 Mode | BLB12 | BLB11 | Protection |
|---|---|---|---|
| 1 | 1 | 1 | No restrictions for SPM or LPM accessing the Boot Loader section. |
| 2 | 1 | 0 | SPM is not allowed to write to the Boot Loader section. |
| 3 | 0 | 0 | SPM is not allowed to write to the Boot Loader section, and LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section. |
| 4 | 0 | 1 | LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section. |

Note: 1. “1” means unprogrammed, “0” means programmed

## Entering the Boot Loader Program

Entering the Boot Loader takes place by a jump or call from the application program. This may be initiated by a trigger such as a command received via USART, or SPI interface. Alternatively, the Boot Reset Fuse can be programmed so that the Reset Vector is pointing to the Boot Flash start address after a reset. In this case, the Boot Loader is started after a reset. After the application code is loaded, the program can start executing the application code. Note that the fuses cannot be changed by the MCU itself. This means that once the Boot Reset Fuse is programmed, the Reset Vector will always point to the Boot Loader Reset and the fuse can only be changed through the serial or parallel programming interface.

**Table 102.** Boot Reset Fuse[1]

| BOOTRST | Reset Address |
|---|---|
| 1 | Reset Vector = Application Reset (address 0x0000) |
| 0 | Reset Vector = Boot Loader Reset (see Table 104 on page 255) |

Note: 1. “1” means unprogrammed, “0” means programmed

### Store Program Memory Control and Status Register – SPMCSR

The Store Program Memory Control and Status Register contains the control bits needed to control the Boot Loader operations.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | SPMIE | RWWSB | – | RWWSRE | BLBSET | PGWRT | PGERS | SPMEN | SPMCSR |
| Read/Write | R/W | R | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 7 – SPMIE: SPM Interrupt Enable**

When the SPMIE bit is written to one, and the I-bit in the Status Register is set (one), the SPM ready interrupt will be enabled. The SPM ready Interrupt will be executed as long as the SPMEN bit in the SPMCSR Register is cleared.

- **Bit 6 – RWWSB: Read-While-Write Section Busy**

When a Self-Programming (Page Erase or Page Write) operation to the RWW section is initiated, the RWWSB will be set (one) by hardware. When the RWWSB bit is set, the RWW section cannot be accessed. The RWWSB bit will be cleared if the RWWSRE bit is written to one after a Self-Programming operation is completed. Alternatively the RWWSB bit will automatically be cleared if a page load operation is initiated.

- **Bit 5 – Res: Reserved Bit**

This bit is a reserved bit in the ATmega325/3250/645/6450 and always read as zero.

- **Bit 4 – RWWSRE: Read-While-Write Section Read Enable**

When programming (Page Erase or Page Write) to the RWW section, the RWW section is blocked for reading (the RWWSB will be set by hardware). To re-enable the RWW section, the user software must wait until the programming is completed (SPMEN will be cleared). Then, if the RWWSRE bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles re-enables the RWW section. The RWW section cannot be re-enabled while the Flash is busy with a Page Erase or a Page Write (SPMEN is set). If the RWWSRE bit is written while the Flash is being loaded, the Flash load operation will abort and the data loaded will be lost.

- **Bit 3 – BLBSET: Boot Lock Bit Set**

If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles sets Boot Lock bits and general Lock bits, according to the data in R0. The data in R1 and the address in the Z-pointer are ignored. The BLBSET bit will automatically be cleared upon completion of the Lock bit set, or if no SPM instruction is executed within four clock cycles.

An LPM instruction within three cycles after BLBSET and SPMEN 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 “Reading the Fuse and Lock Bits from Software” on page 251 for details.

- **Bit 2 – PGWRT: Page Write**

If this bit is written to one at the same time as SPMEN, 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 if the NRWW section is addressed.

- **Bit 1 – PGERS: Page Erase**

If this bit is written to one at the same time as SPMEN, 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 if the NRWW section is addressed.

- **Bit 0 – SPMEN: Store Program Memory Enable**

This bit enables the SPM instruction for the next four clock cycles. If written to one together with either RWWSRE, BLBSET, PGWRT' or PGERS, the following SPM instruction will have a special meaning, see description above. If only SPMEN 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 SPMEN 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 SPMEN 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.

## Addressing the Flash During Self-Programming

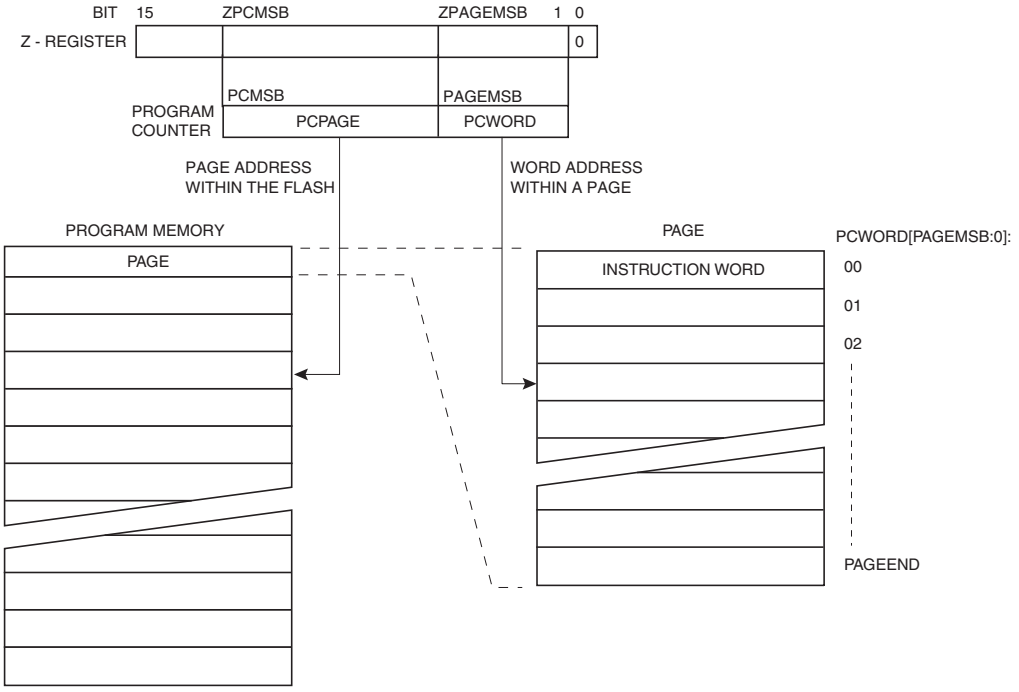
The Z-pointer is used to address the SPM commands.

| 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 116 on page 262), 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 110. Note that the Page Erase and Page Write operations are addressed independently. Therefore it is of major importance that the Boot Loader software addresses the same page in both the Page Erase and Page Write operation. Once a programming operation is initiated, the address is latched and the Z-pointer can be used for other operations.

The only SPM operation that does not use the Z-pointer is Setting the Boot Loader Lock bits. The content of the Z-pointer is ignored and will have no effect on the operation. The LPM instruction does also use 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 110.** Addressing the Flash During SPM[1]




Note: 1. The different variables used in Figure 110 are listed in Table 106 on page 256.
2. PCPAGE and PCWORD are listed in Table 116 on page 262.

## Self-Programming the Flash

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 rewritten. When using alternative 1, the Boot Loader 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. See “Simple Assembly Code Example for a Boot Loader” on page 253 for an assembly code example.

**Performing Page Erase by SPM**

To execute Page Erase, set up the address in the Z-pointer, write “X0000011” 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.

- Page Erase to the RWW section: The NRWW section can be read during the Page Erase.
- Page Erase to the NRWW section: The CPU is halted during the operation.

**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 RWWSRE 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.

**Performing a Page Write**

To execute Page Write, set up the address in the Z-pointer, write “X0000101” 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.

- Page Write to the RWW section: The NRWW section can be read during the Page Write.
- Page Write to the NRWW section: The CPU is halted during the operation.

**Using the SPM Interrupt**

If the SPM interrupt is enabled, the SPM interrupt will generate a constant interrupt when the SPMEN bit in SPMCSR is cleared. This means that the interrupt can be used instead of polling the SPMCSR Register in software. When using the SPM interrupt, the Interrupt Vectors should be moved to the BLS section to avoid that an interrupt is accessing the RWW section when it is blocked for reading. How to move the interrupts is described in “Interrupts” on page 47.

**Consideration While Updating BLS**

Special care must be taken if the user allows the Boot Loader section to be updated by leaving Boot Lock bit11 unprogrammed. An accidental write to the Boot Loader itself can corrupt the entire Boot Loader, and further software updates might be impossible. If it is not necessary to change the Boot Loader software itself, it is recommended to program the Boot Lock bit11 to protect the Boot Loader software from any internal software changes.

**Prevent Reading the RWW Section During Self-Programming**

During Self-Programming (either Page Erase or Page Write), the RWW section is always blocked for reading. The user software itself must prevent that this section is addressed during the self programming operation. The RWWSB in the SPMCSR will be set as long as the RWW section is busy. During Self-Programming the Interrupt Vector table should be moved to the BLS as described in “Interrupts” on page 47, or the interrupts must be disabled. Before addressing the RWW section after the programming is completed, the user software must clear the RWWSB by writing the RWWSRE. See “Simple Assembly Code Example for a Boot Loader” on page 253 for an example.

### Setting the Boot Loader Lock Bits by SPM

To set the Boot Loader Lock bits and general Lock bits, write the desired data to R0, write “X0001001” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| R0 | 1 | 1 | BLB12 | BLB11 | BLB02 | BLB01 | LB2 | LB1 |

See Table 100 and Table 101 for how the different settings of the Boot Loader bits affect the Flash access.

If bits 5..0 in R0 are cleared (zero), the corresponding Lock bit will be programmed if an SPM instruction is executed within four cycles after BLBSET and SPMEN are set in SPMCSR. The Z-pointer is don’t care during this operation, but for future compatibility it is recommended to load the Z-pointer with 0x0001 (same as used for reading the Lock bits). For future compatibility it is also recommended to set bits 7, and 6 in R0 to “1” when writing the Lock bits. When programming the Lock bits the entire Flash can be read during the operation.

### 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 (EEWE) in the EECR Register and verifies that the bit is cleared before writing to the SPMCSR Register.

### 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 BLBSET and SPMEN bits in SPMCSR. When an LPM instruction is executed within three CPU cycles after the BLBSET and SPMEN bits are set in SPMCSR, the value of the Lock bits will be loaded in the destination register. The BLBSET and SPMEN 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 BLBSET and SPMEN are cleared, LPM will work as described in the Instruction set Manual.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Rd | – | – | BLB12 | BLB11 | BLB02 | BLB01 | 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 BLBSET and SPMEN bits in SPMCSR. When an LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in the SPMCSR, the value of the Fuse Low byte (FLB) will be loaded in the destination register as shown below. Refer to Table 111 on page 259 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, load 0x0003 in the Z-pointer. When an LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in the SPMCSR, the value of the Fuse High byte (FHB) will be loaded in the destination register as shown below. Refer to Table 110 on page 259 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 |

When reading the Extended Fuse byte, load 0x0002 in the Z-pointer. When an LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in

the SPMCSR, the value of the Extended Fuse byte (EFB) will be loaded in the destination register as shown below. Refer to Table 109 on page 258 for detailed description and mapping of the Extended Fuse byte.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Rd | – | – | – | – | – | EFB2 | EFB1 | EFB0 |

Fuse and Lock bits that are programmed, will be read as zero. Fuse and Lock bits that are unprogrammed, will be read as one.

### 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. If there is no need for a Boot Loader update in the system, program the Boot Loader Lock bits to prevent any Boot Loader software updates.
2. 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.
3. 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.

### Programming Time for Flash when Using SPM

The calibrated RC Oscillator is used to time Flash accesses. Table 103 shows the typical programming time for Flash accesses from the CPU.

**Table 103.** SPM Programming Time

| Symbol | Min Programming Time | Max Programming Time |
|---|---|---|
| Flash write (Page Erase, Page Write, and write Lock bits by SPM) | 3.7 ms | 4.5 ms |

### Simple Assembly Code Example for a Boot Loader

```
  ;-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<<SPMEN)
  call Do_spm

  ; re-enable the RWW section
  ldi  spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call 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<<SPMEN)
  call 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<<SPMEN)
  call Do_spm

  ; re-enable the RWW section
  ldi  spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call 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
  jmp  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<<RWWSRE) | (1<<SPMEN)
  call Do_spm
  rjmp Return

Do_spm:
  ; check for previous SPM complete
Wait_spm:
  in   temp1, SPMCSR
  sbrc temp1, SPMEN
  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, EEWE
  rjmp Wait_ee
  ; SPM timed sequence
  out  SPMCSR, spmcrval
  spm
  ; restore SREG (to enable interrupts if originally enabled)
  out  SREG, temp2
  ret
```

### ATmega325/3250/645/6450 Boot Loader Parameters

In Table 104 through Table 106, the parameters used in the description of the Self-Programming are given.

**Table 104.** Boot Size Configuration(1)

| BOOTSZ1 | BOOTSZ0 | Boot Size | Pages | Application Flash Section | Boot Loader Flash Section | End Application Section | Boot Reset Address (Start Boot Loader Section) |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 256/ 512 words | 4 | 0x0000 - 0x3EFF/ x0000 - 0x7DFF | 0x3F00- 0x3FFF/ 0x7E00- 0x7FFF | 0x3EFF/ 0x7DFF | 0x3F00/ 0x7E00 |
| 1 | 0 | 512/ 1024 words | 8 | 0x0000 - 0x3DFF/ 0x0000 - 0x7BFF/ | 0x3E00 - 0x3FFF/ 0x7C00- 0x7FFF | 0x3DFF/ 0x7BFF | 0x3E00/ 0x7C00 |
| 0 | 1 | 1024/ 2048 words | 16 | 0x0000 - 0x3BFF/ 0x0000 - 0x77FF | 0x3C00- 0x3FFF/ 0x7800- 0x7FFF | 0x3BFF/ 0x77FF | 0x3C00 0x7800 |
| 0 | 0 | 2048/ 4096 words | 32 | 0x0000 - 0x37FF/ 0x0000 - 0x6FFF | 0x3800- 0x3FFF/ 0x7000- 0x7FFF | 0x37FF/ 0x6FFF | 0x3800/ 0x7000 |

Note: 1. The different BOOTSZ Fuse configurations are shown in Figure 109

**Table 105.** Read-While-Write Limit(1)

| Section | Pages | Address |
|---|---|---|
| Read-While-Write section (RWW) | 224/224 | 0x0000 - x37FF/ 0x0000 - 0x6FFF |
| No Read-While-Write section (NRWW) | 32/32 | 0x3800 - 0x3FFF/ 0x7000-0x7FFF |

Note: 1. For details about these two section, see “NRWW – No Read-While-Write Section” on page 244 and “RWW – Read-While-Write Section” on page 244.

**Table 106.** Explanation of different variables used in Figure 110 and the mapping to the Z-pointer()

| Variable | | Corresponding Z-value | Description |
|---|---|---|---|
| PCMSB | 13/14 | | Most significant bit in the Program Counter. (Program Counter is 14/15 bits PC[13/14:0]) |
| PAGEMSB | 5/6 | | Most significant bit which is used to address the words within one page (64/128 words in a page requires six/seven bits PC [5/6:0]). |
| ZPCMSB | | Z14/15 | Bit in Z-register that is mapped to PCMSB. Because Z0 is not used, the ZPCMSB equals PCMSB + 1. |
| ZPAGEMSB | | Z6/7 | Bit in Z-register that is mapped to PCMSB. Because Z0 is not used, the ZPAGEMSB equals PAGEMSB + 1. |
| PCPAGE | PC[13 /14:6/7] | Z14/15:Z7/8 | Program Counter page address: Page select, for Page Erase and Page Write |
| PCWORD | PC [5/6:0] | Z6/7:Z1 | Program Counter word address: Word select, for filling temporary buffer (must be zero during Page Write operation) |

Z0: should be zero for all SPM commands, byte select for the LPM instruction.
See “Addressing the Flash During Self-Programming” on page 248 for details about the use of Z-pointer during Self-Programming.

# Memory Programming

## Program And Data Memory Lock Bits

The ATmega325/3250/645/6450 provides six Lock bits which can be left unprogrammed (“1”) or can be programmed (“0”) to obtain the additional features listed in Table 108. The Lock bits can only be erased to “1” with the Chip Erase command.

**Table 107.** Lock Bit Byte[1]

| Lock Bit Byte | Bit No | Description | Default Value |
|---|---|---|---|
| | 7 | – | 1 (unprogrammed) |
| | 6 | – | 1 (unprogrammed) |
| BLB12 | 5 | Boot Lock bit | 1 (unprogrammed) |
| BLB11 | 4 | Boot Lock bit | 1 (unprogrammed) |
| BLB02 | 3 | Boot Lock bit | 1 (unprogrammed) |
| BLB01 | 2 | Boot Lock bit | 1 (unprogrammed) |
| LB2 | 1 | Lock bit | 1 (unprogrammed) |
| LB1 | 0 | Lock bit | 1 (unprogrammed) |

Note: 1. “1” means unprogrammed, “0” means programmed

**Table 108.** 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 Boot Lock bits and Fuse bits are locked in both Serial and Parallel Programming mode.[1] |
| **BLB0 Mode** | **BLB02** | **BLB01** | |
| 1 | 1 | 1 | No restrictions for SPM or LPM accessing the Application section. |
| 2 | 1 | 0 | SPM is not allowed to write to the Application section. |
| 3 | 0 | 0 | SPM is not allowed to write to the Application section, and LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section. |
| 4 | 0 | 1 | LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section. |

**Table 108.** Lock Bit Protection Modes[1][2] (Continued)

| Memory Lock Bits | | | Protection Type |
|---|---|---|---|
| **BLB1 Mode** | **BLB12** | **BLB11** | |
| 1 | 1 | 1 | No restrictions for SPM or LPM accessing the Boot Loader section. |
| 2 | 1 | 0 | SPM is not allowed to write to the Boot Loader section. |
| 3 | 0 | 0 | SPM is not allowed to write to the Boot Loader section, and LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section. |
| 4 | 0 | 1 | LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section. |

Notes: 1. Program the Fuse bits and Boot Lock bits before programming the LB1 and LB2.
2. "1" means unprogrammed, "0" means programmed

## Fuse Bits

The ATmega325/3250/645/6450 has three Fuse bytes. Table 109 - Table 111 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 109.** Extended Fuse Byte

| Extended Fuse Byte | Bit No | Description | Default Value |
|---|---|---|---|
| – | 7 | – | 1 |
| – | 6 | – | 1 |
| – | 5 | – | 1 |
| – | 4 | – | 1 |
| – | 5 | – | 1 |
| BODLEVEL1[1] | 2 | Brown-out Detector trigger level | 1 (unprogrammed) |
| BODLEVEL0[1] | 1 | Brown-out Detector trigger level | 1 (unprogrammed) |
| RSTDISBL[2] | 0 | External Reset Disable | 1 (unprogrammed) |

Notes: 1. See Table 17 on page 41 for BODLEVEL Fuse decoding.
2. Port G, PG5 is input only. Pull-up is always on. See "Alternate Functions of Port G" on page 67.

**Table 110.** Fuse High Byte

| Fuse High Byte | Bit No | Description | Default Value |
|---|---|---|---|
| OCDEN[4] | 7 | Enable OCD | 1 (unprogrammed, OCD disabled) |
| JTAGEN[5] | 6 | Enable JTAG | 0 (programmed, JTAG enabled) |
| SPIEN[1] | 5 | Enable Serial Program and Data Downloading | 0 (programmed, SPI prog. enabled) |
| WDTON[3] | 4 | Watchdog Timer always on | 1 (unprogrammed) |
| EESAVE | 3 | EEPROM memory is preserved through the Chip Erase | 1 (unprogrammed, EEPROM not preserved) |
| BOOTSZ1 | 2 | Select Boot Size (see Table 112 for details) | 0 (programmed)[2] |
| BOOTSZ0 | 1 | Select Boot Size (see Table 112 for details) | 0 (programmed)[2] |
| BOOTRST | 0 | Select Reset Vector | 1 (unprogrammed) |

Note:
1. The SPIEN Fuse is not accessible in serial programming mode.
2. The default value of BOOTSZ1..0 results in maximum Boot Size. See Table 104 on page 255 for details.
3. See “Watchdog Timer Control Register – WDTCR” on page 44 for details.
4. Never ship a product with the OCDEN Fuse programmed regardless of the setting of Lock bits and JTAGEN Fuse. A programmed OCDEN Fuse enables some parts of the clock system to be running in all sleep modes. This may increase the power consumption.
5. If the JTAG interface is left unconnected, the JTAGEN fuse should if possible be disabled. This to avoid static current at the TDO pin in the JTAG interface.

**Table 111.** 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] |
| CKSEL3 | 3 | Select Clock source | 0 (programmed)[2] |
| CKSEL2 | 2 | Select Clock source | 0 (programmed)[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 16 on page 39 for details.
2. The default setting of CKSEL3..0 results in internal RC Oscillator @ 8 MHz. See Table 7 on page 28 for details.
3. The CKOUT Fuse allow the system clock to be output on PORTE7. See “Clock Output Buffer” on page 30 for details.
4. See “System Clock Prescaler” on page 31 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.

### 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.

## 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 ATmega325/3250/645/6450 the signature bytes are:

1. 0x000: 0x1E (indicates manufactured by Atmel).
2. 0x001: 0x95/0x96 (indicates Flash memory, refer to “Part Number” on page 218).
3. 0x002: 0x03/0x04 (indicates device, refer to “Part Number” on page 218).

## Calibration Byte

The ATmega325/3250/645/6450 has a byte calibration value for the internal RC 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 RC Oscillator.

## 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 ATmega325/3250/645/6450. Pulses are assumed to be at least 250ns unless otherwise noted.

### Signal Names

In this section, some pins of the ATmega325/3250/645/6450 are referenced by signal names describing their functionality during parallel programming, see Figure 111 and Table 112. Pins not described in the following table are referenced by pin names.

The XA1/XA0 pins determine the action executed when the XTAL1 pin is given a positive pulse. The bit coding is shown in Table 114.

When pulsing $\overline{WR}$ or $\overline{OE}$, the command loaded determines the action executed. The different Commands are shown in Table 115.

**Figure 111.** Parallel Programming

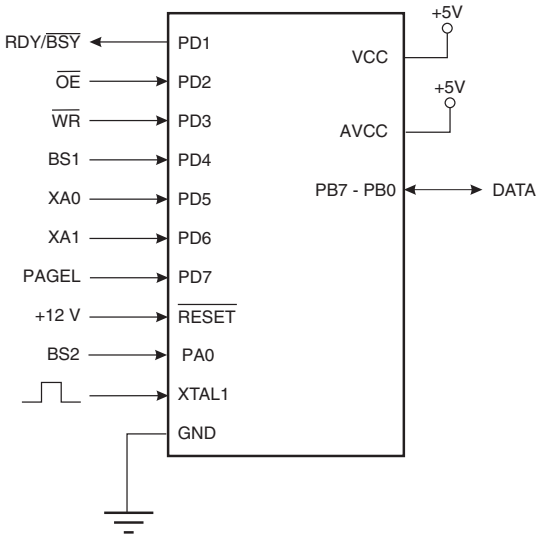



**Table 112.** Pin Name Mapping

| 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 | PA0 | I | Byte Select 2 (“0” selects low byte, “1” selects 2’nd high byte). |
| DATA | PB7-0 | I/O | Bi-directional Data bus (Output when $\overline{OE}$ is low). |

**Table 113.** 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 |

**Table 114.** XA1 and XA0 Coding

| XA1 | XA0 | Action when XTAL1 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 |

**Table 115.** 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 |

**Table 116.** No. of Words in a Page and No. of Pages in the Flash

| Flash Size | Page Size | PCWORD | No. of Pages | PCPAGE | PCMSB |
|---|---|---|---|---|---|
| 16/32K words (32/64K bytes) | 64/128 words | PC[5/6:0] | 256 | PC [13/14:6/7] | 13/14 |

**Table 117.** No. of Words in a Page and No. of Pages in the EEPROM

| EEPROM Size | Page Size | PCWORD | No. of Pages | PCPAGE | EEAMSB |
|---|---|---|---|---|---|
| 1K/2K bytes | 4/8 bytes | EEA[1/2:0] | 256 | EEA [13/14:2/3] | 13/14 |

## Serial Programming Pin Mapping

**Table 118.** Pin Mapping Serial Programming

| Symbol | Pins | I/O | Description |
|---|---|---|---|
| MOSI | PB2 | I | Serial Data in |
| MISO | PB3 | O | Serial Data out |
| SCK | PB1 | I | Serial Clock |

## Parallel Programming

### Enter Programming Mode

The following algorithm puts the device in parallel programming mode:

1. Apply 4.5 - 5.5V between $V_{CC}$ and GND.
2. Set $\overline{\text{RESET}}$ to “0” and toggle XTAL1 at least six times.
3. Set the Prog_enable pins listed in Table 113 on page 261 to “0000” and wait at least 100ns.
4. Apply 11.5 - 12.5V to $\overline{\text{RESET}}$. Any activity on Prog_enable pins within 100 ns after +12V has been applied to $\overline{\text{RESET}}$, will cause the device to fail entering programming mode.
5. Wait at least 50 µs before sending a new command.

### 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.

### 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 EEPRPOM 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 XTAL1 a positive pulse. This loads the command.
5. Give $\overline{\text{WR}}$ a negative pulse. This starts the Chip Erase. RDY/$\overline{\text{BSY}}$ goes low.
6. Wait until RDY/$\overline{\text{BSY}}$ goes high before loading a new command.

### Programming the Flash

The Flash is organized in pages, see Table 116 on page 262. 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 XTAL1 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 XTAL1 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 XTAL1 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 XTAL1 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 113 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 112 on page 265. Note that if less than eight bits are required to address words in the page (pagesize < 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 XTAL1 a positive pulse. This loads the address high byte.

H. Program Page

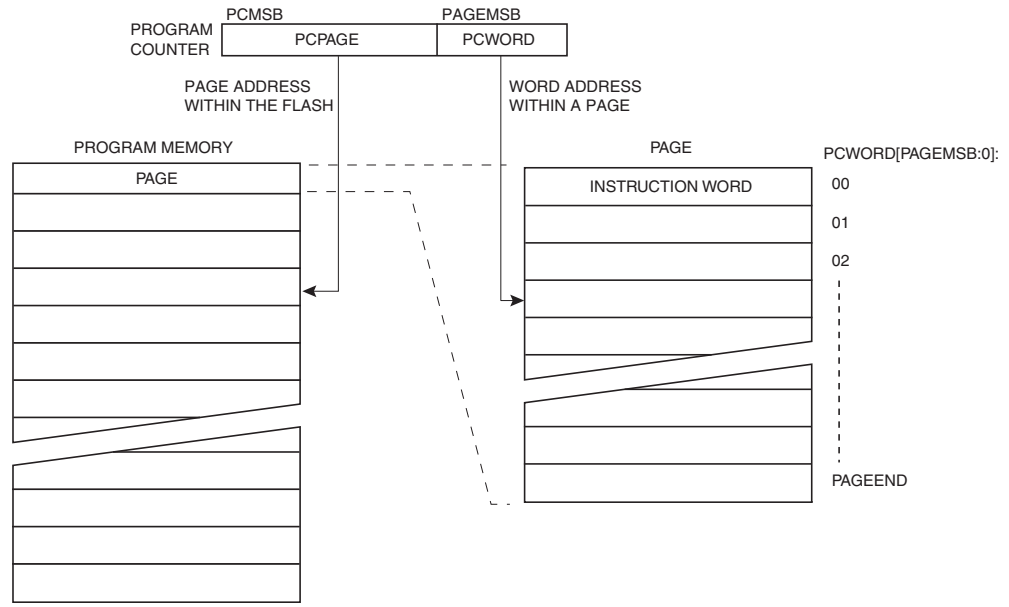
1. Give $\overline{\text{WR}}$ a negative pulse. This starts programming of the entire page of data. RDY/$\overline{\text{BSY}}$ goes low.
2. Wait until RDY/$\overline{\text{BSY}}$ goes high (See Figure 113 for signal waveforms).

I. Repeat B through H until the entire Flash is programmed or until all data has been programmed.

J. End Page Programming

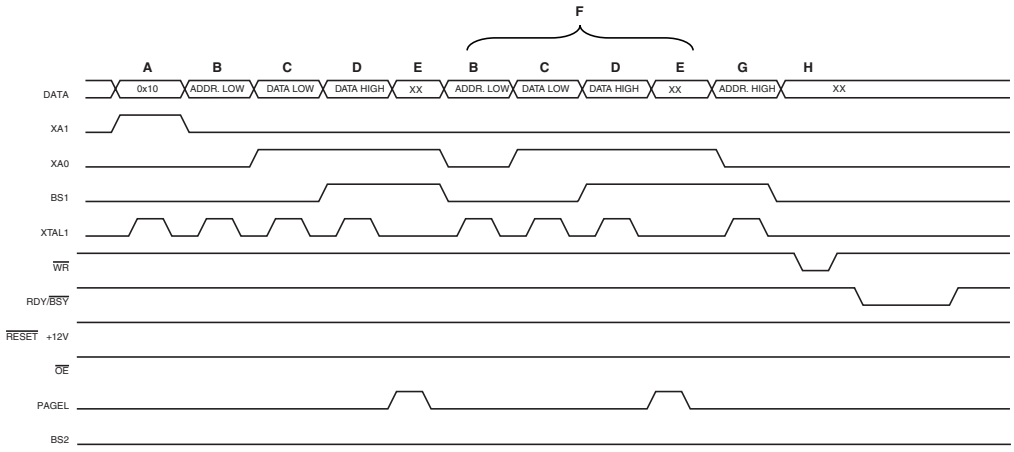
1. 1. Set XA1, XA0 to “10”. This enables command loading.
2. Set DATA to “0000 0000”. This is the command for No Operation.
3. Give XTAL1 a positive pulse. This loads the command, and the internal write signals are reset.

**Figure 112.** Addressing the Flash Which is Organized in Pages[1]




Note: 1. PCPAGE and PCWORD are listed in Table 116 on page 262.

**Figure 113.** Programming the Flash Waveforms[1]




Note: 1. "XX" is don't care. The letters refer to the programming description above.

**Programming the EEPROM**

The EEPROM is organized in pages, see Table 117 on page 262. 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 "Programming the Flash" on page 263 for details on Command, Address and Data loading):

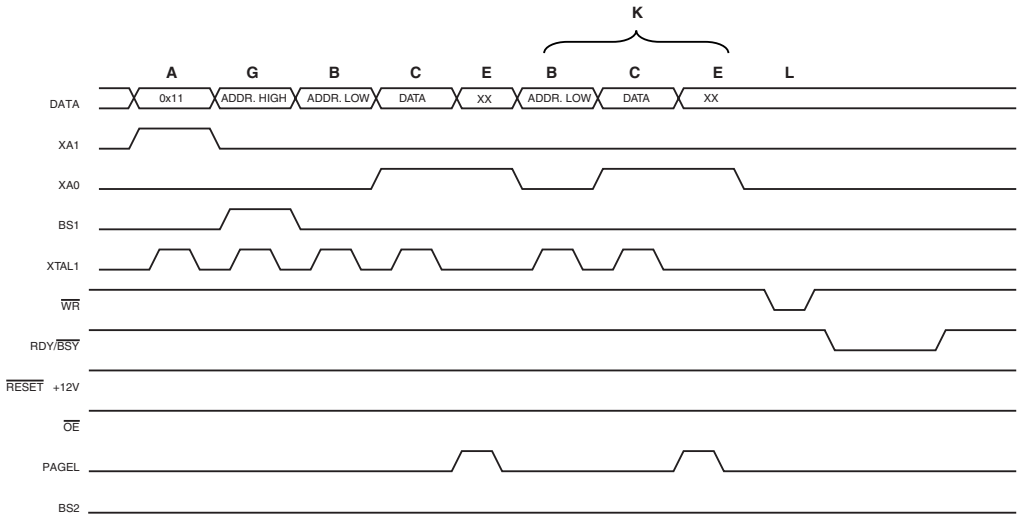
1. A: Load Command "0001 0001".
2. G: Load Address High Byte (0x00 - 0xFF).
3. B: Load Address Low Byte (0x00 - 0xFF).
4. C: Load Data (0x00 - 0xFF).
5. E: Latch data (give PAGEL a positive pulse).

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 114 for signal waveforms).

**Figure 114.** Programming the EEPROM Waveforms




**Reading the Flash**

The algorithm for reading the Flash memory is as follows (refer to “Programming the Flash” on page 263 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”.

**Reading the EEPROM**

The algorithm for reading the EEPROM memory is as follows (refer to “Programming the Flash” on page 263 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”.

**Programming the Fuse Low Bits**

The algorithm for programming the Fuse Low bits is as follows (refer to “Programming the Flash” on page 263 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.

**Programming the Fuse High Bits**

The algorithm for programming the Fuse High bits is as follows (refer to “Programming the Flash” on page 263 for details on Command and Data loading):

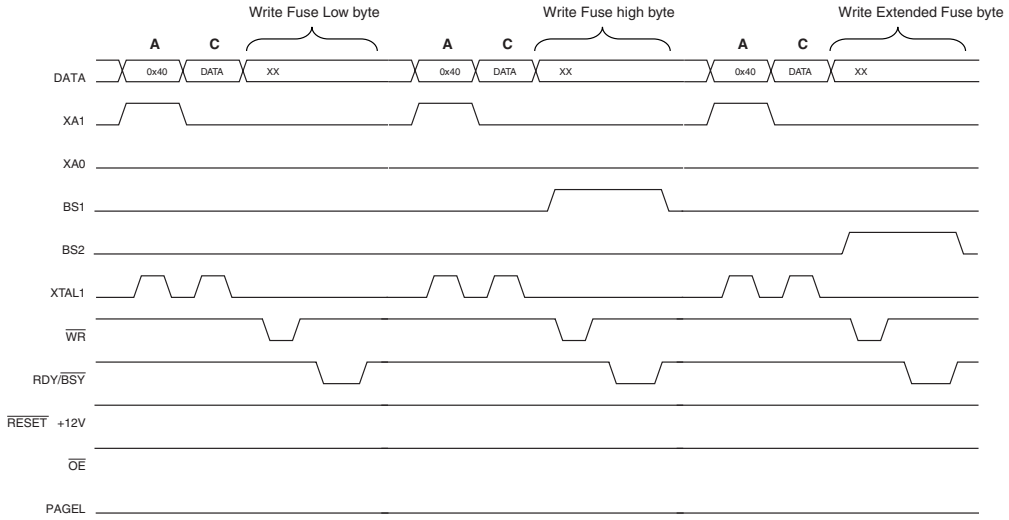
1. A: Load Command “0100 0000”.
2. C: Load Data Low Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Set BS1 to “1” and BS2 to “0”. This selects high data byte.
4. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.
5. Set BS1 to “0”. This selects low data byte.

**Programming the Extended Fuse Bits**

The algorithm for programming the Extended Fuse bits is as follows (refer to “Programming the Flash” on page 263 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 115.** Programming the FUSES Waveforms




**Programming the Lock Bits**

The algorithm for programming the Lock bits is as follows (refer to “Programming the Flash” on page 263 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 Boot Lock bits by any External Programming mode.
3. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.

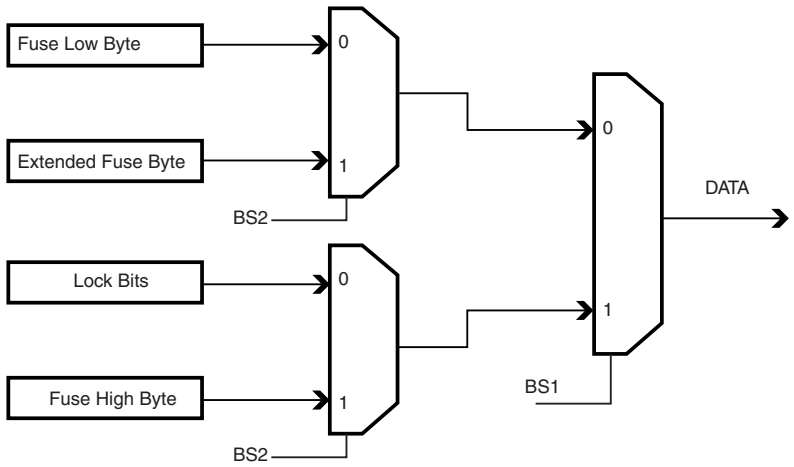
The Lock bits can only be cleared by executing Chip Erase.

### Reading the Fuse and Lock Bits

The algorithm for reading the Fuse and Lock bits is as follows (refer to “Programming the Flash” on page 263 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 Extended Fuse 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 116.** Mapping Between BS1, BS2 and the Fuse and Lock Bits During Read




### Reading the Signature Bytes

The algorithm for reading the Signature bytes is as follows (refer to “Programming the Flash” on page 263 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”.

### Reading the Calibration Byte

The algorithm for reading the Calibration byte is as follows (refer to “Programming the Flash” on page 263 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”.

## Parallel Programming Characteristics

**Figure 117.** Parallel Programming Timing, Including some General Timing Requirements

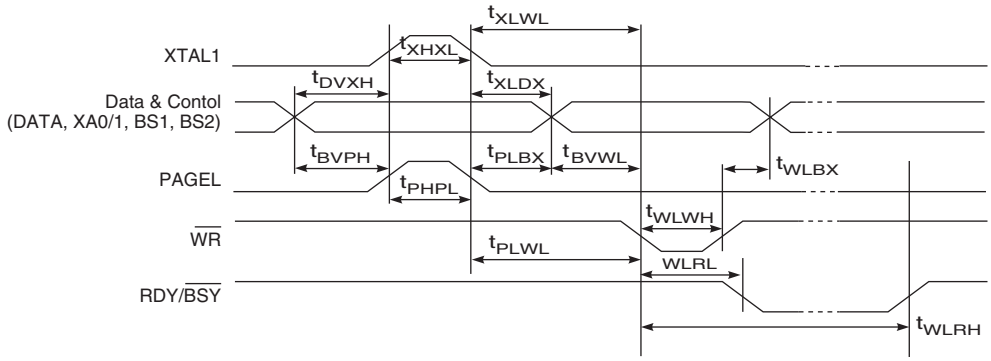
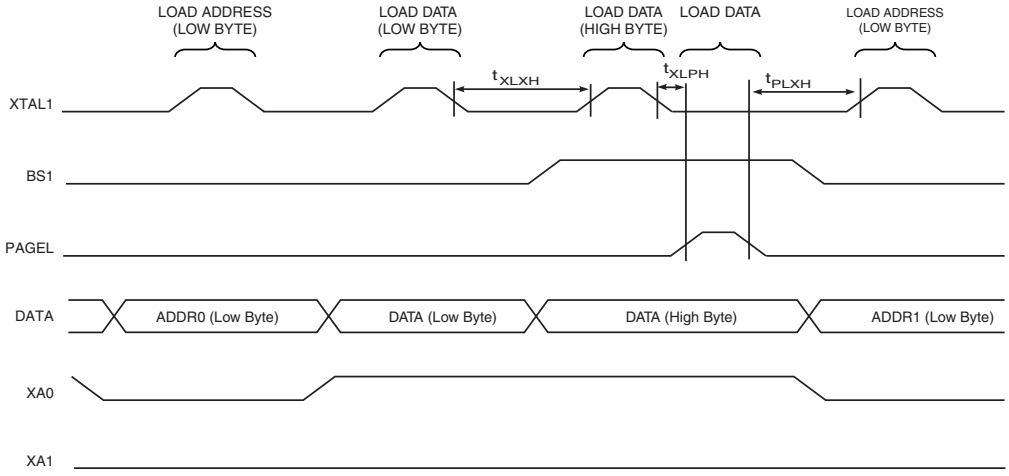
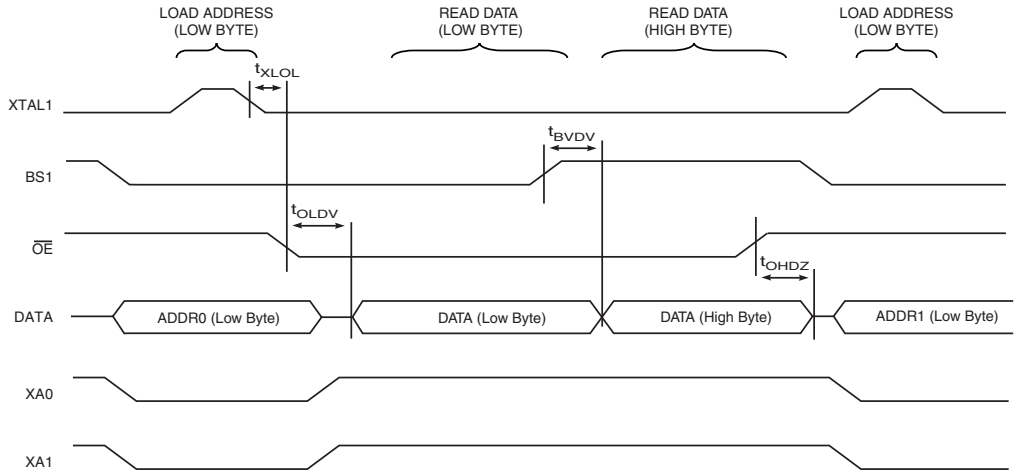



**Figure 118.** Parallel Programming Timing, Loading Sequence with Timing Requirements[1]




Note: 1. The timing requirements shown in Figure 117 (i.e., $t_{DVXH}$, $t_{XHXL}$, and $t_{XLDX}$) also apply to loading operation.

**Figure 119.** Parallel Programming Timing, Reading Sequence (within the Same Page) with Timing Requirements(1)




Note: 1. The timing requirements shown in Figure 117 (i.e., $t_{DVXH}$, $t_{XHXL}$, and $t_{XLDX}$) also apply to reading operation.

**Table 119.** Parallel Programming Characteristics, $V_{CC}$ = 5V ± 10%

| Symbol | Parameter | Min | Typ | Max | Units |
|---|---|---|---|---|---|
| $V_{PP}$ | Programming Enable Voltage | 11.5 | | 12.5 | V |
| $I_{PP}$ | Programming Enable Current | | | 250 | µA |
| $t_{DVXH}$ | Data and Control Valid before XTAL1 High | 67 | | | ns |
| $t_{XLXH}$ | XTAL1 Low to XTAL1 High | 200 | | | ns |
| $t_{XHXL}$ | XTAL1 Pulse Width High | 150 | | | ns |
| $t_{XLDX}$ | Data and Control Hold after XTAL1 Low | 67 | | | ns |
| $t_{XLWL}$ | XTAL1 Low to $\overline{WR}$ Low | 0 | | | ns |
| $t_{XLPH}$ | XTAL1 Low to PAGEL high | 0 | | | ns |
| $t_{PLXH}$ | PAGEL low to XTAL1 high | 150 | | | ns |
| $t_{BVPH}$ | BS1 Valid before PAGEL High | 67 | | | ns |
| $t_{PHPL}$ | PAGEL Pulse Width High | 150 | | | ns |
| $t_{PLBX}$ | BS1 Hold after PAGEL Low | 67 | | | ns |
| $t_{WLBX}$ | BS2/1 Hold after $\overline{WR}$ Low | 67 | | | ns |
| $t_{PLWL}$ | PAGEL Low to $\overline{WR}$ Low | 67 | | | ns |
| $t_{BVWL}$ | BS1 Valid to $\overline{WR}$ Low | 67 | | | ns |
| $t_{WLWH}$ | $\overline{WR}$ Pulse Width Low | 150 | | | ns |
| $t_{WLRL}$ | $\overline{WR}$ Low to RDY/$\overline{BSY}$ Low | 0 | | 1 | µs |
| $t_{WLRH}$ | $\overline{WR}$ Low to RDY/$\overline{BSY}$ High(1) | 3.7 | | 4.5 | ms |
| $t_{WLRH_CE}$ | $\overline{WR}$ Low to RDY/$\overline{BSY}$ High for Chip Erase(2) | 7.5 | | 9 | ms |
| $t_{XLOL}$ | XTAL1 Low to $\overline{OE}$ Low | 0 | | | ns |

**Table 119.** Parallel Programming Characteristics, $V_{CC}$ = 5V ± 10% (Continued)

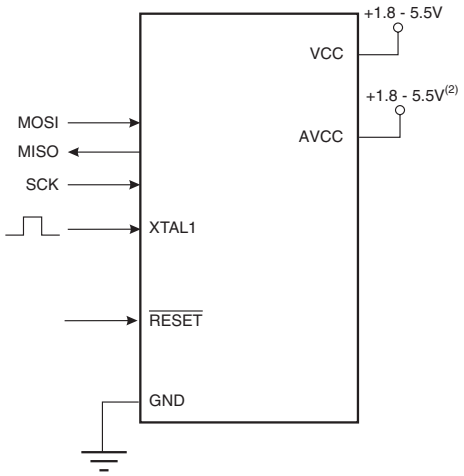
| Symbol | Parameter | Min | Typ | Max | Units |
|---|---|---|---|---|---|
| $t_{BVDV}$ | BS1 Valid to DATA valid | 0 | | 250 | ns |
| $t_{OLDV}$ | $\overline{OE}$ Low to DATA Valid | | | 250 | ns |
| $t_{OHDZ}$ | $\overline{OE}$ High to DATA Tri-stated | | | 250 | ns |

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.

## Serial Downloading

Both the Flash and EEPROM memory arrays can be programmed using the serial SPI bus while $\overline{RESET}$ is pulled to GND. The serial interface consists of pins SCK, MOSI (input) and MISO (output). After $\overline{RESET}$ is set low, the Programming Enable instruction needs to be executed first before program/erase operations can be executed. NOTE, in Table 118 on page 262, the pin mapping for SPI programming is listed. Not all parts use the SPI pins dedicated for the internal SPI interface.

**Figure 120.** 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 XTAL1 pin.
2. $V_{CC}$ - 0.3V < AVCC < $V_{CC}$ + 0.3V, however, AVCC should always be within 1.8 - 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 $f_{ck}$ < 12 MHz, 3 CPU clock cycles for $f_{ck}$ >= 12 MHz

High:> 2 CPU clock cycles for $f_{ck}$ < 12 MHz, 3 CPU clock cycles for $f_{ck}$ >= 12 MHz

**Serial Programming Algorithm**

When writing serial data to the ATmega325/3250/645/6450, data is clocked on the rising edge of SCK.

When reading data from the ATmega325/3250/645/6450, data is clocked on the falling edge of SCK. See Figure 121 for timing details.

To program and verify the ATmega325/3250/645/6450 in the serial programming mode, the following sequence is recommended (See four byte instruction formats in Table 121):

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 page size is found in Table 116 on page 262. 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 8 MSB of the address. If polling is not used, the user must wait at least $t_{WD_FLASH}$ before issuing the next page. (See Table 120.) 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 120.) 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 2 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 4 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 page (See Table 117). 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 120.** Minimum Wait Delay Before Writing the Next Flash or EEPROM Location

| Symbol | Minimum Wait Delay |
|---|---|
| $t_{WD_FUSE}$ | 4.5 ms |
| $t_{WD_FLASH}$ | 4.5 ms |
| $t_{WD_EEPROM}$ | 9.0 ms |
| $t_{WD_ERASE}$ | 9.0 ms |

**Figure 121.** Serial Programming Waveforms

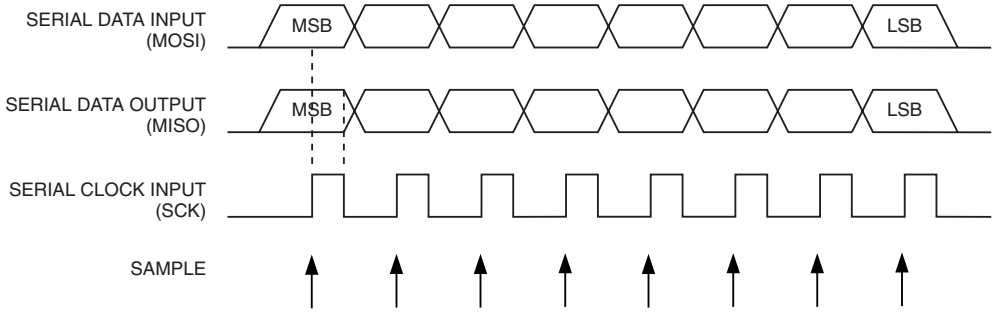



**Table 121.** Serial Programming Instruction Set

| Instruction | Instruction Format | | | | Operation |
|---|---|---|---|---|---|
| | Byte 1 | Byte 2 | Byte 3 | Byte4 | |
| Programming Enable | 1010 1100 | 0101 0011 | xxxx xxxx | xxxx xxxx | Enable Serial Programming after $\overline{\text{RESET}}$ goes low. |
| Chip Erase | 1010 1100 | 100x xxxx | xxxx xxxx | xxxx xxxx | Chip Erase EEPROM and Flash. |
| Read Program Memory | 0010 **H**000 | 000**a aaaa**<br>/<br>00**aa aaaa** | **bbbb bbbb** | **oooo oooo** | Read **H** (high or low) data **o** from Program memory at word address **a**:**b**. |
| Load Program Memory Page | 0100 **H**000 | 000x xxxx | xx**bb bbbb**<br>/<br>x**bbb bbbb** | **iiii iiii** | Write **H** (high or low) data **i** to Program Memory page at word address **b**. Data low byte must be loaded before Data high byte is applied within the same address. |
| Write Program Memory Page | 0100 1100 | 000**a aaaa**<br>/<br>00**aa aaaa** | **bb**xx xxxx<br>/<br>**b**xxx xxxx | xxxx xxxx | Write Program Memory Page at address **a**:**b**. |
| Read EEPROM Memory | 1010 0000 | 000x xx**aa**<br>/<br>000x x**aaa** | **bbbb bbbb** | **oooo oooo** | Read data **o** from EEPROM memory at address **a**:**b**. |
| Write EEPROM Memory | 1100 0000 | 000x xx**aa**<br>/<br>000x x**aaa** | **bbbb bbbb** | **iiii iiii** | Write data **i** to EEPROM memory at address **a**:**b**. |

**Table 121.** Serial Programming Instruction Set (Continued)

| Instruction | Instruction Format | | | | Operation |
|---|---|---|---|---|---|
| | Byte 1 | Byte 2 | Byte 3 | Byte4 | |
| Load EEPROM Memory Page (page access) | 1100 0001 | 0000 0000 | 0000 00**bb**<br>/<br>0000 0**bbb** | **iiii iiii** | Load data **i** to EEPROM memory page buffer. After data is loaded, program EEPROM page. |
| Write EEPROM Memory Page (page access) | 1100 0010 | 00xx xx**aa**<br>/<br>00xx x**aaa** | **bbbb bb**00<br>/<br>**bbbb b**000 | xxxx xxxx | Write EEPROM page at address **a**:**b**. |
| Read Lock bits | 0101 1000 | 0000 0000 | xxxx xxxx | xx**oo oooo** | Read Lock bits. “0” = programmed, “1” = unprogrammed. See Table 107 on page 257 for details. |
| Write Lock bits | 1010 1100 | 111x xxxx | xxxx xxxx | 11**ii iiii** | Write Lock bits. Set bits = “0” to program Lock bits. See Table 107 on page 257 for details. |
| Read Signature Byte | 0011 0000 | 000x xxxx | xxxx xx**bb** | **oooo oooo** | Read Signature Byte **o** at address **b**. |
| Write Fuse bits | 1010 1100 | 1010 0000 | xxxx xxxx | **iiii iiii** | Set bits = “0” to program, “1” to unprogram. See Table 86 on page 205 for details. |
| Write Fuse High bits | 1010 1100 | 1010 1000 | xxxx xxxx | **iiii iiii** | Set bits = “0” to program, “1” to unprogram. See Table 85 on page 199 for details. |
| Write Extended Fuse Bits | 1010 1100 | 1010 0100 | xxxx xxxx | **xxxx xxii** | Set bits = “0” to program, “1” to unprogram. See Table 109 on page 258 for details. |
| Read Fuse bits | 0101 0000 | 0000 0000 | xxxx xxxx | **oooo oooo** | Read Fuse bits. “0” = programmed, “1” = unprogrammed. See Table 86 on page 205 for details. |
| Read Fuse High bits | 0101 1000 | 0000 1000 | xxxx xxxx | **oooo oooo** | Read Fuse High bits. “0” = programmed, “1” = unprogrammed. See Table 85 on page 199 for details. |
| Read Extended Fuse Bits | 0101 0000 | 0000 1000 | xxxx xxxx | **oooo oooo** | Read Extended Fuse bits. “0” = programmed, “1” = unprogrammed. See Table 109 on page 258 for details. |
| Read Calibration Byte | 0011 1000 | 000x xxxx | 0000 0000 | **oooo oooo** | Read Calibration Byte |
| Poll RDY/$\overline{\text{BSY}}$ | 1111 0000 | 0000 0000 | xxxx xxxx | xxxx xxx**o** | If **o** = “1”, a programming operation is still busy. Wait until this bit returns to “0” before applying another command. |

Note: **a** = address high bits, **b** = address low bits, **H** = 0 - Low byte, 1 - High Byte, **o** = data out, **i** = data in, x = don’t care

**SPI Serial Programming Characteristics**

For characteristics of the SPI module see “SPI Timing Characteristics” on page 290.

## Programming via the JTAG Interface

Programming through the JTAG interface requires control of the four JTAG specific pins: TCK, TMS, TDI, and TDO. Control of the reset and clock pins is not required.

To be able to use the JTAG interface, the JTAGEN Fuse must be programmed. The device is default shipped with the fuse programmed. In addition, the JTD bit in MCUCSR must be cleared. Alternatively, if the JTD bit is set, the external reset can be forced low. Then, the JTD bit will be cleared after two chip clocks, and the JTAG pins are available for programming. This provides a means of using the JTAG pins as normal port pins in Running mode while still allowing In-System Programming via the JTAG interface. Note that this technique can not be used when using the JTAG pins for Boundary-scan or On-chip Debug. In these cases the JTAG pins must be dedicated for this purpose.

During programming the clock frequency of the TCK Input must be less than the maximum frequency of the chip. The System Clock Prescaler can not be used to divide the TCK Clock Input into a sufficiently low frequency.

As a definition in this data sheet, the LSB is shifted in and out first of all Shift Registers.

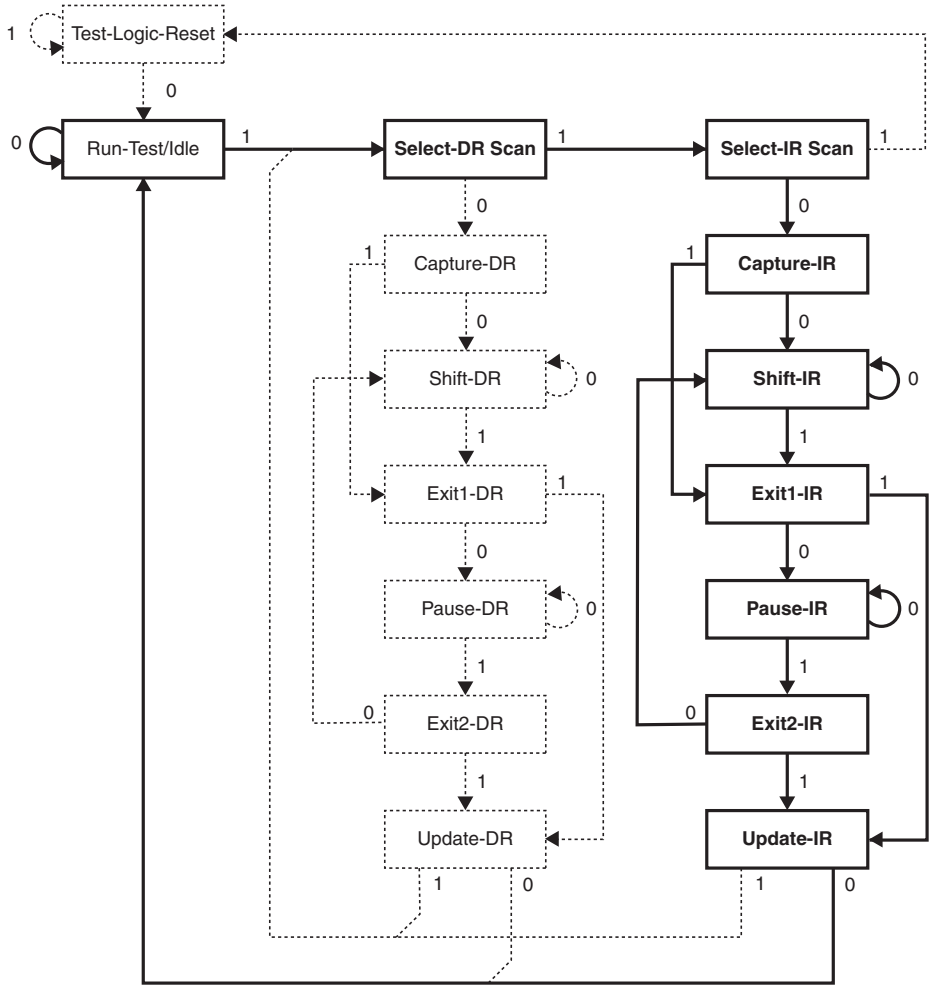
**Programming Specific JTAG Instructions**

The Instruction Register is 4-bit wide, supporting up to 16 instructions. The JTAG instructions useful for programming are listed below.

The OPCODE for each instruction is shown behind the instruction name in hex format. The text describes which Data Register is selected as path between TDI and TDO for each instruction.

The Run-Test/Idle state of the TAP controller is used to generate internal clocks. It can also be used as an idle state between JTAG sequences. The state machine sequence for changing the instruction word is shown in Figure 122.

**Figure 122.** State Machine Sequence for Changing the Instruction Word




**AVR_RESET (0xC)**

The AVR specific public JTAG instruction for setting the AVR device in the Reset mode or taking the device out from the Reset mode. The TAP controller is not reset by this instruction. The one bit Reset Register is selected as Data Register. Note that the reset will be active as long as there is a logic “one” in the Reset Chain. The output from this chain is not latched.

The active states are:

- Shift-DR: The Reset Register is shifted by the TCK input.

**PROG_ENABLE (0x4)**

The AVR specific public JTAG instruction for enabling programming via the JTAG port. The 16-bit Programming Enable Register is selected as Data Register. The active states are the following:

- Shift-DR: The programming enable signature is shifted into the Data Register.
- Update-DR: The programming enable signature is compared to the correct value, and Programming mode is entered if the signature is valid.

**PROG_COMMANDS (0x5)**

The AVR specific public JTAG instruction for entering programming commands via the JTAG port. The 15-bit Programming Command Register is selected as Data Register. The active states are the following:

- Capture-DR: The result of the previous command is loaded into the Data Register.
- Shift-DR: The Data Register is shifted by the TCK input, shifting out the result of the previous command and shifting in the new command.
- Update-DR: The programming command is applied to the Flash inputs
- Run-Test/Idle: One clock cycle is generated, executing the applied command (not always required, see Table 122 below).

**PROG_PAGELOAD (0x6)**

The AVR specific public JTAG instruction to directly load the Flash data page via the JTAG port. An 8-bit Flash Data Byte Register is selected as the Data Register. This is physically the 8 LSBs of the Programming Command Register. The active states are the following:

- Shift-DR: The Flash Data Byte Register is shifted by the TCK input.
- Update-DR: The content of the Flash Data Byte Register is copied into a temporary register. A write sequence is initiated that within 11 TCK cycles loads the content of the temporary register into the Flash page buffer. The AVR automatically alternates between writing the low and the high byte for each new Update-DR state, starting with the low byte for the first Update-DR encountered after entering the PROG_PAGELOAD command. The Program Counter is pre-incremented before writing the low byte, except for the first written byte. This ensures that the first data is written to the address set up by PROG_COMMANDS, and loading the last location in the page buffer does not make the program counter increment into the next page.

**PROG_PAGEREAD (0x7)**

The AVR specific public JTAG instruction to directly capture the Flash content via the JTAG port. An 8-bit Flash Data Byte Register is selected as the Data Register. This is physically the 8 LSBs of the Programming Command Register. The active states are the following:

- Capture-DR: The content of the selected Flash byte is captured into the Flash Data Byte Register. The AVR automatically alternates between reading the low and the high byte for each new Capture-DR state, starting with the low byte for the first Capture-DR encountered after entering the PROG_PAGEREAD command. The Program Counter is post-incremented after reading each high byte, including the first read byte. This ensures that the first data is captured from the first address set up by PROG_COMMANDS, and reading the last location in the page makes the program counter increment into the next page.
- Shift-DR: The Flash Data Byte Register is shifted by the TCK input.

**Data Registers**

The Data Registers are selected by the JTAG instruction registers described in section “Programming Specific JTAG Instructions” on page 275. The Data Registers relevant for programming operations are:

- Reset Register
- Programming Enable Register
- Programming Command Register
- Flash Data Byte Register

### Reset Register

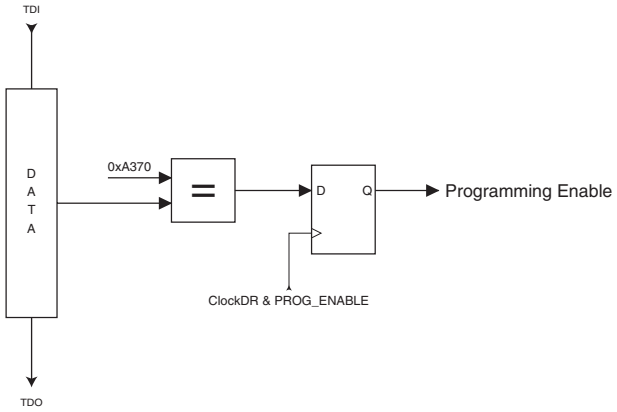
The Reset Register is a Test Data Register used to reset the part during programming. It is required to reset the part before entering Programming mode.

A high value in the Reset Register corresponds to pulling the external reset low. The part is reset as long as there is a high value present in the Reset Register. Depending on the Fuse settings for the clock options, the part will remain reset for a Reset Time-out period (refer to “Clock Sources” on page 26) after releasing the Reset Register. The output from this Data Register is not latched, so the reset will take place immediately, as shown in Figure 100 on page 219.

### Programming Enable Register

The Programming Enable Register is a 16-bit register. The contents of this register is compared to the programming enable signature, binary code 0b1010_0011_0111_0000. When the contents of the register is equal to the programming enable signature, programming via the JTAG port is enabled. The register is reset to 0 on Power-on Reset, and should always be reset when leaving Programming mode.

**Figure 123.** Programming Enable Register




### Programming Command Register

The Programming Command Register is a 15-bit register. This register is used to serially shift in programming commands, and to serially shift out the result of the previous command, if any. The JTAG Programming Instruction Set is shown in Table 122. The state sequence when shifting in the programming commands is illustrated in Figure 125.

**Figure 124.** Programming Command Register

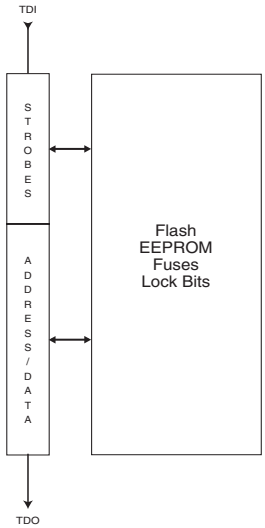


**Table 122.** JTAG Programming Instruction Set

**a** = address high bits, **b** = address low bits, **H** = 0 - Low byte, 1 - High Byte, **o** = data out, **i** = data in, x = don't care

| Instruction | TDI Sequence | TDO Sequence | Notes |
|---|---|---|---|
| 1a. Chip Erase | 0100011_10000000<br>0110001_10000000<br>0110011_10000000<br>0110011_10000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | |
| 1b. Poll for Chip Erase Complete | 0110011_10000000 | xxxxx**o**x_xxxxxxxx | (2) |
| 2a. Enter Flash Write | 0100011_00010000 | xxxxxxx_xxxxxxxx | |
| 2b. Load Address High Byte | 0000111_**aaaaaaaa** | xxxxxxx_xxxxxxxx | (9) |
| 2c. Load Address Low Byte | 0000011_**bbbbbbbb** | xxxxxxx_xxxxxxxx | |
| 2d. Load Data Low Byte | 0010011_**iiiiiiii** | xxxxxxx_xxxxxxxx | |
| 2e. Load Data High Byte | 0010111_**iiiiiiii** | xxxxxxx_xxxxxxxx | |
| 2f. Latch Data | 0110111_00000000<br>1110111_00000000<br>0110111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | (1) |
| 2g. Write Flash Page | 0110111_00000000<br>0110101_00000000<br>0110111_00000000<br>0110111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | (1) |
| 2h. Poll for Page Write Complete | 0110111_00000000 | xxxxx**o**x_xxxxxxxx | (2) |
| 3a. Enter Flash Read | 0100011_00000010 | xxxxxxx_xxxxxxxx | |
| 3b. Load Address High Byte | 0000111_**aaaaaaaa** | xxxxxxx_xxxxxxxx | (9) |
| 3c. Load Address Low Byte | 0000011_**bbbbbbbb** | xxxxxxx_xxxxxxxx | |
| 3d. Read Data Low and High Byte | 0110010_00000000<br>0110110_00000000<br>0110111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_**oooooooo**<br>xxxxxxx_**oooooooo** | <br>Low byte<br>High byte |
| 4a. Enter EEPROM Write | 0100011_00010001 | xxxxxxx_xxxxxxxx | |
| 4b. Load Address High Byte | 0000111_**aaaaaaaa** | xxxxxxx_xxxxxxxx | (9) |
| 4c. Load Address Low Byte | 0000011_**bbbbbbbb** | xxxxxxx_xxxxxxxx | |
| 4d. Load Data Byte | 0010011_**iiiiiiii** | xxxxxxx_xxxxxxxx | |
| 4e. Latch Data | 0110111_00000000<br>1110111_00000000<br>0110111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | (1) |
| 4f. Write EEPROM Page | 0110011_00000000<br>0110001_00000000<br>0110011_00000000<br>0110011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | (1) |
| 4g. Poll for Page Write Complete | 0110011_00000000 | xxxxx**o**x_xxxxxxxx | (2) |
| 5a. Enter EEPROM Read | 0100011_00000011 | xxxxxxx_xxxxxxxx | |
| 5b. Load Address High Byte | 0000111_**aaaaaaaa** | xxxxxxx_xxxxxxxx | (9) |

**Table 122.** JTAG Programming Instruction Set (Continued)

**a** = address high bits, **b** = address low bits, **H** = 0 - Low byte, 1 - High Byte, **o** = data out, **i** = data in, x = don't care

| Instruction | TDI Sequence | TDO Sequence | Notes |
|---|---|---|---|
| 5c. Load Address Low Byte | 0000011_**bbbbbbbb** | xxxxxxx_xxxxxxxx | |
| 5d. Read Data Byte | 0110011_**bbbbbbbb**<br>0110010_00000000<br>0110011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_**oooooooo** | |
| 6a. Enter Fuse Write | 0100011_01000000 | xxxxxxx_xxxxxxxx | |
| 6b. Load Data Low Byte[6] | 0010011_**iiiiiiii** | xxxxxxx_xxxxxxxx | (3) |
| 6c. Write Fuse Extended Byte | 0111011_00000000<br>0111001_00000000<br>0111011_00000000<br>0111011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | (1) |
| 6d. Poll for Fuse Write Complete | 0110111_00000000 | xxxxx**o**x_xxxxxxxx | (2) |
| 6e. Load Data Low Byte[7] | 0010011_**iiiiiiii** | xxxxxxx_xxxxxxxx | (3) |
| 6f. Write Fuse High Byte | 0110111_00000000<br>0110101_00000000<br>0110111_00000000<br>0110111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | (1) |
| 6g. Poll for Fuse Write Complete | 0110111_00000000 | xxxxx**o**x_xxxxxxxx | (2) |
| 6h. Load Data Low Byte[7] | 0010011_**iiiiiiii** | xxxxxxx_xxxxxxxx | (3) |
| 6i. Write Fuse Low Byte | 0110011_00000000<br>0110001_00000000<br>0110011_00000000<br>0110011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | (1) |
| 6j. Poll for Fuse Write Complete | 0110011_00000000 | xxxxx**o**x_xxxxxxxx | (2) |
| 7a. Enter Lock Bit Write | 0100011_00100000 | xxxxxxx_xxxxxxxx | |
| 7b. Load Data Byte[9] | 0010011_11**iiiiii** | xxxxxxx_xxxxxxxx | (4) |
| 7c. Write Lock Bits | 0110011_00000000<br>0110001_00000000<br>0110011_00000000<br>0110011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | (1) |
| 7d. Poll for Lock Bit Write complete | 0110011_00000000 | xxxxx**o**x_xxxxxxxx | (2) |
| 8a. Enter Fuse/Lock Bit Read | 0100011_00000100 | xxxxxxx_xxxxxxxx | |
| 8b. Read Extended Fuse Byte[6] | 0111010_00000000<br>0111011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_**oooooooo** | |
| 8c. Read Fuse High Byte[7] | 0111110_00000000<br>0111111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_**oooooooo** | |
| 8d. Read Fuse Low Byte[8] | 0110010_00000000<br>0110011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_**oooooooo** | |
| 8e. Read Lock Bits[9] | 0110110_00000000<br>0110111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xx**oooooo** | (5) |

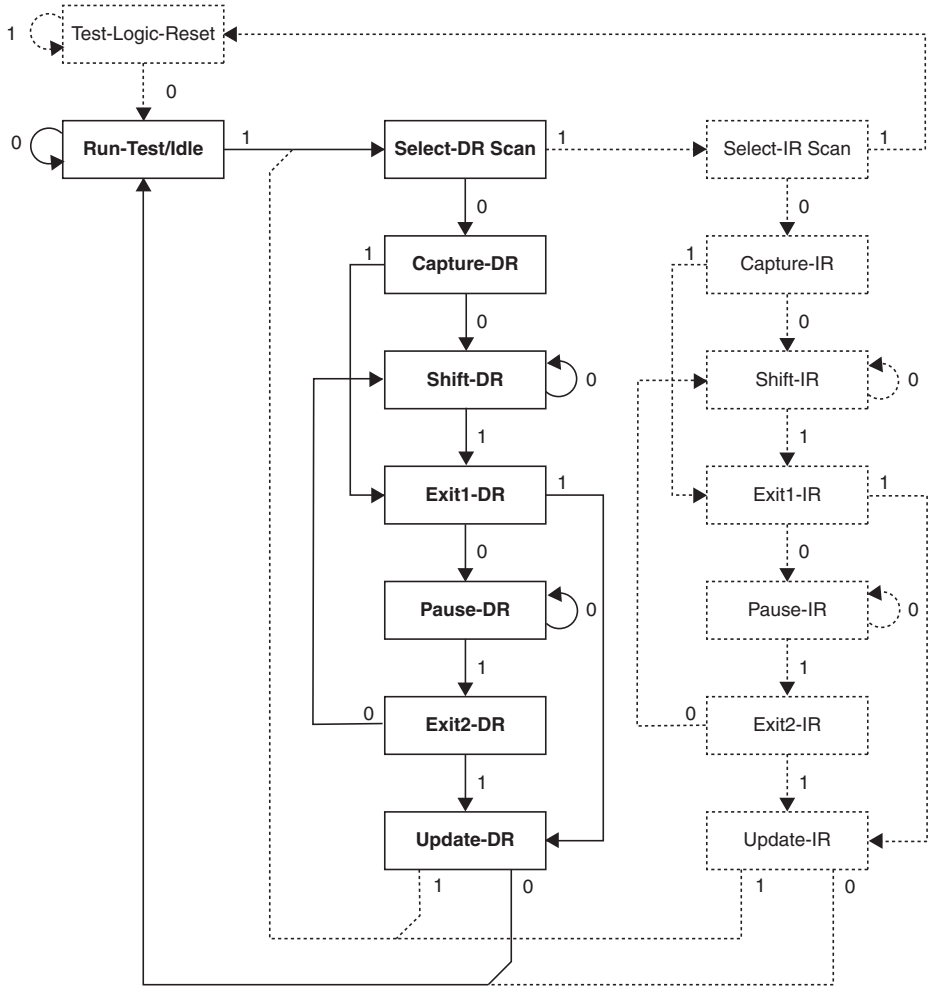
**Table 122.** JTAG Programming Instruction Set (Continued)

**a** = address high bits, **b** = address low bits, **H** = 0 - Low byte, 1 - High Byte, **o** = data out, **i** = data in, x = don't care

| Instruction | TDI Sequence | TDO Sequence | Notes |
|---|---|---|---|
| 8f. Read Fuses and Lock Bits | 0111010_00000000<br>0111110_00000000<br>0110010_00000000<br>0110110_00000000<br>0110111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_**oooooooo**<br>xxxxxxx_**oooooooo**<br>xxxxxxx_**oooooooo**<br>xxxxxxx_**oooooooo** | (5)<br>Fuse Ext. byte<br>Fuse High byte<br>Fuse Low byte<br>Lock bits |
| 9a. Enter Signature Byte Read | 0100011_00001000 | xxxxxxx_xxxxxxxx | |
| 9b. Load Address Byte | 0000011_**bbbbbbbb** | xxxxxxx_xxxxxxxx | |
| 9c. Read Signature Byte | 0110010_00000000<br>0110011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_**oooooooo** | |
| 10a. Enter Calibration Byte Read | 0100011_00001000 | xxxxxxx_xxxxxxxx | |
| 10b. Load Address Byte | 0000011_**bbbbbbbb** | xxxxxxx_xxxxxxxx | |
| 10c. Read Calibration Byte | 0110110_00000000<br>0110111_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_**oooooooo** | |
| 11a. Load No Operation Command | 0100011_00000000<br>0110011_00000000 | xxxxxxx_xxxxxxxx<br>xxxxxxx_xxxxxxxx | |

Notes:
1. This command sequence is not required if the seven MSB are correctly set by the previous command sequence (which is normally the case).
2. Repeat until **o** = "1".
3. Set bits to "0" to program the corresponding Fuse, "1" to unprogram the Fuse.
4. Set bits to "0" to program the corresponding Lock bit, "1" to leave the Lock bit unchanged.
5. "0" = programmed, "1" = unprogrammed.
6. The bit mapping for Fuses Extended byte is listed in Table 109 on page 258
7. The bit mapping for Fuses High byte is listed in Table 110 on page 259
8. The bit mapping for Fuses Low byte is listed in Table 111 on page 259
9. The bit mapping for Lock bits byte is listed in Table 107 on page 257
10. Address bits exceeding PCMSB and EEAMSB (Table 116 and Table 117) are don't care
11. All TDI and TDO sequences are represented by binary digits (0b...).

**Figure 125.** State Machine Sequence for Changing/Reading the Data Word




**Flash Data Byte Register**

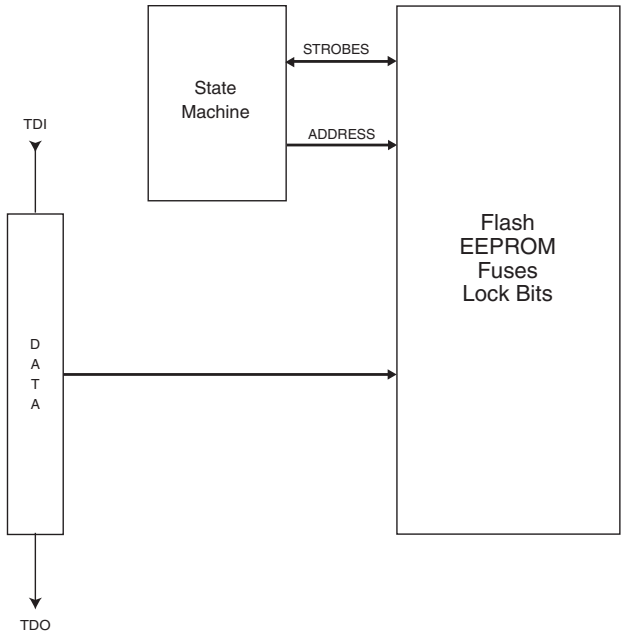
The Flash Data Byte Register provides an efficient way to load the entire Flash page buffer before executing Page Write, or to read out/verify the content of the Flash. A state machine sets up the control signals to the Flash and senses the strobe signals from the Flash, thus only the data words need to be shifted in/out.

The Flash Data Byte Register actually consists of the 8-bit scan chain and a 8-bit temporary register. During page load, the Update-DR state copies the content of the scan chain over to the temporary register and initiates a write sequence that within 11 TCK cycles loads the content of the temporary register into the Flash page buffer. The AVR automatically alternates between writing the low and the high byte for each new Update-DR state, starting with the low byte for the first Update-DR encountered after entering the PROG_PAGELOAD command. The Program Counter is pre-incremented before writing the low byte, except for the first written byte. This ensures that the first data is written to the address set up by PROG_COMMANDS, and loading the last location in the page buffer does not make the Program Counter increment into the next page.

During Page Read, the content of the selected Flash byte is captured into the Flash Data Byte Register during the Capture-DR state. The AVR automatically alternates between reading the low and the high byte for each new Capture-DR state, starting with the low byte for the first Capture-DR encountered after entering the PROG_PAGEREAD command. The Program Counter is post-incremented after reading each high byte,

including the first read byte. This ensures that the first data is captured from the first address set up by PROG_COMMANDS, and reading the last location in the page makes the program counter increment into the next page.

**Figure 126.** Flash Data Byte Register




The state machine controlling the Flash Data Byte Register is clocked by TCK. During normal operation in which eight bits are shifted for each Flash byte, the clock cycles needed to navigate through the TAP controller automatically feeds the state machine for the Flash Data Byte Register with sufficient number of clock pulses to complete its operation transparently for the user. However, if too few bits are shifted between each Update-DR state during page load, the TAP controller should stay in the Run-Test/Idle state for some TCK cycles to ensure that there are at least 11 TCK cycles between each Update-DR state.

**Programming Algorithm**

All references below of type “1a”, “1b”, and so on, refer to Table 122.

**Entering Programming Mode**

1. Enter JTAG instruction AVR_RESET and shift 1 in the Reset Register.
2. Enter instruction PROG_ENABLE and shift 0b1010_0011_0111_0000 in the Programming Enable Register.

**Leaving Programming Mode**

1. Enter JTAG instruction PROG_COMMANDS.
2. Disable all programming instructions by using no operation instruction 11a.
3. Enter instruction PROG_ENABLE and shift 0b0000_0000_0000_0000 in the programming Enable Register.
4. Enter JTAG instruction AVR_RESET and shift 0 in the Reset Register.

**Performing Chip Erase**

1. Enter JTAG instruction PROG_COMMANDS.
2. Start Chip Erase using programming instruction 1a.
3. Poll for Chip Erase complete using programming instruction 1b, or wait for $t_{WLRH_CE}$ (refer to Table 119 on page 270).

**Programming the Flash**

Before programming the Flash a Chip Erase must be performed, see “Performing Chip Erase” on page 283.

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash write using programming instruction 2a.
3. Load address High byte using programming instruction 2b.
4. Load address Low byte using programming instruction 2c.
5. Load data using programming instructions 2d, 2e and 2f.
6. Repeat steps 4 and 5 for all instruction words in the page.
7. Write the page using programming instruction 2g.
8. Poll for Flash write complete using programming instruction 2h, or wait for $t_{WLRH}$ (refer to Table 119 on page 270).
9. Repeat steps 3 to 7 until all data have been programmed.

A more efficient data transfer can be achieved using the PROG_PAGELOAD instruction:

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash write using programming instruction 2a.
3. Load the page address using programming instructions 2b and 2c. PCWORD (refer to Table 116 on page 262) is used to address within one page and must be written as 0.
4. Enter JTAG instruction PROG_PAGELOAD.
5. Load the entire page by shifting in all instruction words in the page byte-by-byte, starting with the LSB of the first instruction in the page and ending with the MSB of the last instruction in the page. Use Update-DR to copy the contents of the Flash Data Byte Register into the Flash page location and to auto-increment the Program Counter before each new word.
6. Enter JTAG instruction PROG_COMMANDS.
7. Write the page using programming instruction 2g.
8. Poll for Flash write complete using programming instruction 2h, or wait for $t_{WLRH}$ (refer to Table 119 on page 270).
9. Repeat steps 3 to 8 until all data have been programmed.

**Reading the Flash**

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash read using programming instruction 3a.
3. Load address using programming instructions 3b and 3c.
4. Read data using programming instruction 3d.
5. Repeat steps 3 and 4 until all data have been read.

A more efficient data transfer can be achieved using the PROG_PAGEREAD instruction:

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash read using programming instruction 3a.
3. Load the page address using programming instructions 3b and 3c. PCWORD (refer to Table 116 on page 262) is used to address within one page and must be written as 0.
4. Enter JTAG instruction PROG_PAGEREAD.
5. Read the entire page (or Flash) by shifting out all instruction words in the page (or Flash), starting with the LSB of the first instruction in the page (Flash) and

ending with the MSB of the last instruction in the page (Flash). The Capture-DR state both captures the data from the Flash, and also auto-increments the program counter after each word is read. Note that Capture-DR comes before the shift-DR state. Hence, the first byte which is shifted out contains valid data.

6. Enter JTAG instruction PROG_COMMANDS.
7. Repeat steps 3 to 6 until all data have been read.

**Programming the EEPROM**

Before programming the EEPROM a Chip Erase must be performed, see "Performing Chip Erase" on page 283.

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable EEPROM write using programming instruction 4a.
3. Load address High byte using programming instruction 4b.
4. Load address Low byte using programming instruction 4c.
5. Load data using programming instructions 4d and 4e.
6. Repeat steps 4 and 5 for all data bytes in the page.
7. Write the data using programming instruction 4f.
8. Poll for EEPROM write complete using programming instruction 4g, or wait for $t_{WLRH}$ (refer to Table 119 on page 270).
9. Repeat steps 3 to 8 until all data have been programmed.

Note that the PROG_PAGELOAD instruction can not be used when programming the EEPROM.

**Reading the EEPROM**

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable EEPROM read using programming instruction 5a.
3. Load address using programming instructions 5b and 5c.
4. Read data using programming instruction 5d.
5. Repeat steps 3 and 4 until all data have been read.

Note that the PROG_PAGEREAD instruction can not be used when reading the EEPROM.

**Programming the Fuses**

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Fuse write using programming instruction 6a.
3. Load data high byte using programming instructions 6b. A bit value of "0" will program the corresponding fuse, a "1" will unprogram the fuse.
4. Write Fuse High byte using programming instruction 6c.
5. Poll for Fuse write complete using programming instruction 6d, or wait for $t_{WLRH}$ (refer to Table 119 on page 270).
6. Load data low byte using programming instructions 6e. A "0" will program the fuse, a "1" will unprogram the fuse.
7. Write Fuse low byte using programming instruction 6f.
8. Poll for Fuse write complete using programming instruction 6g, or wait for $t_{WLRH}$ (refer to Table 119 on page 270).

**Programming the Lock Bits**

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Lock bit write using programming instruction 7a.
3. Load data using programming instructions 7b. A bit value of “0” will program the corresponding lock bit, a “1” will leave the lock bit unchanged.
4. Write Lock bits using programming instruction 7c.
5. Poll for Lock bit write complete using programming instruction 7d, or wait for $t_{WLRH}$ (refer to Table 119 on page 270).

**Reading the Fuses and Lock Bits**

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Fuse/Lock bit read using programming instruction 8a.
3. To read all Fuses and Lock bits, use programming instruction 8e.
   To only read Fuse High byte, use programming instruction 8b.
   To only read Fuse Low byte, use programming instruction 8c.
   To only read Lock bits, use programming instruction 8d.

**Reading the Signature Bytes**

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Signature byte read using programming instruction 9a.
3. Load address 0x00 using programming instruction 9b.
4. Read first signature byte using programming instruction 9c.
5. Repeat steps 3 and 4 with address 0x01 and address 0x02 to read the second and third signature bytes, respectively.

**Reading the Calibration Byte**

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Calibration byte read using programming instruction 10a.
3. Load address 0x00 using programming instruction 10b.

Read the calibration byte using programming instruction 10c.

# Electrical Characteristics

## Absolute Maximum Ratings*

| | |
|---|---|
| Operating Temperature | -55°C to +125°C |
| Storage Temperature | -65°C to +150°C |
| Voltage on any Pin except $\overline{\text{RESET}}$ with respect to Ground | -0.5V to $V_{CC}$+0.5V |
| Voltage on $\overline{\text{RESET}}$ with respect to Ground | -0.5V to +13.0V |
| Maximum Operating Voltage | 6.0V |
| DC Current per I/O Pin | 40.0 mA |
| DC Current $V_{CC}$ and GND Pins | 200.0 mA |

*NOTICE: 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 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.

## DC Characteristics

$T_A$ = -40°C to 85°C, $V_{CC}$ = 1.8V to 5.5V (unless otherwise noted)

| Symbol | Parameter | Condition | Min. | Typ. | Max. | Units |
|---|---|---|---|---|---|---|
| $V_{IL}$ | Input Low Voltage, Except XTAL1 pin | $V_{CC}$ = 1.8V - 2.4V<br>$V_{CC}$ = 2.4V - 5.5V | -0.5<br>-0.5 | | $0.2V_{CC}$(1)<br>$0.3V_{CC}$(1) | V |
| $V_{IL1}$ | Input Low Voltage, XTAL1 pin | $V_{CC}$ = 1.8V - 5.5V | -0.5 | | $0.1V_{CC}$(1) | V |
| $V_{IH}$ | Input High Voltage, Except XTAL1 and RESET pins | $V_{CC}$ = 1.8V - 2.4V<br>$V_{CC}$ = 2.4V - 5.5V | $0.7V_{CC}$(2)<br>$0.6V_{CC}$(2) | | $V_{CC}$ + 0.5<br>$V_{CC}$ + 0.5 | V |
| $V_{IH1}$ | Input High Voltage, XTAL1 pin | $V_{CC}$ = 1.8V - 2.4V<br>$V_{CC}$ = 2.4V - 5.5V | $0.8V_{CC}$(2)<br>$0.7V_{CC}$(2) | | $V_{CC}$ + 0.5<br>$V_{CC}$ + 0.5 | V |
| $V_{IH2}$ | Input High Voltage, RESET pin | $V_{CC}$ = 1.8V - 5.5V | $0.85V_{CC}$(2) | | $V_{CC}$ + 0.5 | V |
| $V_{OL}$ | Output Low Voltage(3), Port A, C, D, E, F, G, H, J | $I_{OL}$ = 10mA, $V_{CC}$ = 5V<br>$I_{OL}$ = 5mA, $V_{CC}$ = 3V | | | 0.7<br>0.5 | V |
| $V_{OL1}$ | Output Low Voltage(3), Port B | $I_{OL}$ = 20mA, $V_{CC}$ = 5V<br>$I_{OL}$ = 10mA, $V_{CC}$ = 3V | | | 0.7<br>0.5 | V |
| $V_{OH}$ | Output High Voltage(4), Port A, C, D, E, F, G, H, J | $I_{OH}$ = -10mA, $V_{CC}$ = 5V<br>$I_{OH}$ = -5mA, $V_{CC}$ = 3V | 4.2<br>2.3 | | | V |
| $V_{OH1}$ | Output High Voltage(4), Port B | $I_{OH}$ = -20mA, $V_{CC}$ = 5V<br>$I_{OH}$ = -10mA, $V_{CC}$ = 3V | 4.2<br>2.3 | | | V |
| $I_{IL}$ | Input Leakage Current I/O Pin | $V_{CC}$ = 5.5V, pin low (absolute value) | | | 1 | µA |
| $I_{IH}$ | Input Leakage Current I/O Pin | $V_{CC}$ = 5.5V, pin high (absolute value) | | | 1 | µA |
| $R_{RST}$ | Reset Pull-up Resistor | | 20 | | 100 | kΩ |
| $R_{PU}$ | I/O Pin Pull-up Resistor | | 20 | | 100 | kΩ |

$T_A$ = -40°C to 85°C, $V_{CC}$ = 1.8V to 5.5V (unless otherwise noted) (Continued)

| Symbol | Parameter | Condition | Min. | Typ. | Max. | Units |
|---|---|---|---|---|---|---|
| $I_{CC}$ | Power Supply Current | Active 1MHz, $V_{CC}$ = 2V (ATmega325/3250/645/6450V) | | | 0.55 | mA |
| | | Active 4MHz, $V_{CC}$ = 3V (ATmega325/3250/645/6450V) | | | 2.5 | mA |
| | | Active 8MHz, $V_{CC}$ = 5V (ATmega325/3250/645/6450) | | | 9 | mA |
| | | Idle 1MHz, $V_{CC}$ = 2V (ATmega325/3250/645/6450V) | | | 0.2 | mA |
| | | Idle 4MHz, $V_{CC}$ = 3V (ATmega325/3250/645/6450V) | | | 0.85 | mA |
| | | Idle 8MHz, $V_{CC}$ = 5V (ATmega325/3250/645/6450) | | | 3 | mA |
| | Power-down mode | WDT enabled, $V_{CC}$ = 3V | | <8 | 15 | µA |
| | | WDT disabled, $V_{CC}$ = 3V | | <1 | 5 | µA |
| $V_{ACIO}$ | Analog Comparator Input Offset Voltage | $V_{CC}$ = 5V<br>$V_{in}$ = $V_{CC}$/2 | | <10 | 40 | mV |
| $I_{ACLK}$ | Analog Comparator Input Leakage Current | $V_{CC}$ = 5V<br>$V_{in}$ = $V_{CC}$/2 | -50 | | 50 | nA |
| $t_{ACID}$ | Analog Comparator Propagation Delay | $V_{CC}$ = 2.7V<br>$V_{CC}$ = 4.0V | | 750<br>500 | | ns |

Note: 1. "Max" means the highest value where the pin is guaranteed to be read as low

2. "Min" means the lowest value where the pin is guaranteed to be read as high

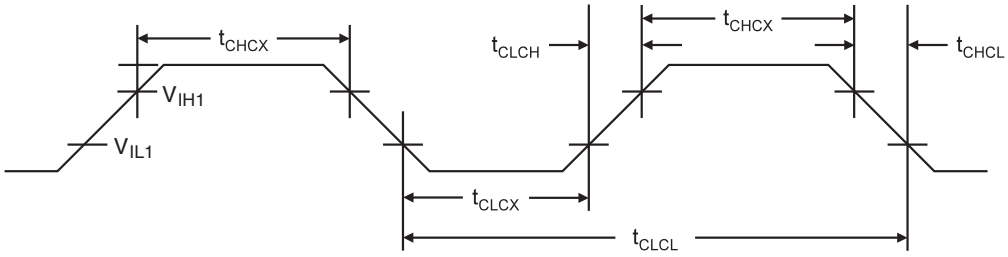
3. Although each I/O port can sink more than the test conditions (20 mA at $V_{CC}$ = 5V, 10 mA at $V_{CC}$ = 3V for Port B and 10 mA at $V_{CC}$ = 5V, 5 mA at $V_{CC}$ = 3V for all other ports) under steady state conditions (non-transient), the following must be observed:
   TQFP and MLF Package:
   1] The sum of all IOL, for all ports, should not exceed 400 mA.
   2] The sum of all IOL, for ports A0 - A7, C4 - C7, G2 should not exceed 100 mA.
   3] The sum of all IOL, for ports B0 - B7, E0 - E7, G3 - G5 should not exceed 100 mA.
   4] The sum of all IOL, for ports D0 - D7, C0 - C3, G0 - G1 should not exceed 100 mA.
   5] The sum of all IOL, for ports F0 - F7, should not exceed 100 mA.
   If IOL exceeds the test condition, VOL may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test condition.

4. Although each I/O port can source more than the test conditions (20 mA at $V_{CC}$ = 5V, 10 mA at $V_{CC}$ = 3V for Port B and 10mA at $V_{CC}$ = 5V, 5 mA at $V_{CC}$ = 3V for all other ports) under steady state conditions (non-transient), the following must be observed:
   TQFP and MLF Package:
   1] The sum of all IOL, for all ports, should not exceed 400 mA.
   2] The sum of all IOL, for ports A0 - A7, C4 - C7, G2 should not exceed 100 mA.
   3] The sum of all IOL, for ports B0 - B7, E0 - E7, G3 - G5 should not exceed 100 mA.
   4] The sum of all IOL, for ports D0 - D7, C0 - C3, G0 - G1 should not exceed 100 mA.
   5] The sum of all IOL, for ports F0 - F7, should not exceed 100 mA.

If IOH exceeds the test condition, VOH may exceed the related specification. Pins are not guaranteed to source current greater than the listed test condition.

## External Clock Drive Waveforms

**Figure 127.** External Clock Drive Waveforms




## External Clock Drive

**Table 123.** External Clock Drive

| Symbol | Parameter | $V_{CC}$=1.8-5.5V | | $V_{CC}$=2.7-5.5V | | $V_{CC}$=4.5-5.5V | | Units |
|---|---|---|---|---|---|---|---|---|
| | | Min. | Max. | Min. | Max. | Min. | Max. | |
| $1/t_{CLCL}$ | Oscillator Frequency | 0 | 1 | 0 | 8 | 0 | 16 | MHz |
| $t_{CLCL}$ | Clock Period | 1000 | | 125 | | 62.5 | | ns |
| $t_{CHCX}$ | High Time | 400 | | 50 | | 25 | | ns |
| $t_{CLCX}$ | Low Time | 400 | | 50 | | 25 | | ns |
| $t_{CLCH}$ | Rise Time | | 2.0 | | 1.6 | | 0.5 | µs |
| $t_{CHCL}$ | Fall Time | | 2.0 | | 1.6 | | 0.5 | µs |
| $\Delta t_{CLCL}$ | Change in period from one clock cycle to the next | | 2 | | 2 | | 2 | % |

## Speed vs. $V_{CC}$

**Figure 128.** Maximum Frequency vs. $V_{CC}$ (4 - 8 MHz).

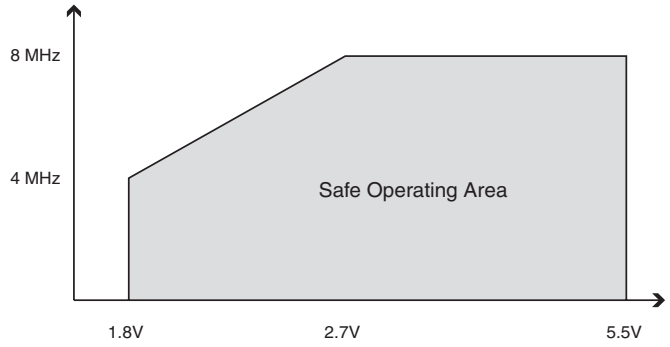
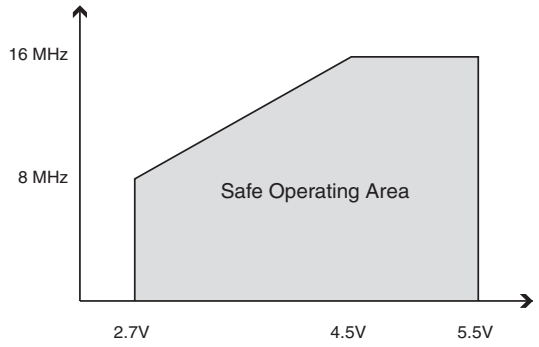


**Figure 129.** Maximum Frequency vs. $V_{CC}$ (8 - 16 MHz).




## SPI Timing Characteristics

See Figure 130 and Figure 131 for details.

**Table 124.** SPI Timing Parameters

| | Description | Mode | Min | Typ | Max | |
|---|---|---|---|---|---|---|
| 1 | SCK period | Master | | See Table 67 | | |
| 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 • $t_{sck}$ | | ns |
| 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 • $t_{ck}$ | | | |
| 11 | SCK high/low(1) | Slave | 2 • $t_{ck}$ | | | |
| 12 | Rise/Fall time | Slave | | 1.6 | | µs |
| 13 | Setup | Slave | 10 | | | |
| 14 | Hold | Slave | $t_{ck}$ | | | |
| 15 | SCK to out | Slave | | 15 | | |
| 16 | SCK to $\overline{SS}$ high | Slave | 20 | | | ns |
| 17 | $\overline{SS}$ high to tri-state | Slave | | 10 | | |
| 18 | $\overline{SS}$ low to SCK | Slave | 20 • $t_{ck}$ | | | |

Note: 1. In SPI Programming mode the minimum SCK high/low period is:
- 2 $t_{CLCL}$ for $f_{CK}$ < 12 MHz
- 3 $t_{CLCL}$ for $f_{CK}$ > 12 MHz

**Figure 130.** SPI Interface Timing Requirements (Master Mode)

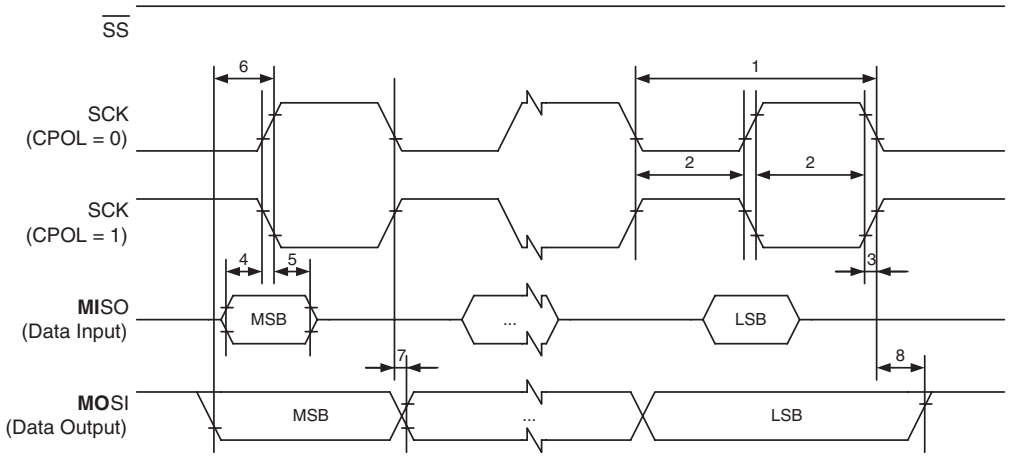
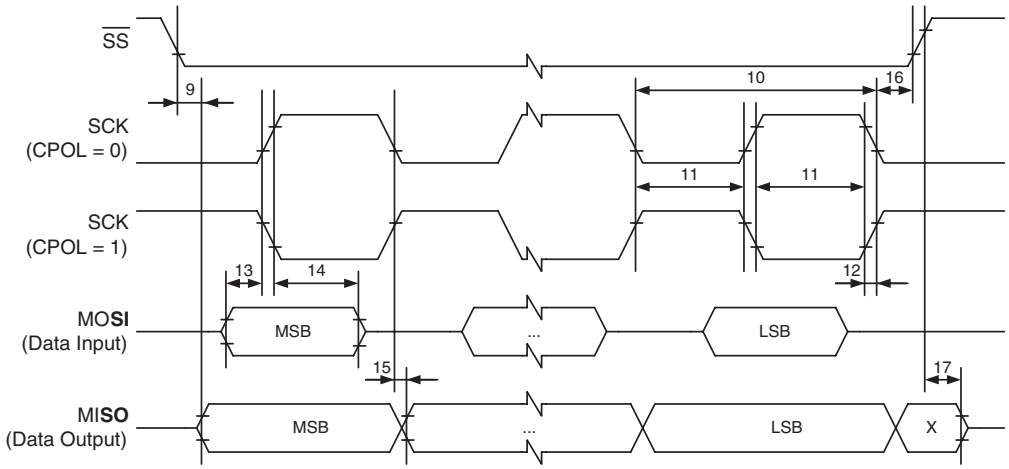



**Figure 131.** SPI Interface Timing Requirements (Slave Mode)



## ADC Characteristics – Preliminary Data

**Table 125.** ADC Characteristics

| Symbol | Parameter | Condition | Min | Typ | Max | Units |
|---|---|---|---|---|---|---|
| | Resolution | Single Ended Conversion | | 10 | | Bits |
| | | Differential Conversion | | 8 | | Bits |
| | Absolute accuracy (Including INL, DNL, quantization error, gain and offset error) | Single Ended Conversion $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200 kHz | | 2 | 2.5 | LSB |
| | | Single Ended Conversion $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 1 MHz | | 4.5 | | LSB |
| | | Single Ended Conversion $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200 kHz Noise Reduction Mode | | 2 | | LSB |
| | | Single Ended Conversion $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 1 MHz Noise Reduction Mode | | 4.5 | | LSB |
| | Integral Non-Linearity (INL) | Single Ended Conversion $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200 kHz | | 0.5 | | LSB |
| | Differential Non-Linearity (DNL) | Single Ended Conversion $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200 kHz | | 0.25 | | LSB |
| | Gain Error | Single Ended Conversion $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200 kHz | | 2 | | LSB |
| | Offset Error | Single Ended Conversion $V_{REF}$ = 4V, $V_{CC}$ = 4V, ADC clock = 200 kHz | | 2 | | LSB |
| | Conversion Time | Free Running Conversion | 13 | | 260 | µs |
| | Clock Frequency | Single Ended Conversion | 50 | | 1000 | kHz |
| AVCC | Analog Supply Voltage | | $V_{CC}$ - 0.3 | | $V_{CC}$ + 0.3 | V |
| $V_{REF}$ | Reference Voltage | Single Ended Conversion | 1.0 | | AVCC | V |
| | | Differential Conversion | 1.0 | | AVCC - 0.5 | V |
| $V_{IN}$ | Input Voltage | Single ended channels | GND | | $V_{REF}$ | V |
| | | Differential Conversion | 0 | | AVCC[1] | V |
| | Input Bandwidth | Single Ended Channels | | 38,5 | | kHz |
| | | Differential Channels | | 4 | | kHz |
| $V_{INT}$ | Internal Voltage Reference | | 1.0 | 1.1 | 1.2 | V |
| $R_{REF}$ | Reference Input Resistance | | | 32 | | kΩ |
| $R_{AIN}$ | Analog Input Resistance | | | 100 | | MΩ |

Note: 1. $V_{DIFF}$ must be below $V_{REF}$

## ATmega325/3250/645/6450 Typical Characteristics – Preliminary Data

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.

All Active- and Idle current consumption measurements are done with all bits in the PRR register set and thus, the corresponding I/O modules are turned off. Also the Analog Comparator is disabled during these measurements. See “Power Reduction Register” on page 35 for details.

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 $C_L*V_{CC}*f$ 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.

### Active Supply Current

**Figure 132.** Active Supply Current vs. Frequency (0.1 - 1.0 MHz)

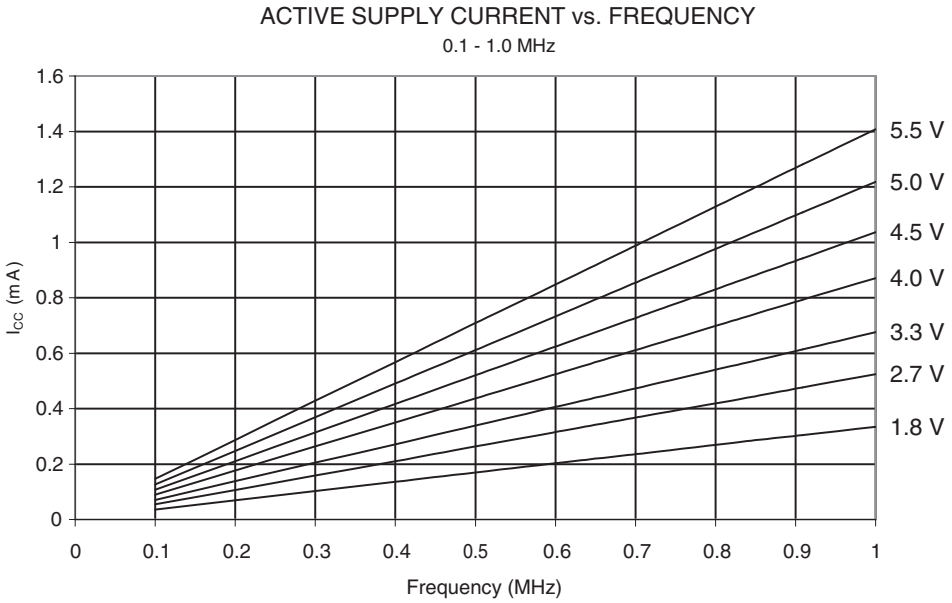


**Figure 133.** Active Supply Current vs. Frequency (1 - 16 MHz))

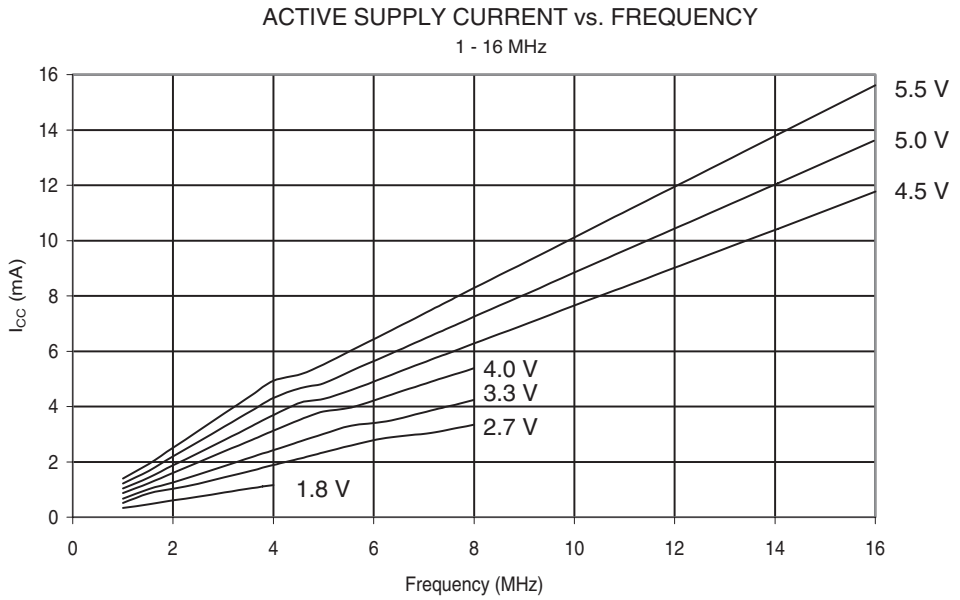



**Figure 134.** Active Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 8 MHz)

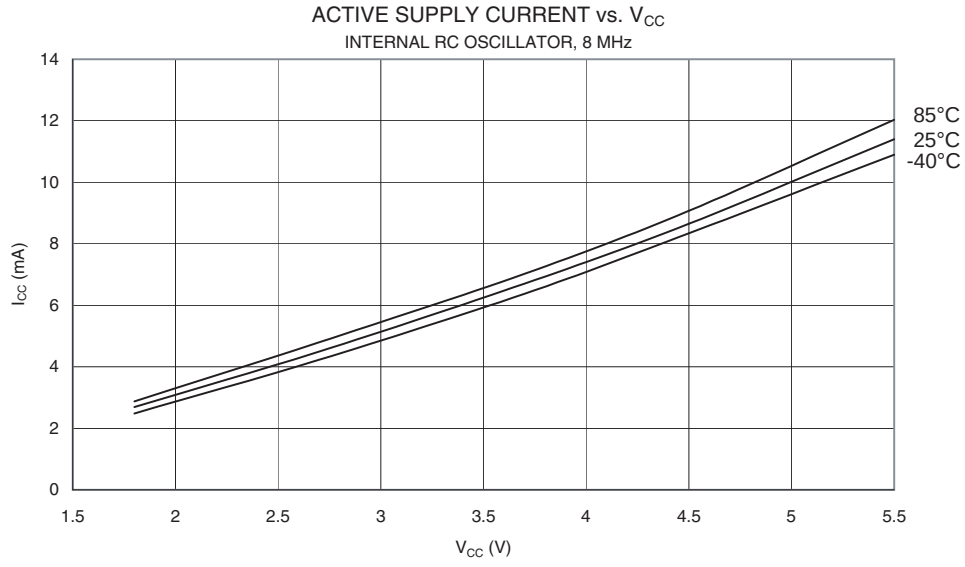


**Figure 135.** Active Supply Current vs. $V_{CC}$ (Internal RC Oscillator, CKDIV8 Programmed, 1 MHz)

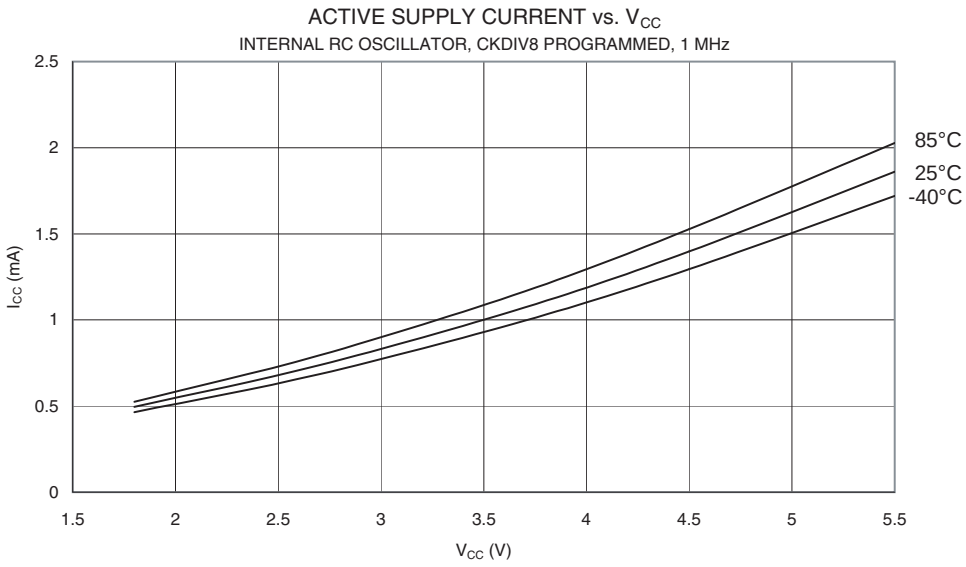
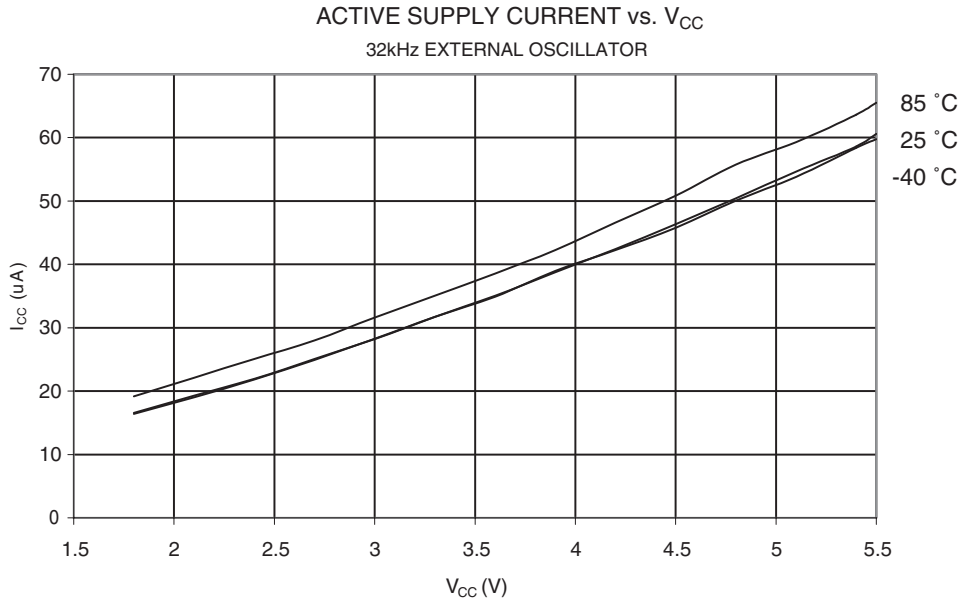



**Figure 136.** Active Supply Current vs. $V_{CC}$ (32 kHz External Oscillator)



### Idle Supply Current

**Figure 137.** Idle Supply Current vs. Frequency (0.1 - 1.0 MHz)

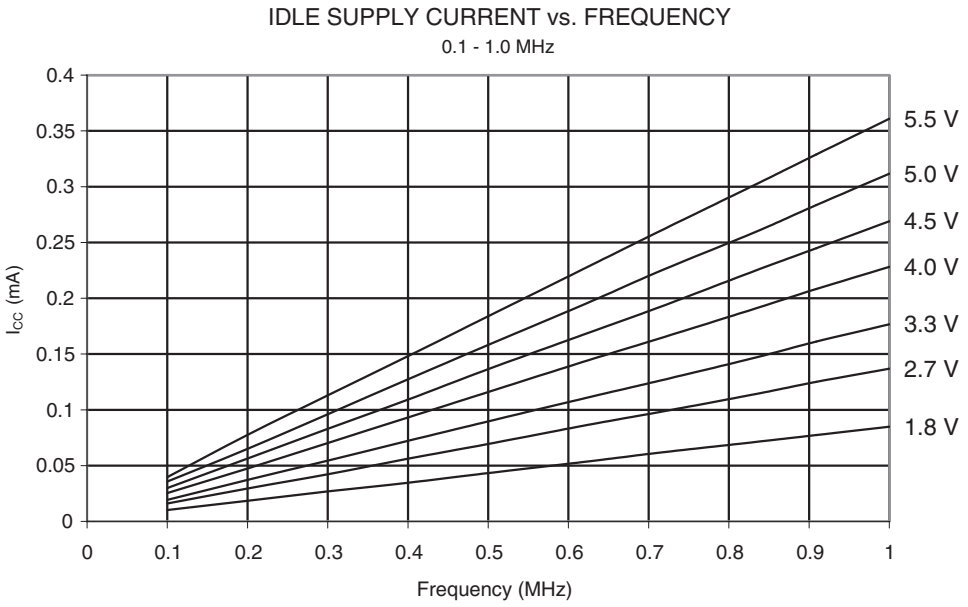



**Figure 138.** Idle Supply Current vs. Frequency (1 - 16 MHz)

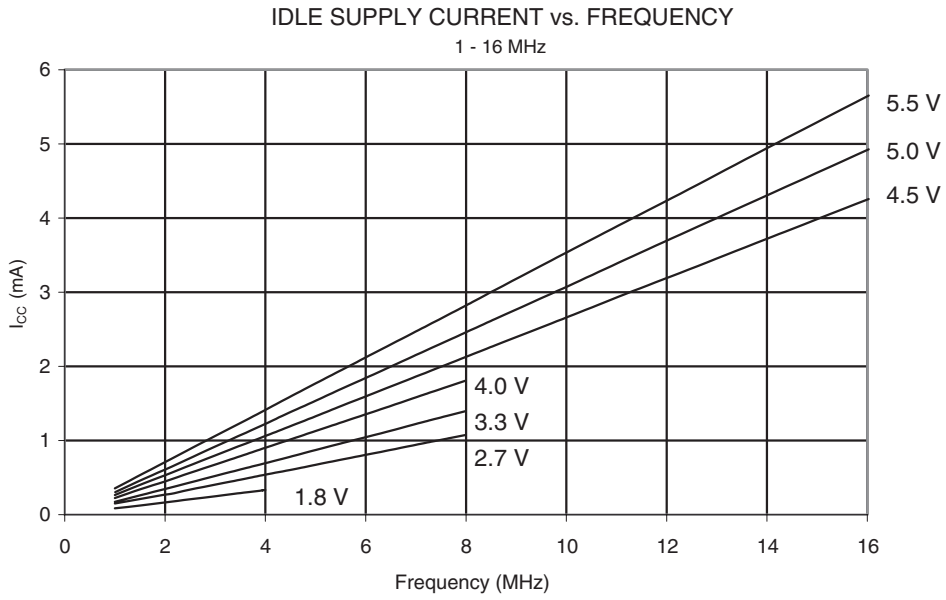


**Figure 139.** Idle Supply Current vs. $V_{CC}$ (Internal RC Oscillator, 8 MHz)

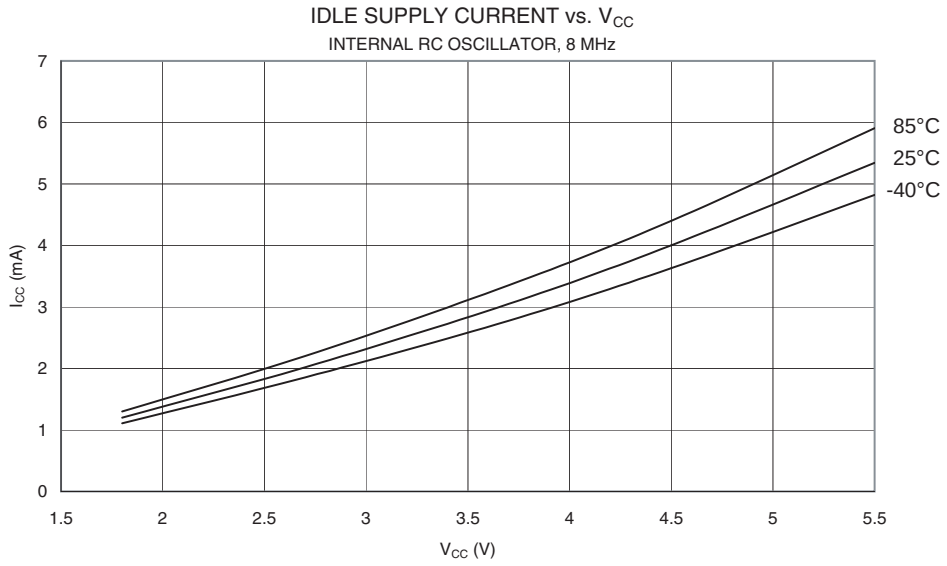



**Figure 140.** Idle Supply Current vs. $V_{CC}$ (Internal RC Oscillator, CKDIV8 Programmed, 1 MHz)

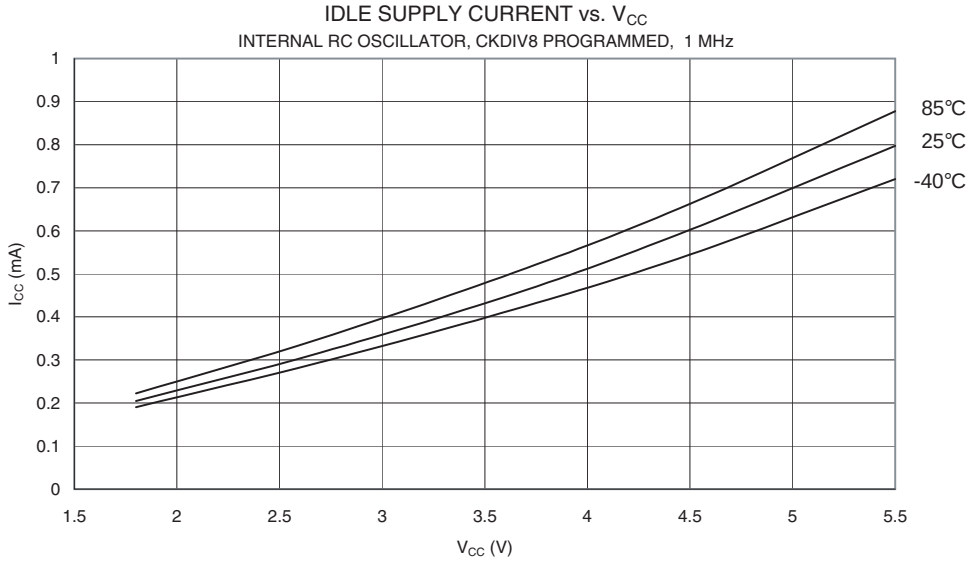
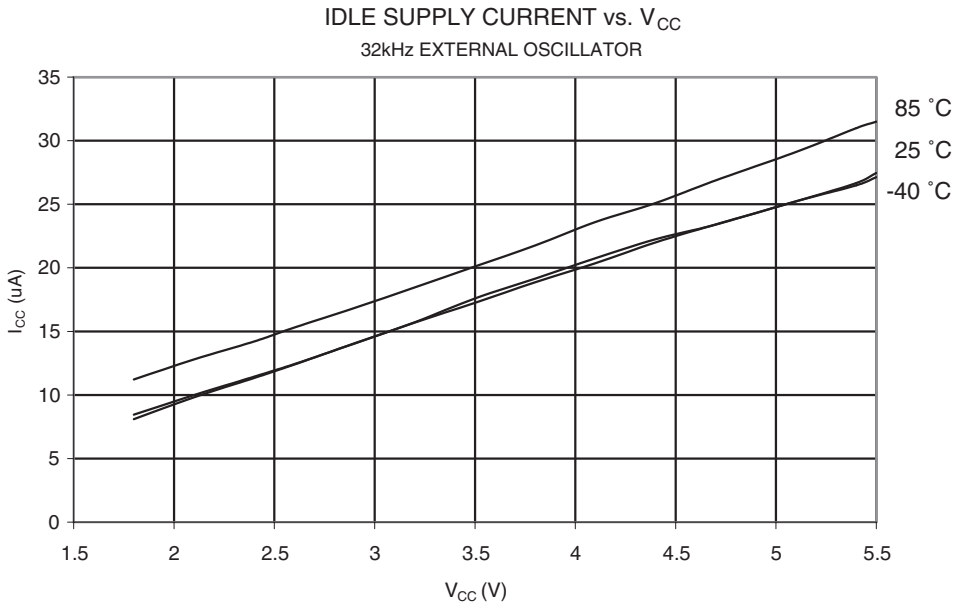


**Figure 141.** Idle Supply Current vs. $V_{CC}$ (32 kHz External Oscillator)




## Power-down Supply Current

**Figure 142.** Power-down Supply Current vs. $V_{CC}$ (Watchdog Timer Disabled)

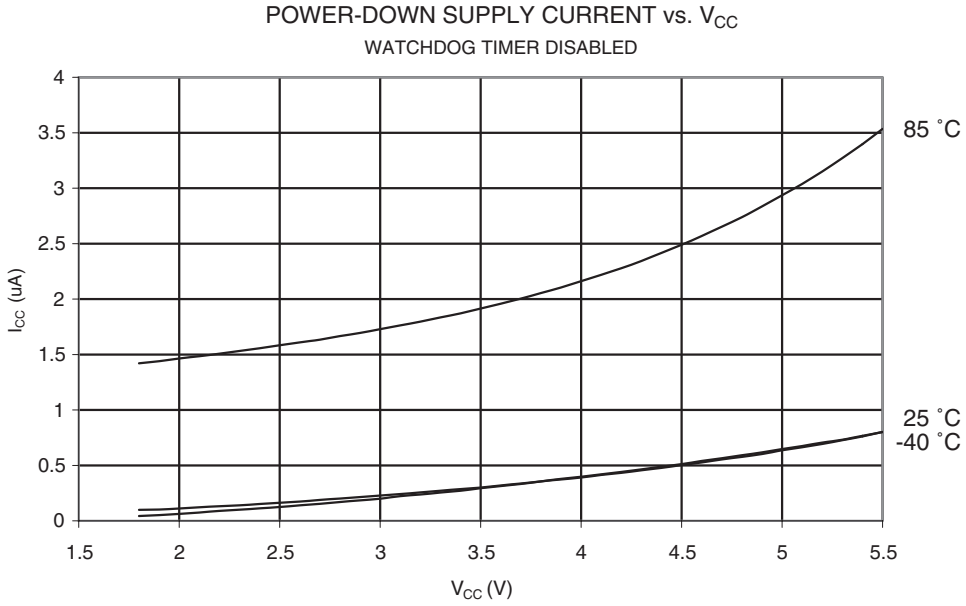
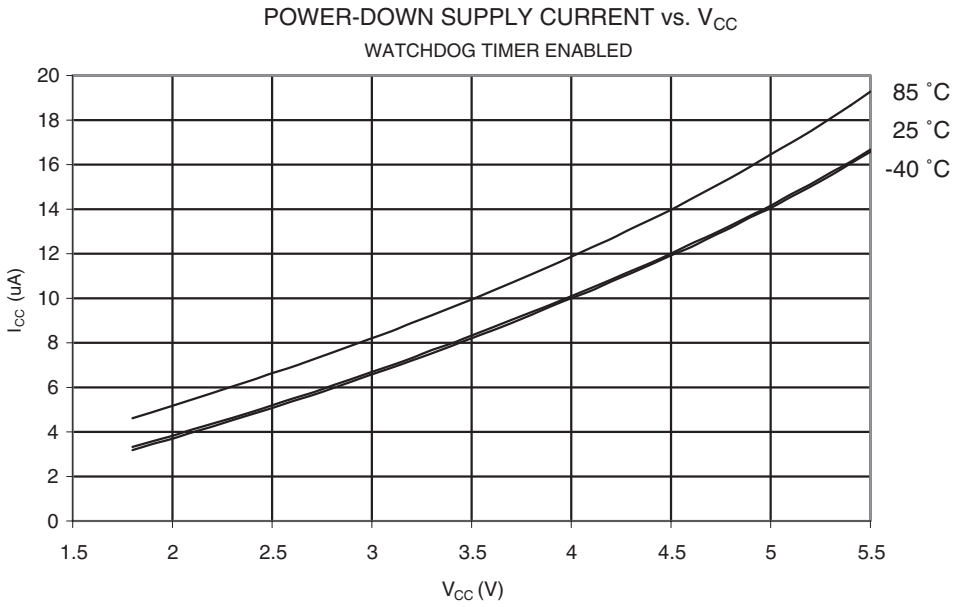
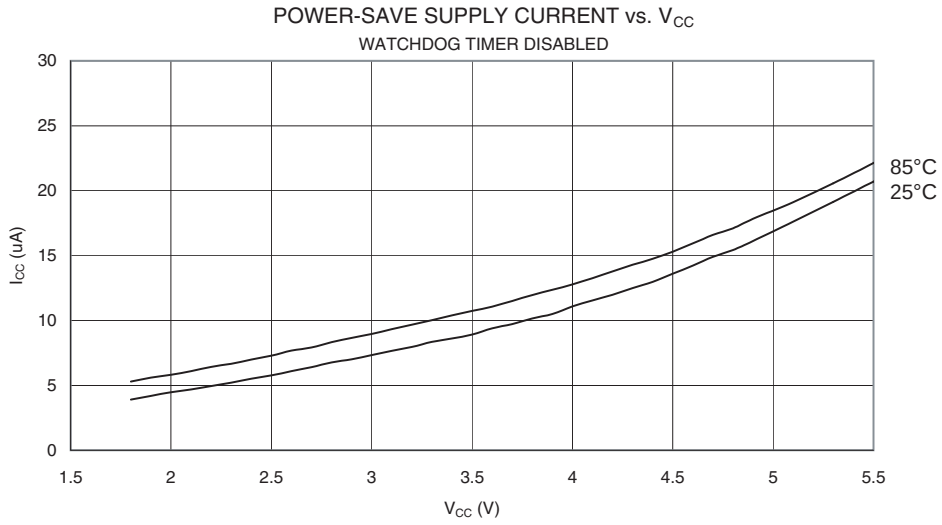


**Figure 143.** Power-down Supply Current vs. $V_{CC}$ (Watchdog Timer Enabled)



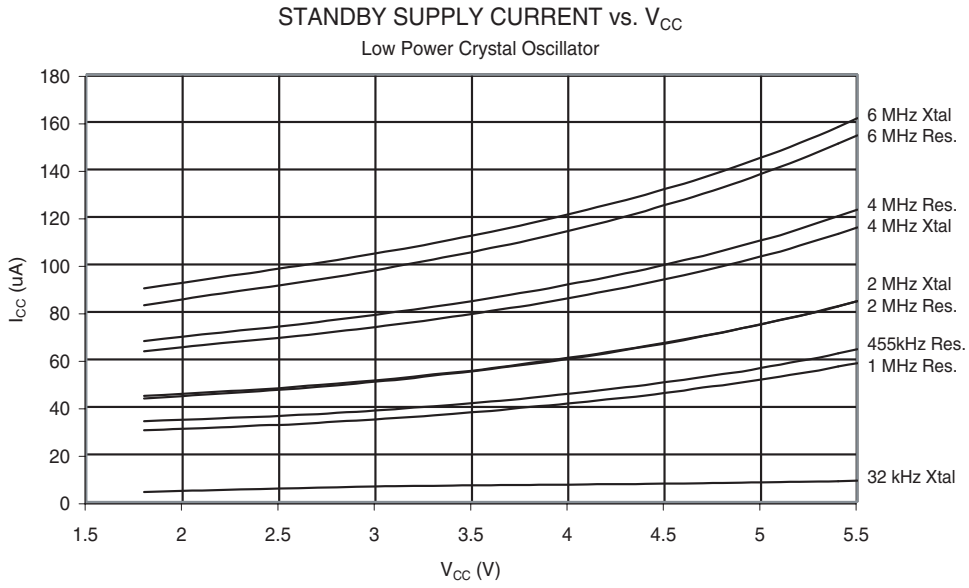

**Power-save Supply Current**

**Figure 144.** Power-save Supply Current vs. $V_{CC}$ (Watchdog Timer Disabled)



**Standby Supply Current**

**Figure 145.** Standby Supply Current vs. $V_{CC}$ (Low Power Crystal Oscillator)




**Pin Pull-up**

**Figure 146.** I/O Pin Pull-up Resistor Current vs. Input Voltage ($V_{CC}$ = 5V)

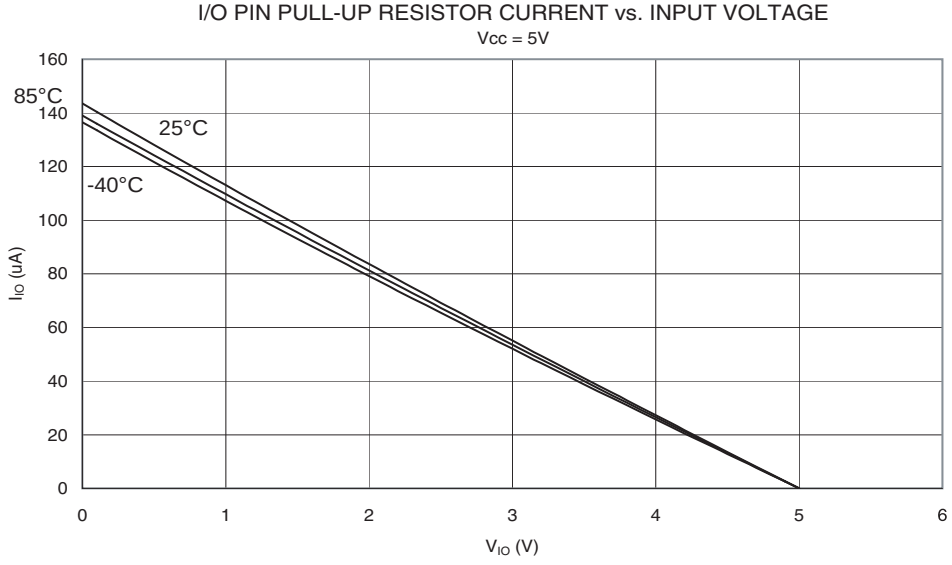


**Figure 147.** I/O Pin Pull-up Resistor Current vs. Input Voltage ($V_{CC}$ = 2.7V)

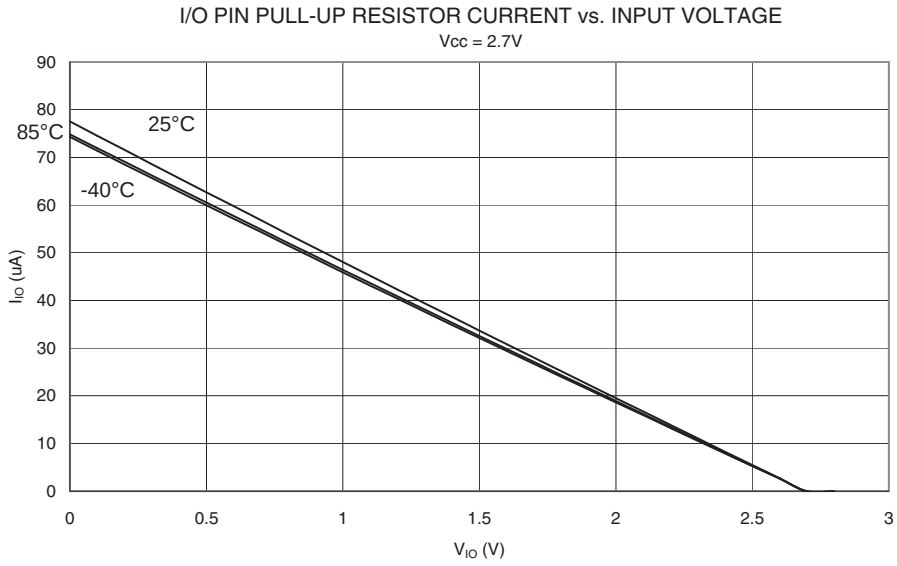



**Figure 148.** I/O Pin Pull-up Resistor Current vs. Input Voltage ($V_{CC}$ = 1.8V)

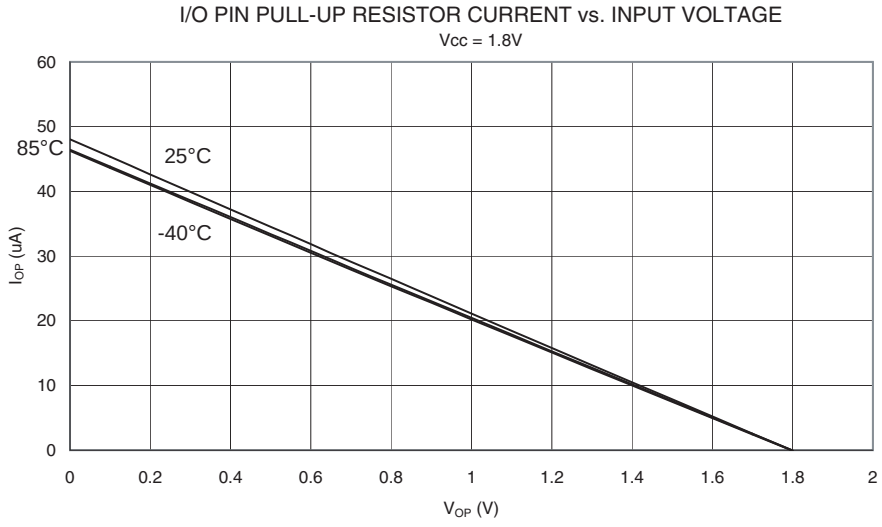


**Figure 149.** Reset Pull-up Resistor Current vs. Reset Pin Voltage ($V_{CC}$ = 5V)

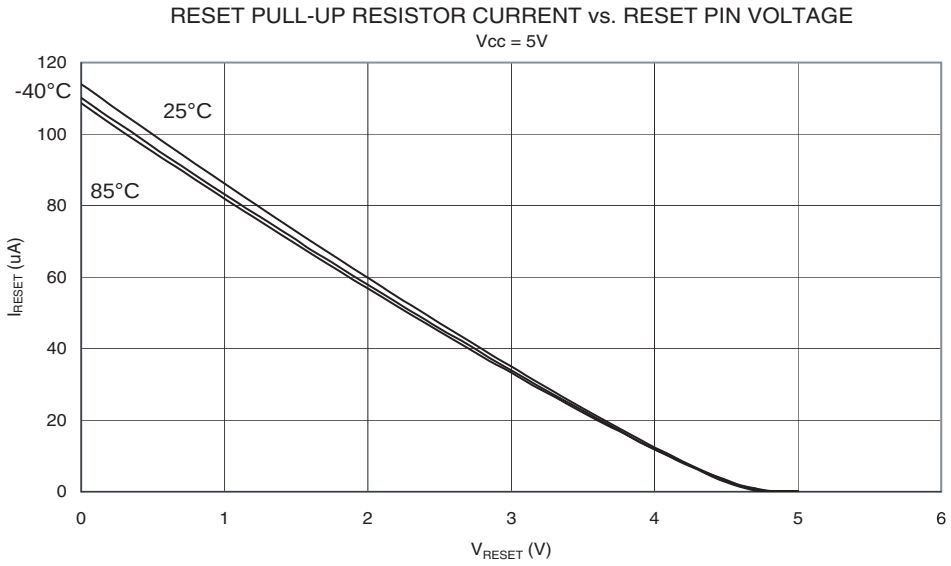



**Figure 150.** Reset Pull-up Resistor Current vs. Reset Pin Voltage ($V_{CC}$ = 2.7V)

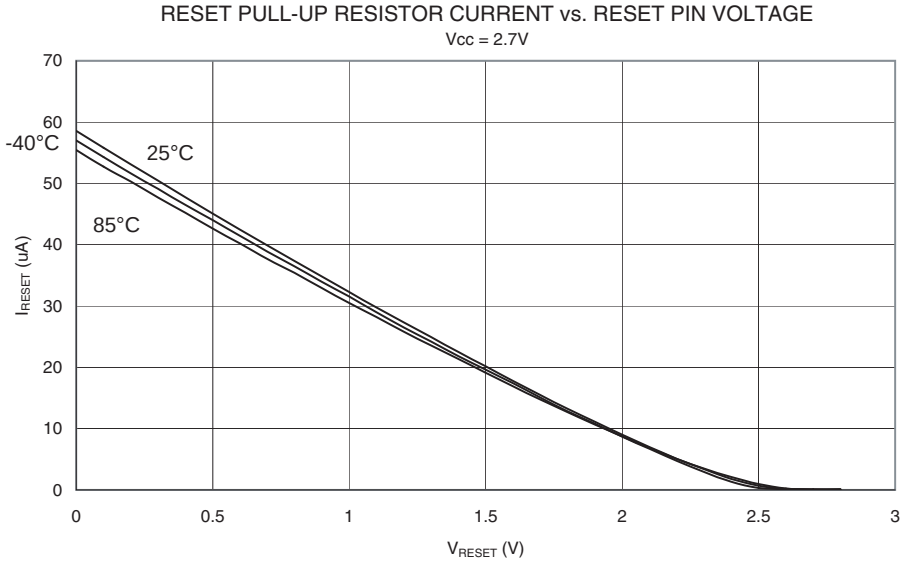
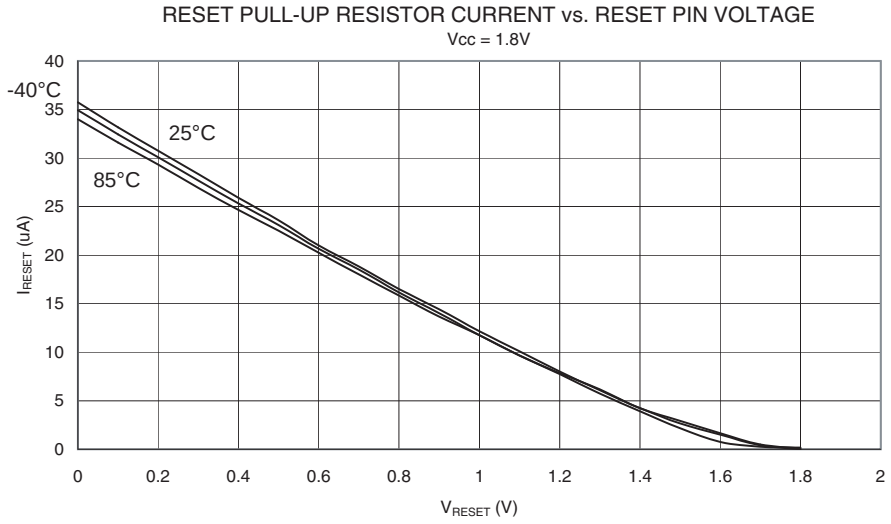


**Figure 151.** Reset Pull-up Resistor Current vs. Reset Pin Voltage ($V_{CC}$ = 1.8V)




## Pin Driver Strength

**Figure 152.** I/O Pin Source Current vs. Output Voltage, Ports A, C, D, E, F, G, H, J ($V_{CC}$ = 5V)

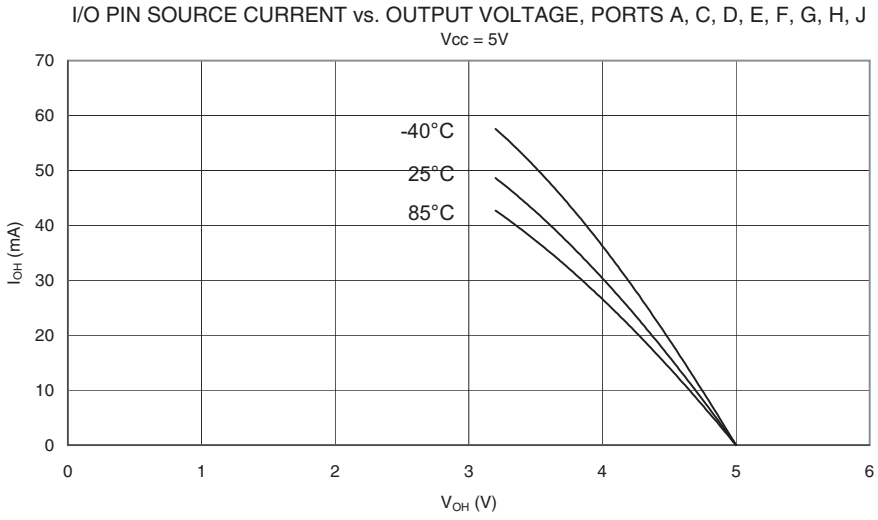


**Figure 153.** I/O Pin Source Current vs. Output Voltage, Ports A, C, D, E, F, G, H, J ($V_{CC}$ = 2.7V)

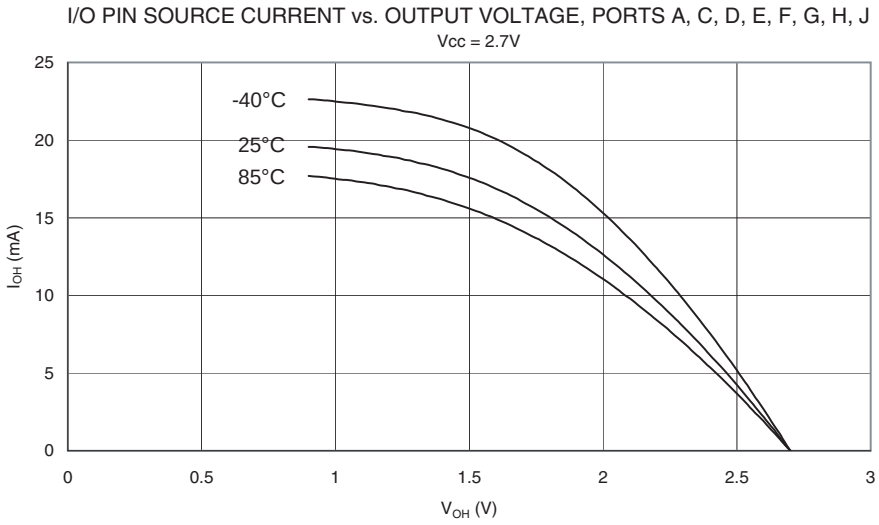



**Figure 154.** I/O Pin Source Current vs. Output Voltage, Ports A, C, D, E, F, G, H, J ($V_{CC}$ = 1.8V)

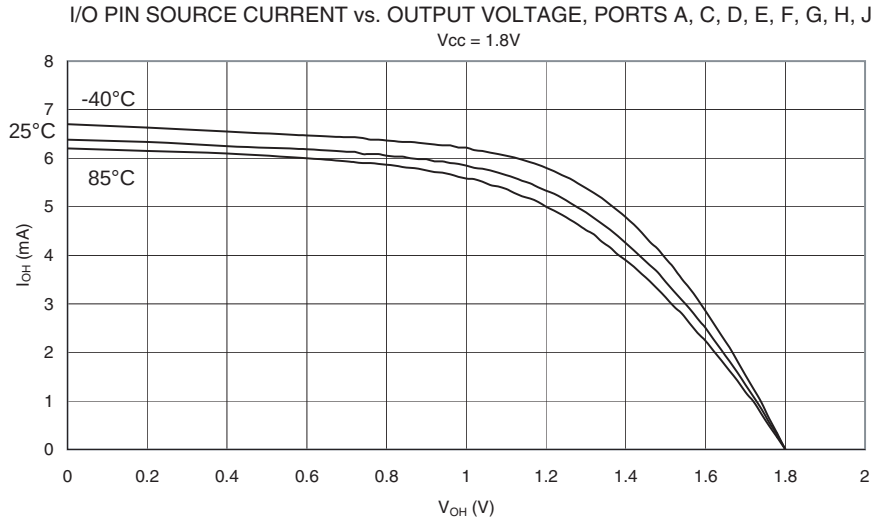


**Figure 155.** I/O Pin Source Current vs. Output Voltage, Port B ($V_{CC}$= 5V)

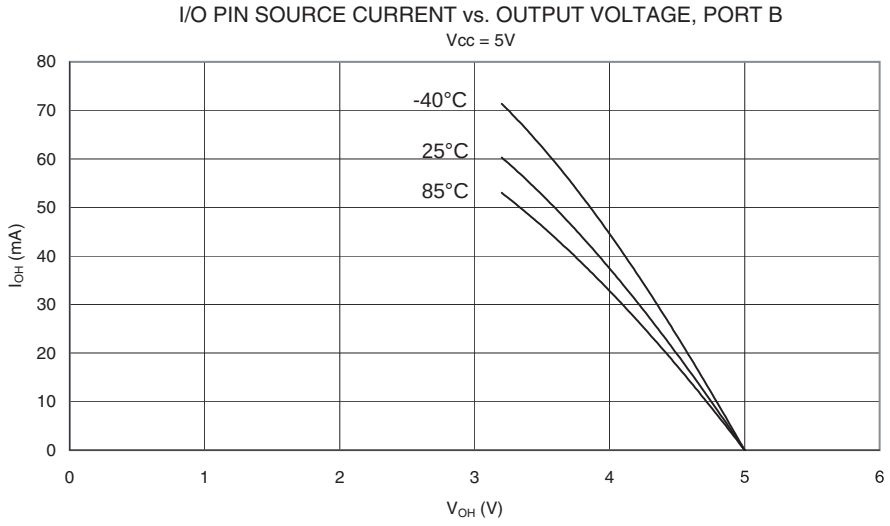



**Figure 156.** I/O Pin Source Current vs. Output Voltage, Port B ($V_{CC}$ = 2.7V)

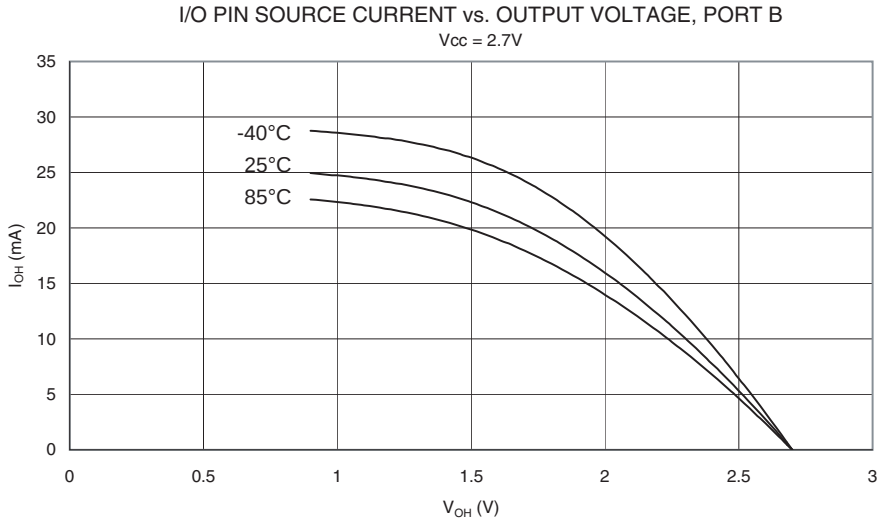


**Figure 157.** I/O Pin Source Current vs. Output Voltage, Port B ($V_{CC}$ = 1.8V)

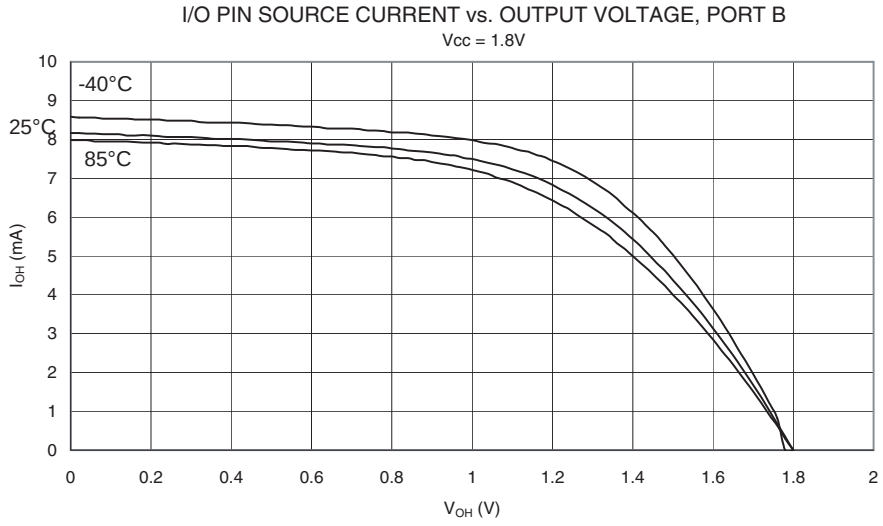



**Figure 158.** I/O Pin Sink Current vs. Output Voltage, Ports A, C, D, E, F, G, H, J ($V_{CC}$ = 5V)

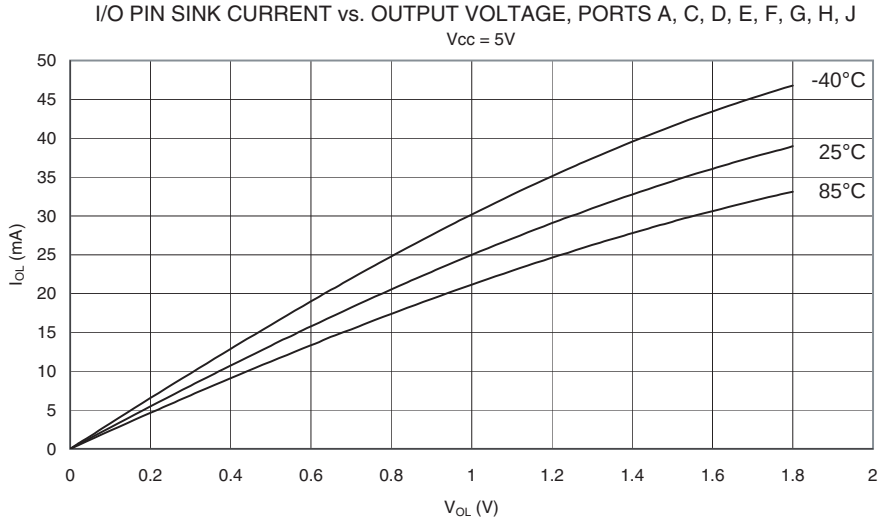


**Figure 159.** I/O Pin Sink Current vs. Output Voltage, Ports A, C, D, E, F, G, H, J ($V_{CC}$ = 2.7V)

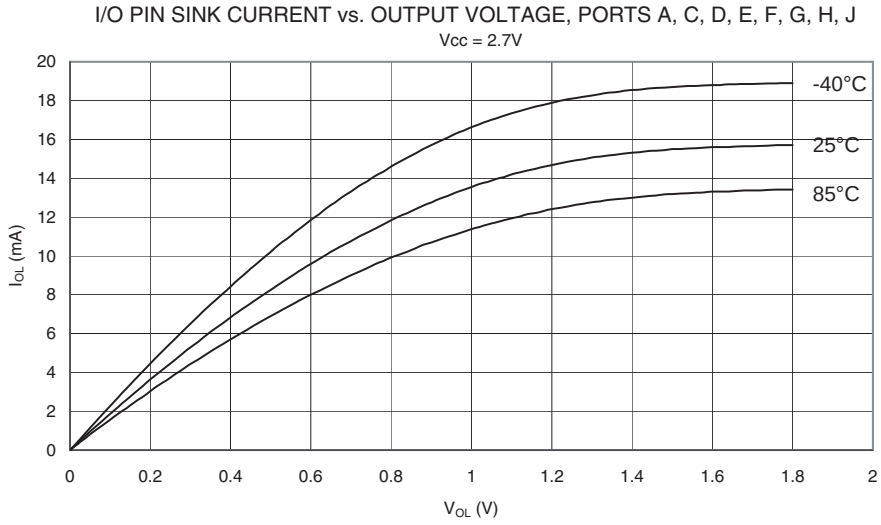



**Figure 160.** I/O Pin Sink Current vs. Output Voltage, Ports A, C, D, E, F, G, H, J ($V_{CC}$ = 1.8V)

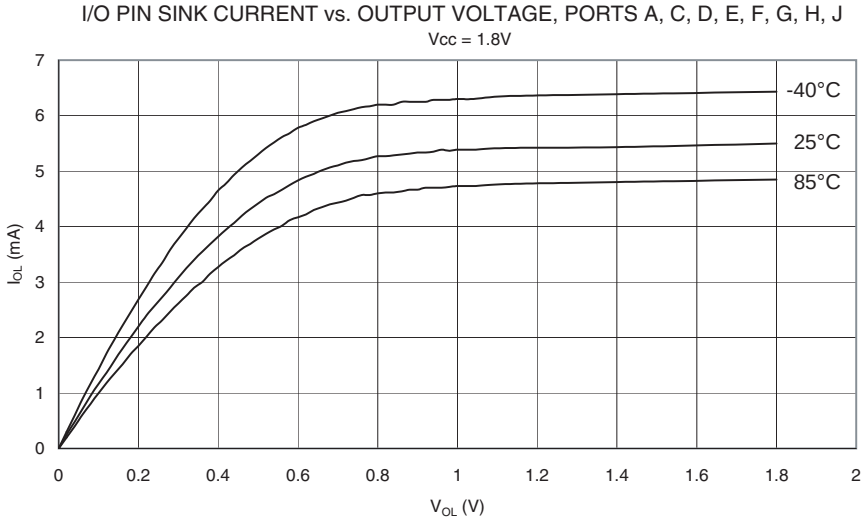


**Figure 161.** I/O Pin Sink Current vs. Output Voltage, Port B ($V_{CC}$ = 5V)

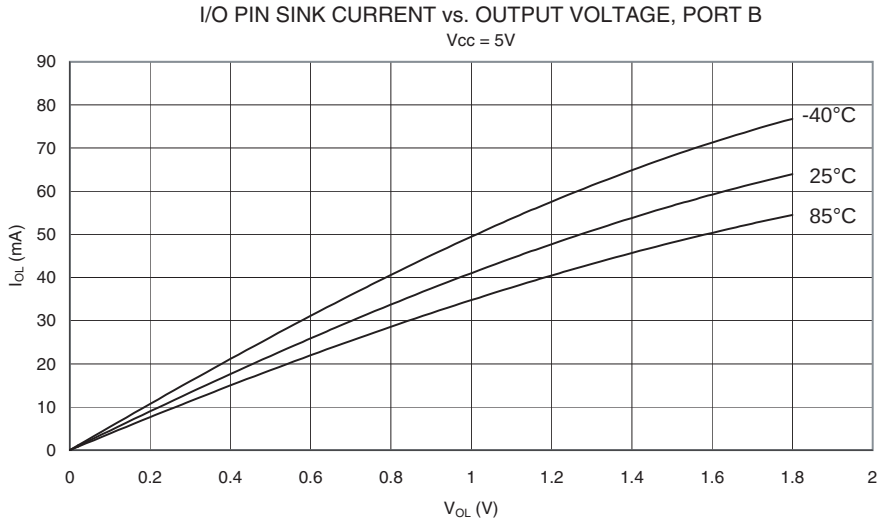



**Figure 162.** I/O Pin Sink Current vs. Output Voltage, Port B ($V_{CC}$ = 2.7V)

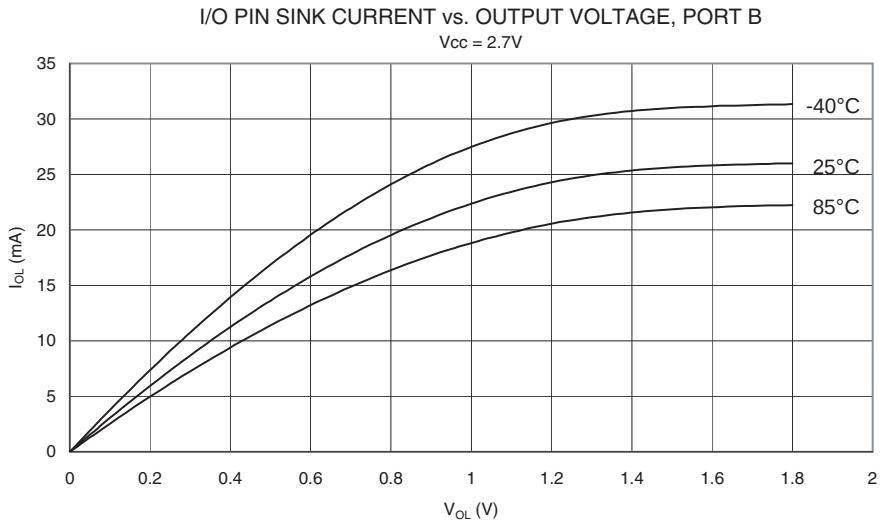
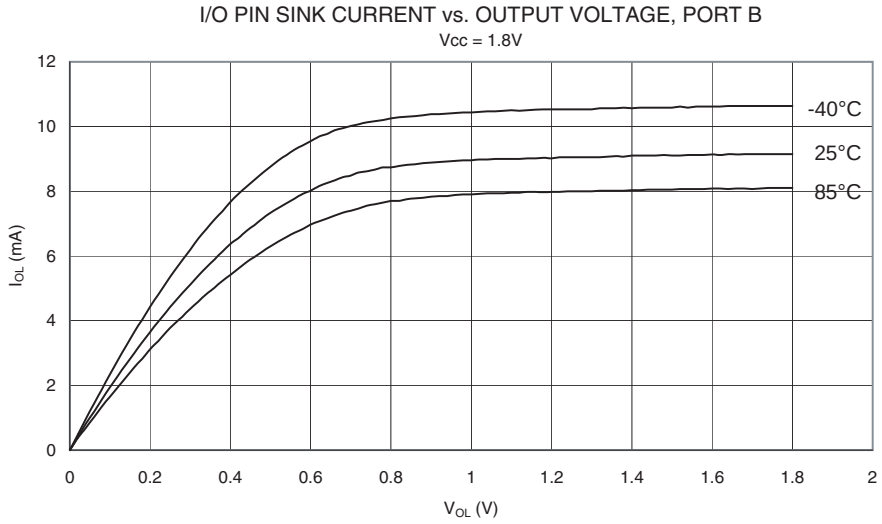


**Figure 163.** I/O Pin Sink Current vs. Output Voltage, Port B ($V_{CC}$ = 1.8V)




**Pin Thresholds and hysteresis**

**Figure 164.** I/O Pin Input Threshold Voltage vs. $V_{CC}$ ($V_{IH}$, I/O Pin Read as “1”)

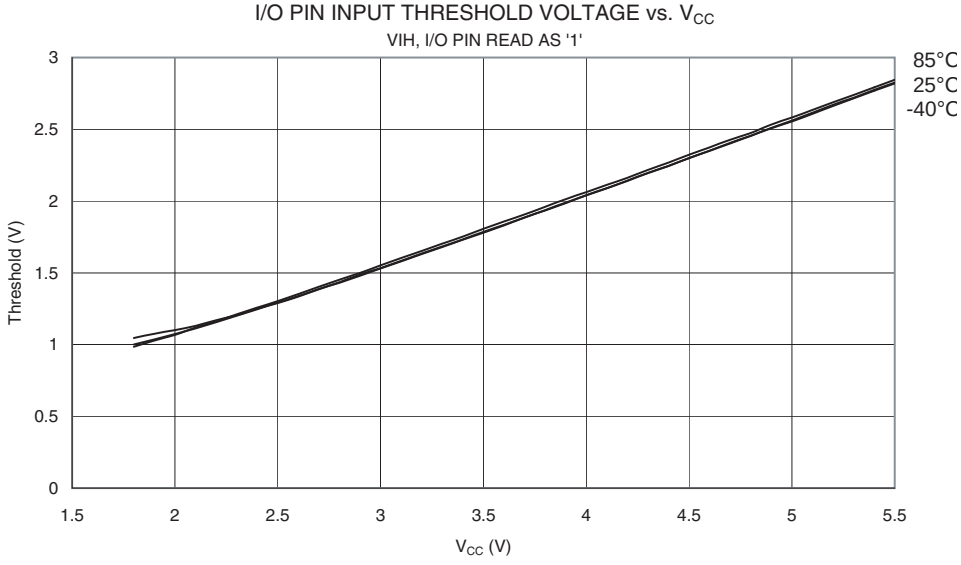


**Figure 165.** I/O Pin Input Threshold Voltage vs. $V_{CC}$ ($V_{IL}$, I/O Pin Read as “0”)

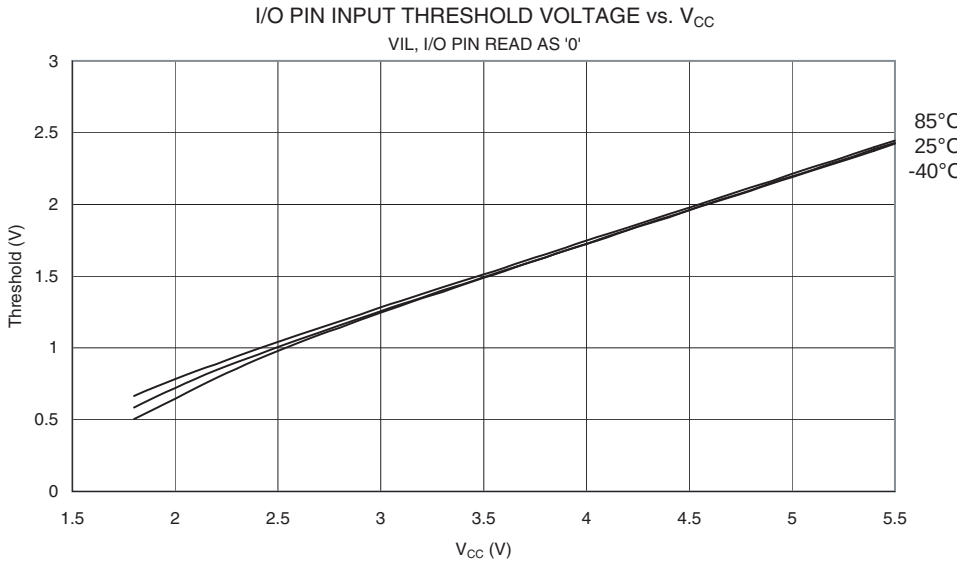



**Figure 166.** I/O Pin Input Hysteresis vs. $V_{CC}$

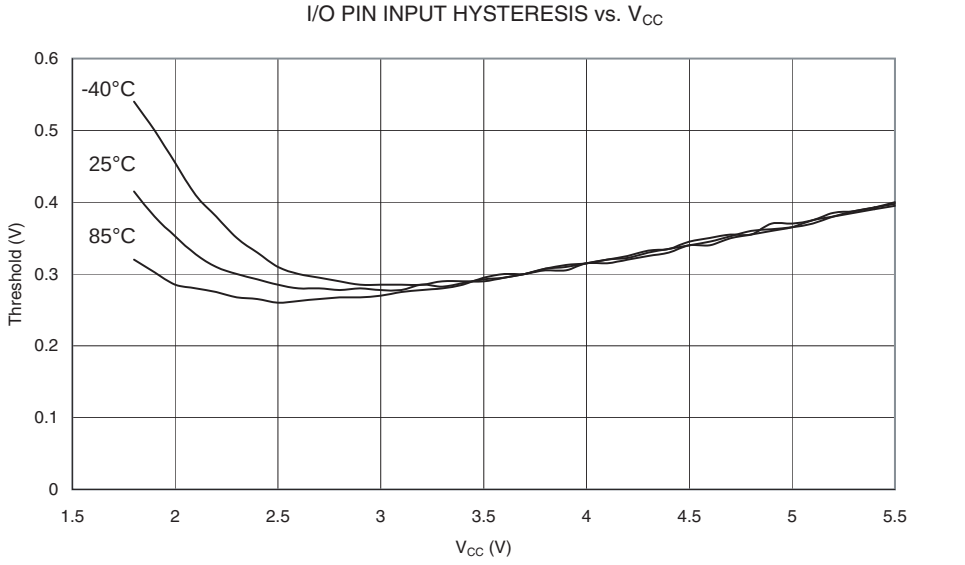


**Figure 167.** Reset Input Threshold Voltage vs. $V_{CC}$ ($V_{IH}$,Reset Pin Read as “1”)

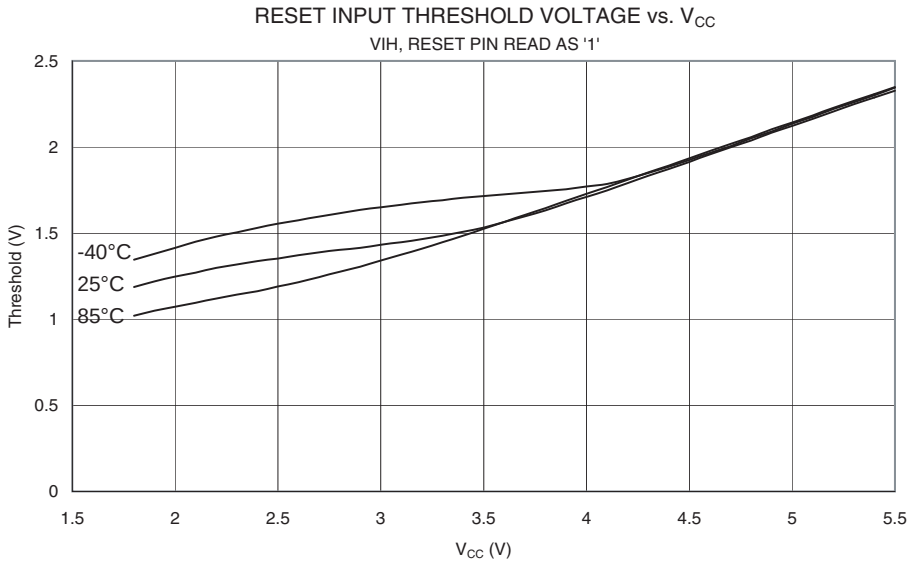



**Figure 168.** Reset Input Threshold Voltage vs. $V_{CC}$ ($V_{IL}$,Reset Pin Read as “0”)

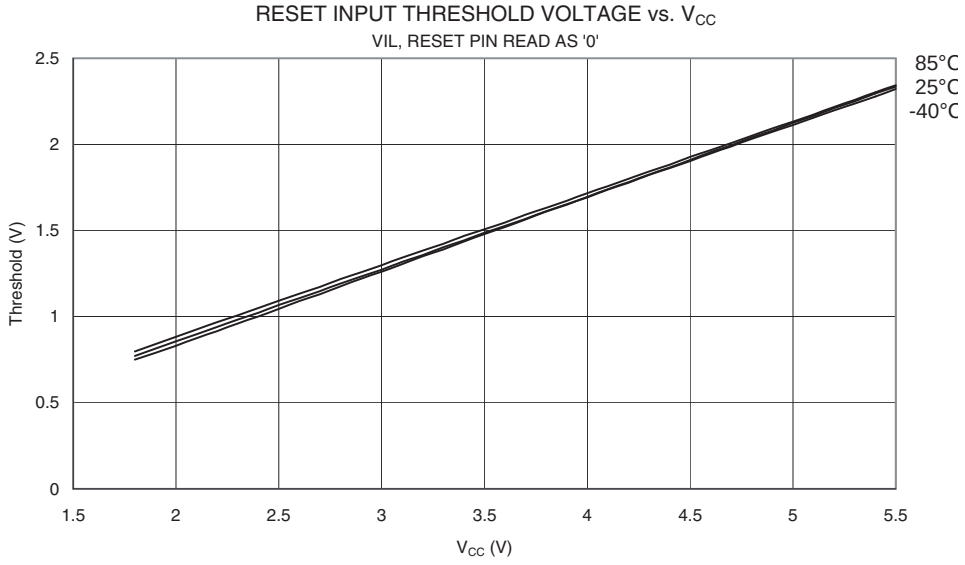
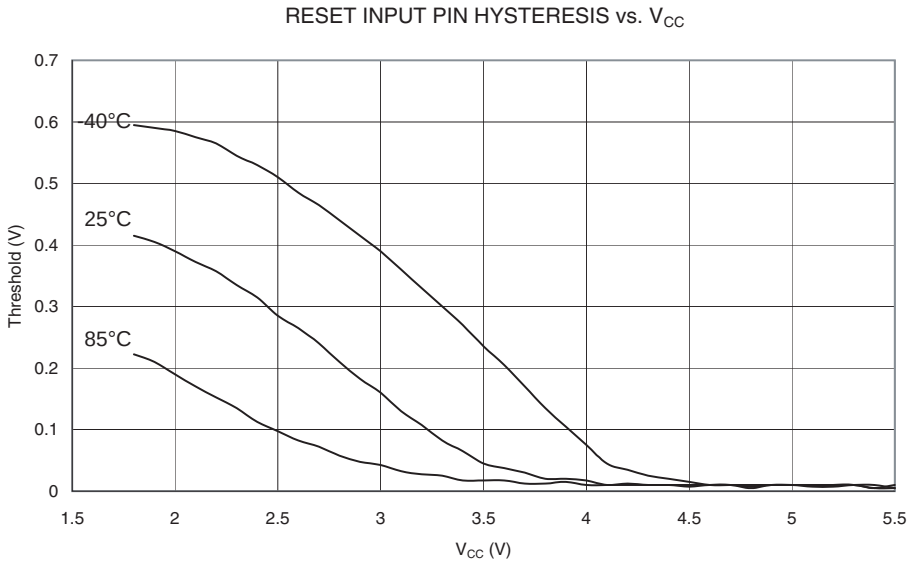


**Figure 169.** Reset Input Pin Hysteresis vs. $V_{CC}$




## BOD Thresholds and Analog Comparator Offset

**Figure 170.** BOD Thresholds vs. Temperature (BOD Level is 4.3V)

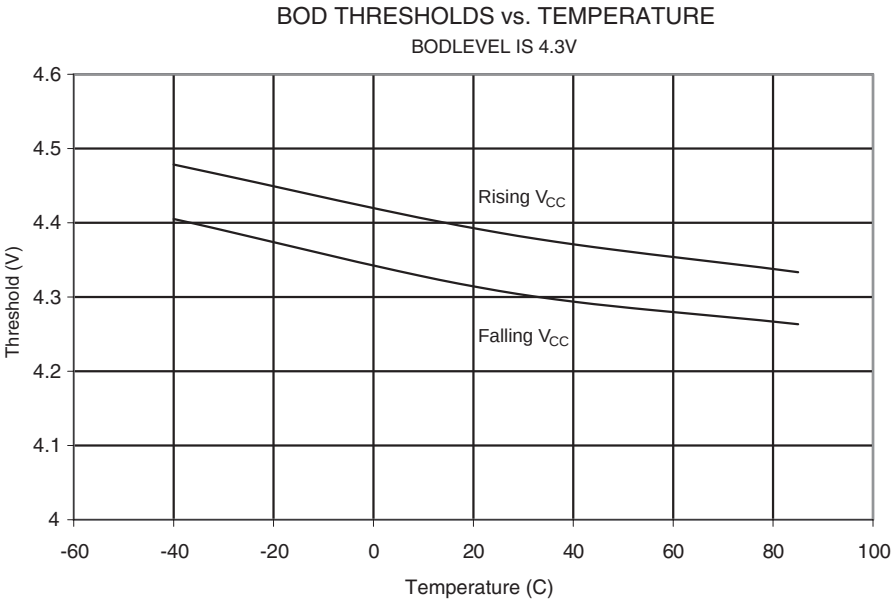


**Figure 171.** BOD Thresholds vs. Temperature (BOD Level is 2.7V)

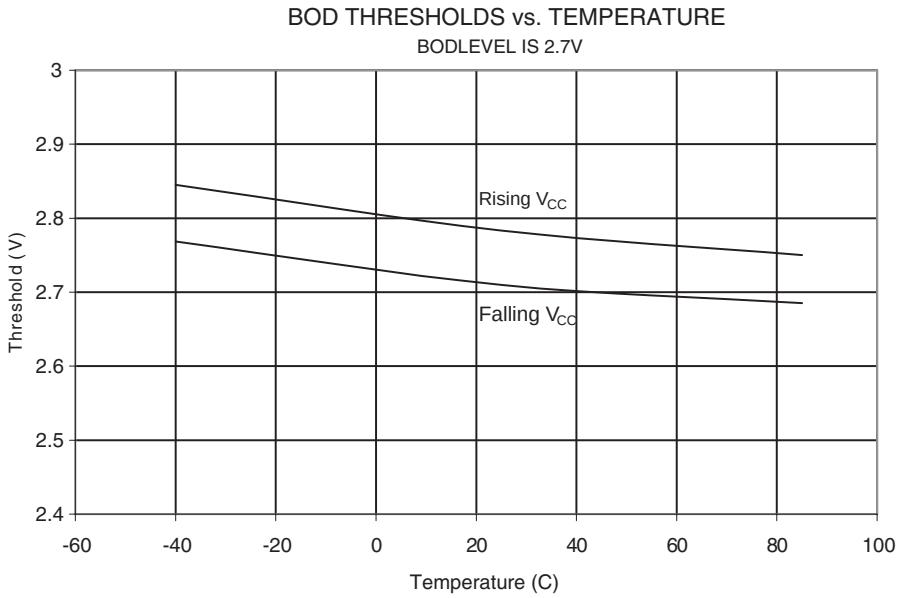



**Figure 172.** BOD Thresholds vs. Temperature (BOD Level is 1.8V)

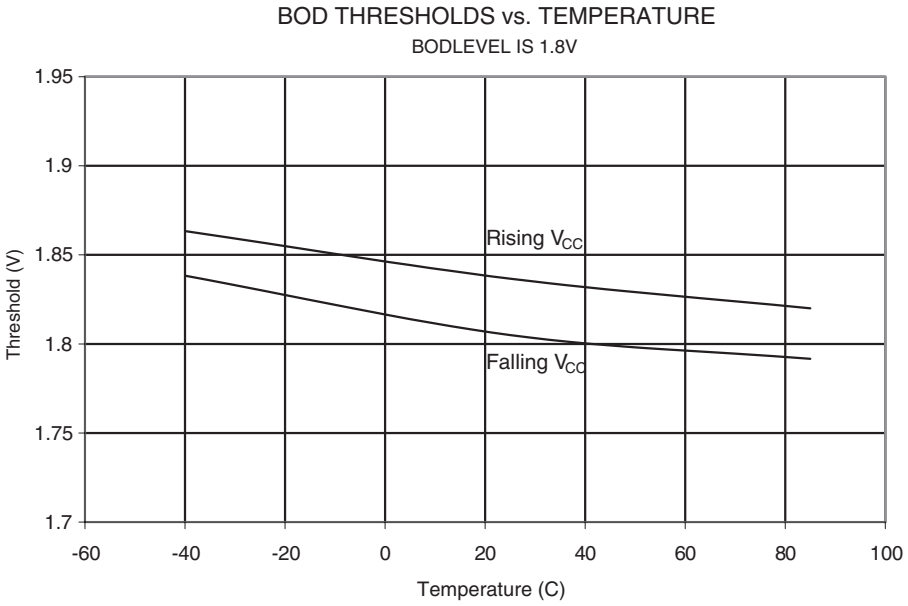


**Figure 173.** Bandgap Voltage vs. $V_{CC}$

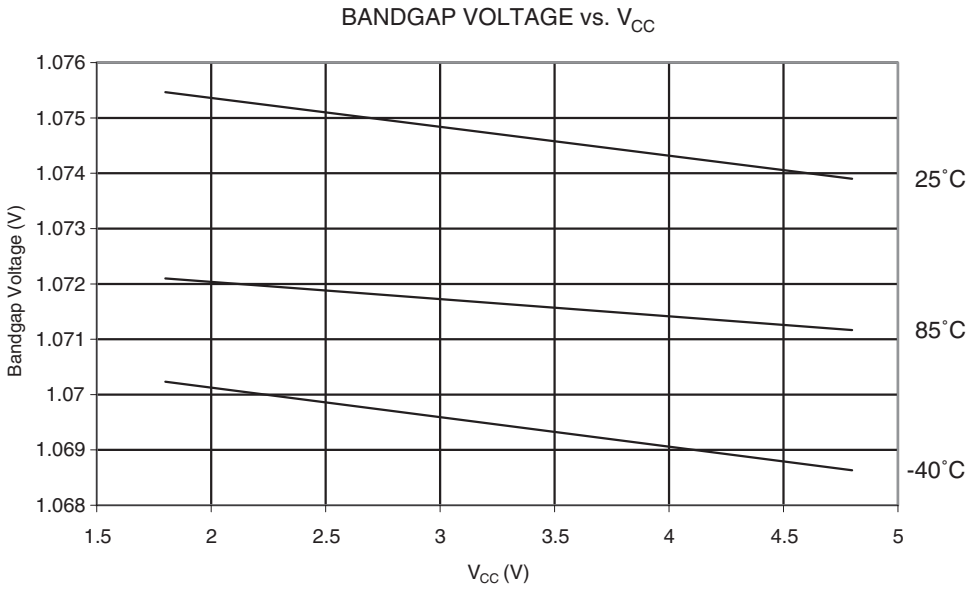



**Figure 174.** Analog Comparator Offset Voltage vs. Common Mode Voltage ($V_{CC}$ = 5V)

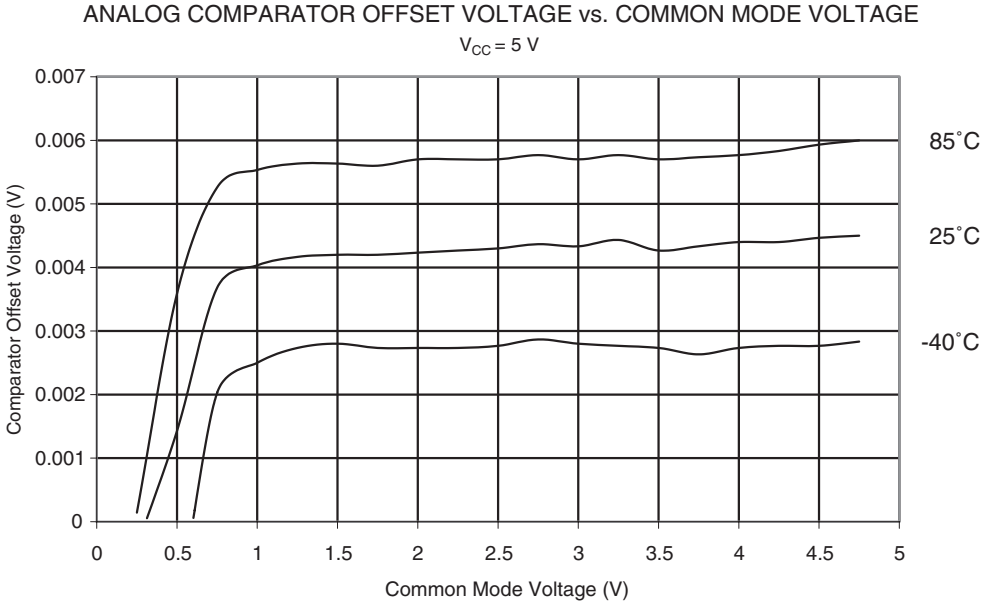
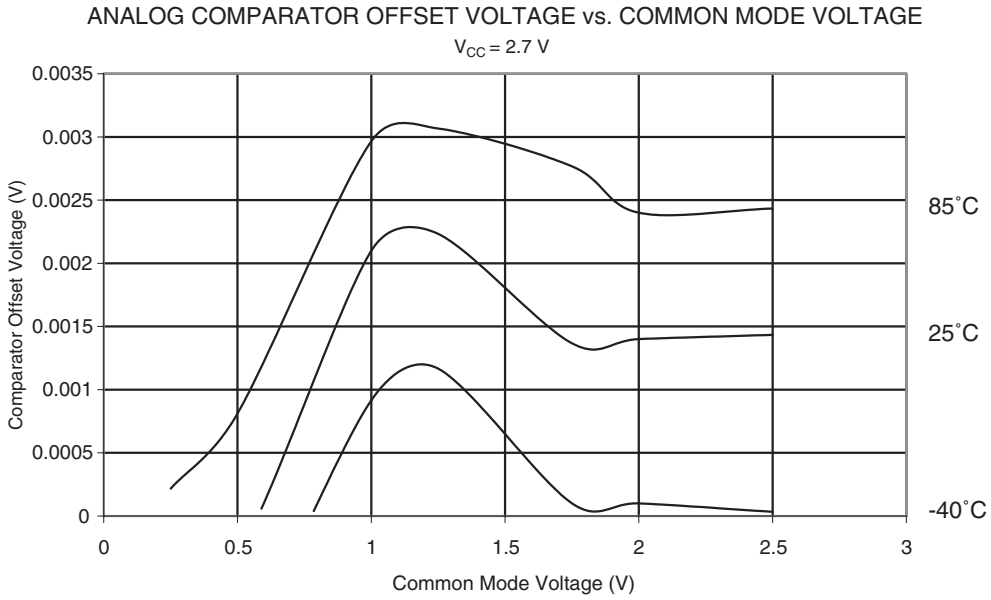


**Figure 175.** Analog Comparator Offset Voltage vs. Common Mode Voltage ($V_{CC}$ = 2.7V)




## Internal Oscillator Speed

**Figure 176.** Watchdog Oscillator Frequency vs. $V_{CC}$

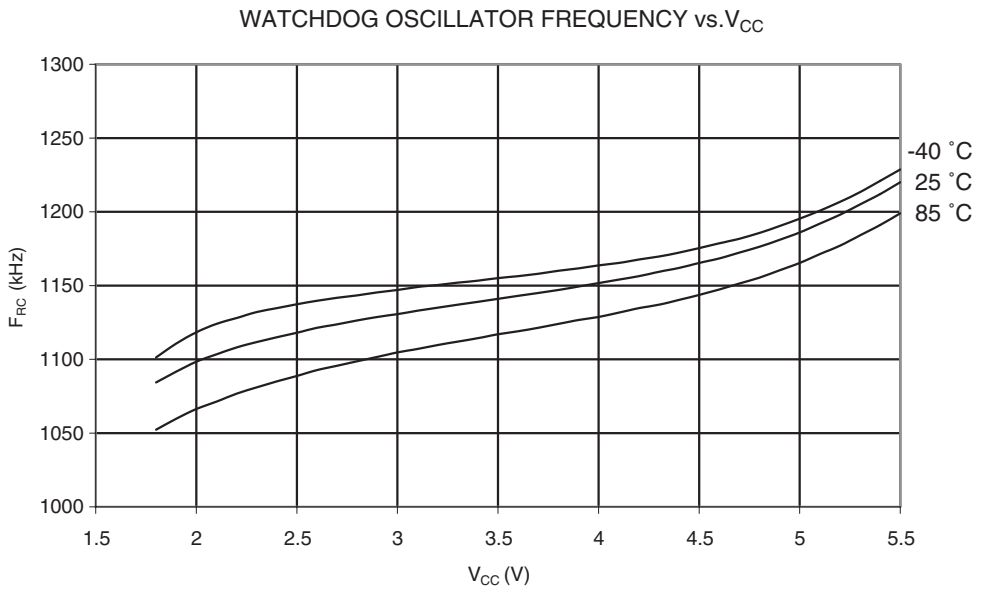


**Figure 177.** Calibrated 8 MHz RC Oscillator Frequency vs. Temperature

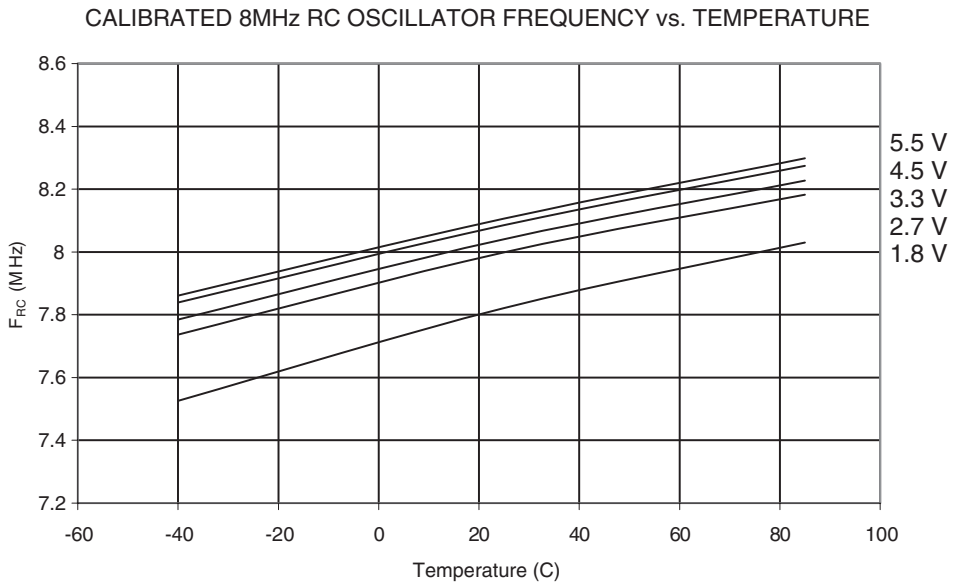



**Figure 178.** Calibrated 8 MHz RC Oscillator Frequency vs. $V_{CC}$

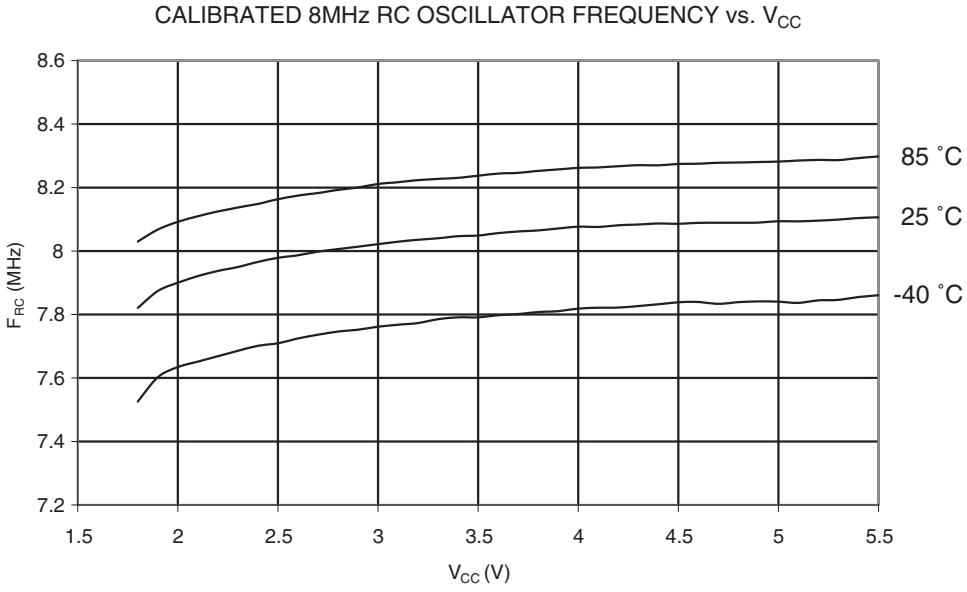
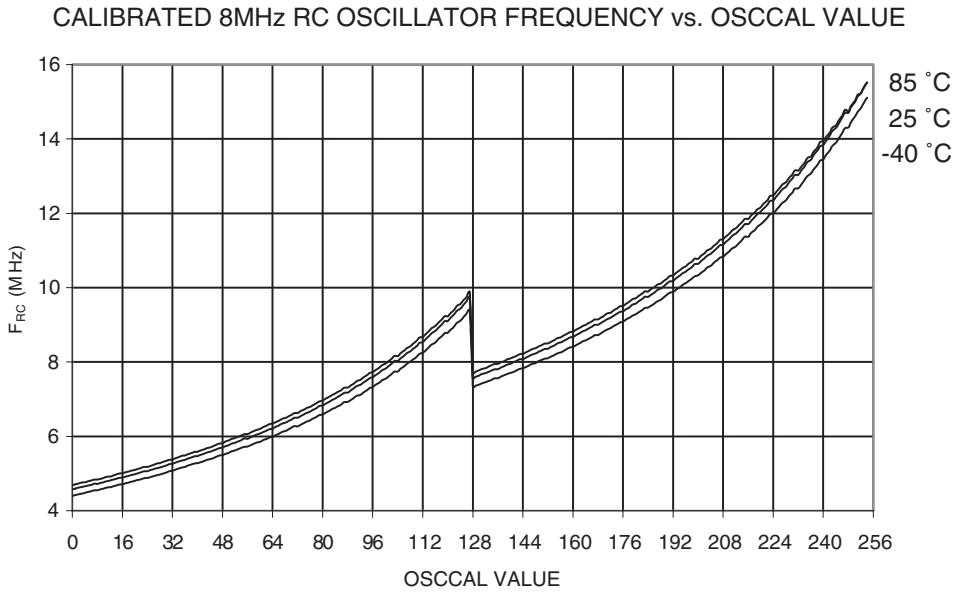


**Figure 179.** Calibrated 8 MHz RC Oscillator Frequency vs. Osccal Value




## Current Consumption of Peripheral Units

**Figure 180.** Brownout Detector Current vs. $V_{CC}$

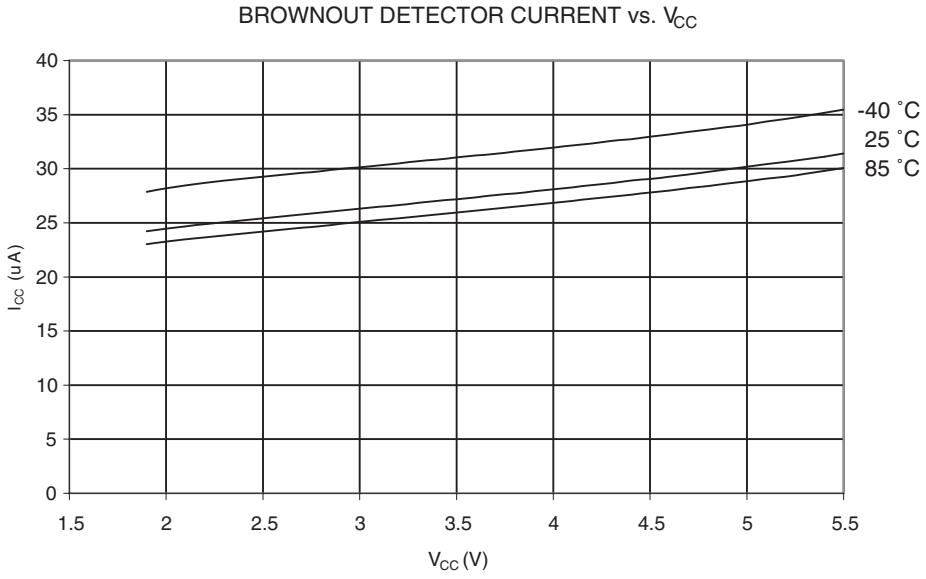


**Figure 181.** ADC Current vs. $V_{CC}$ (AREF = AVCC)

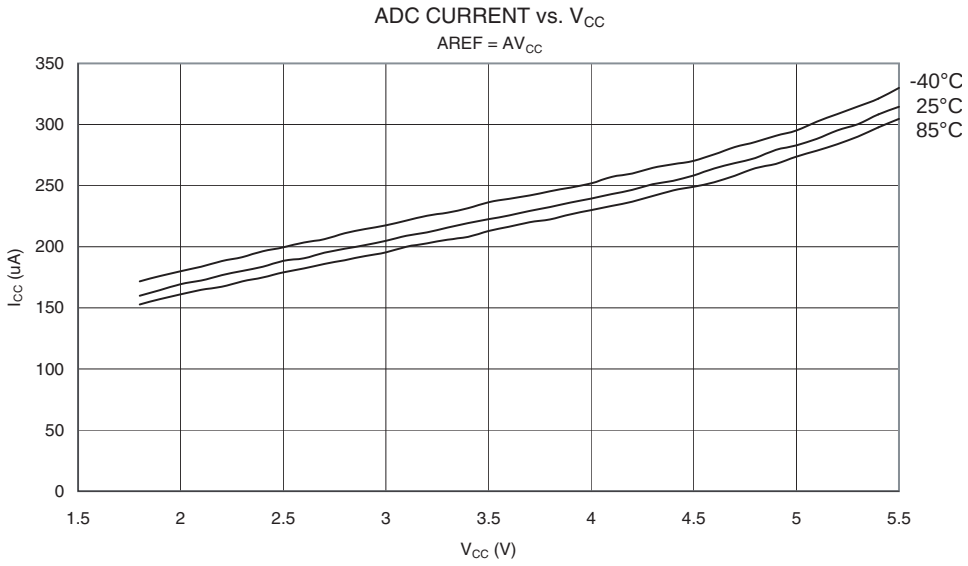



**Figure 182.** AREF External Reference Current vs. $V_{CC}$

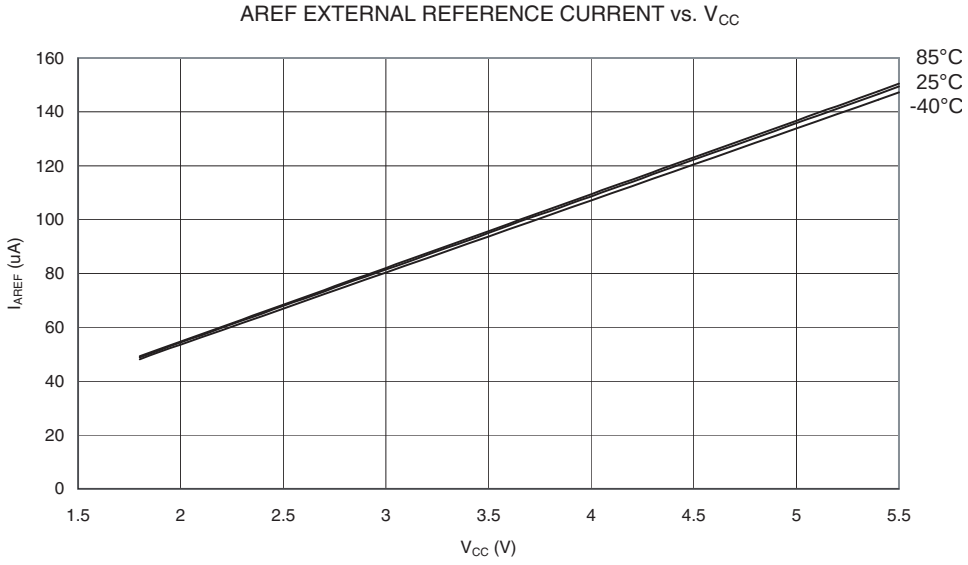


**Figure 183.** 32 kHZ TOSC Current vs. $V_{CC}$ (Watchdog Timer Disabled)

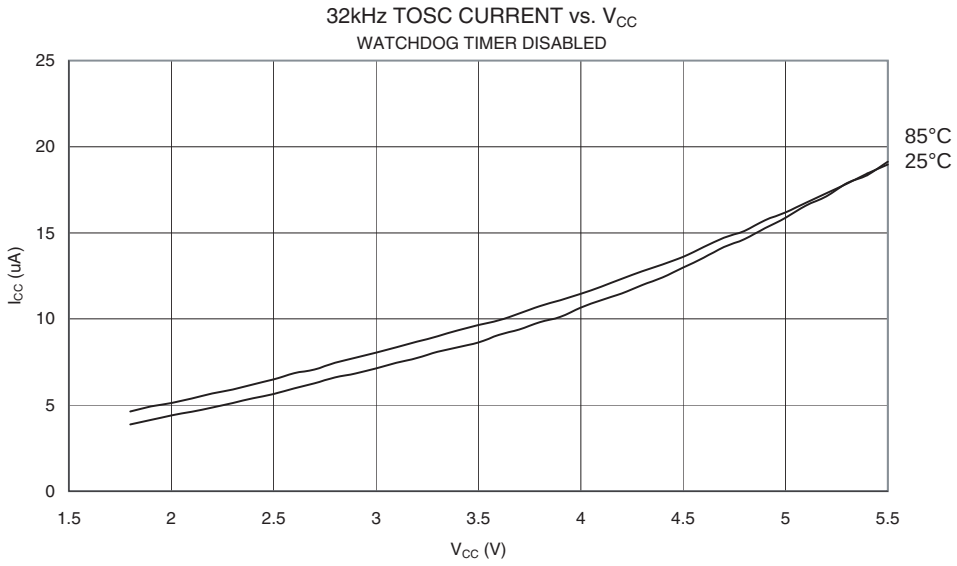



**Figure 184.** Watchdog Timer Current vs. $V_{CC}$

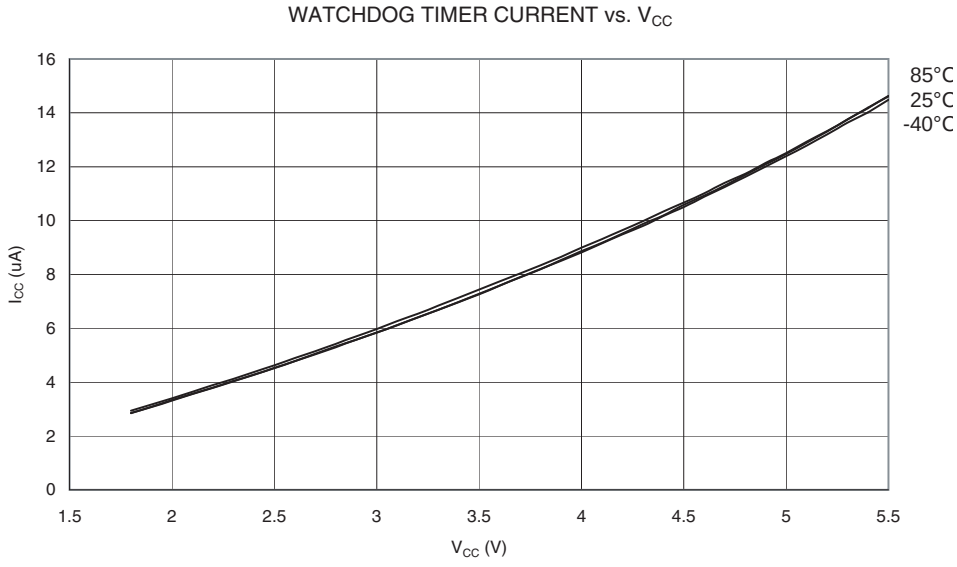


**Figure 185.** Analog Comparator Current vs. $V_{CC}$

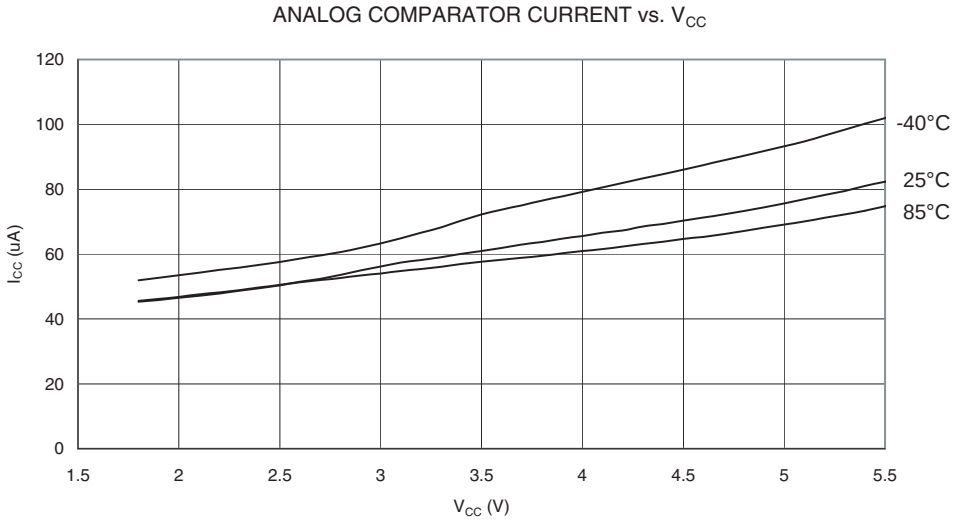
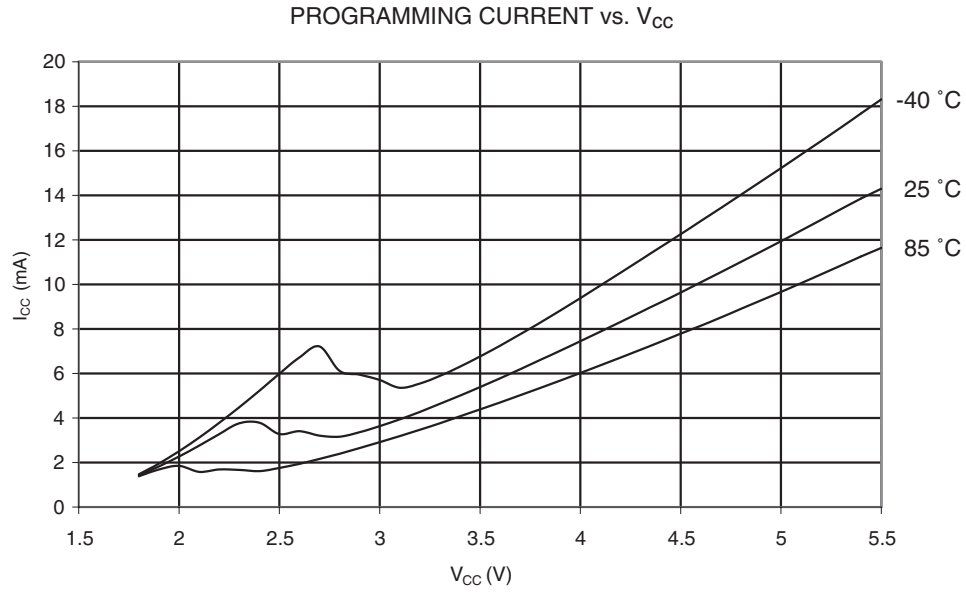



**Figure 186.** Programming Current vs. $V_{CC}$



**Current Consumption in Reset and Reset Pulsewidth**

**Figure 187.** Reset Supply Current vs. $V_{CC}$ (0.1 - 1.0 MHz, Excluding Current Through The Reset Pull-up)

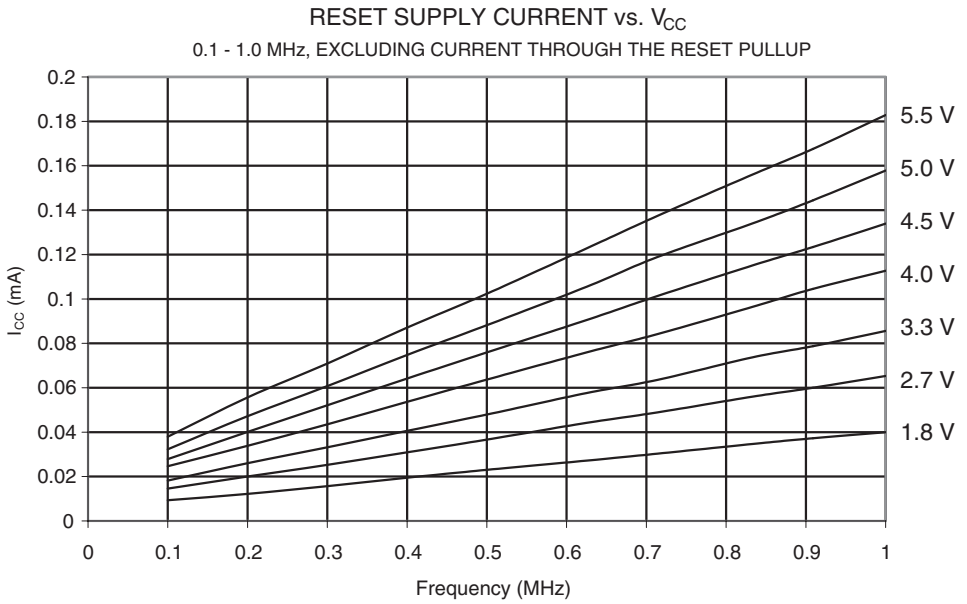



**Figure 188.** Reset Supply Current vs. $V_{CC}$ (1 - 16 MHz, Excluding Current Through The Reset Pull-up)

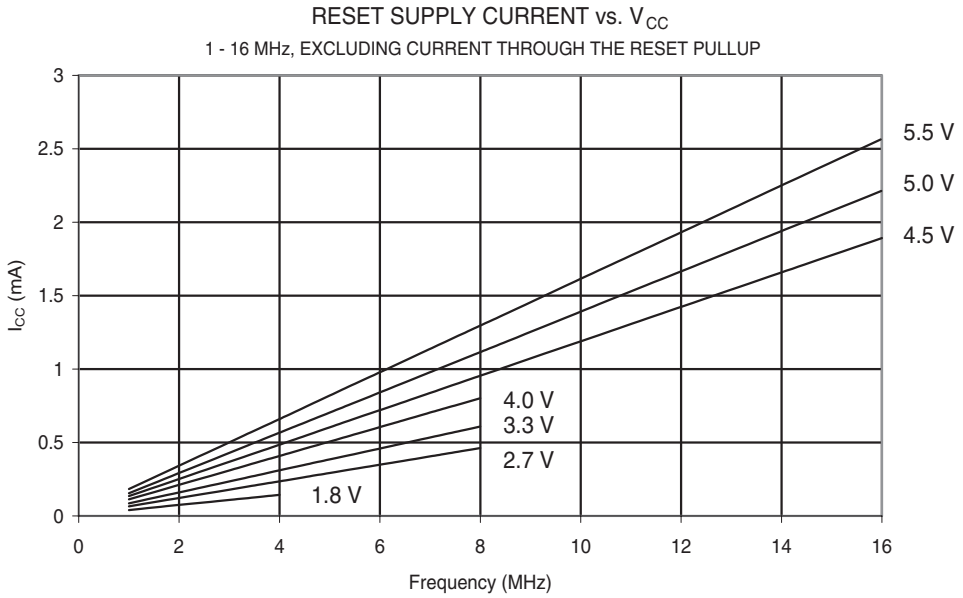
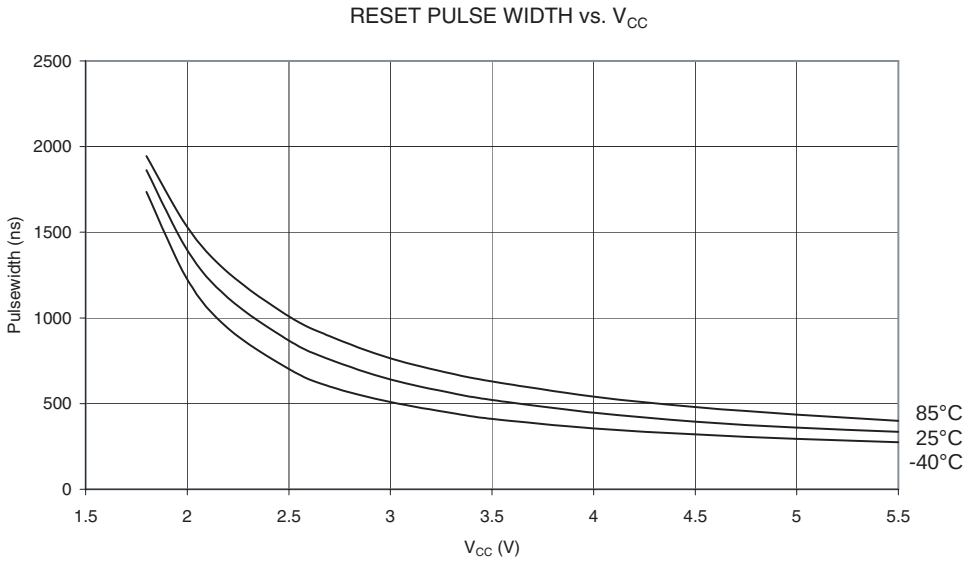


**Figure 189.** Reset Pulse Width vs. $V_{CC}$



## Register Summary

Note: Registers with bold type only available in ATmega3250/6450.

| 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 | - | - | - | - | - | - | - | - | |
| (0xDE) | Reserved | - | - | - | - | - | - | - | - | |
| (0xDD) | **PORTJ** | - | **PORTJ6** | **PORTJ5** | **PORTJ4** | **PORTJ3** | **PORTJ2** | **PORTJ1** | **PORTJ0** | 75 |
| (0xDC) | **DDRJ** | - | **DDJ6** | **DDJ5** | **DDJ4** | **DDJ3** | **DDJ2** | **DDJ1** | **DDJ0** | 75 |
| (0xDB) | **PINJ** | - | **PINJ6** | **PINJ5** | **PINJ4** | **PINJ3** | **PINJ2** | **PINJ1** | **PINJ0** | 75 |
| (0xDA) | **PORTH** | **PORTH7** | **PORTH6** | **PORTH5** | **PORTH4** | **PORTH3** | **PORTH2** | **PORTH1** | **PORTH0** | 75 |
| (0xD9) | **DDRH** | **DDH7** | **DDH6** | **DDH5** | **DDH4** | **DDH3** | **DDH2** | **DDH1** | **DDH0** | 75 |
| (0xD8) | **PINH** | **PINH7** | **PINH6** | **PINH5** | **PINH4** | **PINH3** | **PINH2** | **PINH1** | **PINH0** | 75 |
| (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) | UDR0 | USART0 Data Register | | | | | | | | 170 |
| (0xC5) | UBRR0H | | | | | USART0 Baud Rate Register High | | | | 173 |
| (0xC4) | UBRR0L | USART0 Baud Rate Register Low | | | | | | | | 173 |
| (0xC3) | Reserved | - | - | - | - | - | - | - | - | |

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Page |
|---|---|---|---|---|---|---|---|---|---|---|
| (0xC2) | UCSR0C | - | UMSEL0 | UPM01 | UPM00 | USBS0 | UCSZ01 | UCSZ00 | UCPOL0 | 172 |
| (0xC1) | UCSR0B | RXCIE0 | TXCIE0 | UDRIE0 | RXEN0 | TXEN0 | UCSZ20 | RXB80 | TXB80 | 171 |
| (0xC0) | UCSR0A | RXC0 | TXC0 | UDRE0 | FE0 | DOR0 | UPE0 | U2X0 | MPCM0 | 170 |
| (0xBF) | Reserved | - | - | - | - | - | - | - | - | |
| (0xBE) | Reserved | - | - | - | - | - | - | - | - | |
| (0xBD) | Reserved | - | - | - | - | - | - | - | - | |
| (0xBC) | Reserved | - | - | - | - | - | - | - | - | |
| (0xBB) | Reserved | - | - | - | - | - | - | - | - | |
| (0xBA) | USIDR | USI Data Register | | | | | | | | 185 |
| (0xB9) | USISR | USISIF | USIOIF | USIPF | USIDC | USICNT3 | USICNT2 | USICNT1 | USICNT0 | 186 |
| (0xB8) | USICR | USISIE | USIOIE | USIWM1 | USIWM0 | USICS1 | USICS0 | USICLK | USITC | 186 |
| (0xB7) | Reserved | - | - | - | - | - | - | - | - | |
| (0xB6) | ASSR | - | - | - | EXCLK | AS2 | TCN2UB | OCR2UB | TCR2UB | 138 |
| (0xB5) | Reserved | - | - | - | - | - | - | - | - | |
| (0xB4) | Reserved | - | - | - | - | - | - | - | - | |
| (0xB3) | OCR2A | Timer/Counter 2 Output Compare Register A | | | | | | | | 137 |
| (0xB2) | TCNT2 | Timer/Counter2 | | | | | | | | 137 |
| (0xB1) | Reserved | - | - | - | - | - | - | - | - | |
| (0xB0) | TCCR2A | FOC2A | WGM20 | COM2A1 | COM2A0 | WGM21 | CS22 | CS21 | CS20 | 135 |
| (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 | - | - | - | - | - | - | - | - | |
| (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 | | | | | | | | 121 |
| (0x8A) | OCR1BL | Timer/Counter1 Output Compare Register B Low | | | | | | | | 121 |
| (0x89) | OCR1AH | Timer/Counter1 Output Compare Register A High | | | | | | | | 121 |
| (0x88) | OCR1AL | Timer/Counter1 Output Compare Register A Low | | | | | | | | 121 |
| (0x87) | ICR1H | Timer/Counter1 Input Capture Register High | | | | | | | | 121 |
| (0x86) | ICR1L | Timer/Counter1 Input Capture Register Low | | | | | | | | 121 |
| (0x85) | TCNT1H | Timer/Counter1 High | | | | | | | | 121 |
| (0x84) | TCNT1L | Timer/Counter1 Low | | | | | | | | 121 |

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Page |
|---|---|---|---|---|---|---|---|---|---|---|
| (0x83) | Reserved | - | - | - | - | - | - | - | - | |
| (0x82) | TCCR1C | FOC1A | FOC1B | - | - | - | - | - | - | 120 |
| (0x81) | TCCR1B | ICNC1 | ICES1 | - | WGM13 | WGM12 | CS12 | CS11 | CS10 | 119 |
| (0x80) | TCCR1A | COM1A1 | COM1A0 | COM1B1 | COM1B0 | - | - | WGM11 | WGM10 | 117 |
| (0x7F) | DIDR1 | - | - | - | - | - | - | AIN1D | AIN0D | 193 |
| (0x7E) | DIDR0 | ADC7D | ADC6D | ADC5D | ADC4D | ADC3D | ADC2D | ADC1D | ADC0D | 210 |
| (0x7D) | Reserved | - | - | - | - | - | - | - | - | |
| (0x7C) | ADMUX | REFS1 | REFS0 | ADLAR | MUX4 | MUX3 | MUX2 | MUX1 | MUX0 | 206 |
| (0x7B) | ADCSRB | - | ACME | - | - | - | ADTS2 | ADTS1 | ADTS0 | 190/209 |
| (0x7A) | ADCSRA | ADEN | ADSC | ADATE | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 | 208 |
| (0x79) | ADCH | ADC Data Register High | | | | | | | | 209 |
| (0x78) | ADCL | ADC Data Register Low | | | | | | | | 209 |
| (0x77) | Reserved | - | - | - | - | - | - | - | - | |
| (0x76) | Reserved | - | - | - | - | - | - | - | - | |
| (0x75) | Reserved | - | - | - | - | - | - | - | - | |
| (0x74) | Reserved | - | - | - | - | - | - | - | - | |
| (0x73) | **PCMSK3** | - | **PCINT30** | **PCINT29** | **PCINT28** | **PCINT27** | **PCINT26** | **PCINT25** | **PCINT24** | 78 |
| (0x72) | Reserved | - | - | - | - | - | - | - | - | |
| (0x71) | Reserved | - | - | - | - | - | - | - | - | |
| (0x70) | TIMSK2 | - | - | - | - | - | - | OCIE2A | TOIE2 | 140 |
| (0x6F) | TIMSK1 | - | - | ICIE1 | - | - | OCIE1B | OCIE1A | TOIE1 | 122 |
| (0x6E) | TIMSK0 | - | - | - | - | - | - | OCIE0A | TOIE0 | 93 |
| (0x6D) | **PCMSK2** | **PCINT23** | **PCINT22** | **PCINT21** | **PCINT20** | **PCINT19** | **PCINT18** | **PCINT17** | **PCINT16** | 79 |
| (0x6C) | PCMSK1 | PCINT15 | PCINT14 | PCINT13 | PCINT12 | PCINT11 | PCINT10 | PCINT9 | PCINT8 | 79 |
| (0x6B) | PCMSK0 | PCINT7 | PCINT6 | PCINT5 | PCINT4 | PCINT3 | PCINT2 | PCINT1 | PCINT0 | 79 |
| (0x6A) | Reserved | - | - | - | - | - | - | - | - | |
| (0x69) | EICRA | - | - | - | - | - | - | ISC01 | ISC00 | 76 |
| (0x68) | Reserved | - | - | - | - | - | - | - | - | |
| (0x67) | Reserved | - | - | - | - | - | - | - | - | |
| (0x66) | OSCCAL | Oscillator Calibration Register [CAL7..0] | | | | | | | | 29 |
| (0x65) | Reserved | - | - | - | - | - | - | - | - | |
| (0x64) | PRR | - | - | - | - | PRTIM1 | PRSPI | PSUSART0 | PRADC | 35 |
| (0x63) | Reserved | - | - | - | - | - | - | - | - | |
| (0x62) | Reserved | - | - | - | - | - | - | - | - | |
| (0x61) | CLKPR | CLKPCE | - | - | - | CLKPS3 | CLKPS2 | CLKPS1 | CLKPS0 | 31 |
| (0x60) | WDTCR | - | - | - | WDCE | WDE | WDP2 | WDP1 | WDP0 | 44 |
| 0x3F (0x5F) | SREG | I | T | H | S | V | N | Z | C | 11 |
| 0x3E (0x5E) | SPH | Stack Pointer High | | | | | | | | 13 |
| 0x3D (0x5D) | SPL | Stack Pointer Low | | | | | | | | 13 |
| 0x3C (0x5C) | Reserved | - | - | - | - | - | - | - | - | |
| 0x3B (0x5B) | Reserved | - | - | - | - | - | - | - | - | |
| 0x3A (0x5A) | Reserved | - | - | - | - | - | - | - | - | |
| 0x39 (0x59) | Reserved | - | - | - | - | - | - | - | - | |
| 0x38 (0x58) | Reserved | - | - | - | - | - | - | - | - | |
| 0x37 (0x57) | SPMCSR | SPMIE | RWWSB | - | RWWSRE | BLBSET | PGWRT | PGERS | SPMEN | 247 |
| 0x36 (0x56) | Reserved | | | | | | | | | |
| 0x35 (0x55) | MCUCR | JTD | - | - | PUD | - | - | IVSEL | IVCE | 50/59/220 |
| 0x34 (0x54) | MCUSR | - | - | - | JTRF | WDRF | BORF | EXTRF | PORF | 42 |
| 0x33 (0x53) | SMCR | - | - | - | - | SM2 | SM1 | SM0 | SE | 33 |
| 0x32 (0x52) | Reserved | - | - | - | - | - | - | - | - | |
| 0x31 (0x51) | OCDR | IDRD/OCDR7 | OCDR6 | OCDR5 | OCDR4 | OCDR3 | OCDR2 | OCDR1 | OCDR0 | 216 |
| 0x30 (0x50) | ACSR | ACD | ACBG | ACO | ACI | ACIE | ACIC | ACIS1 | ACIS0 | 190 |
| 0x2F (0x4F) | Reserved | - | - | - | - | - | - | - | - | |
| 0x2E (0x4E) | SPDR | SPI Data Register | | | | | | | | 150 |
| 0x2D (0x4D) | SPSR | SPIF | WCOL | - | - | - | - | - | SPI2X | 150 |
| 0x2C (0x4C) | SPCR | SPIE | SPE | DORD | MSTR | CPOL | CPHA | SPR1 | SPR0 | 148 |
| 0x2B (0x4B) | GPIOR2 | General Purpose I/O Register | | | | | | | | 24 |
| 0x2A (0x4A) | GPIOR1 | General Purpose I/O Register | | | | | | | | 24 |
| 0x29 (0x49) | Reserved | - | - | - | - | - | - | - | - | |
| 0x28 (0x48) | Reserved | - | - | - | - | - | - | - | - | |
| 0x27 (0x47) | OCR0A | Timer/Counter0 Output Compare A | | | | | | | | 93 |
| 0x26 (0x46) | TCNT0 | Timer/Counter0 | | | | | | | | 92 |
| 0x25 (0x45) | Reserved | - | - | - | - | - | - | - | - | |

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Page |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x24 (0x44) | TCCR0A | FOC0A | WGM00 | COM0A1 | COM0A0 | WGM01 | CS02 | CS01 | CS00 | 90 |
| 0x23 (0x43) | GTCCR | TSM | - | - | - | - | - | PSR2 | PSR10 | 95/142 |
| 0x22 (0x42) | EEARH | - | - | - | - | - | EEPROM Address Register High | | | 20 |
| 0x21 (0x41) | EEARL | EEPROM Address Register Low | | | | | | | | 20 |
| 0x20 (0x40) | EEDR | EEPROM Data Register | | | | | | | | 20 |
| 0x1F (0x3F) | EECR | - | - | - | - | EERIE | EEMWE | EEWE | EERE | 20 |
| 0x1E (0x3E) | GPIOR0 | General Purpose I/O Register | | | | | | | | 24 |
| 0x1D (0x3D) | EIMSK | **PCIE3** | **PCIE2** | PCIE1 | PCIE0 | - | - | - | INT0 | 77 |
| 0x1C (0x3C) | EIFR | **PCIF3** | **PCIF2** | PCIF1 | PCIF0 | - | - | - | INTF0 | 78 |
| 0x1B (0x3B) | Reserved | - | - | - | - | - | - | - | - | |
| 0x1A (0x3A) | Reserved | - | - | - | - | - | - | - | - | |
| 0x19 (0x39) | Reserved | - | - | - | - | - | - | - | - | |
| 0x18 (0x38) | Reserved | - | - | - | - | - | - | - | - | |
| 0x17 (0x37) | TIFR2 | - | - | - | - | - | - | OCF2A | TOV2 | 140 |
| 0x16 (0x36) | TIFR1 | - | - | ICF1 | - | - | OCF1B | OCF1A | TOV1 | 122 |
| 0x15 (0x35) | TIFR0 | - | - | - | - | - | - | OCF0A | TOV0 | 93 |
| 0x14 (0x34) | PORTG | - | - | - | PORTG4 | PORTG3 | PORTG2 | PORTG1 | PORTG0 | 75 |
| 0x13 (0x33) | DDRG | - | - | - | DDG4 | DDG3 | DDG2 | DDG1 | DDG0 | 75 |
| 0x12 (0x32) | PING | - | - | PING5 | PING4 | PING3 | PING2 | PING1 | PING0 | 75 |
| 0x11 (0x31) | PORTF | PORTF7 | PORTF6 | PORTF5 | PORTF4 | PORTF3 | PORTF2 | PORTF1 | PORTF0 | 74 |
| 0x10 (0x30) | DDRF | DDF7 | DDF6 | DDF5 | DDF4 | DDF3 | DDF2 | DDF1 | DDF0 | 74 |
| 0x0F (0x2F) | PINF | PINF7 | PINF6 | PINF5 | PINF4 | PINF3 | PINF2 | PINF1 | PINF0 | 74 |
| 0x0E (0x2E) | PORTE | PORTE7 | PORTE6 | PORTE5 | PORTE4 | PORTE3 | PORTE2 | PORTE1 | PORTE0 | 74 |
| 0x0D (0x2D) | DDRE | DDE7 | DDE6 | DDE5 | DDE4 | DDE3 | DDE2 | DDE1 | DDE0 | 74 |
| 0x0C (0x2C) | PINE | PINE7 | PINE6 | PINE5 | PINE4 | PINE3 | PINE2 | PINE1 | PINE0 | 74 |
| 0x0B (0x2B) | PORTD | PORTD7 | PORTD6 | PORTD5 | PORTD4 | PORTD3 | PORTD2 | PORTD1 | PORTD0 | 74 |
| 0x0A (0x2A) | DDRD | DDD7 | DDD6 | DDD5 | DDD4 | DDD3 | DDD2 | DDD1 | DDD0 | 74 |
| 0x09 (0x29) | PIND | PIND7 | PIND6 | PIND5 | PIND4 | PIND3 | PIND2 | PIND1 | PIND0 | 74 |
| 0x08 (0x28) | PORTC | PORTC7 | PORTC6 | PORTC5 | PORTC4 | PORTC3 | PORTC2 | PORTC1 | PORTC0 | 73 |
| 0x07 (0x27) | DDRC | DDC7 | DDC6 | DDC5 | DDC4 | DDC3 | DDC2 | DDC1 | DDC0 | 73 |
| 0x06 (0x26) | PINC | PINC7 | PINC6 | PINC5 | PINC4 | PINC3 | PINC2 | PINC1 | PINC0 | 73 |
| 0x05 (0x25) | PORTB | PORTB7 | PORTB6 | PORTB5 | PORTB4 | PORTB3 | PORTB2 | PORTB1 | PORTB0 | 73 |
| 0x04 (0x24) | DDRB | DDB7 | DDB6 | DDB5 | DDB4 | DDB3 | DDB2 | DDB1 | DDB0 | 73 |
| 0x03 (0x23) | PINB | PINB7 | PINB6 | PINB5 | PINB4 | PINB3 | PINB2 | PINB1 | PINB0 | 73 |
| 0x02 (0x22) | PORTA | PORTA7 | PORTA6 | PORTA5 | PORTA4 | PORTA3 | PORTA2 | PORTA1 | PORTA0 | 73 |
| 0x01 (0x21) | DDRA | DDA7 | DDA6 | DDA5 | DDA4 | DDA3 | DDA2 | DDA1 | DDA0 | 73 |
| 0x00 (0x20) | PINA | PINA7 | PINA6 | PINA5 | PINA4 | PINA3 | PINA2 | PINA1 | PINA0 | 73 |

Note:
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 ATmega325/3250/645/6450 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.

## 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 |
| MUL | Rd, Rr | Multiply Unsigned | R1:R0 ← Rd x Rr | Z,C | 2 |
| MULS | Rd, Rr | Multiply Signed | R1:R0 ← Rd x Rr | Z,C | 2 |
| MULSU | Rd, Rr | Multiply Signed with Unsigned | R1:R0 ← Rd x Rr | Z,C | 2 |
| FMUL | Rd, Rr | Fractional Multiply Unsigned | R1:R0 ← (Rd x Rr) << 1 | Z,C | 2 |
| FMULS | Rd, Rr | Fractional Multiply Signed | R1:R0 ← (Rd x Rr) << 1 | Z,C | 2 |
| FMULSU | Rd, Rr | Fractional Multiply Signed with Unsigned | R1:R0 ← (Rd x Rr) << 1 | Z,C | 2 |
| **BRANCH INSTRUCTIONS** | | | | | |
| RJMP | k | Relative Jump | PC ← PC + k + 1 | None | 2 |
| IJMP | | Indirect Jump to (Z) | PC ← Z | None | 2 |
| JMP | k | Direct Jump | PC ← k | None | 3 |
| RCALL | k | Relative Subroutine Call | PC ← PC + k + 1 | None | 3 |
| ICALL | | Indirect Call to (Z) | PC ← Z | None | 3 |
| CALL | k | Direct Subroutine Call | PC ← k | None | 4 |
| 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 |
| 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 |

| Mnemonics | Operands | Description | Operation | Flags | #Clocks |
|---|---|---|---|---|---|
| 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 |
| 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 |
| 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 |

| Mnemonics | Operands | Description | Operation | Flags | #Clocks |
|---|---|---|---|---|---|
| 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 |

# Ordering Information

## ATmega325

| Speed (MHz)[3] | Power Supply | Ordering Code | Package Type[1] | Operational Range |
|---|---|---|---|---|
| 8 | 1.8 - 5.5V | ATmega325V-8AI<br>ATmega325V-8AU[2]<br>ATmega325V-8MI<br>ATmega325V-8MU[2] | 64A<br>64A<br>64M1<br>64M1 | Industrial<br>(-40°C to 85°C) |
| 16 | 4.5 - 5.5V | ATmega325-16AI<br>ATmega325-16AU[2]<br>ATmega325-16MI<br>ATmega325-16MU[2] | 64A<br>64A<br>64M1<br>64M1 | Industrial<br>(-40°C to 85°C) |

Notes: 1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging alternative, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
3. For Speed vs. $V_{CC}$ see Figure 128 on page 289 and Figure 129 on page 290.

| Package Type | |
|---|---|
| **64A** | 64-lead, 14 x 14 x 1.0 mm, Thin Profile Plastic Quad Flat Package (TQFP) |
| **64M1** | 64-pad, 9 x 9 x 1.0 mm, Micro Lead Frame Package (MLF) |
| **100A** | 100-lead, 14 x 14 x 1.0 mm, 0.5 mm Lead Pitch, Thin Profile Plastic Quad Flat Package (TQFP) |

## ATmega3250

| Speed (MHz)[3] | Power Supply | Ordering Code | Package Type[1] | Operational Range |
|---|---|---|---|---|
| 8 | 1.8 - 5.5V | ATmega3250V-8AI<br>ATmega3250V-8AU[2] | 100A<br>100A | Industrial<br>(-40°C to 85°C) |
| 16 | 4.5 - 5.5V | ATmega3250-16AI<br>ATmega3250-16AU[2] | 100A<br>100A | Industrial<br>(-40°C to 85°C) |

Notes: 1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging alternative, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
3. For Speed vs. $V_{CC}$ see Figure 128 on page 289 and Figure 129 on page 290.

| Package Type | |
|---|---|
| **64A** | 64-lead, 14 x 14 x 1.0 mm, Thin Profile Plastic Quad Flat Package (TQFP) |
| **64M1** | 64-pad, 9 x 9 x 1.0 mm, Micro Lead Frame Package (MLF) |
| **100A** | 100-lead, 14 x 14 x 1.0 mm, 0.5 mm Lead Pitch, Thin Profile Plastic Quad Flat Package (TQFP) |

## ATmega645

| Speed (MHz)[3] | Power Supply | Ordering Code | Package Type[1] | Operational Range |
|---|---|---|---|---|
| 8 | 1.8 - 5.5V | ATmega645V-8AI<br>ATmega645V-8AU[2]<br>ATmega645V-8MI<br>ATmega645V-8MU[2] | 64A<br>64A<br>64M1<br>64M1 | Industrial<br>(-40°C to 85°C) |
| 16 | 4.5 - 5.5V | ATmega645-16AI<br>ATmega645-16AU[2]<br>ATmega645-16MI<br>ATmega645-16MU[2] | 64A<br>64A<br>64M1<br>64M1 | Industrial<br>(-40°C to 85°C) |

Notes: 1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging alternative, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
3. For Speed vs. $V_{CC}$ see Figure 128 on page 289 and Figure 129 on page 290.

| Package Type | |
|---|---|
| **64A** | 64-lead, 14 x 14 x 1.0 mm, Thin Profile Plastic Quad Flat Package (TQFP) |
| **64M1** | 64-pad, 9 x 9 x 1.0 mm, Micro Lead Frame Package (MLF) |
| **100A** | 100-lead, 14 x 14 x 1.0 mm, 0.5 mm Lead Pitch, Thin Profile Plastic Quad Flat Package (TQFP) |

## ATmega6450

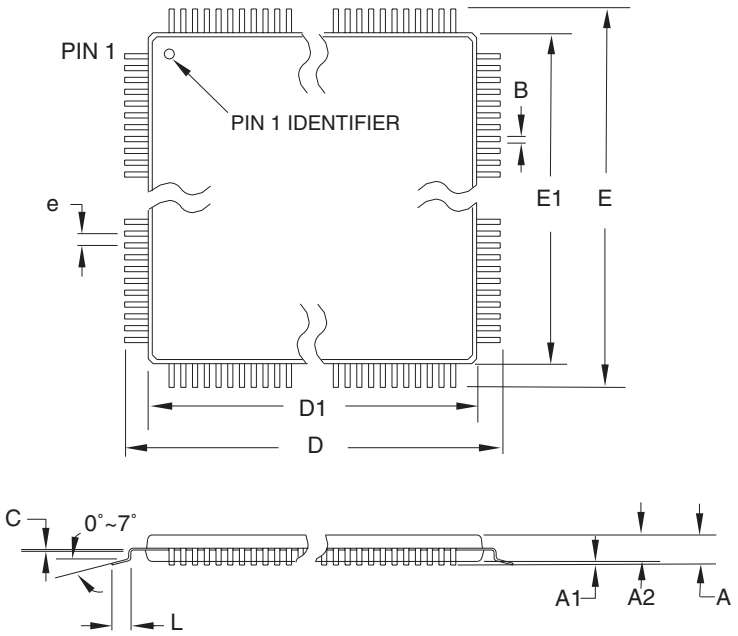
| Speed (MHz)[3] | Power Supply | Ordering Code | Package Type[1] | Operational Range |
|---|---|---|---|---|
| 8 | 1.8 - 5.5V | ATmega6450V-8AI<br>ATmega6450V-8AU[2] | 100A<br>100A | Industrial<br>(-40°C to 85°C) |
| 16 | 4.5 - 5.5V | ATmega6450-16AI<br>ATmega6450-16AU[2] | 100A<br>100A | Industrial<br>(-40°C to 85°C) |

Notes: 1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging alternative, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
3. For Speed vs. $V_{CC}$ see Figure 128 on page 289 and Figure 129 on page 290.

| Package Type | |
|---|---|
| **64A** | 64-lead, 14 x 14 x 1.0 mm, Thin Profile Plastic Quad Flat Package (TQFP) |
| **64M1** | 64-pad, 9 x 9 x 1.0 mm, Micro Lead Frame Package (MLF) |
| **100A** | 100-lead, 14 x 14 x 1.0 mm, 0.5 mm Lead Pitch, Thin Profile Plastic Quad Flat Package (TQFP) |

## Packaging Information

### 64A




**COMMON DIMENSIONS**
(Unit of Measure = mm)

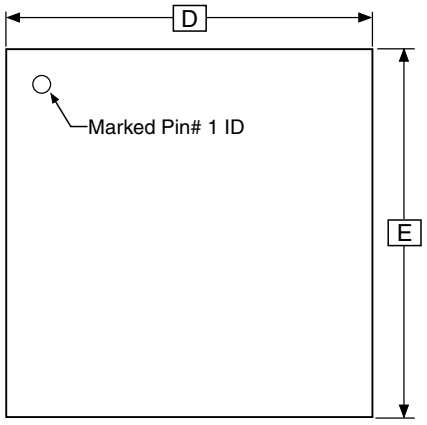
| SYMBOL | MIN | NOM | MAX | NOTE |
|---|---|---|---|---|
| A | – | – | 1.20 | |
| A1 | 0.05 | – | 0.15 | |
| A2 | 0.95 | 1.00 | 1.05 | |
| D | 15.75 | 16.00 | 16.25 | |
| D1 | 13.90 | 14.00 | 14.10 | Note 2 |
| E | 15.75 | 16.00 | 16.25 | |
| E1 | 13.90 | 14.00 | 14.10 | Note 2 |
| B | 0.30 | – | 0.45 | |
| C | 0.09 | – | 0.20 | |
| L | 0.45 | – | 0.75 | |
| e | 0.80 TYP | | | |

Notes:
1. This package conforms to JEDEC reference MS-026, Variation AEB.
2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25 mm per side. Dimensions D1 and E1 are maximum plastic body size dimensions including mold mismatch.
3. Lead coplanarity is 0.10 mm maximum.

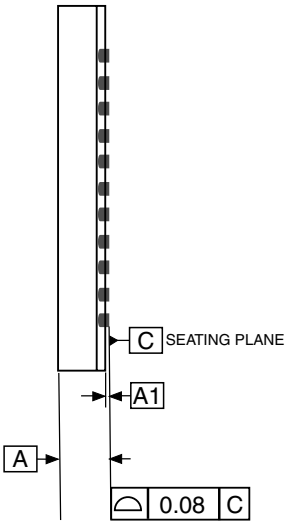
10/5/2001

| | TITLE | DRAWING NO. | REV. |
|---|---|---|---|
| 2325 Orchard Parkway<br>San Jose, CA 95131 | **64A,** 64-lead, 14 x 14 mm Body Size, 1.0 mm Body Thickness, 0.8 mm Lead Pitch, Thin Profile Plastic Quad Flat Package (TQFP) | 64A | B |

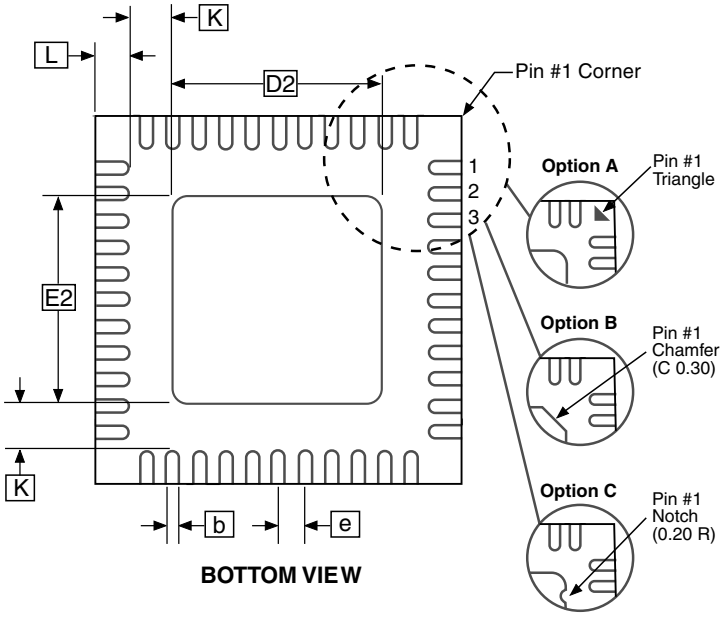
## 64M1




TOP VIEW




SIDE VIEW




BOTTOM VIEW

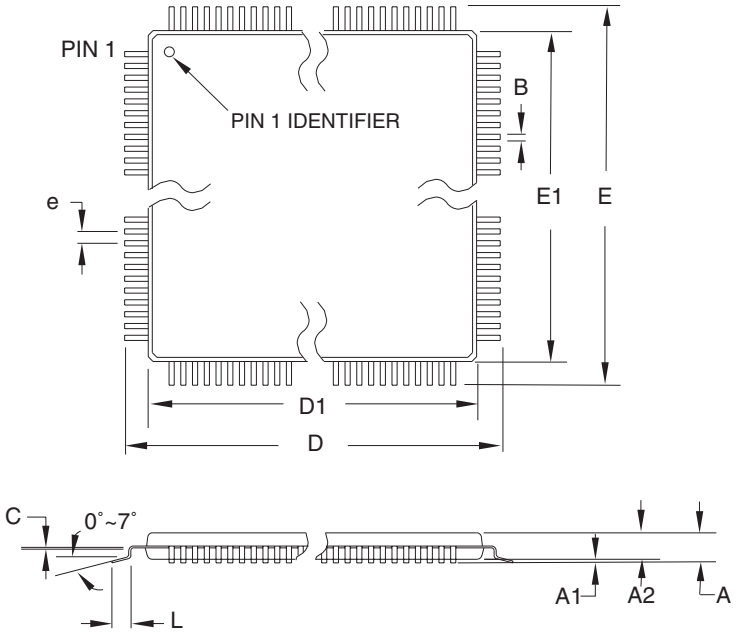
**COMMON DIMENSIONS**
(Unit of Measure = mm)

| SYMBOL | MIN | NOM | MAX | NOTE |
|---|---|---|---|---|
| A | 0.80 | 0.90 | 1.00 | |
| A1 | – | 0.02 | 0.05 | |
| b | 0.23 | 0.25 | 0.28 | |
| D | | 9.00 BSC | | |
| D2 | 5.20 | 5.40 | 5.60 | |
| E | | 9.00 BSC | | |
| E2 | 5.20 | 5.40 | 5.60 | |
| e | | 0.50 BSC | | |
| L | 0.35 | 0.40 | 0.45 | |
| K | 0.20 | – | – | |

Note: JEDEC Standard MO-220, (SAW Singulation) Fig. 1, VMMD.

8/19/04

| | TITLE | DRAWING NO. | REV. |
|---|---|---|---|
| 2325 Orchard Parkway<br>San Jose, CA 95131 | **64M1**, 64-pad, 9 x 9 x 1.0 mm Body, Lead Pitch 0.50 mm,<br>5.40 mm Exposed Pad, Micro Lead Frame Package (MLF) | 64M1 | D |




**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 | 15.75 | 16.00 | 16.25 | |
| D1 | 13.90 | 14.00 | 14.10 | Note 2 |
| E | 15.75 | 16.00 | 16.25 | |
| E1 | 13.90 | 14.00 | 14.10 | Note 2 |
| B | 0.17 | – | 0.27 | |
| C | 0.09 | – | 0.20 | |
| L | 0.45 | – | 0.75 | |
| e | 0.50 TYP | | | |

Notes:
1. This package conforms to JEDEC reference MS-026, Variation AED.
2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25 mm per side. Dimensions D1 and E1 are maximum plastic body size dimensions including mold mismatch.
3. Lead coplanarity is 0.08 mm maximum.

10/5/2001

| | TITLE | DRAWING NO. | REV. |
|---|---|---|---|
| 2325 Orchard Parkway<br>San Jose, CA 95131 | **100A,** 100-lead, 14 x 14 mm Body Size, 1.0 mm Body Thickness, 0.5 mm Lead Pitch, Thin Profile Plastic Quad Flat Package (TQFP) | 100A | C |

# Errata

## ATmega325

There is no errata for this revision of ATmega325.

## ATmega3250

There is no errata for this revision of ATmega3250.

## ATmega645

There is no errata for this revision of ATmega645.

## ATmega6450

There is no errata for this revision of ATmega6450.

## Datasheet Revision History

Please note that the referring page numbers in this section are referring to this document. The referring revision in this section are referring to the document revision.

### Rev. 2570C-11/04

1. **“Features (Continued)” on page 2 updated.**
2. **Table 10 on page 29 updated.**
3. **COM01:0 renamed COM0A1:0 in “8-bit Timer/Counter0 with PWM” on page 80.**
4. **PRR-bit descripton added to “16-bit Timer/Counter1” on page 96, “Serial Peripheral Interface – SPI” on page 143, and “USART0” on page 152.**
5. **“Part Number” on page 218 updated.**
6. **“ATmega325/3250/645/6450 Typical Characteristics – Preliminary Data” on page 293 updated.**
7. **“DC Characteristics” on page 287 updated.**
8. **“Alternate Functions of Port G” on page 67 updated.**

### Rev. 2570B-09/04

1. **Updated “Ordering Information” on page 330.**

### Rev. 2570A-09/04

1. **Initial revision.**

## Table of Contents


*Features* ..... 1

*Features (Continued)* ..... 2

*Pin Configurations* ..... 2

Disclaimer ..... 3

*Overview* ..... 4

Block Diagram ..... 4

Comparison between ATmega325, ATmega3250, ATmega645 and ATmega6450 6

Pin Descriptions ..... 6

*About Code Examples* ..... 8

*AVR CPU Core* ..... 9

Introduction ..... 9

Architectural Overview ..... 9

ALU – Arithmetic Logic Unit ..... 10

Status Register ..... 11

General Purpose Register File ..... 12

Stack Pointer ..... 13

Instruction Execution Timing ..... 14

Reset and Interrupt Handling ..... 14

*AVR ATmega325/3250/645/6450 Memories* ..... 17

In-System Reprogrammable Flash Program Memory ..... 17

SRAM Data Memory ..... 18

EEPROM Data Memory ..... 19

I/O Memory ..... 24

*System Clock and Clock Options* ..... 25

Clock Systems and their Distribution ..... 25

Clock Sources ..... 26

Default Clock Source ..... 26

Crystal Oscillator ..... 26

Low-frequency Crystal Oscillator ..... 28

Calibrated Internal RC Oscillator ..... 28

External Clock ..... 29

Clock Output Buffer ..... 30

Timer/Counter Oscillator ..... 30

System Clock Prescaler ..... 31

*Power Management and Sleep Modes* ..... 33

Idle Mode ..... 34

ADC Noise Reduction Mode ..... 34

Power-down Mode ..... 34
Power-save Mode ..... 34
Standby Mode ..... 35
Power Reduction Register ..... 35
Minimizing Power Consumption ..... 36

***System Control and Reset ..... 38***
Internal Voltage Reference ..... 43
Watchdog Timer ..... 43
Timed Sequences for Changing the Configuration of the Watchdog Timer ..... 46

***Interrupts ..... 47***
Interrupt Vectors in ATmega325/3250/645/6450 ..... 47

***I/O-Ports ..... 52***
Introduction ..... 52
Ports as General Digital I/O ..... 53
Alternate Port Functions ..... 57
Register Description for I/O-Ports ..... 73

***External Interrupts ..... 76***

***8-bit Timer/Counter0 with PWM ..... 80***
Overview ..... 80
Timer/Counter Clock Sources ..... 81
Counter Unit ..... 81
Output Compare Unit ..... 82
Compare Match Output Unit ..... 83
Modes of Operation ..... 85
Timer/Counter Timing Diagrams ..... 89
8-bit Timer/Counter Register Description ..... 90

***Timer/Counter0 and Timer/Counter1 Prescalers ..... 94***

***16-bit Timer/Counter1 ..... 96***
Overview ..... 96
Accessing 16-bit Registers ..... 99
Timer/Counter Clock Sources ..... 102
Counter Unit ..... 102
Input Capture Unit ..... 103
Output Compare Units ..... 105
Compare Match Output Unit ..... 107
Modes of Operation ..... 108
Timer/Counter Timing Diagrams ..... 115
16-bit Timer/Counter Register Description ..... 117

*8-bit Timer/Counter2 with PWM and Asynchronous Operation .. 124*
Overview ... 124
Timer/Counter Clock Sources ... 125
Counter Unit ... 125
Output Compare Unit ... 126
Compare Match Output Unit ... 128
Modes of Operation ... 129
Timer/Counter Timing Diagrams ... 133
8-bit Timer/Counter Register Description ... 135
Asynchronous operation of the Timer/Counter ... 138
Timer/Counter Prescaler ... 141

*Serial Peripheral Interface – SPI ... 143*
Overview ... 143
$\overline{SS}$ Pin Functionality ... 148
Data Modes ... 151

*USART0 ... 152*
Overview ... 152
Clock Generation ... 153
Frame Formats ... 156
USART Initialization ... 157
Data Transmission – The USART Transmitter ... 159
Data Reception – The USART Receiver ... 162
Asynchronous Data Reception ... 165
Multi-processor Communication Mode ... 169
USART Register Description ... 170
Examples of Baud Rate Setting ... 174

*Universal Serial Interface – USI ... 178*
Overview ... 178
Functional Descriptions ... 179
Alternative USI Usage ... 185
USI Register Descriptions ... 185

*Analog Comparator ... 190*
Analog Comparator Multiplexed Input ... 193

*Analog to Digital Converter ... 194*
Features ... 194
Operation ... 195
Starting a Conversion ... 196
Prescaling and Conversion Timing ... 197
Changing Channel or Reference Selection ... 199
ADC Noise Canceler ... 200
ADC Conversion Result ... 204

*JTAG Interface and On-chip Debug System ..... 211*
Overview ..... 211
Test Access Port – TAP ..... 211
TAP Controller ..... 213
Using the Boundary-scan Chain ..... 214
Using the On-chip Debug System ..... 214
On-chip Debug Specific JTAG Instructions ..... 215
On-chip Debug Related Register in I/O Memory ..... 216
Using the JTAG Programming Capabilities ..... 216
Bibliography ..... 216

*IEEE 1149.1 (JTAG) Boundary-scan ..... 217*
Features ..... 217
System Overview ..... 217
Data Registers ..... 217
Boundary-scan Specific JTAG Instructions ..... 219
Boundary-scan Related Register in I/O Memory ..... 220
Boundary-scan Chain ..... 221
ATmega325/3250/645/6450 Boundary-scan Order ..... 230
Boundary-scan Description Language Files ..... 242

*Boot Loader Support – Read-While-Write Self-Programming ..... 243*
Boot Loader Features ..... 243
Application and Boot Loader Flash Sections ..... 243
Read-While-Write and No Read-While-Write Flash Sections ..... 243
Boot Loader Lock Bits ..... 245
Entering the Boot Loader Program ..... 246
Addressing the Flash During Self-Programming ..... 248
Self-Programming the Flash ..... 249

*Memory Programming ..... 257*
Program And Data Memory Lock Bits ..... 257
Fuse Bits ..... 258
Signature Bytes ..... 260
Calibration Byte ..... 260
Parallel Programming Parameters, Pin Mapping, and Commands ..... 260
Serial Programming Pin Mapping ..... 262
Parallel Programming ..... 263
Serial Downloading ..... 271
Programming via the JTAG Interface ..... 275

*Electrical Characteristics ..... 287*
Absolute Maximum Ratings* ..... 287
DC Characteristics ..... 287
External Clock Drive Waveforms ..... 289
External Clock Drive ..... 289

Speed vs. $V_{CC}$ ..... 289
SPI Timing Characteristics ..... 290
ADC Characteristics – Preliminary Data ..... 292

***ATmega325/3250/645/6450 Typical Characteristics – Preliminary Data ..... 293***

***Register Summary ..... 323***

***Instruction Set Summary ..... 327***

***Ordering Information ..... 330***
ATmega325 ..... 330
ATmega3250 ..... 331
ATmega645 ..... 332
ATmega6450 ..... 333

***Packaging Information ..... 334***
64A ..... 334
64M1 ..... 335
100A ..... 336

***Errata ..... 337***
ATmega325 ..... 337
ATmega3250 ..... 337
ATmega645 ..... 337
ATmega6450 ..... 337

***Datasheet Revision History ..... 338***
Rev. 2570C-11/04 ..... 338
Rev. 2570B-09/04 ..... 338
Rev. 2570A-09/04 ..... 338

***Table of Contents ..... i***

## Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

## Regional Headquarters

*Europe*
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

*Asia*
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

*Japan*
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

## Atmel Operations

*Memory*
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

*Microcontrollers*
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

*ASIC/ASSP/Smart Cards*
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

*RF/Automotive*
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

*Biometrics/Imaging/Hi-Rel MPU/*
*High Speed Converters/RF Datacom*
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80


*Literature Requests*
www.atmel.com/literature


**Disclaimer:** The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN ATMEL'S TERMS AND CONDITIONS OF SALE LOCATED ON ATMEL'S WEB SITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

**© Atmel Corporation 2004. All rights reserved.** Atmel®, logo and combinations thereof, AVR®, and AVR Studio® are registered trademarks, and Everywhere You Are™ are the trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.

Printed on recycled paper.



2570C–AVR–11/04