ATmega164P-B/ATmega324P-B/ ATmega644P-B Automotive

8-bit Atmel Microcontroller with 16/32/64Kbytes In-system Programmable Flash

DATASHEET

Features

- High-performance, low-power Atmel® AVR® 8-bit microcontroller
- Advanced RISC architecture
  - 131 powerful instructions – most single-clock cycle execution
  - 32 × 8 general purpose working registers
  - Fully static operation
  - Up to 16MIPS throughput at 16MHz
  - On-chip 2-cycle multiplier
- High endurance non-volatile memory segments
  - 16/32/64Kbytes of in-system self-programmable flash program memory
  - 512/1/2Kbytes EEPROM
  - 1/2/4Kbytes internal SRAM
  - Write/erase cycles: 10,000 flash/ 100,000 EEPROM
  - Data retention: 20 years at 85°C/ 100 years at 25°C(1)
  - Optional boot code section with independent lock bits
    - In-system programming by on-chip boot program
    - True read-while-write operation
  - Programming lock for software security
- QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix acquisition
  - Up to 64 sense channels
- 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 (1)
- Peripheral features
  - Two 8-bit Timer/Counters with separate prescalers and compare modes
  - One/two 16-bit Timer/Counter with separate prescaler, compare mode, and capture mode
  - Real time counter with separate oscillator

---

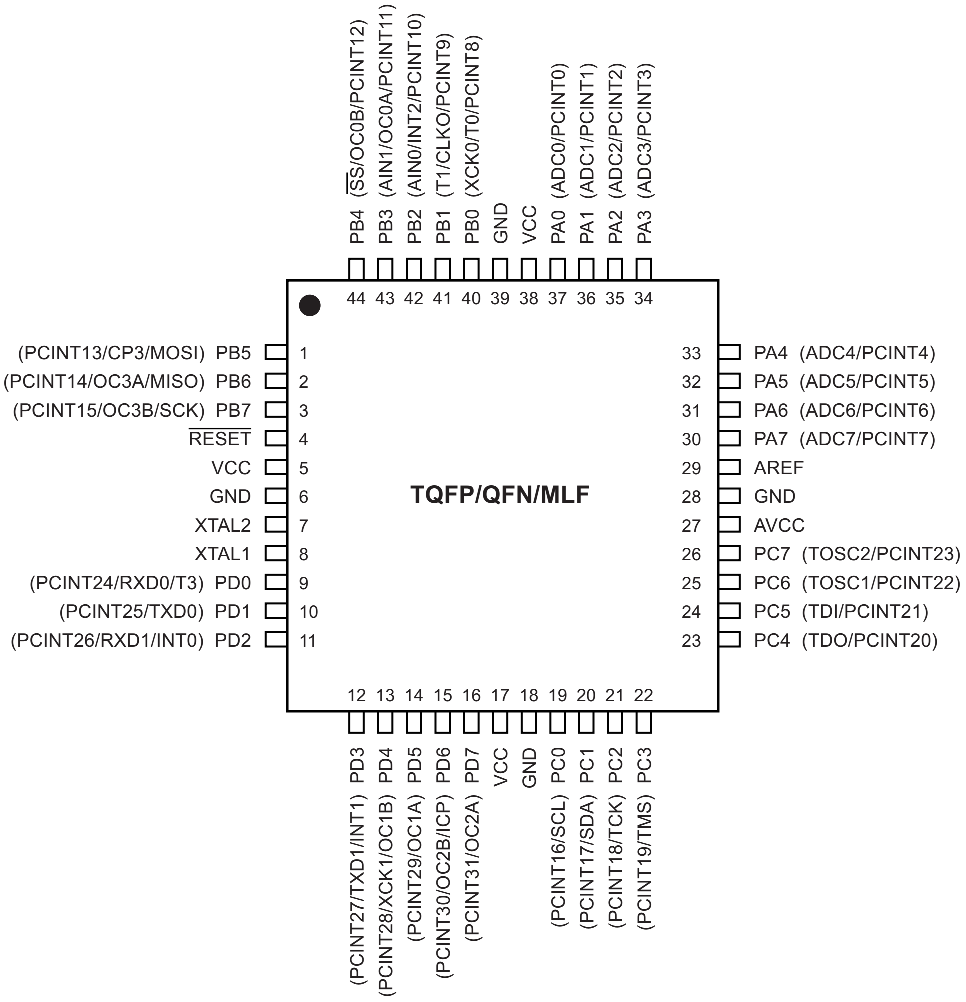
1. See Section 5. “Data Retention” on page 8 for details.

  - Six PWM channels
  - 8-channel, 10-bit ADC
    - Differential mode with selectable gain at 1x, 10x or 200x
  - Byte-oriented two-wire serial interface
  - Two programmable serial USART
  - Master/slave SPI serial interface
  - 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 RC oscillator
  - External and internal interrupt sources
  - Six sleep modes: Idle, ADC noise reduction, power-save, power-down, standby and extended standby
- I/O and packages
  - 32 programmable I/O lines
  - 44-lead TQFP, 44-pad QFN/MLF
- Operating voltages
  - 2.7 to 5.5V
- Speed grades
  - 0 to 8MHz at 2.7 to 5.5V; 0 to 16MHz at 4.5 to 5.5V
- Power consumption at 8MHz, 2.7V, 25°C
  - Active: 4.8mA
  - Idle mode: 1mA
  - Power-down mode: 0.6µA

1. Pin Configurations

1.1 Pinout - TQFP/QFN/MLF for ATmega164P-B/324P-B/644P-B

Figure 1-1. Pinout



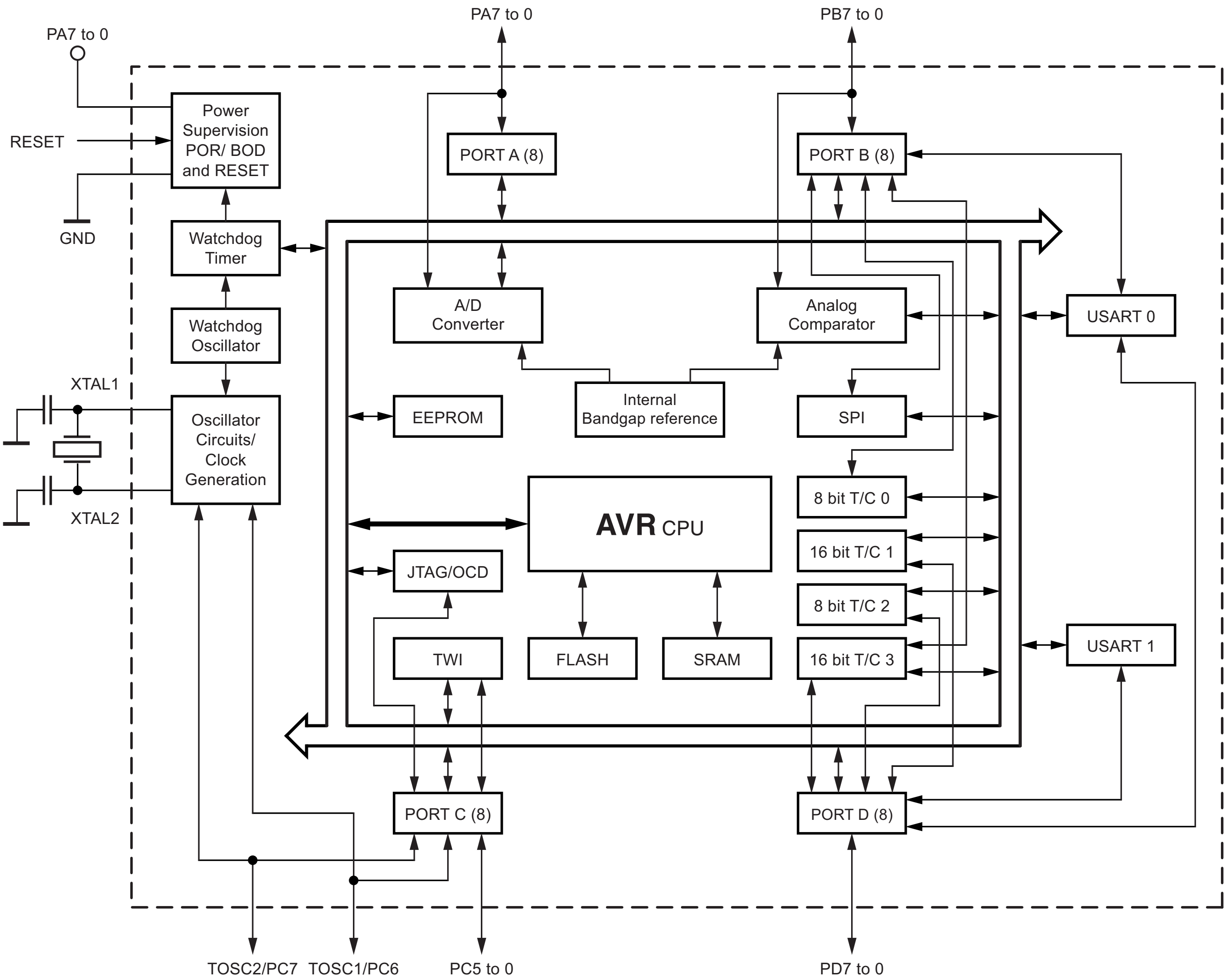
Note: The large center pad underneath the QFN/MLF package should be soldered to ground on the board to ensure good mechanical stability.

2. Overview

The ATmega164P-B/324P-B/644P-B 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 ATmega164P-B/324P-B/644P-B achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

2.1 Block Diagram

Figure 2-1. Block Diagram



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

The ATmega164P-B/324P-B/644P-B provide the following features:

16/32/64Kbytes of in-system programmable flash with read-while-write capabilities, 512/1/2Kbytes EEPROM, 1/2/4Kbytes SRAM, 32 general purpose I/O lines, 32 general purpose working registers, real time counter (RTC), three flexible Timer/Counters with compare modes and PWM, 2 USARTs, a byte oriented 2-wire serial interface, a 8-channel, 10-bit ADC with optional differential input stage with programmable gain, programmable watchdog timer with internal oscillator, an SPI serial port, IEEE std. 1149.1 compliant JTAG test interface, also used for accessing the on-chip debug system and programming and six 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 continues 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. In extended standby mode, both the main oscillator and the asynchronous timer continue to run.

Atmel® offers the QTouch® library for embedding capacitive touch buttons, sliders and wheels functionality into AVR microcontrollers. The patented charge-transfer signal acquisition offers robust sensing and includes fully debounced reporting of touch keys and includes Adjacent Key Suppression® (AKS®) technology for unambiguous detection of key events. The easy-to-use QTouch Suite toolchain allows you to explore, develop and debug your own touch applications.

The device is manufactured using Atmel's high-density nonvolatile memory technology. The on-chip ISP flash allows the program memory to be reprogrammed in-system through an SPI serial interface, by a conventional nonvolatile 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 ATmega164P-B/324P-B/644P-B is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega164P-B/324P-B/644P-B 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.

2.2 Automotive Quality Grade

The ATmega164P-B/324P-B/644P-B have been developed and manufactured according to the most stringent requirements of the international standard ISO-TS-16949. This data sheet contains limit values extracted from the results of extensive characterization (temperature and voltage).

The quality and reliability of theATmega164P-B/324P-B/644P-B have been verified during regular product qualification as per AEC-Q100 grade 1.

As indicated in the ordering information paragraph, the products are available in only one temperature grade.

Table 2-1. Temperature Grade Identification for Automotive Products

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

2.3 Comparison Between ATmega164P-B, ATmega324P-B and ATmega644P-B

Table 2-2. Differences between ATmega164P-B, ATmega324P-B and ATmega644P-B

Device	Flash	EEPROM	RAM	Unit
ATmega164P-B	16K	512	1K	Bytes
ATmega324P-B	32K	1K	2K	
ATmega644P-B	64K	2K	4K	

2.4 Pin Descriptions

2.4.1 VCC

Digital supply voltage.

2.4.2 GND

Ground.

2.4.3 Port A (PA7:PA0)

Port A serves as analog inputs to the analog-to-digital converter.

Port A also serves as 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 A also serves the functions of various special features of the ATmega164P-B/324P-B/644P-B as listed in Section 14.3.1 "Alternate Functions of Port A" on page 63.

2.4.4 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 also serves the functions of various special features of the ATmega164P-B/324P-B/644P-B as listed in Section 14.3.2 "Alternate Functions of Port B" on page 65.

2.4.5 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 C also serves the functions of the JTAG interface, along with special features of the ATmega164P-B/324P-B/644P-B as listed in Section 14.3.3 "Alternate Functions of Port C" on page 68.

2.4.6 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 ATmega164P-B/324P-B/644P-B as listed in Section 14.3.4 "Alternate Functions of Port D" on page 70.

2.4.7 $\overline{\text{RESET}}$

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 Section 28.5 "System and Reset Characteristics" on page 291. Shorter pulses are not guaranteed to generate a reset.

2.4.8 XTAL1

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

2.4.9 XTAL2

Output from the inverting Oscillator amplifier.

2.4.10 AVCC

AVCC is the supply voltage pin for Port A and the Analog-to-digital 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.

2.4.11 AREF

This is the analog reference pin for the analog-to-digital converter.

3. Resources

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

4. About Code Examples

This documentation contains simple code examples that briefly show how to use various parts of the device. 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.

The code examples assume that the part specific header file is included before compilation. 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”.

Note: 1.

5. Data Retention

Reliability Qualification results show that the projected data retention failure rate is much less than 1 PPM over 20 years at 85°C or 100 years at 25°C.

6. Capacitive Touch Sensing

The Atmel QTouch library provides a simple to use solution to realize touch sensitive interfaces on most Atmel AVR microcontrollers. The QTouch library includes support for the QTouch and QMatrix acquisition methods.

Touch sensing can be added to any application by linking the appropriate Atmel QTouch Library for the AVR Microcontroller. This is done by using a simple set of APIs to define the touch channels and sensors, and then calling the touch sensing API’s to retrieve the channel information and determine the touch sensor states.

The QTouch Library is FREE and downloadable from the Atmel website at the following location:

http://www.atmel.com/qtouchlibrary

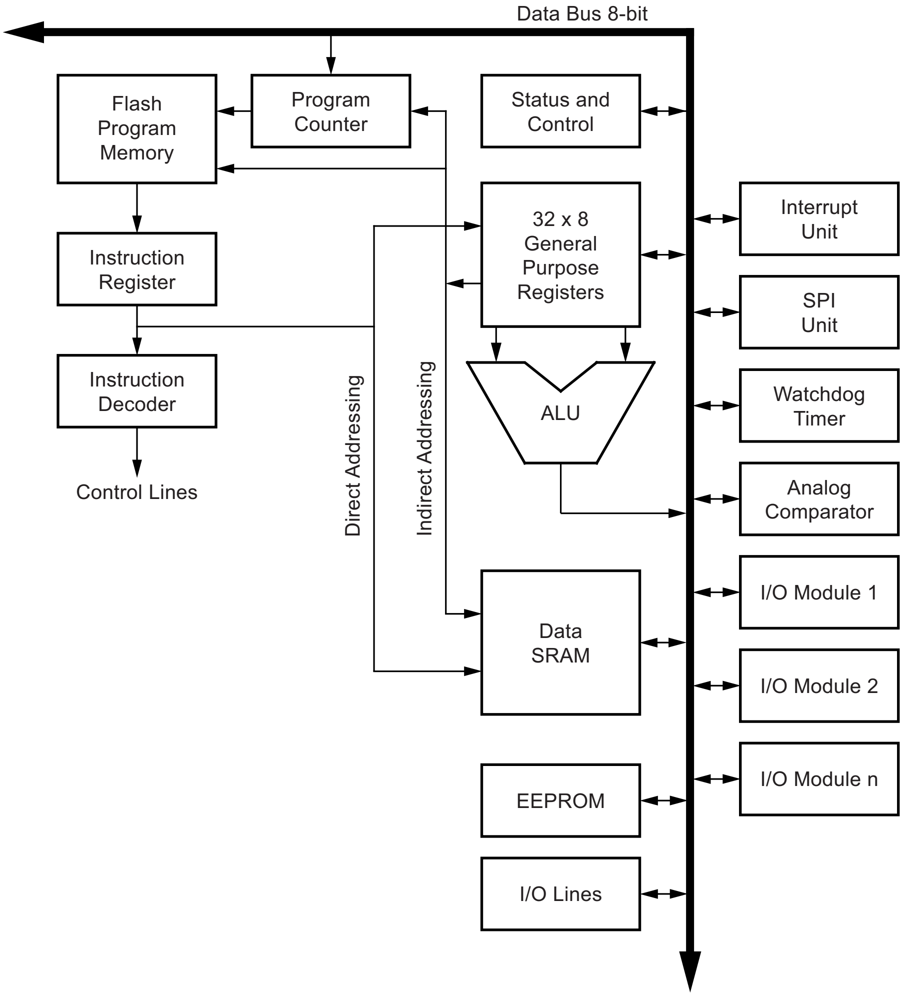
For implementation details and other information, refer to the Atmel QTouch Library User Guide - also available for download from the Atmel website.

7. AVR CPU Core

7.1 Overview

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.

Figure 7-1. Block Diagram of the AVR Architecture



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

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

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

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

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. 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 ATmega164P-B/324P-B/644P-B has extended I/O space from 0x60 - 0xFF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

7.2 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.

7.3 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.

7.3.1 SREG – Status Register

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	**I**	**T**	**H**	**S**	**V**	**N**	**Z**	**C**	**SREG**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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

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

- **Bit 6 – T: Bit Copy Storage**

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

- **Bit 5 – H: Half Carry Flag**

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

- **Bit 4 – S: Sign Bit, S = N ⊕ V**

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

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

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

- **Bit 2 – N: Negative Flag**

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

- **Bit 1 – Z: Zero Flag**

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

- **Bit 0 – C: Carry Flag**

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

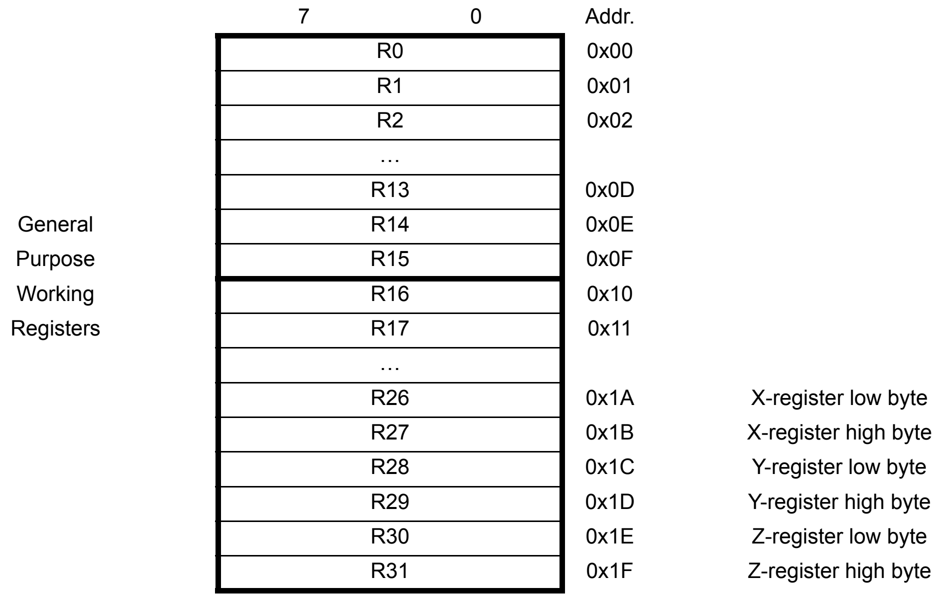
7.4 General Purpose Register File

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 7-2 shows the structure of the 32 general purpose working registers in the CPU.

Figure 7-2. AVR CPU General Purpose Working Registers



	7 0	Addr.	
General Purpose Working Registers	R0	0x00	
	R1	0x01	
	R2	0x02	
	…		
	R13	0x0D	
	R14	0x0E	
	R15	0x0F	
	R16	0x10	
	R17	0x11	
	…		
	R26	0x1A	X-register low byte
	R27	0x1B	X-register high byte
	R28	0x1C	Y-register low byte
	R29	0x1D	Y-register high byte
	R30	0x1E	Z-register low byte
	R31	0x1F	Z-register high byte

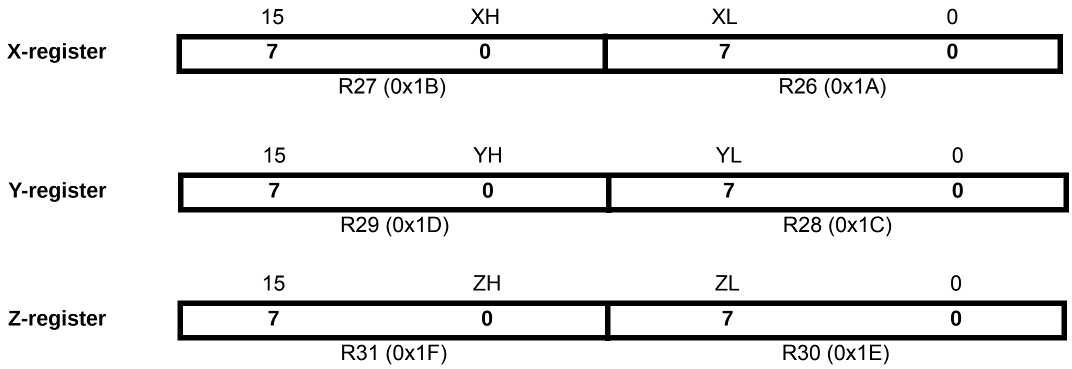
Most of the instructions operating on the register file have direct access to all registers, and most of them are single cycle instructions.

As shown in Figure 7-2, each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user data space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X-, Y- and Z-pointer registers can be set to index any register in the file.

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

The registers R26..R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the data space. The three indirect address registers X, Y, and Z are defined as described in Figure 7-3.

Figure 7-3. The X-, Y-, and Z-registers



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

7.5 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. Note that the stack is implemented as growing from higher to lower memory locations. The stack pointer register always points to the top of the stack. The stack pointer points to the data SRAM stack area where the subroutine and interrupt stacks are located. A stack PUSH command will decrease the stack pointer.

The stack in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. Initial stack pointer value equals the last address of the internal SRAM and the stack pointer must be set to point above start of the SRAM, see Figure 8-2 on page 17.

See Table 7-1 for stack pointer details.

Table 7-1. Stack Pointer instructions

Instruction	Stack pointer	Description
PUSH	Decremented by 1	Data is pushed onto the stack
CALL ICALL RCALL	Decremented by 2	Return address is pushed onto the stack with a subroutine call or interrupt
POP	Incremented by 1	Data is popped from the stack
RET RETI	Incremented by 2	Return address is popped from the stack with return from subroutine or return from interrupt

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, see Table 7-2 on page 13. 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.

7.5.1 SPH and SPL – Stack Pointer High and Stack pointer Low

Bit	15	14	13	12	11	10	9	8	
0x3E (0x5E)	–	–	–	**SP12**	**SP11**	**SP10**	**SP9**	**SP8**	**SPH**
0x3D (0x5D)	**SP7**	**SP6**	**SP5**	**SP4**	**SP3**	**SP2**	**SP1**	**SP0**	**SPL**
	7	6	5	4	3	2	1	0	
Read/Write	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	R/W	
Initial Value	0	0	0	0/1()	0/1()	1/0()	0	0	
	1	1	1	1	1	1	1	1	

Note: 1. Initial values respectively for the ATmega164P-B/324P-B/644P-B

Table 7-2. Stack Pointer Size

Device	Stack Pointer Size
ATmega164P-B	SP[10:0]
ATmega324P-B	SP[11:0]
ATmega644P-B	SP[12:0]

7.6 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 7-4 on page 13 shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast-access register file concept. This is the basic pipelining concept to obtain up to 1MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

Figure 7-4. The Parallel Instruction Fetches and Instruction Executions

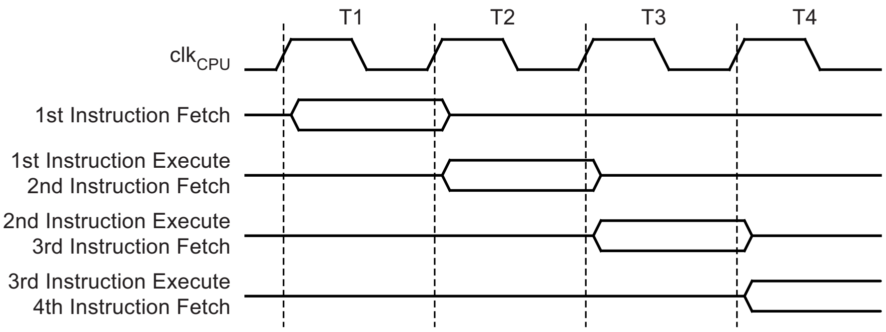
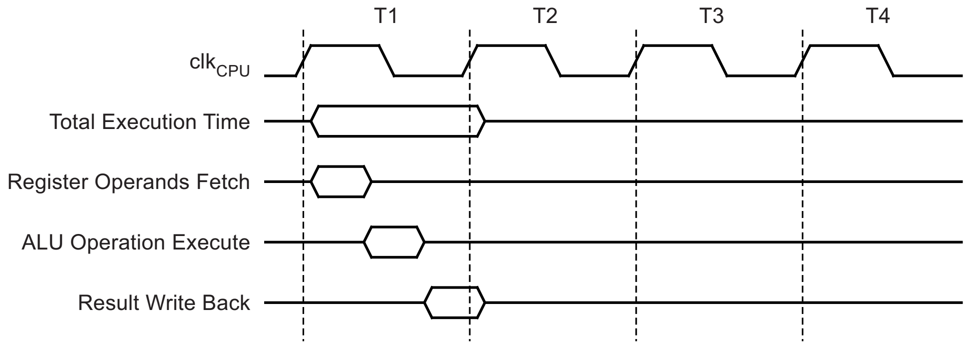


Figure 7-5 shows the internal timing concept for the register file. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

Figure 7-5. Single Cycle ALU Operation



7.7 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 Section 27. "Memory Programming" on page 255 for details.

The lowest addresses in the program memory space are by default defined as the reset and interrupt vectors. The complete list of vectors is shown in Section 12. "Interrupts" on page 49. 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 Section 12. "Interrupts" on page 49 for more information. The reset vector can also be moved to the start of the boot flash section by programming the BOOTRST fuse, see Section 27. "Memory Programming" on page 255.

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

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

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

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

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

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

Assembly Code Example

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

C Code Example

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

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

Assembly Code Example

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

C Code Example

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

7.7.1 Interrupt Response Time

The interrupt execution response for all the enabled AVR interrupts is five clock cycles minimum. After five clock cycles the program vector address for the actual interrupt handling routine is executed. During these five 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 five 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 five clock cycles. During these five clock cycles, the program counter (three bytes) is popped back from the stack, the stack pointer is incremented by three, and the I-bit in SREG is set.

8. AVR Memories

8.1 Overview

This section describes the different memories in the ATmega164P-B/324P-B/644P-B. The AVR architecture has two main memory spaces, the Data Memory and the Program Memory space. In addition, the ATmega164P-B/324P-B/644P-B features an EEPROM memory for data storage. All three memory spaces are linear and regular.

8.2 In-System Reprogrammable Flash Program Memory

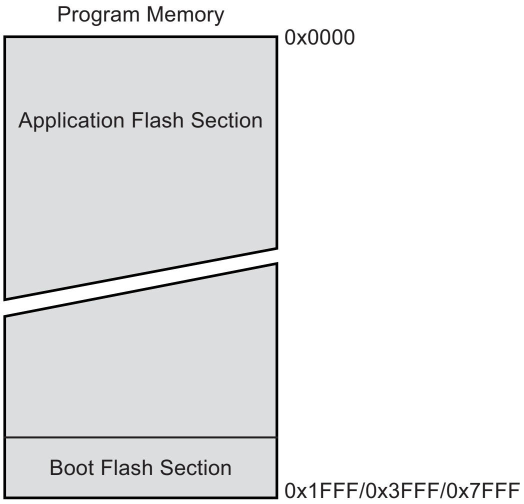
The ATmega164P-B/324P-B/644P-B contains 16/32/64Kbytes on-chip in-system reprogrammable flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the flash is organized as 32/64 × 16. For software security, the flash program memory space is divided into two sections, boot program section and application program section.

The flash memory has an endurance of at least 10,000 write/erase cycles. The ATmega164P-B/324P-B/644P-B program counter (PC) is 15/16 bits wide, thus addressing the 32/64K program memory locations. The operation of boot program section and associated boot lock bits for software protection are described in detail in Section 27. “Memory Programming” on page 255. Section 27. “Memory Programming” on page 255 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 Section 7.6 “Instruction Execution Timing” on page 13.

Figure 8-1. Program Memory Map



8.3 SRAM Data Memory

Figure 8-2 shows how the ATmega164P-B/324P-B/644P-B SRAM Memory is organized.

The ATmega164P-B/324P-B/644P-B is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in the Opcode for the IN and OUT instructions. For the extended I/O space from $060 - $FF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

The first 4,352 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 4,096 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.

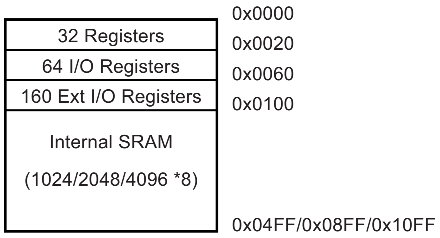
The direct addressing reaches the entire data space.

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 1024/2048/4096 bytes of internal data SRAM in the ATmega164P-B/324P-B/644P-B are all accessible through all these addressing modes. The register file is described in Section 7.4 “General Purpose Register File” on page 11.

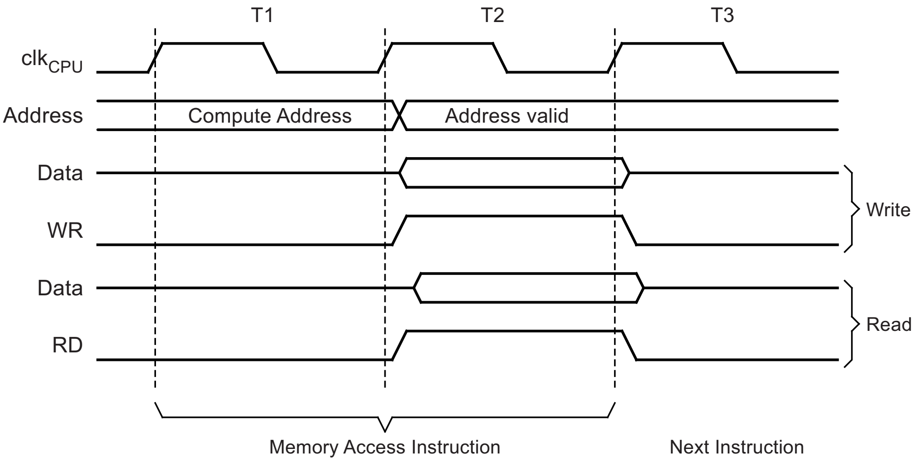
Figure 8-2. Data Memory Map for ATmega164P-B/324P-B/644P-B



8.3.1 Data Memory Access Times

This section describes the general access timing concepts for internal memory access. The internal data SRAM access is performed in two clk_{CPU} cycles as described in Figure 8-3.

Figure 8-3. On-chip Data SRAM Access Cycles



8.4 EEPROM Data Memory

The ATmega164P-B/324P-B/644P-B contains 512/1/2Kbytes 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 Section 27.6 "Parallel Programming Parameters, Pin Mapping, and Commands" on page 259, Section 27.8 "Serial Downloading" on page 270, and Section 27.10 "Programming via the JTAG Interface" on page 274 respectively.

8.4.1 EEPROM Read/Write Access

The EEPROM access registers are accessible in the I/O space. See Section 8.6 "Register Description" on page 19 for details. The write access time for the EEPROM is given in Table 8-2 on page 21. A self-timing function, however, lets the user software detect when the next byte can be written. If the user code contains instructions that write the EEPROM, some precautions must be taken. In heavily filtered power supplies, V_{CC} is likely to rise or fall slowly on power-up/down. This causes the device for some period of time to run at a voltage lower than specified as minimum for the clock frequency used. See Section 8.4.2 "Preventing EEPROM Corruption" on page 18 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.

8.4.2 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.

8.5 I/O Memory

The I/O space definition of the ATmega164P-B/324P-B/644P-B is shown in Section 30. "Register Summary" on page 323.

All ATmega164P-B/324P-B/644P-B 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 ATmega164P-B/324P-B/644P-B 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.

The ATmega164P-B/324P-B/644P-B contains three general purpose I/O registers, see Section 8.6 “Register Description” on page 19. 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.

8.6 Register Description

8.6.1 EEARH and EEARL – The EEPROM Address Register

Bit	15	14	13	12	11	10	9	8	
0x22 (0x42)	–	–	–	–	**EEAR11**	**EEAR10**	**EEAR9**	**EEAR8**	**EEARH**
0x21 (0x41)	**EEAR7**	**EEAR6**	**EEAR5**	**EEAR4**	**EEAR3**	**EEAR2**	**EEAR1**	**EEAR0**	**EEARL**
	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	X	X	X	X	
	X	X	X	X	X	X	X	X	

- **Bits 15:12 – Reserved**

These bits are reserved bits in the ATmega164P-B/324P-B/644P-B and will always read as zero.

- **Bits 11:0 – EEAR8:0: EEPROM Address**

The EEPROM Address Registers – EEARH and EEARL specify the EEPROM address in the 512/1K/2Kbytes EEPROM space. The EEPROM data bytes are addressed linearly between 0 and 511/1023/2047. The initial value of EEAR is undefined. A proper value must be written before the EEPROM may be accessed.

8.6.2 EEDR – The EEPROM Data Register

Bit	7	6	5	4	3	2	1	0	
0x20 (0x40)	**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.

8.6.3 EECR – The EEPROM Control Register

Bit	7	6	5	4	3	2	1	0	
0x1F (0x3F)	–	–	**EEPM1**	**EEPM0**	**EERIE**	**EEMPE**	**EEPE**	**EERE**	**EECR**
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	X	X	0	0	X	0	

- **Bits 7:6 – Reserved**

These bits are reserved bits in the ATmega164P-B/324P-B/644P-B and will always read as zero.

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

The EEPROM programming mode bit setting defines which programming action that will be triggered when writing EEPE. It is possible to program data in one atomic operation (erase the old value and program the new value) or to split the erase and write operations in two different operations. The programming times for the different modes are shown in Table 8-1 on page 20. While EEPE is set, any write to EEPMn will be ignored. During reset, the EEPMn bits will be reset to 0b00 unless the EEPROM is busy programming.

Table 8-1. EEPROM Mode Bits

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

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

Writing EERIE to one enables the EEPROM ready interrupt if the I bit in SREG is set. Writing EERIE to zero disables the interrupt. The EEPROM ready interrupt generates a constant interrupt when EEPE is cleared.

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

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

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

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

1. Wait until EEPE becomes zero.
2. Wait until 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 EEMPE bit while writing a zero to EEPE in EECR.
6. Within four clock cycles after setting EEMPE, write a logical one to EEPE.

The EEPROM can not be programmed during a CPU write to the flash memory. The software must check that the flash programming is completed before initiating a new EEPROM write. 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 Section 27. “Memory Programming” on page 255 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 EEPE bit is cleared by hardware. The user software can poll this bit and wait for a zero before writing the next byte. When EEPE has been set, the CPU is halted for two cycles before the next instruction is executed.

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

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

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

The calibrated Oscillator is used to time the EEPROM accesses. Table 8-2 on page 21 lists the typical programming time for EEPROM access from the CPU.

Table 8-2. EEPROM Programming Time

Symbol	Number of Calibrated RC Oscillator Cycles	Typical Programming Time
EEPROM write (from CPU)	26,368	3.3ms

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(1)

```
    EEPROM_write:
        ; Wait for completion of previous write
        sbic   EECR,EEPE
        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 EEMPE
        sbi    EECR,EEMPE
        ; Start eeprom write by setting EEPE
        sbi    EECR,EEPE
        ret
```

C Code Example(1)

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

Note: 1. See Section 4. “About Code Examples” on page 8.

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(1)

```
EEPROM_read:
      ; Wait for completion of previous write
      sbic   EECR,EEPE
      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(1)

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

Note: 1. See Section 4. “About Code Examples” on page 8.

8.6.4 GPIOR2 – General Purpose I/O Register 2

Bit	7	6	5	4	3	2	1	0	
0x2B (0x4B)	**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	

8.6.5 GPIOR1 – General Purpose I/O Register 1

Bit	7	6	5	4	3	2	1	0	
0x2A (0x4A)	**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	

8.6.6 GPIOR0 – General Purpose I/O Register 0

Bit	7	6	5	4	3	2	1	0	
0x1E (0x3E)	**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	

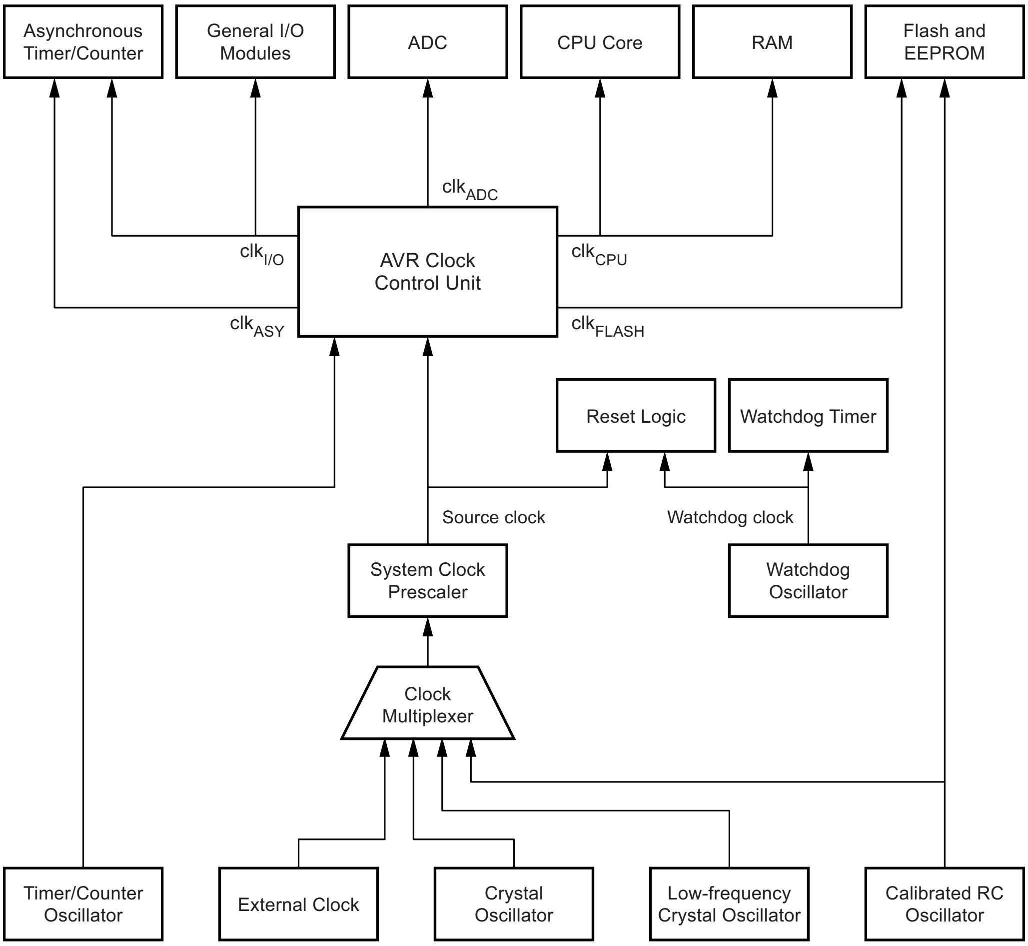
Note: 1. SRWn1 = SRW11 (upper sector) or SRW01 (lower sector), SRWn0 = SRW10 (upper sector) or SRW00 (lower sector). The ALE pulse in period T4 is only present if the next instruction accesses the RAM (internal or external).

9. System Clock and Clock Options

9.1 Clock Systems and their Distribution

Figure 9-1 presents the principal clock systems in the AVR and their distribution. All of the clocks need not be active at a given time. In order to reduce power consumption, the clocks to modules not being used can be halted by using different sleep modes, as described in Section 10. "Power Management and Sleep Modes" on page 34. The clock systems are detailed below.

Figure 9-1. Clock Distribution



9.1.1 CPU Clock – clk_{CPU}

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

9.1.2 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, TWI address recognition in all sleep modes.

9.1.3 Flash Clock – clk_{FLASH}

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

9.1.4 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 32kHz clock crystal. The dedicated clock domain allows using this Timer/Counter as a real-time counter even when the device is in sleep mode.

9.1.5 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.

9.2 Clock Sources

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

Table 9-1. Device Clocking Options Select(1)

Device Clocking Option	CKSEL3..0
Low power crystal oscillator	1111 - 1000
Full swing crystal oscillator	0111 - 0110
Low frequency crystal oscillator	0101 - 0100
Internal 128kHz RC oscillator	0011
Calibrated internal RC oscillator	0010
External clock	0000
Reserved	0001

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

9.2.1 Default Clock Source

The device is shipped with internal RC oscillator at 8.0MHz and with the fuse CKDIV8 programmed, resulting in 1.0MHz system clock. The startup time is set to maximum and time-out period enabled. (CKSEL = “0010”, SUT = “10”, CKDIV8 = “0”). The default setting ensures that all users can make their desired clock source setting using any available programming interface.

9.2.2 Clock Startup Sequence

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

To ensure sufficient V_{CC}, the device issues an internal reset with a time-out delay (t_{TOUT}) after the device reset is released by all other reset sources. Section 10.11.7 "On-chip Debug System" on page 37 describes the start conditions for the internal reset. The delay (t_{TOUT}) is timed from the watchdog oscillator and the number of cycles in the delay is set by the SUTx and CKSELx fuse bits. The selectable delays are shown in Table 9-2. The frequency of the watchdog oscillator is voltage dependent as shown in Section 29. "Typical Characteristics" on page 297.

Table 9-2. Number of Watchdog Oscillator Cycles

Typical Time-out (V_{CC} = 5.0V)	Typical Time-out (V_{CC} = 3.0V)	Number of Cycles
0ms	0ms	0
4.1ms	4.3ms	512
65ms	69ms	8K (8,192)

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

The oscillator is required to oscillate for a minimum number of cycles before the clock is considered stable. An internal ripple counter monitors the oscillator output clock, and keeps the internal reset active for a given number of clock cycles. The reset is then released and the device will start to execute. The recommended oscillator start-up time is dependent on the clock type, and varies from 6 cycles for an externally applied clock to 32Kcycles for a low frequency crystal.

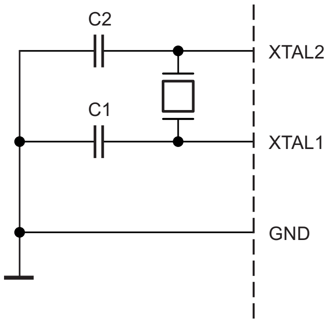
The start-up sequence for the clock includes both the time-out delay and the start-up time when the device starts up from reset. When starting up from power-save or power-down mode, Vcc is assumed to be at a sufficient level and only the start-up time is included.

9.2.3 Clock Source Connections

The pins 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 9-2 on page 25. 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. For ceramic resonators, the capacitor values given by the manufacturer should be used.

Figure 9-2. Crystal Oscillator Connections



9.3 Low Power Crystal Oscillator

This crystal oscillator is a low power oscillator, with reduced voltage swing on the XTAL2 output. It gives the lowest power consumption, but is not capable of driving other clock inputs, and may be more susceptible to noise in noisy environments. In these cases, refer to Section 9.4 "Full Swing Crystal Oscillator" on page 27.

Some initial guidelines for choosing capacitors for use with crystals are given in Table 9-3. The crystal should be connected as described in Section 9.2.3 "Clock Source Connections" on page 25.

The low power 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 9-3.

Table 9-3. Low Power Crystal Oscillator Operating Modes(1)

Frequency Range (MHz)	CKSEL3..1(2)	Recommended Range for Capacitors C1 and C2 (pF)
0.4 - 0.9	100(3)	–
0.9 - 3.0	101	12 - 22
3.0 - 8.0	110	12 - 22
8.0 - 16.0	111	12 - 22

Notes:
1. If the crystal 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. It must be ensured that the resulting divided clock meets the frequency specification of the device.
2. This is the recommended CKSEL settings for the different frequency ranges.
3. 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 9-4.

Table 9-4. Start-up Times for the Low Power Crystal Oscillator Clock Selection

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

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.

9.4 Full Swing Crystal Oscillator

This crystal oscillator is a full swing oscillator, with rail-to-rail swing on the XTAL2 output. This is useful for driving other clock inputs and in noisy environments. The current consumption is higher than the Low Power Crystal Oscillator in Section 9.3 on page 26. Note that the full swing crystal oscillator will only operate for Vcc = 2.7 to 5.5V.

Some initial guidelines for choosing capacitors for use with crystals are given in Table 9-6. The crystal should be connected as described in Section 9.2.3 "Clock Source Connections" on page 25.

The operating mode is selected by the fuses CKSEL3..1 as shown in Table 9-5.

Table 9-5. Full Swing Crystal Oscillator Operating Modes

Frequency Range[1] (MHz)	CKSEL3..1	Recommended Range for Capacitors C1 and C2 (pF)
0.4 - 16	011	12 - 22

Notes: 1. If the crystal 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. It must be ensured that the resulting divided clock meets the frequency specification of the device.

Table 9-6. Start-up Times for the Full Swing Crystal Oscillator Clock Selection

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

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.

9.5 Low Frequency Crystal Oscillator

The low-frequency crystal oscillator is optimized for use with a 32.768kHz watch crystal. When selecting crystals, load capasitance and crystal's equivalent series resistance, ESR must be taken into consideration. Both values are specified by the crystal vendor. ATmega164P-B/324P-B/644P-B oscillator is optimized for very low power consumption, and thus when selecting crystals, see Table 9-7 on page 28 for maximum ESR recommendations on 9pF and 12.5pF crystals.

Table 9-7. Maximum ESR Recommendation for 32.768kHz Watch Crystal

Crystal CL (pF)	Max ESR [kΩ](1)
9.0	65
12.5	30

Note: 1. Maximum ESR is typical value based on characterization

The Low-frequency crystal oscillator provides an internal load capacitance, see Table on page 28 at each TOSC pin.

Table 9-8. Capacitance for Low-frequency Oscillator

Device	32kHz Osc. Type	Cap (Xtal1/Tosc1)	Cap (Xtal2/Tosc2)
ATmega164P-B/324P-B/644P-B	System Osc.	18pF	8pF
	Timer Osc.	6pF	6pF

The capacitance (Ce + Ci) needed at each TOSC pin can be calculated by using:

$$Ce + Ci = 2 \cdot CL - C_s$$

where:

Ce - is optional external capacitors as described in Figure 9-2 on page 25

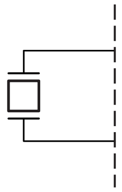
Ci - is the pin capacitance in Table 9-8 on page 28

CL - is the load capacitance for a 32.768kHz crystal specified by the crystal vendor.

C_S - is the total stray capacitance for one TOSC pin.

Crystals specifying load capacitance (CL) higher than the ones given in the Table 9-8 on page 28, require external capacitors applied as described in Figure 9-2 on page 25.

Figure 9-3. Crystal Oscillator Connections



Crystals specifying load capacitance (CL) higher than listed in Table 9-8 on page 28, require external capacitors applied as described in Figure 9-2 on page 25.

To find suitable load capacitance for a 32.768kHz crysal, please consult the crystal datasheet.

When this oscillator is selected, start-up times are determined by the SUT Fuses and CKSEL0 as shown in Table 9-9.

Table 9-9. Start-up Times for the Low Frequency Crystal Oscillator Clock Selection

Power Conditions	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	CKSEL0	SUT1..0
BOD enabled	1K CK	14CK(1)	0	00
Fast rising power	1K CK	14CK + 4.1ms(1)	0	01
Slowly rising power	1K CK	14CK + 65ms(1)	0	10
Reserved			0	11
BOD enabled	32K CK	14CK	1	00
Fast rising power	32K CK	14CK + 4.1ms	1	01
Slowly rising power	32K CK	14CK + 65ms	1	10
Reserved			1	11

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

9.6 Calibrated Internal RC Oscillator

By default, the Internal RC oscillator provides an approximate 8MHz clock. Though voltage and temperature dependent, this clock can be very accurately calibrated by the user. See Table 28-3 on page 290 and Section 29.1.8 "Internal Oscillator Speed" on page 304 and Section 29.2.8 "Internal Oscillator Speed" on page 312 for more details. The device is shipped with the CKDIV8 Fuse programmed. See Section 9.11 "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 9-10. If selected, it will operate with no external components. During reset, hardware loads the pre-programmed default 3V calibration value into the OSCCAL register and thereby automatically calibrates the RC oscillator for 3V operation. If the device is to be used at 5V then the alternate RC oscillator 5V calibration byte (Table 26-5 on page 248) can be read from signature row and stored into the OSCCAL register by the user application program for better 5V frequency accuracy. The accuracy of this calibration is shown as factory calibration in Table 28-3 on page 290.

By changing the OSCCAL register from SW, see Section 9.12.1 "OSCCAL – Oscillator Calibration Register" on page 32, it is possible to get a higher calibration accuracy than by using the factory calibration. The accuracy of this calibration is shown as user calibration in Table 28-3 on page 290.

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 Section 27.4 "Calibration Byte" on page 258.

Table 9-10. Internal Calibrated RC Oscillator Operating Modes

Frequency Range(2) (MHz)	CKSEL3..0
7.3 - 8.1	0010(1)

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

2. If 8MHz frequency exceeds the specification of the device (depends on V_{CC}), the CKDIV8 fuse can be programmed in order to divide the internal frequency by 8.

When this oscillator is selected, start-up times are determined by the SUT fuses as shown in Table 9-11 on page 30.

Table 9-11. 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 + 4.1ms	01
Slowly rising power	6 CK	14CK + 65ms	10[1]
Reserved			11

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

9.7 128kHz Internal Oscillator

The 128kHz internal oscillator is a low power oscillator providing a clock of 128kHz. The frequency is nominal at 3V and 25°C. This clock may be select as the system clock by programming the CKSEL Fuses to “0011” as shown in Table 9-12.

Table 9-12. 128kHz Internal Oscillator Operating Modes[2]

Nominal Frequency	CKSEL3..0
128kHz	0011

Note: 1. Note that the 128kHz oscillator is a very low power clock source, and is not designed for high accuracy.

When this clock source is selected, start-up times are determined by the SUT Fuses as shown in Table 9-13.

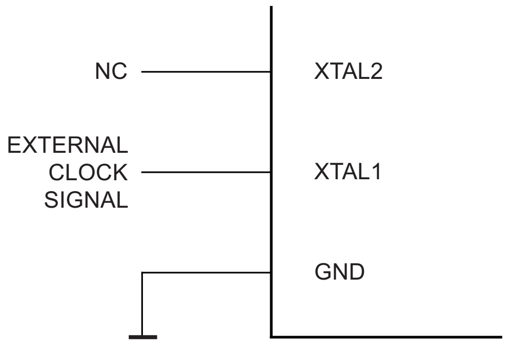
Table 9-13. Start-up Times for the 128kHz Internal Oscillator

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

9.8 External Clock

To drive the device from an external clock source, XTAL1 should be driven as shown in Figure 9-4. To run the device on an external clock, the CKSEL Fuses must be programmed to “0000”.

Figure 9-4. External Clock Drive Configuration



When this clock source is selected, start-up times are determined by the SUT Fuses as shown in Table 9-15.

Table 9-14. Crystal Oscillator Clock Frequency

Nominal Frequency	CKSEL3..0
0 - 16MHz	0000

Table 9-15. Start-up Times for the External 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 + 4.1ms	01
Slowly rising power	6 CK	14CK + 65ms	10
Reserved			11

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

Note that the system clock prescaler can be used to implement run-time changes of the internal clock frequency while still ensuring stable operation. Refer to Section 9.11 "System Clock Prescaler" on page 31 for details.

9.9 Timer/Counter Oscillator

ATmega164P-B/324P-B/644P-B uses the same type of crystal oscillator for low-frequency crystal oscillator and Timer/Counter oscillator. See Section 9.5 "Low Frequency Crystal Oscillator" on page 28 for details on the oscillator and crystal requirements.

The device can operate its Timer/Counter2 from an external 32.768kHz watch crystal or a external clock source. See Section 9.2.3 "Clock Source Connections" on page 25 for details.

Applying an external clock source to TOSC1 can be done if EXTCLK in the ASSR register is written to logic one. See Section 17.11.4 "OCR2A – Output Compare Register A" on page 136 for further description on selecting external clock as input instead of a 32.768kHz watch crystal.

9.10 Clock Output Buffer

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

9.11 System Clock Prescaler

The ATmega164P-B/324P-B/644P-B has a system clock prescaler, and the system clock can be divided by setting the Section 9.12.2 "CLKPR – Clock Prescale Register" on page 32. This feature can be used to decrease the system clock frequency and the power consumption when the requirement for processing power is low. This can be used with all clock source options, and it will affect the clock frequency of the CPU and all synchronous peripherals. $clk_{I/O}$, clk_{ADC}, clk_{CPU}, and clk_{FLASH} are divided by a factor as shown in Table 9-16 on page 33.

When switching between prescaler settings, the system clock prescaler ensures that no glitches occurs in the clock system. It also ensures that no intermediate frequency is higher than neither the clock frequency corresponding to the previous setting, nor the clock frequency corresponding to the new setting.

The ripple counter that implements the prescaler runs at the frequency of the undivided clock, which may be faster than the CPU's clock frequency. Hence, it is not possible to determine the state of the prescaler - even if it were readable, and the exact time it takes to switch from one clock division to the other cannot be exactly predicted. From the time the CLKPS values are written, it takes between T1 + T2 and T1 + 2 × T2 before the new clock frequency is active. In this interval, 2 active clock edges are produced. Here, T1 is the previous clock period, and T2 is the period corresponding to the new prescaler setting.

To avoid unintentional changes of clock frequency, a special write procedure must 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.

9.12 Register Description

9.12.1 OSCCAL – Oscillator Calibration Register

Bit	7	6	5	4	3	2	1	0	
(0x66)	**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. A pre-programmed calibration value is automatically written to this register during chip reset, giving the Factory calibrated frequency as specified in Table 28-3 on page 290. The application software can write this register to change the oscillator frequency. The oscillator can be calibrated to frequencies as specified in Table 28-3 on page 290. 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 8.8MHz. 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.

9.12.2 CLKPR – Clock Prescale Register

Bit	7	6	5	4	3	2	1	0	
(0x61)	**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 9-16 on page 33.

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 9-16. 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

10. Power Management and Sleep Modes

10.1 Overview

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

When enabled, the brown-out detector (BOD) actively monitors the power supply voltage during the sleep periods. To further save power, it is possible to disable the BOD in some sleep modes. See Section 10.3 "BOD Disable" on page 35 for more details.

10.2 Sleep Modes

Figure 9-1 on page 23 presents the different clock systems in the ATmega164P-B/324P-B/644P-B, and their distribution. The figure is helpful in selecting an appropriate sleep mode. Table 10-1 shows the different sleep modes, their wake up sources and BOD disable ability.

Table 10-1. 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	INT2:0 and Pin Change	TWI Address Match	Timer2	SPM/ EEPROM Ready	ADC	WDT Interrupt	Other I/O	Software BOD Disdable
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X	
ADCNRM				X	X	X	X(2)	X	X	X(2)	X	X	X		
Power-down								X	X				X		X
Power-save					X		X(2)	X	X	X			X		X
Standby(1)						X		X	X				X		X
Extended standby					X(2)	X	X(2)	X	X	X			X		X

Notes: 1. Only recommended with external crystal or resonator selected as clock source.

2. If Timer/Counter2 is running in asynchronous mode.

To enter any of the sleep modes, the SE bit in SMCR must be written to logic one and a SLEEP instruction must be executed. The SM2, SM1, and SM0 bits in the SMCR register select which sleep mode will be activated by the SLEEP instruction. See Table 10-2 on page 38 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.

10.3 BOD Disable

When the brown-out detector (BOD) is enabled by BODLEVEL fuses, Table 27-3 on page 256, the BOD is actively monitoring the power supply voltage during a sleep period. To save power, it is possible to disable the BOD by software for some of the sleep modes, see Table 10-1 on page 34. The sleep mode power consumption will then be at the same level as when BOD is globally disabled by fuses. If BOD is disabled in software, the BOD function is turned off immediately after entering the sleep mode. Upon wake-up from sleep, BOD is automatically enabled again. This ensures safe operation in case the V_{CC} level has dropped during the sleep period.

When the BOD has been disabled, the wake-up time from sleep mode will be approximately 60µs to ensure that the BOD is working correctly before the MCU continues executing code.

BOD disable is controlled by bit 6, BODS (BOD Sleep) in the control register MCUCR, see Section 10.12.2 “MCUCR – MCU Control Register” on page 38. Writing this bit to one turns off the BOD in relevant sleep modes, while a zero in this bit keeps BOD active. Default setting keeps BOD active, i.e. BODS set to zero.

Writing to the BODS bit is controlled by a timed sequence and an enable bit, see Section 10.12.2 “MCUCR – MCU Control Register” on page 38.

10.4 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, 2-wire serial interface, Timer/Counters, watchdog, and the interrupt system to continue operating. This sleep mode basically halts clk_{CPU} and clk_{FLASH}, while allowing the other clocks to run.

Idle mode enables the MCU to wake up from external triggered interrupts as well as internal ones like the 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.

10.5 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, 2-wire serial interface address match, Timer/Counter2 and the watchdog to continue operating (if enabled). This sleep mode basically halts clkI/O, clkCPU, and clkFLASH, 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 system reset, a watchdog interrupt, a brown-out reset, a 2-wire serial interface interrupt, a Timer/Counter2 interrupt, an SPM/EEPROM ready interrupt, an external level interrupt on INT7:4 or a pin change interrupt can wake up the MCU from ADC noise reduction mode.

10.6 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 2-wire serial interface, and the watchdog continue operating (if enabled). Only an external reset, a watchdog reset, a brown-out reset, 2-wire serial interface address match, an external level interrupt on PCINT7:4, an external interrupt on INT2:0, or a pin change interrupt can wake up the MCU. This sleep mode basically halts all generated clocks, allowing operation of asynchronous modules only.

Note that if a level triggered interrupt is used for wake-up from power-down mode, the changed level must be held for some time to wake up the MCU. Refer to Section 13. “External Interrupts” on page 53 for details.

When waking up from power-down mode, there is a delay from the wake-up condition occurs until the wake-up becomes effective. This allows the clock to restart and become stable after having been stopped. The wake-up period is defined by the same CKSEL fuses that define the reset time-out period, as described in Section 9.2 “Clock Sources” on page 24.

10.7 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 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 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 the Timer/Counter2 is not using the asynchronous clock, the Timer/Counter oscillator is stopped during sleep. If the 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 the Timer/Counter2.

10.8 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.

10.9 Extended Standby Mode

When the SM2..0 bits are 111 and an external crystal/resonator clock option is selected, the SLEEP instruction makes the MCU enter extended standby mode. This mode is identical to power-save mode with the exception that the oscillator is kept running. From extended standby mode, the device wakes up in six clock cycles.

10.10 Power Reduction Register

The power reduction register (PRR), see Section 10.12.3 "PRR0 – Power Reduction Register 0" on page 39, provides a method to stop the clock to individual peripherals to reduce power consumption. The current state of the peripheral is frozen and the I/O registers can not be read or written. Resources used by the peripheral when stopping the clock will remain occupied, hence the peripheral should in most cases be disabled before stopping the clock. Waking up a peripheral, which is done by clearing the bit in PRR, puts the peripheral in the same state as before shutdown.

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

10.11 Minimizing Power Consumption

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

10.11.1 Analog to Digital Converter

If enabled, the ADC will be enabled in all sleep modes. To save power, the ADC should be disabled before entering any sleep mode. When the ADC is turned off and on again, the next conversion will be an extended conversion. Refer to Section 23. "ADC - Analog-to-digital Converter" on page 207 for details on ADC operation.

10.11.2 Analog Comparator

When entering Idle mode, the analog comparator should be disabled if not used. When entering ADC noise reduction mode, the analog comparator should be disabled. In other sleep modes, the analog comparator is automatically disabled. However, if the analog comparator is set up to use the internal voltage reference as input, the analog comparator should be disabled in all sleep modes. Otherwise, the internal voltage reference will be enabled, independent of sleep mode. Refer to Section 22. "AC - Analog Comparator" on page 204 for details on how to configure the analog comparator.

10.11.3 Brown-out Detector

If the brown-out detector is not needed by the application, this module should be turned off. If the brown-out detector is enabled by the BODLEVEL fuses, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to Section 11.1.4 “Brown-out Detection” on page 42 for details on how to configure the brown-out detector.

10.11.4 Internal Voltage Reference

The internal voltage reference will be enabled when needed by the brown-out detection, the analog comparator or the ADC. If these modules are disabled as described in the sections above, the internal voltage reference will be disabled and it will not be consuming power. When turned on again, the user must allow the reference to start up before the output is used. If the reference is kept on in sleep mode, the output can be used immediately. Refer to Section 11.2 “Internal Voltage Reference” on page 43 for details on the start-up time.

10.11.5 Watchdog Timer

If the watchdog timer is not needed in the application, the module should be turned off. If the watchdog timer is enabled, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to Section 11.3 “Watchdog Timer” on page 44 for details on how to configure the watchdog timer.

10.11.6 Port Pins

When entering a sleep mode, all port pins should be configured to use minimum power. The most important is then to ensure that no pins drive resistive loads. In sleep modes where both the I/O clock ($clk_{I/O}$) and the ADC clock (clk_{ADC}) are stopped, the input buffers of the device will be disabled. This ensures that no power is consumed by the input logic when not needed. In some cases, the input logic is needed for detecting wake-up conditions, and it will then be enabled. Refer to the section Section 14.2.5 “Digital Input Enable and Sleep Modes” on page 61 for details on which pins are enabled. If the input buffer is enabled and the input signal is left floating or have an analog signal level close to $V_{CC}/2$, the input buffer will use excessive power.

For analog input pins, the digital input buffer should be disabled at all times. An analog signal level close to $V_{CC}/2$ on an input pin can cause significant current even in active mode. Digital input buffers can be disabled by writing to the digital input disable registers (DIDR1 and DIDR0). Refer to Section 22.3.3 “DIDR1 – Digital Input Disable Register 1” on page 206 and Section 23.9.5 “DIDR0 – Digital Input Disable Register 0” on page 224 for details.

10.11.7 On-chip Debug System

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

There are three alternative ways to disable the OCD system:

- Disable the OCDEN fuse
- Disable the JTAGEN fuse
- Write one to the JTD bit in MCUCR

10.12 Register Description

10.12.1 SMCR – Sleep Mode Control Register

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

Bit	7	6	5	4	3	2	1	0	
0x33 (0x53)	**–**	**–**	**–**	**–**	**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 10-2.

Table 10-2. 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	Extended standby(1)

Note: 1. Standby modes are only recommended for use with external crystals or resonators.

- **Bit 0 – SE: Sleep Enable**

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

10.12.2 MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	**JTD**	**BODS**	**BODSE**	**PUD**	**–**	**–**	**IVSEL**	**IVCE**	**MCUCR**
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 6 – BODS: BOD Sleep**

The BODS bit must be written to logic one in order to turn off BOD during sleep, see Table 10-1 on page 34. Writing to the BODS bit is controlled by a timed sequence and an enable bit, BODSE in MCUCR. To disable BOD in relevant sleep modes, both BODS and BODSE must first be set to one. Then, to set the BODS bit, BODS must be set to one and BODSE must be set to zero within four clock cycles.

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

- **Bit 5 – BODSE: BOD Sleep Enable**

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

10.12.3 PRR0 – Power Reduction Register 0

Bit	7	6	5	4	3	2	1	0	
(0x64)	PRTWI	PRTIM2	PRTIM0	PRUSART1	PRTIM1	PRSPI	PRUSART0	PRADC	PRR0
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 – PRTWI: Power Reduction TWI**

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

- **Bit 6 – PRTIM2: Power Reduction Timer/Counter2**

Writing a logic one to this bit shuts down the Timer/Counter2 module in synchronous mode (AS2 is 0). When the Timer/Counter2 is enabled, operation will continue like before the shutdown.

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

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

- **Bit 4 – PRUSART1: Power Reduction USART1**

Writing a logic one to this bit shuts down the USART1 by stopping the clock to the module. When waking up the USART1 again, the USART1 should be reinitialized to ensure proper operation.

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

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

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

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

- **Bit 1 – PRUSART0: Power Reduction USART0**

Writing a logic one to this bit shuts down the USART0 by stopping the clock to the module. When waking up the USART0 again, the USART0 should be reinitialized to ensure proper operation.

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

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

10.12.4 PRR1 – Power Reduction Register 1

Bit	7	6	5	4	3	2	1	0	
(0x65)	–	–	–	–	–	–	–	PRTIM3	PRR1
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:1 – Reserved**

- **Bit 0 – PRTIM3: Power Reduction Timer/Counter3**

Writing a logic one to this bit shuts down the Timer/Counter3 module. When the Timer/Counter3 is enabled, operation will continue like before the shutdown.

11. System Control and Reset

11.1 Resetting the AVR

During reset, all I/O registers are set to their initial values, and the program starts execution from the reset vector. The instruction placed at the reset vector must be 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 11-1 on page 41 shows the reset logic. Section 28.5 “System and Reset Characteristics” on page 291 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.

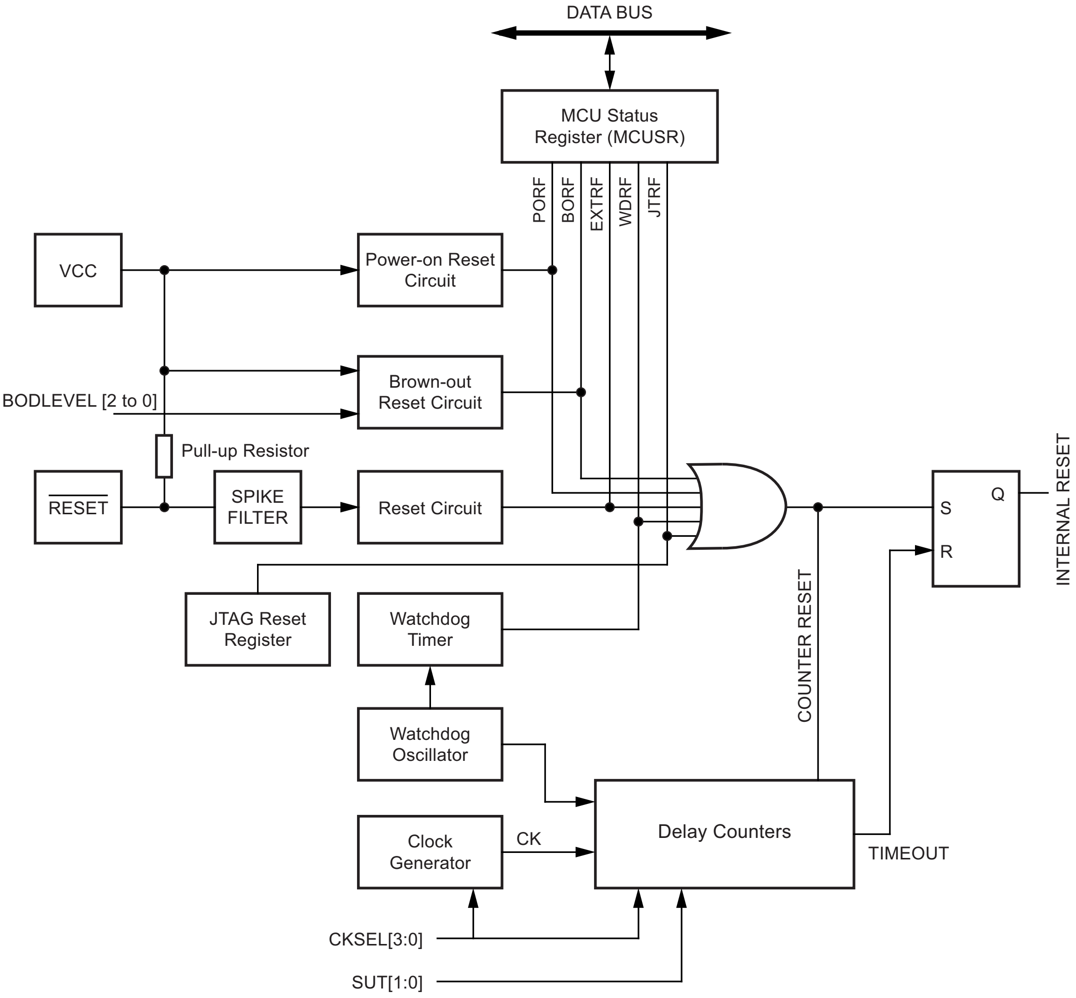
After all reset sources have gone inactive, a delay counter is invoked, stretching the internal reset. This allows the power to reach a stable level before normal operation starts. The time-out period of the delay counter is defined by the user through the SUT and CKSEL Fuses. The different selections for the delay period are presented in Section 9.2 “Clock Sources” on page 24.

11.1.1 Reset Sources

The ATmega164P-B/324P-B/644P-B 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 Section 25. “IEEE 1149.1 (JTAG) Boundary-scan” on page 231 for details.

Figure 11-1. Reset Logic



11.1.2 Power-on Reset

A power-on reset (POR) pulse is generated by an on-chip detection circuit. The detection level is defined in Section 28.5 “System and Reset Characteristics” on page 291. 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 11-2. MCU Start-up, $\overline{\text{RESET}}$ Tied to V_{CC}

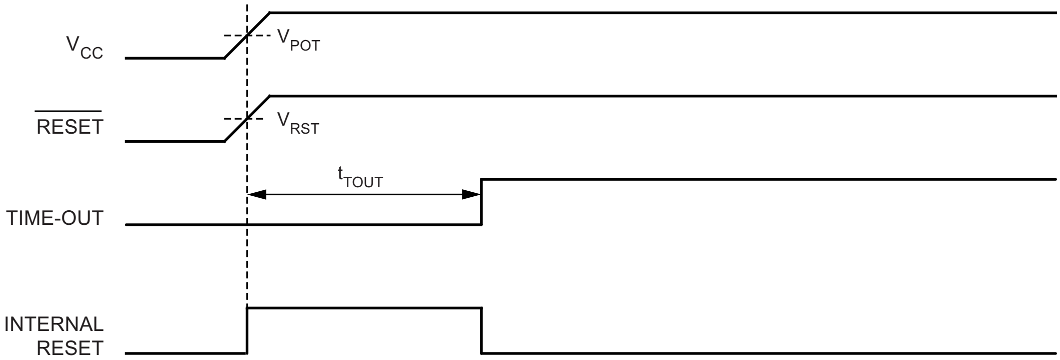
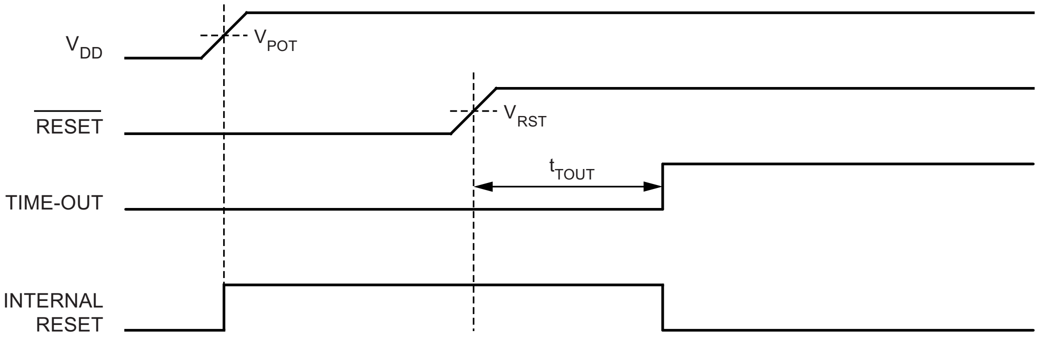


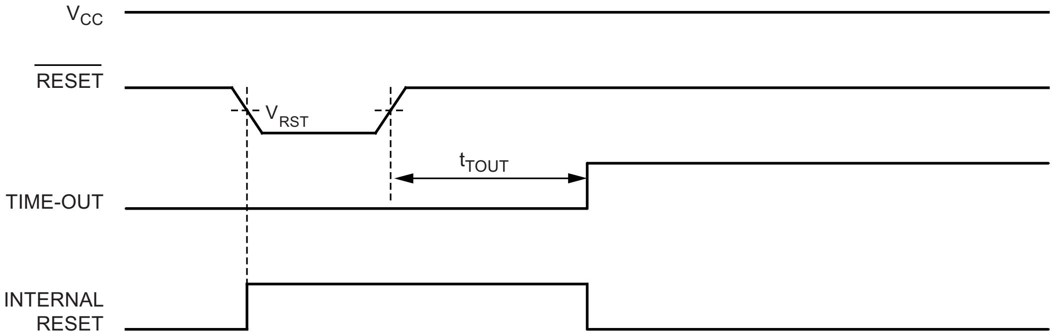
Figure 11-3. **MCU Start-up, $\overline{\text{RESET}}$ Extended Externally**



11.1.3 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 "System and Reset Characteristics" on page 291) 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 11-4. External Reset During Operation



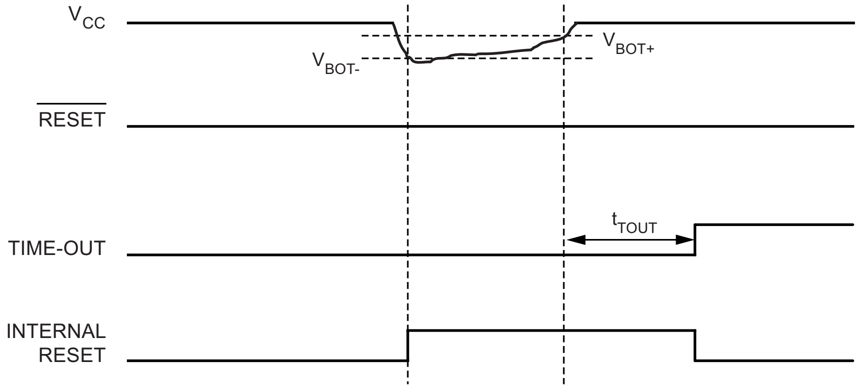
11.1.4 Brown-out Detection

ATmega164P-B/324P-B/644P-B has an on-chip brown-out detection (BOD) circuit for monitoring the V_{CC} level during operation by comparing it to a fixed trigger level. The trigger level for the BOD can be selected by the BODLEVEL fuses. The trigger level has a hysteresis to ensure spike free brown-out detection. The hysteresis on the detection level should be interpreted as $V_{BOT+} = V_{BOT} + V_{HYST}/2$ and $V_{BOT-} = V_{BOT} - V_{HYST}/2$.

When the BOD is enabled, and V_{CC} decreases to a value below the trigger level (V_{BOT-} in Figure 11-5 on page 43), the brown-out reset is immediately activated. When V_{CC} increases above the trigger level (V_{BOT+} in Figure 11-5 on page 43), the delay counter starts the MCU after the time-out period t_{TOUT} has expired.

The BOD circuit will only detect a drop in V_{CC} if the voltage stays below the trigger level for longer than t_{BOD} given in Section 28.5 "System and Reset Characteristics" on page 291.

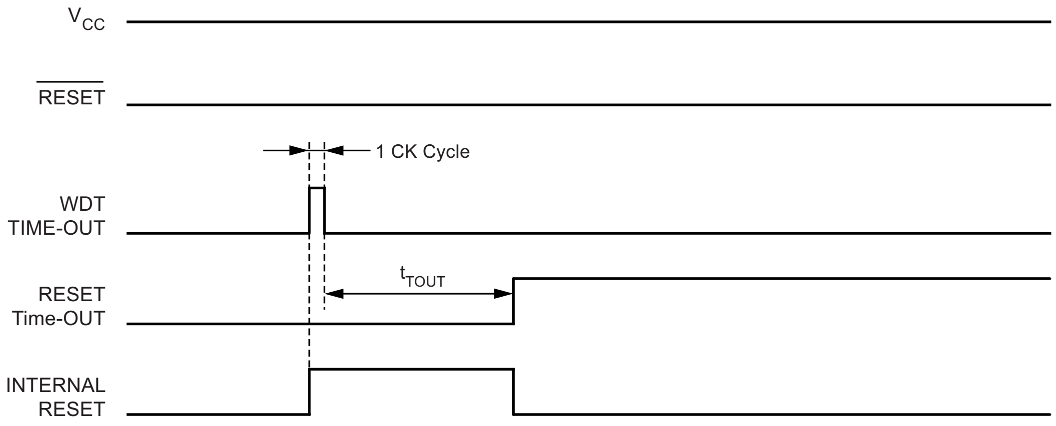
Figure 11-5. Brown-out Reset During Operation



11.1.5 Watchdog Reset

When the watchdog times out, it will generate a short reset pulse of one CK cycle duration. On the falling edge of this pulse, the delay timer starts counting the time-out period t_{TOUT}. Refer to Section 11.3 “Watchdog Timer” on page 44 for details on operation of the watchdog timer.

Figure 11-6. Watchdog Reset During Operation



11.2 Internal Voltage Reference

ATmega164P-B/324P-B/644P-B 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.

11.2.1 Voltage Reference Enable Signals and Start-up Time

The voltage reference has a start-up time that may influence the way it should be used. The start-up time is given in Section 28.5 “System and Reset Characteristics” on page 291. To save power, the reference is not always turned on. The reference is on during the following situations:

1. When the BOD is enabled (by programming the BODLEVEL [2:0] 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.

11.3 Watchdog Timer

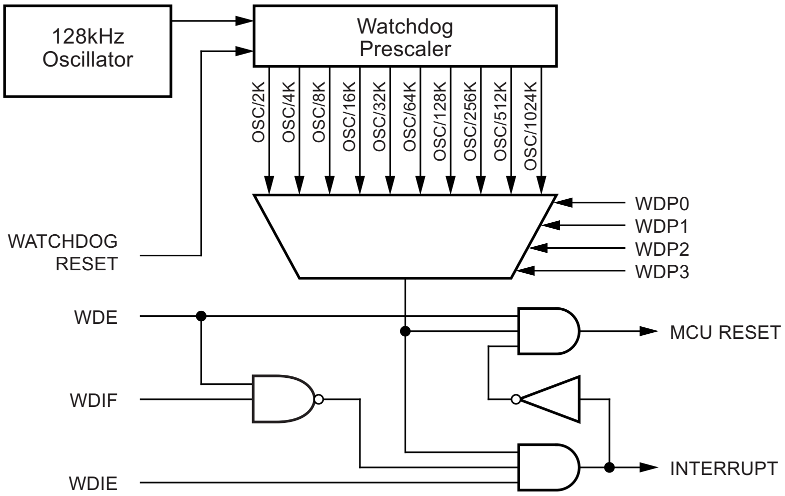
11.3.1 Features

- Clocked from separate on-chip oscillator
- 3 operating modes
  - Interrupt
  - System reset
  - Interrupt and system reset
- Selectable time-out period from 16ms to 8s
- Possible hardware fuse watchdog always on (WDTON) for fail-safe mode

11.3.2 Overview

ATmega164P-B/324P-B/644P-B has an enhanced watchdog timer (WDT). The WDT is a timer counting cycles of a separate on-chip 128kHz oscillator. The WDT gives an interrupt or a system reset when the counter reaches a given time-out value. In normal operation mode, it is required that the system uses the WDR - Watchdog Timer Reset - instruction to restart the counter before the time-out value is reached. If the system doesn't restart the counter, an interrupt or system reset will be issued.

Figure 11-7. Watchdog Timer



In Interrupt mode, the WDT gives an interrupt when the timer expires. This interrupt can be used to wake the device from sleep-modes, and also as a general system timer. One example is to limit the maximum time allowed for certain operations, giving an interrupt when the operation has run longer than expected. In System Reset mode, the WDT gives a reset when the timer expires. This is typically used to prevent system hang-up in case of runaway code. The third mode, Interrupt and System Reset mode, combines the other two modes by first giving an interrupt and then switch to system reset mode. This mode will for instance allow a safe shutdown by saving critical parameters before a system reset.

The watchdog always on (WDTON) fuse, if programmed, will force the watchdog timer to system reset mode. With the fuse programmed the system reset mode bit (WDE) and Interrupt mode bit (WDIE) are locked to 1 and 0 respectively. To further ensure program security, alterations to the watchdog set-up must follow timed sequences. The sequence for clearing WDE and changing time-out configuration is as follows:

1. In the same operation, write a logic one to the watchdog change enable bit (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, write the WDE and watchdog prescaler bits (WDP) as desired, but with the WDCE bit cleared. This must be done in one operation.

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

Assembly Code Example[1]

```
WDT_off:
        ; Turn off global interrupt
        cli
        ; Reset Watchdog Timer
        wdr
        ; Clear WDRF in MCUSR
        in    r16, MCUSR
        andi r16, ~(1<<WDRF)
        out   MCUSR, r16
        ; Write logical one to WDCE and WDE
        ; Keep old prescaler setting to prevent unintentional time-out
        in    r16, WDTCSR
        ori   r16, (1<<WDCE) | (1<<WDE)
        out   WDTCSR, r16
        ; Turn off WDT
        ldi   r16, (0<<WDE)
        out   WDTCSR, r16
        ; Turn on global interrupt
        sei
        ret
```

C Code Example[1]

```
void WDT_off(void)
{
        __disable_interrupt();
        __watchdog_reset();
        /* Clear WDRF in MCUSR */
        MCUSR &= ~(1<<WDRF);
        /* Write logical one to WDCE and WDE */
        /* Keep old prescaler setting to prevent unintentional time-out
*/
        WDTCSR |= (1<<WDCE) | (1<<WDE);
        /* Turn off WDT */
        WDTCSR = 0x00;
        __enable_interrupt();
}
```

Notes: 1. The example code assumes that the part specific header file is included.

2. If the Watchdog is accidentally enabled, for example by a runaway pointer or brown-out condition, the device will be reset and the watchdog Timer will stay enabled. If the code is not set up to handle the watchdog, this might lead to an eternal loop of time-out resets. To avoid this situation, the application software should always clear the watchdog system reset flag (WDRF) and the WDE control bit in the initialization routine, even if the watchdog is not in use.

The following code example shows one assembly and one C function for changing the time-out value of the Watchdog Timer.

Assembly Code Example(1)

```
       WDT_Prescaler_Change:
              ; Turn off global interrupt
              cli
              ; Reset Watchdog Timer
              wdr
              ; Start timed sequence
              in    r16, WDTCSR
              ori   r16, (1<<WDCE) | (1<<WDE)
              out   WDTCSR, r16
              ; -- Got four cycles to set the new values from here -
              ; Set new prescaler(time-out) value = 64K cycles (~0.5 s)
              ldi   r16, (1<<WDE) | (1<<WDP2) | (1<<WDP0)
              out   WDTCSR, r16
              ; -- Finished setting new values, used 2 cycles -
              ; Turn on global interrupt
              sei
              ret
```

C Code Example(1)

```
       void WDT_Prescaler_Change(void)
       {
              __disable_interrupt();
              __watchdog_reset();
              /* Start timed equence */
              WDTCSR |= (1<<WDCE) | (1<<WDE);
              /* Set new prescaler(time-out) value = 64K cycles (~0.5 s) */
              WDTCSR = (1<<WDE) | (1<<WDP2) | (1<<WDP0);
              __enable_interrupt();
       }
```

Notes: 1. The example code assumes that the part specific header file is included.

2. The watchdog timer should be reset before any change of the WDP bits, since a change in the WDP bits can result in a time-out when switching to a shorter time-out period.

11.4 Register Description

11.4.1 MCUSR – MCU Status Register

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

Bit	7	6	5	4	3	2	1	0	
0x34 (0x54)	**–**	**–**	**–**	**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.

11.4.2 WDTCSR – Watchdog Timer Control Register

Bit	7	6	5	4	3	2	1	0	
(0x60)	**WDIF**	**WDIE**	**WDP3**	**WDCE**	**WDE**	**WDP2**	**WDP1**	**WDP0**	**WDTCSR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	0	0	0	

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

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

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

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

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

Table 11-1. Watchdog Timer Configuration

WDTON	WDE	WDIE	Mode	Action on Time-out
1	0	0	Stopped	None
1	0	1	Interrupt mode	Interrupt
1	1	0	System reset mode	Reset
1	1	1	Interrupt and system reset mode	Interrupt, then go to system reset mode
0	x	x	System reset mode	Reset

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

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

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

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

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

- **Bit 5, 2:0 - WDP3:0: Watchdog Timer Prescaler 3, 2, 1 and 0**

The WDP3:0 bits determine the watchdog timer prescaling when the watchdog timer is running. The different prescaling values and their corresponding time-out periods are shown in Table 11-2 on page 48.

Table 11-2. Watchdog Timer Prescale Select

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

12. Interrupts

12.1 Overview

This section describes the specifics of the interrupt handling as performed in ATmega164P-B/324P-B/644P-B. For a general explanation of the AVR interrupt handling, refer to Section 7.7 "Reset and Interrupt Handling" on page 14.

12.2 Interrupt Vectors in ATmega164P-B/324P-B/644P-B

Table 12-1. Reset and Interrupt Vectors

Vector No.	Program Address[2]	Source	Interrupt Definition
1	$0000[1]	RESET	External pin, power-on reset, brown-out reset, watchdog reset, and JTAG AVR reset
2	$0002	INT0	External interrupt request 0
3	$0004	INT1	External interrupt request 1
4	$0006	INT2	External interrupt request 2
5	$0008	PCINT0	Pin change interrupt request 0
6	$000A	PCINT1	Pin change interrupt request 1
7	$000C	PCINT2	Pin change interrupt request 2
8	$000E	PCINT3	Pin change interrupt request 3
9	$0010	WDT	Watchdog time-out interrupt
10	$0012	TIMER2_COMPA	Timer/Counter2 compare match A
11	$0014	TIMER2_COMPB	Timer/Counter2 compare match B
12	$0016	TIMER2_OVF	Timer/Counter2 overflow
13	$0018	TIMER1_CAPT	Timer/Counter1 capture event
14	$001A	TIMER1_COMPA	Timer/Counter1 compare match A
15	$001C	TIMER1_COMPB	Timer/Counter1 compare match B
16	$001E	TIMER1_OVF	Timer/Counter1 overflow
17	$0020	TIMER0_COMPA	Timer/Counter0 compare match A
18	$0022	TIMER0_COMPB	Timer/Counter0 compare match B
19	$0024	TIMER0_OVF	Timer/Counter0 overflow
20	$0026	SPI_STC	SPI serial transfer complete
21	$0028	USART0_RX	USART0 Rx complete
22	$002A	USART0_UDRE	USART0 data register empty
23	$002C	USART0_TX	USART0 Tx complete
24	$002E	ANALOG_COMP	Analog comparator
25	$0030	ADC	ADC conversion complete
26	$0032	EE_READY	EEPROM ready
27	$0034	TWI	2-wire serial interface
28	$0036	SPM_READY	Store program memory ready
29	$0038	USART1_RX	USART1 Rx Complete
30	$003A	USART1_UDRE	USART1 Data Register Empty
31	$003C	USART1_TX	USART1 Tx Complete

Notes:
1. When the BOOTRST fuse is programmed, the device will jump to the boot loader address at reset, see Section 27. "Memory Programming" on page 255.
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.

Table 12-2 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 12-2. **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 26-10 on page 251. 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 ATmega164P-B/324P-B/644P-B is:

```
Address       Labels Code              Comments
0x0000               jmp    RESET         ; Reset
0x0002               jmp    INT0          ; IRQ0
0x0004               jmp    INT1          ; IRQ1
0x0006               jmp    INT2          ; IRQ2
0x0008               jmp    PCINT0        ; PCINT0
0x000A               jmp    PCINT1        ; PCINT1
0x000C               jmp    PCINT2        ; PCINT2
0x000E               jmp    PCINT3        ; PCINT3
0x0010               jmp    WDT           ; Watchdog Timeout
0x0012               jmp    TIM2_COMPA    ; Timer2 CompareA
0x0014               jmp    TIM2_COMPB    ; Timer2 CompareB
0x0016               jmp    TIM2_OVF      ; Timer2 Overflow
0x0018               jmp    TIM1_CAPT     ; Timer1 Capture
0x001A               jmp    TIM1_COMPA    ; Timer1 CompareA
0x001C               jmp    TIM1_COMPB    ; Timer1 CompareB
0x001E               jmp    TIM1_OVF      ; Timer1 Overflow
0x0020               jmp    TIM0_COMPA    ; Timer0 CompareA
0x0022               jmp    TIM0_COMPB    ; Timer0 CompareB
0x0024               jmp    TIM0_OVF      ; Timer0 Overflow
0x0026               jmp    SPI_STC       ; SPI Transfer Complete
0x0028               jmp    USART0_RXC    ; USART0 RX Complete
0x002A               jmp    USART0_UDRE   ; USART0,UDR Empty
0x002C               jmp    USART0_TXC    ; USART0 TX Complete
0x002E               jmp    ANA_COMP      ; Analog Comparator
0x0030               jmp    ADC           ; ADC Conversion Complete
0x0032               jmp    EE_RDY        ; EEPROM Ready
0x0034               jmp    TWI           ; 2-wire Serial
0x0036               jmp    SPM_RDY       ; SPM Ready
0x0038               jmp    USART1_RXC    ; USART1 RX Complete
0x003A               jmp    USART1_UDRE   ; USART1,UDR Empty
0x003C               jmp    USART1_TXC    ; USART1 TX Complete
;
0x003E        RESET: ldi    r16,high(RAMEND); Main program start
0x003F               out    SPH,r16       ; Set Stack Pointer to top of RAM
0x0040               ldi    r16,low(RAMEND)
0x0041               out    SPL,r16
0x0042               sei                  ; Enable interrupts
0x0043               <instr> xxx
...           ...    ...    ...
```

When the BOOTRST fuse is unprogrammed, the boot section size set to 8Kbytes 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
0x00000      RESET: ldi    r16,high(RAMEND); Main program start
0x00001             out    SPH,r16      ; Set Stack Pointer to top of RAM
0x00002             ldi    r16,low(RAMEND)
0x00003             out    SPL,r16
0x00004             sei                 ; Enable interrupts
0x00005             <instr> xxx
;
.org 0x1F002
0x1F002             jmp    EXT_INT0     ; IRQ0 Handler
0x1F004             jmp    EXT_INT1     ; IRQ1 Handler
...                 ...    ...          ;
0x1FO36             jmp    SPM_RDY      ; SPM Ready Handler
```

When the BOOTRST fuse is programmed and the boot section size set to 8Kbytes, the most typical and general program setup for the reset and interrupt vector addresses is:

```
Address      Labels Code                Comments
.org 0x0002
0x00002             jmp    EXT_INT0     ; IRQ0 Handler
0x00004             jmp    EXT_INT1     ; IRQ1 Handler
...                 ...    ...          ;
0x00036             jmp    SPM_RDY      ; SPM Ready Handler
;
.org 0x1F000
0x1F000      RESET: ldi    r16,high(RAMEND); Main program start
0x1F001             out    SPH,r16      ; Set Stack Pointer to top of RAM
0x1F002             ldi    r16,low(RAMEND)
0x1F003             out    SPL,r16
0x1F004             sei                 ; Enable interrupts
0x1F005             <instr> xxx
```

When the BOOTRST fuse is programmed, the boot section size set to 8Kbytes 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 0x1F000
0x1F000             jmp    RESET        ; Reset handler
0x1F002             jmp    EXT_INT0     ; IRQ0 Handler
0x1F004             jmp    EXT_INT1     ; IRQ1 Handler
...                 ...    ...          ;
0x1F036             jmp    SPM_RDY      ; SPM Ready Handler
;
0x1F03E      RESET: ldi    r16,high(RAMEND); Main program start
0x1F03F             out    SPH,r16      ; Set Stack Pointer to top of RAM
0x1F040             ldi    r16,low(RAMEND)
0x1F041             out    SPL,r16
0x1F042             sei    ; Enable interrupts
0x1FO43             <instr> xxx
```

12.2.1 Moving Interrupts Between Application and Boot Space

The general interrupt control register controls the placement of the interrupt vector table.

12.3 Register Description

12.3.1 MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	**JTD**	**BODS**	**BODSE**	**PUD**	**–**	**–**	**IVSEL**	**IVCE**	**MCUCR**
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 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 27. “Memory Programming” on page 255 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 Section 27. “Memory Programming” on page 255 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 the following code example.

Assembly code example

```
Move_interrupts:
       ; Get MCUCR
       in      r16, MCUCR
       mov     r17, r16
       ; Enable change of Interrupt Vectors
       ori     r16, (1<<IVCE)
       out     MCUCR, r16
       ; Move interrupts to Boot Flash section
       ori     r17, (1<<IVSEL)
       out     MCUCR, r17
       ret
```

C code example

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

13. External Interrupts

13.1 Overview

The external interrupts are triggered by the INT2:0 pin or any of the PCINT31:0 pins. Observe that, if enabled, the interrupts will trigger even if the INT2:0 or PCINT31:0 pins are configured as outputs. This feature provides a way of generating a software interrupt.

The pin change interrupt PCI3 will trigger if any enabled PCINT31:24 pin toggle, Pin change interrupt PCI2 will trigger if any enabled PCINT23:16 pin toggles, pin change interrupt PCI1 if any enabled PCINT15:8 toggles and pin change interrupts PCI0 will trigger if any enabled PCINT7:0 pin toggles. PCMSK3, PCMSK2, PCMSK1 and PCMSK0 registers control which pins contribute to the pin change interrupts. Pin change interrupts on PCINT31:0 are detected asynchronously. This implies that these interrupts can be used for waking the part also from sleep modes other than idle mode.

The external 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 registers – EICRA (INT2:0). When the external interrupt is enabled and is configured as level triggered, the interrupt will trigger as long as the pin is held low. Low level interrupts and the edge interrupt on INT2:0 are detected asynchronously. This implies that these interrupts can be used for waking the part also from sleep modes other than idle mode. The I/O clock is halted in all sleep modes except idle mode.

Note that if a level triggered interrupt is used for wake-up from power-down, the required level must be held long enough for the MCU to complete the wake-up to trigger the level interrupt. If the level disappears before the end of the start-up time, the MCU will still wake up, but no interrupt will be generated. The start-up time is defined by the SUT and CKSEL fuses as described in Section 9. “System Clock and Clock Options” on page 23.

13.2 Register Description

13.2.1 EICRA – External Interrupt Control Register A

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

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	**ISC21**	**ISC20**	**ISC11**	**ISC10**	**ISC01**	**ISC00**	**EICRA**
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:6 – Reserved**

These bits are reserved in the Atmel ATmega164P-B/324P-B/644P-B, and will always read as zero.

- **Bits 5:0 – ISC21, ISC20 – ISC00, ISC00: External Interrupt 2 - 0 Sense Control Bits**

The external interrupts 2 - 0 are activated by the external pins INT2:0 if the SREG I-flag and the corresponding interrupt mask in the EIMSK is set. The level and edges on the external pins that activate the interrupts are defined in Table 13-1 on page 54. Edges on INT2:INT0 are registered asynchronously. Pulses on INT2:0 pins wider than the minimum pulse width given in Section 28.6 “External Interrupts Characteristics” on page 291 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. If enabled, a level triggered interrupt will generate an interrupt request as long as the pin is held low.

When changing the ISCn bit, an interrupt can occur. Therefore, it is recommended to first disable INTn by clearing its interrupt enable bit in the EIMSK register. Then, the ISCn bit can be changed. Finally, the INTn interrupt flag should be cleared by writing a logical one to its interrupt flag bit (INTFn) in the EIFR register before the interrupt is re-enabled.

Table 13-1. Interrupt Sense Control[1]

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Any edge of INTn generates asynchronously an interrupt request.
1	0	The falling edge of INTn generates asynchronously an interrupt request.
1	1	The rising edge of INTn generates asynchronously an interrupt request.

Note: 1. n = 2, 1 or 0.
When changing the ISCn1/ISCn0 bits, the interrupt must be disabled by clearing its interrupt enable bit in the EIMSK register. Otherwise an interrupt can occur when the bits are changed.

13.2.2 EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	**INT2**	**INT1**	**IINT0**	**EIMSK**
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 2:0 – INT2:0: External Interrupt Request 2 - 0 Enable**

When an INT2:0 bit is written to one and the I-bit in the status register (SREG) is set (one), the corresponding external pin interrupt is enabled. The interrupt sense control bits in the external Interrupt control register, EICRA, defines whether the external interrupt is activated on rising or falling edge or level sensed. Activity on any of these pins will trigger an interrupt request even if the pin is enabled as an output. This provides a way of generating a software interrupt.

13.2.3 EIFR –External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	–	–	–	–	–	**INTF2**	**INTF1**	**IINTF0**	**EIFR**
Read/Write	R/W	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 2:0 – INTF2:0: External Interrupt Flags 2 - 0**

When an edge or logic change on the INT2:0 pin triggers an interrupt request, INTF2:0 becomes set (one). If the I-bit in SREG and the corresponding interrupt enable bit, INT2:0 in EIMSK, are set (one), the MCU will jump to the 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. These flags are always cleared when INT2:0 are configured as level interrupt. Note that when entering sleep mode with the INT2:0 interrupts disabled, the input buffers on these pins will be disabled. This may cause a logic change in internal signals which will set the INTF2:0 flags. See Section 14.2.5 “Digital Input Enable and Sleep Modes” on page 61 for more information.

13.2.4 PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	**PCIE3**	**PCIE2**	**PCIE1**	**PCIE0**	**PCICR**
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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

When the PCIE3 bit is set (one) and the I-bit in the status register (SREG) is set (one), pin change interrupt 3 is enabled. Any change on any enabled PCINT31:24 pin will cause an interrupt. The corresponding interrupt of pin change interrupt request is executed from the PCI3 interrupt vector. PCINT31:24 pins are enabled individually by the PCMSK3 register.

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

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

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

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

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

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

13.2.5 PCIFR – Pin Change Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1B (0x3B)			–	–	**PCIF3**	**PCIF2**	**PCIF1**	**PCIF0**	**PCIFR**
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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

When a logic change on any PCINT31:24 pin triggers an interrupt request, PCIF3 becomes set (one). If the I-bit in SREG and the PCIE3 bit in PCICR are set (one), the MCU will jump to the corresponding interrupt vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

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

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

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

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

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

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

13.2.6 PCMSK3 – Pin Change Mask Register 3

Bit	7	6	5	4	3	2	1	0	
(0x73)	**PCINT31**	**PCINT30**	**PCINT29**	**PCINT28**	**PCINT27**	**PCINT26**	**PCINT25**	**PCINT24**	**PCMSK3**
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 – PCINT31:24: Pin Change Enable Mask 31:24**

Each PCINT31:24-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT31:24 is set and the PCIE3 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT31:24 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

13.2.7 PCMSK2 – Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
(0x6D)	**PCINT23**	**PCINT22**	**PCINT21**	**PCINT20**	**PCINT19**	**PCINT18**	**PCINT17**	**PCINT16**	**PCMSK2**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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

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

13.2.8 PCMSK1 – Pin Change Mask Register 1

Bit	7	6	5	4	3	2	1	0	
(0x6C)	**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.

13.2.9 PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	**PCINT7**	**PCINT6**	**PCINT5**	**PCINT4**	**PCINT3**	**PCINT2**	**PCINT1**	**PCINT0**	**PCMSK0**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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

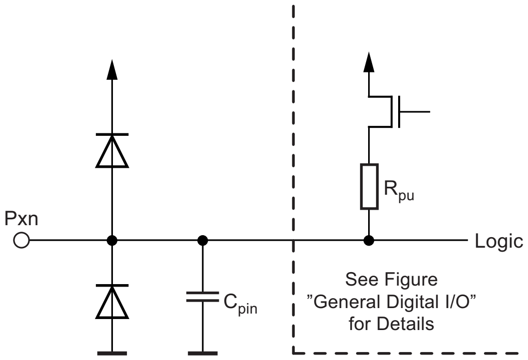
Each PCINT7:0 bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT7:0 is set and the PCIE0 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

14. I/O-Ports

14.1 Overview

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 14-1. Refer to Section 28. “Electrical Characteristics” on page 287 for a complete list of parameters.

Figure 14-1. I/O Pin Equivalent Schematic



All registers and bit references in this section are written in general form. A lower case “x” represents the numbering letter for the port, and a lower case “n” represents the bit number. However, when using the register or bit defines in a program, the precise form must be used. For example, PORTB3 for bit no. 3 in Port B, here documented generally as PORTxn. The physical I/O registers and bit locations are listed in Section “Register Description” on page 72.

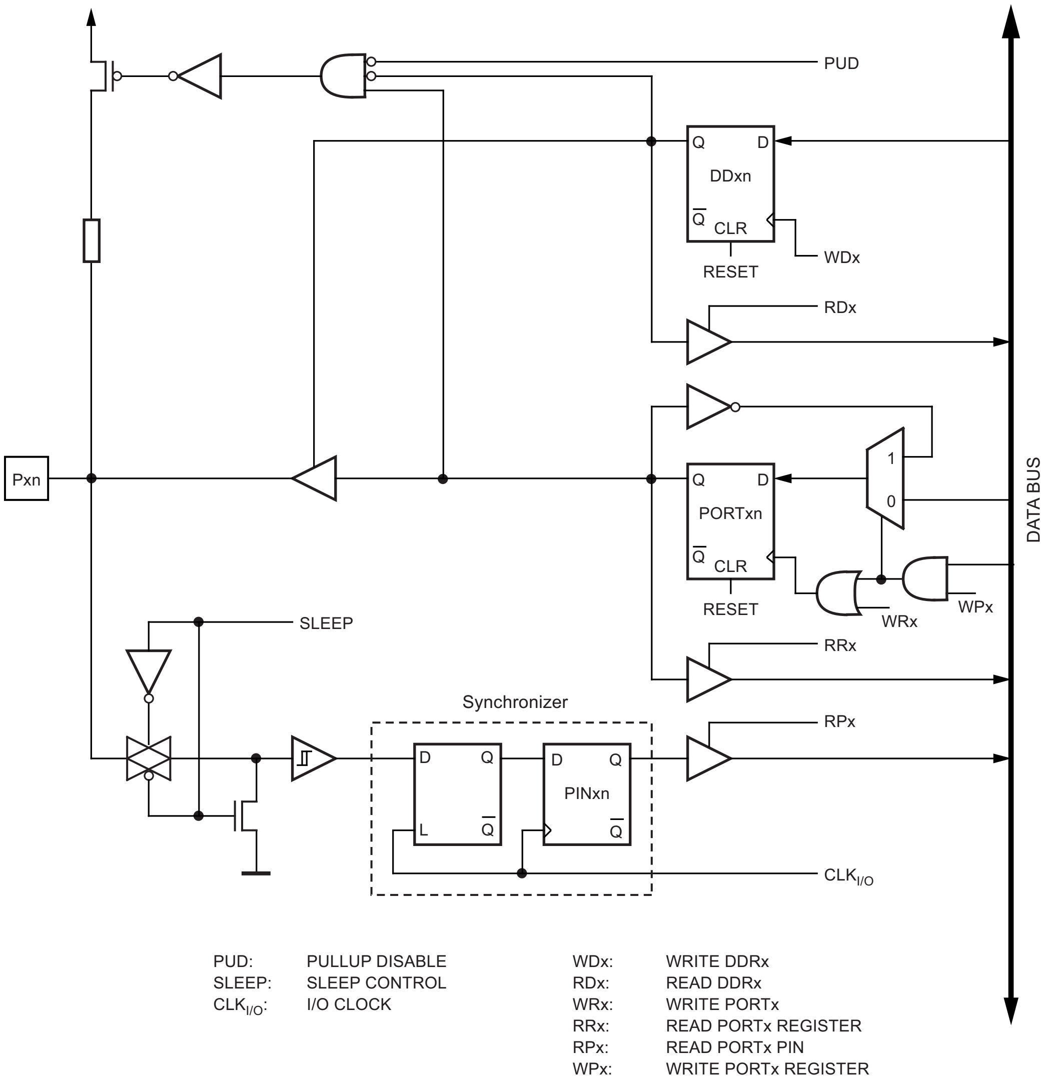
Three I/O memory address locations are allocated for each port, one each for the data register – PORTx, data direction register – DDRx, and the port input pins – PINx. The port input pins I/O location is read only, while the data register and the data direction register are read/write. However, writing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

Using the I/O port as general digital I/O is described in Section 14.2 “Ports as General Digital I/O” on page 58. Most port pins are multiplexed with alternate functions for the peripheral features on the device. How each alternate function interferes with the port pin is described in Section 14.3 “Alternate Port Functions” on page 62. 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.

14.2 Ports as General Digital I/O

The ports are bi-directional I/O ports with optional internal pull-ups. Figure 14-2 shows a functional description of one I/O-port pin, here generically called Pxn.

Figure 14-2. General Digital I/O(1)



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

14.2.1 Configuring the Pin

Each port pin consists of three register bits: DDxn, PORTxn, and PINxn. As shown in Section "Register Description" on page 72, 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).

14.2.2 Toggling the Pin

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

14.2.3 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 14-1 summarizes the control signals for the pin value.

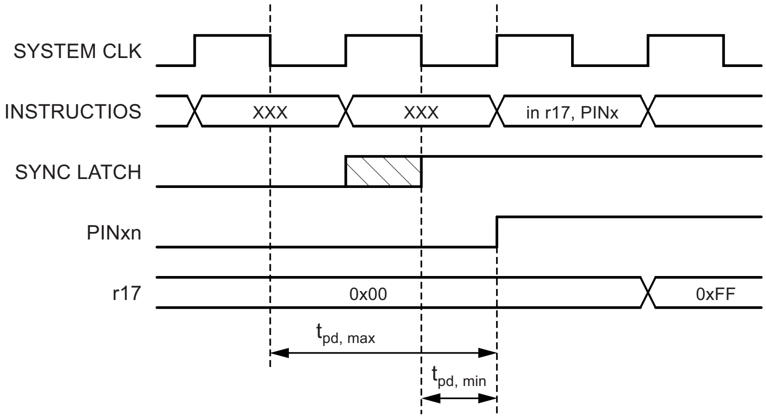
Table 14-1. Port Pin Configurations

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)

14.2.4 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 14-2 on page 58, 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 14-3 on page 60 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.

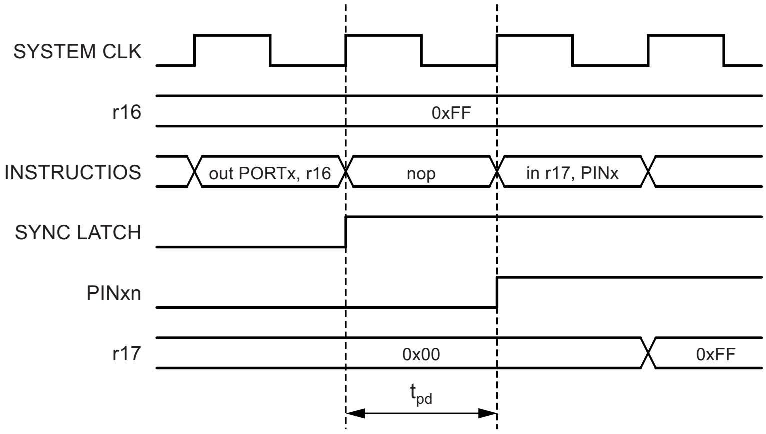
Figure 14-3. Synchronization when Reading an Externally Applied Pin value



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 14-4. 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 14-4. 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.

14.2.5 Digital Input Enable and Sleep Modes

As shown in Figure 14-2 on page 58, 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 Section 14.3 “Alternate Port Functions” on page 62.

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.

14.2.6 Unconnected Pins

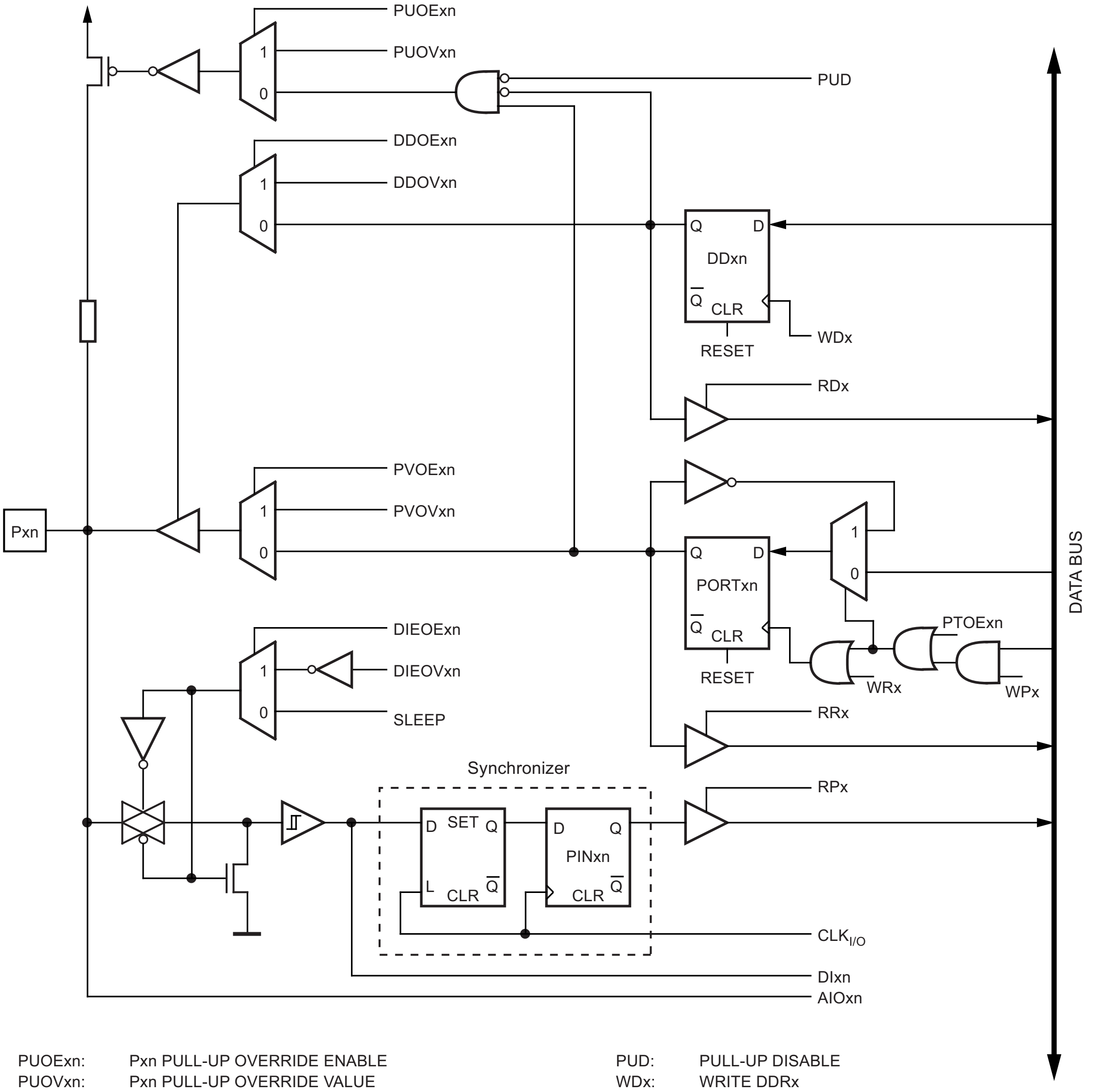
If some pins are unused, it is recommended to ensure that these pins have a defined level. Even though most of the digital inputs are disabled in the deep sleep modes as described above, floating inputs should be avoided to reduce current consumption in all other modes where the digital inputs are enabled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

14.3 Alternate Port Functions

Most port pins have alternate functions in addition to being general digital I/Os. Figure 14-5 shows how the port pin control signals from the simplified Figure 14-2 on page 58 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 14-5. Alternate Port Functions(1)



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

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

Table 14-2 summarizes the function of the overriding signals. The pin and port indexes from Figure 14-5 on page 62 are not shown in the succeeding tables. The overriding signals are generated internally in the modules having the alternate function.

Table 14-2. Generic Description of Overriding Signals for Alternate Functions

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

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

14.3.1 Alternate Functions of Port A

The port A pins with alternate functions are shown in Table 14-3.

Table 14-3. Port A Pins Alternate Functions

Port Pin	Alternate Function
PA7	ADC7 (ADC input channel 7) PCINT7 (pin change interrupt 7)
PA6	ADC6 (ADC input channel 6) PCINT6 (pin change interrupt 6)
PA5	ADC5 (ADC input channel 5) PCINT5 (pin change interrupt 5)
PA4	ADC4 (ADC input channel 4) PCINT4 (pin change interrupt 4)
PA3	ADC3 (ADC input channel 3) PCINT3 (pin change interrupt 3)
PA2	ADC2 (ADC input channel 2) PCINT2 (pin change interrupt 2)
PA1	ADC1 (ADC input channel 1) PCINT1 (pin change interrupt 1)
PA0	ADC0 (ADC input channel 0) PCINT0 (pin change interrupt 0)

- **ADC7:0/PCINT7:0 – Port A, Bit 7:0**

ADC7:0, analog to digital converter, channels 7:0.

PCINT7:0, pin change interrupt source 7:0: The PA7:0 pins can serve as external interrupt sources.

Table 14-4 and Table 14-5 relate the alternate functions of port A to the overriding signals shown in Figure 14-5 on page 62.

Table 14-4. Overriding Signals for Alternate Functions in PA7:PA4

Signal Name	PA7/ADC7/PCINT7	PA6/ADC6/PCINT6	PA5/ADC5/PCINT5	PA4/ADC4/PCINT4
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
DIEOE	PCINT7 × PCIE0 + ADC7D	PCINT6 × PCIE0 + ADC6D	PCINT5 × PCIE0 + ADC5D	PCINT4 × PCIE0 + ADC4D
DIEOV	PCINT7 × PCIE0	PCINT6 × PCIE0	PCINT5 × PCIE0	PCINT4 × PCIE0
DI	PCINT7 INPUT	PCINT6 INPUT	PCINT5 INPUT	PCINT4 INPUT
AIO	ADC7 INPUT	ADC6 INPUT	ADC5 INPUT	ADC4 INPUT

Table 14-5. Overriding Signals for Alternate Functions in PA3:PA0

Signal Name	PA3/ADC3/PCINT3	PA2/ADC2/PCINT2	PA1/ADC1/PCINT1	PA0/ADC0/PCINT0
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
DIEOE	PCINT3 × PCIE0 + ADC3D	PCINT2 × PCIE0 + ADC2D	PCINT1 × PCIE0 + ADC1D	PCINT0 × PCIE0 + ADC0D
DIEOV	PCINT3 × PCIE0	PCINT2 × PCIE0	PCINT1 × PCIE0	PCINT0 × PCIE0
DI	PCINT3 INPUT	PCINT2 INPUT	PCINT1 INPUT	PCINT0 INPUT
AIO	ADC3 INPUT	ADC2 INPUT	ADC1 INPUT	ADC0 INPUT

14.3.2 Alternate Functions of Port B

The port B pins with alternate functions are shown in Table 14-6.

Table 14-6. Port B Pins Alternate Functions

Port Pin	Alternate Functions
PB7	SCK (SPI bus master clock input) OC3B (Timer/Counter 3 output compare match B output) PCINT15 (pin change interrupt 15)
PB6	MISO (SPI bus master input/slave output) OC3A (Timer/Counter 3 output compare match A output) PCINT14 (pin change interrupt 14)
PB5	MOSI (SPI bus master output/slave input) ICP3 (Timer/Counter3 input capture trigger) PCINT13 (pin change interrupt 13)
PB4	$\overline{\text{SS}}$ (SPI slave select input) OC0B (Timer/Counter 0 output compare match B output) PCINT12 (pin change interrupt 12)
PB3	AIN1 (analog comparator negative input) OC0A (Timer/Counter 0 output compare match A output) PCINT11 (pin change interrupt 11)
PB2	AIN0 (analog comparator positive input) INT2 (external interrupt 2 input) PCINT10 (pin change interrupt 10)
PB1	T1 (Timer/Counter 1 external counter input) CLKO (divided system clock output) PCINT9 (pin change interrupt 9)
PB0	T0 (Timer/Counter 0 external counter input) XCK0 (USART0 external clock input/output) PCINT8 (pin change interrupt 8)

The alternate pin configuration is as follows:

- **SCK/OC3B/PCINT15 – Port B, Bit 7**

SCK: Master clock output, slave clock input pin for SPI channel. When the SPI is enabled as a slave, this pin is configured as an input regardless of the setting of DDB7. When the SPI0 is enabled as a master, the data direction of this pin is controlled by DDB7. When the pin is forced to be an input, the pull-up can still be controlled by the PORTB7 bit.

OC3B, output compare match B output: The PB7 pin can serve as an external output for the Timer/Counter3 output compare. The pin has to be configured as an output (DDB7 set "one") to serve this function. The OC3B 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.

- **MISO/OC3A/PCINT14 – Port B, Bit 6**

MISO: Master data input, slave data output pin for SPI channel. When the SPI is enabled as a master, this pin is configured as an input regardless of the setting of DDB6. When the SPI is enabled as a slave, the data direction of this pin is controlled by DDB6. When the pin is forced to be an input, the pull-up can still be controlled by the PORTB6 bit.

OC3A, output compare match A output: The PB6 pin can serve as an external output for the Timer/Counter0 output compare. The pin has to be configured as an output (DDB6 set "one") to serve this function. The OC3A 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.

- **MOSI/ICP3/PCINT13 – Port B, Bit 5**

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

ICP3, input capture pin 3: The PB5 pin can act as an input capture pin for Timer/Counter3.

PCINT13, pin change interrupt source 13: The PB5 pin can serve as an external interrupt source.

- **$\overline{\text{SS}}$/OC0B/PCINT12 – Port B, Bit 4**

$\overline{\text{SS}}$: Slave port select input. When the SPI is enabled as a slave, this pin is configured as an input regardless of the setting of DDB4. 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 DDB4. When the pin is forced to be an input, the pull-up can still be controlled by the PORTB4 bit.

OC0B, output compare match B output: The PB4 pin can serve as an external output for the Timer/Counter0 output compare. The pin has to be configured as an output (DDB4 set “one”) to serve this function. The OC0B 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.

- **AIN1/OC0A/PCINT11, Bit 3**

AIN1, analog comparator negative input. This pin is directly connected to the negative input of the analog comparator.

OC0A, output compare match A output: The PB3 pin can serve as an external output for the Timer/Counter0 output compare. The pin has to be configured as an output (DDB3 set “one”) to serve this function. The OC0A pin is also the output pin for the PWM mode timer function.

PCINT11, pin change interrupt source 11: The PB3 pin can serve as an external interrupt source.

- **AIN0/INT2/PCINT10, Bit 2**

AIN0, analog comparator positive input. This pin is directly connected to the positive input of the analog comparator.

INT2, external interrupt source 2. The PB2 pin can serve as an external interrupt source to the MCU.

PCINT10, pin change interrupt source 10: The PB2 pin can serve as an external interrupt source.

- **T1/CLKO/PCINT9, Bit 1**

T1, Timer/Counter1 counter source.

CLKO, divided system clock: The divided system clock can be output on the PB1 pin. The divided system clock will be output if the CKOUT fuse is programmed, regardless of the PORTB1 and DDB1 settings. It will also be output during reset.

PCINT9, pin change interrupt source 9: The PB1 pin can serve as an external interrupt source.

- **T0/XCK0/PCINT8, Bit 0**

T0, Timer/Counter0 counter source.

XCK0, USART0 external clock. The data direction register (DDB0) controls whether the clock is output (DDD0 set “one”) or input (DDD0 cleared). The XCK0 pin is active only when the USART0 operates in synchronous mode.

PCINT8, pin change interrupt source 8: The PB0 pin can serve as an external interrupt source.

Table 14-7 and Table 14-8 relate the alternate functions of port B to the overriding signals shown in Figure 14-5 on page 62. SPI MSTR INPUT and SPI SLAVE OUTPUT constitute the MISO signal, while MOSI is divided into SPI MSTR OUTPUT and SPI SLAVE INPUT.

Table 14-7. Overriding Signals for Alternate Functions in PB7:PB4

Signal Name	PB7/SCK/PCINT15	PB6/MISO/PCINT14	PB5/MOSI/PCINT13	PB4/$\overline{\text{SS}}$/OC0B/PCINT12
PUOE	SPE × $\overline{\text{MSTR}}$	SPE × MSTR	SPE × $\overline{\text{MSTR}}$	SPE × $\overline{\text{MSTR}}$
PUOV	PORTB7 × $\overline{\text{PUD}}$	PORTB14 × $\overline{\text{PUD}}$	PORTB13 × $\overline{\text{PUD}}$	PORTB12 × $\overline{\text{PUD}}$
DDOE	SPE × $\overline{\text{MSTR}}$	SPE × MSTR	SPE × $\overline{\text{MSTR}}$	SPE × $\overline{\text{MSTR}}$
DDOV	0	0	0	0
PVOE	SPE × MSTR	SPE × $\overline{\text{MSTR}}$	SPE × MSTR	OC0A ENABLE
PVOV	SCK OUTPUT	SPI SLAVE OUTPUT	SPI MSTR OUTPUT	OC0A
DIEOE	PCINT15 × PCIE1	PCINT14 × PCIE1	PCINT13 × PCIE1	PCINT12 × PCIE1
DIEOV	1	1	1	1
DI	SCK INPUT PCINT17 INPUT	SPI MSTR INPUT PCINT14 INPUT	SPI SLAVE INPUT PCINT13 INPUT	SPI $\overline{\text{SS}}$ PCINT12 INPUT
AIO	–	–	–	–

Table 14-8. Overriding Signals for Alternate Functions in PB3:PB0

Signal Name	PB3/AIN1/OC0B/PCINT11	PB2/AIN0/INT2/PCINT10	PB1/T1/CLKO/PCINT9	PB0/T0/XCK/PCINT8
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	CKOUT	0
DDOV	0	0	CKOUT	0
PVOE	OC0B ENABLE	0	CKOUT	0
PVOV	OC0B	0	CLK I/O	0
DIEOE	PCINT11 × PCIE1	INT2 ENABLE PCINT10 × PCIE1	PCINT9 × PCIE1	PCINT8 × PCIE1
DIEOV	1	1	1	1
DI	PCINT11 INPUT	INT2 INPUT PCINT10 INPUT	T1 INPUT PCINT9 INPUT	T0 INPUT PCINT8 INPUT
AIO	AIN1 INPUT	AIN0 INPUT	–	–

14.3.3 Alternate Functions of Port C

The port C pins with alternate functions are shown in Table 14-9.

Table 14-9. Port C Pins Alternate Functions

Port Pin	Alternate Function
PC7	TOSC2 (timer oscillator pin 2) PCINT23 (pin change interrupt 23)
PC6	TOSC1 (timer oscillator pin 1) PCINT22 (pin change interrupt 22)
PC5	TDI (JTAG test data input) PCINT21 (pin change interrupt 21)
PC4	TDO (JTAG test data output) PCINT20 (pin change interrupt 20)
PC3	TMS (JTAG test mode select) PCINT19 (pin change interrupt 19)
PC2	TCK (JTAG test clock) PCINT18 (pin change interrupt 18)
PC1	SDA (2-wire serial bus data input/output line) PCINT17 (pin change interrupt 17)
PC0	SCL (2-wire serial bus clock line) PCINT16 (pin change interrupt 16)

- **TOSC2/PCINT23 – Port C, Bit7**

TOSC2, timer oscillator pin 2. The PC7 pin can serve as an external interrupt source to the MCU.

PCINT23, pin change interrupt source 23: The PC7 pin can serve as an external interrupt source.

- **TOSC1/PCINT22 – Port C, Bit 6**

TOSC1, timer oscillator pin 1. The PC6 pin can serve as an external interrupt source to the MCU.

PCINT22, pin change interrupt source 22: The PC6 pin can serve as an external interrupt source.

- **TDI/PCINT21 – Port C, Bit 5**

TDI, JTAG test data input.

PCINT21, pin change interrupt source 21: The PC5 pin can serve as an external interrupt source.

- **TDO/PCINT20 – Port C, Bit 4**

TDO, JTAG test data output.

PCINT20, pin change interrupt source 20: The PC4 pin can serve as an external interrupt source.

- **TMS/PCINT19 – Port C, Bit 3**

TMS, JTAG test mode select.

PCINT19, pin change interrupt source 19: The PC3 pin can serve as an external interrupt source.

- **TCK/PCINT18 – Port C, Bit 2**

TCK, JTAG test clock.

PCINT18, pin change interrupt source 18: The PC2 pin can serve as an external interrupt source.

- **SDA/PCINT17 – Port C, Bit 1**

SDA, 2-wire serial bus data input/output line.

PCINT17, pin change interrupt source 17: The PC1 pin can serve as an external interrupt source.

- **SCL/PCINT16 – Port C, Bit 0**

SCL, 2-wire serial bus clock line.

PCINT16, pin change interrupt source 16: The PC0 pin can serve as an external interrupt source.

Table 14-10 and Table 14-11 relate the alternate functions of port C to the overriding signals shown in Figure 14-5 on page 62.

Table 14-10. Overriding Signals for Alternate Functions in PC7:PC4

Signal Name	PC7/TOSC2/PCINT23	PC6/TOSC1/PCINT22	PC5/TDI/PCINT21	PC4/TDO/PCINT20
PUOE	AS2 × $\overline{\text{EXCLK}}$	AS2	JTAGEN	JTAGEN
PUOV	0	0	1	1
DDOE	AS2 × $\overline{\text{EXCLK}}$	AS2	JTAGEN	JTAGEN
DDOV	0	0	0	SHIFT_IR + SHIFT_DR
PVOE	0	0	0	JTAGEN
PVOV	0	0	0	TDO
DIEOE	AS2 × $\overline{\text{EXCLK}}$ + PCINT23 × PCIE2	AS2 + PCINT22 × PCIE2	JTAGEN + PCINT21 × PCIE2	JTAGEN + PCINT20 × PCIE2
DIEOV	$\overline{\text{AS2}}$	EXCLK + $\overline{\text{AS2}}$	$\overline{\text{JTAGEN}}$	$\overline{\text{JTAGEN}}$
DI	PCINT23 INPUT	PCINT22 INPUT	PCINT21 INPUT	PCINT20 INPUT
AIO	T/C2 OSC OUTPUT	T/C2 OSC INPUT	TDI INPUT	–

Table 14-11. Overriding Signals for Alternate Functions in PC3:PC0

Signal Name	PC3/TMS/PCINT19	PC2/TCK/PCINT18	PC1/SDA/PCINT17	PC0/SCL/PCINT16
PUOE	JTAGEN	JTAGEN	TWEN	TWEN
PUOV	1	1	PORTC1 × $\overline{\text{PUD}}$	PORTC0 × $\overline{\text{PUD}}$
DDOE	JTAGEN	JTAGEN	TWEN	TWEN
DDOV	0	0	0	0
PVOE	0	0	TWEN	TWEN
PVOV	0	0	SDA OUT	SCL OUT
DIEOE	JTAGEN + PCINT19 × PCIE2	JTAGEN + PCINT18 × PCIE2	PCINT17 × PCIE2	PCINT16 × PCIE2
DIEOV	$\overline{\text{JTAGEN}}$	$\overline{\text{JTAGEN}}$	1	1
DI	PCINT19 INPUT	PCINT18 INPUT	PCINT17 INPUT	PCINT16 INPUT
AIO	TMS INPUT	TCK INPUT	SDA INPUT	SCL INPUT

14.3.4 Alternate Functions of Port D

The port D pins with alternate functions are shown in Table 14-12.

Table 14-12. Port D Pins Alternate Functions

Port Pin	Alternate Function
PD7	OC2A (Timer/Counter2 output compare match A output) PCINT31 (pin change interrupt 31)
PD6	ICP1 (Timer/Counter1 input capture trigger) OC2B (Timer/Counter2 output compare match B output) PCINT30 (pin change interrupt 30)
PD5	OC1A (Timer/Counter1 output compare match A output) PCINT29 (pin change interrupt 29)
PD4	OC1B (Timer/Counter1 output compare match B output) XCK1 (USART1 external clock input/output) PCINT28 (pin change interrupt 28)
PD3	INT1 (external interrupt1 input) TXD1 (USART1 transmit pin) PCINT27 (pin change interrupt 27)
PD2	INT0 (external interrupt0 input) RXD1 (USART1 receive pin) PCINT26 (pin change interrupt 26)
PD1	TXD0 (USART0 transmit pin) PCINT25 (pin change interrupt 25)
PD0	RXD0 (USART0 receive pin) PCINT24 (pin change interrupt 24) T3 (Timer/Counter 3 external counter input)

The alternate pin configuration is as follows:

- **OC2A/PCINT31 – Port D, Bit 7**

OC2A, output compare match A output: The PD7 pin can serve as an external output for the Timer/Counter2 output compare A. The pin has to be configured as an output (DDD7 set (one)) to serve this function. The OC2A pin is also the output pin for the PWM mode timer function.

PCINT31, pin change interrupt source 31:The PD7 pin can serve as an external interrupt source.

- **ICP1/OC2B/PCINT30 – Port D, Bit 6**

ICP1, input capture pin 1: The PD6 pin can act as an input capture pin for Timer/Counter1.

OC2B, output compare match B output: The PD6 pin can serve as an external output for the Timer/Counter2 output compare B. The pin has to be configured as an output (DDD6 set (one)) to serve this function. The OC2B pin is also the output pin for the PWM mode timer function.

PCINT30, pin change interrupt source 30: The PD6 pin can serve as an external interrupt source.

- **OC1A/PCINT29 – Port D, Bit 5**

OC1A, output compare match A output: The PD5 pin can serve as an external output for the Timer/Counter1 output compare A. The pin has to be configured as an output (DDD5 set (one)) to serve this function. The OC1A pin is also the output pin for the PWM mode timer function.

PCINT29, pin change interrupt source 29: The PD5 pin can serve as an external interrupt source.

- **OC1B/XCK1/PCINT28 – Port D, Bit 4**

OC1B, output compare match B output: The PB4 pin can serve as an external output for the Timer/Counter1 output compare B. The pin has to be configured as an output (DDD4 set (one)) to serve this function. The OC1B pin is also the output pin for the PWM mode timer function.

XCK1, USART1 external clock. The data direction register (DDB4) controls whether the clock is output (DDD4 set “one”) or input (DDD4 cleared). The XCK4 pin is active only when the USART1 operates in synchronous mode.

PCINT28, pin change interrupt source 28: The PD4 pin can serve as an external interrupt source.

- **INT1/TXD1/PCINT27 – Port D, Bit 3**

INT1, external interrupt source 1. The PD3 pin can serve as an external interrupt source to the MCU.

TXD1, transmit data (Data output pin for the USART1). When the USART1 transmitter is enabled, this pin is configured as an output regardless of the value of DDD3.

PCINT27, pin change interrupt source 27: The PD3 pin can serve as an external interrupt source.

- **INT0/RXD1/PCINT26 – Port D, Bit 2**

INT0, external interrupt source 0. The PD2 pin can serve as an external interrupt source to the MCU.

RXD1, RXD0, receive Data (data input pin for the USART1). When the USART1 receiver is enabled this pin is configured as an input regardless of the value of DDD2. When the USART forces this pin to be an input, the pull-up can still be controlled by the PORTD2 bit.

PCINT26, pin change interrupt source 26: The PD2 pin can serve as an external interrupt source.

- **TXD0/PCINT25 – Port D, Bit 1**

TXD0, transmit data (Data output pin for the USART0). When the USART0 transmitter is enabled, this pin is configured as an output regardless of the value of DDD1.

PCINT25, pin change interrupt source 25: The PD1 pin can serve as an external interrupt source.

- **RXD0/T3/PCINT24 – Port D, Bit 0**

RXD0, receive data (Data input pin for the USART0). When the USART0 receiver is enabled this pin is configured as an input regardless of the value of DDD0. When the USART forces this pin to be an input, the pull-up can still be controlled by the PORTD0 bit.

T3, Timer/Counter3 counter source.

PCINT24, pin change interrupt source 24: The PD0 pin can serve as an external interrupt source.

Table 14-13 on page 72 and Table 14-14 on page 72 relate the alternate functions of Port D to the overriding signals shown in Figure 14-5 on page 62.

Table 14-13. Overriding Signals for Alternate Functions PD7:PD4

Signal Name	PD7/OC2A/PCINT31	PD6/ICP1/OC2B/PCINT30	PD5/OC1A/PCINT29	PD4/OC1B/XCK1/PCINT28
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	OC2A ENABLE	OC2B ENABLE	OC1A ENABLE	OC1B ENABLE
PVOV	OCA2A	OC2B	OC1A	OC1B
DIEOE	PCINT31 × PCIE3	PCINT30 × PCIE3	PCINT29 × PCIE3	PCINT28 × PCIE3
DIEOV	1	1	1	1
DI	PCINT31 INPUT	ICP1 INPUT PCINT30 INPUT	PCINT29 INPUT	PCINT28 INPUT
AIO	–	–	–	–

Table 14-14. Overriding Signals for Alternate Functions in PD3:PD0(1)

Signal Name	PD3/INT1/TXD1/PCINT27	PD2/INT0/RXD1/PCINT26	PD1/TXD0/PCINT25	PD0/RXD0/PCINT27
PUOE	TXEN1	RXEN1	TXEN0	RXEN1
PUOV	0	PORTD2 × $\overline{\text{PUD}}$	0	PORTD0 × $\overline{\text{PUD}}$
DDOE	TXEN1	RXEN1	TXEN0	RXEN1
DDOV	1	0	1	0
PVOE	TXEN1	0	TXEN0	0
PVOV	TXD1	0	TXD0	0
DIEOE	INT1 ENABLE PCINT27 × PCIE3	INT2 ENABLE PCINT26 × PCIE3	PCINT25 × PCIE3	PCINT24 × PCIE3
DIEOV	1	1	1	1
DI	INT1 INPUT PCINT27 INPUT	INT0 INPUTRXD1 PCINT26 INPUT	PCINT25 INPUT	RXD0PCINT24 INPUT
AIO	–	–	–	–

Note: 1. When enabled, the 2-wire serial interface enables slew-rate controls on the output pins PD0 and PD1. This is not shown in this table. In addition, spike filters are connected between the AIO outputs shown in the port figure and the digital logic of the TWI module.

14.3.5 MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	JTD	BODS	BODSE	PUD	–	–	IVSEL	IVCE	MCUCR
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 4 – PUD: Pull-up Disable**

When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn registers are configured to enable the pull-ups ({DDxn, PORTxn} = 0b01). See Section 14.2.1 “Configuring the Pin” on page 59 for more details about this feature.

14.3.6 PORTA – Port A Data Register

Bit	7	6	5	4	3	2	1	0	
0x02 (0x22)	**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	

14.3.7 DDRA – Port A Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x01 (0x21)	**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	

14.3.8 PINA – Port A Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x00 (0x20)	**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	

14.3.9 PORTB – Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	**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	

14.3.10 DDRB – Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	**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	

14.3.11 PINB – Port B Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	**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	

14.3.12 PORTC – Port C Data Register

Bit	7	6	5	4	3	2	1	0	
0x08 (0x28)	**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	

14.3.13 DDRC – Port C Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x07 (0x27)	**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	

14.3.14 PINC – Port C Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x06 (0x26)	**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	

14.3.15 PORTD – Port D Data Register

Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	**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	

14.3.16 DDRD – Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	**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	

14.3.17 PIND – Port D Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	**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	

15. 8-bit Timer/Counter0 with PWM

15.1 Features

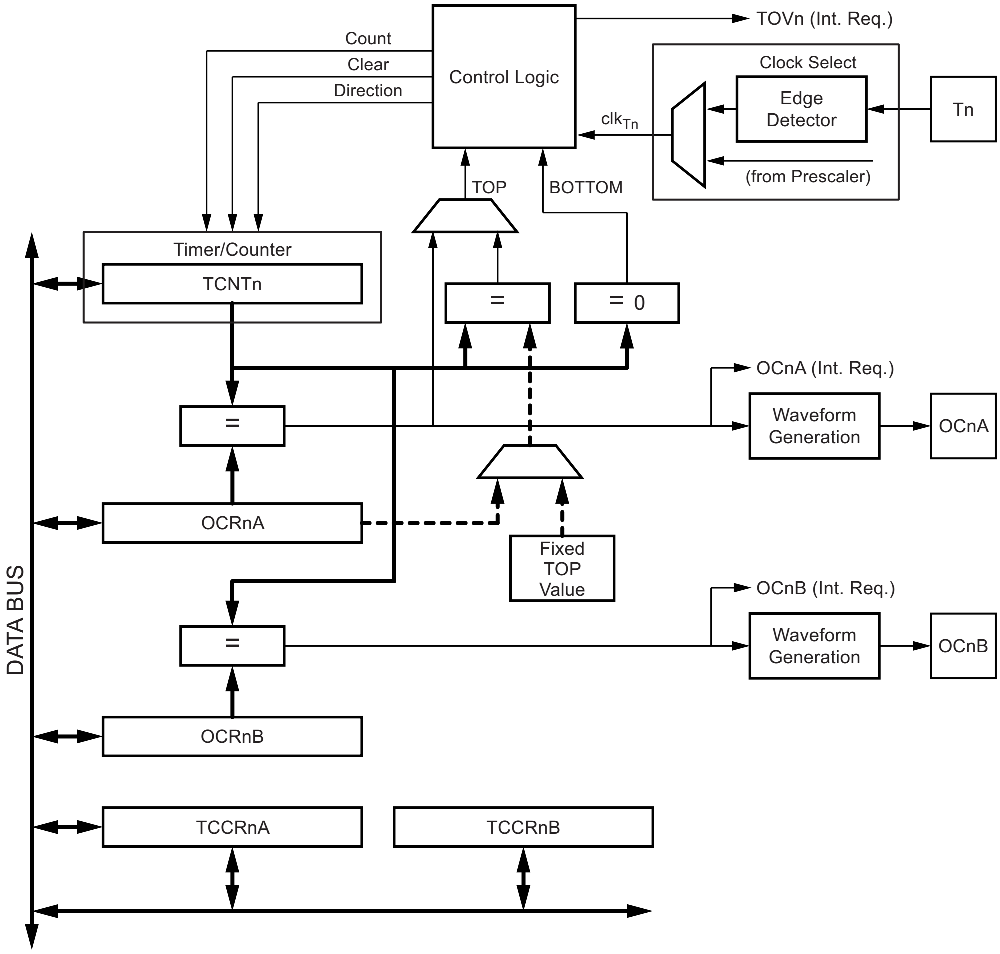
- Two independent output compare units
- Double buffered output compare registers
- Clear timer on compare match (auto reload)
- Glitch free, phase correct pulse width modulator (PWM)
- Variable PWM period
- Frequency generator
- Three independent interrupt sources (TOV0, OCF0A, and OCF0B)

15.2 Overview

Timer/Counter0 is a general purpose 8-bit Timer/Counter module, with two independent output compare units, and with PWM support. It allows accurate program execution timing (event management) and wave generation.

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 15-1. For the actual placement of I/O pins, see Section 1. “Pin Configurations” on page 3. CPU accessible I/O registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O register and bit locations are listed in the Section 15.9 “Register Description” on page 85.

Figure 15-1. 8-bit Timer/Counter Block Diagram



15.2.1 Registers

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

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the T0 pin. The clock select logic block controls which clock source and edge the Timer/Counter uses to increment (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 registers (OCR0A and OCR0B) are 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 pins (OC0A and OC0B). Section 15.5 "Output Compare Unit" on page 77 for details. The compare match event will also set the compare flag (OCF0A or OCF0B) which can be used to generate an output compare interrupt request.

15.2.2 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, in this case compare unit A or compare unit B. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT0 for accessing Timer/Counter0 counter value and so on.

The definitions in Table 15-1 are also used extensively throughout the document.

Table 15-1. Definitions

Parameter	Definition
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.

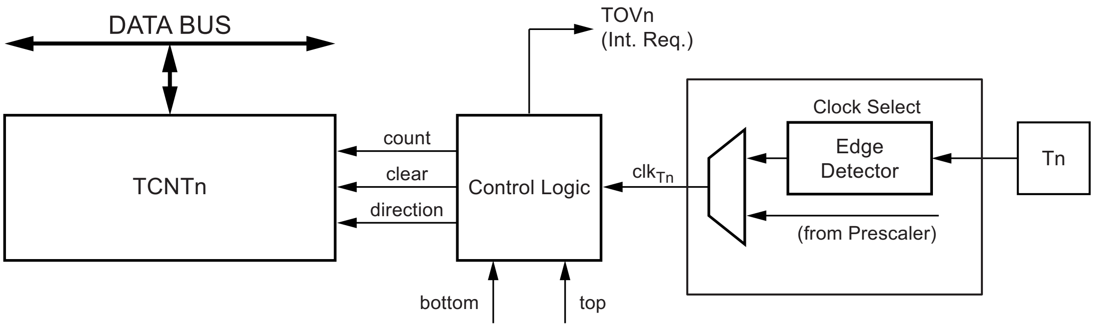
15.3 Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the clock select logic which is controlled by the clock select (CS02:0) bits located in the Timer/Counter control register (TCCR0B). For details on clock sources and prescaler, see Section 17.10 "Timer/Counter Prescaler" on page 131.

15.4 Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Figure 15-2 shows a block diagram of the counter and its surroundings.

Figure 15-2. 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) and the WGM02 bit located in the Timer/Counter control register B (TCCR0B). There are close connections between how the counter behaves (counts) and how waveforms are generated on the output compare outputs OC0A and OC0B. For more details about advanced counting sequences and waveform generation, see Section 15.7 “Modes of Operation” on page 79.

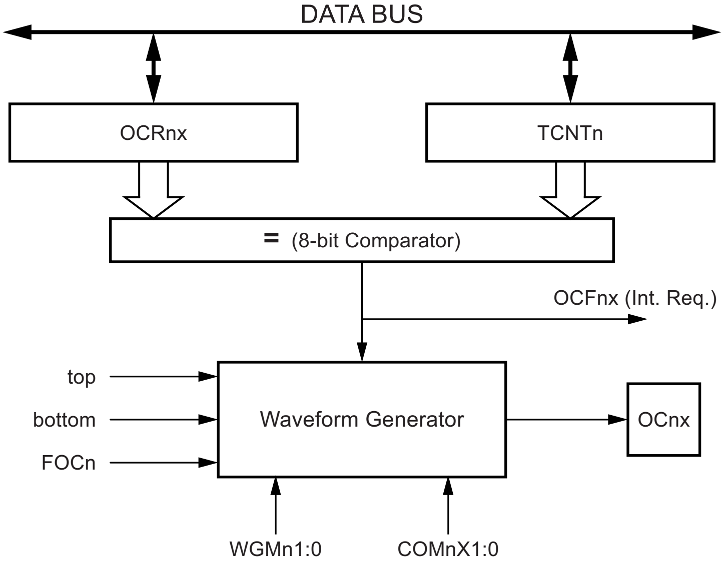
The Timer/Counter overflow flag (TOV0) is set according to the mode of operation selected by the WGM02:0 bits. TOV0 can be used for generating a CPU interrupt.

15.5 Output Compare Unit

The 8-bit comparator continuously compares TCNT0 with the output compare registers (OCR0A and OCR0B). Whenever TCNT0 equals OCR0A or OCR0B, the comparator signals a match. A match will set the output compare flag (OCF0A or OCF0B) at the next timer clock cycle. If the corresponding interrupt is enabled, the output compare flag generates an output compare interrupt. The output compare flag is automatically cleared when the interrupt is executed. Alternatively, the flag can be cleared by software by writing a logical one to its I/O bit location. The waveform generator uses the match signal to generate an output according to operating mode set by the WGM02:0 bits and compare output mode (COM0x1:0) 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 (Section 15.7 “Modes of Operation” on page 79).

Figure 15-3 shows a block diagram of the output compare unit.

Figure 15-3. Output Compare Unit, Block Diagram



The OCR0x registers are 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 OCR0x compare registers 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 OCR0x register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR0x buffer register, and if double buffering is disabled the CPU will access the OCR0x directly.

15.5.1 Force Output Compare

In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a one to the force output compare (FOC0x) bit. Forcing compare match will not set the OCF0x flag or reload/clear the timer, but the OC0x pin will be updated as if a real compare match had occurred (the COM0x1:0 bits settings define whether the OC0x pin is set, cleared or toggled).

15.5.2 Compare Match Blocking by TCNT0 Write

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

15.5.3 Using the Output Compare Unit

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

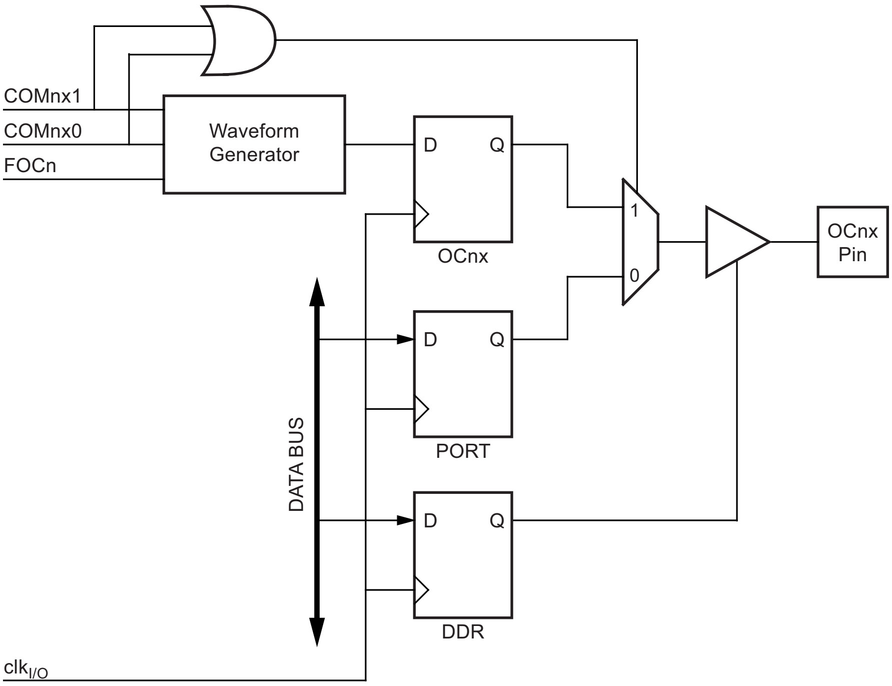
The setup of the OC0x should be performed before setting the data direction register for the port pin to output. The easiest way of setting the OC0x value is to use the force output compare (FOC0x) strobe bits in normal mode. The OC0x registers keep their values even when changing between waveform generation modes.

Be aware that the COM0x1:0 bits are not double buffered together with the compare value. Changing the COM0x1:0 bits will take effect immediately.

15.6 Compare Match Output Unit

The compare output mode (COM0x1:0) bits have two functions. The waveform generator uses the COM0x1:0 bits for defining the output compare (OC0x) state at the next compare match. Also, the COM0x1:0 bits control the OC0x pin output source. Figure 15-4 on page 79 shows a simplified schematic of the logic affected by the COM0x1:0 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 COM0x1:0 bits are shown. When referring to the OC0x state, the reference is for the internal OC0x register, not the OC0x pin. If a system reset occur, the OC0x register is reset to “0”.

Figure 15-4. Compare Match Output Unit, Schematic



The general I/O port function is overridden by the Output Compare (OC0x) from the Waveform Generator if either of the COM0x1:0 bits are set. However, the OC0x 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 OC0x pin (DDR_OC0x) must be set as output before the OC0x 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 OC0x state before the output is enabled. Note that some COM0x1:0 bit settings are reserved for certain modes of operation. See Section 15.9 "Register Description" on page 85.

15.6.1 Compare Output Mode and Waveform Generation

The waveform generator uses the COM0x1:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COM0x1:0 = 0 tells the waveform generator that no action on the OC0x register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 15-2 on page 85. For fast PWM mode, refer to Table 15-3 on page 85, and for phase correct PWM refer to Table 15-4 on page 85.

A change of the COM0x1:0 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 FOC0x strobe bits.

15.7 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 (WGM02:0) and compare output mode (COM0x1:0) bits. The compare output mode bits do not affect the counting sequence, while the waveform generation mode bits do. The COM0x1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM0x1:0 bits control whether the output should be set, cleared, or toggled at a compare match (see Section 16.8 "Compare Match Output Unit" on page 100). For detailed timing information see Section 15.8 "Timer/Counter Timing Diagrams" on page 83.

15.7.1 Normal Mode

The simplest mode of operation is the normal mode (WGM02:0 = 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.

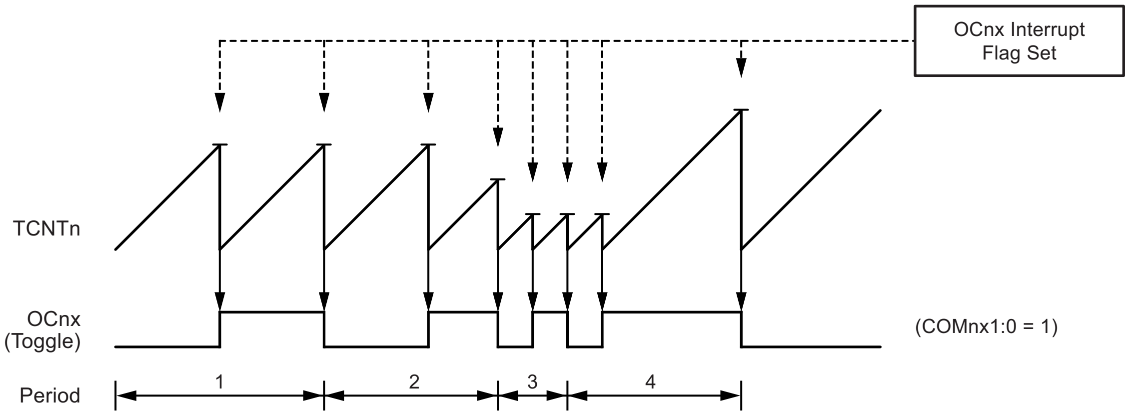
The output compare unit can be used to generate interrupts at some given time. Using the output compare to generate waveforms in normal mode is not recommended, since this will occupy too much of the CPU time.

15.7.2 Clear Timer on Compare Match (CTC) Mode

In clear timer on compare or CTC mode (WGM02:0 = 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 15-5. The counter value (TCNT0) increases until a compare match occurs between TCNT0 and OCR0A, and then counter (TCNT0) is cleared.

Figure 15-5. 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:

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

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

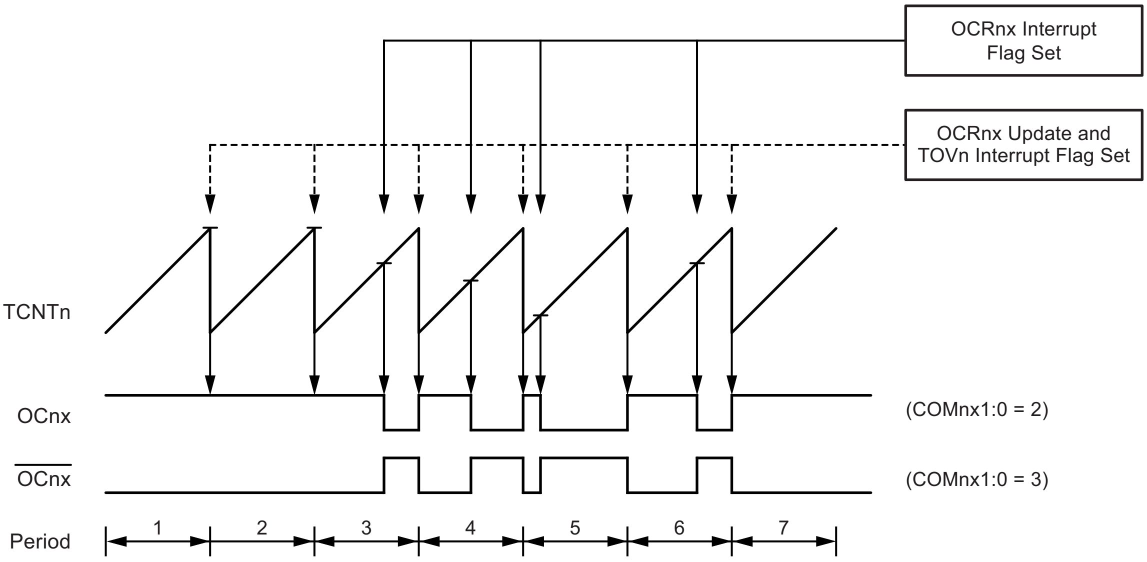
As for the normal mode of operation, the TOV0 flag is set in the same timer clock cycle that the counter counts from MAX to 0x00.

15.7.3 Fast PWM Mode

The fast pulse width modulation or fast PWM mode (WGM02:0 = 3 or 7) 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 TOP then restarts from BOTTOM. TOP is defined as 0xFF when WGM2:0 = 3, and OCR0A when WGM2:0 = 7. In non-inverting compare output mode, the output compare (OC0x) is cleared on the compare match between TCNT0 and OCR0x, 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 TOP value. The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 15-6. 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 OCR0x and TCNT0.

Figure 15-6. Fast PWM Mode, Timing Diagram



The Timer/Counter overflow flag (TOV0) is set each time the counter reaches TOP. 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 OC0x pins. Setting the COM0x1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM0x1:0 to three: Setting the COM0A1:0 bits to one allows the OC0A pin to toggle on compare matches if the WGM02 bit is set. This option is not available for the OC0B pin (See Table 15-3 on page 85). The actual OC0x 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 OC0x register at the compare match between OCR0x and TCNT0, and clearing (or setting) the OC0x 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:

$$f_{OCnxPWM} = \frac{f_{\text{clk\_I/O}}}{N \cdot 256}$$

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.)

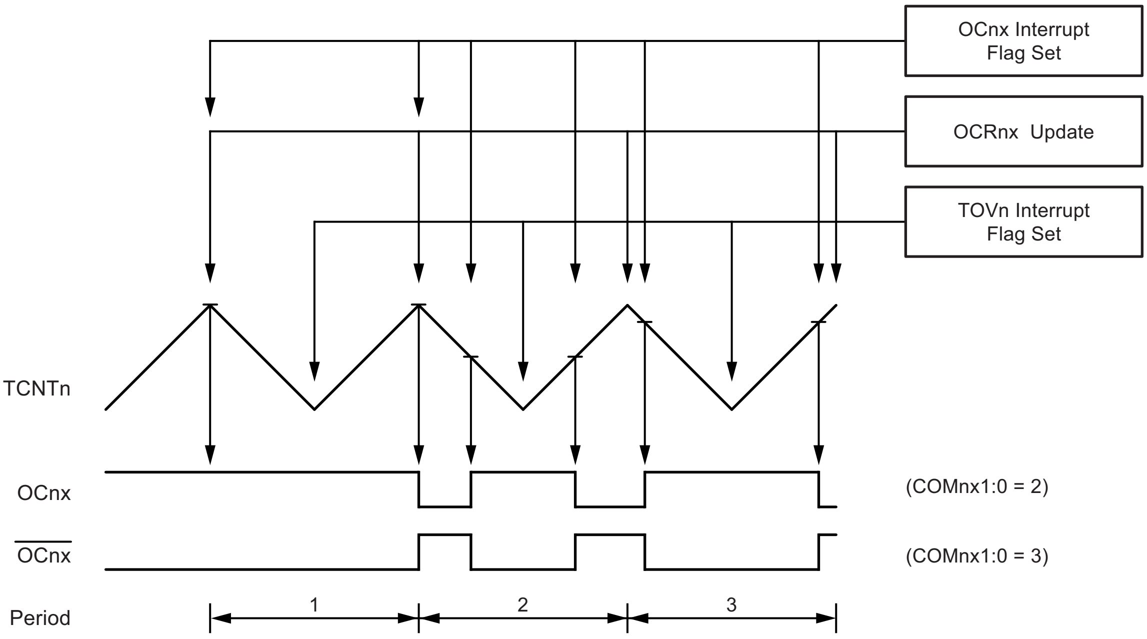
A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC0x to toggle its logical level on each compare match (COM0x1:0 = 1). The waveform generated will have a maximum frequency of $f_{OC0} = f_{clk\_I/O}/2$ when OCR0A is set to zero. This feature is similar to the OC0A toggle in CTC mode, except the double buffer feature of the output compare unit is enabled in the fast PWM mode.

15.7.4 Phase Correct PWM Mode

The phase correct PWM mode (WGM02:0 = 1 or 5) 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 TOP and then from TOP to BOTTOM. TOP is defined as 0xFF when WGM2:0 = 1, and OCR0A when WGM2:0 = 5. In non-inverting compare output mode, the output compare (OC0x) is cleared on the compare match between TCNT0 and OCR0x while upcounting, and set on the compare match while down-counting. 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.

In phase correct PWM mode the counter is incremented until the counter value matches TOP. When the counter reaches TOP, it changes the count direction. The TCNT0 value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 15-7. 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 OCR0x and TCNT0.

Figure 15-7. 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 OC0x pins. Setting the COM0x1:0 bits to two will produce a non-inverted PWM. An inverted PWM output can be generated by setting the COM0x1:0 to three: Setting the COM0A0 bits to one allows the OC0A pin to toggle on compare matches if the WGM02 bit is set. This option is not available for the OC0B pin (See Table 15-4 on page 85). The actual OC0x 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 OC0x register at the compare match between OCR0x and TCNT0 when the counter increments, and setting (or clearing) the OC0x register at compare match between OCR0x and TCNT0 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnxPCPWM} = \frac{f_{\text{clk\_I/O}}}{N \cdot 510}$$

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 15-7 on page 82 OCnx 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 15-7 on page 82. 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.

15.8 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 15-8 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 15-8. Timer/Counter Timing Diagram, no Prescaling

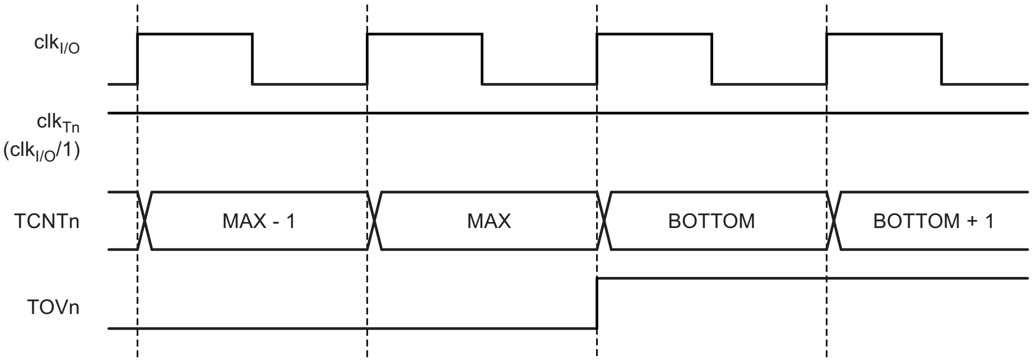


Figure 15-9 shows the same timing data, but with the prescaler enabled.

Figure 15-9. Timer/Counter Timing Diagram, with Prescaler ($f_{clk\_I/O}/8$)

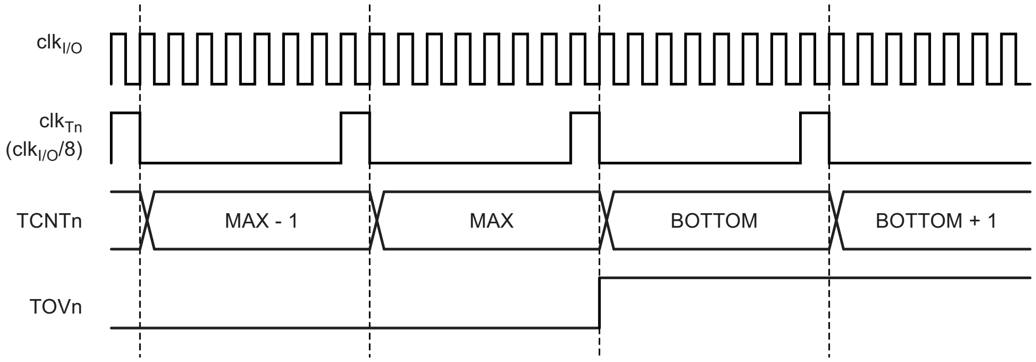


Figure 15-10 shows the setting of OCF0B in all modes and OCF0A in all modes except CTC mode and PWM mode, where OCR0A is TOP.

Figure 15-10.Timer/Counter Timing Diagram, Setting of OCF0x, with Prescaler ($f_{clk\_I/O}/8$)

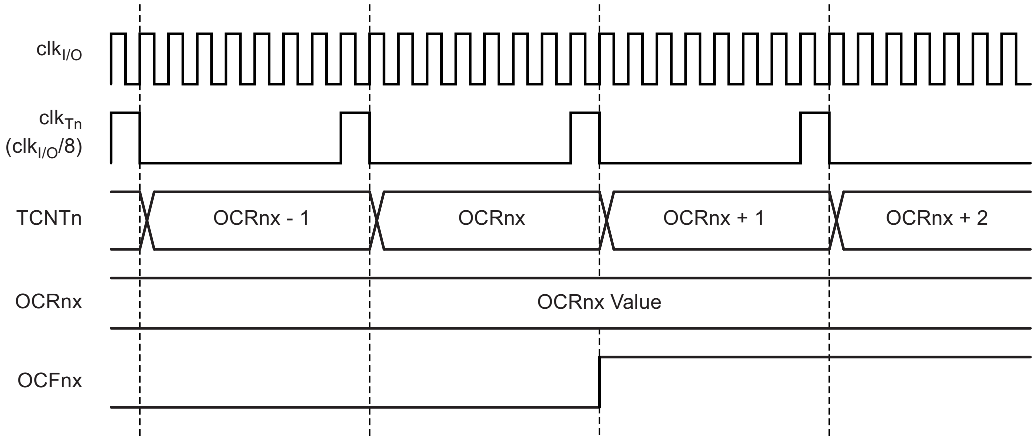
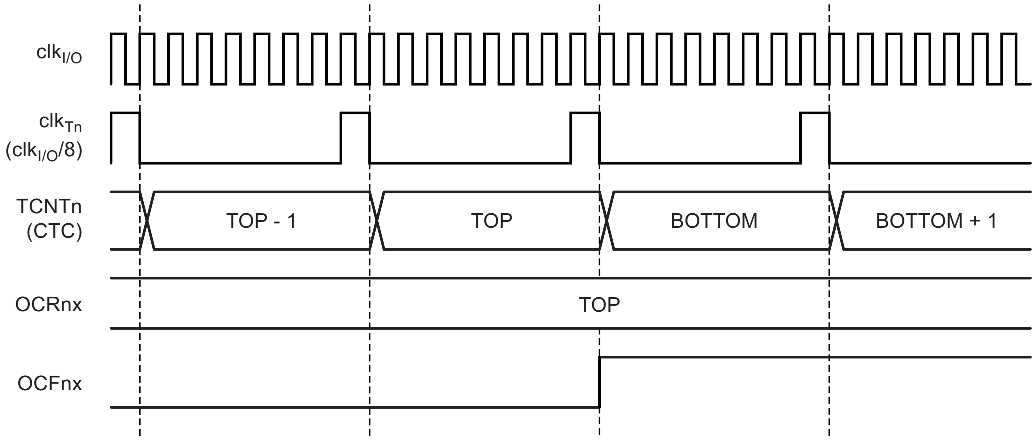


Figure 15-11 shows the setting of OCF0A and the clearing of TCNT0 in CTC mode and fast PWM mode where OCR0A is TOP.

Figure 15-11.Timer/Counter Timing Diagram, Clear Timer on Compare Match mode, with Prescaler ($f_{clk\_I/O}/8$)



15.9 Register Description

15.9.1 TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	**COM0A1**	**COM0A0**	**COM0B1**	**COM0B0**	**–**	**–**	**WGM01**	**WGM00**	**TCCR0A**
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	

- **Bits 7:6 – COM0A1:0: Compare Match Output A 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 WGM02:0 bit setting. Table 15-2 shows the COM0A1:0 bit functionality when the WGM02:0 bits are set to a normal or CTC mode (non-PWM).

Table 15-2. 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 15-3 shows the COM0A1:0 bit functionality when the WGM01:0 bits are set to fast PWM mode.

Table 15-3. Compare Output Mode, Fast PWM Mode(1)

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	WGM02 = 0: Normal port operation, OC0A disconnected. WGM02 = 1: Toggle OC0A on compare match.
1	0	Clear OC0A on compare match, set OC0A at BOTTOM, (non-inverting mode).
1	1	Set OC0A on compare match, clear OC0A at BOTTOM, (inverting mode).

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 BOTTOM. See Section 15.7.3 "Fast PWM Mode" on page 81 for more details.

Table 15-4 shows the COM0A1:0 bit functionality when the WGM02:0 bits are set to phase correct PWM mode.

Table 15-4. Compare Output Mode, Phase Correct PWM Mode(1)

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	WGM02 = 0: Normal port operation, OC0A disconnected. WGM02 = 1: Toggle OC0A on compare match.
1	0	Clear OC0A on compare match when up-counting. Set OC0A on compare match when down-counting.
1	1	Set OC0A on compare match when up-counting. Clear OC0A on compare match when down-counting.

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 Section 15.7.4 "Phase Correct PWM Mode" on page 82 for more details.

- **Bits 5:4 – COM0B1:0: Compare Match Output B Mode**

These bits control the output compare pin (OC0B) behavior. If one or both of the COM0B1:0 bits are set, the OC0B 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 OC0B pin must be set in order to enable the output driver.

When OC0B is connected to the pin, the function of the COM0B1:0 bits depends on the WGM02:0 bit setting. Table 15-2 on page 85 shows the COM0A1:0 bit functionality when the WGM02:0 bits are set to a normal or CTC mode (non-PWM).

Table 15-5. Compare Output Mode, non-PWM Mode

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected.
0	1	Toggle OC0B on compare match
1	0	Clear OC0B on compare match
1	1	Set OC0B on compare match

Table 15-6 shows the COM0B1:0 bit functionality when the WGM02:0 bits are set to fast PWM mode.

Table 15-6. Compare Output Mode, Fast PWM Mode(1)

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected.
0	1	Reserved
1	0	Clear OC0B on compare match, set OC0B at BOTTOM, (non-inverting mode).
1	1	Set OC0B on compare match, clear OC0B at BOTTOM, (inverting mode).

Note: 1. A special case occurs when OCR0B equals TOP and COM0B1 is set. In this case, the compare match is ignored, but the set or clear is done atBOTTOM. See Section 15.7.3 “Fast PWM Mode” on page 81 for more details.

Table 15-7 shows the COM0B1:0 bit functionality when the WGM02:0 bits are set to phase correct PWM mode.

Table 15-7. Compare Output Mode, Phase Correct PWM Mode(1)

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected.
0	1	Reserved
1	0	Clear OC0B on compare match when up-counting. Set OC0B on compare match when down-counting.
1	1	Set OC0B on compare match when up-counting. Clear OC0B on compare match when down-counting.

Note: 1. A special case occurs when OCR0B equals TOP and COM0B1 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See Section 15.7.4 “Phase Correct PWM Mode” on page 82 for more details.

- **Bits 3:2 – Reserved**

These bits are reserved bits in the ATmega164P-B/324P-B/644P-B and will always read as zero.

- **Bits 1:0 – WGM01:0: Waveform Generation Mode**

Combined with the WGM02 bit found in the TCCR0B 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 15-8. Modes of operation supported by the Timer/Counter unit are: normal mode (counter), clear timer on compare match (CTC) mode, and two types of pulse width modulation (PWM) modes (see Section 16.9 “Modes of Operation” on page 101).

Table 15-8. Waveform Generation Mode Bit Description

Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, phase correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, phase correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Notes: 1. MAX = 0xFF

2. BOTTOM = 0x00

15.9.2 TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	**FOC0A**	**FOC0B**	–	–	**WGM02**	**CS02**	**CS01**	**CS00**	**TCCR0B**
Read/Write	W	W	R	R	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 WGM bits specify a non-PWM mode.

However, for ensuring compatibility with future devices, this bit must be set to zero when TCCR0B 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 – FOC0B: Force Output Compare B**

The FOC0B 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 TCCR0B is written when operating in PWM mode. When writing a logical one to the FOC0B bit, an immediate compare match is forced on the waveform generation unit. The OC0B output is changed according to its COM0B1:0 bits setting. Note that the FOC0B bit is implemented as a strobe. Therefore it is the value present in the COM0B1:0 bits that determines the effect of the forced compare.

A FOC0B strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR0B as TOP.

The FOC0B bit is always read as zero.

- **Bits 5:4 – Reserved**

These bits are reserved and will always read as zero.

- **Bit 3 – WGM02: Waveform Generation Mode**

See the description in the Section 15.9.1 “TCCR0A – Timer/Counter Control Register A” on page 85.

- **Bits 2:0 – CS02:0: Clock Select**

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

Table 15-9. 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.

15.9.3 TCNT0 – Timer/Counter Register

Bit	7	6	5	4	3	2	1	0	
0x26 (0x46)	**TCNT0[7:0]**								**TCNT0**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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

15.9.4 OCR0A – Output Compare Register A

Bit	7	6	5	4	3	2	1	0	
0x27 (0x47)	**OCR0A[7:0]**								**OCR0A**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The output compare register A contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an output compare interrupt, or to generate a waveform output on the OC0A pin.

15.9.5 OCR0B – Output Compare Register B

Bit	7	6	5	4	3	2	1	0	
0x28 (0x48)	**OCR0B[7:0]**								**OCR0B**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The output compare register B contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an output compare interrupt, or to generate a waveform output on the OC0B pin.

15.9.6 TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	**OCIE0B**	**OCIE0A**	**TOIE0**	**TIMSK0**
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:3 – Reserved**

These bits are reserved and will always read as zero.

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

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

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

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

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

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

15.9.7 TIFR0 – Timer/Counter 0 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x15 (0x35)	–	–	–	–	–	**OCF0B**	**OCF0A**	**TOV0**	**TIFR0**
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:3 – Reserved**

These bits are reserved and will always read as zero.

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

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

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

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

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

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

The setting of this flag is dependent of the WGM02:0 bit setting. Refer to Table 15-8 on page 87, Section 15-8 “Waveform Generation Mode Bit Description” on page 87.

16. 16-bit Timer/Counter1 and Timer/Counter3 with PWM

16.1 Features

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

16.2 Overview

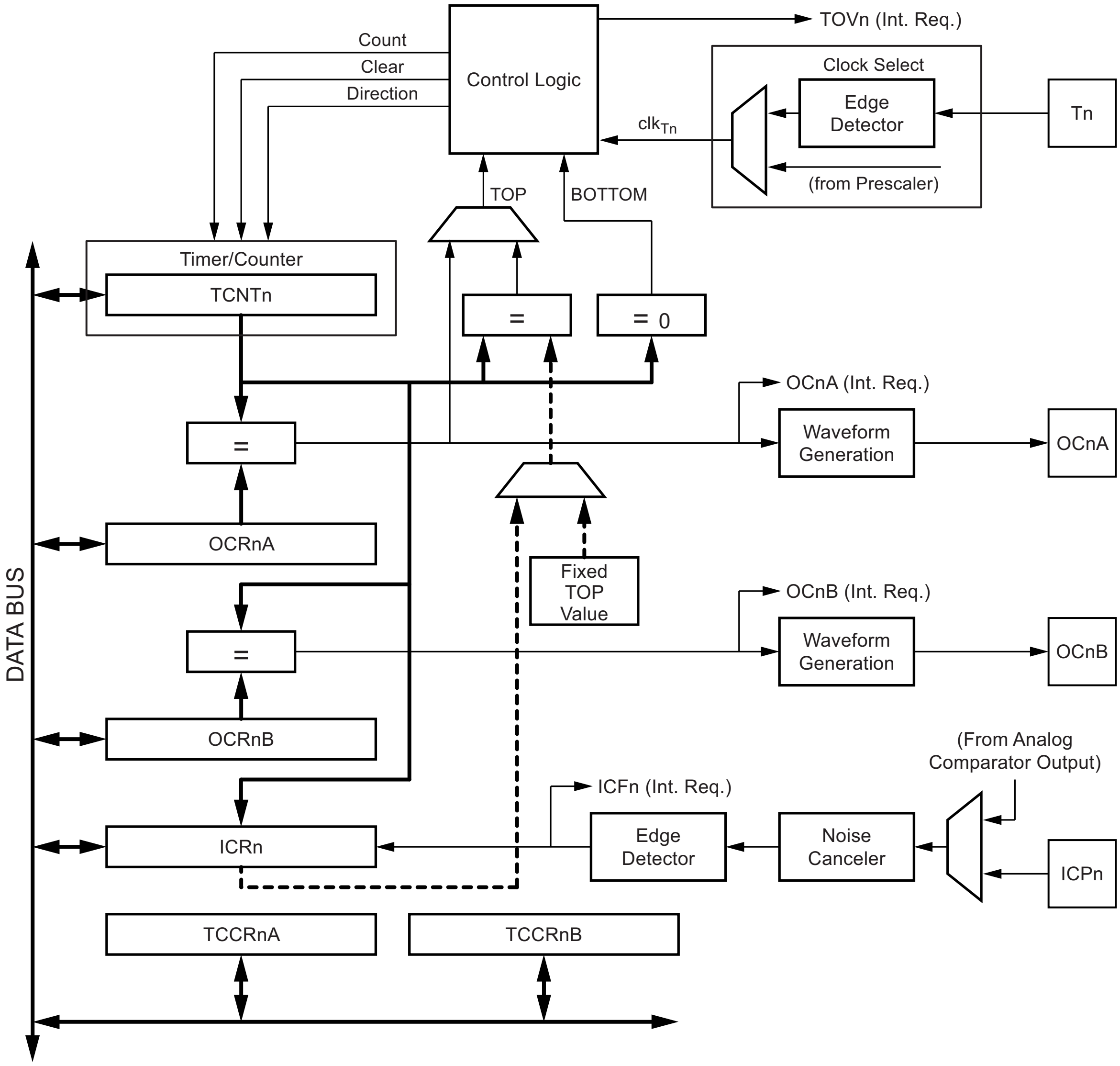
The 16-bit Timer/Counter unit allows accurate program execution timing (event management), wave generation, and signal timing measurement.

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 channel. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT1 for accessing Timer/Counter1 counter value and so on.

A simplified block diagram of the 16-bit Timer/Counter is shown in Figure 16-1 on page 91. For the actual placement of I/O pins, see Section 1. “Pin Configurations” on page 3. CPU accessible I/O registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O register and bit locations are listed in the Section 16.11 “Register Description” on page 109.

The PRTIM1 bit in Section 10.12.3 “PRR0 – Power Reduction Register 0” on page 39 must be written to zero to enable Timer/Counter1 module.

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



Note: 1. Refer to Figure 1-1 on page 3 and Section 14.3 “Alternate Port Functions” on page 62 for Timer/Counter1 pin placement and description.

16.2.1 Registers

The Timer/Counter (TCNTn), output compare registers (OCRnA/B/C), and input capture register (ICRn) are all 16-bit registers. Special procedures must be followed when accessing the 16-bit registers. These procedures are described in the section Section 16.3 “Accessing 16-bit Registers” on page 92. The Timer/Counter control registers (TCCRnA/B/C) 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 (TIFRn). All interrupts are individually masked with the timer interrupt mask register (TIMSKn). TIFRn and TIMSKn are not shown in the figure.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the Tn 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 (clkTn).

The double buffered output compare registers (OCRnA/B/C) 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 (OCnA/B/C). Section 16.7 “Output Compare Units” on page 98. The compare match event will also set the compare match flag (OCFnA/B/C) 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 (ICPn) or on the analog comparator pins (Section 22. “AC - Analog Comparator” on page 204) 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 OCRnA register, the ICRn register, or by a set of fixed values. When using OCRnA as TOP value in a PWM mode, the OCRnA 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 ICRn register can be used as an alternative, freeing the OCRnA to be used as PWM output.

16.2.2 Definitions

The following definitions are used extensively throughout the section:

Table 16-1. Definitions

Parameter	Definition
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 OCRnA or ICRn register. The assignment is dependent of the mode of operation.

16.3 Accessing 16-bit Registers

The TCNTn, OCRnA/B/C, and ICRn 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 OCRnA/B/C 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 OCRnA/B/C and ICRn registers. Note that when using "C", the compiler handles the 16-bit access.

Assembly Code Examples[1]

```
...
; Set TCNTn to 0x01FF
ldi    r17,0x01
ldi    r16,0xFF
out    TCNTnH,r17
out    TCNTnL,r16
; Read TCNTn into r17:r16
in     r16,TCNTnL
in     r17,TCNTnH
...
```

C Code Examples[1]

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

Note: 1. The example code assumes that the part specific header file is included.
For I/O registers located in extended I/O map, "IN", "OUT", "SBIS", "SBIC", "CBI", and "SBI" instructions must be replaced with instructions that allow access to extended I/O. Typically "LDS" and "STS" combined with "SBRS", "SBRC", "SBR", and "CBR".

The assembly code example returns the TCNTn 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 TCNTn register contents. Reading any of the OCRnA/B/C or ICRn registers can be done by using the same principle.

Assembly Code Example[1]

```
TIM16_ReadTCNTn:
        ; Save global interrupt flag
        in     r18,SREG
        ; Disable interrupts
        cli
        ; Read TCNTn into r17:r16
        in     r16,TCNTnL
        in     r17,TCNTnH
        ; Restore global interrupt flag
        out    SREG,r18
        ret
```

C Code Example[1]

```
unsigned int TIM16_ReadTCNTn( void )
{
        unsigned char sreg;
        unsigned int i;
        /* Save global interrupt flag */
        sreg = SREG;
        /* Disable interrupts */
        _CLI();
        /* Read TCNTn into i */
        i = TCNTn;
        /* Restore global interrupt flag */
        SREG = sreg;
        return i;
}
```

Note: 1. The example code assumes that the part specific header file is included.
For I/O registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

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

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

Assembly Code Example(1)

```
       TIM16_WriteTCNTn:
              ; Save global interrupt flag
              in     r18,SREG
              ; Disable interrupts
              cli
              ; Set TCNTn to r17:r16
              out    TCNTnH,r17
              out    TCNTnL,r16
              ; Restore global interrupt flag
              out    SREG,r18
              ret
```

C Code Example(1)

```
       void TIM16_WriteTCNTn(unsigned int i)
       {
              unsigned char sreg;
              unsigned int i;
              /* Save global interrupt flag */
              sreg = SREG;
              /* Disable interrupts */
              _CLI();
              /* Set TCNTn to i */
              TCNTn = i;
              /* Restore global interrupt flag */
              SREG = sreg;
       }
```

Note: 1. The example code assumes that the part specific header file is included.
For I/O registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

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

16.3.1 Reusing the Temporary High Byte Register

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

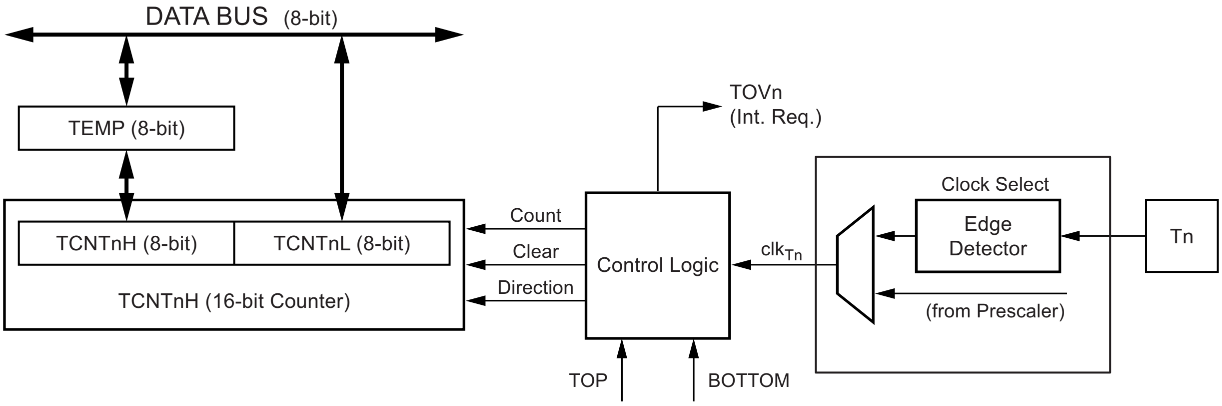
16.4 Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the clock select logic which is controlled by the clock select (CSn2:0) bits located in the Timer/Counter control register *B* (TCCRnB). For details on clock sources and prescaler, see Section 17.10 “Timer/Counter Prescaler” on page 131.

16.5 Counter Unit

The main part of the 16-bit Timer/Counter is the programmable 16-bit bi-directional counter unit. Figure 16-2 on page 96 shows a block diagram of the counter and its surroundings.

Figure 16-2. Counter Unit Block Diagram



Signal description (internal signals):

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

The 16-bit counter is mapped into two 8-bit I/O memory locations: counter high (TCNTnH) containing the upper eight bits of the counter, and counter low (TCNTnL) containing the lower eight bits. The TCNTnH register can only be indirectly accessed by the CPU. When the CPU does an access to the TCNTnH I/O location, the CPU accesses the high byte temporary register (TEMP). The temporary register is updated with the TCNTnH value when the TCNTnL is read, and TCNTnH is updated with the temporary register value when TCNTnL 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 TCNTn 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 (clkTn). The clkTn can be generated from an external or internal clock source, selected by the clock select bits (CSn2:0). When no clock source is selected (CSn2:0 = 0) the timer is stopped. However, the TCNTn value can be accessed by the CPU, independent of whether clkTn 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 (WGMn3:0) located in the Timer/Counter control registers A and B (TCCRnA and TCCRnB). There are close connections between how the counter behaves (counts) and how waveforms are generated on the output compare outputs OCnx. For more details about advanced counting sequences and waveform generation, see Section 16.9 "Modes of Operation" on page 101.

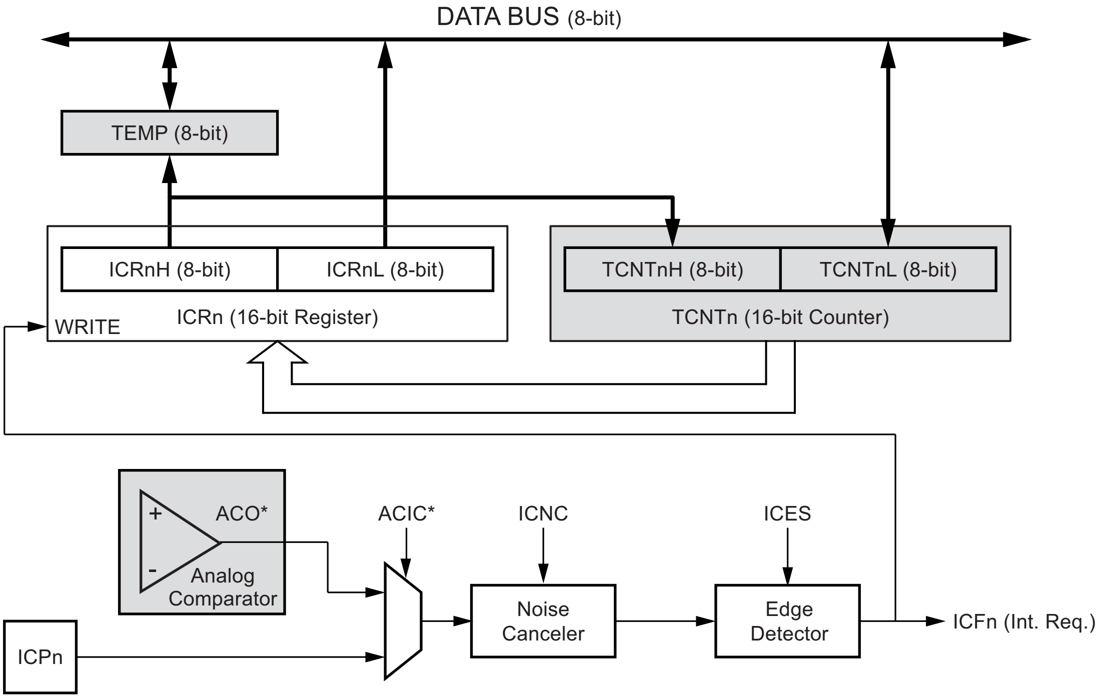
The Timer/Counter overflow flag (TOVn) is set according to the mode of operation selected by the WGMn3:0 bits. TOVn can be used for generating a CPU interrupt.

16.6 Input Capture Unit

The Timer/Counter incorporates an input capture unit that can capture external events and give them a time-stamp indicating time of occurrence. The external signal indicating an event, or multiple events, can be applied via the ICPn 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 16-3 on page 97. 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 16-3. Input Capture Unit Block Diagram



When a change of the logic level (an event) occurs on the input capture pin (ICPn), 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 (TCNTn) is written to the input capture register (ICRn). The input capture flag (ICFn) is set at the same system clock as the TCNTn value is copied into ICRn register. If enabled (ICIEn = 1), the input capture flag generates an input capture interrupt. The ICFn flag is automatically cleared when the interrupt is executed. Alternatively the ICFn 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 (ICRn) is done by first reading the low byte (ICRnL) and then the high byte (ICRnH). When the low byte is read the high byte is copied into the high byte temporary register (TEMP). When the CPU reads the ICRnH I/O location it will access the TEMP register.

The ICRn register can only be written when using a waveform generation mode that utilizes the ICRn register for defining the counter's TOP value. In these cases the waveform Generation mode (WGMn3:0) bits must be set before the TOP value can be written to the ICRn Register. When writing the ICRn register the high byte must be written to the ICRnH I/O location before the low byte is written to ICRnL.

For more information on how to access the 16-bit registers refer to Section 16.3 "Accessing 16-bit Registers" on page 92.

16.6.1 Input Capture Trigger Source

The main trigger source for the input capture unit is the input capture pin (ICPn). 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 (ICPn) and the analog comparator output (ACO) inputs are sampled using the same technique as for the Tn pin (Figure 16-1 on page 91). 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 ICRn to define TOP.

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

16.6.2 Noise Canceler

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

The noise canceler is enabled by setting the input capture noise canceler (ICNCn) bit in Timer/Counter control register B (TCCRnB). 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 ICRn register. The noise canceler uses the system clock and is therefore not affected by the prescaler.

16.6.3 Using the Input Capture Unit

The main challenge when using the input capture unit is to assign enough processor capacity for handling the incoming events. The time between two events is critical. If the processor has not read the captured value in the ICRn register before the next event occurs, the ICRn 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 ICRn 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 ICRn register has been read. After a change of the edge, the input capture flag (ICFn) must be cleared by software (writing a logical one to the I/O bit location). For measuring frequency only, the clearing of the ICFn flag is not required (if an interrupt handler is used).

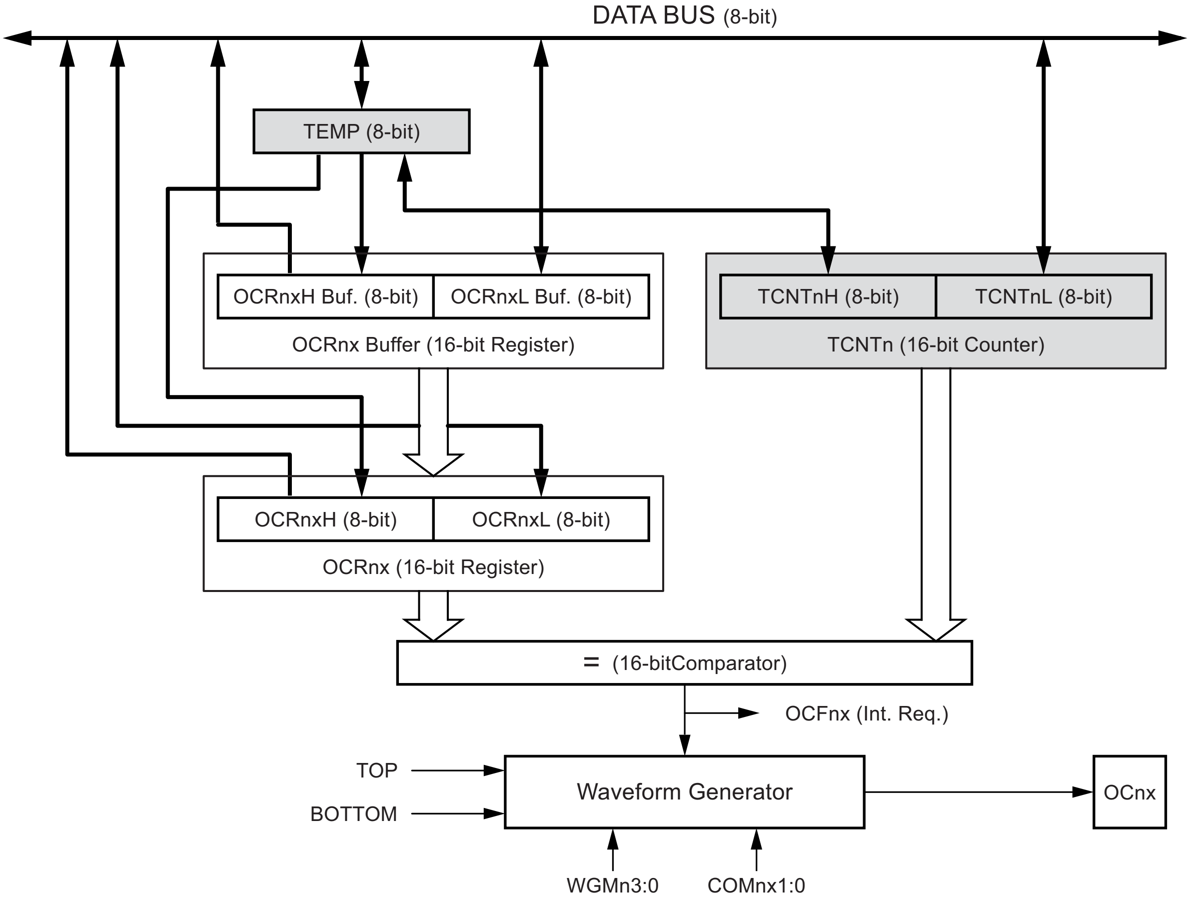
16.7 Output Compare Units

The 16-bit comparator continuously compares TCNTn with the output compare register (OCRnx). If TCNT equals OCRnx the comparator signals a match. A match will set the output compare flag (OCFnx) at the next timer clock cycle. If enabled (OCIEnx = 1), the output compare flag generates an output compare interrupt. The OCFnx flag is automatically cleared when the interrupt is executed. Alternatively the OCFnx 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 (WGMn3:0) bits and compare output mode (COMnx1:0) 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 (Section 16.9 “Modes of Operation” on page 101).

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 16-4 on page 99 shows a block diagram of the output compare unit. The small “n” in the register and bit names indicates the device number (n = n for Timer/Counter n), and the “x” indicates output compare unit (A/B/C). The elements of the block diagram that are not directly a part of the output compare unit are gray shaded.

Figure 16-4. Output Compare Unit, Block Diagram



The OCRnx 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 OCRnx 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 OCRnx register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCRnx buffer register, and if double buffering is disabled the CPU will access the OCRnx 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 OCRnx registers must be done via the TEMP register since the compare of all 16 bits is done continuously. The high byte (OCRnxH) 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 (OCRnxL) is written to the lower eight bits, the high byte will be copied into the upper 8-bits of either the OCRnx buffer or OCRnx compare register in the same system clock cycle.

For more information of how to access the 16-bit registers refer to Section 16.3 “Accessing 16-bit Registers” on page 92.

16.7.1 Force Output Compare

In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a one to the force output compare (FOCnx) bit. Forcing compare match will not set the OCFnx flag or reload/clear the timer, but the OCnx pin will be updated as if a real compare match had occurred (the COMn1:0 bits settings define whether the OCnx pin is set, cleared or toggled).

16.7.2 Compare Match Blocking by TCNTn Write

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

16.7.3 Using the Output Compare Unit

Since writing TCNTn in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNTn when using any of the output compare channels, independent of whether the Timer/Counter is running or not. If the value written to TCNTn equals the OCRnx value, the compare match will be missed, resulting in incorrect waveform generation. Do not write the TCNTn 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 TCNTn value equal to BOTTOM when the counter is downcounting.

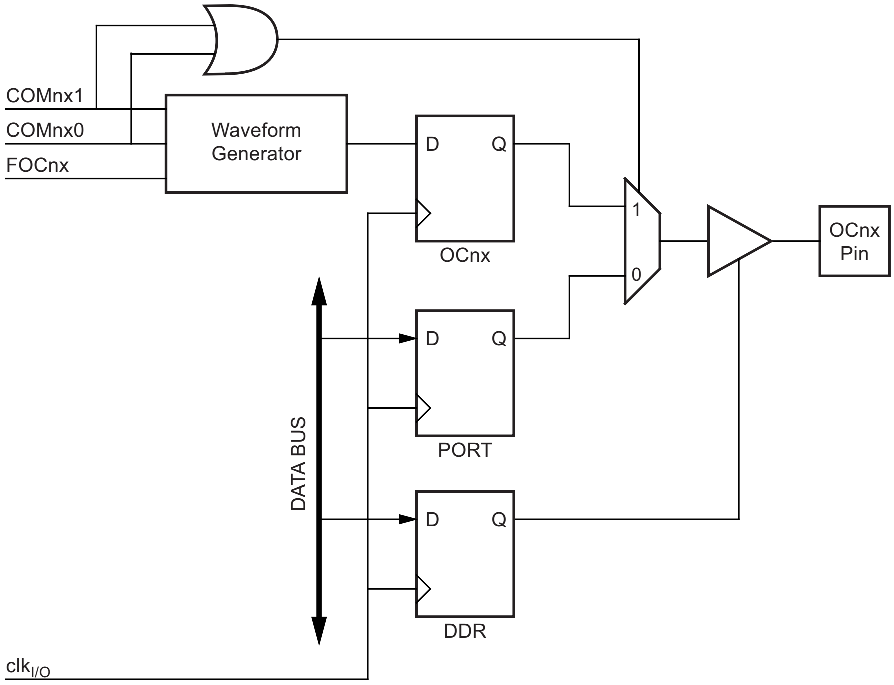
The setup of the OCnx should be performed before setting the data direction register for the port pin to output. The easiest way of setting the OCnx value is to use the force output compare (FOCnx) strobe bits in normal mode. The OCnx register keeps its value even when changing between waveform generation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

Be aware that the COMnx1:0 bits are not double buffered together with the compare value. Changing the COMnx1:0 bits will take effect immediately.

16.8 Compare Match Output Unit

The compare output mode (COMnx1:0) bits have two functions. The waveform generator uses the COMnx1:0 bits for defining the output oompare (OCnx) state at the next compare match. Secondly the COMnx1:0 bits control the OCnx pin output source. Figure 16-5 shows a simplified schematic of the logic affected by the COMnx1:0 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 COMnx1:0 bits are shown. When referring to the OCnx state, the reference is for the internal OCnx register, not the OCnx pin. If a system reset occur, the OCnx register is reset to "0".

Figure 16-5. Compare Match Output Unit, Schematic



The general I/O port function is overridden by the output compare (OCnx) from the waveform generator if either of the COMnx1:0 bits are set. However, the OCnx 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 OCnx pin (DDR_OCnx) must be set as output before the OCnx 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 16-2 on page 109, Table 16-3 on page 110 and Table 16-4 on page 110 for details.

The design of the output compare pin logic allows initialization of the OCnx state before the output is enabled. Note that some COMnx1:0 bit settings are reserved for certain modes of operation. Section 16.11 “Register Description” on page 109.

The COMnx1:0 bits have no effect on the input capture unit.

16.8.1 Compare Output Mode and Waveform Generation

The waveform generator uses the COMnx1:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COMnx1:0 = 0 tells the waveform generator that no action on the OCnx register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 16-2 on page 109. For fast PWM mode refer to Table 16-3 on page 110, and for phase correct and phase and frequency correct PWM refer to Table 16-4 on page 110.

A change of the COMnx1:0 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 FOCnx strobe bits.

16.9 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the output compare pins, is defined by the combination of the waveform generation mode (WGMn3:0) and compare output mode (COMnx1:0) bits. The compare output mode bits do not affect the counting sequence, while the waveform generation mode bits do. The COMnx1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COMnx1:0 bits control whether the output should be set, cleared or toggle at a compare match (Section 16.8 “Compare Match Output Unit” on page 100).

For detailed timing information refer to Section 16.10 “Timer/Counter Timing Diagrams” on page 107.

16.9.1 Normal Mode

The simplest mode of operation is the normal mode (WGMn3:0 = 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 (TOVn) will be set in the same timer clock cycle as the TCNTn becomes zero. The TOVn 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 TOVn 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.

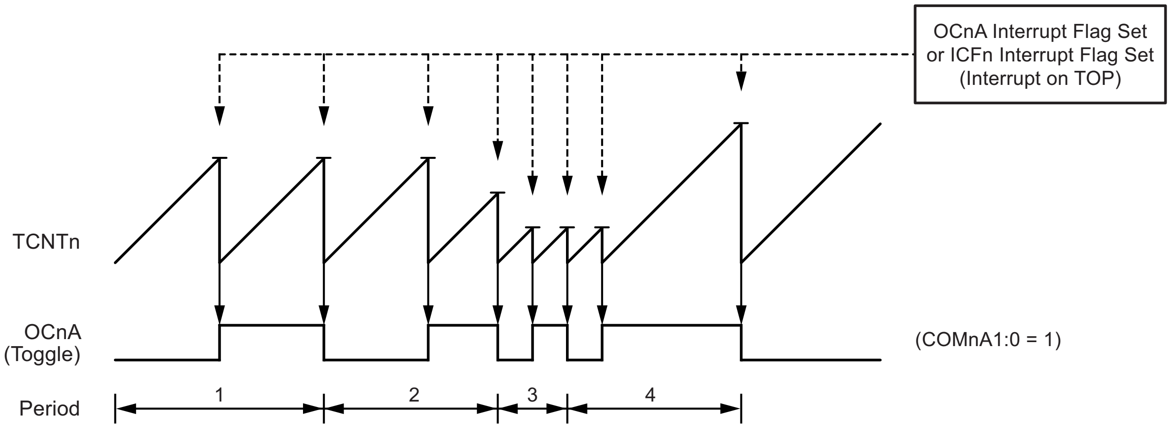
The output compare units can be used to generate interrupts at some given time. Using the output compare to generate waveforms in normal mode is not recommended, since this will occupy too much of the CPU time.

16.9.2 Clear Timer on Compare Match (CTC) Mode

In clear timer on compare or CTC mode (WGMn3:0 = 4 or 12), the OCRnA or ICRn register are used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNTn) matches either the OCRnA (WGMn3:0 = 4) or the ICRn (WGMn3:0 = 12). The OCRnA or ICRn 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 16-6 on page 102. The counter value (TCNTn) increases until a compare match occurs with either OCRnA or ICRn, and then counter (TCNTn) is cleared.

Figure 16-6. CTC Mode, Timing Diagram



An interrupt can be generated at each time the counter value reaches the TOP value by either using the OCFnA or ICFn 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 OCRnA or ICRn is lower than the current value of TCNTn, 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 OCRnA for defining TOP (WGMn3:0 = 15) since the OCRnA then will be double buffered.

For generating a waveform output in CTC mode, the OCnA output can be set to toggle its logical level on each compare match by setting the compare output mode bits to toggle mode (COMnA1:0 = 1). The OCnA value will not be visible on the port pin unless the data direction for the pin is set to output (DDR_OCnA = 1). The waveform generated will have a maximum frequency of $f_{OCnA} = f_{clk\_I/O}/2$ when OCRnA is set to zero (0x0000). The waveform frequency is defined by the following equation:

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

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

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

16.9.3 Fast PWM Mode

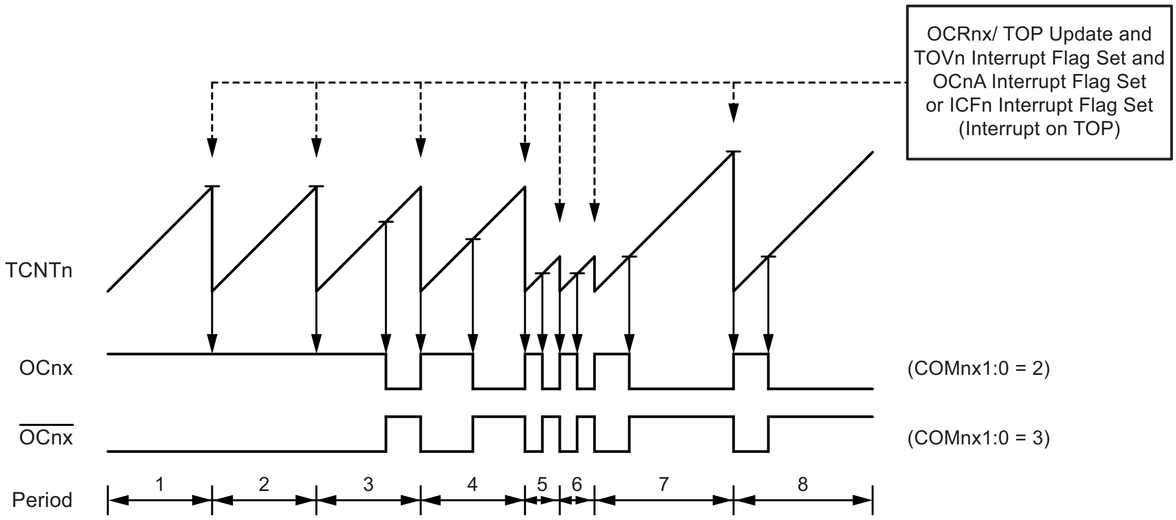
The fast pulse width modulation or fast PWM mode (WGMn3:0 = 5, 6, 7, 14, or 15) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. In non-inverting compare output mode, the output compare (OCnx) is cleared on the compare match between TCNTn and OCRnx, and set at BOTTOM. In inverting compare output mode 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 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 ICRn or OCRnA. The minimum resolution allowed is 2-bit (ICRn or OCRnA set to 0x0003), and the maximum resolution is 16-bit (ICRn or OCRnA 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 (WGMn3:0 = 5, 6, or 7), the value in ICRn (WGMn3:0 = 14), or the value in OCRnA (WGMn3:0 = 15). The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 16-7. The figure shows fast PWM mode when OCRnA or ICRn is used to define TOP. The TCNTn 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 TCNTn slopes represent compare matches between OCRnx and TCNTn. The OCnx interrupt flag will be set when a compare match occurs.

Figure 16-7. Fast PWM Mode, Timing Diagram



The Timer/Counter overflow flag (TOVn) is set each time the counter reaches TOP. In addition the OCnA or ICFn flag is set at the same timer clock cycle as TOVn is set when either OCRnA or ICRn 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 TCNTn and the OCRnx. Note that when using fixed TOP values the unused bits are masked to zero when any of the OCRnx registers are written.

The procedure for updating ICRn differs from updating OCRnA when used for defining the TOP value. The ICRn register is not double buffered. This means that if ICRn 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 ICRn value written is lower than the current value of TCNTn. 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 OCRnA register however, is double buffered. This feature allows the OCRnA I/O location to be written anytime. When the OCRnA I/O location is written the value written will be put into the OCRnA buffer register. The OCRnA compare register will then be updated with the value in the buffer register at the next timer clock cycle the TCNTn matches TOP. The update is done at the same timer clock cycle as the TCNTn is cleared and the TOVn flag is set.

Using the ICRn register for defining TOP works well when using fixed TOP values. By using ICRn, the OCRnA register is free to be used for generating a PWM output on OCnA. However, if the base PWM frequency is actively changed (by changing the TOP value), using the OCRnA 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 OCnx pins. Setting the COMnx1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COMnx1:0 to three (see Table on page 110). The actual OCnx value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OCnx). The PWM waveform is generated by setting (or clearing) the OCnx register at the compare match between OCRnx and TCNTn, and clearing (or setting) the OCnx 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:

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

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

The extreme values for the OCRnx register represents special cases when generating a PWM waveform output in the fast PWM mode. If the OCRnx is set equal to BOTTOM (0x0000) the output will be a narrow spike for each TOP+1 timer clock cycle. Setting the OCRnx equal to TOP will result in a constant high or low output (depending on the polarity of the output set by the COMnx1:0 bits.)

A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OCnA to toggle its logical level on each compare match (COMnA1:0 = 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 $f_{OCnA} = f_{clk\_I/O}/2$ when OCRnA is set to zero (0x0000). This feature is similar to the OCnA toggle in CTC mode, except the double buffer feature of the output compare unit is enabled in the fast PWM mode.

16.9.4 Phase Correct PWM Mode

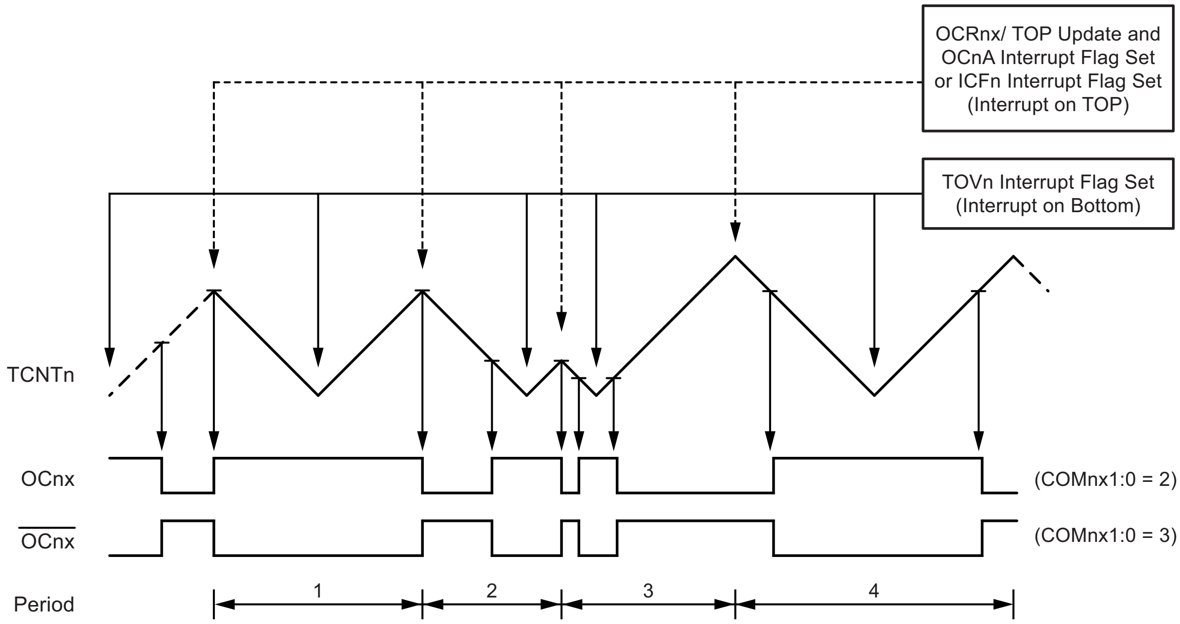
The phase correct pulse width modulation or phase correct PWM mode (WGMn3:0 = 1, 2, 3, 10, or 11) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is, like the phase and frequency correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting compare output mode, the output compare (OCnx) is cleared on the compare match between TCNTn and OCRnx while upcounting, and set on the compare match while downcounting. In inverting output compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

The PWM resolution for the phase correct PWM mode can be fixed to 8-, 9-, or 10-bit, or defined by either ICRn or OCRnA. The minimum resolution allowed is 2-bit (ICRn or OCRnA set to 0x0003), and the maximum resolution is 16-bit (ICRn or OCRnA 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 (WGMn3:0 = 1, 2, or 3), the value in ICRn (WGMn3:0 = 10), or the value in OCRnA (WGMn3:0 = 11). The counter has then reached the TOP and changes the count direction. The TCNTn value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 16-8 on page 105. The figure shows phase correct PWM mode when OCRnA or ICRn is used to define TOP. The TCNTn 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 TCNTn slopes represent compare matches between OCRnx and TCNTn. The OCnx interrupt flag will be set when a compare match occurs.

Figure 16-8. Phase Correct PWM Mode, Timing Diagram



The Timer/Counter overflow flag (TOVn) is set each time the counter reaches BOTTOM. When either OCRnA or ICRn is used for defining the TOP value, the OCnA or ICFn flag is set accordingly at the same timer clock cycle as the OCRnx 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 TCNTn and the OCRnx. Note that when using fixed TOP values, the unused bits are masked to zero when any of the OCRnx registers are written. As the third period shown in Figure 16-8 illustrates, changing the TOP actively while the Timer/Counter is running in the phase correct mode can result in an unsymmetrical output. The reason for this can be found in the time of update of the OCRnx register. Since the OCRnx update occurs at TOP, the PWM period starts and ends at TOP. This implies that the length of the falling slope is determined by the previous TOP value, while the length of the rising slope is determined by the new TOP value. When these two values differ the two slopes of the period will differ in length. The difference in length gives the unsymmetrical result on the output.

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

In phase correct PWM mode, the compare units allow generation of PWM waveforms on the OCnx pins. Setting the COMnx1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COMnx1:0 to three (See Table on page 110). The actual OCnx value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OCnx). The PWM waveform is generated by setting (or clearing) the OCnx register at the compare match between OCRnx and TCNTn when the counter increments, and clearing (or setting) the OCnx register at compare match between OCRnx and TCNTn when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

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

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

The extreme values for the OCRnx register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCRnx 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.

16.9.5 Phase and Frequency Correct PWM Mode

The phase and frequency correct Pulse Width Modulation, or phase and frequency correct PWM mode (WGMn3:0 = 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 (OCnx) is cleared on the compare match between TCNTn and OCRnx while upcounting, and set on the compare match while downcounting. In inverting compare output mode, the operation is inverted. The dual-slope operation gives a lower maximum operation frequency compared to the single-slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

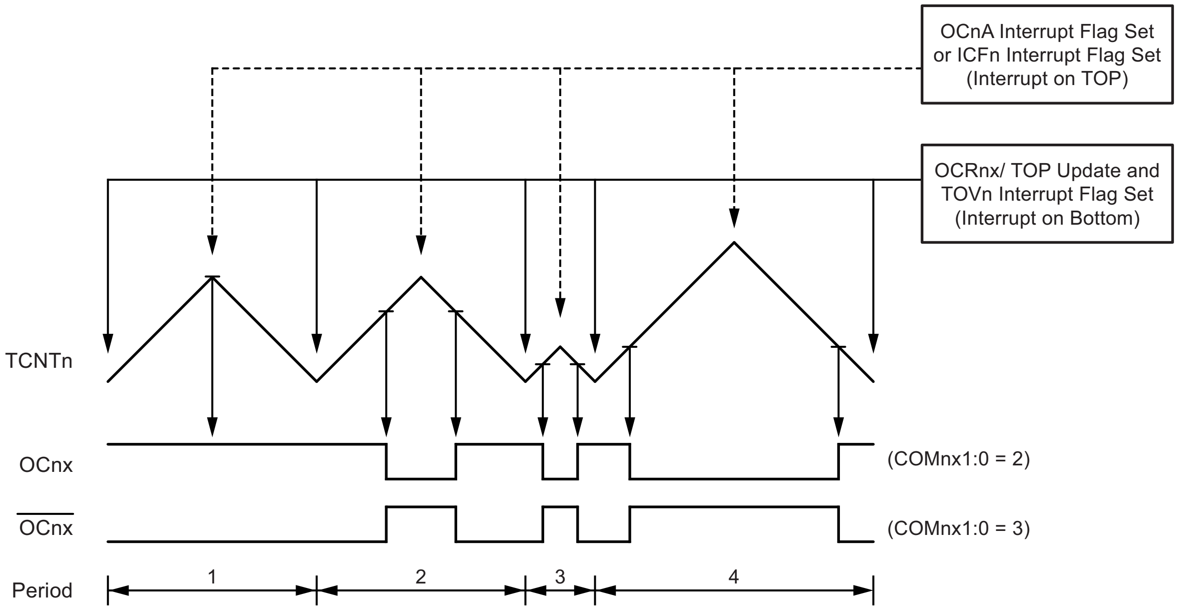
The main difference between the phase correct, and the phase and frequency correct PWM mode is the time the OCRnx register is updated by the OCRnx buffer register, (see Figure 16-8 on page 105 and Figure 16-9).

The PWM resolution for the phase and frequency correct PWM mode can be defined by either ICRn or OCRnA. The minimum resolution allowed is 2-bit (ICRn or OCRnA set to 0x0003), and the maximum resolution is 16-bit (ICRn or OCRnA 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 ICRn (WGMn3:0 = 8), or the value in OCRnA (WGMn3:0 = 9). The counter has then reached the TOP and changes the count direction. The TCNTn 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 16-9. The figure shows phase and frequency correct PWM mode when OCRnA or ICRn is used to define TOP. The TCNTn 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 TCNTn slopes represent compare matches between OCRnx and TCNTn. The OCnx interrupt flag will be set when a compare match occurs.

Figure 16-9. Phase and Frequency Correct PWM Mode, Timing Diagram



The Timer/Counter overflow flag (TOVn) is set at the same timer clock cycle as the OCRnx registers are updated with the double buffer value (at BOTTOM). When either OCRnA or ICRn is used for defining the TOP value, the OCnA or ICFn flag set when TCNTn 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 TCNTn and the OCRnx.

As Figure 16-9 on page 106 shows the output generated is, in contrast to the phase correct mode, symmetrical in all periods. Since the OCRnx 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 ICRn register for defining TOP works well when using fixed TOP values. By using ICRn, the OCRnA register is free to be used for generating a PWM output on OCnA. However, if the base PWM frequency is actively changed by changing the TOP value, using the OCRnA 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 OCnx pins. Setting the COMnx1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COMnx1:0 to three (See Table on page 110). The actual OCnx value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OCnx). The PWM waveform is generated by setting (or clearing) the OCnx register at the compare match between OCRnx and TCNTn when the counter increments, and clearing (or setting) the OCnx register at compare match between OCRnx and TCNTn when the counter decrements. The PWM frequency for the output when using phase and frequency correct PWM can be calculated by the following equation:

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

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

The extreme values for the OCRnx register represents special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCRnx 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.

16.10 Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock ($\mathrm{clk}_{\mathrm{Tn}}$) 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 OCRnx register is updated with the OCRnx buffer value (only for modes utilizing double buffering). Figure 16-10 shows a timing diagram for the setting of OCFnx.

Figure 16-10. Timer/Counter Timing Diagram, Setting of OCFnx, no Prescaling

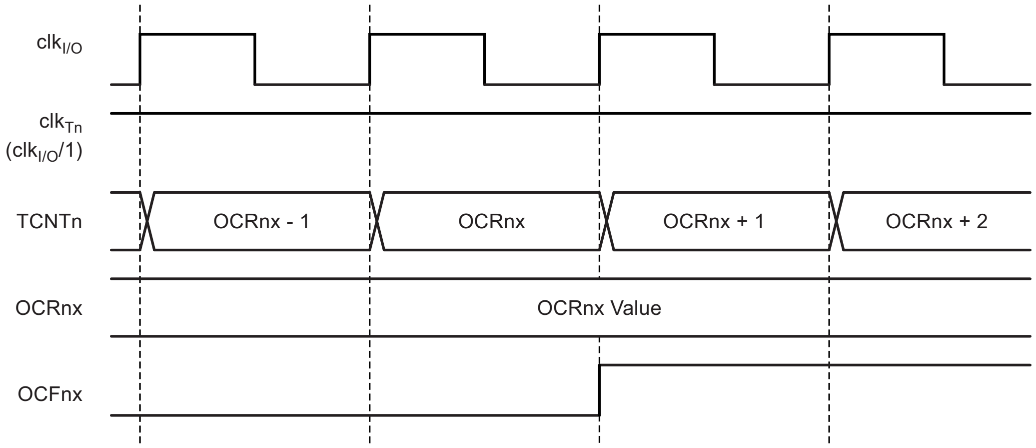


Figure 16-11 shows the same timing data, but with the prescaler enabled.

Figure 16-11. Timer/Counter Timing Diagram, Setting of OCFnx, with Prescaler ($f_{clk\_I/O}/8$)

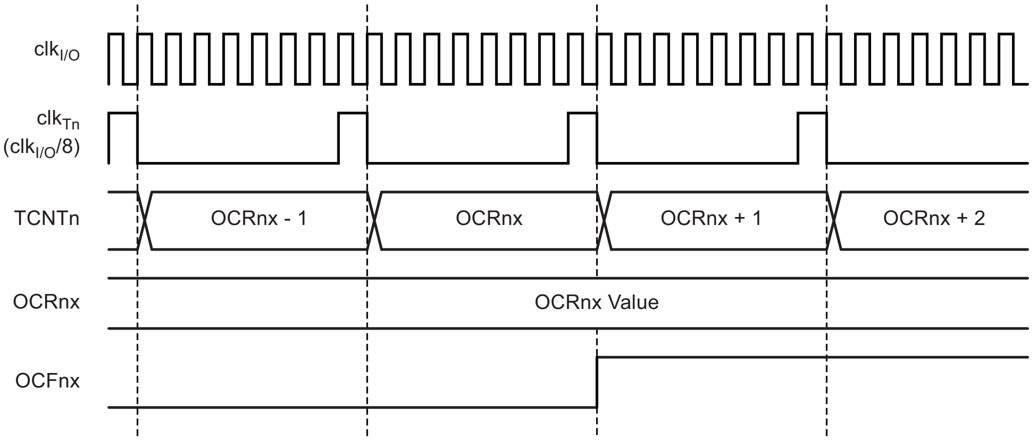


Figure 16-12 shows the count sequence close to TOP in various modes. When using phase and frequency correct PWM mode the OCRnx register is updated at BOTTOM. The timing diagrams will be the same, but TOP should be replaced by BOTTOM, TOP-1 by BOTTOM+1 and so on. The same renaming applies for modes that set the TOVn flag at BOTTOM.

Figure 16-12. Timer/Counter Timing Diagram, no Prescaling

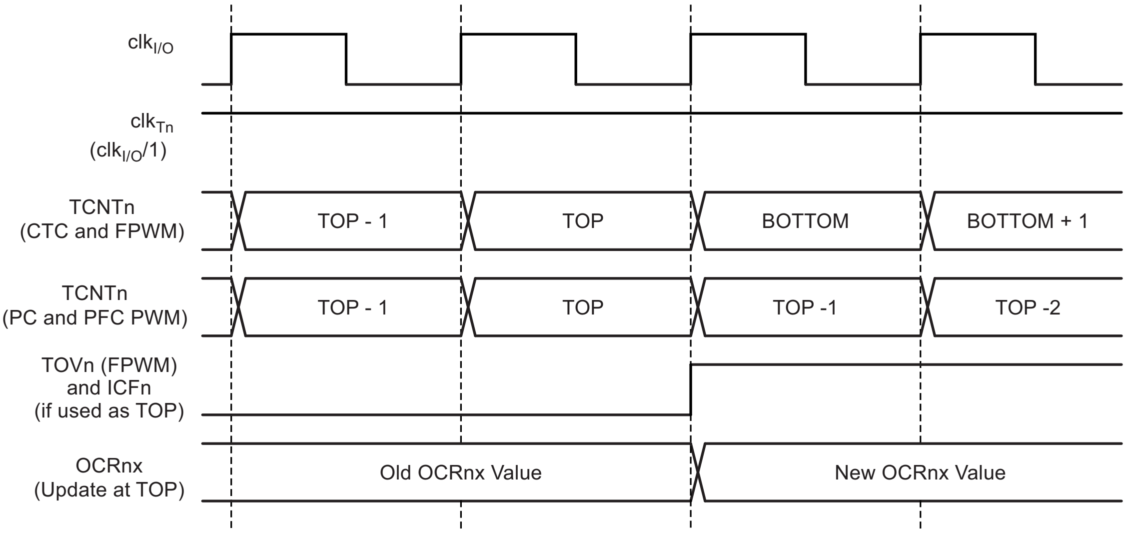
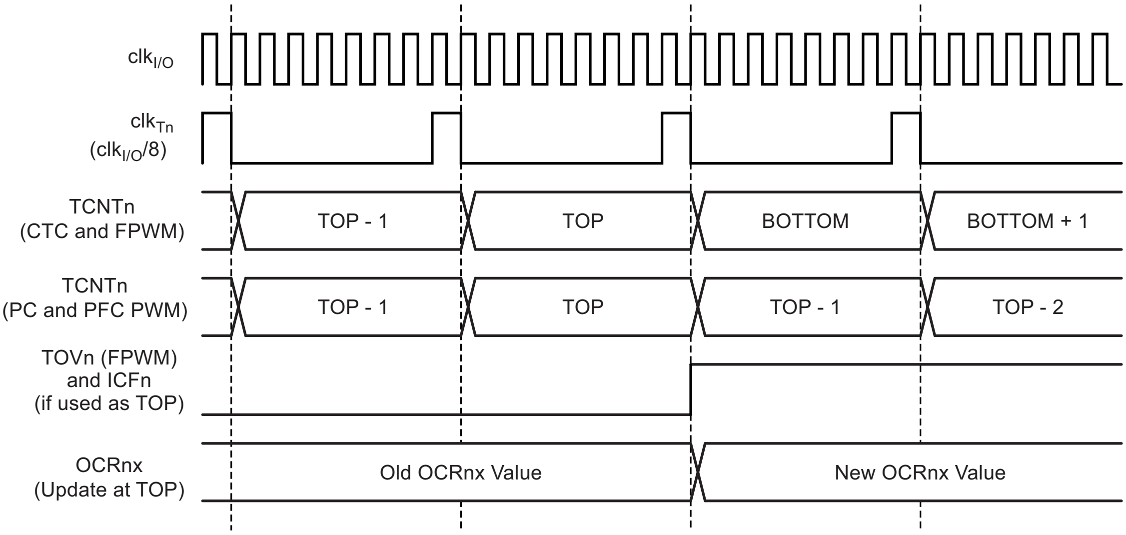


Figure 16-13 shows the same timing data, but with the prescaler enabled.

Figure 16-13. Timer/Counter Timing Diagram, with Prescaler ($f_{clk\_I/O}/8$)



16.11 Register Description

16.11.1 TCCRnA – Timer/Counter n Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	**COMnA1**	**COMnA0**	**COMnB1**	**COMnB0**	**–**	**–**	**WGMn1**	**WGMn0**	**TCCRnA**
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 – COMnA1:0: Compare Output Mode for Channel A**

- **Bit 5:4 – COMnB1:0: Compare Output Mode for Channel B**

The COMnA1:0 and COMnB1:0 control the output compare pins (OCnA and OCnB respectively) behavior. If one or both of the COMnA1:0 bits are written to one, the OCnA output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COMnB1:0 bit are written to one, the OCnB 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 OCnA or OCnB pin must be set in order to enable the output driver.

When the OCnA or OCnB is connected to the pin, the function of the COMnx1:0 bits is dependent of the WGMn3:0 bits setting. Table 16-2 shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to a normal or a CTC mode (non-PWM).

Table 16-2. Compare Output Mode, non-PWM

COMnA1/COMnB1	COMnA0/COMnB0	Description
0	0	Normal port operation, OCnA/OCnB disconnected.
0	1	Toggle OCnA/OCnB on compare match.
1	0	Clear OCnA/OCnB on compare match (set output to low level)
1	1	Set OCnA/OCnB on compare match (set output to high level)

Table 16-3 shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to the fast PWM mode.

Table 16-3. Compare Output Mode, Fast PWM[1]

COMnA1/COMnB1	COMnA0/COMnB0	Description
0	0	Normal port operation, OCnA/OCnB disconnected.
0	1	WGMn3:0 = 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 OCnA/OCnB on compare match, set OCnA/OCnB at BOTTOM (non-inverting mode)
1	1	Set OCnA/OCnB on compare match, clear OCnA/OCnB at BOTTOM (inverting mode)

Note: 1. A special case occurs when OCRnA/OCRnB equals TOP and COMnA1/COMnB1 is set. In this case the compare match is ignored, but the set or clear is done at BOTTOM.
See Section 16.9.3 "Fast PWM Mode" on page 102 for more details.

Table 16-4 shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to the phase correct or the phase and frequency correct, PWM mode.

Table 16-4. Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM[1]

COMnA1/COMnB1	COMnA0/COMnB0	Description
0	0	Normal port operation, OCnA/OCnB disconnected.
0	1	WGMn3:0 = 9 or 11: Toggle OCnA on compare match, OCnB disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OCnA/OCnB on compare match when up-counting. Set OCnA/OCnB on compare match when downcounting.
1	1	Set OCnA/OCnB on compare match when up-counting. Clear OCnA/OCnB on compare match when downcounting.

Note: 1. A special case occurs when OCRnA/OCRnB equals TOP and COMnA1/COMnB1 is set.
See Section 16.9.4 "Phase Correct PWM Mode" on page 104 for more details.

- **Bit 1:0 – WGMn1:0: Waveform Generation Mode**

Combined with the WGMn3:2 bits found in the TCCRnB 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 16-5. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), clear timer on compare match (CTC) mode, and three types of pulse width modulation (PWM) modes. See (Section 16.9 "Modes of Operation" on page 101).

Table 16-5. Waveform Generation Mode Bit Description(1)

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnx at	TOVn 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	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, phase and frequency correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM, phase and frequency correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, phase correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, phase correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Note: 1. The CTCn and PWMn1:0 bit definition names are obsolete. Use the WGMn2:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

16.11.2 TCCRnB – Timer/Counter n Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	**ICNCn**	**ICESn**	**–**	**WGMn3**	**WGMn2**	**CSn2**	**CSn1**	**CSn0**	**TCCRnB**
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 – ICNCn: 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 (ICPn) is filtered. The filter function requires four successive equal valued samples of the ICPn pin for changing its output. The input capture is therefore delayed by four oscillator cycles when the noise canceler is enabled.

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

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

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

When the ICRn is used as TOP value (see description of the WGMn3:0 bits located in the TCCRnA and the TCCRnB register), the ICPn is disconnected and consequently the input capture function is disabled.

- **Bit 5 – Reserved**

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

- **Bit 4:3 – WGMn3:2: Waveform Generation Mode**

See description of Section 16.11.1 "TCCRnA – Timer/Counter n Control Register A" on page 109.

- **Bit 2:0 – CSn2:0: Clock Select**

The three clock select bits select the clock source to be used by the Timer/Counter, see Figure 16-10 on page 107 and Figure 16-11 on page 108.

Table 16-6. Clock Select Bit Description

CSn2	CSn1	CSn0	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 Tn pin. Clock on falling edge.
1	1	1	External clock source on Tn pin. Clock on rising edge.

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

16.11.3 TCCRnC – Timer/Counter n Control Register C

Bit	7	6	5	4	3	2	1	0	
(0x82)	**FOCnA**	**FOCnB**	–	–	–	–	–	–	**TCCRnC**
Read/Write	R/W	R/W	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

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

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

The FOCnA/FOCnB bits are only active when the WGMn3:0 bits specifies a non-PWM mode. However, for ensuring compatibility with future devices, these bits must be set to zero when TCCRnA is written when operating in a PWM mode. When writing a logical one to the FOCnA/FOCnB bit, an immediate compare match is forced on the waveform generation unit. The OCnA/OCnB output is changed according to its COMnx1:0 bits setting. Note that the FOCnA/FOCnB bits are implemented as strobes. Therefore it is the value present in the COMnx1:0 bits that determine the effect of the forced compare.

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

The FOCnA/FOCnB bits are always read as zero.

16.11.4 TCNT1H and TCNT1L –Timer/Counter1

Bit	7	6	5	4	3	2	1	0	
(0x85)	**TCNT1[15:8]**								**TCNT1H**
(0x84)	**TCNT1[7:0]**								**TCNT1L**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The two Timer/Counter I/O locations (TCNT1H and TCNT1L, combined TCNT1) give direct access, both for read and for write operations, to the Timer/Counter unit 16-bit counter. To ensure that both the high and low bytes are read and written simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers.
See Section 16.3 "Accessing 16-bit Registers" on page 92.

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

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

16.11.5 TCNT3H and TCNT3L –Timer/Counter3

Bit	7	6	5	4	3	2	1	0	
(0x95)	**TCNT3[15:8]**								**TCNT3H**
(0x94)	**TCNT3[7:0]**								**TCNT3L**
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 (TCNT3H and TCNT3L, combined TCNT3) give direct access, both for read and for write operations, to the Timer/Counter unit 16-bit counter. To ensure that both the high and low bytes are read and written simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers.
See Section 16.3 "Accessing 16-bit Registers" on page 92.

Modifying the counter (TCNT3) while the counter is running introduces a risk of missing a compare match between TCNT3 and one of the OCRnx registers.

Writing to the TCNT3 register blocks (removes) the compare match on the following timer clock for all compare units.

16.11.6 OCR1AH and OCR1AL – Output Compare Register1 A

Bit	7	6	5	4	3	2	1	0	
(0x89)	**OCR1A[15:8]**								**OCR1AH**
(0x88)	**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	

16.11.7 OCR1BH and OCR1BL – Output Compare Register1 B

Bit	7	6	5	4	3	2	1	0	
(0x8B)	**OCR1B[15:8]**								**OCR1BH**
(0x8A)	**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 OCnx pin.

The output compare registers are 16-bit in size. To ensure that both the high and low bytes are written simultaneously when the CPU writes to these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. See Section 16.3 "Accessing 16-bit Registers" on page 92.

16.11.8 OCR3AH and OCR3AL – Output Compare Register3 A

Bit	7	6	5	4	3	2	1	0	
(0x99)	**OCR3A[15:8]**								**OCR3AH**
(0x98)	**OCR3A[7:0]**								**OCR3AL**
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	

16.11.9 OCR3BH and OCR3BL – Output Compare Register3 B

Bit	7	6	5	4	3	2	1	0	
(0x9B)	**OCR3B[15:8]**								**OCR3BH**
(0x9A)	**OCR3B[7:0]**								**OCR3BL**
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 (TCNT3). A match can be used to generate an output compare interrupt, or to generate a waveform output on the OCnx pin.

The output compare registers are 16-bit in size. To ensure that both the high and low bytes are written simultaneously when the CPU writes to these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. See Section 16.3 "Accessing 16-bit Registers" on page 92.

16.11.10 ICR1H and ICR1L – Input Capture Register 1

Bit	7	6	5	4	3	2	1	0	
(0x87)	**ICR1[15:8]**								**ICR1H**
(0x86)	**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 ICPn pin (or optionally on the analog comparator output for Timer/Counter1). The input capture can be used for defining the counter TOP value.

The input capture register is 16-bit in size. To ensure that both the high and low bytes are read simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. See Section 16.3 "Accessing 16-bit Registers" on page 92.

16.11.11 ICR3H and ICR3L – Input Capture Register 3

Bit	7	6	5	4	3	2	1	0	
(0x97)	**ICR3[15:8]**								**ICR3H**
(0x96)	**ICR3[7:0]**								**ICR3L**
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 (TCNT3) value each time an event occurs on the ICPn pin (or optionally on the analog comparator output for Timer/Counter3). The input capture can be used for defining the counter TOP value.

The input capture register is 16-bit in size. To ensure that both the high and low bytes are read simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. See Section 16.3 “Accessing 16-bit Registers” on page 92.

16.11.12 TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	**ICIE1**	–	–	**OCIE1B**	**OCIE1A**	**TOIE1**	**TIMSK1**
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – Reserved**

These bits are unused and will always read as zero.

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

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 input capture interrupt is enabled. The corresponding interrupt vector (see Section 12. “Interrupts” on page 49) is executed when the ICF1 flag, located in TIFR1, is set.

- **Bit 4:3 – Reserved**

These bits are unused and will always read as zero.

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

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 output compare B match interrupt is enabled. The corresponding interrupt vector (see Section 12. “Interrupts” on page 49) is executed when the OCF1B flag, located in TIFR1, is set.

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

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 output compare A match interrupt is enabled. The corresponding interrupt vector (see Section 12. “Interrupts” on page 49) is executed when the OCF1A flag, located in TIFR1, is set.

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

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 overflow interrupt is enabled. The corresponding interrupt vector (see Section 11.3 “Watchdog Timer” on page 44) is executed when the TOV1 flag, located in TIFR1, is set.

16.11.13 TIMSK3 – Timer/Counter3 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x71)	–	–	ICIE3	–	–	OCIE3B	OCIE3A	TOIE3	**TIMSK3**
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – Reserved**

These bits are unused and will always read as zero.

- **Bit 5 – ICIE3: Timer/Counter3, Input Capture Interrupt Enable**

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 input capture interrupt is enabled. The corresponding interrupt vector (see Section 12. “Interrupts” on page 49) is executed when the ICF3 flag, located in TIFR3, is set.

- **Bit 4:3 – Reserved**

These bits are unused and will always read as zero.

- **Bit 2 – OCIE3B: Timer/Counter3, 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/Counter3 output compare B match interrupt is enabled. The corresponding interrupt vector (see Section 12. “Interrupts” on page 49) is executed when the OCF3B flag, located in TIFR3, is set.

- **Bit 1 – OCIE3A: Timer/Counter3, 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/Counter3 output compare A match interrupt is enabled. The corresponding interrupt vector (see Section 12. “Interrupts” on page 49) is executed when the OCF3A flag, located in TIFR3, is set.

- **Bit 0 – TOIE3: Timer/Counter3, 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/Counter3 overflow interrupt is enabled. The corresponding interrupt vector (see Section 11.3 “Watchdog Timer” on page 44) is executed when the TOV3 flag, located in TIFR3, is set.

16.11.14 TIFR1 – Timer/Counter1 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	**TIFR1**
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – Reserved**

These bits are unused and will always read as zero.

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

This flag is set when a capture event occurs on the ICP1 pin. When the input capture register (ICR1) is set by the WGMn3:0 to be used as the TOP value, the ICF1 flag is set when the counter reaches the TOP value.

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

- **Bit 4:3 – Reserved**

These bits are unused and will always read as zero.

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

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

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

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

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

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

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

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

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

The setting of this flag is dependent of the WGMn3:0 bits setting. In normal and CTC modes, the TOV1 flag is set when the timer overflows. Refer to Table 16-5 on page 111 for the TOV1 flag behavior when using another WGMn3:0 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.

16.11.15 TIFR3 – Timer/Counter3 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x18 (0x38)	**–**	**–**	**ICF3**	**–**	**–**	**OCF3B**	**OCF3A**	**TOV3**	**TIFR3**
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – Reserved**

These bits are unused and will always read as zero.

- **Bit 5 – ICF3: Timer/Counter3, Input Capture Flag**

This flag is set when a capture event occurs on the ICP3 pin. When the input capture register (ICR1) is set by the WGMn3:0 to be used as the TOP value, the ICF3 flag is set when the counter reaches the TOP value.

ICF3 is automatically cleared when the input capture interrupt vector is executed. Alternatively, ICF3 can be cleared by writing a logic one to its bit location.

- **Bit 4:3 – Reserved**

These bits are unused and will always read as zero.

- **Bit 2 – OCF3B: Timer/Counter3, Output Compare B Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT3) value matches the output compare register B (OCR3B).

Note that a forced output compare (FOC3B) strobe will not set the OCF3B flag.

OCF3B is automatically cleared when the output compare match B interrupt vector is executed. Alternatively, OCF3B can be cleared by writing a logic one to its bit location.

- **Bit 1 – OCF3A: Timer/Counter3, Output Compare A Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT3) value matches the output compare register A (OCR3A).

Note that a forced output compare (FOC3A) strobe will not set the OCF3A flag.

OCF3A is automatically cleared when the output compare match A interrupt vector is executed. Alternatively, OCF3A can be cleared by writing a logic one to its bit location.

- **Bit 0 – TOV3: Timer/Counter1, Overflow Flag**

The setting of this flag is dependent of the WGMn3:0 bits setting. In normal and CTC modes, the TOV3 flag is set when the timer overflows. Refer to Table 16-5 on page 111 for the TOV3 flag behavior when using another WGMn3:0 bit setting.

TOV3 is automatically cleared when the Timer/Counter3 overflow interrupt vector is executed. Alternatively, TOV3 can be cleared by writing a logic one to its bit location.

17. 8-bit Timer/Counter2 with PWM and Asynchronous Operation

17.1 Features

- Single channel 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, OCF2A and OCF2B)
- Allows clocking from external 32kHz watch crystal independent of the I/O clock

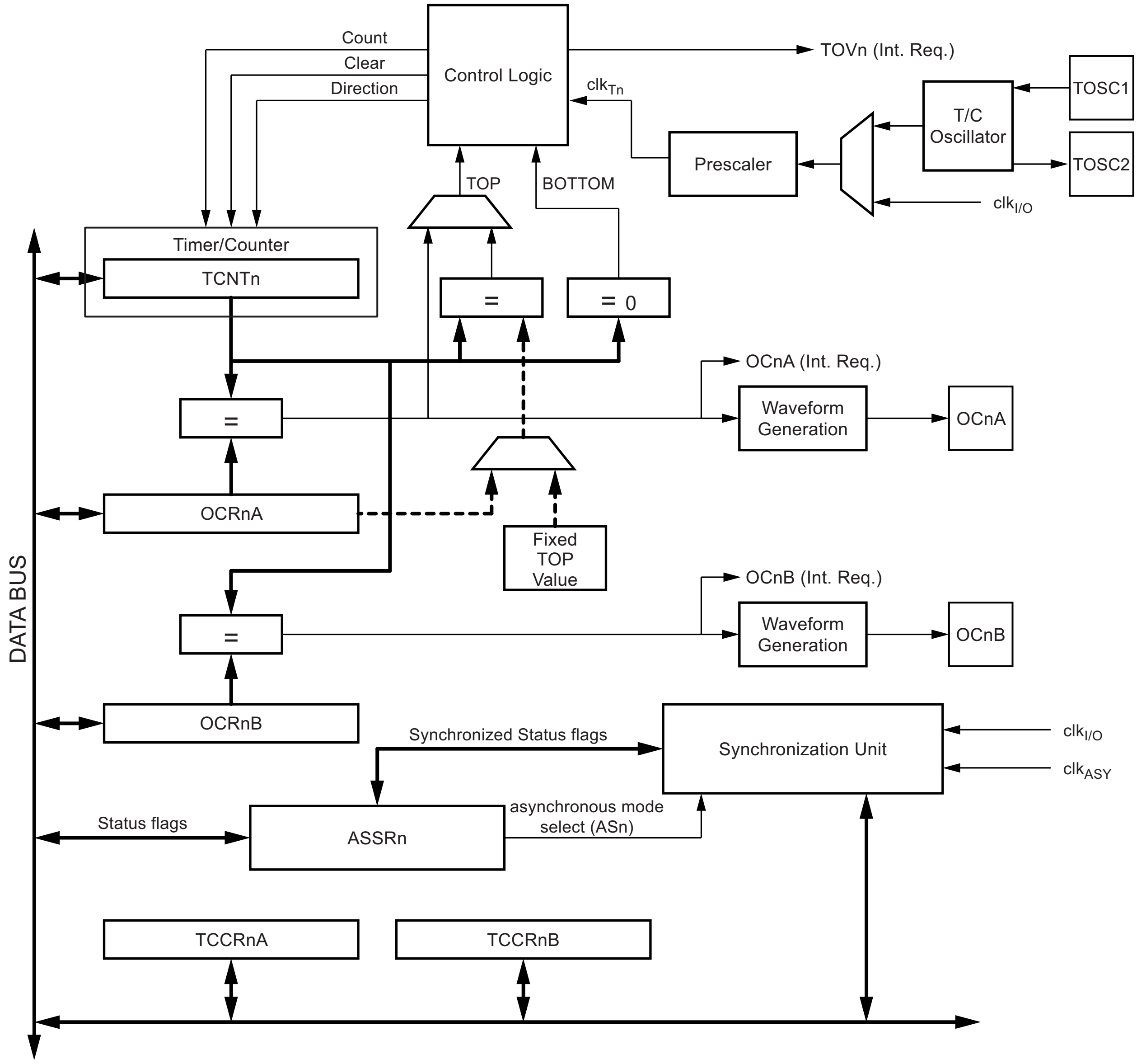
17.2 Overview

Timer/Counter2 is a general purpose, single channel, 8-bit Timer/Counter module.

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 16-12 on page 108. For the actual placement of I/O pins, see Section 1. “Pin Configurations” on page 3. CPU accessible I/O registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O register and bit locations are listed in the Section 17.11 “Register Description” on page 132.

The power reduction Timer/Counter2 bit, PRTIM2, in Section 10.12.3 “PRR0 – Power Reduction Register 0” on page 39 must be written to zero to enable Timer/Counter2 module.

Figure 17-1. 8-bit Timer/Counter Block Diagram



17.2.1 Registers

The Timer/Counter (TCNT2) and output compare register (OCR2A and OCR2B) are 8-bit registers. Interrupt request (abbreviated to 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 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_{T2}).

The double buffered output compare register (OCR2A and OCR2B) are 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 pins (OC2A and OC2B). See Section 17.5 “Output Compare Unit” on page 123 for details. The compare match event will also set the compare flag (OCF2A or OCF2B) which can be used to generate an output compare interrupt request.

17.2.2 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 17-1 are also used extensively throughout the section.

Table 17-1. Definitions

Parameter	Definition
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.

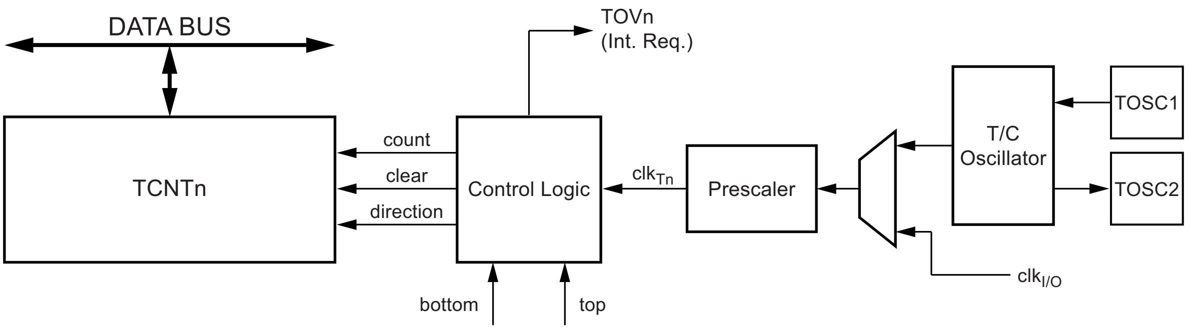
17.3 Timer/Counter Clock Sources

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 Section 17.11.6 “ASSR – Asynchronous Status Register” on page 136. For details on clock sources and prescaler, see Section 17.10 “Timer/Counter Prescaler” on page 131.

17.4 Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Figure 17-2 shows a block diagram of the counter and its surrounding environment.

Figure 17-2. 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_{Tn}	Timer/Counter clock, referred to as clk_{T2} in the following.
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) and the WGM22 located in the Timer/Counter control register B (TCCR2B). There are close connections between how the counter behaves (counts) and how waveforms are generated on the output compare outputs OC2A and OC2B. For more details about advanced counting sequences and waveform generation, see Section 17.7 “Modes of Operation” on page 125.

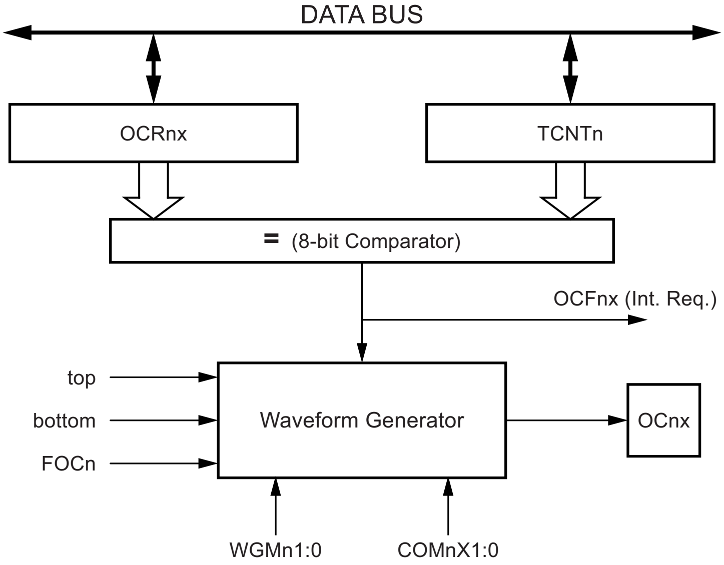
The Timer/Counter overflow flag (TOV2) is set according to the mode of operation selected by the WGM22:0 bits. TOV2 can be used for generating a CPU interrupt.

17.5 Output Compare Unit

The 8-bit comparator continuously compares TCNT2 with the output compare register (OCR2A and OCR2B). Whenever TCNT2 equals OCR2A or OCR2B, the comparator signals a match. A match will set the output compare flag (OCF2A or OCF2B) at the next timer clock cycle. If the corresponding interrupt is enabled, the output compare flag generates an output compare interrupt. The output compare flag is automatically cleared when the interrupt is executed. Alternatively, the output compare 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 WGM22:0 bits and compare output mode (COM2x1:0) 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 Section 17.7 “Modes of Operation” on page 125).

Figure 16-10 on page 107 shows a block diagram of the output compare unit.

Figure 17-3. Output Compare Unit, Block Diagram



The OCR2x 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 OCR2x 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 OCR2x register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR2x buffer register, and if double buffering is disabled the CPU will access the OCR2x directly.

17.5.1 Force Output Compare

In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a one to the force output compare (FOC2x) bit. Forcing compare match will not set the OCF2x flag or reload/clear the timer, but the OC2x pin will be updated as if a real compare match had occurred (the COM2x1:0 bits settings define whether the OC2x pin is set, cleared or toggled).

17.5.2 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 OCR2x to be initialized to the same value as TCNT2 without triggering an interrupt when the Timer/Counter clock is enabled.

17.5.3 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 channel, independently of whether the Timer/Counter is running or not. If the value written to TCNT2 equals the OCR2x 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 downcounting.

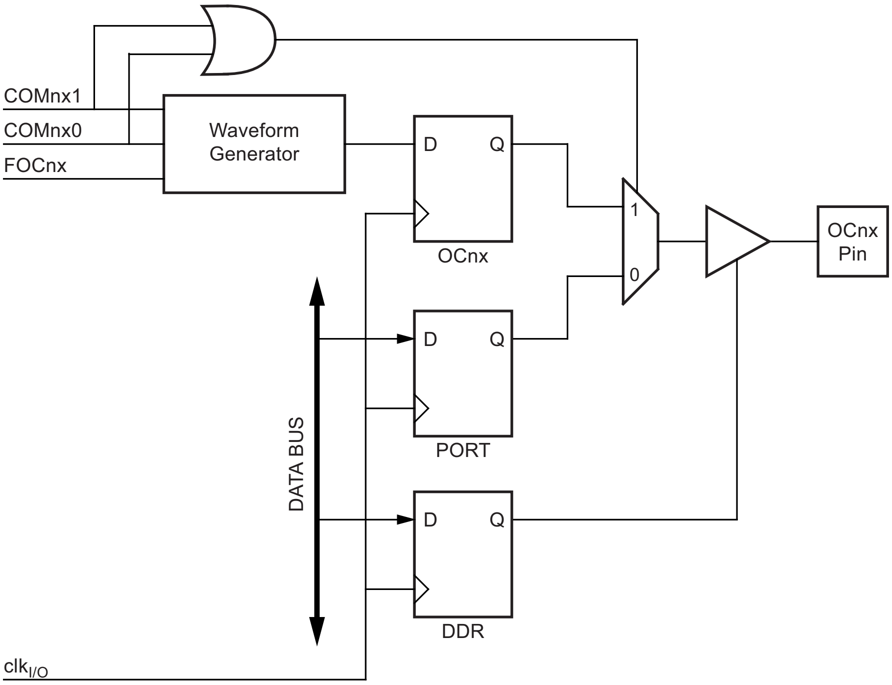
The setup of the OC2x should be performed before setting the data direction register for the port pin to output. The easiest way of setting the OC2x value is to use the force output compare (FOC2x) strobe bit in normal mode. The OC2x register keeps its value even when changing between waveform generation modes.

Be aware that the COM2x1:0 bits are not double buffered together with the compare value. Changing the COM2x1:0 bits will take effect immediately.

17.6 Compare Match Output Unit

The compare output mode (COM2x1:0) bits have two functions. The waveform generator uses the COM2x1:0 bits for defining the output compare (OC2x) state at the next compare match. Also, the COM2x1:0 bits control the OC2x pin output source. Figure 17-4 shows a simplified schematic of the logic affected by the COM2x1:0 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 COM2x1:0 bits are shown. When referring to the OC2x state, the reference is for the internal OC2x register, not the OC2x pin.

Figure 17-4. Compare Match Output Unit, Schematic



The general I/O port function is overridden by the output compare (OC2x) from the waveform generator if either of the COM2x1:0 bits are set. However, the OC2x 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 OC2x pin (DDR_OC2x) must be set as output before the OC2x 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 OC2x state before the output is enabled. Note that some COM2x1:0 bit settings are reserved for certain modes of operation.
See Section 17.11 “Register Description” on page 132.

17.6.1 Compare Output Mode and Waveform Generation

The waveform generator uses the COM2x1:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COM2x1:0 = 0 tells the waveform generator that no action on the OC2x register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 17-5 on page 133. For fast PWM mode, refer to Table 17-6 on page 133, and for phase correct PWM refer to Table 17-7 on page 134.

A change of the COM2x1:0 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 FOC2x strobe bits.

17.7 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 (WGM22:0) and compare output mode (COM2x1:0) bits. The compare output mode bits do not affect the counting sequence, while the waveform generation mode bits do. The COM2x1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM2x1:0 bits control whether the output should be set, cleared, or toggled at a compare match (see Section 17.6 "Compare Match Output Unit" on page 124).

For detailed timing information refer to Section 17.8 "Timer/Counter Timing Diagrams" on page 129.

17.7.1 Normal Mode

The simplest mode of operation is the normal mode (WGM22:0 = 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.

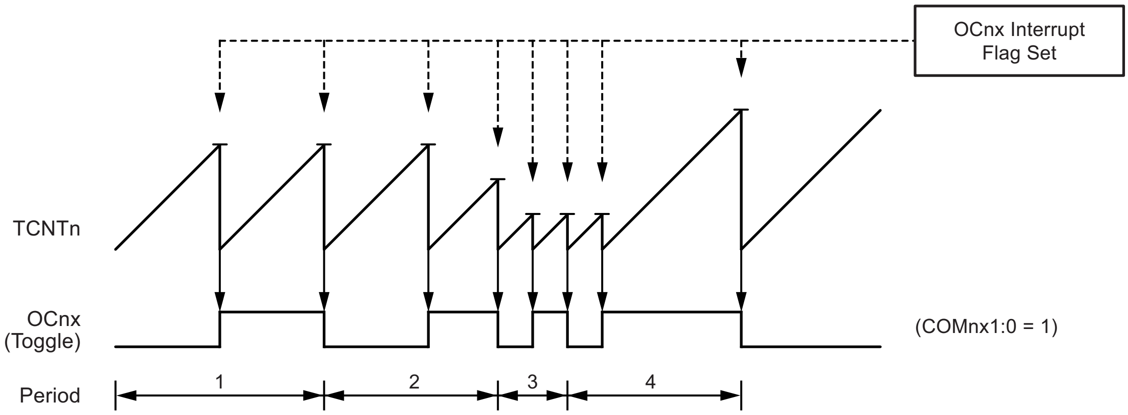
The output compare unit can be used to generate interrupts at some given time. Using the output compare to generate waveforms in normal mode is not recommended, since this will occupy too much of the CPU time.

17.7.2 Clear Timer on Compare Match (CTC) Mode

In clear timer on compare or CTC mode (WGM22:0 = 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 Table 17-5 on page 125. The counter value (TCNT2) increases until a compare match occurs between TCNT2 and OCR2A, and then counter (TCNT2) is cleared.

Figure 17-5. 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 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).

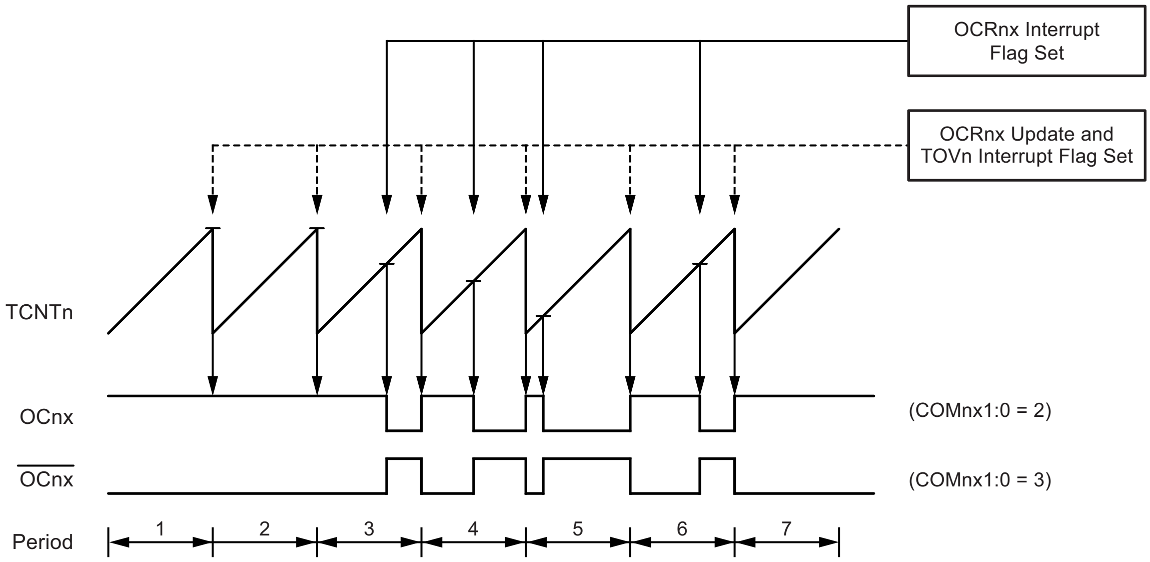
As for the normal mode of operation, the TOV2 flag is set in the same timer clock cycle that the counter counts from MAX to 0x00.

17.7.3 Fast PWM Mode

The fast pulse width modulation or fast PWM mode (WGM22:0 = 3 or 7) 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 TOP then restarts from BOTTOM. TOP is defined as 0xFF when WGM22:0 = 3, and OCR2A when MGM22:0 = 7. In non-inverting compare output mode, the output compare (OC2x) is cleared on the compare match between TCNT2 and OCR2x, 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 TOP value. The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 17-6. 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 OCR2x and TCNT2.

Figure 17-6. Fast PWM Mode, Timing Diagram



The Timer/Counter overflow flag (TOV2) is set each time the counter reaches TOP. 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 OC2x pin. Setting the COM2x1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM2x1:0 to three. TOP is defined as 0xFF when WGM2:0 = 3, and OCR2A when WGM2:0 = 7 (See Table 17-3 on page 132). The actual OC2x 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 OC2x register at the compare match between OCR2x and TCNT2, and clearing (or setting) the OC2x 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:

$$f_{OCnxPWM} = \frac{f_{\text{clk\_I/O}}}{N \cdot 256}$$

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.)

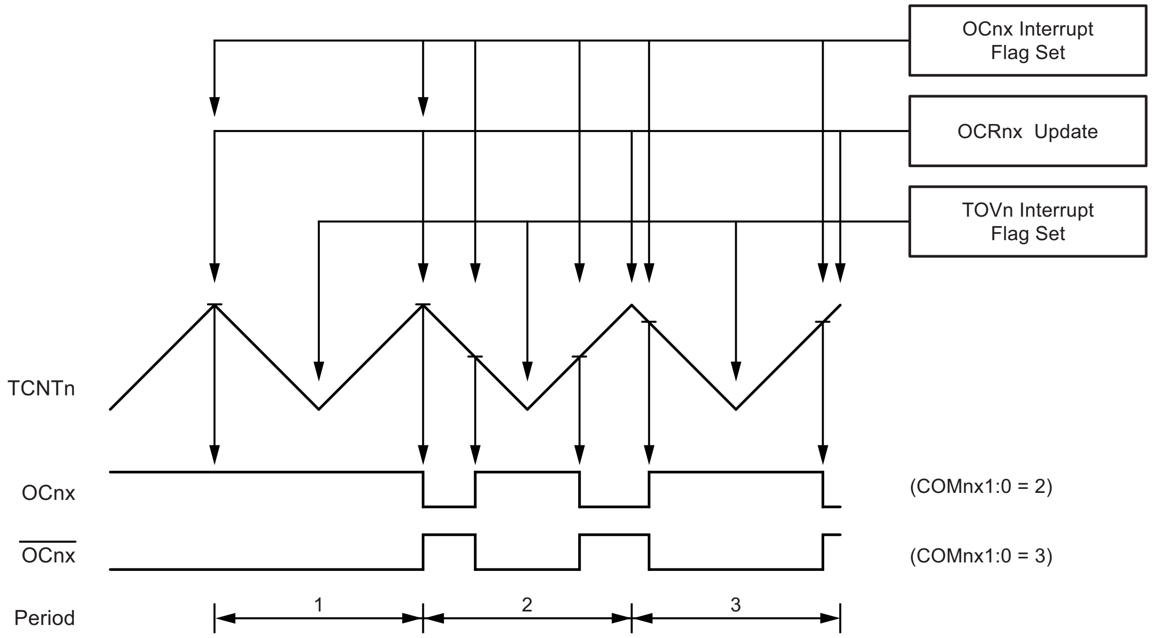
A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OC2x to toggle its logical level on each compare match (COM2x1:0 = 1). The waveform generated will have a maximum frequency of $f_{oc2} = f_{clk\_I/O}/2$ when OCR2A is set to zero. This feature is similar to the OC2A toggle in CTC mode, except the double buffer feature of the output compare unit is enabled in the fast PWM mode.

17.7.4 Phase Correct PWM Mode

The phase correct PWM mode (WGM22:0 = 1 or 5) 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 TOP and then from TOP to BOTTOM. TOP is defined as 0xFF when WGM22:0 = 1, and OCR2A when MGM22:0 = 5. In non-inverting compare output mode, the output compare (OC2x) is cleared on the compare match between TCNT2 and OCR2x while upcounting, and set on the compare match while downcounting. In inverting output compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

In phase correct PWM mode the counter is incremented until the counter value matches TOP. When the counter reaches TOP, it changes the count direction. The TCNT2 value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on Figure 17-7 on page 128. 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 OCR2x and TCNT2.

Figure 17-7. 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 OC2x pin. Setting the COM2x1:0 bits to two will produce a non-inverted PWM. An inverted PWM output can be generated by setting the COM2x1:0 to three. TOP is defined as 0xFF when WGM2:0 = 3, and OCR2A when MGM2:0 = 7 (see Table 17-4 on page 133). The actual OC2x 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 OC2x register at the compare match between OCR2x and TCNT2 when the counter increments, and setting (or clearing) the OC2x register at compare match between OCR2x and TCNT2 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCnxPCPWM} = \frac{f_{\text{clk\_I/O}}}{N \cdot 510}$$

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 17-7 OCnx 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 17-7. 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.

17.8 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 17-8 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 17-8. Timer/Counter Timing Diagram, no Prescaling

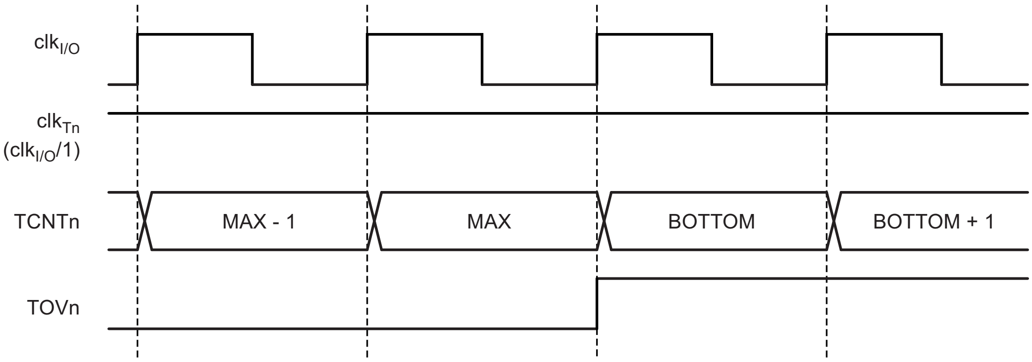


Figure 17-9 shows the same timing data, but with the prescaler enabled.

Figure 17-9. Timer/Counter Timing Diagram, with Prescaler ($f_{clk\_I/O}$/8)

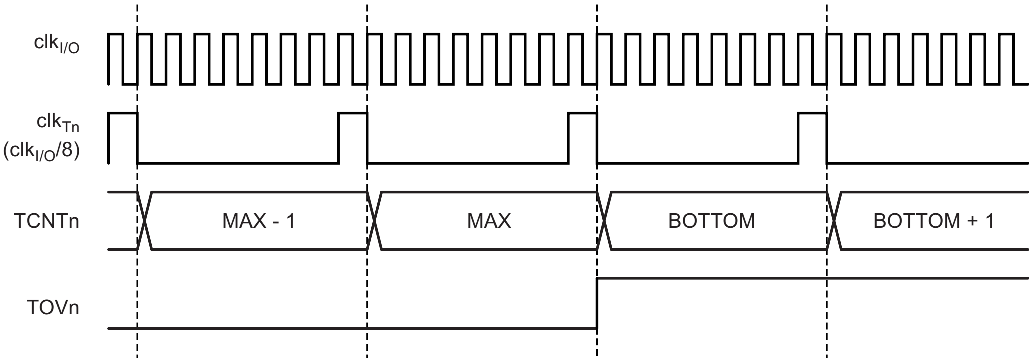


Figure 17-10 shows the setting of OCF2A in all modes except CTC mode.

Figure 17-10. Timer/Counter Timing Diagram, Setting of OCF2A, with Prescaler ($f_{clk\_I/O}$/8)

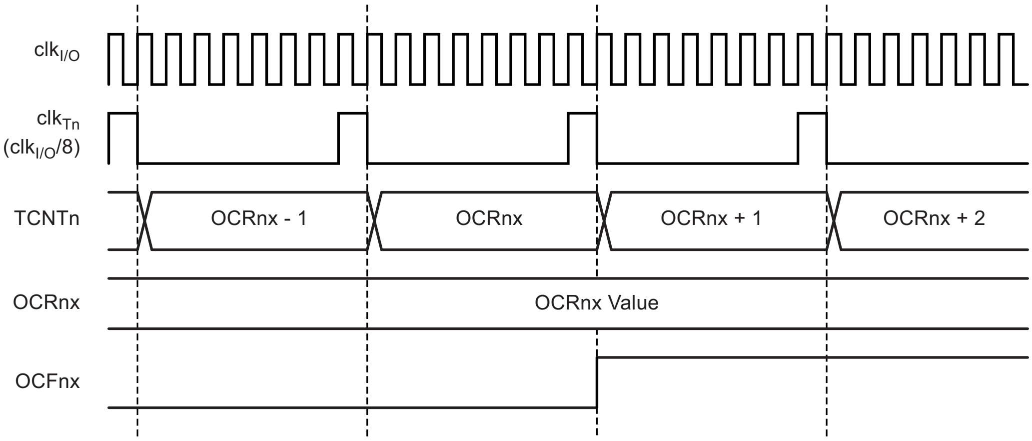
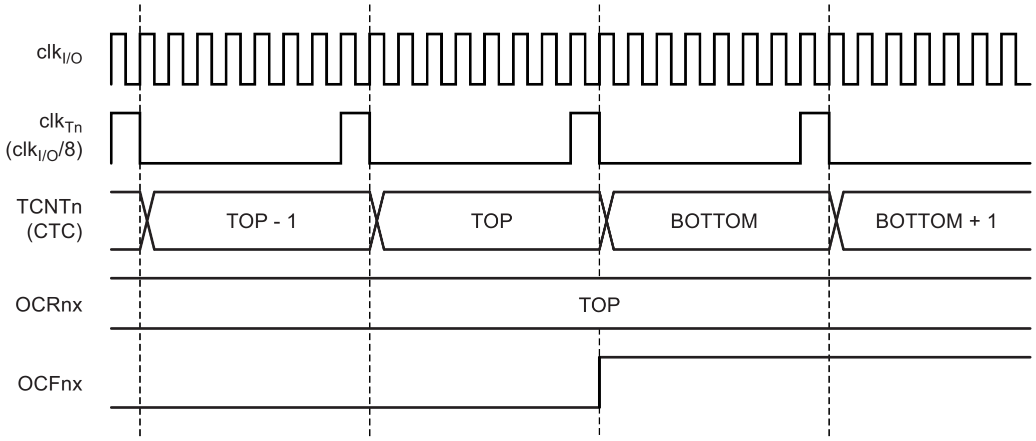


Figure 17-11 shows the setting of OCF2A and the clearing of TCNT2 in CTC mode.

Figure 17-11. Timer/Counter Timing Diagram, Clear Timer on Compare Match mode, with Prescaler ($f_{clk\_I/O}/8$)



17.9 Asynchronous Operation of Timer/Counter2

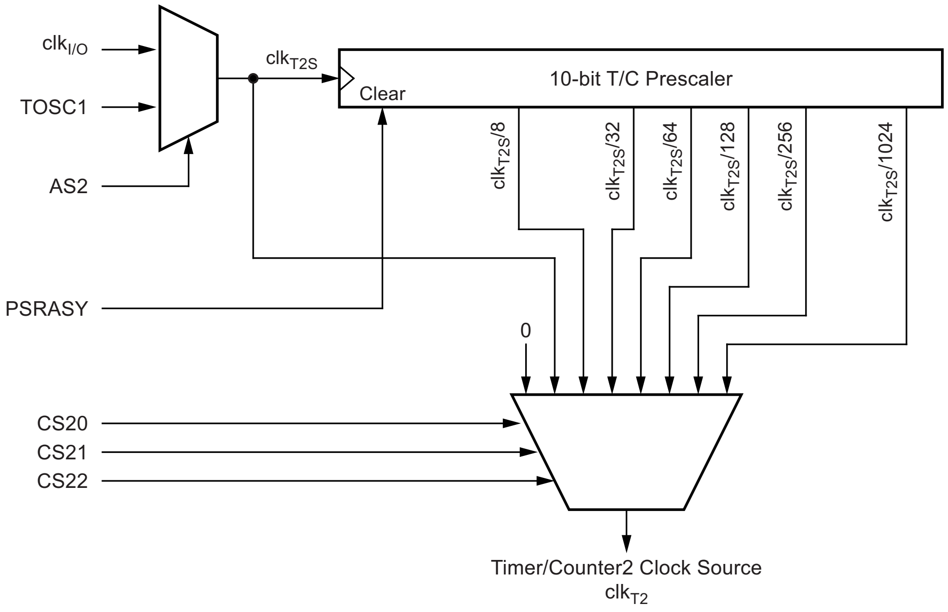
When Timer/Counter2 operates asynchronously, some considerations must be taken.

- Warning: When switching between asynchronous and synchronous clocking of Timer/Counter2, the timer registers TCNT2, OCR2x, and TCCR2x might be corrupted. A safe procedure for switching clock source is:
  1. Disable the Timer/Counter2 interrupts by clearing OCIE2x and TOIE2.
  2. Select clock source by setting AS2 as appropriate.
  3. Write new values to TCNT2, OCR2x, and TCCR2x.
  4. To switch to asynchronous operation: Wait for TCN2UB, OCR2xUB, and TCR2xUB.
  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, OCR2x, or TCCR2x, 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 five mentioned registers have their individual temporary register, which means that e.g. writing to TCNT2 does not disturb an OCR2x 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, OCR2x, or TCCR2x, 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 any of the output compare2 interrupt is used to wake up the device, since the output compare function is disabled during writing to OCR2x or TCNT2. If the write cycle is not finished, and the MCU enters sleep mode before the corresponding OCR2xUB 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 TCCR2x, TCNT2, or OCR2x.
  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 32.768kHz 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 OCR2x or TCCR2x.
    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.

17.10 Timer/Counter Prescaler

Figure 17-12.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$_{IO}$. 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 32.768kHz crystal. By setting the EXCLK bit in the ASSR a 32kHz external clock can be applied. See Section 17.11.6 “ASSR – Asynchronous Status Register” on page 136 for details.

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 PSRASY bit in GTCCR resets the prescaler. This allows the user to operate with a predictable prescaler.

17.11 Register Description

17.11.1 TCCR2A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
(0xB0)	**COM2A1**	**COM2A0**	**COM2B1**	**COM2B0**	**–**	**–**	**WGM21**	**WGM20**	**TCCR2A**
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	

- **Bits 7:6 – COM2A1:0: Compare Match Output A Mode**

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 the 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 WGM22:0 bit setting. Table 17-2 shows the COM2A1:0 bit functionality when the WGM22:0 bits are set to a normal or CTC mode (non-PWM).

Table 17-2. Compare Output Mode, non-PWM Mode

COM2A1	COM2A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC2A on compare match
1	0	Clear OC2A on compare match
1	1	Set OC2A on compare match

Table 17-3 shows the COM2A1:0 bit functionality when the WGM21:0 bits are set to fast PWM mode.

Table 17-3. Compare Output Mode, Fast PWM Mode(1)

COM2A1	COM2A0	Description
0	0	Normal port operation, OC2A disconnected.
0	1	WGM22 = 0: Normal port operation, OC0A disconnected. WGM22 = 1: Toggle OC2A on compare match.
1	0	Clear OC2A on compare match, set OC2A at BOTTOM, (non-inverting mode).
1	1	Set OC2A on compare match, clear OC2A at BOTTOM, (inverting mode).

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 BOTTOM. See Section 17.7.3 “Fast PWM Mode” on page 126 for more details.

Table 17-4 shows the COM2A1:0 bit functionality when the WGM22:0 bits are set to phase correct PWM mode.

Table 17-4. Compare Output Mode, Phase Correct PWM Mode(1)

COM2A1	COM2A0	Description
0	0	Normal port operation, OC2A disconnected.
0	1	WGM22 = 0: Normal port operation, OC2A disconnected. WGM22 = 1: Toggle OC2A on compare match.
1	0	Clear OC2A on compare match when up-counting. Set OC2A on compare match when down-counting.
1	1	Set OC2A on compare match when up-counting. Clear OC2A on compare match when down-counting.

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 Section 17.7.4 “Phase Correct PWM Mode” on page 127 for more details.

- **Bits 5:4 – COM2B1:0: Compare Match Output B Mode**

These bits control the output compare pin (OC2B) behavior. If one or both of the COM2B1:0 bits are set, the OC2B 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 OC2B pin must be set in order to enable the output driver.

When OC2B is connected to the pin, the function of the COM2B1:0 bits depends on the WGM22:0 bit setting. Table 17-5 shows the COM2B1:0 bit functionality when the WGM22:0 bits are set to a normal or CTC mode (non-PWM).

Table 17-5. Compare Output Mode, non-PWM Mode

COM2B1	COM2B0	Description
0	0	Normal port operation, OC2B disconnected.
0	1	Toggle OC2B on compare match
1	0	Clear OC2B on compare match
1	1	Set OC2B on compare match

Table 17-6 shows the COM2B1:0 bit functionality when the WGM22:0 bits are set to fast PWM mode.

Table 17-6. Compare Output Mode, Fast PWM Mode(1)

COM2B1	COM2B0	Description
0	0	Normal port operation, OC2B disconnected.
0	1	Reserved
1	0	Clear OC2B on compare match, set OC2B at BOTTOM, (non-inverting mode).
1	1	Set OC2B on compare match, clear OC2B at BOTTOM, (inverting mode).

Note: 1. A special case occurs when OCR2B equals TOP and COM2B1 is set. In this case, the compare match is ignored, but the set or clear is done at BOTTOM. See Section 17.7.3 “Fast PWM Mode” on page 126 for more details.

Table 17-7 shows the COM2B1:0 bit functionality when the WGM22:0 bits are set to phase correct PWM mode.

Table 17-7. Compare Output Mode, Phase Correct PWM Mode[(1)]

COM2B1	COM2B0	Description
0	0	Normal port operation, OC2B disconnected.
0	1	Reserved
1	0	Clear OC2B on compare match when up-counting. Set OC2B on compare match when down-counting.
1	1	Set OC2B on compare match when up-counting. Clear OC2B on compare match when down-counting.

Note: 1. A special case occurs when OCR2B equals TOP and COM2B1 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See Section 17.7.4 "Phase Correct PWM Mode" on page 127 for more details.

- **Bits 3:2 – Reserved**

These bits are reserved and will always read as zero.

- **Bits 1:0 – WGM21:0: Waveform Generation Mode**

Combined with the WGM22 bit found in the TCCR2B 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 17-8. Modes of operation supported by the Timer/Counter unit are: normal mode (counter), clear timer on compare match (CTC) mode, and two types of pulse width modulation (PWM) modes (see Section 17.7 "Modes of Operation" on page 125).

Table 17-8. Waveform Generation Mode Bit Description

Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on[(1)(2)]
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, phase correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, phase correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Notes: 1. MAX = 0xFF

2. BOTTOM = 0x00

17.11.2 TCCR2B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
(0xB1)	**FOC2A**	**FOC2B**	**–**	**–**	**WGM22**	**CS22**	**CS21**	**CS20**	**TCCR2B**
Read/Write	W	W	R	R	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 TCCR2B 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 – FOC2B: Force Output Compare B**

The FOC2B 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 TCCR2B is written when operating in PWM mode. When writing a logical one to the FOC2B bit, an immediate compare match is forced on the waveform generation unit. The OC2B output is changed according to its COM2B1:0 bits setting. Note that the FOC2B bit is implemented as a strobe. Therefore it is the value present in the COM2B1:0 bits that determines the effect of the forced compare.

A FOC2B strobe will not generate any interrupt, nor will it clear the timer in CTC mode using OCR2B as TOP.

The FOC2B bit is always read as zero.

- **Bits 5:4 – Reserved**

These bits are reserved and will always read as zero.

- **Bit 3 – WGM22: Waveform Generation Mode**

See the description in Section 17.11.1 “TCCR2A – Timer/Counter Control Register A” on page 132.

- **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 17-9.

Table 17-9. 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 prescalerf
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)

17.11.3 TCNT2 – Timer/Counter Register

Bit	7	6	5	4	3	2	1	0	
(0xB2)	**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 OCR2x registers.

17.11.4 OCR2A – Output Compare Register A

Bit	7	6	5	4	3	2	1	0	
(0xB3)	**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.

17.11.5 OCR2B – Output Compare Register B

Bit	7	6	5	4	3	2	1	0	
(0xB4)	**OCR2B[7:0]**								**OCR2B**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The output compare register B contains an 8-bit value that is continuously compared with the counter value (TCNT2). A match can be used to generate an output compare interrupt, or to generate a waveform output on the OC2B pin.

17.11.6 ASSR – Asynchronous Status Register

Bit	7	6	5	4	3	2	1	0	
(0xB6)	–	**EXCLK**	**AS2**	**TCN2UB**	**OCR2AUB**	**OCR2BUB**	**TCR2AUB**	**TCR2BUB**	**ASSR**
Read/Write	R	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 6 – 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 32kHz 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 5 – 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, OCR2B, TCCR2A and TCCR2B might be corrupted.

- **Bit 4 – 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 3 – OCR2AUB: 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 2 – OCR2BUB: Output Compare Register2 Update Busy**

When Timer/Counter2 operates asynchronously and OCR2B is written, this bit becomes set. When OCR2B has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that OCR2B is ready to be updated with a new value.

- **Bit 1 – TCR2AUB: 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.

- **Bit 0 – TCR2BUB: Timer/Counter Control Register2 Update Busy**

When Timer/Counter2 operates asynchronously and TCCR2B is written, this bit becomes set. When TCCR2B has been updated from the temporary storage register, this bit is cleared by hardware. A logical zero in this bit indicates that TCCR2B is ready to be updated with a new value.

If a write is performed to any of the five 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, OCR2B, TCCR2A and TCCR2B are different. When reading TCNT2, the actual timer value is read. When reading OCR2A, OCR2B, TCCR2A and TCCR2B the value in the temporary storage register is read.

17.11.7 TIMSK2 – Timer/Counter2 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x70)	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2	**TIMSK2**
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 2 – OCIE2B: Timer/Counter2 Output Compare Match B Interrupt Enable**

When the OCIE2B bit is written to one and the I-bit in the status register is set (one), the Timer/Counter2 compare match B interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter2 occurs, i.e., when the OCF2B bit is set in the Timer/Counter 2 interrupt flag register – TIFR2.

- **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.

17.11.8 TIFR2 – Timer/Counter2 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x17 (0x37)	–	–	–	–	–	**OCF2B**	**OCF2A**	**TOV2**	**TIFR2**
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 2 – OCF2B: Output Compare Flag 2 B**

The OCF2B bit is set (one) when a compare match occurs between the Timer/Counter2 and the data in OCR2B – output compare register2. OCF2B is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF2B is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE2B (Timer/Counter2 compare match interrupt enable), and OCF2B are set (one), the Timer/Counter2 compare match interrupt is executed.

- **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.

17.11.9 GTCCR – General Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0	
0x23 (0x43)	**TSM**	–	–	–	–	–	**PSRASY**	**PSRSYNC**	**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 PSRASY and PSRSYNC 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 PSRASY and PSRSYNC bits are cleared by hardware, and the Timer/Counters start counting simultaneously.

- **Bit 1 – PSRASY: 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 this page for a description of the Timer/Counter synchronization mode.

- **Bit 0 – PSRSYNC: Prescaler Reset**

When this bit is one, Timer/Counter1 and Timer/Counter0 prescaler will be reset. This bit is normally cleared immediately by hardware, except ifthe 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.

18. SPI – Serial Peripheral Interface

18.1 Features

- Full-duplex, three-wire synchronous data transfer
- Master or slave 0peration
- 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

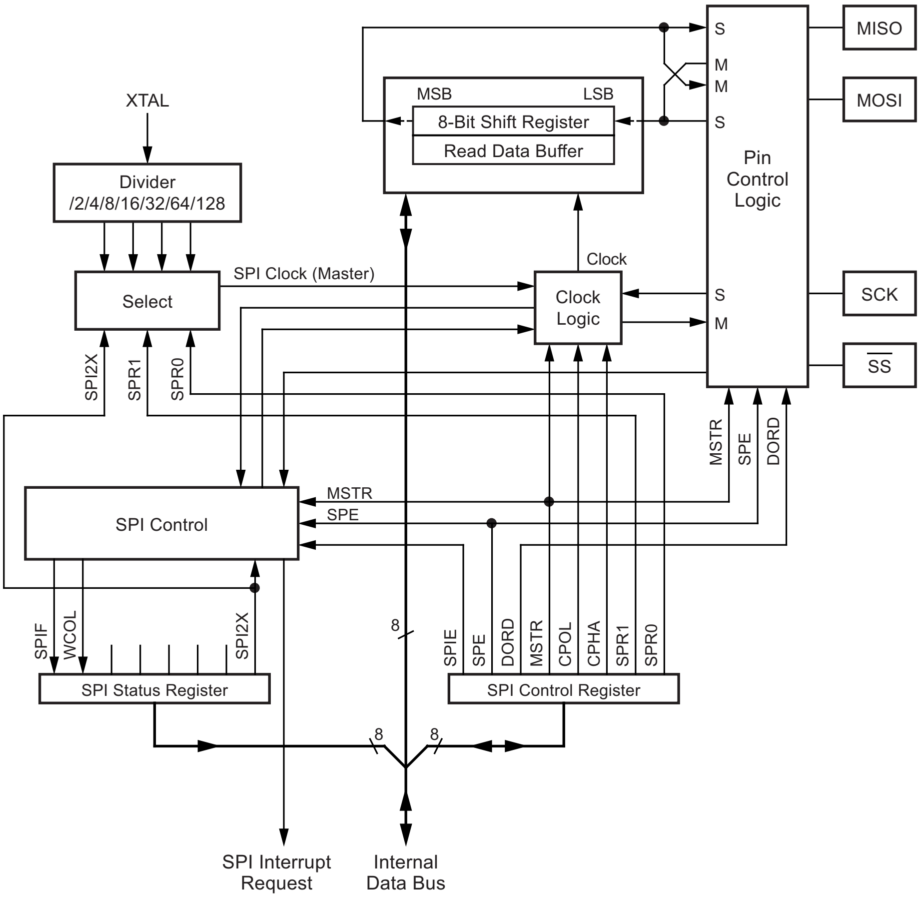
18.2 Overview

The serial peripheral interface (SPI) allows high-speed synchronous data transfer between the Atmel® ATmega164P-B/324P-B/644P-B and peripheral devices or between several AVR® devices.

USART can also be used in master SPI mode, see Section 20. “USART in SPI Mode” on page 169.

The power reduction SPI bit, PRSPI, in Section 10.12.3 “PRR0 – Power Reduction Register 0” on page 39 must be written to zero to enable SPI module.

Figure 18-1. SPI Block Diagram[1]



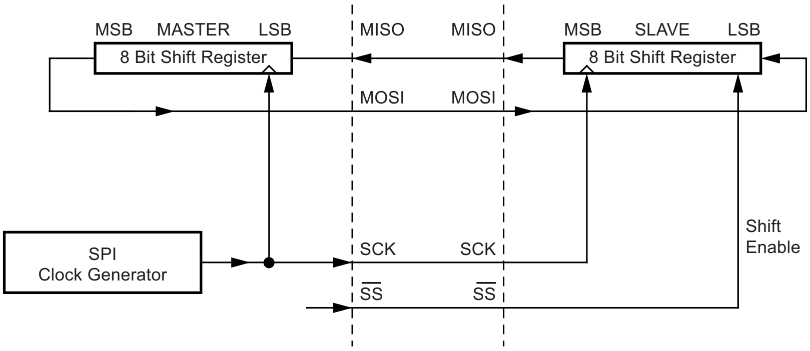
Note: 1. Refer to Figure 1-1 on page 3, and Table 14-6 on page 65 for SPI pin placement.

The interconnection between master and slave CPUs with SPI is shown in Figure 18-2. The system consists of two shift registers, and a master clock generator. The SPI master initiates the communication cycle when pulling low the slave select $\overline{\text{SS}}$ 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, $\overline{\text{SS}}$, line.

When configured as a master, the SPI interface has no automatic control of the $\overline{\text{SS}}$ 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, $\overline{\text{SS}}$ 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 $\overline{\text{SS}}$ 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 $\overline{\text{SS}}$ 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 18-2. SPI Master-slave Interconnection



The system is single buffered in the transmit direction and double buffered in the receive direction. This means that bytes to be transmitted cannot be written to the SPI data register before the entire shift cycle is completed. When receiving data, however, a received character must be read from the SPI data register before the next character has been completely shifted in. Otherwise, the first byte is lost.

In SPI slave mode, the control logic will sample the incoming signal of the SCK pin. To ensure correct sampling of the clock signal, the minimum low and high periods should be:

Low period: longer than 2 CPU clock cycles.

High period: longer than 2 CPU clock cycles.

When the SPI is enabled, the data direction of the MOSI, MISO, SCK, and $\overline{\text{SS}}$ pins is overridden according to Table 18-1. For more details on automatic port overrides, refer to Section 14.3 "Alternate Port Functions" on page 62.

Table 18-1. SPI Pin Overrides[1]

Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User defined	Input
MISO	Input	User defined
SCK	User defined	Input
$\overline{\text{SS}}$	User defined	Input

Note: 1. See Section 14.3.2 "Alternate Functions of Port B" on page 65 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 Section 4. “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 Section 4. "About Code Examples" on page 8.

18.3 $\overline{\text{SS}}$ Pin Functionality

18.3.1 Slave Mode

When the SPI is configured as a slave, the slave select ($\overline{\text{SS}}$) pin is always input. When $\overline{\text{SS}}$ is held low, the SPI is activated, and MISO becomes an output if configured so by the user. All other pins are inputs. When $\overline{\text{SS}}$ 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 $\overline{\text{SS}}$ pin is driven high.

The $\overline{\text{SS}}$ pin is useful for packet/byte synchronization to keep the slave bit counter synchronous with the master clock generator. When the $\overline{\text{SS}}$ 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.

18.3.2 Master Mode

When the SPI is configured as a master (MSTR in SPCR is set), the user can determine the direction of the $\overline{SS}$ pin.

If $\overline{SS}$ is configured as an output, the pin is a general output pin which does not affect the SPI system. Typically, the pin will be driving the $\overline{SS}$ pin of the SPI slave.

If $\overline{SS}$ is configured as an input, it must be held high to ensure master SPI operation. If the $\overline{SS}$ pin is driven low by peripheral circuitry when the SPI is configured as a master with the $\overline{SS}$ pin defined as an input, the SPI system interprets this as another master selecting the SPI as a slave and starting to send data to it. To avoid bus contention, the SPI system takes the following actions:

1. The MSTR bit in SPCR is cleared and the SPI system becomes a slave. As a result of the SPI becoming a slave, the MOSI and SCK pins become inputs.
2. The SPIF flag in SPSR is set, and if the SPI interrupt is enabled, and the I-bit in SREG is set, the interrupt routine will be executed.

Thus, when interrupt-driven SPI transmission is used in master mode, and there exists a possibility that $\overline{SS}$ is driven low, the interrupt should always check that the MSTR bit is still set. If the MSTR bit has been cleared by a slave select, it must be set by the user to re-enable SPI master mode.

18.4 Data Modes

There are four combinations of SCK phase and polarity with respect to serial data, which are determined by control bits CPHA and CPOL. The SPI data transfer formats are shown in Figure 18-3 and Figure 18-4 on page 144. 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 18-3 and Table 18-4 on page 145, as done in Table 18-2.

Table 18-2. SPI Modes

SPI Mode	Conditions	Leading Edge	Trailing Edge
0	CPOL=0, CPHA=0	Sample (rising)	Setup (falling)
1	CPOL=0, CPHA=1	Setup (rising)	Sample (falling)
2	CPOL=1, CPHA=0	Sample (falling)	Setup (rising)
3	CPOL=1, CPHA=1	Setup (falling)	Sample (rising)

Figure 18-3. SPI Transfer Format with CPHA = 0

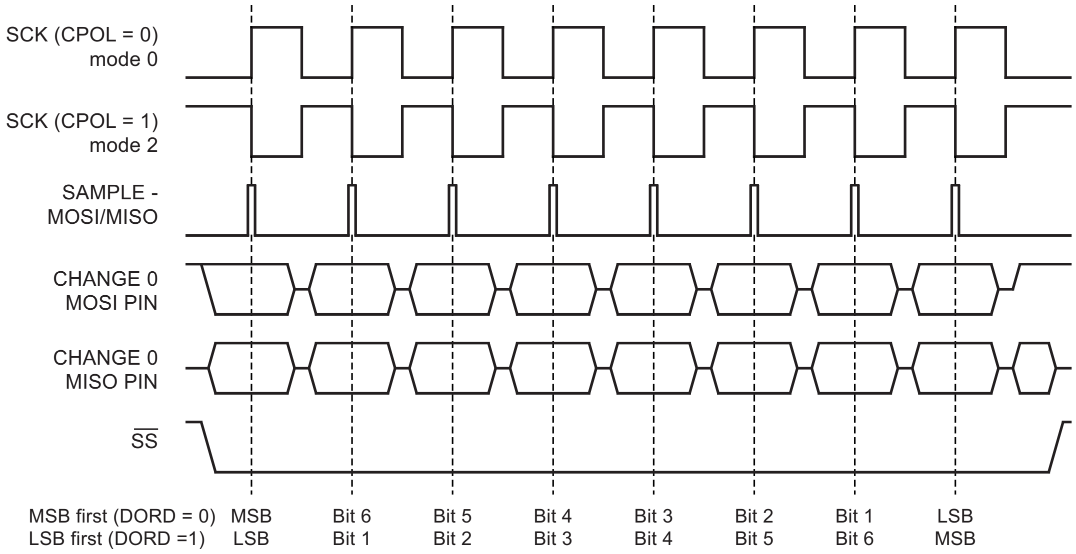
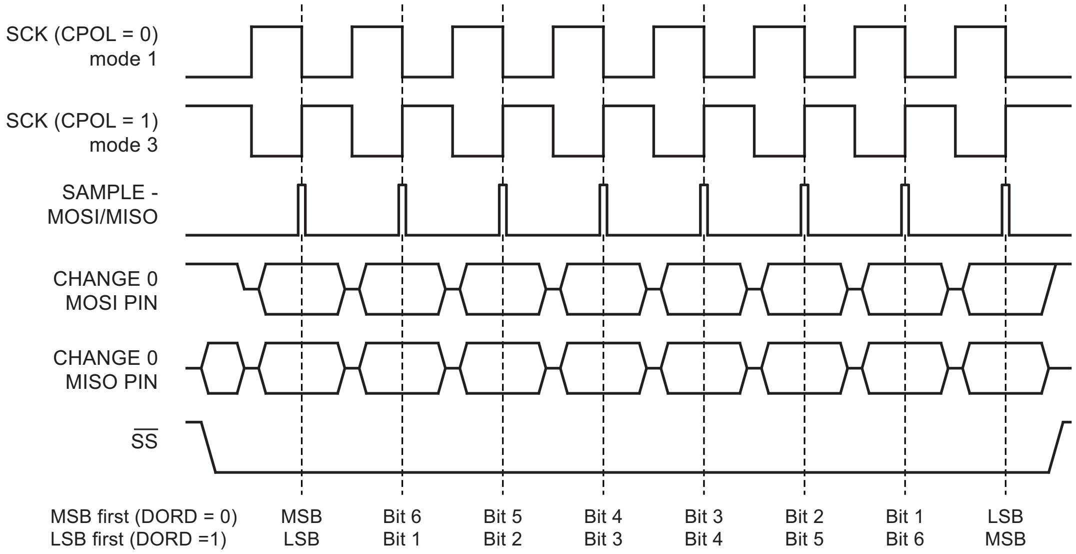


Figure 18-4. SPI Transfer Format with CPHA = 1



18.5 Register Description

18.5.1 SPCR – SPI Control Register

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	**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 18-3 on page 143 and Figure 18-4 on page 144 for an example. The CPOL functionality is summarized below:

Table 18-3. CPOL Functionality

CPOL	Leading Edge	Trailing Edge
0	Rising	Falling
1	Falling	Rising

- **Bit 2 – CPHA: Clock Phase**

The settings of the clock phase bit (CPHA) determine if data is sampled on the leading (first) or trailing (last) edge of SCK. Refer to Figure 18-3 on page 143 and Figure 18-4 on page 144 for an example. The CPOL functionality is summarized below:

Table 18-4. CPHA Functionality

CPHA	Leading Edge	Trailing Edge
0	Sample	Setup
1	Setup	Sample

- **Bits 1:0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0**

These two bits control the SCK rate of the device configured as a Master. SPR1 and SPR0 have no effect on the slave. The relationship between SCK and the oscillator clock frequency f_{osc} is shown in the following table:

Table 18-5. Relationship Between SCK and the Oscillator Frequency

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

18.5.2 SPSR – SPI Status Register

Bit	7	6	5	4	3	2	1	0	
0x2D (0x4D)	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 – Reserved**

These bits are reserved 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 18-5 on page 145). 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 ATmega164P-B/324P-B/644P-B is also used for program memory and EEPROM downloading or uploading. See Section 27.8 “Serial Downloading” on page 270 for serial programming and verification.

18.5.3 SPDR – SPI Data Register

Bit	7	6	5	4	3	2	1	0	
0x2E (0x4E)	**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.

19. USART

19.1 Features

- 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

19.2 USART1 and USART0

The ATmega164P-B/324P-B/644P-B has two USART's, USART0 and USART1.

The functionality for all USART's is described below, most register and bit references in this section are written in general form. A lower case "n" replaces the USART number.

USART0 and USART1 have different I/O registers as shown in Section 30. "Register Summary" on page 323.

19.3 Overview

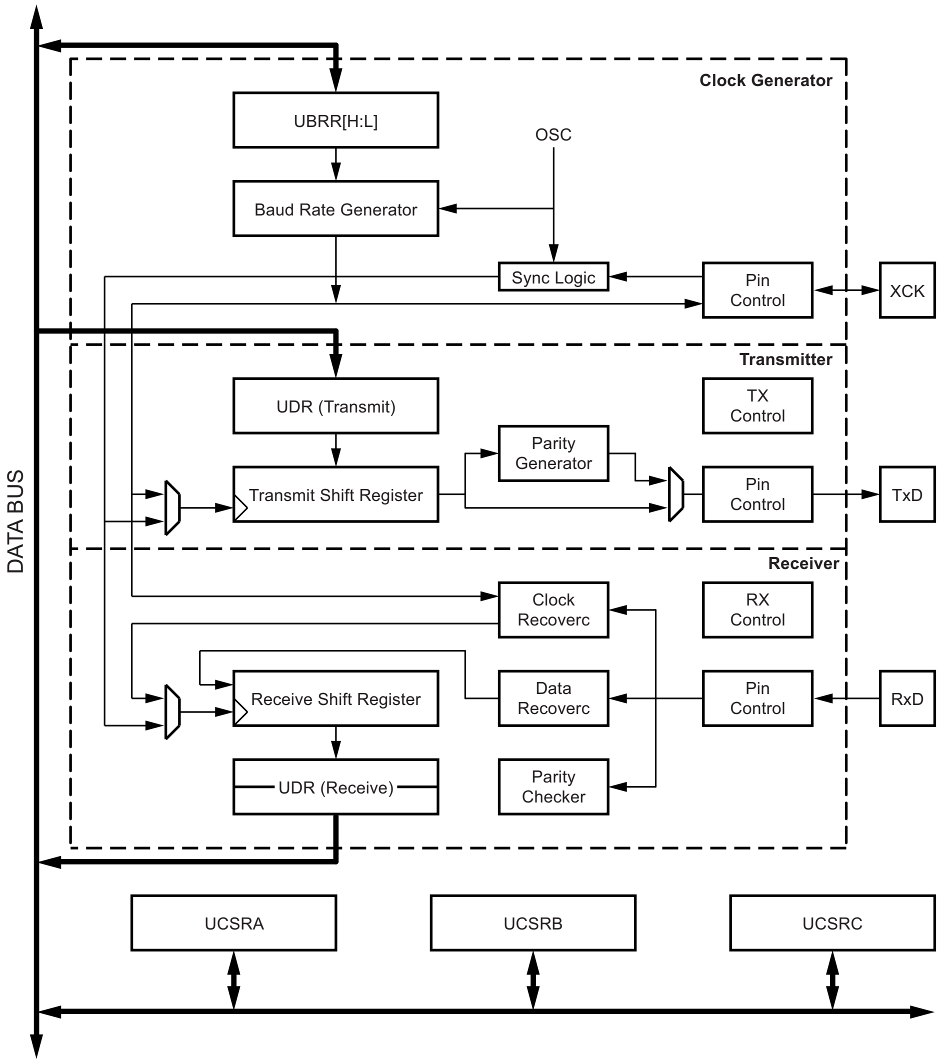
The universal synchronous and asynchronous serial receiver and transmitter (USART) is a highly flexible serial communication device.

A simplified block diagram of the USART transmitter is shown in Figure 19-1 on page 148. CPU accessible I/O registers and I/O pins are shown in bold.

The power reducion USART0 bit, PRUSART0, in Section 10.12.3 "PRR0 – Power Reduction Register 0" on page 39 must be disabled by writing a logical zero to it.

The power reducion USART1 bit, PRUSART1, in Section 10.12.4 "PRR1 – Power Reduction Register 1" on page 39 must be disabled by writing a logical zero to it.

Figure 19-1. USART Block Diagram[1]



Note: 1. See Figure 1-1 on page 3 and Section 14.3 “Alternate Port Functions” on page 62 for USART pin placement.

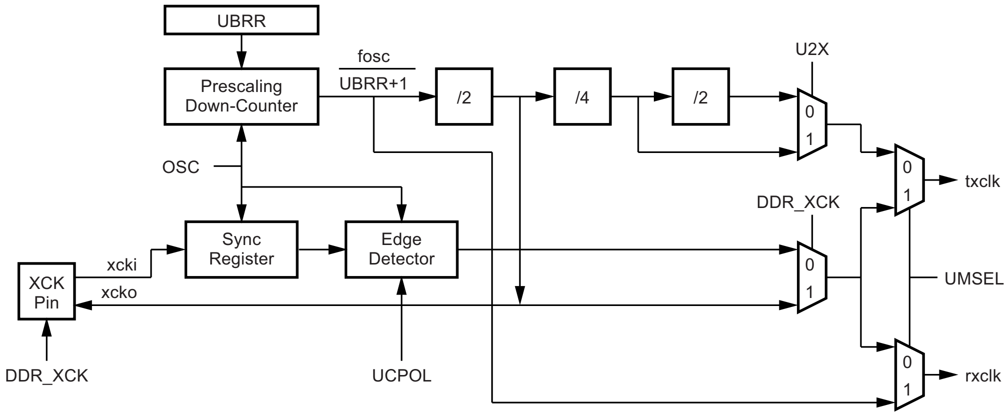
The dashed boxes in the block diagram separate the three main parts of the USART (listed from the top): clock generator, transmitter and receiver. Control registers are shared by all units. The clock generation logic consists of synchronization logic for external clock input used by synchronous slave operation, and the baud rate generator. The XCKn (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.

19.4 Clock Generation

The clock generation logic generates the base clock for the transmitter and receiver. The USARTn 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 XCKn pin (DDR_XCKn) controls whether the clock source is internal (master mode) or external (slave mode). The XCKn pin is only active when using synchronous mode.

Figure 19-2 shows a block diagram of the clock generation logic.

Figure 19-2. 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.
f_{OSC}	XTAL pin frequency (system clock).

19.4.1 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 19-2.

The USART baud rate register (UBRRn) 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 UBRRn value each time the counter has counted down to zero or when the UBRRnL register is written. A clock is generated each time the counter reaches zero. This clock is the baud rate generator clock output (= f_{osc}/(UBRRn+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_XCKn bits.

Table 19-1 contains equations for calculating the baud rate (in bits per second) and for calculating the UBRRn value for each mode of operation using an internally generated clock source.

Table 19-1. 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)	$\mathrm{BAUD} = \frac{f_{OSC}}{16 \times (\mathrm{UBRRn} + 1)}$	$\mathrm{UBRRn} = \frac{f_{OSC}}{16\mathrm{BAUD}} - 1$
Asynchronous double speed mode (U2Xn = 1)	$\mathrm{BAUD} = \frac{f_{OSC}}{8 \times (\mathrm{UBRRn} + 1)}$	$\mathrm{UBRRn} = \frac{f_{OSC}}{8\mathrm{BAUD}} - 1$
Synchronous master mode	$\mathrm{BAUD} = \frac{f_{OSC}}{2 \times (\mathrm{UBRRn} + 1)}$	$\mathrm{UBRRn} = \frac{f_{OSC}}{2\mathrm{BAUD}} - 1$

Notes:
1. The baud rate is defined to be the transfer rate in bit per second (bps)
2. **BAUD** - Baud rate (in bits per second, bps)
3. **f_{OSC}** - System oscillator clock frequency
4. **UBRRn** - Contents of the UBRRnH and UBRRnL registers, (0-4095)

Some examples of UBRRn values for some system clock frequencies are found in Table 19-9 on page 167.

19.4.2 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.

19.4.3 External Clock

External clocking is used by the synchronous slave modes of operation. The description in this section refers to Figure 19-2 on page 149 for details.

External clock input from the XCKn 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 XCKn clock frequency is limited by the following equation:

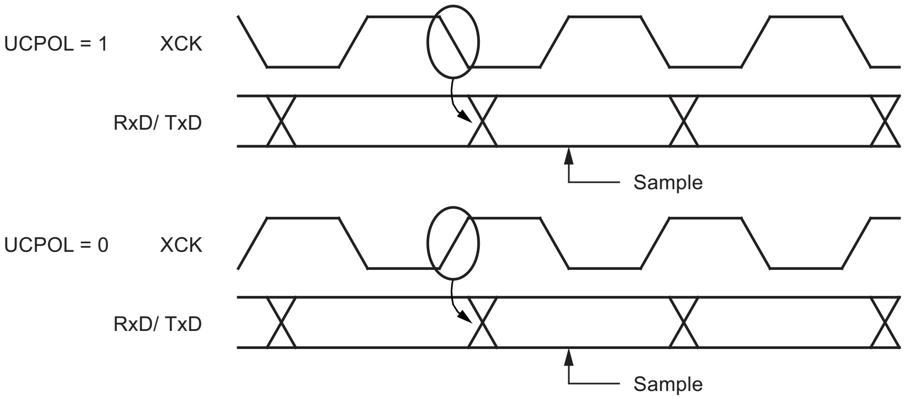
$$f_{XCK} < \frac{f_{OSC}}{4}$$

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.

19.4.4 Synchronous Clock Operation

When synchronous mode is used (UMSELn = 1), the XCKn 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 RxDn) is sampled at the opposite XCKn clock edge of the edge the data output (TxDn) is changed.

Figure 19-3. Synchronous Mode XCKn Timing



The UCPOLn bit UCRSC selects which XCKn clock edge is used for data sampling and which is used for data change. As Figure 19-3 shows, when UCPOLn is zero the data will be changed at rising XCKn edge and sampled at falling XCKn edge. If UCPOLn is set, the data will be changed at falling XCKn edge and sampled at rising XCKn edge.

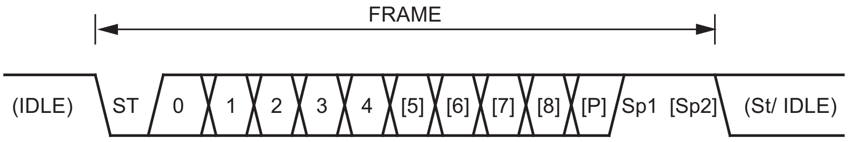
19.5 Frame Formats

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 19-4 illustrates the possible combinations of the frame formats. Bits inside brackets are optional.

Figure 19-4. 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 (RxDn or TxDn). 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 FE (frame error) will therefore only be detected in the cases where the first stop bit is zero.

19.5.1 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:

$$
\begin{aligned}
P_{even} &= d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0 \\
P_{odd} &= d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1
\end{aligned}
$$

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.

19.6 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 the USART interrupts should be disabled.

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 RXC 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    UBRRnH, r17
    out    UBRRnL, r16
    ; Enable receiver and transmitter
    ldi    r16, (1<<RXENn)|(1<<TXENn)
    out    UCSRnB,r16
    ; Set frame format: 8data, 2stop bit
    ldi    r16, (1<<USBSn)|(3<<UCSZn0)
    out    UCSRnC,r16
    ret
```

C Code Example[1]

```
void USART_Init( unsigned int baud )
{
    /* Set baud rate */
    UBRRnH = (unsigned char)(baud>>8);
    UBRRnL = (unsigned char)baud;
    /* Enable receiver and transmitter */
    UCSRnB = (1<<RXENn)|(1<<TXENn);
    /* Set frame format: 8data, 2stop bit */
    UCSRnC = (1<<USBSn)|(3<<UCSZn0);
}
```

Note: 1. See Section 4. “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.

19.7 Data Transmission – The USART Transmitter

The USART transmitter is enabled by setting the transmit enable (TXEN) bit in the UCSRnB register. When the transmitter is enabled, the normal port operation of the TxDn 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 XCKn pin will be overridden and used as transmission clock.

19.7.1 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 XCKn 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   UCSRnA,UDREn
        rjmp   USART_Transmit
        ; Put data (r16) into buffer, sends the data
        out    UDRn,r16
        ret
```

C Code Example[1]

```
void USART_Transmit(unsigned char data)
{
        /* Wait for empty transmit buffer */
        while (!(UCSRnA & (1<<UDREn)))
        ;
        /* Put data into buffer, sends the data */
        UDRn = data;
}
```

Note: 1. See Section 4. "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.

19.7.2 Sending Frames with 9 Data Bit

If 9-bit characters are used (UCSZn = 7), the ninth bit must be written to the TXB8 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    UCSRnA,UDREn
        rjmp    USART_Transmit
        ; Copy 9th bit from r17 to TXB8
        cbi     UCSRnB,TXB8
        sbrc    r17,0
        sbi     UCSRnB,TXB8
        ; Put LSB data (r16) into buffer, sends the data
        out     UDRn,r16
        ret
```

C Code Example(1)(2)

```
void USART_Transmit(unsigned int data)
{
        /* Wait for empty transmit buffer */
        while (!(UCSRnA & (1<<UDREn))))
                            ;
        /* Copy 9th bit to TXB8 */
        UCSRnB &= ~(1<<TXB8);
        if (data & 0x0100)
                UCSRnB |= (1<<TXB8);
        /* Put data into buffer, sends the data */
        UDRn = 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 TXB8 bit of the UCSRnB register is used after initialization.

2. See Section 4. “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.

19.7.3 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 complete 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.

19.7.4 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.

19.7.5 Disabling the Transmitter

The disabling of the transmitter (setting the TXEN 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 TxDn pin.

19.8 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 RxDn 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 XCKn pin will be used as transfer clock.

19.8.1 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 XCKn 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   UCSRnA, RXCn
        rjmp   USART_Receive
        ; Get and return received data from buffer
        in     r16, UDRn
        ret
```

C Code Example[1]

```
unsigned char USART_Receive(void)
{
        /* Wait for data to be received */
        while (!(UCSRnA & (1<<RXCn)))
        ;
        /* Get and return received data from buffer */
        return UDRn;
}
```

Note: 1. See Section 4. “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.

19.8.2 Receiving Frames with 9 Data Bits

If 9-bit characters are used (UCSZn=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    UCSRnA, RXCn
                rjmp    USART_Receive
                ; Get status and 9th bit, then data from buffer
                in      r18, UCSRnA
                in      r17, UCSRnB
                in      r16, UDRn
                ; If error, return -1
                andi    r18,(1<<FEn)|(1<<DORn)|(1<<UPEn)
                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 (!(UCSRnA & (1<<RXCn)))
                                    ;
                /* Get status and 9th bit, then data */
                /* from buffer */
                status = UCSRnA;
                resh = UCSRnB;
                resl = UDRn;
                /* If error, return -1 */
                if (status & (1<<FEn)|(1<<DORn)|(1<<UPEn))
                        return -1;
                /* Filter the 9th bit, then return */
                resh = (resh >> 1) & 0x01;
                return ((resh << 8) | resl);
        }
```

Note: 1. See Section 4. “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.

19.8.3 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.

19.8.4 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 Section 19.5.1 “Parity Bit Calculation” on page 152 and Section 19.8.5 “Parity Checker” on page 158.

19.8.5 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.

19.8.6 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 RxDn port pin. The receiver buffer FIFO will be flushed when the receiver is disabled. Remaining data in the buffer will be lost

19.8.7 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   UCSRnA, RXCn
              ret
              in     r16, UDRn
              rjmp   USART_Flush
```

C Code Example[1]

```
       void USART_Flush( void )
       {
              unsigned char dummy;
              while (UCSRnA & (1<<RXCn)) dummy = UDRn;
       }
```

Note: 1. See Section 4. "About Code Examples" on page 8.

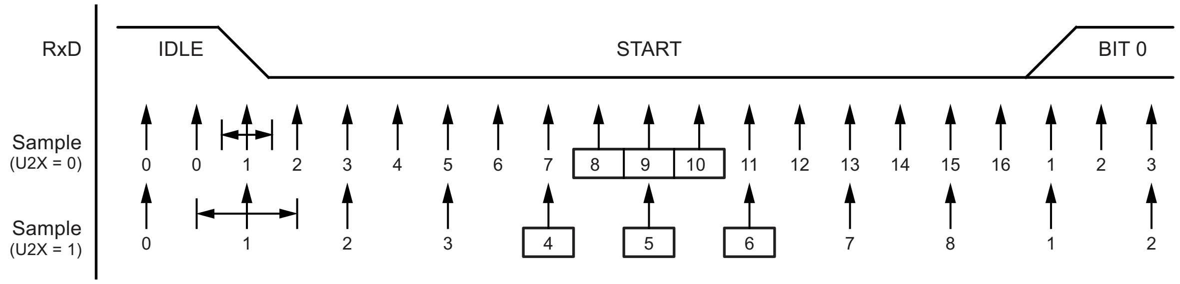
19.9 Asynchronous Data Reception

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 RxDn 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.

19.9.1 Asynchronous Clock Recovery

The clock recovery logic synchronizes internal clock to the incoming serial frames. Figure 19-5 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 RxDn line is idle (i.e., no communication activity).

Figure 19-5. Start Bit Sampling

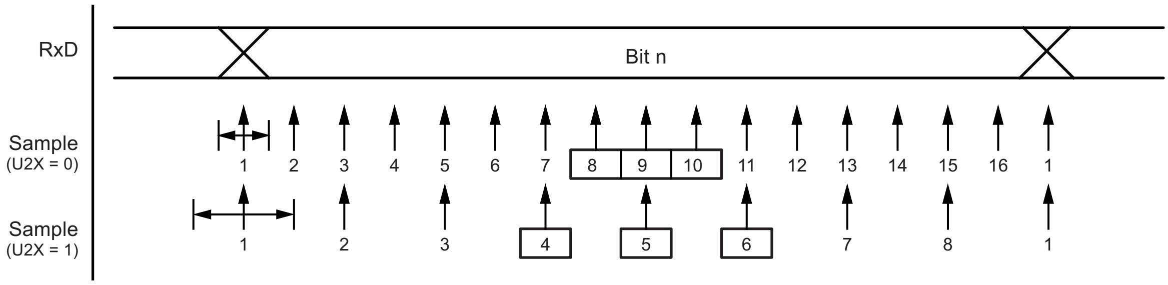


When the clock recovery logic detects a high (idle) to low (start) transition on the RxDn line, the start bit detection sequence is initiated. Let sample 1 denote the first zero-sample as shown in the figure. The clock recovery logic then uses samples 8, 9, and 10 for normal mode, and samples 4, 5, and 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.

19.9.2 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 19-6 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.

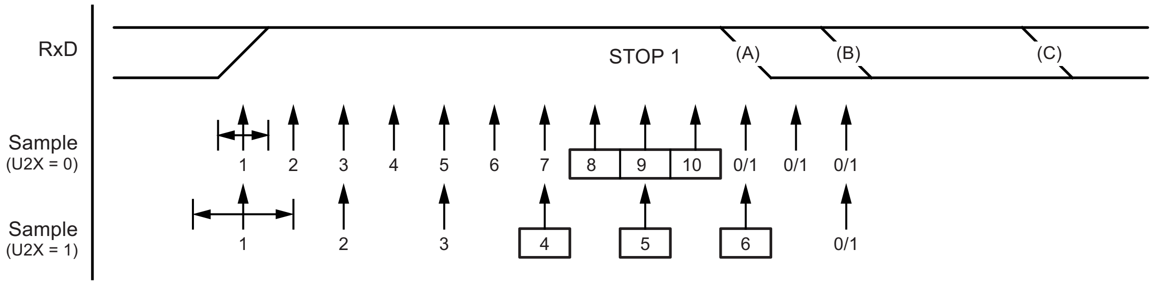
Figure 19-6. Sampling of Data and Parity Bit



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 RxDn 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 19-7 shows the sampling of the stop bit and the earliest possible beginning of the start bit of the next frame.

Figure 19-7. 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 19-7. 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.

19.9.3 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 19-2 on page 161) 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.

$$R_{slow} = \frac{(D+1) \times S}{S - 1 + D \times S + S_F} \qquad R_{fast} = \frac{(D+2) \times S}{(D+1) \times S + S_M}$$

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 19-2 and Table 19-3 list the maximum receiver baud rate error that can be tolerated. Note that normal speed mode has higher toleration of baud rate variations.

Table 19-2. 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 19-3. 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.

19.10 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.

19.10.1 Using MPCMn

For an MCU to act as a master MCU, it can use a 9-bit character frame format (UCSZn = 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.

19.11 Register Description

19.11.1 UDRn – USART I/O Data Register n

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 TxDn 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.

19.11.2 UCSRnA – USART Control and Status Register A

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 Section 19.10 "Multi-processor Communication Mode" on page 162.

19.11.3 UCSRnB – USART Control and Status Register n B

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 n**

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 n**

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 n**

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 n**

Writing this bit to one enables the USART receiver. The receiver will override normal port operation for the RxDn pin when enabled. Disabling the receiver will flush the receive buffer invalidating the FEn, DORn, and UPEn flags.

- **Bit 3 – TXENn: Transmitter Enable n**

Writing this bit to one enables the USART transmitter. The transmitter will override normal port operation for the TxDn 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 TxDn port.

- **Bit 2 – UCSZn2: Character Size n**

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 n**

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 n**

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.

19.11.4 UCSRnC – USART Control and Status Register n C

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
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	1	1	0	

- **Bits 7:6 – UMSELn1:0 USART Mode Select**

These bits select the mode of operation of the USARTn as shown in Table 19-4.

Table 19-4. UMSELn Bits Settings

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)(1)

Note: 1. See Section 20. “USART in SPI Mode” on page 169 for full description of the master SPI mode (MSPIM) operation.

- **Bits 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 UPMn setting. If a mismatch is detected, the UPEn flag in UCSRnA will be set.

Table 19-5. UPMn 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 19-6. 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 19-7. UCSZn 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 (XCKn).

Table 19-8. UCPOLn Bit Settings

UCPOLn	Transmitted Data Changed (Output of TxDn Pin)	Received Data Sampled (Input on RxDn Pin)
0	Rising XCKn edge	Falling XCKn edge
1	Falling XCKn edge	Rising XCKn edge

19.11.5 UBRRnL and UBRRnH – USART Baud Rate Registers

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRR[11:8]				**UBRRnH**
	UBRR[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**

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.

19.12 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 19-9 on page 167 to Table 19-12 on page 168. 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 Section 19.9.3 “Asynchronous Operational Range” on page 161). 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 19-9. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	f_{osc} = 1.0000MHz				f_{osc} = 1.8432MHz				f_{osc} = 2.0000MHz			
	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)	62.5kbps		125kbps		115.2kbps		230.4kbps		125kbps		250kbps	

Note: 1. UBRR = 0, error = 0.0%

Table 19-10. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 3.6864MHz				f_{osc} = 4.0000MHz				f_{osc} = 7.3728MHz			
	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)	230.4kbps		460.8kbps		250kbps		0.5Mbps		460.8kbps		921.6kbps	

Note: 1. UBRR = 0, error = 0.0%

Table 19-11. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 8.0000MHz				f_{osc} = 11.0592MHz				f_{osc} = 14.7456MHz			
	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)	0.5Mbps		1Mbps		691.2kbps		1.3824Mbps		921.6kbps		1.8432Mbps	

Note: 1. UBRR = 0, error = 0.0%

Table 19-12. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 16.0000MHz			
	U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error
2400	416	–0.1%	832	0.0%
4800	207	0.2%	416	–0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	–0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	–0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	–0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	–3.5%	16	2.1%
230.4k	3	8.5%	8	–3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max.(1)	1Mbps		2Mbps	

Note: 1. UBRR = 0, error = 0.0%

20. USART in SPI Mode

20.1 Features

- Full duplex, three-wire synchronous data transfer
- Master operation
- Supports all four SPI modes of operation (mode 0, 1, 2, and 3)
- LSB first or MSB first data transfer (configurable data order)
- Queued operation (double buffered)
- High resolution baud rate generator
- High speed operation (fXCKmax = fCK/2)
- Flexible interrupt generation

20.2 Overview

The universal synchronous and asynchronous serial receiver and transmitter (USART) can be set to a master SPI compliant mode of operation.

Setting both UMSELn1:0 bits to one enables the USART in MSPIM logic. In this mode of operation the SPI master control logic takes direct control over the USART resources. These resources include the transmitter and receiver shift register and buffers, and the baud rate generator. The parity generator and checker, the data and clock recovery logic, and the RX and TX control logic is disabled. The USART RX and TX control logic is replaced by a common SPI transfer control logic. However, the pin control logic and interrupt generation logic is identical in both modes of operation.

The I/O register locations are the same in both modes. However, some of the functionality of the control registers changes when using MSPIM.

20.3 Clock Generation

The clock generation logic generates the base clock for the transmitter and receiver. For USART MSPIM mode of operation only internal clock generation (i.e. master operation) is supported. The data direction register for the XCKn pin (DDR_XCKn) must therefore be set to one (i.e. as output) for the USART in MSPIM to operate correctly. Preferably the DDR_XCKn should be set up before the USART in MSPIM is enabled (i.e. TXENn and RXENn bit set to one).

The internal clock generation used in MSPIM mode is identical to the USART synchronous master mode. The baud rate or UBRRn setting can therefore be calculated using the same equations, see Table 20-1:

Table 20-1. Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRRn Value
Synchronous master mode	$BAUD = \dfrac{f_{OSC}}{2(\boldsymbol{UBRR}n + 1)}$	$\boldsymbol{UBRR}n = \dfrac{f_{OSC}}{2BAUD} - 1$

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
UBRRn	Contents of the UBRRnH and UBRRnL registers, (0-4095

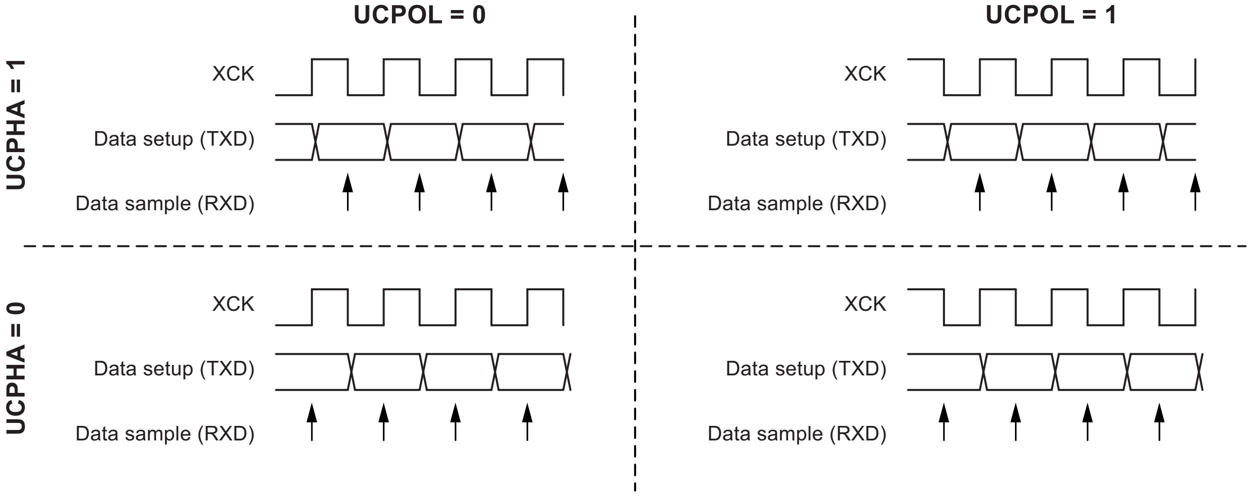
20.4 SPI Data Modes and Timing

There are four combinations of XCKn (SCK) phase and polarity with respect to serial data, which are determined by control bits UCPHAn and UCPOLn. The data transfer timing diagrams are shown in Figure 20-1. Data bits are shifted out and latched in on opposite edges of the XCKn signal, ensuring sufficient time for data signals to stabilize. The UCPOLn and UCPHAn functionality is summarized in Table 20-2. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the receiver and transmitter.

Table 20-2. UCPOLn and UCPHAn Functionality

UCPOLn	UCPHAn	SPI Mode	Leading Edge	Trailing Edge
0	0	0	Sample (rising)	Setup (falling)
0	1	1	Setup (rising)	Sample (falling)
1	0	2	Sample (falling)	Setup (rising)
1	1	3	Setup (falling)	Sample (rising)

Figure 20-1. UCPHAn and UCPOLn Data Transfer Timing Diagrams



20.5 Frame Formats

A serial frame for the MSPIM is defined to be one character of 8 data bits. The USART in MSPIM mode has two valid frame formats:

- 8-bit data with MSB first
- 8-bit data with LSB first

A frame starts with the least or most significant data bit. Then the next data bits, up to a total of eight, are succeeding, ending with the most or least significant bit accordingly. When a complete frame is transmitted, a new frame can directly follow it, or the communication line can be set to an idle (high) state.

The UDORDn bit in UCSRnC sets the frame format used by the USART in MSPIM mode. 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.

16-bit data transfer can be achieved by writing two data bytes to UDRn. A UART transmit complete interrupt will then signal that the 16-bit value has been shifted out.

20.5.1 USART MSPIM Initialization

The USART in MSPIM mode has to be initialized before any communication can take place. The initialization process normally consists of setting the baud rate, setting master mode of operation (by setting DDR_XCKn to one), setting frame format and enabling the transmitter and the receiver. Only the transmitter can operate independently. For interrupt driven USART operation, the global interrupt flag should be cleared (and thus interrupts globally disabled) when doing the initialization.

Note: To ensure immediate initialization of the XCKn output the baud-rate register (UBRRn) must be zero at the time the transmitter is enabled. Contrary to the normal mode USART operation the UBRRn must then be written to the desired value after the transmitter is enabled, but before the first transmission is started. Setting UBRRn to zero before enabling the transmitter is not necessary if the initialization is done immediately after a reset since UBRRn is reset to zero.

Before doing a re-initialization with changed baud rate, data mode, or frame format, be sure that there is 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 polling (no interrupts enabled). 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:
        clr r18
        out UBRRnH,r18
        out UBRRnL,r18
        ; Setting the XCKn port pin as output, enables master mode.
        sbi XCKn_DDR, XCKn
        ; Set MSPI mode of operation and SPI data mode 0.
        ldi r18, (1<<UMSELn1)|(1<<UMSELn0)|(0<<UCPHAn)|(0<<UCPOLn)
        out UCSRnC,r18
        ; Enable receiver and transmitter.
        ldi r18, (1<<RXENn)|(1<<TXENn)
        out UCSRnB,r18
        ; Set baud rate.
        ; IMPORTANT: The Baud Rate must be set after the transmitter is
                        enabled!
        out UBRRnH, r17
        out UBRRnL, r18
        ret
```

C Code Example(1)

```
void USART_Init(unsigned int baud)
{
        UBRRn = 0;
        /* Setting the XCKn port pin as output, enables master mode. */
        XCKn_DDR |= (1<<XCKn);
        /* Set MSPI mode of operation and SPI data mode 0. */
        UCSRnC = (1<<UMSELn1)|(1<<UMSELn0)|(0<<UCPHAn)|(0<<UCPOLn);
        /* Enable receiver and transmitter. */
        UCSRnB = (1<<RXENn)|(1<<TXENn);
        /* Set baud rate. */
        /* IMPORTANT: The Baud Rate must be set after the transmitter is
                        enabled
        */
        UBRRn = baud;
}
```

Note: 1. See Section 4. "About Code Examples" on page 8.

20.6 Data Transfer

Using the USART in MSPI mode requires the transmitter to be enabled, i.e. the TXENn bit in the UCSRnB register is set to one. When the transmitter is enabled, the normal port operation of the TxDn pin is overridden and given the function as the transmitter's serial output. Enabling the receiver is optional and is done by setting the RXENn bit in the UCSRnB register to one. When the receiver is enabled, the normal pin operation of the RxDn pin is overridden and given the function as the receiver's serial input. The XCKn will in both cases be used as the transfer clock.

After initialization the USART is ready for doing data transfers. A data transfer is initiated by writing to the UDRn I/O location. This is the case for both sending and receiving data since the transmitter controls the transfer clock. The data written to UDRn is moved from the transmit buffer to the shift register when the shift register is ready to send a new frame.

Note: To keep the input buffer in sync with the number of data bytes transmitted, the UDRn register must be read once for each byte transmitted. The input buffer operation is identical to normal USART mode, i.e. if an overflow occurs the character last received will be lost, not the first data in the buffer. This means that if four bytes are transferred, byte 1 first, then byte 2, 3, and 4, and the UDRn is not read before all transfers are completed, then byte 3 to be received will be lost, and not byte 1.

The following code examples show a simple USART in MSPIM mode transfer function based on polling of the data register empty (UDREn) flag and the receive complete (RXCn) flag. 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 and the data received will be available in the same register (R16) after the function returns.

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. The function then waits for data to be present in the receive buffer by checking the RXCn flag, before reading the buffer and returning the value.

Assembly Code Example(1)

```
        USART_MSPIM_Transfer:
                ; Wait for empty transmit buffer
                sbis UCSRnA, UDREn
                rjmp USART_MSPIM_Transfer
                ; Put data (r16) into buffer, sends the data
                out UDRn,r16
                ; Wait for data to be received
        USART_MSPIM_Wait_RXCn:
                sbis UCSRnA, RXCn
                rjmp USART_MSPIM_Wait_RXCn
                ; Get and return received data from buffer
                in r16, UDRn
                ret
```

C Code Example(1)

```
        unsigned char USART_Receive(void)
        {
                /* Wait for empty transmit buffer */
                while (!(UCSRnA & (1<<UDREn)));
                /* Put data into buffer, sends the data */
                UDRn = data;
                /* Wait for data to be received */
                while (!(UCSRnA & (1<<RXCn)));
                /* Get and return received data from buffer */
                return UDRn;
        }
```

Note: 1. See Section 4. "About Code Examples" on page 8.

20.6.1 Transmitter and Receiver Flags and Interrupts

The RXCn, TXCn, and UDREn flags and corresponding interrupts in USART in MSPIM mode are identical in function to the normal USART operation. However, the receiver error status flags (FE, DOR, and PE) are not in use and is always read as zero.

20.6.2 Disabling the Transmitter or Receiver

The disabling of the transmitter or receiver in USART in MSPIM mode is identical in function to the normal USART operation.

20.7 AVR USART MSPIM versus AVR SPI

The USART in MSPIM mode is fully compatible with the AVR® SPI regarding:

- Master mode timing diagram.
- The UCPOLn bit functionality is identical to the SPI CPOL bit.
- The UCPHAn bit functionality is identical to the SPI CPHA bit.
- The UDORDn bit functionality is identical to the SPI DORD bit.

However, since the USART in MSPIM mode reuses the USART resources, the use of the USART in MSPIM mode is somewhat different compared to the SPI. In addition to differences of the control register bits, and that only master operation is supported by the USART in MSPIM mode, the following features differ between the two modules:

- The USART in MSPIM mode includes (double) buffering of the transmitter. The SPI has no buffer.
- The USART in MSPIM mode receiver includes an additional buffer level.
- The SPI WCOL (write collision) bit is not included in USART in MSPIM mode.
- The SPI double speed mode (SPI2X) bit is not included. However, the same effect is achieved by setting UBRRn accordingly.
- Interrupt timing is not compatible.
- Pin control differs due to the master only operation of the USART in MSPIM mode.

A comparison of the USART in MSPIM mode and the SPI pins is shown in Table 20-3.

Table 20-3. Comparison of USART in MSPIM mode and SPI pins.

USART_MSPIM	SPI	Comment
TxDn	MOSI	Master out only
RxDn	MISO	Master in only
XCKn	SCK	(Functionally identical)
(N/A)	$\overline{\text{SS}}$	Not supported by USART in MSPIM

20.8 Register Description

The following section describes the registers used for SPI operation using the USART.

20.8.1 UDRn – USART MSPIM I/O Data Register

The function and bit description of the USART data register (UDRn) in MSPI mode is identical to normal USART operation. See Section 19.11.1 “UDRn – USART I/O Data Register n” on page 163.

20.8.2 UCSRnA – USART MSPIM Control and Status Register n A

Bit	7	6	5	4	3	2	1	0	
	RXCn	**TXCn**	**UDREn**	–	–	–	–	–	**UCSRnA**
Read/Write	R/W	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	1	1	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 UDRIE bit). UDREn is set after a reset to indicate that the transmitter is ready.

- **Bit 4:0 – Reserved in MSPI mode**

When in MSPI mode, these bits are reserved for future use. For compatibility with future devices, these bits must be written to zero when UCSRnA is written.

20.8.3 UCSRnB – USART MSPIM Control and Status Register n B

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	**TXCIEn**	**UDRIE**	**RXENn**	**TXENn**	–	–	–	**UCSRnB**
Read/Write	R/W	R/W	R/W	R/W	R/W	R	R	R	
Initial Value	0	0	0	0	0	1	1	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 – UDRIE: 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 UDRIE 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 in MSPIM mode. The receiver will override normal port operation for the RxDn pin when enabled. Disabling the receiver will flush the receive buffer. Only enabling the receiver in MSPI mode (i.e. setting RXENn=1 and TXENn=0) has no meaning since it is the transmitter that controls the transfer clock and since only master mode is supported.

- **Bit 3 – TXENn: Transmitter Enable**

Writing this bit to one enables the USART transmitter. The transmitter will override normal port operation for the TxDn 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 TxDn port.

- **Bit 2:0 – Reserved in MSPI mode**

When in MSPI mode, these bits are reserved for future use. For compatibility with future devices, these bits must be written to zero when UCSRnB is written.

20.8.4 UCSRnC – USART MSPIM Control and Status Register n C

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	–	–	–	UDORDn	UCPHAn	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

- **Bit 7:6 – UMSELn1:0: USART Mode Select**

These bits select the mode of operation of the USART as shown in Table 20-4. See Section 19.11.4 "UCSRnC – USART Control and Status Register n C" on page 165 for full description of the normal USART operation. The MSPIM is enabled when both UMSELn bits are set to one. The UDORDn, UCPHAn, and UCPOLn can be set in the same write operation where the MSPIM is enabled.

Table 20-4. UMSELn Bits Settings

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)

- **Bit 5:3 – Reserved in MSPI mode**

When in MSPI mode, these bits are reserved for future use. For compatibility with future devices, these bits must be written to zero when UCSRnC is written.

- **Bit 2 – UDORDn: Data Order**

When set to one the LSB of the data word is transmitted first. When set to zero the MSB of the data word is transmitted first. Refer to the frame formats section page 4 for details.

- **Bit 1 – UCPHAn: Clock Phase**

The UCPHAn bit setting determine if data is sampled on the leasing edge (first) or tailing (last) edge of XCKn. Refer to the SPI data modes and timing section page 4 for details.

- **Bit 0 – UCPOLn: Clock Polarity**

The UCPOLn bit sets the polarity of the XCKn clock. The combination of the UCPOLn and UCPHAn bit settings determine the timing of the data transfer. Refer to the SPI data modes and Timing section page 4 for details.

20.8.5 UBRRnL and UBRRnH –USART MSPIM Baud Rate Registers

The function and bit description of the baud rate registers in MSPI mode is identical to normal USART operation. See Section 19.11.5 "UBRRnL and UBRRnH – USART Baud Rate Registers" on page 166.

21. Two-wire Serial Interface

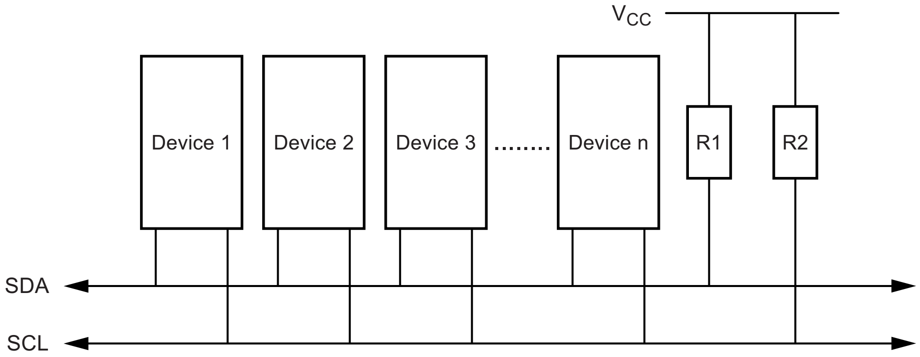
21.1 Features

- Simple yet powerful and flexible communication interface, only two bus lines needed
- Both master and slave operation supported
- Device can operate as transmitter or receiver
- 7-bit address space allows up to 128 different slave addresses
- Multi-master arbitration support
- Up to 400kHz data transfer speed
- Slew-rate limited output drivers
- Noise suppression circuitry rejects spikes on bus lines
- Fully programmable slave address with general call support
- Address recognition causes wake-up when AVR® is in sleep mode

21.2 Two-wire Serial Interface Bus Definition

The two-wire serial interface (TWI) is ideally suited for typical microcontroller applications. The TWI protocol allows the systems designer to interconnect up to 128 different devices using only two bi-directional bus lines, one for clock (SCL) and one for data (SDA). The only external hardware needed to implement the bus is a single pull-up resistor for each of the TWI bus lines. All devices connected to the bus have individual addresses, and mechanisms for resolving bus contention are inherent in the TWI protocol.

Figure 21-1. TWI Bus Interconnection



21.2.1 TWI Terminology

The following definitions are frequently encountered in this section.

Table 21-1. TWI Terminology

Term	Description
Master	The device that initiates and terminates a transmission. The master also generates the SCL clock
Slave	The device addressed by a master
Transmitter	The device placing data on the bus
Receiver	The device reading data from the bus

The power reduction TWI bit, PRTWI bit in Section 10.12.3 “PRR0 – Power Reduction Register 0” on page 39 must be written to zero to enable the 2-wire serial interface.

21.2.2 Electrical Interconnection

As depicted in Figure 21-1 on page 176, both bus lines are connected to the positive supply voltage through pull-up resistors. The bus drivers of all TWI-compliant devices are open-drain or open-collector. This implements a wired-AND function which is essential to the operation of the interface. A low level on a TWI bus line is generated when one or more TWI devices output a zero. A high level is output when all TWI devices trim-state their outputs, allowing the pull-up resistors to pull the line high. Note that all AVR® devices connected to the TWI bus must be powered in order to allow any bus operation.

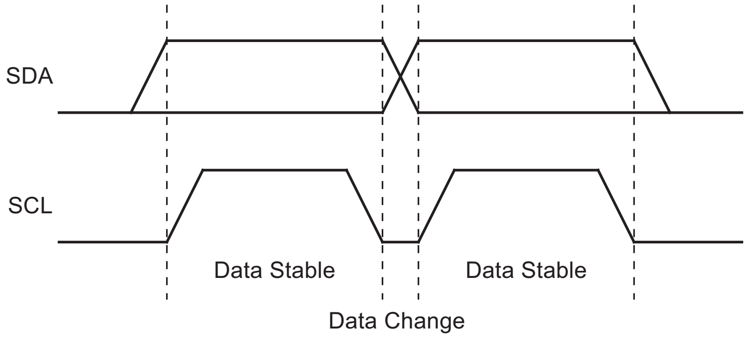
The number of devices that can be connected to the bus is only limited by the bus capacitance limit of 400pF and the 7-bit slave address space. A detailed specification of the electrical characteristics of the TWI is given in Section 28.7 "SPI Timing Characteristics" on page 292. Two different sets of specifications are presented there, one relevant for bus speeds below 100kHz, and one valid for bus speeds up to 400kHz.

21.3 Data Transfer and Frame Format

21.3.1 Transferring Bits

Each data bit transferred on the TWI bus is accompanied by a pulse on the clock line. The level of the data line must be stable when the clock line is high. The only exception to this rule is for generating start and stop conditions.

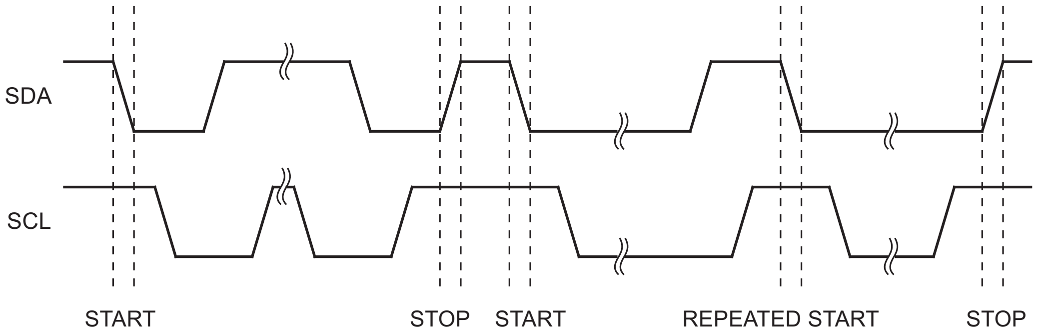
Figure 21-2. Data Validity



21.3.2 START and STOP Conditions

The master initiates and terminates a data transmission. The transmission is initiated when the master issues a START condition on the bus, and it is terminated when the master issues a STOP condition. Between a START and a STOP condition, the bus is considered busy, and no other master should try to seize control of the bus. A special case occurs when a new START condition is issued between a START and STOP condition. This is referred to as a REPEATED START condition, and is used when the master wishes to initiate a new transfer without relinquishing control of the bus. After a REPEATED START, the bus is considered busy until the next STOP. This is identical to the START behavior, and therefore START is used to describe both START and REPEATED START for the remainder of this datasheet, unless otherwise noted. As depicted below, START and STOP conditions are signalled by changing the level of the SDA line when the SCL line is high.

Figure 21-3. START, REPEATED START and STOP conditions



21.3.3 Address Packet Format

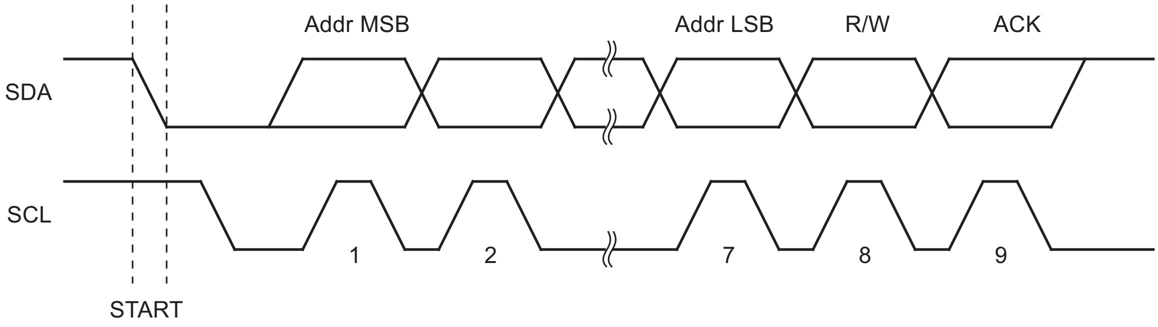
All address packets transmitted on the TWI bus are 9 bits long, consisting of 7 address bits, one READ/WRITE control bit and an acknowledge bit. If the READ/WRITE bit is set, a read operation is to be performed, otherwise a write operation should be performed. When a slave recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle. If the addressed Slave is busy, or for some other reason can not service the master's request, the SDA line should be left high in the ACK clock cycle. The master can then transmit a STOP condition, or a REPEATED START condition to initiate a new transmission. An address packet consisting of a slave address and a READ or a WRITE bit is called SLA+R or SLA+W, respectively.

The MSB of the address byte is transmitted first. Slave addresses can freely be allocated by the designer, but the address 0000 000 is reserved for a general call.

When a general call is issued, all slaves should respond by pulling the SDA line low in the ACK cycle. A general call is used when a master wishes to transmit the same message to several slaves in the system. When the general call address followed by a Write bit is transmitted on the bus, all slaves set up to acknowledge the general call will pull the SDA line low in the ack cycle. The following data packets will then be received by all the slaves that acknowledged the general call. Note that transmitting the general call address followed by a read bit is meaningless, as this would cause contention if several slaves started transmitting different data.

All addresses of the format 1111 xxx should be reserved for future purposes.

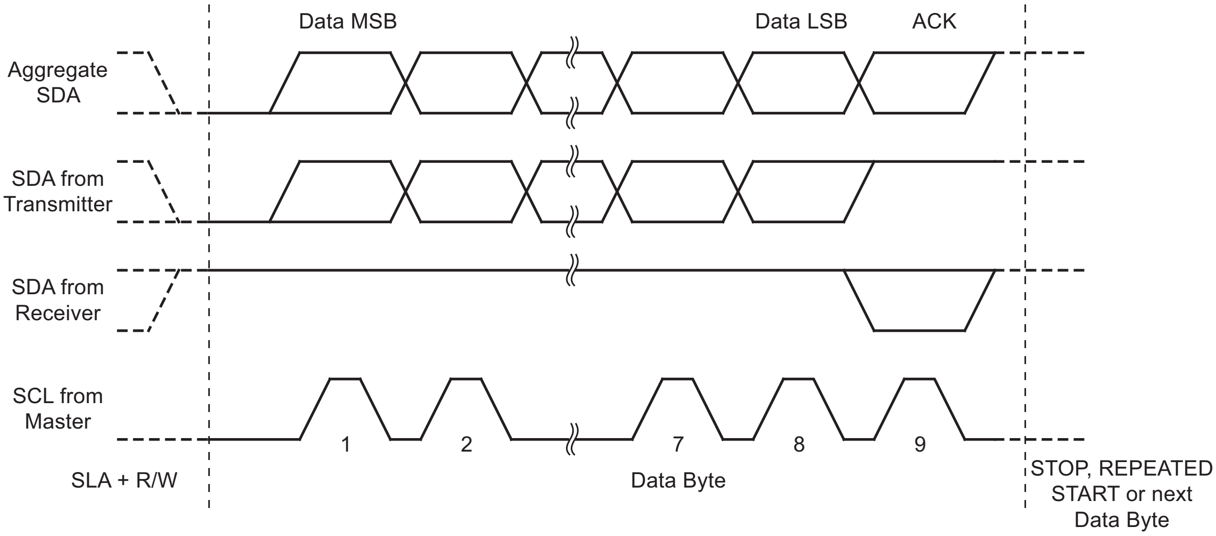
Figure 21-4. Address Packet Format



21.3.4 Data Packet Format

All data packets transmitted on the TWI bus are nine bits long, consisting of one data byte and an acknowledge bit. During a data transfer, the master generates the clock and the START and STOP conditions, while the receiver is responsible for acknowledging the reception. An acknowledge (ACK) is signalled by the receiver pulling the SDA line low during the ninth SCL cycle. If the receiver leaves the SDA line high, a NACK is signalled. When the receiver has received the last byte, or for some reason cannot receive any more bytes, it should inform the transmitter by sending a NACK after the final byte. The MSB of the data byte is transmitted first.

Figure 21-5. Data Packet Format

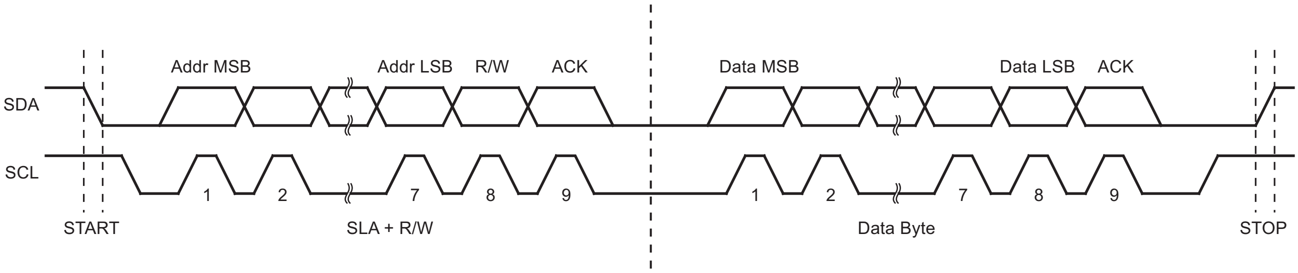


21.3.5 Combining Address and Data Packets into a Transmission

A transmission basically consists of a START condition, a SLA+R/W, one or more data packets and a STOP condition. An empty message, consisting of a START followed by a STOP condition, is illegal. Note that the Wired-ANDing of the SCL line can be used to implement handshaking between the master and the slave. The slave can extend the SCL low period by pulling the SCL line low. This is useful if the clock speed set up by the master is too fast for the slave, or the slave needs extra time for processing between the data transmissions. The slave extending the SCL low period will not affect the SCL high period, which is determined by the master. As a consequence, the slave can reduce the TWI data transfer speed by prolonging the SCL duty cycle.

Figure 21-6 shows a typical data transmission. Note that several data bytes can be transmitted between the SLA+R/W and the STOP condition, depending on the software protocol implemented by the application software.

Figure 21-6. Typical Data Transmission



21.4 Multi-master Bus Systems, Arbitration and Synchronization

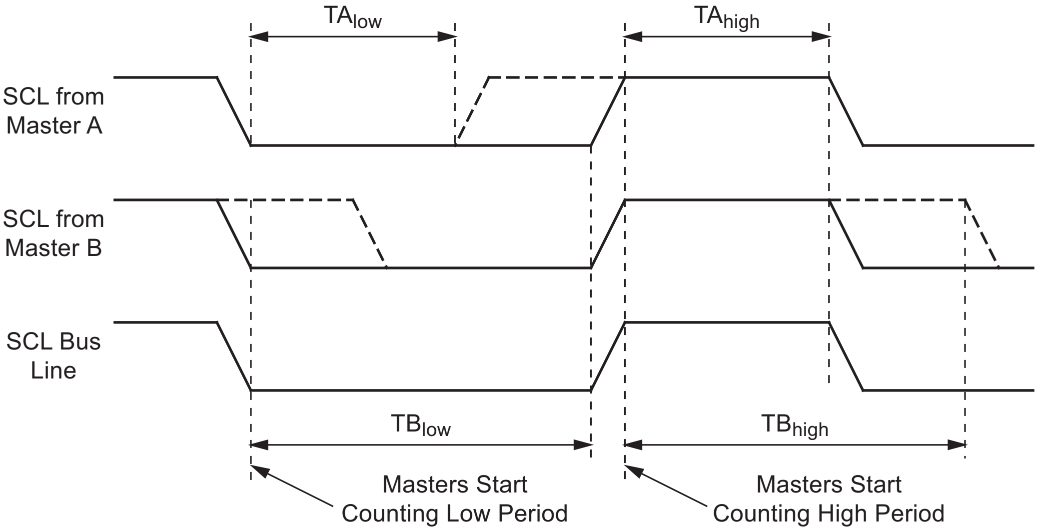
The TWI protocol allows bus systems with several masters. Special concerns have been taken in order to ensure that transmissions will proceed as normal, even if two or more masters initiate a transmission at the same time. Two problems arise in multi-master systems:

- An algorithm must be implemented allowing only one of the masters to complete the transmission. All other masters should cease transmission when they discover that they have lost the selection process. This selection process is called arbitration. When a contending master discovers that it has lost the arbitration process, it should immediately switch to slave mode to check whether it is being addressed by the winning master. The fact that multiple masters have started transmission at the same time should not be detectable to the slaves, i.e. the data being transferred on the bus must not be corrupted.

- Different masters may use different SCL frequencies. A scheme must be devised to synchronize the serial clocks from all masters, in order to let the transmission proceed in a lockstep fashion. This will facilitate the arbitration process.

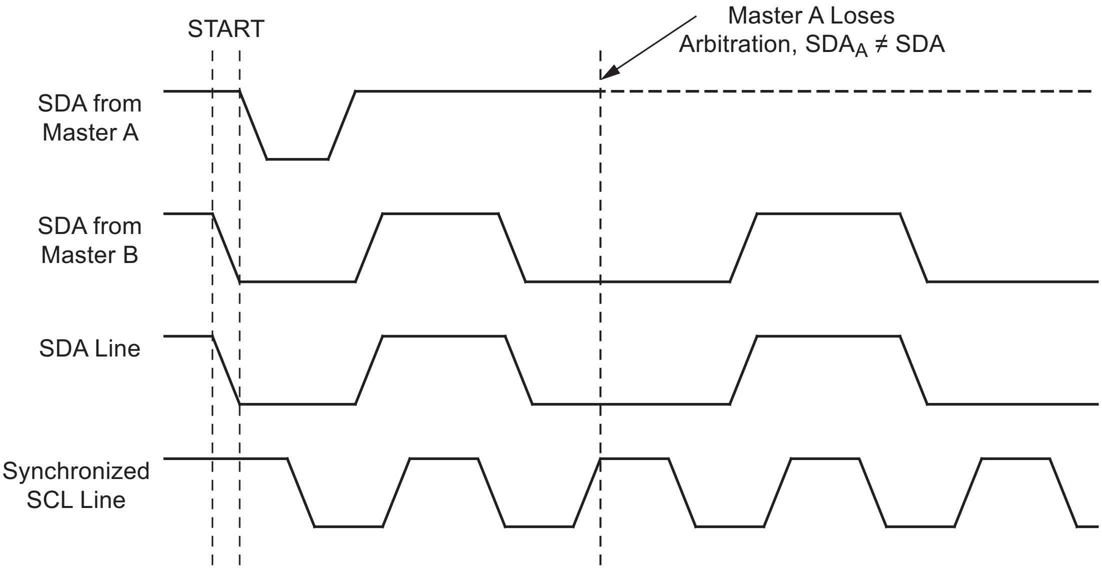
The wired-ANDing of the bus lines is used to solve both these problems. The serial clocks from all masters will be wired-ANDed, yielding a combined clock with a high period equal to the one from the master with the shortest high period. The low period of the combined clock is equal to the low period of the master with the longest low period. Note that all masters listen to the SCL line, effectively starting to count their SCL high and low time-out periods when the combined SCL line goes high or low, respectively.

Figure 21-7. SCL Synchronization Between Multiple Masters



Arbitration is carried out by all masters continuously monitoring the SDA line after outputting data. If the value read from the SDA line does not match the value the master had output, it has lost the arbitration. Note that a master can only lose arbitration when it outputs a high SDA value while another master outputs a low value. The losing master should immediately go to slave mode, checking if it is being addressed by the winning master. The SDA line should be left high, but losing masters are allowed to generate a clock signal until the end of the current data or address packet. Arbitration will continue until only one master remains, and this may take many bits. If several masters are trying to address the same slave, arbitration will continue into the data packet.

Figure 21-8. Arbitration Between Two Masters



Note that arbitration is not allowed between:

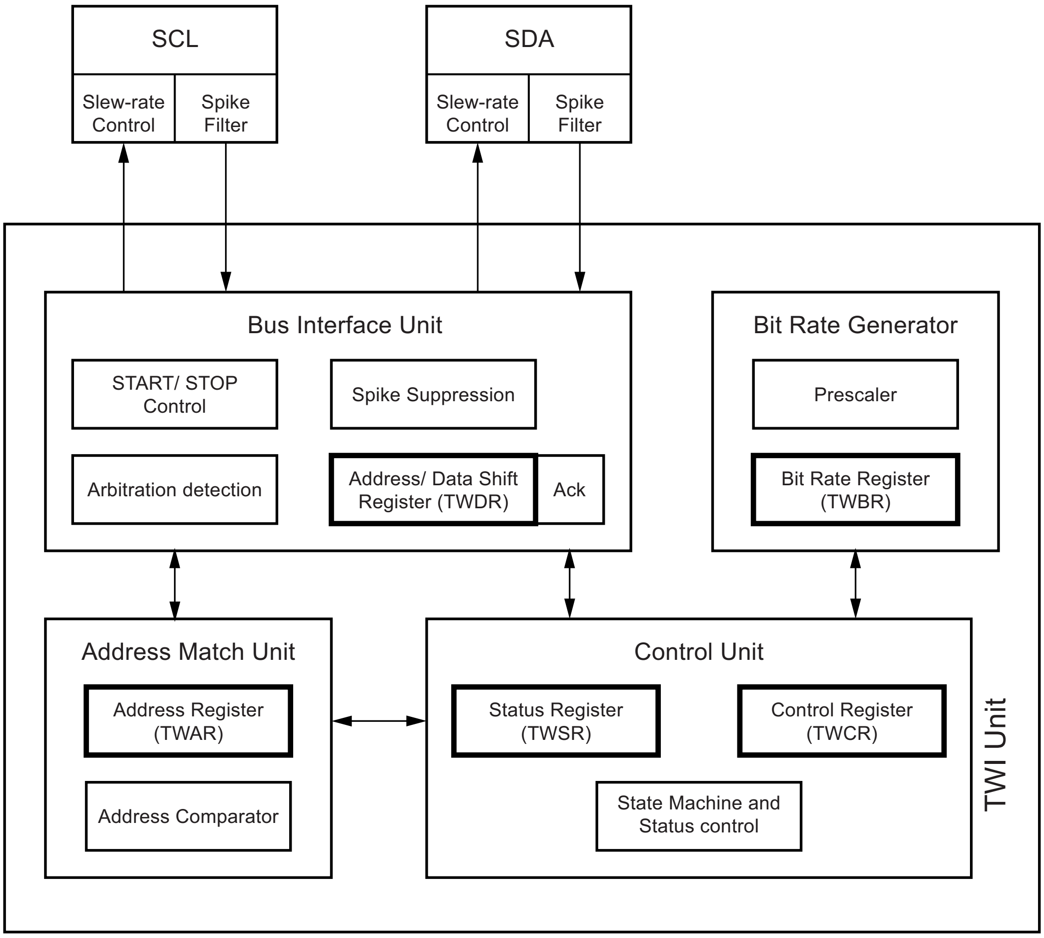
- A REPEATED START condition and a data bit.
- A STOP condition and a data bit.
- A REPEATED START and a STOP condition.

It is the user software's responsibility to ensure that these illegal arbitration conditions never occur. This implies that in multi-master systems, all data transfers must use the same composition of SLA+R/W and data packets. In other words: All transmissions must contain the same number of data packets, otherwise the result of the arbitration is undefined.

21.5 Overview of the TWI Module

The TWI module is comprised of several submodules, as shown in Figure 21-9. All registers drawn in a thick line are accessible through the AVR® data bus.

Figure 21-9. Overview of the TWI Module



21.5.1 SCL and SDA Pins

These pins interface the AVR® TWI with the rest of the MCU system. The output drivers contain a slew-rate limiter in order to conform to the TWI specification. The input stages contain a spike suppression unit removing spikes shorter than 50ns. Note that the internal pull-ups in the AVR pads can be enabled by setting the PORT bits corresponding to the SCL and SDA pins, as explained in the I/O port section. The internal pull-ups can in some systems eliminate the need for external ones.

21.5.2 Bit Rate Generator Unit

This unit controls the period of SCL when operating in a master mode. The SCL period is controlled by settings in the TWI Bit rate register (TWBR) and the prescaler bits in the TWI status register (TWSR). Slave operation does not depend on bit rate or prescaler settings, but the CPU clock frequency in the slave must be at least 16 times higher than the SCL frequency. Note that slaves may prolong the SCL low period, thereby reducing the average TWI bus clock period. The SCL frequency is generated according to the following equation:

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{TWPS}}$$

- TWBR = value of the TWI bit rate register.
- TWPS = value of the prescaler bits in the TWI status register.

Note: Pull-up resistor values should be selected according to the SCL frequency and the capacitive bus line load. See 2-wire serial bus requirements in Table 28-9 on page 293 for value of pull-up resistor.

21.5.3 Bus Interface Unit

This unit contains the data and address shift register (TWDR), a START/STOP controller and arbitration detection hardware. The TWDR contains the address or data bytes to be transmitted, or the address or data bytes received. In addition to the 8-bit TWDR, the bus interface unit also contains a register containing the (N)ACK bit to be transmitted or received. This (N)ACK register is not directly accessible by the application software. However, when receiving, it can be set or cleared by manipulating the TWI control register (TWCR). When in transmitter mode, the value of the received (N)ACK bit can be determined by the value in the TWSR.

The START/STOP controller is responsible for generation and detection of START, REPEATED START, and STOP conditions. The START/STOP controller is able to detect START and STOP conditions even when the AVR MCU is in one of the sleep modes, enabling the MCU to wake up if addressed by a master.

If the TWI has initiated a transmission as master, the arbitration detection hardware continuously monitors the transmission trying to determine if arbitration is in process. If the TWI has lost an arbitration, the control unit is informed. correct action can then be taken and appropriate status codes generated.

21.5.4 Address Match Unit

The address match unit checks if received address bytes match the seven-bit address in the TWI address register (TWAR). If the TWI general call recognition enable (TWGCE) bit in the TWAR is written to one, all incoming address bits will also be compared against the general call address. Upon an address match, the control unit is informed, allowing correct action to be taken. The TWI may or may not acknowledge its address, depending on settings in the TWCR. The address match unit is able to compare addresses even when the AVR MCU is in sleep mode, enabling the MCU to wake up if addressed by a master.

21.5.5 Control Unit

The control unit monitors the TWI bus and generates responses corresponding to settings in the TWI control register (TWCR). When an event requiring the attention of the application occurs on the TWI bus, the TWI interrupt flag (TWINT) is asserted. In the next clock cycle, the TWI status register (TWSR) is updated with a status code identifying the event. The TWSR only contains relevant status information when the TWI interrupt flag is asserted. At all other times, the TWSR contains a special status code indicating that no relevant status information is available. As long as the TWINT flag is set, the SCL line is held low. This allows the application software to complete its tasks before allowing the TWI transmission to continue.

The TWINT flag is set in the following situations:

- After the TWI has transmitted a START/REPEATED START condition.
- After the TWI has transmitted SLA+R/W.
- After the TWI has transmitted an address byte.
- After the TWI has lost arbitration.
- After the TWI has been addressed by own slave address or general call.
- After the TWI has received a data byte.
- After a STOP or REPEATED START has been received while still addressed as a slave.
- When a bus error has occurred due to an illegal START or STOP condition.

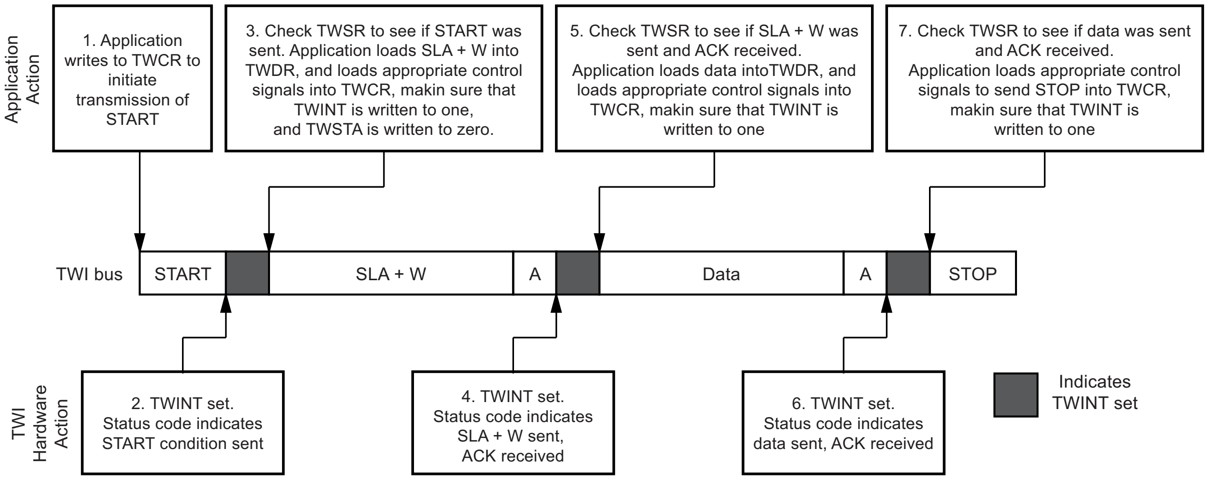
21.6 Using the TWI

The AVR® TWI is byte-oriented and interrupt based. Interrupts are issued after all bus events, like reception of a byte or transmission of a START condition. Because the TWI is interrupt-based, the application software is free to carry on other operations during a TWI byte transfer. Note that the TWI interrupt enable (TWIE) bit in TWCR together with the global interrupt enable bit in SREG allow the application to decide whether or not assertion of the TWINT flag should generate an interrupt request. If the TWIE bit is cleared, the application must poll the TWINT flag in order to detect actions on the TWI bus.

When the TWINT flag is asserted, the TWI has finished an operation and awaits application response. In this case, the TWI status register (TWSR) contains a value indicating the current state of the TWI bus. The application software can then decide how the TWI should behave in the next TWI bus cycle by manipulating the TWCR and TWDR registers.

Figure 21-10 is a simple example of how the application can interface to the TWI hardware. In this example, a master wishes to transmit a single data byte to a slave. This description is quite abstract, a more detailed explanation follows later in this section. A simple code example implementing the desired behavior is also presented.

Figure 21-10. Interfacing the Application to the TWI in a Typical Transmission



1. The first step in a TWI transmission is to transmit a START condition. This is done by writing a specific value into TWCR, instructing the TWI hardware to transmit a START condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the START condition.
2. When the START condition has been transmitted, the TWINT flag in TWCR is set, and TWSR is updated with a status code indicating that the START condition has successfully been sent.

3. The application software should now examine the value of TWSR, to make sure that the START condition was successfully transmitted. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load SLA+W into TWDR. Remember that TWDR is used both for address and data. After TWDR has been loaded with the desired SLA+W, a specific value must be written to TWCR, instructing the TWI hardware to transmit the SLA+W present in TWDR. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the address packet.
4. When the address packet has been transmitted, the TWINT flag in TWCR is set, and TWSR is updated with a status code indicating that the address packet has successfully been sent. The status code will also reflect whether a slave acknowledged the packet or not.
5. The application software should now examine the value of TWSR, to make sure that the address packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load a data packet into TWDR. Subsequently, a specific value must be written to TWCR, instructing the TWI hardware to transmit the data packet present in TWDR. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the data packet.
6. When the data packet has been transmitted, the TWINT flag in TWCR is set, and TWSR is updated with a status code indicating that the data packet has successfully been sent. The status code will also reflect whether a slave acknowledged the packet or not.
7. The application software should now examine the value of TWSR, to make sure that the data packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must write a specific value to TWCR, instructing the TWI hardware to transmit a STOP condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the STOP condition. Note that TWINT is NOT set after a STOP condition has been sent.

Even though this example is simple, it shows the principles involved in all TWI transmissions. These can be summarized as follows:

- When the TWI has finished an operation and expects application response, the TWINT flag is set. The SCL line is pulled low until TWINT is cleared.
- When the TWINT flag is set, the user must update all TWI registers with the value relevant for the next TWI bus cycle. As an example, TWDR must be loaded with the value to be transmitted in the next bus cycle.
- After all TWI register updates and other pending application software tasks have been completed, TWCR is written. When writing TWCR, the TWINT bit should be set. Writing a one to TWINT clears the flag. The TWI will then commence executing whatever operation was specified by the TWCR setting.

In the following an assembly and C implementation of the example is given. Note that the code below assumes that several definitions have been made, for example by using include-files.

Table 21-2. Assembly Code Example

	Assembly Code Example	C Example	Comments
1	**ldi** r16, (1<<TWINT)|(1<<TWSTA)|(1<<TWEN) **out** TWCR, r16	TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)	Send START condition
2	wait1: **in** r16,TWCR **sbrs** r16,TWINT **rjmp** wait1	**while** (!(TWCR & (1<<TWINT))) ;	Wait for TWINT flag set. This indicates that the START condition has been transmitted
3	**in** r16,TWSR **andi** r16, 0xF8 **cpi** r16, START **brne** ERROR	**if** ((TWSR & 0xF8)!= START) ERROR();	Check value of TWI status register. Mask prescaler bits. If status different from START go to ERROR
	ldi r16, SLA_W **out** TWDR, r16 **ldi** r16, (1<<TWINT) | (1<<TWEN) **out** TWCR, r16	TWDR = SLA_W; TWCR = (1<<TWINT) | (1<<TWEN);	Load SLA_W into TWDR register. Clear TWINT bit in TWCR to start transmission of address
4	wait2: **in** r16,TWCR **sbrs** r16,TWINT **rjmp** wait2	**while** (!(TWCR & (1<<TWINT))) ;	Wait for TWINT flag set. This indicates that the SLA+W has been transmitted, and ACK/NACK has been received.
5	**in** r16,TWSR **andi** r16, 0xF8 **cpi** r16, MT_SLA_ACK **brne** ERROR	**if** ((TWSR & 0xF8)!= MT_SLA_ACK) ERROR();	Check value of TWI status register. Mask prescaler bits. If status different from MT_SLA_ACK go to ERROR
	ldi r16, DATA **out** TWDR, r16 **ldi** r16, (1<<TWINT) | (1<<TWEN) **out** TWCR, r16	TWDR = DATA; TWCR = (1<<TWINT) | (1<<TWEN);	Load DATA into TWDR register. Clear TWINT bit in TWCR to start transmission of data
6	wait3: **in** r16,TWCR **sbrs** r16,TWINT **rjmp** wait3	**while** (!(TWCR & (1<<TWINT))) ;	Wait for TWINT flag set. This indicates that the DATA has been transmitted, and ACK/NACK has been received.
7	**in** r16,TWSR **andi** r16, 0xF8 **cpi** r16, MT_DATA_ACK **brne** ERROR	**if** ((TWSR & 0xF8)!= MT_DATA_ACK) ERROR();	Check value of TWI status register. Mask prescaler bits. If status different from MT_DATA_ACK go to ERROR
	ldi r16, (1<<TWINT)|(1<<TWEN)|(1<<TWSTO) **out** TWCR, r16	TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);	Transmit STOP condition

21.7 Transmission Modes

The TWI can operate in one of four major modes. These are named master transmitter (MT), master receiver (MR), slave transmitter (ST) and slave receiver (SR). Several of these modes can be used in the same application. As an example, the TWI can use MT mode to write data into a TWI EEPROM, MR mode to read the data back from the EEPROM. If other masters are present in the system, some of these might transmit data to the TWI, and then SR mode would be used. It is the application software that decides which modes are legal.

The following sections describe each of these modes. Possible status codes are described along with figures detailing data transmission in each of the modes. These figures contain the following abbreviations:

S: START condition

Rs: REPEATED START condition

R: Read bit (high level at SDA)

W: Write bit (low level at SDA)

A: Acknowledge bit (low level at SDA)

$\overline{A}$: Not acknowledge bit (high level at SDA)

Data: 8-bit data byte

P: STOP condition

SLA: Slave address

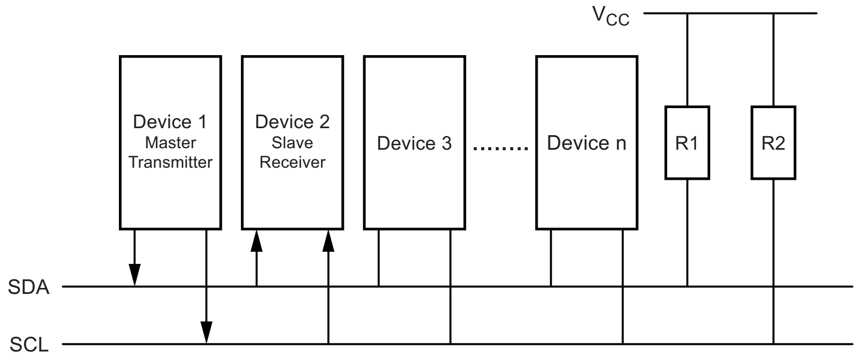
In Figure 21-12 on page 189 to Figure 21-18 on page 198, circles are used to indicate that the TWINT flag is set. The numbers in the circles show the status code held in TWSR, with the prescaler bits masked to zero. At these points, actions must be taken by the application to continue or complete the TWI transfer. The TWI transfer is suspended until the TWINT flag is cleared by software.

When the TWINT flag is set, the status code in TWSR is used to determine the appropriate software action. For each status code, the required software action and details of the following serial transfer are given in Table 21-3 on page 188 to Table 21-6 on page 197. Note that the prescaler bits are masked to zero in these tables.

21.7.1 Master Transmitter Mode

In the master transmitter mode, a number of data bytes are transmitted to a slave receiver (see Figure 21-11). In order to enter a master mode, a START condition must be transmitted. The format of the following address packet determines whether master transmitter or master receiver mode is to be entered. If SLA+W is transmitted, MT mode is entered, if SLA+R is transmitted, MR mode is entered. All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 21-11. Data Transfer in Master Transmitter Mode



A START condition is sent by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	1	0	X	1	0	X

TWEN must be set to enable the 2-wire serial interface, TWSTA must be written to one to transmit a START condition and TWINT must be written to one to clear the TWINT flag. The TWI will then test the 2-wire serial bus and generate a START condition as soon as the bus becomes free. After a START condition has been transmitted, the TWINT flag is set by hardware, and the status code in TWSR will be 0x08 (see Table 21-3). In order to enter MT mode, SLA+W must be transmitted. This is done by writing SLA+W to TWDR. Thereafter the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	0	0	X	1	0	X

When SLA+W have been transmitted and an acknowledgement bit has been received, TWINT is set again and a number of status codes in TWSR are possible. Possible status codes in master mode are 0x18, 0x20, or 0x38. The appropriate action to be taken for each of these status codes is detailed in Table 21-3.

When SLA+W has been successfully transmitted, a data packet should be transmitted. This is done by writing the data byte to TWDR. TWDR must only be written when TWINT is high. If not, the access will be discarded, and the write collision bit (TWWC) will be set in the TWCR register. After updating TWDR, the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	0	0	X	1	0	X

This scheme is repeated until the last byte has been sent and the transfer is ended by generating a STOP condition or a repeated START condition. A STOP condition is generated by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	0	1	X	1	0	X

A REPEATED START condition is generated by writing the following value to TWCR:

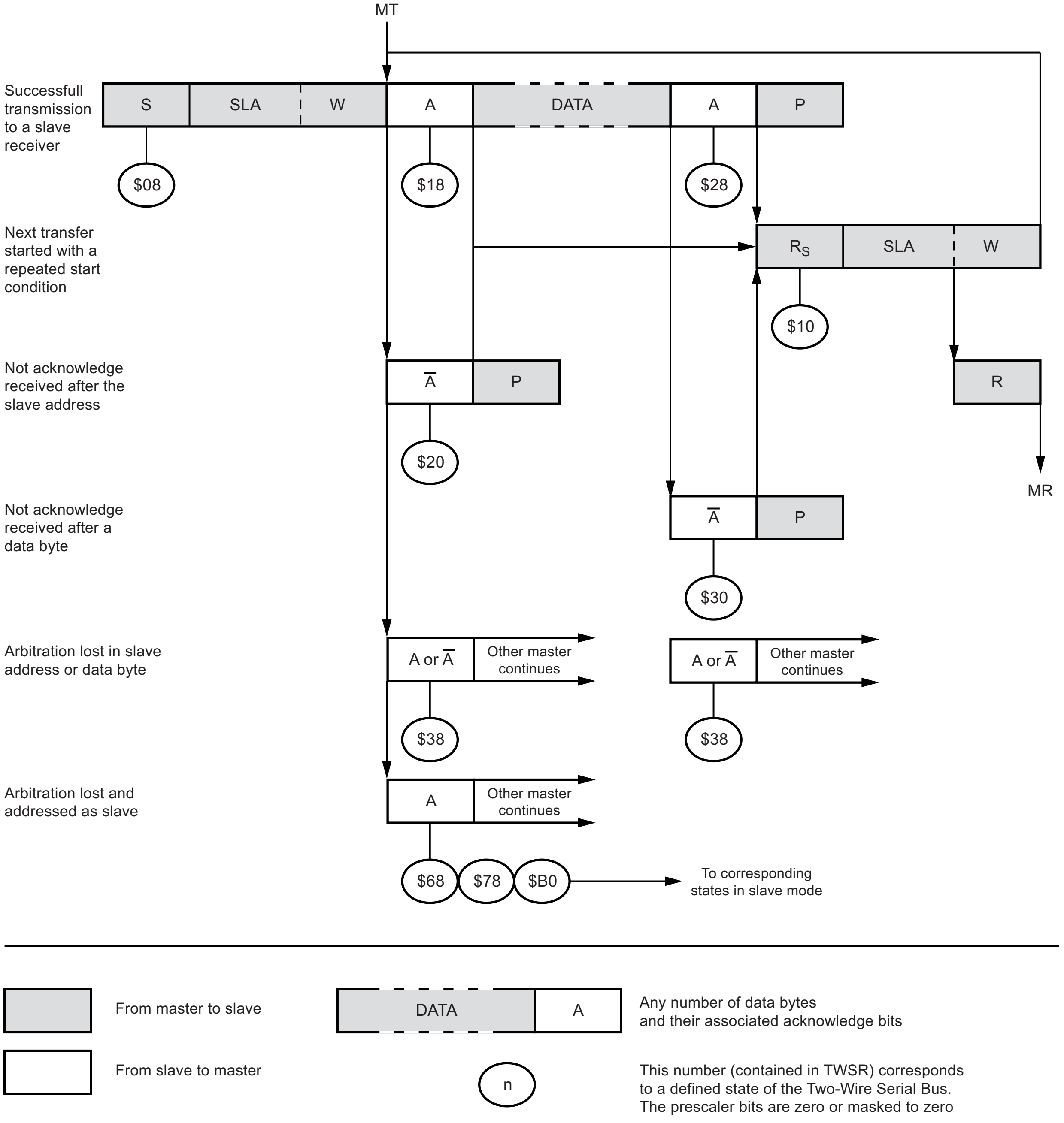
TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	1	0	X	1	0	X

After a repeated START condition (state 0x10) the 2-wire serial interface can access the same slave again, or a new slave without transmitting a STOP condition. Repeated START enables the master to switch between slaves, master transmitter mode and master receiver mode without losing control of the bus.

Table 21-3. Status codes for Master Transmitter Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x08	A START condition has been transmitted	Load SLA+W	0	0	1	X	SLA+W will be transmitted; ACK or NOT ACK will be received
0x10	A repeated START condition has been transmitted	Load SLA+W or	0	0	1	X	SLA+W will be transmitted; ACK or NOT ACK will be received
		Load SLA+R	0	0	1	X	SLA+R will be transmitted; logic will switch to master receiver mode
0x18	SLA+W has been transmitted; ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset
0x20	SLA+W has been transmitted; NOT ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset
0x28	Data byte has been transmitted; ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset
0x30	Data byte has been transmitted; NOT ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset
0x38	Arbitration lost in SLA+W or data bytes	No TWDR action or	0	0	1	X	2-wire serial bus will be released and not addressed slave mode entered
		No TWDR action	1	0	1	X	A START condition will be transmitted when the bus becomes free

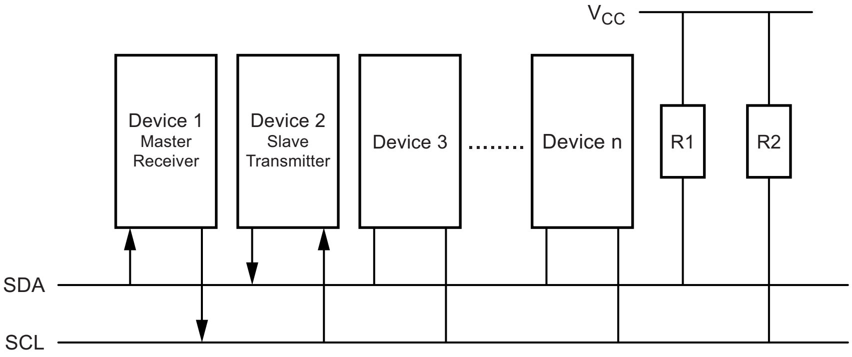
Figure 21-12. Formats and States in the Master Transmitter Mode



21.7.2 Master Receiver Mode

In the master receiver mode, a number of data bytes are received from a slave transmitter (slave see Figure 21-13). In order to enter a master mode, a START condition must be transmitted. The format of the following address packet determines whether master transmitter or master receiver mode is to be entered. If SLA+W is transmitted, MT mode is entered, if SLA+R is transmitted, MR mode is entered. All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 21-13. Data Transfer in Master Receiver Mode



A START condition is sent by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	1	0	X	1	0	X

TWEN must be written to one to enable the 2-wire serial interface, TWSTA must be written to one to transmit a START condition and TWINT must be set to clear the TWINT flag. The TWI will then test the 2-wire serial bus and generate a START condition as soon as the bus becomes free. After a START condition has been transmitted, the TWINT flag is set by hardware, and the status code in TWSR will be 0x08 (See Table 21-3 on page 188). In order to enter MR mode, SLA+R must be transmitted. This is done by writing SLA+R to TWDR. Thereafter the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	0	0	X	1	0	X

When SLA+R have been transmitted and an acknowledgement bit has been received, TWINT is set again and a number of status codes in TWSR are possible. Possible status codes in master mode are 0x38, 0x40, or 0x48. The appropriate action to be taken for each of these status codes is detailed in Table 21-4 on page 191. Received data can be read from the TWDR register when the TWINT flag is set high by hardware. This scheme is repeated until the last byte has been received. After the last byte has been received, the MR should inform the ST by sending a NACK after the last received data byte. The transfer is ended by generating a STOP condition or a repeated START condition.

A STOP condition is generated by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	0	1	X	1	0	X

A REPEATED START condition is generated by writing the following value to TWCR:

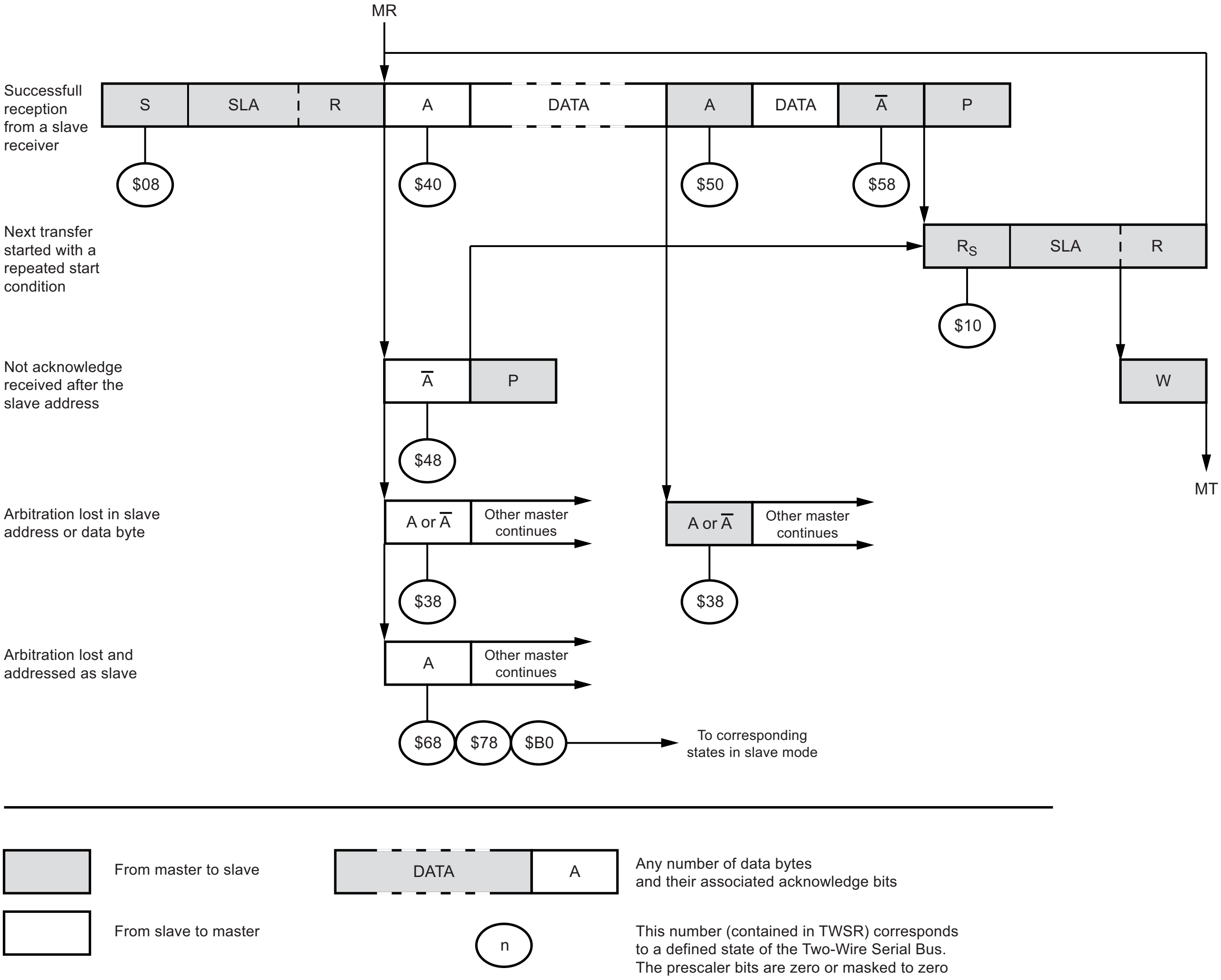
TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
value	1	X	1	0	X	1	0	X

After a repeated START condition (state 0x10) the 2-wire serial interface can access the same slave again, or a new slave without transmitting a STOP condition. Repeated START enables the master to switch between slaves, master transmitter mode and master receiver mode without losing control over the bus.

Table 21-4. Status codes for Master Receiver Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x08	A START condition has been transmitted	Load SLA+R	0	0	1	X	SLA+R will be transmitted ACK or NOT ACK will be received
0x10	A repeated START condition has been transmitted	Load SLA+R or	0	0	1	X	SLA+R will be transmitted ACK or NOT ACK will be received
		Load SLA+W	0	0	1	X	SLA+W will be transmitted logic will switch to master transmitter mode
0x38	Arbitration lost in SLA+R or NOT ACK bit	No TWDR action or	0	0	1	X	2-wire serial bus will be released and not addressed slave mode will be entered
		No TWDR action	1	0	1	X	A START condition will be transmitted when the bus becomes free
0x40	SLA+R has been transmitted; ACK has been received	No TWDR action or	0	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	0	0	1	1	Data byte will be received and ACK will be returned
0x48	SLA+R has been transmitted; NOT ACK has been received	No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset
0x50	Data byte has been received; ACK has been returned	Read data byte or	0	0	1	0	Data byte will be received and NOT ACK will be returned
		Read data byte	0	0	1	1	Data byte will be received and ACK will be returned
0x58	Data byte has been received; NOT ACK has been returned	Read data byte or	1	0	1	X	Repeated START will be transmitted
		Read data byte or	0	1	1	X	STOP condition will be transmitted and TWSTO flag will be reset
		Read data byte	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset

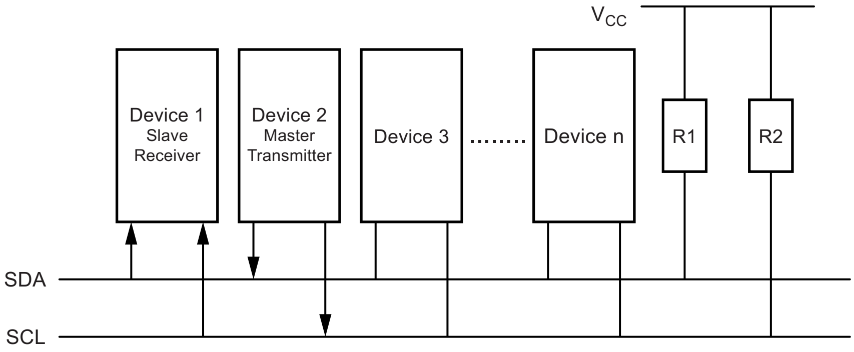
Figure 21-14. Formats and States in the Master Receiver Mode



21.7.3 Slave Receiver Mode

In the slave receiver mode, a number of data bytes are received from a master transmitter (see Figure 21-15). All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 21-15. Data Transfer in Slave Receiver Mode



To initiate the slave receiver mode, TWAR and TWCR must be initialized as follows:

TWAR	**TWA6**	**TWA5**	**TWA4**	**TWA3**	**TWA2**	**TWA1**	**TWA0**	**TWGCE**
value	Device's Own Slave Address							

The upper 7 bits are the address to which the 2-wire serial interface will respond when addressed by a master. If the LSB is set, the TWI will respond to the general call address (0x00), otherwise it will ignore the general call address.

TWCR	**TWINT**	**TWEA**	**TWSTA**	**TWSTO**	**TWWC**	**TWEN**	**–**	**TWIE**
value	0	1	0	0	0	1	0	X

TWEN must be written to one to enable the TWI. The TWEA bit must be written to one to enable the acknowledgement of the device's own slave address or the general call address. TWSTA and TWSTO must be written to zero.

When TWAR and TWCR have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address if enabled) followed by the data direction bit. If the direction bit is "0" (write), the TWI will operate in SR mode, otherwise ST mode is entered. After its own slave address and the write bit have been received, the TWINT flag is set and a valid status code can be read from TWSR. The status code is used to determine the appropriate software action. The appropriate action to be taken for each status code is detailed in Table 21-5 on page 193. The slave receiver mode may also be entered if arbitration is lost while the TWI is in the master mode (see states 0x68 and 0x78).

If the TWEA bit is reset during a transfer, the TWI will return a "not acknowledge" ("1") to SDA after the next received data byte. This can be used to indicate that the slave is not able to receive any more bytes. While TWEA is zero, the TWI does not acknowledge its own slave address. However, the 2-wire serial bus is still monitored and address recognition may resume at any time by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the 2-wire serial bus.

In all sleep modes other than idle mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the 2-wire serial bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock low during the wake up and until the TWINT flag is cleared (by writing it to one). Further data reception will be carried out as normal, with the AVR® clocks running as normal. Observe that if the AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions. Note that the 2-wire serial interface data register – TWDR does not reflect the last byte present on the bus when waking up from these sleep modes.

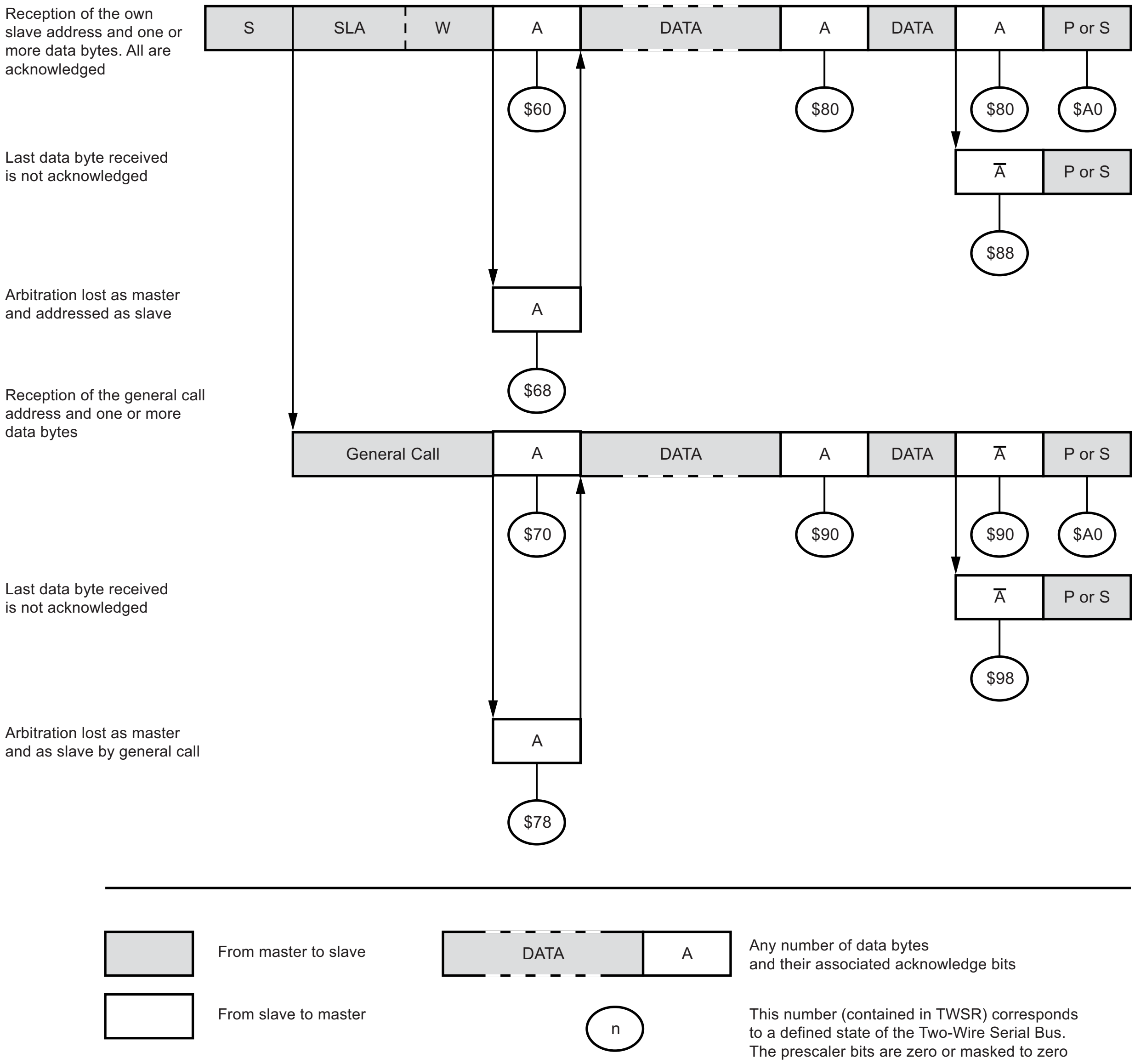
Table 21-5. Status Codes for Slave Receiver Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x60	Own SLA+W has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x68	Arbitration lost in SLA+R/W as master; own SLA+W has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x70	General call address has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x78	Arbitration lost in SLA+R/W as master; general call address has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x80	Previously addressed with own SLA+W; data has been received; ACK has been returned	Read data byte or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		Read data byte	X	0	1	1	Data byte will be received and ACK will be returned

Table 21-5. Status Codes for Slave Receiver Mode (Continued)

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x88	Previously addressed with own SLA+W; data has been received; NOT ACK has been returned	Read data byte or	0	0	1	0	Switched to the not addressed slave mode; no recognition of own SLA or GCA
		Read data byte or	0	0	1	1	Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
		Read data byte or	1	0	1	0	Switched to the not addressed slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		Read data byte	1	0	1	1	Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free
0x90	Previously addressed with general call; data has been received; ACK has been returned	Read data byte or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		Read data byte	X	0	1	1	Data byte will be received and ACK will be returned
0x98	Previously addressed with general call; data has been received; NOT ACK has been returned	Read data byte or	0	0	1	0	Switched to the not addressed slave mode; no recognition of own SLA or GCA
		Read data byte or	0	0	1	1	Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
		Read data byte or	1	0	1	0	Switched to the not addressed slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		Read data byte	1	0	1	1	Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free
0xA0	A STOP condition or repeated START condition has been received while still addressed as slave	No action	0	0	1	0	Switched to the not addressed slave mode; no recognition of own SLA or GCA
			0	0	1	1	Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
			1	0	1	0	Switched to the not addressed slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
			1	0	1	1	Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free

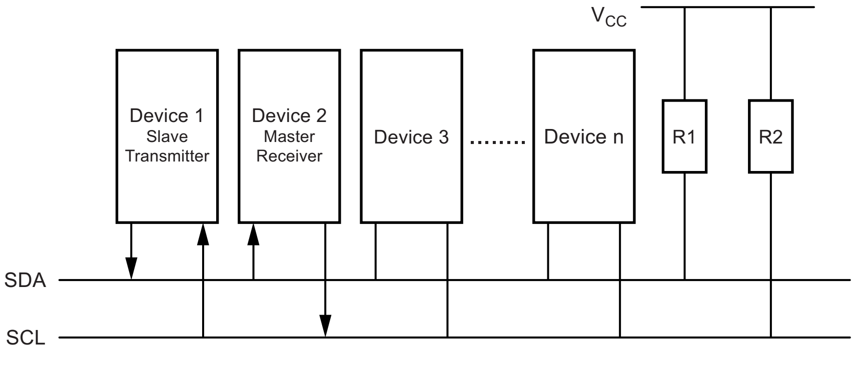
Figure 21-16. Formats and States in the Slave Receiver Mode



21.7.4 Slave Transmitter Mode

In the slave transmitter mode, a number of data bytes are transmitted to a master receiver (see Figure 21-17). All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 21-17. Data Transfer in Slave Transmitter Mode



To initiate the slave transmitter mode, TWAR and TWCR must be initialized as follows:

TWAR	**TWA6**	**TWA5**	**TWA4**	**TWA3**	**TWA2**	**TWA1**	**TWA0**	**TWGCE**
value	Device’s Own Slave Address							

The upper seven bits are the address to which the 2-wire serial interface will respond when addressed by a master. If the LSB is set, the TWI will respond to the general call address (0x00), otherwise it will ignore the general call address.

TWCR	**TWINT**	**TWEA**	**TWSTA**	**TWSTO**	**TWWC**	**TWEN**	**–**	**TWIE**
value	0	1	0	0	0	1	0	X

TWEN must be written to one to enable the TWI. The TWEA bit must be written to one to enable the acknowledgement of the device’s own slave address or the general call address. TWSTA and TWSTO must be written to zero.

When TWAR and TWCR have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address if enabled) followed by the data direction bit. If the direction bit is “1” (read), the TWI will operate in ST mode, otherwise SR mode is entered. After its own slave address and the write bit have been received, the TWINT flag is set and a valid status code can be read from TWSR. The status code is used to determine the appropriate software action. The appropriate action to be taken for each status code is detailed in Table 21-6 on page 197. The slave transmitter mode may also be entered if arbitration is lost while the TWI is in the master mode (see state 0xB0).

If the TWEA bit is written to zero during a transfer, the TWI will transmit the last byte of the transfer. State 0xC0 or state 0xC8 will be entered, depending on whether the master receiver transmits a NACK or ACK after the final byte. The TWI is switched to the not addressed slave mode, and will ignore the master if it continues the transfer. Thus the master receiver receives all “1” as serial data. State 0xC8 is entered if the master demands additional data bytes (by transmitting ACK), even though the slave has transmitted the last byte (TWEA zero and expecting NACK from the master).

While TWEA is zero, the TWI does not respond to its own slave address. However, the 2-wire serial bus is still monitored and address recognition may resume at any time by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the 2-wire serial bus.

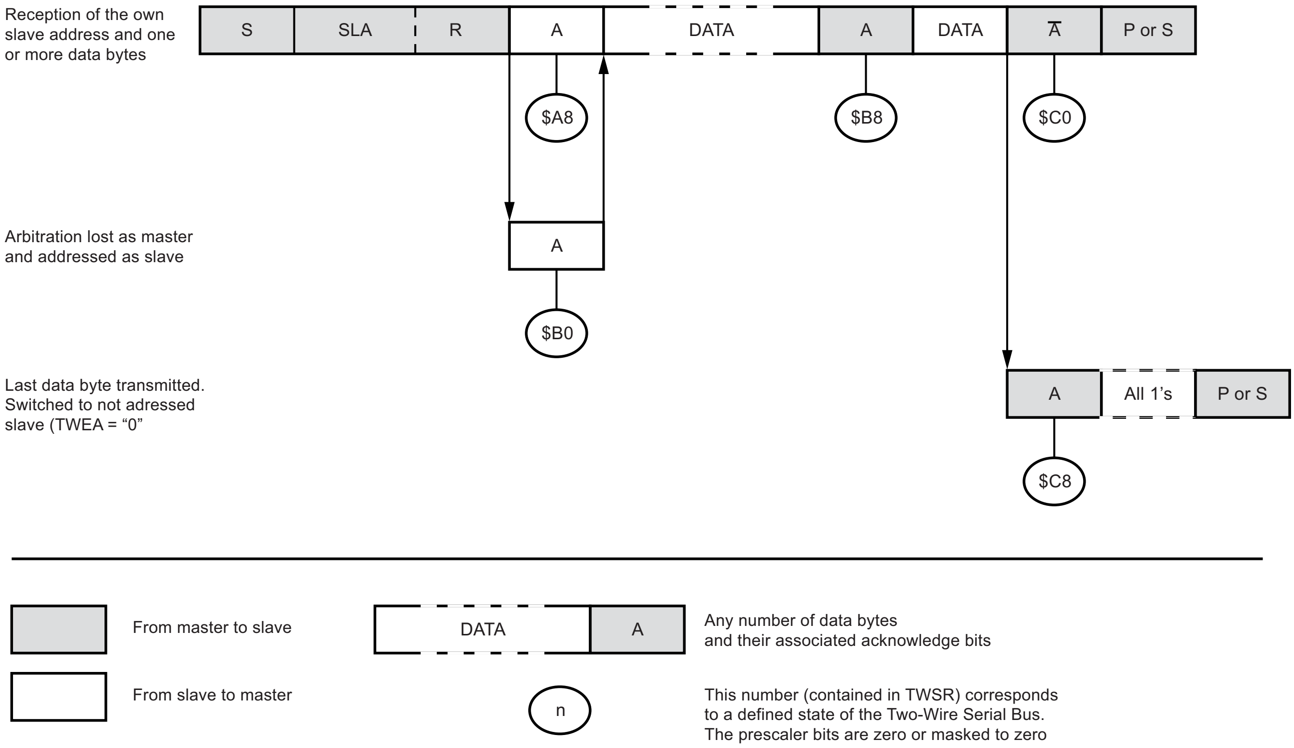
In all sleep modes other than Idle mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the 2-wire serial bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock will low during the wake up and until the TWINT flag is cleared (by writing it to one). Further data transmission will be carried out as normal, with the AVR® clocks running as normal. Observe that if the AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions.

Note that the 2-wire serial interface data register – TWDR does not reflect the last byte present on the bus when waking up from these sleep modes.

Table 21-6. Status Codes for Slave Transmitter Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0xA8	Own SLA+R has been received; ACK has been returned	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received
		Load data byte	X	0	1	1	Data byte will be transmitted and ACK should be received
0xB0	Arbitration lost in SLA+R/W as master; own SLA+R has been received; ACK has been returned	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received
		Load data byte	X	0	1	1	Data byte will be transmitted and ACK should be received
0xB8	Data byte in TWDR has been transmitted; ACK has been received	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received
		Load data byte	X	0	1	1	Data byte will be transmitted and ACK should be received
0xC0	Data byte in TWDR has been transmitted; NOT ACK has been received	No TWDR action or	0	0	1	0	Switched to the not addressed slave mode; no recognition of own SLA or GCA
		No TWDR action or	0	0	1	1	Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
		No TWDR action or	1	0	1	0	Switched to the not addressed slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		No TWDR action	1	0	1	1	Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free
0xC8	Last data byte in TWDR has been transmitted (TWEA = “0”); ACK has been received	No TWDR action or	0	0	1	0	Switched to the not addressed slave mode; no recognition of own SLA or GCA
		No TWDR action or	0	0	1	1	Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
		No TWDR action or	1	0	1	0	Switched to the not addressed slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		No TWDR action	1	0	1	1	Switched to the not addressed slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free

Figure 21-18. Formats and States in the Slave Transmitter Mode



21.7.5 Miscellaneous States

There are two status codes that do not correspond to a defined TWI state, see Table 21-7.

Status 0xF8 indicates that no relevant information is available because the TWINT flag is not set. This occurs between other states, and when the TWI is not involved in a serial transfer.

Status 0x00 indicates that a bus error has occurred during a 2-wire serial bus transfer. A bus error occurs when a START or STOP condition occurs at an illegal position in the format frame. Examples of such illegal positions are during the serial transfer of an address byte, a data byte, or an acknowledge bit. When a bus error occurs, TWINT is set. To recover from a bus error, the TWSTO flag must set and TWINT must be cleared by writing a logic one to it. This causes the TWI to enter the not addressed Slave mode and to clear the TWSTO flag (no other bits in TWCR are affected). The SDA and SCL lines are released, and no STOP condition is transmitted.

Table 21-7. Miscellaneous States

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0xF8	No relevant state information available; TWINT = "0"	No TWDR action	No TWCR action				Wait or proceed current transfer
0x00	Bus error due to an illegal START or STOP condition	No TWDR action	0	1	1	X	Only the internal hardware is affected, no STOP condition is sent on the bus. In all cases, the bus is released and TWSTO is cleared.

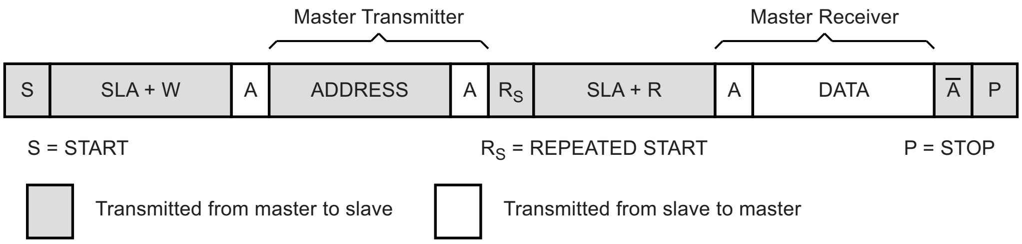
21.7.6 Combining Several TWI Modes

In some cases, several TWI modes must be combined in order to complete the desired action. Consider for example reading data from a serial EEPROM. Typically, such a transfer involves the following steps:

1. The transfer must be initiated.
2. The EEPROM must be instructed what location should be read.
3. The reading must be performed.
4. The transfer must be finished.

Note that data is transmitted both from master to slave and vice versa. The master must instruct the slave what location it wants to read, requiring the use of the MT mode. Subsequently, data must be read from the slave, implying the use of the MR mode. Thus, the transfer direction must be changed. The master must keep control of the bus during all these steps, and the steps should be carried out as an atomical operation. If this principle is violated in a multi master system, another master can alter the data pointer in the EEPROM between steps 2 and 3, and the master will read the wrong data location. Such a change in transfer direction is accomplished by transmitting a REPEATED START between the transmission of the address byte and reception of the data. After a REPEATED START, the master keeps ownership of the bus. The following figure shows the flow in this transfer.

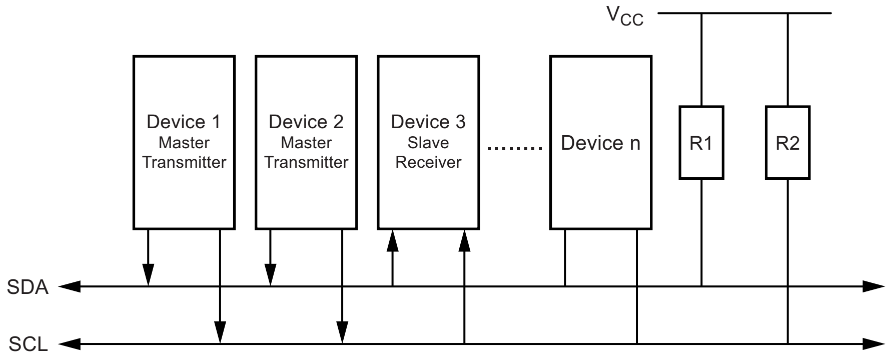
Figure 21-19. Combining Several TWI Modes to Access a Serial EEPROM



21.8 Multi-master Systems and Arbitration

If multiple masters are connected to the same bus, transmissions may be initiated simultaneously by one or more of them. The TWI standard ensures that such situations are handled in such a way that one of the masters will be allowed to proceed with the transfer, and that no data will be lost in the process. An example of an arbitration situation is depicted below, where two masters are trying to transmit data to a slave receiver.

Figure 21-20. An Arbitration Example



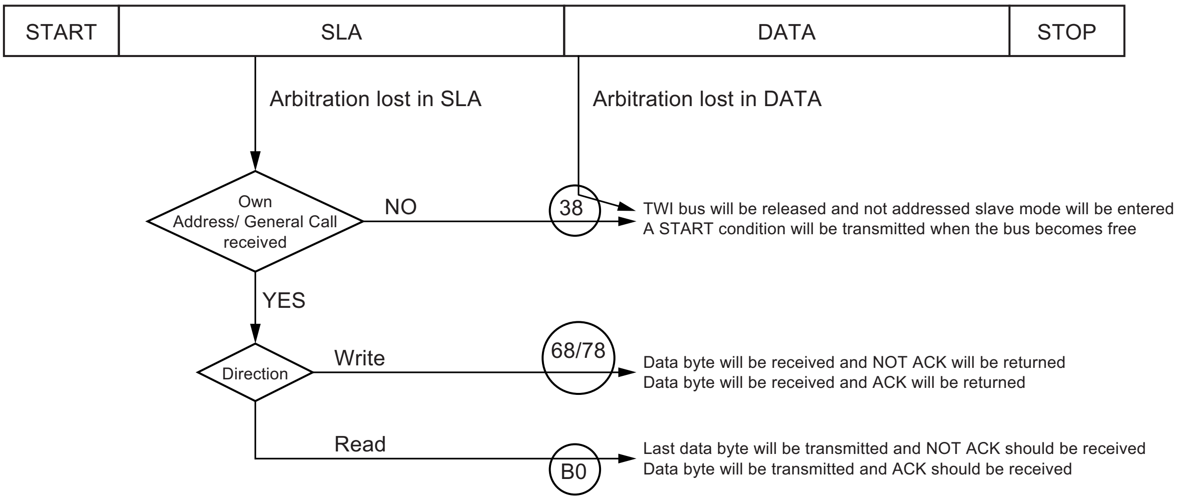
Several different scenarios may arise during arbitration, as described below:

- Two or more masters are performing identical communication with the same slave. In this case, neither the slave nor any of the masters will know about the bus contention.
- Two or more masters are accessing the same slave with different data or direction bit. In this case, arbitration will occur, either in the READ/WRITE bit or in the data bits. The masters trying to output a one on SDA while another master outputs a zero will lose the arbitration. Losing masters will switch to not addressed slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.

- Two or more masters are accessing different slaves. In this case, arbitration will occur in the SLA bits. Masters trying to output a one on SDA while another master outputs a zero will lose the arbitration. Masters losing arbitration in SLA will switch to slave mode to check if they are being addressed by the winning master. If addressed, they will switch to SR or ST mode, depending on the value of the READ/WRITE bit. If they are not being addressed, they will switch to not addressed slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.

This is summarized in Figure 21-21. Possible status values are given in circles.

Figure 21-21. Possible Status Codes Caused by Arbitration



21.9 Register Description

21.9.1 TWBR – TWI Bit Rate Register

Bit	7	6	5	4	3	2	1	0	
(0xB8)	**TWBR7**	**TWBR6**	**TWBR5**	**TWBR4**	**TWBR3**	**TWBR2**	**TWBR1**	**TWBR0**	**TWBR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – TWI Bit Rate Register**

TWBR selects the division factor for the bit rate generator. The bit rate generator is a frequency divider which generates the SCL clock frequency in the master modes. See Section 21.5.2 “Bit Rate Generator Unit” on page 182 for calculating bit rates.

21.9.2 TWCR – TWI Control Register

Bit	7	6	5	4	3	2	1	0	
(0xBC)	**TWINT**	**TWEA**	**TWSTA**	**TWSTO**	**TWWC**	**TWEN**	**–**	**TWIE**	**TWCR**
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The TWCR is used to control the operation of the TWI. It is used to enable the TWI, to initiate a master access by applying a START condition to the bus, to generate a receiver acknowledge, to generate a stop condition, and to control halting of the bus while the data to be written to the bus are written to the TWDR. It also indicates a write collision if data is attempted written to TWDR while the register is inaccessible.

- **Bit 7 – TWINT: TWI Interrupt Flag**

This bit is set by hardware when the TWI has finished its current job and expects application software response. If the I-bit in SREG and TWIE in TWCR are set, the MCU will jump to the TWI interrupt vector. While the TWINT flag is set, the SCL low period is stretched. The TWINT flag must be cleared by software by writing a logic one to it. Note that this flag is not automatically cleared by hardware when executing the interrupt routine. Also note that clearing this flag starts the operation of the TWI, so all accesses to the TWI address register (TWAR), TWI status register (TWSR), and TWI data register (TWDR) must be complete before clearing this flag.

- **Bit 6 – TWEA: TWI Enable Acknowledge Bit**

The TWEA bit controls the generation of the acknowledge pulse. If the TWEA bit is written to one, the ACK pulse is generated on the TWI bus if the following conditions are met:

1. The device’s own slave address has been received.
2. A general call has been received, while the TWGCE bit in the TWAR is set.
3. A data byte has been received in master receiver or slave receiver mode.

By writing the TWEA bit to zero, the device can be virtually disconnected from the 2-wire serial bus temporarily. Address recognition can then be resumed by writing the TWEA bit to one again.

- **Bit 5 – TWSTA: TWI START Condition Bit**

The application writes the TWSTA bit to one when it desires to become a master on the 2-wire serial bus. The TWI hardware checks if the bus is available, and generates a START condition on the bus if it is free. However, if the bus is not free, the TWI waits until a STOP condition is detected, and then generates a new START condition to claim the bus master status. TWSTA must be cleared by software when the START condition has been transmitted.

- **Bit 4 – TWSTO: TWI STOP Condition Bit**

Writing the TWSTO bit to one in master mode will generate a STOP condition on the 2-wire serial bus. When the STOP condition is executed on the bus, the TWSTO bit is cleared automatically. In slave mode, setting the TWSTO bit can be used to recover from an error condition. This will not generate a STOP condition, but the TWI returns to a well-defined unaddressed slave mode and releases the SCL and SDA lines to a high impedance state.

- **Bit 3 – TWWC: TWI Write Collision Flag**

The TWWC bit is set when attempting to write to the TWI data register – TWDR when TWINT is low. This flag is cleared by writing the TWDR register when TWINT is high.

- **Bit 2 – TWEN: TWI Enable Bit**

The TWEN bit enables TWI operation and activates the TWI interface. When TWEN is written to one, the TWI takes control over the I/O pins connected to the SCL and SDA pins, enabling the slew-rate limiters and spike filters. If this bit is written to zero, the TWI is switched off and all TWI transmissions are terminated, regardless of any ongoing operation.

- **Bit 1 – Reserved**

This bit is a reserved bit and will always read as zero.

- **Bit 0 – TWIE: TWI Interrupt Enable**

When this bit is written to one, and the I-bit in SREG is set, the TWI interrupt request will be activated for as long as the TWINT flag is high.

21.9.3 TWSR – TWI Status Register

Bit	7	6	5	4	3	2	1	0	
(0xB9)	**TWS7**	**TWS6**	**TWS5**	**TWS4**	**TWS3**	**–**	**TWPS1**	**TWPS0**	**TWSR**
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

- **Bits 7:3 – TWS: TWI Status**

These 5 bits reflect the status of the TWI logic and the 2-wire serial bus. The different status codes are described Section 21.7 “Transmission Modes” on page 186. Note that the value read from TWSR contains both the 5-bit status value and the 2-bit prescaler value. The application designer should mask the prescaler bits to zero when checking the status bits. This makes status checking independent of prescaler setting. This approach is used in this datasheet, unless otherwise noted.

- **Bit 2 – Reserved**

This bit is reserved and will always read as zero.

- **Bits 1:0 – TWPS: TWI Prescaler Bits**

These bits can be read and written, and control the bit rate prescaler.

Table 21-8. TWI Bit Rate Prescaler

TWPS1	TWPS0	Prescaler Value
0	0	1
0	1	4
1	0	16
1	1	64

To calculate bit rates, see Section 21.5.2 “Bit Rate Generator Unit” on page 182. The value of TWPS1..0 is used in the equation.

21.9.4 TWDR – TWI Data Register

Bit	7	6	5	4	3	2	1	0	
(0xBB)	**TWD7**	**TWD6**	**TWD5**	**TWD4**	**TWD3**	**TWD2**	**TWD1**	**TWD0**	**TWDR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

In transmit mode, TWDR contains the next byte to be transmitted. In receive mode, the TWDR contains the last byte received. It is writable while the TWI is not in the process of shifting a byte. This occurs when the TWI interrupt flag (TWINT) is set by hardware. Note that the data register cannot be initialized by the user before the first interrupt occurs. The data in TWDR remains stable as long as TWINT is set. While data is shifted out, data on the bus is simultaneously shifted in. TWDR always contains the last byte present on the bus, except after a wake up from a sleep mode by the TWI interrupt. In this case, the contents of TWDR is undefined. In the case of a lost bus arbitration, no data is lost in the transition from master to slave. Handling of the ACK bit is controlled automatically by the TWI logic, the CPU cannot access the ACK bit directly.

- **Bits 7:0 – TWD: TWI Data Register**

These eight bits constitute the next data byte to be transmitted, or the latest data byte received on the 2-wire serial bus.

21.9.5 TWAR – TWI (Slave) Address Register

Bit	7	6	5	4	3	2	1	0	
(0xBA)	**TWA6**	**TWA5**	**TWA4**	**TWA3**	**TWA2**	**TWA1**	**TWA0**	**TWGCE**	**TWAR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

The TWAR should be loaded with the 7-bit slave address (in the seven most significant bits of TWAR) to which the TWI will respond when programmed as a slave transmitter or receiver, and not needed in the master modes. In multi master systems, TWAR must be set in masters which can be addressed as Slaves by other masters.

The LSB of TWAR is used to enable recognition of the general call address (0x00). There is an associated address comparator that looks for the slave address (or general call address if enabled) in the received serial address. If a match is found, an interrupt request is generated.

- **Bits 7:1 – TWA: TWI (Slave) Address Register**

These seven bits constitute the slave address of the TWI unit.

- **Bit 0 – TWGCE: TWI General Call Recognition Enable Bit**

If set, this bit enables the recognition of a general call given over the 2-wire serial bus.

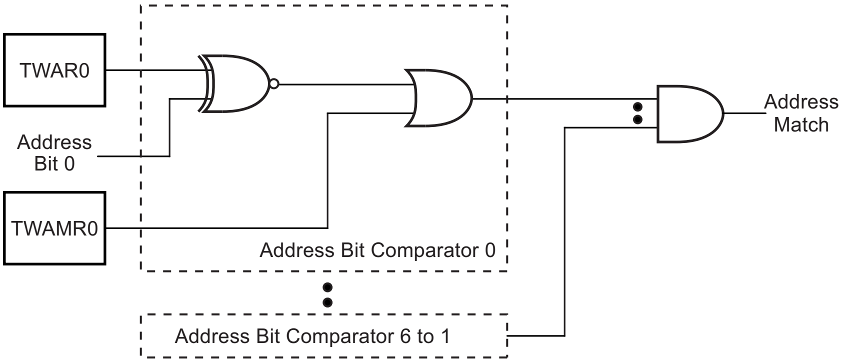
21.9.6 TWAMR – TWI (Slave) Address Mask Register

Bit	7	6	5	4	3	2	1	0	
(0xBD)	**TWAM[6:0]**							**–**	**TWAMR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:1 – TWAM: TWI Address Mask**

The TWAMR can be loaded with a 7-bit Slave Address mask. Each of the bits in TWAMR can mask (disable) the corresponding address bit in the TWI Address Register (TWAR). If the mask bit is set to one then the address match logic ignores the compare between the incoming address bit and the corresponding bit in TWAR. Figure 21-22 shows the address match logic in detail.

Figure 21-22. TWI Address Match Logic, Block Diagram



- **Bit 0 – Reserved**

This bit is reserved and will always read as zero.

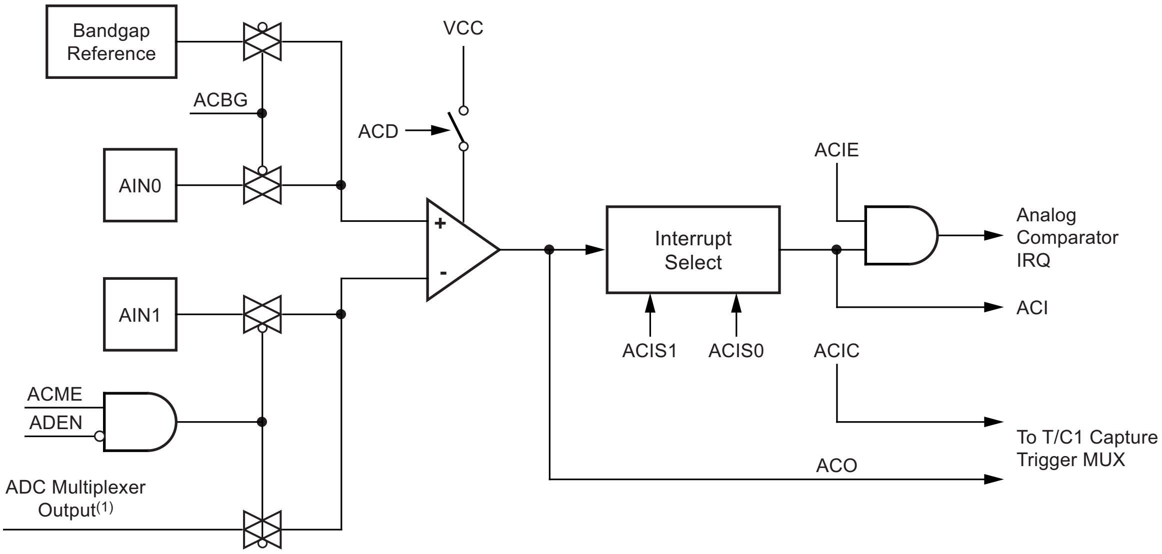
22. AC - Analog Comparator

22.1 Overview

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 22-1.

The power reduction ADC bit, PRADC, in Section 10.12.3 "PRR0 – Power Reduction Register 0" on page 39 must be disabled by writing a logical zero to be able to use the ADC input MUX.

Figure 22-1. Analog Comparator Block Diagram[2]



Notes: 1. See Table 22-1 on page 205.

2. Refer to Figure 1-1 on page 3 and Table 14-5 on page 64 for analog comparator pin placement.

22.2 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 22-1 on page 205. If ACME is cleared or ADEN is set, AIN1 is applied to the negative input to the analog comparator.

Table 22-1. Analog Comparator Mulitiplexed 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

22.3 Register Description

22.3.1 ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
(0x7B)	**–**	**ACME**	**–**	**–**	**-**	**ADTS2**	**ADTS1**	**ADTS0**	**ADCSRB**
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 6 – ACME: Analog Comparator Multiplexer Enable**

When this bit is written logic one and the ADC is switched off (ADEN in ADCSRA is zero), the ADC multiplexer selects the negative input to the analog comparator. When this bit is written logic zero, AIN1 is applied to the negative input of the analog comparator. For a detailed description of this bit, see Section 22.2 “Analog Comparator Multiplexed Input” on page 204.

22.3.2 ACSR – Analog Comparator Control and Status Register

Bit	7	6	5	4	3	2	1	0	
0x30 (0x50)	**ACD**	**ACBG**	**ACO**	**ACI**	**ACIE**	**ACIC**	**ACIS1**	**ACIS0**	**ACSR**
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

- **Bit 7 – ACD: Analog Comparator Disable**

When this bit is written logic one, the power to the analog comparator is switched off. This bit can be set at any time to turn off the analog comparator. This will reduce power consumption in active and idle mode. When changing the ACD bit, the analog comparator interrupt must be disabled by clearing the ACIE bit in ACSR. Otherwise an interrupt can occur when the bit is changed.

- **Bit 6 – ACBG: Analog Comparator Bandgap Select**

When this bit is set, a fixed 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. When bandgap reference is used as input to the analog comparator, it will take a certain time for the voltage to stabilize. If not stabilized, the first conversion may give wrong value. See Section 11.2 “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 22-2.

Table 22-2. ACIS1/ACIS0 Settings

ACIS1	ACIS0	Interrupt Mode
0	0	Comparator interrupt on output toggle.
0	1	Reserved
1	0	Comparator interrupt on falling output edge.
1	1	Comparator interrupt on rising output edge.

When changing the ACIS1/ACIS0 bits, the analog comparator Interrupt must be disabled by clearing its interrupt enable bit in the ACSR register. Otherwise an interrupt can occur when the bits are changed.

22.3.3 DIDR1 – Digital Input Disable Register 1

Bit	7	6	5	4	3	2	1	0	
(0x7F)	–	–	–	–	–	–	**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.

23. ADC - Analog-to-digital Converter

23.1 Features

- 10-bit resolution
- 0.5 LSB integral non-linearity
- ±2 LSB absolute accuracy
- 65 - 260µs conversion time
- Up to 15kSPS at maximum resolution
- 8 multiplexed single ended input channels
- Differential mode with selectable gain at 1x, 10x or 200x
- Optional left adjustment for ADC result readout
- 0 - V_{CC} ADC input voltage range
- 2.7 - V_{CC} differential ADC voltage range
- Selectable 2.56V or 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

23.2 Overview

The Atmel® ATmega164P-B/324P-B/644P-B features a 10-bit successive approximation ADC. The ADC is connected to an 8-channel analog multiplexer which allows 8 single-ended voltage inputs constructed from the pins of Port A.
The single-ended voltage inputs refer to 0V (GND).

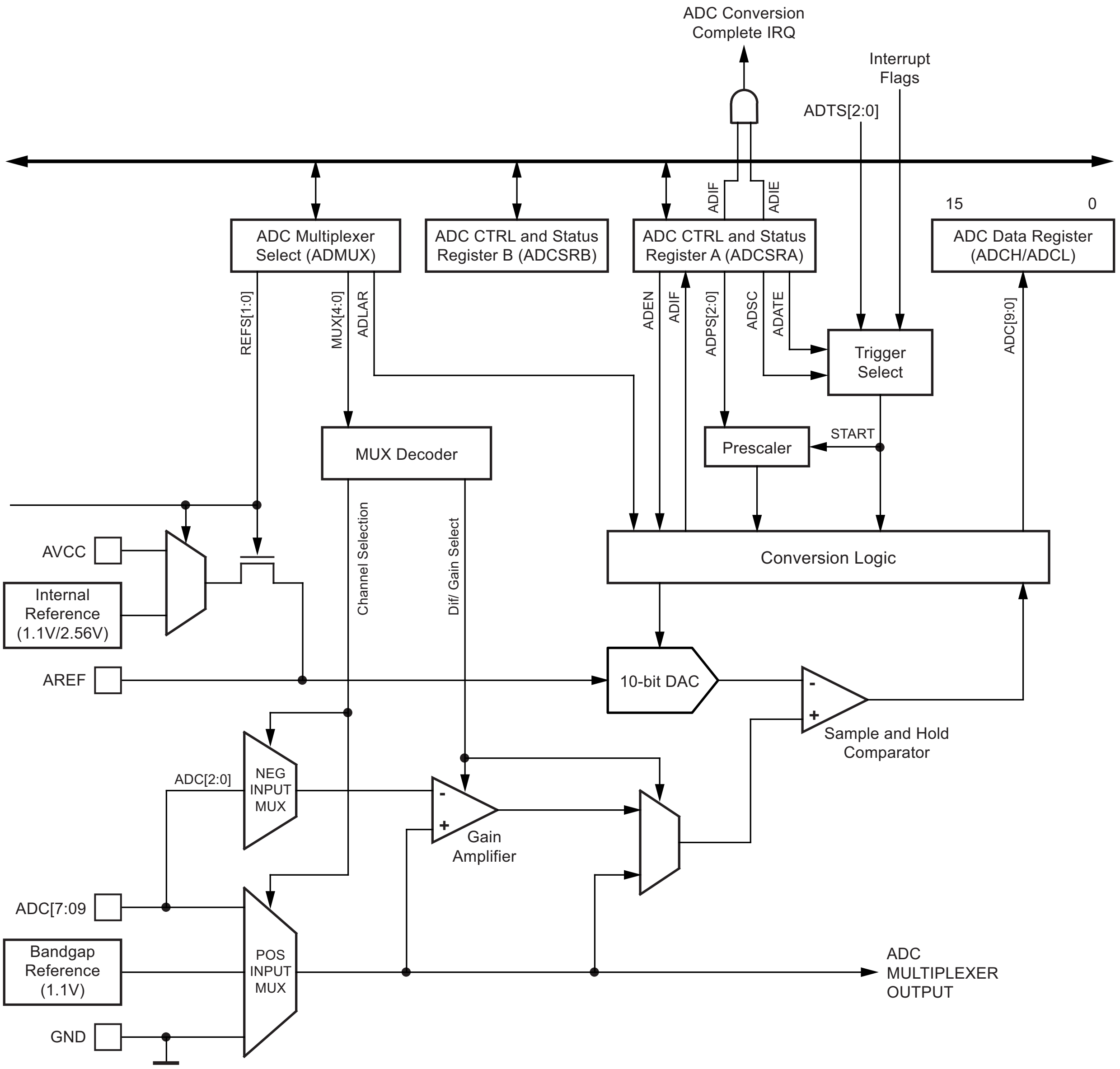
The device also supports 16 differential voltage input combinations. Two of the differential inputs (ADC1, ADC0 and ADC3, ADC2) are equipped with a programmable gain stage. This provides amplification steps of 0 dB (1x), 20 dB (10x), or 46 dB (200x) on the differential input voltage before the A/D conversion. Seven differential analog input channels share a common negative terminal (ADC1), while any other ADC input can be selected as the positive input terminal. If 1x or 10x gain is used, 8-bit resolution can be expected. If 200x gain is used, 6-bit resolution can be expected. Note that internal references of 1.1V should not be used on 10x and 200x gain.

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 23-1 on page 208.

The ADC has a separate analog supply voltage pin, AVCC. AVCC must not differ more than ±0.3 V from V_{CC}. See the Section 23.7 “ADC Noise Canceler” on page 214 on how to connect this pin.

Internal reference voltages of nominally 1.1V, 2.56V or AVCC are provided On-chip. The voltage reference may be externally decoupled at the AREF pin by a capacitor for better noise performance.

Figure 23-1. Analog-to-digital Converter Block Schematic



23.3 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 2.56V 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 and differential gain are 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. A selection of ADC input pins can be selected as positive and negative inputs to the differential gain amplifier.

If differential channels are selected, the differential gain stage amplifies the voltage difference between the selected input channel pair by the selected gain factor. This amplified value then becomes the analog input to the ADC. If single ended channels are used, the gain amplifier is bypassed altogether.

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.

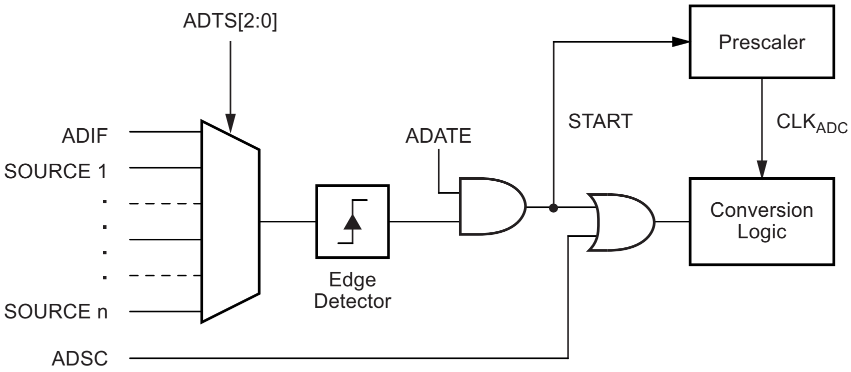
23.4 Starting a Conversion

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 23-2. ADC Auto Trigger Logic

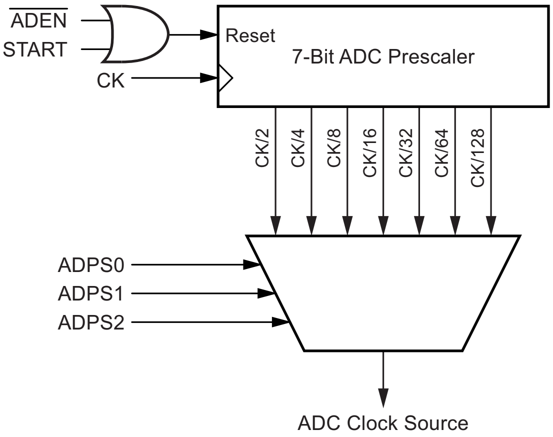


Using the ADC interrupt flag as a trigger source makes the ADC start a new conversion as soon as the ongoing conversion has finished. The ADC then operates in free running mode, constantly sampling and updating the ADC data register. The first conversion must be started by writing a logical one to the ADSC bit in ADCSRA. In this mode the ADC will perform successive conversions independently of whether the ADC interrupt flag, ADIF is cleared or not.

If auto triggering is enabled, single conversions can be started by writing ADSC in ADCSRA to one. ADSC can also be used to determine if a conversion is in progress. The ADSC bit will be read as one during a conversion, independently of how the conversion was started.

23.5 Prescaling and Conversion Timing

Figure 23-3. ADC Prescaler



By default, the successive approximation circuitry requires an input clock frequency between 50kHz and 200kHz to get maximum resolution. If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200kHz to get a higher sample rate.

The ADC module contains a prescaler, which generates an acceptable ADC clock frequency from any CPU frequency above 100kHz. The prescaling is set by the ADPS bits in ADCSRA. The prescaler starts counting from the moment the ADC is switched on by setting the ADEN bit in ADCSRA. The prescaler keeps running for as long as the ADEN bit is set, and is continuously reset when ADEN is low.

When initiating a single ended conversion by setting the ADSC bit in ADCSRA, the conversion starts at the following rising edge of the ADC clock cycle. See Section 23.5.1 "Differential Gain Channels" on page 213 for details on differential conversion timing.

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.

When the bandgap reference voltage is used as input to the ADC, it will take a certain time for the voltage to stabilize. If not stabilized, the first value read after the first conversion may be wrong.

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 a 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 2 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 trigging 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 23-1 on page 212.

Figure 23-4. ADC Timing Diagram, First Conversion (Single Conversion Mode)

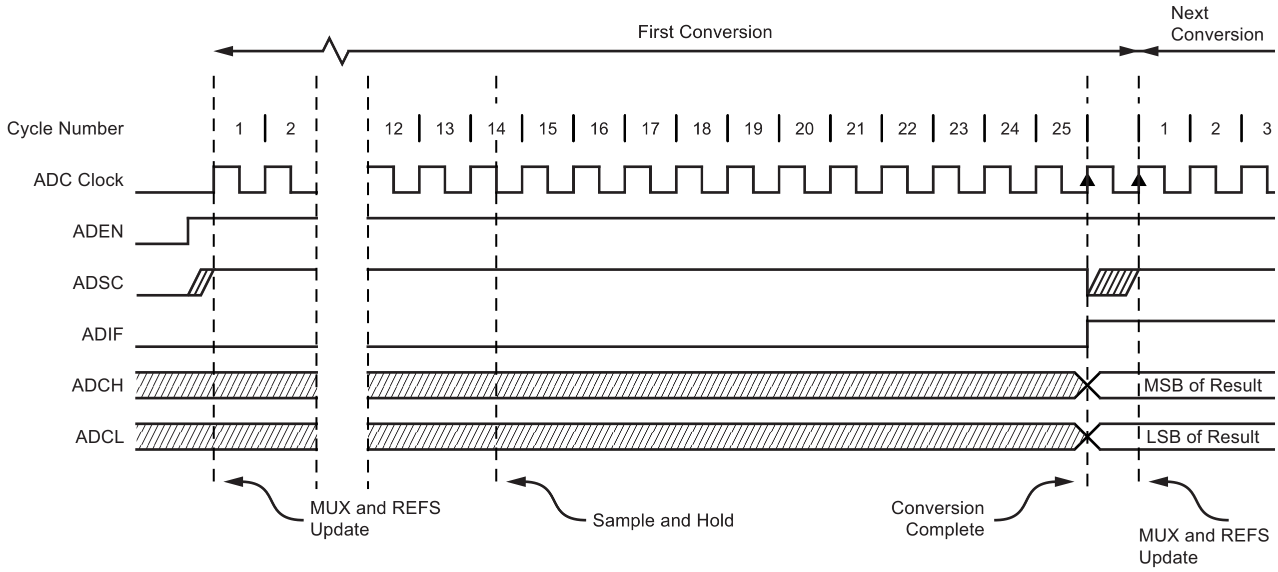


Figure 23-5. ADC Timing Diagram, Single Conversion

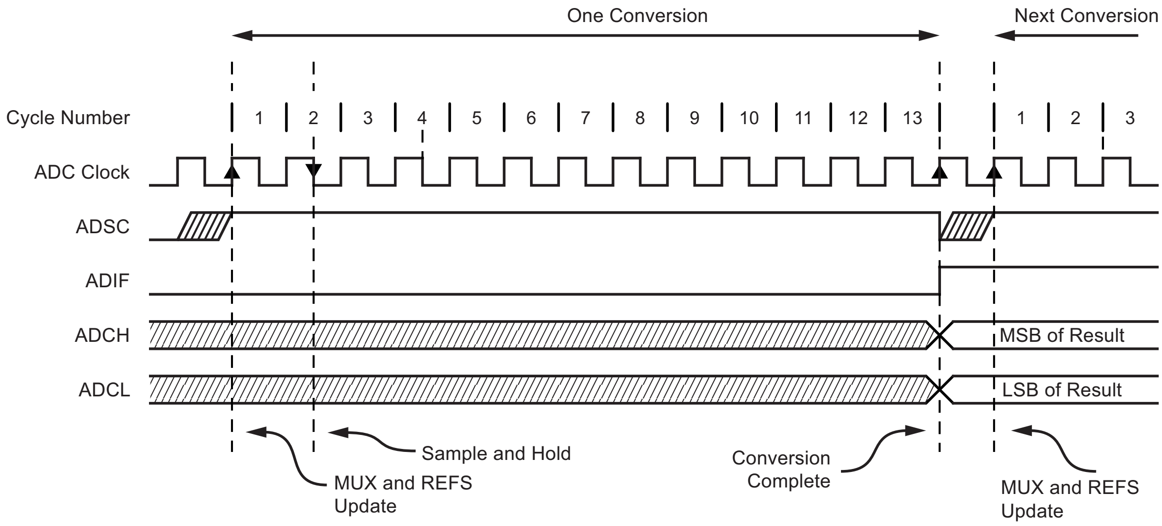


Figure 23-6. ADC Timing Diagram, Auto Triggered Conversion

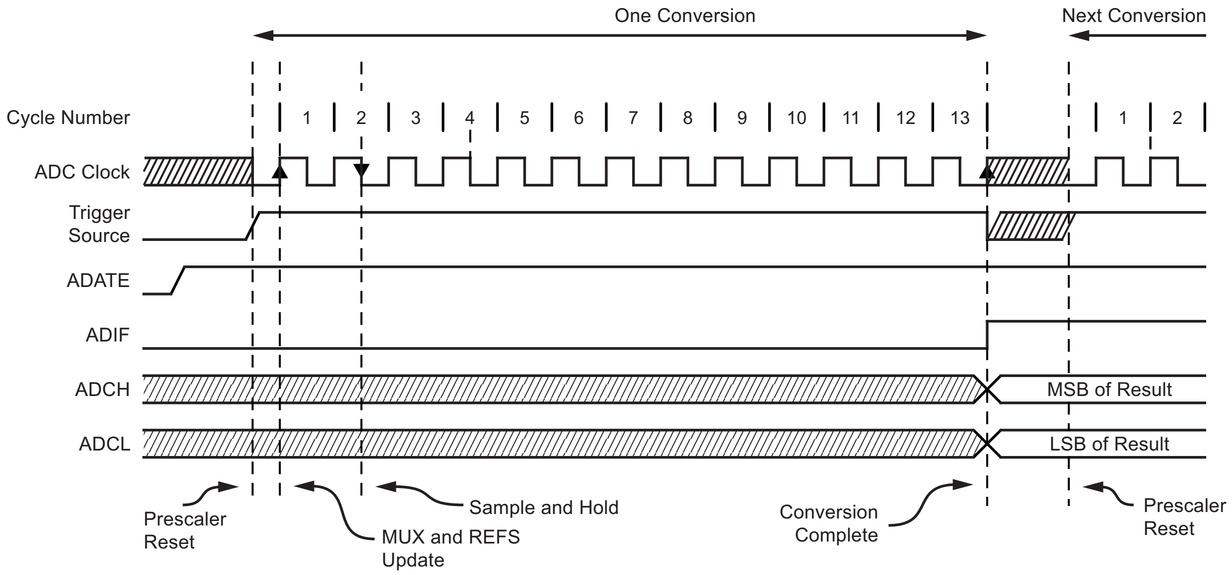


Figure 23-7. ADC Timing Diagram, Free Running Conversion

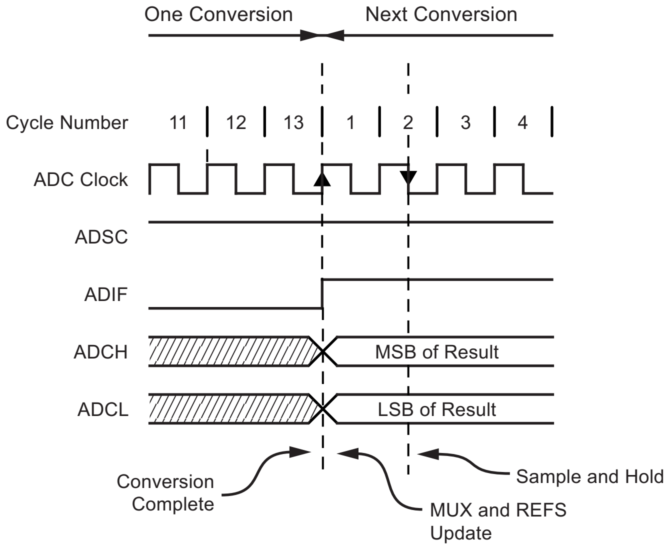


Table 23-1. ADC Conversion Time

Condition	Sample and Hold (Cycles from Start of Conversion)	Conversion Time (Cycles)
First conversion	14.5	25
Normal conversions, single ended	1.5	13
Auto triggered conversions	2	13.5
Normal conversions, differential	1.5/2.5	13/14

23.5.1 Differential Gain Channels

When using differential gain channels, certain aspects of the conversion need to be taken into consideration. Note that the differential channels should not be used with an AREF < 2V.

Differential conversions are synchronized to the internal clock CK_{ADC2} equal to half the ADC clock. This synchronization is done automatically by the ADC interface in such a way that the sample-and-hold occurs at a specific phase of CK_{ADC2}. A conversion initiated by the user (i.e., all single conversions, and the first free running conversion) when CK_{ADC2} is low will take the same amount of time as a single ended conversion (13 ADC clock cycles from the next prescaled clock cycle). A conversion initiated by the user when CK_{ADC2} is high will take 14 ADC clock cycles due to the synchronization mechanism. In free running mode, a new conversion is initiated immediately after the previous conversion completes, and since CK_{ADC2} is high at this time, all automatically started (i.e., all but the first) free running conversions will take 14 ADC clock cycles.

The gain stage is optimized for a bandwidth of 4kHz at all gain settings. Higher frequencies may be subjected to non-linear amplification. An external low-pass filter should be used if the input signal contains higher frequency components than the gain stage bandwidth. Note that the ADC clock frequency is independent of the gain stage bandwidth limitation. For example, the ADC clock period may be 6µs, allowing a channel to be sampled at 12kSPS, regardless of the bandwidth of this channel.

If differential gain channels are used and conversions are started by auto triggering, the ADC must be switched off between conversions. When auto triggering is used, the ADC prescaler is reset before the conversion is started. Since the gain stage is dependent of a stable ADC clock prior to the conversion, this conversion will not be valid. By disabling and then re-enabling the ADC between each conversion (writing ADEN in ADCSRA to “0” then to “1”), only extended conversions are performed. The result from the extended conversions will be valid. See Section 23.5 “Prescaling and Conversion Timing” on page 210 for timing details.

23.6 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.

Special care should be taken when changing differential channels. Once a differential channel has been selected, the gain stage may take as much as 125µs to stabilize to the new value. Thus conversions should not be started within the first 125µs after selecting a new differential channel. Alternatively, conversion results obtained within this period should be discarded.

The same settling time should be observed for the first differential conversion after changing ADC reference (by changing the REFS1:0 bits in ADMUX).

23.6.1 ADC Input Channels

When changing channel selections, the user should observe the following guidelines to ensure that the correct channel is selected:

In single conversion mode, always select the channel before starting the conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the conversion to complete before changing the channel selection.

In free running mode, always select the channel before starting the first conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the first conversion to complete, and then change the channel selection. Since the next conversion has already started automatically, the next result will reflect the previous channel selection. Subsequent conversions will reflect the new channel selection.

When switching to a differential gain channel, the first conversion result may have a poor accuracy due to the required settling time for the automatic offset cancellation circuitry. The user should preferably disregard the first conversion result.

23.6.2 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 2.56V reference, or external AREF pin.

AVCC is connected to the ADC through a passive switch. The internal 2.56V reference is generated from the internal bandgap reference (V_{BG}) through an internal amplifier. 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 2.56V 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.

If differential channels are used, the selected reference should not be closer to AVCC than indicated in Table 28-10 on page 295.

23.7 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. If the ADC is enabled in such sleep modes and the user wants to perform differential conversions, the user is advised to switch the ADC off and on after waking up from sleep to prompt an extended conversion to get a valid result.

23.7.1 Analog Input Circuitry

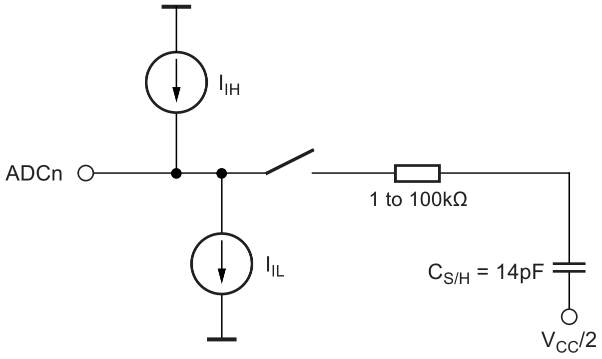
The analog input circuitry for single ended channels is illustrated in Figure 23-8 An analog source applied to ADCn is subjected to the pin capacitance and input leakage of that pin, regardless of whether that channel is selected as input for the ADC. When the channel is selected, the source must drive the S/H capacitor through the series resistance (combined resistance in the input path).

The ADC is optimized for analog signals with an output impedance of approximately 10kΩ or less. If such a source is used, the sampling time will be negligible. If a source with higher impedance is used, the sampling time will depend on how long time the source needs to charge the S/H capacitor, 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.

If differential gain channels are used, the input circuitry looks somewhat different, although source impedances of a few hundred kΩ or less is recommended.

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 23-8. Analog Input Circuitry

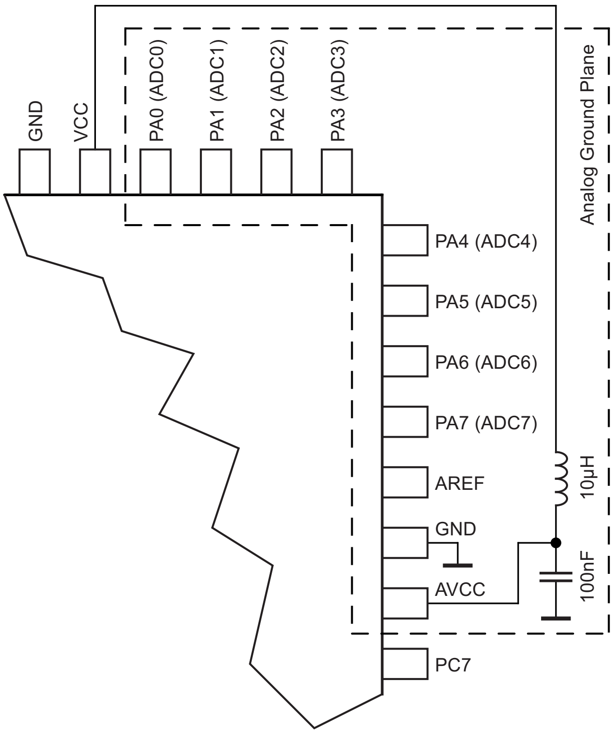


23.7.2 Analog Noise Canceling Techniques

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 23-9.
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 23-9. ADC Power Connections



23.7.3 Offset Compensation Schemes

The gain stage has a built-in offset cancellation circuitry that nulls the offset of differential measurements as much as possible. The remaining offset in the analog path can be measured directly by selecting the same channel for both differential inputs. This offset residue can be then subtracted in software from the measurement results. Using this kind of software based offset correction, offset on any channel can be reduced below one LSB.

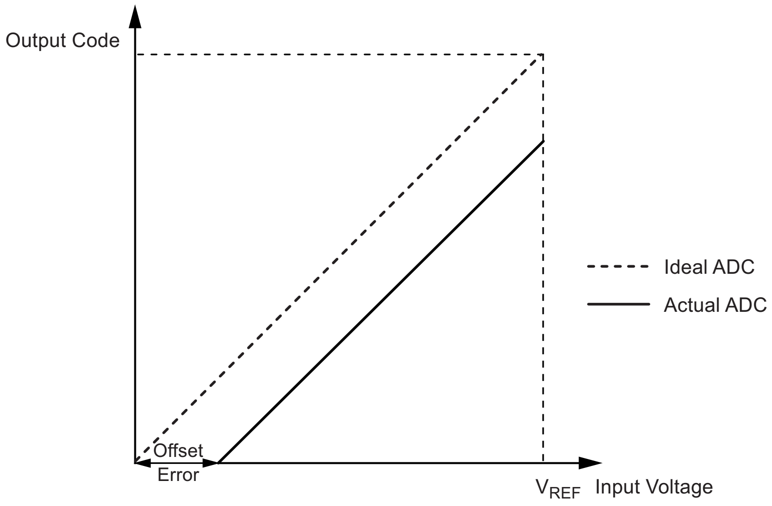
23.7.4 ADC Accuracy Definitions

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:

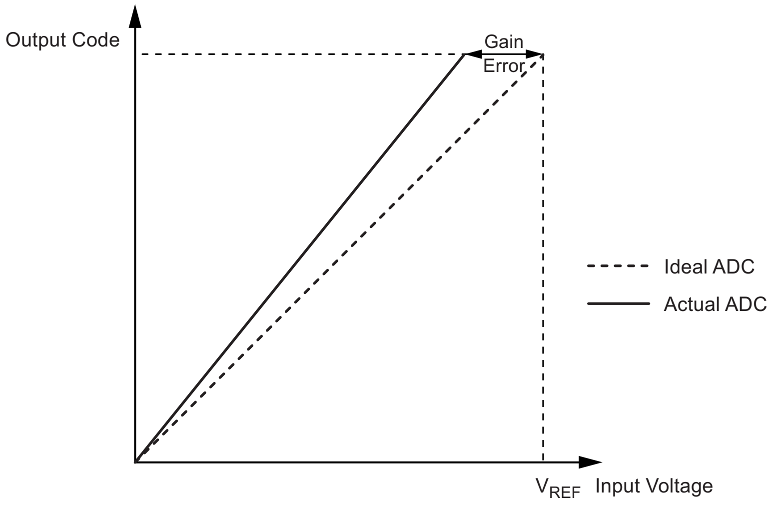
- Offset: The deviation of the first transition (0x000 to 0x001) compared to the ideal transition (at 0.5 LSB). Ideal value: 0 LSB.

Figure 23-10. Offset Error



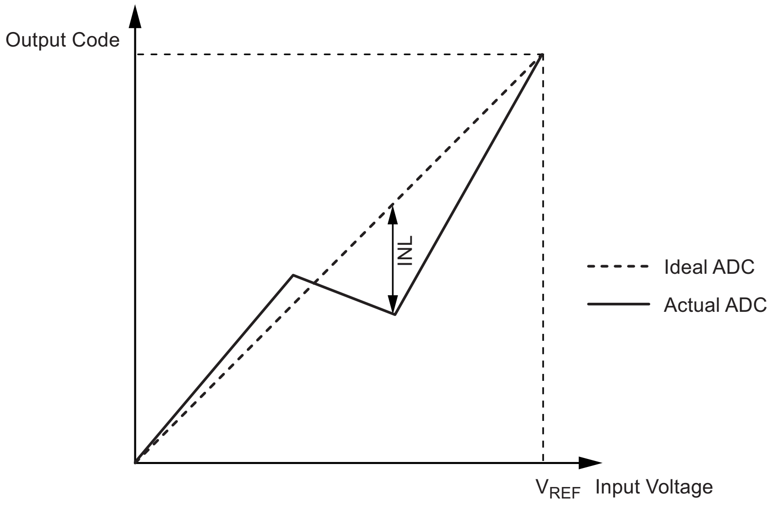
- Gain error: After adjusting for offset, the gain error is found as the deviation of the last transition (0x3FE to 0x3FF) compared to the ideal transition (at 1.5 LSB below maximum). Ideal value: 0 LSB

Figure 23-11. Gain Error



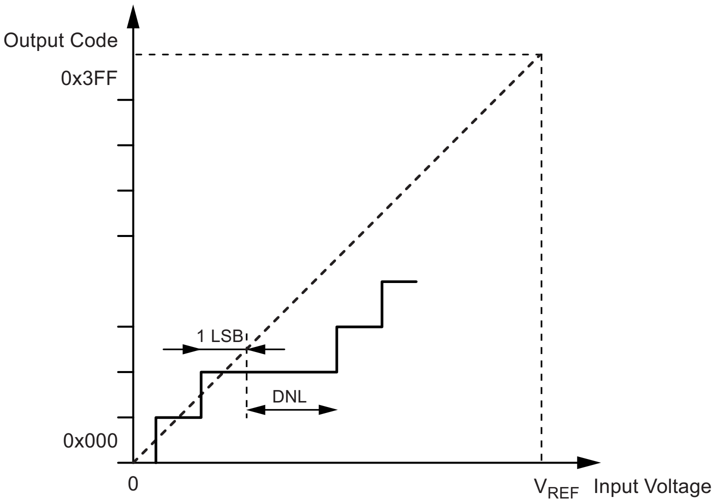
- Integral non-linearity (INL): After adjusting for offset and gain error, the INL is the maximum deviation of an actual transition compared to an ideal transition for any code. Ideal value: 0 LSB.

Figure 23-12. Integral Non-linearity (INL)



- Differential non-linearity (DNL): The maximum deviation of the actual code width (the interval between two adjacent transitions) from the ideal code width (1 LSB). Ideal value: 0 LSB.

Figure 23-13. Differential Non-linearity (DNL)



- Quantization error: Due to the quantization of the input voltage into a finite number of codes, a range of input voltages (1 LSB wide) will code to the same value. Always ±0.5 LSB.
- Absolute accuracy: The maximum deviation of an actual (unadjusted) transition compared to an ideal transition for any code. This is the compound effect of offset, gain error, differential error, non-linearity, and quantization error. ideal value: ±0.5 LSB.

23.8 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 23-3 on page 220 and Table 23-4 on page 221). 0x000 represents analog ground, and 0x3FF represents the selected reference voltage minus one LSB.

If differential channels are used, the result is

$$ADC = \frac{(V_{POS} - V_{NEG}) \cdot GAIN \cdot 512}{V_{REF}}$$

where V_{POS} is the voltage on the positive input pin, V_{NEG} the voltage on the negative input pin, GAIN the selected gain factor, and V_{REF} the selected voltage reference. The result is presented in two's complement form, from 0x200 (-512d) through 0x1FF (+511d). Note that if the user wants to perform a quick polarity check of the results, it is sufficient to read the MSB of the result (ADC9 in ADCH). If this bit is one, the result is negative, and if this bit is zero, the result is positive. Figure 23-14 shows the decoding of the differential input range.

Table 23-2 on page 220 shows the resulting output codes if the differential input channel pair (ADCn - ADCm) is selected with a gain of GAIN and a reference voltage of V_{REF}.

Figure 23-14. Differential Measurement Range

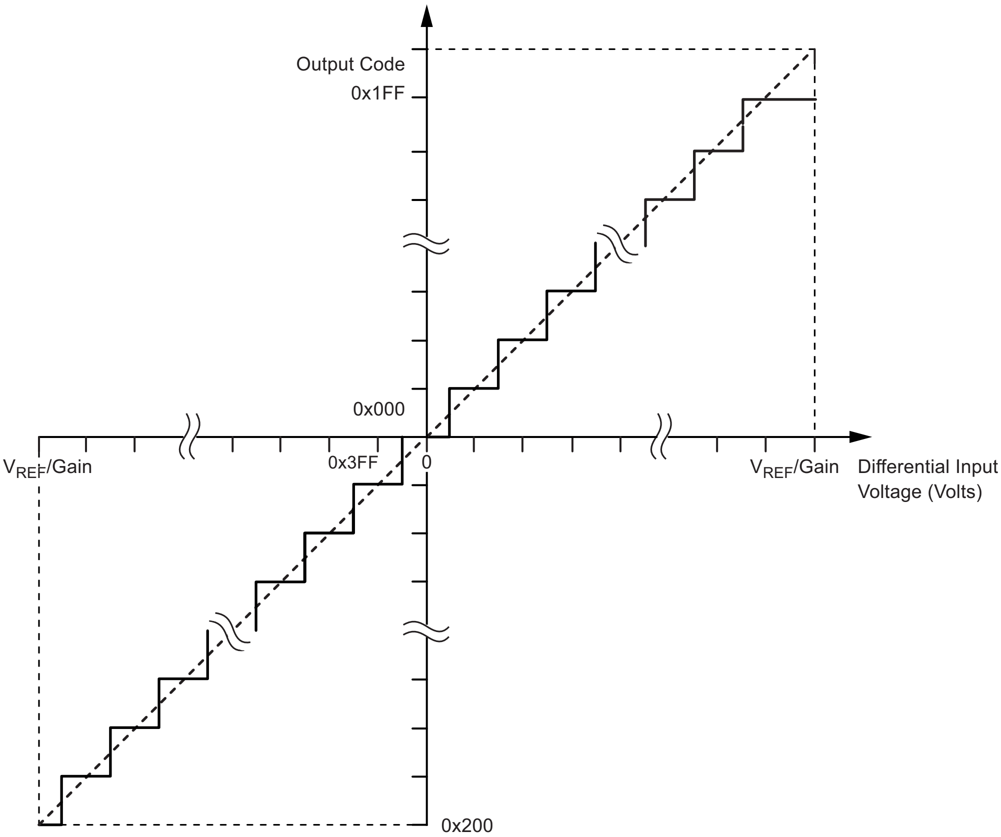


Table 23-2. Correlation between Input Voltage and Output Codes

V_{ADCn}	Read code	Corresponding Decimal Value
$V_{ADCm} + V_{REF}/GAIN$	0x1FF	511
$V_{ADCm} + 0.999\ V_{REF}/GAIN$	0x1FF	511
$V_{ADCm} + 0.998\ V_{REF}/GAIN$	0x1FE	510
...	...	...
$V_{ADCm} + 0.001\ V_{REF}/GAIN$	0x001	1
V_{ADCm}	0x000	0
$V_{ADCm} – 0.001\ V_{REF}/GAIN$	0x3FF	-1
...	...	...
$V_{ADCm} – 0.999\ V_{REF}/GAIN$	0x201	–511
$V_{ADCm} – V_{REF}/GAIN$	0x200	–512

Example: ADMUX = 0xED (ADC3 - ADC2, 10x gain, 2.56V reference, left adjusted result)
Voltage on ADC3 is 300mV, voltage on ADC2 is 500mV.
ADCR = 512 × 10 × (300 – 500) / 2560 = –400 = 0x270
ADCL will thus read 0x00, and ADCH will read 0x9C. Writing zero to ADLAR right adjusts the result: ADCL = 0x70, ADCH = 0x02.

23.9 Register Description

23.9.1 ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	**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 23-3. 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 23-3. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal V_{REF} turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Internal 1.1V voltage reference with external capacitor at AREF pin
1	1	Internal 2.56V voltage reference with external capacitor at AREF pin

Note: If differential channels are selected, only 2.56V should be used as internal voltage reference.

- **Bit 5 – ADLAR: ADC Left Adjust Result**

The ADLAR bit affects the presentation of the ADC conversion result in the ADC data register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC data register immediately, regardless of any ongoing conversions. For a complete description of this bit, see Section 23.9.3 “ADCL and ADCH – The ADC Data Register” on page 223.

- **Bits 4:0 – MUX4:0: Analog Channel and Gain Selection Bits**

The value of these bits selects which combination of analog inputs are connected to the ADC. These bits also select the gain for the differential channels. See Table 23-4 on page 221 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 23-4. Input Channel and Gain Selections

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	1.1V (V_{BG})	N/A		
11111	0 V (GND)			

23.9.2 ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	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 23-5. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

23.9.3 ADCL and ADCH – The ADC Data Register

23.9.3.1 ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	**ADC9**	**ADC8**	**ADCH**
(0x78)	**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	

23.9.3.2 ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
(0x79)	**ADC9**	**ADC8**	**ADC7**	**ADC6**	**ADC5**	**ADC4**	**ADC3**	**ADC2**	**ADCH**
(0x78)	**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. If differential channels are used, the result is presented in two's complement form.

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 Section 23.8 "ADC Conversion Result" on page 219.

23.9.4 ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
(0x7B)	–	**ACME**	–	–	–	**ADTS2**	**ADTS1**	**ADTS0**	**ADCSRB**
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7, 5:3 – Reserved**

These bits are reserved for future use in the Atmel® ATmega164P-B/324P-B/644P-B. For ensuring culpability with future devices, these bits must be written 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 ADTS[2:0] 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 23-6. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free running mode
0	0	1	Analog comparator
0	1	0	External interrupt request 0
0	1	1	Timer/Counter0 compare match
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

23.9.5 DIDR0 – Digital Input Disable Register 0

Bit	7	6	5	4	3	2	1	0	
(0x7E)	**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.

24. JTAG Interface and On-chip Debug System

24.1 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®

24.2 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 Section 27.10 “Programming via the JTAG Interface” on page 274 and Section 25. “IEEE 1149.1 (JTAG) Boundary-scan” on page 231, respectively. The on-chip debug support is considered being private JTAG instructions, and distributed within ATMEL and to selected third party vendors only.

Figure 24-1 on page 226 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.

24.3 TAP – Test Access Port

The JTAG interface is accessed through four of the AVR® 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, the input TAP signals 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 24-1. Block Diagram

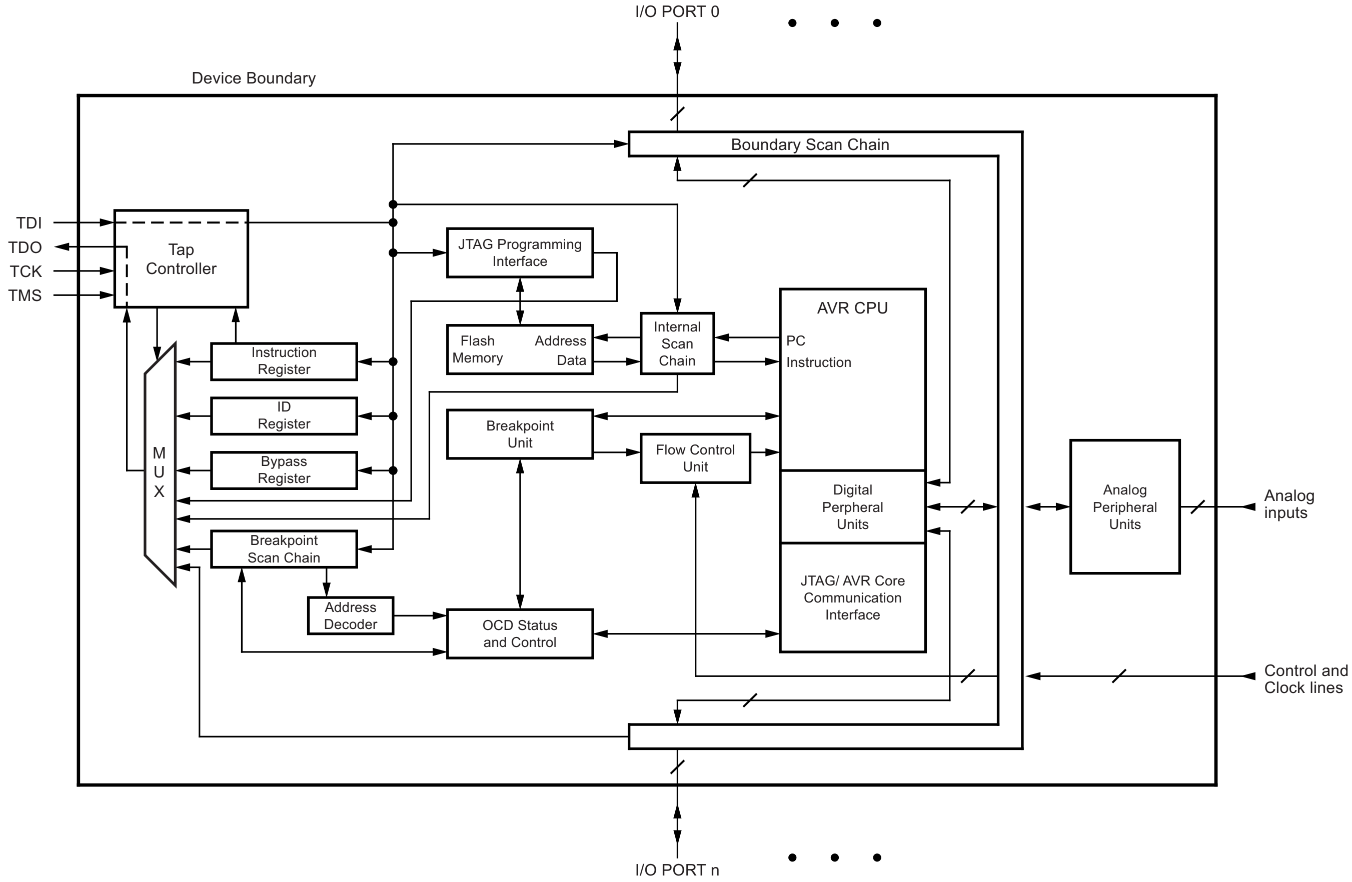
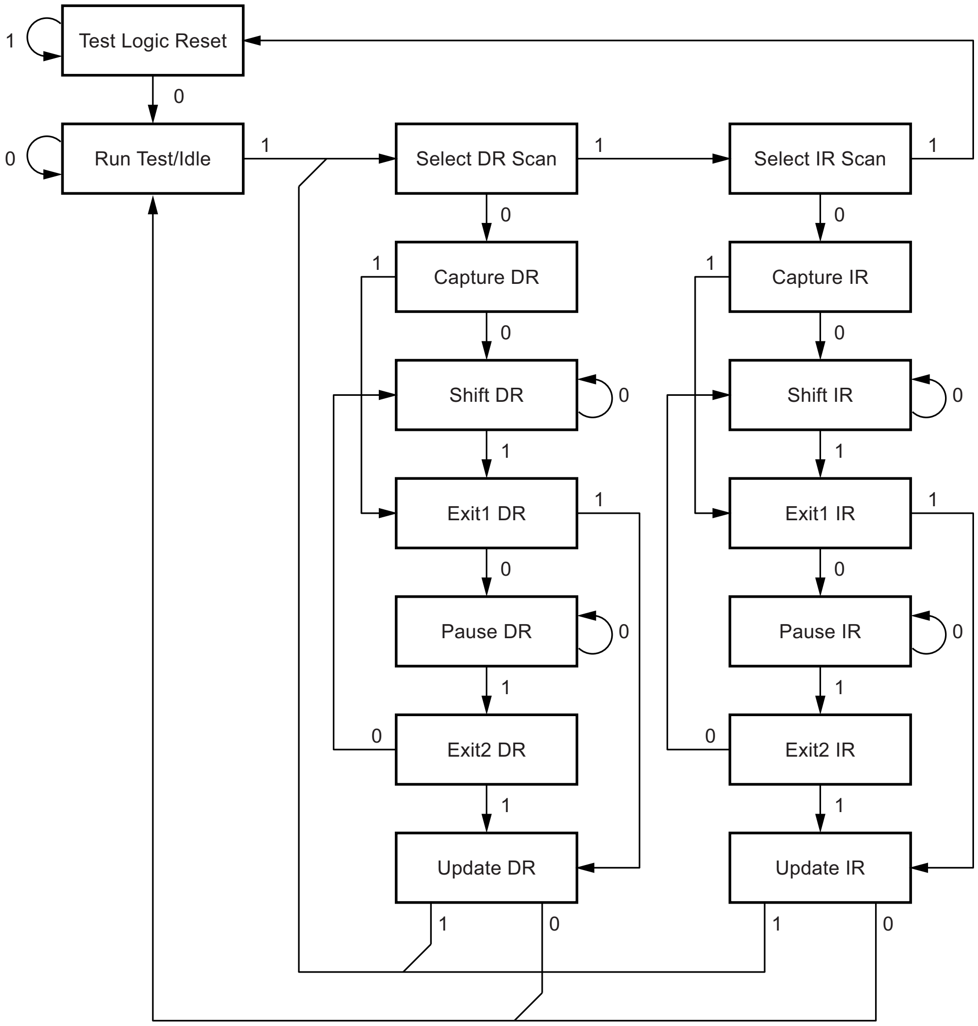


Figure 24-2. TAP Controller State Diagram



24.4 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 24-2 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 3LSBs 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 Section 24.9 “Bibliography” on page 229.

24.5 Using the Boundary-scan Chain

A complete description of the boundary-scan capabilities are given in the section Section 25. “IEEE 1149.1 (JTAG) Boundary-scan” on page 231.

24.6 Using the On-chip Debug System

As shown in Figure 24-1 on page 226, 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 Section 24.7 “On-chip Debug Specific JTAG Instructions” on page 229.

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 AVR Assembler and C programs compiled with third party vendors' compilers.

AVR Studio runs under Microsoft Windows® 95/98/2000 and Microsoft Windows NT®.

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.

24.7 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.

24.7.1 PRIVATE0; 0x8

Private JTAG instruction for accessing on-chip debug system.

24.7.2 PRIVATE1; 0x9

Private JTAG instruction for accessing on-chip debug system.

24.7.3 PRIVATE2; 0xA

Private JTAG instruction for accessing on-chip debug system.

24.7.4 PRIVATE3; 0xB

Private JTAG instruction for accessing on-chip debug system.

24.8 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 Section 27.10 “Programming via the JTAG Interface” on page 274.

24.9 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.

24.10 Register Description

24.10.1 OCDR – On-chip Debug Register

Bit	7	6	5	4	3	2	1	0	
0x31 (0x51)	**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.

25. IEEE 1149.1 (JTAG) Boundary-scan

25.1 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

25.2 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.

25.3 Data Registers

The data registers relevant for boundary-scan operations are:

- Bypass register
- Device identification register
- Reset register
- Boundary-scan chain

25.3.1 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.

25.3.2 Device Identification Register

Figure 25-1 shows the structure of the device identification register.

Figure 25-1. The Format of the Device Identification Register

	MSB			LSB
Bit	31 28	27 12	11 1	0
Device ID	**Version**	**Part Number**	**Manufacturer ID**	**1**
	4 bits	16 bits	11 bits	1-bit

25.3.2.1 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.

25.3.2.2 Part Number

The part number is a 16-bit code identifying the component. The JTAG part number for Atmel® ATmega164P-B/324P-B/644P-B is listed in Table 27-6 on page 258.

25.3.2.3 Manufacturer ID

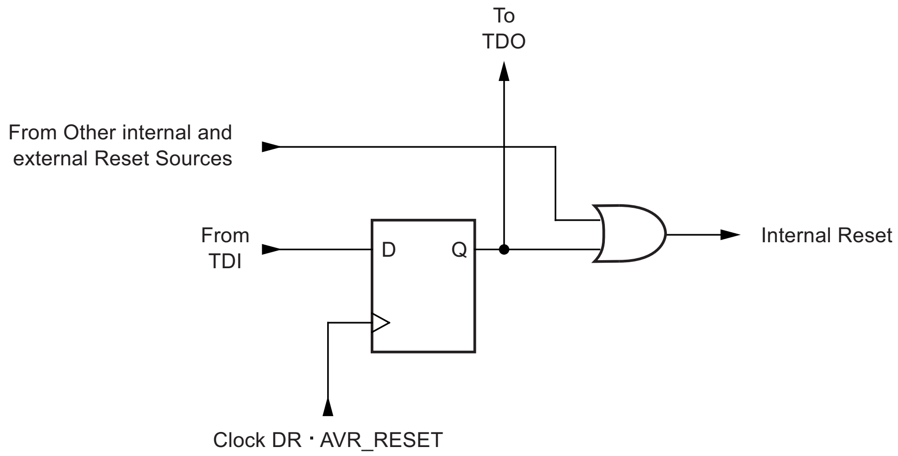
The manufacturer ID is a 11-bit code identifying the manufacturer. The JTAG manufacturer ID for ATMEL is listed in Table 27-6 on page 258.

25.3.3 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 Section 9.2 “Clock Sources” on page 24) 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 25-2 on page 232.

Figure 25-2. Reset Register



25.3.4 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 Section 25.5 “Boundary-scan Chain” on page 234 for a complete description.

25.4 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 datasheet, 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.

25.4.1 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.

25.4.2 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.

25.4.3 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.

25.4.4 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.

25.4.5 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.

25.5 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.

25.5.1 Scanning the Digital Port Pins

Figure 25-3 on page 235 shows the boundary-scan cell for a bi-directional port pin. The pull-up function is disabled during boundary-scan when the JTAG IC contains EXTEST or SAMPLE_PRELOAD. The cell consists of 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 datasheet. Figure 25-4 on page 236 shows a simple digital port pin as described in the Section 14. “I/O-Ports” on page 57. The boundary-scan details from Figure 25-3 on page 235 replaces the dashed box in Figure 25-4 on page 236.

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 $\overline{\text{PUD}} \cdot \overline{\text{DDxn}} \cdot \text{PORTxn}$.

Digital alternate port functions are connected outside the dotted box in Figure 25-4 on page 236 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. There is no scan chain on the interface between the digital and the analog circuitry, but some digital control signal to analog circuitry are turned off to avoid driving contention on the pads.

When JTAG IR contains EXTEST or SAMPLE_PRELOAD the clock is not sent out on the port pins even if the CKOUT fuse is programmed. Even though the clock is output when the JTAG IR contains SAMPLE_PRELOAD, the clock is not sampled by the boundary scan.

Figure 25-3. Boundary-scan Cell for Bi-directional Port Pin with Pull-up Function

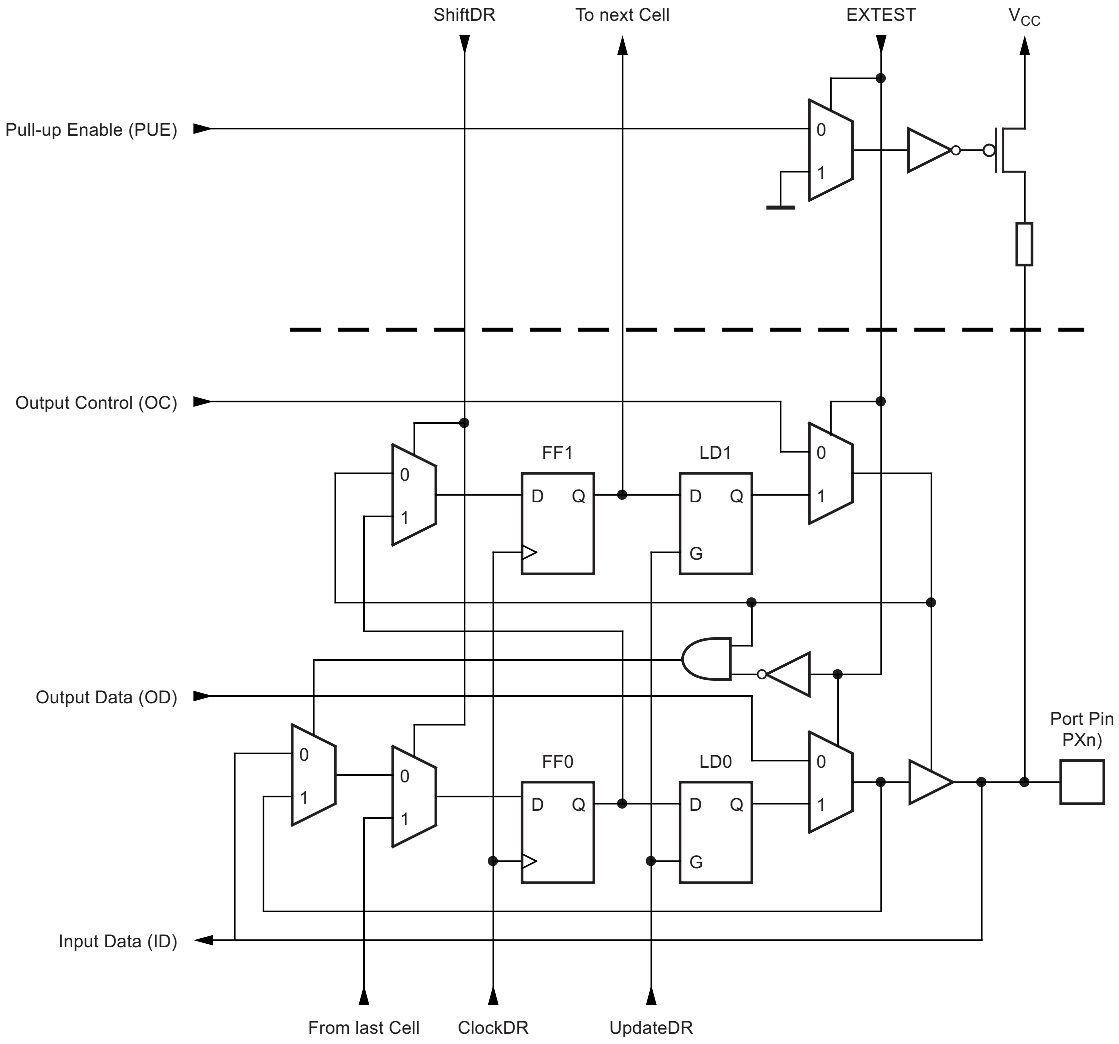
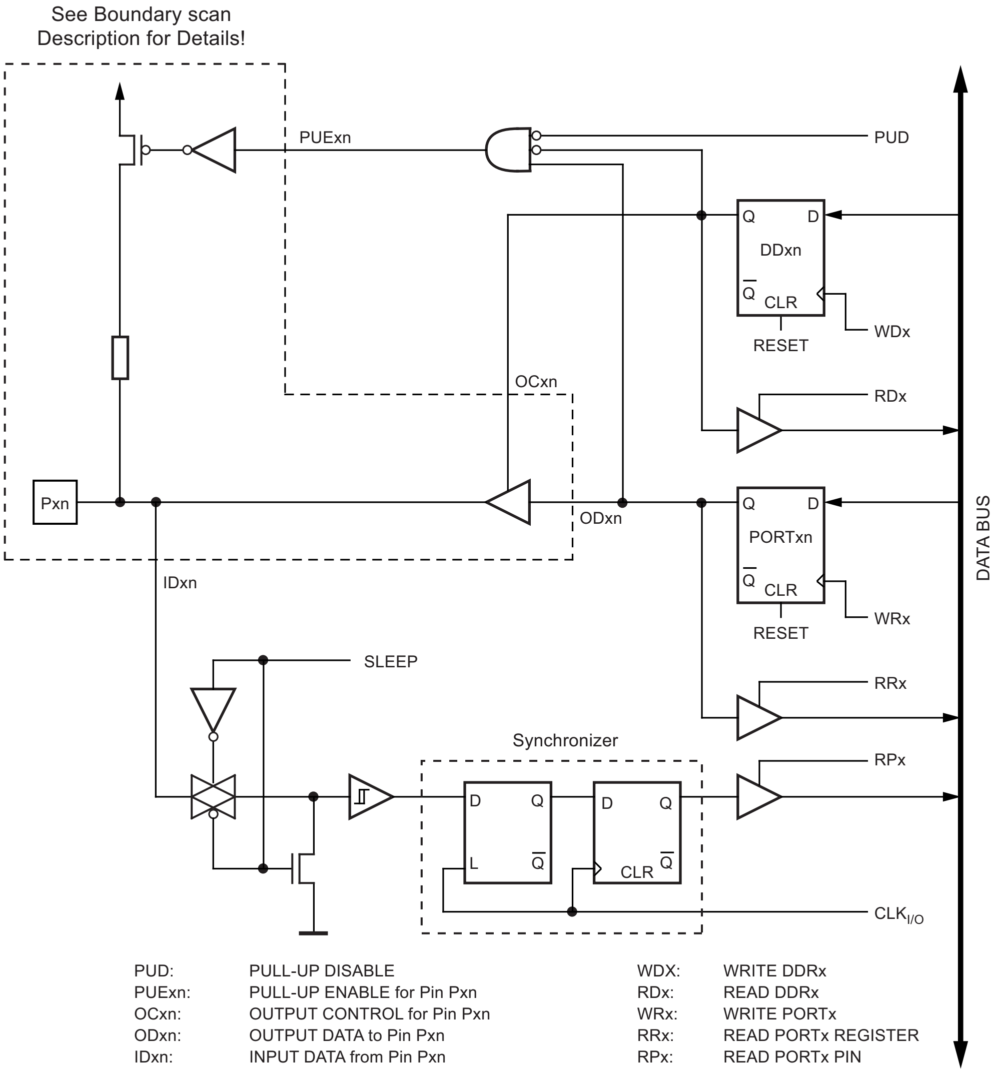


Figure 25-4. General Port Pin Schematic Diagram

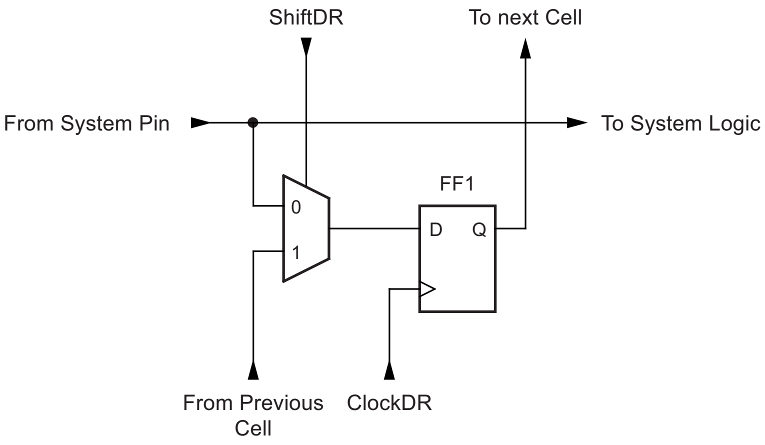


PUD:	PULL-UP DISABLE	WDX:	WRITE DDRx
PUExn:	PULL-UP ENABLE for Pin Pxn	RDx:	READ DDRx
OCxn:	OUTPUT CONTROL for Pin Pxn	WRx:	WRITE PORTx
ODxn:	OUTPUT DATA to Pin Pxn	RRx:	READ PORTx REGISTER
IDxn:	INPUT DATA from Pin Pxn	RPx:	READ PORTx PIN
SLEEP:	SLEEP CONTROL	CLK: $_{I/O}$	I/O CLOCK

25.5.2 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 25-5 is inserted for the 5V reset signal.

Figure 25-5. Observe-only Cell



25.6 ATmega164P-B/324P-B/644P-B Boundary-scan Order

Table 25-1 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 and port K 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 25-3 on page 235, PXn. Data corresponds to FF0, PXn. Control corresponds to FF1, PXn. 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 25-1. ATmega164P-B/324P-B/644P-B Boundary-scan Order

Bit Number	Signal Name	Module
56	PB0.Data	Port B
55	PB0.Control	
54	PB1.Data	
53	PB1.Control	
52	PB2.Data	
51	PB2.Control	
50	PB3.Data	
49	PB3.Control	
48	PB4.Data	
47	PB4.Control	
46	PB5.Data	
45	PB5.Control	
44	PB6.Data	
43	PB6.Control	
42	PB7.Data	
41	PB7.Control	

Table 25-1. ATmega164P-B/324P-B/644P-B Boundary-scan Order (Continued)

Bit Number	Signal Name	Module
40	RSTT	Reset logic (observe only)
39	PD0.Data	Port D
38	PD0.Control	
37	PD1.Data	
36	PD1.Control	
35	PD2.Data	
34	PD2.Control	
33	PD3.Data	
32	PD3.Control	
31	PD4.Data	
30	PD4.Control	
29	PD5.Data	
28	PD5.Control	
27	PD6.Data	
26	PD6.Control	
25	PD7.Data	
24	PD7.Control	
23	PC0.Data	Port C
22	PC0.Control	
21	PC1.Data	
20	PC1.Control	
19	PC6.Data	
18	PC6.Control	
17	PC7.Data	
16	PC7.Control	
15	PA7.Data	
14	PA7.Control	Port A
13	PA6.Data	
12	PA6.Control	
11	PA5.Data	
10	PA5.Control	
9	PA4.Data	
8	PA4.Control	
7	PA3.Data	
6	PA3.Control	
5	PA2.Data	
4	PA2.Control	
3	PA1.Data	
2	PA1.Control	
1	PA0.Data	
0	PA0.Control	

25.7 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. BSDL files are available for Atmel® ATmega164P-B/324P-B/644P-B.

25.8 Register Description

25.8.1 MCUCR – MCU Control Register

The MCU control register contains control bits for general MCU functions.

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	**JTD**	BODS	**BODSE**	**PUD**	–	–	**IVSEL**	**IVCE**	**MCUCR**
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	

- **Bits 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.

25.8.2 MCUSR – MCU Status Register

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

Bit	7	6	5	4	3	2	1	0	
0x34 (0x54)	–	–	–	**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.

26. Boot Loader Support – Read-While-Write Self-Programming

26.1 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 27-7 on page 258) used during programming. The page organization does not affect normal operation.

26.2 Overview

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.

26.3 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 26-2 on page 242). The size of the different sections is configured by the BOOTSZ fuses as shown in Table 26-10 on page 251 and Figure 26-2 on page 242. These two sections can have different level of protection since they have different sets of lock bits.

26.3.1 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 26-2 on page 243. The application section can never store any boot loader code since the SPM instruction is disabled when executed from the application section.

26.3.2 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 26-3 on page 243.

26.4 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 26-1 and Figure 26-1 on page 242. 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.

26.4.1 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 load program memory, call, or jump instructions 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 Section 26.9.1 “SPMCSR – Store Program Memory Control and Status Register” on page 253 for details on how to clear RWWSB.

26.4.2 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 26-1. 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 26-1. Read-while-write versus No Read-while-write

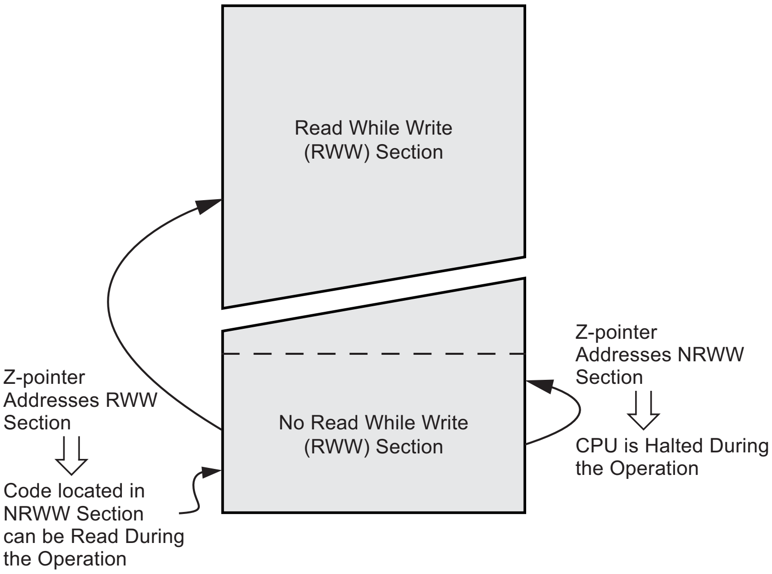
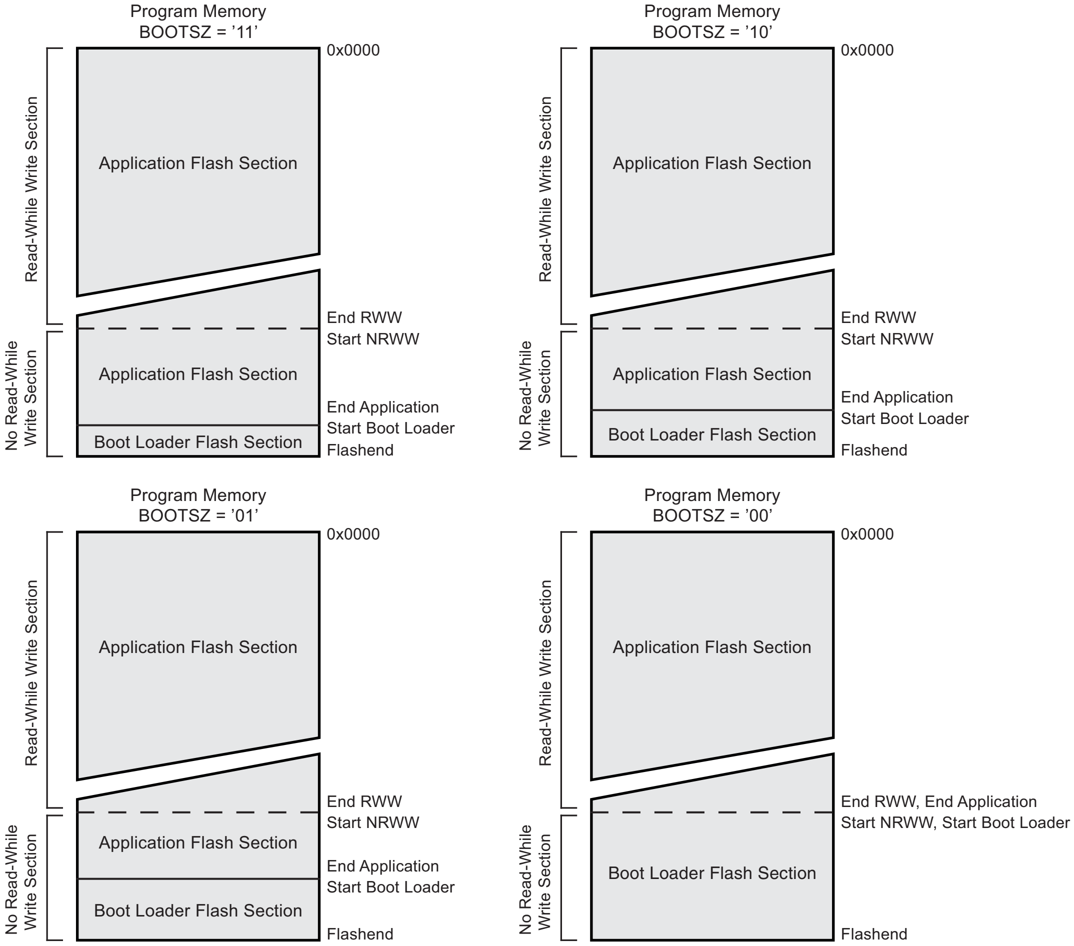


Figure 26-2. Memory Sections



Note: The parameters in the figure above are given in Table 26-10 on page 251.

26.5 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 26-2 and Table 26-3 for further details. The boot 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 allow the programming of the flash memory by SPM instruction. Similarly, the general read/write lock (Lock bit mode 3) does not allow reading nor writing by (E)LPM/SPM, if it is attempted.

Table 26-2. Boot Lock Bit0 Protection Modes (Application Section)(1)

BLB0 Mode	BLB02	BLB01	Protection
1	1	1	No restrictions for SPM or (E)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 (E)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	(E)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 26-3. Boot Lock Bit1 Protection Modes (Boot Loader Section)(1)

BLB1 Mode	BLB12	BLB11	Protection
1	1	1	No restrictions for SPM or (E)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 (E)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	(E)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

26.6 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 26-4. Boot Reset Fuse[1]

BOOTRST	Reset Address
1	Reset vector = application reset (address 0x0000)
0	Reset vector = boot loader reset (see Table 26-10 on page 251)

Note: 1. “1” means unprogrammed, “0” means programmed

26.7 Addressing the Flash During Self-programming

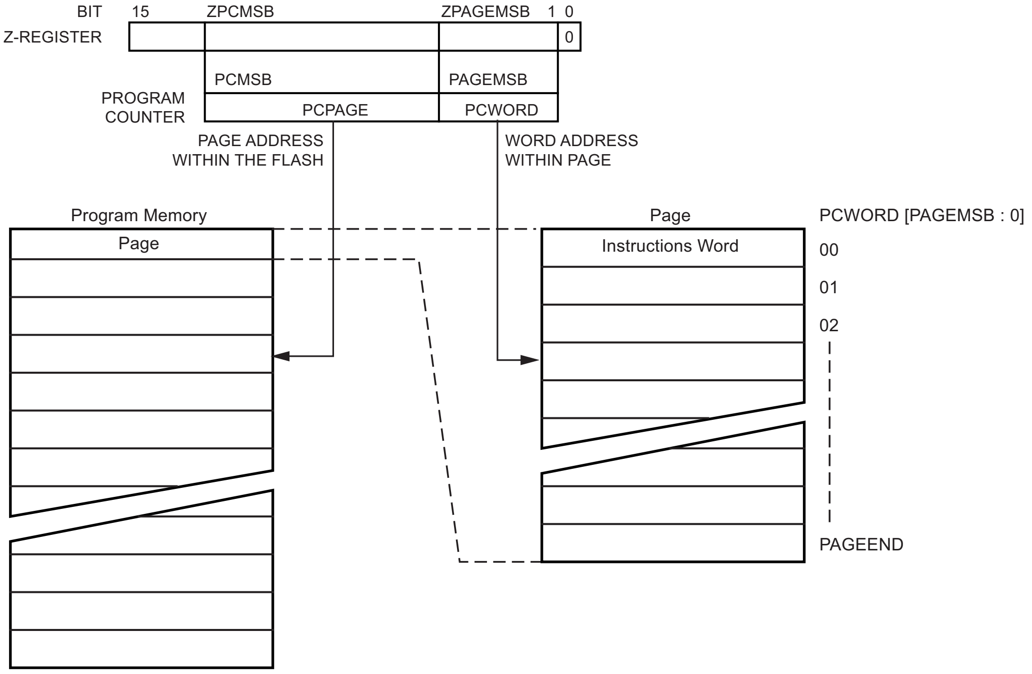
The Z-pointer is used to address the SPM commands. The Z pointer consists of the Z-registers ZL and ZH in the register file, and RAMPZ in the I/O space. The number of bits actually used is implementation dependent. Note that the RAMPZ register is only implemented when the program space is larger than 64Kbytes.

Bit	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8
RAMPZ	**RAMPZ7**	**RAMPZ6**	**RAMPZ5**	**RAMPZ4**	**RAMPZ3**	**RAMPZ2**	**RAMPZ1**	**RAMPZ0**
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 27-7 on page 258), 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 26-3 on page 245. 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 LPM instruction use the Z-pointer to store the address. Since this instruction addresses the flash byte-by-byte, also bit Z0 of the Z-pointer is used.

Figure 26-3. Addressing the Flash During SPM(1)



Note: 1. The different variables used in Figure 26-3 are listed in Table 26-12 on page 252.

26.8 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 Section 26.8.13 “Simple Assembly Code Example for a Boot Loader” on page 249 for an assembly code example.

26.8.1 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.

Note: If an interrupt occurs in the time sequence, the four cycle access cannot be guaranteed. In order to ensure atomic operation, you should disable interrupts before writing to SPMCSR.

26.8.2 Filling the Temporary Buffer (Page Loading)

To write an instruction word, set up the address in the Z-pointer and data in R1:R0, write “00000001” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The content of PCWORD in the Z-register is used to address the data in the temporary buffer. The temporary buffer will auto-erase after a page write operation or by writing the 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.

26.8.3 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.

26.8.4 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 Section 12. “Interrupts” on page 49.

26.8.5 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.

26.8.6 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 Section 12. “Interrupts” on page 49, 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 Section 26.8.13 “Simple Assembly Code Example for a Boot Loader” on page 249 for an example.

26.8.7 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 26-2 on page 243 and Table 26-3 on page 243 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 boot 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 IO_{ck} 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.

26.8.8 EEPROM Write Prevents Writing to SPMCSR

Note that an EEPROM write operation will block all software programming to flash. Reading the fuses and lock bits from software will also be prevented during the EEPROM write operation. It is recommended that the user checks the status bit (EEPE) in the EECR register and verifies that the bit is cleared before writing to the SPMCSR register.

26.8.9 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 (E)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 (E)LPM instruction is executed within three CPU cycles or no SPM instruction is executed within four CPU cycles. When BLBSET and SPMEN are cleared, (E)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 (E)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 27-5 on page 257 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 (E)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 27-4 on page 257 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 (E)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 27-3 on page 256 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.

26.8.10 Reading the Signature Row from Software

To read the signature row from software, load the Z-pointer with the signature byte address given in Table 26-5 and set the SIGRD and SPMEN bits in SPMCSR. When an LPM instruction is executed within three CPU cycles after the SIGRD and SPMEN bits are set in SPMCSR, the signature byte value will be loaded in the destination register. The SIGRD and SPMEN bits will auto-clear upon completion of reading the signature row lock bits or if no LPM instruction is executed within three CPU cycles. When SIGRD and SPMEN are cleared, LPM will work as described in the instruction set manual.

Table 26-5. Signature Row Addressing

Signature Byte	Z-Pointer Address
Device signature byte 1	0x0000
Device signature byte 2	0x0002
Device signature byte 3	0x0004
RC oscillator calibration byte 3V	0x0001
RC oscillator calibration byte 5V	0x0003

Note: All other addresses are reserved for future use.

26.8.11 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.

26.8.12 Programming Time for Flash when Using SPM

The calibrated RC oscillator is used to time flash accesses. Table 26-6 shows the typical programming time for flash accesses from the CPU.

Table 26-6. SPM Programming Time(1)

Symbol	Min Programming Time	Max Programming Time
Flash write (page erase, page write, and write lock bits by SPM)	3.7ms	4.5ms

Note: 1. Minimum and maximum programming times is per individual operation.

26.8.13 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.
.equu                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, EEPE
        rjmp   Wait_ee
        ;      SPM timed sequence
        out    SPMCSR, spmcrval
        spm
        ;restore SREG (to enable interrupts if originally enabled)
        out    SREG, temp2
        ret
```

26.8.14 ATmega164P-B Boot Loader Parameters

In Table 26-7 through Table 26-9 on page 251, the parameters used in the description of the self-programming are given.

Table 26-7. 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	128 words	2	0x0000 - 0x1F7F	0x1F80 - 0x1FFF	0x1F7F	0x1F80
1	0	256 words	4	0x0000 - 0x1EFF	0x1F00 - 0x1FFF	0x1EFF	0x1F00
0	1	512 words	8	0x0000 - 0x1DFF	0x1E00 - 0x1FFF	0x1DFF	0x1E00
0	0	1024 words	16	0x0000 - 0x1BFF	0x1C00 - 0x1FFF	0x1BFF	0x1C00

Note: 1. The different BOOTSZ fuse configurations are shown in Figure 26-2 on page 242.

Table 26-8. Read-While-Write Limit[(1)]

Section	Pages	Address
Read-while-write section (RWW)	112	0x0000 - 0x1BFF
No read-while-write section (NRWW)	16	0x1C00 - 0x1FFF

Note: 1. For details about these two sections, see Section 26.4.2 “NRWW – No Read-While-Write Section” on page 241 and Section 26.4.1 “RWW – Read-While-Write Section” on page 241.

Table 26-9. Explanation of Different Variables used in Figure 26-3 on page 245 and the Mapping to the Z-pointer

Variable		Corresponding Z-value	Description(1)
PCMSB	12		Most significant bit in the program counter. (The program counter is 13 bits PC[12:0])
PAGEMSB	5		Most significant bit which is used to address the words within one page (128 words in a page requires seven bits PC [5:0]).
ZPCMSB		Z13	Bit in Z-pointer that is mapped to PCMSB. Because Z0 is not used, the ZPCMSB equals PCMSB + 1.
ZPAGEMSB		Z6	Bit in Z-pointer that is mapped to PCMSB. Because Z0 is not used, the ZPAGEMSB equals PAGEMSB + 1.
PCPAGE	PC[12:6]	Z14:Z7	Program counter page address: Page select, for page erase and page write
PCWORD	PC[5:0]	Z6:Z1	Program counter word address: Word select, for filling temporary buffer (must be zero during page write operation)

Note: 1. Z0: should be zero for all SPM commands, byte select for the (E)LPM instruction.

See Section 26.7 "Addressing the Flash During Self-programming" on page 244 for details about the use of Z-pointer during self-programming.

26.8.15 ATmega324P-B Boot Loader Parameters

In Table 26-10 through Table 26-12 on page 252, the parameters used in the description of the self-programming are given.

Table 26-10. 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 words	4	0x0000 - 0x3EFF	0x3F00 - 0x3FFF	0x3EFF	0x3F00
1	0	512 words	8	0x0000 - 0x1DFF	0x3E00 - 0x3FFF	0x3DFF	0x3E00
0	1	1024 words	16	0x0000 - 0x1BFF	0x3C00 - 0x3FFF	0x3BFF	0x3C00
0	0	2048 words	32	0x0000 - 0x37FF	0x3800 - 0x3FFF	0x37FF	0x3800

Note: 1. The different BOOTSZ fuse configurations are shown in Figure 26-2 on page 242.

Table 26-11. Read-While-Write Limit(1)

Section	Pages	Address
Read-while-write section (RWW)	224	0x0000 - 0x37FF
No read-while-write section (NRWW)	32	0x3800 - 0x3FFF

Note: 1. For details about these two sections, see Section 26.4.2 "NRWW – No Read-While-Write Section" on page 241 and Section 26.4.1 "RWW – Read-While-Write Section" on page 241.

Table 26-12. Explanation of Different Variables used in Figure 26-3 on page 245 and the Mapping to the Z-pointer

Variable		Corresponding Z-value	Description(1)
PCMSB	13		Most significant bit in the program counter. (The program counter is 14 bits PC[13:0])
PAGEMSB	6		Most significant bit which is used to address the words within one page (128 words in a page requires seven bits PC [5:0]).
ZPCMSB		Z14	Bit in Z-pointer that is mapped to PCMSB. Because Z0 is not used, the ZPCMSB equals PCMSB + 1.
ZPAGEMSB		Z7	Bit in Z-pointer that is mapped to PCMSB. Because Z0 is not used, the ZPAGEMSB equals PAGEMSB + 1.
PCPAGE	PC[13:6]	Z14:Z7	Program counter page address: page select, for page erase and page write
PCWORD	PC[5:0]	Z6:Z1	Program counter word address: Word select, for filling temporary buffer (must be zero during page write operation)

Note: 1. 0: should be zero for all SPM commands, byte select for the (E)LPM instruction.

See Section 26.7 “Addressing the Flash During Self-programming” on page 244 for details about the use of Z-pointer during self-programming.

26.8.16 ZATmega644P-B Boot Loader Parameters

In Table 26-13 through Table 26-15 on page 253, the parameters used in the description of the self-programming are given.

Table 26-13. 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	512 words	4	0x0000 - 0x7DFF	0x7E00 - 0x7FFF	0x7DFF	0x7E00
1	0	1024 words	8	0x0000 - 0x7BFF	0x7C00 - 0x7FFF	0x7BFF	0x7C00
0	1	2048 words	16	0x0000 - 0x77FF	0x7800 - 0x7FFF	0x77FF	0x7800
0	0	4096 words	32	0x0000 - 0x6FFF	0x7000 - 0x7FFF	0x6FFF	0x7000

Note: 1. The different BOOTSZ fuse configurations are shown in Figure 26-2 on page 242.

Table 26-14. Read-while-write Limit(1)

Section	Pages	Address
Read-while-write section (RWW)	224	0x0000 - 0x6FFF
No read-while-write section (NRWW)	32	0x7000 - 0x7FFF

Note: 1. For details about these two sections, see Section 26.4.2 “NRWW – No Read-While-Write Section” on page 241 and Section 26.4.1 “RWW – Read-While-Write Section” on page 241.

Table 26-15. Explanation of Different Variables used in Figure 26-3 on page 245 and the Mapping to the Z-pointer

Variable		Corresponding Z-value	Description[1]
PCMSB	14		Most significant bit in the program counter. (The program counter is 14 bits PC[14:0])
PAGEMSB	7		Most significant bit which is used to address the words within one page (128 words in a page requires seven bits PC [6:0]).
ZPCMSB		Z15	Bit in Z-pointer that is mapped to PCMSB. Because Z0 is not used, the ZPCMSB equals PCMSB + 1.
ZPAGEMSB		Z8	Bit in Z-pointer that is mapped to PCMSB. Because Z0 is not used, the ZPAGEMSB equals PAGEMSB + 1.
PCPAGE	PC[14:7]	Z15:Z7	Program counter page address: page select, for page erase and page write
PCWORD	PC[6:0]	Z7:Z1	Program counter word address: word select, for filling temporary buffer (must be zero during page write operation)

Note: 1. Z0: should be zero for all SPM commands, byte select for the (E)LPM instruction.

See Section 26.7 "Addressing the Flash During Self-programming" on page 244 for details about the use of Z-pointer during self-programming.

26.9 Register Description

26.9.1 SPMCSR – Store Program Memory Control and Status Register

The store program memory control and status register contains the control bits needed to control the boot loader operations.

Bit	7	6	5	4	3	2	1	0	
0x37 (0x57)	**SPMIE**	**RWWSB**	SIGRD	RWWSRE	**BLBSET**	**PGWRT**	**PGERS**	**SPMEN**	**SPMCSR**
Read/Write	R/W	R	R/W	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 – SIGRD: Signature Row Read**

If this bit is written to one at the same time as SPMEN, the next LPM instruction within three clock cycles will read a byte from the signature row into the destination register. see Section 26.8.10 "Reading the Signature Row from Software" on page 248 for details. An SPM instruction within four cycles after SIGRD and SPMEN are set will have no effect. This operation is reserved for future use and should not be used.

- **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, 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 (E)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 Section 26.8.9 "Reading the Fuse and Lock Bits from Software" on page 247 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.

Note: Only one SPM instruction should be active at any time.

27. Memory Programming

27.1 Program And Data Memory Lock Bits

The Atmel® ATmega164P-B/324P-B/644P-B provides six lock bits which can be left unprogrammed (“1”) or can be programmed (“0”) to obtain the additional features listed in Table 27-2. The lock bits can only be erased to “1” with the chip erase command.

Table 27-1. 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 27-2. Lock Bit Protection Modes(1)(2)

Memory Lock Bits			Protection Type
LB Mode	LB2	LB1	
1	1	1	No memory lock features enabled.
2	1	0	Further programming of the flash and EEPROM is disabled in parallel and serial programming mode. The fuse bits are locked in both serial and parallel programming mode.(1)
3	0	0	Further programming and verification of the flash and EEPROM is disabled in parallel, JTAG 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 (E)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 (E)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	(E)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.

Notes: 1. Program the fuse bits and boot lock bits before programming the LB1 and LB2.

2. “1” means unprogrammed, “0” means programmed

Table 27-2. Lock Bit Protection Modes(1)(2) (Continued)

Memory Lock Bits			Protection Type
BLB1 Mode	BLB12	BLB11	
1	1	1	No restrictions for SPM or (E)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 (E)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	(E)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

27.2 Fuse Bits

The Atmel® ATmega164P-B/324P-B/644P-B has four fuse bytes. Table 27-3 - Table 27-5 on page 257 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 27-3. Extended Fuse Byte

Fuse Low Byte	Bit No	Description	Default Value
–	7	–	1
–	6	–	1
–	5	–	1
–	4	–	1
–	3	–	1
BODLEVEL2(1)	2	Brown-out detector trigger level	1 (unprogrammed)
BODLEVEL1(1)	1	Brown-out detector trigger level	0 (programmed)
BODLEVEL0(1)	0	Brown-out detector trigger level	1 (unprogrammed)

Note: 1. See Section 28.5 "System and Reset Characteristics" on page 291 for BODLEVEL fuse decoding (default = 2.7V).

Table 27-4. Fuse High Byte

Fuse High Byte	Bit No	Description	Default Value
OCDEN[4]	7	Enable OCD	1 (unprogrammed, OCD disabled)
JTAGEN	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 27-9 on page 259 for details)	0 (programmed)[2]
BOOTSZ0	1	Select boot size (see Table 27-9 on page 259 for details)	0 (programmed)[2]
BOOTRST	0	Select reset vector	1 (unprogrammed)

Notes: 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 26-10 on page 251 for details.

3. See Section 11.4.2 “WDTCSR – Watchdog Timer Control Register” on page 47 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.

Table 27-5. Fuse Low Byte

Fuse Low Byte	Bit No	Description	Default Value
CKDIV8[4]	7	Divide clock by 8	0 (programmed)
CKOUT[3]	6	Clock output	1 (unprogrammed)
SUT1	5	Select start-up time	1 (unprogrammed)[1]
SUT0	4	Select start-up time	0 (programmed)[1]
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]

Notes: 1. The default value of SUT1..0 results in maximum start-up time for the default clock source. See Section 28.5 “System and Reset Characteristics” on page 291 for details.

2. The default setting of CKSEL3..0 results in internal RC oscillator at 8MHz. See Table 9-1 on page 24 for details.

3. The CKOUT fuse allow the system clock to be output on PORTB1. See Section 9.10 “Clock Output Buffer” on page 31 for details.

4. See Section 9.11 “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.

27.2.1 Latching of Fuses

The fuse values are latched when the device enters programming mode and changes of the fuse values will have no effect until the part leaves programming mode. This does not apply to the EESAVE fuse which will take effect once it is programmed. The fuses are also latched on power-up in normal mode.

27.3 Signature Bytes

All Atmel® microcontrollers have a three-byte signature code which identifies the device. This code can be read in both serial and parallel mode, also when the device is locked. The three bytes reside in a separate address space.

For the Atmel ATmega164P-B/324P-B/644P-B the signature bytes are given in Table 27-6.

Table 27-6. Device and JTAG ID

	Signature Bytes Address			JTAG	
Part	0x000	0x001	0x002	Part Number	Manufacture ID
ATmega164P-B	0x1E	0x94	0x0A	940A	0x1F
ATmega324P-B	0x1E	0x95	0x11	9511	0x1F
ATmega644P-B	0x1E	0x96	0x0A	960A	0x1F

27.4 Calibration Byte

The Atmel ATmega164P-B/324P-B/644P-B has a byte 2 calibration values for the internal RC oscillator. The 3V calibration byte resides in the address 0x0001 in the signature address space and the 5V calibration byte resides in the address 0x0003. During reset, the 3V calibration byte is automatically written into the OSCCAL register to ensure correct frequency of the calibrated RC oscillator.

27.5 Page Size

Table 27-7. No. of Words in a Page and No. of Pages in the Flash

Device	Flash Size	Page Size	PCWORD	No. of Pages	PCPAGE	PCMSB
ATmega164P-B	8K words (16Kbytes)	64 words	PC[5:0]	128	PC[12:6]	12
ATmega324P-B	16K words (32Kbytes)	64 words	PC[5:0]	256	PC[13:6]	13
ATmega644P-B	32K words (64Kbytes)	128 words	PC[6:0]	256	PC[14:6]	14

Table 27-8. No. of Words in a Page and No. of Pages in the EEPROM

Device	EEPROM Size	Page Size	PCWORD	No. of Pages	PCPAGE	EEAMSB
ATmega164P-B	512bytes	4 bytes	EEA[1:0]	128	EEA[8:2]	8
ATmega324P-B	1Kbytes	4 bytes	EEA[1:0]	256	EEA[9:2]	9
ATmega644P-B	2Kbytes	8 bytes	EEA[2:0]	256	EEA[10:2]	10

27.6 Parallel Programming Parameters, Pin Mapping, and Commands

This section describes how to parallel program and verify flash program memory, EEPROM data memory, memory lock bits, and fuse bits in the Atmel ATmega164P-B/324P-B/644P-B. Pulses are assumed to be at least 250ns unless otherwise noted.

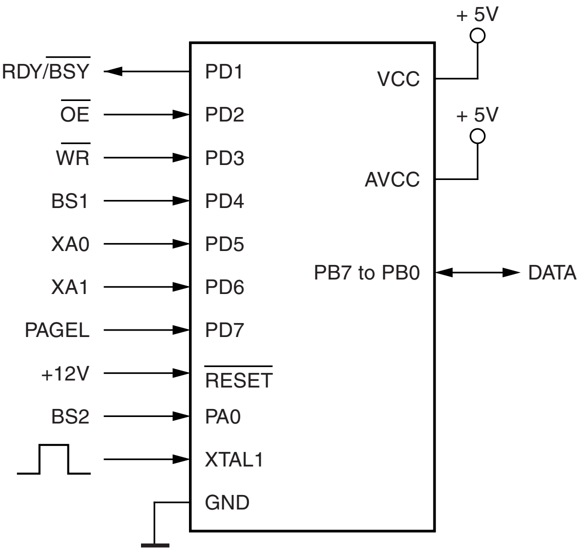
27.6.1 Signal Names

In this section, some pins of the Atmel ATmega164P-B/324P-B/644P-B are referenced by signal names describing their functionality during parallel programming, see Figure 27-1 and Figure 27-9 on page 259. 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 27-12 on page 260.

When pulsing $\overline{\text{WR}}$ or $\overline{\text{OE}}$, the command loaded determines the action executed. The different commands are shown in Table 27-13 on page 260.

Figure 27-1. Parallel Programming(1)



Note: 1. Unused pins should be left floating.

Table 27-9. Pin Name Mapping

Signal Name in Programming Mode	Pin Name	I/O	Function
RDY/$\overline{\text{BSY}}$	PD1	O	0: Device is busy programming, 1: Device is ready for new command.
$\overline{\text{OE}}$	PD2	I	Output enable (active low).
$\overline{\text{WR}}$	PD3	I	Write pulse (active low).
BS1	PD4	I	Byte select 1.
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.
DATA	PB7-0	I/O	Bi-directional data bus (output when $\overline{\text{OE}}$ is low).

Table 27-10. BS2 and BS1 Encoding

BS2	BS1	Flash / EEPROM Address	Flash Data Loading / Reading	Fuse Programming	Reading Fuse and Lock Bits
0	0	Low byte	Low byte	Low byte	Fuse low byte
0	1	High byte	High byte	High byte	Lock bits
1	0	Extended high byte	Reserved	Extended byte	Extended fuse byte
1	1	Reserved	Reserved	Reserved	Fuse high byte

,

Table 27-11. 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 27-12. XA1 and XA0 Encoding

XA1	XA0	Action when XTAL1 is Pulsed
0	0	Load flash or EEPROM address (high or low address byte determined by BS2 and 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 27-13. Command Byte Bit Encoding

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

27.7 Parallel Programming

27.7.1 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 27-11 on page 260 to “0000” and wait at least 100ns.
4. Apply 11.5 - 12.5V to $\overline{\text{RESET}}$. Any activity on prog_enable pins within 100ns after +12V has been applied to $\overline{\text{RESET}}$, will cause the device to fail entering programming mode.
5. Wait at least 50µs before sending a new command.

27.7.2 Considerations for Efficient Programming

The loaded command and address are retained in the device during programming. For efficient programming, the following should be considered.

- The command needs only be loaded once when writing or reading multiple memory locations.
- Skip writing the data value 0xFF, that is the contents of the entire EEPROM (unless the EESAVE fuse is programmed) and flash after a chip erase.
- Address high byte needs only be loaded before programming or reading a new 256 word window in flash or 256 byte EEPROM. This consideration also applies to signature bytes reading.

27.7.3 Chip Erase

The chip erase will erase the flash and EEPROM(1) memories plus lock bits. The lock bits are not reset until the program memory has been completely erased. The fuse bits are not changed. A chip erase must be performed before the flash and/or EEPROM are reprogrammed.

Note: 1. The 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.

27.7.4 Programming the Flash

The flash is organized in pages, see Table 27-7 on page 258. 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 (address bits 7..0)

1. Set XA1, XA0 to “00”. This enables address loading.
2. Set BS2, BS1 to “00”. This selects the address low byte.
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 27-3 on page 263 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 27-2 on page 263. Note that if less than eight bits are required to address words in the page (Page size < 256), the most significant bit(s) in the address low byte are used to address the page when performing a page write.

G. Load address high byte (address bits15..8)

1. Set XA1, XA0 to “00”. This enables address loading.
2. Set BS2, BS1 to “01”. This selects the address high byte.
3. Set DATA = address high byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the address high byte.

H. Load address extended high byte (address bits 23..16)

1. Set XA1, XA0 to “00”. This enables address loading.
2. Set BS2, BS1 to “10”. This selects the address extended high byte.
3. Set DATA = address extended high byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the address high byte.

I. Program page

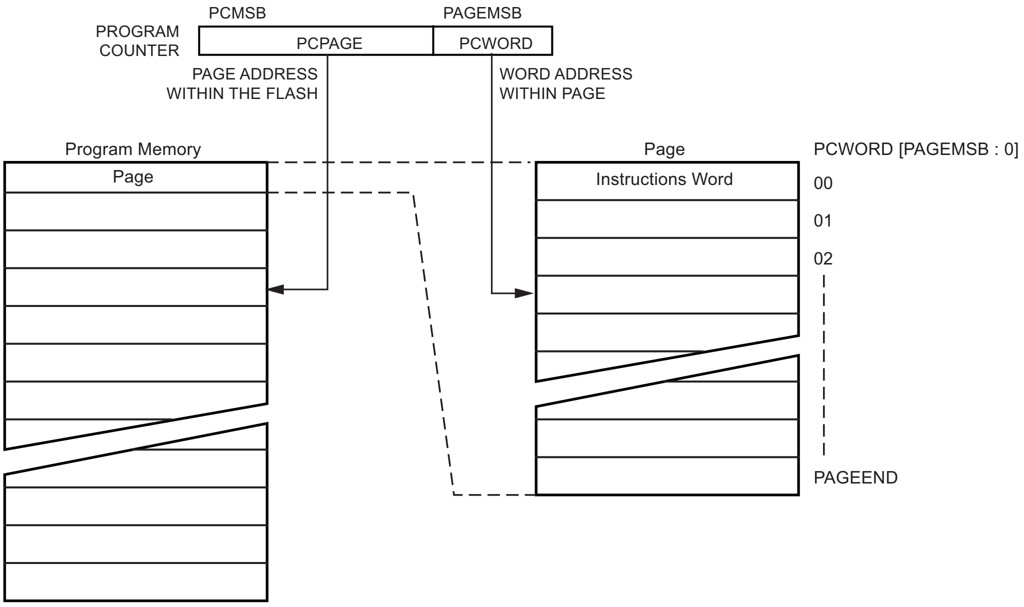
1. Set BS2, BS1 to “00”
2. Give $\overline{\text{WR}}$ a negative pulse. This starts programming of the entire page of data. RDY/$\overline{\text{BSY}}$ goes low.
3. Wait until RDY/$\overline{\text{BSY}}$ goes high (See Figure 27-3 on page 263 for signal waveforms).

J. Repeat B through I until the entire flash is programmed or until all data has been programmed.

K. End page programming

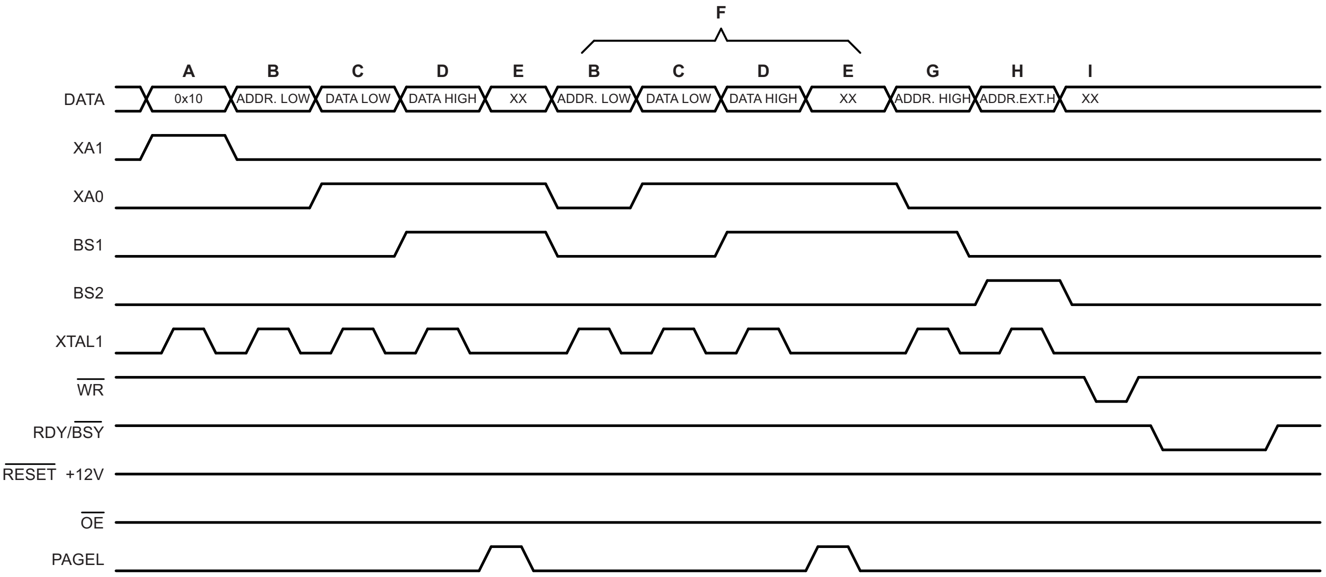
1. 1. Set XA1, XA0 to “10”. This enables command loading.
2. Set DATA to “0000 0000”. This is the command for No operation.
3. Give XTAL1 a positive pulse. This loads the command, and the internal write signals are reset.

Figure 27-2. Addressing the Flash Which is Organized in Pages[1]



Note: 1. PCPAGE and PCWORD are listed in Table 27-7 on page 258.

Figure 27-3. Programming the Flash Waveforms[1]



Note: 1. “XX” is don’t care. The letters refer to the programming description above.

27.7.5 Programming the EEPROM

The EEPROM is organized in pages, see Table 27-8 on page 258. When programming the EEPROM, the program data is latched into a page buffer. This allows one page of data to be programmed simultaneously. The programming algorithm for the EEPROM data memory is as follows (refer to Section 27.7.4 "Programming the Flash" on page 262 for details on command, address and data loading):

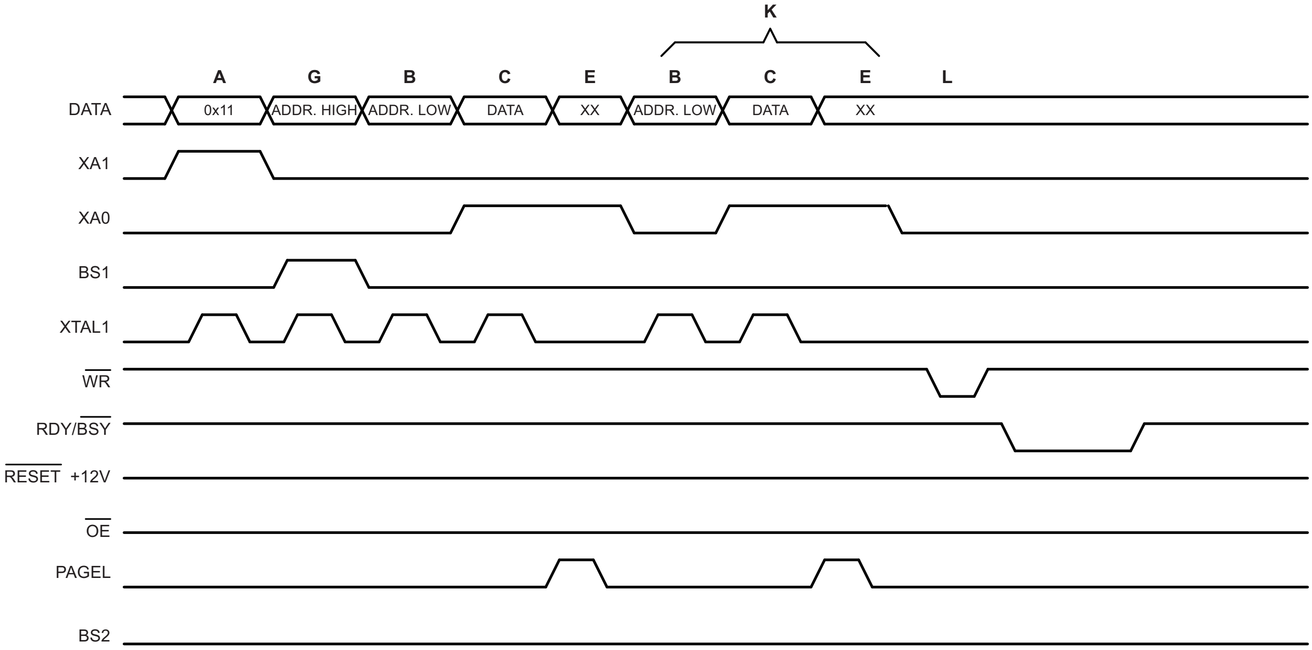
1. A: Load command "0001 0001".
2. G: Load address high byte (0x00 - 0xFF).
3. B: Load address low byte (0x00 - 0xFF).
4. C: Load data (0x00 - 0xFF).
5. E: Latch data (give PAGEL a positive pulse).

K: Repeat 3 through 5 until the entire buffer is filled.

L: Program EEPROM page

1. Set BS2, BS1 to "00".
2. Give $\overline{\text{WR}}$ a negative pulse. This starts programming of the EEPROM page. RDY/$\overline{\text{BSY}}$ goes low.
3. Wait until to RDY/$\overline{\text{BSY}}$ goes high before programming the next page (See Figure 27-4 for signal waveforms).

Figure 27-4. Programming the EEPROM Waveforms



27.7.6 Reading the Flash

The algorithm for reading the flash memory is as follows (refer to Section 27.7.4 “Programming the Flash” on page 262 for details on command and address loading):

1. A: Load command “0000 0010”.
2. H: Load address extended byte (0x00- 0xFF).
3. G: Load address high byte (0x00 - 0xFF).
4. B: Load address low byte (0x00 - 0xFF).
5. Set $\overline{\text{OE}}$ to “0”, and BS1 to “0”. The flash word low byte can now be read at DATA.
6. Set BS to “1”. The flash word high byte can now be read at DATA.
7. Set $\overline{\text{OE}}$ to “1”.

27.7.7 Reading the EEPROM

The algorithm for reading the EEPROM memory is as follows (refer to Section 27.7.4 “Programming the Flash” on page 262 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 $\overline{\text{OE}}$ to “0”, and BS1 to “0”. The EEPROM data byte can now be read at DATA.
5. Set $\overline{\text{OE}}$ to “1”.

27.7.8 Programming the Fuse Low Bits

The algorithm for programming the fuse low bits is as follows (refer to Section 27.7.4 “Programming the Flash” on page 262 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 $\overline{\text{WR}}$ a negative pulse and wait for RDY/$\overline{\text{BSY}}$ to go high.

27.7.9 Programming the Fuse High Bits

The algorithm for programming the fuse high bits is as follows (refer to Section 27.7.4 “Programming the Flash” on page 262 for details on command and data loading):

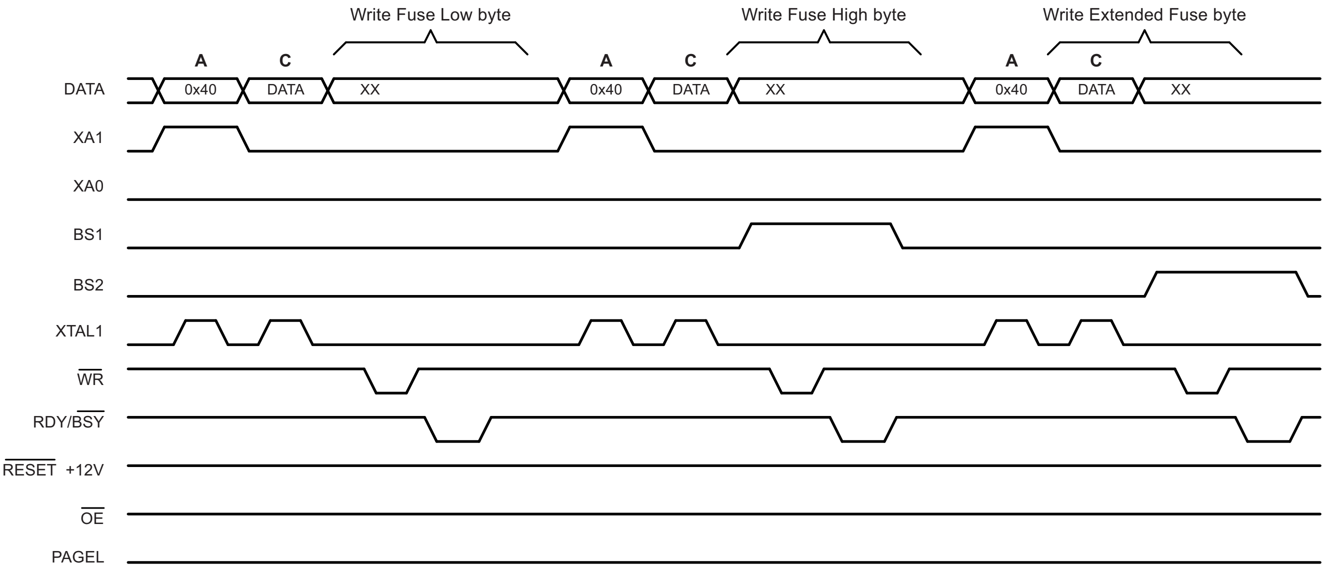
1. A: Load command “0100 0000”.
2. C: Load data low byte. Bit n = “0” programs and bit n = “1” erases the fuse bit.
3. Set BS2, BS1 to “01”. This selects high data byte.
4. Give $\overline{\text{WR}}$ a negative pulse and wait for RDY/$\overline{\text{BSY}}$ to go high.
5. Set BS2, BS1 to “00”. This selects low data byte.

27.7.10 Programming the Extended Fuse Bits

The algorithm for programming the extended fuse bits is as follows (refer to 27.7.4 “Programming the Flash” on page 262 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 BS2, BS1 to “10”. This selects extended data byte.
4. 4. Give $\overline{\text{WR}}$ a negative pulse and wait for RDY/$\overline{\text{BSY}}$ to go high.
5. 5. Set BS2, BS1 to “00”. This selects low data byte.

Figure 27-5. Programming the FUSES Waveforms



27.7.11 Programming the Lock Bits

The algorithm for programming the lock bits is as follows (refer to Section 27.7.4 "Programming the Flash" on page 262 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.

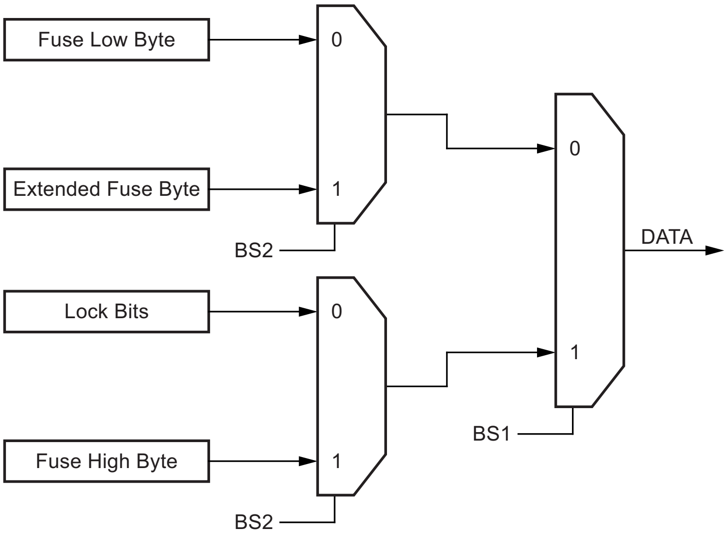
The lock bits can only be cleared by executing chip erase.

27.7.12 Reading the Fuse and Lock Bits

The algorithm for reading the fuse and lock bits is as follows (refer to Section 27.7.4 "Programming the Flash" on page 262 for details on command loading):

1. A: Load command "0000 0100".
2. Set $\overline{OE}$ to "0", and BS2, BS1 to "00". The status of the fuse low bits can now be read at DATA ("0" means programmed).
3. Set $\overline{OE}$ to "0", and BS2, BS1 to "11". The status of the fuse high bits can now be read at DATA ("0" means programmed).
4. Set OE to "0", and BS2, BS1 to "10". The status of the extended fuse bits can now be read at DATA ("0" means programmed).
5. Set $\overline{OE}$ to "0", and BS2, BS1 to "01". The status of the lock bits can now be read at DATA ("0" means programmed).
6. Set $\overline{OE}$ to "1".

Figure 27-6. **Mapping Between BS1, BS2 and the Fuse and Lock Bits During Read**



27.7.13 Reading the Signature Bytes

The algorithm for reading the signature bytes is as follows (refer to Section 27.7.4 “Programming the Flash” on page 262 for details on command and address loading):

1. A: Load command “0000 1000”.
2. B: Load address low byte (0x00 - 0x02).
3. Set $\overline{\text{OE}}$ to “0”, and BS to “0”. The selected signature byte can now be read at DATA.
4. Set $\overline{\text{OE}}$ to “1”.

27.7.14 Reading the Calibration Byte

The algorithm for reading the calibration byte is as follows (refer to Section 27.7.4 “Programming the Flash” on page 262 for details on command and address loading):

1. A: Load command “0000 1000”.
2. B: Load address low byte, 0x00.
3. Set $\overline{\text{OE}}$ to “0”, and BS1 to “1”. The calibration byte can now be read at DATA.
4. Set $\overline{\text{OE}}$ to “1”.

27.7.15 Parallel Programming Characteristics

Table 27-14. Parallel Programming Characteristics, V_{CC} = 5V ±10%

Parameter	Symbol	Min	Typ.	Max	Unit
Programming enable voltage	V_{PP}	11.5		12.5	V
Programming enable current	I_{PP}			250	μA
Data and control valid before XTAL1 high	t_{DVXH}	67			ns
XTAL1 low to XTAL1 high	t_{XLXH}	200			ns
XTAL1 pulse width high	t_{XHXL}	150			ns
Data and control hold after XTAL1 low	t_{XLDX}	67			ns
XTAL1 low to $\overline{\text{WR}}$ low	t_{XLWL}	0			ns
XTAL1 low to PAGEL high	t_{XLPH}	0			ns
PAGEL low to XTAL1 high	t_{PLXH}	150			ns
BS1 valid before PAGEL high	t_{BVPH}	67			ns
PAGEL pulse width high	t_{PHPL}	150			ns
BS1 hold after PAGEL low	t_{PLBX}	67			ns
BS2/1 hold after $\overline{\text{WR}}$ low	t_{WLBX}	67			ns
PAGEL low to $\overline{\text{WR}}$ low	t_{PLWL}	67			ns
BS2/1 valid to $\overline{\text{WR}}$ low	t_{BVWL}	67			ns
$\overline{\text{WR}}$ pulse width low	t_{WLWH}	150			ns
$\overline{\text{WR}}$ low to RDY/$\overline{\text{BSY}}$ low	t_{WLRL}	0		1	μs
$\overline{\text{WR}}$ low to RDY/$\overline{\text{BSY}}$ high(1)	t_{WLRH}	3.7		4.5	ms
$\overline{\text{WR}}$ low to RDY/$\overline{\text{BSY}}$ high for chip erase(2)	t_{WLRH_CE}	7.5		9	ms
XTAL1 low to $\overline{\text{OE}}$ low	t_{XLOL}	0			ns
BS1 valid to DATA valid	t_{BVDV}	0		250	ns
$\overline{\text{OE}}$ low to DATA valid	t_{OLDV}			250	ns
$\overline{\text{OE}}$ high to DATA tri-stated	t_{OHDZ}			250	ns

Notes: 1. t_{WLRH} is valid for the write flash, Write EEPROM, write fuse bits and write lock bits commands.

2. t_{WLRH_CE} is valid for the chip erase command.

Figure 27-7. Parallel Programming Timing, Including some General Timing Requirements

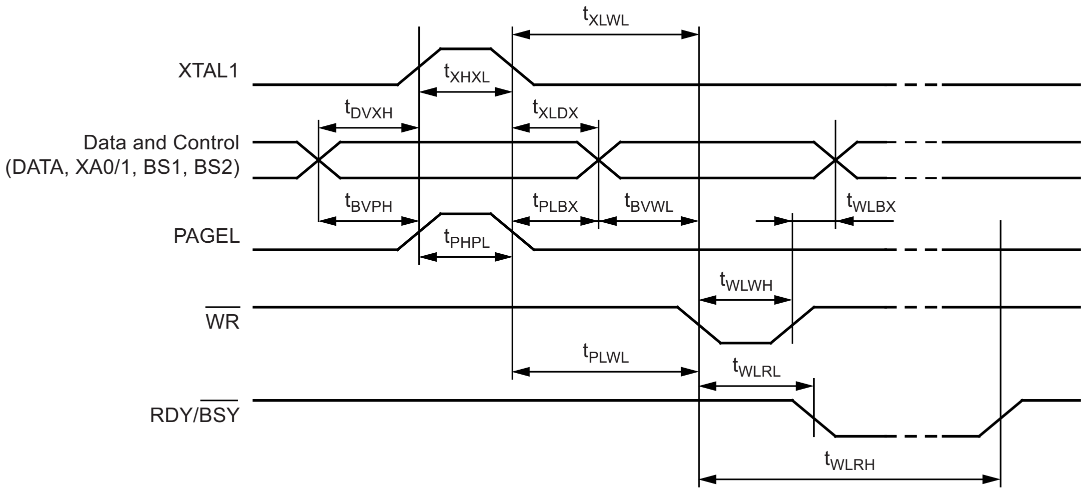
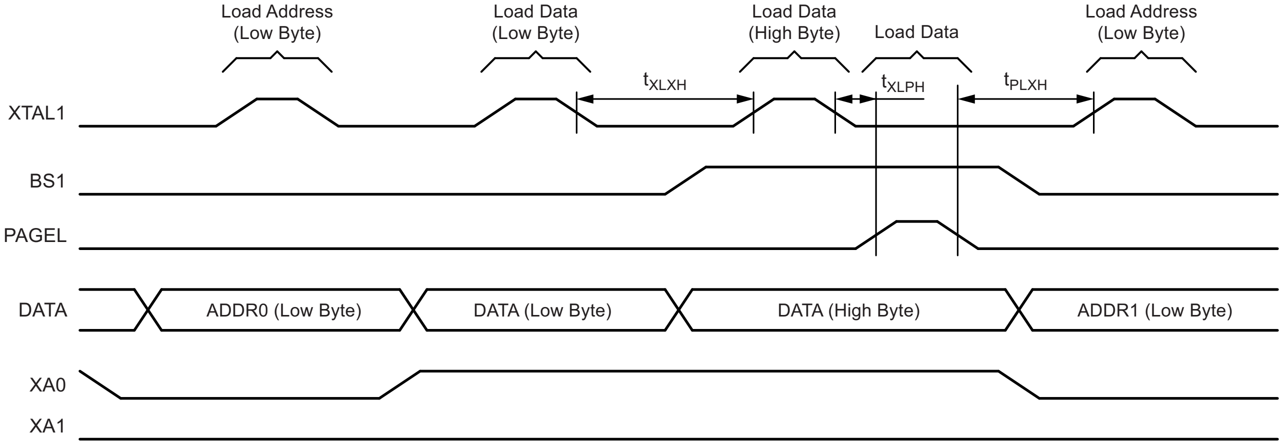
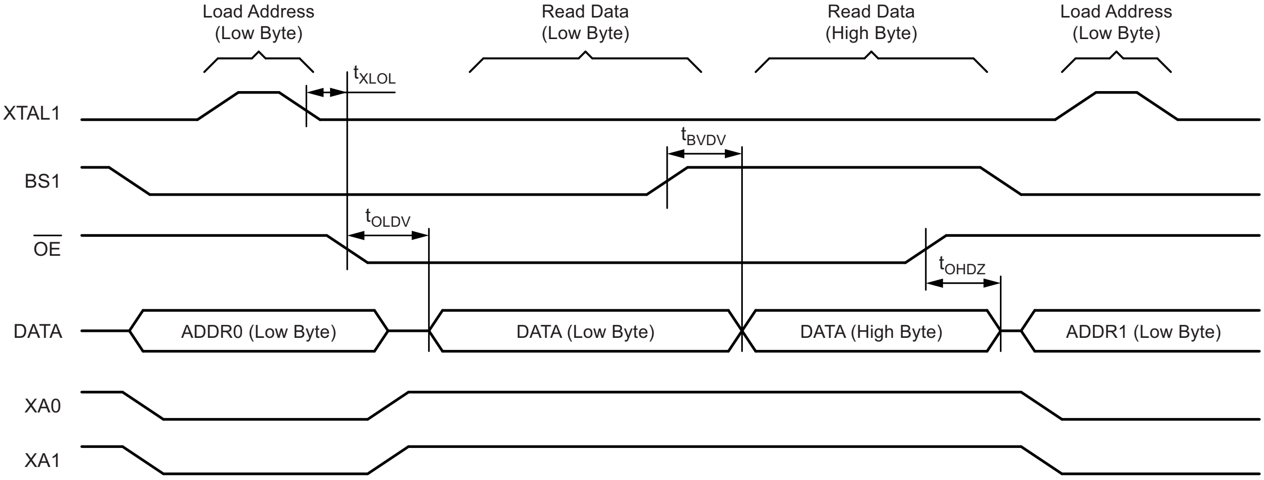


Figure 27-8. Parallel Programming Timing, Loading Sequence with Timing Requirements[(1)]



Note: The timing requirements shown in Figure 27-7 on page 268 (i.e., t_{DVXH}, t_{XHXL}, and t_{XLDX}) also apply to loading operation.

Figure 27-9. Parallel Programming Timing, Reading Sequence (within the Same Page) with Timing Requirements[(1)]



Note: 1. The timing requirements shown in Table 27-7 on page 268 (i.e., t_{DVXH}, t_{XHXL}, and t_{XLDX}) also apply to reading operation.

27.8 Serial Downloading

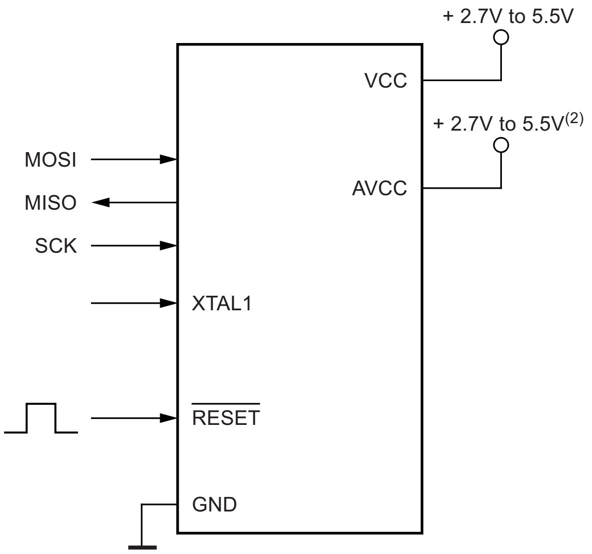
Both the flash and EEPROM memory arrays can be programmed using a serial programming bus while $\overline{\text{RESET}}$ is pulled to GND. The serial programming interface consists of pins SCK, MOSI (input) and MISO (output). After $\overline{\text{RESET}}$ is set low, the programming enable instruction needs to be executed first before program/erase operations can be executed. NOTE, in Table 27-15, the pin mapping for serial programming is listed. Not all packages use the SPI pins dedicated for the internal serial peripheral interface - SPI.

27.8.1 Serial Programming Pin Mapping

Table 27-15. Pin Mapping Serial Programming

Symbol	Pins (PDIP-40)	Pins (TQFP/MLF-44)	I/O	Description
MOSI	PB5	PB5	I	Serial data in
MISO	PB6	PB6	O	Serial data out
SCK	PB7	PB7	I	Serial clock

Figure 27-10. 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 2.7 to 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} < 12MHz, 3 CPU clock cycles for f_{ck} ≥ 12MHz
High:	> 2 CPU clock cycles for f_{ck} < 12MHz, 3 CPU clock cycles for f_{ck} ≥ 12MHz

27.8.2 Serial Programming Algorithm

When writing serial data to the Atmel® ATmega164P-B/324P-B/644P-B, data is clocked on the rising edge of SCK.

When reading data from the Atmel ATmega164P-B/324P-B/644P-B, data is clocked on the falling edge of SCK. See Figure 27-12 on page 273 for timing details.

To program and verify the ATmega164P-B/324P-B/644P-B in the serial programming mode, the following sequence is recommended (see four byte instruction formats in Table 27-17 on page 272):

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 20ms and enable serial programming by sending the programming enable serial instruction to pin MOSI.
3. The serial programming instructions will not work if the communication is out of synchronization. When in sync. the second byte (0x53), will echo back when issuing the third byte of the programming enable instruction. Whether the echo is correct or not, all four bytes of the instruction must be transmitted. If the 0x53 did not echo back, give $\overline{\text{RESET}}$ a positive pulse and issue a new programming enable command.
4. The flash is programmed one page at a time. The memory page is loaded one byte at a time by supplying the 7 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 address lines 15.8. Before issuing this command, make sure the instruction load extended address byte has been used to define the MSB of the address. The extended address byte is stored until the command is re-issued, i.e., the command needs only be issued for the first page, and when crossing the 64KWord boundary. If polling (RDY/$\overline{\text{BSY}}$) is not used, the user must wait at least t_{WD_FLASH} before issuing the next page. (See Table 27-16 on page 271.) Accessing the serial programming interface before the flash write operation completes can result in incorrect programming.
5. 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 is not used, the user must wait at least t_{WD_EEPROM} before issuing the next byte.
   (See Table 27-16 on page 271.) In a chip erased device, no 0xFFs 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. When reading the flash memory, use the instruction load extended address byte to define the upper address byte, which is not included in the read program memory instruction. The extended address byte is stored until the command is re-issued, i.e., the command needs only be issued for the first page, and when crossing the 64KWord boundary.
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 27-16. Minimum Wait Delay Before Writing the Next Flash or EEPROM Location

Symbol	Minimum Wait Delay
t_{WD_FLASH}	4.5ms
t_{WD_EEPROM}	3.6ms
t_{WD_ERASE}	9.0ms

27.9 Serial Programming Instruction set

Table 27-17 and Figure 27-11 on page 273 describes the instruction set.

Table 27-17. Serial Programming Instruction Set (Hexadecimal values)

	Instruction Format			
Instruction/Operation	**Byte 1**	**Byte 2**	**Byte 3**	**Byte4**
Programming enable	$AC	$53	$00	$00
Chip erase (program memory/EEPROM)	$AC	$80	$00	$00
Poll RDY/$\overline{\text{BSY}}$	$F0	$00	$00	data byte out
Load instructions				
Load extended address byte[1]	$4D	$00	Extended adr	$00
Load program memory page, high byte	$48	$00	adr LSB	high data byte in
Load program memory page, low byte	$40	$00	adr LSB	low data byte in
Load EEPROM memory Page (page access)	$C1	$00	0000 000aa	data byte in
Read instructions				
Read program memory, high byte	$28	adr MSB	adr LSB	high data byte out
Read program memory, low byte	$20	adr MSB	adr LSB	low data byte out
Read EEPROM memory	$A0	0000 00aa	aaaa aaaa	data byte out
Read lock bits	$58	$00	$00	data byte out
Read signature byte	$30	$00	0000 000aa	data byte out
Read fuse bits	$50	$00	$00	data byte out
Read fuse high bits	$58	$08	$00	data byte out
Read extended fuse its	$50	$08	$00	data byte out
Read calibration byte	$38	$00	$00	data byte out
Write instructions[6]				
Write program memory page	$4C	adr MSB	adr LSB	$00
Write EEPROM memory	$C0	0000 00aa	aaaa aaaa	data byte in
Write EEPROM memory page (page access)	$C2	0000 00aa	aaaa aa00	$00
Write lock bits	$AC	$E0	$00	data byte in
Write fuse bits	$AC	$A0	$00	data byte in
Write fuse high bits	$AC	$A8	$00	data byte in
Write extended fuse bits	$AC	$A4	$00	data byte in

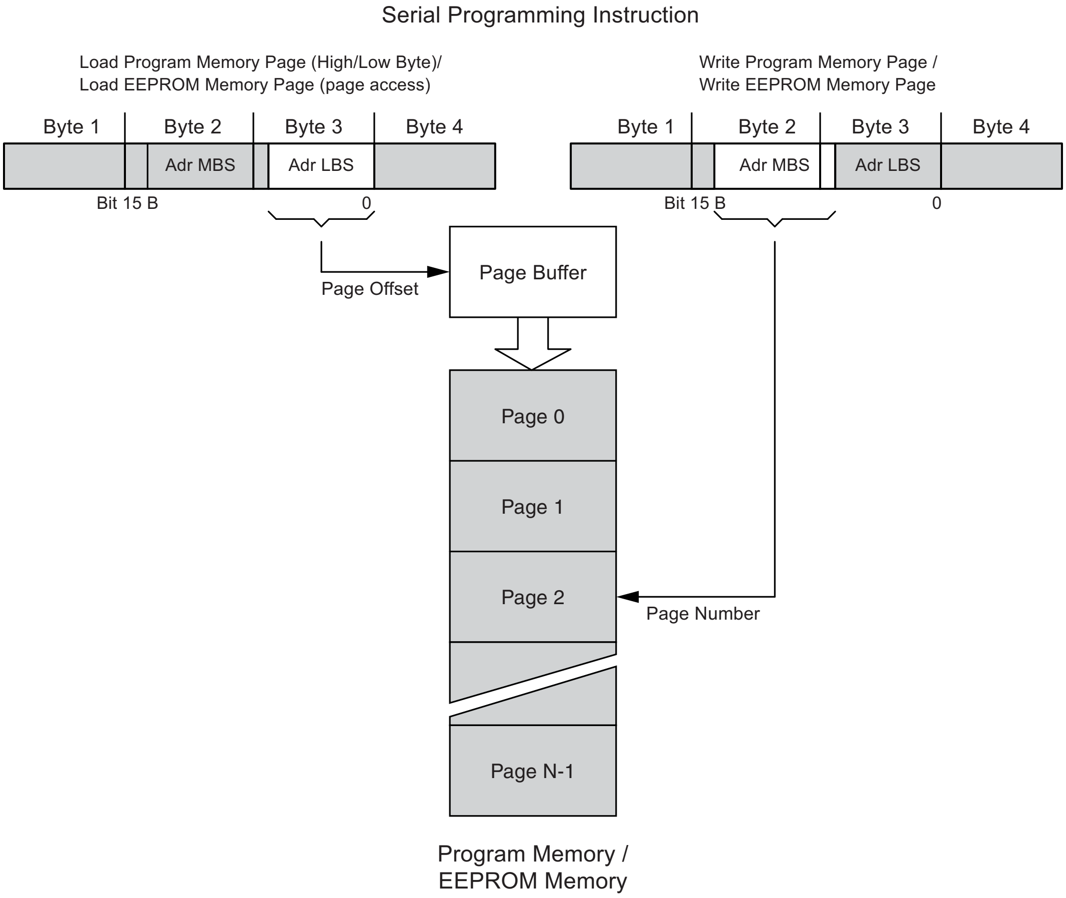
Notes:
1. Not all instructions are applicable for all parts.
2. a = address.
3. Bits are programmed ‘0’, unprogrammed ‘1’.
4. To ensure future compatibility, unused fuses and lock bits should be unprogrammed (‘1’).
5. Refer to the corresponding section for fuse and lock bits, calibration and signature bytes and page size.
6. Instructions accessing program memory use a word address. This address may be random within the page range.
7. See http://www.atmel.com/avr for application notes regarding programming and programmers.

If the LSB in RDY/BSY data byte out is ‘1’, a programming operation is still pending. Wait until this bit returns ‘0’ before the next instruction is carried out.

Within the same page, the low data byte must be loaded prior to the high data byte.

After data is loaded to the page buffer, program the EEPROM page, see Figure 27-11 on page 273.

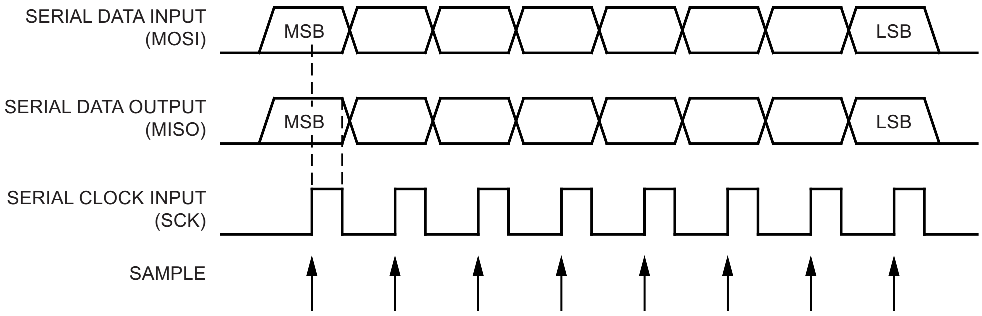
Figure 27-11. Serial Programming Instruction Example



27.9.1 Serial Programming Characteristics

For characteristics of the serial programming module See Section 28.7 “SPI Timing Characteristics” on page 292.

Figure 27-12. Serial Programming Waveforms



27.10 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 MCUCR 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 datasheet, the LSB is shifted in and out first of all shift registers.

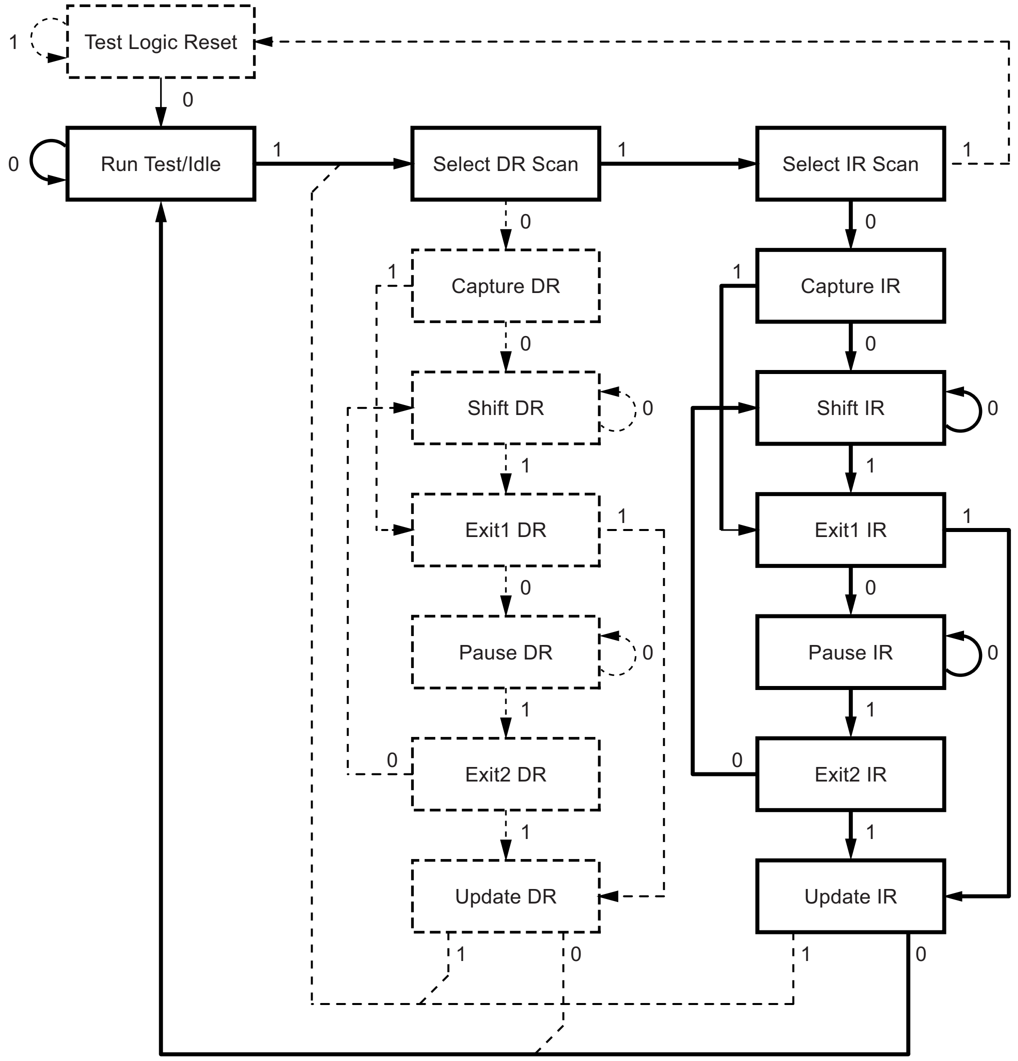
27.10.1 Programming Specific JTAG Instructions

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 27-13 on page 275.

Figure 27-13. State Machine Sequence for Changing the Instruction Word



27.10.2 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.

27.10.3 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.

27.10.4 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

27.10.5 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.

27.10.6 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.

27.10.7 Data Registers

The data registers are selected by the JTAG instruction registers described in section Section 27.10.1 “Programming Specific JTAG Instructions” on page 274. The data registers relevant for programming operations are:

- Reset register
- Programming enable register
- Programming command register
- Flash data byte register

27.10.8 Reset Register

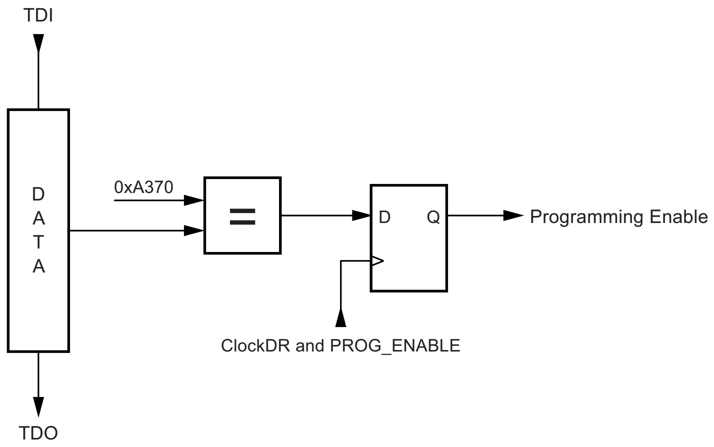
The reset register is a test data register used to reset the part during programming. It is required to reset the part before entering programming mode.

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 Section 9.2 "Clock Sources" on page 24) 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 25-2 on page 232.

27.10.9 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 27-14. Programming Enable Register



27.10.10 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 27-18 on page 279. The state sequence when shifting in the programming commands is illustrated in Figure 27-16 on page 282.

Figure 27-15. Programming Command Register

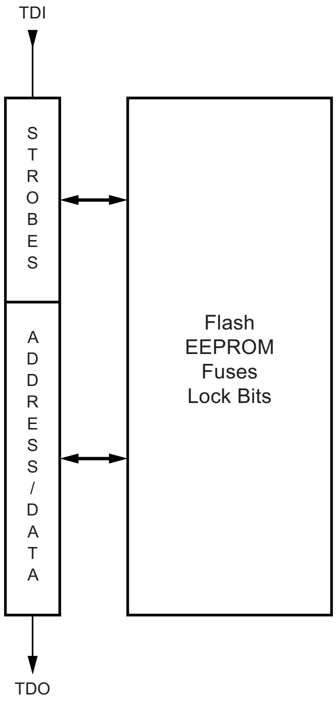


Table 27-18. JTAG Programming Instruction

Set a = address high bits, b = address low bits, c = address extended 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 0110001_10000000 0110011_10000000 0110011_10000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	
1b. Poll for chip erase complete	0110011_10000000	xxxxxox_xxxxxxxx	(2)
2a. Enter flash write	0100011_00010000	xxxxxxx_xxxxxxxx	
2b. Load address extended high byte	0001011_cccccccc	xxxxxxx_xxxxxxxx	(10)
2c. Load address high byte	0000111_aaaaaaaa	xxxxxxx_xxxxxxxx	
2d. Load address low byte	0000011_bbbbbbbb	xxxxxxx_xxxxxxxx	
2e. Load data low byte	0010011_iiiiiiii	xxxxxxx_xxxxxxxx	
2f. Load data high byte	0010111_iiiiiiii	xxxxxxx_xxxxxxxx	
2g. Latch data	0110111_00000000 1110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
2h. Write flash page	0110111_00000000 0110101_00000000 0110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
2i. Poll for page write complete	0110111_00000000	xxxxxox_xxxxxxxx	(2)
3a. Enter flash read	0100011_00000010	xxxxxxx_xxxxxxxx	
3b. Load address extended high byte	0001011_cccccccc	xxxxxxx_xxxxxxxx	(10)
3c. Load address high byte	0000111_aaaaaaaa	xxxxxxx_xxxxxxxx	
3d. Load address low byte	0000011_bbbbbbbb	xxxxxxx_xxxxxxxx	
3e. Read data low and high byte	0110010_00000000 0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_oooooooo xxxxxxx_oooooooo	Low byte High byte
4a. Enter EEPROM write	0100011_00010001	xxxxxxx_xxxxxxxx	
4b. Load address high byte	0000111_aaaaaaaa	xxxxxxx_xxxxxxxx	(10)
4c. Load address low byte	0000011_bbbbbbbb	xxxxxxx_xxxxxxxx	
4d. Load data byte	0010011_iiiiiiii	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 27-3 on page 256
7. The bit mapping for fuses high byte is listed in Table 27-4 on page 257
8. The bit mapping for fuses low byte is listed in Table 27-5 on page 257
9. The bit mapping for lock bits byte is listed in Table 27-1 on page 255
10. Address bits exceeding PCMSB and EEAMSB (Table 27-7 on page 258 and Table 27-8 on page 258) are don't care
11. All TDI and TDO sequences are represented by binary digits (0b...).

Table 27-18. JTAG Programming Instruction (Continued)

Set a = address high bits, b = address low bits, c = address extended bits, H = 0 - Low byte, 1 - High byte,
o = data out, i = data in, x = don't care

Instruction	TDI Sequence	TDO Sequence	Notes
4e. Latch data	0110111_00000000 1110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
4f. Write EEPROM page	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
4g. Poll for page write complete	0110011_00000000	xxxxxox_xxxxxxxx	(2)
5a. Enter EEPROM read	0100011_00000011	xxxxxxx_xxxxxxxx	
5b. Load address high byte	0000111_aaaaaaaa	xxxxxxx_xxxxxxxx	(10)
5c. Load address Low Byte	0000011_bbbbbbbb	xxxxxxx_xxxxxxxx	
5d. Read data byte	0110011_bbbbbbbb 0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx 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 0111001_00000000 0111011_00000000 0111011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
6d. Poll for fuse write complete	0110111_00000000	xxxxxox_xxxxxxxx	(2)
6e. Load data low byte(7)	0010011_iiiiiiii	xxxxxxx_xxxxxxxx	(3)
6f. Write fuse high byte	0110111_00000000 0110101_00000000 0110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
6g. Poll for fuse write complete	0110111_00000000	xxxxxox_xxxxxxxx	(2)
6h. Load data low byte(7)	0010011_iiiiiiii	xxxxxxx_xxxxxxxx	(3)
6i. Write fuse low byte	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)

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 27-3 on page 256
7. The bit mapping for fuses high byte is listed in Table 27-4 on page 257
8. The bit mapping for fuses low byte is listed in Table 27-5 on page 257
9. The bit mapping for lock bits byte is listed in Table 27-1 on page 255
10. Address bits exceeding PCMSB and EEAMSB (Table 27-7 on page 258 and Table 27-8 on page 258) are don't care
11. All TDI and TDO sequences are represented by binary digits (0b...).

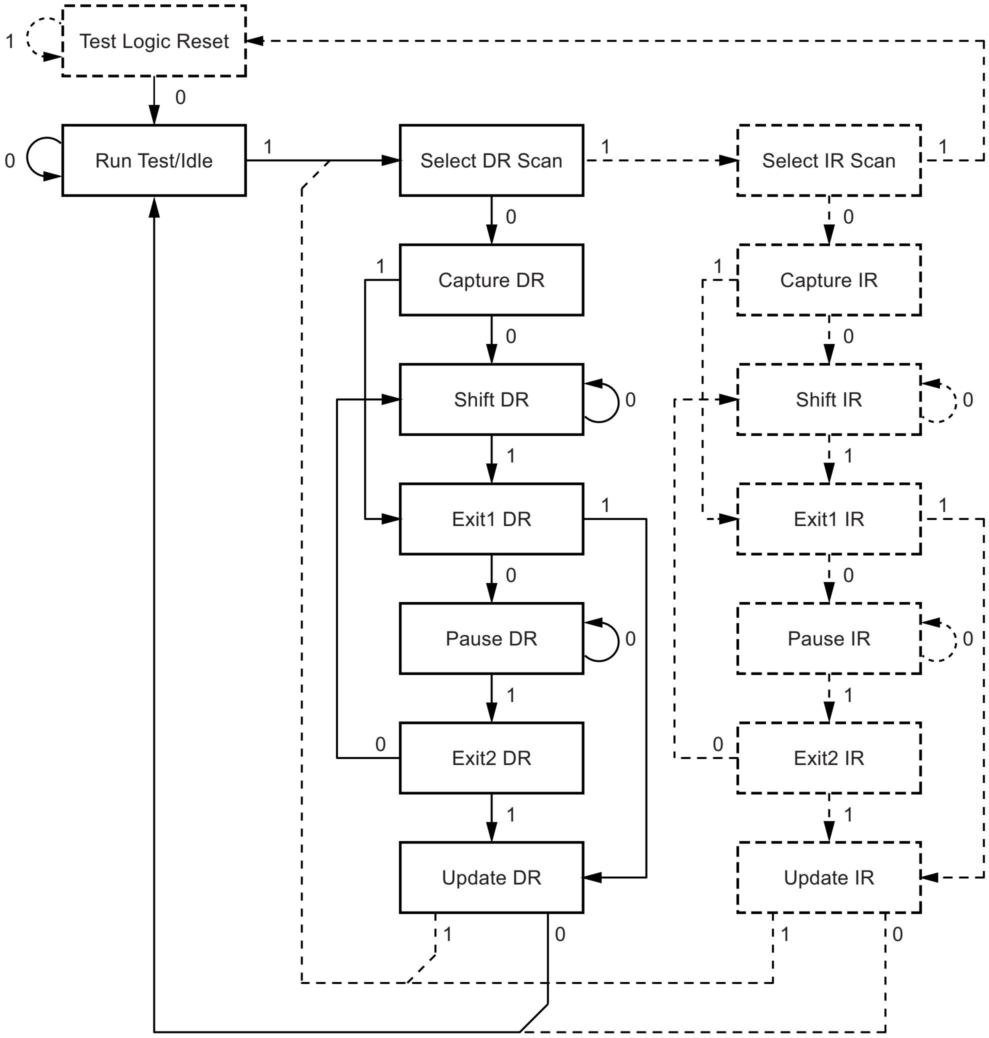
Table 27-18. JTAG Programming Instruction (Continued)

Set a = address high bits, b = address low bits, c = address extended bits, H = 0 - Low byte, 1 - High byte,
o = data out, i = data in, x = don't care

Instruction	TDI Sequence	TDO Sequence	Notes
6j. Poll for fuse write complete	0110011_00000000	xxxxxox_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 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
7d. Poll for lock bit write complete	0110011_00000000	xxxxxox_xxxxxxxx	(2)
8a. Enter fuse/lock bit read	0100011_00000100	xxxxxxx_xxxxxxxx	
8b. Read extended fuse byte[6]	0111010_00000000 0111011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_oooooooo	
8c. Read fuse high byte[7]	0111110_00000000 0111111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_oooooooo	
8d. Read fuse low byte[8]	0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_oooooooo	
8e. Read lock bits[9]	0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxoooooo	(5)
8f. Read fuses and lock bits	0111010_00000000 0111110_00000000 0110010_00000000 0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_oooooooo xxxxxxx_oooooooo xxxxxxx_oooooooo xxxxxxx_oooooooo	(5) Fuse ext. byte Fuse high byte Fuse low byte 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 0110011_00000000	xxxxxxx_xxxxxxxx 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 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_oooooooo	
11a. Load no operation command	0100011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx 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 27-3 on page 256
7. The bit mapping for fuses high byte is listed in Table 27-4 on page 257
8. The bit mapping for fuses low byte is listed in Table 27-5 on page 257
9. The bit mapping for lock bits byte is listed in Table 27-1 on page 255
10. Address bits exceeding PCMSB and EEAMSB (Table 27-7 on page 258 and Table 27-8 on page 258) are don't care
11. All TDI and TDO sequences are represented by binary digits (0b...).

Figure 27-16. State Machine Sequence for Changing/Reading the Data Word



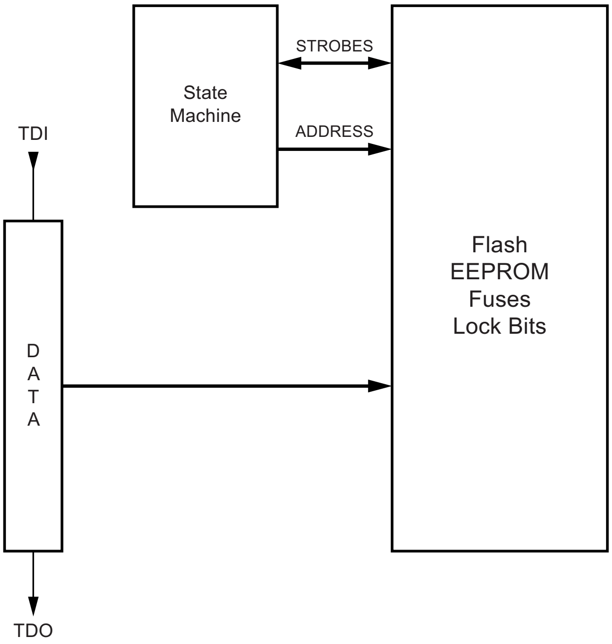
27.10.11 Flash Data Byte Register

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 27-17. 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.

27.10.12 Programming Algorithm

All references below of type “1a”, “1b”, and so on, refer to Table 27-18 on page 279.

27.10.13 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.

27.10.14 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.

27.10.15 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 27-14 on page 268).

27.10.16 Programming the Flash

Before programming the flash a chip erase must be performed, see Section 27.10.15 “Performing Chip Erase” on page 283.

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable flash write using programming instruction 2a.
3. Load address extended high byte using programming instruction 2b.
4. Load address high byte using programming instruction 2c.
5. Load address low byte using programming instruction 2d.
6. Load data using programming instructions 2e, 2f and 2g.
7. Repeat steps 5 and 6 for all instruction words in the page.
8. Write the page using programming instruction 2h.
9. Poll for flash write complete using programming instruction 2i, or wait for t_{WLRH} (refer to Table 27-14 on page 268).
10. Repeat steps 3 to 9 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, 2c and 2d. PCWORD (refer to Table 27-7 on page 258) 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 2h.
8. Poll for flash write complete using programming instruction 2i, or wait for t_{WLRH} (refer to Table 27-14 on page 268).
9. Repeat steps 3 to 8 until all data have been programmed.

27.10.17 Reading the Flash

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable flash read using programming instruction 3a.
3. Load address using programming instructions 3b, 3c and 3d.
4. Read data using programming instruction 3e.
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, 3c and 3d. PCWORD (refer to Table 27-7 on page 258) 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.

27.10.18 Programming the EEPROM

Before programming the EEPROM a chip erase must be performed, See Section 27.10.15 “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 27-14 on page 268).
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.

27.10.19 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.

27.10.20 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 27-14 on page 268).
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 27-14 on page 268).

27.10.21 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 27-14 on page 268).

27.10.22 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.

27.10.23 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.

27.10.24 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.
4. Read the calibration byte using programming instruction 10c.

28. Electrical Characteristics

28.1 Absolute Maximum Ratings*

Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Parameters	Min.	Typ.	Max.	Unit
Operating temperature	–55		+125	°C
Storage temperature	–65		+150	°C
Voltage on any pin except $\overline{\text{RESET}}$ with respect to ground to	–0.5		V_{CC} + 0.5	V
Voltage on $\overline{\text{RESET}}$ with respect to ground	–0.5		+13.0	V
Maximum operating voltage			6.0	V
DC current per I/O Pin		40.0		mA
DC current V_{CC} and GND pins		200.0		mA
Injection current at V_{CC} = 0V		±5.0$^{(1)}$		mA
Injection current at V_{CC} = 5V		±1.0		mA

Note: 1. Maximum current per port = ±30mA

28.2 DC Characteristics

Table 28-1. T_A = –40°C to 125°C, V_{CC} = 2.7V to 5.5V (Unless otherwise Noted)

Parameters	Condition	Symbol	Min.	Typ.	Max.	Unit
Input low voltage, except XTAL1 and reset pin	V_{CC} = 2.7V – 5.5V	V_{IL}	–0.5		0.3$V_{CC}^{(1)}$	V
Input low voltage, XTAL1 pin	V_{CC} = 2.7V – 5.5V	V_{IL1}	–0.5		0.1$V_{CC}^{(1)}$	V
Input low voltage, RESET pin	V_{CC} = 2.7V – 5.5V	V_{IL2}	–0.5		0.2$V_{CC}^{(1)}$	V
Input high voltage, except XTAL1 and RESET pins	V_{CC} = 2.7V – 5.5V	V_{IH}	0.6$V_{CC}^{(2)}$		V_{CC} + 0.5	V
Input high voltage, XTAL1 pin	V_{CC} = 2.7V – 5.5V	V_{IH1}	0.7$V_{CC}^{(2)}$		V_{CC} + 0.5	V

Notes:

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 (20mA at V_{CC} = 5V, 10mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:
   1.) The sum of all IOL, for ports PB0-PB7, XTAL2, PD0-PD7 should not exceed 100mA.
   2.) The sum of all IOL, for ports PA0-PA3, PC0-PC7 should not exceed 100mA.
   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 (20mA at V_{CC} = 5V, 10mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:
   1.) The sum of all IOH, for ports PB0-PB7, XTAL2, PD0-PD7 should not exceed 100mA.
   2.) The sum of all IOH, for ports PA0-PA3, PC0-PC7 should not exceed 100mA.
   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.

Table 28-1. T_A = –40°C to 125°C, V_{CC} = 2.7V to 5.5V (Unless otherwise Noted) (Continued)

Parameters	Condition	Symbol	Min.	Typ.	Max.	Unit
Input high voltage, RESET pin	V_{CC} = 2.7V – 5.5V	V_{IH2}	$0.9V_{CC}$(2)		V_{CC} + 0.5	V
Output low voltage(3)	I_{OL} = 20mA, V_{CC} = 5V I_{OL} = 10mA, V_{CC} = 3V	V_{OL}			0.8 0.5	V
Output high voltage(4)	I_{OH} = –20mA, V_{CC} = 5V I_{OH} = –10mA, V_{CC} = 3V	V_{OH}	4.1 2.3			V
Input leakage Current I/O pin	V_{CC} = 5.5V, pin low (absolute value)	I_{IL}			1	µA
Input leakage Current I/O pin	V_{CC} = 5.5V, pin high (absolute value)	I_{IH}			1	µA
Reset pull-up resistor		R_{RST}	30		60	kΩ
I/O pin pull-up resistor		R_{PU}	20		50	kΩ
Analog comparator Input offset voltage	V_{CC} = 5V, $0.1V_{CC} < V_{in} < V_{CC}$ – 100mV	V_{ACIO}		<10	40	mV
Analog comparator Input leakage current	V_{CC} = 5V $V_{in} = V_{CC}/2$	I_{ACLK}	–50		50	nA
Analog comparator Propagation delay	V_{CC} = 2.7V V_{CC} = 4.0V	t_{ACID}		750 500		ns

Notes:
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 (20mA at V_{CC} = 5V, 10mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:
   1.) The sum of all IOL, for ports PB0-PB7, XTAL2, PD0-PD7 should not exceed 100mA.
   2.) The sum of all IOL, for ports PA0-PA3, PC0-PC7 should not exceed 100mA.
   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 (20mA at V_{CC} = 5V, 10mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:
   1.) The sum of all IOH, for ports PB0-PB7, XTAL2, PD0-PD7 should not exceed 100mA.
   2.) The sum of all IOH, for ports PA0-PA3, PC0-PC7 should not exceed 100mA.
   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.

Table 28-2. T_A = –40°C to +125°C, V_{CC} = 2.7V to 5.5V (Unless otherwise Noted)

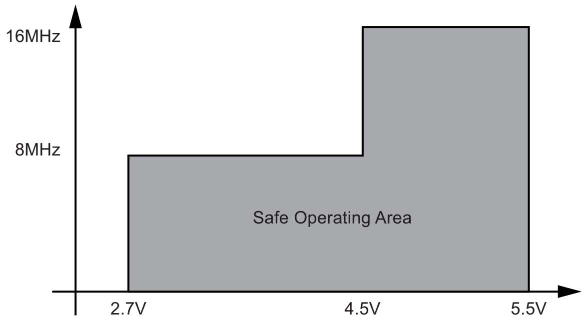
Parameter	Condition	Symbol	Min.	Typ.[2]	Max.	Unit
Power supply current[1]	Active 4MHz, V_{CC} = 3V	I_{CC}		1.4	2.75	mA
	Active 8MHz, V_{CC} = 5V			4.8	10	mA
	Active 16MHz, V_{CC} = 5V			8.6	15	mA
	Idle 4MHz, V_{CC} = 3V			0.25	1.5	mA
	Idle 8MHz, V_{CC} = 5V			1.0	3.0	mA
	Idle 16MHz, V_{CC} = 5V			1.9	4.0	mA
Power-save mode[3]	32kHz TOSC enabled, V_{CC} = 3V			0.6		µA
Power-down mode[3]	WDT enabled, V_{CC} = 3V			4.8	60	µA
	WDT enabled, V_{CC} = 5V			7.3	95	µA
	WDT disabled, V_{CC} = 3V			0.3	54	µA
	WDT disabled, V_{CC} = 5V			0.6	85	µA

Notes: 1. All bits set in the Section 10.12.3 “PRR0 – Power Reduction Register 0” on page 39.
2. Typical values at 25°C. maximum values are test limits in production.
3. The current consumption values include input leakage current.

28.3 Speed Grades

Maximum frequency is depending on V_{CC}. As shown in Figure 28-1, the maximum frequency versus V_{CC} curve is linear between 2.7V < V_{CC} < 4.5V.

Figure 28-1. Maximum Frequency versus V_{CC}, ATmega164P-B/324P-B/644P-B



28.4 Clock Characteristics

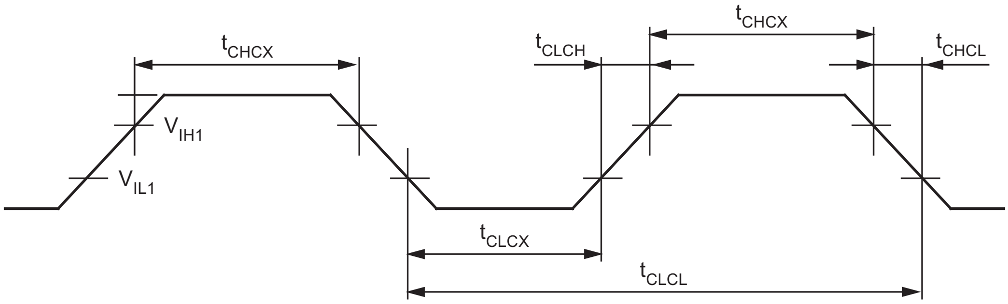
Table 28-3. Calibration Accuracy of Internal RC Oscillator

	Frequency	V_{CC}	Temperature	Calibration Accuracy
Default 3V factory calibration	8.0MHz	3V	25°C	±1.5%
	8.0MHz	2.7 - 5.5V(1)	–40°C / +125°C	±14%
5V factory calibration	8.0MHz	5V	25°C	±1.5%
	8.0MHz	4.5 - 5.5V(1)	–40°C / +125°C	±10%
Watchdog oscillator	128KHz	2.7 - 5.5V(1)	–40°C / +125°C	±40%

Note: 1. Voltage range for Atmel® ATmega164P-B/324P-B/644P-B.

28.4.1 External Clock Drive Waveforms

Figure 28-2. External Clock Drive Waveforms



28.4.2 External Clock Drive

Table 28-4. External Clock Drive

Parameter	Symbol	V_{CC} = 2.7 to 5.5V		V_{CC} = 4.5 to 5.5V		Unit
		Min.	Max.	Min.	Max.	
Oscillator frequency	$1/t_{CLCL}$	0	8	0	16	MHz
Clock period	t_{CLCL}	125		62.5		ns
High time	t_{CHCX}	40		20		ns
Low time	t_{CLCX}	40		20		ns
Rise time	t_{CLCH}		1.6		0.5	μs
Fall time	t_{CHCL}		1.6		0.5	μs
Change in period from one clock cycle to the next	Δt_{CLCL}		2		2	%

28.5 System and Reset Characteristics

Table 28-5. Reset, Brown-out and Internal Voltage Reference Characteristics

Parameter	Condition	Symbol	Min	Typ.	Max	Unit
Power-on reset threshold voltage (rising)		V_{POT}	1.1	1.4	1.6	V
Power-on reset threshold voltage (falling)(1)			0.6	1.3	1.6	V
$\overline{\text{RESET}}$ pin threshold voltage		V_{RST}	$0.2V_{CC}$		$0.9V_{CC}$	V
Minimum pulse width on $\overline{\text{RESET}}$ pin		t_{RST}	2.5			µs
Brown-out detector hysteresis		V_{HYST}		80		mV
Min pulse width on brown-out reset		t_{BOD}		2		µs
Bandgap reference voltage	V_{CC}= 2.7V, T_A = 25°C	V_{BG}	0.98	1.1	1.22	V
Bandgap reference start-up time	V_{CC}= 2.7V, T_A = 25°C	t_{BG}		40	70	µs

Note: 1. The power-on reset will not work unless the supply voltage has been below V_{POT} (falling).

Table 28-6. BODLEVEL Fuse Coding(1)

BODLEVEL 2:0 Fuses	Min V_{BOT}	Typical V_{BOT}	Max V_{BOT}	Unit
111	BOD disabled			
110	1.7	1.8	2.0	V
101	2.5	2.7	2.9	
100	4.0	4.3	4.55	
011	Reserved			
010				
001				
000				

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 = 101 and BODLEVEL = 110.

28.6 External Interrupts Characteristics

Table 28-7. Asynchronous External Interrupt Characteristics

Parameter	Condition	Symbol	Min	Typ.	Max	Unit
Minimum pulse width for asynchronous external interrupt		t_{INT}		50		ns

28.7 SPI Timing Characteristics

See Figure 28-3 and Figure 28-4 on page 293 for details.

Table 28-8. SPI Timing Parameters

	Description	Mode	Min	Typ.	Max	
1	SCK period	Master		See Table 18-5 on page 145		ns
2	SCK high/low	Master		50% duty cycle		
3	Rise/fall time	Master		3.6		
4	Setup	Master		10		
5	Hold	Master		10		
6	Out to SCK	Master		$0.5 \times t_{sck}$		
7	SCK to out	Master		10		
8	SCK to out high	Master		10		
9	$\overline{\text{SS}}$ low to out	Slave		15		
10	SCK period	Slave	$4 \times t_{ck}$			
11	SCK high/low(1)	Slave	$2 \times t_{ck}$			
12	Rise/fall time	Slave			1600	
13	Setup	Slave	10			
14	Hold	Slave	t_{ck}			
15	SCK to out	Slave		15		
16	SCK to $\overline{\text{SS}}$ high	Slave	20			
17	$\overline{\text{SS}}$ high to tri-state	Slave		10		
18	$\overline{\text{SS}}$ low to SCK	Slave	20			

Note: 1. In SPI programming mode the minimum SCK high/low period is:
– 2 t_{CLCL} for $f_{CK} < 12MHz$
–3 t_{CLCL} for $f_{CK} > 12MHz$

Figure 28-3. SPI Interface Timing Requirements (Master Mode)

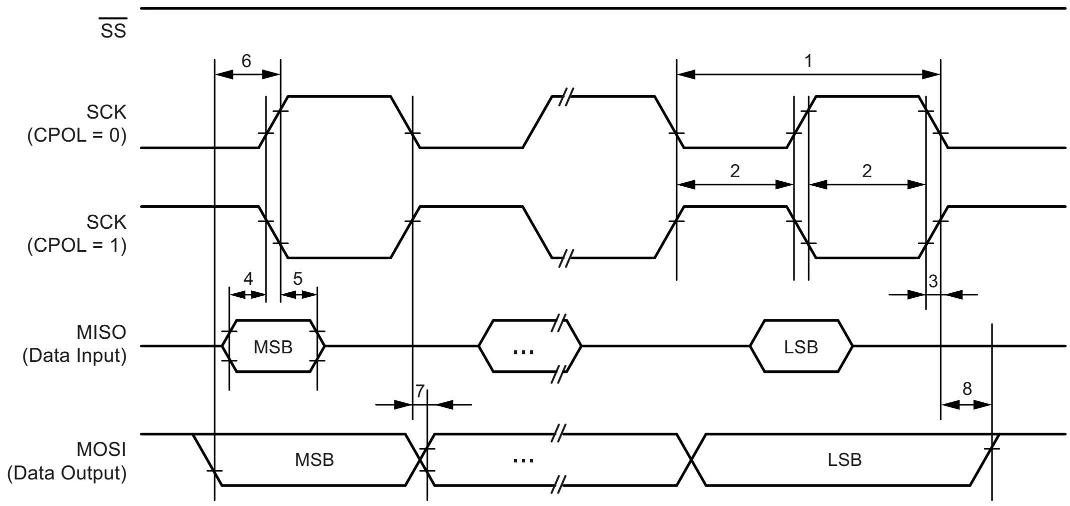
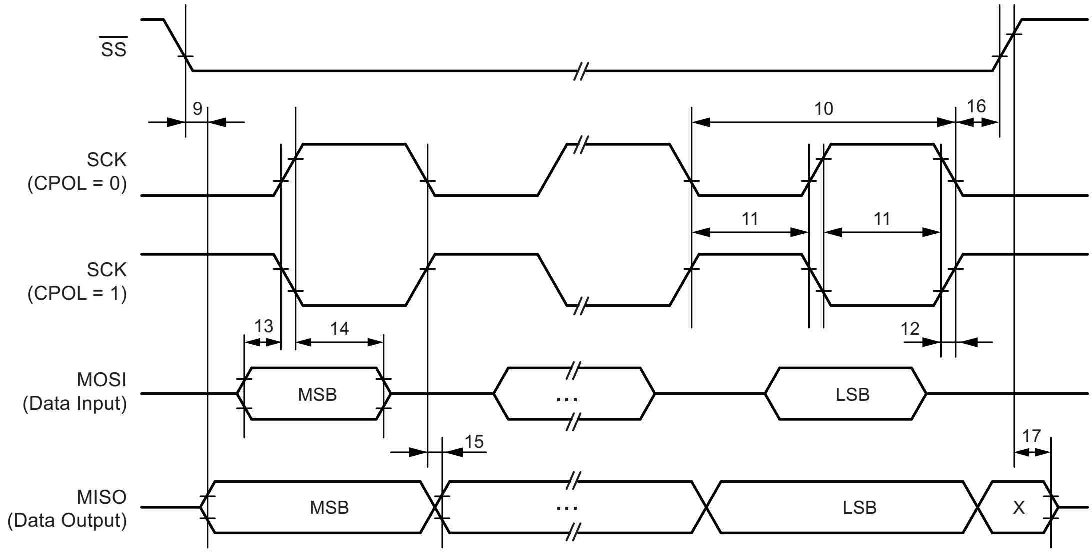


Figure 28-4. SPI Interface Timing Requirements (Slave Mode)



28.8 2-wire Serial Interface Characteristics

Table 28-9 describes the requirements for devices connected to the 2-wire serial bus. The Atmel® ATmega164P-B/324P-B/644P-B 2-wire serial interface meets or exceeds these requirements under the noted conditions.

Timing symbols refer to Figure 28-5 on page 294.

Table 28-9. 2-wire Serial Bus Requirements

Parameter	Condition	Symbol	Min	Max	Unit
Input low-voltage		V_{IL}	–0.5	$0.3\ V_{CC}$	V
Input high-voltage		V_{IH}	$0.7\ V_{CC}$	$V_{CC} + 0.5$	V
Hysteresis of schmitt trigger inputs		$V_{hys}^{(1)}$	$0.05\ V_{CC}^{(2)}$	–	V
Output low-voltage	3mA sink current	$V_{OL}^{(1)}$	0	0.4	V
Rise time for both SDA and SCL		$t_r^{(1)}$	$20 + 0.1C_b^{(2)(3)}$	300	ns
Output fall time from V_{IHmin} to V_{ILmax}	$10pF < C_b < 400pF^{(3)}$	$t_{of}^{(1)}$	$20 + 0.1C_b^{(2)(3)}$	250	ns
Spikes suppressed by input filter		$t_{SP}^{(1)}$	0	$50^{(2)}$	ns
Input current each I/O pin	$0.1V_{CC} < V_i < 0.9V_{CC}$	I_i	–10	10	µA
Capacitance for each I/O pin		$C_i^{(1)}$	–	10	pF
SCL clock frequency	$f_{CK}^{(4)} > \max(16f_{SCL}, 250kHz)^{(5)}$	f_{SCL}	0	400	kHz

Notes:
1. In ATmega164P-B/324P-B/644P-B, this parameter is characterized and not 100% tested.
2. Required only for $f_{SCL} > 100kHz$.
3. Cb = capacitance of one bus line in pF.
4. f_{CK} = CPU clock frequency
5. This requirement applies to all ATmega164P-B/324P-B/644P-B two-wire serial interface operation. Other devices connected to the two-wire serial bus need only obey the general f_{SCL} requirement.
6. The actual low period generated by the ATmega164P-B/324P-B/644P-B two-wire serial interface is $(1/f_{SCL} - 2/f_{CK})$, thus f_{CK} must be greater than 6MHz for the low time requirement to be strictly met at f_{SCL} = 100kHz.
7. The actual low period generated by the ATmega164P-B/324P-B/644P-B two-wire serial interface is $(1/f_{SCL} - 2/f_{CK})$, thus the low time requirement will not be strictly met for f_{SCL} > 308kHz when f_{CK} = 8MHz. Still, ATmega164P-B/324P-B/644P-B devices connected to the bus may communicate at full speed (400kHz) with other ATmega164P-B/324P-B/644P-B devices, as well as any other device with a proper t_{LOW} acceptance margin.

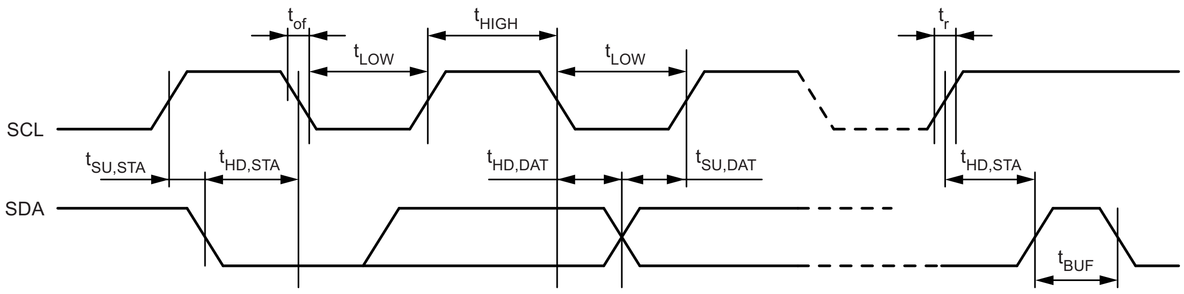
Table 28-9. 2-wire Serial Bus Requirements (Continued)

Parameter	Condition	Symbol	Min	Max	Unit
Value of pull-up resistor	$f_{SCL} \leq 100kHz$	Rp	$\frac{V_{CC} - 0{,}4V}{3mA}$	$\frac{1000ns}{C_b}$	Ω
	$f_{SCL} > 100kHz$	Rp	$\frac{V_{CC} - 0{,}4V}{3mA}$	$\frac{300ns}{C_b}$	Ω
Hold time (repeated) START condition	$f_{SCL} \leq 100kHz$	$t_{HD;STA}$	4.0	–	µs
	$f_{SCL} > 100kHz$		0.6	–	µs
Low period of the SCL clock	$f_{SCL} \leq 100kHz^{(6)}$	t_{LOW}	4.7	–	µs
	$f_{SCL} > 100kHz^{(7)}$		1.3	–	µs
High period of the SCL clock	$f_{SCL} \leq 100kHz$	t_{HIGH}	4.0	–	µs
	$f_{SCL} > 100kHz$		0.6	–	µs
Set-up time for a repeated START condition	$f_{SCL} \leq 100kHz$	$t_{SU;STA}$	4.7	–	µs
	$f_{SCL} > 100kHz$		0.6	–	µs
Data hold time	$f_{SCL} \leq 100kHz$	$t_{HD;DAT}$	0	3.45	µs
	$f_{SCL} > 100kHz$		0	0.9	µs
Data setup time	$f_{SCL} \leq 100kHz$	$t_{SU;DAT}$	250	–	ns
	$f_{SCL} > 100kHz$		100	–	ns
Setup time for STOP condition	$f_{SCL} \leq 100kHz$	$t_{SU;STO}$	4.0	–	µs
	$f_{SCL} > 100kHz$		0.6	–	µs
Bus free time between a STOP and START condition	$f_{SCL} \leq 100kHz$	t_{BUF}	4.7	–	µs
	$f_{SCL} > 100kHz$		1.3	–	µs

Notes:

1. In ATmega164P-B/324P-B/644P-B, this parameter is characterized and not 100% tested.
2. Required only for $f_{SCL} > 100kHz$.
3. Cb = capacitance of one bus line in pF.
4. f_{CK} = CPU clock frequency
5. This requirement applies to all ATmega164P-B/324P-B/644P-B two-wire serial interface operation. Other devices connected to the two-wire serial bus need only obey the general f_{SCL} requirement.
6. The actual low period generated by the ATmega164P-B/324P-B/644P-B two-wire serial interface is $(1/f_{SCL} - 2/f_{CK})$, thus f_{CK} must be greater than 6MHz for the low time requirement to be strictly met at f_{SCL} = 100kHz.
7. The actual low period generated by the ATmega164P-B/324P-B/644P-B two-wire serial interface is $(1/f_{SCL} - 2/f_{CK})$, thus the low time requirement will not be strictly met for f_{SCL} > 308kHz when f_{CK} = 8MHz. Still, ATmega164P-B/324P-B/644P-B devices connected to the bus may communicate at full speed (400kHz) with other ATmega164P-B/324P-B/644P-B devices, as well as any other device with a proper t_{LOW} acceptance margin.

Figure 28-5. 2-wire Serial Bus Timing



28.9 ADC Characteristics

Table 28-10. ADC Characteristics, Single Ended Channel

Parameter	Condition	Symbol	Min	Typ.	Max	Unit
Resolution	Single ended conversion			10		Bit
Absolute accuracy	V_{CC} = 4V, V_{REF} = 4V, ADC clock = 200kHz	TUE		2.5	4	LSB
	V_{CC} = 4V, V_{REF} = 4V, ADC clock = 200kHz, Noise reduction mode on.	TUE		2.5	4	LSB
Integral non linearity	V_{CC} = 4V, V_{REF} = 4V, ADC clock = 200kHz	INL		0.5	1.5	LSB
Differential non linearity	V_{CC} = 4V, V_{REF} = 4V, ADC clock = 200kHz	DNL		0.3	0.7	LSB
Gain error	V_{CC} = 4V, V_{REF} = 4V, ADC clock = 200kHz		–4	–2	4	LSB
Offset error	V_{CC} = 4V, V_{REF} = 4V, ADC clock = 200kHz		4	2	4	LSB
Conversion time	Free running conversion		65		260	µs
Clock frequency	Single ended conversion		50		200	kHz
Analog supply voltage		AVCC	V_{CC} – 0.3		V_{CC} + 0.3	V
Reference voltage		V_{REF}	1.00		AVCC	V
Input voltage		V_{IN}	GND		V_{REF}	V
Internal voltage reference	1.1V	V_{INT1}	0.96	1.1	1.2	V
Internal voltage reference	2.56V, V_{CC} > 2.7V	V_{INT2}	2.33	2.56	2.79	V
Reference input resistance		R_{REF}		30 ±40%		kΩ
Analog input resistance		R_{AIN}		100		MΩ

Table 28-11. ADC Characteristics, Differential Channels

Parameter	Condition	Symbol	Min	Typ.	Max	Unit
Resolution	Gain = 1x			8		Bit
	Gain = 10x			8		
	Gain = 200x			7		
Absolute accuracy (including INL, DNL quantization error and offset error)	Gain = 1x V_{CC} = 5V, V_{REF} = 4V ADC clock = 200kHz	TUE		4.6	7	LSB
	Gain = 10x V_{CC} = 5V, V_{REF} = 4V ADC clock = 200kHz			4.8	8	
	Gain = 200x V_{CC} = 5V, V_{REF} = 4V ADC clock = 200kHz			1.0	4	
Integral non-linearity (INL)	Gain = 1x V_{CC} = 5V, V_{REF} = 4V ADC clock = 200kHz	INL		0.3	1.5	
	Gain = 10x V_{CC} = 5V, V_{REF} = 4V ADC clock = 200kHz			0.3	1.5	
	Gain = 200x V_{CC} = 5V, V_{REF} = 4V ADC clock = 200kHz			0.3	1.5	
Differential non-linearity (DNL)	Gain = 1x V_{CC} = 5V, V_{REF} = 4V ADC clock = 200kHz	DNL		0.2	1.0	
	Gain = 10x V_{CC} = 5V, V_{REF} = 4V ADC clock = 200kHz			0.2	1.0	
	Gain = 200x V_{CC} = 5V, V_{REF} = 4V ADC clock = 200kHz			0.3	1.0	
Gain error	Gain = 1x V_{CC} = 5V, V_{REF} = 4V ADC clock = 200kHz		–12	–9	–4	LSB
	Gain = 10x V_{CC} = 5V, V_{REF} = 4V ADC clock = 200kHz		–12	–9	–4	
	Gain = 200x V_{CC} = 5V, V_{REF} = 4V ADC clock = 200kHz		–3	–1	3	
Offset error	Gain = 1x V_{CC} = 5V, V_{REF} = 4V ADC clock = 200kHz		–4	0.3	4	
	Gain = 10x V_{CC} = 5V, V_{REF} = 4V ADC clock = 200kHz		–4	0.2	4	
	Gain = 200x V_{CC} = 5V, V_{REF} = 4V ADC clock = 200kHz		–4	-0.5	3	
Reference voltage		V_{REF}	2.56		AVCC – 0.5	V

29. Typical Characteristics

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 registers set and thus, the corresponding I/O modules are turned off. Also the analog comparator is disabled during these measurements. The power consumption in power-down mode is independent of clock selection.

The current consumption is a function of several factors such as: operating voltage, operating frequency, loading of I/O pins, switching rate of I/O pins, code executed and ambient temperature. The dominating factors are operating voltage and frequency.

The current drawn from capacitive loaded pins may be estimated (for one pin) as $C_L \times V_{CC} \times f$ where C_L = load capacitance, V_{CC} = operating voltage and f = average switching frequency of I/O pin.

The parts are characterized at frequencies higher than test limits. Parts are not guaranteed to function properly at frequencies higher than the ordering code indicates.

The difference between current consumption in power-down mode with watchdog timer enabled and power-down mode with watchdog timer disabled represents the differential current drawn by the watchdog timer.

29.1 ATmega164P-B Typical Characteristics

29.1.1 Active Supply Current

Figure 29-1. ATmega164P-B: Active Supply Current versus Low Frequency (0.1 - 1.0MHz)

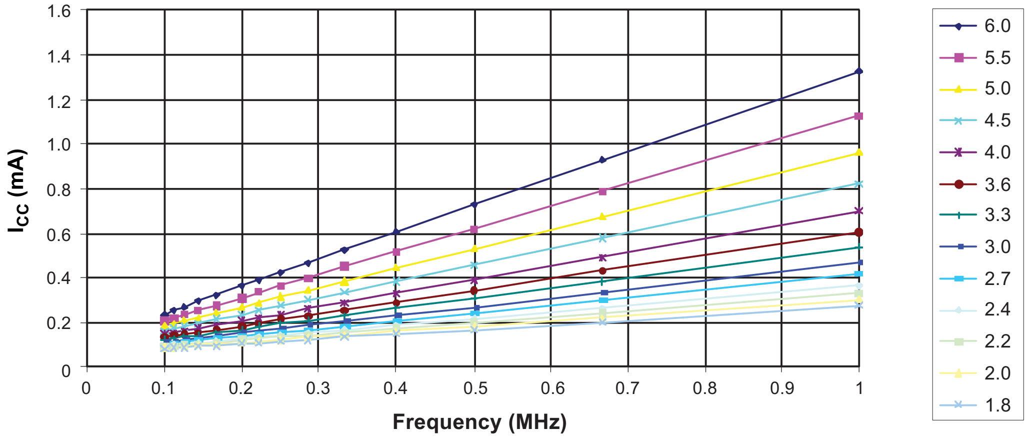
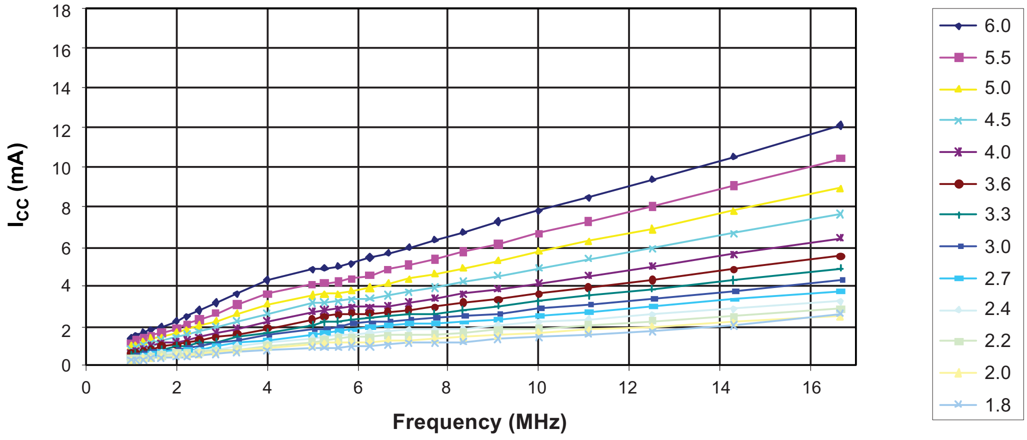


Figure 29-2. ATmega164P-B: Active Supply Current versus Frequency (1 - 16MHz)



29.1.2 Idle Supply Current

Figure 29-3. ATmega164P-B: Idle Supply Current versus Low Frequency (0.1 - 1.0MHz)

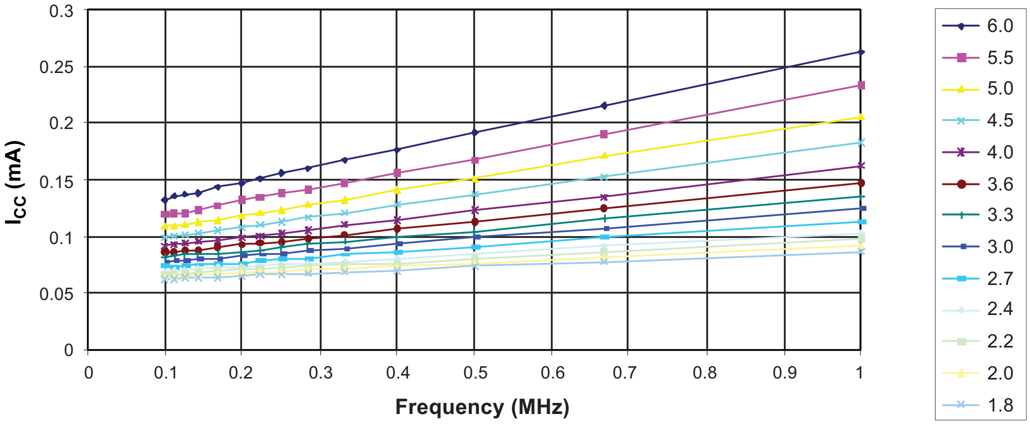
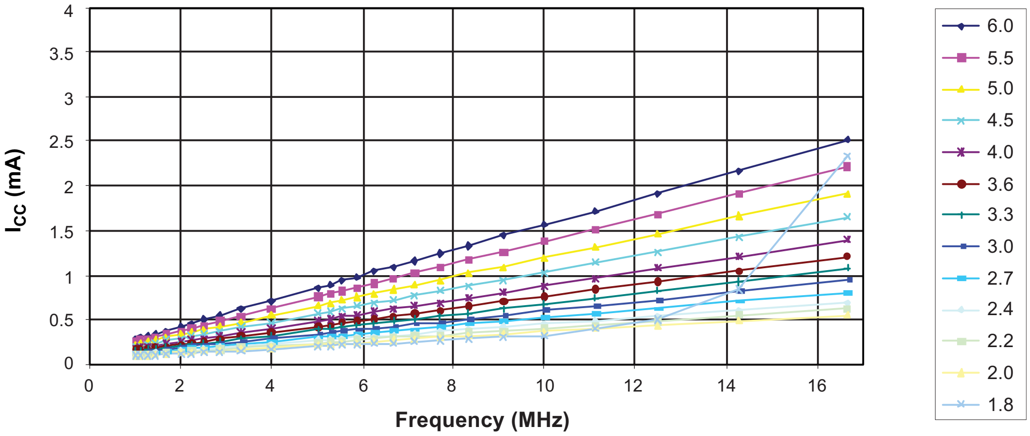


Figure 29-4. ATmega164P-B: Idle Supply Current versus Frequency (1 - 16MHz)



29.1.3 Power-down Supply Current

Figure 29-5. ATmega164P-B: Power-down Supply Current versus V_{CC} (Watchdog Timer Disabled)

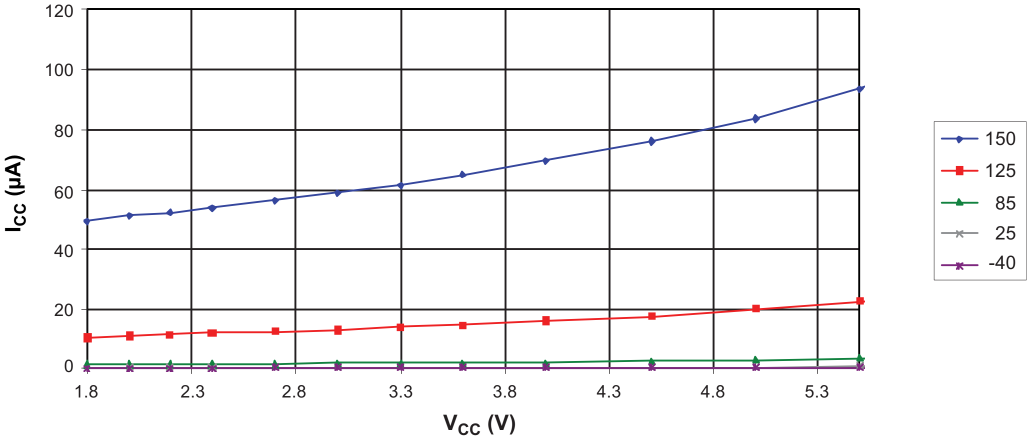
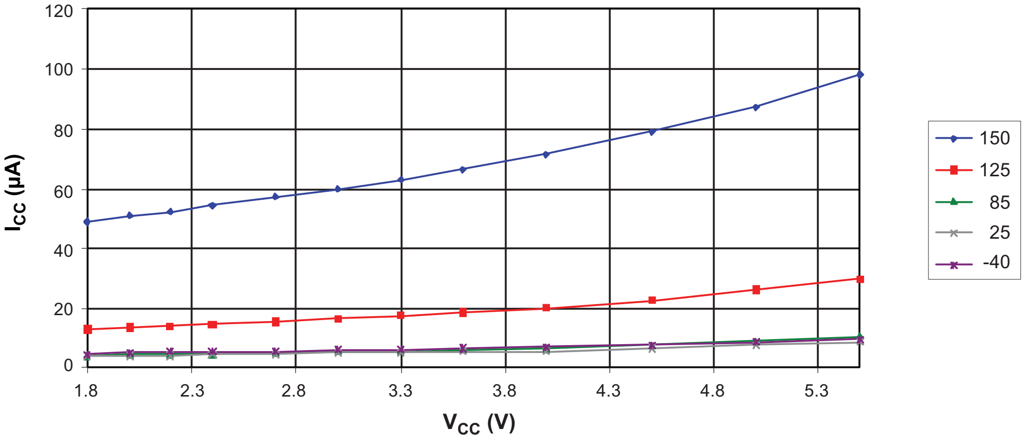


Figure 29-6. ATmega164P-B: Power-down Supply Current versus V_{CC} (Watchdog Timer Enabled)



29.1.4 Pin Pull-up

Figure 29-7. ATmega164P-B: I/O Pin Pull-up Resistor Current versus Input Voltage (V_{CC} = 5V)

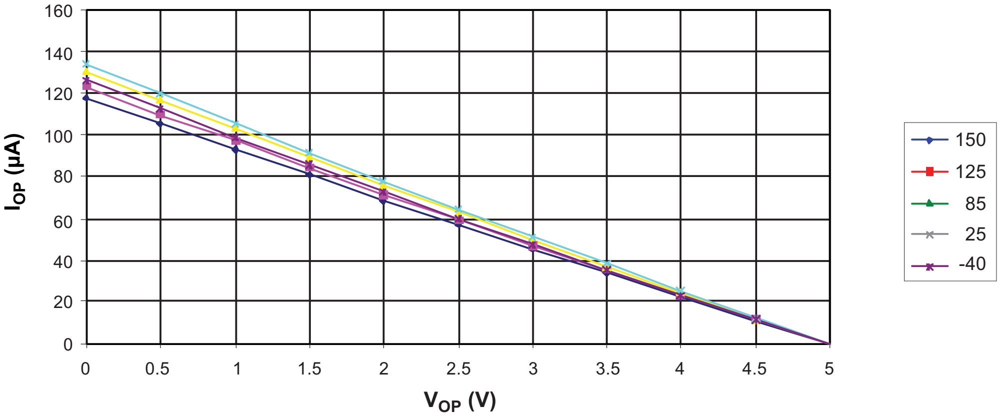
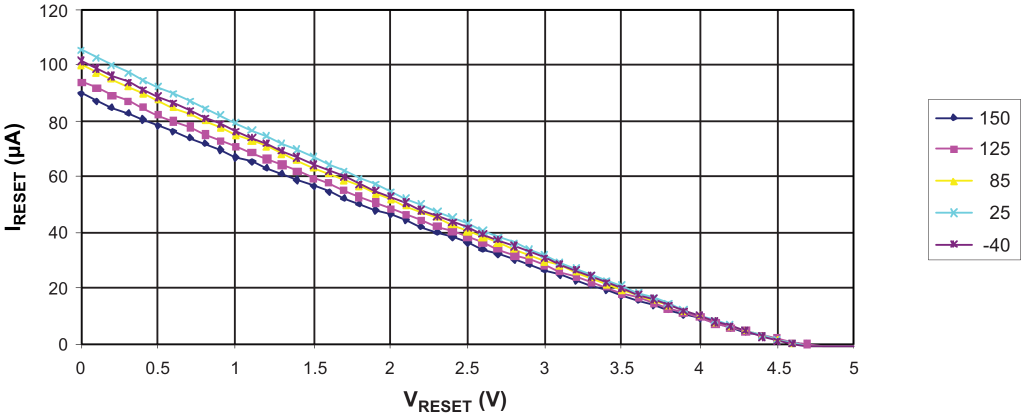


Figure 29-8. ATmega164P-B: Reset Pull-up Resistor Current versus Reset Pin Voltage (V_{CC} =5V)



29.1.5 Pin Driver Strength

Figure 29-9. ATmega164P-B: I/O Pin Output Voltage versus Sink Current (V_{CC} = 3V)

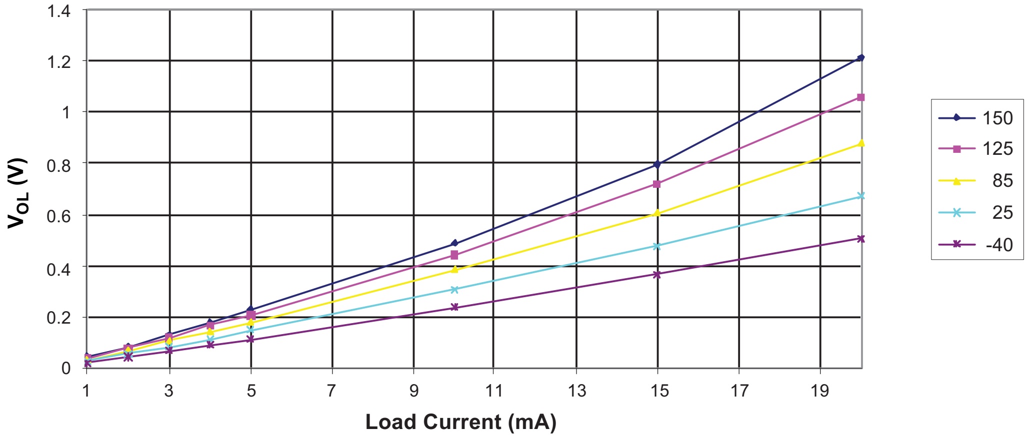


Figure 29-10. ATmega164P-B: I/O Pin Output Voltage versus Sink Current (V_{CC} = 5V)

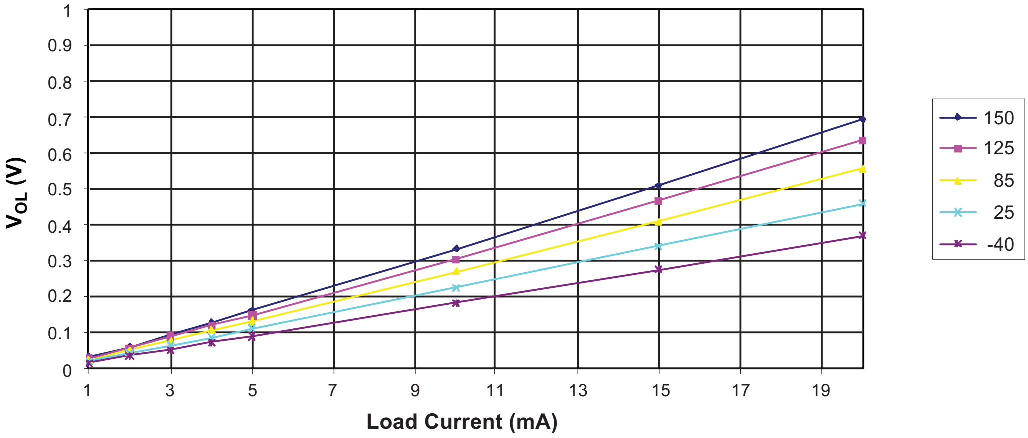


Figure 29-11. ATmega164P-B: I/O Pin Output Voltage versus Source Current (V_{CC} = 3V)

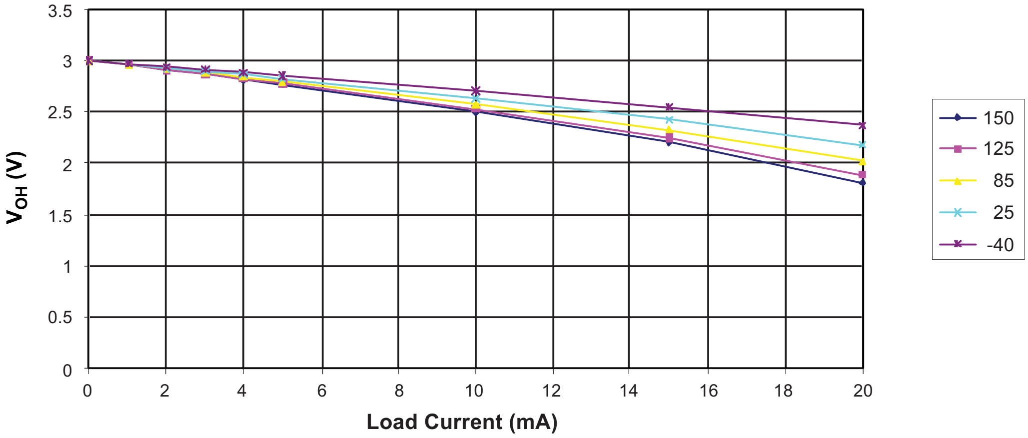
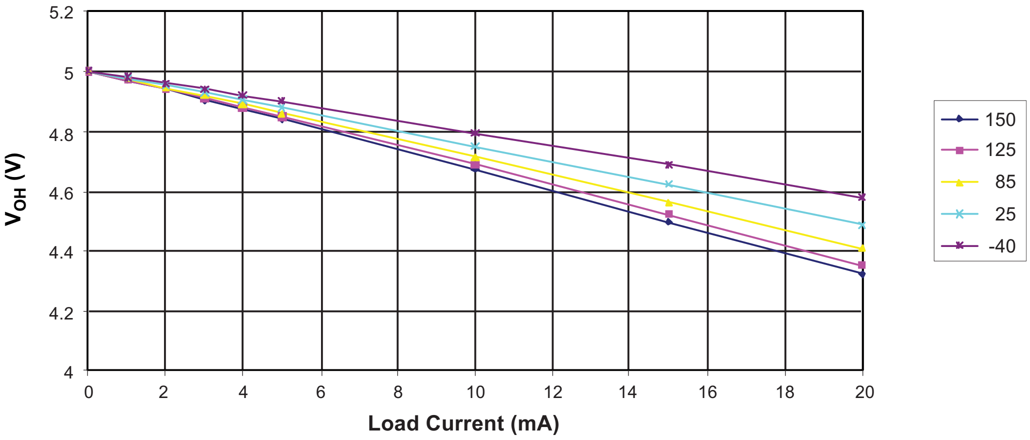


Figure 29-12. ATmega164P-B: I/O Pin Output Voltage versus Source Current (V_{CC} = 5V)



29.1.6 Pin Threshold

Figure 29-13. ATmega164P-B: I/O Pin Input Threshold versus V_{CC} (V_{IH}, I/O Pin Read as ‘1’)

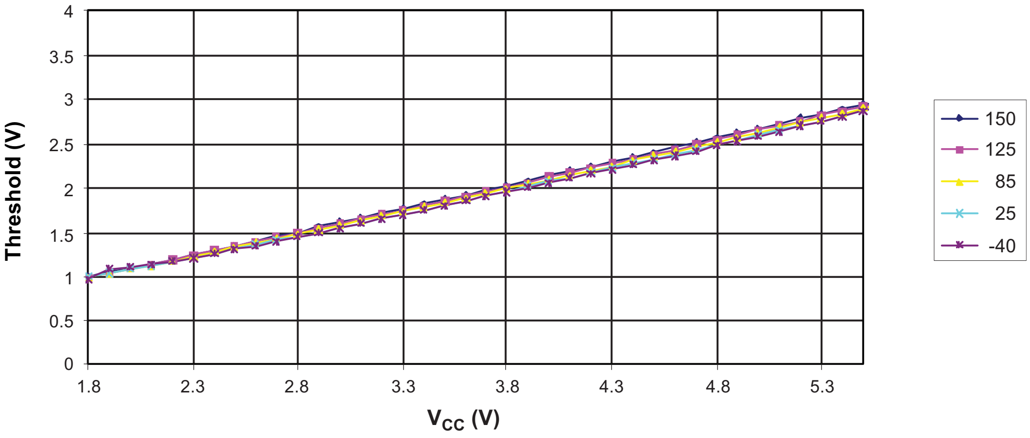


Figure 29-14. ATmega164P-B: I/O Pin Input Threshold versus V_{CC} (V_{IL}, I/O Pin Read as ‘0’)

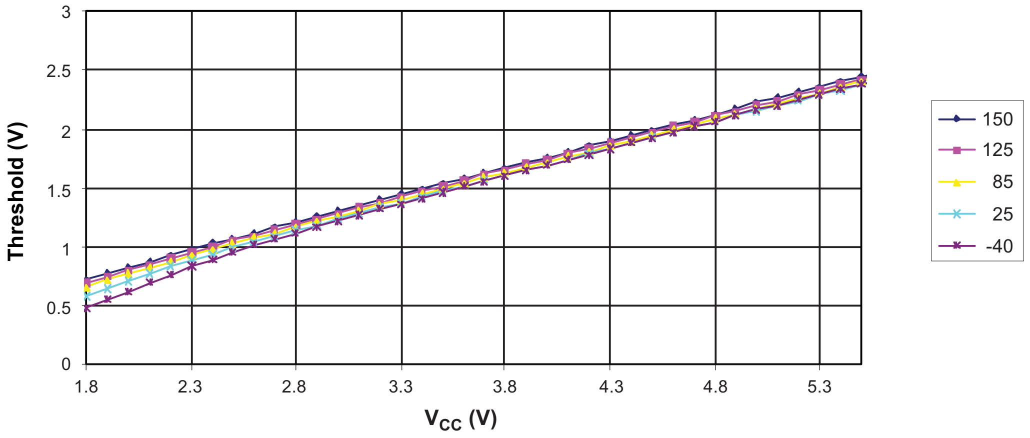


Figure 29-15. ATmega164P-B: Reset Pin Input Threshold versus V_{CC} (V_{IH}, I/O Pin Read as '1')

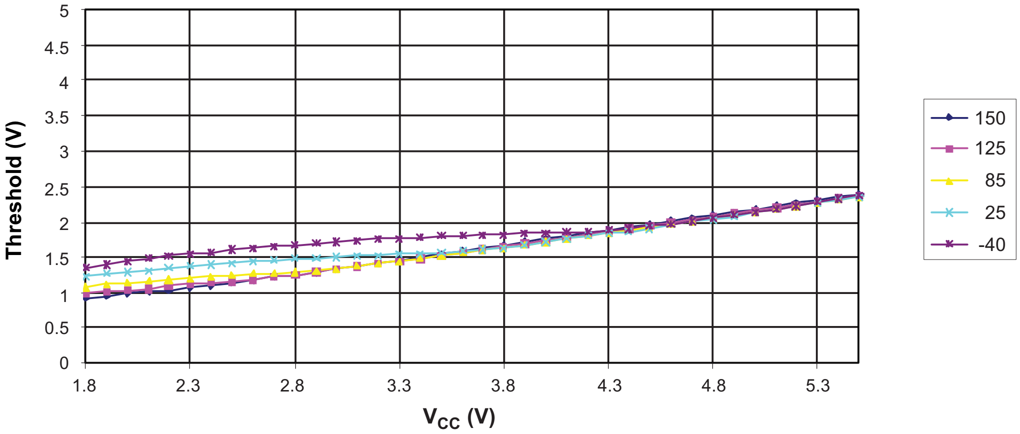
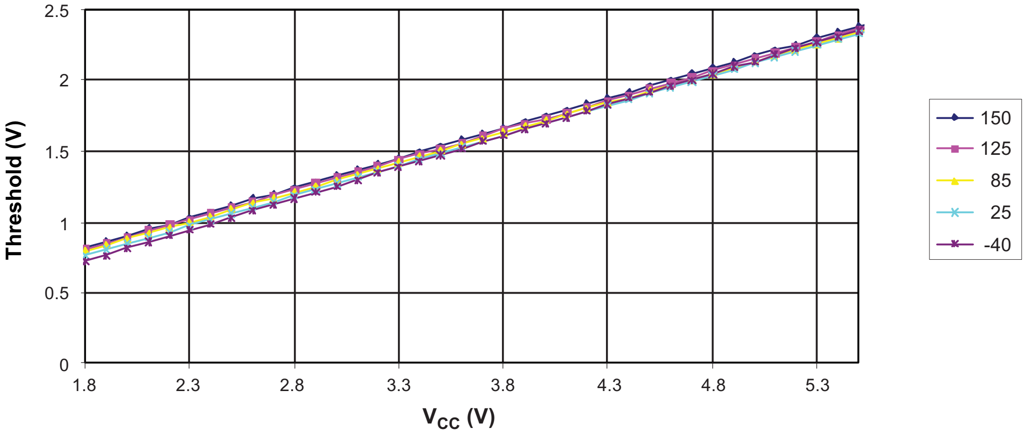


Figure 29-16. ATmega164P-B: Reset Pin Input Threshold versus V_{CC} (V_{IL}, I/O Pin Read as '0')



29.1.7 BOD Threshold

Figure 29-17. ATmega164P-B: BOD Threshold versus Temperature (V_{BOT} = 4.3V)

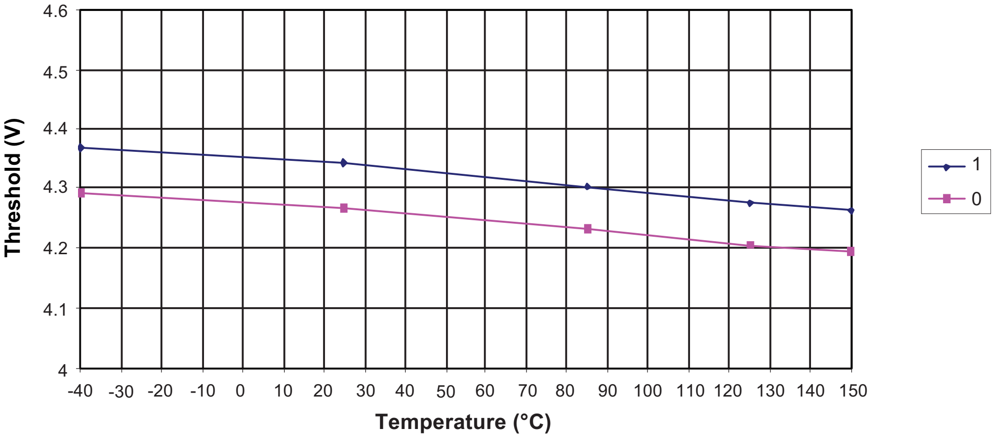


Figure 29-18. ATmega164P-B: BOD Threshold versus Temperature (V_{BOT} = 2.7V)

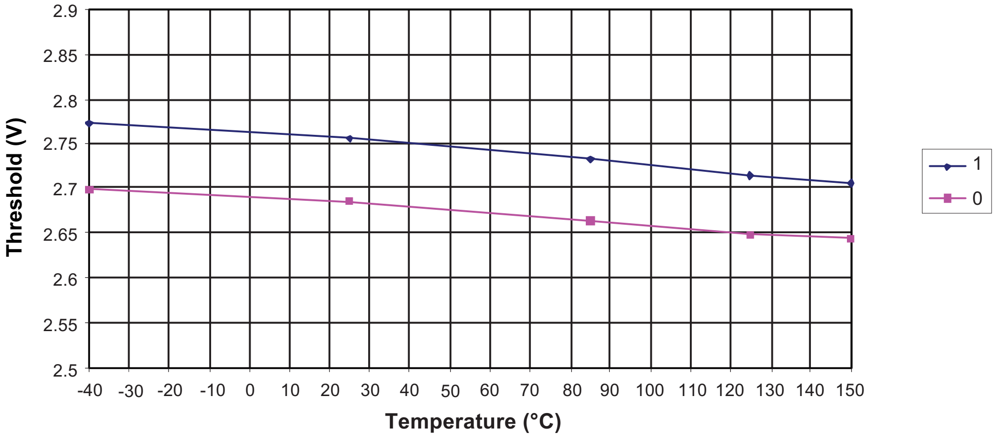


Figure 29-19. ATmega164P-B: Calibrated Bandgap Voltage versus V_{CC}

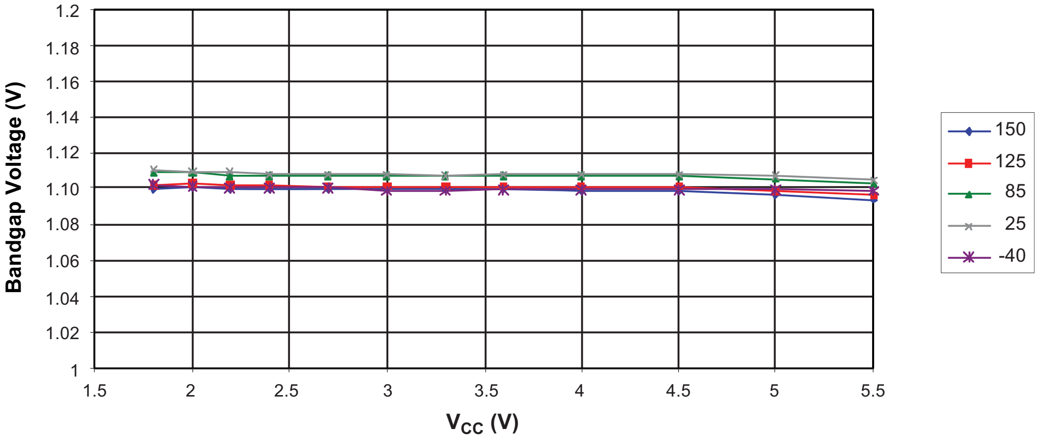
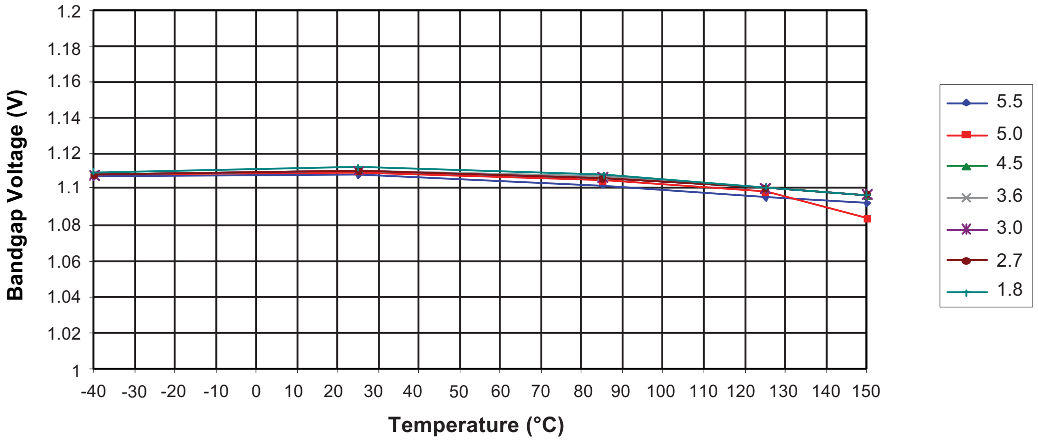


Figure 29-20. ATmega164P-B: Bandgap Voltage versus Temperature



29.1.8 Internal Oscillator Speed

Figure 29-21. ATmega164P-B: Watchdog Oscillator Frequency versus Temperature

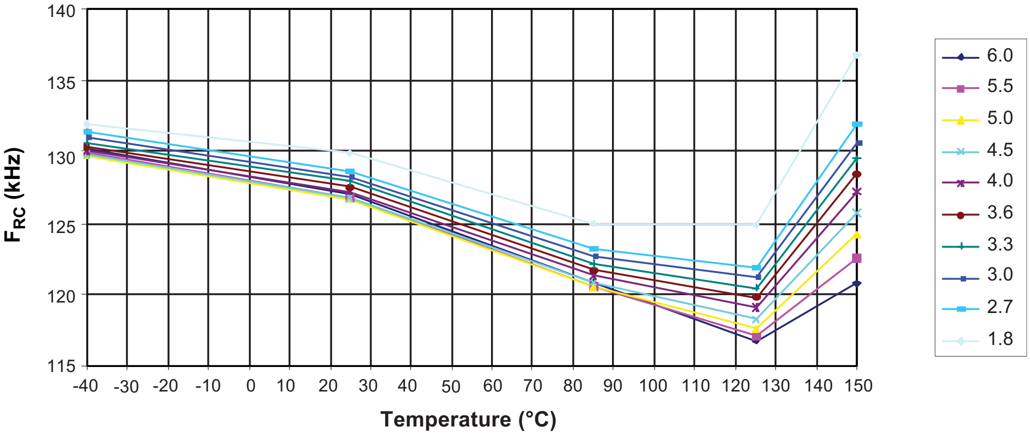


Figure 29-22. ATmega164P-B: Watchdog Oscillator Frequency versus V_{CC}

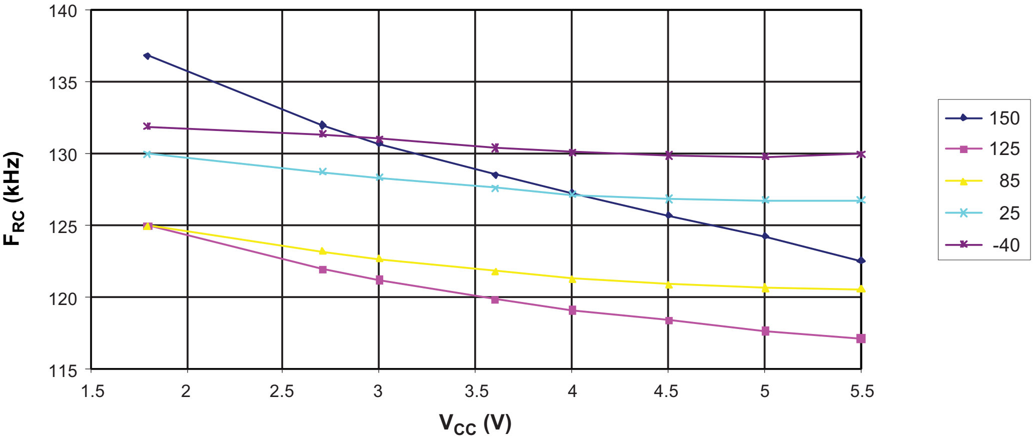


Figure 29-23. ATmega164P-B: Calibrated 8MHz RC Oscillator versus V_{CC}

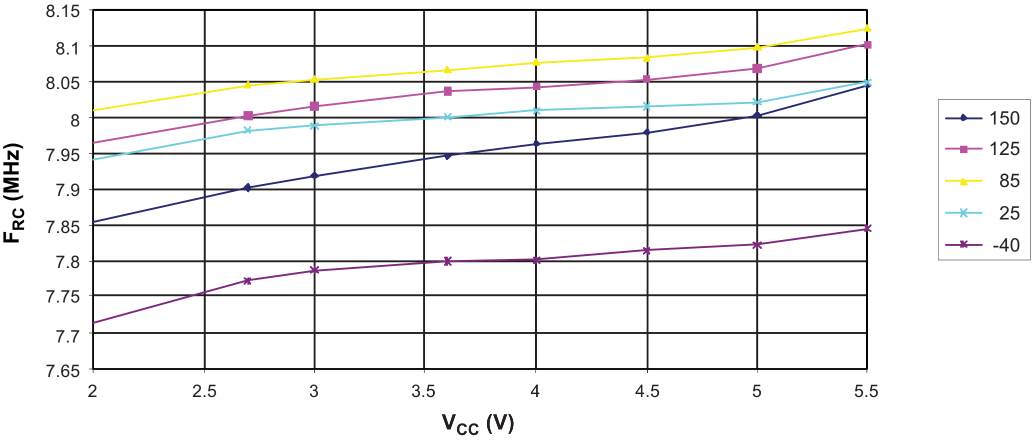


Figure 29-24. ATmega164P-B: Calibrated 8MHz RC Oscillator versus Temperature

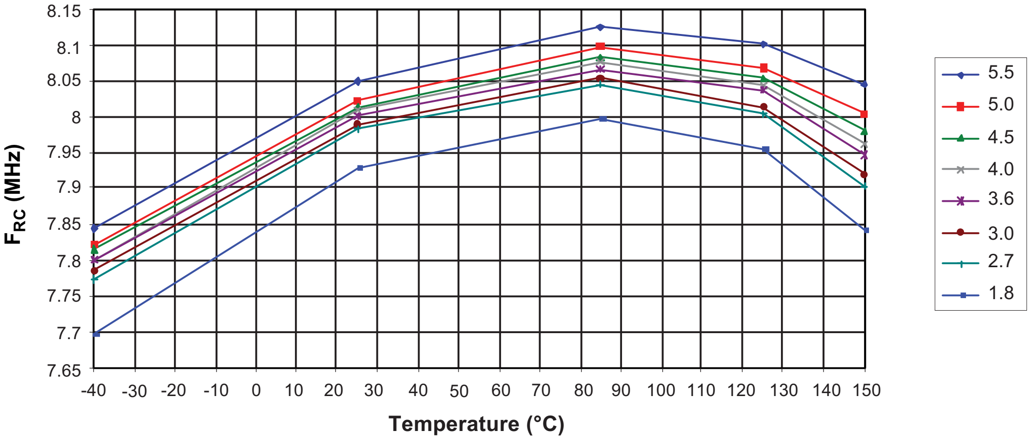
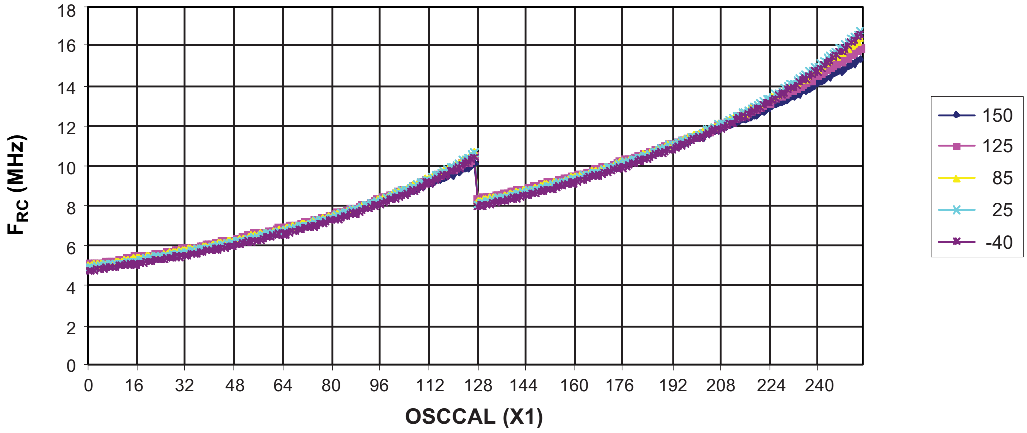


Figure 29-25. ATmega164P-B: Calibrated 8MHz RC Oscillator versus OSCCAL Value



29.2 ATmega324P-B Typical Characteristics

29.2.1 Active Supply Current

Figure 29-26. ATmega324P-B: Active Supply Current versus Low Frequency (0.1 - 1.0MHz)

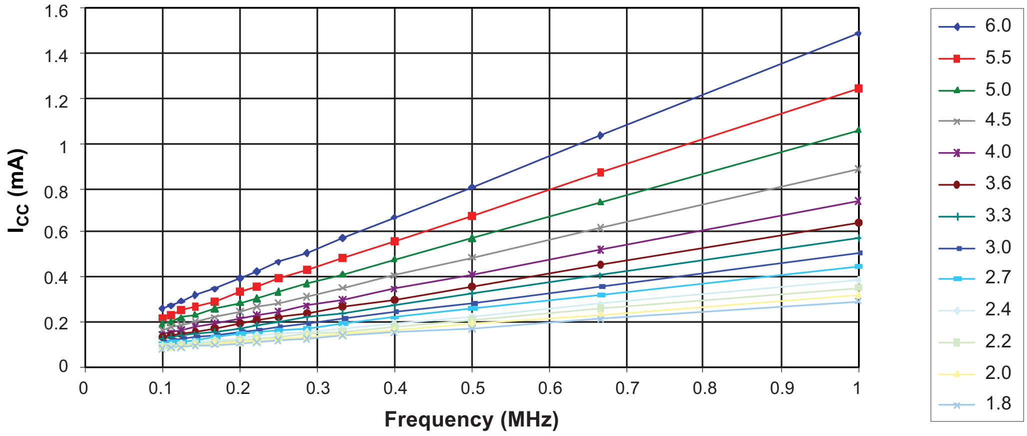
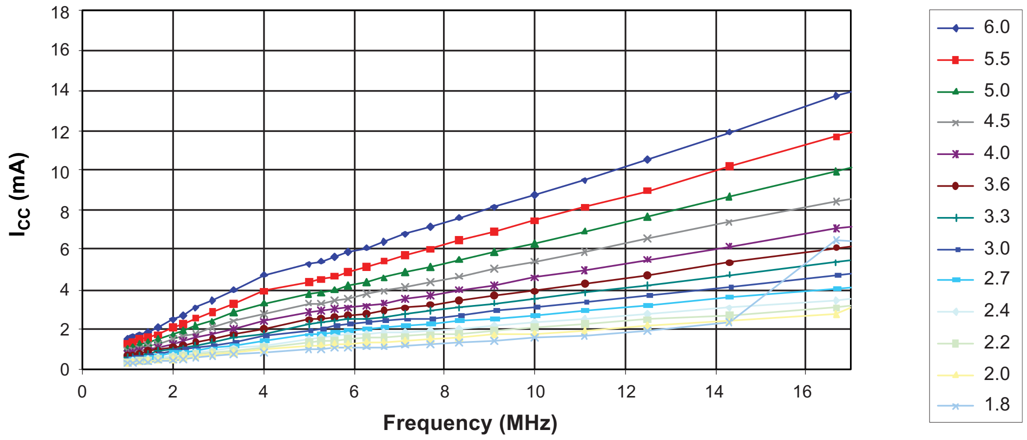


Figure 29-27. ATmega324P-B: Active Supply Current versus Frequency (1 - 16MHz)



29.2.2 Idle Supply Current

Figure 29-28. ATmega324P-B: Idle Supply Current versus Low Frequency (0.1 - 1.0MHz)

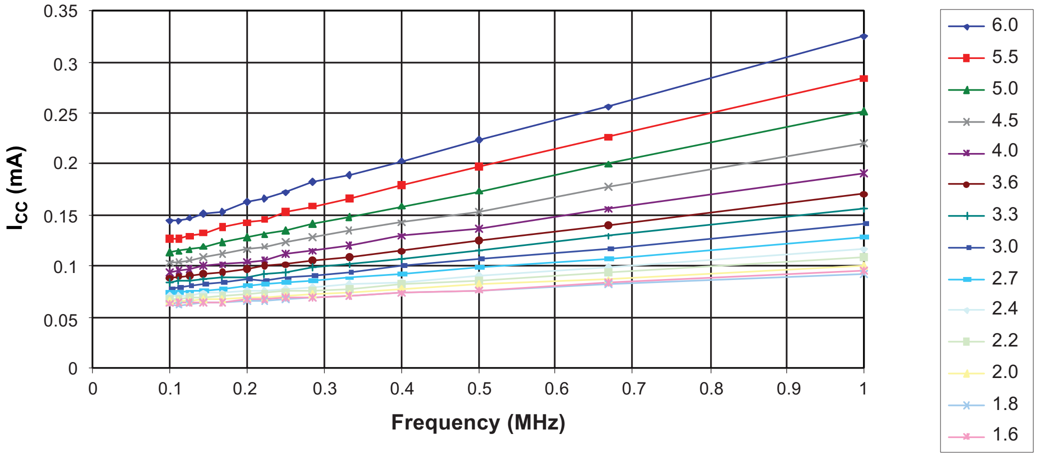
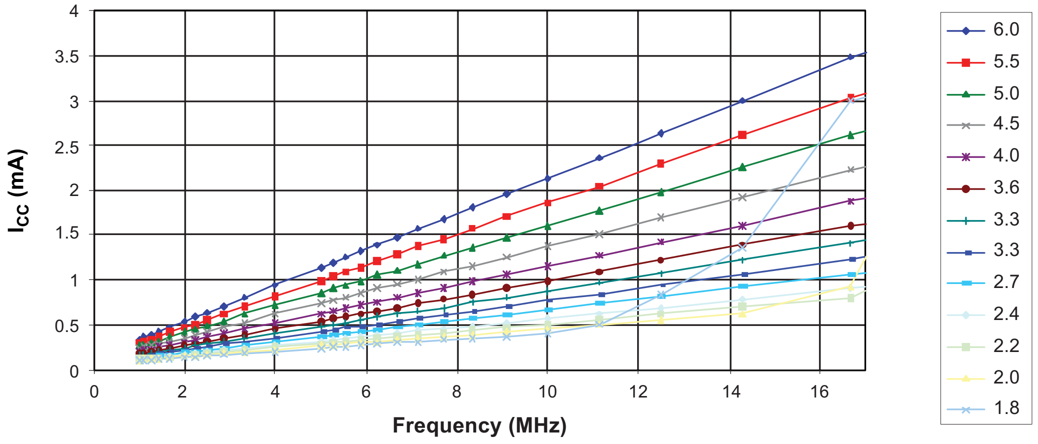


Figure 29-29. ATmega324P-B: Idle Supply Current versus Frequency (1 - 16MHz)



29.2.3 Power-down Supply Current

Figure 29-30. ATmega324P-B: Power-down Supply Current versus V_{CC} (Watchdog Timer Disabled)

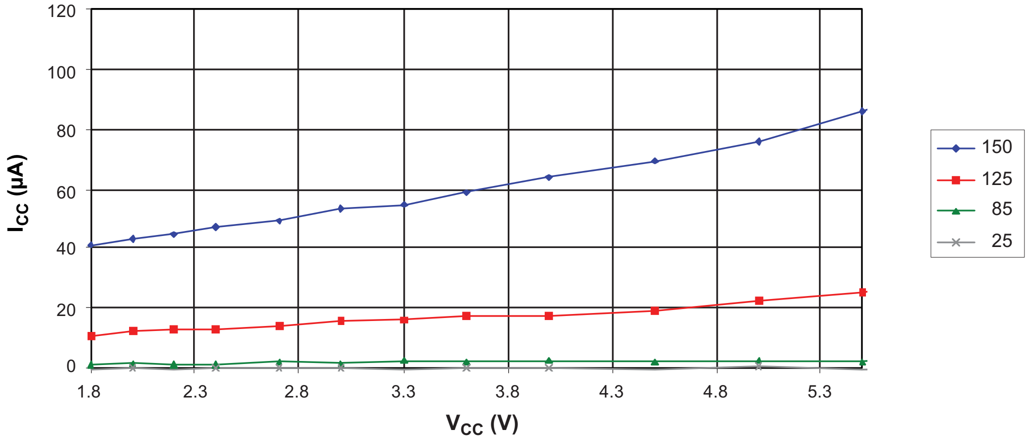
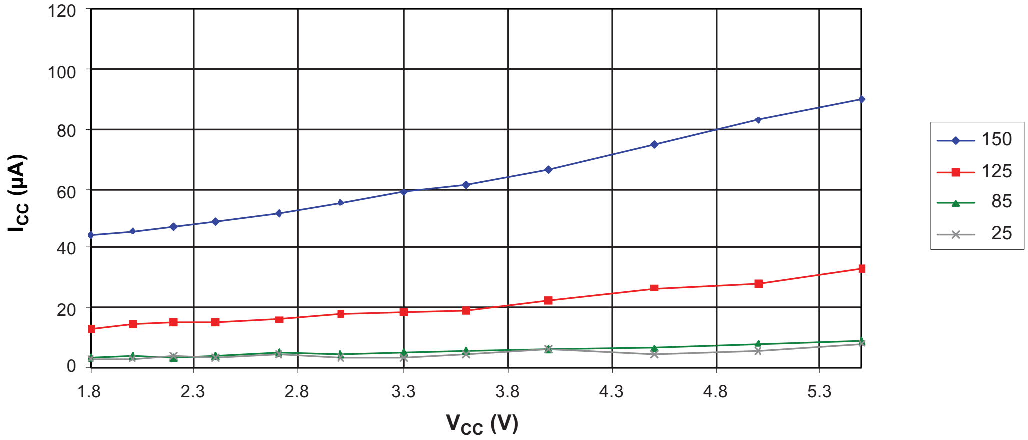


Figure 29-31. ATmega324P-B: Power-down Supply Current versus V_{CC} (Watchdog Timer Enabled)



29.2.4 Pin Pull-up

Figure 29-32. ATmega324P-B: I/O Pin Pull-up Resistor Current versus Input Voltage (V_{CC} = 5V)

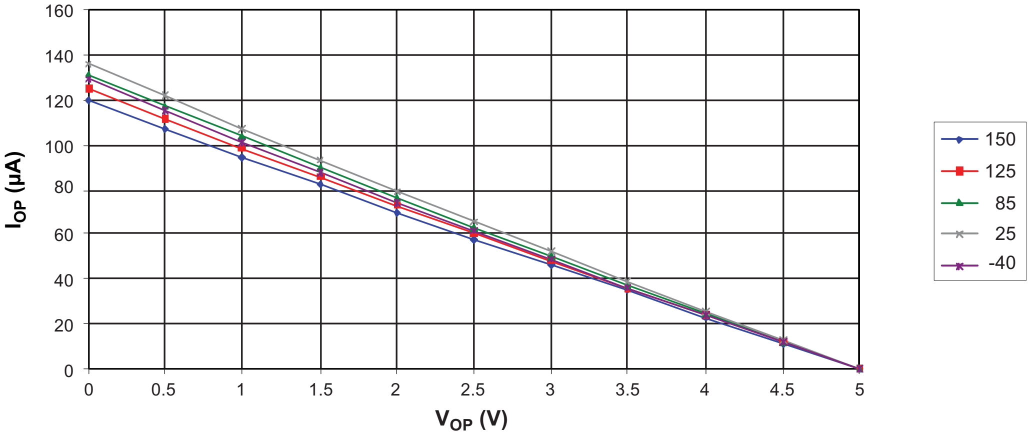
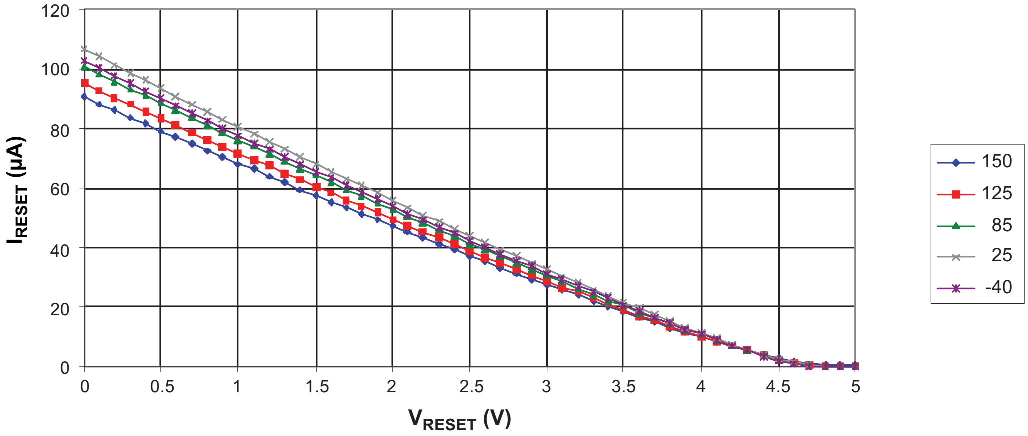


Figure 29-33. ATmega324P-B: Reset Pull-up Resistor Current versus Reset Pin Voltage (V_{CC} = 5V)



29.2.5 Pin Driver Strength

Figure 29-34. ATmega324P-B: I/O Pin Output Voltage versus Sink Current (V_{CC} = 3V)

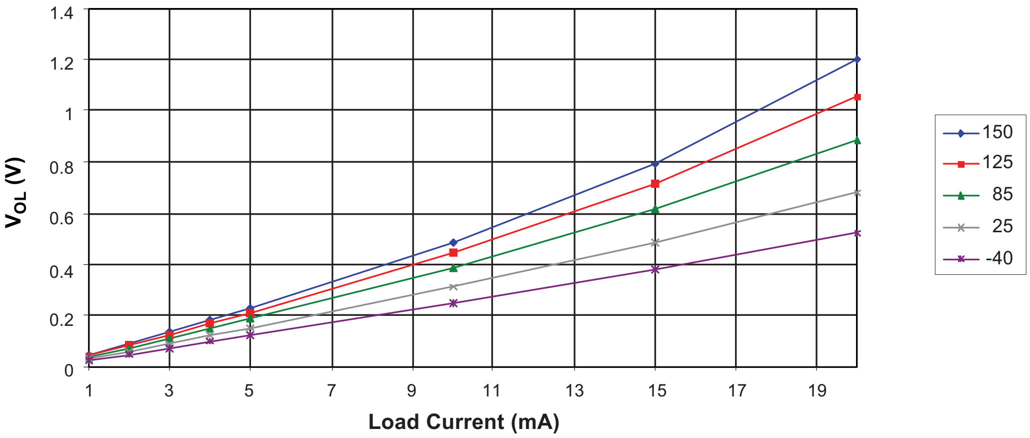


Figure 29-35. ATmega324P-B: I/O Pin Output Voltage versus Sink Current (V_{CC} = 5V)

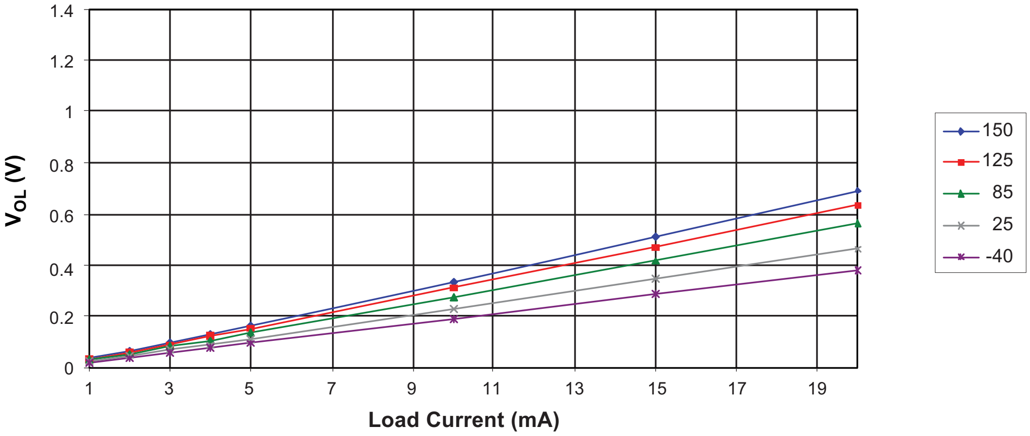


Figure 29-36. ATmega324P-B: I/O Pin Output Voltage versus Source Current (V_{CC} = 3V)

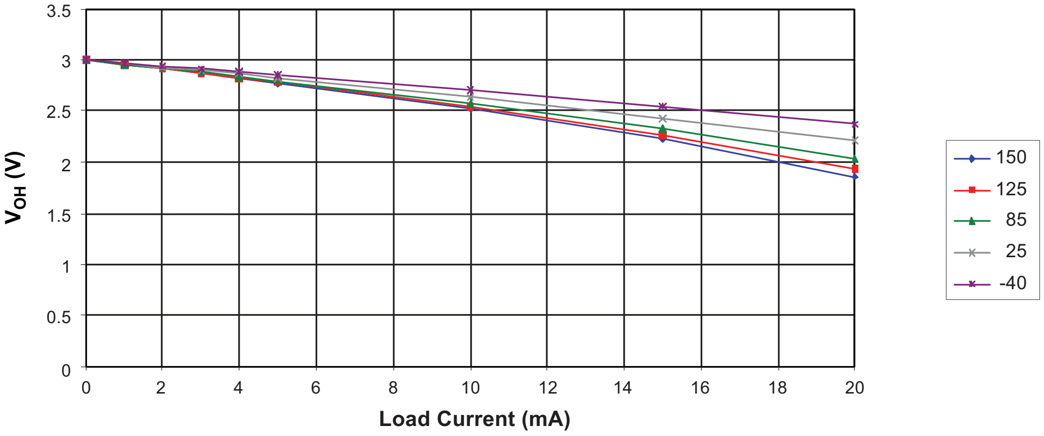
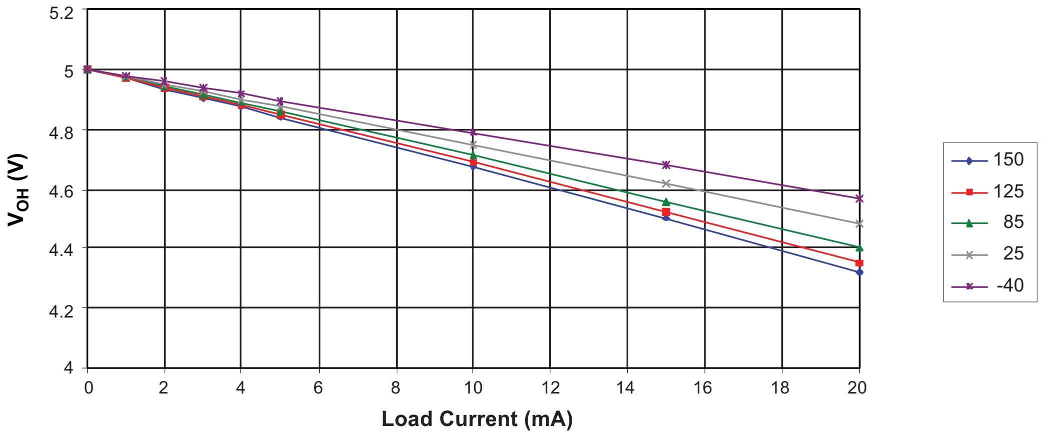


Figure 29-37. ATmega324P-B: I/O Pin Output Voltage versus Source Current (V_{CC} = 5V)



29.2.6 Pin Threshold

Figure 29-38. ATmega324P-B: I/O Pin Input Threshold versus V_{CC} (V_{IH}, I/O Pin Read as '1')

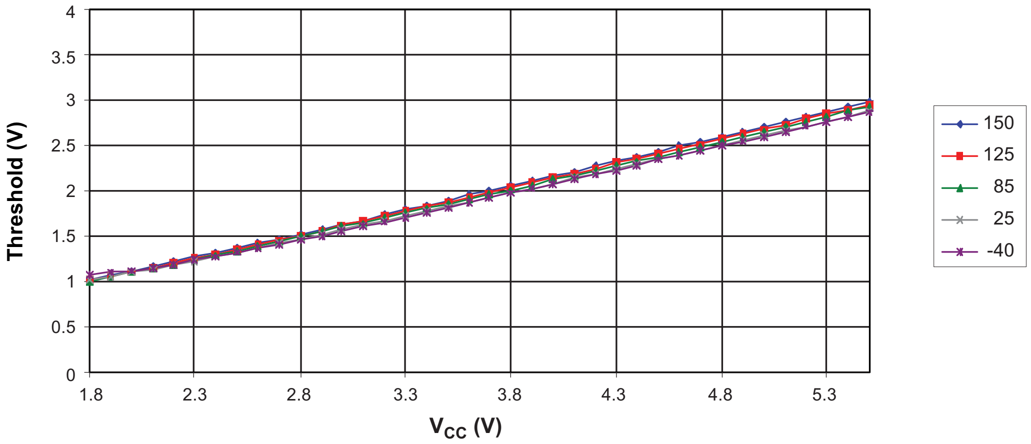


Figure 29-39. ATmega324P-B: I/O Pin Input Threshold versus V_{CC} (V_{IL}, I/O Pin Read as '0')

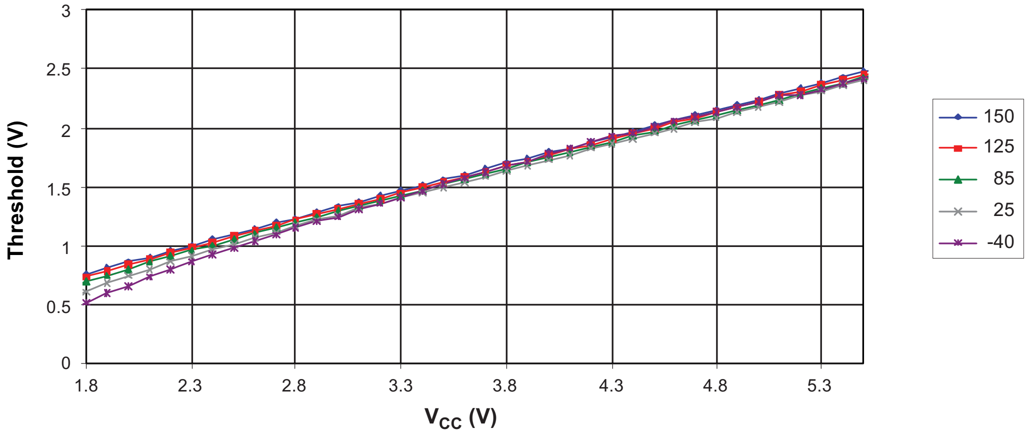


Figure 29-40. ATmega324P-B: Reset Pin Input Threshold versus V_{CC} (V_{IH}, I/O Pin Read as '1')

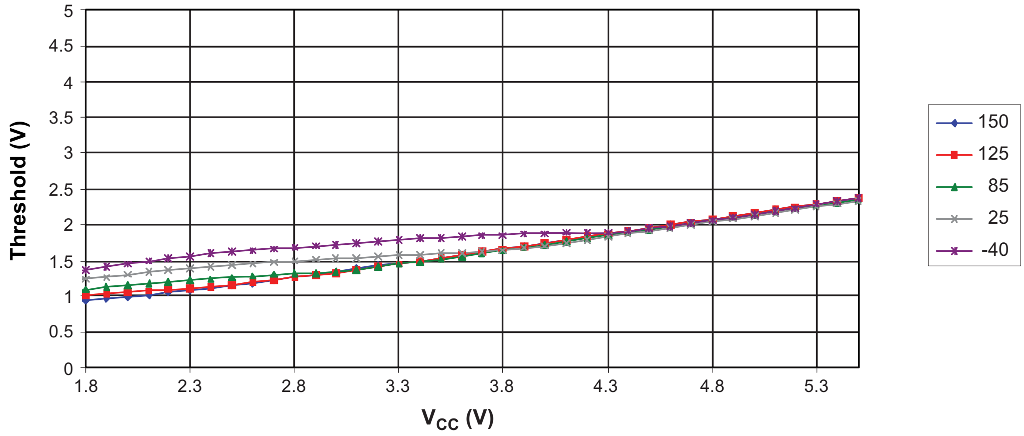
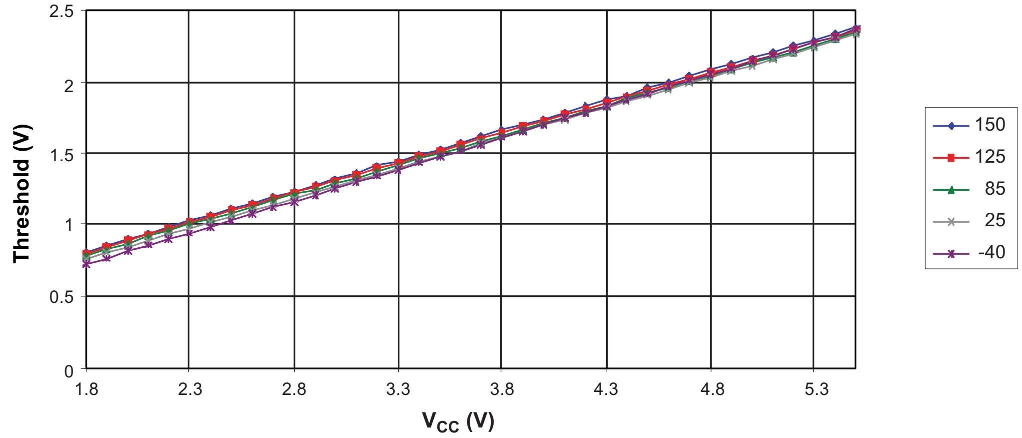


Figure 29-41. ATmega324P-B: Reset Pin Input Threshold versus V_{CC} (V_{IL}, I/O Pin Read as '0')



29.2.7 BOD Threshold

Figure 29-42. ATmega324P-B: BOD Threshold versus Temperature (V_{BOT} = 4.3V)

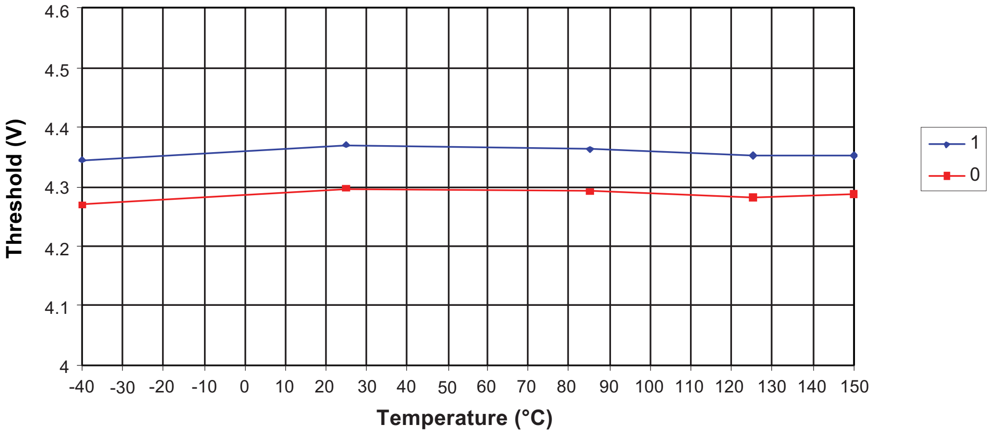


Figure 29-43. ATmega324P-B: BOD Threshold versus Temperature (V_{BOT} = 2.7V)

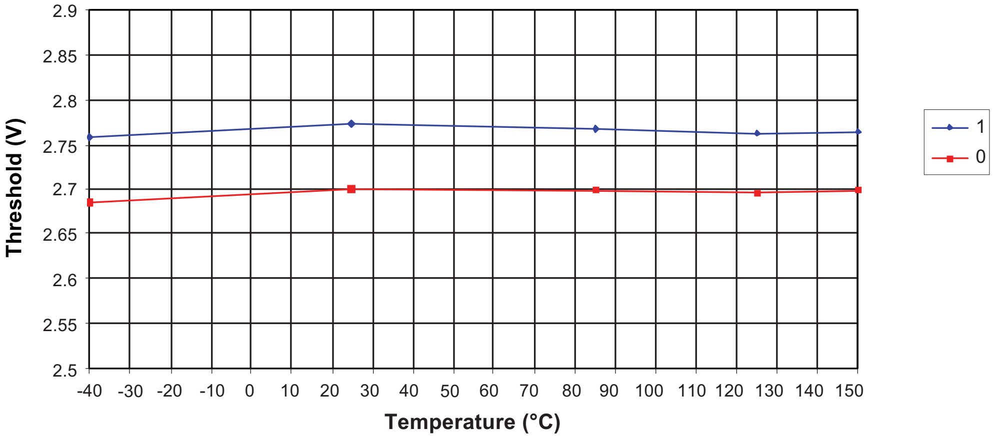


Figure 29-44. ATmega324P-B: Calibrated Bandgap Voltage versus V_{CC}

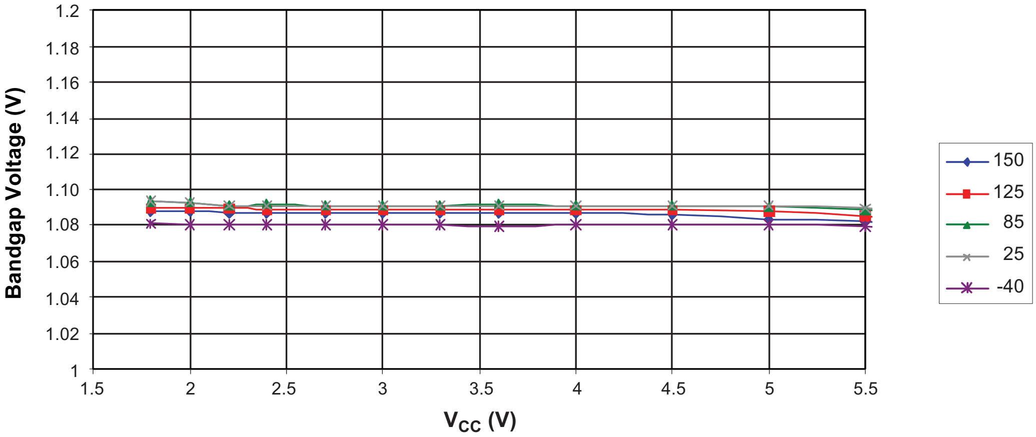
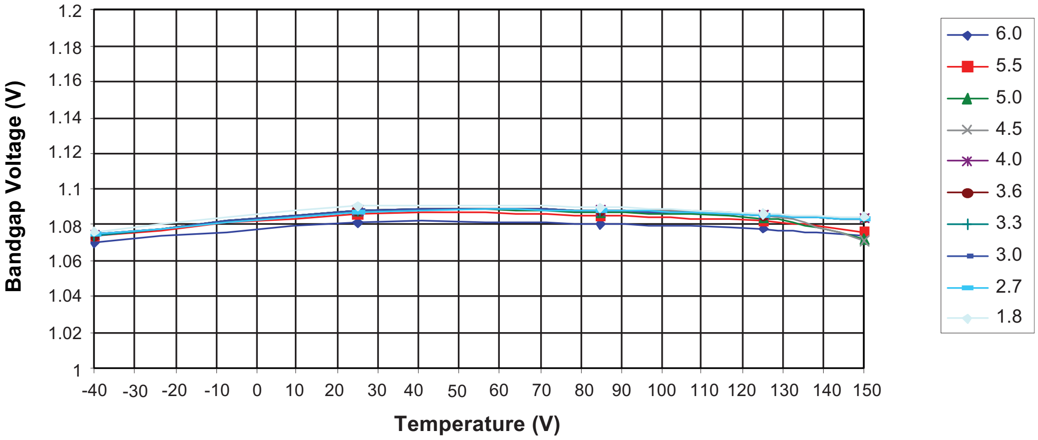


Figure 29-45. ATmega324P-B: Bandgap Voltage versus Temperature



29.2.8 Internal Oscillator Speed

Figure 29-46. ATmega324P-B: Watchdog Oscillator Frequency versus Temperature

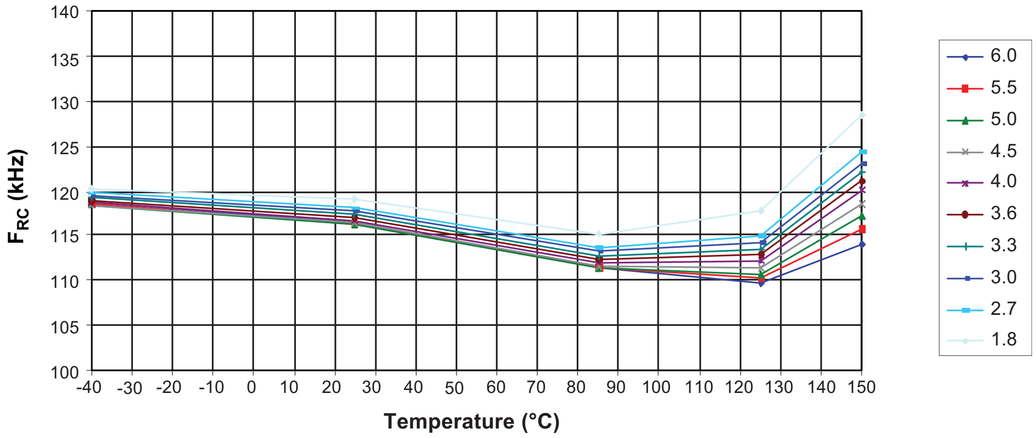


Figure 29-47. ATmega324P-B: Watchdog Oscillator Frequency versus V_{CC}

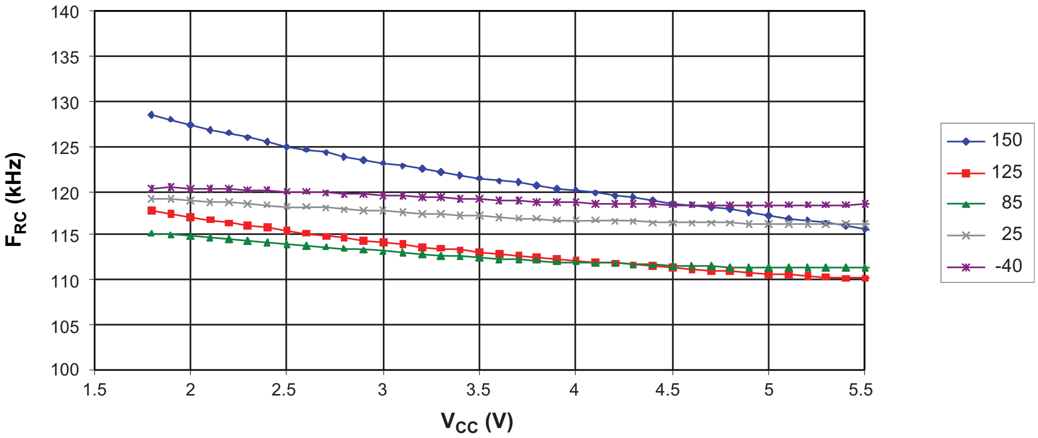


Figure 29-48. ATmega324P-B: Calibrated 8MHz RC Oscillator versus V_{CC}

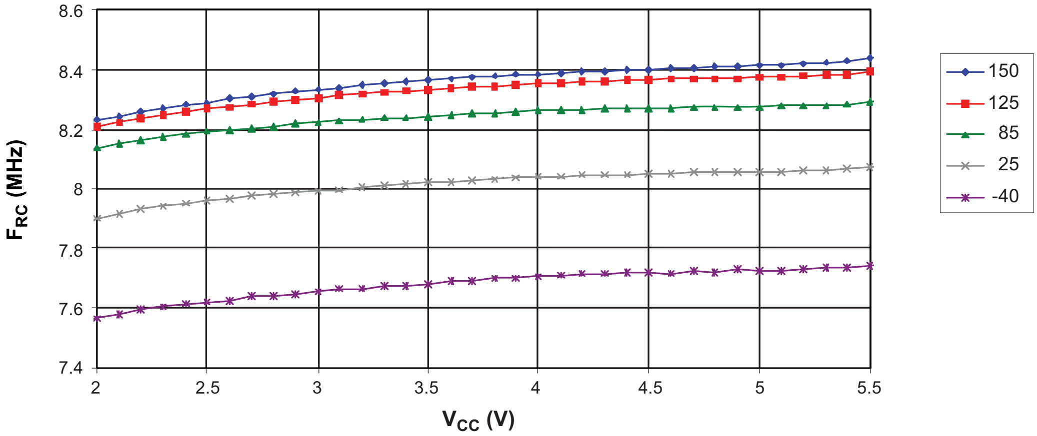


Figure 29-49. ATmega324P-B: Calibrated 8MHz RC Oscillator versus Temperature

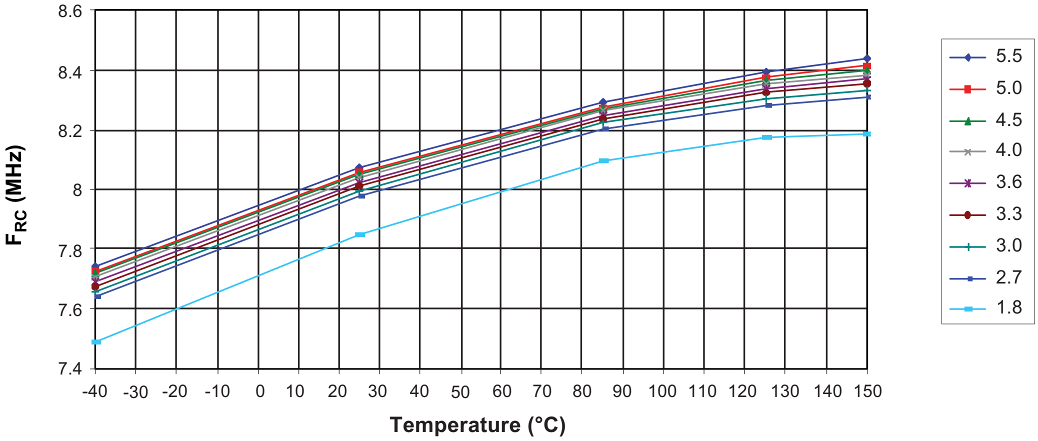
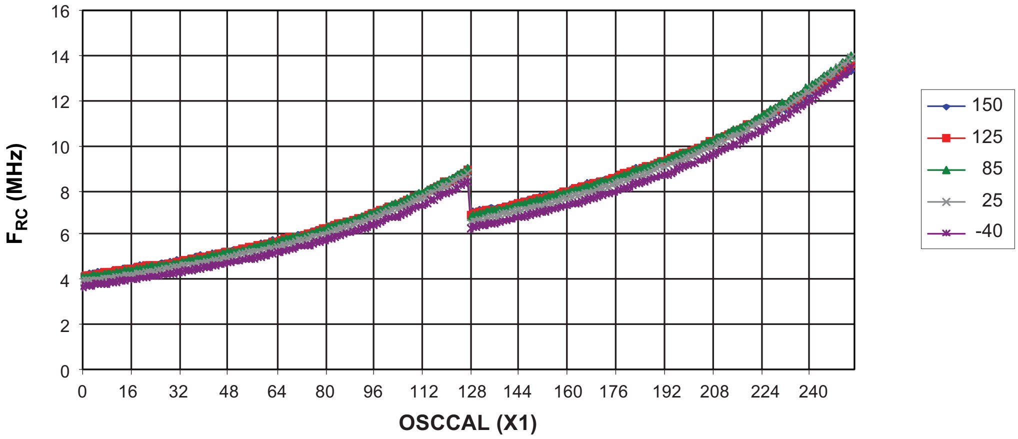


Figure 29-50. ATmega324P-B: Calibrated 8MHz RC Oscillator versus OSCCAL Value



29.3 ATmega644P-B Typical Characteristics

29.3.1 Active Supply Current

Figure 29-51. ATmega644P-B: Active Supply Current versus Low Frequency (0.1 - 1.0MHz)

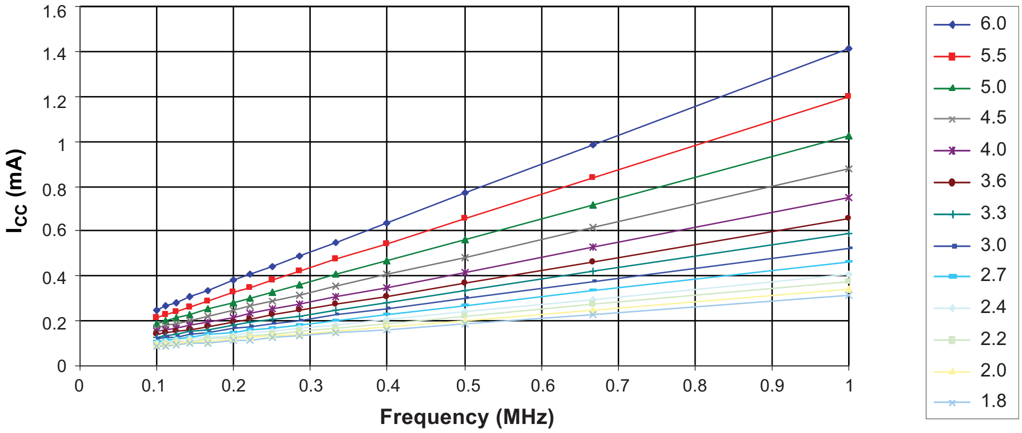
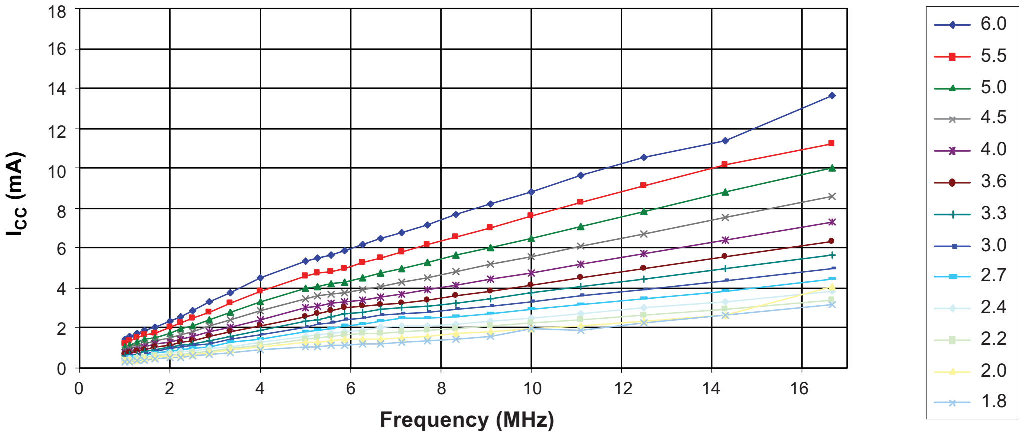


Figure 29-52. ATmega644P-B: Active Supply Current versus Frequency (1 - 16MHz)



29.3.2 Idle Supply Current

Figure 29-53. ATmega644P-B: Idle Supply Current versus Low Frequency (0.1 - 1.0MHz)

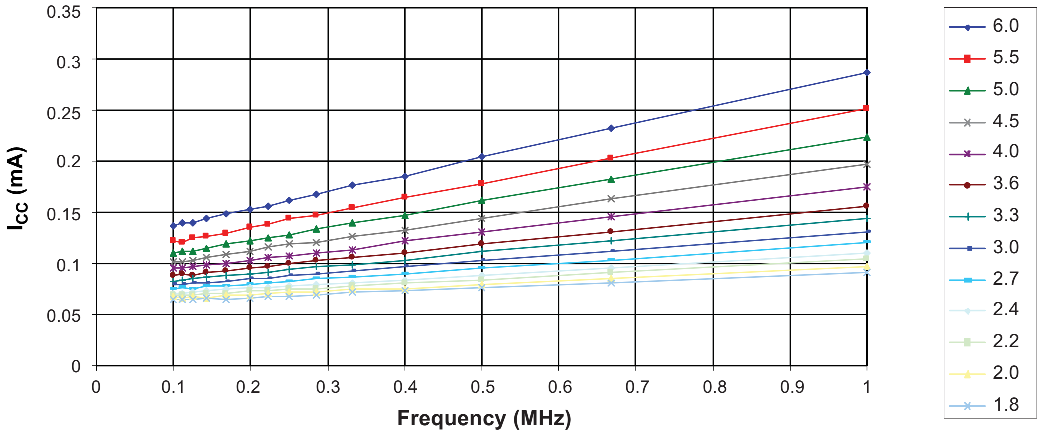
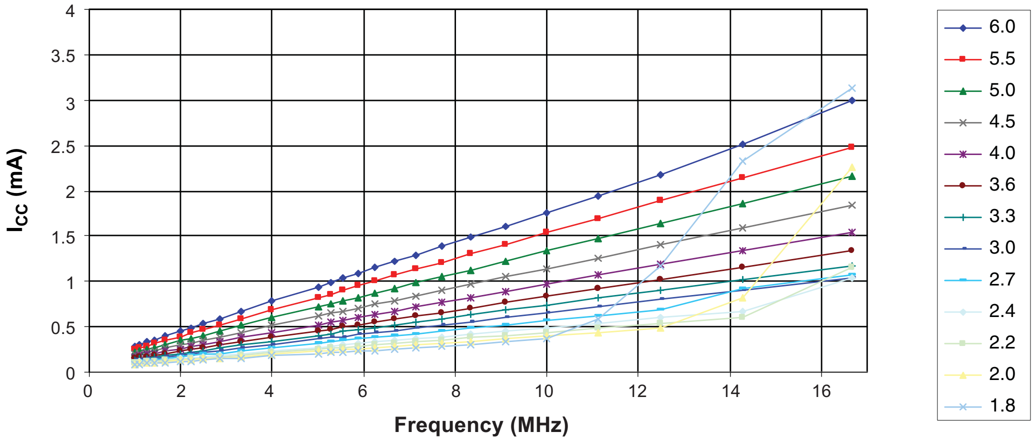


Figure 29-54. ATmega644P-B: Idle Supply Current versus Frequency (1 - 16MHz)



29.3.3 Power-down Supply Current

Figure 29-55. ATmega644P-B: Power-down Supply Current versus V_{CC} (Watchdog Timer Disabled)

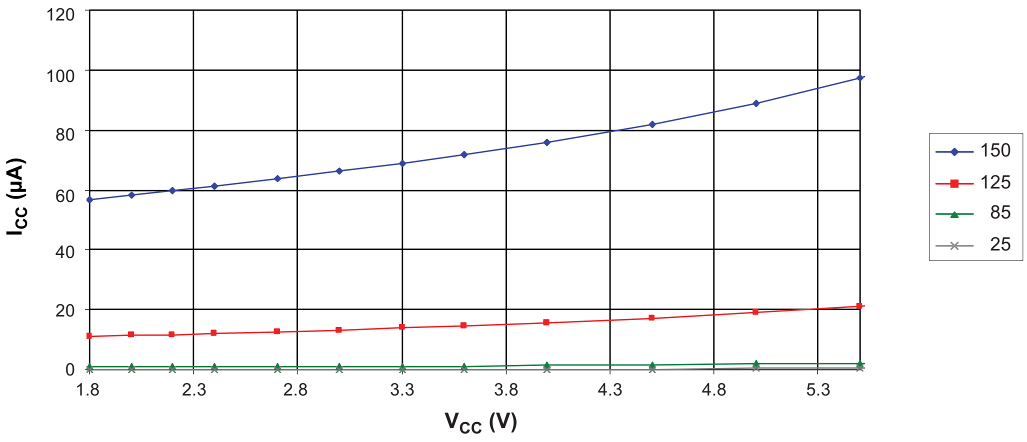
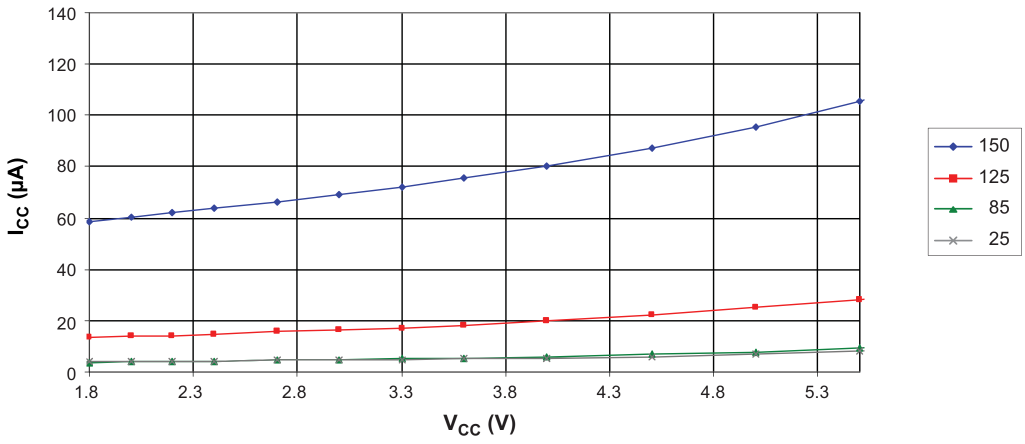


Figure 29-56. ATmega644P-B: Power-down Supply Current versus V_{CC} (Watchdog Timer Enabled)



29.3.4 Pin Pull-up

Figure 29-57. ATmega644P-B: I/O Pin Pull-up Resistor Current versus Input Voltage (V_{CC} = 5V)

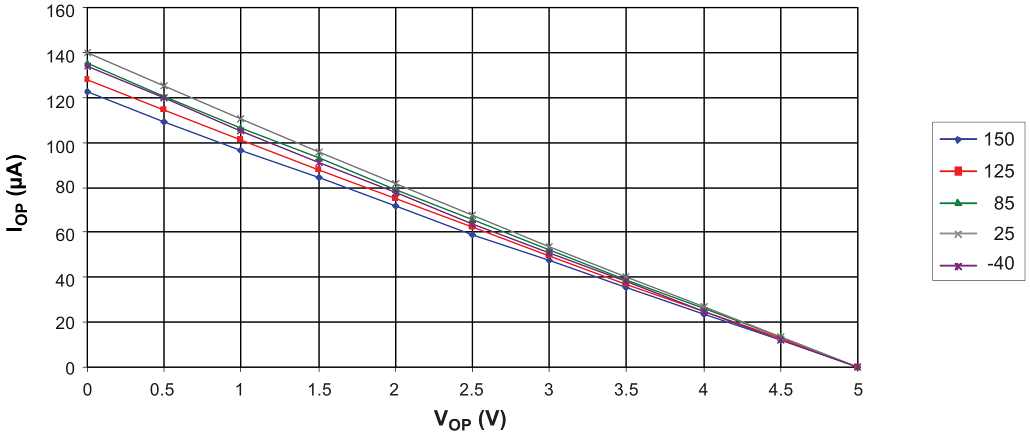
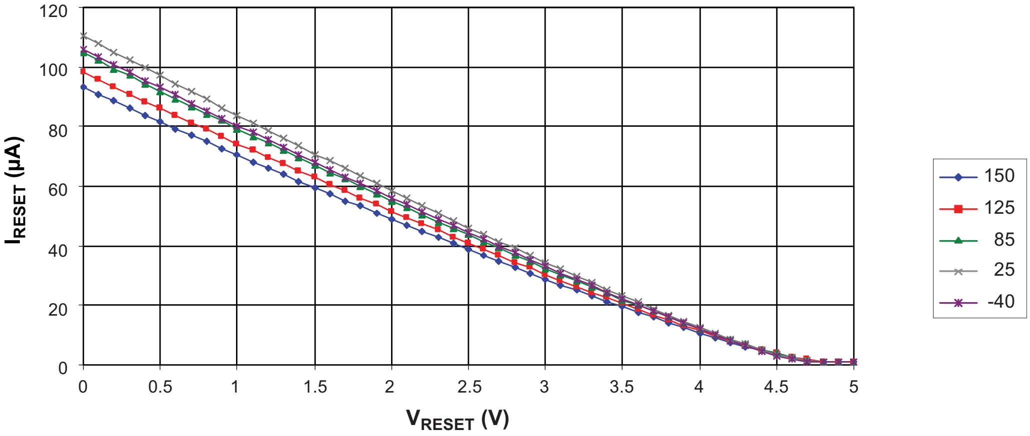


Figure 29-58. ATmega644P-B: Reset Pull-up Resistor Current versus Reset Pin Voltage (V_{CC} = 5V)



29.3.5 Pin Driver Strength

Figure 29-59. ATmega644P-B: I/O Pin Output Voltage versus Sink Current (V_{CC} = 3V)

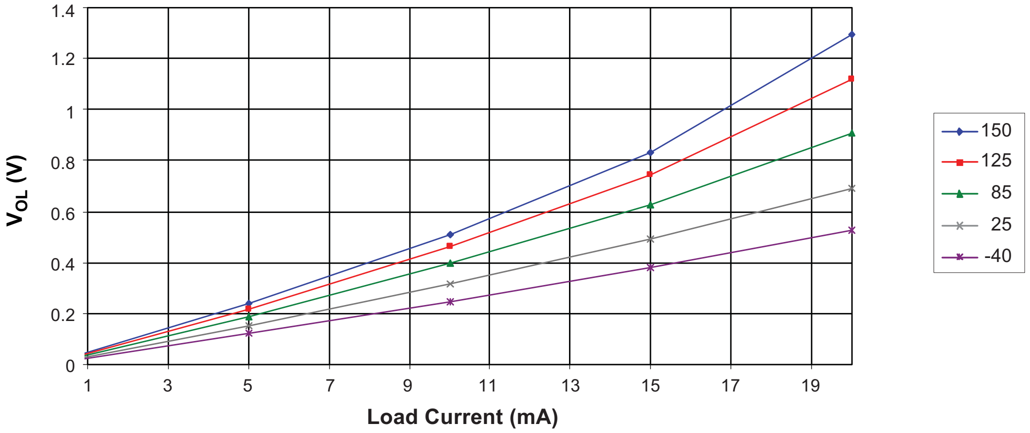


Figure 29-60. ATmega644P-B: I/O Pin Output Voltage versus Sink Current (V_{CC} = 5V)

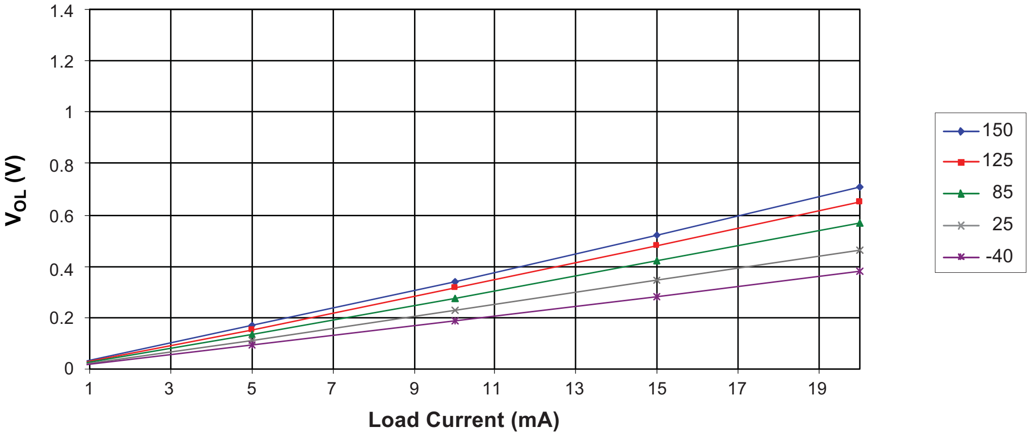


Figure 29-61. ATmega644P-B: I/O Pin Output Voltage versus Source Current (V_{CC} = 3V)

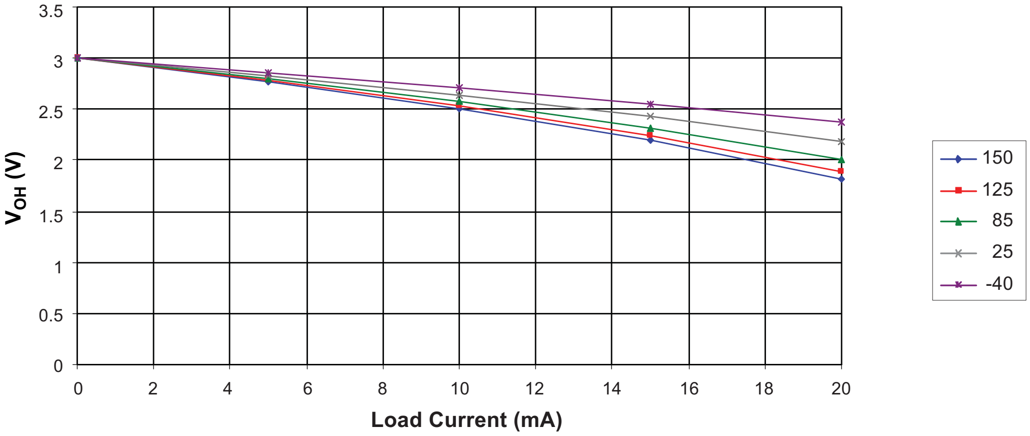
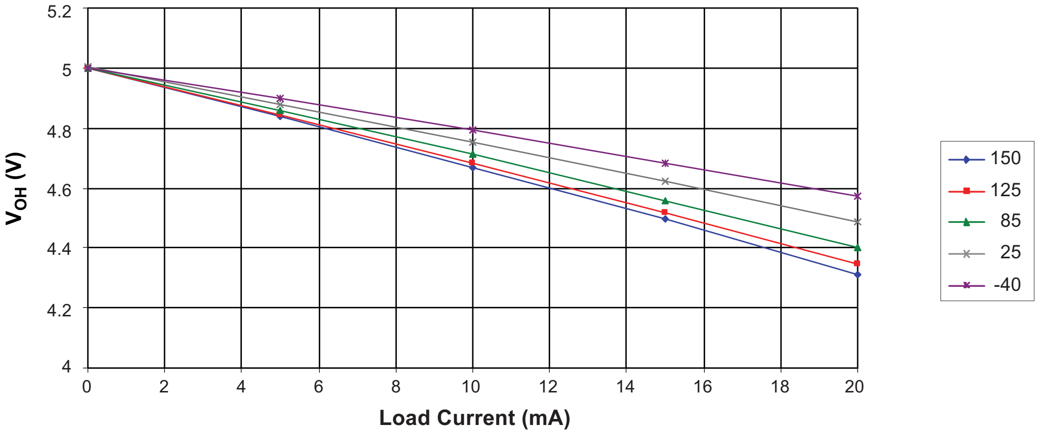


Figure 29-62. ATmega644P-B: I/O Pin Output Voltage versus Source Current (V_{CC} = 5V)



29.3.6 Pin Threshold

Figure 29-63. ATmega644P-B: I/O Pin Input Threshold versus V_{CC} (V_{IH}, I/O Pin Read as '1')

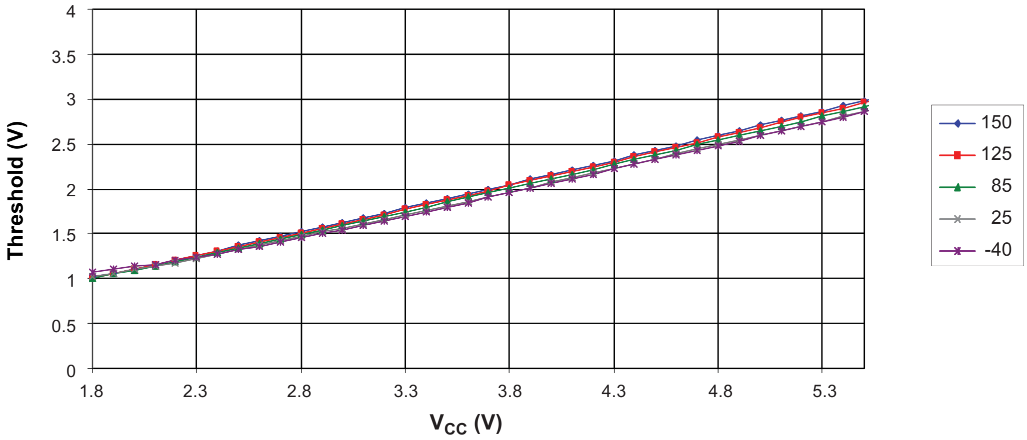


Figure 29-64. ATmega644P-B: I/O Pin Input Threshold versus V_{CC} (V_{IL}, I/O Pin Read as '0')

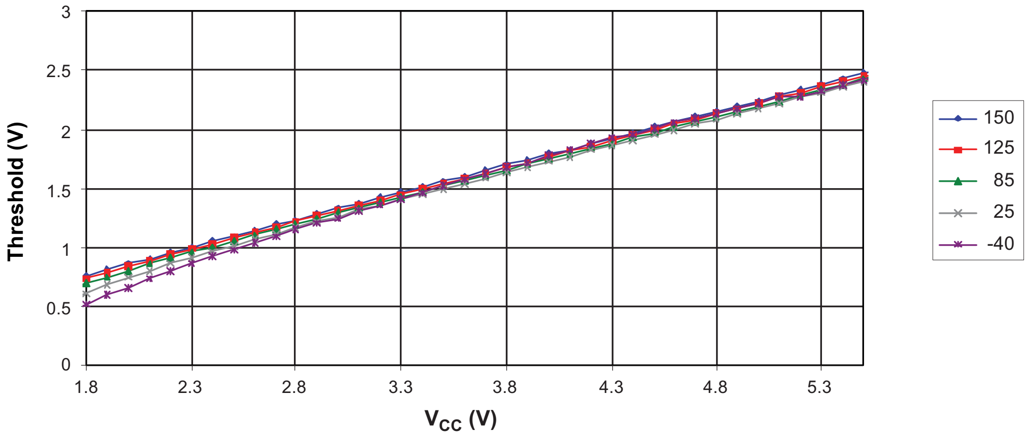


Figure 29-65. ATmega644P-B: Reset Pin Input Threshold versus V_{CC} (V_{IH}, I/O Pin Read as '1')

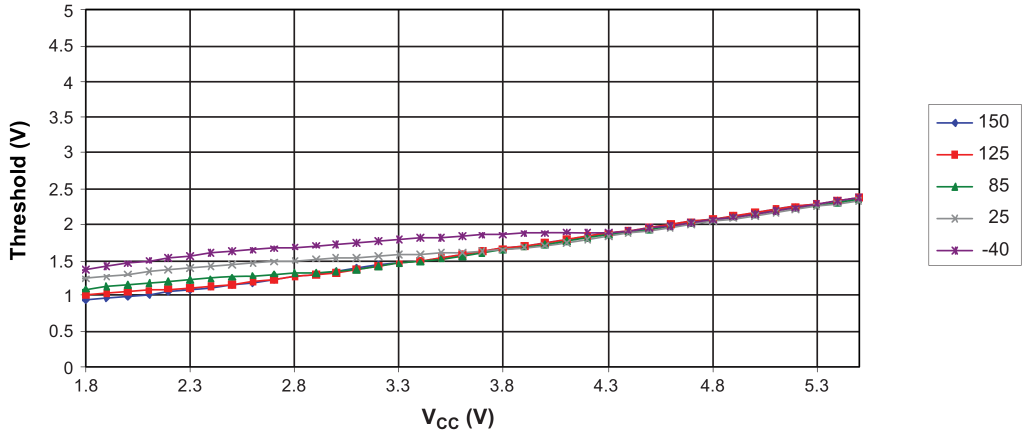
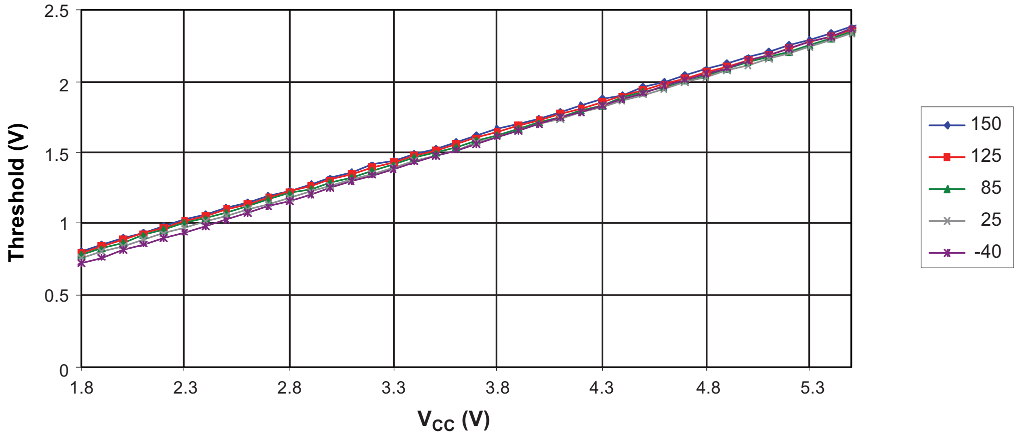


Figure 29-66. ATmega644P-B: Reset Pin Input Threshold versus V_{CC} (V_{IL}, I/O Pin Read as '0')



29.3.7 BOD Threshold

Figure 29-67. ATmega644P-B: BOD Threshold versus Temperature (V_{BOT} = 4.3V)

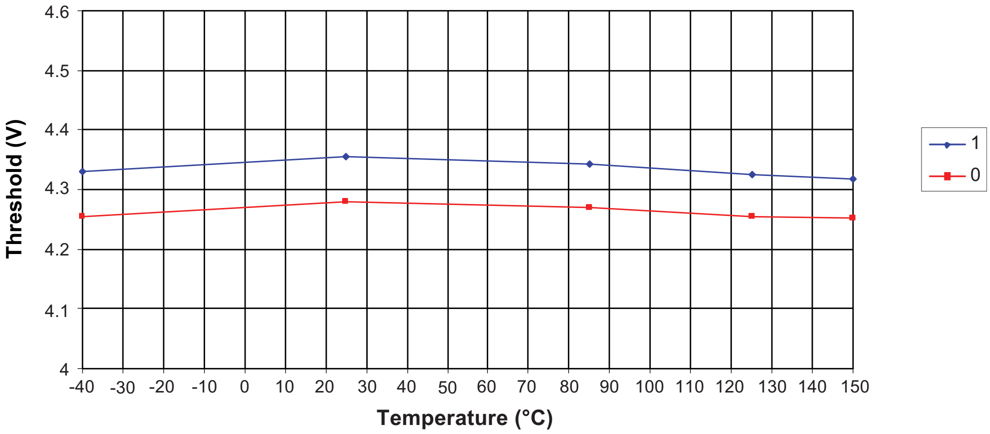


Figure 29-68. ATmega644P-B: BOD Threshold versus Temperature (V_{BOT} = 2.7V)

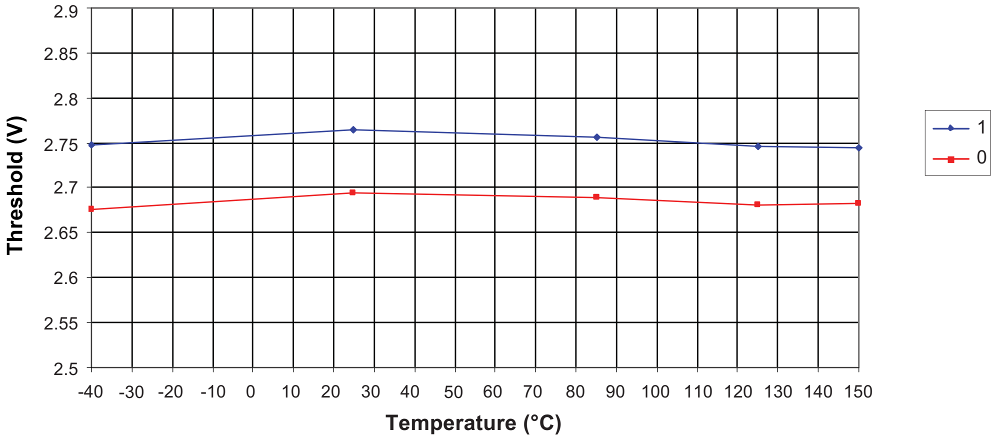


Figure 29-69. ATmega644P-B: Calibrated Bandgap Voltage versus V_{CC}

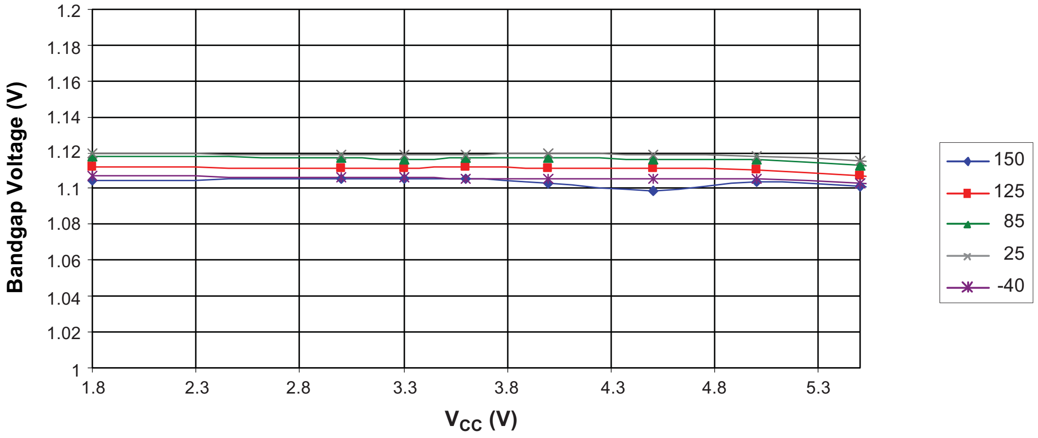
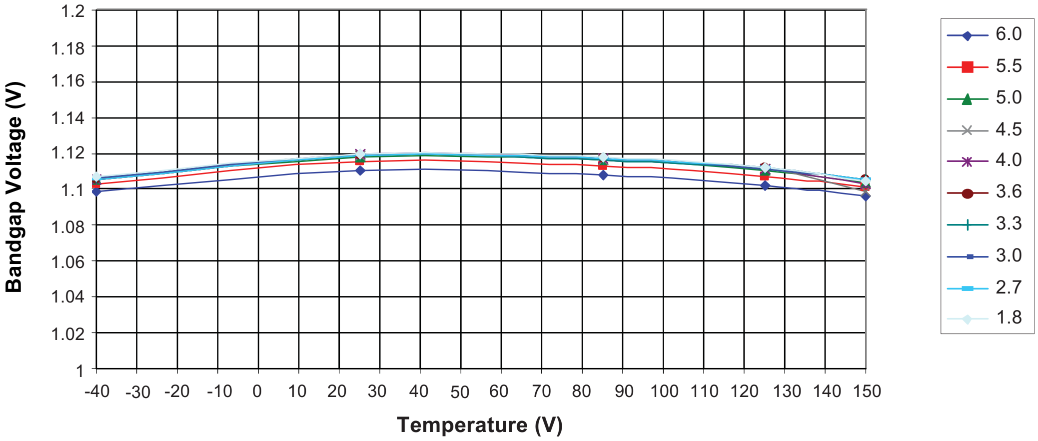


Figure 29-70. ATmega644P-B: Bandgap Voltage versus Temperature



29.3.8 Internal Oscillator Speed

Figure 29-71. ATmega644P-B: Watchdog Oscillator Frequency versus Temperature

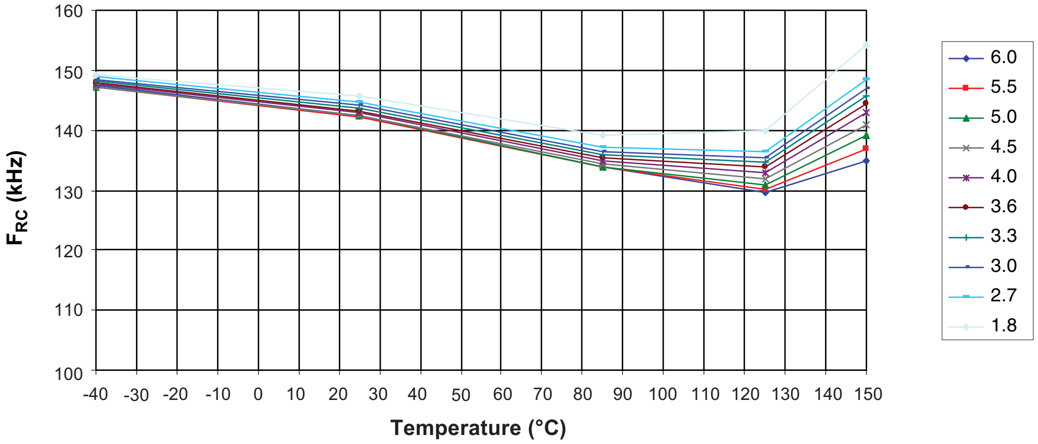


Figure 29-72. ATmega644P-B: Watchdog Oscillator Frequency versus V_{CC}

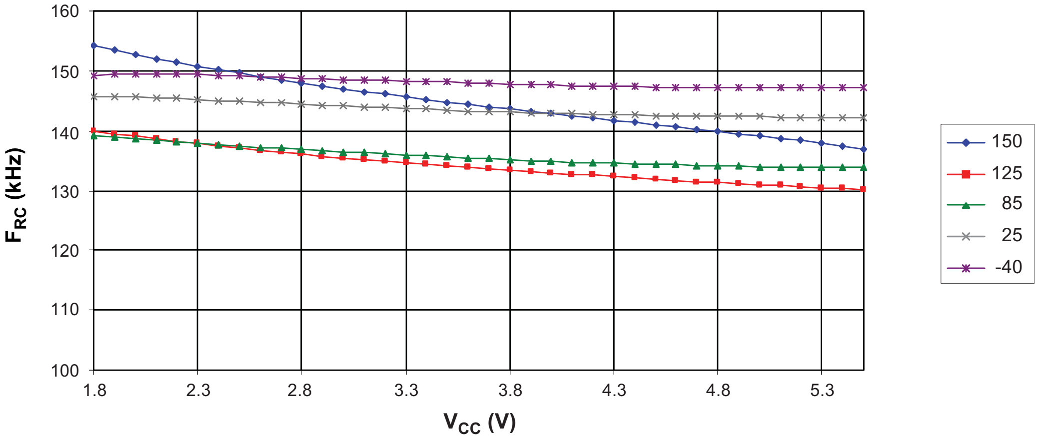


Figure 29-73. ATmega644P-B: Calibrated 8MHz RC Oscillator versus V_{CC}

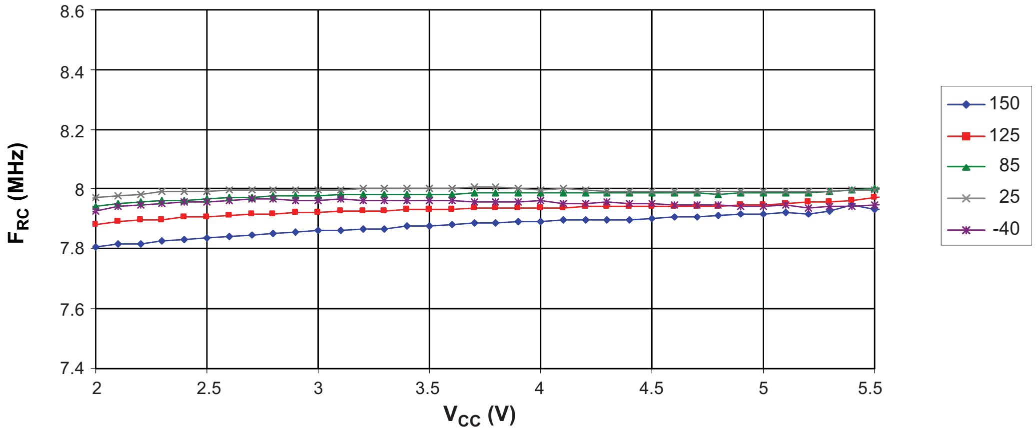


Figure 29-74. ATmega644P-B: Calibrated 8MHz RC Oscillator versus Temperature

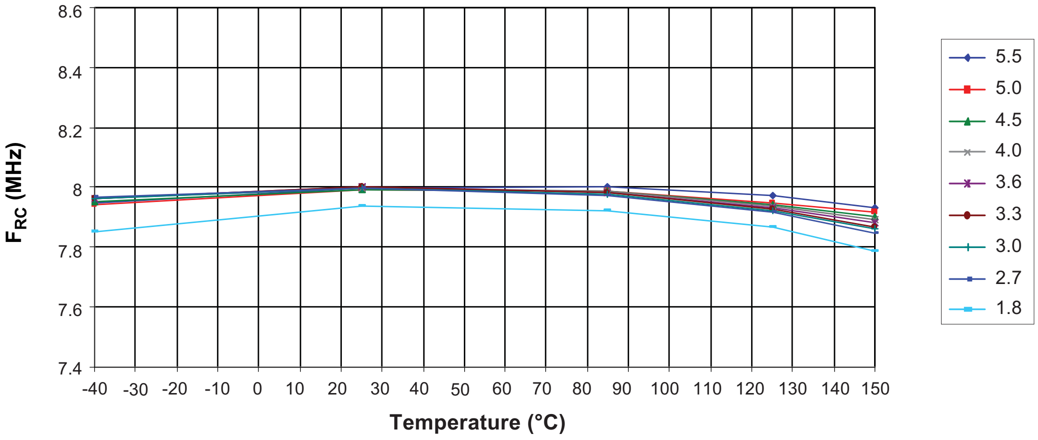
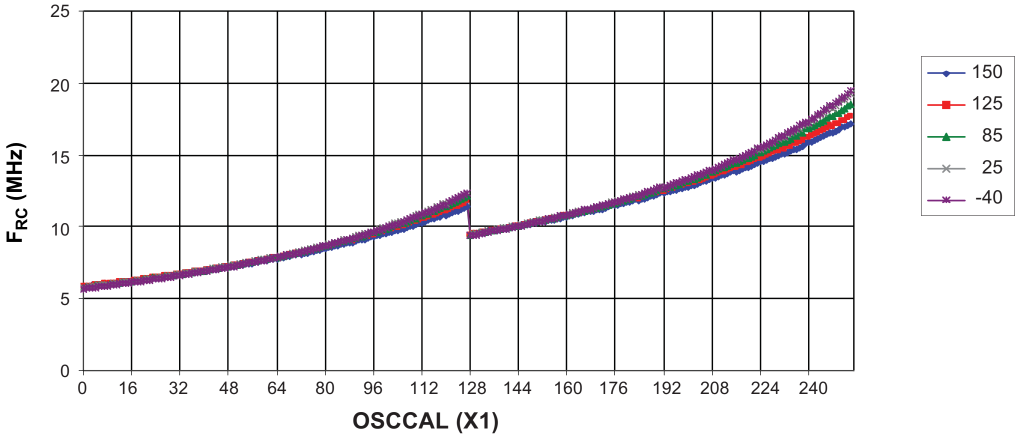


Figure 29-75. ATmega644P-B: Calibrated 8MHz RC Oscillator versus OSCCAL Value



30. Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xFF)	Reserved	-	-	-	-		-	-	-	
(0xFE)	Reserved	-	-	-	-	-	-	-	-	
(0xFD)	Reserved	-	-	-	-	-	-	-	-	
(0xFC)	Reserved TCCR2B	-	-	-	-	-	-	-	-	
(0xFB)	Reserved	-	-	-	-		-	-	-	
(0xFA)	Reserved	-	-	-	-	-	-	-	-	
(0xF9)	Reserved	-	-	-	-		-	-	-	
(0xF8)	Reserved	-	-	-	-	-	-	-	-	
(0xF7)	Reserved	-	-	-	-	-	-	-	-	
(0xF6)	Reserved TCCR2B	-	-	-	-	-	-	-	-	
(0xF5)	Reserved	-	-	-	-		-	-	-	
(0xF4)	Reserved	-	-	-	-	-	-	-	-	
(0xF3)	Reserved	-	-	-	-	-	-	-	-	
(0xF2)	Reserved TCCR2B	-	-	-	-	-	-	-	-	
(0xF1)	Reserved	-	-	-	-		-	-	-	
(0xF0)	Reserved	-	-	-	-	-	-	-	-	
(0xEF)	Reserved	-	-	-	-		-	-	-	
(0xEE)	Reserved	-	-	-	-	-	-	-	-	
(0xED)	Reserved	-	-	-	-	-	-	-	-	
(0xEC)	Reserved TCCR2B	-	-	-	-	-	-	-	-	
(0xEB)	Reserved	-	-	-	-		-	-	-	
(0xEA)	Reserved	-	-	-	-	-	-	-	-	
(0xE9)	Reserved	-	-	-	-	-	-	-	-	
(0xE8)	Reserved TCCR2B	-	-	-	-	-	-	-	-	
(0xE7)	Reserved	-	-	-	-		-	-	-	
(0xE6)	Reserved	-	-	-	-	-	-	-	-	
(0xE5)	Reserved	-	-	-	-	-	-	-	-	
(0xE4)	Reserved TCCR2B	-	-	-	-	-	-	-	-	
(0xE3)	Reserved	-	-	-	-		-	-	-	
(0xE2)	Reserved	-	-	-	-	-	-	-	-	
(0xE1)	Reserved	-	-	-	-	-	-	-	-	
(0xE0)	Reserved	-	-	-	-		-	-	-	

Notes:

1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O registers within the address range $00 - $1F 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 the CBI and SBI instructions will operate on all bits in the I/O register, writing a one back into any flag read as set, thus clearing the flag. 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 $00 - $3F must be used. When addressing I/O registers as data space using LD and ST instructions, $20 must be added to these addresses.
   The ATmega164P-B/324P-B/644P-B 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 $60 - $FF, only the ST/STS/STD and LD/LDS/LDD instructions can be used.
5. USART in SPI master mode.

30. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xDF)	Reserved	-	-	-	-	-	-	-	-	
(0xDE)	Reserved	-	-	-	-	-	-	-	-	
(0xDD)	Reserved TCCR2B	-	-	-	-	-	-	-	-	
(0xDC)	Reserved	-	-	-	-		-	-	-	
(0xDB)	Reserved	-	-	-	-	-	-	-	-	
(0xDA)	Reserved	-	-	-	-	-	-	-	-	
(0xD9)	Reserved	-	-	-	-	-	-	-	-	
(0xD8)	Reserved	-	-	-	-	-	-	-	-	
(0xD7)	Reserved	-	-	-	-	-	-	-	-	
(0xD6)	Reserved	-	-	-	-	-	-	-	-	
(0xD5)	Reserved	-	-	-	-	-	-	-	-	
(0xD4)	Reserved	-	-	-	-	-	-	-	-	
(0xD3)	Reserved	-	-	-	-	-	-	-	-	
(0xD2)	Reserved	-	-	-	-	-	-	-	-	
(0xD1)	Reserved	-	-	-	-	-	-	-	-	
(0xD0)	Reserved	-	-	-	-	-	-	-	-	
(0xCF)	Reserved	-	-	-	-	-	-	-	-	
(0xCE)	UDR1	USART1 I/O data register USART0 I/O data register								163
(0xCD)	UBRR1H	-	-	-	-	USART1 baud rate register high byte				166/175
(0xCC)	UBRR1L	USART1 baud rate Register low byte								166/175
(0xCB)	Reserved	-	-	-	-	-	-	-	-	
(0xCA)	UCSR1C	UMSEL11	UMSEL10	UPM11	UPM10	USBS1	UCSZ11/UDORD0[5]	UCSZ10/UCPHA0[5]	UCPOL1	165/175
(0xC9)	UCSR1B	RXCIE1	TXCIE1	UDRIE1	RXEN1	TXEN1	UCSZ12	RXB81	TXB81	164/174
(0xC8)	UCSR1A	RXC1	TXC1	UDRE1	FE1	DOR1	UPE1	U2X1	MPCM1	163/173
(0xC7)	Reserved	-	-	-	-	-	-	-	-	
(0xC6)	UDR0	USART0 I/O data register USART0 I/O data register								163
(0xC5)	UBRR0H	-	-	-	-	USART0 baud rate register high byte				166/175
(0xC4)	UBRR0L	USART0 baud rate register low byte								166/175
(0xC3)	Reserved	-	-	-	-	-	-	-	-	
(0xC2)	UCSR0C	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01/UDORD0[5]	UCSZ00/UCPHA0[5]	UCPOL0	165/175
(0xC1)	UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80	164/174
(0xC0)	UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0	163/173
(0xBF)	Reserved	-	-	-	-	-	-	-	-	
(0xBE)	Reserved	-	-	-	-	-	-	-	-	

Notes:

1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O registers within the address range $00 - $1F 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 the CBI and SBI instructions will operate on all bits in the I/O register, writing a one back into any flag read as set, thus clearing the flag. 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 $00 - $3F must be used. When addressing I/O registers as data space using LD and ST instructions, $20 must be added to these addresses.
   The ATmega164P-B/324P-B/644P-B 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 $60 - $FF, only the ST/STS/STD and LD/LDS/LDD instructions can be used.
5. USART in SPI master mode.

30. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xBD)	TWAMR	TWAM6	TWAM5	TWAM4	TWAM3	TWAM2	TWAM1	TWAM0	-	203
(0xBC)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE	200
(0xBB)	TWDR	2-wire serial interface data register								202
(0xBA)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	203
(0xB9)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	-	TWPS1	TWPS0	202
(0xB8)	TWBR	2-wire serial interface bit rate register								200
(0xB7)	Reserved	-	-	-	-	-	-	-	-	
(0xB6)	ASSR	-	EXCLK	AS2	TCN2UB	OCR2AB	OCR2BUB	TCR2AUB	TCR2BB	136
(0xB5)	Reserved	-	-	-	-	-	-	-	-	
(0xB4)	OCR2B	Timer/Counter2 output compare register B								136
(0xB3)	OCR2A	Timer/Counter2 output compare register A								136
(0xB2)	TCNT2	Timer/Counter2 (8 bit)								136
(0xB1)	TCCR2B	FOC2A	FOC2B	-	-	WGM22	CS22	CS21	CS20	135
(0xB0)	TCCR2A	COM2A1	COM2A0	COM2B1	COM2B0	-	-	WGM21	WGM20	132
(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)	OCR3BH	Timer/Counter3 - output compare register B high byte								114
(0x9A)	OCR3BL	Timer/Counter3 - output compare register B low byte								114

Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O registers within the address range $00 - $1F 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 the CBI and SBI instructions will operate on all bits in the I/O register, writing a one back into any flag read as set, thus clearing the flag. 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 $00 - $3F must be used. When addressing I/O registers as data space using LD and ST instructions, $20 must be added to these addresses.
   The ATmega164P-B/324P-B/644P-B 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 $60 - $FF, only the ST/STS/STD and LD/LDS/LDD instructions can be used.
5. USART in SPI master mode.

30. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0x99)	OCR3AH	Timer/Counter3 - output compare register A high byte								113
(0x98)	OCR3AL	Timer/Counter3 - output compare register A low byte								113
(0x97)	ICR3H	Timer/Counter3 - input capture register high byte								114
(0x96)	ICR3L	Timer/Counter3 - input capture register low byte								114
(0x95)	TCNT3H	Timer/Counter3 - counter register high byte								113
(0x94)	TCNT3L	Timer/Counter3 - counter register low byte								113
(0x93)	Reserved	-	-	-	-	-	-	-	-	
(0x92)	TCCR3C	FOC3A	FOC3B	-	-	-	-	-	-	112
(0x91)	TCCR3B	ICNC3	ICES3	-	WGM33	WGM32	CS32	CS31	CS30	111
(0x90)	TCCR3A	COM3A1	COM3A0	COM3B1	COM3B0	-	-	WGM31	WGM30	109
(0x8F)	Reserved	-	-	-	-	-	-	-	-	
(0x8E)	Reserved	-	-	-	-	-	-	-	-	
(0x8D)	Reserved	-	-	-	-	-	-	-	-	
(0x8C)	Reserved	-	-	-	-	-	-	-	-	
(0x8B)	OCR1BH	Timer/Counter1 - output compare register B high byte								114
(0x8A)	OCR1BL	Timer/Counter1 - output compare register B low byte								114
(0x89)	OCR1AH	Timer/Counter1 - output compare register A high byte								113
(0x88)	OCR1AL	Timer/Counter1 - output compare register A low byte								113
(0x87)	ICR1H	Timer/Counter1 - input capture register high byte								114
(0x86)	ICR1L	Timer/Counter1 - input capture register low byte								114
(0x85)	TCNT1H	Timer/Counter1 - counter register high byte								113
(0x84)	TCNT1L	Timer/Counter1 - counter register low byte								113
(0x83)	Reserved	-	-	-	-	-	-	-	-	
(0x82)	TCCR1C	FOC1A	FOC1B	-	-	-	-	-	-	112
(0x81)	TCCR1B	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	111
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	-	-	WGM11	WGM10	109
(0x7F)	DIDR1	-	-	-	-	-	-	AIN1D	AIN0D	206
(0x7E)	DIDR0	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	224
(0x7D)	Reserved	-	-	-	-	-	-	-	-	
(0x7C)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	220
(0x7B)	ADCSRB	-	ACME	-	-	-	ADTS2	ADTS1	ADTS0	205
(0x7A)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	222
(0x79)	ADCH	ADC data register high byte								223
(0x78)	ADCL	ADC data register low byte								223
(0x77)	Reserved	-	-	-	-	-	-	-	-	
(0x76)	Reserved	-	-	-	-	-	-	-	-	

Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O registers within the address range $00 - $1F 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 the CBI and SBI instructions will operate on all bits in the I/O register, writing a one back into any flag read as set, thus clearing the flag. 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 $00 - $3F must be used. When addressing I/O registers as data space using LD and ST instructions, $20 must be added to these addresses.
   The ATmega164P-B/324P-B/644P-B 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 $60 - $FF, only the ST/STS/STD and LD/LDS/LDD instructions can be used.
5. USART in SPI master mode.

30. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0x75)	Reserved	-	-	-	-	-	-	-	-	
(0x74)	Reserved	-	-	-	-	-	-	-	-	
(0x73)	PCMSK3	PCINT31	PCINT30	PCINT29	PCINT28	PCINT27	PCINT26	PCINT25	PCINT24	55
(0x72)	Reserved	-	-	-	-	-	-	-	-	
(0x71)	TIMSK3	-	-	ICIE3	-	-	OCIE3B	OCIE3A	TOIE3	116
(0x70)	TIMSK2	-	-	-	-	-	OCIE2B	OCIE2A	TOIE2	137
(0x6F)	TIMSK1	-	-	ICIE1	-	-	OCIE1B	OCIE1A	TOIE1	115
(0x6E)	TIMSK0	-	-	-	-	-	OCIE0B	OCIE0A	TOIE0	89
(0x6D)	PCMSK2	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	56
(0x6C)	PCMSK1	PCINT15	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	56
(0x6B)	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	56
(0x6A)	Reserved	-	-	-	-	-	-	-	-	
(0x69)	EICRA	-	-	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	53
(0x68)	PCICR	-	-	-	-	PCIE3	PCIE2	PCIE1	PCIE0	54
(0x67)	Reserved	-	-	-	-	-	-	-	-	
(0x66)	OSCCAL	Oscillator calibration register								32
(0x65)	Reserved	-	-	-	-	-	-	-	-	
(0x64)	PRR	PRTWI	PRTIM2	PRTIM0	PRUSART1	PRTIM1	PRSPI	PRUSART0	PRADC	39
(0x63)	Reserved	-	-	-	-	-	-	-	-	
(0x62)	Reserved	-	-	-	-	-	-	-	-	
(0x61)	CLKPR	CLKPCE	-	-	-	CLKPS3	CLKPS2	CLKPS1	CLKPS0	32
(0x60)	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	47
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	10
0x3E (0x5E)	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	10
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	10
0x3C (0x5C)	Reserved	-	-	-	-	-	-	-	-	
0x3B (0x5B)	Reserved	-	-	-	-	-	-	-	-	
0x3A (0x5A)	Reserved	-	-	-	-	-	-	-	-	
0x39 (0x59)	Reserved	-	-	-	-	-	-	-	-	
0x38 (0x58)	Reserved	-	-	-	-	-	-	-	-	

Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O registers within the address range $00 - $1F 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 the CBI and SBI instructions will operate on all bits in the I/O register, writing a one back into any flag read as set, thus clearing the flag. 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 $00 - $3F must be used. When addressing I/O registers as data space using LD and ST instructions, $20 must be added to these addresses.
   The ATmega164P-B/324P-B/644P-B 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 $60 - $FF, only the ST/STS/STD and LD/LDS/LDD instructions can be used.
5. USART in SPI master mode.

30. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x37 (0x57)	SPMCSR	SPMIE	RWWSB	SIGRD	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	253
0x36 (0x56)	Reserved	-	-	-	-	-	-	-	-	
0x35 (0x55)	MCUCR	JTD	BODS	BODSE	PUD	-	-	IVSEL	IVCE	72/239
0x34 (0x54)	MCUSR	-	-	-	JTRF	WDRF	BORF	EXTRF	PORF	47/239
0x33 (0x53)	SMCR	-	-	-	-	SM2	SM1	SM0	SE	38
0x32 (0x52)	Reserved	-	-	-	-	-	-	-	-	
0x31 (0x51)	OCDR	On-chip debug register								230
0x30 (0x50)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	222
0x2F (0x4F)	Reserved	-	-	-	-	-	-	-	-	
0x2E (0x4E)	SPDR	SPI 0 data register								146
0x2D (0x4D)	SPSR	SPIF0	WCOL0	-	-	-	-	-	SPI2X0	145
0x2C (0x4C)	SPCR	SPIE0	SPE0	DORD0	MSTR0	CPOL0	CPHA0	SPR01	SPR00	144
0x2B (0x4B)	GPIOR2	General purpose I/O register 2								22
0x2A (0x4A)	GPIOR1	General purpose I/O register 1								22
0x29 (0x49)	Reserved	-	-	-	-	-	-	-	-	
0x28 (0x48)	OCR0B	Timer/Counter0 output compare register B								88
0x27 (0x47)	OCR0A	Timer/Counter0 output compare register A								88
0x26 (0x46)	TCNT0	Timer/Counter0 (8 bit)								88
0x25 (0x45)	TCCR0B	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00	87
0x24 (0x44)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00	88

Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O registers within the address range $00 - $1F 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 the CBI and SBI instructions will operate on all bits in the I/O register, writing a one back into any flag read as set, thus clearing the flag. 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 $00 - $3F must be used. When addressing I/O registers as data space using LD and ST instructions, $20 must be added to these addresses.
   The ATmega164P-B/324P-B/644P-B 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 $60 - $FF, only the ST/STS/STD and LD/LDS/LDD instructions can be used.
5. USART in SPI master mode.

30. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x23 (0x43)	GTCCR	TSM	-	-	-	-	-	PSRASY	PSRSYNC	138
0x22 (0x42)	EEARH	-	-	-	-	EEPROM address register high byte				19
0x21 (0x41)	EEARL	EEPROM address register low byte								19
0x20 (0x40)	EEDR	EEPROM data register								19
0x1F (0x3F)	EECR	-	-	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	19
0x1E (0x3E)	GPIOR0	General purpose I/O register 0								22
0x1D (0x3D)	EIMSK	-	-	-	-	-	INT2	INT1	INT0	54
0x1C (0x3C)	EIFR	-	-	-	-	-	INTF2	INTF1	INTF0	54
0x1B (0x3B)	PCIFR	-	-	-	-	PCIF3	PCIF2	PCIF1	PCIF0	55
0x1A (0x3A)	Reserved	-	-	-	-	-	-	-	-	
0x19 (0x39)	Reserved	-	-	-	-	-	-	-	-	
0x18 (0x38)	TIFR3	-	-	ICF3	-	-	OCF3B	OCF3A	TOV3	118
0x17 (0x37)	TIFR2	-	-	-	-	-	OCF2B	OCF2A	TOV2	138
0x16 (0x36)	TIFR1	-	-	ICF1	-	-	OCF1B	OCF1A	TOV1	117
0x15 (0x35)	TIFR0	-	-	-	-	-	OCF0B	OCF0A	TOV0	89
0x14 (0x34)	Reserved	-	-	-	-	-	-	-	-	
0x13 (0x33)	Reserved	-	-	-	-	-	-	-	-	
0x12 (0x32)	Reserved	-	-	-	-	-	-	-	-	
0x11 (0x31)	Reserved	-	-	-	-	-	-	-	-	
0x10 (0x30)	Reserved	-	-	-	-	-	-	-	-	

Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O registers within the address range $00 - $1F 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 the CBI and SBI instructions will operate on all bits in the I/O register, writing a one back into any flag read as set, thus clearing the flag. 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 $00 - $3F must be used. When addressing I/O registers as data space using LD and ST instructions, $20 must be added to these addresses.
   The ATmega164P-B/324P-B/644P-B 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 $60 - $FF, only the ST/STS/STD and LD/LDS/LDD instructions can be used.
5. USART in SPI master mode.

30. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x0F (0x2F)	Reserved	-	-	-	-	-	-	-	-	
0x0E (0x2E)	Reserved	-	-	-	-	-	-	-	-	
0x0D (0x2D)	Reserved	-	-	-	-	-	-	-	-	
0x0C (0x2C)	Reserved	-	-	-	-	-	-	-	-	
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	74
0x06 (0x26)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	74
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

Notes:

1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O registers within the address range $00 - $1F 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 the CBI and SBI instructions will operate on all bits in the I/O register, writing a one back into any flag read as set, thus clearing the flag. 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 $00 - $3F must be used. When addressing I/O registers as data space using LD and ST instructions, $20 must be added to these addresses.
   The ATmega164P-B/324P-B/644P-B 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 $60 - $FF, only the ST/STS/STD and LD/LDS/LDD instructions can be used.
5. USART in SPI master mode.

31. 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	4
ICALL		Indirect call to (Z)	PC ← Z	None	4
CALL	k	Direct subroutine call	PC ← k	None	5
RET		Subroutine return	PC ← STACK	None	5
RETI		Interrupt return	PC ← STACK	I	5
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

31. Instruction Set Summary (Continued)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
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
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

31. Instruction Set Summary (Continued)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
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

31. Instruction Set Summary (Continued)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
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
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

32. Ordering Information

Speed (MHz)[3]	Power Supply	Ordering Code[2]	Package[1]	Operational Range
16	2.7 – 5.5V	ATmega164P-B15AZ ATmega164P-B15MZ ATmega324P-B15AZ ATmega324P-B15MZ ATmega644P-B15AZ ATmega644P-B15MZ	ML PW ML PW ML PW	Automotive (–40°C to 125°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, complies to the European directive for restriction of hazardous substances (RoHS directive). Also halide free and fully green.
3. For speed versus V_{CC} see Section 28.3 “Speed Grades” on page 289.

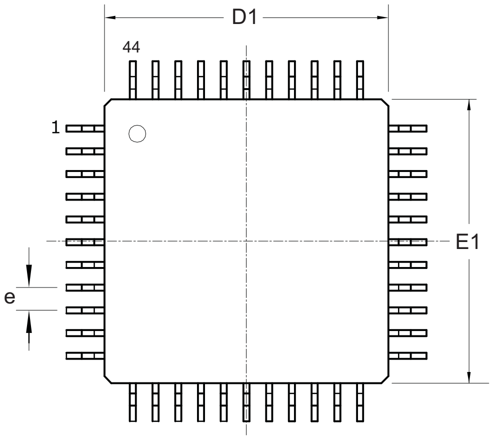
33. Packaging Information

Table 33-1. Package Types

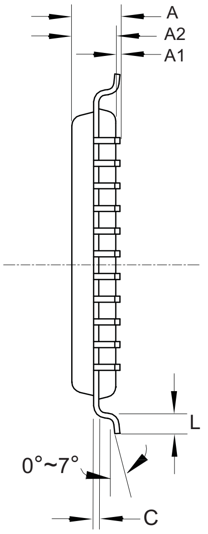
Package Type	
ML	44-lead, Thin (1.0mm) plastic gull wing quad flat package (TQFP)
PW	44-pad, 7 x 7 x 1.0mm body, lead pitch 0.50mm, quad flat no-lead/micro lead frame package (QFN/MLF)

Figure 33-1. ML

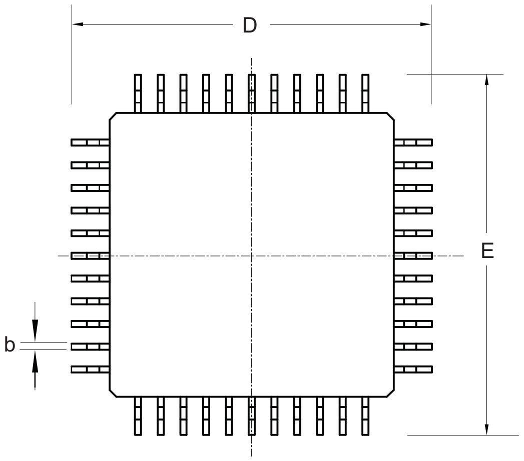
Drawings not scaled



Top View



Side View



Bottom View

COMMON DIMENSIONS
(Unit of Measure = mm)

Symbol	MIN	NOM	MAX	NOTE
A	––––	––––	1.20	
A1	0.05	––––	0.15	
A2	0.95	1.00	1.05	
D/E	11.75	12.00	12.25	
D1/E1	9.90	10.00	10.10	2
C	0.09	––––	0.20	
L	0.45	––––	0.75	
b	0.30	––––	0.45	
e	0.80 TYP.			
n	44			

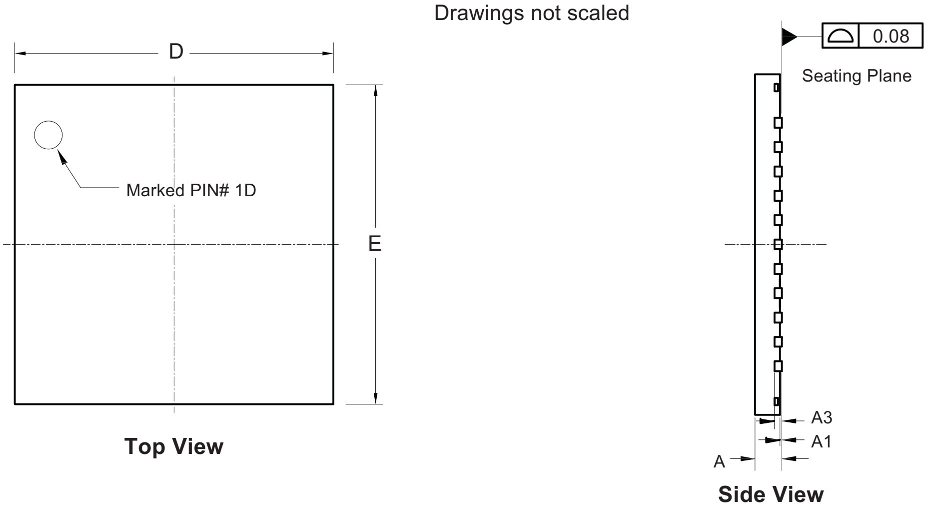
Notes:
1. This drawing is for general information only. Refer to JEDEC Drawing MS-026, Variation ACB.
2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25mm per side. Dimensions D1 and E1 are maximum plastic body size dimensions including mold mismatch.
3. Lead coplanarity is 0.10mm maximum.

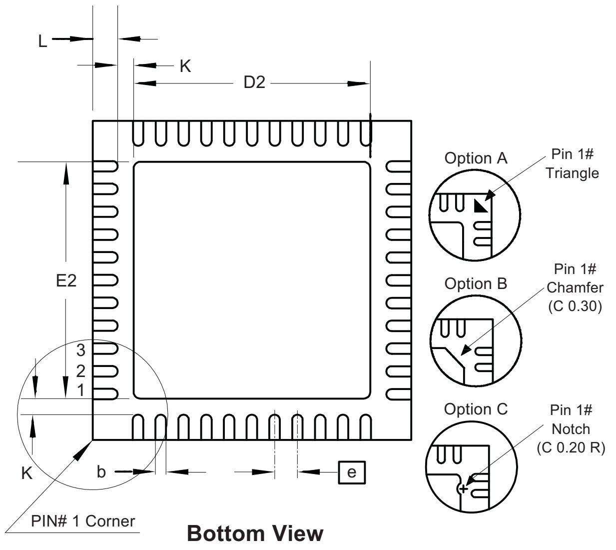
02/28/12

	TITLE	GPC	DRAWING NO.	REV.
Atmel **Package Drawing Contact:** packagedrawings@atmel.com	ML, 44 Lds - 0.80mm Pitch, 10x10x1.00mm Body size Thin Profile Plastic Quad Flat Package (TQFP)	AIX	ML	I

Figure 33-2. PW

Drawings not scaled





COMMON DIMENSIONS
(Unit of Measure = mm)

Symbol	MIN	NOM	MAX	NOTE
A	0.80	––––	1.00	
A1	––––	0.02	0.05	
A3	0.20 REF			
D/E	6.90	7.00	7.10	
D2/E2	5.00	5.20	5.40	
L	0.45	0.55	0.65	
K	––––	0.35	––––	
b	0.18	0.23	0.30	2
e	0.50 BSC			
n	44			

Notes:
1. This drawing is for general information only. Refer to JEDEC Drawing MO-220, Variation VKKD-1, for proper dimensions, tolerances, datums, etc.
2. Dimensions b applies to metallized terminal and is measured between 0.15mm and 0.30mm from the terminal tip. If the terminal has the optical radius on the other end of the terminal, the dimensions should not be measured in that radius area.

02/17/12

Atmel **Package Drawing Contact:** packagedrawings@atmel.com	TITLE PW, 44 Leads - 0.50mm Pitch, 7x7x1mm Body size Very Thin Quad Flat Package (Punched) (VQFN) Sawn	GPC ZCP	DRAWING NO. PW	REV. H

34. Errata

34.1 Errata for ATmega164P-B

34.1.1 Rev. E

No known errata.

34.2 Errata for ATmega324P-B

34.2.1 Rev. F

No known errata.

34.3 Errata for ATmega644P-B

34.3.1 Rev. G

No known errata.

35. Revision History

Please note that the following page numbers referred to in this section refer to the specific revision mentioned, not to this document.

Revision No.	History
9255E-AVR-08/14	• Put datasheet in the latest template

36. Table of Contents

Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**

© 2014 Atmel Corporation. / Rev.: 9255E–AVR–08/14

Atmel®, Atmel logo and combinations thereof, Enabling Unlimited Possibilities®, AVR®, AKS®, QTouch®, and others are registered trademarks or trademarks of Atmel Corporation in U.S. and other countries. Windows® is a registered trademark of Microsoft Corporation in U.S. and or other countries. Other terms and product names may be trademarks of others.

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 THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, 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 AND 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 products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER: Atmel products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death (“Safety-Critical Applications”) without an Atmel officer's specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Atmel products are not designed nor intended for use in military or aerospace applications or environments unless specifically designated by Atmel as military-grade. Atmel products are not designed nor intended for use in automotive applications unless specifically designated by Atmel as automotive-grade.