Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 125 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
- Non-volatile Program and Data Memories
  - 8K/16K/32K Bytes of In-System Self-Programmable Flash
  - 512/512/1024 EEPROM
  - 512/512/1024 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 hardware-activated after reset
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- USB 2.0 Full-speed Device Module with Interrupt on Transfer Completion
  - Complies fully with Universal Serial Bus Specification REV 2.0
  - 48 MHz PLL for Full-speed Bus Operation : data transfer rates at 12 Mbit/s
  - Fully independant 176 bytes USB DPRAM for endpoint memory allocation
  - Endpoint 0 for Control Transfers: from 8 up to 64-bytes
  - 4 Programmable Endpoints:
    - IN or Out Directions
    - Bulk, Interrupt and IsochronousTransfers
    - Programmable maximum packet size from 8 to 64 bytes
    - Programmable single or double buffer
  - Suspend/Resume Interrupts
  - Microcontroller reset on USB Bus Reset without detach
  - USB Bus Disconnection on Microcontroller Request
- Peripheral Features
  - One 8-bit Timer/Counters with Separate Prescaler and Compare Mode (two 8-bit PWM channels)
  - One 16-bit Timer/Counter with Separate Prescaler, Compare and Capture Mode (three 8-bit PWM channels)
  - USART with SPI master only mode and hardware flow control (RTS/CTS)
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- On Chip Debug Interface (debugWIRE)
- Special Microcontroller Features
  - Power-On Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Five Sleep Modes: Idle, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 22 Programmable I/O Lines
  - QFN32 (5x5mm) / TQFP32 packages
- Operating Voltages
  - 2.7 - 5.5V
- Operating temperature
  - Industrial (-40°C to +85°C)
- Maximum Frequency
  - 8 MHz at 2.7V - Industrial range
  - 16 MHz at 4.5V - Industrial range

Note: 1. See “Data Retention” on page 6 for details.

8-bit AVR® Microcontroller with 8/16/32K Bytes of ISP Flash and USB Controller

ATmega8U2

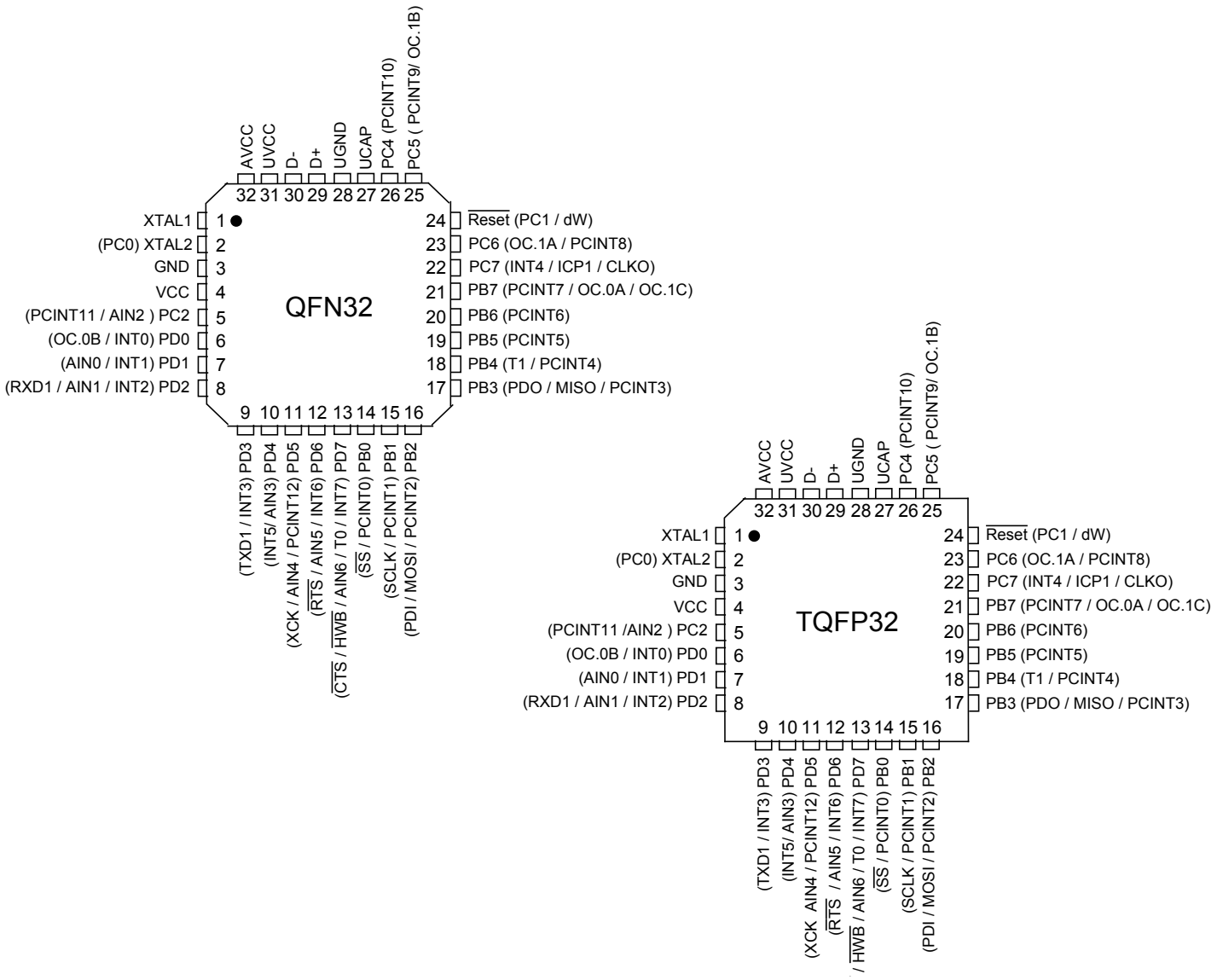
ATmega16U2

ATmega32U2

7799E–AVR–09/2012

1. Pin Configurations

Figure 1-1. Pinout



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

1.1 Disclaimer

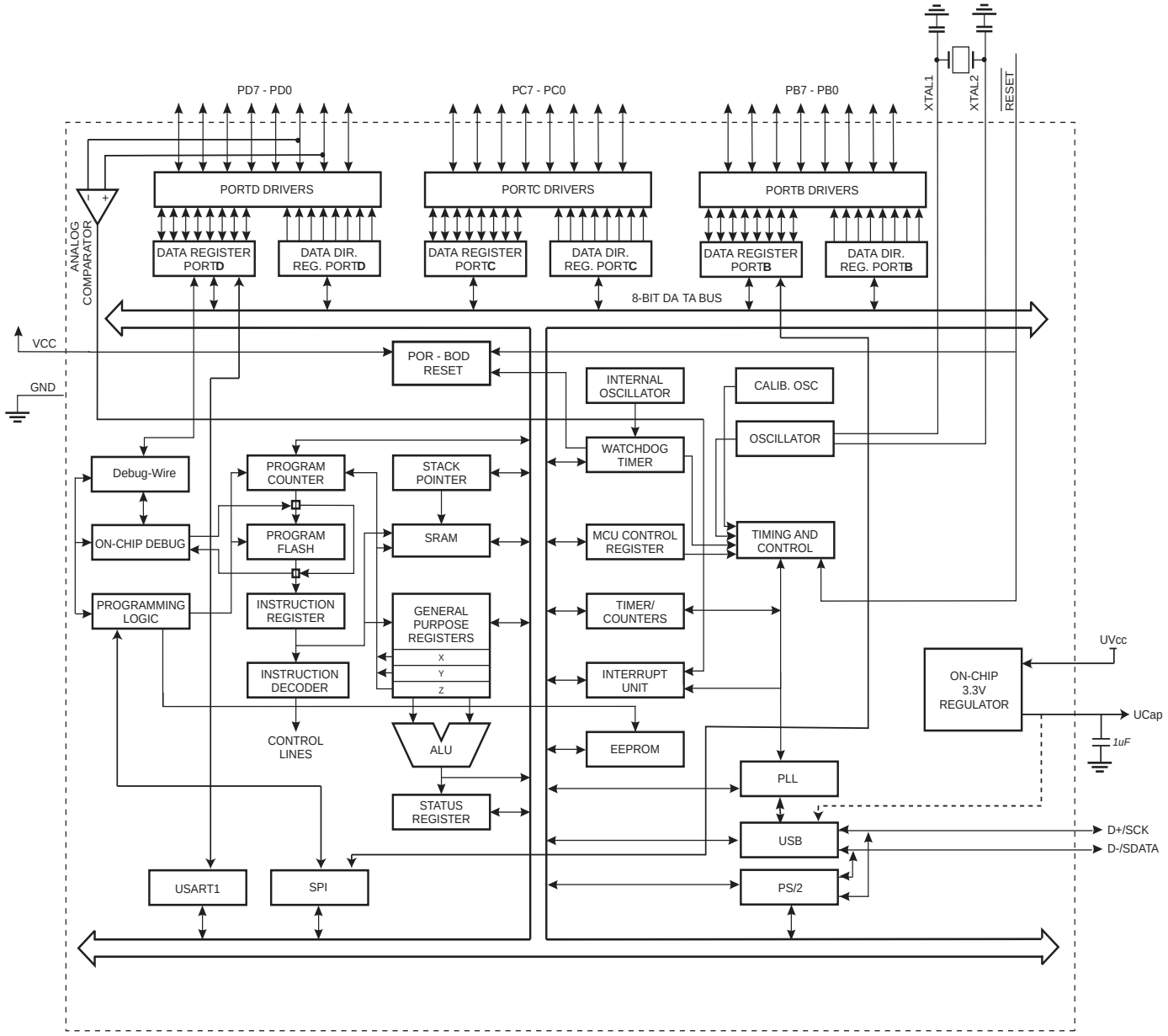
Typical values contained in this datasheet are based on simulations and characterization of other AVR microcontrollers manufactured on the same process technology. Min and Max values will be available after the device is characterized.

2. Overview

The ATmega8U2/16U2/32U2 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 ATmega8U2/16U2/32U2 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 ATmega8U2/16U2/32U2 provides the following features: 8K/16K/32K Bytes of In-System Programmable Flash with Read-While-Write capabilities, 512/512/1024 Bytes EEPROM, 512/512/1024 SRAM, 22 general purpose I/O lines, 32 general purpose working registers, two flexible Timer/Counters with compare modes and PWM, one USART, a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, debugWIRE interface, also used for accessing the On-chip Debug system and programming and five software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or Hardware Reset. In 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, the main Oscillator continues to run.

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 ATmega8U2/16U2/32U2 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega8U2/16U2/32U2 are 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 Pin Descriptions

2.2.1 VCC

Digital supply voltage.

2.2.2 GND

Ground.

2.2.3 AVCC

AVCC is the supply voltage pin (input) for all analog features (Analog Comparator, PLL). It should be externally connected to VCC through a low-pass filter.

2.2.4 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 ATmega8U2/16U2/32U2 as listed on-page 74.

2.2.5 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 various special features of the ATmega8U2/16U2/32U2 as listed on page 77.

2.2.6 Port D (PD7..PD0)

Port D serves as analog inputs to the analog comparator.

Port D also serves as an 8-bit bi-directional I/O port, if the analog comparator is not used (concerns PD2/PD1 pins). Port pins can provide 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.

2.2.7 D-

USB Full Speed Negative Data Upstream Port

2.2.8 D+

USB Full Speed Positive Data Upstream Port

2.2.9 UGND

USB Ground.

2.2.10 UVCC

USB Pads Internal Regulator Input supply voltage.

2.2.11 UCAP

USB Pads Internal Regulator Output supply voltage. Should be connected to an external capacitor (1µF).

2.2.12 $\overline{\text{RESET}}$/PC1/dW

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 "System Control and Reset" on page 47. Shorter pulses are not guaranteed to generate a reset. This pin alternatively serves as debugWire channel or as generic I/O. The configuration depends on the fuses RSTDISBL and DWEN.

2.2.13 XTAL1

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

2.2.14 XTAL2/PC0

Output from the inverting Oscillator amplifier if enabled by Fuse. Also serves as a generic I/O.

3. Resources

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

4. 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.

These 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".

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.

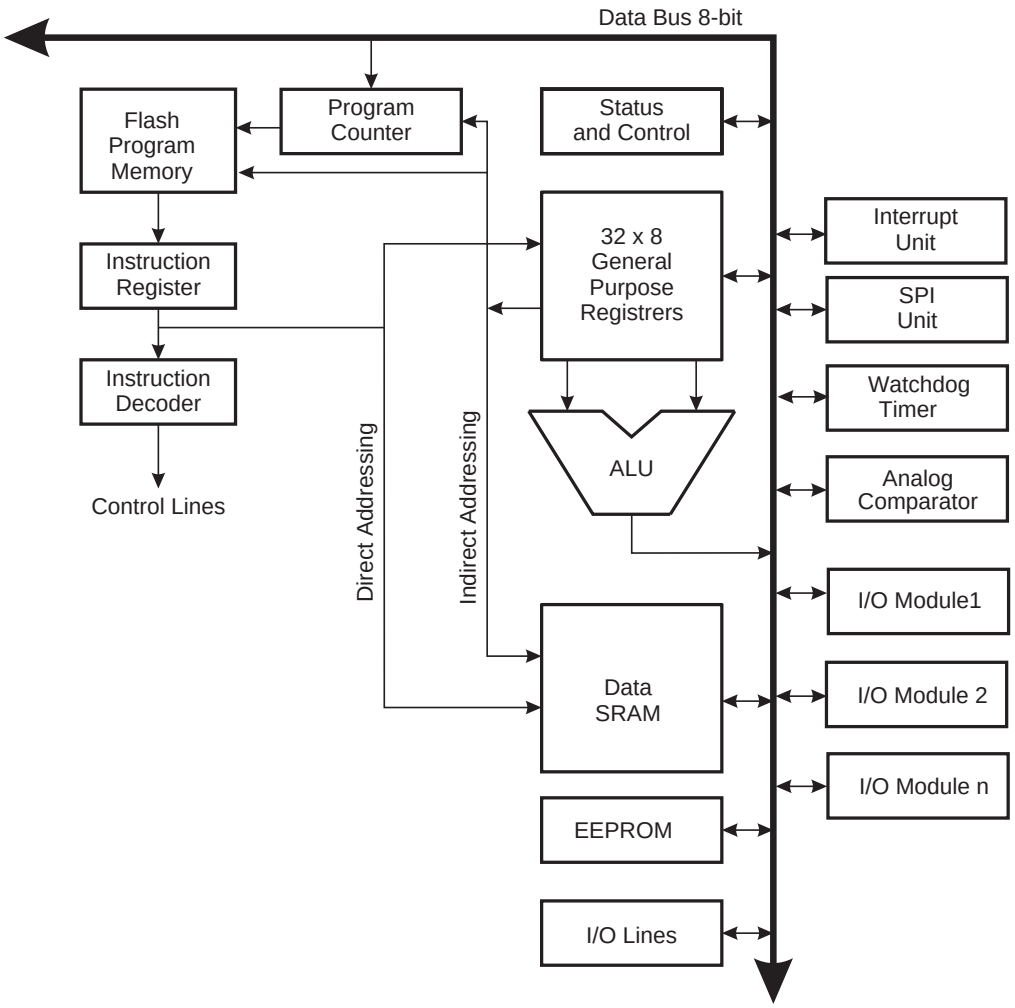
6. AVR CPU Core

6.1 Introduction

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

6.2 Architectural Overview

Figure 6-1. 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 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 ATmega8U2/16U2/32U2 has Extended I/O space from 0x60 - 0xFF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

6.3 ALU – Arithmetic Logic Unit

The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. See the “Instruction Set” section for a detailed description.

6.4 Status Register

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform

conditional operations. Note that the Status Register is updated after all ALU operations, as specified in the Instruction Set Reference. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

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

6.4.1 SREG – Status Register

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.

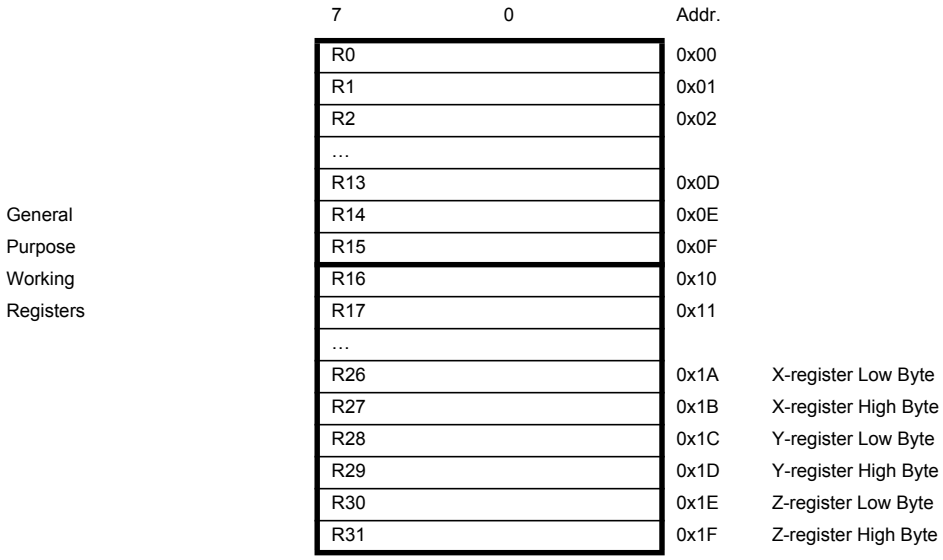
6.5 General Purpose Register File

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

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

Figure 6-2 shows the structure of the 32 general purpose working registers in the CPU.

Figure 6-2. AVR CPU General Purpose Working Registers



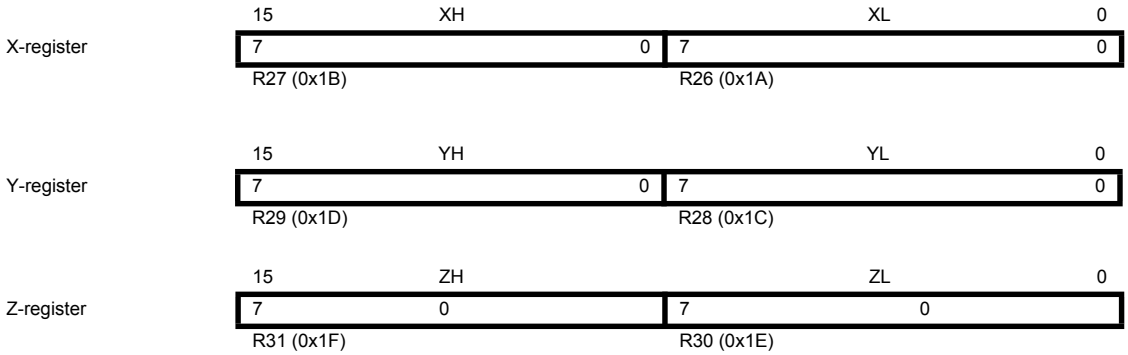
Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions.

As shown in Figure 6-2, 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.

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

The registers R26..R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the data space. The three indirect address registers X, Y, and Z are defined as described in Figure 6-3.

Figure 6-3. 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).

6.6 Stack Pointer

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. 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 7-2 on page 18.

See Table 6-1 for Stack Pointer details.

Table 6-1. 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. 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.

6.6.1 SPH and SPL – Stack Pointer High and Low Register

Bit	15	14	13	12	11	10	9	8	
0x3E (0x5E)	SP15	SP14	SP13	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/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	
	1	1	1	1	1	1	1	1	

6.7 Instruction Execution Timing

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

Figure 6-4 shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast-access Register File concept. This is the basic pipelining concept to obtain up to 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

Figure 6-4. The Parallel Instruction Fetches and Instruction Executions

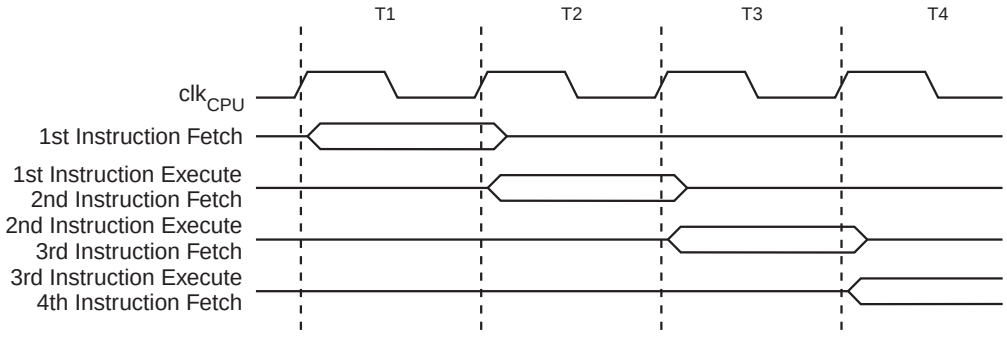
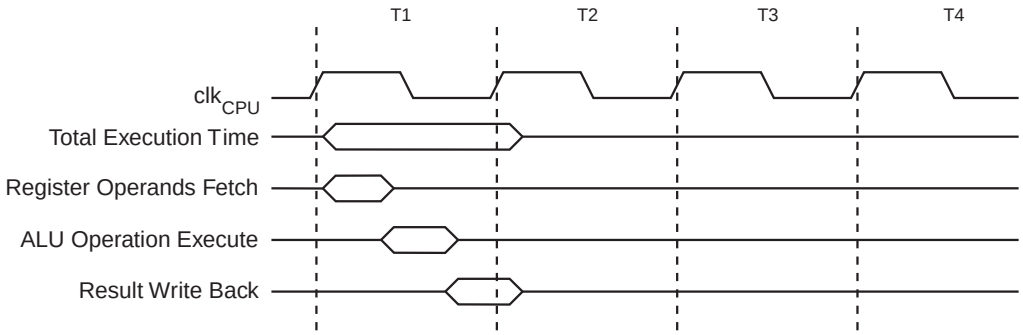


Figure 6-5 shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

Figure 6-5. Single Cycle ALU Operation



6.8 Reset and Interrupt Handling

The AVR provides several different interrupt sources. These interrupts and the separate Reset Vector each have a separate program vector in the program memory space. All interrupts are assigned individual enable bits which must be written logic one together with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt. Depending on the Program Counter value, interrupts may be automatically disabled when Boot Lock bits BLB02 or BLB12 are programmed. This feature improves software security. See the section “Memory Programming” on page 246 for details.

The lowest addresses in the program memory space are by default defined as the Reset and Interrupt Vectors. The complete list of vectors is shown in “Interrupts” on page 64. The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0 – the External Interrupt Request 0. The Interrupt Vectors can be moved to the start of the Boot Flash section by setting the IVSEL bit in the MCU Control Register (MCUCR). Refer to “Interrupts” on page 64 for more information. The Reset Vector can also be moved to the start of the Boot Flash section by programming the BOOTRST Fuse, see “Memory Programming” on page 246.

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) */
```

6.8.1 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.

7. AVR Memories

This section describes the different memories in the ATmega8U2/16U2/32U2. The AVR architecture has two main memory spaces, the Data Memory and the Program Memory space. In addition, the ATmega8U2/16U2/32U2 features an EEPROM Memory for data storage. All three memory spaces are linear and regular.

7.1 In-System Reprogrammable Flash Program Memory

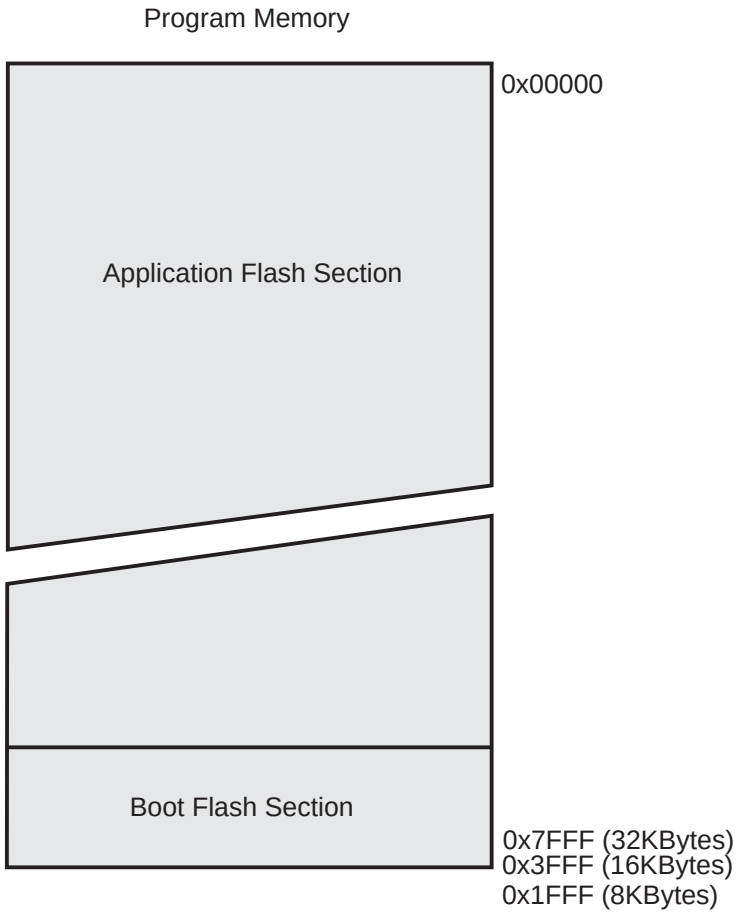
The ATmega8U2/16U2/32U2 contains 8K/16K/32K bytes On-chip In-System Reprogrammable Flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 4K x 16, 8K x 16. For software security, the Flash Program memory space is divided into two sections, Boot Program section and Application Program section.

The Flash memory has an endurance of at least 100,000 write/erase cycles. The ATmega8U2/16U2/32U2 Program Counter (PC) is 16 bits wide, thus addressing the 8K/16K/32K program memory locations. The operation of Boot Program section and associated Boot Lock bits for software protection are described in detail in “Memory Programming” on page 246. “Memory Programming” on page 246 contains a detailed description on Flash data serial downloading using the SPI pins or the debugWIRE interface.

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

Timing diagrams for instruction fetch and execution are presented in “Instruction Execution Timing” on page 12.

Figure 7-1. Program Memory Map



7.2 SRAM Data Memory

Figure 7-2 shows how the ATmega8U2/16U2/32U2 SRAM Memory is organized.

The ATmega8U2/16U2/32U2 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 - $0FF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

The first 768 Data Memory locations address 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 512 locations of 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.

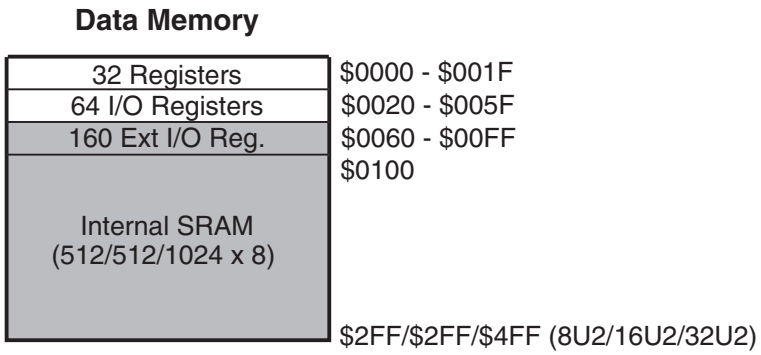
The direct addressing reaches the entire data space.

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, and the 512/512/1024bytes of internal data SRAM in the ATmega8U2/16U2/32U2 are all accessible through all these addressing modes. The Register File is described in “General Purpose Register File” on page 10.

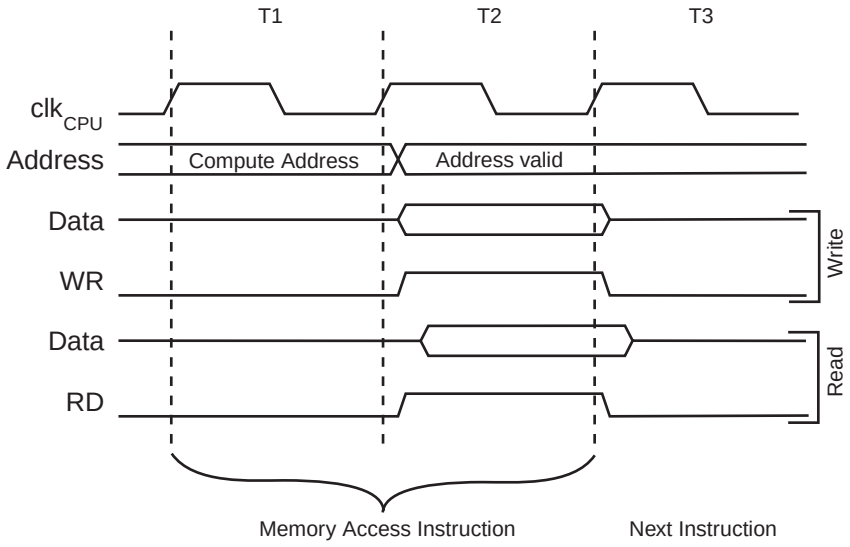
Figure 7-2. Data Memory Map



7.2.1 Data Memory Access Times

This section describes the general access timing concepts for internal memory access. The internal data SRAM access is performed in two clk_{CPU} cycles as described in Figure 7-3.

Figure 7-3. On-chip Data SRAM Access Cycles



7.3 EEPROM Data Memory

The ATmega8U2/16U2/32U2 contains 512/512/1024 bytes of data EEPROM memory. It is organized as a separate data space, in which single bytes can be read and written. The EEPROM has an endurance of at least 100,000 write/erase cycles. The access between the EEPROM and the CPU is described in the following, specifying the EEPROM Address Registers, the EEPROM Data Register, and the EEPROM Control Register.

For a detailed description of SPI, debugWIRE and Parallel data downloading to the EEPROM, see page 259, page 244, and page 250 respectively.

7.3.1 EEPROM Read/Write Access

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

The write access time for the EEPROM is given in Table 7-2 on page 22. A self-timing function, however, lets the user software detect when the next byte can be written. If the user code contains instructions that write the EEPROM, some precautions must be taken. In heavily filtered power supplies, V_{CC} is likely to rise or fall slowly on power-up/down. This causes the device for some period of time to run at a voltage lower than specified as minimum for the clock frequency used. See “Preventing EEPROM Corruption” on page 19. 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.

7.3.2 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.

7.4 I/O Memory

The I/O space definition of the ATmega8U2/16U2/32U2 is shown in “Register Summary” on page 288.

All ATmega8U2/16U2/32U2 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 ATmega8U2/16U2/32U2 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 - 0x1FF 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.

7.4.1 General Purpose I/O Registers

The ATmega8U2/16U2/32U2 contains three General Purpose I/O Registers. These registers can be used for storing any information, and they are particularly useful for storing global variables and Status Flags. General Purpose I/O Registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI, CBI, SBIS, and SBIC instructions.

7.5 Register Description

7.5.1 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 – Res: Reserved Bits**

These bits are reserved and will always read as zero.

- **Bits 11:0 – EEAR[8:0]: EEPROM Address**

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

7.5.2 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 – EEDR[7:0]: 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.

7.5.3 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 – Res: Reserved Bits**

These bits are reserved bits 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 7-1. While EEPE is set, any write to EEPMn will be ignored. During reset, the EEPMn bits will be reset to 0b00 unless the EEPROM is busy programming.

Table 7-1. EEPROM Mode Bits

EEPM1	EEPM0	Programming Time	Operation
0	0	3.4 ms	Erase and Write in one operation (Atomic Operation)
0	1	1.8 ms	Erase Only
1	0	1.8 ms	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 SELFPRGEN in SPMCSR becomes zero.
3. Write new EEPROM address to EEAR (optional).
4. Write new EEPROM data to EEDR (optional).
5. Write a logical one to the EEMPE bit while writing a zero to EEPE in EECR.
6. Within four clock cycles after setting EEMPE, write a logical one to EEPE.

The EEPROM can not be programmed during a CPU write to the Flash memory. The software must check that the Flash programming is completed before initiating a new EEPROM write. 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 "Memory Programming" on page 246 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 7-2 lists the typical programming time for EEPROM access from the CPU.

Table 7-2. EEPROM Programming Time

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

The following code examples show one assembly and one C function for writing to the EEPROM. The examples assume that interrupts are controlled (e.g. by disabling interrupts globally) so that no interrupts will occur during execution of these functions. The examples also assume that no Flash Boot Loader is present in the software. If such code is present, the EEPROM write function must also wait for any ongoing SPM command to finish.

Assembly Code Example[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 “Code Examples” on page 6.

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 “Code Examples” on page 6.

7.5.4 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	

7.5.5 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	

7.5.6 GPIOR0 – General Purpose I/O Register 0

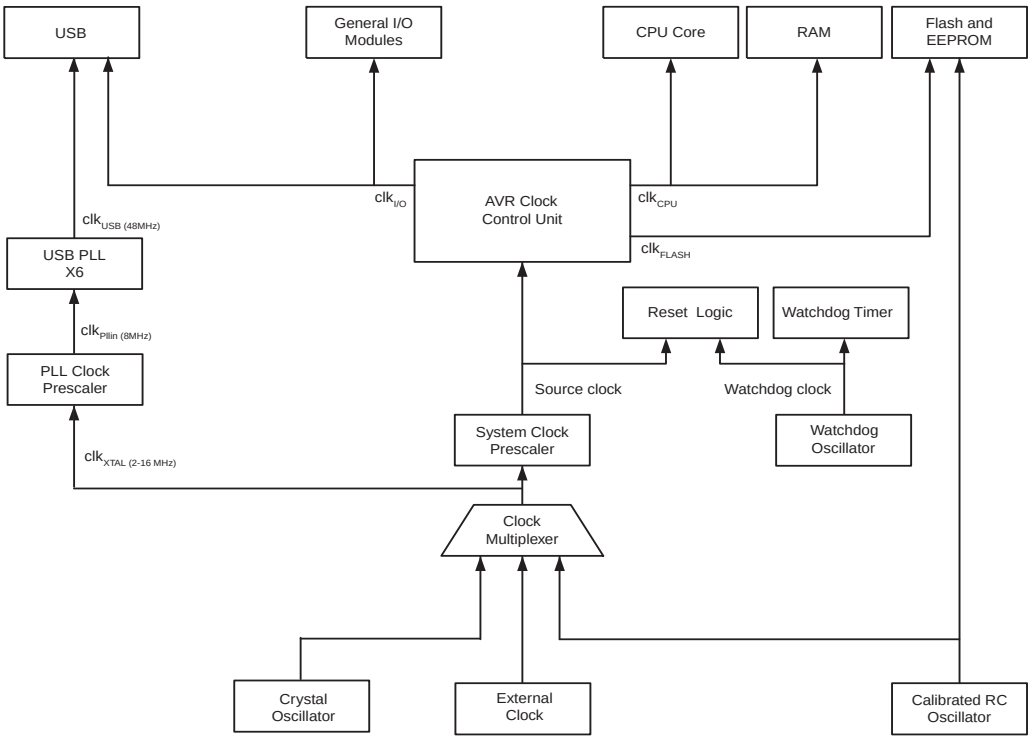
Bit	7	6	5	4	3	2	1	0	
0x1E (0x3E)	MSB							LSB	GPIOR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

8. System Clock and Clock Options

8.1 Clock Systems and their Distribution

Figure 8-1 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 "Power Management and Sleep Modes" on page 42. The clock systems are detailed below.

Figure 8-1. Clock Distribution



8.1.1 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.

8.1.2 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.

8.1.3 Flash Clock – clk_{FLASH}

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

8.1.4 USB Clock – clk_{USB}

The USB is provided with a dedicated clock domain. This clock is generated with an on-chip PLL running at 48 MHz. The PLL always multiply its input frequency by 6. Thus the PLL clock register should be programmed by software to generate a 8 MHz clock on the PLL input.

8.2 Clock Switch

In the ATmega8U2/16U2/32U2 product, the Clock Multiplexer and the System Clock Prescaler can be modified by software.

8.2.1 Exemple of use

The modification can occur when the device enters in USB Suspend mode. It then switches from External Clock to Calibrated RC Oscillator in order to reduce consumption. In such a configuration, the External Clock is disabled.

The firmware can use the watchdog timer to be woken-up from power-down in order to check if there is an event on the application.

If an event occurs on the application or if the USB controller signals a non-idle state on the USB line (Resume for example), the firmware switches the Clock Multiplexer from the Calibrated RC Oscillator to the External Clock.

Figure 8-2. Example of clock switching with wake-up from USB Host

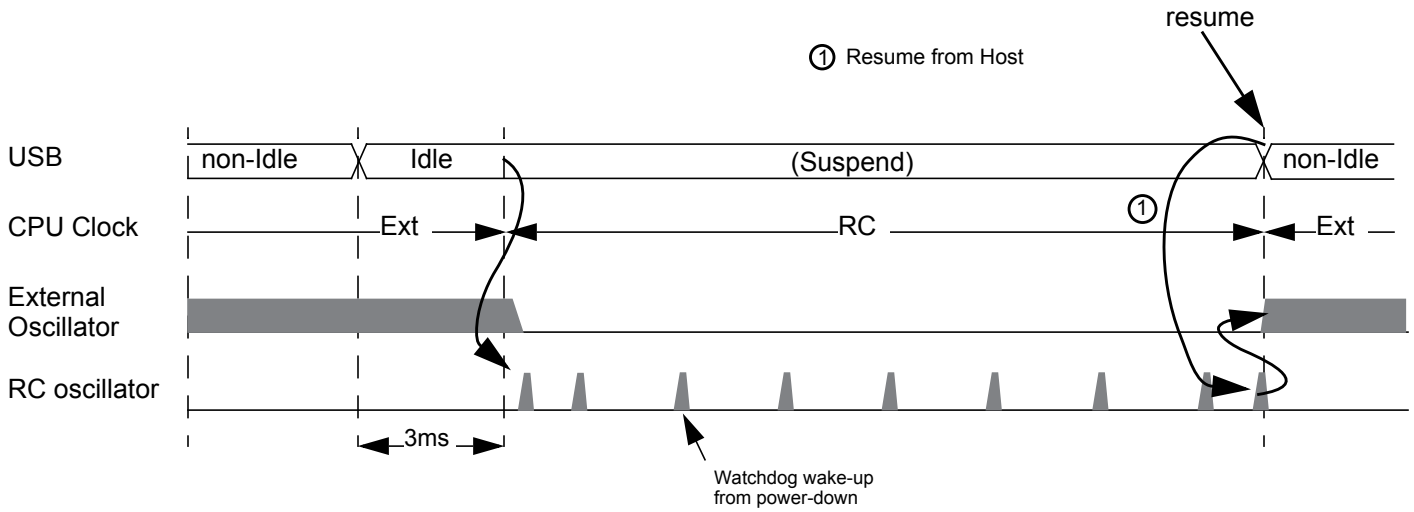
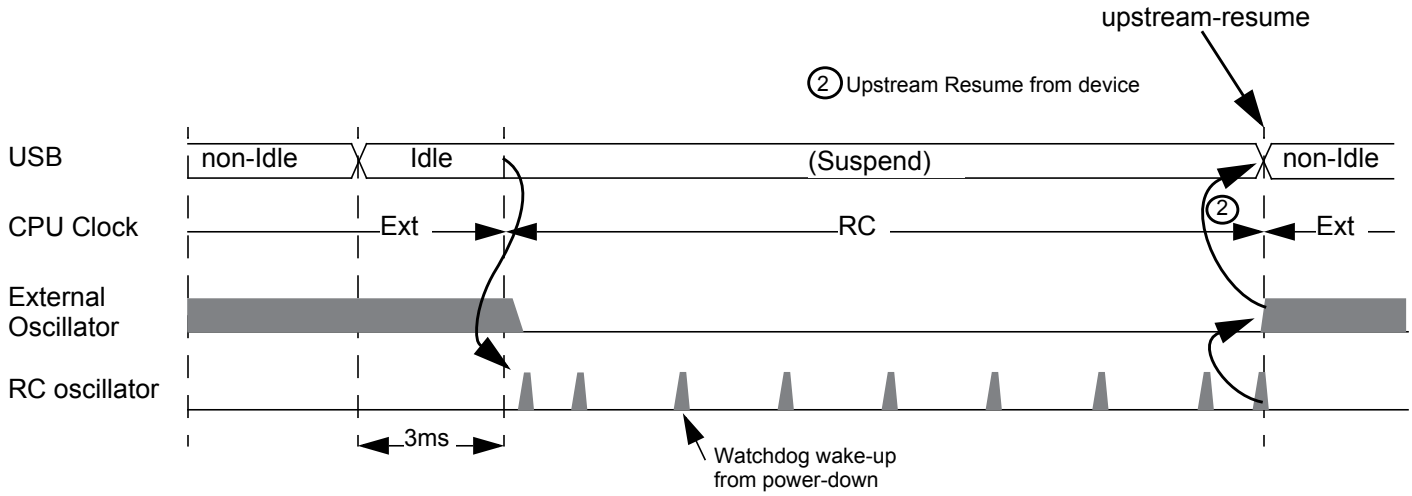


Figure 8-3. Example of clock switching with wake-up from Device



8.2.2 Clock switch Algorythm

8.2.2.1 Swith from external clock to RC clock

```
if (Usb_suspend_detected())         // if (UDINT.SUSPI == 1)
{
   Usb_ack_suspend();               // UDINT.SUSPI = 0;
   Usb_freeze_clock();              // USBCON.FRZCLK = 1;
   Disable_pll();                   // PLLCSR.PLLE = 0;
   Enable_RC_clock();               // CLKSEL0.RCE = 1;
   while (!RC_clock_ready());       // while (CLKSTA.RCON != 1);
   Select_RC_clock();               // CLKSEL0.CLKS = 0;
   Disable_external_clock();        // CLKSEL0.EXTE = 0;
}
```

8.2.2.2 Switch from RC clock to external clock

```
if (Usb_wake_up_detected())         // if (UDINT.WAKEUPI == 1)
{
   Usb_ack_wake_up();               // UDINT.WAKEUPI = 0;
   Enable_external_clock();         // CKSEL0.EXTE = 1;
   while (!External_clock_ready()); // while (CLKSTA.EXTON != 1);
   Select_external_clock();         // CLKSEL0.CLKS = 1;
   Enable_pll();                    // PLLCSR.PLLE = 1;
   Disable_RC_clock();              // CLKSEL0.RCE = 0;
   while (!Pll_ready());            // while (PLLCSR.PLOCK != 1);
   Usb_unfreeze_clock();            // USBCON.FRZCLK = 0;
}
```

8.3 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 8-1. Device Clocking Options Select(1)

Device Clocking Option	CKSEL3:0
Low Power Crystal Oscillator	1111 - 1000
Full Swing Crystal Oscillator	0111 - 0110
Reserved	0101 - 0100
Reserved	0011
Calibrated Internal RC Oscillator	0010
External Clock	0000
Reserved	0001

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

8.3.1 Default Clock Source

The device is shipped with internal RC oscillator at 8.0 MHz and with the fuse CKDIV8 programmed, resulting in 1.0 MHz 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.

8.3.2 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. “On-chip Debug System” on page 45 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 8-2. The frequency of the Watchdog Oscillator is voltage dependent as shown in “Typical Characteristics” on page 273.

Table 8-2. Number of Watchdog Oscillator Cycles

Typ Time-out (V_{CC} = 5.0V)	Typ Time-out (V_{CC} = 3.0V)	Number of Cycles
0 ms	0 ms	0
4.1 ms	4.3 ms	512
65 ms	69 ms	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 32K cycles 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.

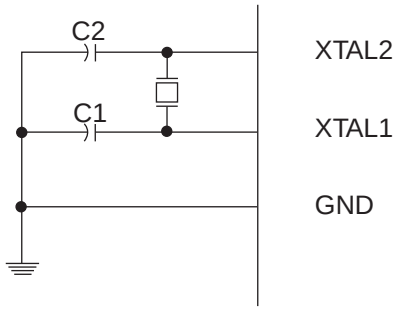
8.4 Low Power Crystal Oscillator

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 8-4. Either a quartz crystal or a ceramic resonator may be used.

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.

C1 and C2 should always be equal for both crystals and resonators. The optimal value of the capacitors depends on the crystal or resonator in use, the amount of stray capacitance, and the electromagnetic noise of the environment. Some initial guidelines for choosing capacitors for use with crystals are given in Table 8-3. For ceramic resonators, the capacitor values given by the manufacturer should be used.

Figure 8-4. Crystal Oscillator Connections



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 8-3.

Table 8-3. Low Power Crystal Oscillator Operating Modes[3]

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

Notes: 1. The frequency ranges are preliminary values. Actual values are TBD.
2. This option should not be used with crystals, only with ceramic resonators.

3. If 8 MHz frequency exceeds the specification of the device (depends on V_{CC}), the CKDIV8 Fuse can be programmed in order to divide the internal frequency by 8. It must be ensured that the resulting divided clock meets the frequency specification of the device.

The CKSEL0 Fuse together with the SUT1..0 Fuses select the start-up times as shown in Table 8-4.

Table 8-4. 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 + 4.1 ms(1)	0	00
Ceramic resonator, slowly rising power	258 CK	14CK + 65 ms(1)	0	01
Ceramic resonator, BOD enabled	1K CK	14CK(2)	0	10
Ceramic resonator, fast rising power	1K CK	14CK + 4.1 ms(2)	0	11
Ceramic resonator, slowly rising power	1K CK	14CK + 65 ms(2)	1	00
Crystal Oscillator, BOD enabled	16K CK	14CK	1	01
Crystal Oscillator, fast rising power	16K CK	14CK + 4.1 ms	1	10
Crystal Oscillator, slowly rising power	16K CK	14CK + 65 ms	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.

Table 8-5. Start-up times for the internal calibrated RC Oscillator clock selection

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

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

8.5 Full Swing Crystal Oscillator

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 8-4. Either a quartz crystal or a ceramic resonator may be used.

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” on page 30. Note that the Full Swing Crystal Oscillator will only operate for V_{CC} = 2.7 - 5.5 volts.

C1 and C2 should always be equal for both crystals and resonators. The optimal value of the capacitors depends on the crystal or resonator in use, the amount of stray capacitance, and the electromagnetic noise of the environment. Some initial guidelines for choosing capacitors for use with crystals are given in Table 1. For ceramic resonators, the capacitor values given by the manufacturer should be used.

Table 1. 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 + 4.1 ms(1)	0	00
Ceramic resonator, slowly rising power	258 CK	14CK + 65 ms(1)	0	01
Ceramic resonator, BOD enabled	1K CK	14CK(2)	0	10
Ceramic resonator, fast rising power	1K CK	14CK + 4.1 ms(2)	0	11
Ceramic resonator, slowly rising power	1K CK	14CK + 65 ms(2)	1	00
Crystal Oscillator, BOD enabled	16K CK	14CK	1	01
Crystal Oscillator, fast rising power	16K CK	14CK + 4.1 ms	1	10
Crystal Oscillator, slowly rising power	16K CK	14CK + 65 ms	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.

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.

8.6 Calibrated Internal RC Oscillator

By default, the Internal RC Oscillator provides an approximate 8 MHz clock. Though voltage and temperature dependent, this clock can be very accurately calibrated by the the user. See Table 26-1 on page 266 for more details. The device is shipped with the CKDIV8 Fuse programmed. See "System Clock Prescaler" on page 35 for more details.

This clock may be selected as the system clock by programming the CKSEL Fuses as shown in Table 8-6. If selected, it will operate with no external components. During reset, hardware loads the pre-programmed calibration value into the OSCCAL Register and thereby automatically calibrates the RC Oscillator. The accuracy of this calibration is shown as Factory calibration in Table 26-1 on page 266.

By changing the OSCCAL register from SW, see "OSCCAL – Oscillator Calibration Register" on page 38, 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 26-1 on page 266.

When this Oscillator is used as the chip clock, the Watchdog Oscillator will still be used for the Watchdog Timer and for the Reset Time-out. For more information on the pre-programmed calibration value, see the section "Calibration Byte" on page 249.

Table 8-6. Internal Calibrated RC Oscillator Operating Modes(3)

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

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

When this Oscillator is selected, start-up times are determined by the SUT Fuses as shown in Table 8-5 on page 31.

Table 8-7. Start-up times for the internal calibrated RC Oscillator clock selection

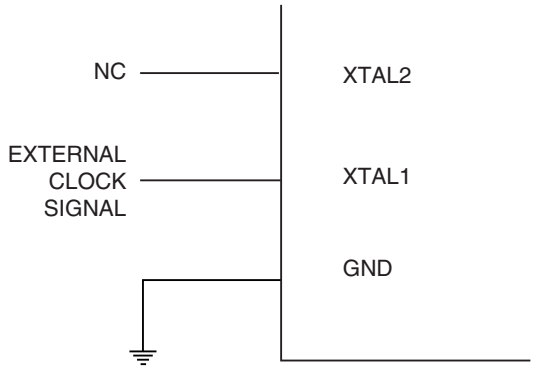
Power Conditions	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	SUT1..0
BOD enabled	6 CK	14 CK	00
Fast rising power	6 CK	14 CK + 4.1 ms	01
Slowly rising power	6 CK	14 CK + 65 ms(1)	10
Reserved			11

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

8.7 External Clock

The device can utilize a external clock source as shown in Figure 8-5. To run the device on an external clock, the CKSEL Fuses must be programmed as shown in Table 8-1.

Figure 8-5. External Clock Drive Configuration



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

Table 8-8. 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 + 4.1 ms	01
Slowly rising power	6 CK	14CK + 65 ms	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 “System Clock Prescaler” on page 35 for details.

8.8 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.

8.9 System Clock Prescaler

The ATmega8U2/16U2/32U2 has a system clock prescaler, and the system clock can be divided by setting the “CLKPR – Clock Prescale Register” on page 39. 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_{CPU}, and clk_{FLASH} are divided by a factor as shown in Table 8-9 on page 40.

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.

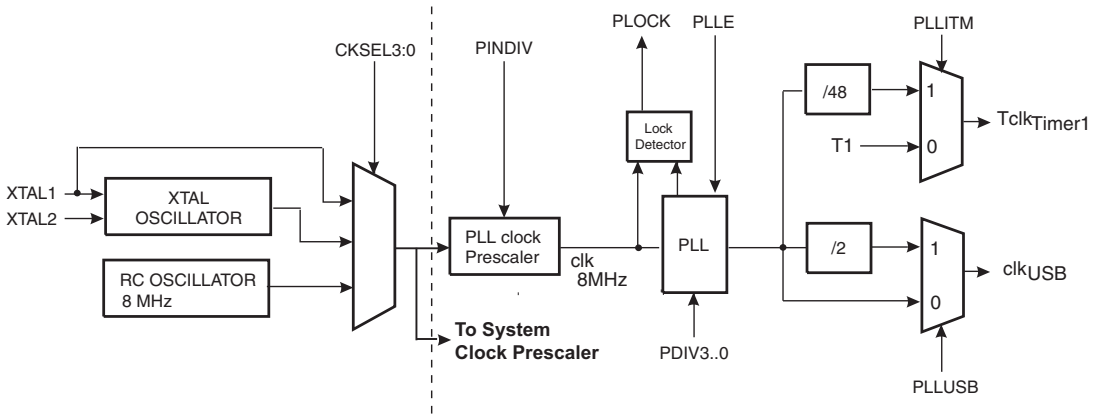
8.10 PLL

The PLL is used to generate internal high frequency (48 MHz) clock for USB interface, the PLL input is generated from an external low-frequency (the crystal oscillator or external clock input pin from XTAL1).

8.10.1 Internal PLL for USB interface

The internal PLL in ATmega8U2/16U2/32U2 generates a clock frequency that is 6x multiplied from nominally 8 MHz input. The source of the 8 MHz PLL input clock is the output of the internal PLL clock prescaler that generates the 8 MHz.

Figure 8-6. PLL Clocking System



8.11 Register Description

8.11.1 CLKSEL0 – Clock Selection Register 0

Bit	7	6	5	4	3	2	1	0	
(0xD0)	RCSUT1	RCSUT0	EXSUT1	EXSUT0	RCE	EXTE	-	CLKS	CLKSEL0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	See Bit Description				

- **Bit 7:6 – RCSUT[1:0]: SUT for RC oscillator**

These 2 bits are the SUT value for the RC Oscillator. If the RC oscillator is selected by fuse bits, the SUT fuse are copied into these bits. A firmware change will not have any effect because this additionnal start-up time is only used after a reset and not after a clock switch.

- **Bit 5:4 – EXSUT[1:0]: SUT for External Oscillator / Low Power Oscillator**

These 2 bits are the SUT value for the External Oscillator / Low Power Oscillator. If the External oscillator / Low Power Oscillator is selected by fuse bits, the SUT fuse are copyed into these bits. The firmware can modify these bits by writing a new value. This value will be used at the next start of the External Oscillator / Low Power Oscillator.

- **Bit 3 – RCE: Enable RC Oscillator**

The RCE bit must be written to logic one to enable the RC Oscillator. The RCE bit must be written to logic zero to disable the RC Oscillator.

- **Bit 2 – EXTE: Enable External Oscillator / Low Power Oscillator**

The OSCE bit must be written to logic one to enable External Oscillator / Low Power Oscillator. The OSCE bit must be written to logic zero to disable the External Oscillator / Low Power Oscillator.

- **Bit 0 – CLKS: Clock Selector**

The CLKS bit must be written to logic one to select the External Oscillator / Low Power Oscillator as CPU clock. The CLKS bit must be written to logic zero to select the RC Oscillator as CPU clock. After a reset, the CLKS bit is set by hardware if the External Oscillator / Low Power Oscil-

lator is selected by the fuse bits configuration. The firmware has to check if the clock is correctly started before selected it.

8.11.2 CLKSEL1 – Clock Selection Register 1

Bit	7	6	5	4	3	2	1	0	
(0xD1)	RCCKSEL3	RCCKSEL2	RCCKSEL1	RCCKSEL0	EXCKSEL3	EXCKSEL2	EXCKSEL1	EXCKSEL0	CLKSEL1
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- **Bit 7:4 – RCCKSEL[3:0]: CKSEL for RC oscillator**

Clock configuration for the RC Oscillator. After a reset, this part of the register is loaded with the 0010b value that corresponds to the RC oscillator. Modifying this value by firmware before switching to RC oscillator is prohibited because the RC clock will not start.

- **Bit 3:0 – EXCKSEL[3:0]: CKSEL for External oscillator / Low Power Oscillator**

Clock configuration for the External Oscillator / Low Power Oscillator. After a reset, if the External oscillator / Low Power Oscillator is selected by fuse bits, this part of the register is loaded with the fuse configuration. Firmware can modify it to change the start-up time after the clock switch.

8.11.3 CLKSTA – Clock Status Register

Bit	7	6	5	4	3	2	1	0	
(0xD2)	-	-	-	-	-	-	RCON	EXTON	CLKSTA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	See Bit Description				

- **Bit 7:2 - Res: Reserved bits**

These bits are reserved and will always read as zero.

- **Bit 1 – RCON: RC Oscillator On**

This bit is set by hardware to one if the RC Oscillator is running.
This bit is set by hardware to zero if the RC Oscillator is stoped.

- **Bit 0 – EXTON: External Oscillator / Low Power Oscillator On**

This bit is set by hardware to one if the External Oscillator / Low Power Oscillator is running.
This bit is set by hardware to zero if the External Oscillator / Low Power Oscillator is stoped.

8.11.4 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 – CAL[7:0]: 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 26-1 on page 266. The application software can write this register to change

the oscillator frequency. The oscillator can be calibrated to frequencies as specified in Table 26-1 on page 266. Calibration outside that range is not guaranteed.

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

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

The CAL[6:0] 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.

8.11.5 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.

- **Bit 6:4 - Reserved bits**

These bits are reserved and will always read as zero.

- **Bits 3:0 – CLKPS[3:0]: 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 8-9.

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 8-9. Clock Prescaler Select

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

8.11.6 PLLCSR – PLL Control and Status Register

Bit	7	6	5	4	3	2	1	0	
0x29 (0x49)	–	–	–	DIV5	DIV3	PINDIV	PLLE	PLOCK	PLLCSR
Read/Write	R	R	R	R/W	R	R	R/W	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:5 – Res: Reserved Bits**

These bits are reserved bits in the ATmega8U2/16U2/32U2 and always read as zero.

- **Bit 4 – DIV5 PLL Input Prescaler (1:5)**

- **Bit 3 – DIV3 PLL Input Prescaler (1:3)**

- **Bit 2 – PINDIV PLL Input Prescaler (1:1, 1:2)**

These bits allow to configure the PLL input prescaler to generate the 8MHz input clock for the PLL from either a 8 or 16 MHz input.

When using a 8 MHz clock source, this bit must be set to 0 before enabling PLL (1:1).

When using a 16 MHz clock source, this bit must be set to 1 before enabling PLL (1:2).

- **Bit 3:2 – Res: Reserved Bits**

These bits are reserved and always read as zero.

- **Bit 1 – PLLE: PLL Enable**

When the PLLE is set, the PLL is started. Note that the Calibrated 8 MHz Internal RC oscillator is automatically enabled when the PLLE bit is set and with PINMUX (see PLLFRQ register) is set. The PLL must be disabled before entering Power down mode in order to stop Internal RC Oscillator and avoid extra-consumption.

- **Bit 0 – PLOCK: PLL Lock Detector**

When the PLOCK bit is set, the PLL is locked to the reference clock. After the PLL is enabled, it takes about several ms for the PLL to lock. To clear PLOCK, clear PLLE.

9. Power Management and Sleep Modes

9.1 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.

9.2 Sleep Modes

Figure 8-1 on page 26 presents the different clock systems in the ATmega8U2/16U2/32U2, and their distribution. The figure is helpful in selecting an appropriate sleep mode. shows the different sleep modes and their wake up sources.

Table 9-1. 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}**	**Main Clock Source Enabled**	**INT[7:0] and PCINT12-0**	**SPM/ EEPROM Ready**	**WDT Interrupt**	**Other I/O**	**USB Synchronous Interrupts**	**USB Asynchonous Interrupts(3)**
Idle			X	X	X	X	X	X	X	X
Power-down					X(2)		X			X
Power-save					X(2)		X			X
Standby(1)				X	X(2)		X			X
Extended Standby				X	X(2)		X			X

Notes: 1. Only recommended with external crystal or resonator selected as clock source.
2. For INT[7:4], only level interrupt.
3. Asynchronous USB interrupt is WAKEUPI only.

To enter any of the five sleep modes, the SE bit in SMCR must be written to logic one and a SLEEP instruction must be executed. The SM2, SM1, and SM0 bits in the SMCR Register select which sleep mode (Idle, Power-down, Power-save, Standby or Extended standby) will be activated by the SLEEP instruction. See Table 9-2 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.

9.3 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 USB, SPI, USART, Analog Comparator, 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, USART Transmit Complete or some USB interrupts (like SOFI, WAKEUPI...). 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.

9.4 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 INT7:4, an external interrupt on INT3:0, a pin change interrupt or an asynchronous USB interrupt source (WAKEUPI only), can wake up the MCU. This sleep mode basically halts all generated clocks, allowing operation of asynchronous modules only.

Note that if a level triggered interrupt is used for wake-up from Power-down mode, the changed level must be held for some time to wake up the MCU. Refer to “External Interrupts” on page 84 for details.

When waking up from Power-down mode, there is a delay from the wake-up condition occurs until the wake-up becomes effective. This allows the clock to restart and become stable after having been stopped. The wake-up period is defined by the same CKSEL Fuses that define the Reset Time-out period, as described in “Clock Sources” on page 29.

9.5 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. This mode has been conserved for compatibility purpose with higher-end products.

9.6 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.

9.7 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. So Extended Standby Mode is equivalent to Standy Mode, but is also conserved for compatibility purpose. From Extended Standby mode, the device wakes up in six clock cycle.

9.8 Power Reduction Register

The Power Reduction Registers (PRR0 and PRR1), provides a method to stop the clock to individual peripherals to reduce power consumption. See “PRR0 – Power Reduction Register 0” and “PRR1 – Power Reduction Register 1” on page 46 for details. The current state of the peripheral is frozen and the I/O registers can not be read or written. Resources used by the peripheral

when stopping the clock will remain occupied, hence the peripheral should in most cases be disabled before stopping the clock. Waking up a module, which is done by clearing the bit in PRR, puts the module in the same state as before shutdown.

Module shutdown can be used in Idle mode and Active mode to significantly reduce the overall power consumption.

9.9 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.

9.9.1 Analog Comparator

When entering Idle mode, the Analog Comparator should be disabled if not used. In other sleep modes, the Analog Comparator is automatically disabled. However, if the Analog Comparator is set up to use the Internal Voltage Reference as input, the Analog Comparator should be disabled in all sleep modes. Otherwise, the Internal Voltage Reference will be enabled, independent of sleep mode. Refer to "Analog Comparator" on page 223 for details on how to configure the Analog Comparator.

9.9.2 Brown-out Detector

If the Brown-out Detector is not needed by the application, this module should be turned off. If the Brown-out Detector is enabled by the BODLEVEL Fuses, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to "Brown-out Detection" on page 50 for details on how to configure the Brown-out Detector.

9.9.3 Internal Voltage Reference

The Internal Voltage Reference will be enabled when needed by the Brown-out Detection, or the Analog Comparator. If these modules are disabled as described in the sections above, the internal voltage reference will be disabled and it will not be consuming power. When turned on again, the user must allow the reference to start up before the output is used. If the reference is kept on in sleep mode, the output can be used immediately. Refer to "Internal Voltage Reference" on page 51 for details on the start-up time.

9.9.4 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 "Interrupts" on page 64 for details on how to configure the Watchdog Timer.

9.9.5 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 the I/O clock ($clk_{I/O}$) is stopped, the input buffers of the device will be disabled. This ensures that no power is consumed by the input logic when not needed. In some cases, the input logic is needed for detecting wake-up conditions, and it will then be enabled. Refer to the section "Digital Input Enable and Sleep Modes" on page 71 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). Refer to "DIDR1 – Digital Input Disable Register 1" on page 225 for details.

9.9.6 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.

9.10 Register Description

9.10.1 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	

- **Bit 7:4 - Reserved bits**

These bits are reserved and will always read as zero.

- **Bits 3:1 – SM[2:0]: Sleep Mode Select Bits 2, 1, and 0**

These bits select between the five available sleep modes as shown in Table 9-2.

Table 9-2. Sleep Mode Select

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	Reserved
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.

9.10.2 PRR0 – Power Reduction Register 0

Bit	7	6	5	4	3	2	1	0	
(0x64)	-	-	PRTIM0	–	PRTIM1	PRSPI	-	-	PRR0
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 - Res: Reserved bits**

These bits are reserved and will always read as zero.

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

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

- **Bit 4 - Res: Reserved bit**

This bit is reserved and will always read as zero.

- **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 - Res: Reserved bit**

These bits are reserved and will always read as zero.

- **Bit 0 - Res: Reserved bit**

These bits are reserved and will always read as zero.

9.10.3 PRR1 – Power Reduction Register 1

Bit	7	6	5	4	3	2	1	0	
(0x65)	PRUSB	–	–	–	-	–	–	PRUSART1	PRR1
Read/Write	R/W	R	R	R	R/W	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 - PRUSB: Power Reduction USB**

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

- **Bit 6:1 - Res: Reserved bits**

These bits are reserved and will always read as zero.

- **Bit 0 - 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 re initialized to ensure proper operation.

10. System Control and Reset

10.1 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 10-1 shows the reset logic. “System and Reset Characteristics” on page 267 defines the electrical parameters of the reset circuitry.

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

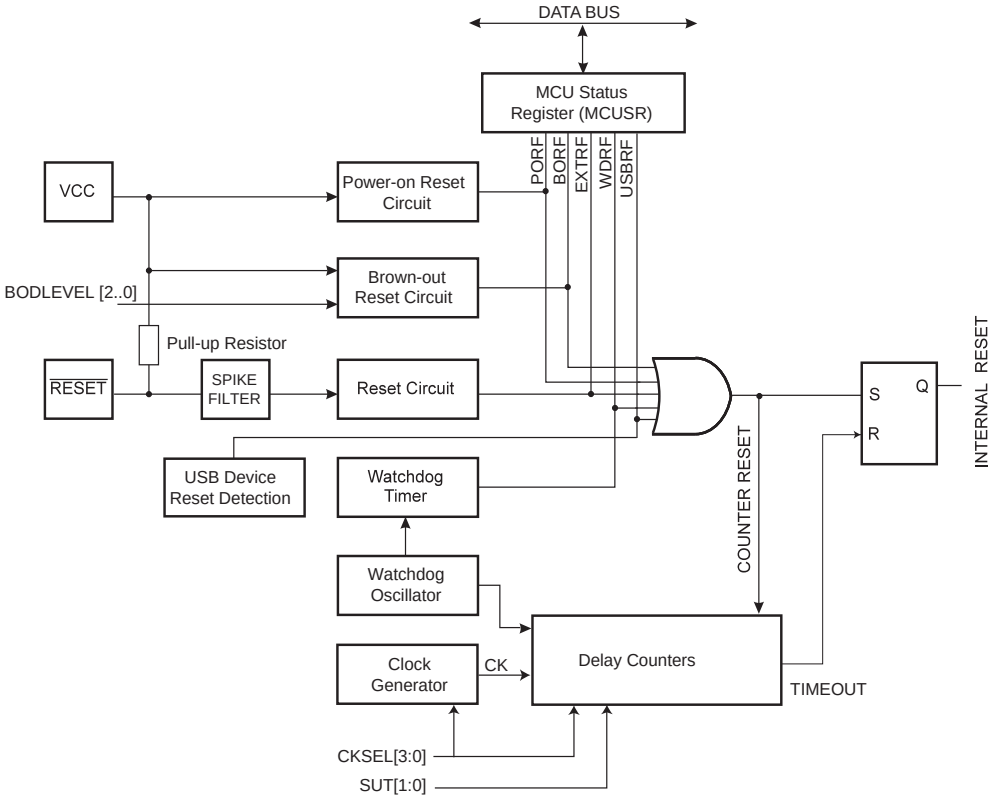
After all reset sources have gone inactive, a delay counter is invoked, stretching the internal reset. This allows the power to reach a stable level before normal operation starts. The time-out period of the delay counter is defined by the user through the SUT and CKSEL Fuses. The different selections for the delay period are presented in “Clock Sources” on page 29.

10.2 Reset Sources

The ATmega8U2/16U2/32U2 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.
- USB Reset. The MCU is reset when the USB macro is enabled and detects a USB Reset. Note that with this reset the USB macro remains enabled so that the device stays attached to the bus.

Figure 10-1. Reset Logic



10.2.1 Power-on Reset

A Power-on Reset (POR) pulse is generated by an On-chip detection circuit. The detection level is defined in “System and Reset Characteristics” on page 267. 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 10-2. MCU Start-up, $\overline{\text{RESET}}$ Tied to V_{CC}

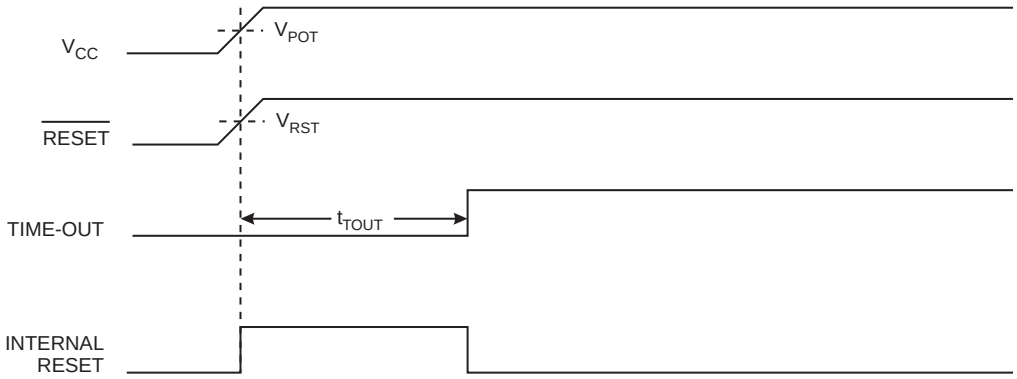
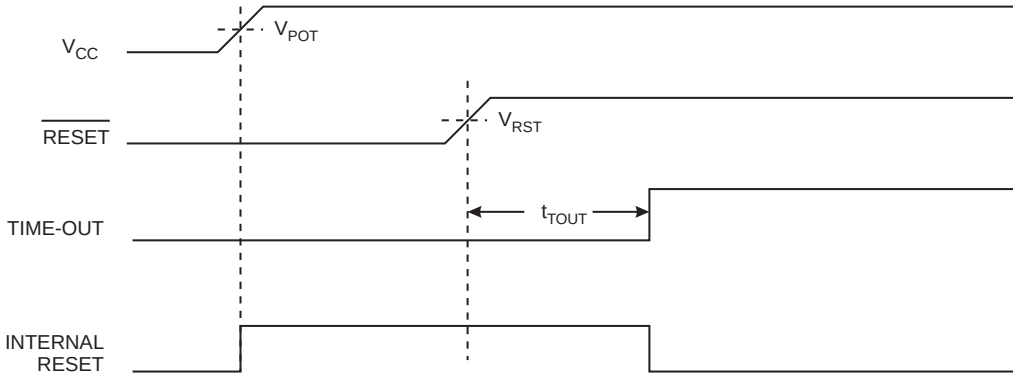


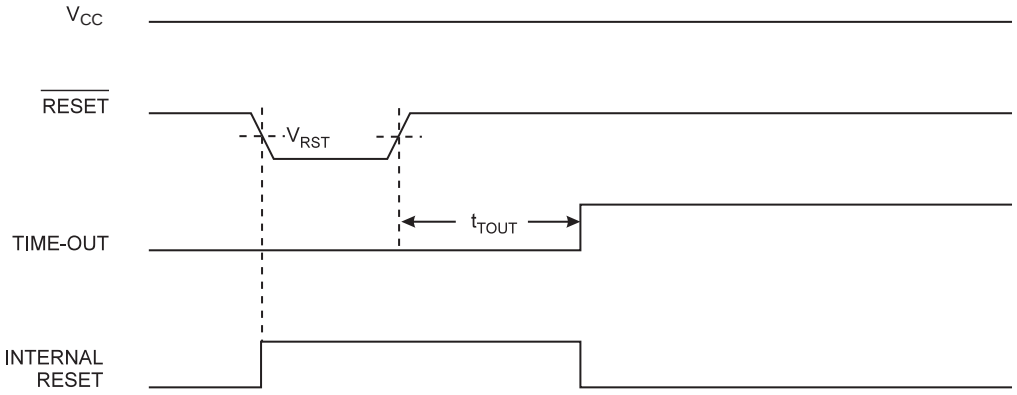
Figure 10-3. MCU Start-up, $\overline{\text{RESET}}$ Extended Externally



10.2.2 External Reset

An External Reset is generated by a low level on the $\overline{\text{RESET}}$ pin. Reset pulses longer than the minimum pulse width (see "System and Reset Characteristics" on page 267) 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 10-4. External Reset During Operation

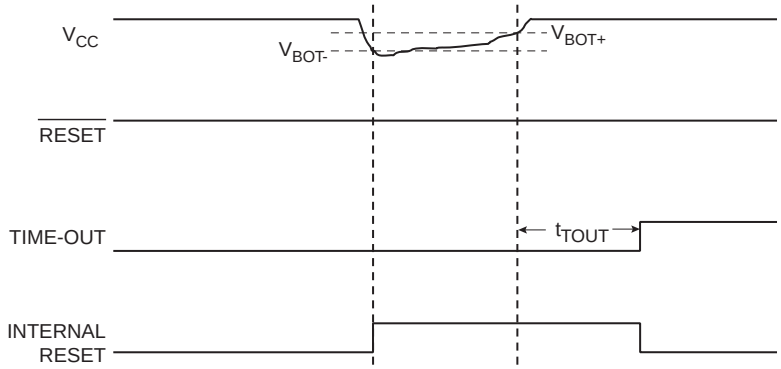


10.2.3 Brown-out Detection

ATmega8U2/16U2/32U2 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 10-5), the Brown-out Reset is immediately activated. When V_{CC} increases above the trigger level (V_{BOT+} in Figure 10-5), the delay counter starts the MCU after the Time-out period t_{TOUT} has expired.

The BOD circuit will only detect a drop in V_{CC} if the voltage stays below the trigger level for longer than t_{BOD} given in "System and Reset Characteristics" on page 267.

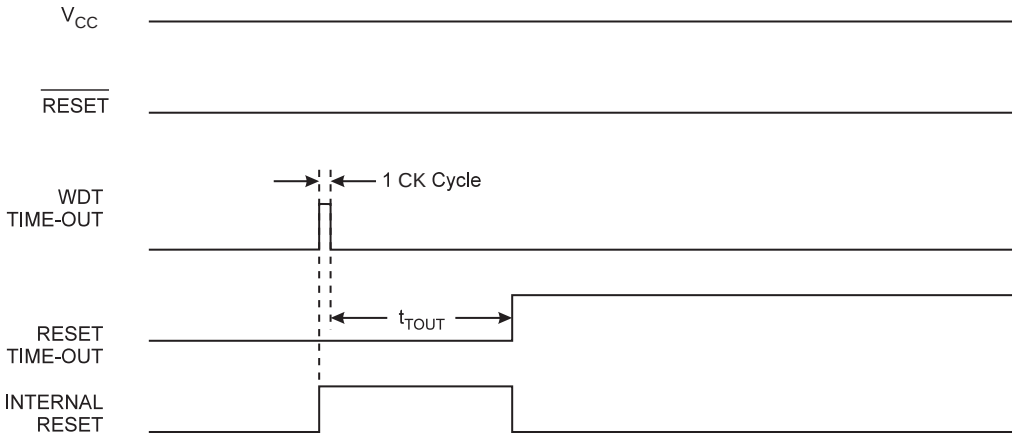
Figure 10-5. Brown-out Reset During Operation



10.2.4 Watchdog Reset

When the Watchdog times out, it will generate a short reset pulse of one CK cycle duration. On the falling edge of this pulse, the delay timer starts counting the Time-out period t_{TOUT}. Refer to "Watchdog Timer" on page 51 for details on operation of the Watchdog Timer.

Figure 10-6. Watchdog Reset During Operation

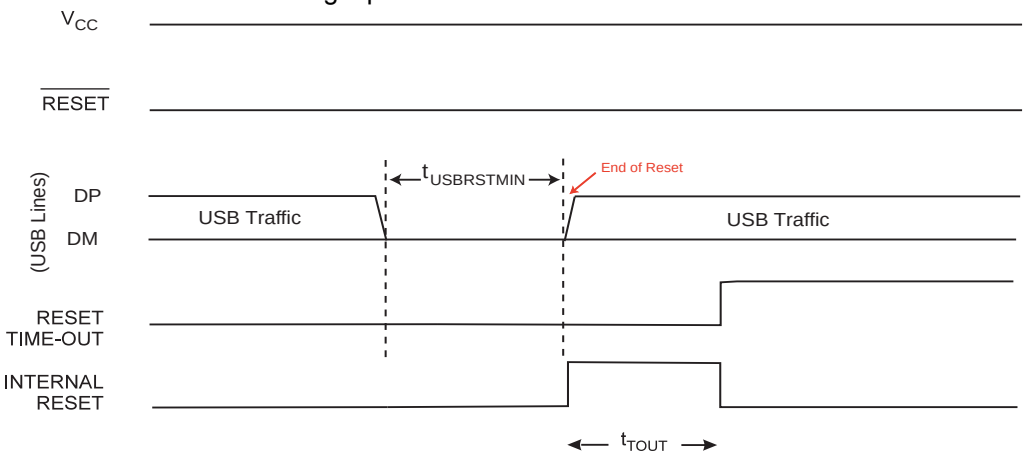


10.2.5 USB Reset

When the USB macro is enabled and configured with the USB reset MCU feature enabled, and if a valid USB Reset signalling is detected, the microcontroller is reset unless the USB macro

that remains enabled. This allows the device to stay attached to the bus during and after the reset, while enhancing firmware reliability.

Figure 10-7. USB Reset During Operation



10.3 Internal Voltage Reference

ATmega8U2/16U2/32U2 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.

10.3.1 Voltage Reference Enable Signals and Start-up Time

The voltage reference has a start-up time that may influence the way it should be used. The start-up time is given in “System and Reset Characteristics” on page 267. 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).

Thus, when the BOD is not enabled, after setting the ACBG bit, the user must always allow the reference to start up before the output from the Analog Comparator 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.

10.4 Watchdog Timer

10.4.1 Features

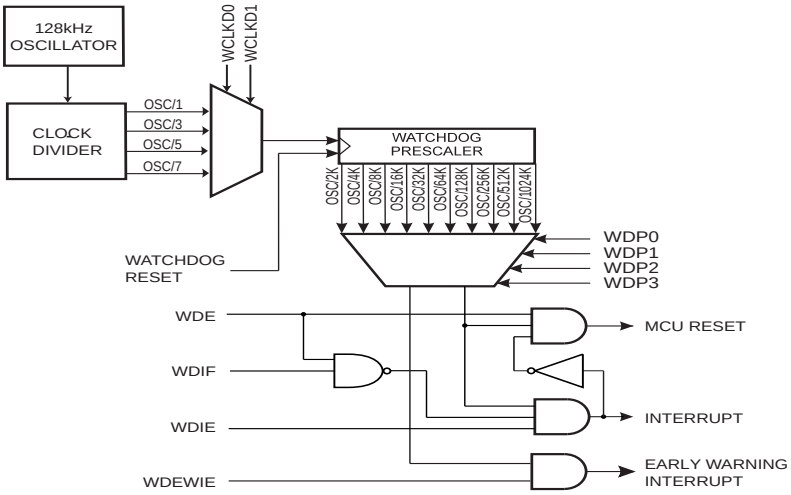
- **Clocked from separate On-chip Oscillator**
- **3 Operating modes**
  - **Interrupt**
  - **System Reset**
  - **Interrupt and System ResetSelectable Time-out period from 16ms to 8s**
- **Possible Hardware fuse Watchdog always on (WDTON) for fail-safe mode**
- **Early warning after one Time-Out period reached, programmable Reset (see operating modes) after 2 Time-Out periods reached.**

10.4.2 Overview

ATmega8U2/16U2/32U2 has an Enhanced Watchdog Timer (WDT). The WDT is a timer counting cycles of a separate on-chip 128 kHz oscillator. The WDT gives a early warning interrupt

when the counter reaches a given time-out value. The WDT gives an interrupt or a system reset when the counter reaches two times the 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 10-8. Watchdog Timer



In Interrupt mode, the WDT gives an interrupt when the timer expires two times. 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 two times. 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.

In addition to these modes, the early warning interrupt can be enabled in order to generate an interrupt when the WDT counter expires the first time.

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 or changing time-out configuration is as follows:

1. In the same operation, write a logic one to the Watchdog change enable bits WDCE and WDE. A logic one must be written to WDE regardless of the previous value of the WDE bit and even if it will be cleared after the operation.
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.

While the WDT prescaler allows only even division factors (2, 4, 8...), the WDT peripheral also includes a clock divider that directly acts on the clock source. This divider handles odd division factors (3, 5, 7). In combination with the prescaler, a large number of time-out values can be obtained.

The divider factor change is also ruled by the secure timed sequence : first the WDE and WDCE bits must be set, and then four cycles are available to load the new divider value into the WDTCKD register. Be aware that after this operation WDE will still be set. So keep in mind the importance of order of operations. When setting up the WDT in Interrupt mode with specific values of prescaler and divider, the divider register must be loaded before the prescaler register :

1. Set WDCE and WDE
2. Load the divider factor into WDTCKD
3. Wait WDCE being automatically cleared (just wait 2 more cycles)
4. Set again WDCE and WDE
5. Clear WDE, set WDIE and load the prescaler factor into WDTCSR in a same operation
6. Now the system is properly configured for Interrupt only mode. Inverting the two operations would have been resulted into “Reset and Interrupt mode” and needed a third operation to clear WDE.

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, (0xff & (0<<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();
}
```

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

Note: 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 initialisation 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();
}
```

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

Note: 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.

10.5 Register Description

10.5.1 MCUSR – MCU Status Register

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

Bit	7	6	5	4	3	2	1	0	
0x34 (0x54)	–	–	USBRF	–	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 7:6 – Res: Reserved Bit**

These bits are reserved and will always read as zero.

- **Bit 5 – USBRF: USB Reset Flag**

This bit is set if a USB Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

- **Bit 4 – Res: Reserved Bit**

This bit is reserved and will always read as zero.

- **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.

10.5.2 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 twice in the Watchdog Timer and if the Watchdog Timer is configured for interrupt. WDIF is automatically 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. Two consecutives times-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 reinitialize the 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 10-1. Watchdog Timer Configuration

WDTON (Fuse)	WDE	WDIE	Mode	Action on 2x Time-out
1 (unprogrammed)	0	0	Stopped	None
1 (unprogrammed)	0	1	Interrupt Mode	Interrupt
1 (unprogrammed)	1	0	System Reset Mode	Reset
1 (unprogrammed)	1	1	Interrupt and System Reset Mode	Interrupt, then go to System Reset Mode
0 (programmed)	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 - WDP[3:0]: 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 on page 58.

10.5.3 WDTCKD – Watchdog Timer Clock Divider Register

Bit	7	6	5	4	3	2	1	0	
(0x62)	-	-	WDEWIF-CM	WCLKD2	WDEWIF	WDEWIE	WCLKD1	WCLKD0	WDTCKD
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	

- **Bit 7:6 - Res: Reserved bits**

These bits are reserved and will always read as zero.

- **Bit 5 - WDEWIFCL: Watchdog Early Warning Flag Clear Mode**

When this bit has been set by software, the WDEWIF interrupt flag is not cleared by hardware when entering the Watchdog Interrupt subroutine (it has to be cleared by software by writing a logic one to the flag).

When cleared, the WDEWIF is cleared by hardware when executing the corresponding interrupt handling vector.

- **Bit 4 - WCLKD2 bit: Watchdog Timer Clock Divider**

See “Bit 1:0 - WCLKD[1:0]: Watchdog Timer Clock Divider” on page 58.

- **Bit 3 - WDEWIF: Watchdog Early Warning Interrupt Flag**

This bit is set when a first time-out occurs in the Watchdog Timer and if the WDEWIE bit is enabled. WDEWIF is automatically cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, WDIF can be cleared by writing a logic one to the flag. When the I-bit in SREG and WDEWIE are set, the Watchdog Time-out Interrupt is executed.

- **Bit 2 - WDEWIE: Watchdog Early Warning Interrupt Enable**

When this bit has been set by software, an interrupt will be generated on the watchdog interrupt vector when the Early warning flag is set to one by hardware.

- **Bit 1:0 - WCLKD[1:0]: Watchdog Timer Clock Divider**

Table 10-2. Watchdog Timer Clock Divider Configuration

WCLKD2	WCLKD1	WCLKD0	Mode
0	0	0	$Clk_{WDT} = Clk_{128k}$
0	0	1	$Clk_{WDT} = Clk_{128k} / 3$
0	1	0	$Clk_{WDT} = Clk_{128k} / 5$
0	1	1	$Clk_{WDT} = Clk_{128k} / 7$
1	0	0	$Clk_{WDT} = Clk_{128k} / 9$
1	0	1	$Clk_{WDT} = Clk_{128k} / 11$
1	1	0	$Clk_{WDT} = Clk_{128k} / 13$
1	1	1	$Clk_{WDT} = Clk_{128k} / 15$

Table 10-3. Watchdog Timer Prescale Select, DIV = 0 (CLKwdt = CLK128 / 1)

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles before 1st time-out (Early warning)	Early warning Typical Time-out at V_{CC} = 5.0V	Watchdog Reset/Interrupt Typical Time-out at V_{CC} = 5.0V
0	0	0	0	2K (2048) cycles	16 ms	32 ms
0	0	0	1	4K (4096) cycles	32 ms	64 ms
0	0	1	0	8K (8192) cycles	64 ms	128 ms
0	0	1	1	16K (16384) cycles	0.125 s	0.250 s
0	1	0	0	32K (32768) cycles	0.25 s	0.5 s
0	1	0	1	64K (65536) cycles	0.5 s	1.0 s
0	1	1	0	128K (131072) cycles	1.0 s	2.0 s
0	1	1	1	256K (262144) cycles	2.0 s	4.0 s
1	0	0	0	512K (524288) cycles	4.0 s	8.0 s
1	0	0	1	1024K (1048576) cycles	8.0 s	16.0 s

Table 10-3. Watchdog Timer Prescale Select, DIV = 0 (CLKwdt = CLK128 / 1) (Continued)

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles before 1st time-out (Early warning)	Early warning Typical Time-out at V_{CC} = 5.0V	Watchdog Reset/Interrupt Typical Time-out at V_{CC} = 5.0V
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			

Table 10-4. Watchdog Timer Prescale Select, DIV = 1 (CLKwdt = CLK128 / 3)

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles before 1st time-out (Early warning)	Early warning Typical Time-out at V_{CC} = 5.0V	Watchdog Reset/Interrupt Typical Time-out at V_{CC} = 5.0V
0	0	0	0	2K (2048) cycles	48 ms	96 ms
0	0	0	1	4K (4096) cycles	96 ms	192 ms
0	0	1	0	8K (8192) cycles	192 ms	384 ms
0	0	1	1	16K (16384) cycles	0.375 s	0.75 s
0	1	0	0	32K (32768) cycles	0.75 s	1.5 s
0	1	0	1	64K (65536) cycles	1.5 s	3 s
0	1	1	0	128K (131072) cycles	3 s	6 s
0	1	1	1	256K (262144) cycles	6 s	12 s
1	0	0	0	512K (524288) cycles	12 s	24 s
1	0	0	1	1024K (1048576) cycles	24 s	48 s
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			

Table 10-5. Watchdog Timer Prescale Select, DIV = 2 (CLKwdt = CLK128 / 5)

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles before 1st time-out (Early warning)	Early warning Typical Time-out at V_{CC} = 5.0V	Watchdog Reset/Interrupt Typical Time-out at V_{CC} = 5.0V
0	0	0	0	2K (2048) cycles	80 ms	160 ms
0	0	0	1	4K (4096) cycles	160 ms	320 ms
0	0	1	0	8K (8192) cycles	320 ms	640 ms
0	0	1	1	16K (16384) cycles	0.625 s	1.25 s
0	1	0	0	32K (32768) cycles	1.25 s	2.5 s
0	1	0	1	64K (65536) cycles	2.5 s	5 s
0	1	1	0	128K (131072) cycles	5 s	10 s
0	1	1	1	256K (262144) cycles	10 s	20 s
1	0	0	0	512K (524288) cycles	20 s	40 s
1	0	0	1	1024K (1048576) cycles	40 s	80 s
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			

Table 10-6. Watchdog Timer Prescale Select, DIV = 3 (CLKwdt = CLK128 / 7)

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles before 1st time-out (Early warning)	Early warning Typical Time-out at V_{CC} = 5.0V	Watchdog Reset/Interrupt Typical Time-out at V_{CC} = 5.0V
0	0	0	0	2K (2048) cycles	112 ms	224 ms
0	0	0	1	4K (4096) cycles	224 ms	448 ms
0	0	1	0	8K (8192) cycles	448 ms	896 ms
0	0	1	1	16K (16384) cycles	0.875 s	1.75 s
0	1	0	0	32K (32768) cycles	1.75 s	3.5 s
0	1	0	1	64K (65536) cycles	3.5 s	7 s
0	1	1	0	128K (131072) cycles	7 s	14 s
0	1	1	1	256K (262144) cycles	14 s	28 s
1	0	0	0	512K (524288) cycles	28 s	56 s
1	0	0	1	1024K (1048576) cycles	56 s	112 s

Table 10-6. Watchdog Timer Prescale Select, DIV = 3 (CLKwdt = CLK128 / 7) (Continued)

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles before 1st time-out (Early warning)	Early warning Typical Time-out at V_{CC} = 5.0V	Watchdog Reset/Interrupt Typical Time-out at V_{CC} = 5.0V
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			

Table 10-7. Watchdog Timer Prescale Select, DIV = 4 (CLKwdt = CLK128 / 9)

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles before 1st time-out (Early warning)	Early warning Typical Time-out at V_{CC} = 5.0V	Watchdog Reset/Interrupt Typical Time-out at V_{CC} = 5.0V
0	0	0	0	2K (2048) cycles	72ms	144 ms
0	0	0	1	4K (4096) cycles	144 ms	288 ms
0	0	1	0	8K (8192) cycles	288 ms	576 ms
0	0	1	1	16K (16384) cycles	576 s	1.15 s
0	1	0	0	32K (32768) cycles	1.1 s	2.3 s
0	1	0	1	64K (65536) cycles	2.3 s	4.6 s
0	1	1	0	128K (131072) cycles	4.6 s	9.2 s
0	1	1	1	256K (262144) cycles	9.2 s	18.4s
1	0	0	0	512K (524288) cycles	18.4 s	36.8 s
1	0	0	1	1024K (1048576) cycles	36.8 s	73 s
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			

Table 10-8. Watchdog Timer Prescale Select, DIV = 5 (CLKwdt = CLK128 / 11)

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles before 1st time-out (Early warning)	Early warning Typical Time-out at V_{CC} = 5.0V	Watchdog Reset/Interrupt Typical Time-out at V_{CC} = 5.0V
0	0	0	0	2K (2048) cycles	88 ms	176 ms
0	0	0	1	4K (4096) cycles	176 ms	352 ms
0	0	1	0	8K (8192) cycles	352 ms	704 ms
0	0	1	1	16K (16384) cycles	704 ms	1.4 s
0	1	0	0	32K (32768) cycles	1.4 s	2.8 s
0	1	0	1	64K (65536) cycles	2.8 s	5.6 s
0	1	1	0	128K (131072) cycles	5.6 s	11.2 s
0	1	1	1	256K (262144) cycles	11.2 s	22.5 s
1	0	0	0	512K (524288) cycles	22.5 s	45 s
1	0	0	1	1024K (1048576) cycles	45s	90 s
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			

Table 10-9. Watchdog Timer Prescale Select, DIV = 6(CLKwdt = CLK128 / 13)

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles before 1st time-out (Early warning)	Early warning Typical Time-out at V_{CC} = 5.0V	Watchdog Reset/Interrupt Typical Time-out at V_{CC} = 5.0V
0	0	0	0	2K (2048) cycles	104 ms	208 ms
0	0	0	1	4K (4096) cycles	208 ms	416 ms
0	0	1	0	8K (8192) cycles	416 ms	832 ms
0	0	1	1	16K (16384) cycles	832 ms	1.64 s
0	1	0	0	32K (32768) cycles	1.6 s	3.3 s
0	1	0	1	64K (65536) cycles	3.3 s	6.6 s
0	1	1	0	128K (131072) cycles	6.6 s	13.3 s
0	1	1	1	256K (262144) cycles	13.3 s	26.6 s
1	0	0	0	512K (524288) cycles	26.6 s	53.2 s
1	0	0	1	1024K (1048576) cycles	53.2 s	106.4 s

Table 10-9. Watchdog Timer Prescale Select, DIV = 6(CLKwdt = CLK128 / 13) (Continued)

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles before 1st time-out (Early warning)	Early warning Typical Time-out at V_{CC} = 5.0V	Watchdog Reset/Interrupt Typical Time-out at V_{CC} = 5.0V
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			

Table 10-10. Watchdog Timer Prescale Select, DIV = 7 (CLKwdt = CLK128 / 15)

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles before 1st time-out (Early warning)	Early warning Typical Time-out at V_{CC} = 5.0V	Watchdog Reset/Interrupt Typical Time-out at V_{CC} = 5.0V
0	0	0	0	2K (2048) cycles	120 ms	240 ms
0	0	0	1	4K (4096) cycles	240 ms	480 ms
0	0	1	0	8K (8192) cycles	480 ms	960 ms
0	0	1	1	16K (16384) cycles	0.960 s	1.9 s
0	1	0	0	32K (32768) cycles	1.92 s	3.8 s
0	1	0	1	64K (65536) cycles	3.8 s	7.6 s
0	1	1	0	128K (131072) cycles	7.6 s	15.3 s
0	1	1	1	256K (262144) cycles	15.3 s	30.7 s
1	0	0	0	512K (524288) cycles	30.7 s	61.4 s
1	0	0	1	1024K (1048576) cycles	61.4 s	122 s
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			

11. Interrupts

11.1 Overview

This section describes the specifics of the interrupt handling as performed in ATmega8U2/16U2/32U2. For a general explanation of the AVR interrupt handling, refer to "Reset and Interrupt Handling" on page 13.

11.2 Interrupt Vectors in ATmega8U2/16U2/32U2

Table 11-1. 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, USB Reset and debugWIRE 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	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	PCINT0	Pin Change Interrupt Request 0
11	$0014	PCINT1	Pin Change Interrupt Request 1
12	$0016	USB General	USB General Interrupt request
13	$0018	USB Endpoint	USB Endpoint Interrupt request
14	$001A	WDT	Watchdog Time-out Interrupt
15	$001C	TIMER1 CAPT	Timer/Counter1 Capture Event
16	$001E	TIMER1 COMPA	Timer/Counter1 Compare Match A
17	$0020	TIMER1 COMPB	Timer/Counter1 Compare Match B
18	$0022	TIMER1 COMPC	Timer/Counter1 Compare Match C
19	$0024	TIMER1 OVF	Timer/Counter1 Overflow
20	$0026	TIMER0 COMPA	Timer/Counter0 Compare Match A
21	$0028	TIMER0 COMPB	Timer/Counter0 Compare match B
22	$002A	TIMER0 OVF	Timer/Counter0 Overflow
23	$002C	SPI, STC	SPI Serial Transfer Complete
24	$002E	USART1 RX	USART1 Rx Complete
25	$0030	USART1 UDRE	USART1 Data Register Empty
26	$0032	USART1TX	USART1 Tx Complete

Table 11-1. Reset and Interrupt Vectors (Continued)

Vector No.	Program Address(2)	Source	Interrupt Definition
27	$0034	ANALOG COMP	Analog Comparator
28	$0036	EE READY	EEPROM Ready
29	$0038	SPM READY	Store Program Memory Ready

Notes: 1. When the BOOTRST Fuse is programmed, the device will jump to the Boot Loader address at reset, see "Memory Programming" on page 246.

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. Moreover, contrary to other 8K/16K devices, the interrupt vectors spacing remains identical (2 words) for both 8KB and 16KB versions.

Table 11-2 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 11-2. 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 23-8 on page 239. For the BOOTRST Fuse "1" means unprogrammed while "0" means programmed.

11.2.1 Moving Interrupts Between Application and Boot Space

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

11.3 Register Description

11.3.1 MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	JTD	–	–	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R/W	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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

When the IVSEL bit is cleared (zero), the Interrupt Vectors are placed at the start of the Flash memory. When this bit is set (one), the Interrupt Vectors are moved to the beginning of the Boot Loader section of the Flash. The actual address of the start of the Boot Flash Section is determined by the BOOTSZ Fuses. Refer to the section "Memory Programming" on page 246 for details. To avoid unintentional changes of Interrupt Vector tables, a special write procedure must be followed to change the IVSEL bit:

a. Write the Interrupt Vector Change Enable (IVCE) bit to one.

b. 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 "Memory Programming" on page 246 for details on Boot Lock bits.

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

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

Assembly Code Example

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

C Code Example

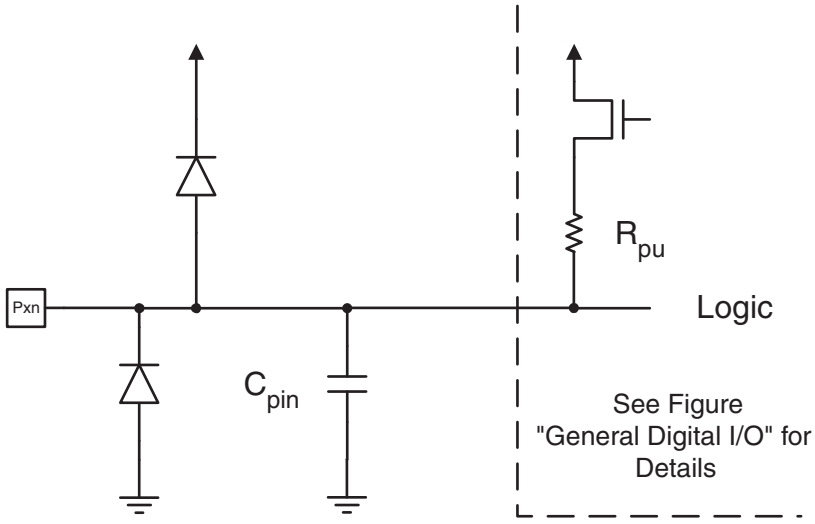
```
void Move_interrupts(void)
{
  /* Enable change of Interrupt Vectors */
  MCUCR = (1<<IVCE);
  /* Move interrupts to Boot Flash section */
  MCUCR = (1<<IVSEL);
}
```

12. I/O-Ports

12.1 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 12-1. Refer to "Electrical Characteristics" on page 264 for a complete list of parameters.

Figure 12-1. 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 "Register Description for I/O-Ports" on page 82.

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

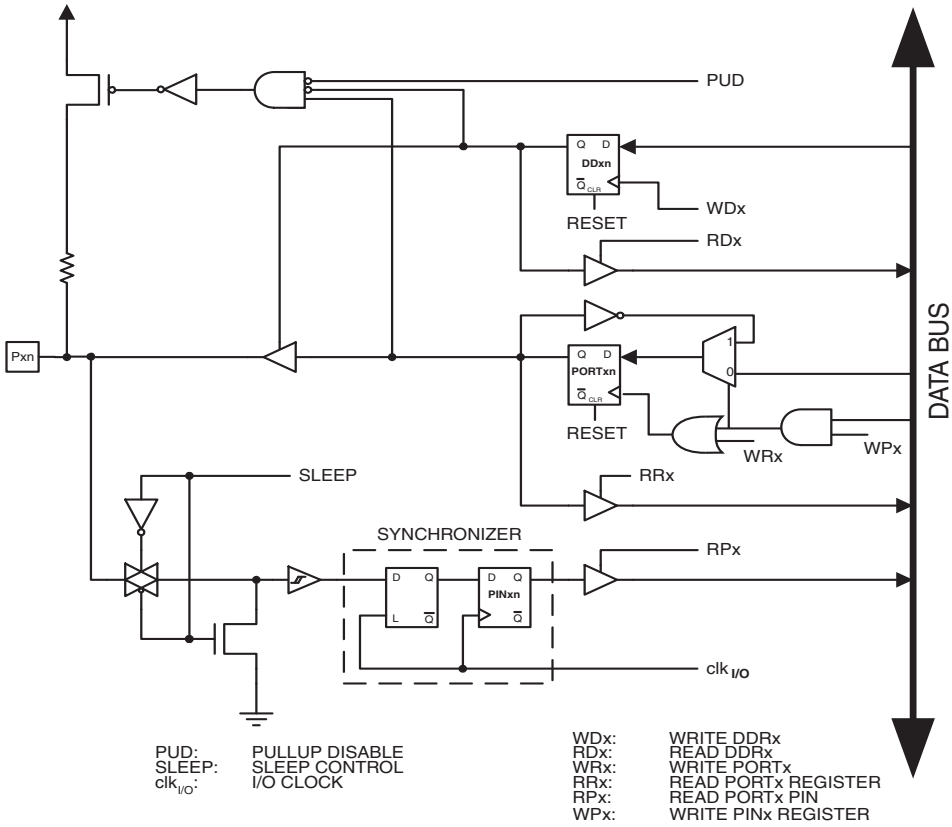
Using the I/O port as General Digital I/O is described in "Ports as General Digital I/O" on page 68. Most port pins are multiplexed with alternate functions for the peripheral features on the device. How each alternate function interferes with the port pin is described in "Alternate Port Functions" on page 72. 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.

12.2 Ports as General Digital I/O

The ports are bi-directional I/O ports with optional internal pull-ups. Figure 12-2 shows a functional description of one I/O-port pin, here generically called Pxn.

Figure 12-2. 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.

12.2.1 Configuring the Pin

Each port pin consists of three register bits: DDxn, PORTxn, and PINxn. As shown in "Register Description for I/O-Ports" on page 82, 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).

12.2.2 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.

12.2.3 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) occurs. 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 12-1 summarizes the control signals for the pin value.

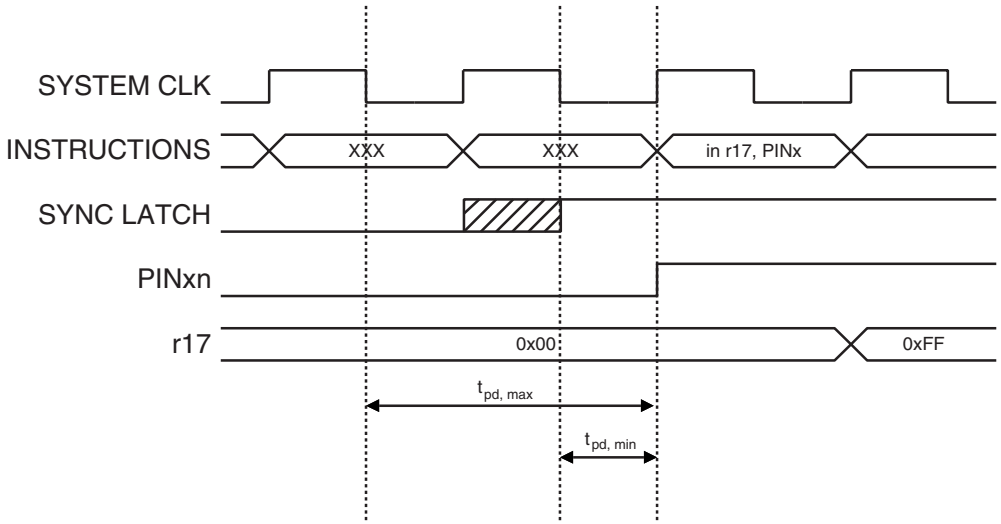
Table 12-1. Port Pin Configurations

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)

12.2.4 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 12-2, 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 12-3 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.

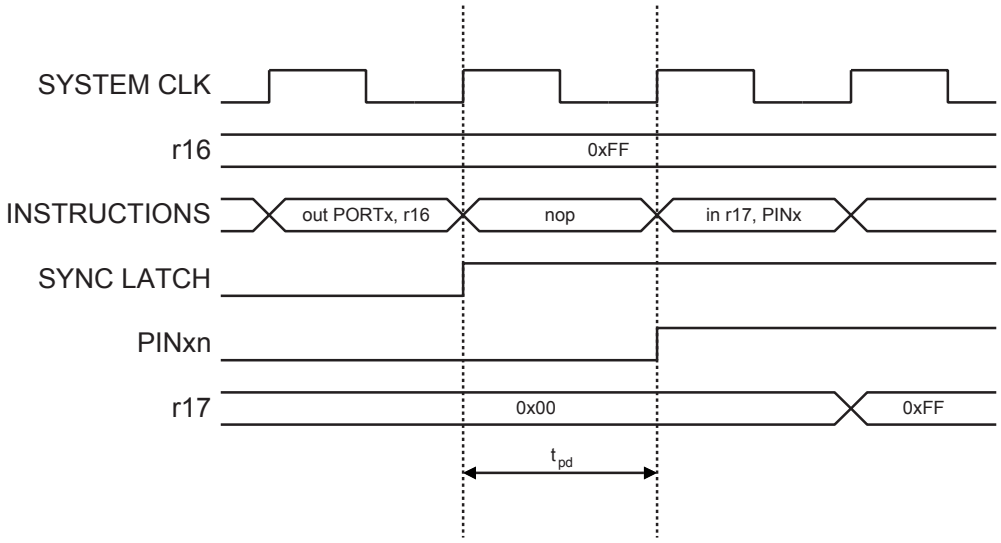
Figure 12-3. Synchronization when Reading an Externally Applied Pin value



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 12-4. 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 12-4. 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.

12.2.5 Digital Input Enable and Sleep Modes

As shown in Figure 12-2, the digital input signal can be clamped to ground at the input of the schmitt-trigger. The signal denoted SLEEP in the figure, is set by the MCU Sleep Controller in Power-down mode, Power-save mode, and Standby mode to avoid high power consumption if some input signals are left floating, or have an analog signal level close to $V_{CC}/2$.

SLEEP is overridden for port pins enabled as external interrupt pins. If the external interrupt request is not enabled, SLEEP is active also for these pins. SLEEP is also overridden by various other alternate functions as described in “Alternate Port Functions” on page 72.

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.

12.2.6 Unconnected Pins

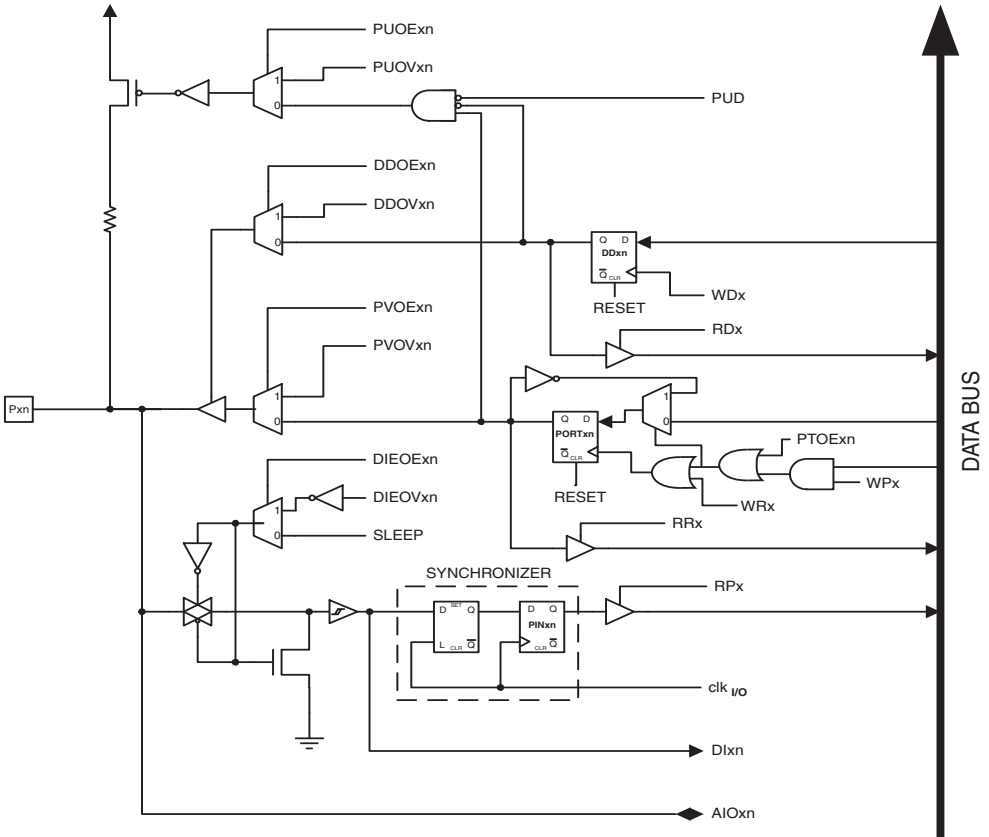
If some pins are unused, it is recommended to ensure that these pins have a defined level. Even though most of the digital inputs are disabled in the deep sleep modes as described above, floating inputs should be avoided to reduce current consumption in all other modes where the digital inputs are enabled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

12.3 Alternate Port Functions

Most port pins have alternate functions in addition to being general digital I/Os. Figure 12-5 shows how the port pin control signals from the simplified Figure 12-2 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 12-5. Alternate Port Functions(1)



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

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

Table 12-2 summarizes the function of the overriding signals. The pin and port indexes from Figure 12-5 are not shown in the succeeding tables. The overriding signals are generated internally in the modules having the alternate function.

Table 12-2. 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.

12.3.1 Alternate Functions of Port B

The Port B pins with alternate functions are shown in Table 12-3.

Table 12-3. Port B Pins Alternate Functions

Port Pin	Alternate Functions
PB7	OC0A/OC1C/PCINT7 (Output Compare and PWM Output A for Timer/Counter0, Output Compare and PWM Output C for Timer/Counter1 or Pin Change Interrupt 7)
PB6	PCINT6 (Pin Change Interrupt 6)
PB5	PCINT5 (Pin Change Interrupt 5)
PB4	T1/PCINT4 (Timer/Counter1 Clock Input or Pin Change Interrupt 4)
PB3	PDO/MISO/PCINT3 (Programming Data Output or SPI Bus Master Input/Slave Output or Pin Change Interrupt 3)
PB2	PDI/MOSI/PCINT2 (Programming Data Input or SPI Bus Master Output/Slave Input or Pin Change Interrupt 2)
PB1	SCLK/PCINT1 (SPI Bus Serial Clock or Pin Change Interrupt 1)
PB0	$\overline{SS}$/PCINT0 (SPI Slave Select input or Pin Change Interrupt 0)

The alternate pin configuration is as follows:

- **OC0A/OC1C/PCINT7, Bit 7**

OC0A, Output Compare Match A output: The PB7 pin can serve as an external output for the Timer/Counter0 Output Compare. The pin has to be configured as an output (DDB7 set “one”) to serve this function. The OC0A pin is also the output pin for the PWM mode timer function.

OC1C, Output Compare Match C output: The PB7 pin can serve as an external output for the Timer/Counter1 Output Compare C. The pin has to be configured as an output (DDB7 set “one”) to serve this function. The OC1C pin is also the output pin for the PWM mode timer function.

PCINT7, Pin Change Interrupt source 7: The PB7 pin can serve as an external interrupt source.

- **PCINT6, Bit 6**

PCINT6, Pin Change Interrupt source 6: The PB6 pin can serve as an external interrupt source.

- **PCINT5, Bit 5**

PCINT5, Pin Change Interrupt source 5: The PB5 pin can serve as an external interrupt source.

- **T1/PCINT4, Bit 4**

T1, Timer/Counter1 counter source.

PCINT4, Pin Change Interrupt source 4: The PB4 pin can serve as an external interrupt source.

- **PDO/MISO/PCINT3 – Port B, Bit 3**

PDO, SPI Serial Programming Data Output. During Serial Program Downloading, this pin is used as data output line for the AT90USB82/162.

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 DDB3. When the SPI is enabled as a slave, the data direction of this pin is controlled by DDB3. When the pin is forced to be an input, the pull-up can still be controlled by the PORTB3 bit.

PCINT3, Pin Change Interrupt source 3: The PB3 pin can serve as an external interrupt source.

- **PDI/MOSI/PCINT2 – Port B, Bit 2**

PDI, SPI Serial Programming Data Input. During Serial Program Downloading, this pin is used as data input line for the AT90USB82/162.

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 DDB2. When the SPI is enabled as a master, the data direction of this pin is controlled by DDB2. When the pin is forced to be an input, the pull-up can still be controlled by the PORTB2 bit.

PCINT2, Pin Change Interrupt source 2: The PB2 pin can serve as an external interrupt source.

- **SCK/PCINT1 – Port B, Bit 1**

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 DDB1. When the SPI0 is enabled as a master, the data direction of this pin is controlled by DDB1. When the pin is forced to be an input, the pull-up can still be controlled by the PORTB1 bit. This pin also serves as Clock for the Serial Programming interface.

PCINT1, Pin Change Interrupt source 1: The PB1 pin can serve as an external interrupt source.

- **$\overline{SS}$/PCINT0 – Port B, Bit 0**

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

PCINT0, Pin Change Interrupt source 0: The PB0 pin can serve as an external interrupt source.

Table 12-4 and Table 12-5 relate the alternate functions of Port B to the overriding signals shown in Figure 12-5 on page 72. SPI MSTR INPUT and SPI SLAVE OUTPUT constitute the MISO signal, while MOSI is divided into SPI MSTR OUTPUT and SPI SLAVE INPUT.

PCINT0, Pin Change Interrupt source 0: The PB0 pin can serve as an external interrupt source

.Table 12-4 and Table 12-5 relate the alternate functions of Port B to the overriding signals shown in Figure 12-5 on page 72. SPI MSTR INPUT and SPI SLAVE OUTPUT constitute the MISO signal, while MOSI is divided into SPI MSTR OUTPUT and SPI SLAVE INPUT..

Table 12-4. Overriding Signals for Alternate Functions in PB7..PB4

Signal Name	PB7/OC0A/OC1C/ PCINT7	PB6/PCINT6	PB5/PCINT5	PB4/T1/PCINT4
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	OC0A/OC1C ENABLE	0	0	0
PVOV	OC0A/OC1C	0	0	0
DIEOE	PCINT7 • PCIE0	PCINT6 • PCIE0	PCINT5 • PCIE0	PCINT4 • PCIE0
DIEOV	1	1	1	1
DI	PCINT7 INPUT	PCINT6 INPUT	PCINT5 INPUT	PCINT4 INPUT T1 INPUT
AIO	–	–	–	–

Table 12-5. Overriding Signals for Alternate Functions in PB3..PB0

Signal Name	PB3/MISO/PCINT3/ PDO	PB2/MOSI/PCINT2/ PDI	PB1/SCK/ PCINT1	PB0/$\overline{SS}$/PCINT0
PUOE	SPE • MSTR	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$
PUOV	PORTB3 • $\overline{PUD}$	PORTB2 • $\overline{PUD}$	PORTB1 • $\overline{PUD}$	PORTB0 • $\overline{PUD}$
DDOE	SPE • MSTR	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$
DDOV	0	0	0	0
PVOE	SPE • $\overline{MSTR}$	SPE • MSTR	SPE • MSTR	0
PVOV	SPI SLAVE OUTPUT	SPI MSTR OUTPUT	SCK OUTPUT	0
DIEOE	PCINT3 • PCIE0	PCINT2 • PCIE0	PCINT1 • PCIE0	PCINT0 • PCIE0
DIEOV	1	1	1	1
DI	SPI MSTR INPUT PCINT3 INPUT	SPI SLAVE INPUT PCINT2 INPUT	SCK INPUT PCINT1 INPUT	SPI $\overline{SS}$ PCINT0 INPUT
AIO	–	–	–	–

12.3.2 Alternate Functions of Port C

The Port C alternate function is as follows:

Table 12-6. Port C Pins Alternate Functions

Port Pin	Alternate Function
PC7	ICP1/INT4/CLKO
PC6	PCINT8/OC1A
PC5	PCINT9/OC1B
PC4	PCINT10
-	-
PC2	PCINT11
PC1	$\overline{\text{Reset}}$, dW
PC0	XTAL2

The alternate pin configuration is as follows:

- **ICP1/INT4/CLK0, Bit 7**

ICP1, Input Capture pin 1 :The PC7 pin can act as an input capture for Timer/Counter1.

INT4, External Interrupt source 4 : The PC7 pin can serve as an external interrupt source to the MCU.

CLK0, Clock Output : The PC7 pin can serve as oscillator clock ouput if the feature is enabled by fuse.

- **PCINT8/OC1A, Bit 6**

PCINT8, Pin Change Interrupt source 8 : The PC6 pin can serve as an external interrupt source.

OC1A, Output Compare Match A output: The PC6 pin can serve as an external output for the Timer/Counter1 Output Compare. The pin has to be configured as an output (DDC6 set “one”) to serve this function. The OC1A pin is also the output pin for the PWM mode timer function.

- **PCINT9/OC1B, Bit 5**

PCINT9, Pin Change Interrupt source 9: The PC5 pin can serve as an external interrupt source.

OC1B, Output Compare Match B output: The PC5 pin can serve as an external output for the Timer/Counter1 Output Compare. The pin has to be configured as an output (DDC5 set “one”) to serve this function. The OC1B pin is also the output pin for the PWM mode timer function.

- **PCINT10, Bit 4**

PCINT10, Pin Change Interrupt source 10 : The PC4 pin can serve as an external interrupt source.

- **PCINT11, Bit 2**

PCINT11, Pin Change Interrupt source 11 : The PC2 pin can serve as an external interrupt source.

- **$\overline{\text{Reset}}$/dW, Bit 1**

$\overline{\text{Reset}}$, Reset input. External Reset input is active low and enabled by unprogramming ("1") the RSTDISBL Fuse. Pullup is activated and output driver and digital input are deactivated when the pin is used as the RESET pin.

dW, debugWire channel. When the debugWIRE Enable (DWEN) Fuse is programmed and Lock bits are unprogrammed, the debugWIRE system within the target device is activated. The RESET port pin is configured as a wired -AND (open-drain) bi-directional I/O pin with pull-up enabled and becomes the communication gateway between the target and the emulator.

- **$\overline{\text{XTAL2}}$, Bit 0**

XTAL2, Oscillator. The PC0 pin can serve as Inverting Output for internal Oscillator amplifier.

Table 12-7 and Table 12-8 relate the alternate functions of Port C to the overriding signals shown in Figure 12-5 on page 72.

Table 12-7. Overriding Signals for Alternate Functions in PC7..PC4

Signal Name	PC7/ICP1/INT4/CLK0	PC6/PCINT8/ OC1A	PC5/PCINT9/ OC1B	PC4/PCINT10
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	OC1A ENABLE	OC1B ENABLE	0
PVOV	0	OC1A	OC1B	0
DIEOE	INT4 ENABLE	PCINT8 ENABLE	PCINT9 ENABLE	PCINT10 ENABLE
DIEOV	1	1	1	1
DI	INT4 INPUT	PCINT8 INPUT	PCINT9 INPUT	PCINT10 INPUT
AIO	–	–	–	–

Table 12-8. Overriding Signals for Alternate Functions in PC2..PC0

Signal Name	PC2/PCINT11	PC1/$\overline{\text{RESET}}$/dW	PC0/XTAL2
PUOE	0	0	0
PUOV	0	0	0
DDOE	0	0	0
DDOV	0	0	0
PVOE	0	0	0
PVOV	0	0	0
DIEOE	PCINT11 ENABLE	0	0
DIEOV	1	0	0
DI	PCINT11 INPUT	–	–
AIO	–	–	–

12.3.3 Alternate Functions of Port D

The Port D pins with alternate functions are shown in Table 12-9.

Table 12-9. Port D Pins Alternate Functions

Port Pin	Alternate Function
PD7	$\overline{\text{HWB}}$/TO/INT7/$\overline{\text{CTS}}$
PD6	INT6/$\overline{\text{RTS}}$
PD5	XCK1/PCINT12 (USART1 External Clock Input/Output)
PD4	INT5
PD3	$\overline{\text{INT3}}$/TXD1 (External Interrupt3 Input or USART1 Transmit Pin)
PD2	$\overline{\text{INT2}}$/AIN1/RXD1(External Interrupt2 Input or USART1 Receive Pin)
PD1	$\overline{\text{INT1}}$/AIN0 (External Interrupt1 Input)
PD0	$\overline{\text{INT0}}$/OC0B (External Interrupt0 Input)

The alternate pin configuration is as follows:

- **$\overline{\text{HWB}}$/TO/INT7/$\overline{\text{CTS}}$, Bit 7**

$\overline{\text{HWB}}$, Hardware Boot : The PD7 pin can serve as

TO, Timer/Counter0 counter source.

INT7, External Interrupt source 7: The PD7 pin can serve as an external interrupt source to the MCU.

$\overline{\text{CTS}}$, USART1 Transmitter Flow Control. This pin can control the transmitter in function of its state.

- **INT6/$\overline{\text{RTS}}$,Bit 6**

INT6, External Interrupt source 6: The PD6 pin can serve as an external interrupt source to the MCU.

$\overline{\text{RTS}}$, USART1 Receiver Flow Control. This pin can control the receiver in function of its state.

- **XCK1/PCINT12, Bit 5**

XCK1, USART1 External Clock : The data direction register DDRD5 controls whether the clock is output (DDRD5 set) or input (DDRD5 cleared). The XCK1 pin is active only when the USART1 operates in Synchronous Mode.

PCINT12, Pin Change Interrupt source 12: The PD5 pin can serve as an external interrupt source.

- **INT5, Bit 4**

INT5, External Interrupt source 5: The PD4 pin can serve as an external interrupt source to the MCU.

- **$\overline{\text{INT3}}$/TXD1, Bit 3**

INT3, External Interrupt source 3: The PD3 pin can serve as an external interrupt source to the MCU.

TXD1, USART1 Transmit Data : When the USART1 Transmitter is enabled, this pin is configured as an ouput regardless of DDRD3.

- **$\overline{\text{INT2}}$/AIN1/RXD1, Bit 2**

INT2, External Interrupt source 2: The PD2 pin can serve as an external interrupt source to the MCU.

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

RXD1, USART1 Receive Data : When the USART1 Receiver is enabled, this pin is configured as an input regardless of DDRD2. When the USART forces this pin to be an input, the pull-up can still be controlled by the PORTD2 bit.

- **$\overline{\text{INT1}}$/AIN0, Bit 1**

INT1, External Interrupt source 1: The PD1 pin can serve as an external interrupt source to the MCU.

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

- **$\overline{\text{INT0}}$/OC0B, Bit 0**

INT0, External Interrupt source 0: The PD0 pin can serve as an external interrupt source to the MCU.

OC0B, Output Compare Match B output: The PD0 pin can serve as an external output for the Timer/Counter0 Output Compare. The pin has to be configured as an output (DDD0 set “one”) to serve this function. The OC0B pin is also the output pin for the PWM mode timer function.

Table 12-10 and Table 12-11 relates the alternate functions of Port D to the overriding signals shown in Figure 12-5 on page 72.

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

Signal Name	PD7/T0/INT7/ HBW/$\overline{\text{CTS}}$	PD6/INT6/ $\overline{\text{RTS}}$	PD5/XCK/PCINT12	PD4/INT5
PUOE	CTS	RTS	0	0
PUOV	PORTD7 • $\overline{\text{PUD}}$	0	0	0
DDOE	CTS	RTS	0	0
DDOV	0	1	0	0
PVOE	0	$\overline{\text{RTS}}$ OUTPUT ENABLE	XCK OUTPUT ENABLE	0
PVOV	0	$\overline{\text{RTS}}$ OUTPUT	XCK1 OUTPUT	0
DIEOE	INT7/$\overline{\text{CTS}}$ ENABLE	INT6 ENABLE	PCINT12 ENABLE	INT5 ENABLE
DIEOV	1	1	1	1
DI	T0 INPUT INT7 INPUT $\overline{\text{CTS}}$ INPUT	INT6 INPUT	XCK INPUT PCINT12 INPUT	INT5 INPUT
AIO	–	–	–	–

Table 12-11. Overriding Signals for Alternate Functions in PD3..PD0[1]

Signal Name	PD3/INT3/TXD1	PD2/INT2/RXD1/ AIN1	PD1/INT1/AIN0	PD0/INT0/OC0B
PUOE	TXEN1	RXEN1	0	0
PUOV	0	PORTD2 • $\overline{\text{PUD}}$	0	0
DDOE	TXEN1	RXEN1	0	0
DDOV	1	0	0	0
PVOE	TXEN1	0	0	OC0B ENABLE
PVOV	TXD1	0	0	OC0B
DIEOE	INT3 ENABLE	INT2 ENABLE AIN1 ENABLE	INT1 ENABLE AIN0 ENABLE	INT0 ENABLE
DIEOV	1	AIN1 ENABLE	AIN0 ENABLE	1
DI	INT3 INPUT	INT2 INPUT/RXD1	INT1 INPUT	INT0 INPUT
AIO	–	AIN1 INPUT	AIN0 INPUT	–

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.

12.4 Register Description for I/O-Ports

12.4.1 MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	JTD	–	–	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R/W	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 4 – PUD: Pull-up Disable**

When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-ups ({DDxn, PORTxn} = 0b01). See “Configuring the Pin” on page 68 for more details about this feature.

12.4.2 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	

12.4.3 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	

12.4.4 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	

12.4.5 PORTC – Port C Data Register

Bit	7	6	5	4	3	2	1	0	
0x08 (0x28)	PORTC7	PORTC6	PORTC5	PORTC4	-	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

12.4.6 DDRC – Port C Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x07 (0x27)	DDC7	DDC6	DDC5	DDC4	-	DDC2	DDC1	DDC0	DDRC
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

12.4.7 PINC – Port C Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x06 (0x26)	PINC7	PINC6	PINC5	PINC4	-	PINC2	PINC1	PINC0	PINC
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

12.4.8 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	

12.4.9 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	

12.4.10 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	

13. External Interrupts

13.1 Overview

The External Interrupts are triggered by the INT[7:0] pin or any of the PCINT[12:0] pins. Observe that, if enabled, the interrupts will trigger even if the INT[7:0] or PCINT[12:0] pins are configured as outputs. This feature provides a way of generating a software interrupt.

The Pin change interrupt PCI0 will trigger if any enabled PCINT[7:0] pin toggles. PCMSK0 Register control which pins contribute to the pin change interrupts. The Pin change interrupt PCI1 will trigger if any enabled PCINT[12:8] pin toggles. PCMSK1 Register control which pins contribute to the pin change interrupts. Pin change interrupts on PCINT[12:0] 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 (INT[3:0]) and EICRB (INT[7:4]). When the external interrupt is enabled and is configured as level triggered, the interrupt will trigger as long as the pin is held low. Note that recognition of falling or rising edge interrupts on INT[7:4] requires the presence of an I/O clock, described in “System Clock and Clock Options” on page 26. Low level interrupts and the edge interrupt on INT[3:0] 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 “System Clock and Clock Options” on page 26.

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)	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
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 – ISC31, ISC30 – ISC00, ISC00: External Interrupt 3:0 Sense Control Bits**

The External Interrupts 3:0 are activated by the external pins INT[3:0] 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. Edges on INT[3:0] are registered asynchronously. Pulses on INT[3:0] pins wider than the minimum pulse width given in “External Interrupts Characteristics” on page 268 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 = 3, 2, 1or 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 EICRB – External Interrupt Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x6A)	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
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 – ISC71, ISC70 - ISC41, ISC40: External Interrupt 7:4 Sense Control Bits**

The External Interrupts [7:4] are activated by the external pins INT[7:4] 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 13-2. The value on the INT[7:4] pins are sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. Observe that CPU clock frequency can be lower than the XTAL frequency if the XTAL divider is enabled. 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.

Table 13-2. Interrupt Sense Control(1)

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Any logical change on INTn generates an interrupt request
1	0	The falling edge between two samples of INTn generates an interrupt request.
1	1	The rising edge between two samples of INTn generates an interrupt request.

Note: 1. n = 7, 6, 5 or 4.
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.

13.2.3 EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	INT7	INT6	INT5	INT4	INT3	INT2	INT1	IINT0	EIMSK
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 – INT[7:0]: External Interrupt Request 7:0 Enable**

When an INT[7:0] 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 Registers – EICRA and EICRB – 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.

13.2.4 EIFR – External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	INTF0	EIFR
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 – INTF[7:0]: External Interrupt Flags 7:0**

When an edge or logic change on the INT[7:0] pin triggers an interrupt request, INTF[7:0] becomes set (one). If the I-bit in SREG and the corresponding interrupt enable bit, INT[7:0] 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 INT[7:0] are configured as level interrupt. Note that when entering sleep mode with the INT[3:0] interrupts disabled, the input buffers on these pins will be disabled. This may cause a logic change in internal signals which will set the INTF[3:0] flags. See “Digital Input Enable and Sleep Modes” on page 71 for more information.

13.2.5 PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	-	-	–	–	–	–	PCIE1	PCIE0	PCICR
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 – PCIE[1:0]: Pin Change Interrupt Enable 1:0**

When the PCIE1/0 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), Pin Change interrupt 1/0 is enabled. Any change on any enabled PCINT[12:8]/[7:0] pin will cause an interrupt. The corresponding interrupt of Pin Change Interrupt Request is executed from the PCI1/0 Interrupt Vector. PCINT[12:8]/[7:0] pins are enabled individually by the PCMSK1/0 Register.

13.2.6 PCIFR – Pin Change Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1B (0x3B)	-	-	–	–	–	–	PCIF1	PCIF0	PCIFR
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 – PCIF[1:0]: Pin Change Interrupt Flag 1:0**

When a logic change on any PCINT[12:8]/[7:0] pin triggers an interrupt request, PCIF1/0 becomes set (one). If the I-bit in SREG and the PCIE1/0 bit in EIMSK are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

13.2.7 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 – PCINT[7:0]: Pin Change Enable Mask 7:0**

Each PCINT[7:0] bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT[7:0] is set and the PCIE0 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT[7:0] 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)	-	-	-	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
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	

- **Bit 4:0 – PCINT[12:8]: Pin Change Enable Mask 12:8**

Each PCINT[12:8] bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT[12:8] is set and the PCIE1 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT[12:8] is cleared, pin change interrupt on the corresponding I/O pin is disabled.

14. Timer/Counter0 and Timer/Counter1 Prescalers

14.1 Overview

Timer/Counter0 and 1 share the same prescaler module, but the Timer/Counters can have different prescaler settings. The description below applies to all Timer/Counters. Tn is used as a general name, n = 0 or 1.

14.2 Internal Clock Source

The Timer/Counter can be clocked directly by the system clock (by setting the CSn[2:0] = 1). This provides the fastest operation, with a maximum Timer/Counter clock frequency equal to system clock frequency ($f_{CLK_I/O}$). Alternatively, one of four taps from the prescaler can be used as a clock source. The prescaled clock has a frequency of either $f_{CLK_I/O}/8$, $f_{CLK_I/O}/64$, $f_{CLK_I/O}/256$, or $f_{CLK_I/O}/1024$.

14.3 Prescaler Reset

The prescaler is free running, i.e., operates independently of the Clock Select logic of the Timer/Counter, and it is shared by the Timer/Counter Tn. Since the prescaler is not affected by the Timer/Counter's clock select, the state of the prescaler will have implications for situations where a prescaled clock is used. One example of prescaling artifacts occurs when the timer is enabled and clocked by the prescaler (6 > CSn[2:0] > 1). The number of system clock cycles from when the timer is enabled to the first count occurs can be from 1 to N+1 system clock cycles, where N equals the prescaler divisor (8, 64, 256, or 1024).

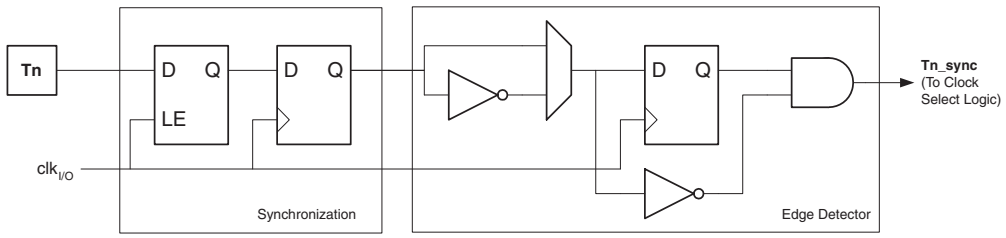
It is possible to use the prescaler reset for synchronizing the Timer/Counter to program execution. However, care must be taken if the other Timer/Counter that shares the same prescaler also uses prescaling. A prescaler reset will affect the prescaler period for all Timer/Counters it is connected to.

14.4 External Clock Source

An external clock source applied to the Tn pin can be used as Timer/Counter clock (clk_{Tn}). The Tn pin is sampled once every system clock cycle by the pin synchronization logic. The synchronized (sampled) signal is then passed through the edge detector. Figure 14-1 shows a functional equivalent block diagram of the Tn synchronization and edge detector logic. The registers are clocked at the positive edge of the internal system clock ($clk_{I/O}$). The latch is transparent in the high period of the internal system clock.

The edge detector generates one clk_{Tn} pulse for each positive (CSn2:0 = 7) or negative (CSn2:0 = 6) edge it detects.

Figure 14-1. Tn/T0 Pin Sampling



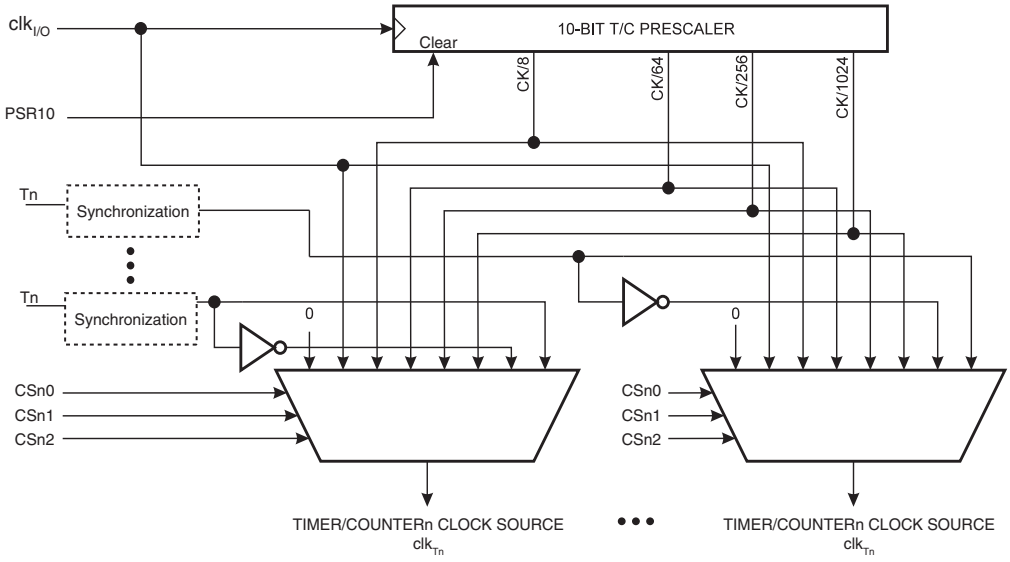
The synchronization and edge detector logic introduces a delay of 2.5 to 3.5 system clock cycles from an edge has been applied to the Tn pin to the counter is updated.

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

Each half period of the external clock applied must be longer than one system clock cycle to ensure correct sampling. The external clock must be guaranteed to have less than half the system clock frequency ($f_{ExtClk} < f_{clk_I/O}/2$) given a 50/50% duty cycle. Since the edge detector uses sampling, the maximum frequency of an external clock it can detect is half the sampling frequency (Nyquist sampling theorem). However, due to variation of the system clock frequency and duty cycle caused by Oscillator source (crystal, resonator, and capacitors) tolerances, it is recommended that maximum frequency of an external clock source is less than $f_{clk_I/O}/2.5$.

An external clock source can not be prescaled.

Figure 14-2. Prescaler for synchronous Timer/Counters



14.5 Register Description

14.5.1 GTCCR – General Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0	
0x23 (0x43)	TSM	–	–	–	–	–	-	PSRSYNC	GTCCR
Read/Write	R/W	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – TSM: Timer/Counter Synchronization Mode**

Writing the TSM bit to one activates the Timer/Counter Synchronization mode. In this mode, the value that is written to the 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.

- **Bits 6:1 – Res: Reserved**

These bits are reserved and will always read as zero.

- **Bit 0 – PSRSYNC: Prescaler Reset for Synchronous Timer/Counters**

When this bit is one, Timer/Counter0 and Timer/Counter1, Timer/Counter3, Timer/Counter4 and Timer/Counter5 prescaler will be Reset. This bit is normally cleared immediately by hardware, except if the TSM bit is set. Note that Timer/Counter0 and Timer/Counter1 share the same prescaler and a reset of this prescaler will affect all timers.

15. 8-bit Timer/Counter0 with PWM

15.1 Features

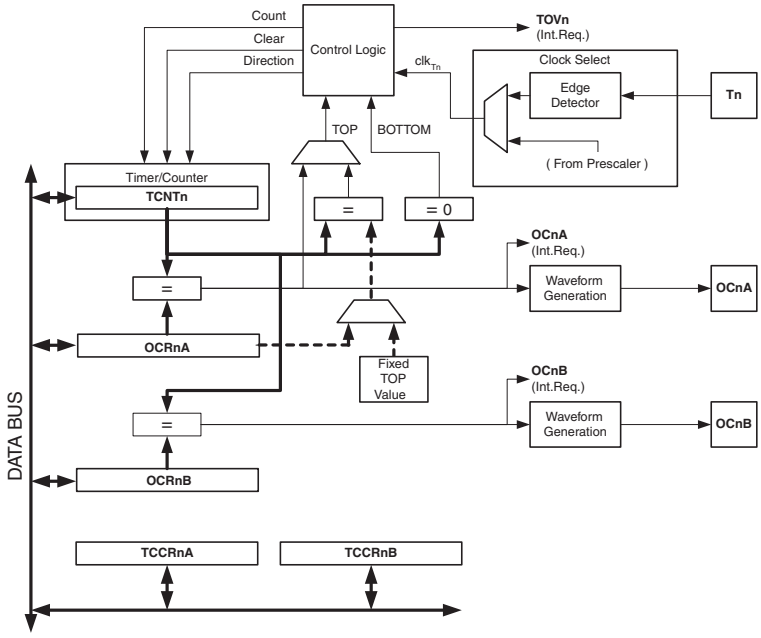
- **Two Independent Output Compare Units**
- **Double Buffered Output Compare Registers**
- **Clear Timer on Compare Match (Auto Reload)**
- **Glitch Free, Phase Correct Pulse Width Modulator (PWM)**
- **Variable PWM Period**
- **Frequency Generator**
- **Three Independent Interrupt Sources (TOV0, OCF0A, 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, refer to “Pinout” on page 2. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the “Register Description” on page 102.

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). See “Output Compare Unit” on page 93. 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

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.

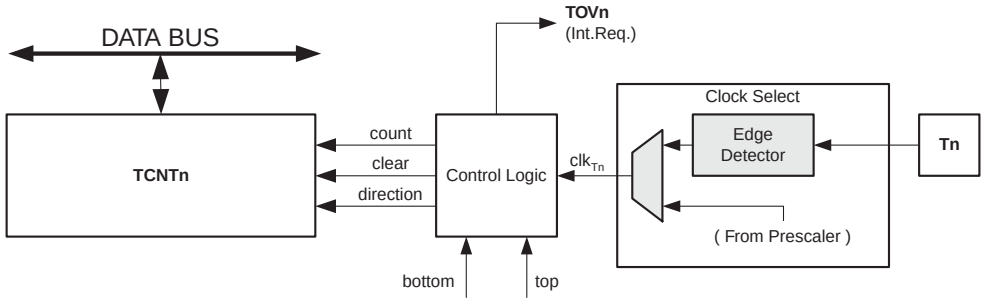
15.3 Timer/Counter Clock Sources

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 “Timer/Counter0 and Timer/Counter1 Prescalers” on page 88.

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 (CS0[2:0]). When no clock source is selected (CS0[2:0] = 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 “Modes of Operation” on page 96.

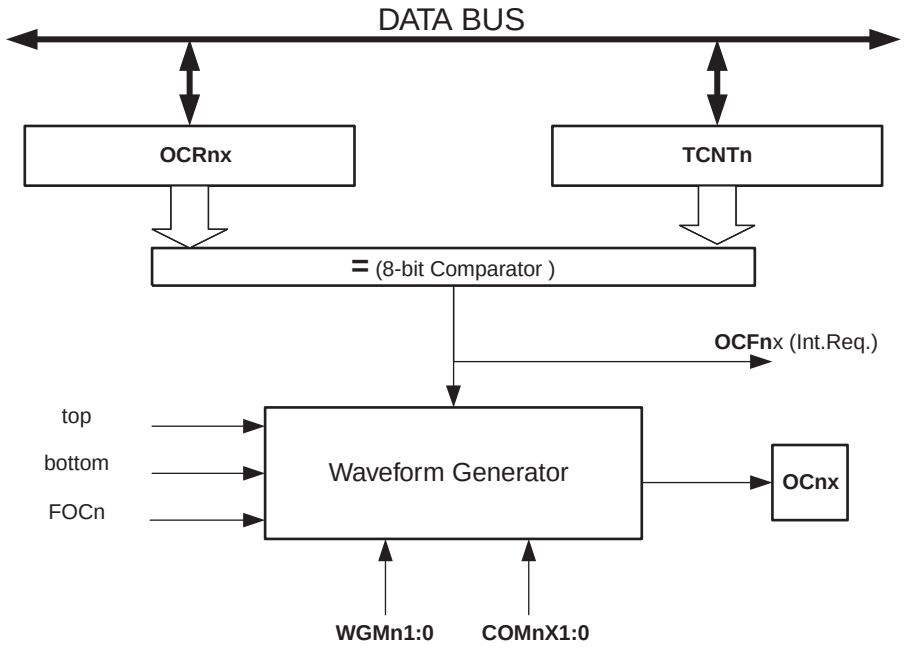
The Timer/Counter Overflow Flag (TOV0) is set according to the mode of operation selected by the WGM0[2:0] bits. TOV0 can be used for generating a CPU interrupt.

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 WGM0[2:0] bits and Compare Output mode (COM0x[1:0]) bits. The max and bottom signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation (“Modes of Operation” on page 96).

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.

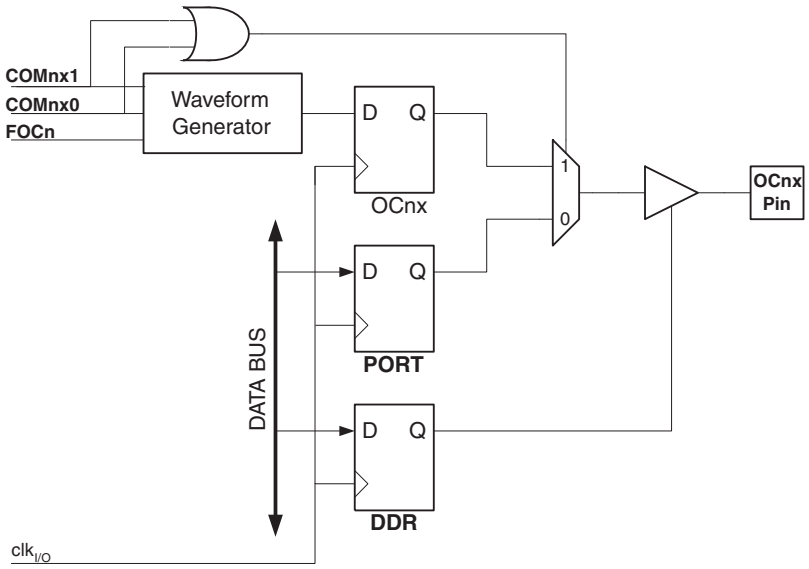
The setup of the OC0x should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OC0x value is to use the Force Output Compare (FOC0x) strobe bits in Normal mode. The OC0x Registers keep their values even when changing between Waveform Generation modes.

Be aware that the COM0x[1:0] bits are not double buffered together with the compare value. Changing the COM0x[1:0] bits will take effect immediately.

15.6 Compare Match Output Unit

The Compare Output mode (COM0x[1:0]) bits have two functions. The Waveform Generator uses the COM0x[1:0] bits for defining the Output Compare (OC0x) state at the next Compare Match. Also, the COM0x[1:0] bits control the OC0x pin output source. Figure 15-4 shows a simplified schematic of the logic affected by the COM0x[1:0] 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 COM0x[1:0] 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 “Register Description” on page 102.

15.6.1 Compare Output Mode and Waveform Generation

The Waveform Generator uses the COM0x[1:0] bits differently in Normal, CTC, and PWM modes. For all modes, setting the COM0x[1:0] = 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 102. For fast PWM mode, refer to Table 15-3 on page 102, and for phase correct PWM refer to Table 15-4 on page 103.

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 (WGM0[2:0]) and Compare Output mode (COM0x[1:0]) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM0x[1:0] bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM0x[1:0] bits control whether the output should be set, cleared, or toggled at a Compare Match (See “Compare Match Output Unit” on page 95.).

For detailed timing information see “Timer/Counter Timing Diagrams” on page 100.

15.7.1 Normal Mode

The simplest mode of operation is the Normal mode (WGM0[2:0] = 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.

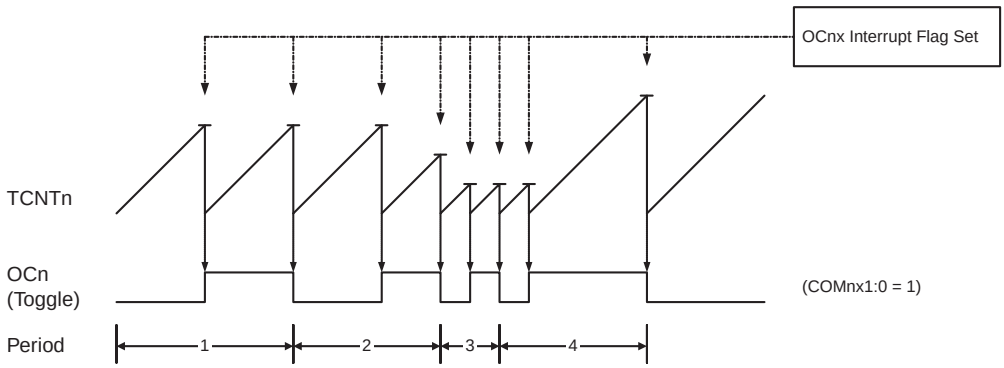
The Output Compare Unit can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

15.7.2 Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC mode (WGM0[2:0] = 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 (COM0A[1:0] = 1). The OC0A value will not be visible on the port pin unless the data direction for the pin is set to output. The waveform generated will have a maximum frequency of $f_{OC0} = f_{clk_I/O}/2$ when OCR0A is set to zero (0x00). The waveform frequency is defined by the following equation:

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

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

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

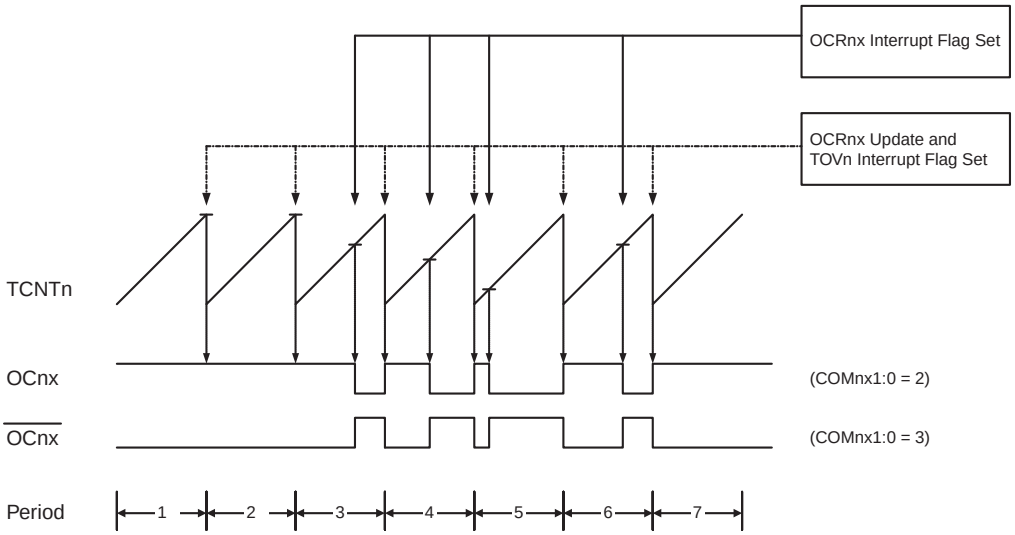
15.7.3 Fast PWM Mode

The fast Pulse Width Modulation or fast PWM mode (WGM0[2:0] = 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 WGM0[2:0] = 3, and OCR0A when WGM0[2:0] = 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 COM0x[1:0] bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM0x[1:0] to three: Setting the COM0A[1:0] 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 102). 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:

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

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 (COM0x[1:0] = 1). The waveform generated will have a maximum frequency of $f_{OC0} = f_{clk_I/O}/2$ when OCR0A is set to zero. This

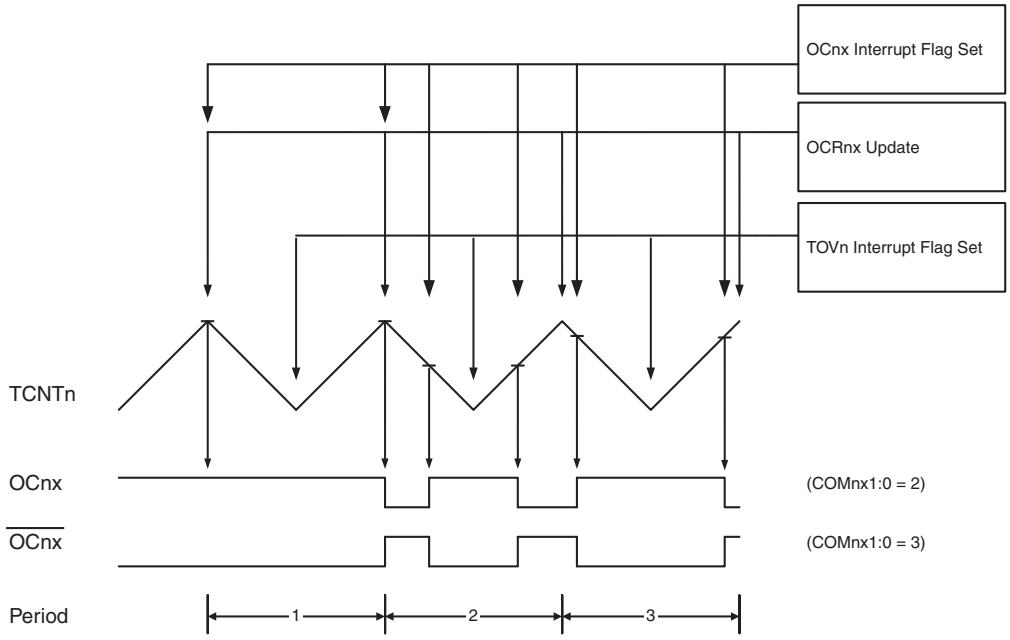
feature is similar to the OC0A toggle in CTC mode, except the double buffer feature of the Output Compare unit is enabled in the fast PWM mode.

15.7.4 Phase Correct PWM Mode

The phase correct PWM mode (WGM0[2:0] = 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 WGM0[2:0] = 1, and OCR0A when WGM0[2:0] = 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 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 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 COM0x[1:0] to three: Setting the COM0A0 bit 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 103). 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:

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

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 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. 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

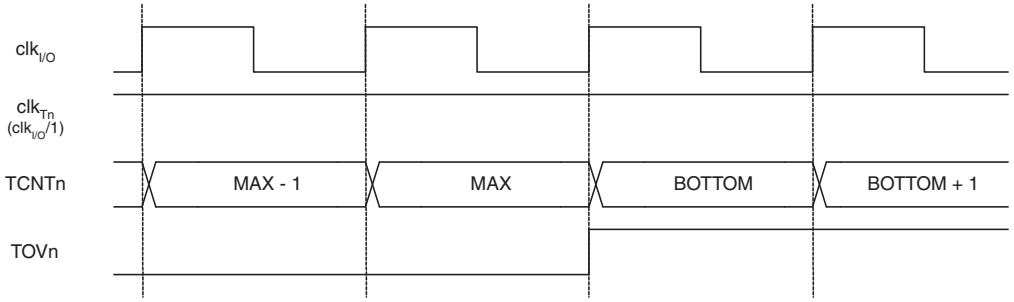


Figure 15-9 shows the same timing data, but with the prescaler enabled.

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

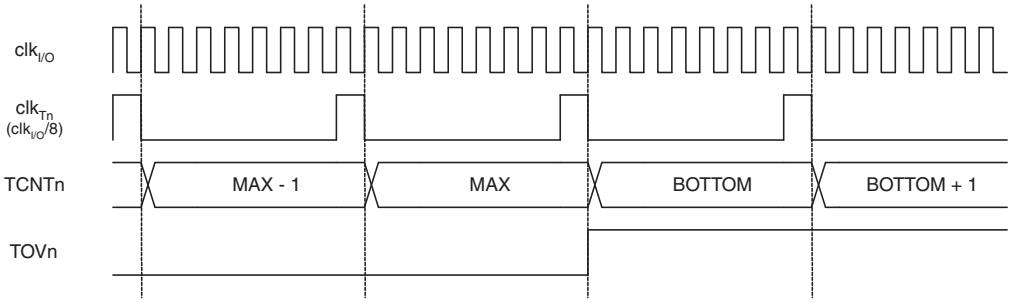


Figure 15-10 shows the setting of OCF0B in all modes and OCF0A in all modes except CTC mode and PWM mode, where OCR0A is TOP.

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

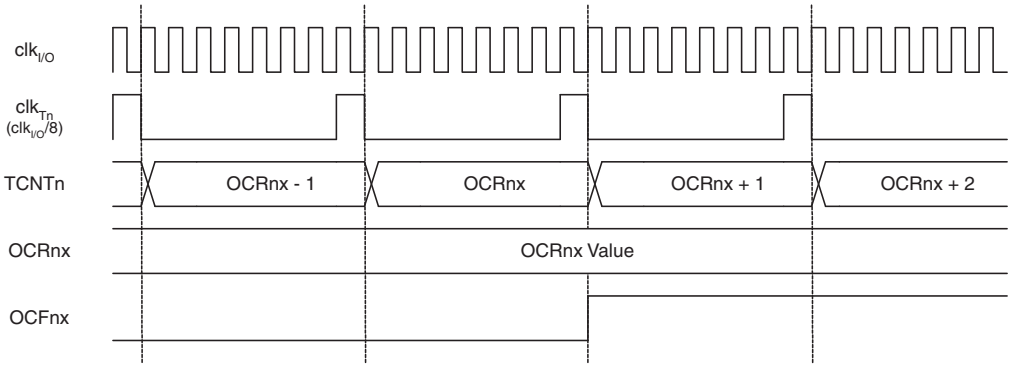
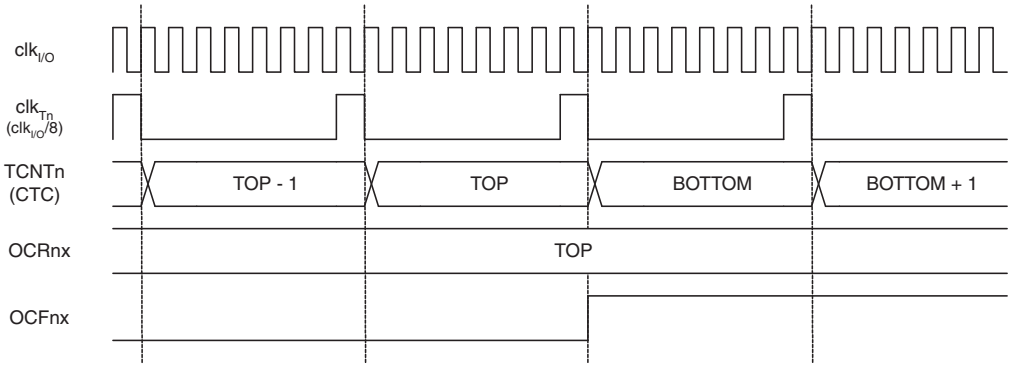


Figure 15-11 shows the setting of OCF0A and the clearing of TCNT0 in CTC mode and fast PWM mode where OCR0A is TOP.

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 – COM0A[1:0]: 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 COM0A[1:0] bits depends on the WGM0[2:0] bit setting. Table 15-2 shows the COM0A[1:0] bit functionality when the WGM0[2:0] 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 COM0A[1:0] bit functionality when the WGM0[1:0] 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 TOP
1	1	Set OC0A on Compare Match, clear OC0A at TOP

Note: 1. A special case occurs when OCR0A equals TOP and COM0A1 is set. In this case, the Compare Match is ignored, but the set or clear is done at TOP. See “Fast PWM Mode” on page 97 for more details.

Table 15-4 shows the COM0A1:0 bit functionality when the WGM0[2:0] 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 "Phase Correct PWM Mode" on page 99 for more details.

- **Bits 5:4 – COM0B[1:0]: Compare Match Output B Mode**

These bits control the Output Compare pin (OC0B) behavior. If one or both of the COM0B[1:0] 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 COM0B[1:0] bits depends on the WGM0[2:0] bit setting. Table 15-2 shows the COM0A[1:0] bit functionality when the WGM0[2:0] 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 15-3 shows the COM0B[1:0] bit functionality when the WGM0[2:0] 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 TOP
1	1	Set OC0B on Compare Match, clear OC0B at TOP

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 "Fast PWM Mode" on page 97 for more details.

Table 15-4 shows the COM0B[1:0] bit functionality when the WGM0[2:0] 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 "Phase Correct PWM Mode" on page 99 for more details.

- **Bits 3:2 – Res: Reserved Bits**

These bits are reserved and will always read as zero.

- **Bits 1:0 – WGM0[1:0]: 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 "Modes of Operation" on page 96).

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	TOP	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	TOP	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 COM0B[1:0] bits setting. Note that the FOC0B bit is implemented as a strobe. Therefore it is the value present in the COM0B[1:0] 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 – Res: Reserved Bits**

These bits are reserved bits and will always read as zero.

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

See the description in the “TCCR0A – Timer/Counter Control Register A” on page 102.

- **Bits 2:0 – CS0[2:0]: 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)

Table 15-9. Clock Select Bit Description (Continued)

CS02	CS01	CS00	Description
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 – Res: Reserved Bits**

These bits are reserved bits 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 – Res: Reserved Bits**

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 WGM0[2:0] bit setting. Refer to Table 15-8, “Waveform Generation Mode Bit Description” on page 104.

16. 16-bit Timer/Counter 1 with PWM

16.1 Features

- **True 16-bit Design (i.e., Allows 16-bit PWM)**
- **Three 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**
- **Five independent interrupt sources (TOV1, OCF1A, OCF1B, OCF1C, ICF1)**

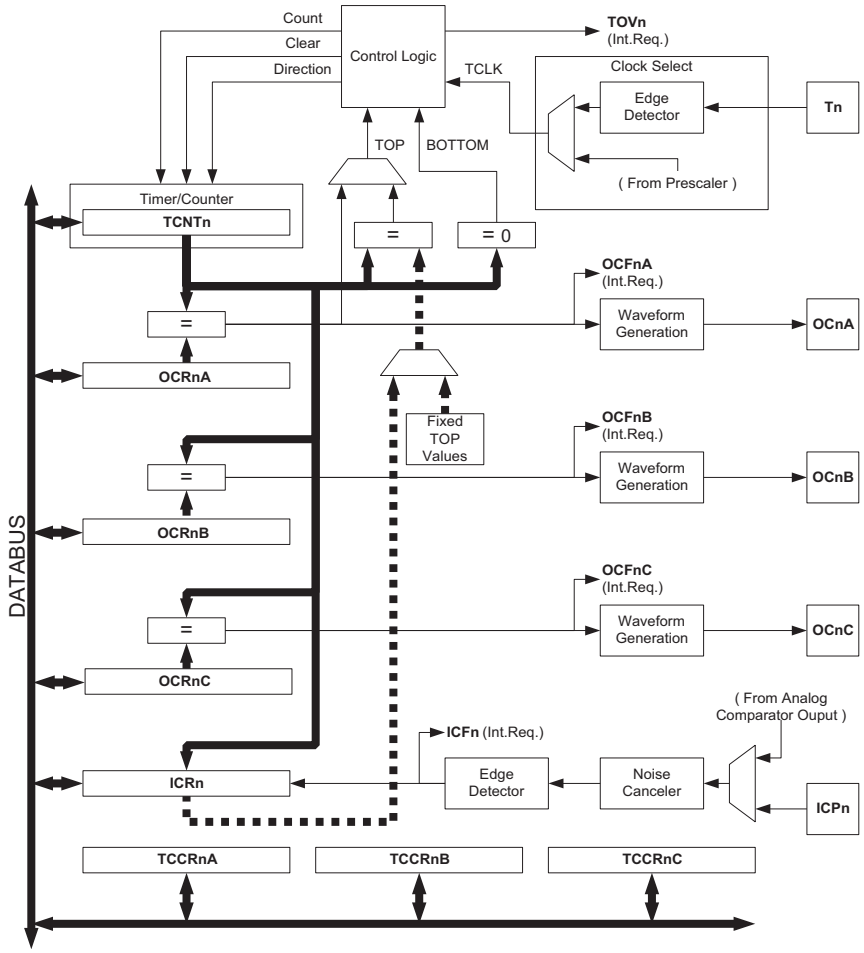
16.2 Overview

The 16-bit Timer/Counter 1 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 (for this product, only n=1 is available), 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. For the actual placement of I/O pins, see “Pinout” on page 2. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the “16-bit Timer/Counter 1 with PWM” on page 108.

The Power Reduction Timer/Counter1 bit, PRTIM1, in “PRR0 – Power Reduction Register 0” on page 46 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 2, Table 12-3 on page 74, and Table 12-6 on page 77 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 "Accessing 16-bit Registers" on page 110. The Timer/Counter Control Registers (TCCRnA/B/C) are 8-bit registers and have no CPU access restrictions. Interrupt requests (shorten as Int.Req.) 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 since these registers are shared by other timer units.

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 (clk_{Tn}).

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).

See “Output Compare Units” on page 117.. 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 (See “Analog Comparator” on page 223.) 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 document:

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. See “Code Examples” on page 6.

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

Note: 1. See “Code Examples” on page 6.

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

Note: 1. See “Code Examples” on page 6.

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.

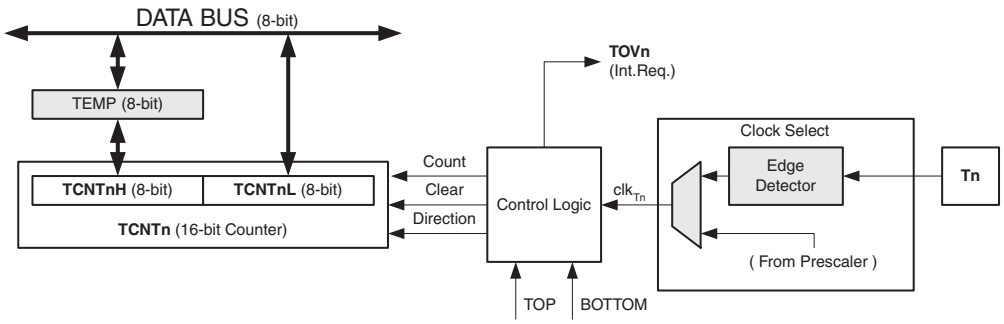
16.4 Timer/Counter Clock Sources

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 “Timer/Counter0 and Timer/Counter1 Prescalers” on page 88.

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 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* (clk_{Tn}). The clk_{Tn} 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 clk_{Tn} 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 “Modes of Operation” on page 120.

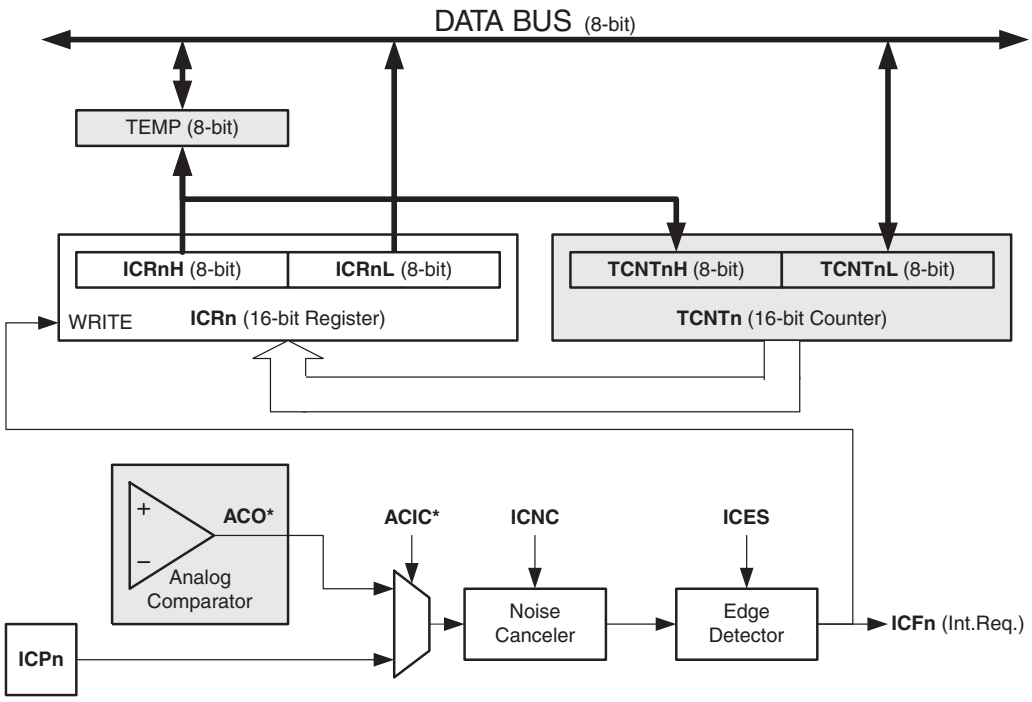
The Timer/Counter Overflow Flag (TOVn) is set according to the mode of operation selected by the WGMn3:0 bits. TOVn can be used for generating a CPU interrupt.

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, for the Timer/Counter1 only, 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. 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



Note: The Analog Comparator Output (ACO) can only trigger the Timer/Counter1 ICP – not Timer/Counter3, 4 or 5.

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 (TICIEn = 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 "Accessing 16-bit Registers" on page 110.

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 14-1 on page 88). 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).

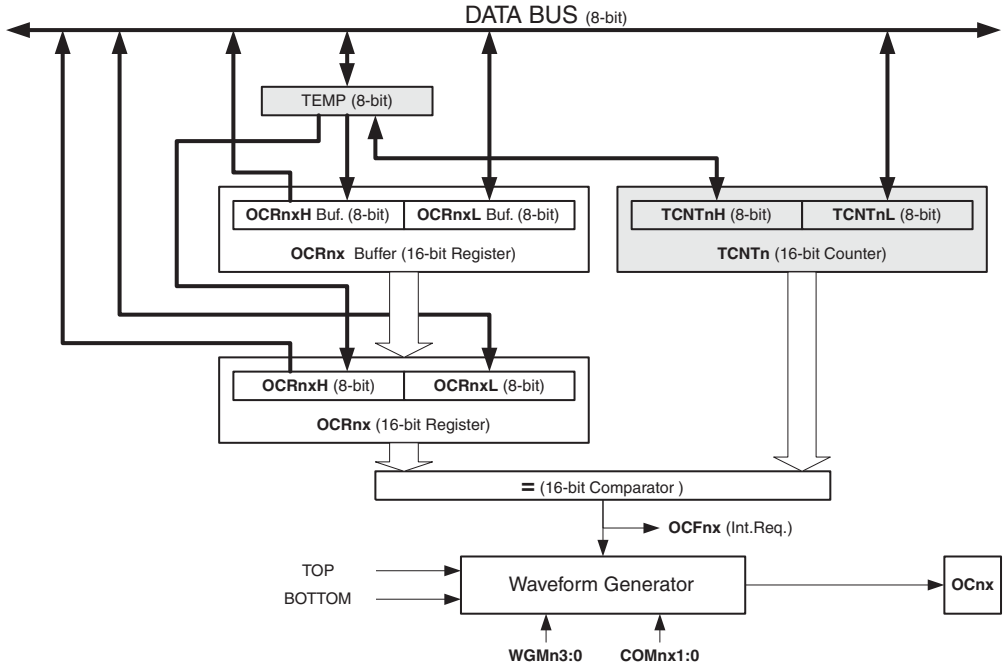
16.7 Output Compare Units

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 (See "Modes of Operation" on page 120.)

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 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 “Accessing 16-bit Registers” on page 110.

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.

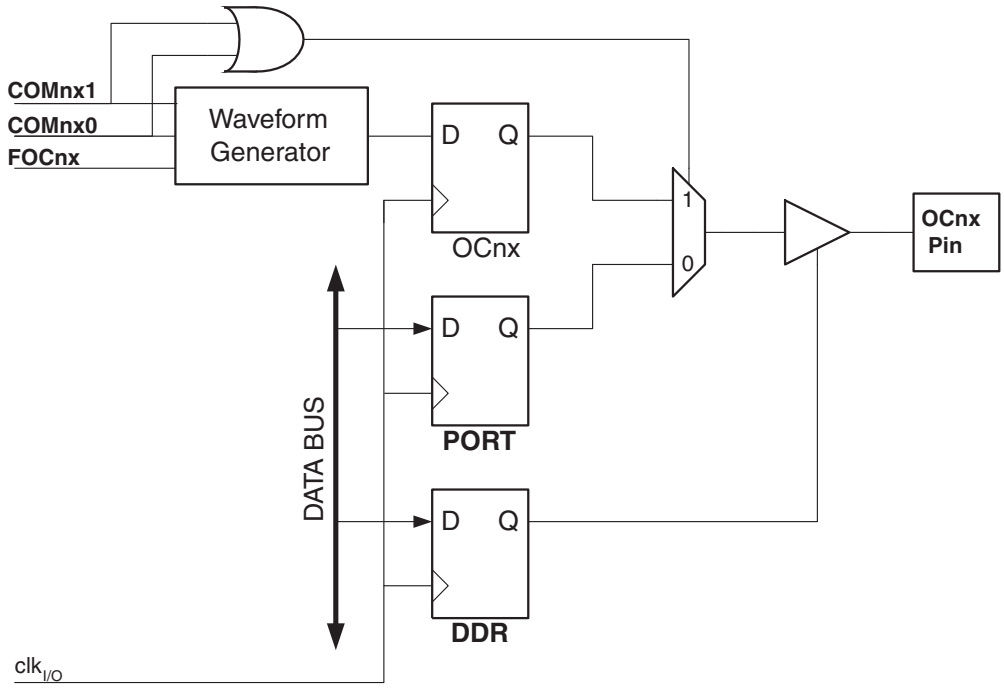
The setup of the OCnx should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OCnx value is to use the Force Output Compare (FOCnx) strobe bits in Normal mode. The OCnx Register keeps its value even when changing between Waveform Generation 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 Compare (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 16-1, Table 16-2 and Table 16-3 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. See “16-bit Timer/Counter 1 with PWM” on page 108.

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 16-1 on page 130. For fast PWM mode refer to Table 16-2 on page 130, and for phase correct and phase and frequency correct PWM refer to Table 16-3 on page 131.

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 (See “Compare Match Output Unit” on page 119.)

For detailed timing information refer to “Timer/Counter Timing Diagrams” on page 127.

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.

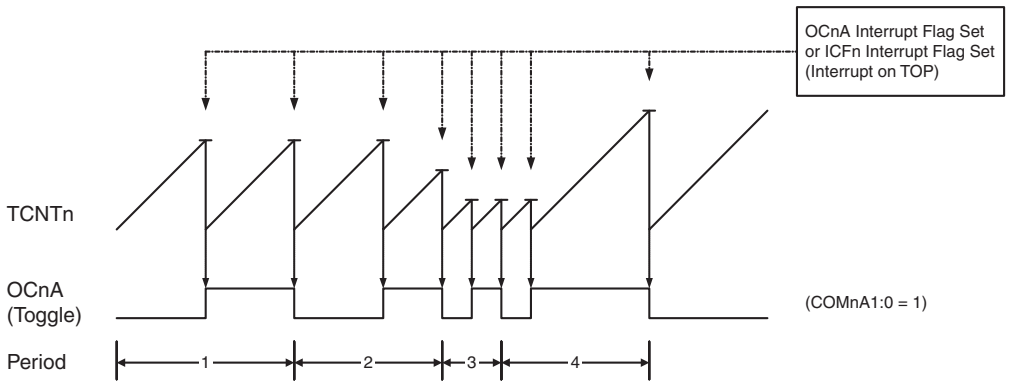
The Output Compare units can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

16.9.2 Clear Timer on Compare Match (CTC) Mode

In *Clear Timer on Compare* or CTC mode (WGMn[3:0] = 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 (WGMn[3:0] = 4) or the ICRn (WGMn[3:0] = 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. 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:

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

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

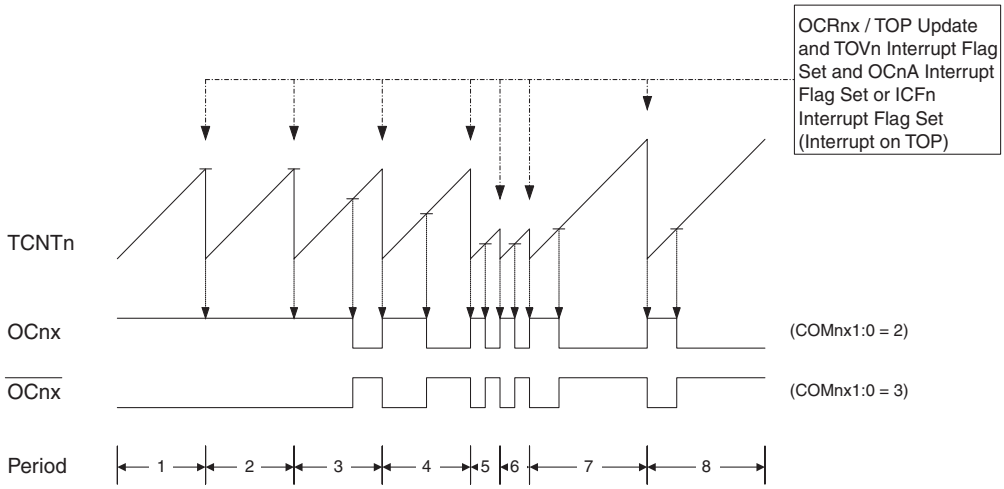
The *fast Pulse Width Modulation* or fast PWM mode (WGMn[3:0] = 5, 6, 7, 14, or 15) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OCnx) is set on the compare match between TCNTn and OCRnx, and cleared at TOP. In inverting Compare Output mode output is cleared on compare match and set at TOP. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct and phase and frequency correct PWM modes that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), hence reduces total system cost.

The PWM resolution for fast PWM can be fixed to 8-, 9-, or 10-bit, or defined by either 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 (WGMn[3:0] = 5, 6, or 7), the value in ICRn (WGMn[3:0] = 14), or the value in OCRnA (WGMn[3:0] = 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 130). 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:

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

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 (WGMn[3:0] = 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.

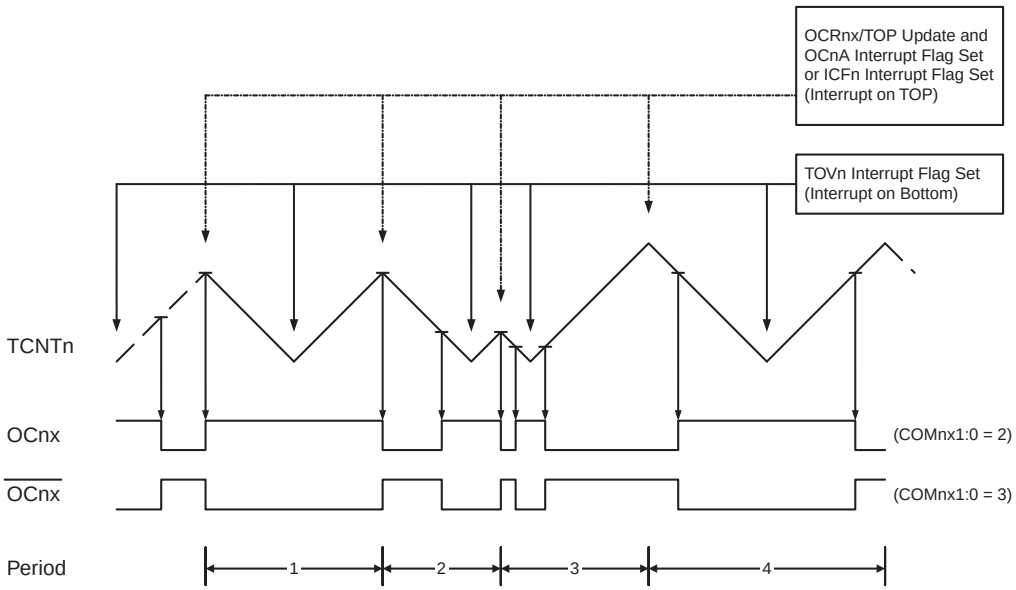
The PWM resolution for the phase correct PWM mode can be fixed to 8-, 9-, or 10-bit, or defined by either ICRn or OCRnA. The minimum resolution allowed is 2-bit (ICRn or OCRnA set to

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 (WGMn[3:0] = 1, 2, or 3), the value in ICRn (WGMn[3:0] = 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. 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 Reg-

ister. 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 COMnx[1:0] bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COMnx[1:0] to three (See Table 16-3 on page 131). 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:

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

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 COM1A[1:0] = 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 (WGMn[3:0] = 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 and Figure 16-9).

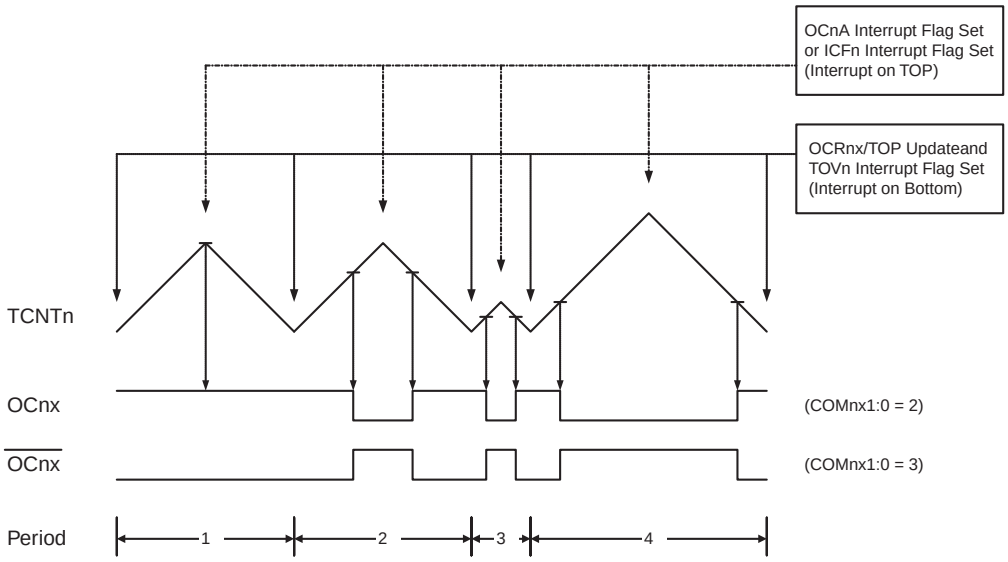
The PWM resolution for the phase and frequency correct PWM mode can be defined by either ICRn or OCRnA. The minimum resolution allowed is 2-bit (ICRn or OCRnA set to 0x0003), and

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 (WGMn[3:0] = 8), or the value in OCRnA (WGMn[3:0] = 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 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 COMnx[1:0] 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 16-3 on page 131). 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:

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

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 (WGM1[3:0] = 9) and COM1A[1:0] = 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 (clk_{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

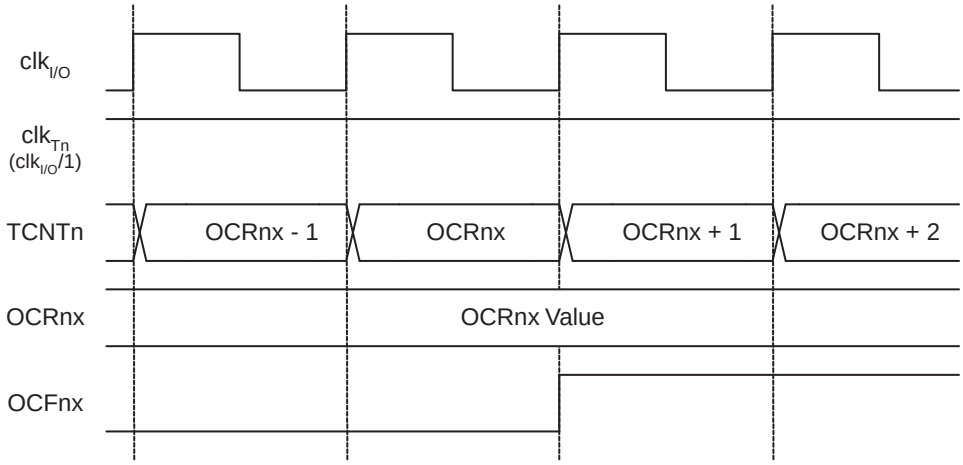


Figure 16-11 shows the same timing data, but with the prescaler enabled.

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

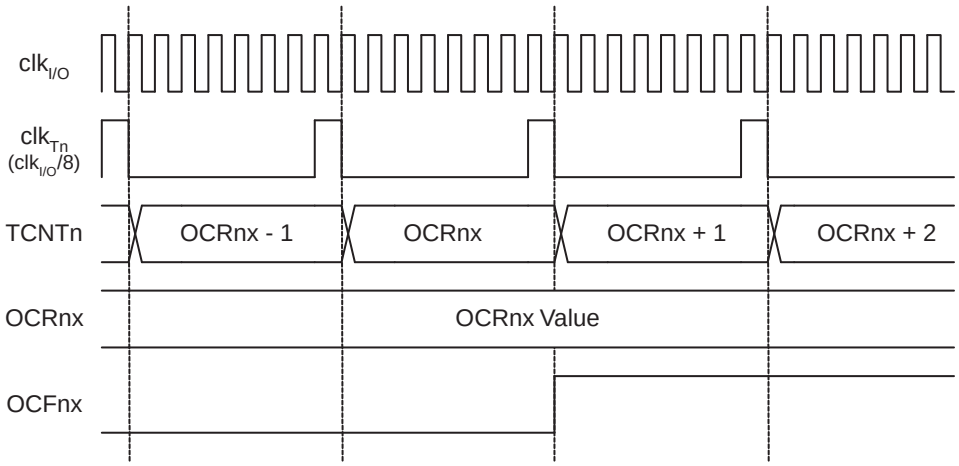


Figure 16-12 shows the count sequence close to TOP in various modes. When using phase and frequency correct PWM mode the OCRnx Register is updated at BOTTOM. The timing diagrams will be the same, but TOP should be replaced by BOTTOM, TOP-1 by BOTTOM+1 and so on. The same renaming applies for modes that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

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

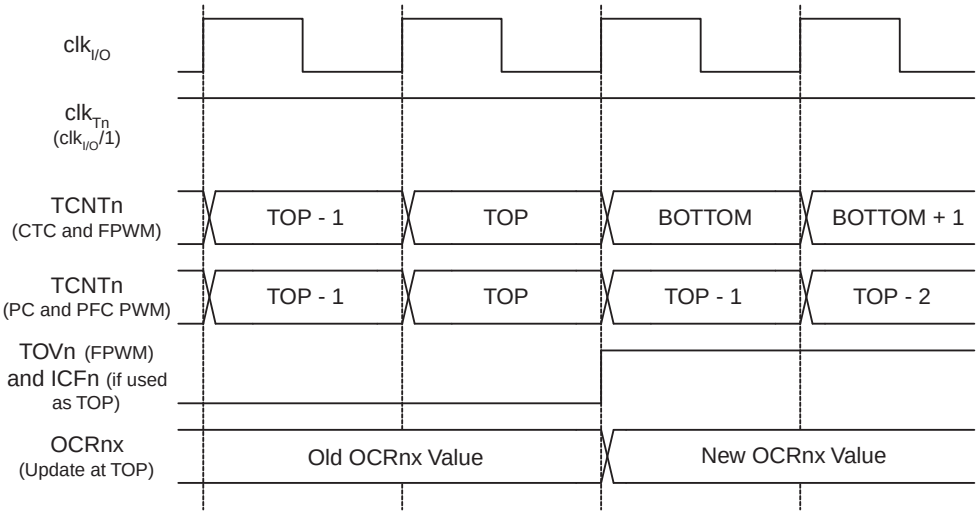
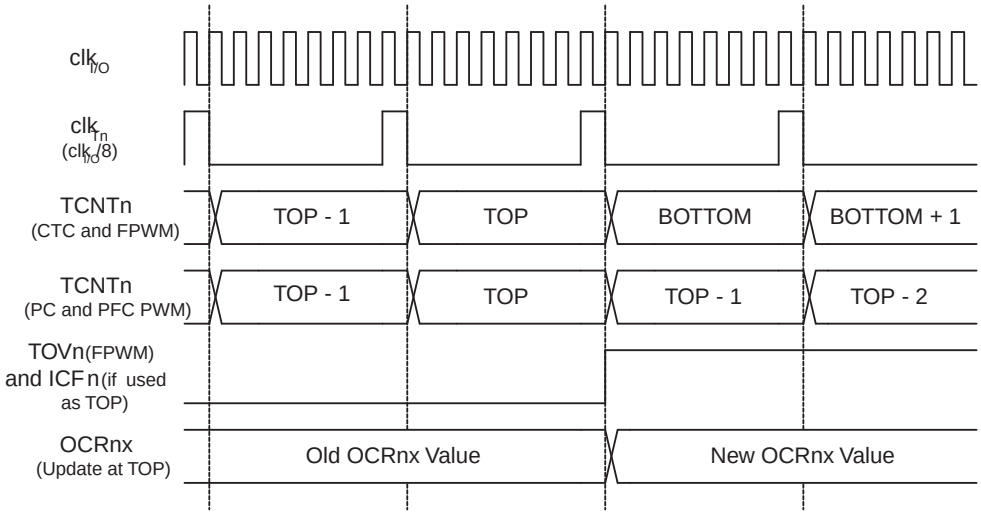


Figure 16-13 shows the same timing data, but with the prescaler enabled.

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



16.11 Register Description

16.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
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 – COMnA1:0: Compare Output Mode for Channel A**
- **Bit 5:4 – COMnB1:0: Compare Output Mode for Channel B**
- **Bit 3:2 – COMnC1:0: Compare Output Mode for Channel C**

The COMnA[1:0], COMnB[1:0], and COMnC[1:0] control the output compare pins (OCnA, OCnB, and OCnC respectively) behavior. If one or both of the COMnA[1:0] 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 COMnB[1:0] bits are written to one, the OCnB output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COMnC[1:0] bits are written to one, the OCnC 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, OCnB or OCnC pin must be set in order to enable the output driver.

When the OCnA, OCnB or OCnC is connected to the pin, the function of the COMnx[1:0] bits is dependent of the WGMn[3:0] bits setting. Table 16-1 shows the COMnx[1:0] bit functionality when the WGMn[3:0] bits are set to a normal or a CTC mode (non-PWM).

Table 16-1. Compare Output Mode, non-PWM

COMnA1/COMnB1/ COMnC1	COMnA0/COMnB0/ COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected.
0	1	Toggle OCnA/OCnB/OCnC on compare match.
1	0	Clear OCnA/OCnB/OCnC on compare match (set output to low level).
1	1	Set OCnA/OCnB/OCnC on compare match (set output to high level).

Table 16-2 shows the COMnx[1:0] bit functionality when the WGMn[3:0] bits are set to the fast PWM mode.

Table 16-2. Compare Output Mode, Fast PWM

COMnA1/COMnB1/ COMnC0	COMnA0/COMnB0/ COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected.
0	1	WGM1[3:0] = 14 or 15: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected.
1	0	Clear OCnA/OCnB/OCnC on compare match, set OCnA/OCnB/OCnC at TOP
1	1	Set OCnA/OCnB/OCnC on compare match, clear OCnA/OCnB/OCnC at TOP

Note: A special case occurs when OCRnA/OCRnB/OCRnC equals TOP and COMnA1/COMnB1/COMnC1 is set. In this case the compare match is ignored, but the set or clear is done at TOP. See “Fast PWM Mode” on page 97. for more details.

Table 16-3 shows the COMnx[1:0] bit functionality when the WGMn[3:0] bits are set to the phase correct and frequency correct PWM mode.

Table 16-3. Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM

COMnA1/COMnB/ COMnC1	COMnA0/COMnB0/ COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected.
0	1	WGM1[3:0] = 8, 9 10 or 11: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected.
1	0	Clear OCnA/OCnB/OCnC on compare match when up-counting. Set OCnA/OCnB/OCnC on compare match when downcounting.
1	1	Set OCnA/OCnB/OCnC on compare match when up-counting. Clear OCnA/OCnB/OCnC on compare match when downcounting.

Note: A special case occurs when OCRnA/OCRnB/OCRnC equals TOP and COMnA1/COMnB1//COMnC1 is set. See “Phase Correct PWM Mode” on page 99. for more details.

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

Combined with the WGMn[3:2] 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 16-4. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare match (CTC) mode, and three types of Pulse Width Modulation (PWM) modes. (See “Modes of Operation” on page 96.).

Table 16-4. 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	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	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	TOP	TOP
15	1	1	1	1	Fast PWM	OCRnA	TOP	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 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – 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 WGMn[3:0] bits located in the TCCRnA and the TCCRnB Register), the ICPn is disconnected and consequently the input capture function is disabled.

- **Bit 5 – Reserved Bit**

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 – WGMn[3:2]: Waveform Generation Mode**

See TCCRnA Register description.

- **Bit 2:0 – CSn[2:0]: Clock Select**

The three clock select bits select the clock source to be used by the Timer/Counter, see Figure 15-1 and Figure 15-2.

Table 16-5. 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 TCCR1C – Timer/Counter1 Control Register C

Bit	7	6	5	4	3	2	1	0	
(0x82)	FOC1A	FOC1B	FOC1C	–	–	–	–	–	TCCR1C
Read/Write	W	W	W	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**
- **Bit 5 – FOCnC: Force Output Compare for Channel C**

The FOCnA/FOCnB/FOCnC bits are only active when the WGMn[3:0] bits specifies a non-PWM mode. When writing a logical one to the FOCnA/FOCnB/FOCnC bit, an immediate compare match is forced on the waveform generation unit. The OCnA/OCnB/OCnC output is changed according to its COMnx[1:0] bits setting. Note that the FOCnA/FOCnB/FOCnC 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/FOCnC 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/FOCnB bits are always read as zero.

- **Bit 4:0 – Res: Reserved Bits**

These bits are reserved for future use. For ensuring compatibility with future devices, these bits must be written to zero when TCCRnC is written.

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 (TCNTnH and TCNTnL, combined TCNTn) give direct access, both for read and for write operations, to the Timer/Counter unit 16-bit counter. To ensure that both the high and low bytes are read and written simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This temporary register is shared by all the other 16-bit registers. See “Accessing 16-bit Registers” on page 110.

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

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

16.11.5 OCR1AH and OCR1AL – Output Compare Register 1 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	

16.11.6 OCR1BH and OCR1BL – Output Compare Register 1 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	

16.11.7 OCR1CH and OCR1CL – Output Compare Register 1 C

Bit	7	6	5	4	3	2	1	0	
(0x8D)	OCR1C[15:8]								OCR1CH
(0x8C)	OCR1C[7:0]								OCR1CL
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 (TCNTn). 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 “Accessing 16-bit Registers” on page 110.

16.11.8 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	

IThe Input Capture is updated with the counter (TCNTn) 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 “Accessing 16-bit Registers” on page 110.

16.11.9 TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	OCIE1C	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 5 – ICIEn: Timer/Countern, 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/Countern Input Capture interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 64.) is executed when the ICFn Flag, located in TIFRn, is set.

Bit 3 – OCIEnC: Timer/Countern, Output Compare C 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/Countern Output Compare C Match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 64.) is executed when the OCFnC Flag, located in TIFRn, is set.

- **Bit 2 – OCIEnB: Timer/Countern, 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/Countern Output Compare B Match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 64.) is executed when the OCFnB Flag, located in TIFRn, is set.

- **Bit 1 – OCIEnA: Timer/Countern, 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/Countern Output Compare A Match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 64.) is executed when the OCFnA Flag, located in TIFRn, is set.

- **Bit 0 – TOIEn: Timer/Countern, 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/Countern Overflow interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 64.) is executed when the TOVn Flag, located in TIFRn, is set.

16.11.10 TIFR1 – Timer/Counter1 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	OCF1C	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 5 – ICFn: Timer/Countern, Input Capture Flag**

This flag is set when a capture event occurs on the ICPn pin. When the Input Capture Register (ICRn) is set by the WGMn[3:0] to be used as the TOP value, the ICFn Flag is set when the counter reaches the TOP value.

ICFn is automatically cleared when the Input Capture Interrupt Vector is executed. Alternatively, ICFn can be cleared by writing a logic one to its bit location.

- **Bit 3 – OCFnC: Timer/Countern, Output Compare C Match Flag**

This flag is set in the timer clock cycle after the counter (TCNTn) value matches the Output Compare Register C (OCRnC).

Note that a Forced Output Compare (FOCnC) strobe will not set the OCFnC Flag.

OCFnC is automatically cleared when the Output Compare Match C Interrupt Vector is executed. Alternatively, OCFnC can be cleared by writing a logic one to its bit location.

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

This flag is set in the timer clock cycle after the counter (TCNTn) value matches the Output Compare Register B (OCRnB).

Note that a Forced Output Compare (FOCnB) strobe will not set the OCFnB Flag.

OCFnB is automatically cleared when the Output Compare Match B Interrupt Vector is executed. Alternatively, OCFnB 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 (TCNTn value matches the Output Compare Register A (OCRnA).

Note that a Forced Output Compare (FOCnA) strobe will not set the OCFnA Flag.

OCFnA is automatically cleared when the Output Compare Match A Interrupt Vector is executed. Alternatively, OCFnA can be cleared by writing a logic one to its bit location.

- **Bit 0 – TOVn: Timer/Countern, Overflow Flag**

The setting of this flag is dependent of the WGMn[3:0] bits setting. In Normal and CTC modes, the TOVn Flag is set when the timer overflows. Refer to Table 16-4 on page 132 for the TOVn Flag behavior when using another WGMn[3:0] bit setting.

TOVn is automatically cleared when the Timer/Countern Overflow Interrupt Vector is executed. Alternatively, TOVn can be cleared by writing a logic one to its bit location.

17. SPI – Serial Peripheral Interface

17.1 Features

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

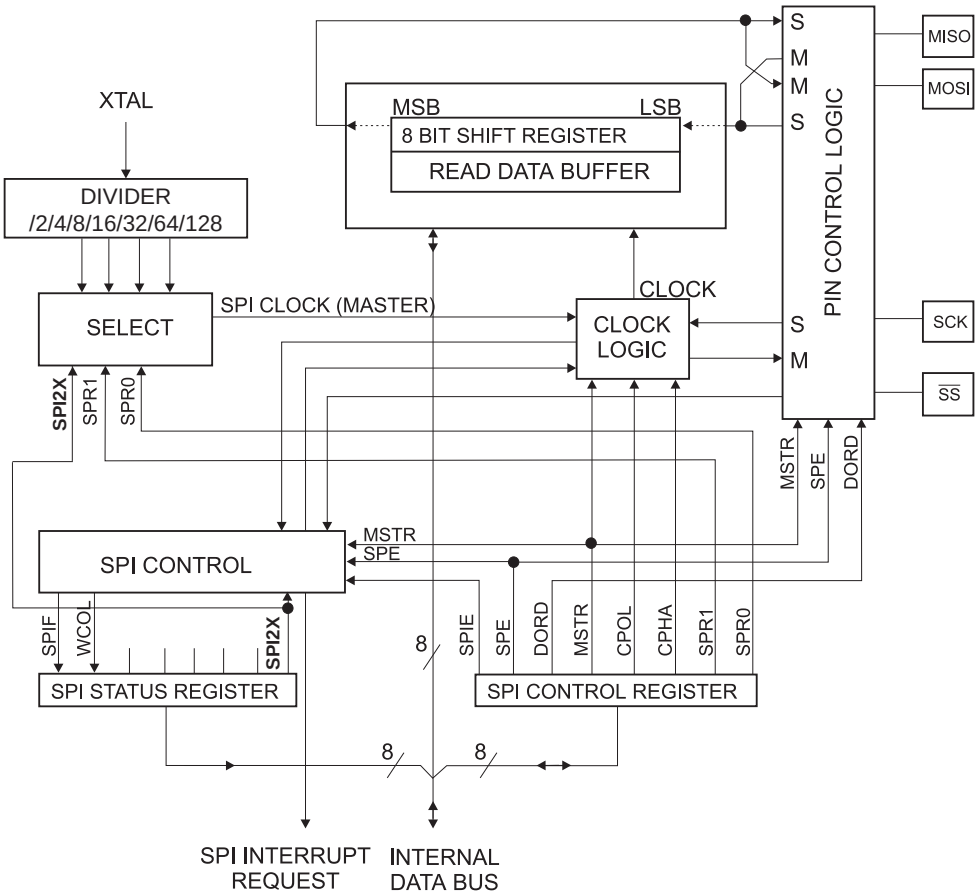
17.2 Overview

The Serial Peripheral Interface (SPI) allows high-speed synchronous data transfer between the ATmega8U2/16U2/32U2 and peripheral devices or between several AVR devices.

USART can also be used in Master SPI mode, see “USART in SPI Mode” on page 176.

The Power Reduction SPI bit, PRSPI, in “Minimizing Power Consumption” on page 44 on page 50 must be written to zero to enable SPI module.

Figure 17-1. SPI Block Diagram[1]



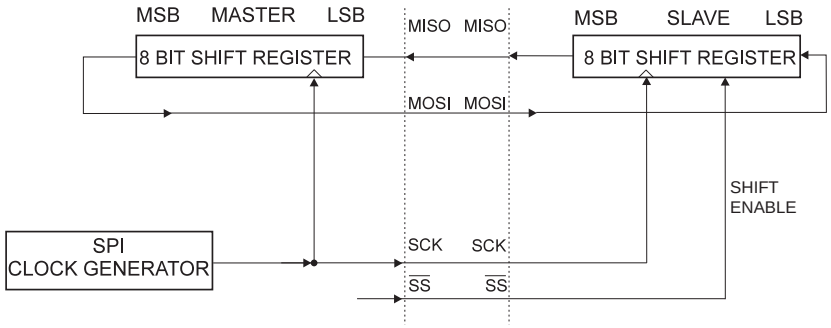
Note: 1. Refer to Figure 1-1 on page 2, and Table 12-6 on page 77 for SPI pin placement.

The interconnection between Master and Slave CPUs with SPI is shown in Figure 17-2. The system consists of two shift Registers, and a Master clock generator. The SPI Master initiates the communication cycle when pulling low the Slave Select $\overline{SS}$ pin of the desired Slave. Master and Slave prepare the data to be sent in their respective shift Registers, and the Master generates the required clock pulses on the SCK line to interchange data. Data is always shifted from Master to Slave on the Master Out – Slave In, MOSI, line, and from Slave to Master on the Master In – Slave Out, MISO, line. After each data packet, the Master will synchronize the Slave by pulling high the Slave Select, $\overline{SS}$, line.

When configured as a Master, the SPI interface has no automatic control of the $\overline{SS}$ line. This must be handled by user software before communication can start. When this is done, writing a byte to the SPI Data Register starts the SPI clock generator, and the hardware shifts the eight bits into the Slave. After shifting one byte, the SPI clock generator stops, setting the end of Transmission Flag (SPIF). If the SPI Interrupt Enable bit (SPIE) in the SPCR Register is set, an interrupt is requested. The Master may continue to shift the next byte by writing it into SPDR, or signal the end of packet by pulling high the Slave Select, $\overline{SS}$ line. The last incoming byte will be kept in the Buffer Register for later use.

When configured as a Slave, the SPI interface will remain sleeping with MISO tri-stated as long as the $\overline{SS}$ pin is driven high. In this state, software may update the contents of the SPI Data Register, SPDR, but the data will not be shifted out by incoming clock pulses on the SCK pin until the $\overline{SS}$ pin is driven low. As one byte has been completely shifted, the end of Transmission Flag, SPIF is set. If the SPI Interrupt Enable bit, SPIE, in the SPCR Register is set, an interrupt is requested. The Slave may continue to place new data to be sent into SPDR before reading the incoming data. The last incoming byte will be kept in the Buffer Register for later use.

Figure 17-2. SPI Master-slave Interconnection



The system is single buffered in the transmit direction and double buffered in the receive direction. This means that bytes to be transmitted cannot be written to the SPI Data Register before the entire shift cycle is completed. When receiving data, however, a received character must be read from the SPI Data Register before the next character has been completely shifted in. Otherwise, the first byte is lost.

In SPI Slave mode, the control logic will sample the incoming signal of the SCK pin. To ensure correct sampling of the clock signal, the frequency of the SPI clock should never exceed $f_{osc}/4$.

When the SPI is enabled, the data direction of the MOSI, MISO, SCK, and $\overline{SS}$ pins is overridden according to Table 17-1. For more details on automatic port overrides, refer to “Alternate Port

Functions” on page 72.

Table 17-1. SPI Pin Overrides(1)

Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User Defined	Input
MISO	Input	User Defined
SCK	User Defined	Input
$\overline{SS}$	User Defined	Input

Note: 1. See “Alternate Functions of Port B” on page 74 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 “Code Examples” on page 6.

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 “Code Examples” on page 6.

17.3 $\overline{SS}$ Pin Functionality

17.3.1 Slave Mode

When the SPI is configured as a Slave, the Slave Select ($\overline{SS}$) pin is always input. When $\overline{SS}$ is held low, the SPI is activated, and MISO becomes an output if configured so by the user. All other pins are inputs. When $\overline{SS}$ is driven high, all pins are inputs, and the SPI is passive, which

means that it will not receive incoming data. Note that the SPI logic will be reset once the $\overline{SS}$ pin is driven high.

The $\overline{SS}$ pin is useful for packet/byte synchronization to keep the slave bit counter synchronous with the master clock generator. When the $\overline{SS}$ pin is driven high, the SPI slave will immediately reset the send and receive logic, and drop any partially received data in the Shift Register.

17.3.2 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.

17.4 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 17-3 and Figure 17-4. 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 17-3 and Table 17-4, as done below:

Table 17-2. CPOL Functionality

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

Figure 17-3. SPI Transfer Format with CPHA = 0

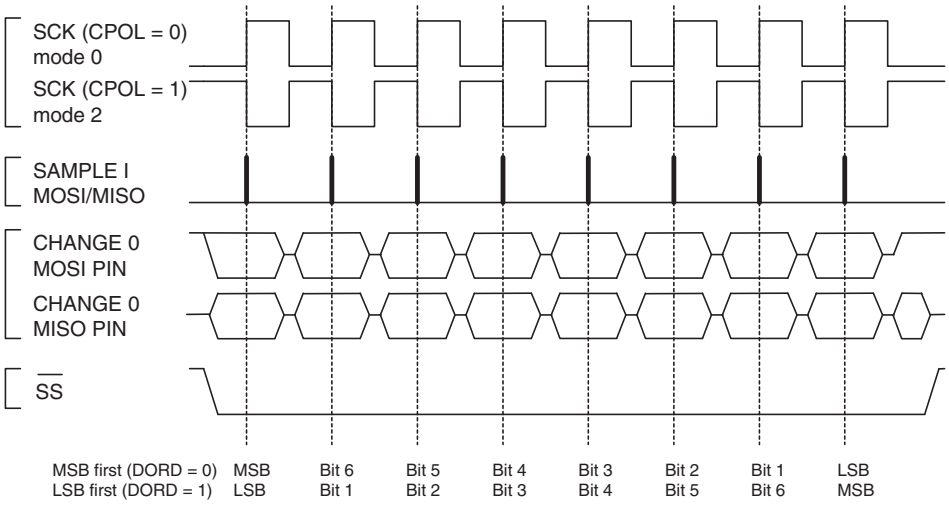
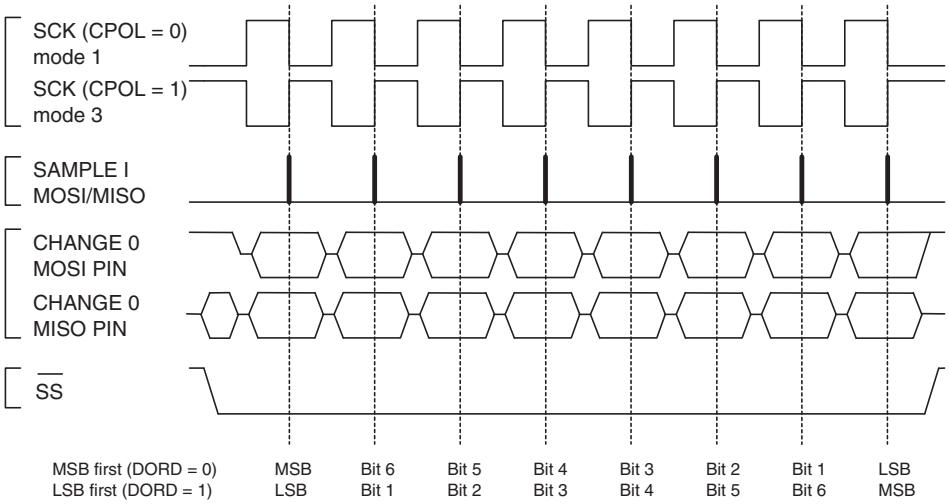


Figure 17-4. SPI Transfer Format with CPHA = 1



17.5 Register Description

17.5.1 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 17-3 and Figure 17-4 for an example. The CPOL functionality is summarized below:

Table 17-3. 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 17-3 and Figure 17-4 for an example. The CPOL functionality is summarized below:

Table 17-4. 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 17-5. 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	$f_{osc}/64$

17.5.2 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 – Res: Reserved Bits**

These bits are reserved bits in the ATmega8U2/16U2/32U2 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 17-5). 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 ATmega8U2/16U2/32U2 is also used for program memory and EEPROM downloading or uploading. See page 259 for serial programming and verification.

17.5.3 SPDR – SPI Data Register

Bit	7	6	5	4	3	2	1	0	
0x2E (0x4E)	MSB	–	–	–	–	–	–	LSB	SPDR
Read/Write	R/W	R	R	R	R	R	R	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.

18. USART

18.1 Features

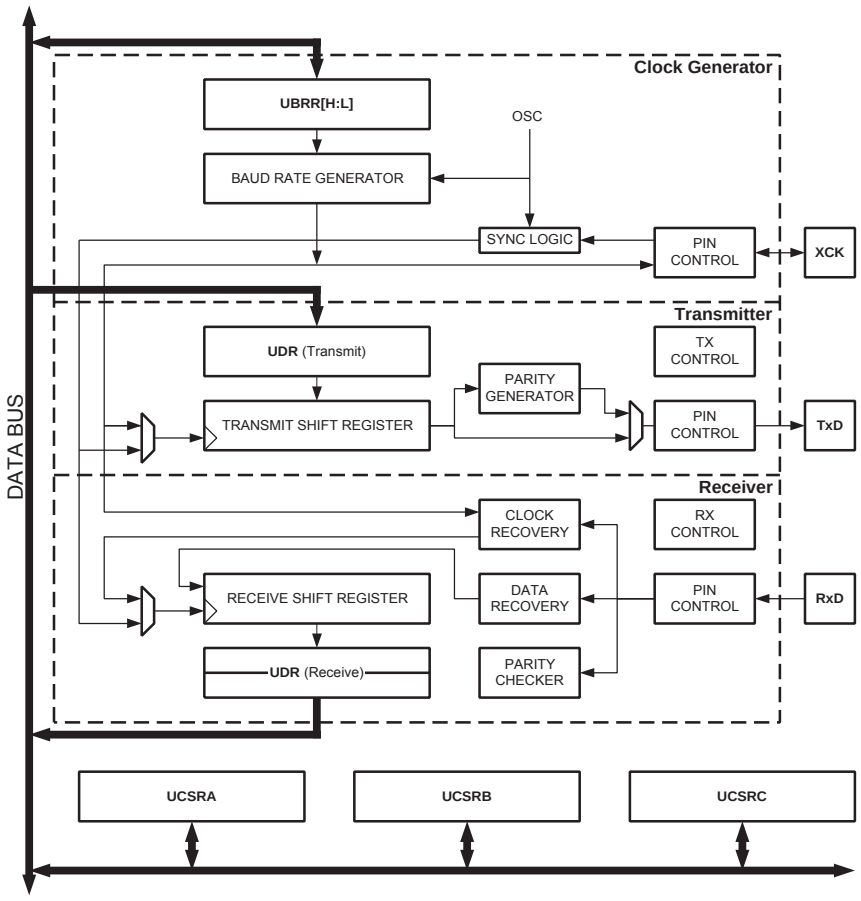
- **Full Duplex Operation (Independent Serial Receive and Transmit Registers)**
- **Asynchronous or Synchronous Operation**
- **Flow control CTS/RTS signals hardware management**
- **Master or Slave Clocked Synchronous Operation**
- **High Resolution Baud Rate Generator**
- **Supports Serial Frames with 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**

18.2 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 18-1 on page 149. CPU accessible I/O Registers and I/O pins are shown in bold.

Figure 18-1. USART Block Diagram[1]



Note: 1. See Figure 1-1 on page 2, Table 12-9 on page 79 and 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.

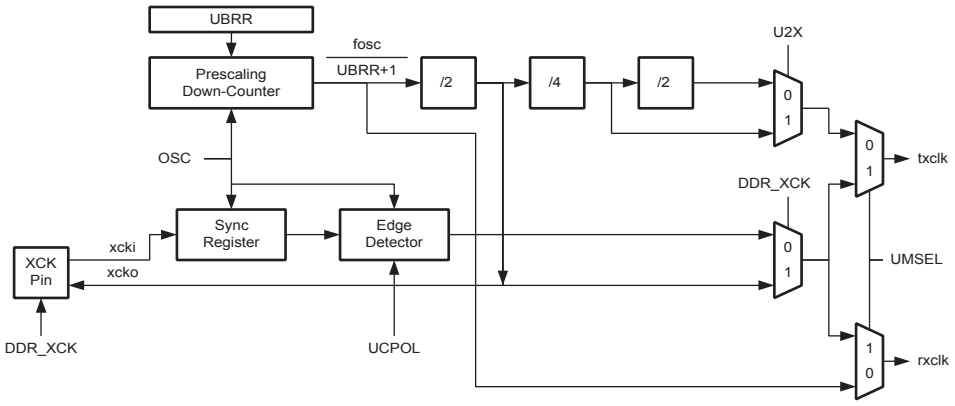
18.3 Clock Generation

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 18-2 shows a block diagram of the clock generation logic.

Figure 18-2. 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).

18.3.1 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 18-2.

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 UBRRLn 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 18-1 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 18-1. 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)	$BAUD = \frac{f_{OSC}}{16(\boldsymbol{UBRR}n + 1)}$	$\boldsymbol{UBRR}n = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed mode (U2Xn = 1)	$BAUD = \frac{f_{OSC}}{8(\boldsymbol{UBRR}n + 1)}$	$\boldsymbol{UBRR}n = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master mode	$BAUD = \frac{f_{OSC}}{2(\boldsymbol{UBRR}n + 1)}$	$\boldsymbol{UBRR}n = \frac{f_{OSC}}{2BAUD} - 1$

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

BAUD Baud rate (in bits per second, bps)

f_{osc} System Oscillator clock frequency

UBRRn Contents of the UBRRHn and UBRRLn Registers, (0-4095)

Some examples of UBRRn values for some system clock frequencies are found in Table 18-9 on page 172.

18.3.2 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.

18.3.3 External Clock

External clocking is used by the synchronous slave modes of operation. The description in this section refers to Figure 18-2 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:

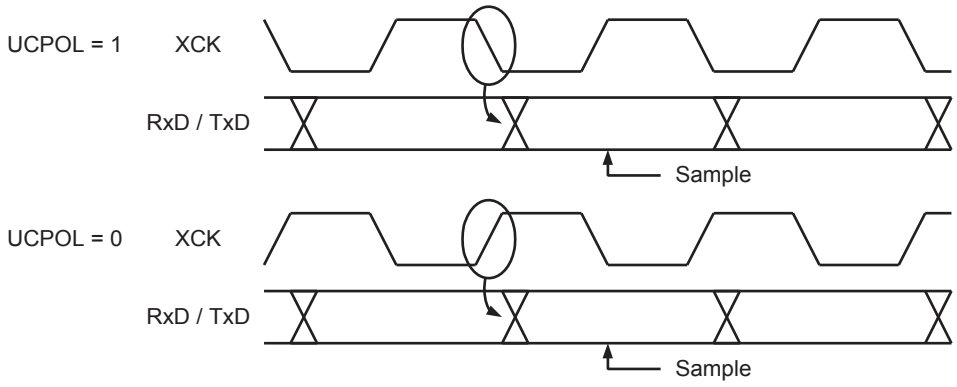
$$f_{XCK} < \frac{f_{OSC}}{4}$$

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.

18.3.4 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 18-3. 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 18-3 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.

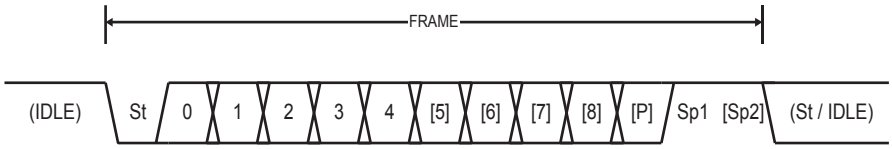
18.4 Frame Formats

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 18-4 illustrates the possible combinations of the frame formats. Bits inside brackets are optional.

Figure 18-4. 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.

18.4.1 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::

$$P_{even} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

P_{even} Parity bit using even parity

P^{odd} Parity bit using odd parity

d_n Data bit n of the character

If used, the parity bit is located between the last data bit and first stop bit of a serial frame.

18.5 USART Initialization

The USART has to be initialized before any communication can take place. The initialization process normally consists of setting the baud rate, setting frame format and enabling the Transmitter or the Receiver depending on the usage. For interrupt driven USART operation, the Global Interrupt Flag should be cleared (and interrupts globally disabled) when doing the initialization.

Before doing a re-initialization with changed baud rate or frame format, be sure that there are no ongoing transmissions during the period the registers are changed. The TXCn Flag can be used to check that the Transmitter has completed all transfers, and the 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  UBRRHn, r17
  out  UBRRLn, 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 */
  UBRRHn = (unsigned char)(baud>>8);
  UBRRLn = (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 “Code Examples” on page 6.

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.

18.6 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.

18.6.1 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 "Code Examples" on page 6.

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.

18.6.2 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 “Code Examples” on page 6.

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.

18.6.3 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 Compete Interrupt Enable (TXCIEn) bit in UCSRnB is set, the USART Transmit Complete Interrupt will be executed when the TXCn Flag becomes set (provided that global interrupts are enabled). When the transmit complete interrupt is used, the interrupt handling routine does not have to clear the TXCn Flag, this is done automatically when the interrupt is executed.

18.6.4 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.

18.6.5 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.

18.7 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.

18.7.1 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 “Code Examples” on page 6.

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.

18.7.2 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 “Code Examples” on page 6.

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.

18.7.3 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.

18.7.4 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 “Parity Bit Calculation” on page 153 and “Parity Checker” on page 160.

18.7.5 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.

18.7.6 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

18.7.7 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 “Code Examples” on page 6.

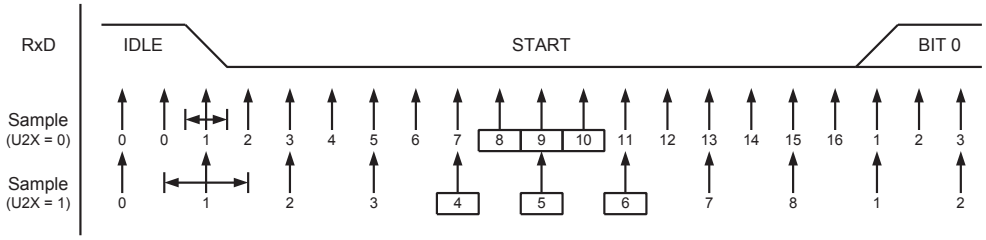
18.8 Asynchronous Data Reception

The USART includes a clock recovery and a data recovery unit for handling asynchronous data reception. The clock recovery logic is used for synchronizing the internally generated baud rate clock to the incoming asynchronous serial frames at the 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.

18.8.1 Asynchronous Clock Recovery

The clock recovery logic synchronizes internal clock to the incoming serial frames. Figure 18-5 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 18-5. Start Bit Sampling

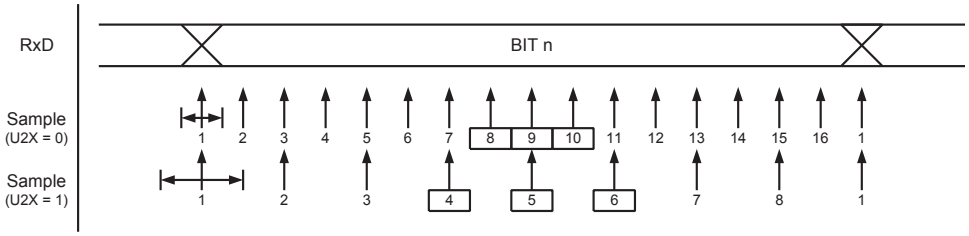


When the clock recovery logic detects a high (idle) to low (start) transition on the RxDn line, the start bit detection sequence is initiated. Let sample 1 denote the first zero-sample as shown in the figure. The clock recovery logic then uses samples 8, 9, and 10 for Normal mode, and samples 4, 5, and 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.

18.8.2 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 18-6 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.

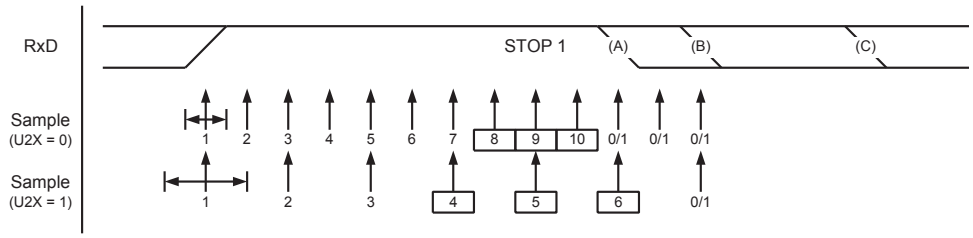
Figure 18-6. Sampling of Data and Parity Bit



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 18-7 shows the sampling of the stop bit and the earliest possible beginning of the start bit of the next frame.

Figure 18-7. 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 18-7. 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.

18.8.3 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 18-2) base frequency, the Receiver will not be able to synchronize the frames to the start bit.

The following equations can be used to calculate the ratio of the incoming data rate and internal receiver baud rate.

$$R_{slow} = \frac{(D+1)S}{S-1+D\cdot S+S_F} \qquad R_{fast} = \frac{(D+2)S}{(D+1)S+S_M}$$

D Sum of character size and parity size (D = 5 to 10 bit)

S Samples per bit. S = 16 for Normal Speed mode and S = 8 for Double Speed mode.

S_F First sample number used for majority voting. S_F = 8 for normal speed and S_F = 4 for Double Speed mode.

S_M Middle sample number used for majority voting. S_M = 9 for normal speed and S_M = 5 for Double Speed mode.

R_{slow} is the ratio of the slowest incoming data rate that can be accepted in relation to the receiver baud rate. R_{fast} is the ratio of the fastest incoming data rate that can be accepted in relation to the receiver baud rate.

Table 18-2 and Table 18-3 list the maximum receiver baud rate error that can be tolerated. Note that Normal Speed mode has higher toleration of baud rate variations.

Table 18-2. 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 18-3. 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.

18.9 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.

18.9.1 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:

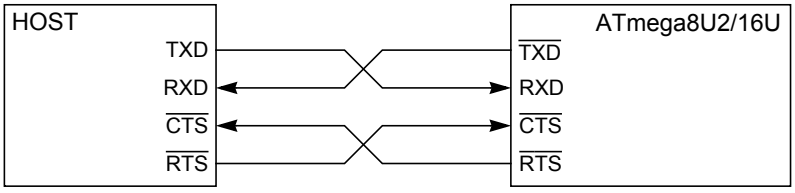
1. All Slave MCUs are in Multi-processor Communication mode (MPCMn in UCSRnA is set).
2. The Master MCU sends an address frame, and all slaves receive and read this frame. In the Slave MCUs, the RXCn Flag in UCSRnA will be set as normal.
3. Each Slave MCU reads the UDRn Register and determines if it 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.

18.10 Hardware Flow Control

The hardware flow control can be enabled by software.
$\overline{CTS}$: (Clear to Send)
$\overline{RTS}$: (Request to Send)



18.10.1 Receiver Flow Control

The reception flow can be controlled by hardware using the $\overline{RTS}$ pin. The aim of the flow control is to inform the external transmitter when the internal receive Fifo is full. Thus the transmitter can

stop sending characters. $\overline{RTS}$ usage and so associated flow control is enabled using RTSEN bit in UCSRnD.
Figure 18-8. shows a reception example.

Figure 18-8. Reception Flow Control Waveform Example

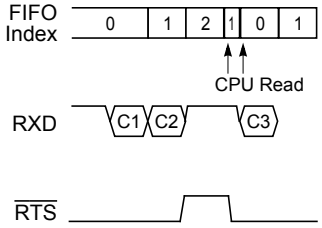
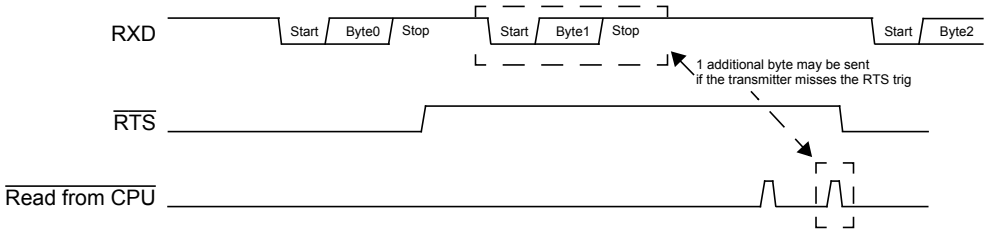


Figure 18-9. $\overline{RTS}$ behavior



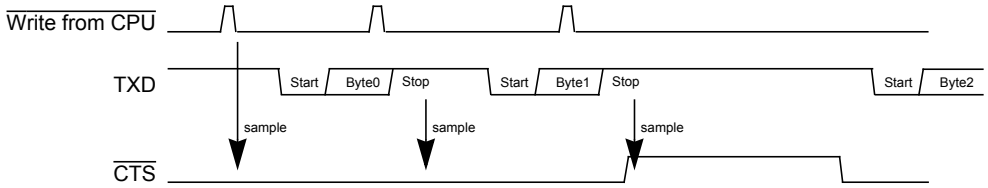
$\overline{RTS}$ will rise at 2/3 of the last received stop bit if the receive fifo is full.

To ensure reliable transmissions, even after a $\overline{RTS}$ rise, an extra-data can still be received and stored in the Receive Shift Register.

18.10.2 Transmission Flow Control

The transmission flow can be controlled by hardware using the $\overline{CTS}$ pin controlled by the external receiver. The aim of the flow control is to stop transmission when the receiver is full of data ($\overline{CTS}$ = 1). $\overline{CTS}$ usage and so associated flow control is enabled using CTSEN bit in UCSRnD. **The $\overline{CTS}$ pin is sampled at each CPU write and at the middle of the last stop bit that is curently being sent.**

Figure 18-10. $\overline{CTS}$ behavior



18.11 Register Description

18.11.1 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.

18.11.2 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 “Multi-processor Communication Mode” on page 164.

18.11.3 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.

18.11.4 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 – UMSELn[1:0] USART Mode Select**

These bits select the mode of operation of the USARTn as shown in Table 18-4..

Table 18-4. 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 “USART in SPI Mode” on page 176 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 18-5. 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 18-6. 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 18-7. 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 18-8. 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

18.11.5 UCSRnD – USART Control and Status Register n D

Bit	7	6	5	4	3	2	1	0	
	-	-	-	-	-	-	CTSEN	RTSEN	UCSRnD
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 1 – CTSEN : USART CTS Enable**

Set this bit to one by firmware to enable the transmission flow control (CTS). Transmission is allowed if $\overline{CTS}$ = 0.
Set this bit to zero by firmware to disable the transmission flow control (CTS). Transmission is always allowed.

- **Bits 0 – RTSEN : USART RTS Enable**

Set this bit to one by firmware to enable the receive flow control (RTS).
Set this bit to zero by firmware to disable the receive flow control (RTS).

18.11.6 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 Bits**

These bits are reserved for future use. For compatibility with future devices, these bit must be written to zero when UBRRH is written.

- **Bit 11:0 – UBRR[11:0]: 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.

18.12 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 18-9 to Table 18-12. UBRR values which yield an actual baud rate differing less than 0.5% from the target baud rate, are bold in the table. Higher error ratings are acceptable, but the Receiver will have less noise resistance when the error ratings are high, especially for large serial frames (see “Asynchronous Operational Range” on page 163). 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 18-9. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	f_{osc} = 1.0000 MHz				f_{osc} = 1.8432 MHz				f_{osc} = 2.0000 MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max. (1)	62.5 kbps		125 kbps		115.2 kbps		230.4 kbps		125 kbps		250 kbps	

1. UBRR = 0, Error = 0.0%

Table 18-10. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 3.6864 MHz				f_{osc} = 4.0000 MHz				f_{osc} = 7.3728 MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	95	0.0%	191	0.0%	103	0.2%	207	0.2%	191	0.0%	383	0.0%
4800	47	0.0%	95	0.0%	51	0.2%	103	0.2%	95	0.0%	191	0.0%
9600	23	0.0%	47	0.0%	25	0.2%	51	0.2%	47	0.0%	95	0.0%
14.4k	15	0.0%	31	0.0%	16	2.1%	34	-0.8%	31	0.0%	63	0.0%
19.2k	11	0.0%	23	0.0%	12	0.2%	25	0.2%	23	0.0%	47	0.0%
28.8k	7	0.0%	15	0.0%	8	-3.5%	16	2.1%	15	0.0%	31	0.0%
38.4k	5	0.0%	11	0.0%	6	-7.0%	12	0.2%	11	0.0%	23	0.0%
57.6k	3	0.0%	7	0.0%	3	8.5%	8	-3.5%	7	0.0%	15	0.0%
76.8k	2	0.0%	5	0.0%	2	8.5%	6	-7.0%	5	0.0%	11	0.0%
115.2k	1	0.0%	3	0.0%	1	8.5%	3	8.5%	3	0.0%	7	0.0%
230.4k	0	0.0%	1	0.0%	0	8.5%	1	8.5%	1	0.0%	3	0.0%
250k	0	-7.8%	1	-7.8%	0	0.0%	1	0.0%	1	-7.8%	3	-7.8%
0.5M	–	–	0	-7.8%	–	–	0	0.0%	0	-7.8%	1	-7.8%
1M	–	–	–	–	–	–	–	–	–	–	0	-7.8%
Max. (1)	230.4 kbps		460.8 kbps		250 kbps		0.5 Mbps		460.8 kbps		921.6 kbps	

1. UBRR = 0, Error = 0.0%

Table 18-11. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 8.0000 MHz				f_{osc} = 11.0592 MHz				f_{osc} = 14.7456 MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	-7.8%	6	5.3%
0.5M	0	0.0%	1	0.0%	–	–	2	-7.8%	1	-7.8%	3	-7.8%
1M	–	–	0	0.0%	–	–	–	–	0	-7.8%	1	-7.8%
Max. (1)	0.5 Mbps		1 Mbps		691.2 kbps		1.3824 Mbps		921.6 kbps		1.8432 Mbps	

1. UBRR = 0, Error = 0.0%

Table 18-12. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 16.0000 MHz				f_{osc} = 18.4320 MHz				f_{osc} = 20.0000 MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250k	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	–	–	4	-7.8%	–	–	4	0.0%
1M	0	0.0%	1	0.0%	–	–	–	–	–	–	–	–
Max. (1)	1 Mbps		2 Mbps		1.152 Mbps		2.304 Mbps		1.25 Mbps		2.5 Mbps	

1. UBRR = 0, Error = 0.0%

19. USART in SPI Mode

19.1 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**

19.2 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.

19.3 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 19-1:

Table 19-1. Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRRn Value
Synchronous Master mode	$BAUD = \frac{f_{OSC}}{2(UBRRn+1)}$	$UBRRn = \frac{f_{OSC}}{2BAUD} - 1$

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

BAUD Baud rate (in bits per second, bps)

f_{osc} System Oscillator clock frequency

UBRRn Contents of the UBRRnH and UBRRnL Registers, (0-4095)

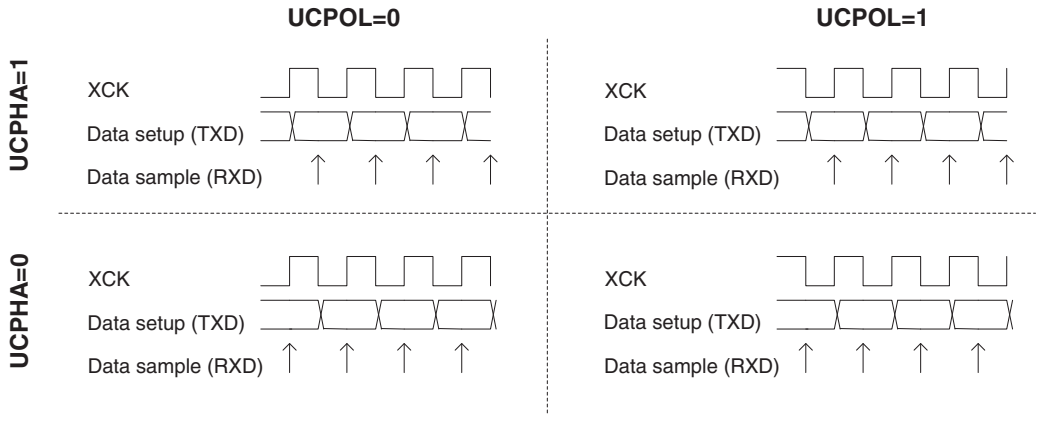
19.4 SPI Data Modes and Timing

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 19-1. 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 19-2. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the Receiver and Transmitter.

Table 19-2. 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 19-1. UCPHAn and UCPOLn data transfer timing diagrams.



19.5 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.

19.5.1 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 “Code Examples” on page 6.

19.6 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 “Code Examples” on page 6.

19.6.1 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.

19.6.2 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.

19.7 Register Description

The following section describes the registers used for SPI operation using the USART.

19.7.1 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 “UDRn – USART I/O Data Register n” on page 167.

19.7.2 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 Bits 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.

19.7.3 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 Bits 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.

19.7.4 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 - UMSELn[1:0]: USART Mode Select**

These bits select the mode of operation of the USART as shown in Table 19-3. See “UCSRnC – USART Control and Status Register n C” on page 169 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 19-3. 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 Bits 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.

19.7.5 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 “UBRRnL and UBRRnH – USART Baud Rate Registers” on page 171.

19.8 AVR USART MSPIM vs. 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 19-4 on page 184.

Table 19-4. 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)	$\overline{SS}$	Not supported by USART in MSPIM

20. USB Controller

20.1 Features

- **USB 2.0 Full-speed device**
- **Ping-pong mode (dual bank), with transparent switch**
- **176 bytes of DPRAM**
  - **1 endpoint of 64 bytes max (default control endpoint)**
  - **2 endpoints of 64 bytes max (one bank)**
  - **2 endpoints of 64 bytes max (one or two banks)**

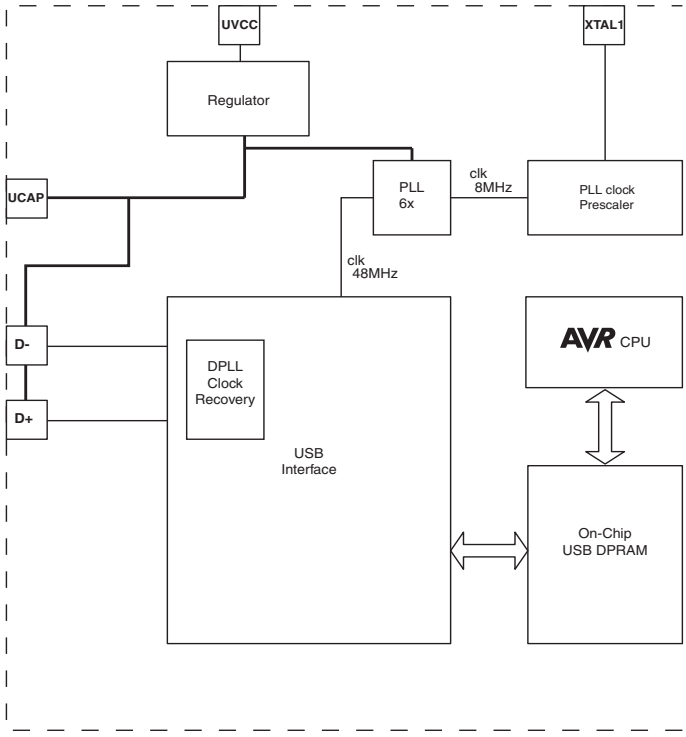
20.2 Overview

The USB controller provides the hardware to implement a USB2.0 compliant Full-Speed USB device in the ATmega8U2/16U2/32U2. A simplified block diagram of the USB controller is shown in Figure 20-1 on page 185.

The USB controller requires a 48 MHz ±0.25% reference clock for USB Full-Speed compliance. This clock is generated by an internal PLL. The reference clock to the PLL must be provided from an external crystal or an external clock input. Only these two clock options will be able to provide a reference clock within the accuracy and jitter requirements of the USB specification. See section "System Clock and Clock Options" on page 26 for details on the ATmega8U2/16U2/32U2 system clock and clock options.

To comply to the USB specifications electrical characteristics, the USB Pads (D+ or D-) must be powered at 3.0V to 3.6V. As the ATmega8U2/16U2/32U2 can be powered up to 5.5V, an internal regulator is provided to correctly power the USB pads. See "USB Module Powering Options" on page 186 for details on the powering options available for the USB controller

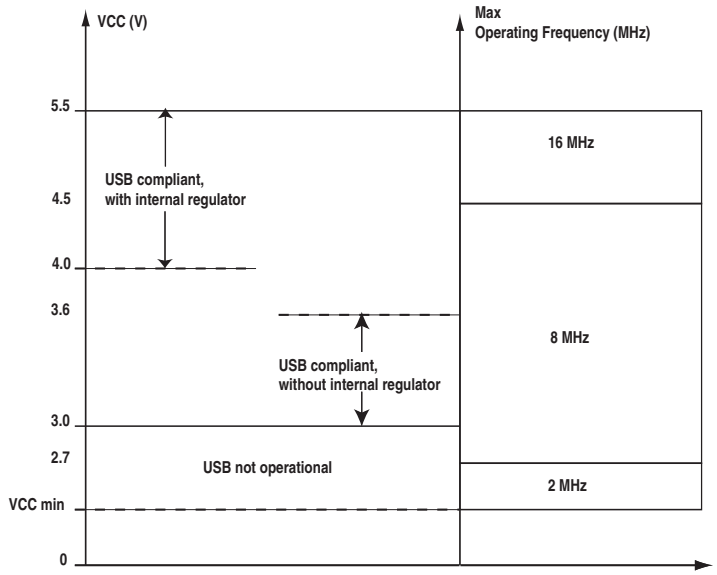
Figure 20-1. USB controller Block Diagram



20.3 USB Module Powering Options

Depending on the selected target application power supply (V_{CC}), the ATmega8U2/16U2/32U2 USB controller requires different powering schemes, see Figure 20-2 on page 186.

Figure 20-2. Operating modes versus frequency and power-supply



20.3.1 Bus Powered device

The following figures show typical implementations for different powering schemes.

Figure 20-3. Typical Bus powered application with 5V I/O

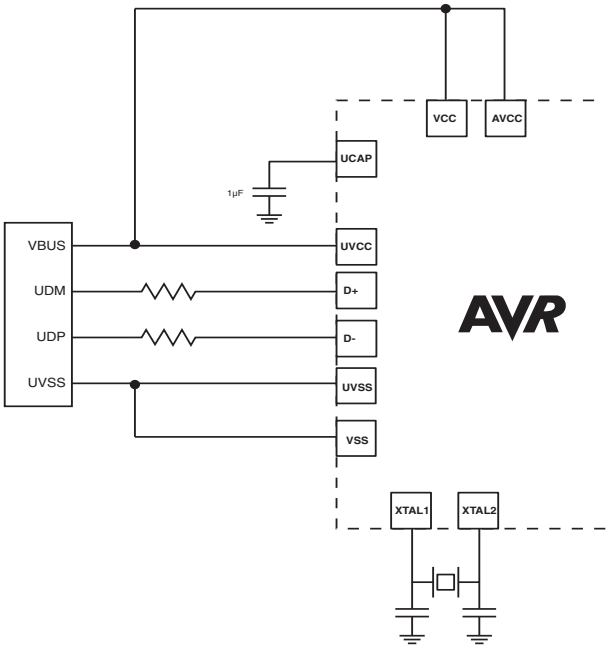
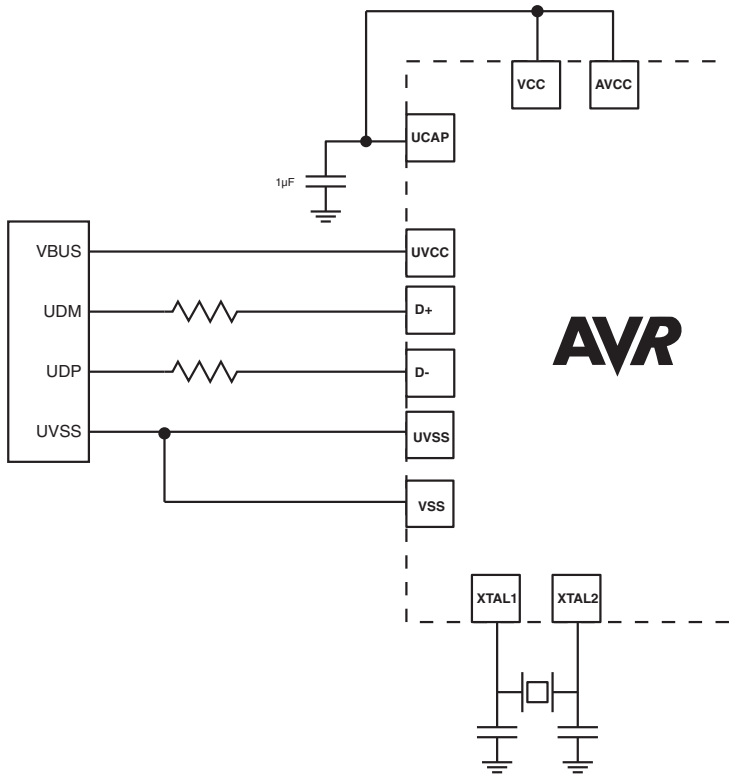


Figure 20-4. Typical Bus powered application with 3.3V I/O



20.3.2 Self Powered device

Figure 20-5. Typical Self powered application with 4.0V to 5.5V I/O.

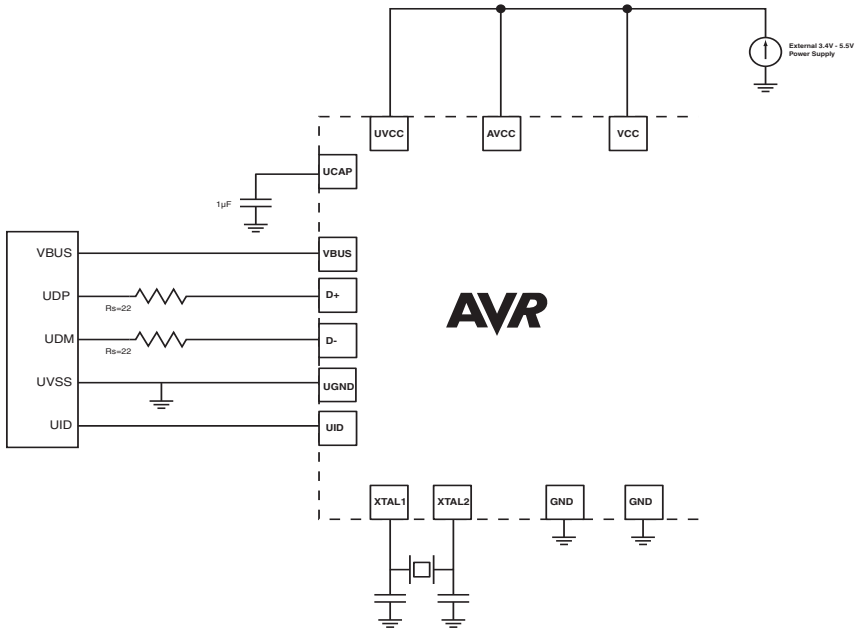
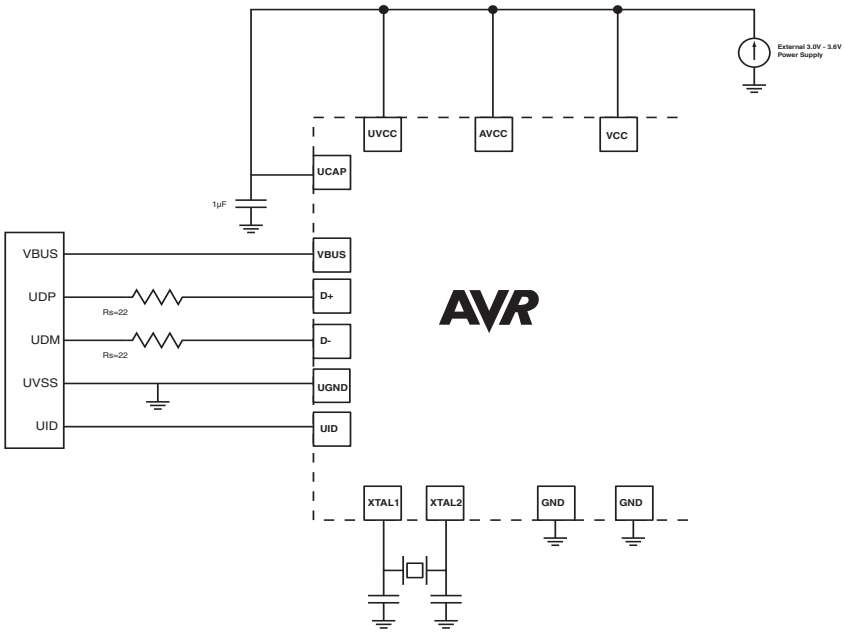


Figure 20-6. Typical Self powered application with 3.0V to 3.6 I/O[1]



Note: 1. The internal 3.3V regulator is bypassed. Disable the regulator to avoid additional power consumption. See the "REGCR – Regulator Control Register" on page 196 for details.

20.3.3 Design guidelines

The following design guidelines should be met:

- Serial resistors on USB Data lines must have 22 Ohms value (+/- 5%).
- Traces from the input USB receptacle (or from the cable connection in the case of a tethered device) to the USB microcontroller pads should be as short as possible, and follow differential traces routing rules (same length, as near as possible and avoid vias accumulation).
- Voltage transient / ESD suppressors may also be used to prevent USB pads to be damaged by external disturbances.
- U_{cap} capacitor should be 1µF (+/- 10%) for correct operation.

In addition it is highly recommended to connect a 10µF capacitor to the VBUS line

20.4 General Operating Modes

20.4.1 Introduction

The USB controller is disabled and reset after a hardware reset generated by:

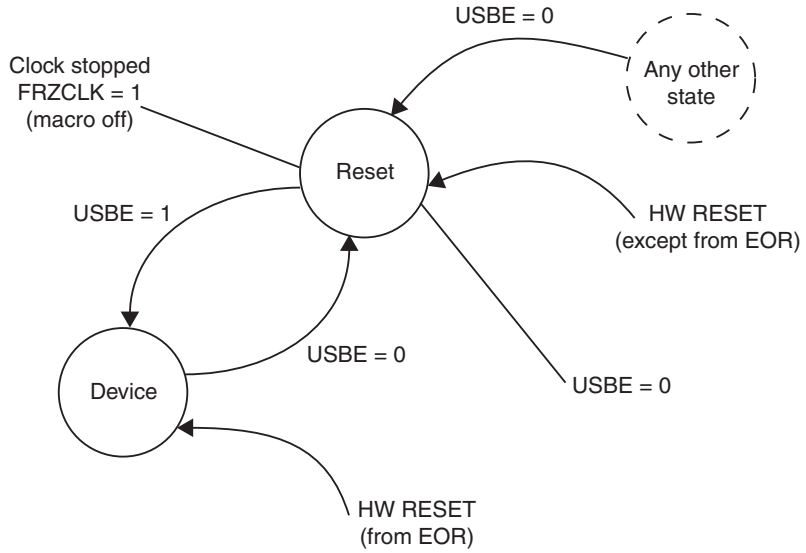
– Power on reset
– External reset
– Watchdog reset
– Brown out reset
– debugWIRE reset
– USB End Of Reset

In the case of USB End Of Reset (EOR), the USB controller is reset, but not disabled. Therefore the device remains attached.

20.4.2 Power-on and reset

Figure 20-7 on page 189 illustrates the USB controller main states on power-on:

Figure 20-7. USB controller states after reset



When the USB controller is in reset state:

- USBE is not set
- the USB controller clock is stopped in order to minimize the power consumption (FRZCLK=1)
- the USB controller is disabled
- USB is in the suspend mode
- the Device USB controllers internal state is reset
- The DPACC bit and the DPADD10:0 field can be set by software. The DPRAM is not cleared.
- The SPDCONF bits can be set by software

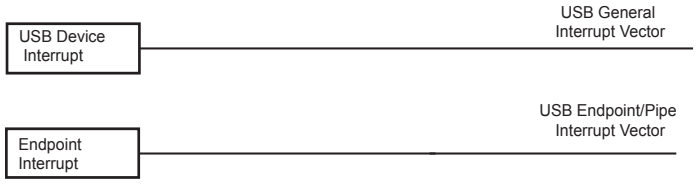
After setting USBE, the USB Controller enters in the Device state.

The USB Controller can at any time be reset by clearing USBE.

20.4.3 Interrupts

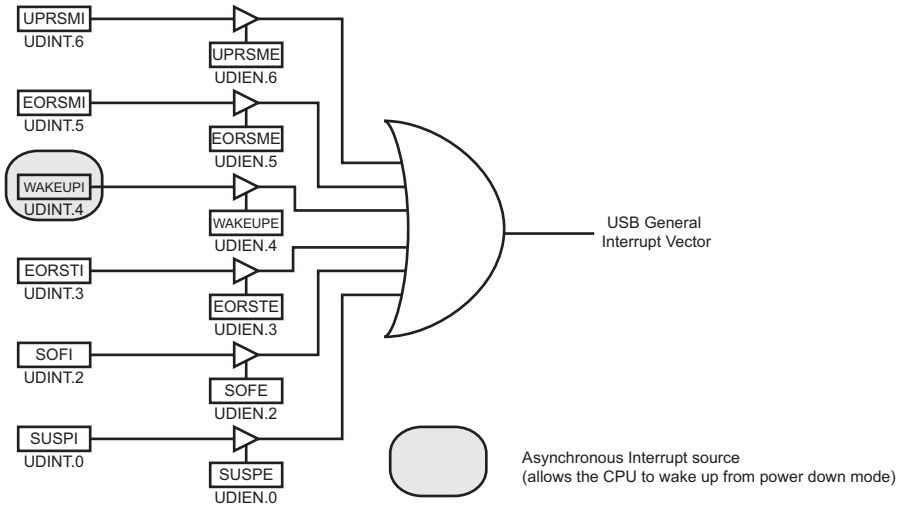
Two interrupts vectors are assigned to the USB controller.

Figure 20-8. USB Interrupt System



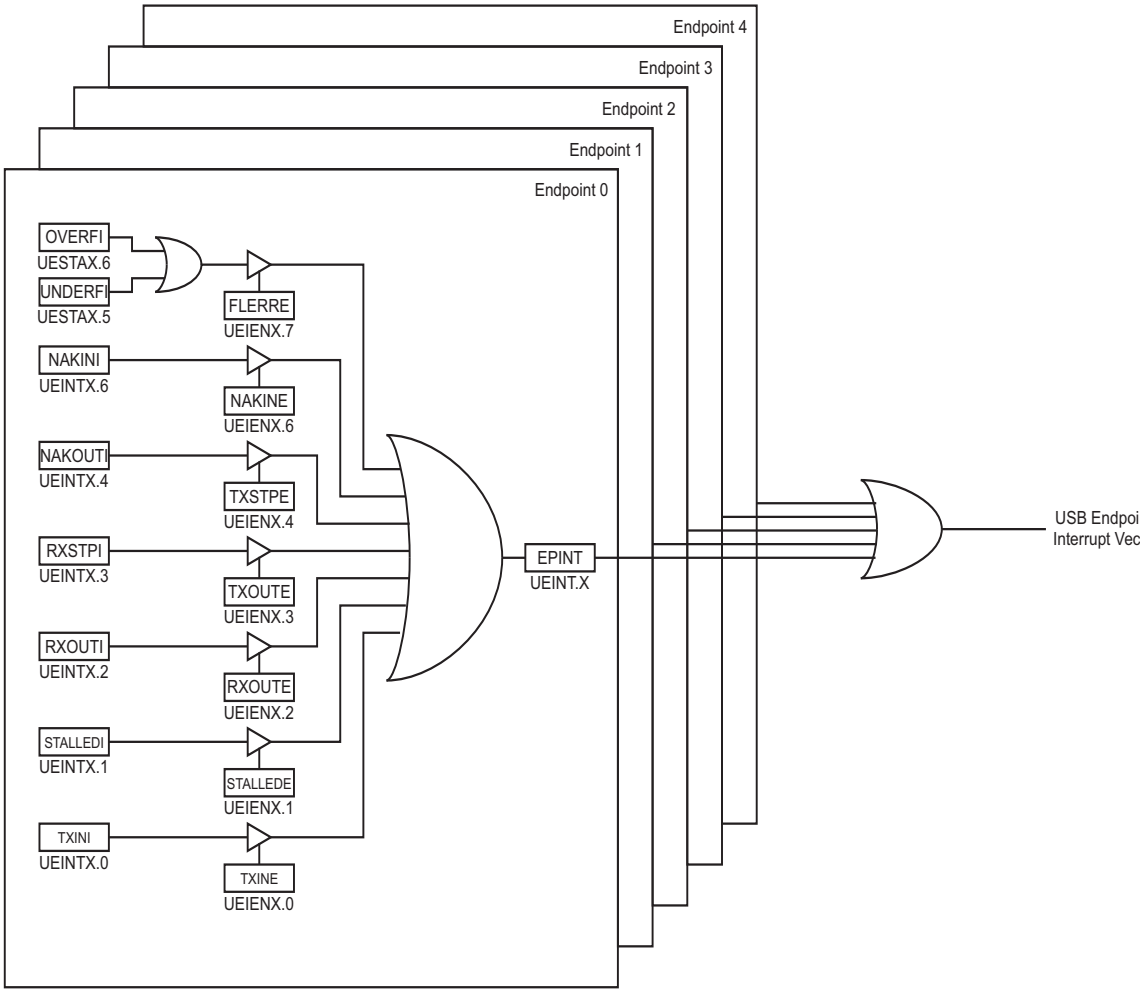
The USB module distinguishes between USB General events and USB Endpoints events.

Figure 20-9. USB General interrupt vector sources



The WAKEUP interrupt allows device wake-up from power-down mode, and is an asynchronous interrupt, triggering each time a state change is detected on the data lines. The other interrupts are synchronous and will be detected only if the USB clock is enabled (FRZCLK bit set).

Figure 20-10. USB Endpoint Interrupt vector sources



Each endpoint has 8 interrupts sources associated with flags, and each source can be enabled to trigger the corresponding endpoint interrupt.

If, for an endpoint, at least one of the sources is enabled to trigger interrupt, the corresponding event(s) will make the program branch to the USB Endpoint Interrupt vector. The user may determine the source (endpoint) of the interrupt by reading the “UDINT – USB Device Interrupt Register” on page 210.

20.5 Power modes

20.5.1 Idle mode

In Idle mode, the CPU is halted (CPU clock stopped). The Idle mode is taken wether the USB controller is running or not. The CPU can wake up on any USB interrupts.

20.5.2 Power-down

In Power-down mode, the oscillator is stopped and halts all the clocks (CPU and peripherals). The USB controller wakes up when:

- the WAKEUPI interrupt is triggered (single asynchronous interrupt)

20.5.3 Freeze clock

The firmware has the ability to freeze the clock of USB controller by setting the FRZCLK bit, and thereby reduce the power consumption. When FRZCLK is set, it is still possible to access to the following registers:

- USBCON
- DPRAM direct access registers (DPADD7:0, UEDATX)
- UDCON
- UDINT
- UDIEN

When FRZCLK is set, only the asynchronous interrupt may be triggered:

- WAKEUPI

20.6 Memory management

The controller does only support the following memory allocation management.

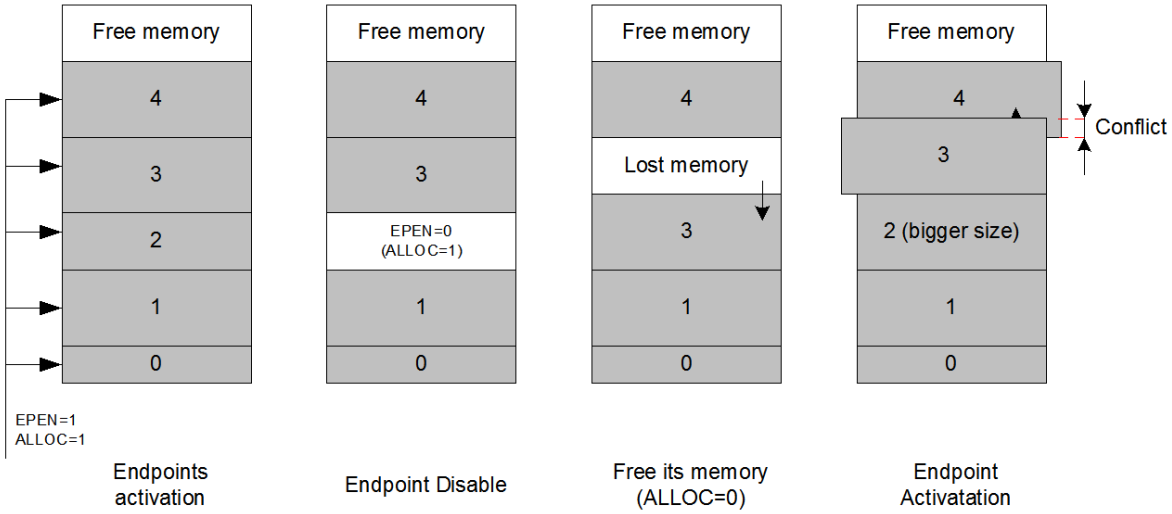
The reservation of an Endpoint can only be made in the increasing order (Endpoint 0 to the last Endpoint). The firmware shall thus configure them in the same order.

The reservation of an Endpoint k^i is done when its ALLOC bit is set. Then, the hardware allocates the memory and insert it between the Endpoints k^{i-1} and k^{i+1}. The k^{i+1} Endpoint memory “slides” up and its data is lost. Note that the k^{i+2} and upper Endpoint memory does not slide.

Clearing an Endpoint enable (EPEN) does not clear either its ALLOC bit, or its configuration (EPSIZE/PSIZE, EPBK/PBK). To free its memory, the firmware should clear ALLOC. Then, the k^{i+1} Endpoint memory automatically slides down. Note that the k^{i+2} and upper Endpoint memory does not slide.

The following figure illustrates the allocation and reorganization of the USB memory in a typical example:

Table 20-1. Allocation and reorganization USB memory flow



- Endpoints activation:
  Endpoint 0 to Endpoint 4 are configured, in the growing order. The memory of each is reserved in the DPRAM.
- Endpoint disable:
  The Endpoint 2 is disabled (EPEN=0), but its memory reservation is internally kept by the controller.
- Free its memory:
  The ALLOC bit is cleared: the Endpoint 3 slides down, but the Endpoint 4 does not slide.
- Endpoint activation:
  The firmware chooses to reconfigure the Endpoint 2, but with a bigger size. The controller reserved the memory after the endpoint 1 memory and automatically slide the Endpoint 3. The Endpoint 4 does not move and a memory conflict appear, in that both Endpoint 3 and 4 use a common area. The data of those endpoints are potentially lost.

Note that:

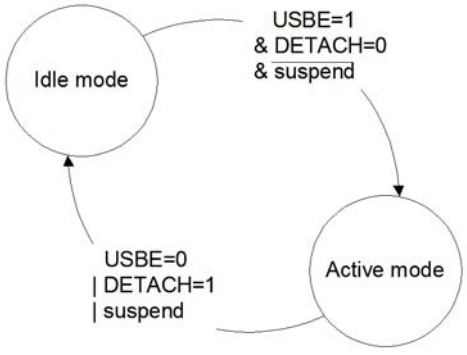
- The data of Endpoint 0 is never lost at activation or deactivation of a higher Endpoint. The data is lost only if the Endpoint 0 is deactivated.
- Deactivate and reactivate the same Endpoint with the same parameters does not lead to a slide of the higher endpoints. For those endpoints, the data are preserved.
- CFGOK is set by hardware even in the case that there is a “conflict” in the memory allocation.

20.7 PAD suspend

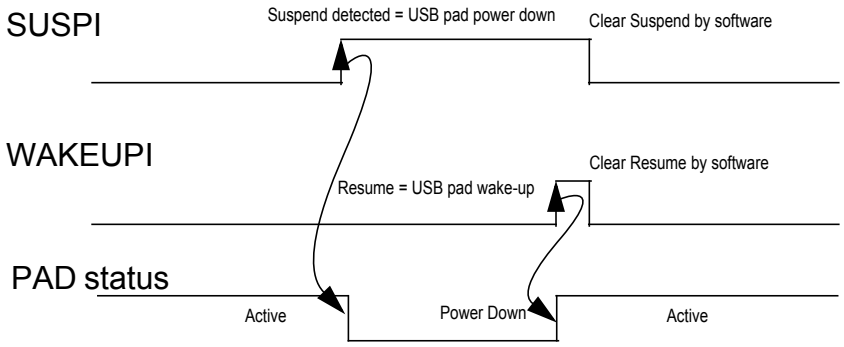
The next figures illustrates the pad behaviour:

- In the Idle mode, the pad is put in low power consumption mode.
- In the Active mode, the pad is working.

Figure 20-11. Pad behaviour



The SUSPI flag indicated that a suspend state has been detected on the USB bus. This flag automatically put the USB pad in Idle. The detection of a non-idle event sets the WAKEUPI flag and wakes-up the USB pad.



Moreover, the pad can also be put in the Idle mode if the DETACH bit is set. It come back in the Active mode when the DETACH bit is cleared.

20.8 D+/D- Read/write

The level of D+ and D- can be read and written using the UPOE register. The USB controller has to be enabled to write a value. For read operation, the USB controller can be enabled or disabled.

20.9 USB Software Operating modes

Depending on the USB operating mode, the software should perform some of the following operations:

Power On the USB interface

- Configure PLL interface
- Enable PLL
- Check PLL lock
- Enable USB interface
- Configure USB interface (USB Endpoint 0 configuration)
- Attach USB device

Power Off the USB interface

- Detach USB device
- Disable USB interface
- Disable PLL

Suspending the USB interface

- Clear Suspend Bit
- Set USB suspend clock
- Disable PLL

- Be sure to have interrupts enabled (WAKEUPE) to exit sleep mode
- Put the MCU in sleep mode

Resuming the USB interface

- Enable PLL
- Wait PLL lock
- Clear USB suspend clock
- Clear Resume information

20.10 Registers Description

20.10.1 USBCON – USB General Control Registers

Bit	7	6	5	4	3	2	1	0	
(0xD8)	USBE	-	FRZLK	-	-	-	-	-	USBCON
Read/Write	R/W	R	R/W	R	R	R	R	R	
Initial Value	0	0	1	0	0	0	0	0	

- **Bit 7 – USBE: USB macro Enable Bit**

Writing this bit to one enables the USB controller and the USB data buffers (D+ and D-). Clearing this bit disables the USB controller and buffers. When cleared the USB controller is reset.

- **Bit 6 – Res: Reserved**

This bit is reserved and should always read as zero.

- **Bit 5 – FRZCLK: Freeze USB Clock Bit**

Writing this bit to one disables the internal clock for the USB controller, and tehreby freezing it. Activating this mode reduces power consumption. All the USB flags are kept unchanged. Only the “Resume detection” is still active in this mode.

Writing this bit to zero unfreezes the USB controller and allows full operation of the USB interface.

- **Bits 4:0 – Res: Reserved**

These bits are reserved and should always read as zero.

20.10.2 UPOE – USB Software Output Enable register

Bit	7	6	5	4	3	2	1	0	
(0xFB)	UPWE1	UPWE0	UPDRV1	UPDRV0	-	-	DPI	DMI	UPOE
Read/Write	R/W	R/W	R/W	R/W	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – UPWE[1:0]: USB Buffers Direct Drive enable configuration**

These bits select the mode of operation of the USB buffers according to Table 20-2. The possible configurations of these bits allows to enable or disable the USB buffers direct drive by software. When direct drive for USB buffers is enable, the UPDRV[1:0] values are output to the

buffers.

Table 20-2. UPWE[I:0] Bits Settings

UPWE1	UPWE0	Mode
0	0	Direct drive is disabled.
0	1	Reserved
1	0	Direct drive of DP/DM (UPDRV[1:0] values)
1	1	Reserved

- **Bit 5:4 – UPDRV[1:0]: USB direct drive values**

These bits are relevant only when one of the direct drive modes for USB is enable. When UPWE[1:0] is 1:0 the values of these bits are output to USB.

The value written to UPDRV1 is output to D+.

The value written to UPDRV0 is output to D-.

- **Bits 3:2 – Res: Reserved**

These bits are reserved and should always read as zero.

- **Bit 1 – DPI: D+ Input value**

This bit is read only, the value read from this bit reflects the D+ pin (USB buffer). This bit is set one by hardware if a one logic level is read on D+. This bit is set to zero by hardware if a zero logic level is read on D+.

- **Bit 0 – DMI: D- Input value**

This bit is read only, the value read from this bit reflects the D- pin (USB buffer). This bit is set one by hardware if a logic one is read on D-. This bit is set to zero by hardware if a logic zero logic is read on D-.

20.10.3 REGCR – Regulator Control Register

Bit	7	6	5	4	3	2	1	0	
(0x63)	-	-	-	-	-	-	-	REGDIS	REGCR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 0 – REGDIS: Regulator Disable**

Writing this bit to a logic one disables the internal 3.3V regulator. Writing this bit to a logic zero enables the regulstor.

21. USB Device Operating modes

21.1 Overview

The USB device controller supports full speed data transfers. In addition to the default control endpoint, it provides four other endpoints, which can be configured in control, bulk, interrupt or isochronous modes:

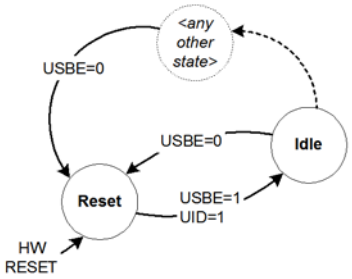
- Endpoint 0: Programmable size FIFO up to 64 bytes, default control endpoint
- Endpoint 1 and 2: Programmable size FIFO up to 64 bytes.
- Endpoint 3 and 4: Programmable size FIFO up to 64 bytes with ping-pong mode.

The controller starts in the “idle” mode. In this mode, the pad consumption is reduced to the minimum.

21.2 Power-on and reset

The next diagram explains the USB device controller main states on power-on:

Figure 21-1. USB device controller states after reset



The reset state of the Device controller is:

- the macro clock is stopped in order to minimize the power consumption (FRZCLK set),
- the USB device controller internal state is reset (all the registers are reset to their default value. Note that DETACH is set.)
- the endpoint banks are reset
- the D+ pull up are not activated (mode Detach)

The D+ pull-up will be activated as soon as the DETACH bit is cleared.

The macro is in the ‘Idle’ state after reset **with a minimum power consumption** and does not need to have the PLL activated to enter in this state.

The USB device controller can at any time be reset by clearing USBE.

21.3 Endpoint reset

An endpoint can be reset at any time by setting in the UERST register the bit corresponding to the endpoint (EPRSTx). This resets:

- the internal state machine on that endpoint,

- the Rx and Tx banks are cleared and their internal pointers are restored,
- the UEINTX, UESTA0X and UESTA1X are restored to their reset value.

The data toggle field remains unchanged.

The other registers remain unchanged.

The endpoint configuration remains active and the endpoint is still enabled.

The endpoint reset may be associated with a clear of the data toggle command (RSTDT bit) as an answer to the CLEAR_FEATURE USB command.

21.4 USB reset

When an USB reset is detected on the USB line (SEO state with a minimal duration of 100µs), the next operations are performed by the controller:

- All the endpoints are disabled.
- The default control endpoint remains configured.
- The data toggle of the default control endpoint is cleared.

If the hardware reset function is selected, a reset is generated to the CPU core without disabling the USB controller (that remains in the same state than after a USB Reset).

21.5 Endpoint selection

Prior to any operation performed by the CPU, the endpoint must first be selected. This is done by setting the EPNUM[2:0] bits (in UENUM register) with the endpoint number which will be managed by the CPU.

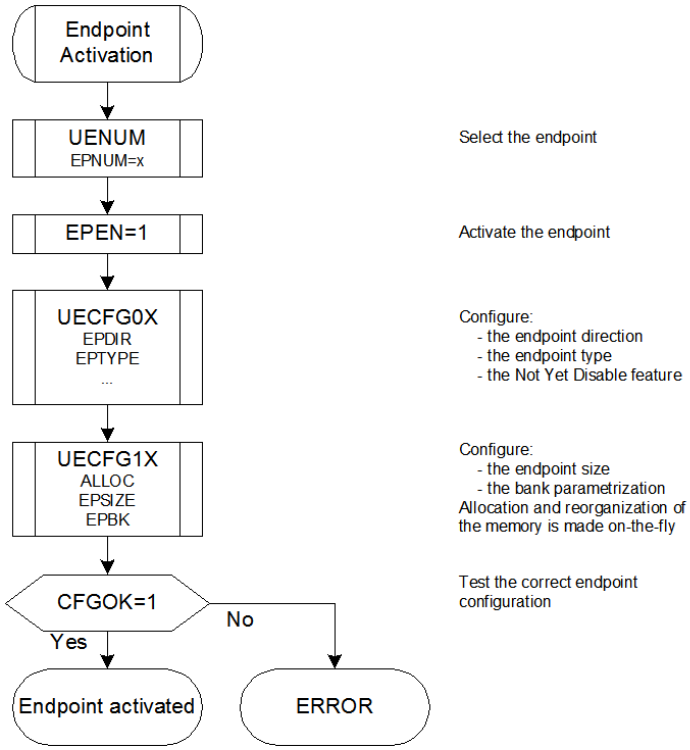
The CPU can then access to the various endpoint registers and data.

21.6 Endpoint activation

The endpoint is maintained under reset as long as the EPEN bit is not set.

The following flow must be respected in order to activate an endpoint:

Figure 21-2. Endpoint activation flow:



As long as the endpoint is not correctly configured (CFGOK cleared), the hardware does not acknowledge the packets sent by the host.

CFGOK will not be set if the Endpoint size parameter is bigger than the DPRAM size.

A clear of EPEN acts as an endpoint reset (see “Endpoint reset” on page 197 for more details). It also performs the next operation:

- The configuration of the endpoint is kept (EPSIZE, EPBK, ALLOC kept)
- It resets the data toggle field.
- The DPRAM memory associated to the endpoint is still reserved.

See “Memory management” on page 192 for more details about the memory allocation/reorganization.

21.7 Address Setup

The USB device address is set up according to the USB protocol:

- the USB device, after power-up, responds at address 0
- the host sends a **SETUP** command (SET_ADDRESS(addr)),
- the firmware records that address in UADD, but keep ADDEN cleared,
- the USB device sends an **IN** command of 0 bytes (IN 0 Zero Length Packet) to acknowledge the transaction,
- then, the firmware may enable the USB device address by setting ADDEN. The only accepted address by the controller is the one stored in UADD.

ADDEN and UADD shall not be written at the same time.

UADD contains the default address 00h after a power-up or an USB reset.

ADDEN is cleared by hardware:

- after a power-up reset,
- when an USB reset is received,
- or when the macro is disabled (USBE cleared)

When this bit is cleared, the default device address 00h is used.

21.8 Suspend, Wake-up and Resume

After the USB line has been inactive for a period of 3 ms (J state), the controller set the SUSPI flag and triggers the corresponding interrupt if enabled. The firmware may then set the FRZCLK bit.

The CPU can also, depending on software architecture, disable the PLL and/or enter in the idle mode to reduce the power consumption (especially in a bus powered application).

There are two ways to recover from the Suspend mode:

1. Clear the FRZCLK bit. This is possible if the CPU is not in the Idle mode.
2. If the CPU is in idle mode, enable the WAKEUPI interrupt (WAKEUPE set). Then, as soon as an non-idle signal is seen by the controller, the WAKEUPI interrupt is triggered. The firmware shall then clear the FRZCLK bit to restart the transfer.

There are no relationship between the SUSPI interrupt and the WAKEUPI interrupt: the WAKEUPI interrupt is triggered as soon as there are non-idle patterns on the data lines. Thus, the WAKEUPI interrupt can occurs even if the controller is not in the “suspend” mode.

When the WAKEUPI interrupt is triggered, if the SUSPI interrupt bit was already set, it is cleared by hardware.

When the SUSPI interrupt is triggered, if the WAKEUPI interrupt bit was already set, it is cleared by hardware.

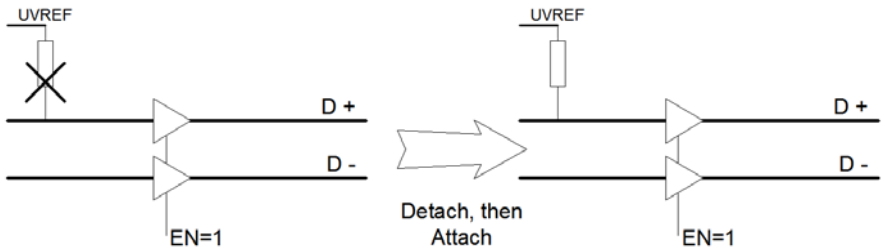
21.9 Detach

The reset value of the DETACH bit is 1.

It is possible to re-enumerate a device, simply by setting and clearing the DETACH bit (the line discharge time must be taken in account).

- When the USB device controller is in full-speed mode, setting DETACH will disconnect the pull-up on the D+. Then, clearing DETACH will connect the pull-up on the D+.

Figure 21-3. Detach a device in Full-speed:



21.10 Remote Wake-up

The Remote Wake-up (or upstream resume) request is the only operation allowed to be sent by the device on its own initiative. Anyway, to do that, the device should first have received a DEVICE_REMOTE_WAKEUP request from the host.

- First, the USB controller must have detected the “suspend” state of the line: the remote wake-up can only be sent if the SUSPI bit is set.
- The firmware has then the ability to set RMWKUP to send the “upstream resume” stream. This will automatically be done by the controller after 5ms of inactivity on the USB line.
- When the controller starts to send the “upstream resume”, the UPRSMI flag is set and interrupt is triggered (if enabled). If SUSPI was set, SUSPI is cleared by hardware.
- RMWKUP is automatically cleared by hardware at the end of the “upstream resume”.
- After that, if the controller detects a good “End Of Resume” signal from the host, an EORSMI interrupt is triggered (if enabled).

21.11 STALL request

For each endpoint, the STALL management is performed using 2 bits:

– STALLRQ (enable stall request)

– STALLRQC (disable stall request)

– STALLEDI (stall sent interrupt)

To send a STALL handshake at the next request, the STALLRQ request bit has to be set. All following requests will be handshak’ed with a STALL until the STALLRQC bit is set.

Setting STALLRQC automatically clears the STALLRQ bit. The STALLRQC bit is also immediately cleared by hardware after being set by software. Thus, the firmware will never read this bit as set.

Each time the STALL handshake is sent, the STALLEDI flag is set by the USB controller and the EPINTx interrupt will be triggered (if enabled).

The incoming packets will be discarded (RXOUTI and RWAL will not be set).

The host will then send a command to reset the STALL: the firmware just has to set the STALLRQC bit and to reset the endpoint.

21.11.1 Special consideration for Control Endpoints

A SETUP request is always ACK’ed.

If a STALL request is set for a Control Endpoint and if a SETUP request occurs, the SETUP request has to be ACK’ed and the STALLRQ request and STALLEDI sent flags are automatically reset (RXSETUPI set, TXIN cleared, STALLED cleared, TXINI cleared...).

This management simplifies the enumeration process management. If a command is not supported or contains an error, the firmware set the STALL request flag and can return to the main task, waiting for the next SETUP request.

This function is compliant with the Chapter 8 test that sends extra status for a GET_DESCRIPTOR. The firmware sets the STALL request just after receiving the status. All extra status will be automatically STALL’ed until the next SETUP request.

21.11.2 STALL handshake and Retry mechanism

The Retry mechanism has priority over the STALL handshake. A STALL handshake is sent if the STALLRQ request bit is set and if there is no retry required.

21.12 CONTROL endpoint management

A SETUP request is always ACK'ed. When a new setup packet is received, the RXSTPI interrupt is triggered (if enabled). The RXOUTI interrupt is not triggered.

The FIFOCON and RWAL fields are irrelevant with CONTROL endpoints. The firmware shall thus never use them on that endpoints. When read, their value is always 0.

CONTROL endpoints are managed by the following bits:

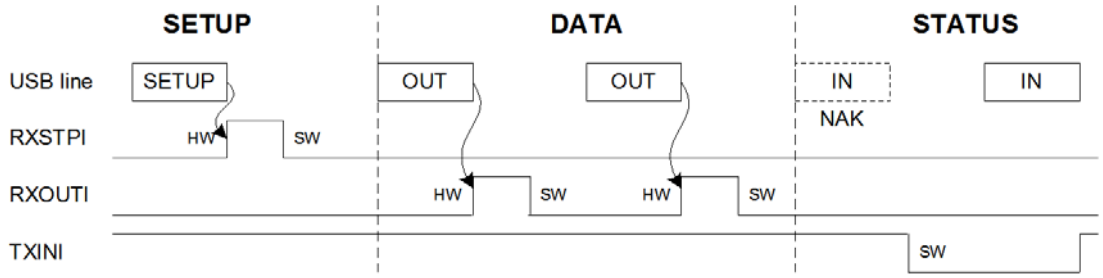
- RXSTPI is set when a new SETUP is received. It shall be cleared by firmware to acknowledge the packet **and to clear the endpoint bank**.
- RXOUTI is set when a new OUT data is received. It shall be cleared by firmware to acknowledge the packet **and to clear the endpoint bank**.
- TXINI is set when the bank is ready to accept a new IN packet. It shall be cleared by firmware to **send the packet and to clear the endpoint bank**.

CONTROL endpoints should not be managed by interrupts, but only by polling the status bits.

21.12.1 Control Write

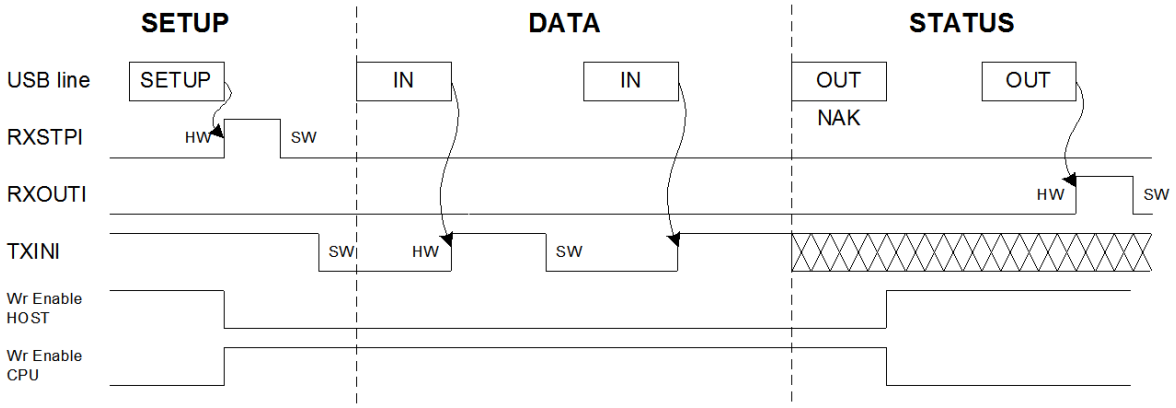
The next figure shows a control write transaction. During the status stage, the controller will not necessary send a NAK at the first IN token:

- If the firmware knows the exact number of descriptor bytes that must be read, it can then anticipate on the status stage and send a ZLP for the next IN token,
- or it can read the bytes and poll NAKINI, which tells that all the bytes have been sent by the host, and the transaction is now in the status stage.



21.12.2 Control Read

The next figure shows a control read transaction. The USB controller has to manage the simultaneous write requests from the CPU and the USB host:



A NAK handshake is always generated at the first status stage command.

When the controller detect the status stage, all the data written by the CPU are erased, and clearing TXINI has no effects.

The firmware checks if the transmission is complete or if the reception is complete.

The OUT retry is always ACK'ed. This reception:

- set the RXOUTI flag (received OUT data)

- set the TXINI flag (data sent, ready to accept new data)

software algorithm:

```
set transmit ready
wait (transmit complete OR Receive complete)
if receive complete, clear flag and return
if transmit complete, continue
```

Once the OUT status stage has been received, the USB controller waits for a SETUP request. The SETUP request have priority over any other request and has to be ACK'ed. This means that any other flag should be cleared and the fifo reset when a SETUP is received.

WARNING: the byte counter is reset when a OUT Zero Length Packet is received. The firmware has to take care of this.

21.13 OUT endpoint management

OUT packets are sent by the host. All the data can be read by the CPU, which acknowledges or not the bank when it is empty.

21.13.1 Overview

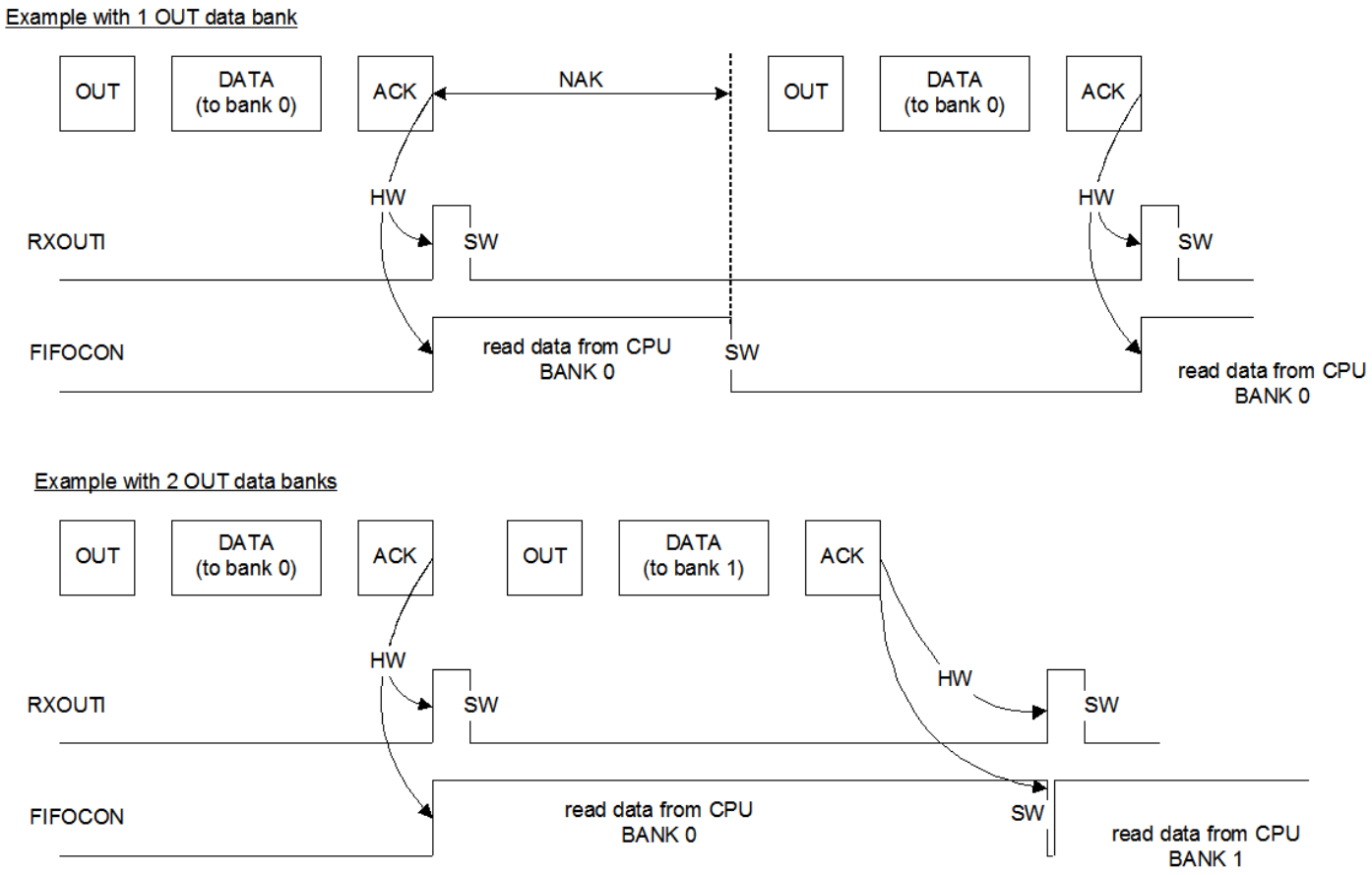
The Endpoint must be configured first.

21.13.1.1 *"Manual" mode*

Each time the current bank is full, the RXOUTI and the FIFOCON bits are set. This triggers an interrupt if the RXOUTE bit is set. The firmware can acknowledge the USB interrupt by clearing the RXOUTI bit. The Firmware read the data and clear the FIFOCON bit in order to free the current bank. If the OUT Endpoint is composed of multiple banks, clearing the FIFOCON bit will switch to the next bank. The RXOUTI and FIFOCON bits are then updated by hardware in accordance with the status of the new bank.

RXOUTI shall always be cleared before clearing FIFOCON.

The RWAL bit always reflects the state of the current bank. This bit is set if the firmware can read data from the bank, and cleared by hardware when the bank is empty.



21.13.2 Detailed description

The data are read by the CPU, following the next flow:

- When the bank is filled by the host, an endpoint interrupt (EPINTx) is triggered, if enabled (RXOUTE set) and RXOUTI is set. The CPU can also poll RXOUTI or FIFOCON, depending on the software architecture,
- The CPU acknowledges the interrupt by clearing RXOUTI,
- The CPU can read the number of byte (N) in the current bank (N=BYCT),
- The CPU can read the data from the current bank ("N" read of UEDATX),

- The CPU can free the bank by clearing FIFOCON when all the data is read, that is:
- after “N” read of UEDATX,
- as soon as RWAL is cleared by hardware.

If the endpoint uses 2 banks, the second one can be filled by the HOST while the current one is being read by the CPU. Then, when the CPU clear FIFOCON, the next bank may be already ready and RXOUTI is set immediately.

21.14 IN endpoint management

IN packets are sent by the USB device controller, upon an IN request from the host. All the data can be written by the CPU, which acknowledge or not the bank when it is full.

21.14.1 Overview

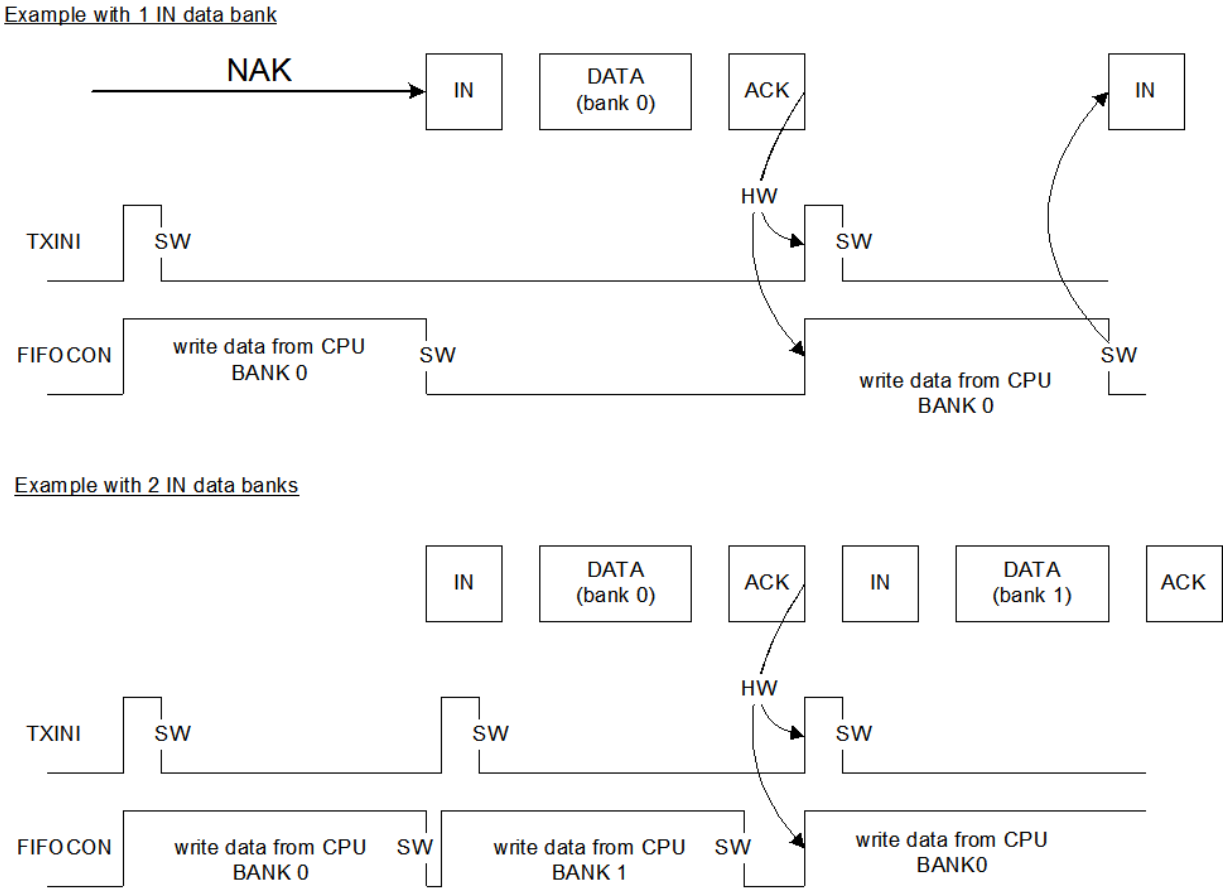
The Endpoint must be configured first.

21.14.1.1 “Manual” mode

The TXINI bit is set by hardware when the current bank becomes free. This triggers an interrupt if the TXINE bit is set. The FIFOCON bit is set at the same time. The CPU writes into the FIFO and clears the FIFOCON bit to allow the USB controller to send the data. If the IN Endpoint is composed of multiple banks, this also switches to the next data bank. The TXINI and FIFOCON bits are automatically updated by hardware regarding the status of the next bank.

TXINI shall always be cleared before clearing FIFOCON.

The RWAL bit always reflects the state of the current bank. This bit is set if the firmware can write data to the bank, and cleared by hardware when the bank is full.



21.14.2 Detailed description

The data are written by the CPU, following the next flow:

- When the bank is empty, an endpoint interrupt (EPINTx) is triggered, if enabled (TXINE set) and TXINI is set. The CPU can also poll TXINI or FIFOCON, depending the software architecture choice,
- The CPU acknowledges the interrupt by clearing TXINI,
- The CPU can write the data into the current bank (write in UEDATX),
- The CPU can free the bank by clearing FIFOCON when all the data are written, that is:
- after “N” write into UEDATX
- as soon as RWAL is cleared by hardware.

If the endpoint uses 2 banks, the second one can be read by the HOST while the current is being written by the CPU. Then, when the CPU clears FIFOCON, the next bank may be already ready (free) and TXINI is set immediately.

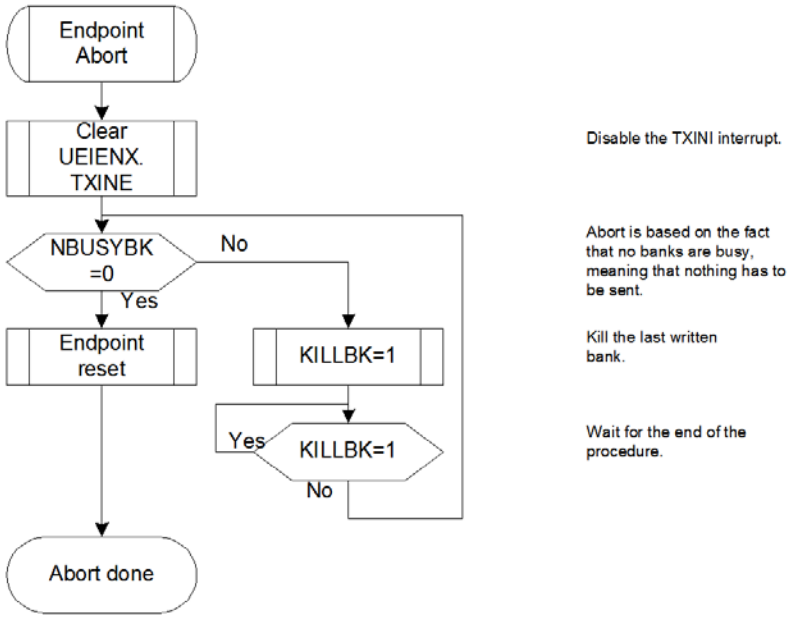
21.14.2.1 Abort

An “abort” stage can be produced by the host in some situations:

- In a control transaction: ZLP data OUT received during a IN stage,
- In an isochronous IN transaction: ZLP data OUT received on the OUT endpoint during a IN stage on the IN endpoint

The KILLBK bit is used to kill the last “written” bank. The best way to manage this abort is to perform the following operations:

Table 21-1. Abort flow



21.15 Isochronous mode

21.15.1 Underflow

An underflow can occur during IN stage if the host attempts to read a bank which is empty. In this situation, the UNDERFI interrupt is triggered.

An underflow can also occur during OUT stage if the host send a packet while the banks are already full. Typically, he CPU is not fast enough. The packet is lost.

It is not possible to have underflow error during OUT stage, in the CPU side, since the CPU should read only if the bank is ready to give data (RXOUTI=1 or RWAL=1)

21.15.2 CRC Error

A CRC error can occur during OUT stage if the USB controller detects a bad received packet. In this situation, the STALLEDI interrupt is triggered. This does not prevent the RXOUTI interrupt from being triggered.

21.16 Overflow

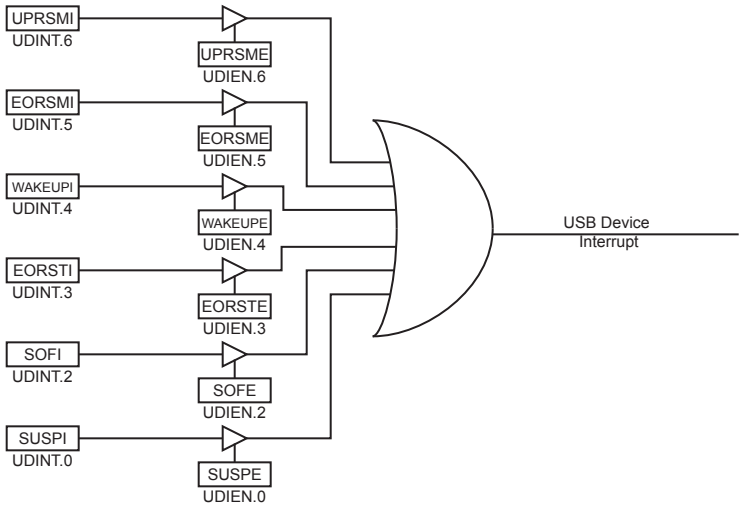
In Control, Isochronous, Bulk or Interrupt Endpoint, an overflow can occur during OUT stage, if the host attempts to write in a bank that is too small for the packet. In this situation, the OVERFI interrupt is triggered (if enabled). The packet is acknowledged and the RXOUTI interrupt is also triggered (if enabled). The bank is filled with the first bytes of the packet.

It is not possible to have overflow error during IN stage, in the CPU side, since the CPU should write only if the bank is ready to access data (TXINI=1 or RWAL=1).

21.17 Interrupts

The next figure shows all the interrupts sources:

Figure 21-4. USB Device Controller Interrupt System



There are 2 kind of interrupts: processing (i.e. their generation are part of the normal processing) and exception (errors).

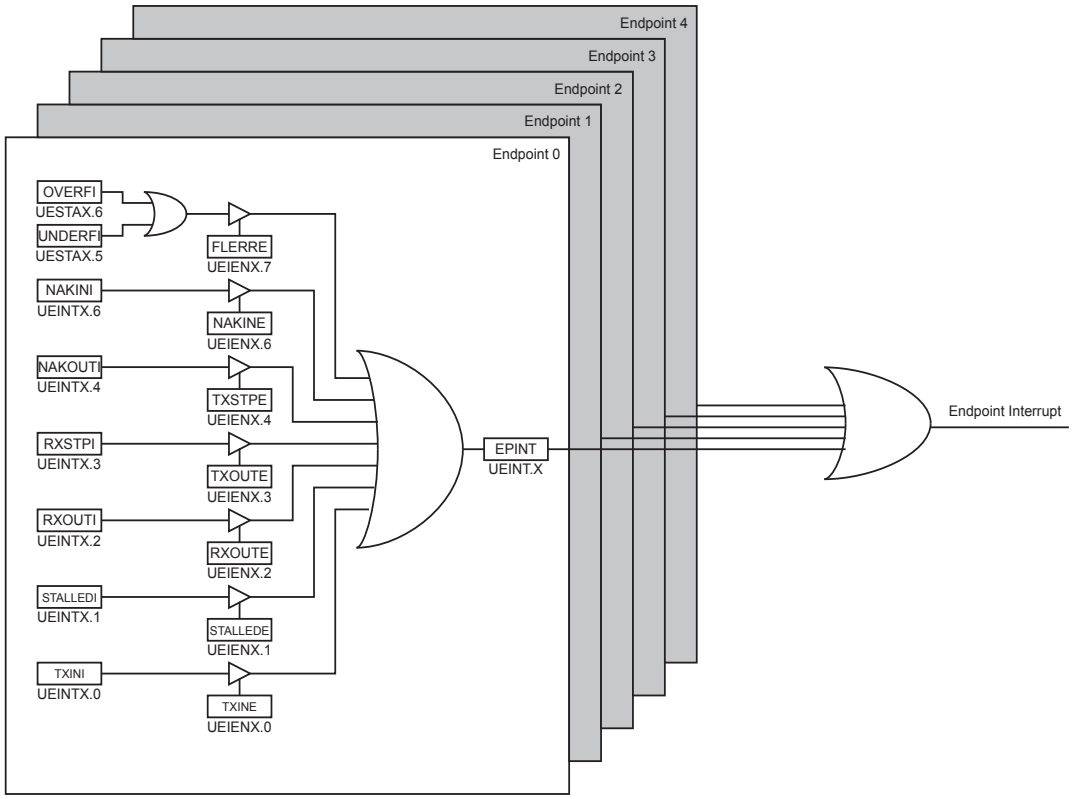
Processing interrupts are generated when:

- Upstream resume(**UPRSMI**)
- End of resume(**EORSMI**)
- Wake up(**WAKEUPI**)
- End of reset (Speed Initialization)(**EORSTI**)
- Start of frame(**SOFI**, if FNCERR=0)
- Suspend detected after 3 ms of inactivity(**SUSPI**)

Exception Interrupts are generated when:

- CRC error in frame number of SOF(**SOFI**, FNCERR=1)

Figure 21-5. USB Device Controller Endpoint Interrupt System



Processing interrupts are generated when:

- Ready to accept IN data(**EPINTx**, TXINI=1)
- Received OUT data(**EPINTx**, RXOUTI=1)
- Received SETUP(**EPINTx**, RXSTPI=1)

Exception Interrupts are generated when:

- Stalled packet(**EPINTx**, STALLEDI=1)
- CRC error on OUT in isochronous mode(**EPINTx**, STALLEDI=1)
- Overflow(**EPINTx**, OVERFI=1)
- Underflow in isochronous mode(**EPINTx**, UNDERFI=1)
- NAK IN sent(**EPINTx**, NAKINI=1)
- NAK OUT sent(**EPINTx**, NAKOUTI=1)

21.18 Register Description

21.18.1 UDCON – USB Device Control Registers

Bit	7	6	5	4	3	2	1	0	
(0xE0)	-	-	-	-	-	**RSTCPU**	**RMWKUP**	**DETACH**	**UDCON**
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	1	

- **Bits 7:3 – Res: Reserved**

These bits are reserved and will always read as zero.

- **Bit 2 – RSTCPU: USB Reset CPU Bit**

Writing this bit to one allows the CPU controller to reset the CPU when a USB bus reset condition is detected. When this mode is activated, the next USB bus reset event allows to reset the CPU and all peripherals except the USB controller. This mode allows to perform a software reset, but keep the USB device attached to the bus.

This bit is reset when the USB controller is disabled or when writing this bit to zero by firmware. Writing this bit to zero makes the CPU system reset independent from the USB bus reset event.

- **Bit 1 – RMWKUP: Remote Wake-up Bit**

Writing this bit to one allows the USB controller to generate an “upstream-resume” packet on the USB bus. This bit is immediately cleared by hardware and can not be read back to one. Writing this bit to zero has no effect.

See “Remote Wake-up” on page 201 for more details.

- **Bit 0 – DETACH: Detach Bit**

Writing this bit to one (default value) disables the USB D+ internal pull-up. This makes the USB device controller physically “detached” from the USB bus. Writing this bit to zero enables the D+ internal pull-up and physically connects the USB device controller to the USB bus. See “Detach” on page 200 for more details.

21.18.2 UDINT – USB Device Interrupt Register

Bit	7	6	5	4	3	2	1	0	
(0xE1)	-	UPRSMI	EORSMI	WAKEUPI	EORSTI	SOFI	-	SUSPI	UDINT
Read/Write	R	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 – Res: Reserved**

This bit is reserved and should always read as zero.

- **Bit 6 – UPRSMI: Upstream Resume Interrupt Flag**

This flag is set by hardware when the USB controller has successfully sent the Upstream Resume sequence (See description of “Bit 1 – RMWKUP: Remote Wake-up Bit” on page 210). If UPRSME is set, the UPRSMI flag can generate a “USB general interrupt”. Writing this bit to zero acknowledges the interrupt source (USB clocks must be enabled before). Writing this bit to one has no effect.

- **Bit 5 – EORSMI: End Of Resume Interrupt Flag**

This flag is set by hardware when the USB controller detects an End Of Resume sequence on the USB initiated by the host. If the EORSME bit is set, the EORSMI flag can generate a “USB general interrupt”. Writing this bit to zero acknowledges the interrupt source (USB clocks must be enabled before). Writing this bit to one has no effect.

- **Bit 4 – WAKEUPI: Wake-up CPU Interrupt Flag**

This flag is set by hardware when the USB controller detects a non-idle signal from the USB lines. This WAKEUPI flag can generate a “USB general interrupt” if WAKEUPE bit is set. Writing this bit to zero acknowledges the interrupt source. Writing this bit to one has no effect.Shall be cleared by software. Setting by software has no effect.

See “Suspend, Wake-up and Resume” on page 200 for more details.

- **Bit 3 – EORSTI: End Of Reset Interrupt Flag**

This flag is set by hardware when the USB controller detects an “End Of Reset” event on the USB lines. has been detected by the USB controller. This EORSTI flag can generate a “USB general interrupt” if EORSTE bit is set. Writing this bit to zero acknowledges the interrupt source (USB clocks must be enabled before). Writing this bit to one has no effect.

Shall be cleared by software. Setting by software has no effect.

- **Bit 2 – SOFI: Start Of Frame Interrupt Flag**

This flag is set by hardware when the USB controller detects a Start Of Frame PID (SOF) on the USB lines. This SOFI flag can generate a “USB general interrupt” if SOFE bit is set. Writing this bit to zero acknowledges the interrupt source (USB clocks must be enabled before). Writing this bit to one has no effect.

- **Bit 1 – Res: Reserved**

This bit is reserved and will always read as zero.

- **Bit 0 – SUSPI: Suspend Interrupt Flag**

This flag is set by hardware when the USB controller detects a suspend state on the bus (idle state for more than 3ms). This SUSPI flag can generate a USB general interrupt if SUSPE bit is set. Writing this bit to zero acknowledges the interrupt source (USB clocks must be enabled before). Writing this bit to one has no effect.

See “Suspend, Wake-up and Resume” on page 200 for more details.

The interrupt flag bits are set even if their corresponding ‘Enable’ bits is not set.

21.18.3 UDIEN – USB Device Interrupt Enable Register

Bit	7	6	5	4	3	2	1	0	
(0xE2)	-	UPRSME	EORSME	WAKEUPE	EORSTE	SOFE	-	SUSPE	UDIEN
Read/Write	R	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 – Res: Reserved**

This bit is reserved and will always read as zero.

- **Bit 6 – UPRSME: Upstream Resume Interrupt Enable Bit**

Writing this bit to one enables interrupt on UPRSMI flag. An Upstream resume interrupt will be generated only if the UPRSME bit is set to one, the Global Interrupt Flag in SREG is written to one and the UPRSMI bit is set.

- **Bit 5 – EORSME: End Of Resume Interrupt Enable Bit**

Writing this bit to one enables interrupt on EORSMI flag. An end of resume Upstream resume interrupt will be generated only if the EORSME bit is set to one, the Global Interrupt Flag in SREG is written to one, and the EORSMI bit is set.

- **Bit 4 – WAKEUPE: Wake-up CPU Interrupt Enable Bit**

Writing this bit to one enables interrupt on WAKEUPI flag. A wake-up interrupt will be generated only if the WAKEUPE bit is set to one, the Global Interrupt Flag in SREG is written to one, and the WAKEUPI bit is set.

- **Bit 3 – EORSTE: End Of Reset Interrupt Enable Bit**

Writing this bit to one enables interrupt on EORSTI flag. A USB reset interrupt will be generated only if the EORSTE bit is set to one, the Global Interrupt Flag in SREG is written to one, and the EORSTI bit is set.

- **Bit 2 – SOFE: Start Of Frame Interrupt Enable Bit**

Writing this bit to one enables interrupt on SOFI flag. A Start of Frame USB reset interrupt will be generated only if the SOFE bit is set to one, the Global Interrupt Flag in SREG is written to one, and the SOFI bit is set.

- **Bit 1 – Res: Reserved**

This bit is reserved and will always read as zero.

- **Bit 0 – SUSPE: Suspend Interrupt Enable Bit**

Writing this bit to one enables interrupt on SUSPI flag. A suspend interrupt will be generated only if the SUSPE bit is set to one, the Global Interrupt Flag in SREG is written to one, and the SUSPI bit is set.

21.18.4 UDADDR – USB Device Address Register

Bit	7	6	5	4	3	2	1	0	
(0xE3)	ADDEN	UADD[6:0]							UDADDR
Read/Write	R/W	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ADDEN: Address Enable Bit**

Writing this bit to one will enable the UADD[6:0] field as device address for the USB controller. When this bit is set the USB device controller will be able to answer all requests on the USB that refer to the UADD[6:0] USB bus address.

See “Address Setup” on page 199 for more details.

- **Bits 6:0 – UADD[6:0]: USB Address Bits**

These bits contain the USB device address, thatthe USB controller should answer on the USB bus. This address should be enabled writing one to the ADDEN bit.

21.18.5 UDFNUMH – USB Device Frame Number High Register

Bit	7	6	5	4	3	2	1	0	
(0xE5)	-	-	-	-	-	FNUM[10:8]			UDFNUMH
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:3 – Res: Reserved**

These bits are reserved and will always read as zero.

- **Bits 2:0 – FNUM[10:8]: Frame Number Upper Flag**

These bits are read-only and updated by the hardware USB controller. These bits contains the 3 MSB of the 11-bits Frame Number information. The content of these bits is updated with the last received SOF packet. These bits are updated even if a corrupted SOF has been received. When a corrupted SOF number is detected, the FNCERR bit of UDMFN is set.

21.18.6 UDFNUML – USB Device Frame Number Low Register

Bit	7	6	5	4	3	2	1	0	
(0xE4)	FNUM[7:0]								UDFNUML
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – FNUM: Frame Number Lower Flag**

These bits are read-only and updated by the hardware USB controller. These bits contains the 8 LSB of the 11-bits Frame Number information. The content of these bits is updated with the last received SOF packet. These bits are updated even if a corrupted SOF has been received. When a corrupted SOF number is detected, the FNCERR bit of UDMFN is set.

21.18.7 UDMFN – USB Device Micro Frame Number

Bit	7	6	5	4	3	2	1	0	
(0xE6)	-	-	-	FNCERR	-	-	-	-	UDMFN
Read/Write	R	R	R	R/W	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:5 – Res: Reserved**

These bits are reserved and will always read as zero.

- **Bit 4 – FNCERR: Frame Number CRC Error Flag**

This bit is set by the USB controller when a corrupted frame number in Start of frame packet is received. When an incorrect frame number is detected both SOFI flag and this bit are set.

- **Bits 3:0 – Res: Reserved**

These bits are reserved and will always read as zero.

21.18.8 UENUM – USB Endpoint Number Register

Bit	7	6	5	4	3	2	1	0	
(0xE9)	-	-	-	-	-	EPNUM[2:0]			UENUM
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:3 – Res: Reserved**

These bits are reserved and will always read as zero.

- **Bits 2:0 – EPNUM[2:0] Endpoint Number Bits**

Writing these bits allows to select the hardware endpoint number that can be accessed by the CPU interface. This register select the target endpoint number for UECONEX, UECFG0X, UECFG1X, UESTA0X, UESTA1X, UEINTX, UEIENX, UEDATX, UEBCLX registers. See “Endpoint selection” on page 198 for more details.

21.18.9 UERST – USB Endpoint Reset Register

Bit	7	6	5	4	3	2	1	0	
(0xEA)	-	-	-	EPRST D4	EPRST D3	EPRST D2	EPRST D1	EPRST D0	UERST
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:5 – Res: Reserved**

These bits are reserved and will always read as zero.

- **Bits 4:0 – EPRST[4:0]: Endpoint FIFO Reset Bits**

Writing this bit to one keeps the selected endpoint (UENUM register value) under reset state. selected. Writing this bit to zero completes the endpoint reset operation and makes the endpoint usable. See “Endpoint reset” on page 197 for more information.

21.18.10 UECONX – USB Endpoint Control Register

Bit	7	6	5	4	3	2	1	0	
(0xEB)	-	-	STALLRQ	STALLRQC	RSTDT	-	-	EPEN	UECONX
Read/Write	R	R	R/W	R/W	R/W	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:6 – Res: Reserved**

These bits are reserved and will always read as zero.

- **Bit 5 – STALLRQ: STALL Request Handshake Bit**

Writing this bit to one allows the USB controller to generate a STALL answer for the next SETUP transaction received. This bit is cleared by hardware when the STALL handshake is sent or when a new SETUP token is received. Writing this bit to zero has no effect. The STALL handshake can be abort using STALLRQC bit.

See “STALL request” on page 201 for more details.

- **Bit 4 – STALLRQC: STALL Request Clear Handshake Bit**

Writing this bit to one disables the pending STALL handshake mechanism triggered by STALLRQ bit. This bit can not be write to zero, it is cleared by hardware immediately after the write to one operation.

See “STALL request” on page 201 for more details.

- **Bit 3 – RSTDT: Reset Data Toggle Bit**

Writing this bit to one allows to reset the data toggle bit field for the selected endpoint. This bit can not be write to zero, it is cleared by hardware immediately after the write to one operation.

- **Bits 2:1 – Res: Reserved**

These bits are reserved and will always read as zero.

- **Bit 0 – EPEN: Endpoint Enable Bit**

Writing this bit to one enables the selected endpoint. When the endpoint is enabled it can be configured and used by the USB controller. Endpoint 0 shall always be enabled after a hardware or USB reset and participate in the device configuration. Writing this bit to zero disables the current endpoint.

See “Endpoint activation” on page 198 for more details.

21.18.11 UECFG0X – USB Endpoint Configuration 0 Register

Bit	7	6	5	4	3	2	1	0	
(0xEC)	EPTYPE1:0		-	-	-	-	-	EPDIR	UECFG0X
Read/Write	R/W	R/W	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – EPTYPE[1:0]: Endpoint Type Bits**

These bits configure the endpoint type for the selected endpoint as shown in Table 21-2.

Table 21-2. EPTYPE[1:0] Bits Settings

EPTYPE1	EPTYPE0	Endpoint Type Configuration
0	0	Control Type
0	1	Isochronous Type
1	0	Bulk Type
1	1	Interrupt Type

- **Bits 5:1 – Res: Reserved**

These bits are reserved and will always read as zero.

- **Bit 0 – EPDIR: Endpoint Direction Bit**

Writing this bit to one configures the selected endpoint in the IN direction. Writing this bit to zero configure the endpoint in the OUT direction. This bit is relevant for bulk, interrupt or isochronous endpoints. Using this bit with a control endpoint has no effect (control endpoints are bidirectional).

21.18.12 UECFG1X – USB Endpoint Configuration 1 Register

Bit	7	6	5	4	3	2	1	0	
(0xED)	-	EPSIZE[2:0]			EPBK1:0		ALLOC	-	UECFG1X
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – Res: Reserved**

This bit is reserved and will always read as zero.

- **Bit 6:4 – EPSIZE[2:0]: Endpoint Size Bits**

These bits configure the endpoint size for the selected endpoint as shown in Table 21-3.

Table 21-3. EPSIZE[2:0] Bits Settings

EPSIZE2	EPSIZE1	EPSIZE0	Endpoint Size
0	0	0	8 Bytes
0	0	1	16 Bytes
0	1	0	32 Bytes
0	1	1	64 Bytes
1	0	0	Reserved.
1	0	1	
1	1	0	
1	1	1	

- **Bits 3:2 – EPBK[1:0]: Endpoint Bank Bits**

These bits configure the number of banks that is allocated to the selected endpoint as shown in Table 21-3.

Table 21-4. EPBK[1:0] Bits Settings

EPBK1	EPBK0	Endpoint Size
0	0	One Bank
0	1	Two Banks
1	0	Reserved
1	1	

- **Bit 1 – ALLOC: Endpoint Allocation Bit**

Writing this to one allows to allocate the specified amount of memory (endpoint size x number of banks) for the selected endpoint. Writing this bit to zero allows to free the previously allocated memory for the selected endpoint.

See Section 21.6, page 198 for more details.

- **Bit 0 – Res: Reserved**

This bit is reserved and will always read as zero.

21.18.13 UESTA0X – USB Endpoint Status 0 Register

Bit	7	6	5	4	3	2	1	0	
(0xEE)	CFGOK	OVERFI	UNDERFI	-	DTSEQ1:0		NBUSYBK1:0		UESTA0X
Read/Write	R	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – CFGOK: Configuration Status Flag**

This flag bit is set by hardware when the selected endpoint size parameter (EPSIZE) and number of banks (EPBK) are correct compared to the max FIFO capacity. This bit is updated when the bit ALLOC is set, if the USB controller can not allocate the correct amount of memory for the selected endpoint, this flag bit will be cleared.

If this bit is cleared, the user should reprogram the UECFG1X register with correct EPSIZE and EPBK values.

- **Bit 6 – OVERFI: Overflow Error Interrupt Flag**

This flag is set when an overflow error occurs for an isochronous endpoint.This OVERFI flag can generate a “USB endpoint interrupt” if FLERRE bit is set. Writing this bit to zero acknowledges the interrupt source (USB clocks must be enabled before). Writing this bit to one has no effect.

See “Isochronous mode” on page 207 for more details.

- **Bit 5 – UNDERFI: Underflow Error Interrupt Flag**

This flag is set when an underflow error occurs for an isochronous endpoint.This UNDERFI flag can generate a “USB endpoint interrupt” if FLERRE bit is set. Writing this bit to zero acknowledges the interrupt source (USB clocks must be enabled before). Writing this bit to one has no effect.

See “Isochronous mode” on page 207 for more details.

- **Bit 4 – Res: Reserved**

This bit is reserved and will always read as zero.

- **Bit 3:2 – DTSEQ[1:0]: Data Toggle Sequencing Flag**

These flags are set by hardware to indicate the PID data of the current bank as shown in Table 21-5.

For OUT transfer, this value indicates the last data toggle received on the current bank. For IN transfer, it indicates the Toggle that will be used for the next packet to be sent. This is not relative to the current bank.

Table 21-5. DTSEQ[1:0] Bits Settings

DTSEQ1	DTSEQ1	PID DATA
0	0	DATA0
0	1	DATA1
1	0	Reserved.
1	1	

- **Bit 1:0 – NBUSYBK[1:0]: Busy Bank Flag**

These flags are set by hardware to indicate the number of busy bank for the selected endpoint as shown in Table 21-6.

For IN endpoint, it indicates the number of busy bank(s), filled by the user, ready for IN transfer. For OUT endpoint, it indicates the number of busy bank(s) filled by OUT transaction from the host.

Table 21-6. NBUSYBK[1:0] Bits Settings

NBUSYBK1	NBUSYBK0	Number of busy banks
0	0	All banks are free
0	1	1 busy bank
1	0	2 busy banks
1	1	Reserved

21.18.14 UESTA1X – USB Endpoint Status 1 Register

Bit	7	6	5	4	3	2	1	0	
(0xEF)	-	-	-	-	-	CTRLDIR	CURRBK[1:0]		UESTA1X
Read/Write	R	R	R	R	R	R	R		
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:3 – Res: Reserved**

These bits are reserved and will always read as zero.

- **Bit 2 – CTRLDIR: Control Direction**

This flag is updated by the USB controller when a SETUP packet has been received. This flag bit can be used for debug purpose to give the direction of the following packet. Reading one from this flag means that the following packet is for an IN request, reading zero for an OUT request.

- **Bits 1:0 – CURRBK[1:0]: Current Bank**

These flags are set by hardware to indicate the current bank number in used with the selected endpoint as shown in Table 21-6. These flags are not relevant for control endpoint (control endpoint can not be configured in dual bank mode).These flags can be used for debug purpose and are optional for data transfer with endpoint in dual bank mode.

Table 21-7. CURRBK[1:0] Bits Settings

CURRBK1	CURRBK0	Current Bank Number
0	0	Bank 0
0	1	Bank 1
1	0	Reserved
1	1	

21.18.15 UEINTX – USB Endpoint Interrupt Register

Bit	7	6	5	4	3	2	1	0	
(0xE8)	FIFOCON	NAKINI	RWAL	NAKOUTI	RXSTPI	RXOUTI	STALLEDI	TXINI	UEINTX
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 – FIFOCON: FIFO Control Bit**

This bit can only be written to zero by software. Writing this bit to one has no effect. The behavior of this bit depends on the direction of the selected endpoint.

- For OUT or CONTROL Endpoints:

This flag is set by the USB controller when a new OUT message is stored in the current bank. In this situation RXOUT or RXSTP flags are also updated at the same time. Writing this bit to zero frees the current bank and switches to the next bank.

- For IN Endpoints:

This flag is set by the USB controller when the current bank is free and can be loaded with new data bytes. In this situation TXIN flag is also updated at the same time. Writing this bit to zero sends the FIFO content and to switch the next bank.

- **Bit 6 – NAKINI: NAK IN Received Interrupt Flag**

This flag is set when a NAK handshake has been sent in response to a IN request from the host. This NAKINI flag can generate a “USB endpoint interrupt” if NAKINE bit is set. Writing this bit to zero acknowledges the interrupt source (USB clocks must be enabled before). Writing this bit to one has no effect.

- **Bit 5 – RWAL: Read/Write Allowed Flag**

This flag is set by the USB controller and is relevant for all endpoint types except control endpoint. For an IN endpoint, this flag is set when the current bank is not full i.e. the firmware can push at least one more byte into the FIFO (UPDATx register). For an OUT endpoint, this flag is set when the current bank is not empty i.e. the firmware can read from the FIFO (UPDATx register). When the STALLRQ bit is set or one of the endpoint error is set, this flag can not be set.

- **Bit 4 – NAKOUTI: NAK OUT Received Interrupt Flag**

This flag is set by the USB controller when a NAK handshake has been sent in response of a OUT request from the host. This NAKOUTI flag can generate a “USB endpoint interrupt” if NAKOUTE bit is set. Writing this bit to zero acknowledges the interrupt source (USB clocks must be enabled before). Writing this bit to one has no effect.

- **Bit 3 – RXSTPI: Received SETUP Interrupt Flag**

This flag is set by the USB controller when a new valid (error free) SETUP packet has been received from the host. This RXSTPI flag can generate a “USB endpoint interrupt” if RXSTPE bit is set. Writing this bit to zero acknowledges the interrupt source (USB clocks must be enabled before). Writing this bit to one has no effect.

- **Bit 2 – RXOUTI / KILLBK: Received OUT Data Interrupt Flag**

Depending on the direction of the endpoint, this bit has two functions:

- Endpoint OUT direction (RXOUTI flag):

This flag is set by the USB controller when the current bank contains a new packet. This RXOUTI flag can generate a “USB endpoint interrupt” if RXOUTE bit is set. Writing this bit to zero acknowledges the interrupt source (USB clocks must be enabled before). Writing this bit to one has no effect for an OUT endpoint.

- Endpoint IN direction (KILLBK bit)

Writing this bit to one kills the last loaded bank. This sequence can be used to cancelled a previously loaded endpoint. Clearing by software has no effect. See page 206 for more details on the Abort.

- **Bit 1 – STALLEDI: STALLEDI Interrupt Flag**

This flag is set by the USB controller when STALL handshake has been sent, or when a CRC error has been detected for an isochronous OUT endpoint. This STALLEDI flag can generate a “USB endpoint interrupt” if STALLEDE bit is set. Writing this bit to zero acknowledges the interrupt source (USB clocks must be enabled before). Writing this bit to one has no effect.

- **Bit 0 – TXINI: Transmitter Ready Interrupt Flag**

This flag is set by the USB controller when the current bank is free and can be filled. This TXINI flag can generate a “USB endpoint interrupt” if TXINE bit is set. Writing this bit to zero acknowledges the interrupt source (USB clocks must be enabled before). Writing this bit to one has no effect.

21.18.16 UEIENX – USB Endpoint Interrupt Enable Register

Bit	7	6	5	4	3	2	1	0	
(0xF0)	FLERRE	NAKINE	-	NAKOUTE	RXSTPE	RXOUTE	STALLEDE	TXINE	UEIENX
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 – FLERRE: Flow Error Interrupt Enable Flag**

Writing this bit to one enables interrupt on OVERFI or UNDERFI flags. An overflow or underflow interrupt will be generated only if the FLERRE bit is set to one, the Global Interrupt Flag in SREG is written to one, and the OVERFI or UNDERFI flags are set.

- **Bit 6 – NAKINE: NAK IN Interrupt Enable Bit**

Writing this bit to one enables interrupt on NAKINI flag. A NAK IN interrupt will be generated only if the NAKINE bit is set to one, the Global Interrupt Flag in SREG is written to one, and the NAKINI is set.

- **Bit 5 – Res: Reserved**

This bit is reserved and will always read as zero.

- **Bit 4 – NAKOUTE: NAK OUT Interrupt Enable Bit**

Writing this bit to one enables interrupt on NAKOUTI flag. A NAKOUT interrupt will be generated only if the NAKOUTE bit is set to one, the Global Interrupt Flag in SREG is written to one, and the NAKOUTI is set.

- **Bit 3 – RXSTPE: Received SETUP Interrupt Enable Flag**

Writing this bit to one enables interrupt on RXSTPI flag. A receiveD setup interrupt will be generated only if the RXSTPE bit is set to one, the Global Interrupt Flag in SREG is written to one, and the RXSTPI is set.

- **Bit 2 – RXOUTE: Received OUT Data Interrupt Enable Flag**

Writing this bit to one enables interrupt on RXOUTI flag. A receiveD OUT interrupt will be generated only if the RXOUTE bit is set to one, the Global Interrupt Flag in SREG is written to one, and the RXOUTI is set.

- **Bit 1 – STALLEDE: Stalled Interrupt Enable Flag**

Writing this bit to one enables interrupt on STALLEDI flag. A sent STALL interrupt will be generated only if the STALLEDE bit is set to one, the Global Interrupt Flag in SREG is written to one, and the STALLEDI is set.

- **Bit 0 – TXINE: Transmitter Ready Interrupt Enable Flag**

Writing this bit to one enables interrupt on TXINI flag. A transmitter ready interrupt will be generated only if the TXINE bit is set to one, the Global Interrupt Flag in SREG is written to one, and the TXINI is set.

21.18.17 UEDATX – USB Data Endpoint Register

Bit	7	6	5	4	3	2	1	0	
(0xF1)	DAT D7	DAT D6	DAT D5	DAT D4	DAT D3	DAT D2	DAT D1	DAT D0	UEDATX
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 – DAT[7:0]: Data Bits**

The USB Data Endpoint register is a read/write register used for data transfer between the Register File and the USB device controller. Writing to the register pushes the data byte into the current bank of the selected endpoint. Reading the register pops extracts one data byte from the current bank of the selected endpoint.

21.18.18 UEBCLX – USB Endpoint Byte Count Register

Bit	7	6	5	4	3	2	1	0	
(0xF2)	BYCT D7	BYCT D6	BYCT D5	BYCT D4	BYCT D3	BYCT D2	BYCT D1	BYCT D0	UEBCLX
Read/Write	R	R	R	R	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – BYCT[7:0]:Byte Count Bits**

This register is read only. Its content is updated by the USB controller.

- For IN endpoint:

This register contains the number of byte currently loaded into the current bank of the selected endpoint. The content of this register is incremented after each write access to the endpoint data register.

- For OUT endpoint:

This register contains the number of received byte into the current bank of the selected endpoint. The content of this register is decremented after each write access to the endpoint data register.

21.18.19 UEINT – USB Endpoint Number interrupt Register

Bit	7	6	5	4	3	2	1	0	
(0xF4)	-	-	-	EPINT4:0					UEINT
Read/Write	R	R	R	R	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:5 – Res: Reserved**

The value read from these bits is always 0. Do not set these bits.

- **Bits 4:0 – EPINT[4:0]: Endpoint Interrupts Bits**

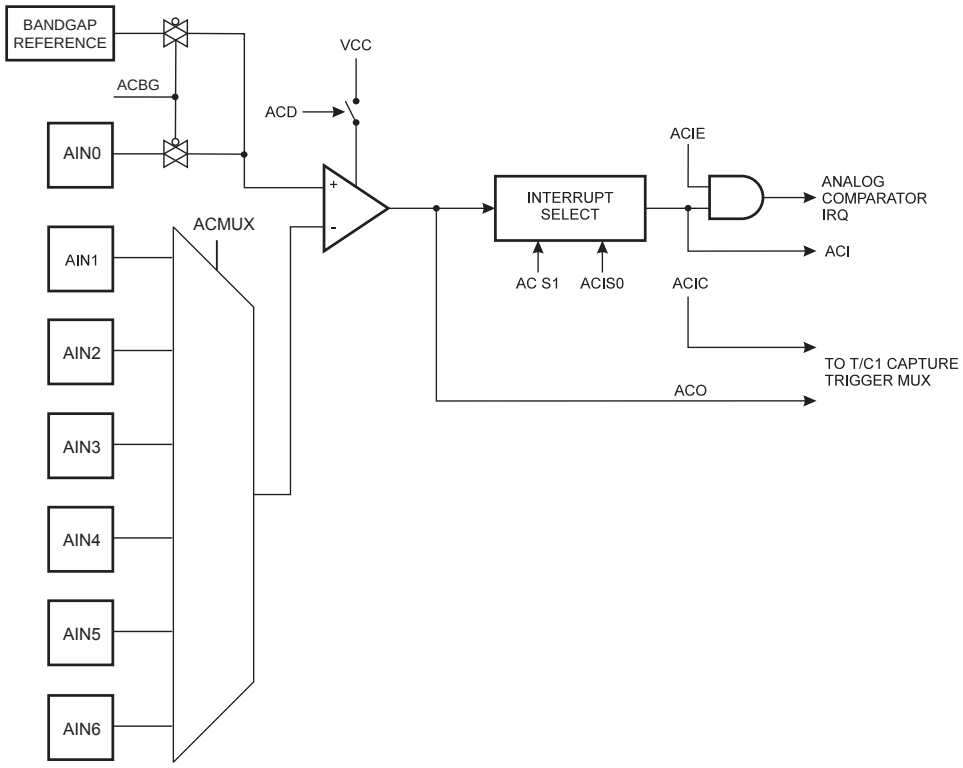
These flags are updated by the USB controller when a USB endpoint interrupt occurs (at least one bit in UEINTX set). Each bit in this field indicates which endpoint number has generated a USB endpoint interrupt request. Each one of these bits are independently cleared by hardware when their respective interrupt source is served.

22. 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. User can also replace by software the AIN0 input by the internal Bandgap reference.

Figure 22-1. Analog Comparator Block Diagram(1)



Notes: 1. Refer to Figure 1-1 on page 2 and Table 12-9 on page 79 for Analog Comparator pin placement.

22.2 Register Description

22.2.1 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. See “Internal Voltage Reference” on page 51.

- **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 22-1.

Table 22-1. 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.

22.2.2 ACMUX – Analog Comparator Input Multiplexer

Bit	7	6	5	4	3	2	1	0	
(0x7D)	–	–	–	–	–	CMUX2	CMUX1	CMUX0	ACMUX
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, 0 – CMUX2:0: Analog Comparator Selection Bits**

The value of these bits selects which combination of analog inputs are connected to the analog comparator.

The different settings are shown in Table 22-2.

Table 22-2. CMUX2:0 Settings

CMUX2	CMUX1	CMUX0	Comparator Input
0	0	0	AIN1
0	0	1	AIN2
0	1	0	AIN3
0	1	1	AIN4
1	0	0	AIN5
1	0	1	AIN6
1	1	0	Reserved
1	1	1	Reserved

22.2.3 DIDR1 – Digital Input Disable Register 1

Bit	7	6	5	4	3	2	1	0	
	–	AIN6D	AIN5D	AIN4D	AIN3D	AIN2D	AIN1D	AIN0D	DIDR1
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 1, 0 – AIN1D, AIN0D: AIN1, AIN0 Digital Input Disable**

When this bit is written logic one, the digital input buffer on the AINx 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. Boot Loader Support – Read-While-Write Self-Programming

23.1 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 25-7 on page 249) used during programming. The page organization does not affect normal operation.

23.2 Overivew

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.

23.3 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 23-2). The size of the different sections is configured by the BOOTSZ Fuses as shown in Table 23-8 on page 239 and Figure 23-2. These two sections can have different level of protection since they have different sets of Lock bits.

23.3.1 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 23-2 on page 230. The Application section can never store any Boot Loader code since the SPM instruction is disabled when executed from the Application section.

23.3.2 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 23-3 on page 230.

23.4 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 23-1 and Figure 23-1 on page 228. 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.

23.4.1 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 ongoing 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 “SPMCSR – Store Program Memory Control and Status Register” on page 242. for details on how to clear RWWSB.

23.4.2 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 23-1. 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 23-1. Read-While-Write vs. No Read-While-Write

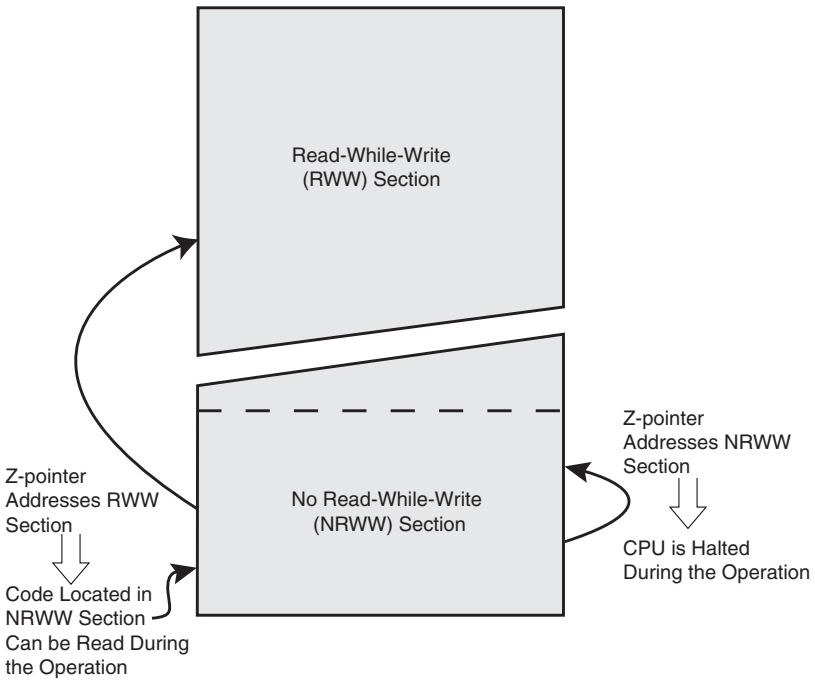
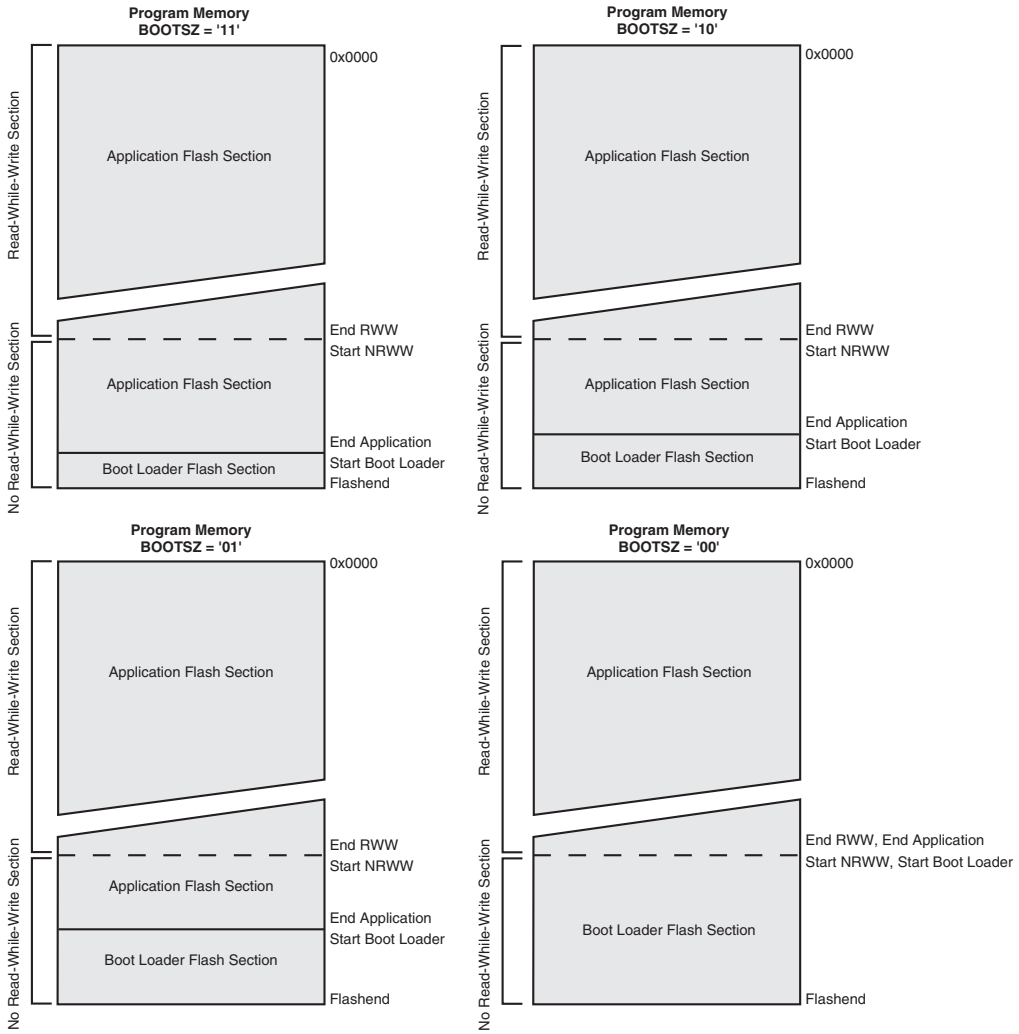


Figure 23-2. Memory Sections



Note: 1. The parameters in the figure above are given in Table 23-8 on page 239.

23.5 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 23-2 and Table 23-3 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 control the programming of the Flash memory by SPM instruction. Similarly, the general Read/Write Lock (Lock Bit mode 1) does not control reading nor writing by (E)LPM/SPM, if it is attempted.

Table 23-2. 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 23-3. 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

23.6 Entering the Boot Loader Program

The bootloader can be executed with three different conditions:

23.6.1 Regular application conditions.

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.

23.6.2 Boot Reset Fuse

The Boot Reset Fuse (BOOTRST) 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 23-4. Boot Reset Fuse(1)

BOOTRST	Reset Address
1	Reset Vector = Application Reset (address 0x0000)
0	Reset Vector = Boot Loader Reset (see Table 23-8 on page 239)

Note: 1. “1” means unprogrammed, “0” means programmed

23.6.3 External Hardware conditions

The Hardware Boot Enable Fuse (HWBE) can be programmed (See Table 23-5) so that upon special hardware conditions under reset, the bootloader execution is forced after reset.

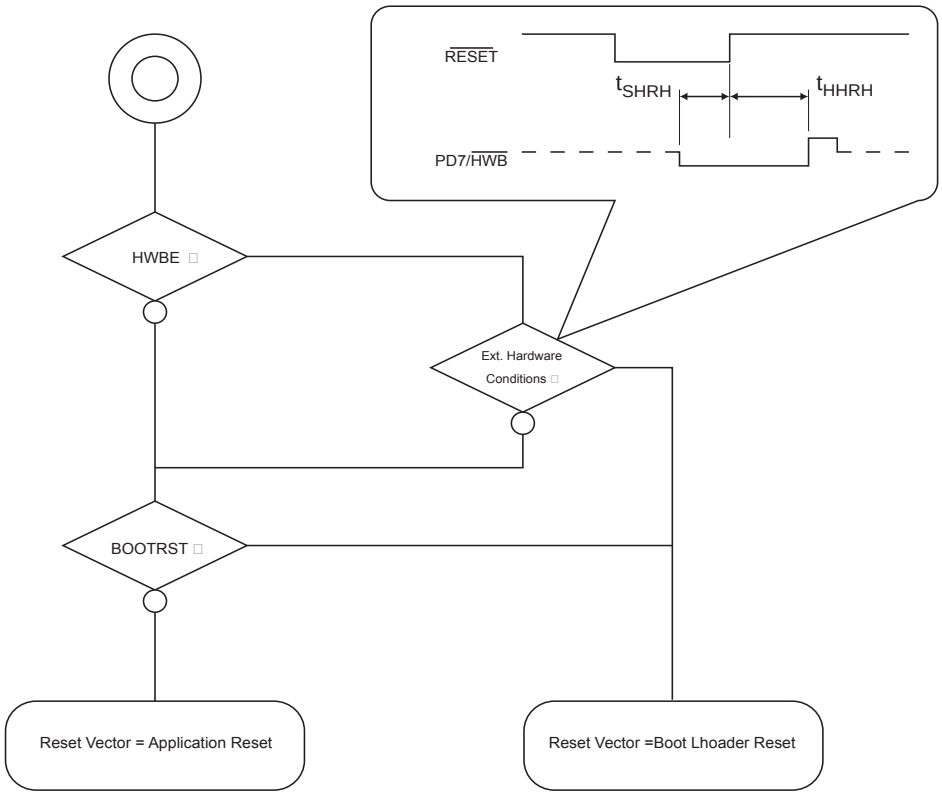
Table 23-5. Hardware Boot Enable Fuse(1)

HWBE	Reset Address
1	PD7/$\overline{\text{HWB}}$ pin can not be used to force Boot Loader execution after reset
0	PD7/$\overline{\text{HWB}}$ pin is used during reset to force bootloader execution after reset

Note: 1. “1” means unprogrammed, “0” means programmed

When the HWBE fuse is enable the PD7/$\overline{\text{HWB}}$ pin is configured as input during reset and sampled during reset rising edge. When PD7/$\overline{\text{HWB}}$ pin is ‘0’ during reset rising edge, the reset vector will be set as the Boot Loader Reset address and the Boot Loader will be executed (See Figures 23-3).

Figure 23-3. Boot Process Description



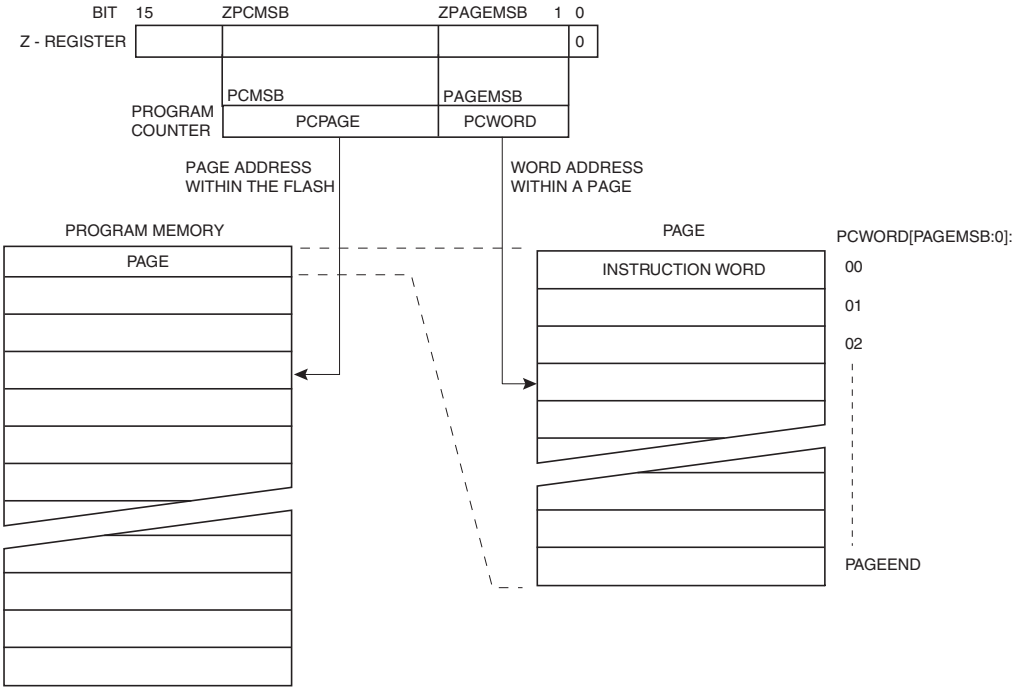
23.7 Addressing the Flash During Self-Programming

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. The number of bits actually used is implementation dependent.

Since the Flash is organized in pages (see Table 25-7 on page 249), 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 23-4. 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 (E)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 23-4. Addressing the Flash During SPM[1]



Note: 1. The different variables used in Figure 23-4 are listed in Table 23-10 on page 239.

23.8 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 “Simple Assembly Code Example for a Boot Loader” on page 237 for an assembly code example.

23.8.1 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.

23.8.2 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.

23.8.3 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.

23.8.4 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 “Interrupts” on page 64.

23.8.5 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.

23.8.6 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 “Interrupts” on page 64, or the interrupts must be disabled. Before addressing the RWW section after the programming is completed, the user software must clear the RWWSB by writing the RWWSRE. See “Simple Assembly Code Example for a Boot Loader” on

page 237 for an example.

23.8.7 Setting the Boot Loader Lock Bits by SPM

To set the Boot Loader Lock bits, write the desired data to R0, write “X0001001” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The only accessible Lock bits are the Boot Lock bits that may prevent the Application and Boot Loader section from any software update by the MCU.

Bit	7	6	5	4	3	2	1	0
R0	1	1	BLB12	BLB11	BLB02	BLB01	1	1

See Table 23-2 and Table 23-3 for how the different settings of the Boot Loader bits affect the Flash access.

If bits 5..2 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 lock bits). For future compatibility it is also recommended to set bits 7, 6, 1, and 0 in R0 to “1” when writing the Lock bits. When programming the Lock bits the entire Flash can be read during the operation.

23.8.8 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.

23.8.9 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 25-5 on page 248 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 25-4 on page 248 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 25-3 on page 247 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.

23.8.10 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 23-6 on page 236 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

ATmega8U2/16U2/32U2 includes a unique 10 bytes serial number located in the signature row. This unique serial number can be used as a USB serial number in the device enumeration process. The pointer addresses to access this unique serial number are given in Table 23-6 on page 236..

Table 23-6. 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	0x0001
Unique Serial Number	From 0x000E to 0x0018

Note: All other addresses are reserved for future use.

23.8.11 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.

23.8.12 Programming Time for Flash when Using SPM

The calibrated RC Oscillator is used to time Flash accesses. Table 23-7 shows the typical programming time for Flash accesses from the CPU.

Table 23-7. SPM Programming Time

Symbol	Min Programming Time	Max Programming Time
Flash write (Page Erase, Page Write, and write Lock bits by SPM)	3.7 ms	4.5 ms

23.8.13 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.
.equ PAGESIZEB = PAGESIZE*2   ;PAGESIZEB is page size in BYTES, not words
.org SMALLBOOTSTART
Write_page:
  ; Page Erase
  ldi  spmcrval, (1<<PGERS) | (1<<SPMEN)
  call Do_spm

  ; re-enable the RWW section
  ldi  spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call Do_spm

  ; transfer data from RAM to Flash page buffer
  ldi  looplo, low(PAGESIZEB)   ;init loop variable
  ldi  loophi, high(PAGESIZEB)  ;not required for PAGESIZEB<=256
Wrloop:
  ld   r0, Y+
  ld   r1, Y+
  ldi  spmcrval, (1<<SPMEN)
```

```
  call Do_spm
  adiw ZH:ZL, 2
  sbiw loophi:looplo, 2         ;use subi for PAGESIZEB<=256
  brne Wrloop

  ; execute Page Write
  subi ZL, low(PAGESIZEB)       ;restore pointer
  sbci ZH, high(PAGESIZEB)      ;not required for PAGESIZEB<=256
  ldi  spmcrval, (1<<PGWRT) | (1<<SPMEN)
  call Do_spm

  ; re-enable the RWW section
  ldi  spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call Do_spm

  ; read back and check, optional
  ldi  looplo, low(PAGESIZEB)   ;init loop variable
  ldi  loophi, high(PAGESIZEB)  ;not required for PAGESIZEB<=256
  subi YL, low(PAGESIZEB)       ;restore pointer
  sbci YH, high(PAGESIZEB)
Rdloop:
  elpm 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
```

23.8.14 ATmega8U2 Boot Loader Parameters

In Table 23-8 through Table 23-10, the parameters used in the description of the Self-Programming are given.

Table 23-8. Boot Size Configuration[1](Word Addresses)

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 - 0xEFF	0xF00 - 0xFFF	0xEFF	0xF00
1	0	512 words	8	0x0000 - 0xDFF	0xE00 - 0xFFF	0xDFF	0xE00
0	1	1024 words	16	0x0000 - 0xBFF	0xC00 - 0xFFF	0xBFF	0xC00
0	0	2048 words	32	0x0000 - 0x7FF	0x800 - 0xFFF	0x7FF	0x800

(Page size = 64 words = 128 bytes)

Note: 1. The different BOOTSZ Fuse configurations are shown in Figure 23-2.

Table 23-9. Read-While-Write Limit[1]

Section	Pages	Address
Read-While-Write section (RWW)	32	0x0000 - 0x07FF
No Read-While-Write section (NRWW)	32	0x0800 - 0x0FFF

Note: 1. For details about these two section, see “NRWW – No Read-While-Write Section” on page 227 and “RWW – Read-While-Write Section” on page 227.

Table 23-10. Explanation of different variables used in Figure 23-4 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 (64 words in a page requires six 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]	Z13: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 “Addressing the Flash During Self-Programming” on page 232 for details about the use of Z-pointer during Self-Programming.

23.8.15 ATmega16U2 Boot Loader Parameters

In Table 23-11 through Table 23-13, the parameters used in the description of the Self-Programming are given.

Table 23-11. Boot Size Configuration[1](Word Addresses)

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 - 0x1EFF	0x1F00 - 0x1FFF	0x1EFF	0x1F00
1	0	512 words	8	0x0000 - 0x1DFF	0x1E00 - 0x1FFF	0x1DFF	0x1E00
0	1	1024 words	16	0x0000 - 0x1BFF	0x1C00 - 0x1FFF	0x1BFF	0x1C00
0	0	2048 words	32	0x0000 - 0x17FF	0x1800 - 0x1FFF	0x17FF	0x1800

(Page size = 64 words = 128 bytes)

Note: 1. The different BOOTSZ Fuse configurations are shown in Figure 23-2.

Table 23-12. Read-While-Write Limit[1]

Section	Pages	Address
Read-While-Write section (RWW)	96	0x0000 - 0x17FF
No Read-While-Write section (NRWW)	32	0x1800 - 0x1FFF

Note: 1. For details about these two section, see "NRWW – No Read-While-Write Section" on page 227 and "RWW – Read-While-Write Section" on page 227.

Table 23-13. Explanation of different variables used in Figure 23-4 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 (64 words in a page requires six 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]	Z13: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 "Addressing the Flash During Self-Programming" on page 232 for details about the use of Z-pointer during Self-Programming.

23.8.16 ATmega32U2 Boot Loader Parameters

In Table 23-14 through Table 23-16, the parameters used in the description of the Self-Programming are given.

Table 23-14. Boot Size Configuration[1](Word Addresses)

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 - 0x3DFF	0x3E00 - 0x3FFF	0x3DFF	0x3E00
0	1	1024 words	16	0x0000 - 0x3BFF	0x3C00 - 0x3FFF	0x3BFF	0x3C00
0	0	2048 words	32	0x0000 - 0x37FF	0x3800 - 0x3FFF	0x37FF	0x3800

(Page size = 64 words = 128 bytes)

Note: 1. The different BOOTSZ Fuse configurations are shown in Figure 23-2.

Table 23-15. 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 section, see "NRWW – No Read-While-Write Section" on page 227 and "RWW – Read-While-Write Section" on page 227.

Table 23-16. Explanation of different variables used in Figure 23-4 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 (64 words in a page requires six 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]	Z13: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 "Addressing the Flash During Self-Programming" on page 232 for details about the use of Z-pointer during Self-Programming.

23.9 Register Description

23.9.1 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 “Reading the Signature Row from Software” on page 236 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 “Reading the Fuse and Lock Bits from Software” on page 235 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.

24. debugWIRE On-chip Debug System

24.1 Features

- **Complete Program Flow Control**
- **Emulates All On-chip Functions, Both Digital and Analog, except RESET Pin**
- **Real-time Operation**
- **Symbolic Debugging Support (Both at C and Assembler Source Level, or for Other HLLs)**
- **Unlimited Number of Program Break Points (Using Software Break Points)**
- **Non-intrusive Operation**
- **Electrical Characteristics Identical to Real Device**
- **Automatic Configuration System**
- **High-Speed Operation**
- **Programming of Non-volatile Memories**

24.2 Overview

The debugWIRE On-chip debug system uses a One-wire, bi-directional interface to control the program flow, execute AVR instructions in the CPU and to program the different non-volatile memories.

24.3 Physical Interface

When the debugWIRE Enable (DWEN) Fuse is programmed and Lock bits are unprogrammed, the debugWIRE system within the target device is activated. The RESET port pin is configured as a wire-AND (open-drain) bi-directional I/O pin and becomes the communication gateway between target and emulator.

Figure 24-1. The debugWIRE Setup

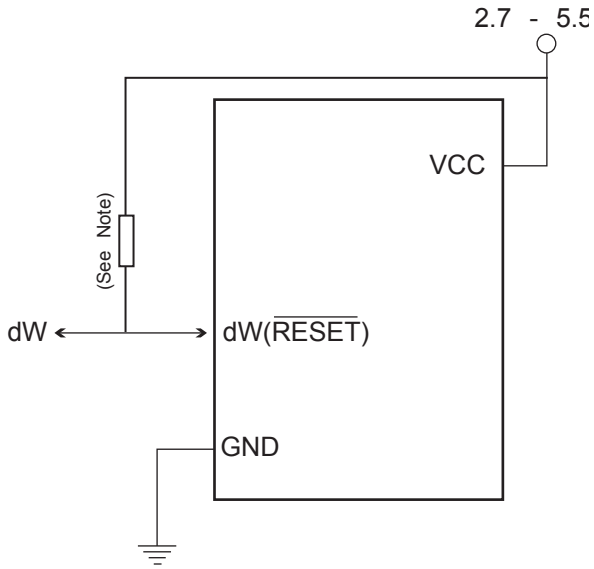


Figure 24-1 shows the schematic of a target MCU, with debugWIRE enabled, and the emulator connector. The system clock is not affected by debugWIRE and will always be the clock source selected by the CKSEL Fuses.

When designing a system where debugWIRE will be used, the following observations must be made for correct operation:

- Connecting the RESET pin directly to V_{CC} will not work.
- Any capacitors (or additionnal circuitry) connected to the RESET pin must be disconnected when using debugWire.
- All external reset sources must be disconnected.

Note: some releases of JTAG Ice mkII firmware may require a pull-up resistor with a value between 8 and 14 kOhms when operating at 5V.

24.4 Software Break Points

debugWIRE supports Program memory Break Points by the AVR Break instruction. Setting a Break Point in AVR Studio® will insert a BREAK instruction in the Program memory. The instruction replaced by the BREAK instruction will be stored. When program execution is continued, the stored instruction will be executed before continuing from the Program memory. A break can be inserted manually by putting the BREAK instruction in the program.

The Flash must be re-programmed each time a Break Point is changed. This is automatically handled by AVR Studio through the debugWIRE interface. The use of Break Points will therefore reduce the Flash Data retention. Devices used for debugging purposes should not be shipped to end customers.

24.5 Limitations of debugWIRE

The debugWIRE communication pin (dW) is physically located on the same pin as External Reset (RESET). An External Reset source is therefore not supported when the debugWIRE is enabled.

The debugWIRE system accurately emulates all I/O functions when running at full speed, i.e., when the program in the CPU is running. When the CPU is stopped, care must be taken while accessing some of the I/O Registers via the debugger (AVR Studio).

A programmed DWEN Fuse enables some parts of the clock system to be running in all sleep modes. This will increase the power consumption while in sleep. Thus, the DWEN Fuse should be disabled when debugWire is not used.

24.6 Register Description

24.6.1 DWDR – debugWire Data Register

Bit	7	6	5	4	3	2	1	0	
0x31 (0x51)	DWDR[7:0]								DWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The DWDR Register provides a communication channel from the running program in the MCU to the debugger. This register is only accessible by the debugWIRE and can therefore not be used as a general purpose register in the normal operations.

25. Memory Programming

25.1 Program And Data Memory Lock Bits

The ATmega8U2/16U2/32U2 provides six Lock bits which can be left unprogrammed (“1”) or can be programmed (“0”) to obtain the additional features listed in Table 25-2. The Lock bits can only be erased to “1” with the Chip Erase command.

Table 25-1. 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	0 (programmed)
BLB02	3	Boot Lock bit	1 (unprogrammed)
BLB01	2	Boot Lock bit	1 (unprogrammed)
LB2	1	Lock bit	0 (programmed)
LB1	0	Lock bit	0 (programmed)

Note: 1. “1” means unprogrammed, “0” means programmed

Table 25-2. Lock Bit Protection Modes(1)(2)

Memory Lock Bits			Protection Type
LB Mode	**LB2**	**LB1**	
1	1	1	No memory lock features enabled.
2	1	0	Further programming of the Flash and EEPROM is disabled in Parallel and Serial Programming mode. The Fuse bits are locked in both Serial and Parallel Programming mode.(1)
3	0	0	Further programming and verification of the Flash and EEPROM is disabled in Parallel and Serial Programming mode. The 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.

Table 25-2. 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

25.2 Fuse Bits

The ATmega8U2/16U2/32U2 has three Fuse bytes. Table 25-3 - Table 25-5 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 25-3. Extended Fuse Byte

Fuse Low Byte	Bit No	Description	Default Value : 0xF4
–	7	–	1
–	6	–	1
–	5	–	1
–	4	–	1
HWBE	3	Hardware Boot Enable	0 (programmed)
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	0 (programmed)

Note: 1. See "System and Reset Characteristics" on page 267 for BODLEVEL Fuse decoding.

Table 25-4. Fuse High Byte

Fuse High Byte	Bit No	Description	Default Value : 0xD9
DWEN[4]	7	Enable debugWIRE (and disable Reset capability	1 (unprogrammed, debugWIRE disabled)
RSTDSBL	6	Disable Reset (pin can be used as general purpose I/O)	1 (unprogrammed, Reset enabled)
SPIEN[1]	5	Enable Serial Program and Data Downloading	0 (programmed, SPI prog. enabled)
WDTON[3]	4	Watchdog Timer always ON	1 (unprogrammed)[3]
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed, EEPROM not preserved)
BOOTSZ1	2	Select Boot Size (see Table 25-9 for details)	0 (programmed)[2]
BOOTSZ0	1	Select Boot Size (see Table 25-9 for details)	0 (programmed)[2]
BOOTRST	0	Select Bootloader Address as Reset Vector	1 (unprogrammed, Reset vector @0x0000)

Note: 1. The SPIEN Fuse is not accessible in serial programming mode.
2. The default value of BOOTSZ1..0 results in maximum Boot Size. See Table 23-8 on page 239 for details.
3. See "WDTCSR – Watchdog Timer Control Register" on page 56 for details.
4. Never ship a product with the DWEN Fuse programmed regardless of the setting of Lock bits and RSTDSBL Fuse. A programmed DWEN Fuse enables some parts of the clock system to be running in all sleep modes. This may increase the power consumption.

Table 25-5. Fuse Low Byte

Fuse Low Byte	Bit No	Description	Default Value : 0x5E
CKDIV8[4]	7	Divide clock by 8	0 (programmed)
CKOUT[3]	6	Clock output	1 (unprogrammed)
SUT1	5	Select start-up time	0 (programmed)[1]
SUT0	4	Select start-up time	1 (unprogrammed)[1]
CKSEL3	3	Select Clock source	1 (unprogrammed)[2]
CKSEL2	2	Select Clock source	1 (unprogrammed)[2]
CKSEL1	1	Select Clock source	1 (unprogrammed)[2]
CKSEL0	0	Select Clock source	0 (programmed)[2]

Note: 1. The default value of SUT1..0 results in maximum start-up time for the default clock source. See "System and Reset Characteristics" on page 267 for details.
2. The default setting of CKSEL3..0 results in External crystal Oscillator 8MHz. See Table 8-1 on page 29 for details.
3. The CKOUT Fuse allow the system clock to be output on PORTC7. See "Clock Output Buffer" on page 35 for details.
4. See "System Clock Prescaler" on page 35 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.

25.2.1 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.

25.3 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 ATmega8U2/16U2/32U2 the signature bytes are given in Table 25-6.

Table 25-6. Device and JTAG ID

	Signature Bytes Address			JTAG	
Part	0x000	0x001	0x002	Part Number	Manufacture ID
ATmega8U2	0x1E	0x93	0x89	9389	0x1F
ATmega16U2	0x1E	0x94	0x89	9489	0x1F
ATmega32U2	0x1E	0x95	0x8A	958A	0x1F

25.4 Calibration Byte

The ATmega8U2/16U2/32U2 has a byte calibration value for the internal RC Oscillator. This byte resides in the high byte of address 0x000 in the signature address space. During reset, this byte is automatically written into the OSCCAL Register to ensure correct frequency of the calibrated RC Oscillator.

25.5 Page Size

Table 25-7. No. of Words in a Page and No. of Pages in the Flash

Device	Flash Size	Page Size	PCWORD	No. of Pages	PCPAGE	PCMSB
ATmega8U2	4K words (8Kbytes)	32 words	PC[4:0]	128	PC[11:6]	11
ATmega16U2	8K words (16Kbytes)	64 words	PC[5:0]	128	PC[12:6]	12
ATmega32U2	16K words (32Kbytes)	64 words	PC[5:0]	256	PC[13:6]	13

Table 25-8. No. of Bytes in a Page and No. of Pages in the EEPROM

Device	EEPROM Size	Page Size	PCWORD	No. of Pages	PCPAGE	EEAMSB
ATmega8U2	256 bytes	4 bytes	EEA[1:0]	128	EEA[8:2]	8
ATmega16U2	512 bytes	4 bytes	EEA[1:0]	128	EEA[8:2]	8
ATmega32U2	1K bytes	4 bytes	EEA[1:0]	256	EEA[9:2]	9

25.6 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 ATmega8U2/16U2/32U2. Pulses are assumed to be at least 250 ns unless otherwise noted.

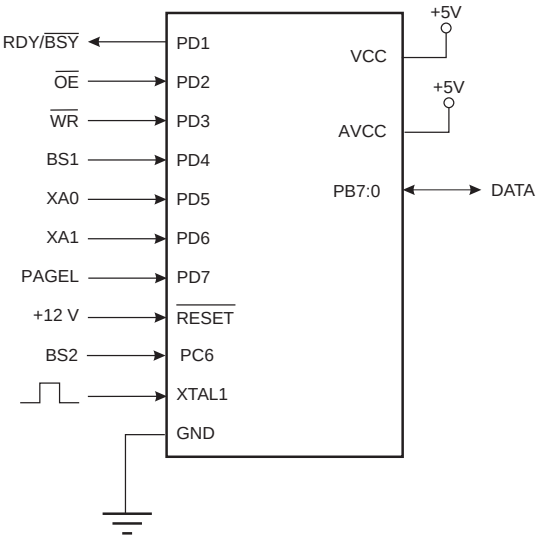
25.6.1 Signal Names

In this section, some pins of the ATmega8U2/16U2/32U2 are referenced by signal names describing their functionality during parallel programming, see Figure 25-1 and Table 25-9. 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 25-12.

When pulsing $\overline{WR}$ or $\overline{OE}$, the command loaded determines the action executed. The different commands are shown in Table 25-13.

Figure 25-1. Parallel Programming(1)



Note: 1. Unused Pins should be left floating.

Table 25-9. Pin Name Mapping

Signal Name in Programming Mode	Pin Name	I/O	Function
RDY/$\overline{BSY}$	PD1	O	0: Device is busy programming, 1: Device is ready for new command.
$\overline{OE}$	PD2	I	Output Enable (Active low).
$\overline{WR}$	PD3	I	Write Pulse (Active low).
BS1	PD4	I	Byte Select 1.
XA0	PD5	I	XTAL Action Bit 0
XA1	PD6	I	XTAL Action Bit 1

Table 25-9. Pin Name Mapping

Signal Name in Programming Mode	Pin Name	I/O	Function
PAGEL	PD7	I	Program Memory and EEPROM data Page Load.
BS2	PC6	I	Byte Select 2.
DATA	PB7-0	I/O	Bi-directional Data bus (Output when $\overline{OE}$ is low).

Table 25-10. 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	Lockbits
1	0	Extended High Byte	Reserved	Extended Byte	Extended Fuse Byte
1	1	Reserved	Reserved	Reserved	Fuse High Byte

Table 25-11. 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 25-12. 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 25-13. 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

Table 25-13. Command Byte Bit Encoding

Command Byte	Command Executed
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

25.7 Parallel Programming

25.7.1 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 25-11 on page 251 to “0000” and wait at least 100 ns.
4. Apply 11.5 - 12.5V to $\overline{\text{RESET}}$. Any activity on Prog_enable pins within 100 ns after +12V has been applied to $\overline{\text{RESET}}$, will cause the device to fail entering programming mode.
5. Wait at least 50 µs before sending a new command.

25.7.2 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.

25.7.3 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.

25.7.4 Programming the Flash

The Flash is organized in pages, see Table 25-7 on page 249. 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 25-3 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 25-2 on page 254. Note that if less than eight bits are required to address words in the page (pagesize < 256), the most significant bit(s) in the address low byte are used to address the page when performing a Page Write.

G. Load Address High byte (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

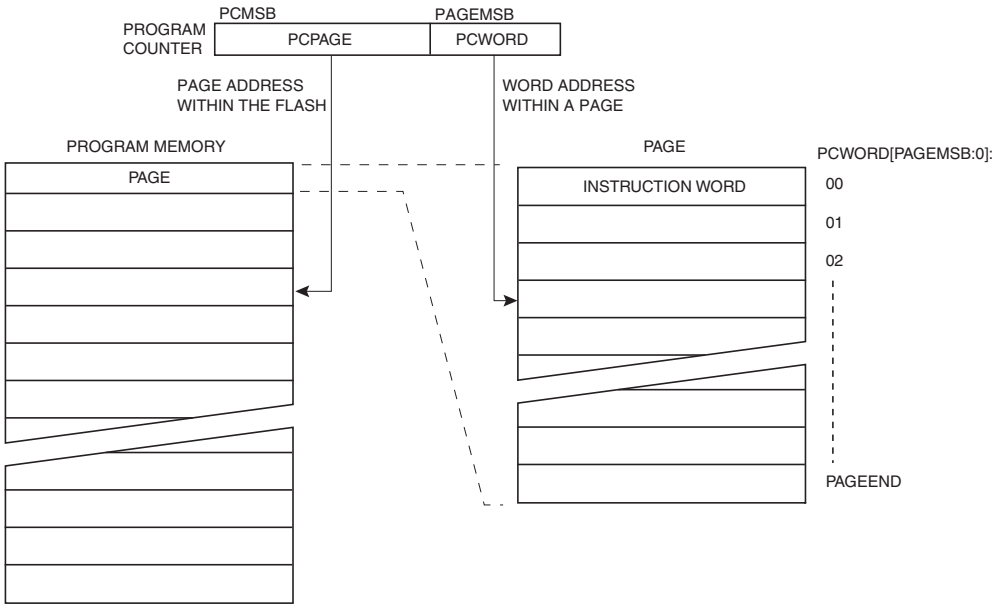
1. Set BS2, BS1 to “00”
2. Give $\overline{WR}$ a negative pulse. This starts programming of the entire page of data. RDY/$\overline{BSY}$ goes low.
3. Wait until RDY/$\overline{BSY}$ goes high (See Figure 25-3 for signal waveforms).

J. Repeat B through I until the entire Flash is programmed or until all data has been programmed.

K. End Page Programming

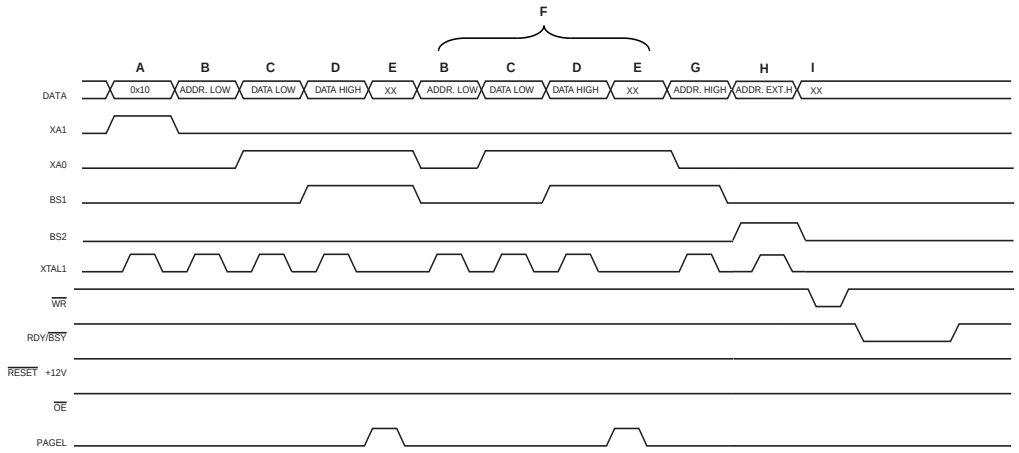
1. 1. Set XA1, XA0 to “10”. This enables command loading.
2. Set DATA to “0000 0000”. This is the command for No Operation.
3. Give XTAL1 a positive pulse. This loads the command, and the internal write signals are reset.

Figure 25-2. Addressing the Flash Which is Organized in Pages(1)



Note: 1. PCPAGE and PCWORD are listed in Table 25-7 on page 249.

Figure 25-3. Programming the Flash Waveforms[1]



Note: 1. “XX” is don’t care. The letters refer to the programming description above.

25.7.5 Programming the EEPROM

The EEPROM is organized in pages, see Table 25-8 on page 249. When programming the EEPROM, the program data is latched into a page buffer. This allows one page of data to be programmed simultaneously. The programming algorithm for the EEPROM data memory is as follows (refer to “Programming the Flash” on page 253 for details on Command, Address and Data loading):

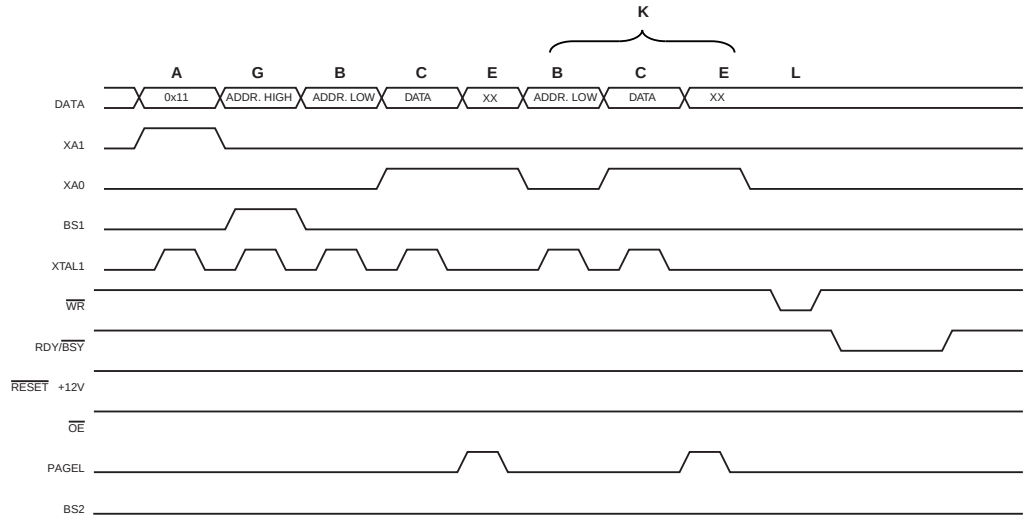
1. A: Load Command “0001 0001”.
2. G: Load Address High Byte (0x00 - 0xFF).
3. B: Load Address Low Byte (0x00 - 0xFF).
4. C: Load Data (0x00 - 0xFF).
5. E: Latch data (give PAGEL a positive pulse).

K: Repeat 3 through 5 until the entire buffer is filled.

L: Program EEPROM page

1. Set BS2, BS1 to “00”.
2. Give $\overline{WR}$ a negative pulse. This starts programming of the EEPROM page. RDY/$\overline{BSY}$ goes low.
3. Wait until to RDY/$\overline{BSY}$ goes high before programming the next page (See Figure 25-4 for signal waveforms).

Figure 25-4. Programming the EEPROM Waveforms



25.7.6 Reading the Flash

The algorithm for reading the Flash memory is as follows (refer to “Programming the Flash” on page 253 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 $\overline{OE}$ 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 $\overline{OE}$ to “1”.

25.7.7 Reading the EEPROM

The algorithm for reading the EEPROM memory is as follows (refer to “Programming the Flash” on page 253 for details on Command and Address loading):

1. A: Load Command “0000 0011”.
2. G: Load Address High Byte (0x00 - 0xFF).
3. B: Load Address Low Byte (0x00 - 0xFF).
4. Set $\overline{OE}$ to “0”, and BS1 to “0”. The EEPROM Data byte can now be read at DATA.
5. Set $\overline{OE}$ to “1”.

25.7.8 Programming the Fuse Low Bits

The algorithm for programming the Fuse Low bits is as follows (refer to “Programming the Flash” on page 253 for details on Command and Data loading):

1. A: Load Command “0100 0000”.
2. C: Load Data Low Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.

25.7.9 Programming the Fuse High Bits

The algorithm for programming the Fuse High bits is as follows (refer to “Programming the Flash” on page 253 for details on Command and Data loading):

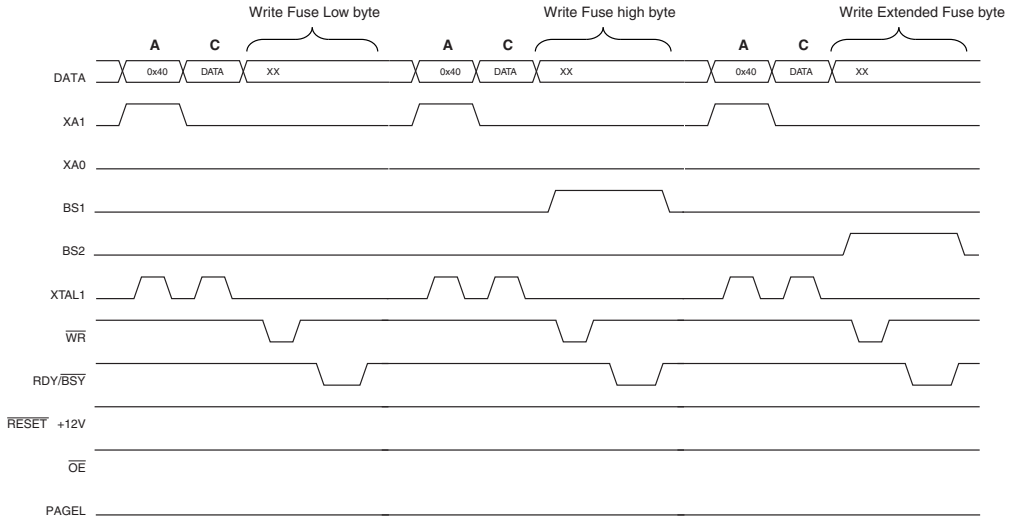
1. A: Load Command “0100 0000”.
2. C: Load Data Low Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Set BS2, BS1 to “01”. This selects high data byte.
4. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.
5. Set BS2, BS1 to “00”. This selects low data byte.

25.7.10 Programming the Extended Fuse Bits

The algorithm for programming the Extended Fuse bits is as follows (refer to “Programming the Flash” on page 253 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 $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.
5. 5. Set BS2, BS1 to “00”. This selects low data byte.

Figure 25-5. Programming the FUSES Waveforms



25.7.11 Programming the Lock Bits

The algorithm for programming the Lock bits is as follows (refer to “Programming the Flash” on page 253 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.

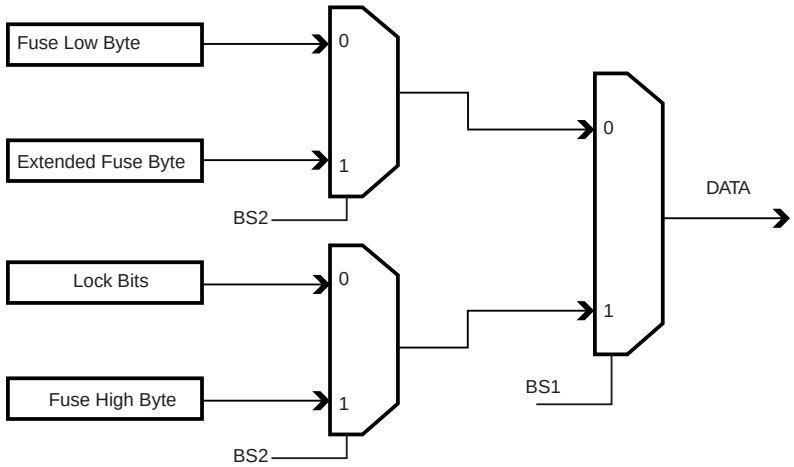
The Lock bits can only be cleared by executing Chip Erase.

25.7.12 Reading the Fuse and Lock Bits

The algorithm for reading the Fuse and Lock bits is as follows (refer to "Programming the Flash" on page 253 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 25-6. Mapping Between BS1, BS2 and the Fuse and Lock Bits During Read



25.7.13 Reading the Signature Bytes

The algorithm for reading the Signature bytes is as follows (refer to "Programming the Flash" on page 253 for details on Command and Address loading):

1. A: Load Command "0000 1000".
2. B: Load Address Low Byte (0x00 - 0x02).
3. Set $\overline{OE}$ to "0", and BS to "0". The selected Signature byte can now be read at DATA.
4. Set $\overline{OE}$ to "1".

25.7.14 Reading the Calibration Byte

The algorithm for reading the Calibration byte is as follows (refer to "Programming the Flash" on page 253 for details on Command and Address loading):

1. A: Load Command "0000 1000".
2. B: Load Address Low Byte, 0x00.
3. Set $\overline{OE}$ to "0", and BS1 to "1". The Calibration byte can now be read at DATA.
4. Set $\overline{OE}$ to "1".

25.8 Serial Downloading

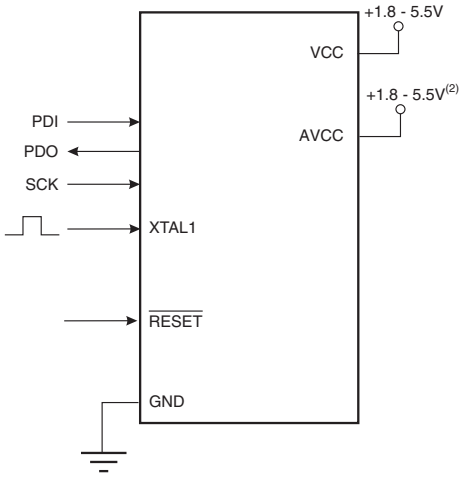
Both the Flash and EEPROM memory arrays can be programmed using a serial programming bus while $\overline{\text{RESET}}$ is pulled to GND. The serial programming interface consists of pins SCK, PDI (input) and PDO (output). After $\overline{\text{RESET}}$ is set low, the Programming Enable instruction needs to be executed first before program/erase operations can be executed. NOTE, in Table 25-14 on page 259, the pin mapping for serial programming is listed. Not all packages use the SPI pins dedicated for the internal Serial Peripheral Interface - SPI.

25.9 Serial Programming Pin Mapping

Table 25-14. Pin Mapping Serial Programming

Symbol	Pins	I/O	Description
PDI	PB2	I	Serial Data in
PDO	PB3	O	Serial Data out
SCK	PB1	I	Serial Clock

Figure 25-7. Serial Programming and Verify(1)



Notes: 1. If the device is clocked by the internal Oscillator, it is no need to connect a clock source to the XTAL1 pin.

2. V_{CC} - 0.3V < AVCC < V_{CC} + 0.3V, however, AVCC should always be within 1.8 - 5.5V

When programming the EEPROM, an auto-erase cycle is built into the self-timed programming operation (in the Serial mode ONLY) and there is no need to first execute the Chip Erase instruction. The Chip Erase operation turns the content of every memory location in both the Program and EEPROM arrays into 0xFF.

Depending on CKSEL Fuses, a valid clock must be present. The minimum low and high periods for the serial clock (SCK) input are defined as follows:

Low: > 2 CPU clock cycles for f_{ck} < 12 MHz, 3 CPU clock cycles for f_{ck} >= 12 MHz

High: > 2 CPU clock cycles for f_{ck} < 12 MHz, 3 CPU clock cycles for f_{ck} >= 12 MHz

25.9.1 Serial Programming Algorithm

When writing serial data to the ATmega8U2/16U2/32U2, data is clocked on the rising edge of SCK.

When reading data from the ATmega8U2/16U2/32U2, data is clocked on the falling edge of SCK. See Figure 25-8 for timing details.

To program and verify the ATmega8U2/16U2/32U2 in the serial programming mode, the following sequence is recommended (See four byte instruction formats in Table 25-16):

1. Power-up sequence:
   Apply power between V_{CC} and GND while $\overline{\text{RESET}}$ and SCK are set to “0”. In some systems, the programmer can not guarantee that SCK is held low during power-up. In this case, $\overline{\text{RESET}}$ must be given a positive pulse of at least two CPU clock cycles duration after SCK has been set to “0”.
2. Wait for at least 20 ms and enable serial programming by sending the Programming Enable serial instruction to pin PDI.
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 to be issued for the first page, since the memory size is not larger than 64KWord. 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 25-15.) 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 25-15.) 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 PDO. 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 to be issued for the first page, since the memory size is not larger than 64KWord.
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 25-15. Minimum Wait Delay Before Writing the Next Flash or EEPROM Location

Symbol	Minimum Wait Delay
t_{WD_FLASH}	4.5 ms
t_{WD_EEPROM}	9.0 ms
t_{WD_ERASE}	9.0 ms

Figure 25-8. Serial Programming Waveforms

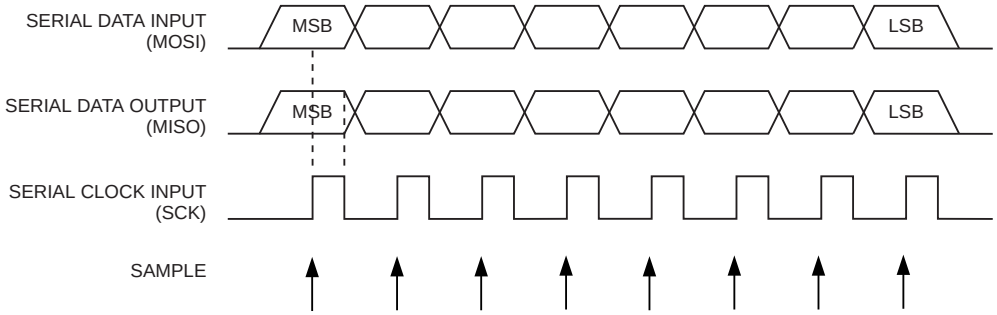


Table 25-16. Serial Programming Instruction Set

Instruction	Instruction Format				Operation
	Byte 1	Byte 2	Byte 3	Byte4	
Programming Enable	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	Enable Serial Programming after $\overline{\text{RESET}}$ goes low.
Chip Erase	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	Chip Erase EEPROM and Flash.
Load Extended Address Byte	0100 1101	0000 0000	cccc cccc	xxxx xxxx	Defines Extended Address Byte for Read Program Memory and Write Program Memory Page.
Read Program Memory	0010 **H**000	**aaaa aaaa**	**bbbb bbbb**	**oooo oooo**	Read **H** (high or low) data **o** from Program memory at word address **c**:**a**:**b**.
Load Program Memory Page	0100 **H**000	xxxx xxxx	xx**bb bbbb**	**iiii iiii**	Write **H** (high or low) data **i** to Program Memory page at word address **b**. Data low byte must be loaded before Data high byte is applied within the same address.
Write Program Memory Page	0100 1100	**aaaa aaaa**	**bb**xx xxxx	xxxx xxxx	Write Program Memory Page at address **c**:**a**:**b**.
Read EEPROM Memory	1010 0000	0000 **aaaa**	**bbbb bbbb**	**oooo oooo**	Read data **o** from EEPROM memory at address **a**:**b**.
Write EEPROM Memory	1100 0000	0000 **aaaa**	**bbbb bbbb**	**iiii iiii**	Write data **i** to EEPROM memory at address **a**:**b**.
Load EEPROM Memory Page (page access)	1100 0001	0000 0000	0000 00**bb**	**iiii iiii**	Load data **i** to EEPROM memory page buffer. After data is loaded, program EEPROM page.
Write EEPROM Memory Page (page access)	1100 0010	0000 **aaaa**	**bbbb bb00**	xxxx xxxx	Write EEPROM page at address **a**:**b**.
Read Lock bits	0101 1000	0000 0000	xxxx xxxx	xx**oo oooo**	Read Lock bits. “0” = programmed, “1” = unprogrammed. See Table 25-1 on page 246 for details.
Write Lock bits	1010 1100	111x xxxx	xxxx xxxx	11**ii iiii**	Write Lock bits. Set bits = “0” to program Lock bits. See Table 25-1 on page 246 for details.
Read Signature Byte	0011 0000	000x xxxx	xxxx xx**bb**	**oooo oooo**	Read Signature Byte **o** at address **b**.
Write Fuse bits	1010 1100	1010 0000	xxxx xxxx	**iiii iiii**	Set bits = “0” to program, “1” to unprogram.
Write Fuse High bits	1010 1100	1010 1000	xxxx xxxx	**iiii iiii**	Set bits = “0” to program, “1” to unprogram.
Write Extended Fuse Bits	1010 1100	1010 0100	xxxx xxxx	**iiii iiii**	Set bits = “0” to program, “1” to unprogram. See Table 25-3 on page 247 for details.
Read Fuse bits	0101 0000	0000 0000	xxxx xxxx	**oooo oooo**	Read Fuse bits. “0” = programmed, “1” = unprogrammed.
Read Fuse High bits	0101 1000	0000 1000	xxxx xxxx	**oooo oooo**	Read Fuse High bits. “0” = programmed, “1” = unprogrammed.

Table 25-16. Serial Programming Instruction Set (Continued)

Instruction	Instruction Format				Operation
	Byte 1	Byte 2	Byte 3	Byte4	
Read Extended Fuse Bits	0101 0000	0000 1000	xxxx xxxx	**oooo oooo**	Read Extended Fuse bits. “0” = programmed, “1” = unprogrammed. See Table 25-3 on page 247 for details.
Read Calibration Byte	0011 1000	000x xxxx	0000 0000	**oooo oooo**	Read Calibration Byte
Poll RDY/$\overline{\text{BSY}}$	1111 0000	0000 0000	xxxx xxxx	xxxx xxx**o**	If **o** = “1”, a programming operation is still busy. Wait until this bit returns to “0” before applying another command.

Note: **a** = address high bits, **b** = address low bits, **c** = address extended bits, **H** = 0 - Low byte, 1 - High Byte, **o** = data out, **i** = data in, x = don’t care

25.9.2 Serial Programming Characteristics

For characteristics of the Serial Programming module see “SPI Timing Characteristics” on page 269.

26. Electrical Characteristics

26.1 Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except $\overline{\text{RESET}}$ & UVcc with respect to Ground[7]	-0.5V to V_{CC}+0.5V
Voltage on $\overline{\text{RESET}}$ with respect to Ground	-0.5V to +13.0V
Voltage on UVcc with respect to Ground	-0.5V to +6.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0 mA
DC Current V_{CC} and GND Pins	200.0 mA

*NOTICE: Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

26.2 DC Characteristics

T_A = -40°C to 85°C, V_{CC} = 2.7V to 5.5V (unless otherwise noted)

Symbol	Parameter	Condition	Min.[5]	Typ.	Max.[5]	Units
V_{IL}	Input Low Voltage, Standard IOs[8]	V_{CC} = 2.7V - 5.5V	-0.5		0.8	V
V_{IL1}	Input Low Voltage, XTAL1 pin	V_{CC} = 2.7V - 5.5V	-0.5		$0.1V_{CC}$[1]	V
V_{IL2}	Input Low Voltage, RESET pin	V_{CC} = 2.7V - 5.5V	-0.5		$0.1V_{CC}$[1]	V
V_{IH}	Input High Voltage, Standard IOs[8]	V_{CC} = 2.7V - 5.5V	2		V_{CC} + 0.5	V
V_{IH1}	Input High Voltage, XTAL1 pin	V_{CC} = 2.7V - 5.5V	$0.7V_{CC}$[2]		V_{CC} + 0.5	V
V_{IH2}	Input High Voltage, RESET pin	V_{CC} = 2.7V - 5.5V	$0.9V_{CC}$[2]		V_{CC} + 0.5	V
V_{OL}	Output Low Voltage[3], Standard IOs[8], MOSI/MISO pins	I_{OL} = 10mA, V_{CC} = 5V I_{OL} = 5mA, V_{CC} = 3V			0.7 0.5	V
V_{OH}	Output High Voltage[4], Standard IOs[8], MOSI/MISO pins	I_{OH} = -10mA, V_{CC} = 5V I_{OH} = -5mA, V_{CC} = 3V	4.2 2.3			V
I_{IL}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin low (absolute value)			1	µA
I_{IH}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin high (absolute value)			1	µA
R_{RST}	Reset Pull-up Resistor		30		60	kΩ
R_{PU}	I/O Pin Pull-up Resistor		20		50	kΩ

T_A = -40°C to 85°C, V_{CC} = 2.7V to 5.5V (unless otherwise noted) (Continued)

Symbol	Parameter	Condition	Min.[5]	Typ.	Max.[5]	Units
R_{PUDP}	USB D+ Internal Pull-Up	Idle mode	900		1500	Ω
		Streaming mode	1425		3090	Ω
I_{CC}	Power Supply Current[6]	Active 8 MHz, V_{CC} = 3V regulator disabled		4	6	mA
		Active 16 MHz, V_{CC} = 5V regulator enabled		13.5	21	mA
		Idle 8 MHz, V_{CC} = 3V regulator disabled		0.8	1.2	mA
		Idle 16 MHz, V_{CC} = 5V regulator enabled		3.2	4.0	mA
	Power-down mode	WDT disabled, regulator disabled,V_{CC} = 3V		5	10	µA
		WDT enabled, regulator disabled,V_{CC} = 3V		10	15	µA
		WDT, BOD, regulator enabled, Vcc = 5V		40	65	µA
	Standby mode - 8MHZ XTAL	WDT disabled, BOD Enabled, regulator disabled, Vcc = 3V		250		µA
		WDT disabled, BOD, regulator enabled, Vcc = 5V		350		µA
V_{ACIO}	Analog Comparator Input Offset Voltage	V_{CC} = 5V V_{in} = V_{CC}/2		<10	40	mV
I_{ACLK}	Analog Comparator Input Leakage Current	V_{CC} = 5V V_{in} = V_{CC}/2	-50		50	nA
t_{ACID}	Analog Comparator Propagation Delay	V_{CC} = 2.7V V_{CC} = 4.0V		750 500		ns
Rusb	USB Series resistor (external)			22±5%		Ω
Vreg	Regulator Output Voltage	Vcc≥4.0V, I≤100mA, C_{UCAP}=1µF±20%	3.0	3.3	3.6	V
UVcc			4		5.5	V

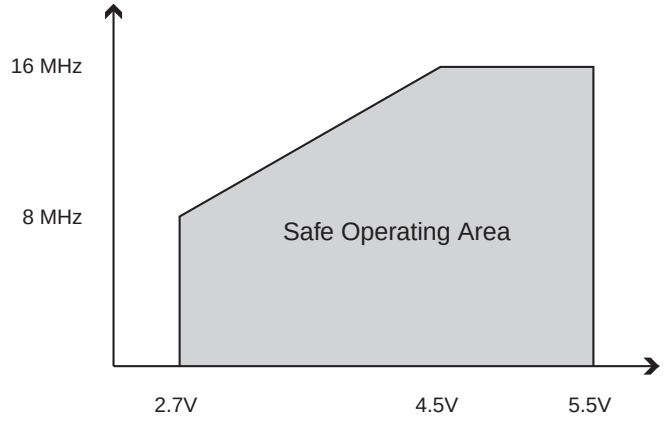
Note:
1. "Max" means the highest value where the pin is guaranteed to be read as low
2. "Min" means the lowest value where the pin is guaranteed to be read as high
3. Although each I/O port can sink more than the test conditions (20mA at VCC = 5V, 10mA at VCC = 3V) under steady state conditions (non-transient), the following must be observed:
   1.)The sum of all IOL, for ports B0-B7, C0-C7, D0-D7 should not exceed 150 mA.
   If IOL exceeds the test condition, VOL may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test condition.
4. Although each I/O port can source more than the test conditions (20mA at VCC = 5V, 10mA at VCC = 3V) under steady state conditions (non-transient), the following must be observed:
   1.)The sum of all IOL, for ports B0-B7, C0-C7, D0-D7 should not exceed 150 mA.
   If IOH exceeds the test condition, VOH may exceed the related specification. Pins are not guaranteed to source current greater than the listed test condition.

5. All DC Characteristics contained in this datasheet are based on simulation and characterization of other AVR microcontrollers manufactured in the same process technology. These values are preliminary values representing design targets, and will be updated after characterization of actual silicon
6. Values with “PRR1 – Power Reduction Register 1” disabled (0x00).
7. As specified in the USB Electrical chapter, the D+/D- pads can withstand voltages down to -1V applied through a 39Ω resistor (in series with the external 39Ω resistor).
8. All IOs Except XTAL1 and Reset pins

26.3 Speed Grades

Maximum frequency is depending on V_{CC}. As shown in Figure 26-1, the Maximum Frequency vs. V_{CC} curve is linear between 2.7V < V_{CC} < 4.5V.

Figure 26-1. Maximum Frequency vs. V_{CC}, ATmega8U2/16U2/32U2



26.4 Clock Characteristics

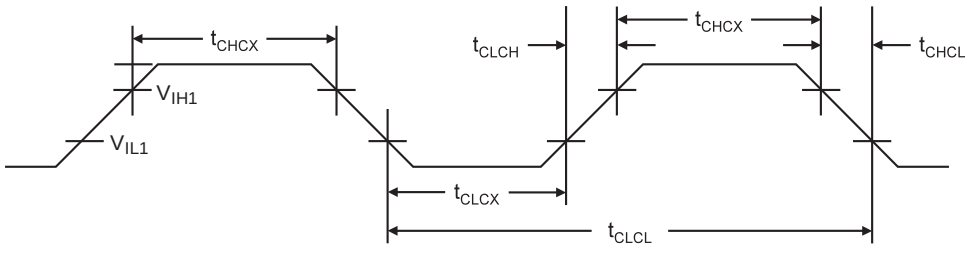
26.4.1 Calibrated Internal RC Oscillator Accuracy

Table 26-1. Calibration Accuracy of Internal RC Oscillator

	Frequency	V_{CC}	Temperature	Calibration Accuracy
Factory Calibration	8.0 MHz	3V	25°C	±10%
User Calibration	7.3 - 8.1 MHz	2.7V - 5.5V	-40°C - 85°C	±1%

26.4.2 External Clock Drive Waveforms

Figure 26-2. External Clock Drive Waveforms



26.4.3 External Clock Drive

Table 26-2. External Clock Drive

Symbol	Parameter	V_{CC}=2.7-5.5V		V_{CC}=4.5-5.5V		Units
		Min.	Max.	Min.	Max.	
$1/t_{CLCL}$	Oscillator Frequency	0	8	0	16	MHz
t_{CLCL}	Clock Period	125		62.5		ns
t_{CHCX}	High Time	50		25		ns
t_{CLCX}	Low Time	50		25		ns
t_{CLCH}	Rise Time		1.6		0.5	µs
t_{CHCL}	Fall Time		1.6		0.5	µs
Δt_{CLCL}	Change in period from one clock cycle to the next		2		2	%

Note: All DC Characteristics contained in this datasheet are based on simulation and characterization of other AVR microcontrollers manufactured in the same process technology. These values are preliminary values representing design targets, and will be updated after characterization of actual silicon.

26.5 System and Reset Characteristics

Table 26-3. Reset, Brown-out and Internal Voltage Reference Characteristics

Symbol	Parameter	Condition	Min	Typ	Max	Units
V_{POT}	Power-on Reset Threshold Voltage (rising)			1.4	2.3	V
	Power-on Reset Threshold Voltage (falling)(Note:)			1.3	2.3	V
V_{POR}	V_{CC} Start Voltage to ensure internal Power-on Reset signal		-0.1		0.1	V
V_{CCRR}	V_{CC} Rise Rate to ensure internal Power_on Reset signal		0.3			V/ms
t_{RST}	Minimum pulse width on $\overline{\text{RESET}}$ Pin	5V, 25°C		400		ns
V_{HYST}	Brown-out Detector Hysteresis			50		mV
t_{BOD}	Min Pulse Width on Brown-out Reset					ns
V_{BG}	Bandgap reference voltage	V_{CC} = 2.7V - 5.5V	1.0	1.1	1.2	V
t_{BG}	Bandgap reference start-up time	-		40	70	µs
I_{BG}	Bandgap reference current consumption	-		10		µA

Note: The POR will not work unless the supply voltage has been below VPOT (falling)

Table 26-4. BODLEVEL Fuse Coding

BODLEVEL 2..0 Fuses	Min V_{BOT} (1)	Typ V_{BOT}	Max V_{BOT} (1)	Units
111	BOD Disabled			
110	2.5	2.7	2.9	V
101	RESERVED			
100		3.0		
011		3.5		
010	RESERVED			
001		4.0		
000	4.1	4.3	4.5	

Note: 1. The test is performed using BODLEVEL = 000 and 110.

26.6 External Interrupts Characteristics

Table 26-5. Asynchronous External Interrupt Characteristics

Symbol	Parameter	Condition	Min	Typ	Max	Units
t_{INT}	Minimum pulse width for asynchronous external interrupt			50		ns

26.7 SPI Timing Characteristics

See Figure 26-3 and Figure 26-7 for details.

Table 26-6. SPI Timing Parameters

	Description	Mode	Min	Typ	Max	
1	SCK period	Master		See Table 17-5		ns
2	SCK high/low	Master		50% duty cycle		
3	Rise/Fall time	Master		TBD		
4	Setup	Master		10		
5	Hold	Master		10		
6	Out to SCK	Master		$0.5 \cdot t_{sck}$		
7	SCK to out	Master		10		
8	SCK to out high	Master		10		
9	$\overline{SS}$ low to out	Slave		15		
10	SCK period	Slave	$4 \cdot t_{ck}$			
11	SCK high/low(1)	Slave	$2 \cdot t_{ck}$			
12	Rise/Fall time	Slave		TBD		
13	Setup	Slave	10			
14	Hold	Slave	t_{ck}			
15	SCK to out	Slave		15		
16	SCK to $\overline{SS}$ high	Slave	20			
17	$\overline{SS}$ high to tri-state	Slave		10		
18	$\overline{SS}$ low to SCK	Slave	20			

Note: 1. In SPI Programming mode the minimum SCK high/low period is:
- 2 t_{CLCL} for f_{CK} < 12 MHz
- 3 t_{CLCL} for f_{CK} > 12 MHz

Figure 26-3. SPI Interface Timing Requirements (Master Mode)

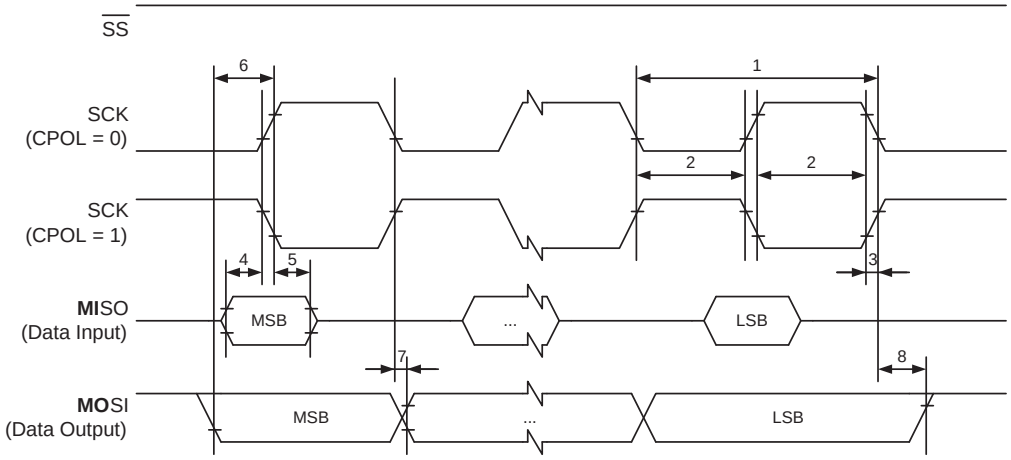
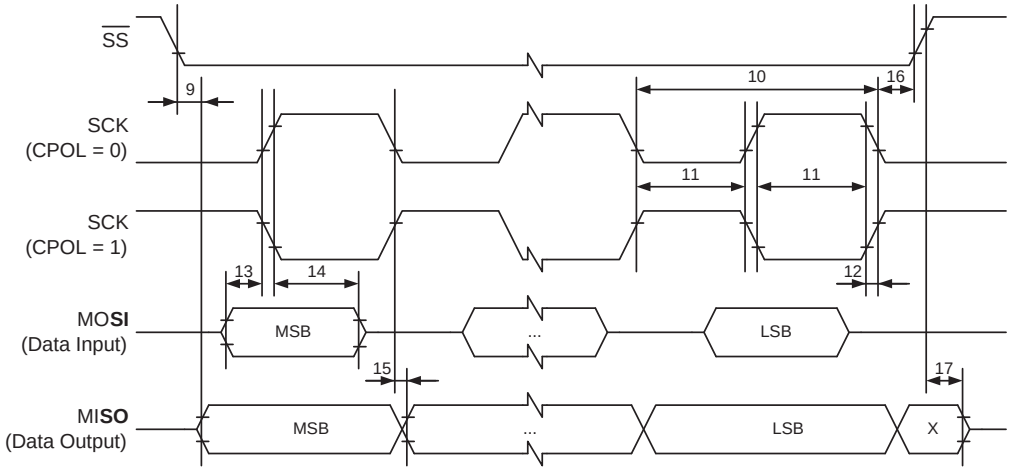


Table 26-7. SPI Interface Timing Requirements (Slave Mode)



26.8 Hardware Boot EntranceTiming Characteristics

Figure 26-4. Hardware Boot Timing Requirements

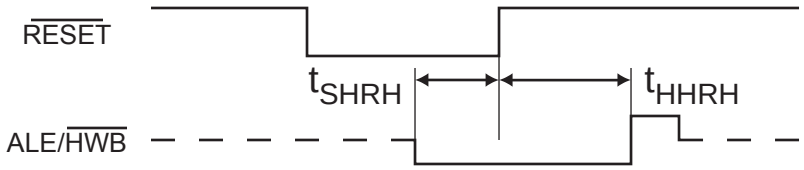


Table 26-8. Hardware Boot Timings

Symbol	Parameter	Min	Max
tSHRH	HWB low Setup before Reset High	0	
tHHRH	HWB low Hold after Reset High	StartUpTime(SUT) + Time Out Delay(TOUT)	

26.9 Parallel Programming Characteristics

Figure 26-5. Parallel Programming Timing, Including some General Timing Requirements

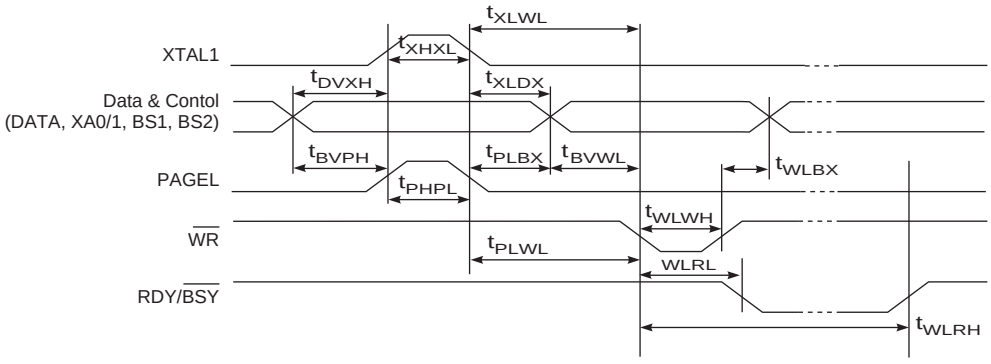
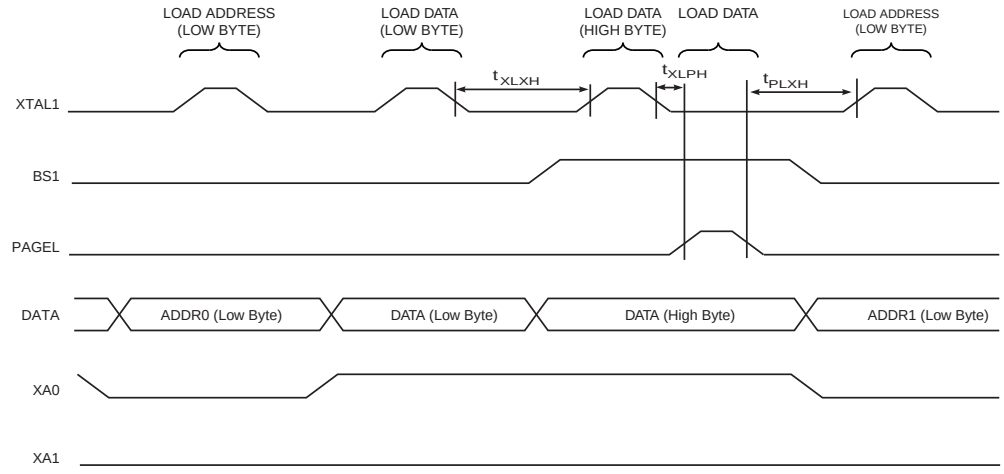
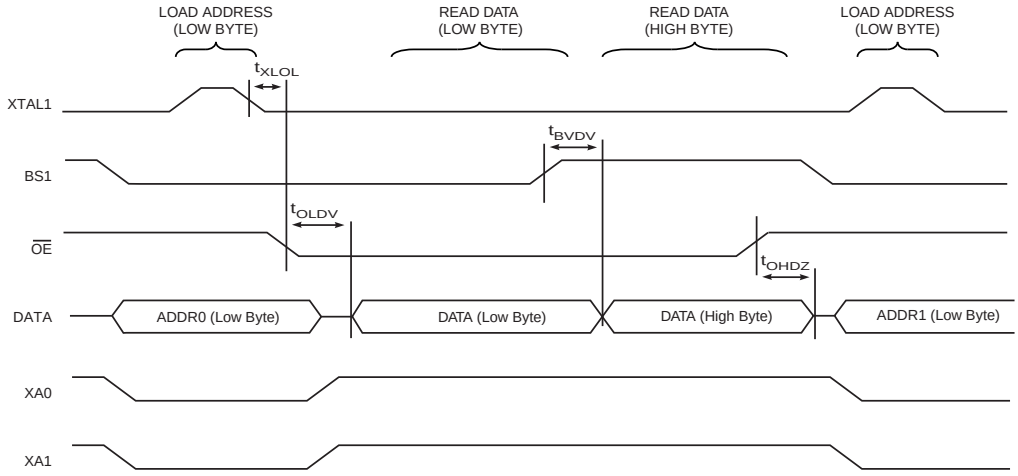


Figure 26-6. Parallel Programming Timing, Loading Sequence with Timing Requirements(1)



Note: 1. The timing requirements shown in Figure 26-5 (i.e., t_{DVXH}, t_{XHXL}, and t_{XLDX}) also apply to loading operation.

Figure 26-7. Parallel Programming Timing, Reading Sequence (within the Same Page) with Timing Requirements(1)



Note: 1. The timing requirements shown in Figure 26-5 (i.e., t_{DVXH}, t_{XHXL}, and t_{XLDX}) also apply to reading operation.

Table 26-9. Parallel Programming Characteristics, V_{CC} = 5V ± 10%

Symbol	Parameter	Min	Typ	Max	Units
V_{PP}	Programming Enable Voltage	11.5		12.5	V
I_{PP}	Programming Enable Current			250	µA
t_{DVXH}	Data and Control Valid before XTAL1 High	67			ns
t_{XLXH}	XTAL1 Low to XTAL1 High	200			ns
t_{XHXL}	XTAL1 Pulse Width High	150			ns
t_{XLDX}	Data and Control Hold after XTAL1 Low	67			ns

Table 26-9. Parallel Programming Characteristics, V_{CC} = 5V ± 10% (Continued)

Symbol	Parameter	Min	Typ	Max	Units
t_{XLWL}	XTAL1 Low to $\overline{WR}$ Low	0			ns
t_{XLPH}	XTAL1 Low to PAGEL high	0			ns
t_{PLXH}	PAGEL low to XTAL1 high	150			ns
t_{BVPH}	BS1 Valid before PAGEL High	67			ns
t_{PHPL}	PAGEL Pulse Width High	150			ns
t_{PLBX}	BS1 Hold after PAGEL Low	67			ns
t_{WLBX}	BS2/1 Hold after $\overline{WR}$ Low	67			ns
t_{PLWL}	PAGEL Low to $\overline{WR}$ Low	67			ns
t_{BVWL}	BS2/1 Valid to $\overline{WR}$ Low	67			ns
t_{WLWH}	$\overline{WR}$ Pulse Width Low	150			ns
t_{WLRL}	$\overline{WR}$ Low to RDY/$\overline{BSY}$ Low	0		1	µs
t_{WLRH}	$\overline{WR}$ Low to RDY/$\overline{BSY}$ High(1)	3.7		4.5	ms
t_{WLRH_CE}	$\overline{WR}$ Low to RDY/$\overline{BSY}$ High for Chip Erase(2)	7.5		9	ms
t_{XLOL}	XTAL1 Low to $\overline{OE}$ Low	0			ns
t_{BVDV}	BS1 Valid to DATA valid	0		250	ns
t_{OLDV}	$\overline{OE}$ Low to DATA Valid			250	ns
t_{OHDZ}	$\overline{OE}$ High to DATA Tri-stated			250	ns

Notes: 1. t_{WLRH} is valid for the Write Flash, Write EEPROM, Write Fuse bits and Write Lock bits commands.

2. t_{WLRH_CE} is valid for the Chip Erase command.

27. 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*V_{CC}*f where C_L = load capacitance, V_{CC} = operating voltage and f = average switching frequency of I/O pin.

The parts are characterized at frequencies higher than test limits. Parts are not guaranteed to function properly at frequencies higher than the ordering code indicates.

The difference between current consumption in Power-down mode with Watchdog Timer enabled and Power-down mode with Watchdog Timer disabled represents the differential current drawn by the Watchdog Timer.

27.1 Active Supply Current

Figure 27-1. Active Supply Current vs. Frequency (Regulator Enabled T = 85°C)

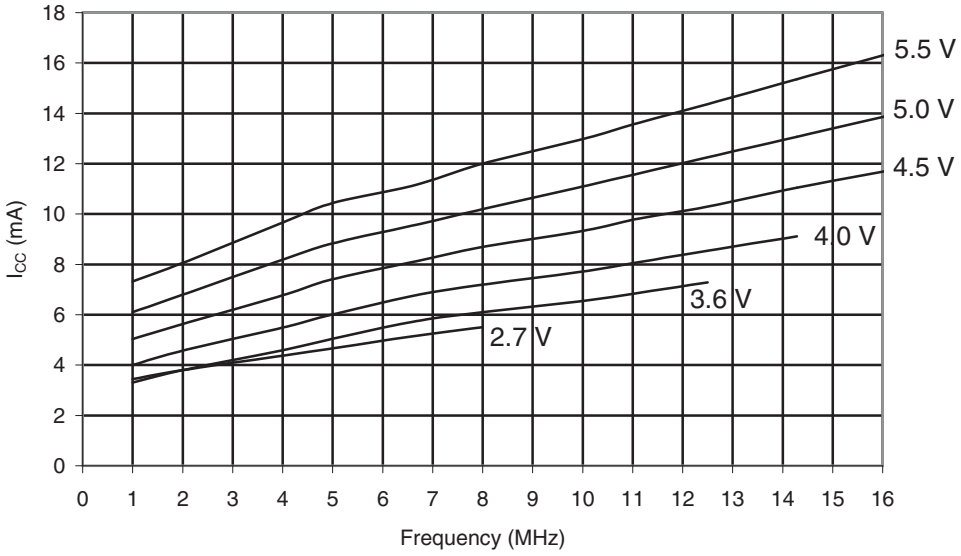
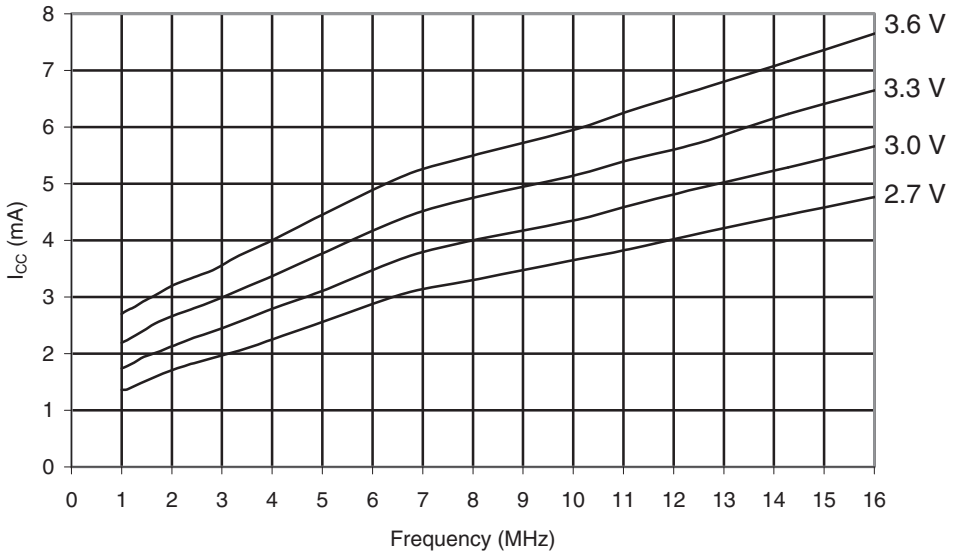


Figure 27-2. Active Supply Current vs. Frequency (Regulator Disabled T = 85°C)



27.2 Idle Supply Current

Figure 27-3. Idle Supply Current vs. Frequency (Regulator Enabled T = 85°C)

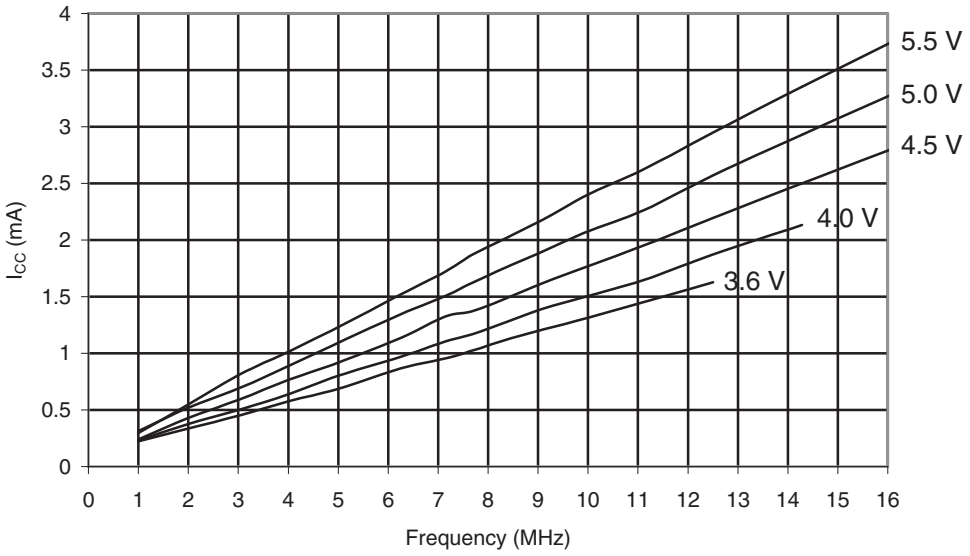
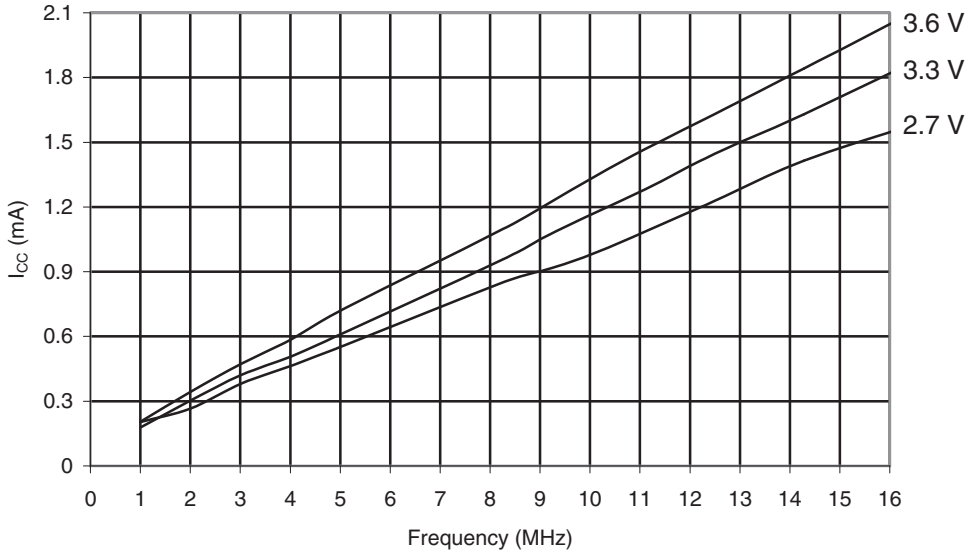


Figure 27-4. Idle Supply Current vs. Frequency (Regulator Disabled T = 85°C



27.3 Power-down Supply Current

Figure 27-5. Power-Down Supply Current vs. V_{CC} (Watchdog Timer Disabled)

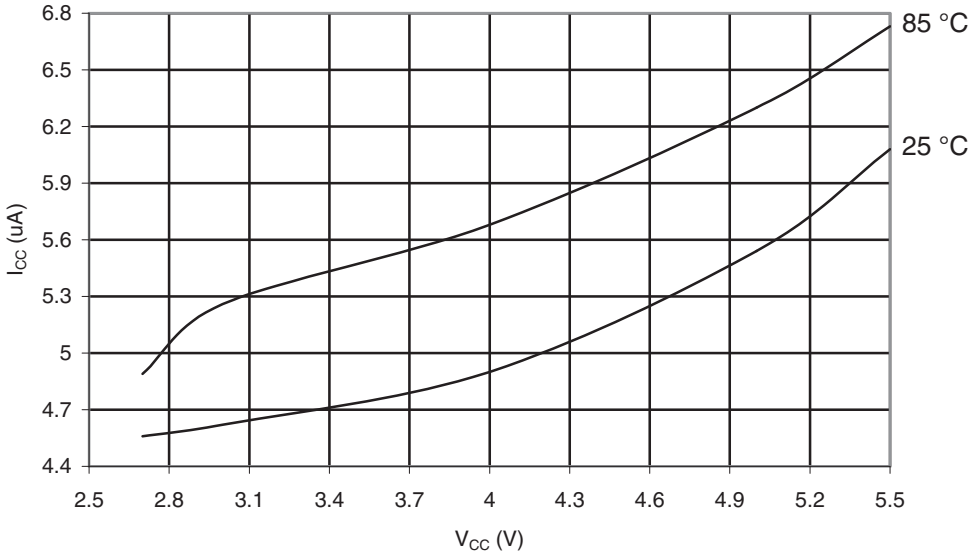


Figure 27-6. Power-Down Supply Current vs. V_{CC} (Watchdog Timer Enabled)

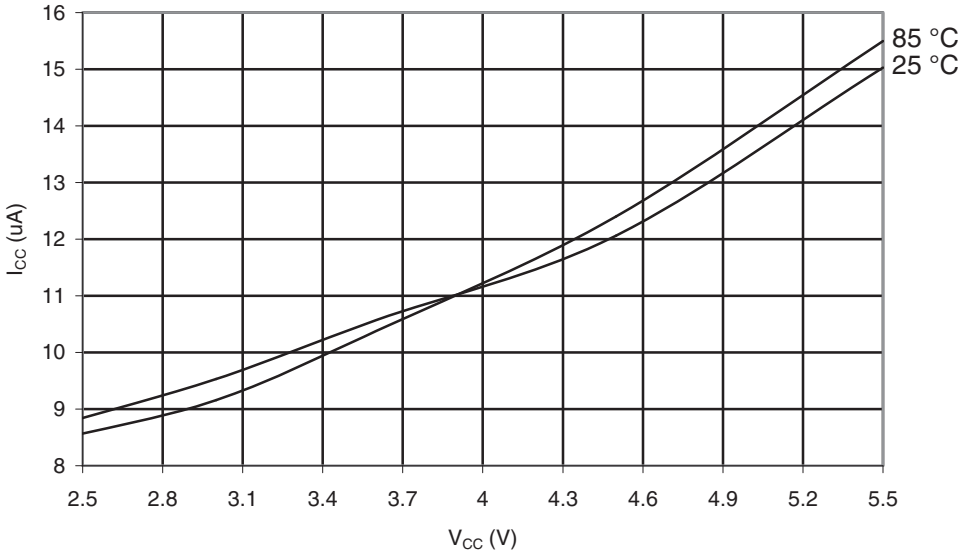
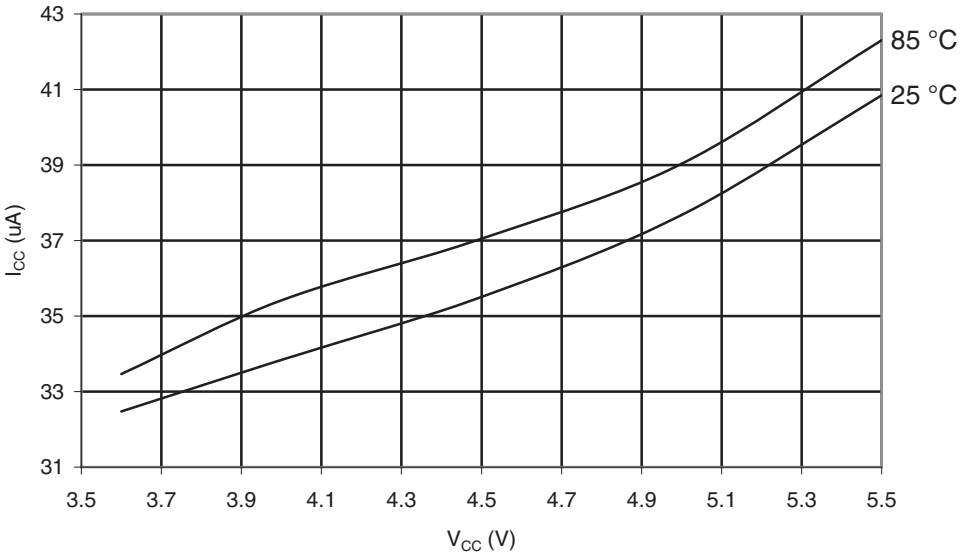


Figure 27-7. Power-Down Supply Current vs. V_{CC} (WDT Enabled BODEN)



27.4 Pin Pull-Up

Figure 27-8. I/O Pin Pull-up Resistor Current vs. Input Voltage (V_{CC} = 5 V)

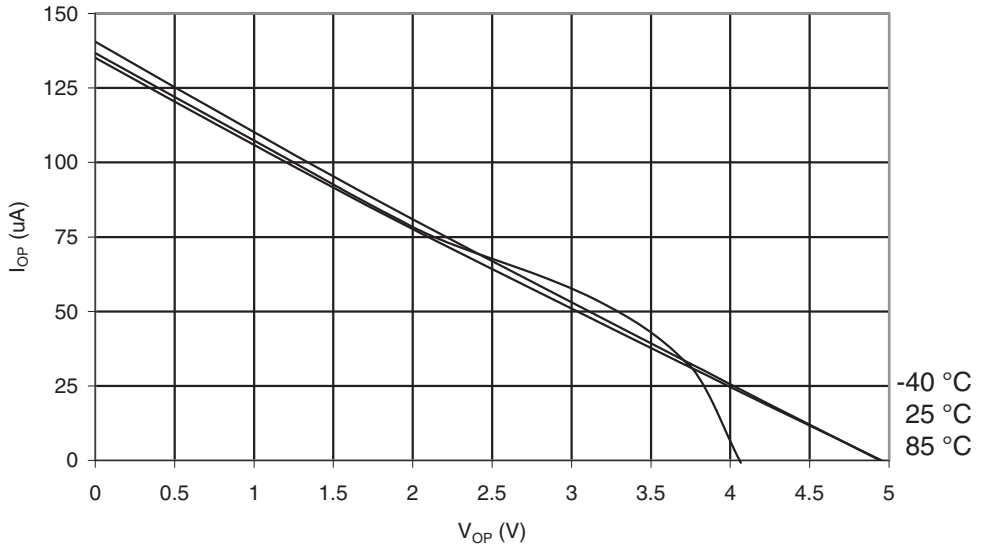
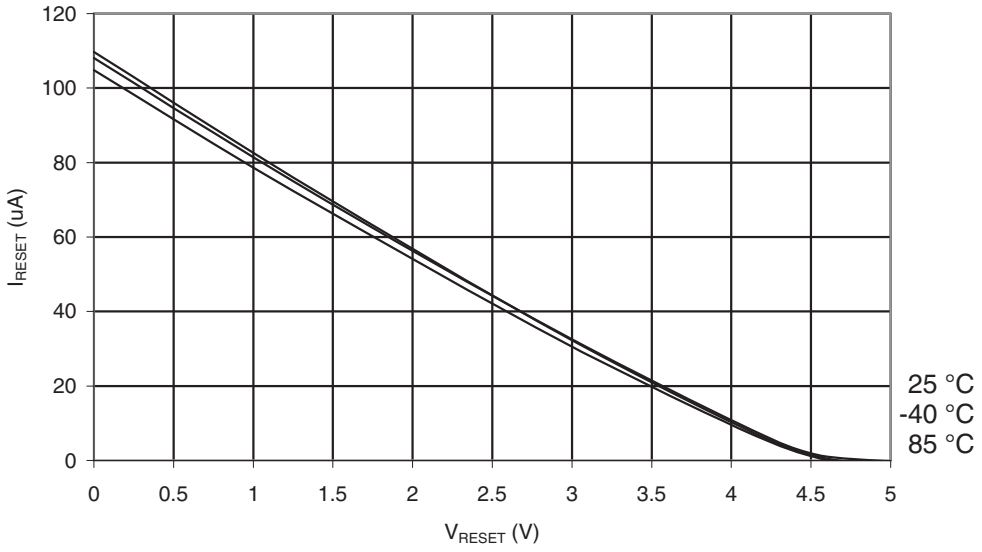


Figure 27-9. Reset Pull-up Resistor Current vs. Reset Pin Voltage (V_{CC} = 5 V)



27.5 Pin Driver Strength

Figure 27-10. I/O Pin Output Voltage vs. Sink Current(V_{CC} = 3 V)

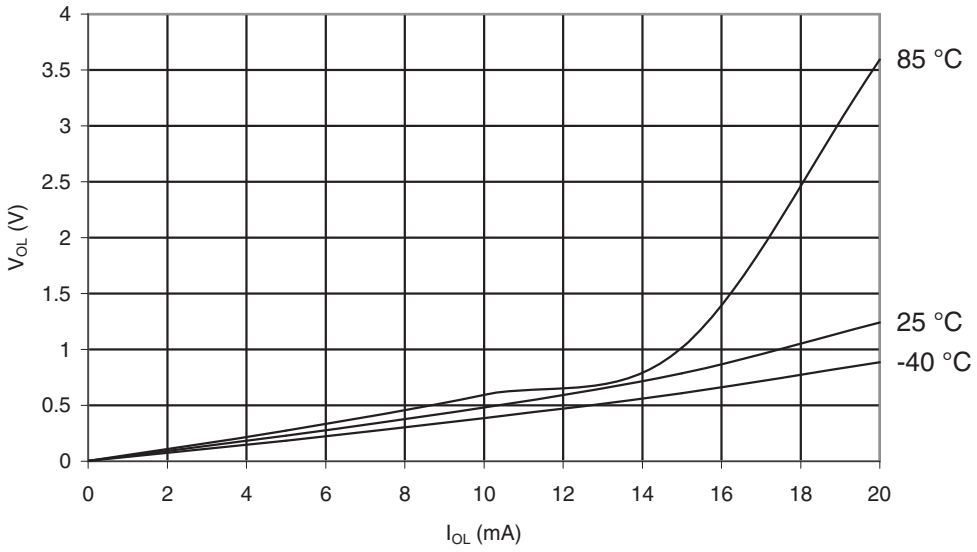


Figure 27-11. I/O Pin Output Voltage vs. Sink Current(V_{CC} = 5 V)

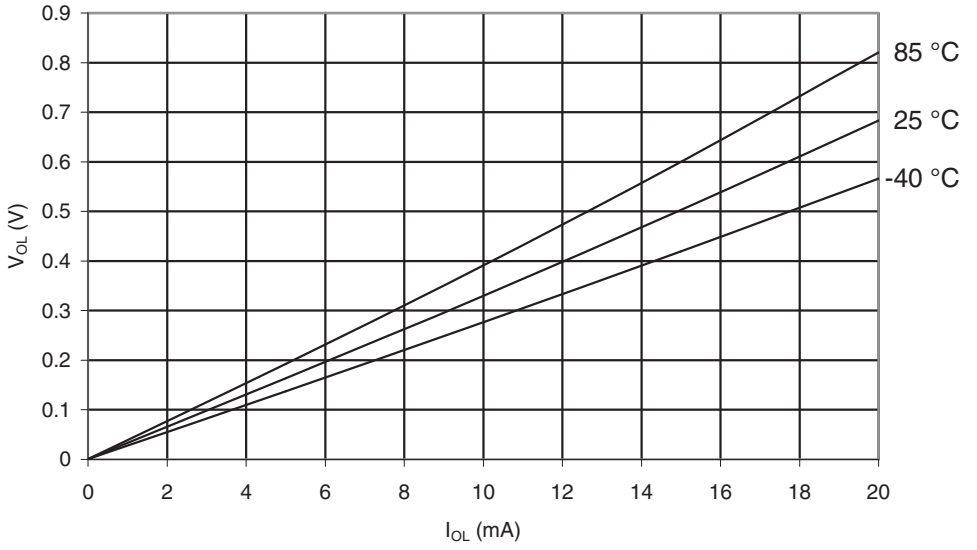


Figure 27-12. I/O Pin Output Voltage vs. Source Current(Vcc = 3 V)

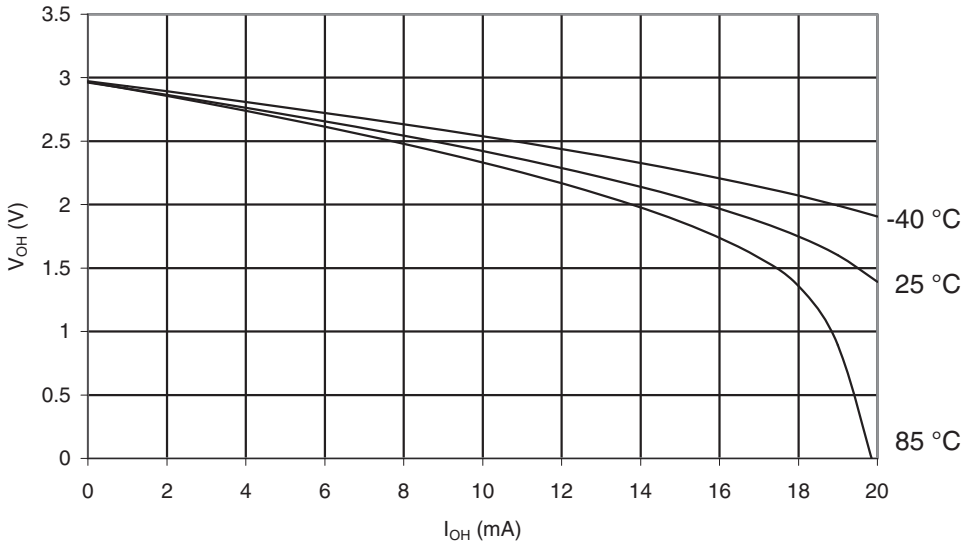


Figure 27-13. I/O Pin Output Voltage vs. Source Current(V_{CC} = 5 V)

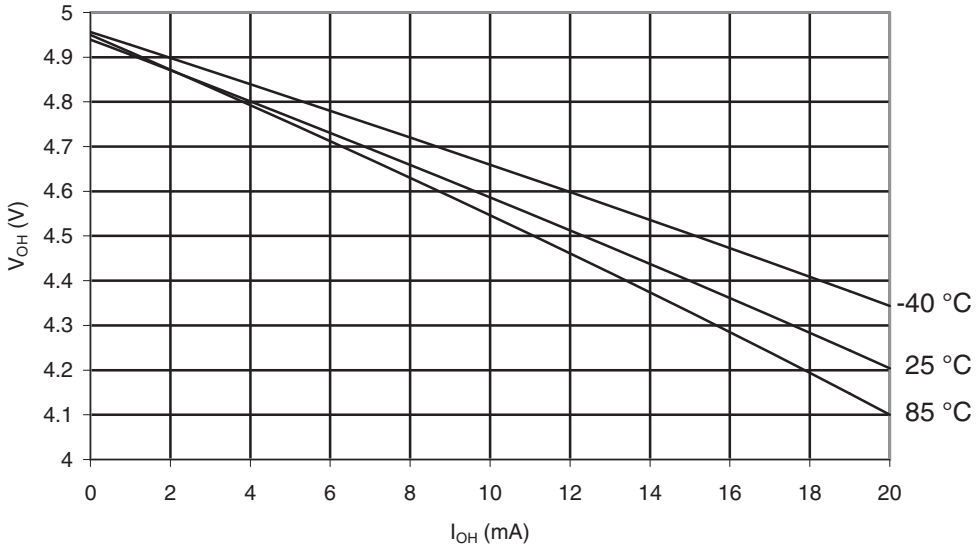
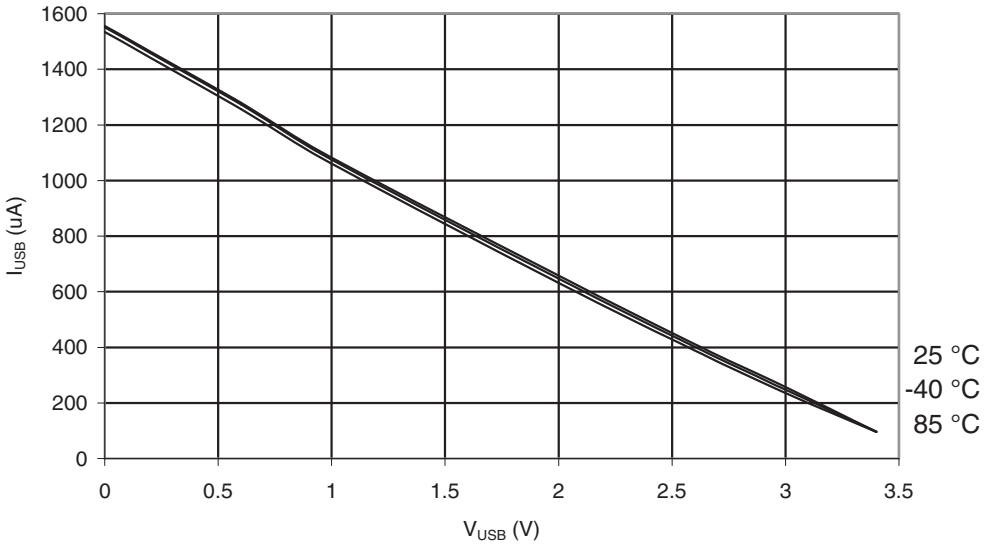


Figure 27-14. USB DP HI Pull-Up Resistor Current vs. USB Pin Voltage



27.6 Pin Threshold and Hysteresis

Figure 27-15. I/O Pin Input Threshold Voltage vs. V_{CC} (V_{IH}, I/O Pin read as '1')

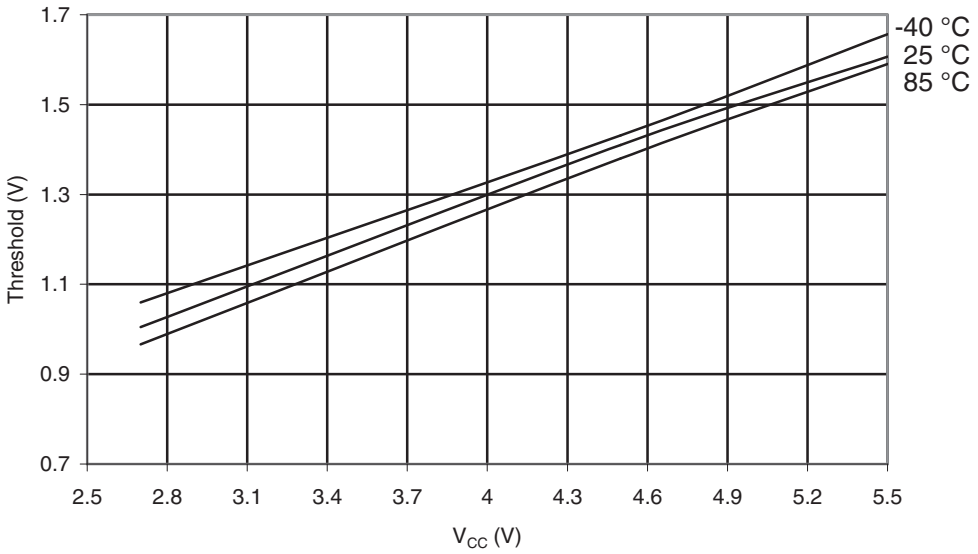
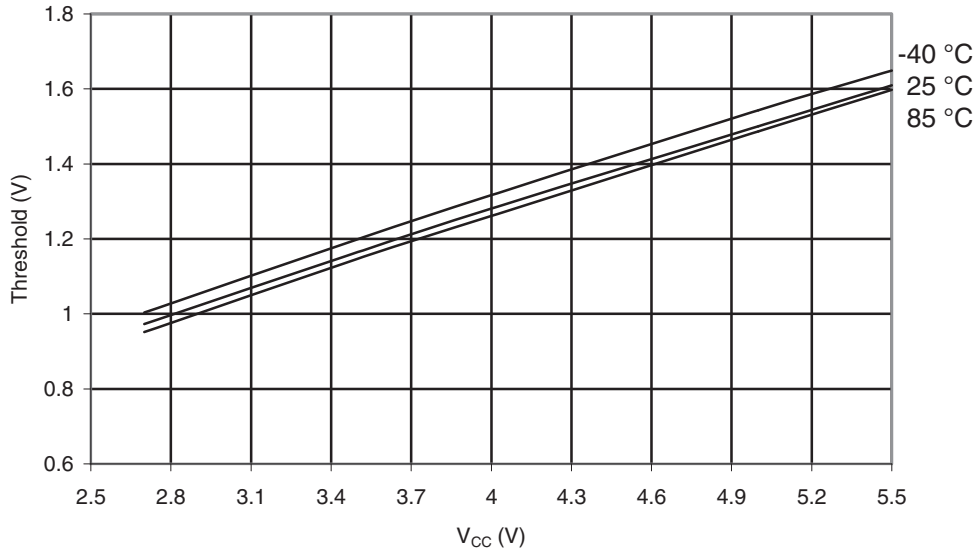


Figure 27-16. I/O Pin Input Threshold Voltage vs. V_{CC} (V_{IL}, I/O Pin read as '0')



27.7 BOD Threshold

Figure 27-17. BOD Thresholds vs. Temperature (BODLEVEL is 2.7 V)

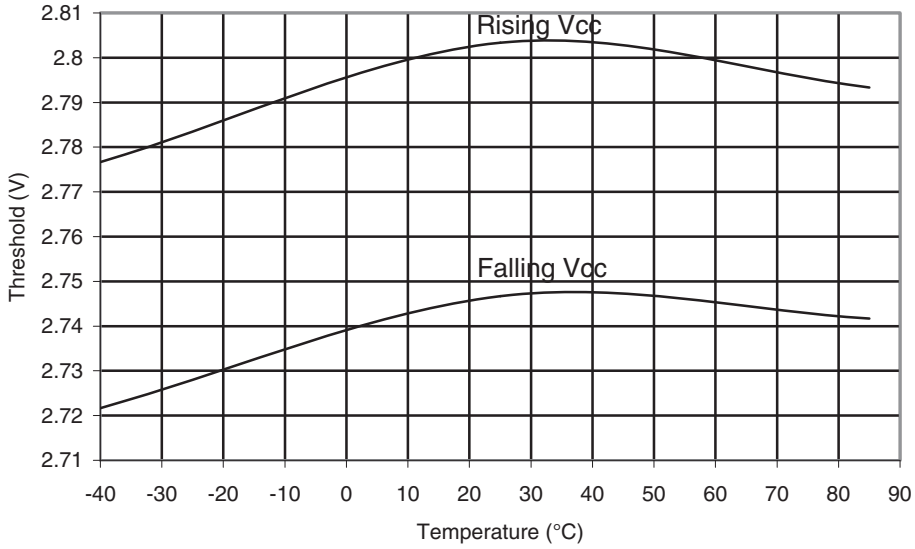


Figure 27-18. BOD Thresholds vs. Temperature (BODLEVEL is 3.5 V)

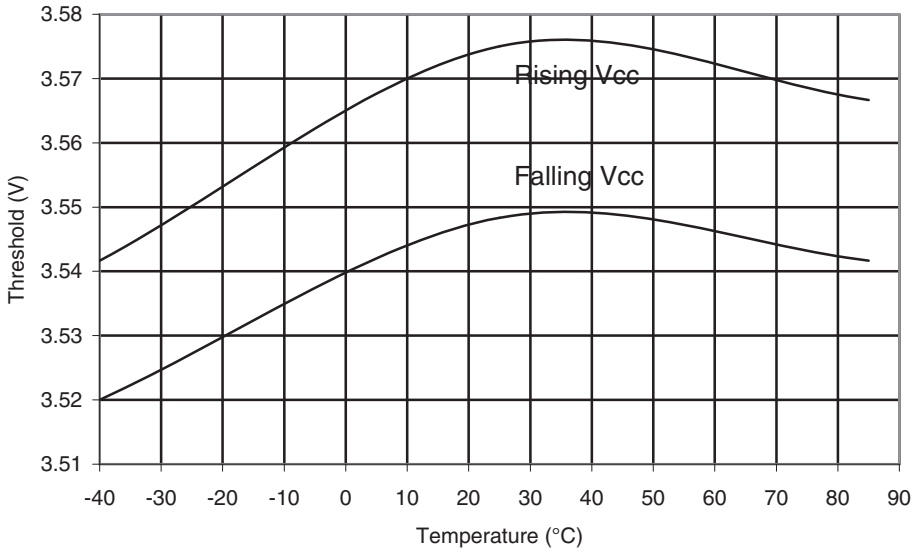


Figure 27-19. BOD Thresholds vs. Temperature (BODLEVEL is 4.3 V)

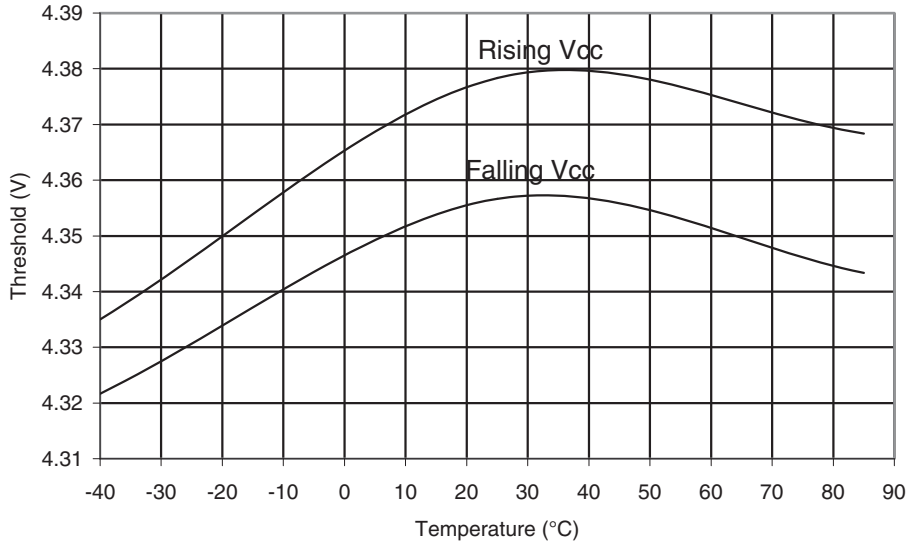
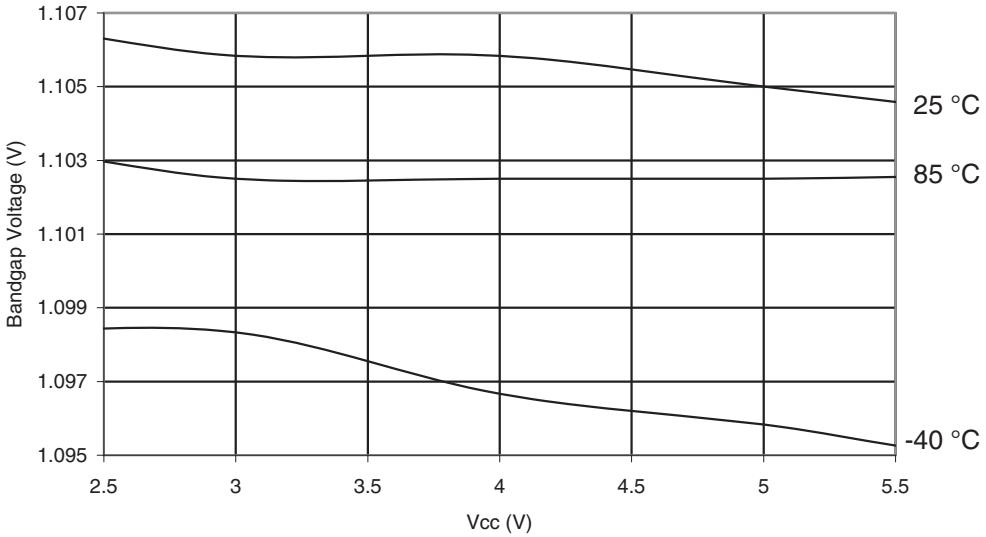


Figure 27-20. Bandgap Voltage vs. Vcc



27.8 Internal Oscilllator Speed

Figure 27-21. Watchdog Oscillator Frequency vs. Temperature

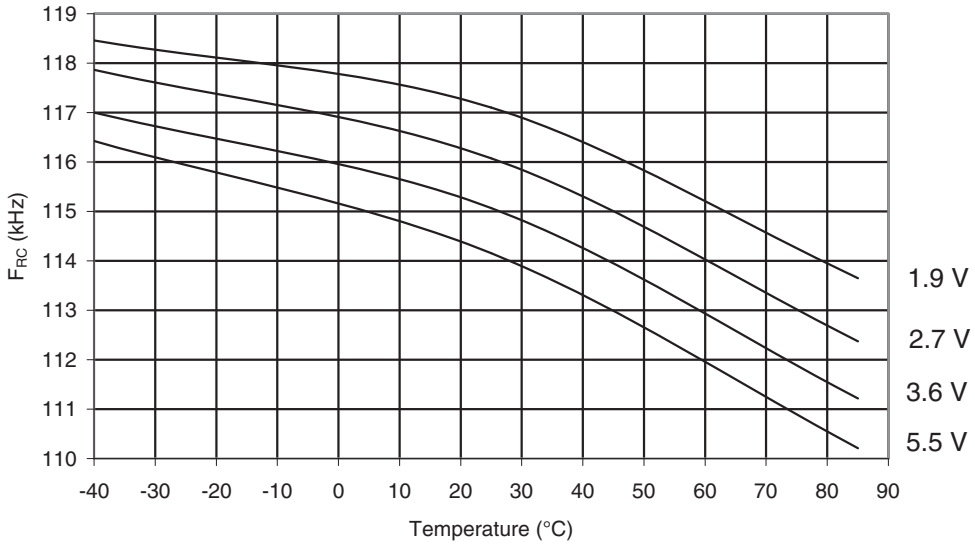


Figure 27-22. Watchdog Oscillator Frequency vs. V_{CC}

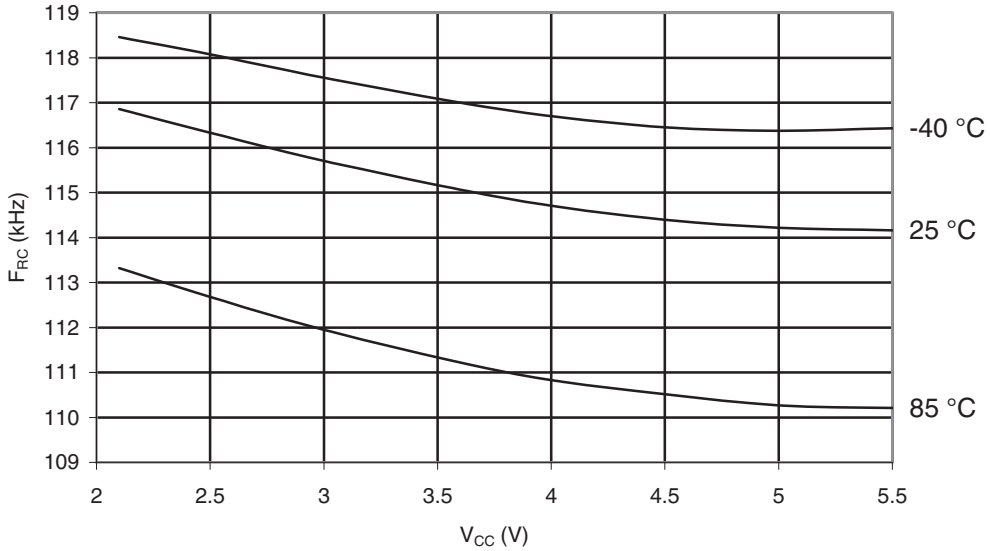


Figure 27-23. Calibrated 8 MHz RC Oscillator Frequency vs. V_{CC}

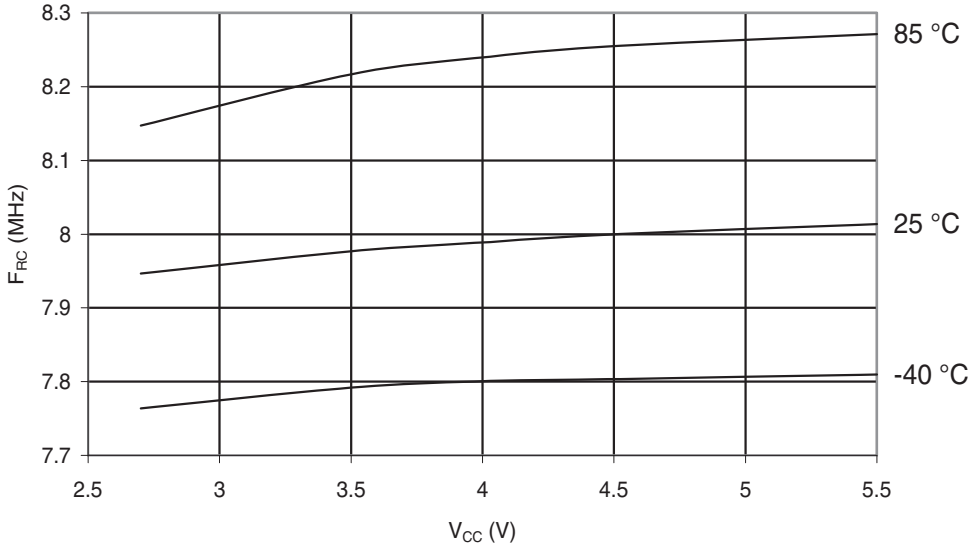


Figure 27-24. Calibrated 8 MHz RC Oscillator Frequency vs. Temperature

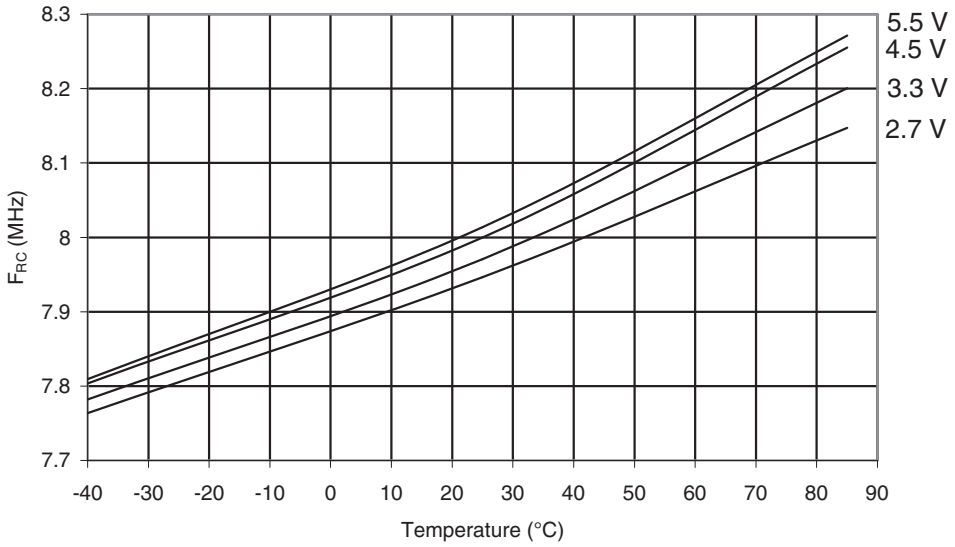
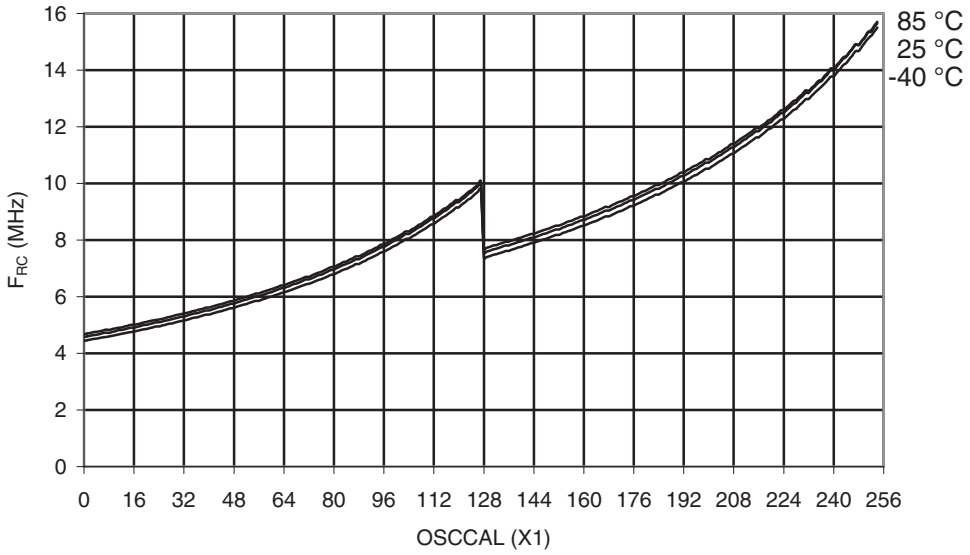


Figure 27-25. Calibrated 8 MHz RC Oscillator Frequency vs. OSCCAL Value



27.9 Current Consumption of Peripheral Units

Figure 27-26. USB Regulator Level vs. V_{CC}

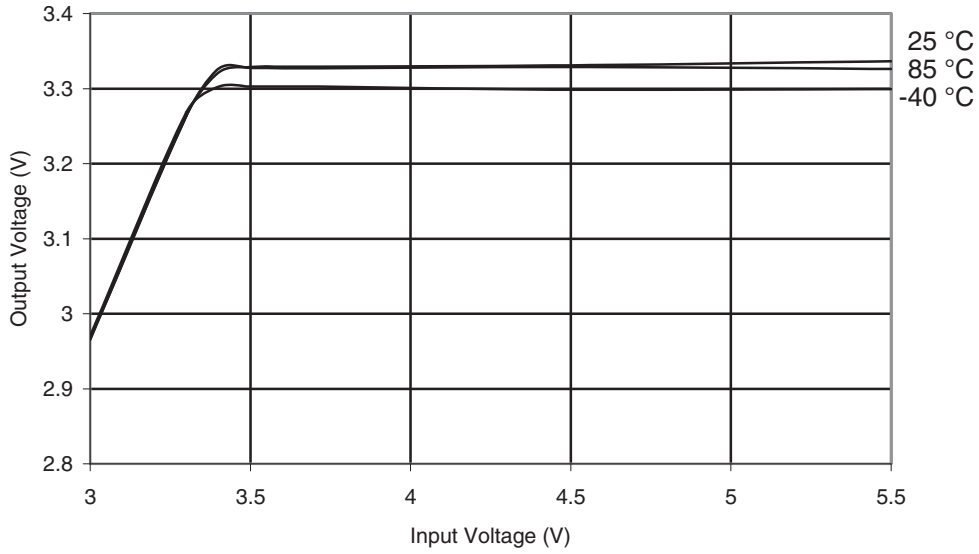
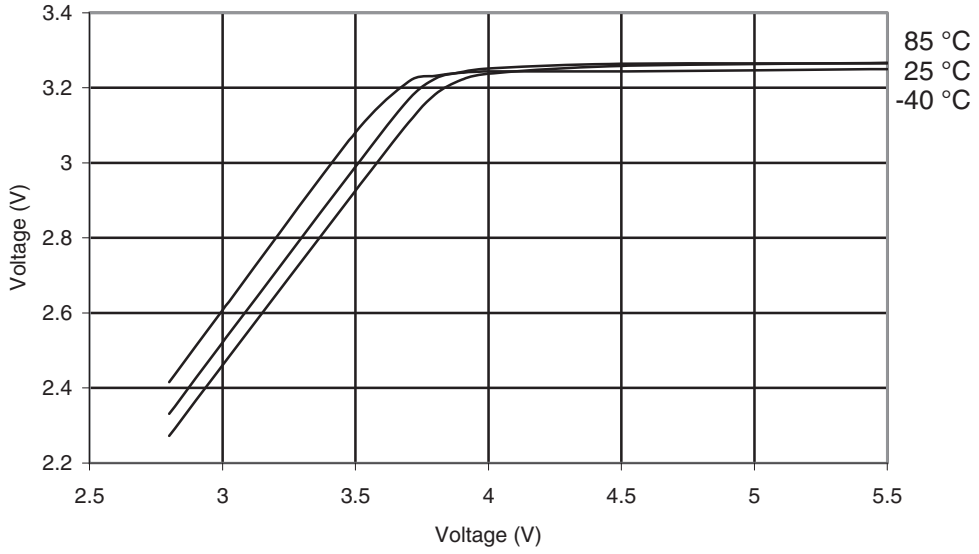
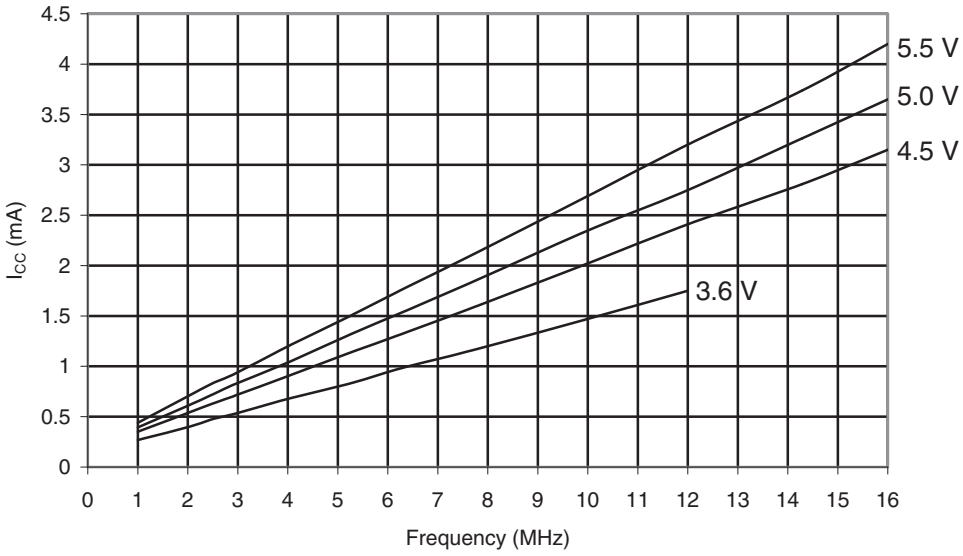


Figure 27-27. USB Regulator Level with load 75 Ω vs. V_{CC}



27.10 Current Consumption in Reset and Reset Pulsewidth

Figure 27-28. Reset Supply Current vs. Frequency (Excluding Current Through the Reset Pullup)



28. Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xFF)	Reserved	-	-	-	-	-	-	-	-	
(0xFE)	Reserved	-	-	-	-	-	-	-	-	
(0xFD)	Reserved	-	-	-	-	-	-	-	-	
(0xFC)	Reserved	-	-	-	-	-	-	-	-	
(0xFB)	UPOE	UPWE1	UPWE0	UPDRV1	UPDRV0	SCKI	DATAI	DPI	DMI	page 195
(0xFA)	Reserved	-	-	-	-	-	-	-	-	
(0xF9)	Reserved	-	-	-	-	-	-	-	-	
(0xF8)	Reserved	-	-	-	-	-	-	-	-	
(0xF7)	Reserved	-	-	-	-	-	-	-	-	
(0xF6)	Reserved	-	-	-	-	-	-	-	-	
(0xF5)	Reserved	-	-	-	-	-	-	-	-	
(0xF4)	UEINT		-	-	EPINT4:0					page 222
(0xF3)	Reserved	-	-	-	-	-	-	-	-	
(0xF2)	UEBCLX	BYCT7:0								page 221
(0xF1)	UEDATX	DAT7:0								page 221
(0xF0)	UEIENX	FLERRE	NAKINE	-	NAKOUTE	RXSTPE	RXOUTE	STALLEDE	TXINE	page 220
(0xEF)	UESTA1X	-	-	-	-	-	CTRLDIR	CURRBK1:0		page 218
(0xEE)	UESTA0X	CFGOK	OVERFI	UNDERFI	-	DTSEQ1:0		NBUSYBK1:0		page 217
(0xED)	UECFG1X	-	EPSIZE2:0			EPBK1:0		ALLOC	-	page 216
(0xEC)	UECFG0X	EPTYPE1:0		-	-	-	-	-	EPDIR	page 215
(0xEB)	UECONX	-	-	STALLRQ	STALLRQC	RSTDT	-	-	EPEN	page 214
(0xEA)	UERST	-	-	-	EPRST4:0					page 214
(0xE9)	UENUM	-	-	-	-	-	EPNUM2:0			page 214
(0xE8)	UEINTX	FIFOCON	NAKINI	RWAL	NAKOUTI	RXSTPI	RXOUTI	STALLEDI	TXINI	page 219
(0xE7)	Reserved	-	-	-	-	-	-	-	-	
(0xE6)	UDMFN	-	-	-	FNCERR	-	-	-	-	page 213
(0xE5)	UDFNUMH	-	-	-	-	-	FNUM10:8			page 213
(0xE4)	UDFNUML	FNUM7:0								page 213
(0xE3)	UDADDR	ADDEN	UADD6:0							page 212
(0xE2)	UDIEN	-	UPRSME	EORSME	WAKEUPE	EORSTE	SOFE	-	SUSPE	page 211
(0xE1)	UDINT	-	UPRSMI	EORSMI	WAKEUPI	EORSTI	SOFI	-	SUSPI	page 210
(0xE0)	UDCON	-	-	-	RPUTX	-	RSTCPU	RMWKUP	DETACH	page 209
(0xDF)	Reserved	-	-	-	-	-	-	-	-	
(0xDE)	Reserved	-	-	-	-	-	-	-	-	
(0xDD)	Reserved	-	-	-	-	-	-	-	-	
(0xDC)	Reserved	-	-	-	-	-	-	-	-	
(0xDB)	Reserved	-	-	-	-	-	-	-	-	
(0xDA)	Reserved	-	-	-	-	-	-	-	-	
(0xD9)	Reserved	-	-	-	-	-	-	-	-	
(0xD8)	USBCON	USBE	-	FRZCLK	-	-	-	-	-	page 195
(0xD7)	Reserved	-	-	-	-	-	-	-	-	
(0xD6)	Reserved	-	-	-	-	-	-	-	-	
(0xD5)	Reserved	-	-	-	-	-	-	-	-	
(0xD4)	Reserved	-	-	-	-	-	-	-	-	
(0xD3)	Reserved	-	-	-	-	-	-	-	-	
(0xD2)	CLKSTA	-	-	-	-	-	-	RCON	EXTON	page 38
(0xD1)	CLKSEL1	RCCKSEL3	RCCKSEL2	RCCKSEL1	RCCKSEL0	EXCKSEL3	EXCKSEL2	EXCKSEL1	EXCKSEL0	page 38
(0xD0)	CLKSEL0	RCSUT1	RCSUT0	EXSUT1	EXSUT0	RCE	EXTE	-	CLKS	page 37
(0xCF)	Reserved	-	-	-	-	-	-	-	-	
(0xCE)	UDR1	USART1 I/O Data Register								page 167
(0xCD)	UBRR1H	-	-	-	-	USART1 Baud Rate Register High Byte				page 171
(0xCC)	UBRR1L	USART1 Baud Rate Register Low Byte								page 171
(0xCB)	UCSR1D	-	-	-	-	-	-	CTSEN	RTSEN	page 171
(0xCA)	UCSR1C	UMSEL11	UMSEL10	UPM11	UPM10	USBS1	UCSZ11	UCSZ10	UCPOL1	page 169
(0xC9)	UCSR1B	RXCIE1	TXCIE1	UDRIE1	RXEN1	TXEN1	UCSZ12	RXB81	TXB81	page 168
(0xC8)	UCSR1A	RXC1	TXC1	UDRE1	FE1	DOR1	PE1	U2X1	MPCM1	page 167
(0xC7)	Reserved	-	-	-	-	-	-	-	-	
(0xC6)	Reserved	-	-	-	-	-	-	-	-	
(0xC5)	Reserved	-	-	-	-	-	-	-	-	
(0xC4)	Reserved	-	-	-	-	-	-	-	-	
(0xC3)	Reserved	-	-	-	-	-	-	-	-	
(0xC2)	Reserved	-	-	-	-	-	-	-	-	
(0xC1)	Reserved	-	-	-	-	-	-	-	-	
(0xC0)	Reserved	-	-	-	-	-	-	-	-	
(0xBF)	Reserved	-	-	-	-	-	-	-	-	

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xBE)	Reserved	-	-	-	-	-	-	-	-	
(0xBD)	Reserved	-	-	-	-	-	-	-	-	
(0xBC)	Reserved	-	-	-	-	-	-	-	-	
(0xBB)	Reserved	-	-	-	-	-	-	-	-	
(0xBA)	Reserved	-	-	-	-	-	-	-	-	
(0xB9)	Reserved	-	-	-	-	-	-	-	-	
(0xB8)	Reserved	-	-	-	-	-	-	-	-	
(0xB7)	Reserved	-	-	-	-	-	-	-	-	
(0xB6)	Reserved	-	-	-	-	-	-	-	-	
(0xB5)	Reserved	-	-	-	-	-	-	-	-	
(0xB4)	Reserved	-	-	-	-	-	-	-	-	
(0xB3)	Reserved	-	-	-	-	-	-	-	-	
(0xB2)	Reserved	-	-	-	-	-	-	-	-	
(0xB1)	Reserved	-	-	-	-	-	-	-	-	
(0xB0)	Reserved	-	-	-	-	-	-	-	-	
(0xAF)	Reserved	-	-	-	-	-	-	-	-	
(0xAE)	Reserved	-	-	-	-	-	-	-	-	
(0xAD)	Reserved	-	-	-	-	-	-	-	-	
(0xAC)	Reserved	-	-	-	-	-	-	-	-	
(0xAB)	Reserved	-	-	-	-	-	-	-	-	
(0xAA)	Reserved	-	-	-	-	-	-	-	-	
(0xA9)	Reserved	-	-	-	-	-	-	-	-	
(0xA8)	Reserved	-	-	-	-	-	-	-	-	
(0xA7)	Reserved	-	-	-	-	-	-	-	-	
(0xA6)	Reserved	-	-	-	-	-	-	-	-	
(0xA5)	Reserved	-	-	-	-	-	-	-	-	
(0xA4)	Reserved	-	-	-	-	-	-	-	-	
(0xA3)	Reserved	-	-	-	-	-	-	-	-	
(0xA2)	Reserved	-	-	-	-	-	-	-	-	
(0xA1)	Reserved	-	-	-	-	-	-	-	-	
(0xA0)	Reserved	-	-	-	-	-	-	-	-	
(0x9F)	Reserved	-	-	-	-	-	-	-	-	
(0x9E)	Reserved	-	-	-	-	-	-	-	-	
(0x9D)	Reserved	-	-	-	-	-	-	-	-	
(0x9C)	Reserved	-	-	-	-	-	-	-	-	
(0x9B)	Reserved	-	-	-	-	-	-	-	-	
(0x9A)	Reserved	-	-	-	-	-	-	-	-	
(0x99)	Reserved	-	-	-	-	-	-	-	-	
(0x98)	Reserved	-	-	-	-	-	-	-	-	
(0x97)	Reserved	-	-	-	-	-	-	-	-	
(0x96)	Reserved	-	-	-	-	-	-	-	-	
(0x95)	Reserved	-	-	-	-	-	-	-	-	
(0x94)	Reserved	-	-	-	-	-	-	-	-	
(0x93)	Reserved	-	-	-	-	-	-	-	-	
(0x92)	Reserved	-	-	-	-	-	-	-	-	
(0x91)	Reserved	-	-	-	-	-	-	-	-	
(0x90)	Reserved	-	-	-	-	-	-	-	-	
(0x8F)	Reserved	-	-	-	-	-	-	-	-	
(0x8E)	Reserved	-	-	-	-	-	-	-	-	
(0x8D)	OCR1CH	Timer/Counter1 - Output Compare Register C High Byte								page 135
(0x8C)	OCR1CL	Timer/Counter1 - Output Compare Register C Low Byte								page 135
(0x8B)	OCR1BH	Timer/Counter1 - Output Compare Register B High Byte								page 135
(0x8A)	OCR1BL	Timer/Counter1 - Output Compare Register B Low Byte								page 135
(0x89)	OCR1AH	Timer/Counter1 - Output Compare Register A High Byte								page 135
(0x88)	OCR1AL	Timer/Counter1 - Output Compare Register A Low Byte								page 135
(0x87)	ICR1H	Timer/Counter1 - Input Capture Register High Byte								page 135
(0x86)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte								page 135
(0x85)	TCNT1H	Timer/Counter1 - Counter Register High Byte								page 134
(0x84)	TCNT1L	Timer/Counter1 - Counter Register Low Byte								page 134
(0x83)	Reserved	-	-	-	-	-	-	-	-	
(0x82)	TCCR1C	FOC1A	FOC1B	FOC1C	-	-	-	-	-	page 134
(0x81)	TCCR1B	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	page 133
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	page 129
(0x7F)	DIDR1	-	AIN6D	AIN5D	AIN4D	AIN3D	AIN2D	AIN1D	AIN0D	page 225
(0x7E)	Reserved	-	-	-	-	-	-	-	-	
(0x7D)	ACMUX	-	-	-	-	-	CMUX2	CMUX1	CMUX0	page 225

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0x7C)	Reserved	-	-	-	-	-	-	-	-	
(0x7B)	Reserved	-	-	-	-	-	-	-	-	
(0x7A)	Reserved	-	-	-	-	-	-	-	-	
(0x79)	Reserved	-	-	-	-	-	-	-	-	
(0x78)	Reserved	-	-	-	-	-	-	-	-	
(0x77)	Reserved	-	-	-	-	-	-	-	-	
(0x76)	Reserved	-	-	-	-	-	-	-	-	
(0x75)	Reserved	-	-	-	-	-	-	-	-	
(0x74)	Reserved	-	-	-	-	-	-	-	-	
(0x73)	Reserved	-	-	-	-	-	-	-	-	
(0x72)	Reserved	-	-	-	-	-	-	-	-	
(0x71)	Reserved	-	-	-	-	-	-	-	-	
(0x70)	Reserved	-	-	-	-	-	-	-	-	
(0x6F)	TIMSK1	-	-	ICIE1	-	OCIE1C	OCIE1B	OCIE1A	TOIE1	page 135
(0x6E)	TIMSK0	-	-	-	-	-	OCIE0B	OCIE0A	TOIE0	page 106
(0x6D)	Reserved	-	-	-	-	-	-	-	-	
(0x6C)	PCMSK1	-	-	-	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	page 87
(0x6B)	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	page 87
(0x6A)	EICRB	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	page 85
(0x69)	EICRA	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	page 84
(0x68)	PCICR	-	-	-	-	-	-	PCIE1	PCIE0	page 86
(0x67)	Reserved	-	-	-	-	-	-	-	-	
(0x66)	OSCCAL	Oscillator Calibration Register								page 38
(0x65)	PRR1	PRUSB	-	-	-	-	-	-	PRUSART1	page 46
(0x64)	PRR0	-	-	PRTIM0	-	PRTIM1	PRSPI	-	-	page 46
(0x63)	REGCR	-	-	-	-	-	-	-	REGDIS	page 196
(0x62)	WDTCKD	-	-	WDEWIFCM	WCLKD2	WDEWIF	WDEWIE	WCLKD1	WCLKD0	page 57
(0x61)	CLKPR	CLKPCE	-	-	-	CLKPS3	CLKPS2	CLKPS1	CLKPS0	page 39
(0x60)	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	page 56
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	page 9
0x3E (0x5E)	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	page 12
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	page 12
0x3C (0x5C)	Reserved	-	-	-	-	-	-	-	-	
0x3B (0x5B)	Reserved	-	-	-	-	-	-	-	-	
0x3A (0x5A)	Reserved	-	-	-	-	-	-	-	-	
0x39 (0x59)	Reserved	-	-	-	-	-	-	-	-	
0x38 (0x58)	Reserved	-	-	-	-	-	-	-	-	
0x37 (0x57)	SPMCSR	SPMIE	RWWSB	SIGRD	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	page 242
0x36 (0x56)	Reserved	-	-	-	-	-	-	-	-	
0x35 (0x55)	MCUCR	-	-	-	-	-	-	IVSEL	IVCE	page 65, 82
0x34 (0x54)	MCUSR	-	-	USBRF	-	WDRF	BORF	EXTRF	PORF	page 55
0x33 (0x53)	SMCR	-	-	-	-	SM2	SM1	SM0	SE	page 45
0x32 (0x52)	Reserved	-	-	-	-	-	-	-	-	
0x31 (0x51)	DWDR	debugWIRE Data Register								page 245
0x30 (0x50)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	page 224
0x2F (0x4F)	Reserved	-	-	-	-	-	-	-	-	
0x2E (0x4E)	SPDR	SPI Data Register								page 147
0x2D (0x4D)	SPSR	SPIF	WCOL	-	-	-	-	-	SPI2X	page 146
0x2C (0x4C)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	page 145
0x2B (0x4B)	GPIOR2	General Purpose I/O Register 2								page 24
0x2A (0x4A)	GPIOR1	General Purpose I/O Register 1								page 24
0x29 (0x49)	PLLCSR	-	-	-	PLLP2	PLLP1	PLLP0	PLLE	PLOCK	page 40
0x28 (0x48)	OCR0B	Timer/Counter0 Output Compare Register B								page 106
0x27 (0x47)	OCR0A	Timer/Counter0 Output Compare Register A								page 106
0x26 (0x46)	TCNT0	Timer/Counter0 (8 Bit)								page 106
0x25 (0x45)	TCCR0B	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00	page 105
0x24 (0x44)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00	page 105
0x23 (0x43)	GTCCR	TSM	-	-	-	-	-	PSRASY	PSRSYNC	page 89
0x22 (0x42)	EEARH	-	-	-	-	EEPROM Address Register High Byte				page 20
0x21 (0x41)	EEARL	EEPROM Address Register Low Byte								page 20
0x20 (0x40)	EEDR	EEPROM Data Register								page 20
0x1F (0x3F)	EECR	-	-	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	page 21
0x1E (0x3E)	GPIOR0	General Purpose I/O Register 0								page 25
0x1D (0x3D)	EIMSK	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	page 86
0x1C (0x3C)	EIFR	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	INTF0	page 86
0x1B (0x3B)	PCIFR	-	-	-	-	-	-	PCIF1	PCIF0	page 86

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x1A (0x3A)	Reserved	-	-	-	-	-	-	-	-	
0x19 (0x39)	Reserved	-	-	-	-	-	-	-	-	
0x18 (0x38)	Reserved	-	-	-	-	-	-	-	-	
0x17 (0x37)	Reserved	-	-	-	-	-	-	-	-	
0x16 (0x36)	TIFR1	-	-	ICF1	-	OCF1C	OCF1B	OCF1A	TOV1	page 136
0x15 (0x35)	TIFR0	-	-	-	-	-	OCF0B	OCF0A	TOV0	page 107
0x14 (0x34)	Reserved	-	-	-	-	-	-	-	-	
0x13 (0x33)	Reserved	-	-	-	-	-	-	-	-	
0x12 (0x32)	Reserved	-	-	-	-	-	-	-	-	
0x11 (0x31)	Reserved	-	-	-	-	-	-	-	-	
0x10 (0x30)	Reserved	-	-	-	-	-	-	-	-	
0x0F (0x2F)	Reserved	-	-	-	-	-	-	-	-	
0x0E (0x2E)	Reserved	-	-	-	-	-	-	-	-	
0x0D (0x2D)	Reserved	-	-	-	-	-	-	-	-	
0x0C (0x2C)	Reserved	-	-	-	-	-	-	-	-	
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	page 83
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	page 83
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	page 83
0x08 (0x28)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	-	PORTC2	PORTC1	PORTC0	page 82
0x07 (0x27)	DDRC	DDC7	DDC6	DDC5	DDC4	-	DDC2	DDC1	DDC0	page 82
0x06 (0x26)	PINC	PINC7	PINC6	PINC5	PINC4	-	PINC2	PINC1	PINC0	page 82
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	page 82
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	page 82
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	page 82
0x02 (0x22)	Reserved	-	-	-	-	-	-	-	-	
0x01 (0x21)	Reserved	-	-	-	-	-	-	-	-	
0x00 (0x20)	Reserved	-	-	-	-	-	-	-	-	

Note:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Moreover reserved bits are not guaranteed to be read as “0”. 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 ATmega8U2/16U2/32U2 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 - $1FF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

29. Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
		ARITHMETIC AND LOGIC INSTRUCTIONS			
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
		BRANCH INSTRUCTIONS			
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
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
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

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
		DATA TRANSFER INSTRUCTIONS			
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
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

30. Ordering Information

30.1 ATmega8U2

Speed	Power Supply	Ordering Code	Package	Operational Range
16 MHz	2.7 - 5.5V	ATmega8U2-AU	32A	-40°C to +85°C
		ATmega8U2-MU	32M1-A	

Package Type	
32A	32-lead, 7 x7 x 1.2 mm, lead pitch 0.8 mm Thin Quad Flat Package
32M1	32-pad, 5 x 5 x 1 mm body, pad pitch 0.50 mm Quad Flat No lead (QFN)

30.2 ATmega16U2

Speed	Power Supply	Ordering Code	Package	Operational Range
16 MHz	2.7 - 5.5V	ATmega16U2-AU	32A	-40°C to +85°C
		ATmega16U2-MU	32M1-A	

Package Type	
32A	32-lead, 7 x7 x 1.2 mm, lead pitch 0.8 mm Thin Quad Flat Package
32M1	32-pad, 5 x 5 x 1 mm body, pad pitch 0.50 mm Quad Flat No lead (QFN)

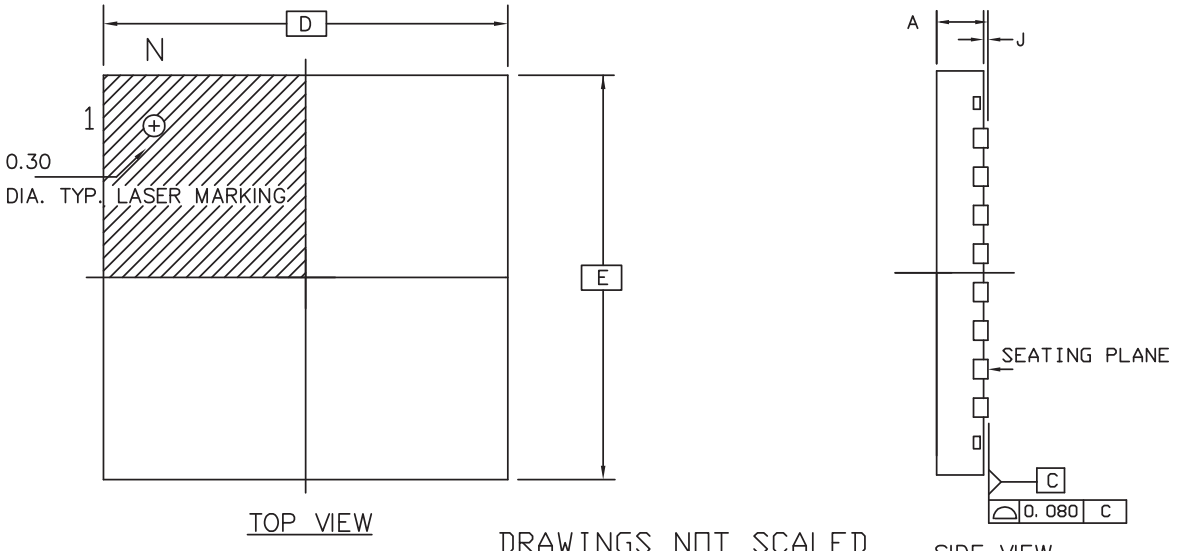
30.3 ATmega32U2

Speed	Power Supply	Ordering Code	Package	Operational Range
16 MHz	2.7 - 5.5V	ATmega32U2-AU	32A	-40°C to +85°C
		ATmega32U2-MU	32M1-A	

Package Type	
32A	32-lead, 7 x7 x 1.2 mm, lead pitch 0.8 mm Thin Quad Flat Package
32M1	32-pad, 5 x 5 x 1 mm body, pad pitch 0.50 mm Quad Flat No lead (QFN)

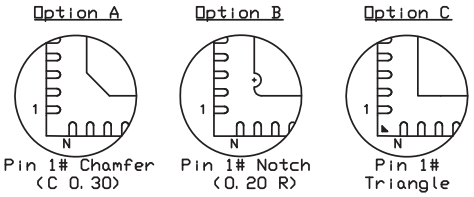
31. Packaging Information

31.1 QFN32



COMMON DIMENSIONS IN MM

SYMBOL	MIN.	NOM.	MAX.	NOTES
A	0.80	----	1.00	
J	0.00	----	0.05	
D/E		5.00 BSC		
D2/E2	3.50	3.60	3.70	
N		32		
e		0.50 BSC		
L	0.30	0.40	0.50	
b	0.18	0.25	0.30	

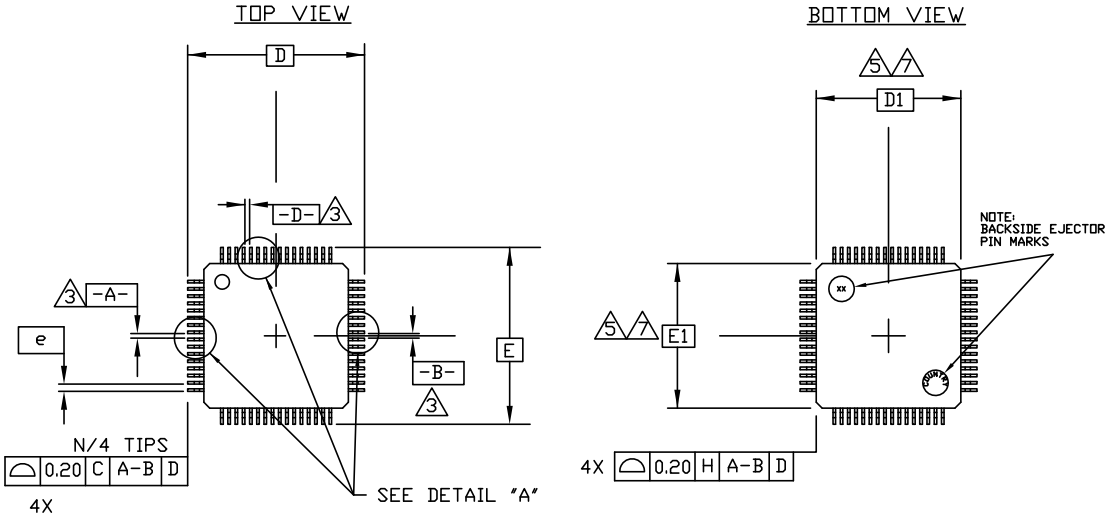


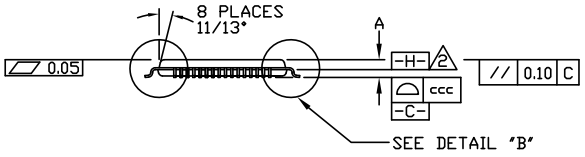
Compliant JEDEC Standard MO-220 variation VHHD-2

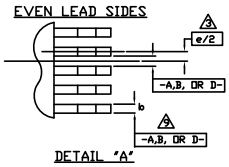
13/10/08

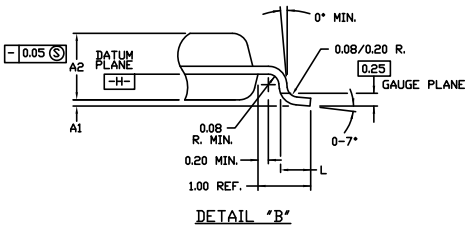
	TITLE	DRAWING No.	REV.
Atmel Nantes S.A. La Chantrerie - BP 70602 44306 Nantes Cedex 3 - France	PG - 32 Leads - 5.0x5.0mm body - 0.5mm pitch Quad Flat No Lead Package (QFN)	PG	A

31.2 TQFP32









SYMBOL	JEDEC VARIATION ALL DIMENSIONS IN MILLIMETERS			NOTE
	MIN.	NOM.	MAX.	
A			1.20	
A_1	0.05		0.15	
A_2	0.95	1.00	1.05	
D		9.00 BSC.		
D_1		7.00 BSC.		
E		9.00 BSC.		
E_1		7.00 BSC.		
L	0.45	0.60	0.75	
N		32		
e		0.80 BSC.		
b	0.30	0.37	0.45	
ccc			0.10	

32. Errata

32.1 Errata ATmega8U2

The revision letter in this section refers to the revision of the ATmega8U2 device.

32.1.1 rev. A and rev B

- **Full Swing oscillator**

1. Full Swing oscillator

The maximum frequency for the Full Swing Crystal Oscillator is 8MHz. For Crystal frequencies > 8MHz the Full Swing Crystal Oscillator is not guaranteed to operate correctly.

Problem fix/Workaround

If a Crystal with frequency > 8MHz is used, the Low Power Crystal Oscillator option should be used instead. See table 8-1 for an overview of the Device Clocking Options. Note that the Low Power Crystal Oscillator will not provide full rail-to-rail swing on the XTAL2 pin. If system clock output is needed to drive other clock inputs while running from the Low Power Crystal Oscillator, the system clock can be output on PORTC7 by programming the CKOUT fuse.

32.2 Errata ATmega16U2

The revision letter in this section refers to the revision of the ATmega16U2 device.

32.2.1 rev. A and rev B

- **Full Swing oscillator**

1. Full Swing oscillator

The maximum frequency for the Full Swing Crystal Oscillator is 8MHz. For Crystal frequencies > 8MHz the Full Swing Crystal Oscillator is not guaranteed to operate correctly.

Problem fix/Workaround

If a Crystal with frequency > 8MHz is used, the Low Power Crystal Oscillator option should be used instead. See table 8-1 for an overview of the Device Clocking Options. Note that the Low Power Crystal Oscillator will not provide full rail-to-rail swing on the XTAL2 pin. If system clock output is needed to drive other clock inputs while running from the Low Power Crystal Oscillator, the system clock can be output on PORTC7 by programming the CKOUT fuse.

32.3 Errata ATmega32U2

The revision letter in this section refers to the revision of the ATmega32U2 device.

32.3.1 rev. C

No Known Errata

32.3.2 rev. A and rev B

- **Full Swing oscillator**

1. Full Swing oscillator

The maximum frequency for the Full Swing Crystal Oscillator is 8MHz. For Crystal frequencies > 8MHz the Full Swing Crystal Oscillator is not guaranteed to operate correctly.

Problem fix/Workaround

If a Crystal with frequency > 8MHz is used, the Low Power Crystal Oscillator option should be used instead. See table 8-1 for an overview of the Device Clocking Options. Note that the Low Power Crystal Oscillator will not provide full rail-to-rail swing on the XTAL2 pin. If system clock output is needed to drive other clock inputs while running from the Low Power Crystal Oscillator, the system clock can be output on PORTC7 by programming the CKOUT fuse.

33. Datasheet Revision History

Please note that the referring page numbers in this section are referred to this document. The referring revision in this section are referring to the document revision.

33.1 Rev.7799E – 09/12

1. Renamed package name in Figure 1-1 on page 2 from VQFP32 to TQFP32.
2. Corrected typos.

33.2 Rev. 7799D – 11/10

1. Updated the footnote on page 2. Removed the VQFP from the footnote
2. Updated Section 20-4 ”Typical Bus powered application with 3.3V I/O” on page 187.
3. Updated Figure 20-6 on page 188. By connecting UVCC to 3V power-supply.
4. Updated Table 21-2 on page 215. 10: Bulk Type, and 01: Isochronous Type
5. Added UVCC limits in Electrical Characteristics
6. Updated “Electrical Characteristics” on page 264. Added USB D+ Internal Pull-up (streaming mode)
7. Updated “Register Summary” on page 288. Added DIDR1 (adress: 0x7F)
8. Removed Figure 27-26: USB Regulator Consumption with load 75Ω vs. Vcc

33.3 Rev. 7799C – 12/09

1. Updated “Features” on page 1.
2. Added description of “AVCC” on page 4.
3. Updated Figure 7-2 on page 18.
4. Updated Figure 20-3 on page 186 and Figure 20-4 on page 187.
5. Updated “Fuse Bits” on page 247.
6. Updated “DC Characteristics” on page 264.
7. Updated Table 26-3 on page 267, by removing Vrst.
8. Updated Table 26-4 on page 268.
9. Updated “Typical Characteristics” on page 273.
10. Added new “Errata” on page 299.

33.4 Rev. 7799B – 06/09

1. Updated “Typical Characteristics” on page 273.

33.5 Rev. 7799A – 03/09

1. Initial revision.

Table of Contents

Features *1*

1 Pin Configurations *2*

1.1 Disclaimer 2

2 Overview *3*

2.1 Block Diagram 3

2.2 Pin Descriptions 4

3 Resources *6*

4 Code Examples *6*

5 Data Retention *6*

6 AVR CPU Core *7*

6.1 Introduction 7

6.2 Architectural Overview 7

6.3 ALU – Arithmetic Logic Unit 8

6.4 Status Register 8

6.5 General Purpose Register File 10

6.6 Stack Pointer 11

6.7 Instruction Execution Timing 12

6.8 Reset and Interrupt Handling 13

7 AVR Memories *16*

7.1 In-System Reprogrammable Flash Program Memory 16

7.2 SRAM Data Memory 17

7.3 EEPROM Data Memory 18

7.4 I/O Memory 19

7.5 Register Description 20

8 System Clock and Clock Options *26*

8.1 Clock Systems and their Distribution 26

8.2 Clock Switch 27

8.3 Clock Sources 29

8.4 Low Power Crystal Oscillator 30

8.5 Full Swing Crystal Oscillator 32

8.6 Calibrated Internal RC Oscillator 33

8.7 External Clock 35

8.8Clock Output Buffer 35
8.9System Clock Prescaler 35
8.10PLL 36
8.11Register Description 37

9 Power Management and Sleep Modes 42

9.1Overview 42
9.2Sleep Modes 42
9.3Idle Mode 42
9.4Power-down Mode 43
9.5Power-save Mode 43
9.6Standby Mode 43
9.7Extended Standby Mode 43
9.8Power Reduction Register 43
9.9Minimizing Power Consumption 44
9.10Register Description 45

10 System Control and Reset 47

10.1Resetting the AVR 47
10.2Reset Sources 47
10.3Internal Voltage Reference 51
10.4Watchdog Timer 51
10.5Register Description 55

11 Interrupts 64

11.1Overview 64
11.2Interrupt Vectors in ATmega8U2/16U2/32U2 64
11.3Register Description 65

12 I/O-Ports 67

12.1Overview 67
12.2Ports as General Digital I/O 68
12.3Alternate Port Functions 72
12.4Register Description for I/O-Ports 82

13 External Interrupts 84

13.1Overview 84
13.2Register Description 84

14 Timer/Counter0 and Timer/Counter1 Prescalers 88

14.1Overview 88
14.2Internal Clock Source 88
14.3Prescaler Reset 88
14.4External Clock Source 88
14.5Register Description 89

15 8-bit Timer/Counter0 with PWM 91

15.1Features 91
15.2Overview 91
15.3Timer/Counter Clock Sources 92
15.4Counter Unit 92
15.5Output Compare Unit 93
15.6Compare Match Output Unit 95
15.7Modes of Operation 96
15.8Timer/Counter Timing Diagrams 100
15.9Register Description 102

16 16-bit Timer/Counter 1 with PWM 108

16.1Features 108
16.2Overview 108
16.3Accessing 16-bit Registers 110
16.4Timer/Counter Clock Sources 113
16.5Counter Unit 114
16.6Input Capture Unit 115
16.7Output Compare Units 117
16.8Compare Match Output Unit 119
16.9Modes of Operation 120
16.10Timer/Counter Timing Diagrams 127
16.11Register Description 129

17 SPI – Serial Peripheral Interface 138

17.1Features 138
17.2Overview 138
17.3$\overline{SS}$ Pin Functionality 142
17.4Data Modes 143
17.5Register Description 145

18 USART 148

18.1Features 148

18.2Overview 148
18.3Clock Generation 149
18.4Frame Formats 152
18.5USART Initialization 154
18.6Data Transmission – The USART Transmitter 155
18.7Data Reception – The USART Receiver 157
18.8Asynchronous Data Reception 161
18.9Multi-processor Communication Mode 164
18.10Hardware Flow Control 165
18.11Register Description 167
18.12Examples of Baud Rate Setting 171

19 USART in SPI Mode 176

19.1Features 176
19.2Overview 176
19.3Clock Generation 176
19.4SPI Data Modes and Timing 177
19.5Frame Formats 178
19.6Data Transfer 179
19.7Register Description 181
19.8AVR USART MSPIM vs. AVR SPI 183

20 USB Controller 185

20.1Features 185
20.2Overview 185
20.3USB Module Powering Options 186
20.4General Operating Modes 189
20.5Power modes 191
20.6Memory management 192
20.7PAD suspend 193
20.8D+/D- Read/write 194
20.9USB Software Operating modes 194
20.10Registers Description 195

21 USB Device Operating modes 197

21.1Overview 197
21.2Power-on and reset 197
21.3Endpoint reset 197

21.4USB reset 198
21.5Endpoint selection 198
21.6Endpoint activation 198
21.7Address Setup 199
21.8Suspend, Wake-up and Resume 200
21.9Detach 200
21.10Remote Wake-up 201
21.11STALL request 201
21.12CONTROL endpoint management 202
21.13OUT endpoint management 203
21.14IN endpoint management 205
21.15Isochronous mode 207
21.16Overflow 207
21.17Interrupts 208
21.18Register Description 209

22 Analog Comparator 223

22.1Overview 223
22.2Register Description 224

23 Boot Loader Support – Read-While-Write Self-Programming 226

23.1Features 226
23.2Overivew 226
23.3Application and Boot Loader Flash Sections 226
23.4Read-While-Write and No Read-While-Write Flash Sections 227
23.5Boot Loader Lock Bits 229
23.6Entering the Boot Loader Program 230
23.7Addressing the Flash During Self-Programming 232
23.8Self-Programming the Flash 233
23.9Register Description 242

24 debugWIRE On-chip Debug System 244

24.1Features 244
24.2Overview 244
24.3Physical Interface 244
24.4Software Break Points 245
24.5Limitations of debugWIRE 245
24.6Register Description 245

25 Memory Programming 246

25.1 Program And Data Memory Lock Bits 246
25.2 Fuse Bits 247
25.3 Signature Bytes 249
25.4 Calibration Byte 249
25.5 Page Size 249
25.6 Parallel Programming Parameters, Pin Mapping, and Commands 250
25.7 Parallel Programming 252
25.8 Serial Downloading 259
25.9 Serial Programming Pin Mapping 259

26 Electrical Characteristics 264

26.1 Absolute Maximum Ratings* 264
26.2 DC Characteristics 264
26.3 Speed Grades 266
26.4 Clock Characteristics 266
26.5 System and Reset Characteristics 267
26.6 External Interrupts Characteristics 268
26.7 SPI Timing Characteristics 269
26.8 Hardware Boot Entrance Timing Characteristics 270
26.9 Parallel Programming Characteristics 270

27 Typical Characteristics 273

27.1 Active Supply Current 273
27.2 Idle Supply Current 274
27.3 Power-down Supply Current 275
27.4 Pin Pull-Up 277
27.5 Pin Driver Strength 278
27.6 Pin Threshold and Hysteresis 280
27.7 BOD Threshold 281
27.8 Internal Oscilllator Speed 283
27.9 Current Consumption of Peripheral Units 286
27.10 Current Consumption in Reset and Reset Pulsewidth 287

28 Register Summary 288

29 Instruction Set Summary 292

30 Ordering Information 294

30.1ATmega8U2 294
30.2ATmega16U2 295
30.3ATmega32U2 296

31 Packaging Information 297

31.1QFN32 297
31.2TQFP32 298

32 Errata 299

32.1Errata ATmega8U2 299
32.2Errata ATmega16U2 299
32.3Errata ATmega32U2 299

33 Datasheet Revision History 301

33.1Rev.7799E – 09/12 301
33.2Rev. 7799D – 11/10 301
33.3Rev. 7799C – 12/09 301
33.4Rev. 7799B – 06/09 301
33.5Rev. 7799A – 03/09 302

Table of Contents i

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
JAPAN
Tel: (+81)(3) 3523-3551
Fax: (+81)(3) 3523-7581

© 2010 Atmel Corporation. All rights reserved. / Rev. CORP072610

Atmel®, logo and combinations thereof, and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries. 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 product 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.