Features

- High Performance, Low Power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 124 Powerful Instructions - Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 1 MIPS Throughput at 1 MHz
- Nonvolatile Program and Data Memories
  - 40K Bytes of In-System Self-Programmable Flash, Endurance: 10,000 Write/Erase Cycles
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - 512 bytes EEPROM, Endurance: 100,000 Write/Erase Cycles
  - 2K Bytes Internal SRAM
  - Programming Lock for Software Security
- On-chip Debugging
  - Extensive On-chip Debug Support
  - Available through JTAG interface
- Battery Management Features
  - Two, Three, or Four Cells in Series
  - Deep Under-voltage Protection
  - Over-current Protection (Charge and Discharge)
  - Short-circuit Protection (Discharge)
  - Integrated Cell Balancing FETs
  - High Voltage Outputs to Drive Charge/Precharge/Discharge FETs
- Peripheral Features
  - One 8-bit Timer/Counter with Separate Prescaler, Compare Mode, and PWM
  - One 16-bit Timer/Counter with Separate Prescaler and Compare Mode
  - 12-bit Voltage ADC, Eight External and Two Internal ADC Inputs
  - High Resolution Coulomb Counter ADC for Current Measurements
  - TWI Serial Interface for SM-Bus
  - Programmable Wake-up Timer
  - Programmable Watchdog Timer
- Special Microcontroller Features
  - Power-on Reset
  - On-chip Voltage Regulator
  - External and Internal Interrupt Sources
  - Four Sleep Modes: Idle, Power-save, Power-down, and Power-off
- Packages
  - 48-pin LQFP
- Operating Voltage: 4.0 - 25V
- Maximum Withstand Voltage (High-voltage pins): 28V
- Temperature Range: -30°C to 85°C
  - Speed Grade: 1 MHz

8-bit AVR® Microcontroller with 40K Bytes In-System Programmable Flash

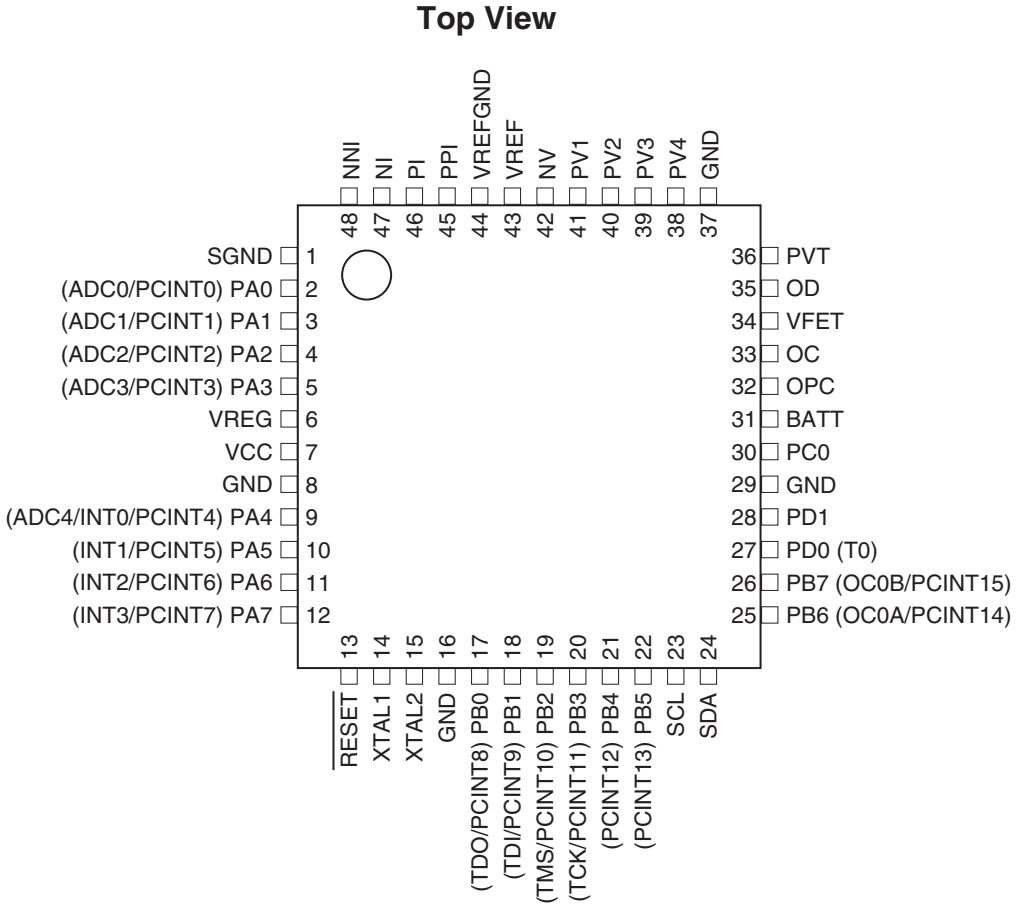
ATmega406

Preliminary

2548C–AVR–05/05

1. Pin Configurations

Figure 1-1. Pinout ATmega406.



1.1 Disclaimer

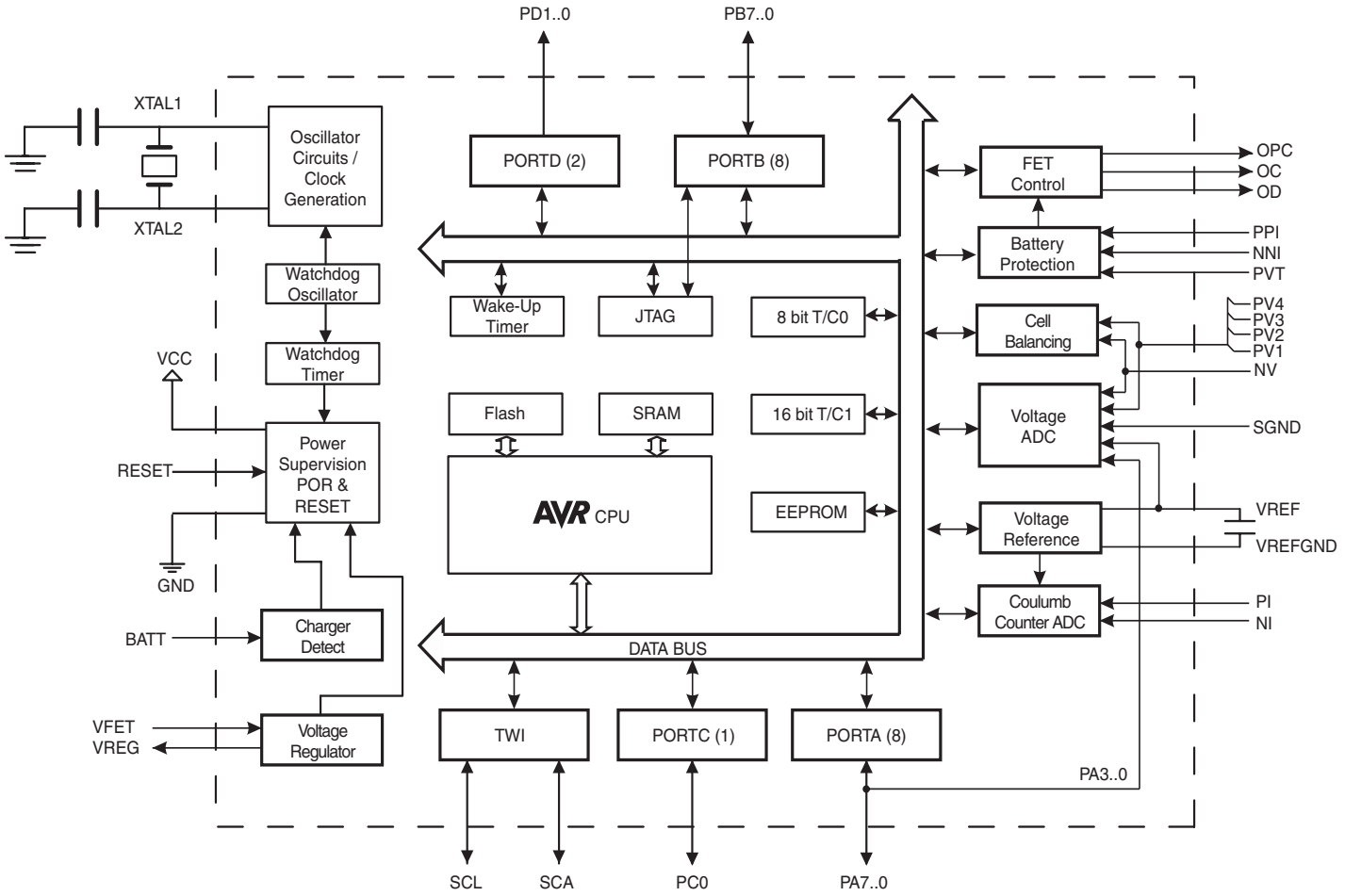
Typical values contained in this datasheet are based on simulations and characterization of other AVR microcontrollers manufactured on the same process technology. Min and Max values will be available after the device is characterized.

2. Overview

The ATmega406 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 ATmega406 achieves throughputs approaching 1 MIPS at 1 MHz.

2.1 Block Diagram

Figure 2-1. Block Diagram



The ATmega406 provides the following features: a Voltage Regulator, dedicated Battery Protection Circuitry, integrated cell balancing FETs, high-voltage analog front-end, and an MCU with two ADCs with On-chip voltage reference for battery fuel gauging.

The voltage regulator operates at a wide range of voltages, 4.0 - 25 volts. This voltage is regulated to a constant supply voltage of nominally 3.3 volts for the integrated logic and analog functions.

The battery protection monitors the battery voltage and charge/discharge current to detect illegal conditions and protect the battery from these when required. The illegal conditions are deep under-voltage during discharging, short-circuit during discharging and over-current during charging and discharging.

The integrated cell balancing FETs allow cell balancing algorithms to be implemented in software.

The MCU provides the following features: 40K bytes of In-System Programmable Flash with Read-While-Write capabilities, 512 bytes EEPROM, 2K byte SRAM, 32 general purpose working registers, 18 general purpose I/O lines, 11 high-voltage I/O lines, a JTAG Interface for On-chip Debugging support and programming, two flexible Timer/Counters with PWM and compare modes, one Wake-up Timer, an SM-Bus compliant TWI module, internal and external interrupts, a 12-bit Sigma Delta ADC for voltage and temperature measurements, a high resolution Sigma Delta ADC for Coulomb Counting and instantaneous current measurements, a programmable Watchdog Timer with internal Oscillator, and four software selectable power saving modes.

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 Idle mode stops the CPU while allowing the other chip function to continue functioning. The Power-down mode allows the voltage regulator, battery protection, regulator current detection, Watchdog Timer, and Wake-up Timer to operate, while disabling all other chip functions until the next Interrupt or Hardware Reset. In Power-save mode, the Wake-up Timer and Coulomb Counter ADC continues to run.

The device is manufactured using Atmel's high voltage high density non-volatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System, by a conventional non-volatile memory programmer or by an On-chip Boot program running on the AVR core. The Boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash, fuel gauging ADCs, dedicated battery protection circuitry, Cell Balancing FETs, and a voltage regulator on a monolithic chip, the Atmel ATmega406 is a powerful microcontroller that provides a highly flexible and cost effective solution for Li-ion Smart Battery applications.

The ATmega406 AVR is supported with a full suite of program and system development tools including: C Compilers, Macro Assemblers, Program Debugger/Simulators, and On-chip Debugger.

2.2 Pin Descriptions

2.2.1 VFET

Input to the internal voltage regulator.

2.2.2 VCC

Digital supply voltage. Normally connected to VREG.

2.2.3 VREG

Output from the internal voltage regulator.

2.2.4 VREF

Internal Voltage Reference for external decoupling.

2.2.5 VREFGND

Ground for decoupling of Internal Voltage Reference.

2.2.6 GND

Ground

2.2.7 SGND

Signal Ground.

2.2.8 Port A (PA7:PA0)

PA3:PA0 serves as the analog inputs to the Voltage A/D Converter.

Port A also serves as a low-voltage 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). As inputs, Port A pins that are externally pulled low will source current if the pull-up resistors are activated. The Port A pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port A also serves the functions of various special features of the ATmega406 as listed in ”Alternate Functions of Port A” on page 66.

2.2.9 Port B (PB7:PB0)

Port B is a low-voltage 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). 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 ATmega406 as listed in ”Alternate Functions of Port B” on page 68.

2.2.10 Port C (PC0)

Port C is a high voltage Open Drain output port.

2.2.11 Port D (PD1:PD0)

Port D is a low-voltage 2-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). As inputs, Port D pins that are externally pulled low will source current if the pull-up

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

Port D also serves the functions of various special features of the ATmega406 as listed in ”Alternate Functions of Port D” on page 70.

2.2.12 SCL

SMBUS clock, Open Drain bidirectional pin.

2.2.13 SDA

SMBUS data, Open Drain bidirectional pin.

2.2.14 OC

High voltage output to drive Charge FET.

2.2.15 OD

High voltage output to drive Discharge FET.

2.2.16 OPC

High voltage output to drive Pre-charge FET.

2.2.17 NI

NI is the filtered negative input from the current sense resistor.

2.2.18 NNI

NNI is the unfiltered negative input from the current sense resistor.

2.2.19 PI

PI is the filtered positive input from the current sense resistor.

2.2.20 PPI

PPI is the unfiltered positive input from the current sense resistor.

2.2.21 NV/PV1/PV2/PV3/PV4

NV, PV1, PV2, PV3, and PV4 are the inputs for battery cells 1, 2, 3, and 4.

2.2.22 PVT

PVT defines the pull-up level for the OD output.

2.2.23 BATT

Input for detecting when a charger is connected. This pin also defines the pull-up level for OC and OPC outputs.

2.2.24 RESET

Reset input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running. The minimum pulse length is given in Table 11 on page 38. Shorter pulses are not guaranteed to generate a reset.

2.2.25 XTAL1

Input to the inverting Oscillator amplifier.

2.2.26 XTAL2

Output from the inverting Oscillator amplifier.

3. About Code Examples

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

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

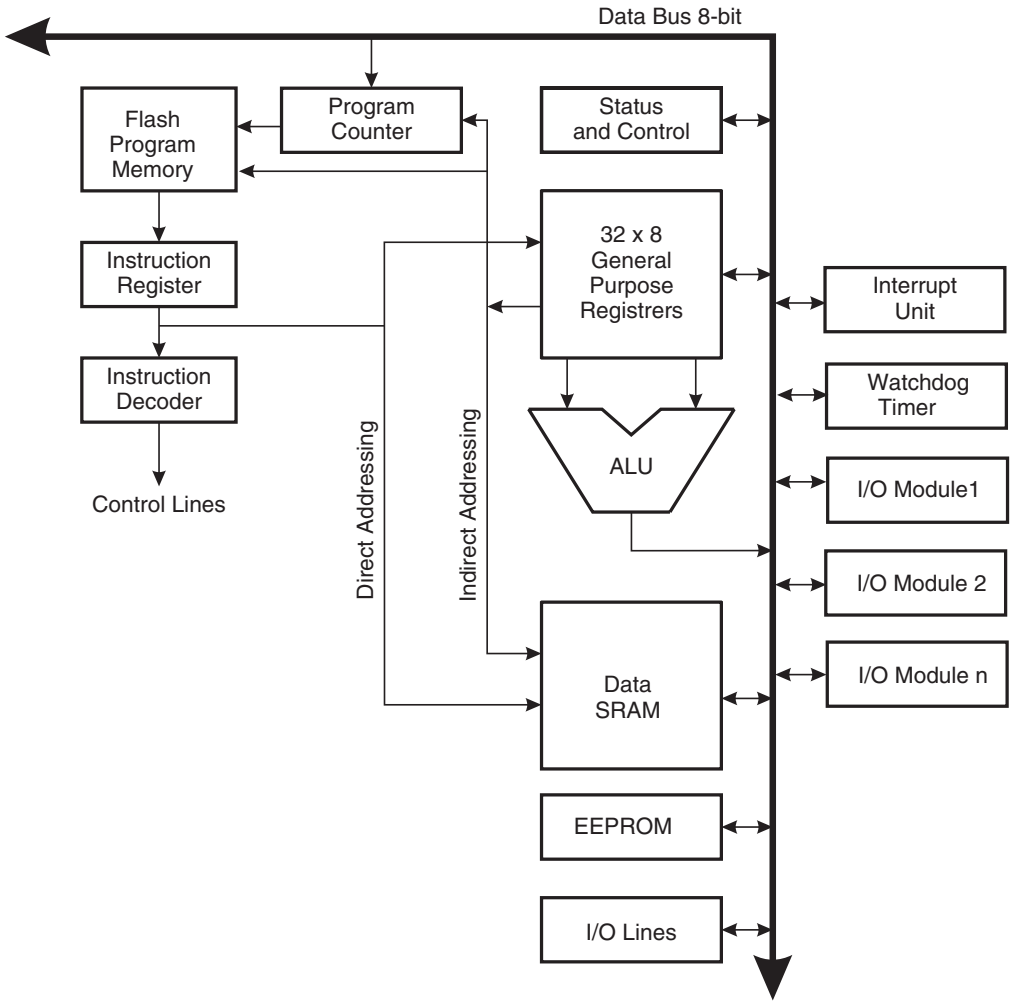
4. AVR CPU Core

4.1 Introduction

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

4.2 Architectural Overview

Figure 4-1. Block Diagram of the AVR Architecture



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

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

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

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

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. 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 ATmega406 has Extended I/O space from 0x60 - 0xFF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

4.3 ALU – Arithmetic Logic Unit

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

4.4 Status Register

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

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

The AVR Status Register – SREG – is defined as:

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

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

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

- **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 ”AVR Instruction Set” 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 ”AVR Instruction Set” 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 ”AVR Instruction Set” for detailed information.

- **Bit 2 – N: Negative Flag**

The Negative Flag N indicates a negative result in an arithmetic or logic operation. See the ”AVR Instruction Set” for detailed information.

- **Bit 1 – Z: Zero Flag**

The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See the ”AVR Instruction Set” for detailed information.

- **Bit 0 – C: Carry Flag**

The Carry Flag C indicates a carry in an arithmetic or logic operation. See the ”AVR Instruction Set” for detailed information.

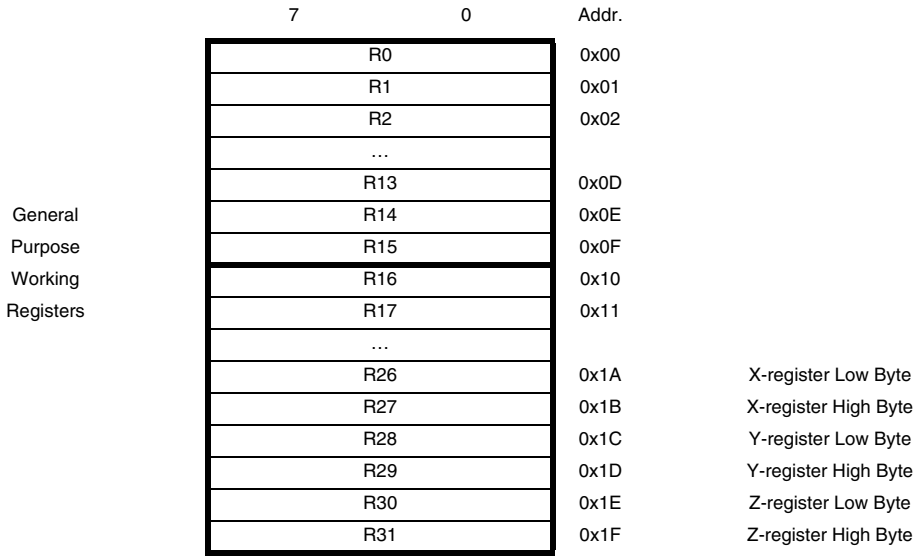
4.5 General Purpose Register File

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

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

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

Figure 4-2. AVR CPU General Purpose Working Registers



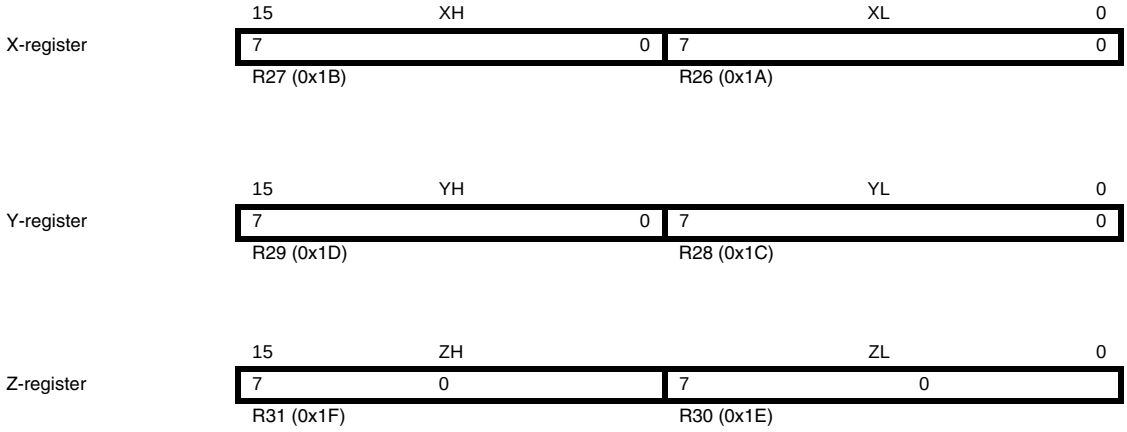
Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions.

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

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

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

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



In the different addressing modes these address registers have functions as fixed displacement, automatic increment, and automatic decrement (see the ”AVR Instruction Set” description for details).

4.6 Stack Pointer

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

The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. This Stack space in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. The Stack Pointer must be set to point above 0x100. The Stack Pointer is decremented by one when data is pushed onto the Stack with the PUSH instruction, and it is decremented by two when the return address is pushed onto the Stack with subroutine call or interrupt. The Stack Pointer is incremented by one when data is popped from the Stack with the POP instruction, and it is incremented by two when data is popped from the Stack with return from subroutine RET or return from interrupt RETI.

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

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

4.7 Instruction Execution Timing

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

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

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

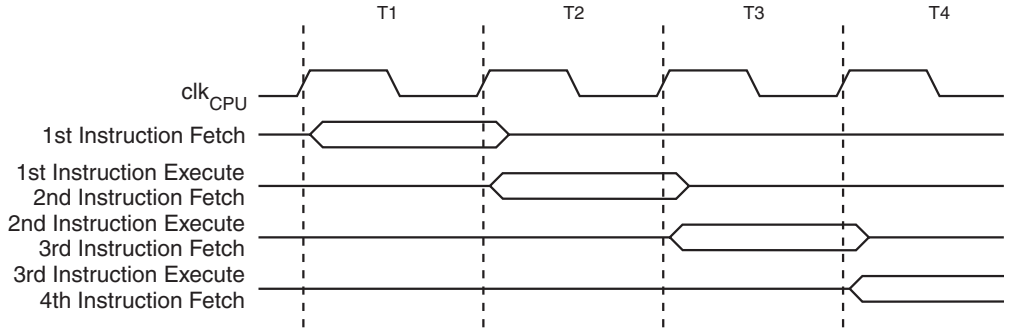
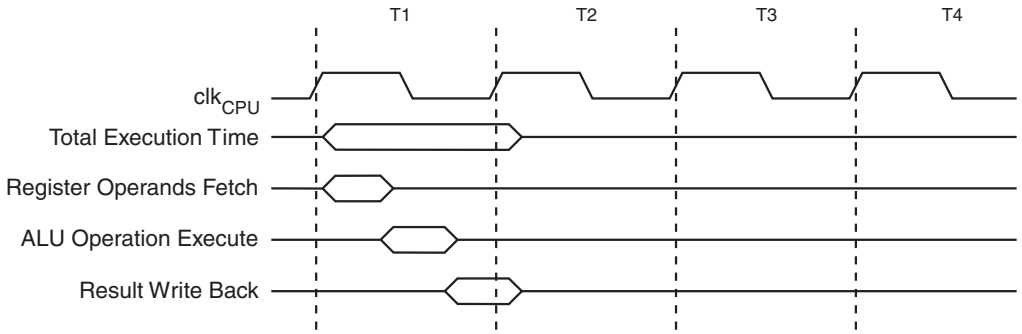


Figure 4-5 shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

Figure 4-5. Single Cycle ALU Operation



4.8 Reset and Interrupt Handling

The AVR provides several different interrupt sources. These interrupts and the separate Reset Vector each have a separate program vector in the program memory space. All interrupts are assigned individual enable bits which must be written logic one together with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt. Depending on the Program Counter value, interrupts may be automatically disabled when Boot Lock bits BLB02 or BLB12 are programmed. This feature improves software security. See the section ”Memory Programming” on page 185 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 49. The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority. 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 49 for more information. The Reset Vector can also be moved to the start of the Boot Flash section by programming the BOOTRST Fuse, see ”Boot Loader Support – Read-While-Write Self-Programming” on page 170.

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

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

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

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

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

When using the CLI instruction to disable interrupts, the interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the

CLI instruction. The following example shows how this can be used to avoid interrupts during the timed EEPROM write sequence.

Assembly Code Example

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

C Code Example

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

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

Assembly Code Example

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

C Code Example

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

4.8.1 Interrupt Response Time

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

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

5. AVR ATmega406 Memories

This section describes the different memories in the ATmega406. The AVR architecture has two main memory spaces, the Data Memory and the Program Memory space. In addition, the ATmega406 features an EEPROM Memory for data storage. All three memory spaces are linear and regular.

5.1 In-System Reprogrammable Flash Program Memory

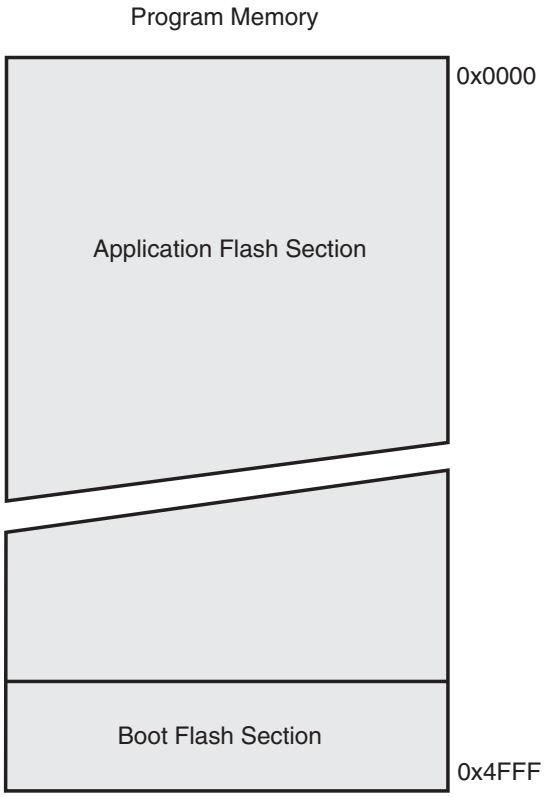
The ATmega406 contains 40K bytes On-chip In-System Reprogrammable Flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 20K x 16. For software security, the Flash Program memory space is divided into two sections, Boot Program section and Application Program section.

The Flash memory has an endurance of at least 10,000 write/erase cycles. The ATmega406 Program Counter (PC) is 15 bits wide, thus addressing the 20K program memory locations. The operation of Boot Program section and associated Boot Lock bits for software protection are described in detail in ”Boot Loader Support – Read-While-Write Self-Programming” on page 170. ”Memory Programming” on page 185 contains a detailed description on Flash data serial downloading.

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

Timing diagrams for instruction fetch and execution are presented in ”Instruction Execution Timing” on page 13.

Figure 5-1. Program Memory Map



5.2 SRAM Data Memory

Figure 5-2 shows how the ATmega406 SRAM Memory is organized.

The ATmega406 is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in the Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

The lower 2,304 data memory locations address both the Register File, the I/O memory, Extended I/O memory, and the internal data SRAM. The first 32 locations address the Register File, the next 64 location the standard I/O memory, then 160 locations of Extended I/O memory, and the next 2,048 locations address the internal data SRAM.

The five different addressing modes for the data memory cover: Direct, Indirect with Displacement, Indirect, Indirect with Pre-decrement, and Indirect with Post-increment. In the Register File, registers R26 to R31 feature the indirect addressing pointer registers.

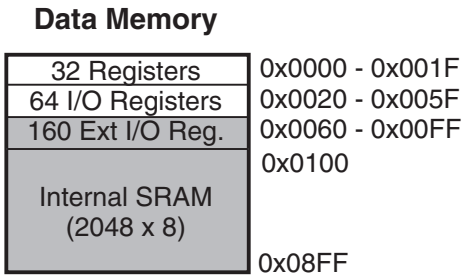
The direct addressing reaches the entire data space.

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

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

The 32 general purpose working registers, 64 I/O Registers, 160 Extended I/O Registers, and the 2,048 bytes of internal data SRAM in the ATmega406 are all accessible through all these addressing modes. The Register File is described in ”General Purpose Register File” on page 11.

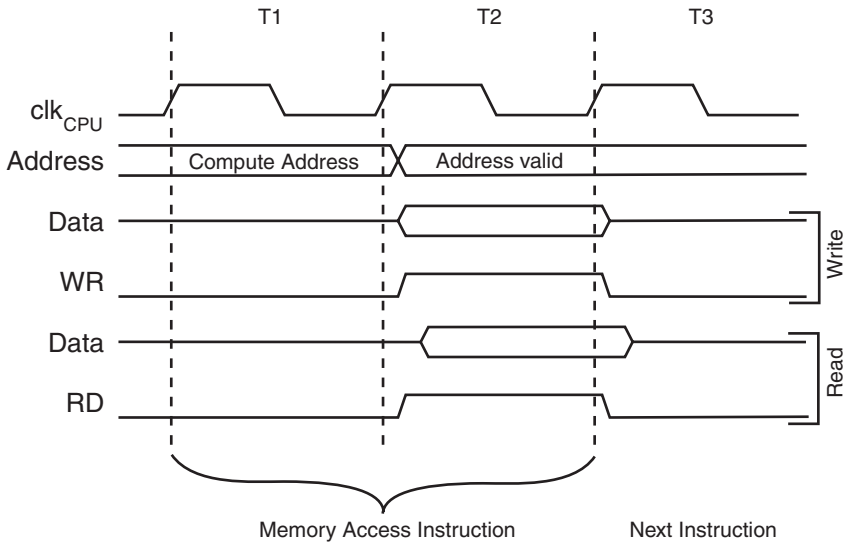
Figure 5-2. Data Memory Map



5.2.1 Data Memory Access Times

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

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



5.3 EEPROM Data Memory

The ATmega406 contains 512 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 Serial and Parallel data downloading to the EEPROM, see page 198 and page 188 respectively.

5.3.1 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 5-1. 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 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.

5.3.2 The EEPROM Address Register – EEARH and EEARL

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

- **Bits 15:9 – Res: Reserved Bits**

These bits are reserved bits in the ATmega406 and will always read as zero.

- **Bits 8:0 – EEAR8:0: 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 511. The initial value of EEAR is undefined. A proper value must be written before the EEPROM may be accessed.

5.3.3 The EEPROM Data Register – EEDR

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

- **Bits 7:0 – EEDR7:0: EEPROM Data**

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

5.3.4 The EEPROM Control Register – EECR

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

- **Bits 7:6 – Res: Reserved Bits**

These bits are reserved bits in the ATmega406 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 5-1. While EEPE is set, any write to EEPMn will be ignored. During reset, the EEPMn bits will be reset to 0b00 unless the EEPROM is busy programming.

Table 5-1. EEPROM Mode Bits

EEPM1	EEPM0	Programming Time	Operation
0	0	3.4 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 ”Boot Loader Support – Read-While-Write Self-Programming” on page 170 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 5-2 lists the typical programming time for EEPROM access from the CPU.

Table 5-2. 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

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

C Code Example

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

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

Assembly Code Example

```
EEPROM_read:
  ; Wait for completion of previous write
  sbic EECR,EEWE
  rjmp EEPROM_read
  ; Set up address (r18:r17) in address register
  out  EEARH, r18
  out  EEARL, r17
  ; Start eeprom read by writing EERE
  sbi  EECR,EERE
  ; Read data from data register
  in   r16,EEDR
  ret
```

C Code Example

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

5.4 I/O Memory

The I/O space definition of the ATmega406 is shown in ”Register Summary” on page 216.

All ATmega406 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 ATmega406 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

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

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

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

5.4.1 General Purpose I/O Registers

The ATmega406 contains three General Purpose I/O Registers. These registers can be used for storing any information, and they are particularly useful for storing global variables and Status Flags. General Purpose I/O Registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI, CBI, SBIS, and SBIC instructions.

5.4.2 General Purpose I/O Register 2 – GPIOR2

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

5.4.3 General Purpose I/O Register 1 – GPIOR1

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

5.4.4 General Purpose I/O Register 0 – GPIOR0

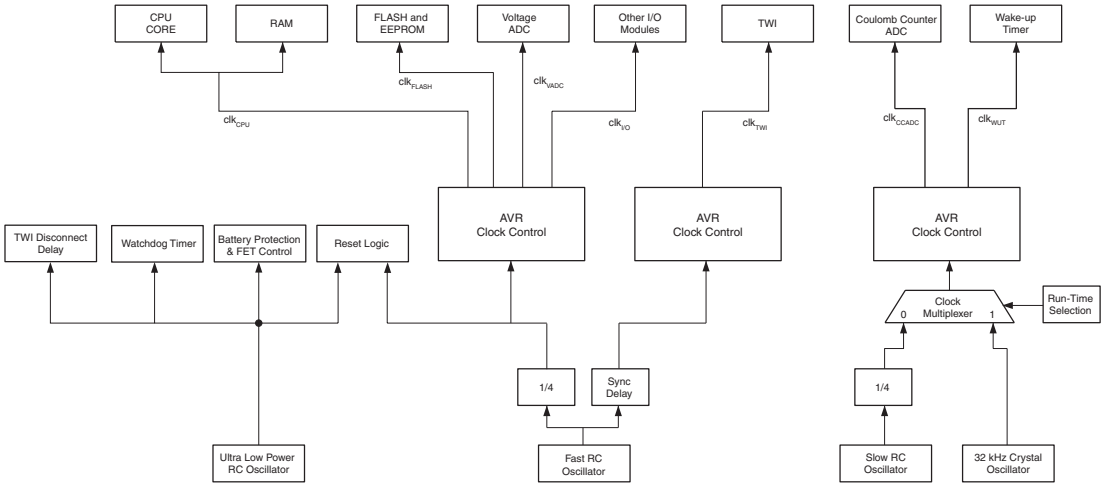
Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	GPIOR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

6. System Clock and Clock Options

6.1 Clock Systems and their Distribution

Figure 6-1 presents the principal clock systems in the AVR and their distribution. All of the clocks need not be active at a given time. In order to reduce power consumption, the clocks to modules not being used can be halted by using different sleep modes, as described in "Power Management and Sleep Modes" on page 31. The clock systems are detailed below.

Figure 6-1. Clock Distribution



6.1.1 CPU Clock – clk_{CPU}

The CPU clock is routed to parts of the system concerned with operation of the AVR core. Examples of such modules are the General Purpose Register File, the Status Register and the data memory holding the Stack Pointer. Halting the CPU clock inhibits the core from performing general operations and calculations.

6.1.2 TWI Clock - clk_{TWI}

The TWI module is provided with a dedicated clock domain. This is because the TWI module requires a 4 MHz clock to achieve the specified Data Transfer Speed. It also allows power reduction by halting the clk_{TWI} clock when TWI communication is not used. Note that address match detection in the TWI module is carried out asynchronously when clk_{TWI} is halted, enabling TWI address watch detection in all sleep modes except Power-off.

6.1.3 I/O Clock – $clk_{I/O}$

The I/O clock is used by the majority of the I/O modules. 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.

6.1.4 Flash Clock – clk_{FLASH}

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

6.1.5 Voltage ADC Clock – clk_{VADC}

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

6.1.6 Coulomb Counter ADC Clock - clk_{CCADC}

The Coulomb Counter ADC is provided with a dedicated clock domain. This allows operating the Coulomb Counter ADC in low power modes like Power-save for continuous current measurements.

6.1.7 Watchdog Timer and Battery Protection Clock

The Watchdog Timer and Battery Protection are provided with a dedicated clock domain. This allows operation in all modes except Power-off. It also allows very low power operation by utilizing an Ultra Low Power RC Oscillator dedicated to this purpose.

6.2 Clock Sources

The device has the following clock sources. The clocks are input to the AVR clock generator, and routed to the appropriate modules.

6.3 Calibrated Fast RC Oscillator

The calibrated Fast RC Oscillator by default provides a 4.0 MHz clock, which is divided down to 1.0 MHz to all modules except the TWI. The frequency is nominal value at 25°C. This clock will operate with no external components. During reset, hardware loads the calibration byte into the FOSCCAL Register and thereby automatically calibrates the Fast RC Oscillator. At 25°C, this calibration gives a frequency of 4 MHz ± 3%. The oscillator can be calibrated to any frequency in the range 3.7 - 4.0 MHz within ±1% accuracy, by changing the FOSCCAL register. For more information on the pre-programmed calibration value, see the section ”Calibration Bytes” on page 187.

When this Oscillator is selected, start-up times are determined by the SUT Fuses as shown in Table 6-1 on page 26.

Table 6-1. Start-up times for the internal calibrated RC Oscillator clock selection

SUT1:0	Start-up Time from Power-down and Power-save	Additional Delay from Reset
00	6 CK	14CK
01	6 CK	14CK + 4.1 ms
10	6 CK	14CK + 65 ms(1)
11	Reserved	

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

6.3.1 Fast RC Oscillator Calibration Register – FOSCCAL

Bit	7	6	5	4	3	2	1	0	
	FCAL7	FCAL6	FCAL5	FCAL4	FCAL3	FCAL2	FCAL1	FCAL0	FOSCCAL
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 – FCAL7:0: Fast RC Oscillator Calibration Value**

The Fast RC Oscillator Calibration Register is used to trim the Fast RC Oscillator to remove process variations from the oscillator frequency. The factory-calibrated value is automatically written to this register during chip reset, giving an oscillator frequency of 4.0 MHz at 25°C. The application software can write this register to change the oscillator frequency. The oscillator can be calibrated to any frequency in the range 3.7 - 4.0 MHz within ±1% accuracy. Calibration outside that range is not guaranteed.

Note that this oscillator is used to time EEPROM and Flash write accesses, and these write times will be affected accordingly. If the EEPROM or Flash are written, do not calibrate to more than 4.4 MHz. Otherwise, the EEPROM or Flash write may fail.

The FCAL7 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 FOSCCAL = 0x7F gives a higher frequency than FOSCCAL = 0x80.

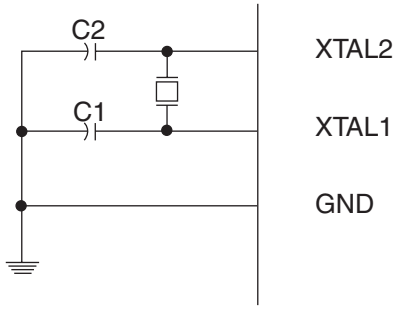
The FCAL6:0 bits are used to tune the frequency within the selected range. A setting of 0x00 gives the lowest frequency in that range, and a setting of 0x7F gives the highest frequency in the range. Incrementing FCAL6:0 by 1 will give a frequency increment of less than 2% in the frequency range 3.7 - 4.0 MHz.

6.4 32 kHz Crystal Oscillator

XTAL1 and XTAL2 are input and output, respectively, of an inverting amplifier which can be configured for use as an On-chip Oscillator, as shown in Figure 6-2. This Oscillator is optimized for use with a 32.768 kHz watch crystal.

C1 and C2 should always be equal. The optimal value of the capacitors depends on the crystal or resonator in use, the amount of stray capacitance, and the electromagnetic noise of the environment. For information on how to choose capacitors and other details on Oscillator operation, refer to the 32 kHz Crystal Oscillator application note.

Figure 6-2. 32 kHz Crystal Oscillator Connections



6.5 Slow RC Oscillator

The Slow RC Oscillator provides a fixed 131 kHz clock. This clock source can be used as a backup clock source in case of 32 kHz Crystal Oscillator failure. It can also be used as the only Run-Time clock source in systems where the resulting clock accuracy is acceptable. To provide good accuracy when used as a Run-Time clock source, the slow RC Oscillator has a calibration byte stored in the signature address space. See the section "Calibration Bytes" on page 187. In order to get the actual timeout periods, the application software must use this calibration byte to scale the WUT time-outs found in Table 9-1 on page 48.

6.6 Ultra Low Power RC Oscillator

The Ultra Low Power RC Oscillator (ULP Oscillator) provides a clock of 128 kHz. It operates at very low power consumption, at the expense of frequency accuracy.

6.7 CPU, I/O, Flash, and Voltage ADC Clock

The clock source for the CPU, I/O, Flash, and Voltage ADC is the calibrated Fast RC Oscillator. Note that the Calibrated Fast RC Oscillator will provide a 4 MHz clock to the TWI module and a 1 MHz clock to all other modules.

When the CPU wakes up from Power-down or Power-save, the CPU clock source is used to time the start-up, ensuring a stable clock before instruction execution starts. When the CPU starts from reset, there is an additional delay allowing the voltage regulator to reach a stable level before commencing normal operation. The Ultra Low Power RC Oscillator is used for timing this real-time part of the start-up time. Start-up times are determined by the SUT Fuses as shown in Table 6-2. The number of Ultra Low Power RC Oscillator cycles used for each time-out is shown in Table 6-3.

Table 6-2. Start-up Times for the Calibrated Fast RC Oscillator

SUT1:0	Start-up Time from Power-down and Power-save	Additional Delay from Reset
00	6 CK	14CK
01	6 CK	14CK + 3.9 ms
10	6 CK	14CK + 62.5 ms
11	Reserved	

Table 6-3. Number of Ultra Low Power RC Oscillator Cycles

Typ Time-out	Number of Cycles
3.9 ms	500
62.5 ms	8000

6.8 Coulomb Counter ADC and Wake-up Timer Clock

The Coulomb Counter ADC and Wake-up Timer clock operates asynchronously with the CPU clock, to allow low power operation in sleep modes. The clock source is either the 32 kHz Crystal Oscillator, or the Slow RC Oscillator (divided by 4). The selected clock is input to the AVR Clock Control Unit, and is routed to the appropriate modules.

The clock source for the Coulomb Counter ADC and Wake-up Timer is selected by an I/O bit in the Clock Control and Status Register, see ”Run-Time Clock Source Select” on page 29 for details.

6.9 Watchdog Timer and Battery Protection Clock

The clock source for the Watchdog Timer and Battery Protection is the Ultra Low Power RC Oscillator. The Oscillator is automatically enabled in all operational modes where either the Watchdog Timer, the Battery Protection, or both, are enabled. It is also enabled during reset.

6.10 Run-Time Clock Source Select

The clock source for the Coulomb Counter ADC and Wake-up Timer is run-time selectable as either the 32 kHz Crystal Oscillator, or the Slow RC oscillator (divided by 4). The clock source is selected by an I/O bit in the Clock Control and Status Register.

The 32 kHz Crystal Oscillator is the recommended clock source in order to achieve the highest clock accuracy. The Slow RC Oscillator is provided as a clock source for low cost systems, or as an alternate clock source in case of crystal clock failure. If the CPU detects that the crystal clock is not operating correctly, it can switch to the Slow RC Oscillator as a less accurate, but still functional, backup solution.

6.10.1 Clock Control and Status Register - CCSR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	XOE	ACS	CCSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:2 - Res: Reserved Bits**

These bits are reserved bits in the ATmega406 and will always read as zero.

- **Bit 1 - XOE: 32 kHz Crystal Oscillator Enable**

The XOE bit is used to enable the 32 kHz Crystal Oscillator before it is selected as clock source. This allows the Oscillator clock to stabilize prior to use. The 32 kHz Crystal Oscillator requires approximately two seconds to stabilize, this must be timed by the user software. If the software tries to write a one to ACS and a zero to XOE at the same time, both XOE and ACS will be cleared by the hardware. Thus, while the 32 kHz Crystal Oscillator is disabled it is not possible to select it as a clock source .

- **Bit 0 - ACS: Asynchronous Clock Select**

The ACS bit is used to selected the source of the asynchronous clock for the Coulomb Counter ADC and Wake-up Timer. The Slow RC Oscillator is selected when this bit is cleared (zero). The 32 kHz Crystal Oscillator is selected when this bit is set (one).

The selected clock source and oscillator enable conditions are illustrated in Table 6-4.

Table 6-4. Asynchronous Clock Source and Oscillator Enable Conditions

Sleep Mode	32 kHz Crystal Oscillator Enable	Slow RC Oscillator Enable
Power-off or Power-down	0	0
Other Sleep Modes	XOE	ACS & (CADEN | WUTEN)
Active Mode	XOE	1

Recommended algorithm for switching from the RC Oscillator to the Crystal Oscillator as the asynchronous clock for the Coulomb Counter ADC and Wake-up Timer:

1. Enable the Crystal Oscillator by setting the XOE bit (one).
2. Enable the Wake-up Timer, select a two second timeout, and reset the Wake-up Timer (”Wake-up Timer” on page 47 for details).
3. Wait for the Wake-up Timer time-out.
4. Switch to the Crystal Oscillator by setting the ACS bit (one) while keeping the XOE bit set (one).
5. Optional: Wait for another Wake-up Timer time-out, to ensure the Crystal Oscillator is operating correctly. This can be done by enabling another timer interrupt with significantly longer time-out, and checking that the Wake-up Timer time-out occurs first.

Recommended algorithm for switching from the Crystal Oscillator to the RC Oscillator as the asynchronous clock for the Coulomb Counter ADC and Wake-up Timer:

1. Switch to the RC Oscillator by clearing the ACS bit (zero) while keeping the XOE bit set (one).
2. Disable the Crystal Oscillator by clearing the XOE bit (zero) while keeping the ACS bit cleared (zero).

7. Power Management and Sleep Modes

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

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 SM2:0 bits in the SMCR Register select which sleep mode (Idle, ADC Noise Reduction, Power-down, Power-save, or Power-off) will be activated by the SLEEP instruction. See Table 7-1 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 any sleep mode except Power-off. If a reset occurs during sleep mode, the MCU wakes up and executes from the Reset Vector. The MCU will reset when returning from Power-off mode.

Figure 6-1 on page 25 presents the different clock systems in the ATmega406, and their distribution. The figure is helpful in selecting an appropriate sleep mode.

7.0.1 Sleep Mode Control Register – SMCR

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

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	SM2	SM1	SM0	SE	SMCR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:4 – Res: Reserved Bits**

These bits are reserved bits in the ATmega406, and will always read as zero.

- **Bits 3:1 – SM2:0: Sleep Mode Select Bits 2, 1 and 0**

These bits select between the five available sleep modes as shown in Table 7-1.

Table 7-1. Sleep Mode Select

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

Note: 1. SMCR is auto-cleared after 4 cycles when this value is set and the SE bit is written to logic one. To enter this mode, execute SLEEP instruction within 4 cycles after writing SE to logic one.

- **Bit 0 – SE: Sleep Enable**

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

7.1 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 all peripheral functions 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 interrupt.

7.2 ADC Noise Reduction Mode

When the SM2:0 bits are written to 001, the SLEEP instruction makes the MCU enter ADC Noise Reduction mode, stopping the CPU but allowing the Voltage ADC (V-ADC), Wake-up Timer (WUT), Watchdog Timer (WDT), Coulomb Counter (CC), Current Battery Protection (CBP), Voltage Battery Protection (VBP), Wake-up on Regular Current (WURC), 32 kHz crystal Oscillator (XOSC_32K) or Slow RC Oscillator (RCOSC_SLOW), and the Ultra Low Power RC Oscillator (RCOSC_ULP) to continue operating. This sleep mode basically halts $clk_{I/O}$, clk_{CPU}, and clk_{FLASH}, while allowing the other clocks to run.

This improves the noise environment for the Voltage ADC, enabling higher resolution measurements.

7.3 Power-save Mode

When the SM2:0 bits are written to 011, the SLEEP instruction makes the MCU enter Power-save mode. In this mode, the internal Fast RC Oscillator (RCOSC_FAST) is stopped, while Wake-up Timer (WUT), Watchdog Timer (WDT), Coulomb Counter (CC), Current Battery Protection (CBP), Voltage Battery Protection (VBP), Wake-up on Regular Current (WURC), 32 kHz crystal Oscillator (XOSC_32K) or Slow RC Oscillator (RCOSC_SLOW) and the Ultra Low Power RC Oscillator (RCOSC_ULP) continue operating.

This mode will be the default mode when application software does not require operation of CPU, Flash or any of the periphery units running at the Fast internal Oscillator (RCOSC_FAST).

If the current through the sense resistor is so small that the Coulomb Counter cannot measure it accurately, Regular Current detection should be enabled to reduce power consumption. The WUT keeps accurately track of the time so that battery self discharge can be calculated.

Note that if a level triggered interrupt is used for wake-up from Power-save mode, the changed level must be held for some time to wake up the MCU. Refer to "External Interrupts" on page 54 for details.

When waking up from Power-save 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 in "Clock Sources" on page 26.

7.4 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 Fast RC Oscillator (RCOSC_FAST), 32 kHz crystal Oscillator

(XOSC_32K), and Slow RC Oscillator (RCOSC_SLOW) are stopped, while the external interrupts and the Watchdog continue operating (if enabled). This sleep mode basically halts all generated clocks, allowing operation of asynchronous modules only.

7.5 Power-off Mode

When the SM2:0 bits are written to 100, the SLEEP instruction makes the CPU ask the voltage regulator to shut off power to the CPU, leaving only the Regulator and the Charger Detect Circuitry to be operational.

Table 7-2. Active modules in different Sleep Modes

Module	Mode					
	Active	**Idle**	**ADC NRM**	**Power-save**	**Power-down**	**Power-off**
RCOSC_FAST	X	X	X			
RCOSC_ULP	X	X	X	X	X	
XOSC_32K/ RCOSC_SLOW	X	X	X	X		
CPU	X					
Flash	X					
8-bit Timer/16-bit Timer	X	X				
SMBus	X	X	X(1)	X(1)	X(1)	
V-ADC	X	X	X			
CC-ADC	X	X	X	X		
External Interrupts	X	X	X	X	X	
CBP(2)	X	X	X	X	X	
VBP	X	X	X	X	X	
WDT	X	X	X	X	X	
WUT	X	X	X	X		
VREG	X	X	X	X	X	X
CHARGER_DETECT						X

Note: 1. Address Match and Bus Connect/Disconnect Wake-up only.

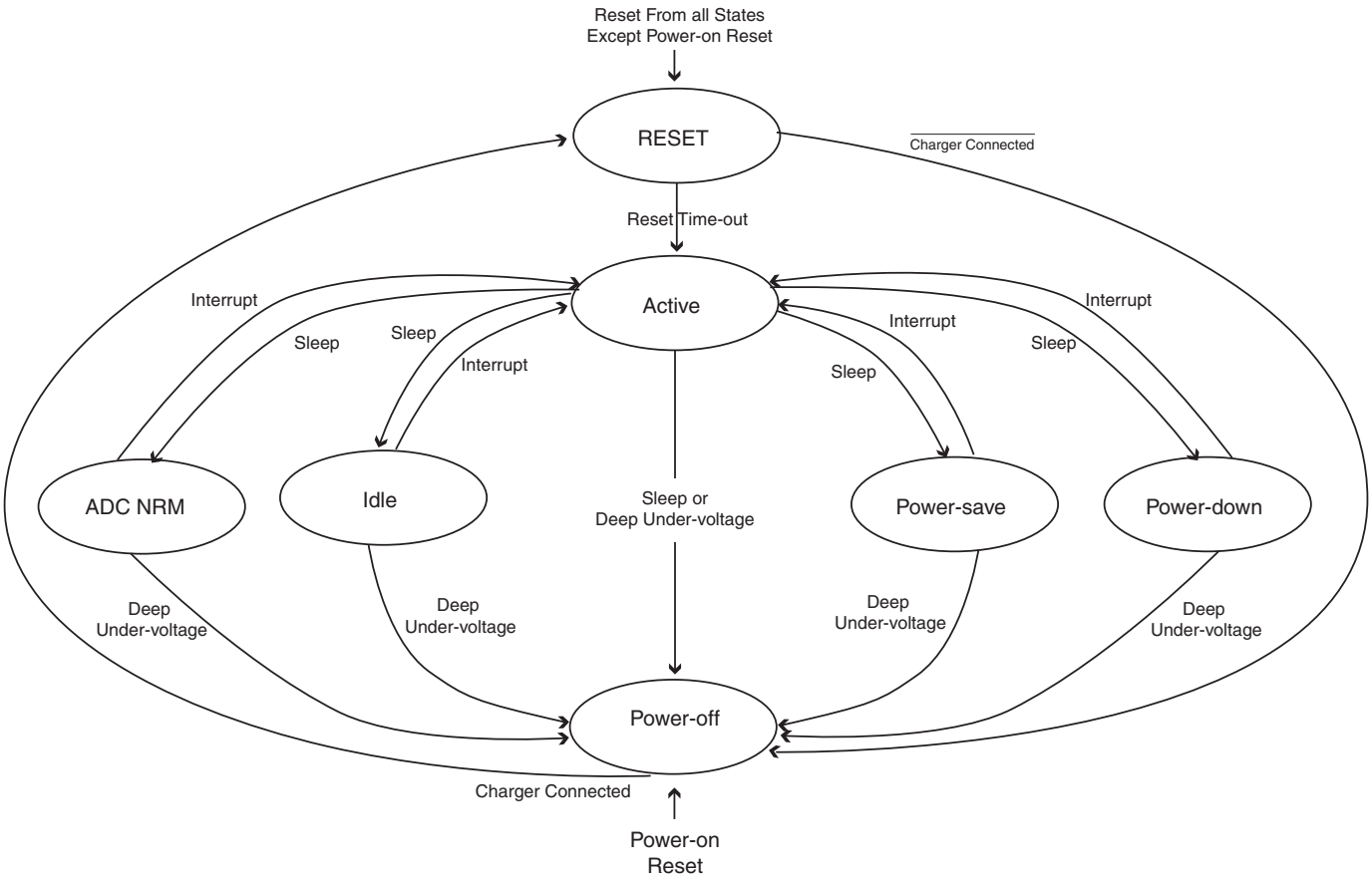
2. When Discharge-FET is switched off, Short-circuit Protection is automatically disabled to reduce current consumption.

Table 7-3. Wake-up Sources for Sleep Modes

Mode	Wake-up sources										
	Wake-up on Regular Current	Battery Protection Interrupts	External Interrupts	SMBus Address Match and Bus Connect/Disconnect	WDT	WUT	SPM/EEPROM Ready	CC-ADC	V-ADC	Other I/O	Charger Connect
Idle	X	X	X	X	X	X	X	X	X	X	
ADC NRM	X	X	X	X	X	X	X	X	X		
Power-save	X	X	X	X	X	X		X			
Power-down		X	X	X	X						
Power-off											X

The sleep mode state diagram is shown in Figure 7-1.

Figure 7-1. Sleep Mode State Diagram



7.6 Power Reduction Register

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

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

7.6.1 Power Reduction Register 0 - PRR0

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	PRTWI	PRTIM1	PRTIM0	PRVADC	PRR0
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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

These bits are reserved in ATmega406 and will always read as zero.

- **Bit 3 - PRTWI: Power Reduction TWI**

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

- **Bit 2 - 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 1 - 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 0 - PRVADC: Power Reduction V-ADC**

Writing a logic one to this bit shuts down the V-ADC. The V-ADC must be disabled before shut down.

7.7 Minimizing Power Consumption

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

7.7.1 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 except Power-off. The Watchdog Timer current consumption is significant only in Power-down mode. See "Watchdog Timer" on page 42 for details on how to configure the Watchdog Timer.

7.7.2 Port Pins

When entering a sleep mode, all port pins should be configured to use minimum power. The most important is then to ensure that no pins drive resistive loads. In sleep modes where both the I/O clock ($clk_{I/O}$) and the ADC clock (clk_{ADC}) are stopped, the input buffers of the device will be disabled. This ensures that no power is consumed by the input logic when not needed. In some cases, the input logic is needed for detecting wake-up conditions, and it will then be enabled. See "Digital Input Enable and Sleep Modes" on page 62 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 $V_{REG}/2$, 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 $V_{REG}/2$ 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 Register. Refer to "Digital Input Disable Register 0 – DIDR0" on page 114 for details.

7.7.3 On-chip Debug System

If the On-chip debug system is enabled by 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.

7.7.4 Battery Protection

If one of the Battery Protection features is not needed by the application, this feature should be disabled, see "Battery Protection Control Register – BPCR" on page 121. When the Discharge FET is switched off, the Short-Circuit Circuitry will automatically be stopped in order to minimize power consumption. The current consumption in the Battery Protection circuitry is only significant in Power-down mode.

8. System Control and Reset

8.1 Resetting the AVR

During reset, all I/O Registers are set to their initial values, and the program starts execution from the Reset Vector. The instruction placed at the Reset Vector must be 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 8-1 shows the reset logic. Table 8-1 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 voltage regulator 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 Fuses. The different selections for the delay period are presented in ”Clock Sources” on page 26.

8.2 Reset Sources

The ATmega406 has five sources of reset:

- The Power-on Reset module generates a Power-on Reset when the supply voltage at the V_{FET} pin is below the Power-on Reset Threshold, V_{POT}, for a time longer than the Power-on Reset Response time, T_{POR}. This will activate Power-off mode.
- Charger Connect Reset. If the chip is in Power-off mode, the Charger Detect module generates a reset pulse when a charger is connected. The CHARGER CONNECTED signal also enables transition to the RESET state. This is the only way to exit from Power-off mode. See Figure 7-1 on page 34.
- External Reset. The MCU is reset when a low level is present on the 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 V_{CC} is below the Brown-out Reset Threshold, V_{BOT}. See ”Brown-out Detection” on page 40 for details.
- JTAG AVR Reset. The MCU is Reset as long as there is a logic one in the Reset Register, one of the scan chains of the JTAG system. See ”JTAG Interface and On-chip Debug System” on page 164 for details.

Figure 8-1. Reset Logic

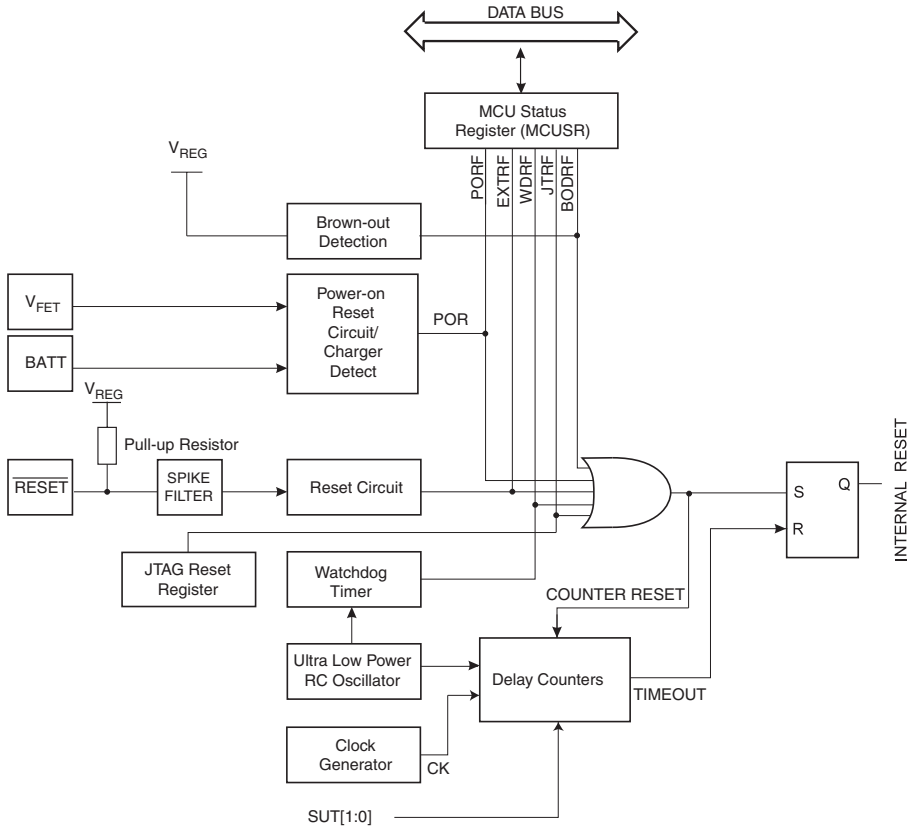


Table 8-1. Reset Characteristics

Symbol	Parameter	Condition	Min	Typ	Max	Units
V_{POT}	Power-on Threshold Voltage		3.0		4.0	V
V_{COT}	Charger-on Threshold Voltage		6	7	8	V
V_{RST}	$\overline{\text{RESET}}$ Pin Threshold Voltage	V_{REG} = 3.3V	0.66		2.8	V
t_{RST}	Minimum pulse width on $\overline{\text{RESET}}$ Pin			900		ns

8.3 Power-on Reset and Charger Connect

When the supply voltage at the VFET pin rise above the Power-On Threshold, VPOT, the regulator will start up in Power-off mode. The chip will stay in Power-off mode until the Charger Detect module detects a charger.

In order to detect a charger, the voltage at the BATT pin must rise above the VCOT level. This will issue a Power-On Reset (POR), and the chip enters RESET mode. When the Delay Counter times out, the chip will enter Active mode. See Figure 7-1 on page 34.

Figure 8-2. Powering up from battery cell side before charger side.

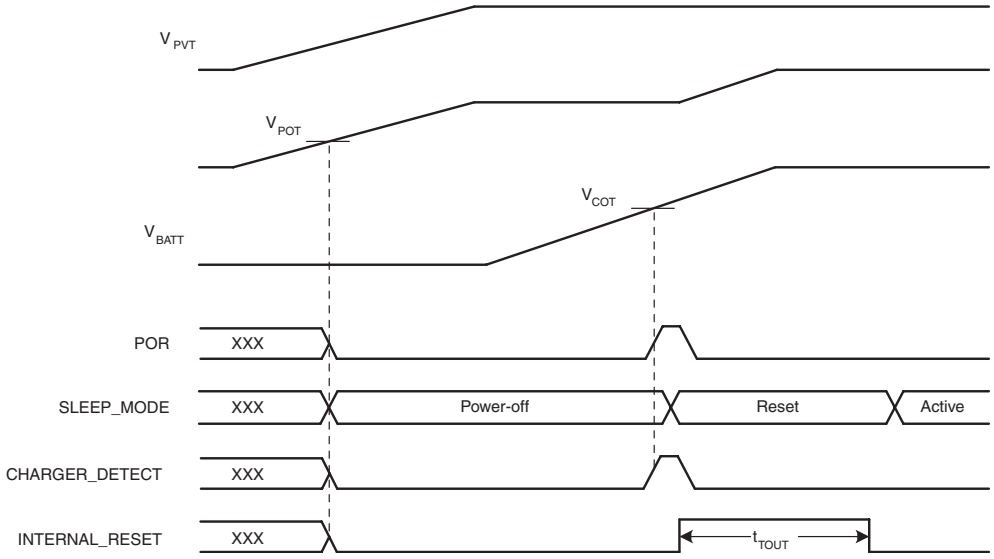
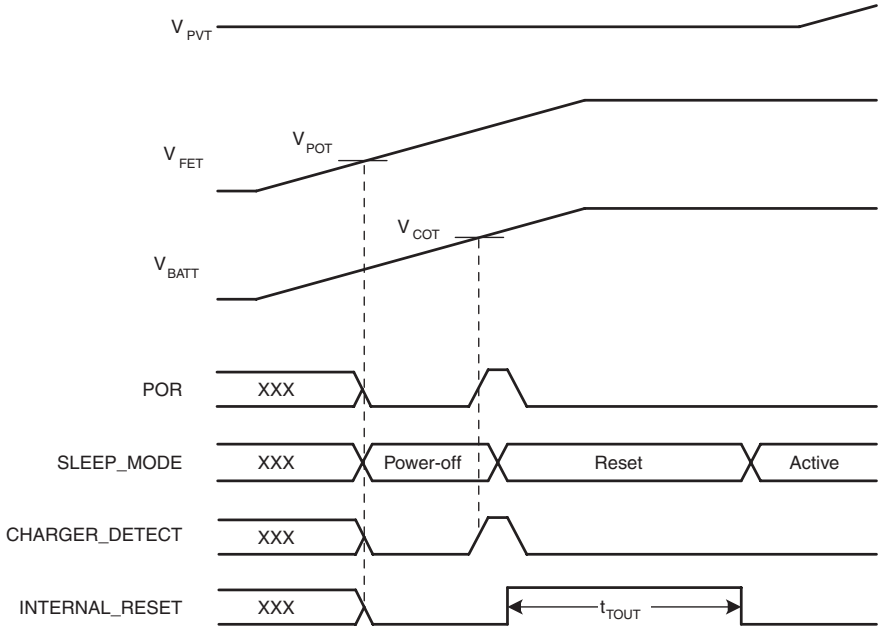


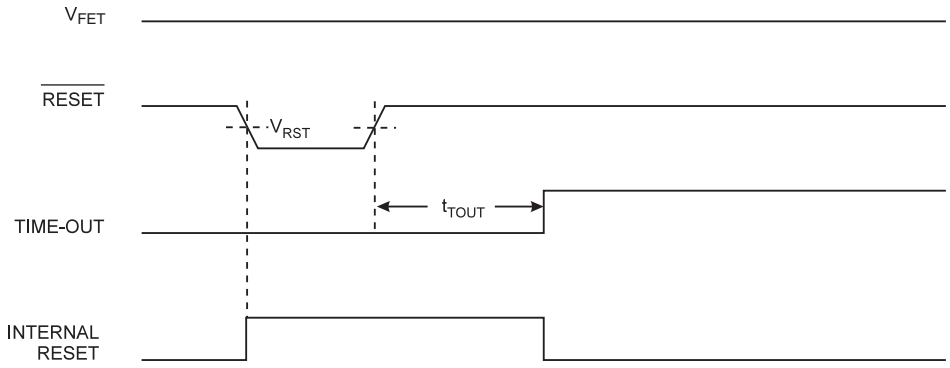
Figure 8-3. Powering up from charger side.



8.4 External Reset

An External Reset is generated by a low level on the $\overline{\text{RESET}}$ pin. Reset pulses longer than the minimum pulse width (see Table 8-1) 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.

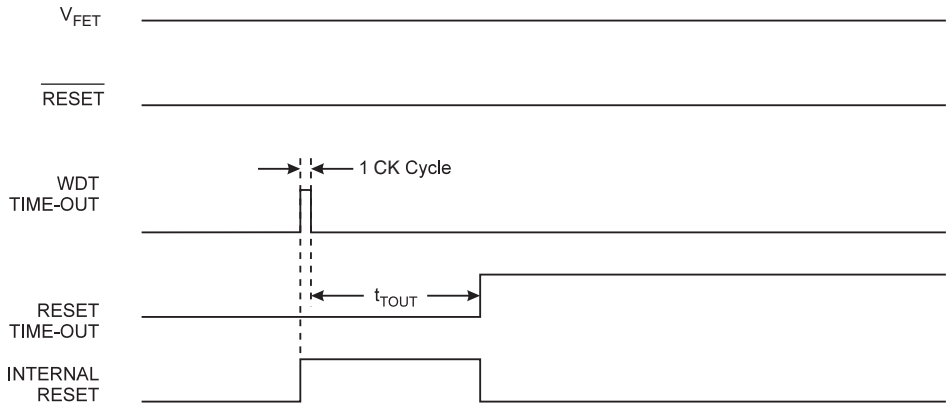
Figure 8-4. External Reset During Operation



8.5 Watchdog Reset

When the Watchdog times out, it will generate a short reset pulse of one CK cycle duration. On the falling edge of this pulse, the delay timer starts counting the Time-out period t_{TOUT}. Refer to page 42 for details on operation of the Watchdog Timer.

Figure 8-5. Watchdog Reset During Operation



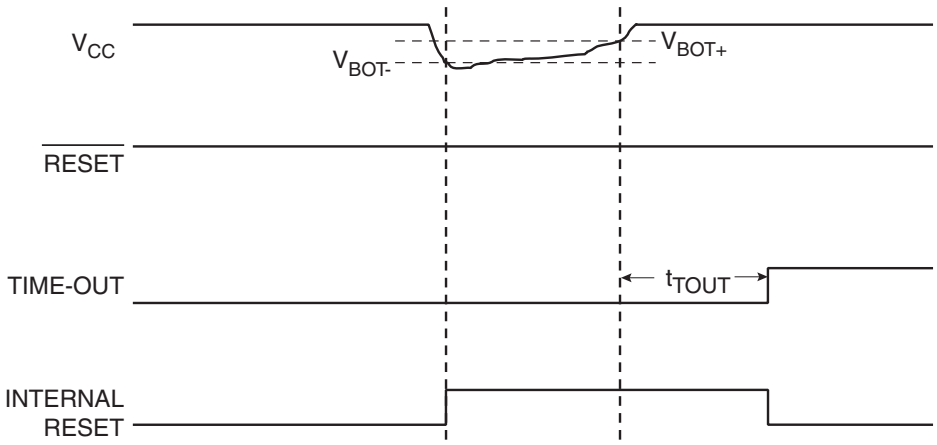
8.6 Brown-out Detection

ATmega406 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 V_{BOT} = 2.7V. 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 $V_{BOT-} = V_{BOT} - V_{HYST}/2$.

The BOD is automatically enabled in all modes of operation, except in Power-off mode.

When the BOD is enabled, and V_{CC} decreases to a value below the trigger level (V_{BOT-} in Figure 8-6), the Brown-out Reset is immediately activated. When V_{CC} increases above the trigger level (V_{BOT+} in Figure 8-6), the delay counter starts the MCU after the Time-out period t_{TOUT} has expired.

Figure 8-6. Brown-out Reset During Operation



8.7 MCU Status Register – MCUSR

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

Bit	7	6	5	4	3	2	1	0	
	–	–	–	JTRF	WDRF	BODRF	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					

- **Bits 7:5 – Res: Reserved Bits**

These bits are reserved bits in the ATmega406, and will always read as zero.

- **Bit 4 – JTRF: JTAG Reset Flag**

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

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

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

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

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

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

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

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

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

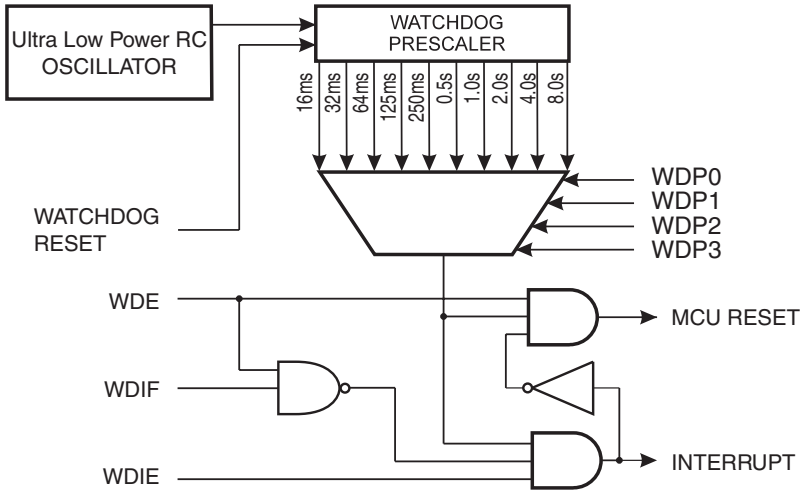
To make use of the Reset flags to identify a reset condition, the user should read and then reset the MCUSR as early as possible in the program. If the register is cleared before another reset occurs, the source of the reset can be found by examining the reset flags.

8.8 Watchdog Timer

ATmega406 has an Enhanced Watchdog Timer (WDT). The main features are:

- **Clocked from separate On-chip Oscillator**
- **3 Operating modes**
  - **Interrupt**
  - **System Reset**
  - **Interrupt and System Reset**
- **Selectable Time-out period from 16ms to 8s**
- **Possible Hardware fuse Watchdog always on (WDTON) for fail-safe mode**

Figure 8-7. Watchdog Timer



The Watchdog Timer (WDT) is a timer counting cycles of the Ultra Low Power RC Oscillator that runs at 128 kHz. The WDT gives an interrupt or a system reset when the counter reaches a given time-out value. In normal operation mode, it is required that the system uses the WDR - Watchdog Timer Reset - instruction to restart the counter before the time-out value is reached. If the system doesn't restart the counter, an interrupt or system reset will be issued.

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

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

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

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

Assembly Code Example[1]

```
WDT_off:
  ; Turn off global interrupt
  cli
  ; Reset Watchdog Timer
  wdr
  ; Clear WDRF in MCUSR
  in    r16, MCUSR
  andi  r16, (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. See "About Code Examples" on page 7.

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. "About Code Examples" on page 7.

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.

8.8.1 Watchdog Timer Control Register - WDTCSR

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

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

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

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

When this bit is written to one and the I-bit in the Status Register is set, the Watchdog Interrupt is enabled. If WDE is cleared in combination with this setting, the Watchdog Timer is in Interrupt Mode, and the corresponding interrupt is executed if time-out in the Watchdog Timer occurs.

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

Table 8-2. Watchdog Timer Configuration

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

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

The WDP3:0 bits determine the Watchdog Timer prescaling when the Watchdog Timer is enabled. The different prescaling values and their corresponding Timeout Periods are shown in Table 8-3.

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

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

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

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

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

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

The WDP3:0 bits determine the Watchdog Timer prescaling when the Watchdog Timer is enabled. The different prescaling values and their corresponding Timeout Periods are shown in Table 8-3..

Table 8-3. Watchdog Timer Prescale Select

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 5.0V
0	0	0	0	2K cycles	16 ms
0	0	0	1	4K cycles	32 ms
0	0	1	0	8K cycles	64 ms
0	0	1	1	16K cycles	0.125 s
0	1	0	0	32K cycles	0.25 s
0	1	0	1	64K cycles	0.5 s
0	1	1	0	128K cycles	1.0 s
0	1	1	1	256K cycles	2.0 s
1	0	0	0	512K cycles	4.0 s
1	0	0	1	1024K cycles	8.0 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		

9. Wake-up Timer

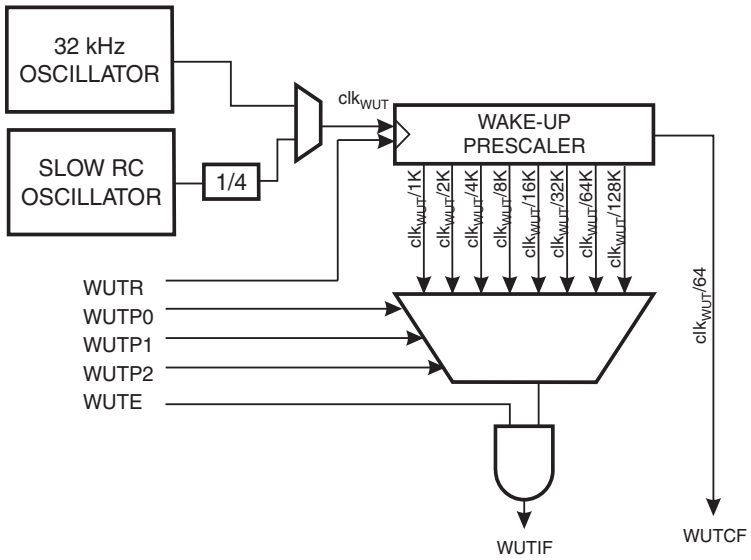
The following section describes the Wake-up Timer in the ATmega406.

- **One Wake-up Timer Interrupt**
- **8 Selectable Time-out Periods**
- **Separate Wake-up Timer Calibration Flag**
- **Separate Clock Source**

9.1 Overview

The Wake-up Timer is clocked either from the Slow RC Oscillator or from the external 32 kHz crystal oscillator. See ”Run-Time Clock Source Select” on page 29 for details. By controlling the Wake-up Timer prescaler, the Wake-up interval can be adjusted from 31.25 ms to 4 s. Eight different clock cycle periods can be selected to determine the Time-out period.

Figure 9-1. Wake-up Timer



9.1.1 Wake-up Timer Control and Status Register – WUTCSR

Bit	7	6	5	4	3	2	1	0	
	WUTIF	WUTIE	WUTCF	WUTR	WUTE	WUTP2	WUTP1	WUTP0	WUTCSR
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 – WUTIF: Wake-up Timer Interrupt Flag**

The WUTIF bit is set (one) when an overflow occurs in the Wake-up Timer. WUTIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, WUTIF is cleared by writing a logic one to the flag. When the SREG I-bit, WUTIE (Wake-up Timer Interrupt Enable), and WUTIF are set (one), the Wake-up Timer interrupt is executed.

- **Bit 6 – WUTIE: Wake-up Timer Interrupt Enable**

When the WUTIE bit and the I-bit in the Status Register are set (one), the Wake-up Timer interrupt is enabled. The corresponding interrupt is executed if a Wake-up Timer overflow occurs, i.e., when the WUTIF bit is set.

- **Bit 5 – WUTCF: Wake-up Timer Calibration Flag**

The WUTCF bit is set every 1.95 ms (256 Slow RC OScillator clocks or 64 32 kHz Crystal Oscillator clocks). WUTCF is cleared by writing a logic one to the flag. WUTCF can be used to calibrate the Fast RC Oscillator to the 32 kHz oscillator or the Slow RC Oscillator.

- **Bit 4 – WUTR: Wake-up Timer Reset**

When WUTR bit is written to one, the Wake-up Timer is reset, and starts counting from zero. The WUTR bit is always read as zero.

- **Bit 3 – WUTE: Wake-up Timer Enable**

When the WUTE bit is set (one) the Wake-up Timer is enabled, and if the WUTE is cleared (zero) the Wake-up Timer function is disabled. It is recommended to reset the Wake-up Timer when enabling it, by simultaneously setting the WUTR and WUTE bits.

- **Bits 2:0 – WUTP2, WUTP1, WUTP0: Wake-up Timer Prescaler 2, 1, and 0**

The WUTP2, WUTP1 and WUTP0 bits determine the Wake-up Timer prescaling when the Wake-up Timer is enabled. The different prescaling values and their corresponding time-out periods are shown in Table 9-1. The Wake-up Timer should always be reset when changing these bits.

Table 9-1. Wake-up Timer Prescale Select

WUTP2	WUTP1	WUTP0	Number of Slow RC Oscillator Cycles	Number of 32kHz Crystal Oscillator Cycles	Typical Time-out
0	0	0	4K(4096)	1K(1024)	31.25 ms
0	0	1	8K(8192)	2K(2048)	62.5 ms
0	1	0	16K(16384)	4K(4096)	125 ms
0	1	1	32K(32768)	8K(8192)	250 ms
1	0	0	64K(65536)	16K(16384)	0.5 s
1	0	1	128K(131072)	32K(32768)	1 s
1	1	0	256K(262144)	64K(65536)	2 s
1	1	1	512K(524288)	128K(131072)	4 s

10. Interrupts

This section describes the specifics of the interrupt handling as performed in ATmega406. For a general explanation of the AVR interrupt handling, refer to ”Reset and Interrupt Handling” on page 14.

10.1 Interrupt Vectors in ATmega406

Table 10-1. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
1	0x0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	0x0002	BPINT	Battery Protection Interrupt
3	0x0004	INT0	External Interrupt Request 0
4	0x0006	INT1	External Interrupt Request 1
5	0x0008	INT2	External Interrupt Request 2
6	0x000A	INT3	External Interrupt Request 3
7	0x000C	PCINT0	Pin Change Interrupt 0
8	0x000E	PCINT1	Pin Change Interrupt 1
9	0x0010	WDT	Watchdog Time-out Interrupt
10	0x0012	WAKE_UP	Wake-up Timer Overflow
11	0x0014	TIMER1 COMP	Timer 1 Compare Match
12	0x0016	TIMER1 OVF	Timer 1 Overflow
13	0x0018	TIMER0 COMPA	Timer 0 Compare Match A
14	0x001A	TIMER0 COMPB	Timer 0 Compare Match B
15	0x001C	TIMER0 OVF	Timer 0 Overflow
16	0x001E	TWI BUS C/D	Two-wire Bus Connect/Disconnect
17	0x0020	TWI	Two-wire Serial Interface
18	0x0022	VADC	Voltage ADC Conversion Complete
19	0x0024	CCADC CONV	CC-ADC Instantaneous Current Conversion Complete
20	0x0026	CCADC REG CUR	CC-ADC Regular Current
21	0x0028	CCADC ACC	CC-ADC Accumulate Current Conversion Complete
22	0x002A	EE READY	EEPROM Ready
23	0x002C	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 ”Boot Loader Support – Read-While-Write Self-Programming” on page 170.
2. When the IVSEL bit in MCUCR is set, Interrupt Vectors will be moved to the start of the Boot Flash Section. The address of each Interrupt Vector will then be the address in this table added to the start address of the Boot Flash Section.

Table 10-2 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 10-2. Reset and Interrupt Vectors Placement[1]

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

Note: 1. The Boot Reset Address is shown in Table 26-7 on page 183. For the BOOTRST Fuse "1" means unprogrammed while "0" means programmed.

The most typical and general program setup for the Reset and Interrupt Vector Addresses in ATmega406 is:

```
Address   Labels   Code                        Comments
0x0000             jmp       RESET             ; Reset Handler
0x0002             jmp       BPINT             ; Battery Protection Interrupt Handler
0x0004             jmp       EXT_INT0          ; External Interrupt Request 0 Handler
0x0006             jmp       EXT_INT1          ; External Interrupt Request 1 Handler
0x0008             jmp       EXT_INT2          ; External Interrupt Request 2 Handler
0x000A             jmp       EXT_INT3          ; External Interrupt Request 3 Handler
0x000C             jmp       PCINT0            ; Pin Change Interrupt 0 Handler
0x000E             jmp       PCINT1            ; Pin Change Interrupt 1 Handler
0x0010             jmp       WDT               ; Watchdog Time-out Interrupt
0x0012             jmp       WAKE_UP           ; Wake-up Timer Overflow
0x0014             jmp       TIM1_COMP         ; Timer1 Compare Handler
0x0016             jmp       TIM1_OVF          ; Timer1 Overflow Handler
0X0018             jmp       TIM0_COMPA        ; Timer0 CompareA Handler
0x001A             jmp       TIM0_COMPB        ; Timer0 CompareB Handler
0x001C             jmp       TIM0_OVF          ; Timer0 Overflow Handler
0x001E             jmp       TWI_BUS_CD        ; Two-wire Bus Connect/Disconnect Handler
0x0020             jmp       TWI               ; Two-wire Serial Interface Handler
0x0022             jmp       VADC              ; Voltage ADC Conversion Complete Handler
0x0024             jmp       CCADC_CONV        ; CC-ADC Instantaneous Current Conversion Complete Handler
0x0026             jmp       CCADC_REC_CUR     ; CC-ADC Regular Current Handler
0x0028             jmp       CCADC_ACC         ; CC-ADC Accumulate Current Conversion Complete Handler
0x002A             jmp       EE_RDY            ; EEPROM Ready Handler
0x002C             jmp       SPM_RDY           ; Store Program Memory Ready Handler
;
0x002E    RESET:   ldi       r16, high(RAMEND) ; Main program start
0x002F             out       SPH,r16           ; Set Stack Pointer to top of RAM
0x0030             ldi       r16, low(RAMEND)
0x0031             out       SPL,r16
0x0032             sei                         ; Enable interrupts
0x0033             <instr>   xxx
0x0034    ...      ...       ...
;
```

When the BOOTRST Fuse is unprogrammed, the Boot section size set to 2K bytes and the IVSEL bit in the MCUCR Register is set before any interrupts are enabled, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address  Labels   Code                    Comments
0x0000   RESET:   ldi     r16,high(RAMEND); Main program start
0x0001            out     SPH,r16         ; Set Stack Pointer to top of RAM
0x0002            ldi     r16,low(RAMEND)
0x0003            out     SPL,r16
0x0004            sei                     ; Enable interrupts
0x0005            <instr>  xxx
;
.org 0x4C02
0x4C02            jmp     BPINT           ; Battery Protection Interrupt Handler
0x4C04            jmp     EXT_INT0        ; External Interrupt Request 0 Handler
...               ...     ...             ;
0x4C2C            jmp     SPM_RDY         ; Store Program Memory Ready Handler
```

When the BOOTRST Fuse is programmed and the Boot section size set to 2K bytes, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address  Labels   Code                    Comments
.org 0x0002
0x0002            jmp     BPINT           ; Battery Protection Interrupt Handler
0x0004            jmp     EXT_INT0        ; External Interrupt Request 0 Handler
...               ...     ...             ;
0x002C      jmp     SPM_RDY         ; Store Program Memory Ready Handler
;
.org 0x4C00
0x4C00   RESET:   ldi     r16,high(RAMEND); Main program start
0x4C01            out     SPH,r16         ; Set Stack Pointer to top of RAM
0x4C02            ldi     r16,low(RAMEND)
0x4C03            out     SPL,r16
0x4C04            sei                     ; Enable interrupts
0x4C05            <instr>  xxx
```

When the BOOTRST Fuse is programmed, the Boot section size set to 2K bytes and the IVSEL bit in the MCUCR Register is set before any interrupts are enabled, the most typical and general program setup for the Reset and Interrupt Vector Addresses is:

```
Address  Labels Code                  Comments
;
.org 0x4C00
0x4C00            jmp     RESET           ; Reset handler
0x4C02            jmp     BPINT           ; Battery Protection Interrupt Handler
0x4C04            jmp     EXT_INT0        ; External Interrupt Request 0 Handler
...               ...     ...             ;
0x4C2C            jmp     SPM_RDY         ; Store Program Memory Ready Handler
;
0x4C2E   RESET:   ldi     r16,high(RAMEND); Main program start
```

```
0x4C2F          out     SPH,r16         ; Set Stack Pointer to top of RAM
0x4C30          ldi     r16,low(RAMEND)
0x4C31          out     SPL,r16
0x4C32          sei                     ; Enable interrupts
0x4C33          <instr>  xxx
```

10.1.1 Moving Interrupts Between Application and Boot Space

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

10.1.2 MCU Control Register – MCUCR

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

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

When the IVSEL bit is cleared (zero), the Interrupt Vectors are placed at the start of the Flash memory. When this bit is set (one), the Interrupt Vectors are moved to the beginning of the Boot Loader section of the Flash. The actual address of the start of the Boot Flash Section is determined by the BOOTSZ Fuses. Refer to the section ”Boot Loader Support – Read-While-Write Self-Programming” on page 170 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 ”Boot Loader Support – Read-While-Write Self-Programming” on page 170 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);
}
```

11. External Interrupts

11.1 Overview

The External Interrupts are triggered by the INT3:0 pins. Observe that, if enabled, the interrupts will trigger even if the INT3:0 pins are configured as outputs. This feature provides a way of generating a software interrupt. 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 Register – EICRA. When the external interrupt is enabled and is configured as level triggered, the interrupt will trigger as long as the pin is held low. Interrupts 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 mode, the changed level must be held for some time to wake up the MCU. This makes the MCU less sensitive to noise. The changed level is sampled twice by the Slow RC Oscillator clock. The period of the Slow RC Oscillator is 7.8 µs (nominal) at 25°C. The MCU will wake up if the input has the required level during this sampling or if it is held until the end of the start-up time. The start-up time is defined by the SUT fuses as described in ”System Clock and Clock Options” on page 25. If the level is sampled twice by the Slow RC Oscillator clock but disappears before the end of the start-up time, the MCU will still wake up, but no interrupt will be generated. The required level must be held long enough for the MCU to complete the wake up to trigger the level interrupt.

11.1.1 External Interrupt Control Register A – EICRA

Bit	7	6	5	4	3	2	1	0	
	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 - ISC01, ISC00: External Interrupt 3 - 0 Sense Control Bits**

The External Interrupts 3 - 0 are activated by the external pins INT3:0 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 11-1. Edges on INT3:INT0 are registered asynchronously. Pulses on INT3:0 pins wider than the minimum pulse width given in Table 11-2 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 11-1. Interrupt Sense Control

ISCn1	ISCn0	Description[1]
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 of INTn generates an interrupt request.
1	1	The rising edge of INTn generates an interrupt request.

Note: 1. n = 3, 2, 1, or 0.
When changing the ISCn1/ISCn0 bits, the interrupt must be disabled by clearing its Interrupt Enable bit in the EIMSK Register. Otherwise an interrupt can occur when the bits are changed.

Table 11-2. Asynchronous External Interrupt Characteristics

Symbol	Parameter	Condition	Min	Typ	Max	Units
t_{INT}	Minimum pulse width for asynchronous external interrupt			50		ns

11.1.2 External Interrupt Mask Register – EIMSK

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	INT3	INT2	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:4 – RES: Reserved Bits**

These bits are reserved bits ins the ATmega406, and will always read as zero.

- **Bits 3:0 – INT3 - INT0: External Interrupt Request 3 - 0 Enable**

When an INT3 – INT0 bit is written to one and the I-bit in the Status Register (SREG) is set (one), the corresponding external pin interrupt is enabled. The Interrupt Sense Control bits in the External Interrupt Control Register – EICRA – defines whether the external interrupt is activated on rising or falling edge or level sensed. Activity on any of these pins will trigger an interrupt request even if the pin is enabled as an output. This provides a way of generating a software interrupt.

11.1.3 External Interrupt Flag Register – EIFR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	INTF3	INTF2	INTF1	INTF0	EIFR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:4 – RES: Reserved Bits**

These bits are reserved bits ins the ATmega406, and will always read as zero.

- **Bits 3:0 – INTF3 - INTF0: External Interrupt Flags 3 - 0**

When an edge or logic change on the INT3:0 pin triggers an interrupt request, INTF3:0 becomes set (one). If the I-bit in SREG and the corresponding interrupt enable bit, INT3:0 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 INT3:0 are configured as level interrupt. Note that when entering sleep mode with the INT3:0 interrupts disabled, the input buffers on these pins will be disabled. This may cause a logic change in internal signals which will set the INTF3:0 flags. See ”Digital Input Enable and Sleep Modes” on page 62 for more information.

11.1.4 Pin Change Interrupt Control Register - PCICR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	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 7:2 - Res: Reserved Bits**

These bits are reserved bits in the ATmega406, and will always read as zero.

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

When the PCIE1 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), pin change interrupt 1 is enabled. Any change on any enabled PCINT15:8 pin will cause an interrupt. The corresponding interrupt of Pin Change Interrupt Request is executed from the PCI1 Interrupt Vector. PCINT15:8 pins are enabled individually by the PCMSK1 Register.

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

When the PCIE0 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), pin change interrupt 0 is enabled. Any change on any enabled PCINT7:0 pin will cause an interrupt. The corresponding interrupt of Pin Change Interrupt Request is executed from the PCI0 Interrupt Vector. PCINT7:0 pins are enabled individually by the PCMSK0 Register.

11.1.5 Pin Change Interrupt Flag Register - PCIFR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	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 7:2 - Res: Reserved Bits**

These bits are reserved bits in the ATmega406, and will always read as zero.

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

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

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

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

11.1.6 Pin Change Mask Register 1 – PCMSK1

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

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

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

11.1.7 Pin Change Mask Register 0 – PCMSK0

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

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

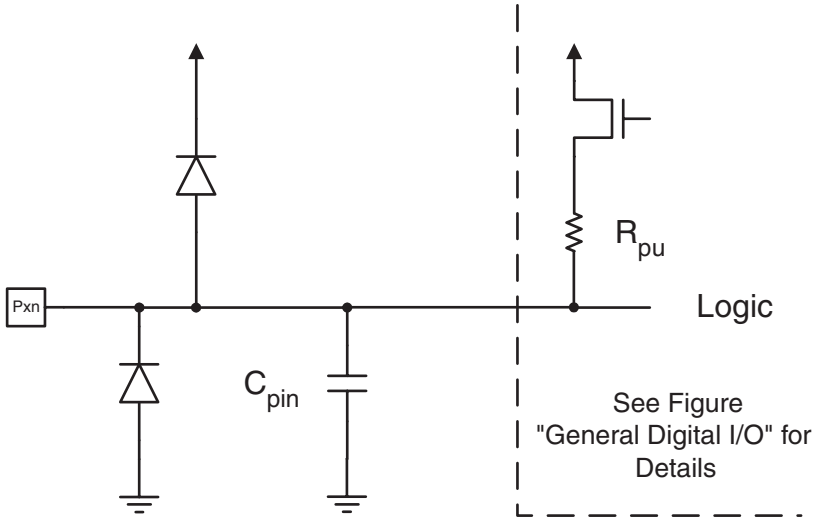
Each PCINT7:0 bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT7:0 is set and the PCIE0 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT7:0 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

12. Low Voltage I/O-Ports

12.1 Introduction

All low voltage 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). All low voltage port pins have individually selectable pull-up resistors with a supply-voltage invariant resistance. All I/O pins have protection diodes to both V_{REG} and Ground as indicated in Figure 12-1. Refer to "Electrical Characteristics" on page 211 for a complete list of parameters.

Figure 12-1. Low Voltage 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 71.

Three I/O memory address locations are allocated for each low voltage 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 low voltage pins in all ports when set.

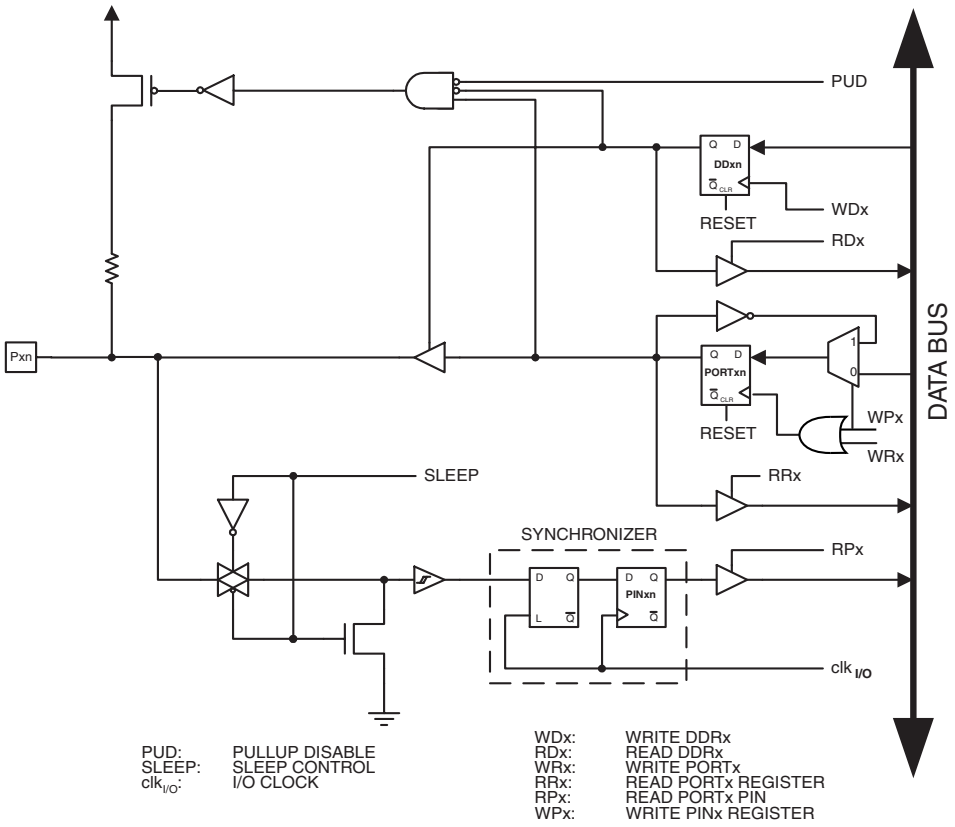
Using the I/O port as General Digital I/O is described in "Low Voltage Ports as General Digital I/O" on page 59. Many low voltage port pins are multiplexed with alternate functions for the peripheral features on the device. How each alternate function interferes with the port pin is described in "Alternate Port Functions" on page 63. 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 Low Voltage Ports as General Digital I/O

The low voltage 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 Low Voltage 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 71, 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) must occur. Normally, the pull-up enabled state is fully acceptable, as a high-impedant environment will not notice the difference between a strong high driver and a pull-up. If this is not the case, the PUD bit in the MCUCR Register can be set to disable all pull-ups in all ports.

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

Table 12-1 summarizes the control signals for the pin value.

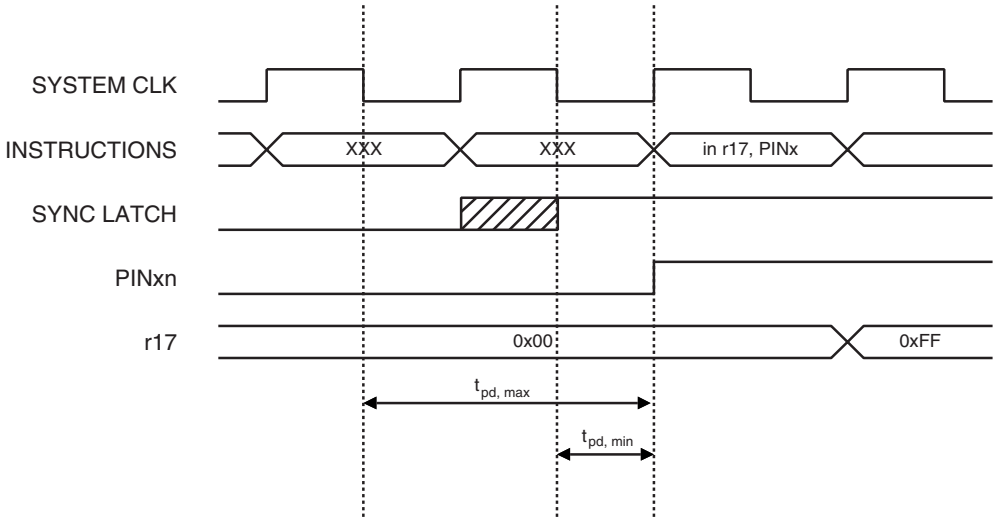
Table 12-1. Port Pin Configurations

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.

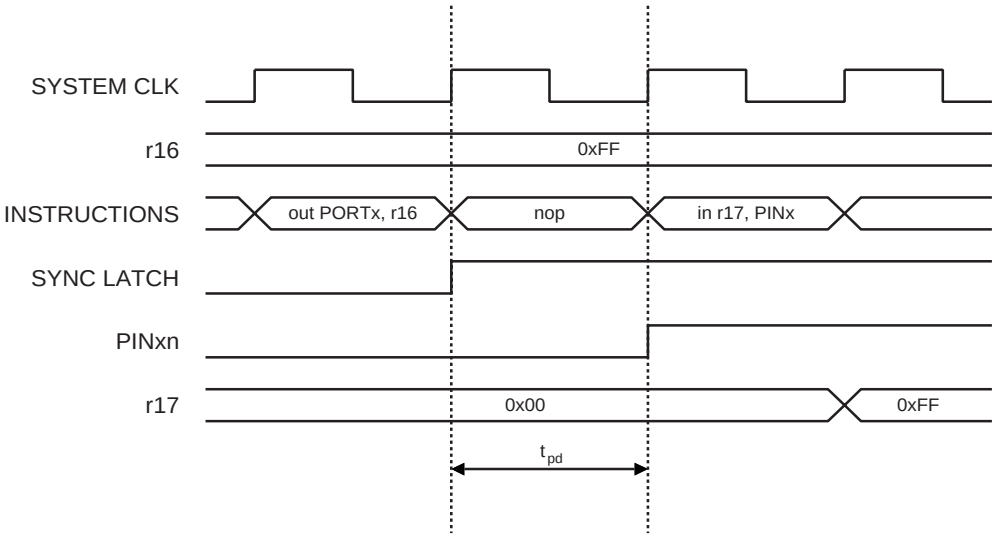
Figure 12-3. Synchronization when Reading an Externally Applied Pin value



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*/
_NOP();
/* 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 and Power-save mode to avoid high power consumption if some input signals are left floating, or have an analog signal level close to $V_{REG}/2$.

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 63.

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

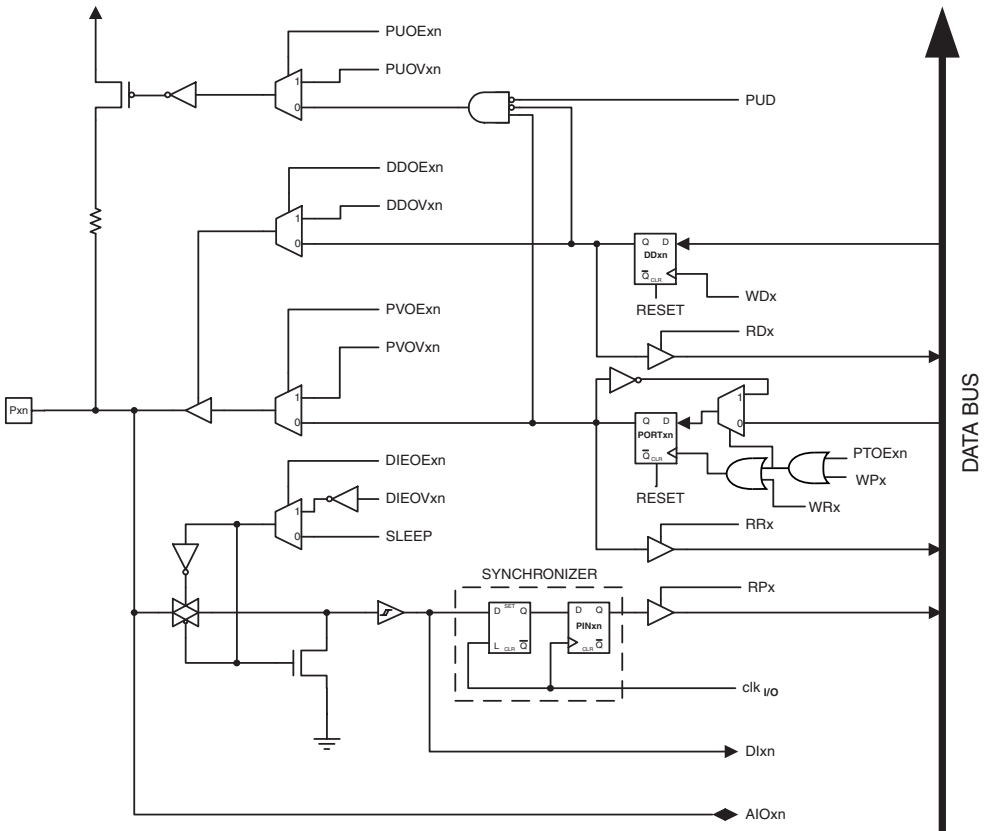
If some pins are unused, it is recommended to ensure that these pins have a defined level. Even though most of the digital inputs are disabled in the deep sleep modes as described above, floating inputs should be avoided to reduce current consumption in all other modes where the digital inputs are enabled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

Many low voltage 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 MCU Control Register – MCUCR

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

- **Bit 4 – PUD: Pull-up Disable**

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

12.3.2 Alternate Functions of Port A

The Port A has an alternate function as input pins to the Voltage ADC.

Table 12-3. Port A Pins Alternate Functions

Port Pin	Alternate Function
PA7	INT3 (External Interrupt 3) PCINT7 (Pin Change Interrupt 7)
PA6	INT2 (External Interrupt 2) PCINT6 (Pin Change Interrupt 6)
PA5	INT1 (External Interrupt 1) PCINT5 (Pin Change Interrupt 5)
PA4	ADC4 (ADC Input Channel 4) INT0 (External Interrupt 0) PCINT4 (Pin Change Interrupt 4)
PA3	ADC3 (ADC Input Channel 3) PCINT3 (Pin Change Interrupt 3)
PA2	ADC2 (ADC Input Channel 2) PCINT2 (Pin Change Interrupt 2)
PA1	ADC1 (ADC Input Channel 1) PCINT1 (Pin Change Interrupt 1)
PA0	ADC0 (ADC Input Channel 0) PCINT0 (Pin Change Interrupt 0)

The alternate pin configuration is as follows:

- **ADC4/INT3:0/PCINT7:4 – Port A, Bit 7:4**

Analog to Digital Converter, Channel 4.

INT3 - INT0, External Interrupt Sources 3:0. The PA7:4 pins can serve as external interrupt sources to the MCU.

PCINT7 - PCINT4, Pin Change Interrupt Sources 7:4. The PA7:4 pins can serve as external interrupt sources to the MCU.

- **ADC3:0/PCINT3:0 – Port A, Bit 3:0**

Analog to Digital Converter, Channels 3:0.

PCINT3 - PCINT0, Pin Change Interrupt Sources 3:0. The PA3:0 pins can serve as external interrupt sources to the MCU.

Table 12-4 and Table 12-5 relates the alternate functions of Port A to the overriding signals shown in Figure 12-5 on page 64.

Table 12-4. Overriding Signals for Alternate Functions in PA7:PA4

Signal Name	PA7/INT3/ PCINT7	PA6/INT2/ PCINT6	PA5/INT1/ PCINT5	PA4/ADC4 INT0/PCINT4
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
PTOE	–	–	–	–
DIEOE	INT3 ENABLE	INT2 ENABLE	INT1 ENABLE	INT0 ENABLE
DIEOV	INT3 ENABLE	INT2 ENABLE	INT1 ENABLE	INT0 ENABLE
DI	INT3 INPUT/ PCINT7 INPUT	INT2 INPUT/ PCINT6 INPUT	INT1 INPUT/ PCINT5 INPUT	INT0 INPUT/ PCINT4 INPUT
AIO	–	–	–	ADC4 INPUT

Table 12-5. Overriding Signals for Alternate Functions in PA3:PA0

Signal Name	PA3/ADC3/ PCINT3	PA2/ADC2/ PCINT2	PA1/ADC1/ PCINT1	PA0/ADC0/ PCINT0
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
PTOE	–	–	–	–
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	PCINT3 INPUT	PCINT2 INPUT	PCINT1 INPUT	PCINT0 INPUT
AIO	ADC3 INPUT	ADC2 INPUT	ADC1 INPUT	ADC0 INPUT

12.3.3 Alternate Functions of Port B

The Port B pins with alternate functions are shown in Table 12-6.

Table 12-6. Port B Pins Alternate Functions

Port Pin	Alternate Functions
PB7	OC0B (Output Compare and PWM Output B for Timer/Counter0) PCINT15 (Pin Change Interrupt 15)
PB6	OC0A (Output Compare and PWM Output A for Timer/Counter0) PCINT14 (Pin Change Interrupt 14)
PB5	PCINT13 (Pin Change Interrupt 13)
PB4	PCINT12 (Pin Change Interrupt 12)
PB3	TCK (JTAG Test Clock) PCINT11 (Pin Change Interrupt 11)
PB2	TMS (JTAG Test Mode Select) PCINT10 (Pin Change Interrupt 10)
PB1	TDI (JTAG Test Data Input/) PCINT9 (Pin Change Interrupt 9)
PB0	TDO (JTAG Test Data Output) PCINT8 (Pin Change Interrupt 8)

The alternate pin configuration is as follows:

- **OC0B/PCINT15 – Port B, Bit 7**

OC0B, Output Compare Match B 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 OC0B pin is also the output pin for the PWM mode timer function.

PCINT15, Pin Change Interrupt Source 15. The PB7 pin can serve as external interrupt source to the MCU.

- **OC0A/PCINT14 – Port B, Bit 6**

OC0A, Output Compare Match A output: The PB6 pin can serve as an external output for the Timer/Counter0 Output Compare. The pin has to be configured as an output (DDB6 set (one)) to serve this function. The OC0A pin is also the output pin for the PWM mode timer function.

PCINT14, Pin Change Interrupt Source 14. The PB6 pin can serve as external interrupt source to the MCU.

- **PCINT13:12 – Port B, Bit 5:4**

PCINT13 - PCINT12, Pin Change Interrupt Source 13:12. The PB5:4 pinS can serve as external interrupt sources to the MCU.

- **TCK/PCINT11 – Port B, Bit 3**

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

PCINT11, Pin Change Interrupt Source 11. The PB3 pin can serve as external interrupt source to the MCU.

- **TMS/PCINT10 – Port B, Bit 2**

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

PCINT10, Pin Change Interrupt Source 10. The PB2 pin can serve as external interrupt source to the MCU.

- **TDI/PCINT9 – Port B, Bit 1**

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

PCINT9, Pin Change Interrupt Source 9. The PB1 pin can serve as external interrupt source to the MCU.

- **TDO/PCINT8 – Port B, Bit 0**

TDO, JTAG Test Data Output: Serial output data from Instruction Register or Data Register. When the JTAG Interface is enabled, this pin can not be used as an I/O pin.

PCINT8, Pin Change Interrupt Source 8. The PB0 pin can serve as external interrupt source to the MCU.

Table 12-7 and Table 12-8 relate the alternate functions of Port B to the overriding signals shown in Figure 12-5 on page 64.

Table 12-7. Overriding Signals for Alternate Functions in PB7:PB4

Signal Name	PB7/OCOB/ PCINT15	PB6/OCOA/ PCINT14	PB5/ PCINT13	PB4/ PCINT12
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	OC0B Enable	OC0A Enable		0
PVOV	OC0B	OC0A		0
PTOE	–	–	–	–
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	PCINT15 INPUT	PCINT14 INPUT	PCINT13 INPUT	PCINT12 INPUT
AIO	–	–	–	–

.

Table 12-8. Overriding Signals for Alternate Functions in PB3:PB0

Signal Name	PB3/TCK/ PCINT11	PB2/TMS/ PCINT10	PB1/TDI/ PCINT9	PB0/TDO/ PCINT8
PUOE	JTAGEN	JTAGEN	JTAGEN	JTAGEN
PUOV	1	1	1	0
DDOE	JTAGEN	JTAGEN	JTAGEN	JTAGEN
DDOV	0	0	0	SHIFT_IR + SHIFT_DR
PVOE	0	0	0	JTAGEN
PVOV	0	0	0	TDO
PTOE	–	–	–	–
DIEOE	JTAGEN	JTAGEN	JTAGEN	JTAGEN
DIEOV	0	0	0	0
DI	TCK/PCINT11 INPUT	TMS/PCINT10 INPUT	TDI/PCINT9 INPUT	PCINT8 INPUT
AIO	–	–	–	–

12.3.4 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
PD0	T0 (Timer/Counter0 Clock Input)

The alternate pin configuration is as follows:

- **T0 – Port B, Bit 0**

T0, Timer/Counter0 Counter Source.

Table 12-10 on page 71 relates the alternate functions of Port D to the overriding signals shown in Figure 12-5 on page 64.

Table 12-10. Overriding Signals for Alternate Functions in PD1:PD0

Signal Name	PD1	PD0/T0
PUOE	0	0
PUOV	0	0
DDOE	0	0
DDOV	0	0
PVOE		0
PVOV		0
PTOE	–	–
DIEOE	0	0
DIEOV	0	0
DI	–	T0 Input
AIO	–	–

12.4 Register Description for I/O-Ports

12.4.1 Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

12.4.2 Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

12.4.3 Port A Input Pins Address – PINA

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

12.4.4 Port B Data Register – PORTB

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

12.4.5 Port B Data Direction Register – DDRB

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

12.4.6 Port B Input Pins Address – PINB

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

12.4.7 Port D Data Register – PORTD

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	PORTD1	PORTD0	PORTD
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

12.4.8 Port D Data Direction Register – DDRD

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	DDD1	DDD0	DDRD
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

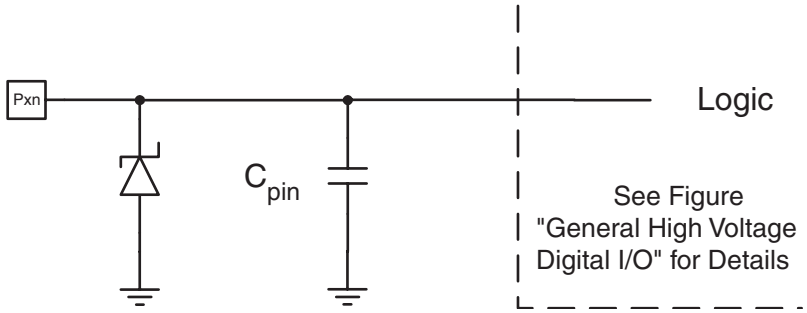
12.4.9 Port D Input Pins Address – PIND

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	N/A	N/A	

13. High Voltage I/O Ports

All high voltage AVR ports have true Read-Modify-Write functionality when used as general digital I/O ports. This means that the state of one port pin can be changed without unintentionally changing the state of any other pin with the SBI and CBI instructions. All high voltage I/O pins have protection Zener diodes to Ground as indicated in Figure 13-1. See ”Electrical Characteristics” on page 211 for a complete list of parameters.

Figure 13-1. High Voltage 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, PORTC3 for bit number three in Port C, here documented generally as PORTxn. The physical I/O Registers and bit locations are listed in ”Register Description for High Voltage Output Ports” on page 74.

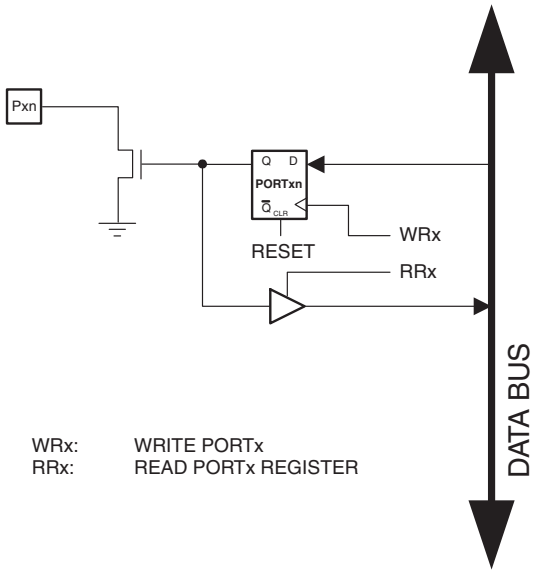
One I/O Memory address location is allocated for each high voltage port, the Data Register – PORTx. The Data Register is read/write.

Using the I/O port as General Digital Output is described in ”High Voltage Ports as General Digital Outputs” on page 73.

13.1 High Voltage Ports as General Digital Outputs

The high voltage ports are high voltage tolerant open collector output ports. Figure 13-2 shows a functional description of one output port pin, here generically called Pxn.

Figure 13-2. General High Voltage Digital I/O[1]



Note: 1. WRx and RRx are common to all pins within the same port.

13.2 Configuring the Pin

Each port pin has one register bit: PORTxn. As shown in ”Register Description for High Voltage Output Ports” on page 74, the PORTxn bits are accessed at the PORTx I/O address. If PORTxn is written logic one, the port pin is driven low (zero). If PORTxn is written logic zero, the port pin is tri-stated. The port pins are tri-stated when a reset condition becomes active, even if no clocks are running.

13.3 Register Description for High Voltage Output Ports

13.3.1 Port C Data Register – PORTC

Bit	7	6	5	4	3	2	1	0	
	–							PORTC0	PORTC
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14. 8-bit Timer/Counter0 with PWM

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. The main features are:

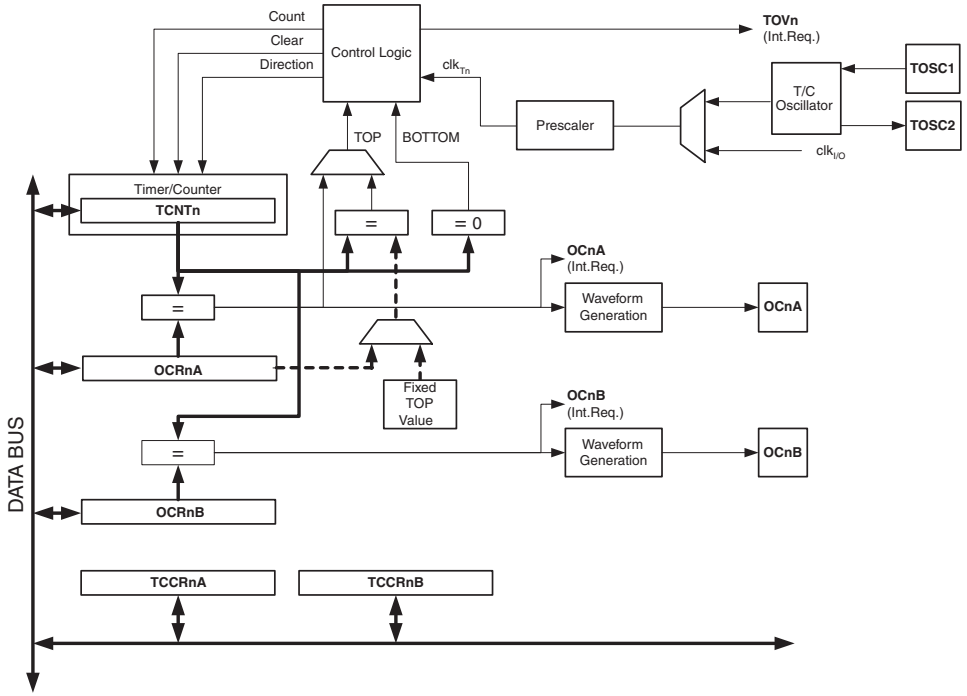
- **Two Independent Output Compare Units**
- **Double Buffered Output Compare Registers**
- **Clear Timer on Compare Match (Auto Reload)**
- **Glitch Free, Phase Correct Pulse Width Modulator (PWM)**
- **Variable PWM Period**
- **Frequency Generator**
- **Three Independent Interrupt Sources (TOV0, OCF0A, and OCF0B)**

14.1 Overview

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 14-1. For the actual placement of I/O pins, refer to ”Pinout ATmega406.” on page 2. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the ”8-bit Timer/Counter Register Description” on page 86.

The PRTIM0 bit in ”Power Reduction Register 0 - PRR0” on page 35 must be written to zero to enable Timer/Counter0 module.

Figure 14-1. 8-bit Timer/Counter Block Diagram



14.1.1 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 14-1 are also used extensively throughout the document.

Table 14-1. 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.

14.1.2 Registers

The Timer/Counter (TCNT0) and Output Compare Registers (OCR0A and OCR0B) are 8-bit registers. Interrupt request (abbreviated to Int.Req. in the figure) signals are all visible in the Timer Interrupt Flag Register (TIFR0). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK0). TIFR0 and TIMSK0 are not shown in the figure.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the T0 pin. The Clock Select logic block controls which clock source and edge the Timer/Counter uses to increment (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 Section “14.4.3” on page 78. 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.

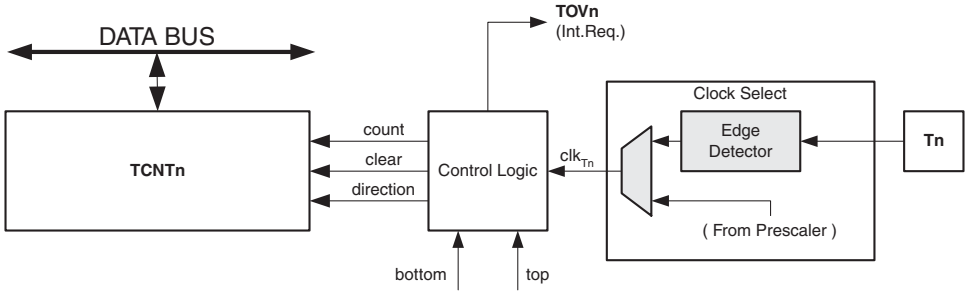
14.2 Timer/Counter Clock Sources

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 101.

14.3 Counter Unit

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Figure 14-2 shows a block diagram of the counter and its surroundings.

Figure 14-2. Counter Unit Block Diagram



Signal description (internal signals):

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

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

The counting sequence is determined by the setting of the WGM01 and WGM00 bits located in the Timer/Counter Control Register (TCCR0A) and the WGM02 bit located in the Timer/Counter Control Register B (TCCR0B). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare outputs OC0A and OC0B. For more details about advanced counting sequences and waveform generation, see ”Modes of Operation” on page 80.

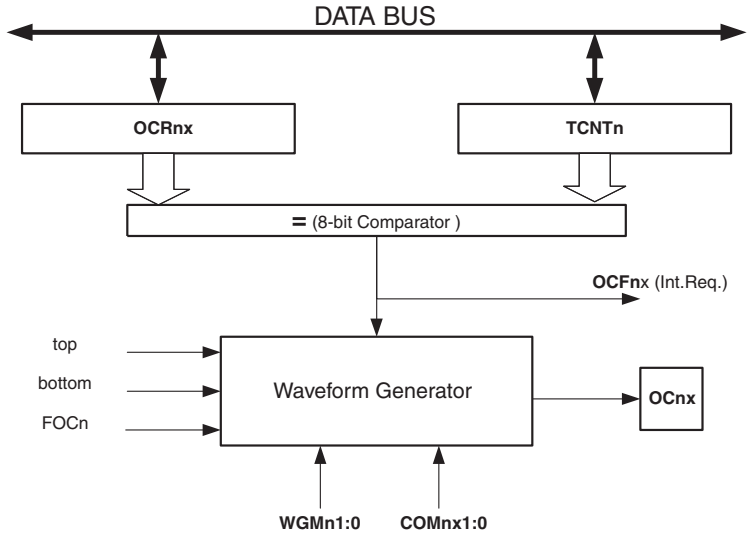
The Timer/Counter Overflow Flag (TOV0) is set according to the mode of operation selected by the WGM02:0 bits. TOV0 can be used for generating a CPU interrupt.

14.4 Output Compare Unit

The 8-bit comparator continuously compares TCNT0 with the Output Compare Registers (OCR0A and OCR0B). Whenever TCNT0 equals OCR0A or OCR0B, the comparator signals a match. A match will set the Output Compare Flag (OCF0A or OCF0B) at the next timer clock cycle. If the corresponding interrupt is enabled, the Output Compare Flag generates an Output Compare interrupt. The Output Compare Flag is automatically cleared when the interrupt is executed. Alternatively, the flag can be cleared by software by writing a logical one to its I/O bit location. The Waveform Generator uses the match signal to generate an output according to operating mode set by the WGM02:0 bits and Compare Output mode (COM0x1:0) bits. The max and bottom signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation (”Modes of Operation” on page 80).

Figure 14-3 shows a block diagram of the Output Compare unit.

Figure 14-3. 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.

14.4.1 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).

14.4.2 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.

14.4.3 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 downcounting.

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 Com-

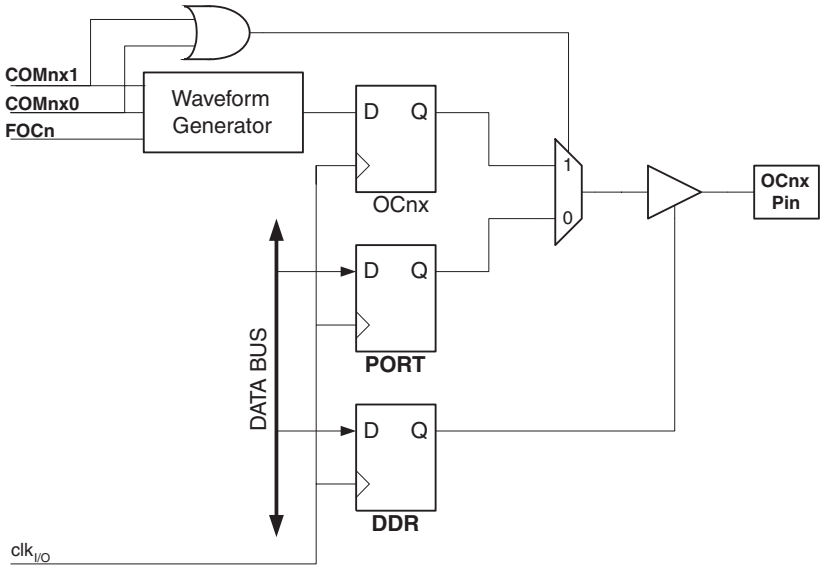
pare (FOC0x) strobe bits in Normal mode. The OC0x Registers keep their values even when changing between Waveform Generation modes.

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

14.5 Compare Match Output Unit

The Compare Output mode (COM0x1:0) bits have two functions. The Waveform Generator uses the COM0x1:0 bits for defining the Output Compare (OC0x) state at the next compare match. Also, the COM0x1:0 bits control the OC0x pin output source. Figure 14-4 shows a simplified schematic of the logic affected by the COM0x1:0 bit setting. The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O port control registers (DDR and PORT) that are affected by the COM0x1:0 bits are shown. When referring to the OC0x state, the reference is for the internal OC0x Register, not the OC0x pin. If a system reset occur, the OC0x Register is reset to “0”.

Figure 14-4. Compare Match Output Unit, Schematic



The general I/O port function is overridden by the Output Compare (OC0x) from the Waveform Generator if either of the COM0x1:0 bits are set. However, the OC0x pin direction (input or output) is still controlled by the Data Direction Register (DDR) for the port pin. The Data Direction Register bit for the OC0x pin (DDR_OC0x) must be set as output before the OC0x value is visible on the pin. The port override function is independent of the Waveform Generation mode.

The design of the Output Compare pin logic allows initialization of the OC0x state before the output is enabled. Note that some COM0x1:0 bit settings are reserved for certain modes of operation. See Section “14.8” on page 86.

14.5.1 Compare Output Mode and Waveform Generation

The Waveform Generator uses the COM0x1:0 bits differently in Normal, CTC, and PWM modes. For all modes, setting the COM0x1:0 = 0 tells the Waveform Generator that no action on the OC0x Register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 14-2 on page 86. For fast PWM mode, refer to Table 14-3 on page 86, and for phase correct PWM refer to Table 14-4 on page 87.

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.

14.6 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the Waveform Generation mode (WGM02:0) and Compare Output mode (COM0x1:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM0x1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM0x1:0 bits control whether the output should be set, cleared, or toggled at a compare match (See Section “14.5” on page 79.).

For detailed timing information refer to ”Timer/Counter Timing Diagrams” on page 84.

14.6.1 Normal Mode

The simplest mode of operation is the Normal mode (WGM02:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 8-bit value (TOP = 0xFF) and then restarts from the bottom (0x00). In normal operation the Timer/Counter Overflow Flag (TOV0) will be set in the same timer clock cycle as the TCNT0 becomes zero. The TOV0 Flag in this case behaves like a ninth bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV0 Flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime.

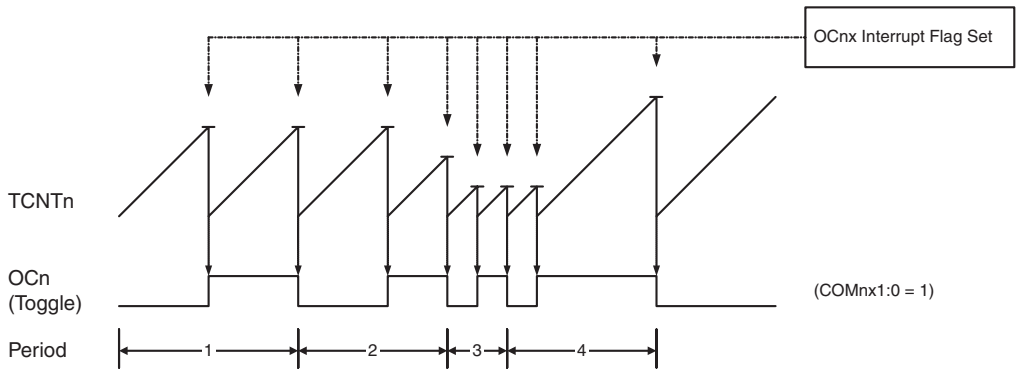
The Output Compare unit can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

14.6.2 Clear Timer on Compare Match (CTC) Mode

In Clear Timer on Compare or CTC mode (WGM02:0 = 2), the OCR0A Register is used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT0) matches the OCR0A. The OCR0A defines the top value for the counter, hence also its resolution. This mode allows greater control of the compare match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 14-5. The counter value (TCNT0) increases until a compare match occurs between TCNT0 and OCR0A, and then counter (TCNT0) is cleared.

Figure 14-5. CTC Mode, Timing Diagram



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

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

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

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.

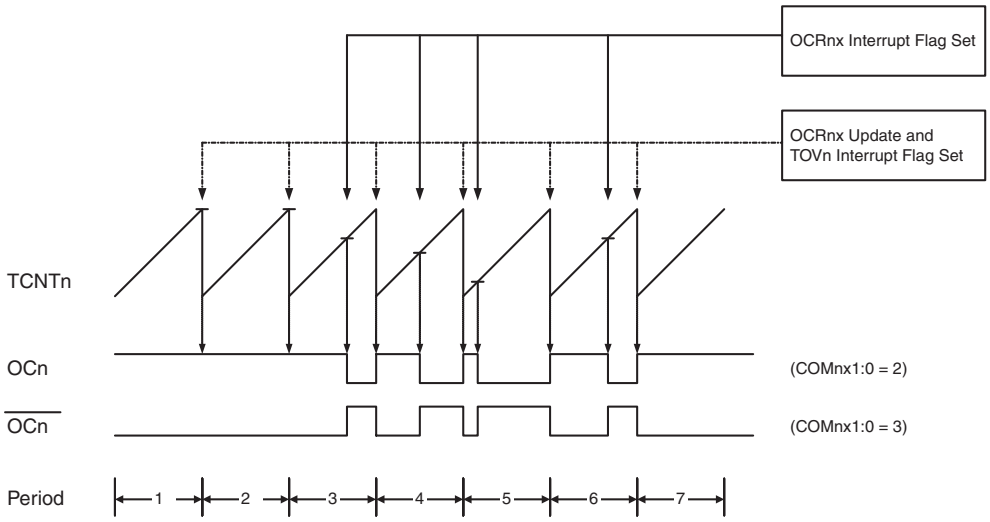
14.6.3 Fast PWM Mode

The fast Pulse Width Modulation or fast PWM mode (WGM02:0 = 3 or 7) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM option by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. TOP is defined as 0xFF when WGM2:0 = 3, and OCR0A when WGM2:0 = 7. In non-inverting Compare Output mode, the Output Compare (OC0x) is cleared on the compare match between TCNT0 and OCR0x, and set at BOTTOM. In inverting Compare Output mode, the output is set on compare match and cleared at BOTTOM. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct PWM mode that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), and therefore reduces total system cost.

In fast PWM mode, the counter is incremented until the counter value matches the TOP value. The counter is then cleared at the following timer clock cycle. The timing diagram for the fast

PWM mode is shown in Figure 14-6. 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 14-6. Fast PWM Mode, Timing Diagram



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

In fast PWM mode, the compare unit allows generation of PWM waveforms on the OC0x pins. Setting the COM0x1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM0x1:0 to three: Setting the COM0A1:0 bits to one allows the OC0A pin to toggle on Compare Matches if the WGM02 bit is set. This option is not available for the OC0B pin (see Table 14-6 on page 87). 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 (COM0x1:0 = 1). The waveform generated will have a maximum frequency of $f_{OC0} = f_{clk\_I/O}/2$ when OCR0A is set to zero. This

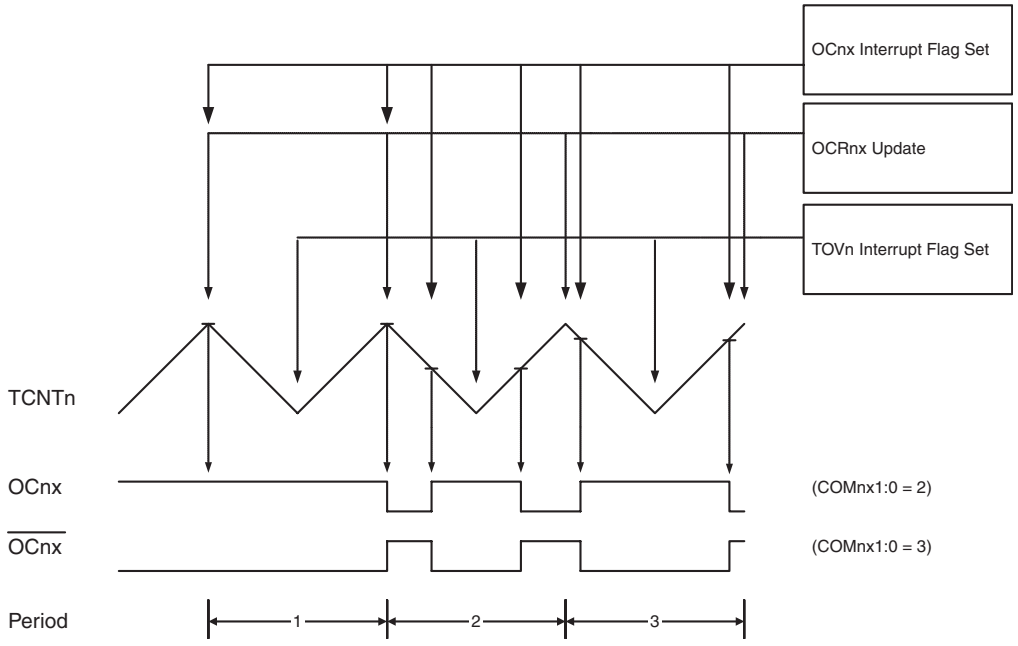
feature is similar to the OC0A toggle in CTC mode, except the double buffer feature of the Output Compare unit is enabled in the fast PWM mode.

14.6.4 Phase Correct PWM Mode

The phase correct PWM mode (WGM02:0 = 1 or 5) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is based on a dual-slope operation. The counter counts repeatedly from BOTTOM to TOP and then from TOP to BOTTOM. TOP is defined as 0xFF when WGM2:0 = 1, and OCR0A when WGM2:0 = 5. In non-inverting Compare Output mode, the Output Compare (OC0x) is cleared on the compare match between TCNT0 and OCR0x while upcounting, and set on the compare match while 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 14-7. 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 14-7. Phase Correct PWM Mode, Timing Diagram



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

In phase correct PWM mode, the compare unit allows generation of PWM waveforms on the OC0x pins. Setting the COM0x1:0 bits to two will produce a non-inverted PWM. An inverted PWM output can be generated by setting the COM0x1:0 to three: Setting the COM0A0 bits to

one allows the OC0A pin to toggle on Compare Matches if the WGM02 bit is set. This option is not available for the OC0B pin (see Table 14-7 on page 88). 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 14-7 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.

- OCRnx changes its value from MAX, like in Figure 14-7. 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 OCnx 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 OCRnx, and for that reason misses the Compare Match and hence the OCnx change that would have happened on the way up.

14.7 Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock (clk_{T0}) is therefore shown as a clock enable signal in the following figures. The figures include information on when interrupt flags are set. Figure 14-8 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 14-8. Timer/Counter Timing Diagram, no Prescaling

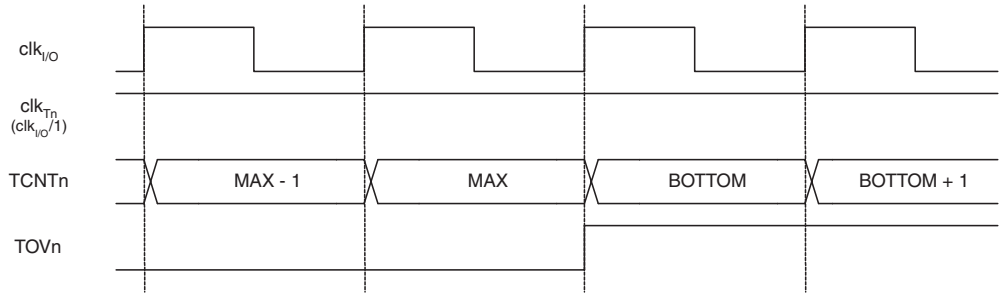


Figure 14-9 shows the same timing data, but with the prescaler enabled.

Figure 14-9. Timer/Counter Timing Diagram, with Prescaler ($f_{clk\_I/O}/8$)

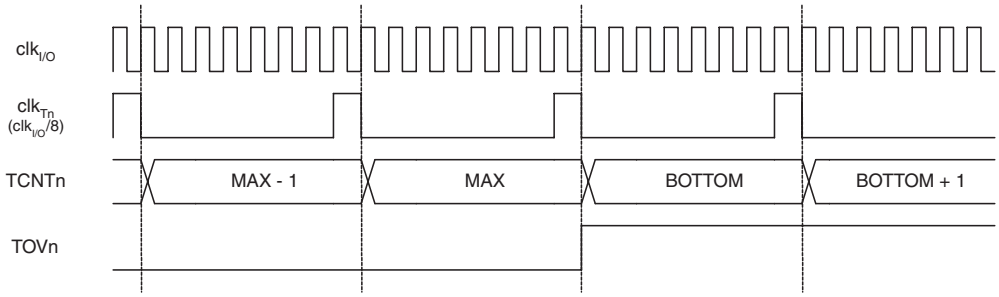


Figure 14-10 shows the setting of OCF0B in all modes and OCF0A in all modes except CTC mode and PWM mode, where OCR0A is TOP.

Figure 14-10. Timer/Counter Timing Diagram, Setting of OCF0x, with Prescaler ($f_{clk\_I/O}/8$)

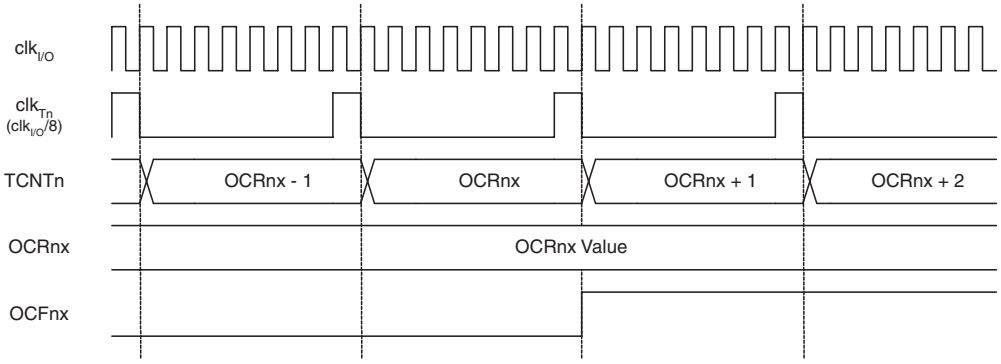
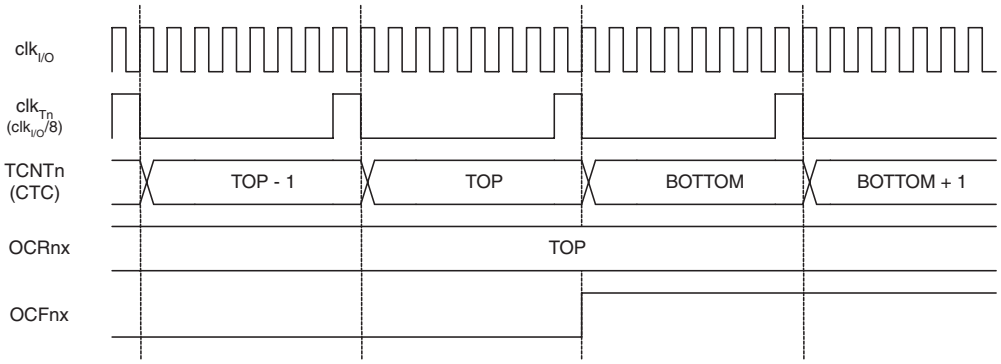


Figure 14-11 shows the setting of OCF0A and the clearing of TCNT0 in CTC mode and fast PWM mode where OCR0A is TOP.

Figure 14-11. Timer/Counter Timing Diagram, Clear Timer on Compare Match mode, with Prescaler ($f_{clk\_I/O}/8$)



14.8 8-bit Timer/Counter Register Description

14.8.1 Timer/Counter Control Register A – TCCR0A

Bit	7	6	5	4	3	2	1	0	
	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:6 – COM0A1:0: Compare Match Output A Mode**

These bits control the Output Compare pin (OC0A) behavior. If one or both of the COM0A1:0 bits are set, the OC0A output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to the OC0A pin must be set in order to enable the output driver.

When OC0A is connected to the pin, the function of the COM0A1:0 bits depends on the WGM02:0 bit setting. Table 14-2 shows the COM0A1:0 bit functionality when the WGM02:0 bits are set to a normal or CTC mode (non-PWM).

Table 14-2. 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 14-3 shows the COM0A1:0 bit functionality when the WGM01:0 bits are set to fast PWM mode.

Table 14-3. 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 81 for more details.

Table 14-4 shows the COM0A1:0 bit functionality when the WGM02:0 bits are set to phase correct PWM mode.

Table 14-4. 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 83 for more details.

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

These bits control the Output Compare pin (OC0B) behavior. If one or both of the COM0B1:0 bits are set, the OC0B output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to the OC0B pin must be set in order to enable the output driver.

When OC0B is connected to the pin, the function of the COM0B1:0 bits depends on the WGM02:0 bit setting. Table 14-5 shows the COM0B1:0 bit functionality when the WGM02:0 bits are set to a normal or CTC mode (non-PWM).

Table 14-5. 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 14-6 shows the COM0B1:0 bit functionality when the WGM02:0 bits are set to fast PWM mode.

Table 14-6. 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 81 for more details.

Table 14-7 shows the COM0B1:0 bit functionality when the WGM02:0 bits are set to phase correct PWM mode.

Table 14-7. 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 83 for more details.

- **Bits 3, 2 – Res: Reserved Bits**

These bits are reserved bits in the ATmega406 and will always read as zero.

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

Combined with the WGM02 bit found in the TCCR0B Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see Table 14-8. 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 80).

Table 14-8. Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	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

14.8.2 Timer/Counter Control Register B – TCCR0B

Bit	7	6	5	4	3	2	1	0	
	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC0A: Force Output Compare A**

The FOC0A bit is only active when the WGM bits specify a non-PWM mode.

However, for ensuring compatibility with future devices, this bit must be set to zero when TCCR0B is written when operating in PWM mode. When writing a logical one to the FOC0A bit, an immediate Compare Match is forced on the Waveform Generation unit. The OC0A output is changed according to its COM0A1:0 bits setting. Note that the FOC0A bit is implemented as a strobe. Therefore it is the value present in the COM0A1:0 bits that determines the effect of the forced compare.

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

The FOC0A bit is always read as zero.

- **Bit 6 – FOC0B: Force Output Compare B**

The FOC0B bit is only active when the WGM bits specify a non-PWM mode.

However, for ensuring compatibility with future devices, this bit must be set to zero when TCCR0B is written when operating in PWM mode. When writing a logical one to the FOC0B bit, an immediate Compare Match is forced on the Waveform Generation unit. The OC0B output is changed according to its COM0B1:0 bits setting. Note that the FOC0B bit is implemented as a strobe. Therefore it is the value present in the COM0B1:0 bits that determines the effect of the forced compare.

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

The FOC0B bit is always read as zero.

- **Bits 5:4 – Res: Reserved Bits**

These bits are reserved bits in the ATmega406 and will always read as zero.

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

See the description in the ”Timer/Counter Control Register A – TCCR0A” on page 86.

- **Bits 2:0 – CS02:0: Clock Select**

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

Table 14-9. 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 14-9. 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.

14.8.3 Timer/Counter Register – TCNT0

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

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

14.8.4 Output Compare Register A – OCR0A

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

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

14.8.5 Output Compare Register B – OCR0B

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

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

14.8.6 Timer/Counter Interrupt Mask Register 0 – TIMSK0

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

- **Bits 7:3 – Res: Reserved Bits**

These bits are reserved bits in the ATmega406 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.

14.8.7 Timer/Counter 0 Interrupt Flag Register – TIFR0

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

- **Bits 7:3 – Res: Reserved Bits**

These bits are reserved bits in the ATmega406 and will always read as zero.

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

The OCF0B bit is set when a Compare Match occurs between the Timer/Counter and the data in OCR0B – Output Compare Register0 B. OCF0B is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF0B is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE0B (Timer/Counter Compare B Match Interrupt Enable), and OCF0B are set, the Timer/Counter Compare Match Interrupt is executed.

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

The OCF0A bit is set when a Compare Match occurs between the Timer/Counter0 and the data in OCR0A – Output Compare Register0. OCF0A is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF0A is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE0A (Timer/Counter0 Compare Match Interrupt Enable), and OCF0A are set, the Timer/Counter0 Compare Match Interrupt is executed.

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

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

The setting of this flag is dependent of the WGM02:0 bit setting. Refer to Table 14-8, ”Waveform Generation Mode Bit Description” on page 88.

15. 16-bit Timer/Counter1

The 16-bit Timer/Counter unit allows accurate program execution timing (event management). The main features are:

- **One Output Compare Unit**
- **Clear Timer on Compare Match (Auto Reload)**
- **Two Independent Interrupt Sources (TOV1 and OCF1A)**

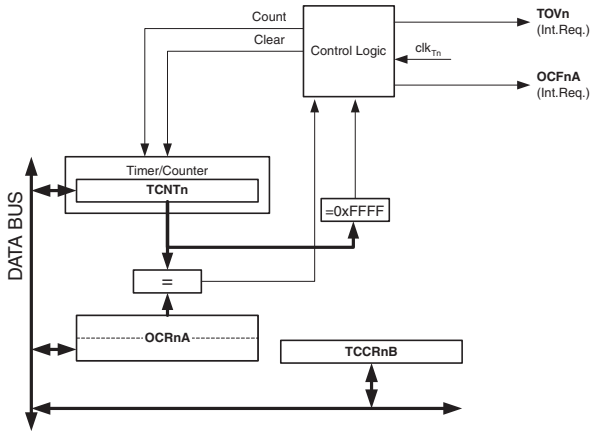
15.1 Overview

Most register and bit references in this document are written in general form. A lower case “n” replaces the Timer/Counter number, and a lower case “x” replaces the output compare unit channel. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT1 for accessing Timer/Counter1 counter value and so on. The physical I/O register and bit locations for ATmega406 are listed in the ”16-bit Timer/Counter Register Description” on page 98.

A simplified block diagram of the 16-bit Timer/Counter is shown in Figure 15-1. CPU accessible I/O registers, including I/O bits and I/O pins, are shown in bold.

The PRTIM1 bit in ”Power Reduction Register 0 - PRR0” on page 35 must be written to zero to enable TImer/Counter1 module.

Figure 15-1. 16-bit Timer/Counter Block Diagram



15.1.1 Registers

The *Timer/Counter* (TCNT1) and the *Output Compare Register* (OCR1A) are both 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 94.The Timer/Counter Control Register (TCCR1B) is an 8-bit register an has no CPU access restrictions. Interrupt requests (abbreviated to Int.Req. in the figure) signals are visible in the *Timer Interrupt Flag Register* (TIFR). Both interrupts are individually masked with the *Timer Interrupt Mask Register* (TIMSK). TIFR and TIMSK are not shown in the figure.

The Timer/Counter is clocked internally via the prescaler. The Clock Select logic block controls which clock source the Timer/Counter uses to increment its value. The Timer/Counter is inactive when no clock source is selected. The output from the clock select logic is referred to as the timer clock (clk_{T1}).

The Output Compare Register (OCR1A) is compared with the Timer/Counter value at all time. The compare match event will set the Compare Match Flag (OCF1A) which can be used to generate an output compare interrupt request.

15.2 Accessing 16-bit Registers

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

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

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

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

Assembly Code Examples[1]

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

C Code Examples[1]

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

Note: 1. See ”About Code Examples” on page 7.

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

It is important to notice that accessing 16-bit registers are atomic operations. If an interrupt occurs between the two instructions accessing the 16-bit register, and the interrupt code

updates the temporary register by accessing the same or any other of the 16-bit timer registers, then the result of the access outside the interrupt will be corrupted. Therefore, when both the main code and the interrupt code update the temporary register, the main code must disable the interrupts during the 16-bit access.

The following code examples show how to do an atomic read of the TCNT1 Register contents. Reading the OCR1A Register can be done using the same principle.

Assembly Code Example[1]

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

C Code Example[1]

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

Note: 1. See "About Code Examples" on page 7.

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

The following code examples show how to do an atomic write of the TCNT1 Register contents. Writing to the OCR1A Register can be done using the same principle.

Assembly Code Example[1]

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

C Code Example[1]

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

Note: 1. See "About Code Examples" on page 7.

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

15.2.1 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.

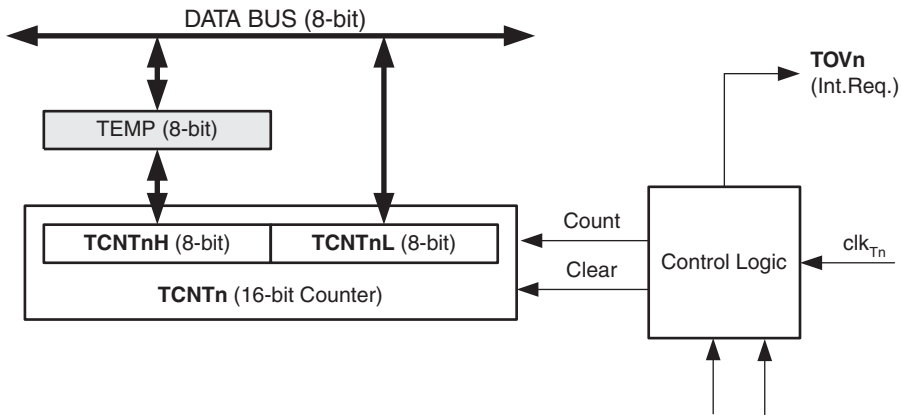
15.3 Timer/Counter Clock Sources

The Timer/Counter is clocked by an internal clock source. The clock source is selected by the Clock Select logic which is controlled by the *Clock Select* (CS1[2:0]) bits located in the *Timer/Counter Control Register B* (TCCR1B). For details on clock sources and prescaler, see "Timer/Counter0 and Timer/Counter1 Prescalers" on page 101.

15.4 Counter Unit

The main part of the 16-bit Timer/Counter is the programmable 16-bit 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 TCNT1 by 1.
Clear	Clear TCNT1 (set all bits to zero).
clk_{T1}	Timer/Counter clock.

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

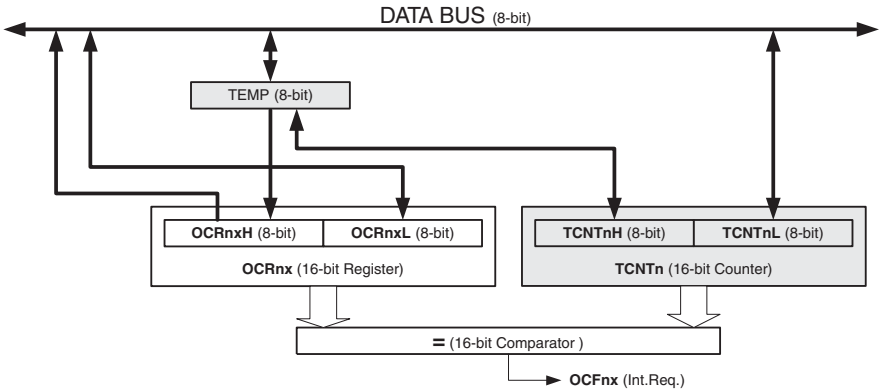
Depending on the mode of operation used, the counter is cleared or incremented at each *Timer Clock* (clk_{T1}). The clk_{T1} is generated from an internal clock source, selected by the *Clock Select* bits (CS1[2:0]). When no clock source is selected (CS1[2:0] = 0) the timer is stopped. However, the TCNT1 value can be accessed by the CPU, independent of whether clk_{T1} is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

15.5 Output Compare Unit

The 16-bit comparator continuously compares TCNT1 with the *Output Compare Register* (OCR1A). If TCNT equals OCR1A the comparator signals a match. A match will set the *Output Compare Flag* (OCF1A) at the next timer clock cycle. If enabled (OCIE1A = 1), the output compare flag generates an output compare interrupt. The OCF1A flag is automatically cleared when the interrupt is executed. Alternatively the OCF1A flag can be cleared by software by writing a logical one to its I/O bit location.

Figure 15-3 shows a block diagram of the output compare unit. The small “n” in the register and bit names indicates the device number (n = 1 for Timer/Counter1), and the “x” indicates output compare unit (A). The elements of the block diagram that are not directly a part of the output compare unit are gray shaded.

Figure 15-3. Output Compare Unit, Block Diagram



15.5.1 Compare Match Blocking by TCNT1 Write

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

15.5.2 Using the Output Compare Unit

Since writing TCNT1 in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNT1 when using any of the output compare channels, independent of whether the Timer/Counter is running or not. If the value written to TCNT1 equals the OCR1x value, the compare match will be missed.

15.6 16-bit Timer/Counter Register Description

15.6.1 Timer/Counter1 Control Register B – TCCR1B

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	CTC1	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 7:4 – Res: Reserved Bits**

These bits is a reserved bit in the ATmega406 and always reads as zero.

- **Bit 3 – CTC1: Clear Timer/Counter1 on Compare Match**

When the CTC1 control bit is set (one), Timer/Counter1 is reset to 0x00 in the CPU clock cycle after a compare match.

- **Bit 2:0 – CS1[2:0]: Clock Select**

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

Table 15-1. CS1[2:0] - Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/32 (From prescaler)
1	0	0	$clk_{I/O}$/64 (From prescaler)
1	0	1	$clk_{I/O}$/128 (From prescaler)
1	1	0	$clk_{I/O}$/256 (From prescaler)
1	1	1	$clk_{I/O}$/1024 (From prescaler)

15.6.2 Timer/Counter1 – TCNT1H and TCNT1L

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

The two *Timer/Counter* I/O locations (TCNT1H and TCNT1L, combined TCNT1) give direct access, both for read and for write operations, to the Timer/Counter unit 16-bit counter. To ensure that both the high and low bytes are read and written simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary high byte register (TEMP). This temporary register is shared by all the other 16-bit registers. See “Accessing 16-bit Registers” on page 94.

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

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

15.6.3 Output Compare Register 1 A – OCR1AH and OCR1AL

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

The Output Compare Register contains a 16-bit value that is continuously compared with the counter value (TCNT1).

The Output Compare Register is 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 94.

15.6.4 Timer/Counter Interrupt Mask Register 1 – TIMSK1

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

- **Bit 7:2 – Res: Reserved Bits**

These bits are reserved bits in the ATmega406 and always reads as zero.

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

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare A Match interrupt is enabled. The corresponding Interrupt Vector (see “Reset and Interrupt Handling” on page 14) is executed when the OCF1A flag, located in TIFR1, is set.

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

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 overflow interrupt is enabled. The corresponding Interrupt Vector (see “Reset and Interrupt Handling” on page 14) is executed when the TOV1 flag, located in TIFR1, is set.

15.6.5 Timer/Counter Interrupt Flag Register – TIFR1

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

- **Bit 7:2 – Res: Reserved Bits**

These bits are reserved bits in the ATmega406 and always reads as zero.

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

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the Output Compare Register A (OCR1A).

OCF1A is automatically cleared when the Output compare Match A Interrupt Vector is executed. Alternatively, OCF1A can be cleared by writing a logic one to its bit location.

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

TOV1 Flag is set when the Timer overflows.

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

16. Timer/Counter0 and Timer/Counter1 Prescalers

Timer/Counter1 and Timer/Counter0 share the same prescaler module, but the Timer/Counters can have different prescaler settings. The description below applies to both Timer/Counter1 and Timer/Counter0.

16.0.1 Internal Clock Source

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

16.0.2 Prescaler Reset

The prescaler is free running, i.e., operates independently of the Clock Select logic of the Timer/Counter, and it is shared by Timer/Counter1 and Timer/Counter0. Since the prescaler is not affected by the Timer/Counter's clock select, the state of the prescaler will have implications for situations where a prescaled clock is used. One example of prescaling artifacts occurs when the timer is enabled and clocked by the prescaler (6 > CSn2:0 > 1). The number of system clock cycles from when the timer is enabled to the first count occurs can be from 1 to N+1 system clock cycles, where N equals the prescaler divisor (8, 64, 256, or 1024).

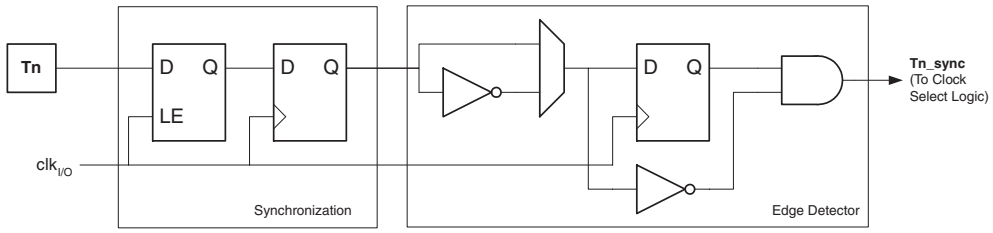
It is possible to use the prescaler reset for synchronizing the Timer/Counter to program execution. However, care must be taken if the other Timer/Counter that shares the same prescaler also uses prescaling. A prescaler reset will affect the prescaler period for all Timer/Counters it is connected to.

16.0.3 External Clock Source

An external clock source applied to the T1/T0 pin can be used as Timer/Counter clock (clk_{T1}/clk_{T0}). The T1/T0 pin is sampled once every system clock cycle by the pin synchronization logic. The synchronized (sampled) signal is then passed through the edge detector. Figure 16-1 shows a functional equivalent block diagram of the T1/T0 synchronization and edge detector logic. The registers are clocked at the positive edge of the internal system clock ($clk_{I/O}$). The latch is transparent in the high period of the internal system clock.

The edge detector generates one clk_{T1}/clk_{T0} pulse for each positive (CSn2:0 = 7) or negative (CSn2:0 = 6) edge it detects.

Figure 16-1. T1/T0 Pin Sampling



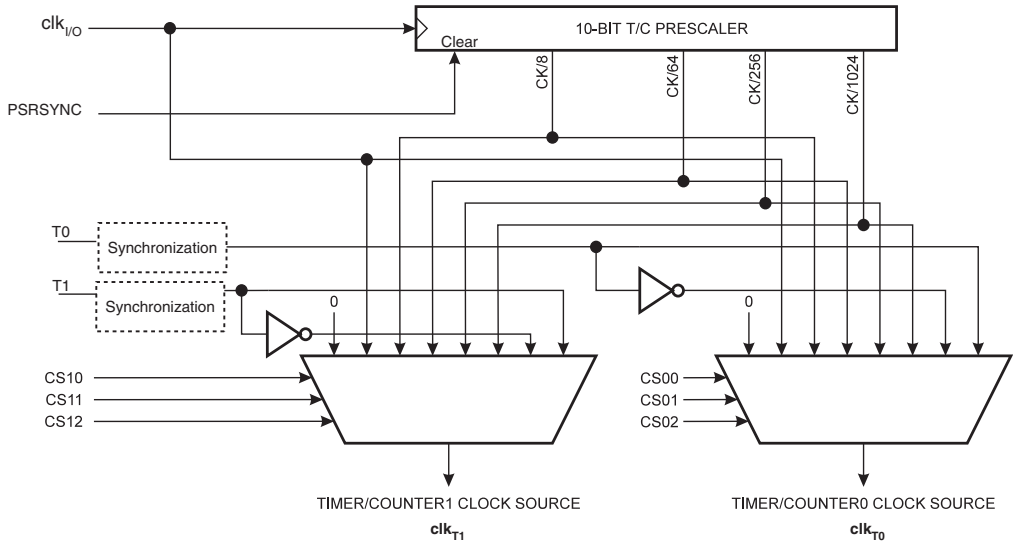
The synchronization and edge detector logic introduces a delay of 2.5 to 3.5 system clock cycles from an edge has been applied to the T1/T0 pin to the counter is updated.

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

Each half period of the external clock applied must be longer than one system clock cycle to ensure correct sampling. The external clock must be guaranteed to have less than half the system clock frequency ($f_{ExtClk} < f_{clk\_I/O}/2$) given a 50/50% 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 16-2. Prescaler for Timer/Counter0 and Timer/Counter1 (1)



Note: 1. The synchronization logic on the input pins (T1/T0) is shown in Figure 16-1.

16.0.4 General Timer/Counter Control Register – GTCCR

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

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

Writing the TSM bit to one activates the Timer/Counter Synchronization mode. In this mode, the value that is written to the PSRSYNC bit is kept, hence keeping the corresponding prescaler reset signals asserted. This ensures that the corresponding Timer/Counters are halted and can be configured to the same value without the risk of one of them advancing during configuration. When the TSM bit is written to zero, the PSRSYNC bit is cleared by hardware, and the Timer/Counters start counting simultaneously.

- **Bit 0 – PSRSYNC: Prescaler Reset**

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

17. Coulomb Counter - Dedicated Fuel Gauging Sigma-delta ADC

17.1 Features

- **Sampled System Coulomb Counter**
- **Low Power Sigma-Delta ADC Optimized for Coulomb Counting**
- **Instantaneous Current Output with 3.9 ms Conversion Time**
- **Accumulate Current Output with Programmable Conversion Time: 125, 250, 500 or 1000 ms**
- **Input voltage Range Larger than ± 0.15V, Allowing Measurement of More than ± 30A @ 5 mΩ**
- **53.7 µV Resolution (10.7 mA @ 5 mΩ) for Instantaneous Current Output**
- **1.68 µV Resolution (0.335 mA @ 5 mΩ) for Accumulate Current Output**
- **Input Offset Less than 10µV for the ADC**
- **Interrupt on Instantaneous Current Conversion Complete**
- **Interrupt on Accumulate Current Conversion Complete**
- **Interrupt on Regular Current with Programmable Compare Level and Programmable Sampling Interval: 250, 500, 1000 or 2000 ms**

ATmega406 features a dedicated Sigma-Delta ADC (CC-ADC) optimized for Coulomb Counting to sample the charge or discharge current flowing through the external sense resistor R_{SENSE}. Two different output values are provided, Instantaneous Current and Accumulate Current. The Instantaneous Current Output has a short conversion time at the cost of lower resolution. The Accumulate Current Output provides a highly accurate current measurement for Coulomb Counting.

The sampling Coulomb Counter provides a highly accurate and flexible solution. Accuracy can easily be traded against conversion time. It also provides Regular Current detection. This allows ultra-low power operation in Power-save mode when small charge or discharge currents are flowing.

17.2 Operation

When enabled, the CC-ADC continuously measures the voltage over the external sense resistor R_{SENSE}. The conversion time for Instantaneous Current is fixed to 3.9 ms allowing the output to closely follow the input signal. After each Instantaneous Current conversion an interrupt is generated if the interrupt is enabled. The resolution for this conversion is 53.7 µV.

The Accumulate Current output is a high-resolution, high accuracy output with programmable conversion time. The converted value is an accurate measurement of the average current flow during one conversion period. The CC-ADC generates an interrupt each time a new Accumulate Current conversion has finished if the interrupt is enabled. While the CC-ADC is converting, the CPU can enter sleep mode and wait for an interrupt from the Accumulate Current conversion. After adding the new Accumulate Current value for Coulomb Counting, the CPU can go back to sleep again. This reduces the CPU workload, and allows more time spent in low power modes, reducing power consumption.

In-system offset voltage for the CC-ADC is typically in the range 0 - 100 µV. To compensate for this offset error, a CC-ADC offset value should be stored in EEPROM and subtracted from each Accumulate Current conversions before the resulting value is added for Coloumb Counting. The CC-ADC offset value can be found by performing a CC-ADC conversion at typical temperature with zero current flowing through R_{SENSE}.

When the battery is not used or the current level stays very low for a long time, it is recommended to estimate the charge flow instead of using the CC-ADC for Coloumb Counting. The

charge flow estimation should be based on the self-discharge rate of the battery and the standby current of the battery system.

The CC-ADC can generate an interrupt if the result of an Instantaneous Current conversion is greater than a programmable threshold. This allows the detection of a Regular Current condition. This function is available in Active mode and all sleep modes except Power-down and Power-off mode. This allows an ultra-low power operation in Power-save, where the CC-ADC can be configured to enter a Regular Current detection mode with a programmable current sampling interval. By setting the CADSE bit in CADCSRA, the Coulomb Counter will repeatedly do one Instantaneous Current conversion, before it is being turned off for a timing interval specified by the CADSI bits in CADCSRA. This allows operating the Regular Current detection while keeping the Coulomb Counter off most of the time.

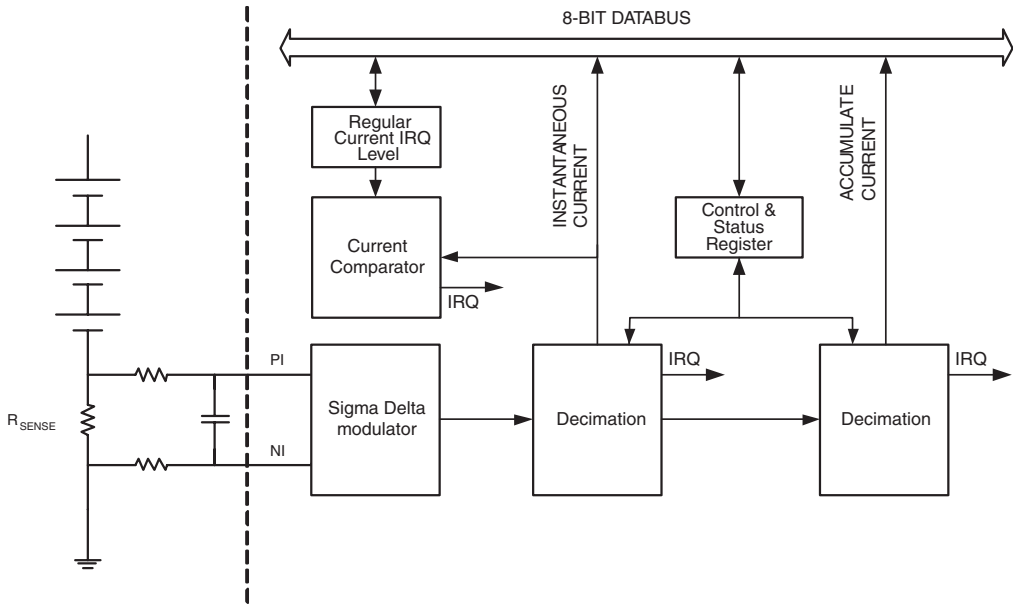
The Coulomb Counter is disabled in Power-down mode. In this mode, time measurements and the battery self-discharge characteristics should be used to estimate the charge flow. When waking up from Power-down mode, the CC-ADC will automatically resume continuous operation.

The CC-ADC is enabled by setting the CC-ADC Enable bit, CADEN, in CADCSRA. Note that the bandgap voltage reference must be enabled separately, see “Bandgap Calibration C Register – BGCCR” on page 116.

The CC-ADC will not consume power when CADEN is cleared. It is therefore recommended to switch off the CC-ADC whenever the Coulomb Counter or Regular Current Detection functions are not used. The CC-ADC is automatically disabled in Power-down and Power-off mode.

After the CC-ADC is enabled, either by setting the CADEN bit or leaving Power-down with CADEN already set, the first four conversions do not contain useful data and should be ignored. This also applies after clearing the CADSE bit.

Figure 17-1. Coulomb Counter Block Diagram



17.2.1 CC-ADC Control and Status Register A – CADCSRA

Bit	7	6	5	4	3	2	1	0	
	CADEN	–	CADUB	CADAS1	CADAS0	CADSI1	CADSI0	CADSE	CADCSRA
Read/Write	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – CADEN: CC-ADC Enable**

When the CADEN bit is cleared (zero), the CC-ADC is disabled, and any ongoing conversions will be terminated. When the CADEN bit is set (one), the CC-ADC will continuously measure the voltage drop over the external sense resistor R_{SENSE}. In Power-off, the CC-ADC is always disabled. Note that the bandgap voltage reference must be enabled separately, see “Bandgap Calibration C Register – BGCCR” on page 116.

- **Bit 6 – Res: Reserved**

This bit is reserved bit in the ATmega406 and will always read as zero.

- **Bit 5 - CADUB: CC-ADC Update Busy**

The CC-ADC operates in a different clock domain than the CPU. Whenever a new value is written to CADCSRA, CADRCC or CADRDC, this value must be synchronized to the CC-ADC clock domain. Subsequent writes to these registers will be blocked during this synchronization. The CADUB bit will be read as one while any of these registers is being synchronized, and will be read as zero when neither register is being synchronized. The synchronization and blocking is independent per register, thus all registers can be updated once the CADUB flag has been read as zero.

- **Bits 4:3 – CADAS1:0: CC-ADC Accumulate Current Select**

The CADAS bits select the conversion time for the Accumulate Current output as shown in Table 17-1.

Table 17-1. CC-ADC Accumulate Current Conversion Time

CADAS1:0	CC-ADC Accumulate Current Conversion Time
00	125 ms
01	250 ms
10	500 ms
11	1 s

- **Bits 2:1 – CADSI1:0: CC-ADC Current Sampling Interval**

The CADSI bits determine the current sampling interval for the Regular Current detection as shown in Table 17-2. The current sampling interval is only used if the CADSE bit is set.

Table 17-2. CC-ADC Regular Current Sampling Interval

CADSI1:0	CC-ADC Regular Current Sampling Interval
00	250 ms (+ sampling time)
01	500 ms (+ sampling time)
10	1 s (+ sampling time)
11	2 s (+ sampling time)

Note: Sampling time ~ 16 ms.

- **Bit 0 – CADSE: CC-ADC Current Sampling Enable**

When the CADSE bit is written to one, the ongoing CC-ADC conversion is aborted, and the CC-ADC enters Regular Current detection mode.

17.2.2 CC-ADC Control and Status Register B – CADCSRB

Bit	7	6	5	4	3	2	1	0	
	–	CADACIE	CADRCIE	CADICIE	–	CADACIF	CADRCIF	CADICIF	CADCSRB
Read/Write	R	R/W	R/W	R/W	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 bits in the ATmega406 and will always read as zero.

- **Bit 6 – CADACIE: CC-ADC Accumulate Current Interrupt Enable**

When the CADACIE bit is set (one), and the I-bit in the Status Register is set (one), the CC-ADC Accumulate Current Interrupt is enabled.

- **Bit 5 – CADRCIE: CC-ADC Regular Current Interrupt Enable**

When the CADRCIE bit is set (one), and the I-bit in the Status Register is set (one), the CC-ADC Regular Current Interrupt is enabled.

- **Bit 4 – CADICIE: CC-ADC Instantaneous Current Interrupt Enable**

When the CADICIE bit is set (one), and the I-bit in the Status Register is set (one), the CC-ADC Instantaneous Current Interrupt is enabled.

- **Bit 2 – CADACIF: CC-ADC Accumulate Current Interrupt Flag**

The CADACIF bit is set (one) after the Accumulate Current conversion has completed. The CC-ADC Accumulate Current Interrupt is executed if the CADACIE bit and the I-bit in SREG are set (one). CADACIF is cleared by hardware when executing the corresponding Interrupt Handling Vector. Alternatively, CADACIF is cleared by writing a logic one to the flag.

- **Bit 1 – CADRCIF: CC-ADC Regular Current Interrupt Flag**

The CADRCIF bit is set (one) when the absolute value of the result of the last CC-ADC conversion is greater than, or equal to, the compare values set by the CC-ADC Regular Charge/Discharge Current Level Registers. A positive value is compared to the Regular Charge Current Level, and a negative value is compared to the Regular Discharge Current Level. The CC-ADC Regular Current Interrupt is executed if the CADRCIE bit and the I-bit in SREG are set (one). CADRCIF is cleared by hardware when executing the corresponding Interrupt Handling vector. Alternatively, CADRCIF is cleared by writing a logic one to the flag.

- **Bit 0 – CADICIF: CC-ADC Instantaneous Current Interrupt Flag**

The CADICIF bit is set (one) when a CC-ADC Instantaneous Current conversion is completed. The CC-ADC Instantaneous Current Interrupt is executed if the CADICIE bit and the I-bit in SREG are set (one). CADICIF is cleared by hardware when executing the corresponding Interrupt Handling vector. Alternatively, CADICIF is cleared by writing a logic one to the flag.

17.2.3 CC-ADC Instantaneous Current – CADICH and CADICL

Bit	15	14	13	12	11	10	9	8	
	CADIC[15:8]								CADICH
	CADIC[7:0]								CADICL
Bit	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

When a CC-ADC Instantaneous Current conversion is complete, the result is found in these two registers. CADIC15:0 represents the converted result in 2's complement format, sign extended to 16 bits.

When CADICL is read, the CC-ADC Instantaneous Current register is not updated until CADCH is read. Reading the registers in the sequence CADICL, CADICH will ensure that consistent values are read.

17.2.4 CC-ADC Accumulate Current– CADAC3, CADAC2, CADAC1 and CADAC0

Bit	31	30	29	28	27	26	25	24	
	23	22	21	20	19	18	17	16	
	15	14	13	12	11	10	9	8	
	7	6	5	4	3	2	1	0	
	CADAC[31:24]								CADAC3
	CADAC[23:16]								CADAC2
	CADAC[15:8]								CADAC1
	CADAC[7:0]								CADAC0
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

The CADAC3, CADAC2, CADAC1 and CADAC0 Registers contain the Accumulate Current measurements in 2’s complement format, sign extended to 32 bits.

When CADAC0 is read, the CC-ADC Accumulate Current register is not updated until CADAC3 is read. Reading the registers in the sequence CADAC0, CADAC1, CADAC2, CADAC3 will ensure that consistent values are read.

17.2.5 CC-ADC Regular Charge Current – CADRCC

Bit	7	6	5	4	3	2	1	0	
	CADRCC[7:0]								CADRCC
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 CC-ADC Regular Charge Current Register determines the threshold level for the Regular Charge Current detection. When the result of a CC-ADC Instantaneous Current conversion is positive with a value greater than, or equal to, the Regular Charge Current level, the CC-ADC Regular Current Interrupt Flag is set.

The value in this register is specified in 2's complement format, and it defines the eight least significant bits of the Regular Charge Current level. The most significant bits of the Regular Charge Current level are always zero. The programmable range for the Regular Charge Current level is given in Table 17-3.

Table 17-3. Programmable Range for the Regular Charge Current Level

		Minimum	Maximum	Step Size
Voltage (µV)		0	13700	53.7
Current (mA)	R_{SENSE} = 5 mΩ	0	2740	10.7
	R_{SENSE} = 7 mΩ	0	1957	7.7

The CC-ADC Regular Charge Current Register does not affect the setting of the CC-ADC Conversion Complete Interrupt Flag.

17.2.6 CC-ADC Regular Discharge Current – CADRDC

Bit	7	6	5	4	3	2	1	0	
	CADRDC[7:0]								CADRDC
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 CC-ADC Regular Discharge Current Register determines the threshold level for the Regular Discharge Current detection. When the result of a CC-ADC Instantaneous Current conversion is negative with an absolute value greater than, or equal to, the Regular Discharge Current level, the CC-ADC Regular Current Interrupt Flag is set.

The value in this register is specified in 2's complement format, and it defines the eight least significant bits of the Regular Discharge Current level. The most significant bits of the Regular Charge Current level are always one. The programmable range for the Regular Discharge Current level is given in Table 17-4.

Table 17-4. Programmable Range for the Regular Discharge Current Level

		Minimum	Maximum	Step Size
Voltage (µV)		0	13700	53.7
Current (mA)	R_{SENSE} = 5 mΩ	0	2740	10.7
	R_{SENSE} = 7 mΩ	0	1957	7.7

The CC-ADC Regular Discharge Current Register does not affect the setting of the CC-ADC Conversion Complete Interrupt Flag.

18. Voltage Regulator

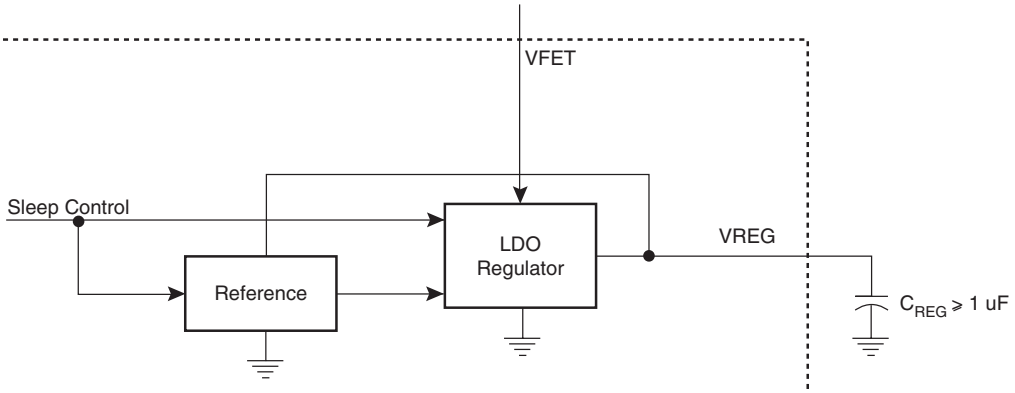
ATmega406 is supplied by the V_{FET} terminal. Operating voltage range at the V_{FET} terminal is 4.0 - 25 volts. The internal Voltage Regulator regulates this voltage down to 3.3V which is a suitable supply voltage for the internal logic, I/O lines, and analog circuitry.

The voltage regulator contains a power consumption control module. When the chip enters low power modes, the voltage regulator will reduce the consumption in the regulator itself, further contributing to low power consumption.

The block diagram of the Voltage Regulator is shown in Figure 18-1. An external decoupling capacitor of 1 µF or larger is required for stable operation of the Voltage Regulator. A larger capacitor will allow larger load currents and increase start-up time. See Electrical Characteristics for more details.

The block diagram of the Voltage Regulator is shown in Figure 18-1.

Figure 18-1. Voltage Regulator



19. Voltage ADC – 10-channel General Purpose 12-bit Sigma-Delta ADC

19.1 Features

- **12-bit Resolution**
- **±1 LSB Accuracy**
- **519µs Conversion Time**
- **Four Differential Input Channels for Cell Voltage Measurements**
- **Six Single Ended Input Channels**
- **0 to 0.9 x VREF Input Voltage Range**
- **0.2x Pre-scaling of Cell Voltages and VREG**
- **Interrupt on V-ADC Conversion Complete**

The ATmega406 features a 12-bit Sigma-Delta ADC. Automatic offset cancellation technique reduces the input offset voltage to less than 0.5 mV.

The Voltage ADC (V-ADC) is connected to ten different sources through the Input Multiplexer. There are four differential channels for Cell Voltage measurements. These channels are scaled 0.2x to comply with the Full Scale range of the V-ADC. In addition there are six single ended channels referenced to SGND. One channel is for measuring the internal temperature sensor VPTAT and five channels for measuring the ADC input pins at Port A. ADC3:0 are not scaled, meaning that full-scale reading corresponds to 1.1 V. ADC4 is scaled by 0.2x, meaning that full-scale reading corresponds to 5.5 V. The ADC4 input can be used to measure the voltage at the PA4 pin when this pin is used to supply an external thermistor, see Figure 28-1 on page 210.

To obtain a total absolute accuracy better than ± 0.25% for the cell voltage measurements, calibration registers for the individual cell voltage gain in the analog front-end is provided. A factory calibration value is stored in the signature row, see Section 26.7.10 ”Reading the Signature Row from Software” on page 180. The V-ADC conversion of a cell voltage must be scaled with the corresponding calibration value by software to correct for gain error in the analog front-end.

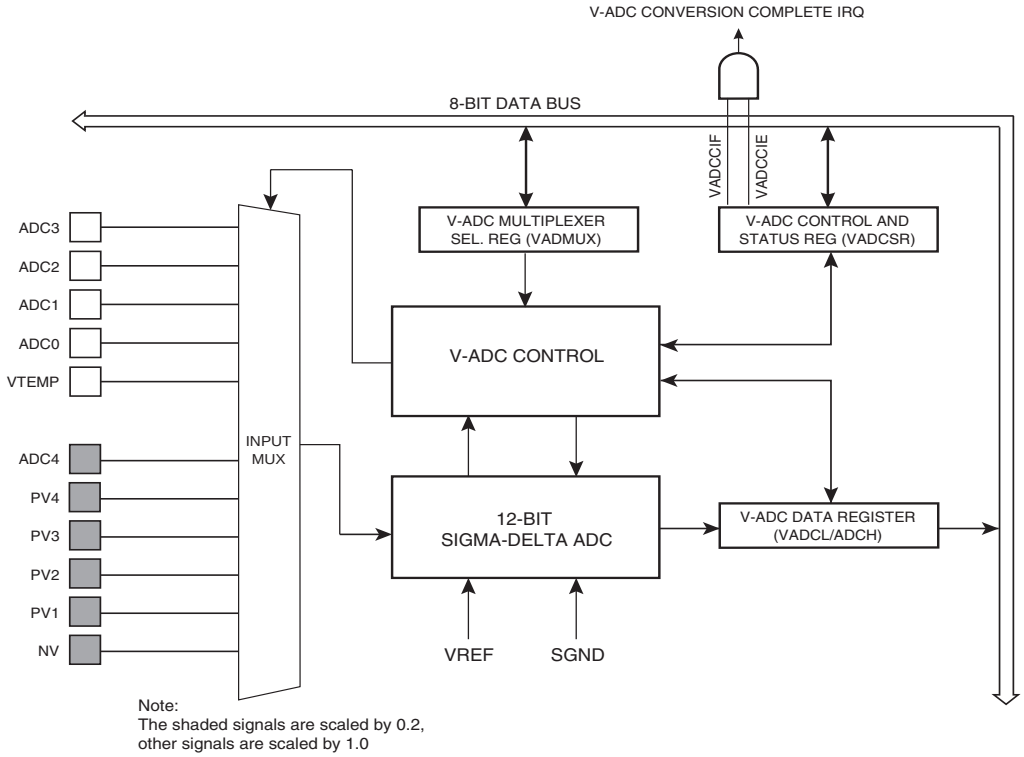
The PRVADC bit in ”Power Reduction Register 0 - PRR0” on page 35 must be written to zero to enable V-ADC module.

19.2 Operation

To enable V-ADC conversions, the V-ADC Enable bit, VADEN, in V-ADC Control and Status Register – VADCSR must be set. If this bit is cleared, the V-ADC will be switched off, and any ongoing conversions will be terminated. The V-ADC is automatically disabled in Power-down and Power-off mode. Note that the bandgap voltage reference must be enabled and disabled separately, see “Bandgap Calibration C Register – BGCCR” on page 116.

To perform a V-ADC conversion, the analog input channel must first be selected by writing to the VADMUX bits in VADMUX. When a logical one is written to the V-ADC Start Conversion bit VADSC, a conversion of the selected channel will start. The VADSC bit stays high as long as the conversion is in progress and will be cleared by hardware when the conversion is completed. If a different data channel is selected while a conversion is in progress, the ADC will finish the current conversion before performing the channel change. When a conversion is finished the V-ADC Conversion Complete Interrupt Flag – VADCCIF is set. One 12-bit conversion takes 519 µs to complete from the start bit is set to the interrupt flag is set. To ensure that correct data is read, both high and low byte data registers should be read before starting a new conversion.

Figure 19-1. Voltage ADC Block Schematic



19.2.1 V-ADC Multiplexer Selection Register – VADMUX

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	VADMUX3	VADMUX2	VADMUX1	VADMUX0	VADMUX
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:4 – RES: Reserved Bits**

These bits are reserved bits in the ATmega406 and will always read as zero.

- **Bit 3:0 – VADMUX3:0: V-ADC Channel Selection Bits**

The VADMUX bits determine the V-ADC channel selection. See Table 19-1 on page 112.

Table 19-1. VADMUX channel selection

VADMUX3:0	Channel Selected	Scale
0001	CELL 1	0.2
0010	CELL 2	0.2
0011	CELL 3	0.2
0100	CELL 4	0.2
0101	ADC4	0.2
0110	VTEMP	1.0
0111	ADC0	1.0

Table 19-1. VADMUX channel selection (Continued)

VADMUX3:0	Channel Selected	Scale
1000	ADC1	1.0
1001	ADC2	1.0
1010	ADC3	1.0

19.2.2 V-ADC Control and Status Register – VADCSR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	VADEN	VADSC	VADCCIF	VADCCIE	VADCSR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:4 – RES: Reserved Bits**

These bits are reserved bits in the ATmega406 and will always read as zero.

- **Bit 3 – VADEN: V-ADC Enable**

Writing this bit to one enables V-ADC conversion. By writing it to zero, the V-ADC is turned off. Turning the V-ADC off while a conversion is in progress will terminate this conversion. Note that the bandgap voltage reference must be enabled separately, see “Bandgap Calibration C Register – BGCCR” on page 116.

- **Bit 2 – VADSC: Voltage ADC Start Conversion**

Write this bit to one to start a new conversion of the selected channel.

VADSC will read as one as long as the conversion is not finished. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect. VADSC will automatically be cleared when the VADEN bit is written to zero.

- **Bit 1 – VADCCIF: V-ADC Conversion Complete Interrupt Flag**

This bit is set when a V-ADC conversion completes and the data registers are updated. The V-ADC Conversion Complete Interrupt is executed if the VADCCIE bit and the I-bit in SREG are set. VADCCIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, VADCCIF is cleared by writing a logical one to the flag. Beware that if doing a Read-Modify-Write on VADCSR, a pending interrupt can be disabled.

- **Bit 0 – VADCCIE: V-ADC Conversion Complete Interrupt Enable**

When this bit is written to one and the I-bit in SREG is set, the V-ADC Conversion Complete Interrupt is activated.

19.2.3 The V-ADC Data Register – VADCL and VADCH

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	VADC[11:8]				VADCH
	VADC[7:0]								VADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

When a V-ADC conversion is complete, the result is found in these two registers. To ensure that correct data is read, the data registers must be read before starting a new conversion.

- **VADC11:0: V-ADC Conversion Result**

These bits represent the result from the conversion. To obtain maximum accuracy of cell voltage measurements, the V-ADC measurements of cells 1 to 4 must be adjusted by the corresponding calibration values found in the device signature row. "Reading the Fuse and Lock Bits from Software" on page 180 for details.

19.2.4 Digital Input Disable Register 0 – DIDR0

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	VADC3D	VADC2D	VADC1D	VADC0D	DIDR0
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:4 – Res: Reserved Bits**

These bits are reserved for future use. To ensure compatibility with future devices, these bits must be written to zero when DIDR0 is written.

- **Bit 3:0 – VADC3D:VADC0D: V-ADC3:0 Digital Input Disable**

When this bit is written logic one, the digital input buffer on the corresponding V-ADC pin is disabled. The corresponding PIN Register bit will always read as zero when this bit is set. When an analog signal is applied to the VADC3:0 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.

20. Voltage Reference and Temperature Sensor

20.1 Features

- **Accurate Voltage Reference of 1.100V**
- **± 0.1% Accuracy After Calibration (2 mV Calibration Steps)**
- **Temperature Drift Less than 80 ppm/°C after Calibration**
- **Alternate Low Power Voltage Reference for Voltage Regulator**
- **Internal Temperature Sensor**
- **Possibility for Runtime Compensation of Temperature Drift in Both Voltage Reference and On-chip Oscillators**
- **External Decoupling for Optimum Noise Performance**
- **Low Power Consumption**

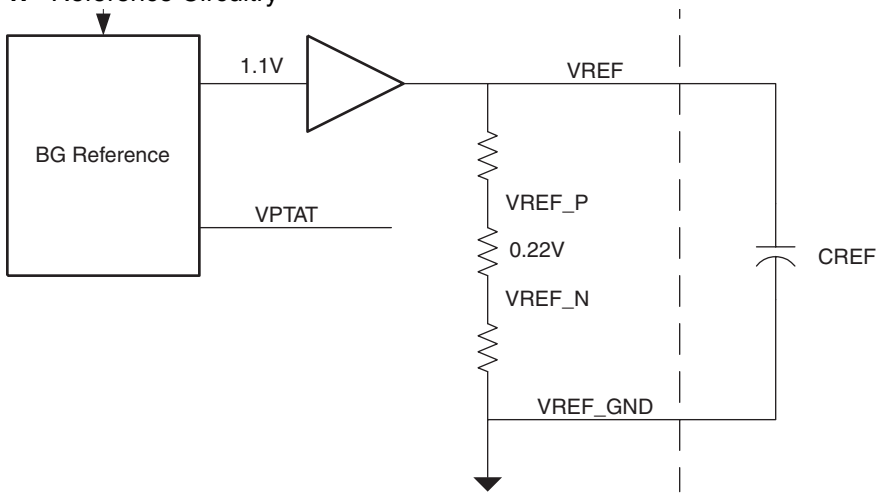
A low power band-gap reference provides ATmega406 with an accurate On-chip voltage reference V_{REF} of 1.100V. This reference voltage is used as reference for the On-chip Voltage Regulator, the V-ADC and the CC-ADC. The reference to the ADCs uses a buffer with external decoupling capacitor to enable excellent noise performance with minimum power consumption. The reference voltage V_{REF_P}/V_{REF_N} to the CC-ADC is scaled to match the full scale requirement at the current sense input pins. This configuration also enables concurrent operation of both V-ADC and CC-ADC.

To guaranty ultra low temperature drift after factory calibration, ATmega406 features a two-step calibration algorithm. The first step is performed at 85°C and the second at room temperature. By default, Atmel factory calibration is performed at 85°C, and the result is stored in Flash. The customer can easily implement the second calibration step in their test flow. This requires an accurate input voltage and a stable room temperature. Temperature drift after this calibration is guarantied by design and characterization to be less than 80 ppm/°C from 0°C to 60°C and 100 ppm/°C from 0°C to 85°C. The BG Calibration C Register can also be altered runtime to implement temperature compensation in software. Very high accuracy for any temperature inside the temperature range can thus be achieved at the cost of extra calibration steps.

A lower power, less accurate voltage reference source exists. This voltage reference source is chosen as reference for the voltage regulator whenever the band-gap voltage reference is disabled. This voltage reference source is not available for the V-ADC and CC-ADC.

ATmega406 has an On-chip temperature sensor for monitoring the die temperature. A voltage Proportional-To-Absolute-Temperature, V_{PTAT}, is generated in the voltage reference circuit and connected to the multiplexer at the V-ADC input. This temperature sensor can be used for runtime compensation of temperature drift in both the voltage reference and the On-chip Oscillator. To get the absolute temperature in degrees Kelvin, the measured V_{PTAT} voltage must be scaled with the VPTAT factory calibration value stored in the signature row. See ”Reading the Signature Row from Software” on page 180 for details.

Figure 20-1. Reference Circuitry



20.2 Register Description for Voltage Reference and Temperature Sensor

20.2.1 Bandgap Calibration C Register – BGCCR

Bit	7	6	5	4	3	2	1	0	
	BGEN	–	BGCC5	BGCC4	BGCC3	BGCC2	BGCC1	BGCC0	BGCCR
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 - BGEN**

This bit is not available from revision E and on of the ATmega406. A complete description is found in the revision A of this document.

- **Bit 6 – Res: Reserved Bit**

This bit is reserved for future use.

- **Bit 5:0 – BGCC5:0: BG Calibration of PTAT Current**

These bits are used for trimming of the nominal value of the bandgap reference voltage. These bits are binary coded. Minimum VREF: 000000, maximum VREF: 111111. Step size approximately 2 mV.

20.2.2 Bandgap Calibration R Register – BGCRR

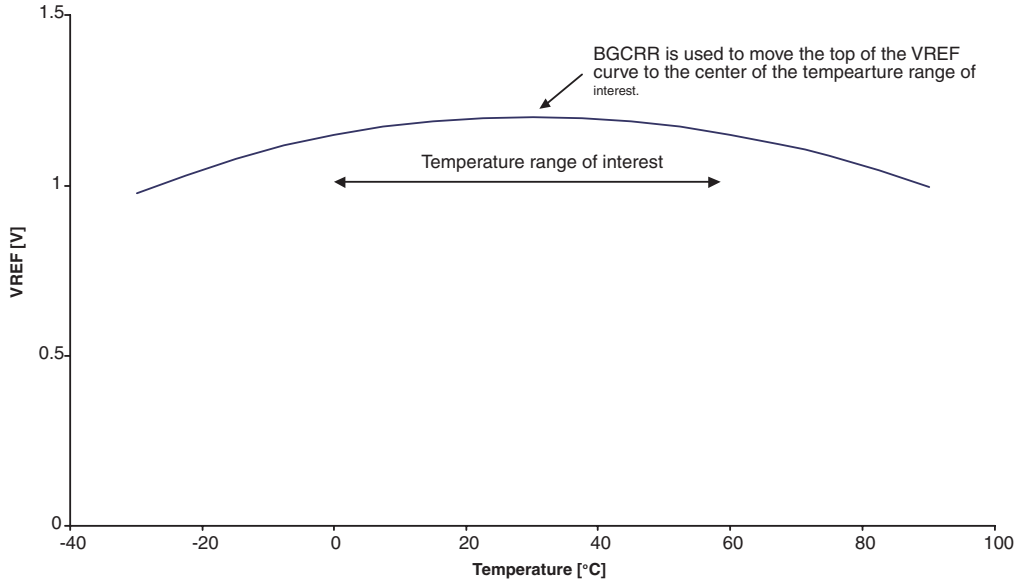
Bit	7	6	5	4	3	2	1	0	
	BGCR7	BGCR6	BGCR5	BGCR4	BGCR3	BGCR2	BGCR1	BGCR0	BGCRR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:0 – BGCR7:0: BG Calibration of Resistor ladder**

These bits are used for temperature gradient adjustment of the bandgap reference. Figure 20-2 illustrates VREF as a function of temperature. VREF has a positive temperature coefficient at low temperatures and negative temperature coefficient at high temperatures. Depending on the process variations, the top of the VREF curve may be located at higher or lower temperatures. To minimize the temperature drift in the temperature range of interest, BGCRR is used to adjust

the top of the curve towards the centre of the temperature range of interest. The BGCRR bits are temperature coded resulting in 9 possible settings: 00000000, 00000001, 00000011, 00000111, … , 11111111. The value 00000000 shifts the top of the VREF curve to the highest possible temperature, and the value 11111111 shifts the top of the VREF curve to the lowest possible temperature.

Figure 20-2. Illustration of VREF as a function of temperature.



21. Battery Protection

21.1 Features

- **Deep Under-voltage Protection**
- **Charge Over-current Protection**
- **Discharge Over-current Protection**
- **Short-circuit Protection**
- **Programmable and Lockable Detection Levels and Reaction Times**
- **Autonomous Operation Independent of CPU**

If the voltage at the VFET pin falls below the programmable Deep Under-voltage detection level, C-FET, PC-FET, and D-FET are disabled and the chip is set in Power-off mode to reduce power consumption to a minimum.

The Current Battery Protection circuitry (CBP) monitors the charge and discharge current and disables C-FET, PC-FET, and D-FET if an over-current or short-circuit condition is detected. There are three different programmable detection levels: Discharge Over-current Detection Level, Charge Over-current Detection Level and Short-circuit Detection Level. The external filter at the PI/NI input pins will cause too large delay for short-circuit detection. Therefore the separate PPI/NNI inputs are used for Current Battery Protection. There are two different programmable delays for activating Current Battery Protection: Short-circuit Reaction Time and Over-current Reaction Time. After Current Battery Protection has been activated, the application software must re-enable the FETs. The Battery Protection hardware provides a hold-off time of 1 second before software can re-enable the discharge FET. This provides safety in case the application software should unintentionally re-enable the discharge FET too early.

The activation of a protection also issues an interrupt to the CPU. The battery protection interrupts can be individually enabled and disabled by the CPU.

The effect of the various battery protection types is given in Table 21-1.

Table 21-1. Effect of Battery Protection Types

Battery Protection Type	Interrupt Requests	C-FET	D-FET	PC-FET	Cell Balancing FETs	MCU
Deep Under-voltage Detected	CPU Reset on exit	**Disabled**	**Disabled**	**Disabled**	**Disabled**	**Power-off**
Discharge Over-current Protection	Entry and exit	**Disabled**	**Disabled**	**Disabled**	Operational	Operational
Charge Over-current Protection	Entry and exit	**Disabled**	**Disabled**	**Disabled**	Operational	Operational
Short-circuit Protection	Entry and exit	**Disabled**	**Disabled**	**Disabled**	Operational	Operational

In order to reduce power consumption, both Short-circuit and Discharge Over-current Protection are automatically deactivated when the D-FET is disabled. The Charge Over-current Protection is disabled when both the C-FET and the PC-FET are disabled. Note however that Charge Over-current Protection is never automatically disabled when any of the C-FET or PC-FETs are controlled by PWM.

21.2 Deep Under-voltage Protection

The Deep Under-voltage Protection ensures that the battery cells will not be discharged deeper than the programmable Deep Under-voltage detection level. If the voltage at the VFET pin is below this level for a time longer than the programmable delay time, C-FET, PC-FET and D-FET are automatically switched off and the chip enters Power-off mode. The Deep Under-voltage Early Warning interrupt flag (DUVIF) will be set 250 ms before the chip enters Power-off. This will give the CPU a chance to take necessary actions before the power is switched off.

The device will remain in the Power-off mode until a charger is connected. When a charger is detected, a normal power-up sequence is started and the chip initializes to default state.

The Deep Under-voltage delay time and Deep Under-voltage detection level are set in the Battery Protection Deep Under-voltage Register (BPDUV). The Parameter Registers can be locked after the initial configuration, prohibiting any further updates until the next Hardware Reset.

Refer to ”Register Description for Battery Protection” on page 121 for register descriptions.

21.3 Discharge Over-current Protection

The Current Battery Protection (CBP) monitors the cell current by sampling the shunt resistor voltage at the PPI/NNI input pins. A differential operational amplifier amplifies the voltage with a suitable gain. The output from the operational amplifier is compared to an accurate, programmable On-chip voltage reference by an Analog Comparator. If the shunt resistor voltage is above the Discharge Over-current Detection level for a time longer than Over-current Protection Reaction Time, the chip activates Discharge Over-current Protection. A sampled system clocked by the internal ULP Oscillator is used for Over-current and Short-circuit Protection. This ensures a reliable clock source, off-set cancellation and low power consumption.

When the Discharge Over-current Protection is activated, the external D-FET, PC-FET, and C-FET are disabled and a Current Protection Timer is started. This timer ensures that the FETs are disabled for at least one second. The application software must then set the DFE and CFE bits in the FET Control and Status Register to re-enable normal operation. If the D-FET is re-enabled while the loading of the battery still is too large, the Discharge Over-current Protection will be activated again.

21.4 Charge Over-current Protection

If the voltage at the PPI/NNI pins is above the Charge Over-current Detection level for a time longer than Over-current Protection Reaction Time, the chip activates Charge Over-current Protection.

When the Charge Over-current Protection is activated, the external D-FET, PC-FET, and C-FET are disabled and a Current Protection Timer is started. This timer ensures that the FETs are disabled for at least one second. The application software must then set the DFE and CFE bits in the FET Control and Status Register to re-enable normal operation. If the C-FET is re-enabled and the charger continues to supply too high currents, the Charge Over-current Protection will be activated again.

21.5 Short-circuit Protection

A second level of high current detection is provided to enable a faster response time to very large discharge currents. If a discharge current larger than the Short-circuit Detection Level is

present for a period longer than Short-circuit Reaction Time, the Short-circuit Protection is activated.

When the Short-circuit Protection is activated, the external D-FET, PC-FET, and C-FET are disabled and a Current Protection Timer is started. This timer ensures that the D-FET, PC-FET, and C-FET are disabled for at least one second. The application software must then set the DFE and CFE bits in the FET Control and Status Register to re-enable normal operation. If the D-FET is re-enabled before the cause of the short-circuit condition is removed, the Short-circuit Protection will be activated again.

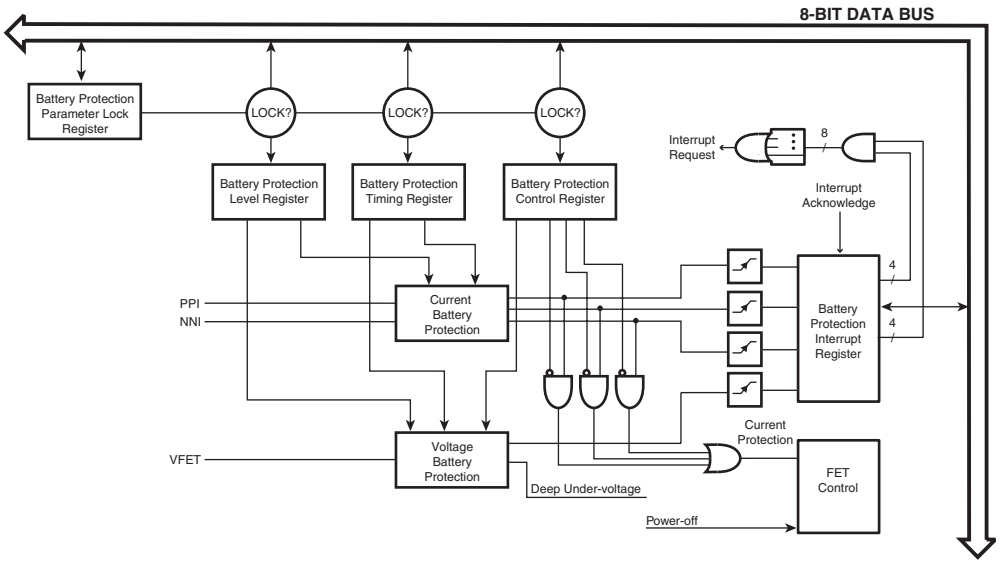
The Over-current and Short-circuit Protection parameters are programmable to adapt to different types of batteries. The parameters are set by writing to I/O Registers. The Parameter Registers can be locked after the initial configuration, prohibiting any further updates until the next Hardware Reset.

Refer to ”Register Description for Battery Protection” on page 121 for register descriptions.

21.6 Battery Protection CPU Interface

The Battery Protection CPU Interface is illustrated in Figure 21-1.

Figure 21-1. Battery Protection CPU Interface



Each protection has an Interrupt Flag. Each Flag can be read and cleared by the CPU, and each flag has an individual interrupt enable. All enabled flags are combined into a single battery protection interrupt request to the CPU. This interrupt can wake up the CPU from any operation mode, except Power-off. The interrupt flags are cleared by by writing a logig ‘1’ to their bit locations from the CPU.

Note that there are neither flags nor status bits indicating that the chip has entered the Power Off mode. This is because the CPU is powered down in this mode. The CPU will, however be able to detect that it came from a Power-off situation by monitoring CPU reset flags when it resumes operation.

21.7 Register Description for Battery Protection

The Battery Protection module operates in a different clock domain than the CPU. Whenever a new value is written to BPCR, BPDUV, BPOCD, BPSCD, or CPBTR, the value must be synchronized to the Battery Protection clock domain. Subsequent writes to this register should not be made during this synchronization. Therefore, after writing to one of these registers, the same register should not be re-written within the next 8 CPU clock periods. Note that each register is synchronized independently of the others.

21.7.1 Battery Protection Parameter Lock Register – BPPLR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	BPPLE	BPPL	BPPLR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:2 – Res: Reserved Bits**

These bits are reserved bits in the ATmega406 and will always read as zero.

- **Bit 1 – BPPLE: Battery Protection Parameter Lock Enable**

- **Bit 0 – BPPL: Battery Protection Parameter Lock**

The Battery Protection parameters set in the Battery Protection Parameter Registers and the disable function set in the Battery Protection Disable Register can be locked from any further software updates. Once locked, these registers cannot be accessed until the next hardware reset. This provides a safe method for protecting these registers from unintentional modification by software runaway. It is recommended that software sets these registers shortly after reset, and then protects these registers from any further updates.

To lock these registers, the following algorithm must be followed:

1. In the same operation, write a logic one to BPPLE and BPPL.
2. Within the next four clock cycles, in the same operation. write a logic zero to BPPLE and a logic one to BPPL.

The Battery Protection Parameter Registers are BPCR, CBPTR, BPOCP, BPSCD and BPDUV.

21.7.2 Battery Protection Control Register – BPCR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	DUVD	SCD	DCD	CCD	BPCR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:4 – Res: Reserved Bits**

These bits are reserved bits in the ATmega406 and will always read as zero.

- **Bit 3 – DUVD: Deep Under-voltage Protection Disable**

When the DUVD bit is set, the Deep Under-voltage Protection is disabled. The Deep Under-voltage Detection will be disabled, and any Deep Under-voltage condition will be ignored

- **Bit 2 – SCD: Short Circuit Protection Disabled**

When the SCD bit is set, the Short-circuit Protection is disabled. The Short-circuit Detection will be disabled, and any Short-circuit condition will be ignored.

- **Bit 1 – DCD: Discharge Over-current Protection Disable**

When the DCD bit is set, the Discharge Over-current Protection is disabled. The Discharge Over-current Detection will be disabled, and any Discharge Over-current condition will be ignored.

- **Bit 0 – CCD: Charge Over-current Protection Disable**

When the CCD bit is set, the Charge Over-current Protection is disabled. The Charge Over-current Detection will be disabled, and any Charge Over-current condition will be ignored.

21.7.3 Current Battery Protection Timing Register – CBPTR

Bit	7	6	5	4	3	2	1	0	
	SCPT[3:0]				OCPT[3:0]				CBPTR
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 7:4 – SCPT3:0: Short-circuit Protection Timing**

These bits control the delay of the Short-circuit Protection. See Table 21-2.

Table 21-2. SCPT[3:0] with Corresponding Short-circuit Delay Time

Short-circuit Protection Reaction Time							
SCPT[3:0]	**Typ**	**SCPT[3:0]**	**Typ**	**SCPT[3:0]**	**Typ**	**SCPT[3:0]**	**Typ**
0000	61 µs	0100	305 µs	1000	610 µs	1100	1098 µs
0001	122 µs	0101	366 µs	1001	732 µs	1101	1220 µs
0010	183 µs	0110	427 µs	1010	854 µs	1110	1342 µs
0011	244 µs	0111	488 µs	1011	976 µs	1111	1464 µs

- **Bit 3:0 – OCPT3:0: Over-current Protection Timing**

These bits control the delay of the Charge and Discharge Current Protection. See Table 21-3. Note that the same setting applies to both types of over-current protection.

Table 21-3. OCPT[3:0] with Corresponding Over-current Delay Time

Over-current Protection Reaction Time							
OCPT[3:0]	**Typ**	**OCPT[3:0]**	**Typ**	**OCPT[3:0]**	**Typ**	**OCPT[3:0]**	**Typ**
0000	1 ms	0100	8 ms	1000	16 ms	1100	24 ms
0001	2 ms	0101	10 ms	1001	18 ms	1101	26 ms
0010	4 ms	0110	12 ms	1010	20 ms	1110	28 ms
0011	6 ms	0111	14 ms	1011	22 ms	1111	30 ms

21.7.4 Battery Protection Over-current Detection Level Register – BPOCD

Bit	7	6	5	4	3	2	1	0	
	DCDL[3:0]				CCDL[3:0]				BPOCD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:4 – DCDL3:0: Discharge Over-current Detection Level**

These bits set the R_{SENSE} voltage level for detection of Discharge Over-current, as defined in Table 21-4.

Table 21-4. DCDL[3:0] with Corresponding R_{SENSE} Voltage for Discharge Over-current Detection Level

Discharge Over-current Protection Detection Level							
DCDL[3:0]	**Typ**	**DCDL[3:0]**	**Typ**	**DCDL[3:0]**	**Typ**	**DCDL[3:0]**	**Typ**
0000	0.050V	0100	0.070V	1000	0.110V	1100	0.160V
0001	0.055V	0101	0.080V	1001	0.120V	1101	0.180V
0010	0.060V	0110	0.090V	1010	0.130V	1110	0.200V
0011	0.065V	0111	0.100V	1011	0.140V	1111	0.220V

- **Bits 3:0 – CCDL3:0: Charge Over-current Detection Level**

These bits set the R_{SENSE} voltage level for detection of Charge Over-current, as defined in Table 21-5.

Table 21-5. CCDL[3:0] with Corresponding R_{SENSE} Voltage for Charge Over-current Detection Level

Charge Over-current Protection Detection Level							
CCDL[3:0]	**Typ**	**CCDL[3:0]**	**Typ**	**CCDL[3:0]**	**Typ**	**CCDL[3:0]**	**Typ**
0000	0.050V	0100	0.070V	1000	0.110V	1100	0.160V
0001	0.055V	0101	0.080V	1001	0.120V	1101	0.180V
0010	0.060V	0110	0.090V	1010	0.130V	1110	0.200V
0011	0.065V	0111	0.100V	1011	0.140V	1111	0.220V

21.7.5 Battery Protection Short-circuit Detection Level Register – BPSCD

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	SCDL[3:0]				BPSCD
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:4 – Res: Reserved Bits**

These bits are reserved bits in the ATmega406 and will always read as zero.

- **Bits 3:0 – SCDL3:0: Short-circuit Detection Level**

These bits set the RSENSE voltage level for detection of Short-circuit in the discharge direction, as defined in Table 21-6.

Table 21-6. SCDL[3:0] with Corresponding R_{SENSE} Voltage for Short-circuit Detection Level

Short-circuit Protection Detection Level							
SCDL[3:0]	**Typ**	**SCDL[3:0]**	**Typ**	**SCDL[3:0]**	**Typ**	**SCDL[3:0]**	**Typ**
0000	0.100V	0100	0.140V	1000	0.220V	1100	0.320V
0001	0.110V	0101	0.160V	1001	0.240V	1101	0.360V
0010	0.120V	0110	0.180V	1010	0.260V	1110	0.400V
0011	0.130V	0111	0.200V	1011	0.280V	1111	0.440V

21.7.6 Battery Protection Deep Under Voltage Register – BPDUV

Bit	7	6	5	4	3	2	1	0	
	–	–	DUVT[1:0]		DUDL[3:0]				BPDUV
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 bits in the ATmega406 and will always read as zero.

- **Bits 5:4 – DUVT1:0: Deep Under-voltage Timing**

These bits set the Deep Under-voltage Protection delay.

Table 21-7. DUVT[1:0] with Corresponding Deep Under-voltage Delay

DUVT1:0	Deep Under-voltage Delay
00	750 ms
01	1000 ms
10	1250 ms
11	1500 ms

- **Bits 3:0 – DUDL3:0: Deep Under-voltage Detection Level**

These bits set the Deep Under-voltage detection level.

Table 21-8. DUDL[3:0] with Corresponding Deep Under-voltage Detection Level

DUDL[3:0]	Typ	DUDL[3:0]	Typ
0000	4.71V	1000	7.23V
0001	5.03V	1001	7.54V
0010	5.34V	1010	7.86V
0011	5.66V	1011	8.17V
0100	5.97V	1100	8.49V
0101	6.29V	1101	8.80V
0110	6.60V	1110	9.11V
0111	6.91V	1111	9.43V

21.7.7 Battery Protection Interrupt Register – BPIR

Bit	7	6	5	4	3	2	1	0	
	DUVIF	COCIF	DOCIF	SCIF	DUVIE	COCIE	DOCIE	SCIE	BPIR
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 – DUVIF: Deep Under-voltage Early Warning Interrupt Flag**

If the voltage at VFET pin is below the Deep Under-voltage detection level and only 250 ms is left of the Deep Under-voltage delay, DUVIF becomes set. The flag must be cleared by writing a logical one to it.

- **Bit 6 – COCIF: Charge Over-current Protection Activated Interrupt Flag**

When the Charge Over-current Protection is activated, COCIF becomes set. The flag must be cleared by writing a logical one to it.

- **Bit 5 – DOCIF: Discharge Over-current Protection Activated Interrupt Flag**

When the Discharge Over-current Protection is activated, DOCIF becomes set. The flag must be cleared by writing a logical one to it.

- **Bit 4 – SCIF: Short-circuit Protection Activated Interrupt Flag**

When the Short-circuit Protection is activated, SCIF becomes set. The flag must be cleared by writing a logical one to it.

- **Bit 3 – DUVIE: Deep Under-voltage Early Warning Interrupt Enable**

The DUVIE bit enables interrupt caused by the Deep Under-voltage Early Warning Interrupt Flag

- **Bit 2 – COCIE: Charge Over-current Protection Activated Interrupt Enable**

The COCIE bit enables interrupt caused by the Charge Over-current Protection Activated Interrupt Flag.

- **Bit 1 – DOCIE: Discharge Over-current Protection Activated Interrupt Enable**

The DOCIE bit enables interrupt caused by the Discharge Over-current Protection Activated Interrupt Flag.

- **Bit 0 – SCIE: Short-circuit Protection Activated Interrupt Enable**

The SCIE bit enables interrupt caused by the Short-circuit Protection Activated Interrupt Flag.

If one of the Battery Protection Interrupt Flags is set, and the corresponding Interrupt Enable bit and the I-bit in the Status Register (SREG) are set, the MCU will jump to the Battery Protection interrupt vector. The application software must read the Battery Protection Interrupt Register to determine the cause of the interrupt. The interrupt flags will not be cleared when the interrupt routine is executed, they must be cleared by writing a logical one to them.

22. FET Control

In addition to the FET disable control signals from the battery protection circuitry, the CPU may disable the Charge FET (C-FET), the Discharge FET (D-FET), or both, by writing to the FET Control Register. Note that the CPU is never allowed to enable a FET that is disabled by the battery protection circuitry. The FET control is shown in Figure 22-1.

The PWM output from the 8-bit Timer/Counter0, OC0B, can be configured to drive the C-FET, Precharge FET (PC-FET) or both directly. This can be useful for controlling the charging of the battery cells. The PWM is configured by the COM0B1:0 and WGM02:0 bits in the TCCR0A/TCCR0B registers. Note that the OC0B pins does not need to be configured as an output. This means that the PWM output can be used to drive the C-FET and/or the PC-FET without occupying the OC0B-pin.

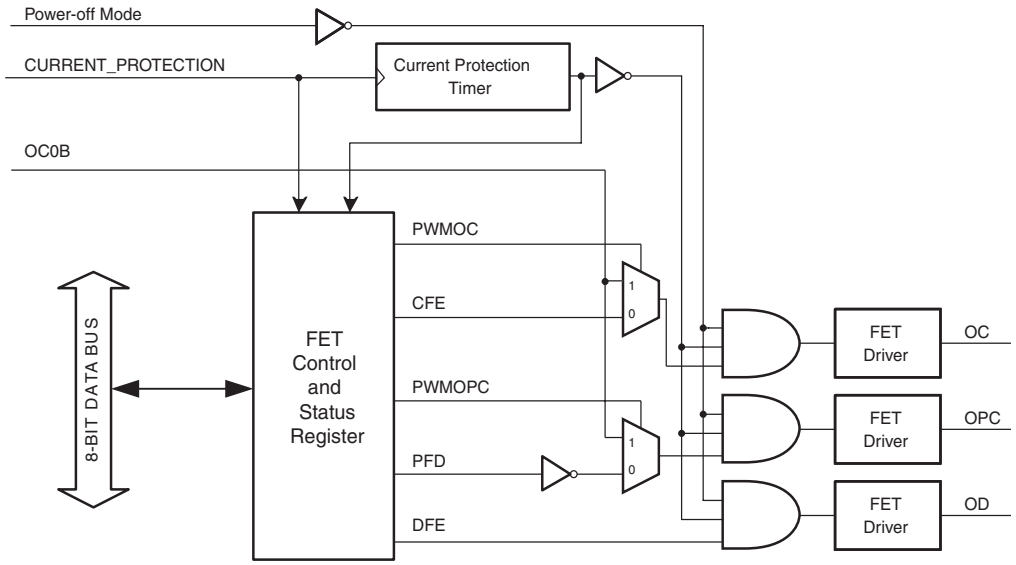
If C-FET is disabled and D-FET enabled, discharge current will run through the body-drain diode of the C-FET and vice versa. To avoid the potential heat problem from this situation, software must ensure that D-FET is not disabled when a charge current is flowing, and that C-FET is not disabled when a discharge current is flowing.

If the battery has been deeply discharged, large surge currents may result when a charger is connected. In this case, it is recommended to first pre charge the battery through a current limiting resistor. For this purpose, ATmega406 provides a Precharge FET (PC-FET) control output. This output is default enabled.

If ATmega406 has entered the Power-off mode, all FET control outputs will be disabled. When a charger is connected, the CPU will wake up. When waking up from Power-off mode, the C-FET and D-FET control outputs will remain disabled while PC-FET is default enabled. When the CPU detects that the cell voltages have risen enough to allow normal charging, it should enable the C-FET and D-FET control outputs and disable the PC-FET control output.

If the Current Battery Protection has been activated, the Current Protection Timer will ensure a hold-off time of 1 second before software can re-enable the external FETs.

Figure 22-1. FET Control Block Diagram



22.1 Register Description for FET Control

The FET Controller operates in a different clock domain than the CPU. Whenever a new value is written to the FCSR, the value must be synchronized to the FET Controller clock domain. Subsequent writes to this register should not be made during this synchronization. Therefore, after writing to this register, it should not be re-written within the next 8 CPU clock periods.

22.1.1 FET Control and Status Register – FCSR

Bit	7	6	5	4	3	2	1	0	
	–	–	PWMOC	PWMOPC	CPS	DFE	CFE	PFD	FCSR
Read/Write	R	R	R/W	R/W	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:6 – Res: Reserved Bits**

These bits are reserved bits in the ATmega406, and will always read as zero.

- **Bit 5 – PWMOC: Pulse Width Modulation of OC output**

When the PWMOC is cleared (zero), the CFE bit and the battery protection circuitry controls the OC output. When this bit is set (one), the OC output will be controlled by the PWM output from the 8-bit Timer/Counter0 and the battery protection circuitry.

- **Bit 4 – PWMOPC: Pulse Width Modulation of OPC output**

When the PWMOPC is cleared (zero), the PFD bit and the battery protection circuitry controls the OPC output. When this bit is set (one), the OC output will be controlled by the PWM output from the 8-bit Timer/Counter0 and the battery protection circuitry.

- **Bit 3 – CPS: Current Protection Status**

The CPS bit shows the status of the Current Protection. This bit is set (one) when the Current Protection Timer is activated, and is cleared (zero) when the hold-off time has elapsed.

- **Bit 2 – DFE: Discharge FET Enable**

When the DFE bit is cleared (zero), the Discharge FET will be disabled regardless of the state of the Battery Protection circuitry. When this bit is set (one), the Discharge FET state is determined by the Battery Protection circuitry. This bit will be cleared when CURRENT_PROTECTION is set (one).

- **Bit 1 – CFE: Charge FET Enable**

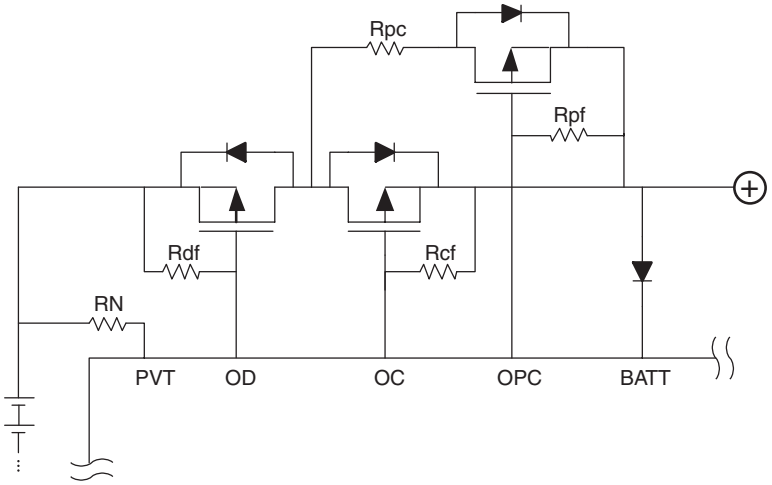
When the CFE bit is cleared (zero), the Charge FET will be disabled regardless of the state of the Battery Protection circuitry. When this bit is set (one), the Charge FET state is determined by the Battery Protection circuitry. This bit will be cleared when CURRENT_PROTECTION is set (one).

- **Bit 0 – PFD: Precharge FET Disable**

The PFD bit provides complete control of the Precharge FET. When the PFD bit is cleared (zero), the Precharge FET will be enabled. When the PFD bit is set (one), the Precharge FET will be disabled. This bit will be set when CURRENT_PROTECTION is set (one).

22.2 FET Driver

Figure 22-2. Connection of external FETs



The connection of external FETs to OD, OC, and OPC is shown in Figure 22-2.

When switching on an FET, the output pulls the gate quickly low to avoid heating of the FET. When the FET is switched completely on, the output changes operation mode in order to reduce current consumption. The gate-source voltage for the FET when switched on, $|V_{GS\_ON}|$, is limited to 13V ± 15%.

When disabling an external FET, the FET Driver output quickly pushes the gate voltage to the source pin potential, making the gate-source voltage of the FET close to zero. This disables the FET, and the FET Driver output switches operation mode to high impedance in order to reduce current consumption. The external resistor will keep the gate-source voltage at zero until the FET is enabled again and its gate is pulled low as explained above.

Table 22-1. Timing of FET Driver Outputs

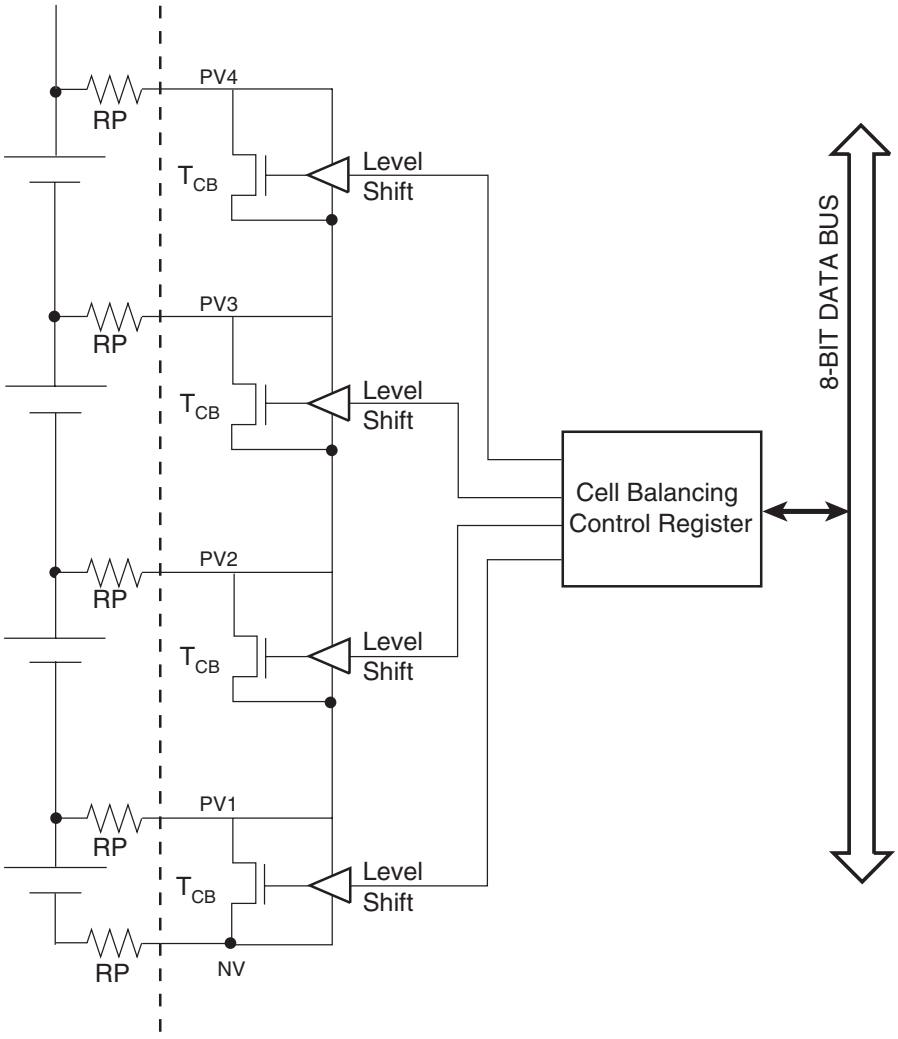
Parameter	Condition	Min	Typ	Max	Unit
$\|V_{GS\_ON}\|$		11	TBD	15	V
OC/OD Rise time (10 - 90%) (Switching OFF)	CL = 10 nF		10	50	µs
OC/OD Fall time (V_{GS} = 0 - V_{GS} = -5V) (Switching ON)	CL = 10 nF		TBD	100	µs
OPC Rise time (10 - 90%) (Switching OFF)	CL = 1 nF		100	500	µs
OPC Fall time (V_{GS} = 0 - V_{GS} = -5V) (Switching ON)	CL = 1 nF		100	500	µs

23. Cell Balancing

ATmega406 incorporates cell balancing FETs. The chip provides one cell balancing FET for each battery cell in series. The FETs are directly controlled by the application software, allowing the cell balancing algorithms to be implemented in software. The FETs are connected in parallel with the individual battery cells. The cell balancing is illustrated in Figure 23-1. The figure shows a four-cell configuration. The cell balancing FETs are disabled in the Power-off mode.

Typical current through the Cell Balancing FETs (T_{CB}) is 2 mA. The Cell Balancing FETs are controlled by the CBCR. Neighbouring FETs cannot be simultaneously enabled. If trying to enable two neighbouring FETs, both will be disabled.

Figure 23-1. Cell Balancing



23.0.1 Cell Balancing Control Register – CBCR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	CBE4	CBE3	CBE2	CBE1	CBCR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:4 – Res: Reserved Bits**

These bits are reserved bits in the ATmega406 and will always read as zero.

- **Bit 3 – CBE4: Cell Balancing Enable 4**

When this bit is set, the integrated Cell Balancing FET between terminals PV4 and PV3 will be enabled. When the bit is cleared, the Cell Balancing FET will be disabled. The Cell Balancing FETs are always disabled in Power-off mode. CBE4 cannot be set if CBE3 is set.

- **Bit 2 – CBE3: Cell Balancing Enable 3**

When this bit is set, the integrated Cell Balancing FET between terminals PV3 and PV2 will be enabled. When the bit is cleared, the Cell Balancing FET will be disabled. The Cell Balancing FETs are always disabled in Power-off mode. CBE3 cannot be set if CBE2 or CBE4 is set.

- **Bit 1 – CBE2: Cell Balancing Enable 2**

When this bit is set, the integrated Cell Balancing FET between terminals PV2 and PV1 will be enabled. When the bit is cleared, the Cell Balancing FET will be disabled. The Cell Balancing FETs are always disabled in Power-off mode. CBE2 cannot be set if CBE1 or CBE3 is set.

- **Bit 0 – CBE1: Cell Balancing Enable 1**

When this bit is set (one), the integrated Cell Balancing FET between terminals PV1 and NV will be enabled. When the bit is cleared (zero), the Cell Balancing FET will be disabled. The Cell Balancing FETs are always disabled in Power-off mode. CBE1 cannot be set if CBE2 is set.

24. 2-wire Serial Interface

24.1 Features

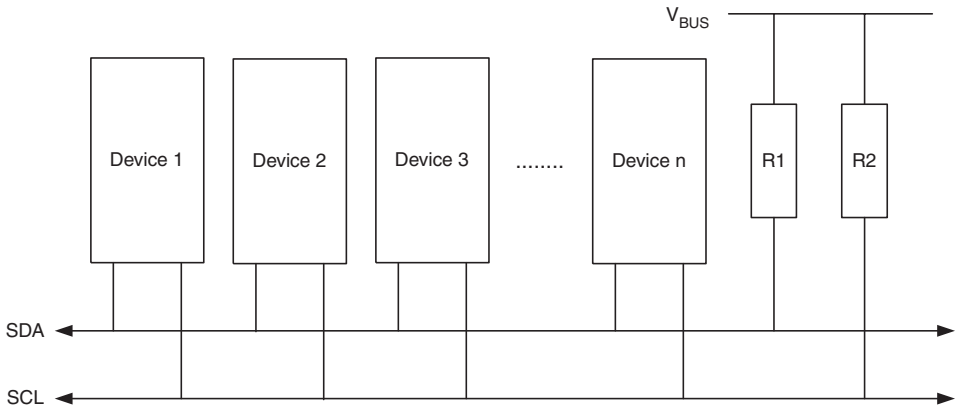
- **Simple yet Powerful and Flexible Communication Interface, Only Two Bus Lines Needed**
- **Both Master and Slave Operation Supported**
- **Device can Operate as Transmitter or Receiver**
- **7-bit Address Space allows up to 128 Different Slave Addresses**
- **Multi-master Arbitration Support**
- **Operates on 4 MHz Clock, achieving up to 100 kHz Data Transfer Speed**
- **Slew-rate Limited Output Drivers**
- **Noise Suppression Circuitry Rejects Spikes on Bus Lines**
- **Fully Programmable Slave Address with General Call Support**
- **Address Recognition Causes Wake-up when AVR is in Sleep Mode**

24.2 Two-wire Serial Interface Bus Definition

The Two-wire Serial Interface (TWI) is ideally suited for typical microcontroller applications. The TWI protocol allows the systems designer to interconnect up to 128 different devices using only two bi-directional bus lines, one for clock (SCL) and one for data (SDA). The only external hardware needed to implement the bus is a single pull-up resistor for each of the TWI bus lines. All devices connected to the bus have individual addresses, and mechanisms for resolving bus contention are inherent in the TWI protocol.

The PRTWI bit in ”Power Reduction Register 0 - PRR0” on page 35 must be written to zero to enable TWI module.

Figure 24-1. TWI Bus Interconnection



24.2.1 TWI Terminology

The following definitions are frequently encountered in this section.

Table 24-1. TWI Terminology

Term	Description
Master	The device that initiates and terminates a transmission. The Master also generates the SCL clock.
Slave	The device addressed by a Master.
Transmitter	The device placing data on the bus.
Receiver	The device reading data from the bus.

24.2.2 Electrical Interconnection

As depicted in Figure 24-1, both bus lines are connected to the positive supply voltage through pull-up resistors. The bus drivers of all TWI-compliant devices are open-drain or open-collector. This implements a wired-AND function which is essential to the operation of the interface. A low level on a TWI bus line is generated when one or more TWI devices output a zero. A high level is output when all TWI devices tri-state their outputs, allowing the pull-up resistors to pull the line high. Note that all AVR devices connected to the TWI bus must be powered in order to allow any bus operation.

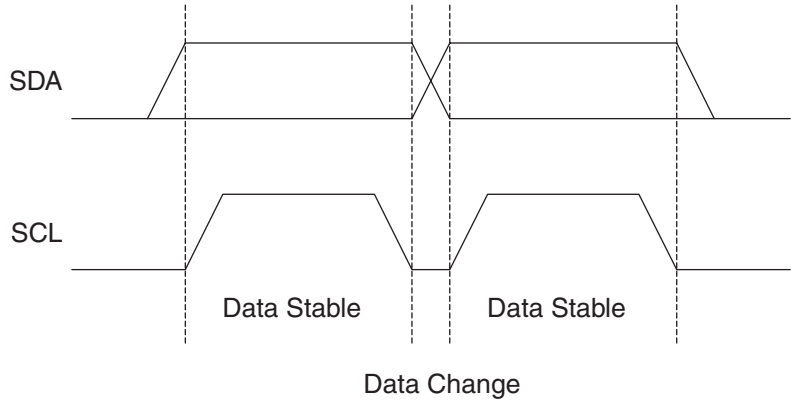
The number of devices that can be connected to the bus is only limited by the bus capacitance limit of 400 pF and the 7-bit slave address space. A detailed specification of the electrical characteristics of the TWI is given in ”2-wire Serial Interface Characteristics” on page 213.

24.3 Data Transfer and Frame Format

24.3.1 Transferring Bits

Each data bit transferred on the TWI bus is accompanied by a pulse on the clock line. The level of the data line must be stable when the clock line is high. The only exception to this rule is for generating start and stop conditions.

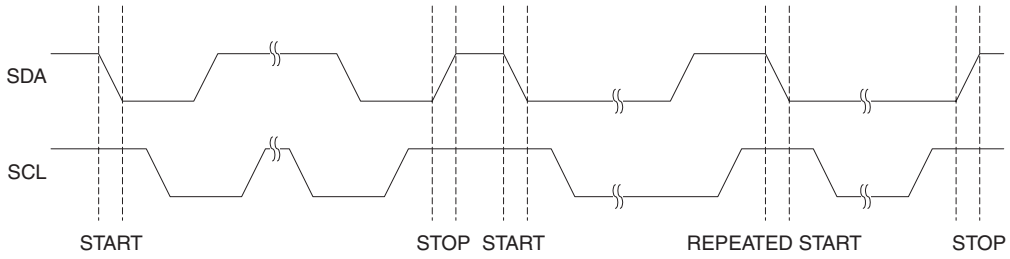
Figure 24-2. Data Validity



24.3.2 START and STOP Conditions

The Master initiates and terminates a data transmission. The transmission is initiated when the Master issues a START condition on the bus, and it is terminated when the Master issues a STOP condition. Between a START and a STOP condition, the bus is considered busy, and no other Master should try to seize control of the bus. A special case occurs when a new START condition is issued between a START and STOP condition. This is referred to as a REPEATED START condition, and is used when the Master wishes to initiate a new transfer without relinquishing control of the bus. After a REPEATED START, the bus is considered busy until the next STOP. This is identical to the START behavior, and therefore START is used to describe both START and REPEATED START for the remainder of this datasheet, unless otherwise noted. As depicted below, START and STOP conditions are signalled by changing the level of the SDA line when the SCL line is high.

Figure 24-3. START, REPEATED START, and STOP Conditions



24.3.3 Address Packet Format

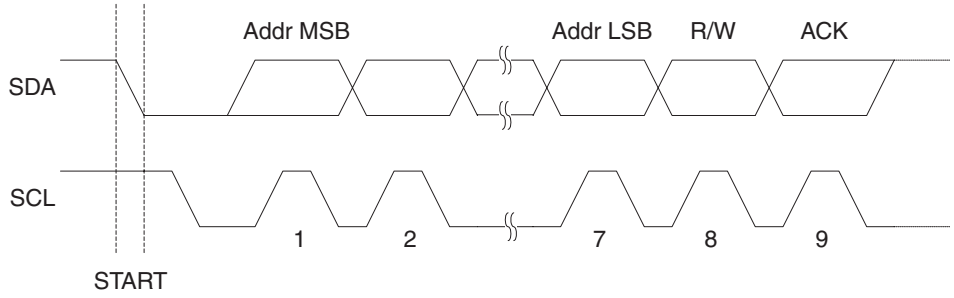
All address packets transmitted on the TWI bus are nine bits long, consisting of seven address bits, one READ/WRITE control bit and an acknowledge bit. If the READ/WRITE bit is set, a read operation is to be performed, otherwise a write operation should be performed. When a slave recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle. If the addressed Slave is busy, or for some other reason can not service the Master's request, the SDA line should be left high in the ACK clock cycle. The Master can then transmit a STOP condition, or a REPEATED START condition to initiate a new transmission. An address packet consisting of a slave address and a READ or a WRITE bit is called SLA+R or SLA+W, respectively.

The MSB of the address byte is transmitted first. Slave addresses can freely be allocated by the designer, but the address 0000 000 is reserved for a general call.

When a general call is issued, all slaves should respond by pulling the SDA line low in the ACK cycle. A general call is used when a Master wishes to transmit the same message to several slaves in the system. When the general call address followed by a write bit is transmitted on the bus, all slaves set up to acknowledge the general call will pull the SDA line low in the ack cycle. The following data packets will then be received by all the slaves that acknowledged the general call. Note that transmitting the general call address followed by a Read bit is meaningless, as this would cause contention if several slaves started transmitting different data.

All addresses of the format 1111 xxx should be reserved for future purposes.

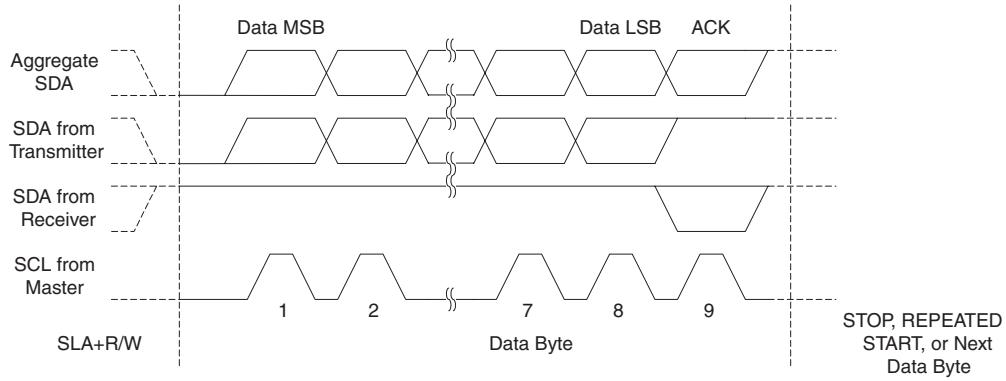
Figure 24-4. Address Packet Format



24.3.4 Data Packet Format

All data packets transmitted on the TWI bus are nine bits long, consisting of one data byte and an acknowledge bit. During a data transfer, the Master generates the clock and the START and STOP conditions, while the Receiver is responsible for acknowledging the reception. An Acknowledge (ACK) is signalled by the Receiver pulling the SDA line low during the ninth SCL cycle. If the Receiver leaves the SDA line high, a NACK is signalled. When the Receiver has received the last byte, or for some reason cannot receive any more bytes, it should inform the Transmitter by sending a NACK after the final byte. The MSB of the data byte is transmitted first.

Figure 24-5. Data Packet Format

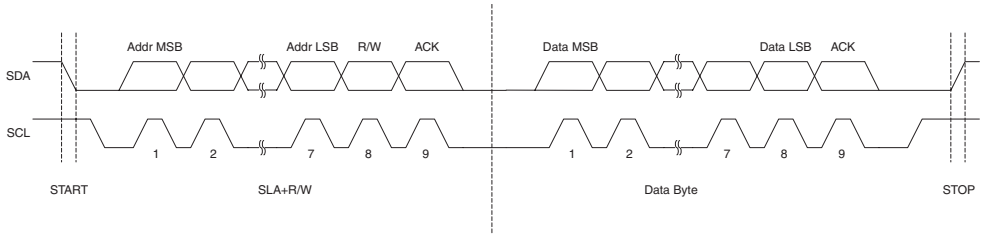


24.3.5 Combining Address and Data Packets Into a Transmission

A transmission basically consists of a START condition, a SLA+R/W, one or more data packets and a STOP condition. An empty message, consisting of a START followed by a STOP condition, is illegal. Note that the wired-ANDing of the SCL line can be used to implement handshaking between the Master and the Slave. The Slave can extend the SCL low period by pulling the SCL line low. This is useful if the clock speed set up by the Master is too fast for the Slave, or the Slave needs extra time for processing between the data transmissions. The Slave extending the SCL low period will not affect the SCL high period, which is determined by the Master. As a consequence, the Slave can reduce the TWI data transfer speed by prolonging the SCL duty cycle.

Figure 24-6 shows a typical data transmission. Note that several data bytes can be transmitted between the SLA+R/W and the STOP condition, depending on the software protocol implemented by the application software.

Figure 24-6. Typical Data Transmission



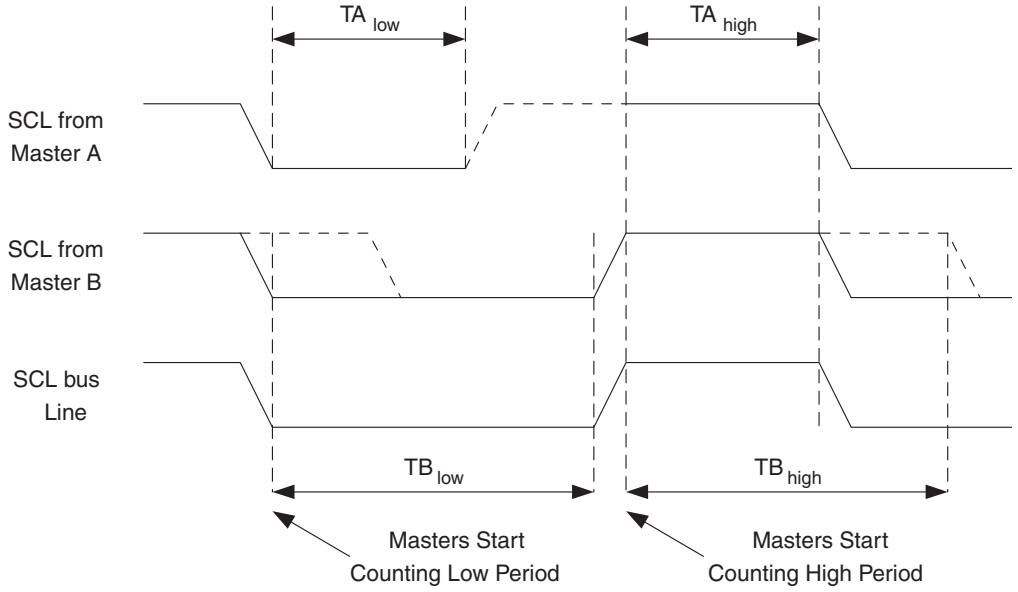
24.4 Multi-master Bus Systems, Arbitration and Synchronization

The TWI protocol allows bus systems with several masters. Special concerns have been taken in order to ensure that transmissions will proceed as normal, even if two or more masters initiate a transmission at the same time. Two problems arise in multi-master systems:

- An algorithm must be implemented allowing only one of the masters to complete the transmission. All other masters should cease transmission when they discover that they have lost the selection process. This selection process is called arbitration. When a contending master discovers that it has lost the arbitration process, it should immediately switch to Slave mode to check whether it is being addressed by the winning master. The fact that multiple masters have started transmission at the same time should not be detectable to the slaves (i.e., the data being transferred on the bus must not be corrupted).
- Different masters may use different SCL frequencies. A scheme must be devised to synchronize the serial clocks from all masters, in order to let the transmission proceed in a lockstep fashion. This will facilitate the arbitration process.

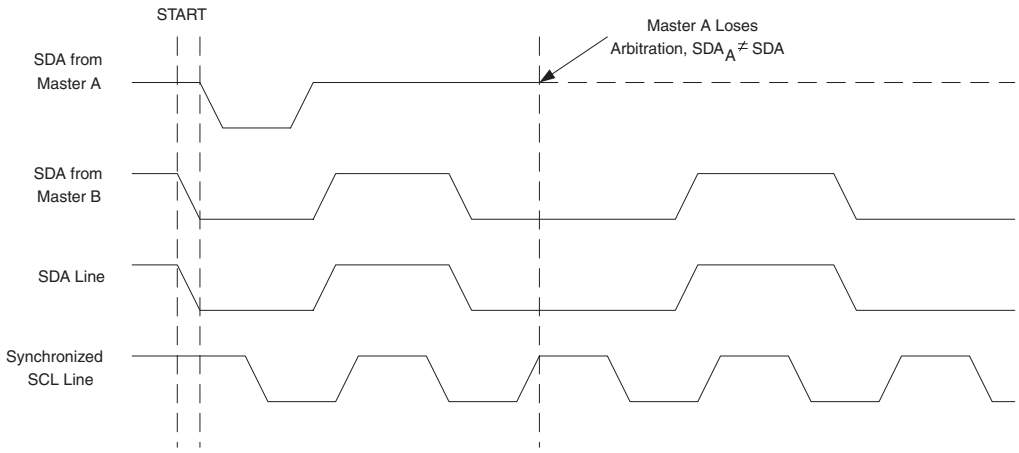
The wired-ANDing of the bus lines is used to solve both these problems. The serial clocks from all masters will be wired-ANDed, yielding a combined clock with a high period equal to the one from the master with the shortest high period. The low period of the combined clock is equal to the low period of the master with the longest low period. Note that all masters listen to the SCL line, effectively starting to count their SCL high and low Time-out periods when the combined SCL line goes high or low, respectively.

Figure 24-7. SCL Synchronization between Multiple Masters



Arbitration is carried out by all masters continuously monitoring the SDA line after outputting data. If the value read from the SDA line does not match the value the master had output, it has lost the arbitration. Note that a master can only lose arbitration when it outputs a high SDA value while another master outputs a low value. The losing master should immediately go to Slave mode, checking if it is being addressed by the winning master. The SDA line should be left high, but losing masters are allowed to generate a clock signal until the end of the current data or address packet. Arbitration will continue until only one master remains, and this may take many bits. If several masters are trying to address the same slave, arbitration will continue into the data packet.

Figure 24-8. Arbitration between Two Masters



Note that arbitration is not allowed between:

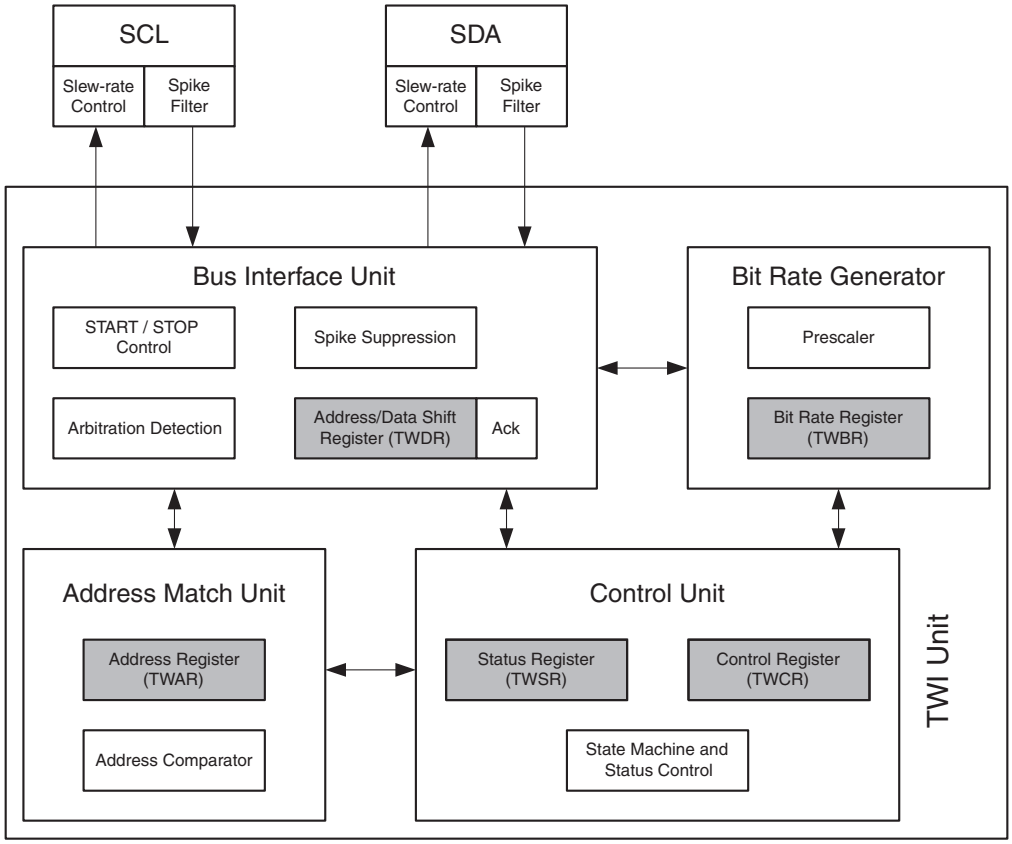
- A REPEATED START condition and a data bit.
- A STOP condition and a data bit.
- A REPEATED START and a STOP condition.

It is the user software's responsibility to ensure that these illegal arbitration conditions never occur. This implies that in multi-master systems, all data transfers must use the same composition of SLA+R/W and data packets. In other words: All transmissions must contain the same number of data packets, otherwise the result of the arbitration is undefined.

24.5 Overview of the TWI Module

The TWI module is comprised of several submodules, as shown in Figure 24-9. The shaded registers are accessible through the AVR data bus.

Figure 24-9. Overview of the TWI Module



24.5.1 SCL and SDA Pins

These pins interface the AVR TWI with the rest of the MCU system. The output drivers contain a slew-rate limiter in order to conform to the TWI specification. The input stages contain a spike suppression unit removing spikes shorter than 50 ns.

24.5.2 Bit Rate Generator Unit

This unit controls the period of SCL when operating in a Master mode. The SCL period is controlled by settings in the TWI Bit Rate Register (TWBR) and the Prescaler bits in the TWI Status Register (TWSR). Slave operation does not depend on Bit Rate or Prescaler settings, but the CPU clock frequency in the slave must be at least 16 times higher than the SCL frequency. Note

that slaves may prolong the SCL low period, thereby reducing the average TWI bus clock period. The SCL frequency is generated according to the following equation:

$$\text{SCL frequency} = \frac{\text{TWI Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{TWPS}}$$

- TWBR = Value of the TWI Bit Rate Register.
- TWPS = Value of the prescaler bits in the TWI Status Register.

Notes: 1. TWBR should be 10 or higher if the TWI operates in Master mode. If TWBR is lower than 10, the master may produce an incorrect output on SDA and SCL for the reminder of the byte. The problem occurs when operating the TWI in Master mode, sending Start + SLA + R/W to a slave (a slave does not need to be connected to the bus for the condition to happen).
2. The TWI clock is 4 MHz, see "Calibrated Fast RC Oscillator" on page 26.

24.5.3 Bus Interface Unit

This unit contains the Data and Address Shift Register (TWDR), a START/STOP Controller and Arbitration detection hardware. The TWDR contains the address or data bytes to be transmitted, or the address or data bytes received. In addition to the 8-bit TWDR, the Bus Interface Unit also contains a register containing the (N)ACK bit to be transmitted or received. This (N)ACK Register is not directly accessible by the application software. However, when receiving, it can be set or cleared by manipulating the TWI Control Register (TWCR). When in Transmitter mode, the value of the received (N)ACK bit can be determined by the value in the TWSR.

The START/STOP Controller is responsible for generation and detection of START, REPEATED START, and STOP conditions. The START/STOP controller is able to detect START and STOP conditions even when the AVR MCU is in one of the sleep modes, enabling the MCU to wake up if addressed by a Master.

If the TWI has initiated a transmission as Master, the Arbitration Detection hardware continuously monitors the transmission trying to determine if arbitration is in process. If the TWI has lost an arbitration, the Control Unit is informed. Correct action can then be taken and appropriate status codes generated.

24.5.4 Address Match Unit

The Address Match unit checks if received address bytes match the 7-bit address in the TWI Address Register (TWAR). If the TWI General Call Recognition Enable (TWGCE) bit in the TWAR is written to one, all incoming address bits will also be compared against the General Call address. Upon an address match, the Control unit is informed, allowing correct action to be taken. The TWI may or may not acknowledge its address, depending on settings in the TWCR. The Address Match unit is able to compare addresses even when the AVR MCU is in sleep mode, enabling the MCU to wake-up if addressed by a Master.

24.5.5 Control Unit

The Control unit monitors the TWI bus and generates responses corresponding to settings in the TWI Control Register (TWCR). When an event requiring the attention of the application occurs on the TWI bus, the TWI Interrupt Flag (TWINT) is asserted. In the next clock cycle, the TWI Status Register (TWSR) is updated with a status code identifying the event. The TWSR only contains relevant status information when the TWI interrupt flag is asserted. At all other times, the TWSR contains a special status code indicating that no relevant status information is available. As long as the TWINT flag is set, the SCL line is held low. This allows the application software to complete its tasks before allowing the TWI transmission to continue.

The TWINT flag is set in the following situations:

- After the TWI has transmitted a START/REPEATED START condition.
- After the TWI has transmitted SLA+R/W.
- After the TWI has transmitted an address byte.
- After the TWI has lost arbitration.
- After the TWI has been addressed by own slave address or general call.
- After the TWI has received a data byte.
- After a STOP or REPEATED START has been received while still addressed as a Slave.
- When a bus error has occurred due to an illegal START or STOP condition.

24.6 TWI Register Description

24.6.1 TWI Bit Rate Register – TWBR

Bit	7	6	5	4	3	2	1	0	
	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:0 – TWI Bit Rate Register**

TWBR selects the division factor for the bit rate generator. The bit rate generator is a frequency divider which generates the SCL clock frequency in the Master modes. See ”Bit Rate Generator Unit” on page 137 for calculating bit rates.

24.6.2 TWI Control Register – TWCR

Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The TWCR is used to control the operation of the TWI. It is used to enable the TWI, to initiate a Master access by applying a START condition to the bus, to generate a Receiver acknowledge, to generate a stop condition, and to control halting of the bus while the data to be written to the bus are written to the TWDR. It also indicates a write collision if data is attempted written to TWDR while the register is inaccessible.

- **Bit 7 – TWINT: TWI Interrupt Flag**

This bit is set by hardware when the TWI has finished its current job and expects application software response. If the I-bit in SREG and TWIE in TWCR are set, the MCU will jump to the TWI Interrupt Vector. While the TWINT flag is set, the SCL low period is stretched. The TWINT flag must be cleared by software by writing a logic one to it. Note that this flag is not automatically cleared by hardware when executing the interrupt routine. Also note that clearing this flag starts the operation of the TWI, so all accesses to the TWI Address Register (TWAR), TWI Status Register (TWSR), and TWI Data Register (TWDR) must be complete before clearing this flag.

- **Bit 6 – TWEA: TWI Enable Acknowledge Bit**

The TWEA bit controls the generation of the acknowledge pulse. If the TWEA bit is written to one, the ACK pulse is generated on the TWI bus if the following conditions are met:

1. The device’s own slave address has been received.
2. A general call has been received, while the TWGCE bit in the TWAR is set.
3. A data byte has been received in Master Receiver or Slave Receiver mode.

By writing the TWEA bit to zero, the device can be virtually disconnected from the Two-wire Serial Bus temporarily. Address recognition can then be resumed by writing the TWEA bit to one again.

- **Bit 5 – TWSTA: TWI START Condition Bit**

The application writes the TWSTA bit to one when it desires to become a Master on the Two-wire Serial Bus. The TWI hardware checks if the bus is available, and generates a START con-

dition on the bus if it is free. However, if the bus is not free, the TWI waits until a STOP condition is detected, and then generates a new START condition to claim the Bus Master status. TWSTA is cleared by the TWI hardware when the START condition has been transmitted.

- **Bit 4 – TWSTO: TWI STOP Condition Bit**

Writing the TWSTO bit to one in Master mode will generate a STOP condition on the Two-wire Serial Bus. When the STOP condition is executed on the bus, the TWSTO bit is cleared automatically. In Slave mode, setting the TWSTO bit can be used to recover from an error condition. This will not generate a STOP condition, but the TWI returns to a well-defined unaddressed Slave mode and releases the SCL and SDA lines to a high impedance state.

- **Bit 3 – TWWC: TWI Write Collision Flag**

The TWWC bit is set when attempting to write to the TWI Data Register – TWDR when TWINT is low. This flag is cleared by writing the TWDR Register when TWINT is high.

- **Bit 2 – TWEN: TWI Enable Bit**

The TWEN bit enables TWI operation and activates the TWI interface. When TWEN is written to one, the TWI takes control over the I/O pins connected to the SCL and SDA pins, enabling the slew-rate limiters and spike filters. If this bit is written to zero, the TWI is switched off and all TWI transmissions are terminated, regardless of any ongoing operation.

- **Bit 1 – Res: Reserved Bit**

This bit is a reserved bit and will always read as zero.

- **Bit 0 – TWIE: TWI Interrupt Enable**

When this bit is written to one, and the I-bit in SREG is set, the TWI interrupt request will be activated for as long as the TWINT flag is high.

24.6.3 TWI Status Register – TWSR

Bit	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

- **Bits 7:3 – TWS: TWI Status**

These five bits reflect the status of the TWI logic and the Two-wire Serial Bus. The different status codes are described in Table 24-3 on page 148 through Table 24-6 on page 158. Note that the value read from TWSR contains both the 5-bit status value and the 2-bit prescaler value. The application designer should mask the prescaler bits to zero when checking the status bits. This makes status checking independent of prescaler setting. This approach is used in this datasheet, unless otherwise noted.

- **Bit 2 – Res: Reserved Bit**

This bit is reserved and will always read as zero.

- **Bits 1:0 – TWPS: TWI Prescaler Bits**

These bits can be read and written, and control the bit rate prescaler.

Table 24-2. TWI Bit Rate Prescaler

TWPS1	TWPS0	Prescaler Value
0	0	1
0	1	4
1	0	16
1	1	64

To calculate bit rates, see ”Bit Rate Generator Unit” on page 137. The value of TWPS1:0 is used in the equation.

24.6.4 TWI Data Register – TWDR

Bit	7	6	5	4	3	2	1	0	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

In Transmit mode, TWDR contains the next byte to be transmitted. In Receive mode, the TWDR contains the last byte received. It is writable while the TWI is not in the process of shifting a byte. This occurs when the TWI Interrupt Flag (TWINT) is set by hardware. Note that the data register cannot be initialized by the user before the first interrupt occurs. The data in TWDR remains stable as long as TWINT is set. While data is shifted out, data on the bus is simultaneously shifted in. TWDR always contains the last byte present on the bus, except after a wake-up from a sleep mode by the TWI interrupt. In this case, the contents of TWDR is undefined. In the case of a lost bus arbitration, no data is lost in the transition from Master to Slave. Handling of the ACK bit is controlled automatically by the TWI logic, the CPU cannot access the ACK bit directly.

- **Bits 7:0 – TWD: TWI Data Register**

These eight bits constitute the next data byte to be transmitted, or the latest data byte received on the Two-wire Serial Bus.

24.6.5 TWI (Slave) Address Register – TWAR

Bit	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

The TWAR should be loaded with the 7-bit slave address (in the seven most significant bits of TWAR) to which the TWI will respond when programmed as a slave transmitter or Receiver, and not needed in the Master modes. In multi-master systems, TWAR must be set in masters which can be addressed as slaves by other masters.

The LSB of TWAR is used to enable recognition of the general call address (0x00). There is an associated address comparator that looks for the slave address (or general call address if enabled) in the received serial address. If a match is found, an interrupt request is generated.

- **Bits 7:1 – TWA: TWI (Slave) Address Register**

These seven bits constitute the slave address of the TWI unit.

- **Bit 0 – TWGCE: TWI General Call Recognition Enable Bit**

If set, this bit enables the recognition of a General Call given over the Two-wire Serial Bus.

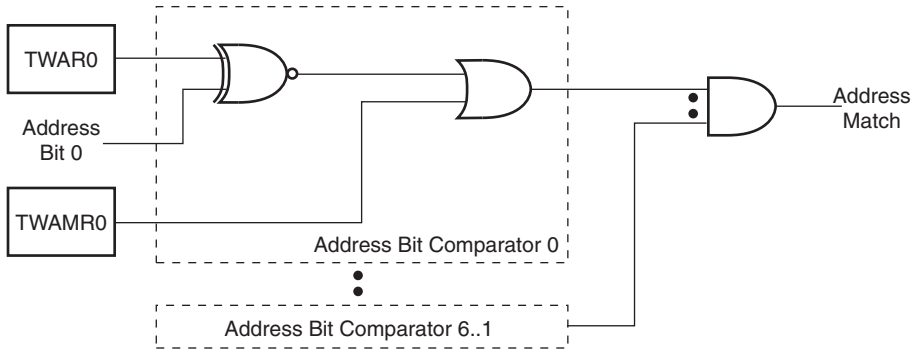
24.6.6 TWI (Slave) Address Mask Register – TWAMR

Bit	7	6	5	4	3	2	1	0	
	TWAM[6:0]							–	TWAMR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:1 – TWAM: TWI Address Mask**

The TWAMR can be loaded with a 7-bit Slave Address mask. Each of the bits in TWAMR can mask (disable) the corresponding address bits in the TWI Address Register (TWAR). If the mask bit is set to one then the address match logic ignores the compare between the incoming address bit and the corresponding bit in TWAR. Figure 24-10 shown the address match logic in detail.

Figure 24-10. TWI Address Match Logic, Block Diagram



- **Bit 0 – Res: Reserved Bit**

This bit is an unused bit in the ATmega406, and will always read as zero.

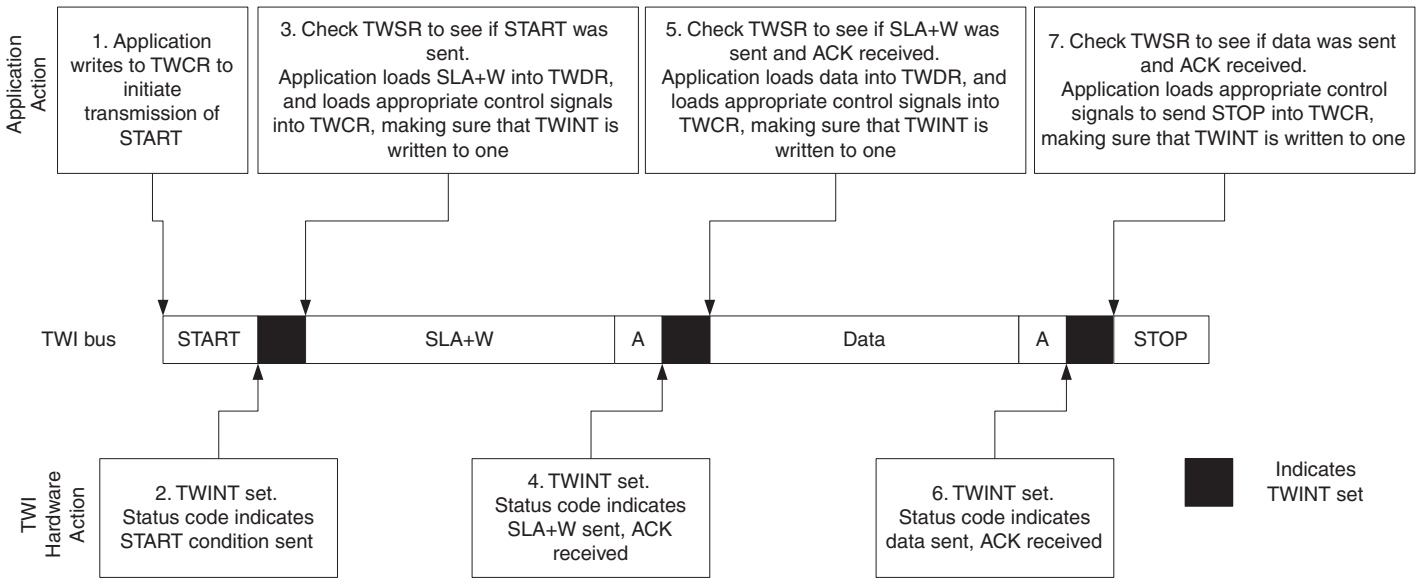
24.7 Using the TWI

The AVR TWI is byte-oriented and interrupt based. Interrupts are issued after all bus events, like reception of a byte or transmission of a START condition. Because the TWI is interrupt-based, the application software is free to carry on other operations during a TWI byte transfer. Note that the TWI Interrupt Enable (TWIE) bit in TWCR together with the Global Interrupt Enable bit in SREG allow the application to decide whether or not assertion of the TWINT flag should generate an interrupt request. If the TWIE bit is cleared, the application must poll the TWINT flag in order to detect actions on the TWI bus.

When the TWINT flag is asserted, the TWI has finished an operation and awaits application response. In this case, the TWI Status Register (TWSR) contains a value indicating the current state of the TWI bus. The application software can then decide how the TWI should behave in the next TWI bus cycle by manipulating the TWCR and TWDR registers.

Figure 24-11 is a simple example of how the application can interface to the TWI hardware. In this example, a Master wishes to transmit a single data byte to a Slave. This description is quite abstract, a more detailed explanation follows later in this section. A simple code example implementing the desired behavior is also presented.

Figure 24-11. Interfacing the Application to the TWI in a Typical Transmission



1. The first step in a TWI transmission is to transmit a START condition. This is done by writing a specific value into TWCR, instructing the TWI hardware to transmit a START condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the START condition.
2. When the START condition has been transmitted, the TWINT flag in TWCR is set, and TWSR is updated with a status code indicating that the START condition has successfully been sent.
3. The application software should now examine the value of TWSR, to make sure that the START condition was successfully transmitted. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load SLA+W into TWDR. Remember that TWDR is used both for address and data. After TWDR has been loaded with the desired SLA+W, a specific value must be written to TWCR, instructing the TWI hardware to transmit the SLA+W present in TWDR. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the address packet.
4. When the address packet has been transmitted, the TWINT flag in TWCR is set, and TWSR is updated with a status code indicating that the address packet has successfully been sent. The status code will also reflect whether a slave acknowledged the packet or not.
5. The application software should now examine the value of TWSR, to make sure that the address packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load a data packet into TWDR. Subsequently, a specific value must be written to TWCR, instructing the TWI hardware to transmit the data packet present in TWDR. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will

not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the data packet.

6. When the data packet has been transmitted, the TWINT flag in TWCR is set, and TWSR is updated with a status code indicating that the data packet has successfully been sent. The status code will also reflect whether a slave acknowledged the packet or not.
7. The application software should now examine the value of TWSR, to make sure that the data packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must write a specific value to TWCR, instructing the TWI hardware to transmit a STOP condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the STOP condition. Note that TWINT is NOT set after a STOP condition has been sent.

Even though this example is simple, it shows the principles involved in all TWI transmissions. These can be summarized as follows:

- When the TWI has finished an operation and expects application response, the TWINT flag is set. The SCL line is pulled low until TWINT is cleared.
- When the TWINT flag is set, the user must update all TWI registers with the value relevant for the next TWI bus cycle. As an example, TWDR must be loaded with the value to be transmitted in the next bus cycle.
- After all TWI Register updates and other pending application software tasks have been completed, TWCR is written. When writing TWCR, the TWINT bit should be set. Writing a one to TWINT clears the flag. The TWI will then commence executing whatever operation was specified by the TWCR setting.

In the following an assembly and C implementation of the example is given. Note that the code below assumes that several definitions have been made for example by using include-files.

	Assembly code example[1]	C example[1]	Comments
1	`ldi r16, (1<<TWINT)|(1<<TWSTA)| (1<<TWEN)` `out TWCR, r16`	`TWCR = (1<<TWINT)|(1<<TWSTA)| (1<<TWEN)`	Send START condition

2	`wait1:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait1`
		`while (!(TWCR & (1<<TWINT)))`
		`;`
			Wait for TWINT flag set. This indicates that the START condition has been transmitted

3	`in r16,TWSR`
	`andi r16, 0xF8`
	`cpi r16, START`
	`brne ERROR`
		`if ((TWSR & 0xF8) != START)`
		`ERROR();`
			Check value of TWI Status Register. Mask prescaler bits. If status different from START go to ERROR

	Assembly code example[1]	C example[1]	Comments

4	`ldi r16, SLA_W`
	`out TWDR, r16`
	`ldi r16, (1<<TWINT) | (1<<TWEN)`
	`out TWCR, r16`
		`TWDR = SLA_W;`
		`TWCR = (1<<TWINT) | (1<<TWEN);`
			Load SLA_W into TWDR Register. Clear TWINT bit in TWCR to start transmission of address

	`wait2:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait2`
		`while (!(TWCR & (1<<TWINT)))`
		`;`
			Wait for TWINT flag set. This indicates that the SLA+W has been transmitted, and ACK/NACK has been received.

5	`in r16,TWSR`
	`andi r16, 0xF8`
	`cpi r16, MT_SLA_ACK`
	`brne ERROR`
		`if ((TWSR & 0xF8) != MT_SLA_ACK)`
		`ERROR();`
			Check value of TWI Status Register. Mask prescaler bits. If status different from MT_SLA_ACK go to ERROR

	`ldi r16, DATA`
	`out TWDR, r16`
	`ldi r16, (1<<TWINT) | (1<<TWEN)`
	`out TWCR, r16`
		`TWDR = DATA;`
		`TWCR = (1<<TWINT) | (1<<TWEN);`
			Load DATA into TWDR Register. Clear TWINT bit in TWCR to start transmission of data

6	`wait3:`
	`in r16,TWCR`
	`sbrs r16,TWINT`
	`rjmp wait3`
		`while (!(TWCR & (1<<TWINT)))`
		`;`
			Wait for TWINT flag set. This indicates that the DATA has been transmitted, and ACK/NACK has been received.

7	`in r16,TWSR`
	`andi r16, 0xF8`
	`cpi r16, MT_DATA_ACK`
	`brne ERROR`
		`if ((TWSR & 0xF8) != MT_DATA_ACK)`
		`ERROR();`
			Check value of TWI Status Register. Mask prescaler bits. If status different from MT_DATA_ACK go to ERROR

	`ldi r16, (1<<TWINT)|(1<<TWEN)| (1<<TWSTO)` `out TWCR, r16`	`TWCR = (1<<TWINT)|(1<<TWEN)| (1<<TWSTO);`	Transmit STOP condition

Note: 1. See "About Code Examples" on page 7.

24.8 Transmission Modes

The TWI can operate in one of four major modes. These are named Master Transmitter (MT), Master Receiver (MR), Slave Transmitter (ST) and Slave Receiver (SR). Several of these modes can be used in the same application. As an example, the TWI can use MT mode to write data into a TWI EEPROM, MR mode to read the data back from the EEPROM. If other masters are present in the system, some of these might transmit data to the TWI, and then SR mode would be used. It is the application software that decides which modes are legal.

The following sections describe each of these modes. Possible status codes are described along with figures detailing data transmission in each of the modes. These figures contain the following abbreviations:

S: START condition

Rs:	REPEATED START condition
R:	Read bit (high level at SDA)
W:	Write bit (low level at SDA)
A:	Acknowledge bit (low level at SDA)
$\overline{A}$:	Not acknowledge bit (high level at SDA)
Data:	8-bit data byte
P:	STOP condition
SLA:	Slave Address

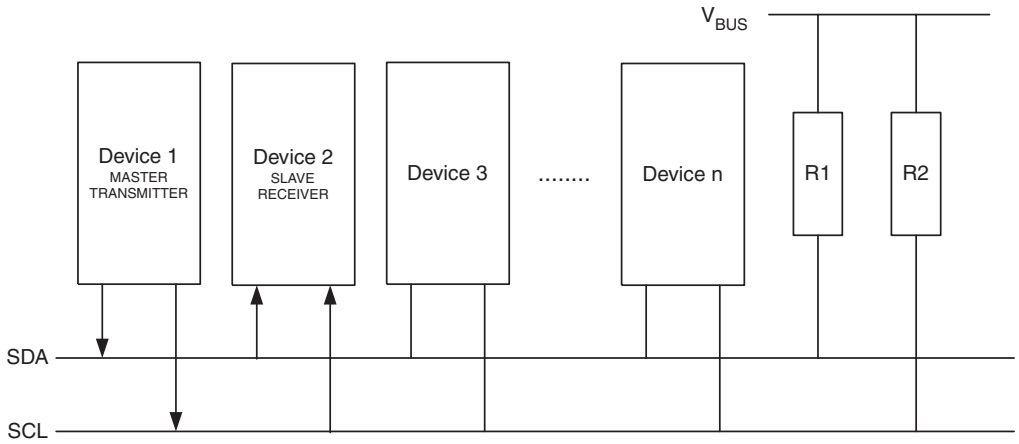
In Figure 24-13 to Figure 24-19, circles are used to indicate that the TWINT flag is set. The numbers in the circles show the status code held in TWSR, with the prescaler bits masked to zero. At these points, actions must be taken by the application to continue or complete the TWI transfer. The TWI transfer is suspended until the TWINT flag is cleared by software.

When the TWINT flag is set, the status code in TWSR is used to determine the appropriate software action. For each status code, the required software action and details of the following serial transfer are given in Table 24-3 to Table 24-6. Note that the prescaler bits are masked to zero in these tables.

24.8.1 Master Transmitter Mode

In the Master Transmitter mode, a number of data bytes are transmitted to a slave receiver (see Figure 24-12). In order to enter a Master mode, a START condition must be transmitted. The format of the following address packet determines whether Master Transmitter or Master Receiver mode is to be entered. If SLA+W is transmitted, MT mode is entered, if SLA+R is transmitted, MR mode is entered. All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 24-12. Data Transfer in Master Transmitter Mode



A START condition is sent by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	1	0	X	1	0	X

TWEN must be set to enable the Two-wire Serial Interface, TWSTA must be written to one to transmit a START condition and TWINT must be written to one to clear the TWINT flag. The TWI

will then test the Two-wire Serial Bus and generate a START condition as soon as the bus becomes free. After a START condition has been transmitted, the TWINT flag is set by hardware, and the status code in TWSR will be 0x08 (see Table 24-3). In order to enter MT mode, SLA+W must be transmitted. This is done by writing SLA+W to TWDR. Thereafter the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	0	0	X	1	0	X

When SLA+W have been transmitted and an acknowledgment bit has been received, TWINT is set again and a number of status codes in TWSR are possible. Possible status codes in Master mode are 0x18, 0x20, or 0x38. The appropriate action to be taken for each of these status codes is detailed in Table 24-3.

When SLA+W has been successfully transmitted, a data packet should be transmitted. This is done by writing the data byte to TWDR. TWDR must only be written when TWINT is high. If not, the access will be discarded, and the Write Collision bit (TWWC) will be set in the TWCR Register. After updating TWDR, the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	0	0	X	1	0	X

This scheme is repeated until the last byte has been sent and the transfer is ended by generating a STOP condition or a repeated START condition. A STOP condition is generated by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	0	1	X	1	0	X

A REPEATED START condition is generated by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	1	0	X	1	0	X

After a repeated START condition (state 0x10) the Two-wire Serial Interface can access the same slave again, or a new slave without transmitting a STOP condition. Repeated START enables the master to switch between slaves, Master Transmitter mode and Master Receiver mode without losing control of the bus.

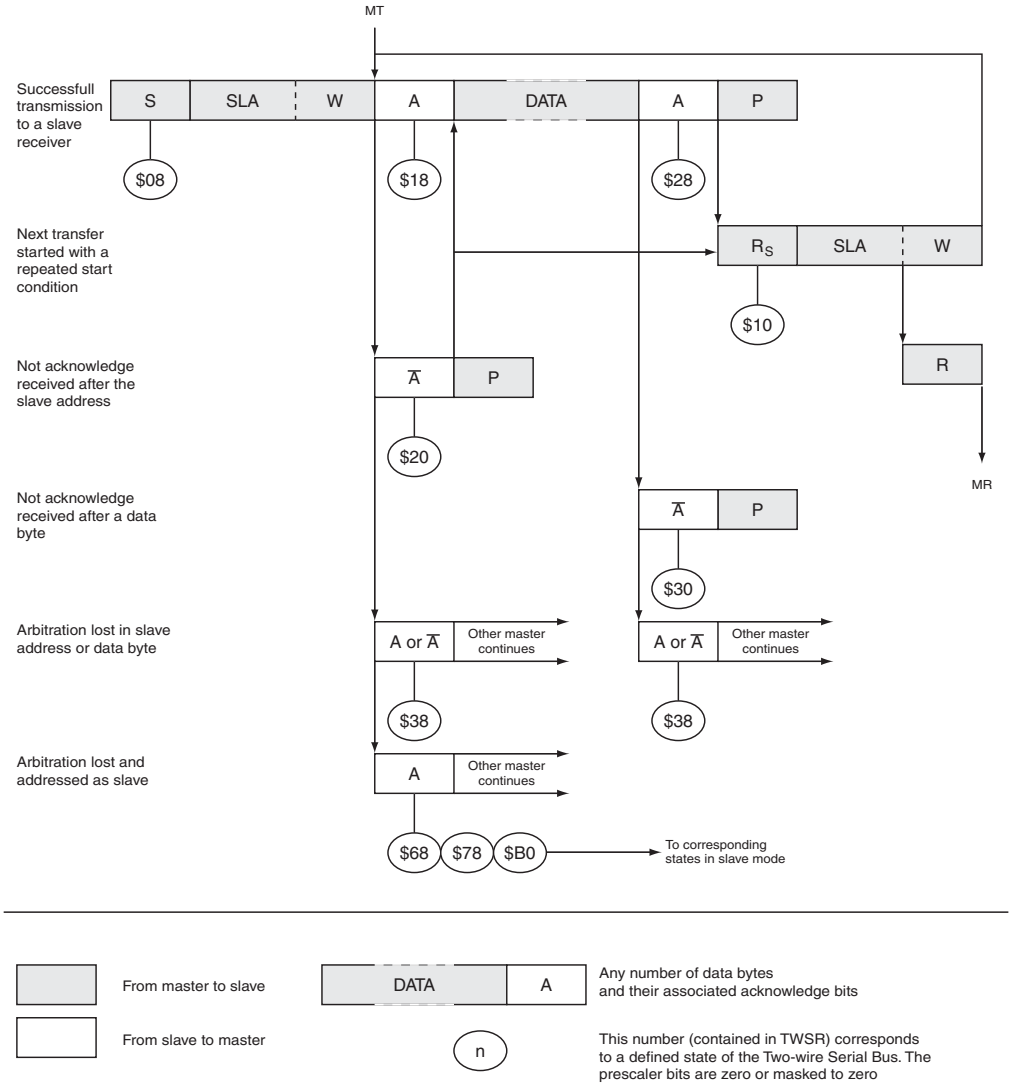
Table 24-3. Status Codes for Master Transmitter Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR STA	To TWCR STO	To TWCR TWINT	To TWCR TWEA	
0x08	A START condition has been transmitted	Load SLA+W	X	0	1	X	SLA+W will be transmitted; ACK or NOT ACK will be received
0x10	A repeated START condition has been transmitted	Load SLA+W or	X	0	1	X	SLA+W will be transmitted; ACK or NOT ACK will be received
		Load SLA+R	X	0	1	X	SLA+R will be transmitted; Logic will switch to Master Receiver mode
0x18	SLA+W has been transmitted; ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset

Table 24-3. Status Codes for Master Transmitter Mode

0x20	SLA+W has been transmitted; NOT ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset
0x28	Data byte has been transmitted; ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset
0x30	Data byte has been transmitted; NOT ACK has been received	Load data byte or	0	0	1	X	Data byte will be transmitted and ACK or NOT ACK will be received
		No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset
0x38	Arbitration lost in SLA+W or data bytes	No TWDR action or	0	0	1	X	Two-wire Serial Bus will be released and not addressed slave mode entered
		No TWDR action	1	0	1	X	A START condition will be transmitted when the bus becomes free

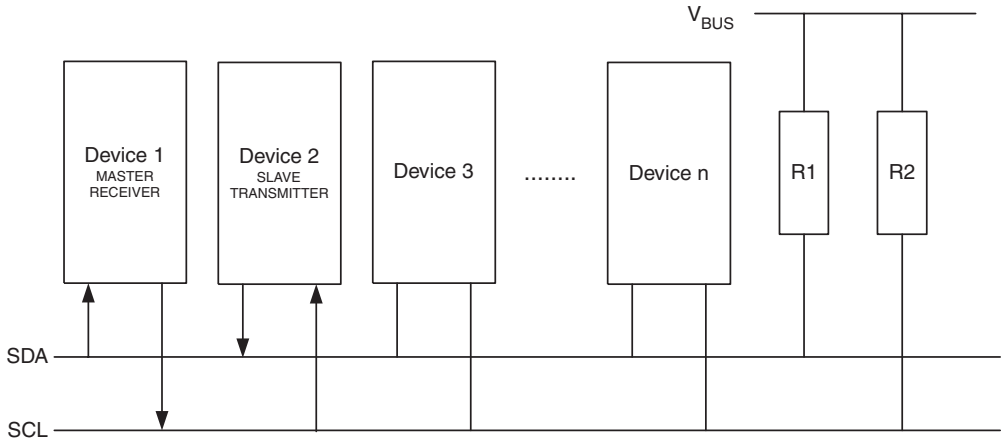
Figure 24-13. Formats and States in the Master Transmitter Mode



24.8.2 Master Receiver Mode

In the Master Receiver mode, a number of data bytes are received from a slave transmitter (see Figure 24-14). In order to enter a Master mode, a START condition must be transmitted. The format of the following address packet determines whether Master Transmitter or Master Receiver mode is to be entered. If SLA+W is transmitted, MT mode is entered, if SLA+R is transmitted, MR mode is entered. All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 24-14. Data Transfer in Master Receiver Mode



A START condition is sent by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	1	0	X	1	0	X

TWEN must be written to one to enable the Two-wire Serial Interface, TWSTA must be written to one to transmit a START condition and TWINT must be set to clear the TWINT flag. The TWI will then test the Two-wire Serial Bus and generate a START condition as soon as the bus becomes free. After a START condition has been transmitted, the TWINT flag is set by hardware, and the status code in TWSR will be 0x08 (see Table 24-3). In order to enter MR mode, SLA+R must be transmitted. This is done by writing SLA+R to TWDR. Thereafter the TWINT bit should be cleared (by writing it to one) to continue the transfer. This is accomplished by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	0	0	X	1	0	X

When SLA+R have been transmitted and an acknowledgment bit has been received, TWINT is set again and a number of status codes in TWSR are possible. Possible status codes in Master mode are 0x38, 0x40, or 0x48. The appropriate action to be taken for each of these status codes is detailed in Table 24-13. Received data can be read from the TWDR Register when the TWINT flag is set high by hardware. This scheme is repeated until the last byte has been received. After the last byte has been received, the MR should inform the ST by sending a NACK after the last received data byte. The transfer is ended by generating a STOP condition or a repeated START condition. A STOP condition is generated by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	0	1	X	1	0	X

A REPEATED START condition is generated by writing the following value to TWCR:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	1	X	1	0	X	1	0	X

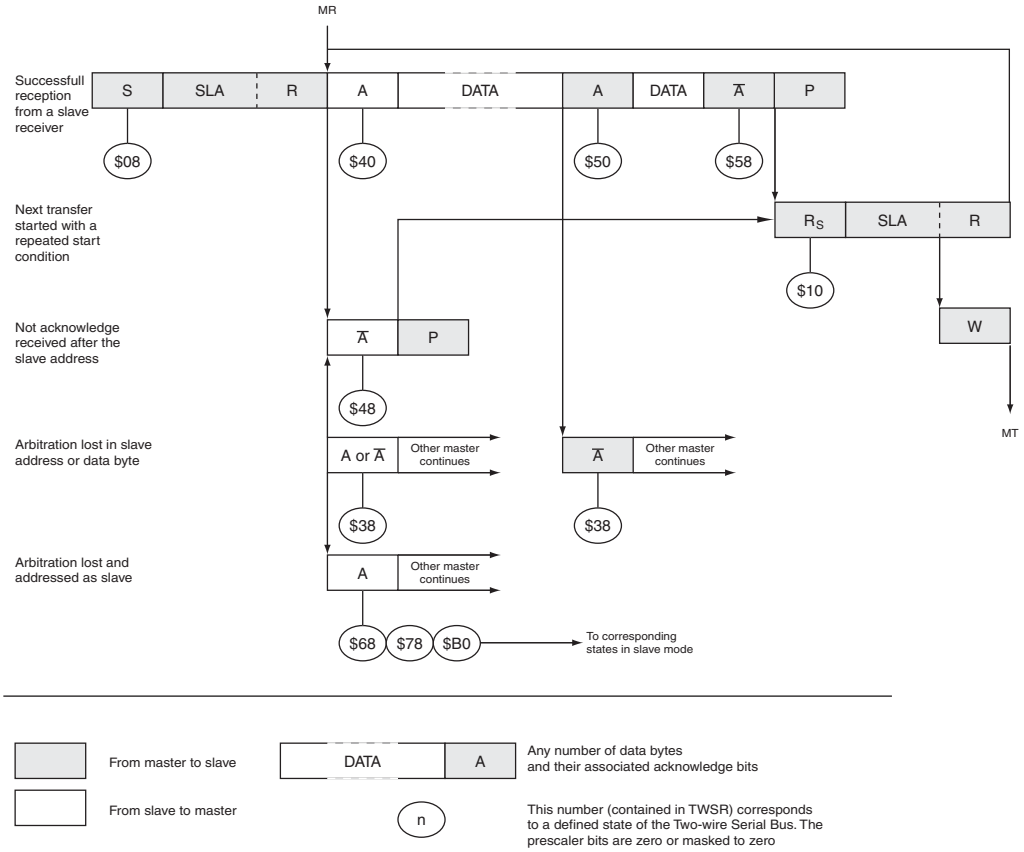
After a repeated START condition (state 0x10) the Two-wire Serial Interface can access the same slave again, or a new slave without transmitting a STOP condition. Repeated START

enables the master to switch between slaves, Master Transmitter mode and Master Receiver mode without losing control over the bus.

Table 24-4. Status Codes for Master Receiver Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0x08	A START condition has been transmitted	Load SLA+R	X	0	1	X	SLA+R will be transmitted ACK or NOT ACK will be received
0x10	A repeated START condition has been transmitted	Load SLA+R or	X	0	1	X	SLA+R will be transmitted ACK or NOT ACK will be received
		Load SLA+W	X	0	1	X	SLA+W will be transmitted Logic will switch to Master Transmitter mode
0x38	Arbitration lost in SLA+R or NOT ACK bit	No TWDR action or	0	0	1	X	Two-wire Serial Bus will be released and not addressed Slave mode will be entered
		No TWDR action	1	0	1	X	A START condition will be transmitted when the bus becomes free
0x40	SLA+R has been transmitted; ACK has been received	No TWDR action or	0	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	0	0	1	1	Data byte will be received and ACK will be returned
0x48	SLA+R has been transmitted; NOT ACK has been received	No TWDR action or	1	0	1	X	Repeated START will be transmitted
		No TWDR action or	0	1	1	X	STOP condition will be transmitted and TWSTO flag will be reset
		No TWDR action	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset
0x50	Data byte has been received; ACK has been returned	Read data byte or	0	0	1	0	Data byte will be received and NOT ACK will be returned
		Read data byte	0	0	1	1	Data byte will be received and ACK will be returned
0x58	Data byte has been received; NOT ACK has been returned	Read data byte or	1	0	1	X	Repeated START will be transmitted
		Read data byte or	0	1	1	X	STOP condition will be transmitted and TWSTO flag will be reset
		Read data byte	1	1	1	X	STOP condition followed by a START condition will be transmitted and TWSTO flag will be reset

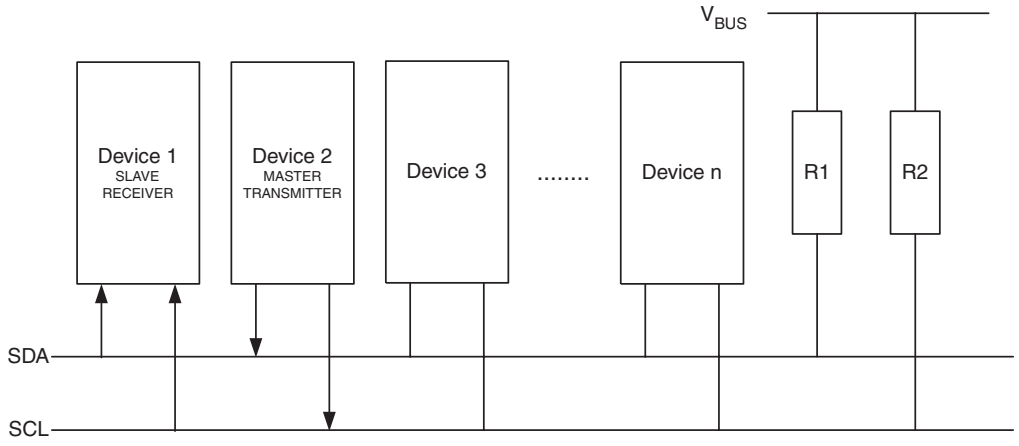
Figure 24-15. Formats and States in the Master Receiver Mode



24.8.3 Slave Receiver Mode

In the Slave Receiver mode, a number of data bytes are received from a master transmitter (see Figure 24-16). All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 24-16. Data Transfer in Slave Receiver Mode



To initiate the Slave Receiver mode, TWAR and TWCR must be initialized as follows:

TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
Value	Device's Own Slave Address							

The upper seven bits are the address to which the Two-wire Serial Interface will respond when addressed by a master. If the LSB is set, the TWI will respond to the general call address (0x00), otherwise it will ignore the general call address.

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	0	1	0	0	0	1	0	X

TWEN must be written to one to enable the TWI. The TWEA bit must be written to one to enable the acknowledgment of the device's own slave address or the general call address. TWSTA and TWSTO must be written to zero.

When TWAR and TWCR have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address if enabled) followed by the data direction bit. If the direction bit is "0" (write), the TWI will operate in SR mode, otherwise ST mode is entered. After its own slave address and the write bit have been received, the TWINT flag is set and a valid status code can be read from TWSR. The status code is used to determine the appropriate software action. The appropriate action to be taken for each status code is detailed in Table 24-5. The Slave Receiver mode may also be entered if arbitration is lost while the TWI is in the Master mode (see states 0x68 and 0x78).

If the TWEA bit is reset during a transfer, the TWI will return a "Not Acknowledge" ("1") to SDA after the next received data byte. This can be used to indicate that the slave is not able to receive any more bytes. While TWEA is zero, the TWI does not acknowledge its own slave address. However, the Two-wire Serial Bus is still monitored and address recognition may resume at any time by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the Two-wire Serial Bus.

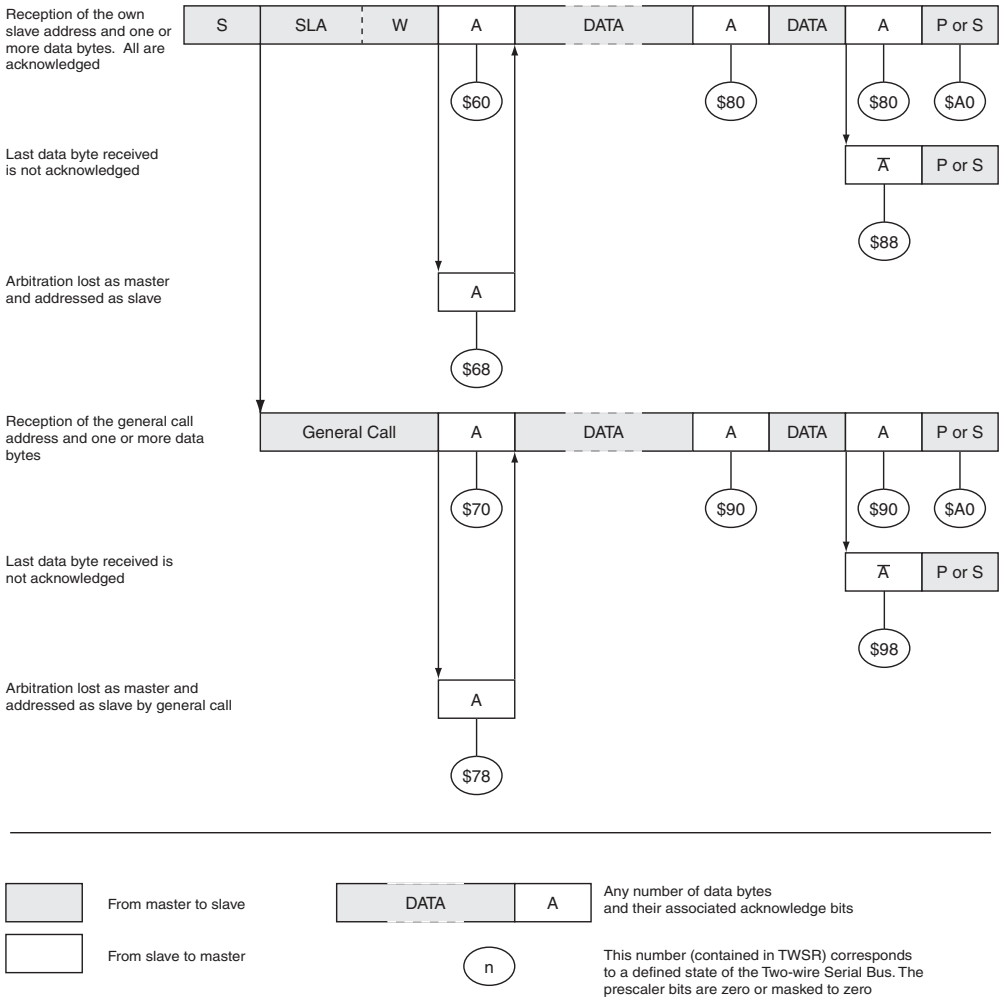
In all sleep modes other than Idle mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the Two-wire Serial Bus clock as a clock source. The part will then wake-up from sleep and the TWI will hold the SCL clock low during the wake up and until the TWINT flag is cleared (by writing it to one). Further data reception will be carried out as normal, with the AVR clocks running as normal. Observe that if the AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions.

Note that the Two-wire Serial Interface Data Register – TWDR does not reflect the last byte present on the bus when waking up from these Sleep modes.

Table 24-5. Status Codes for Slave Receiver Mode

Status Code (TWSR) Prescaler Bits Are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR STA	STO	TWINT	TWEA	
0x60	Own SLA+W has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x68	Arbitration lost in SLA+R/W as master; own SLA+W has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x70	General call address has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x78	Arbitration lost in SLA+R/W as master; General call address has been received; ACK has been returned	No TWDR action or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		No TWDR action	X	0	1	1	Data byte will be received and ACK will be returned
0x80	Previously addressed with own SLA+W; data has been received; ACK has been returned	Read data byte or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		Read data byte	X	0	1	1	Data byte will be received and ACK will be returned
0x88	Previously addressed with own SLA+W; data has been received; NOT ACK has been returned	Read data byte or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		Read data byte or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
		Read data byte or	1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		Read data byte	1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free
0x90	Previously addressed with general call; data has been received; ACK has been returned	Read data byte or	X	0	1	0	Data byte will be received and NOT ACK will be returned
		Read data byte	X	0	1	1	Data byte will be received and ACK will be returned
0x98	Previously addressed with general call; data has been received; NOT ACK has been returned	Read data byte or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		Read data byte or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
		Read data byte or	1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		Read data byte	1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free
0xA0	A STOP condition or repeated START condition has been received while still addressed as slave	Read data byte or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		Read data byte or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
		Read data byte or	1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		Read data byte	1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free

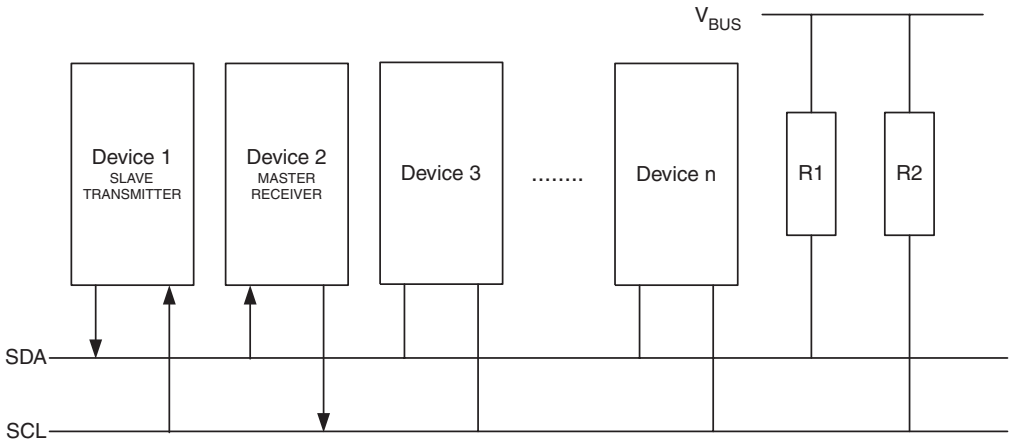
Figure 24-17. Formats and States in the Slave Receiver Mode



24.8.4 Slave Transmitter Mode

In the Slave Transmitter mode, a number of data bytes are transmitted to a master receiver (see Figure 24-18). All the status codes mentioned in this section assume that the prescaler bits are zero or are masked to zero.

Figure 24-18. Data Transfer in Slave Transmitter Mode



To initiate the Slave Transmitter mode, TWAR and TWCR must be initialized as follows:

TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
Value	Device's Own Slave Address							

The upper seven bits are the address to which the Two-wire Serial Interface will respond when addressed by a master. If the LSB is set, the TWI will respond to the general call address (0x00), otherwise it will ignore the general call address.

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
Value	0	1	0	0	0	1	0	X

TWEN must be written to one to enable the TWI. The TWEA bit must be written to one to enable the acknowledgment of the device's own slave address or the general call address. TWSTA and TWSTO must be written to zero.

When TWAR and TWCR have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address if enabled) followed by the data direction bit. If the direction bit is "1" (read), the TWI will operate in ST mode, otherwise SR mode is entered. After its own slave address and the write bit have been received, the TWINT flag is set and a valid status code can be read from TWSR. The status code is used to determine the appropriate software action. The appropriate action to be taken for each status code is detailed in Table 24-6. The Slave Transmitter mode may also be entered if arbitration is lost while the TWI is in the Master mode (see state 0xB0).

If the TWEA bit is written to zero during a transfer, the TWI will transmit the last byte of the transfer. State 0xC0 or state 0xC8 will be entered, depending on whether the master receiver transmits a NACK or ACK after the final byte. The TWI is switched to the not addressed Slave mode, and will ignore the master if it continues the transfer. Thus the master receiver receives all "1" as serial data. State 0xC8 is entered if the master demands additional data bytes (by transmitting ACK), even though the slave has transmitted the last byte (TWEA zero and expecting NACK from the master).

While TWEA is zero, the TWI does not respond to its own slave address. However, the Two-wire Serial Bus is still monitored and address recognition may resume at any time by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the Two-wire Serial Bus.

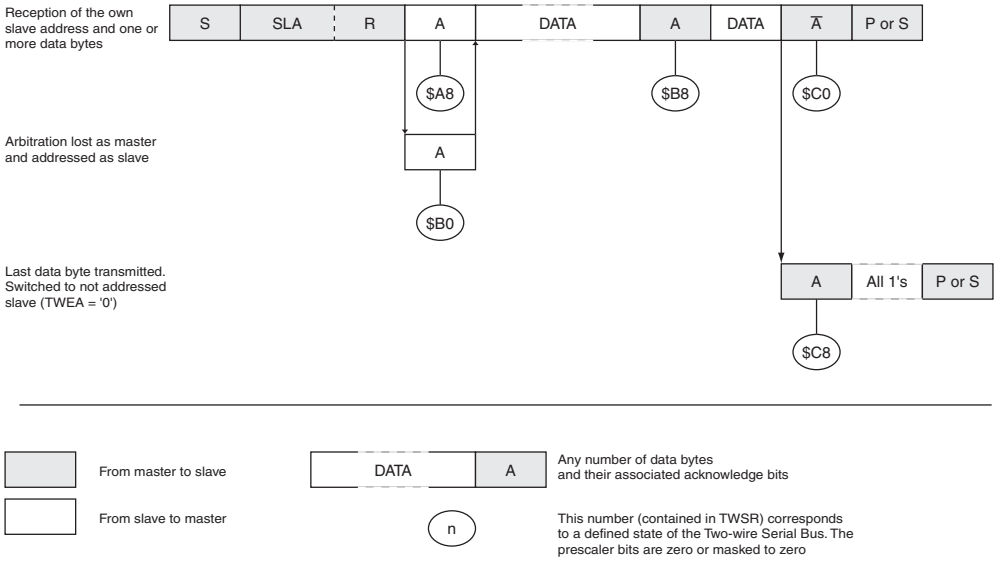
In all sleep modes other than Idle mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the Two-wire Serial Bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock will low during the wake up and until the TWINT flag is cleared (by writing it to one). Further data transmission will be carried out as normal, with the AVR clocks running as normal. Observe that if the AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions.

Note that the Two-wire Serial Interface Data Register – TWDR – does not reflect the last byte present on the bus when waking up from these sleep modes.

Table 24-6. Status Codes for Slave Transmitter Mode

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0xA8	Own SLA+R has been received; ACK has been returned	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received
		Load data byte	X	0	1	1	Data byte will be transmitted and ACK should be received
0xB0	Arbitration lost in SLA+R/W as master; own SLA+R has been received; ACK has been returned	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received
		Load data byte	X	0	1	1	Data byte will be transmitted and ACK should be received
0xB8	Data byte in TWDR has been transmitted; ACK has been received	Load data byte or	X	0	1	0	Last data byte will be transmitted and NOT ACK should be received
		Load data byte	X	0	1	1	Data byte will be transmitted and ACK should be received
0xC0	Data byte in TWDR has been transmitted; NOT ACK has been received	No TWDR action or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		No TWDR action or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
		No TWDR action or	1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		No TWDR action	1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free
0xC8	Last data byte in TWDR has been transmitted (TWEA = “0”); ACK has been received	No TWDR action or	0	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA
		No TWDR action or	0	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”
		No TWDR action or	1	0	1	0	Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free
		No TWDR action	1	0	1	1	Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free

Figure 24-19. Formats and States in the Slave Transmitter Mode



24.8.5 Miscellaneous States

There are two status codes that do not correspond to a defined TWI state, see Table 24-7.

Status 0xF8 indicates that no relevant information is available because the TWINT flag is not set. This occurs between other states, and when the TWI is not involved in a serial transfer.

Status 0x00 indicates that a bus error has occurred during a Two-wire Serial Bus transfer. A bus error occurs when a START or STOP condition occurs at an illegal position in the format frame. Examples of such illegal positions are during the serial transfer of an address byte, a data byte, or an acknowledge bit. When a bus error occurs, TWINT is set. To recover from a bus error, the TWSTO flag must set and TWINT must be cleared by writing a logic one to it. This causes the TWI to enter the not addressed Slave mode and to clear the TWSTO flag (no other bits in TWCR are affected). The SDA and SCL lines are released, and no STOP condition is transmitted.

Table 24-7. Miscellaneous States

Status Code (TWSR) Prescaler Bits are 0	Status of the Two-wire Serial Bus and Two-wire Serial Interface hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0xF8	No relevant state information available; TWINT = “0”	No TWDR action	No TWCR action				Wait or proceed current transfer
0x00	Bus error due to an illegal START or STOP condition	No TWDR action	0	1	1	X	Only the internal hardware is affected, no STOP condition is sent on the bus. In all cases, the bus is released and TWSTO is cleared.

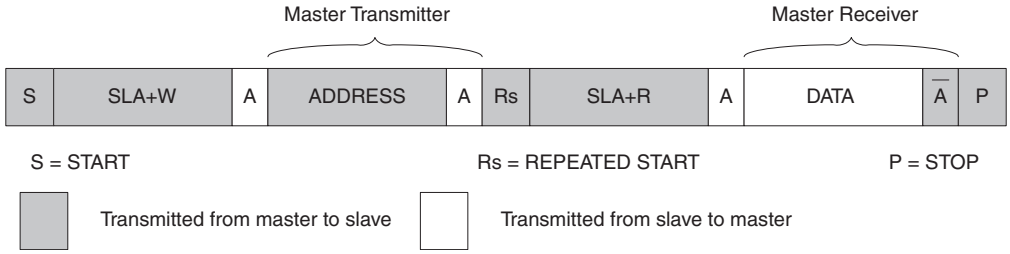
24.8.6 Combining Several TWI Modes

In some cases, several TWI modes must be combined in order to complete the desired action. Consider for example reading data from a serial EEPROM. Typically, such a transfer involves the following steps:

1. The transfer must be initiated.
2. The EEPROM must be instructed what location should be read.
3. The reading must be performed.
4. The transfer must be finished.

Note that data is transmitted both from Master to Slave and vice versa. The Master must instruct the slave what location it wants to read, requiring the use of the MT mode. Subsequently, data must be read from the slave, implying the use of the MR mode. Thus, the transfer direction must be changed. The Master must keep control of the bus during all these steps, and the steps should be carried out as an atomic operation. If this principle is violated in a multi-master system, another master can alter the data pointer in the EEPROM between steps 2 and 3, and the master will read the wrong data location. Such a change in transfer direction is accomplished by transmitting a REPEATED START between the transmission of the address byte and reception of the data. After a REPEATED START, the master keeps ownership of the bus. The following figure shows the flow in this transfer.

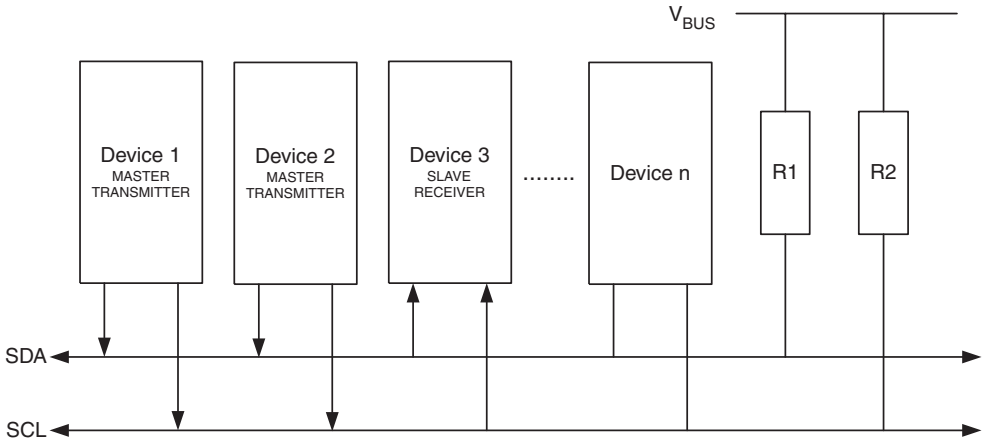
Figure 24-20. Combining Several TWI Modes to Access a Serial EEPROM



24.9 Multi-master Systems and Arbitration

If multiple masters are connected to the same bus, transmissions may be initiated simultaneously by one or more of them. The TWI standard ensures that such situations are handled in such a way that one of the masters will be allowed to proceed with the transfer, and that no data will be lost in the process. An example of an arbitration situation is depicted below, where two masters are trying to transmit data to a slave receiver.

Figure 24-21. An Arbitration Example

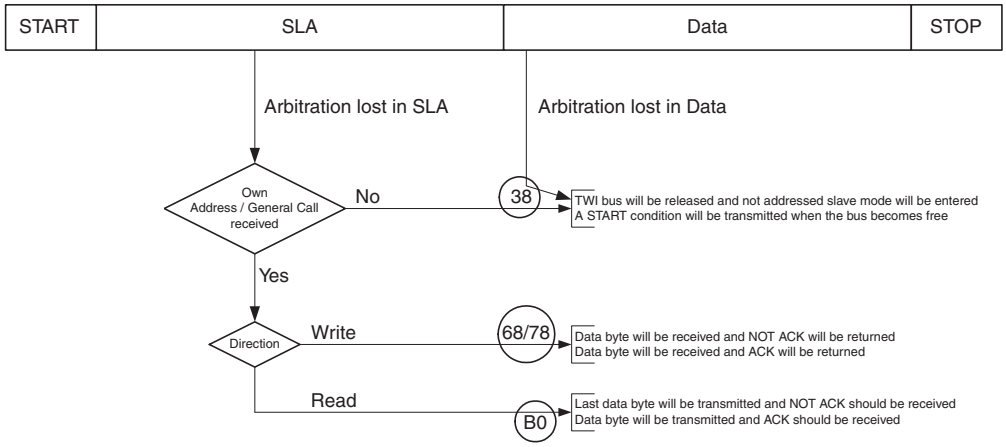


Several different scenarios may arise during arbitration, as described below:

- Two or more masters are performing identical communication with the same slave. In this case, neither the slave nor any of the masters will know about the bus contention.
- Two or more masters are accessing the same slave with different data or direction bit. In this case, arbitration will occur, either in the READ/WRITE bit or in the data bits. The masters trying to output a one on SDA while another master outputs a zero will lose the arbitration. Losing masters will switch to not addressed Slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.
- Two or more masters are accessing different slaves. In this case, arbitration will occur in the SLA bits. Masters trying to output a one on SDA while another master outputs a zero will lose the arbitration. Masters losing arbitration in SLA will switch to Slave mode to check if they are being addressed by the winning master. If addressed, they will switch to SR or ST mode, depending on the value of the READ/WRITE bit. If they are not being addressed, they will switch to not addressed Slave mode or wait until the bus is free and transmit a new START condition, depending on application software action.

This is summarized in Figure 24-22. Possible status values are given in circles.

Figure 24-22. Possible Status Codes Caused by Arbitration



24.10 Bus Connect/Disconnect for Two-wire Serial Interface

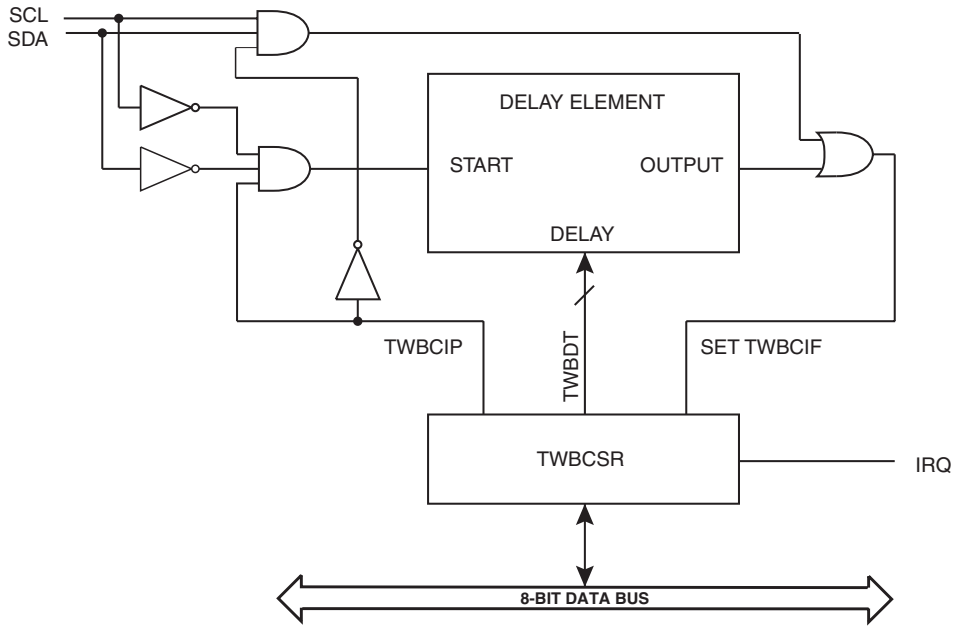
The Bus Connect/Disconnect module is an addition to the TWI Interface. Based on a configuration bit, an interrupt can be generated either when the TWI bus is connected or disconnected.

Figure 24-23 illustrates the Bus Connect/Disconnect logic, where SDA and SCL are the TWI data and clock lines, respectively.

When the TWI bus is connected, both the SDA and the SCL lines will become high simultaneously. If the TWBCIP bit is cleared, the interrupt will be executed if enabled. Once the bus is connected, the TWBCIP bit should be set. This enables detection of when the bus is disconnected, and prevents repetitive interrupts every time both the SDA and SCL lines are high (e.g. bus IDLE state).

When the TWI bus is disconnected, both the SDA and the SCL lines will become low simultaneously. If the TWBCIP bit is set, the interrupt will be executed if enabled and if both lines remain low for a configurable time period. By adding this time constraint, unwanted interrupts caused by both lines going low during normal bus communication is prevented.

Figure 24-23. Overview of Bus Connect/Disconnect.



24.10.1 TWI Bus Control and Status Register - TWBCSR

Bit	7	6	5	4	3	2	1	0	
	TWBCIF	**TWBCIE**	**–**	**–**	**–**	**TWBDT1**	**TWBDT0**	**TWBCIP**	**TWBCSR**
Read/Write	R/W	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	**X**	0	0	0	0	0	0	0	

- **Bit 7 - TWBCIF: TWI Bus Connect/Disconnect Interrupt Flag**

Based on the TWBCIP bit, the TWBCIF bit is set when the TWI bus is connected or disconnected. TWBCIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, TWBCIF is cleared by writing a logic one to the flag. When the SREG I-bit, TWBCIE (TWI Bus Connect/Disconnect Interrupt Enable), and TWBCIF are set, the TWI Bus

Connect/Disconnect Interrupt is executed. If both SDA and SCL are high during reset, TWBCIF will be set after reset. Otherwise TWBCIF will be cleared after reset.

- **Bit 6 - TWBCIE: TWI Bus Connect/Disconnect Interrupt Enable**

When the TWBCIE bit and the I-bit in the Status Register are set, the TWI Bus Connect/Disconnect Interrupt is enabled. The corresponding interrupt is executed if a TWI Bus Connect/Disconnect occurs, i.e., when the TWBCIE bit is set.

- **Bit 5:3 - Res: Reserved Bits**

These bits are reserved bits in the ATmega406 and will always read as zero.

- **Bit 2:1 - TWBDT1, TWBDT0: TWI Bus Disconnect Time-out Period**

The TWBDT bits decides how long both the TWI data (SDA) and clock (SCL) signals must be low before generating the TWI Bus Disconnect Interrupt. The different configuration values and their corresponding time-out periods are shown in Table 24-8.

Table 24-8. TW Bus Disconnect Time-out Period

TWBDT1	TWBDT0	TWI Bus Disconnect Time-out Period
0	0	250 ms
0	1	500 ms
1	0	1000 ms
1	1	2000 ms

- **Bit 0 - TWBCIP: TWI Bus Connect/Disconnect Interrupt Polarity**

The TWBCIP bit decide if the TWI Bus Connect/Disconnect Interrupt Flag (TWBCIF) should be set on a Bus Connect or a Bus Disconnect. If TWBCIP is cleared, the TWBCIF flag is set on a Bus Connect. If TWBCIP is set, the TWBCIF flag is set on a Bus Disconnect.

25. JTAG Interface and On-chip Debug System

25.1 Features

- **JTAG (IEEE std. 1149.1 Compliant) Interface**
- **Debugger Access to:**
  - **All Internal Peripheral Units**
  - **Internal and External RAM**
  - **The Internal Register File**
  - **Program Counter**
  - **EEPROM and Flash Memories**
- **Extensive On-chip Debug Support for Break Conditions, Including**
  - **AVR Break Instruction**
  - **Break on Change of Program Memory Flow**
  - **Single Step Break**
  - **Program Memory Break Points on Single Address or Address Range**
  - **Data Memory Break Points on Single Address or Address Range**
- **Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface**
- **On-chip Debugging Supported by AVR Studio®**

25.2 Overview

The AVR IEEE std. 1149.1 compliant JTAG interface can be used for

- Programming the non-volatile memories, Fuses and Lock bits
- On-chip debugging

A brief description is given in the following sections. Detailed descriptions for Programming via the JTAG interface can be found in the section ”Programming via the JTAG Interface” on page 198. The On-chip Debug support is considered being private JTAG instructions, and distributed within ATMEL and to selected third party vendors only.

Figure 25-1 shows a block diagram of the JTAG interface and the On-chip Debug system. The TAP Controller is a state machine controlled by the TCK and TMS signals. The TAP Controller selects either the JTAG Instruction Register or one of several Data Registers as the scan chain (Shift Register) between the TDI – input and TDO – output. The Instruction Register holds JTAG instructions controlling the behavior of a Data Register.

The JTAG Programming Interface (actually consisting of several physical and virtual Data Registers) is used for serial programming via the JTAG interface. The Internal Scan Chain and Break Point Scan Chain are used for On-chip debugging only.

25.3 Test Access Port – TAP

The JTAG interface is accessed through four of the AVR’s pins. In JTAG terminology, these pins constitute the Test Access Port – TAP. These pins are:

- TMS: Test mode select. This pin is used for navigating through the TAP-controller state machine.
- TCK: Test Clock. JTAG operation is synchronous to TCK.
- TDI: Test Data In. Serial input data to be shifted in to the Instruction Register or Data Register (Scan Chains).
- TDO: Test Data Out. Serial output data from Instruction Register or Data Register.

The IEEE std. 1149.1 also specifies an optional TAP signal; TRST – Test ReSeT – which is not provided.

When the JTAGEN Fuse is unprogrammed, these four TAP pins are normal port pins, and the TAP controller is in reset. When programmed, the input TAP signals are internally pulled high and the JTAG is enabled for programming. The device is shipped with this fuse programmed.

For the On-chip Debug system, in addition to the JTAG interface pins, the $\overline{\text{RESET}}$ pin is monitored by the debugger to be able to detect external reset sources. The debugger can also pull the $\overline{\text{RESET}}$ pin low to reset the whole system, assuming only open collectors on the reset line are used in the application.

Figure 25-1. Block Diagram

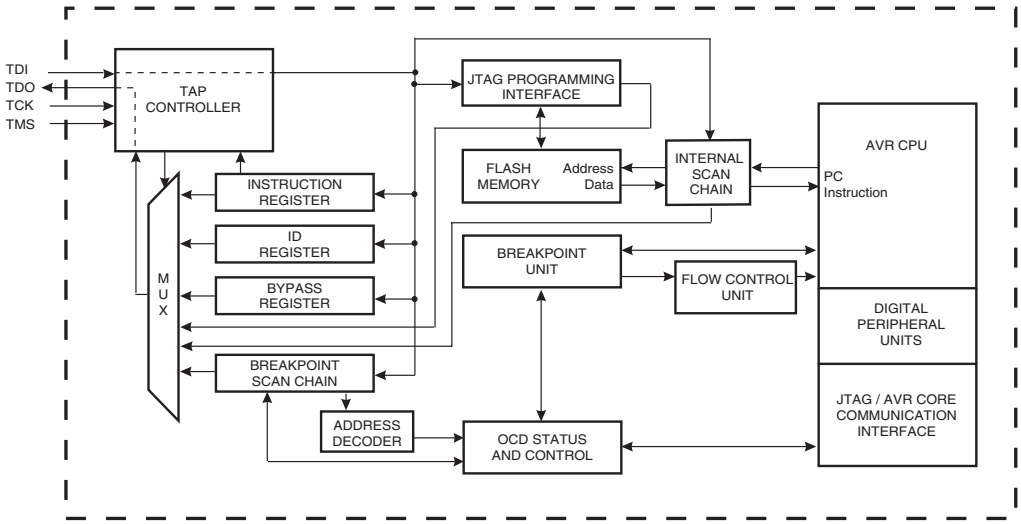
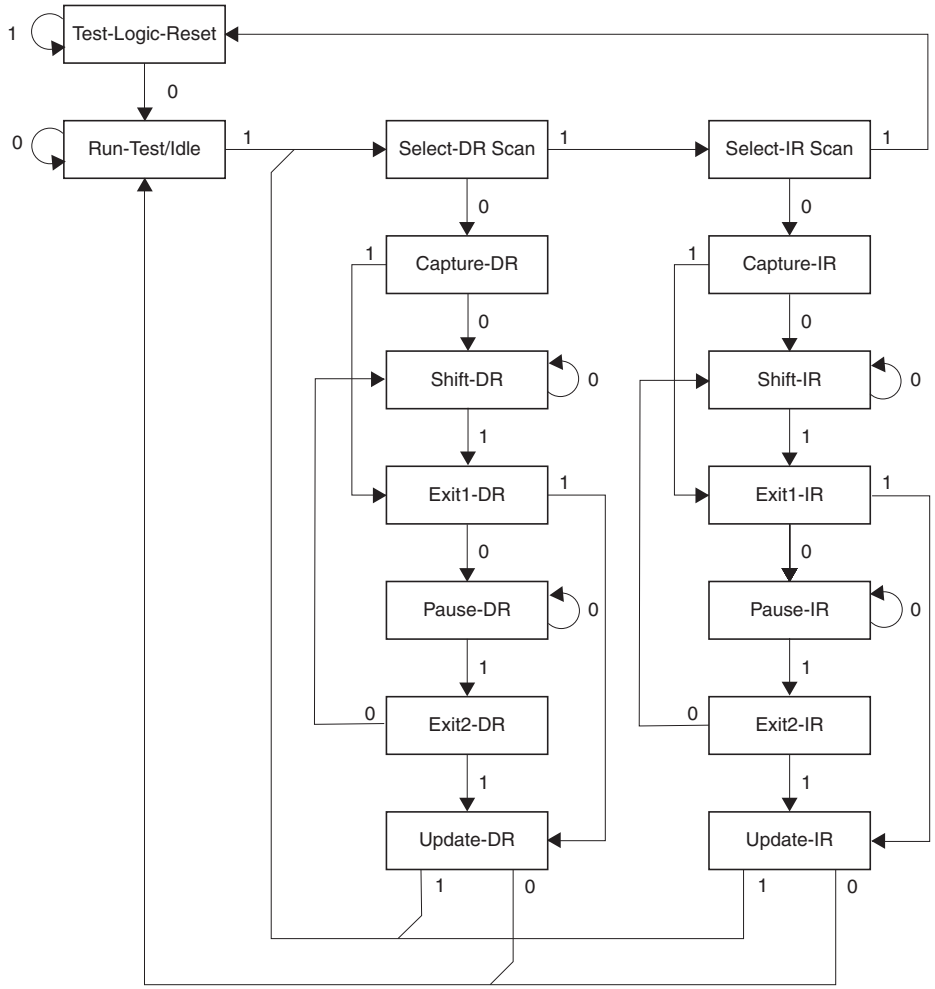


Figure 25-2. TAP Controller State Diagram



25.4 TAP Controller

The TAP controller is a 16-state finite state machine that controls the operation of the JTAG programming circuitry, or On-chip Debug system. The state transitions depicted in Figure 25-2 depend on the signal present on TMS (shown adjacent to each state transition) at the time of the rising edge at TCK. The initial state after a Power-on Reset is Test-Logic-Reset.

As a definition in this document, the LSB is shifted in and out first for all Shift Registers.

Assuming Run-Test/Idle is the present state, a typical scenario for using the JTAG interface is:

- At the TMS input, apply the sequence 1, 1, 0, 0 at the rising edges of TCK to enter the Shift Instruction Register – Shift-IR state. While in this state, shift the four bits of the JTAG instructions into the JTAG Instruction Register from the TDI input at the rising edge of TCK. The TMS input must be held low during input of the 3 LSBs in order to remain in the Shift-IR state. The MSB of the instruction is shifted in when this state is left by setting TMS high. While the instruction is shifted in from the TDI pin, the captured IR-state 0x01 is shifted out on the TDO pin. The JTAG Instruction selects a particular Data Register as path between TDI and TDO and controls the circuitry surrounding the selected Data Register.

- Apply the TMS sequence 1, 1, 0 to re-enter the Run-Test/Idle state. The instruction is latched onto the parallel output from the Shift Register path in the Update-IR state. The Exit-IR, Pause-IR, and Exit2-IR states are only used for navigating the state machine.
- At the TMS input, apply the sequence 1, 0, 0 at the rising edges of TCK to enter the Shift Data Register – Shift-DR state. While in this state, upload the selected data register (selected by the present JTAG instruction in the JTAG Instruction Register) from the TDI input at the rising edge of TCK. In order to remain in the Shift-DR state, the TMS input must be held low during input of all bits except the MSB. The MSB of the data is shifted in when this state is left by setting TMS high. While the data register is shifted in from the TDI pin, the parallel inputs to the data register captured in the Capture-DR state is shifted out on the TDO pin.
- Apply the TMS sequence 1, 1, 0 to re-enter the Run-Test/Idle state. If the selected data register has a latched parallel-output, the latching takes place in the Update-DR state. The Exit-DR, Pause-DR, and Exit2-DR states are only used for navigating the state machine.

As shown in the state diagram, the Run-Test/Idle state need not be entered between selecting JTAG instruction and using data registers, and some JTAG instructions may select certain functions to be performed in the Run-Test/Idle, making it unsuitable as an Idle state.

Note: Independent of the initial state of the TAP Controller, the Test-Logic-Reset state can always be entered by holding TMS high for five TCK clock periods.

For detailed information on the JTAG specification, refer to the literature listed in ”JTAG Interface and On-chip Debug System” on page 164.

25.5 Using the On-chip Debug System

As shown in Figure 25-1, the hardware support for On-chip Debugging consists mainly of

- A scan chain on the interface between the internal AVR CPU and the internal peripheral units.
- Break Point unit.
- Communication interface between the CPU and JTAG system.

All read or modify/write operations needed for implementing the Debugger are done by applying AVR instructions via the internal AVR CPU Scan Chain. The CPU sends the result to an I/O memory mapped location which is part of the communication interface between the CPU and the JTAG system.

The Break Point Unit implements Break on Change of Program Flow, Single Step Break, two Program Memory Break Points, and two combined Break Points. Together, the four Break Points can be configured as either:

- 4 single Program Memory Break Points.
- 3 Single Program Memory Break Point + 1 single Data Memory Break Point.
- 2 single Program Memory Break Points + 2 single Data Memory Break Points.
- 2 single Program Memory Break Points + 1 Program Memory Break Point with mask (“range Break Point”).
- 2 single Program Memory Break Points + 1 Data Memory Break Point with mask (“range Break Point”).

A debugger, like the AVR Studio, may however use one or more of these resources for its internal purpose, leaving less flexibility to the end-user.

A list of the On-chip Debug specific JTAG instructions is given in ”On-chip Debug Specific JTAG Instructions” on page 168.

The JTAGEN Fuse must be programmed to enable the JTAG Test Access Port. In addition, the OCDEN Fuse must be programmed and no Lock bits must be set for the On-chip debug system to work. As a security feature, the On-chip debug system is disabled when either of the LB1 or LB2 Lock bits are set. Otherwise, the On-chip debug system would have provided a back-door into a secured device.

The AVR Studio enables the user to fully control execution of programs on an AVR device with On-chip Debug capability, AVR In-Circuit Emulator, or the built-in AVR Instruction Set Simulator. AVR Studio® supports source level execution of Assembly programs assembled with Atmel Corporation's AVR Assembler and C programs compiled with third party vendors' compilers.

AVR Studio runs under Microsoft® Windows® 95/98/2000 and Microsoft Windows NT®.

For a full description of the AVR Studio, please refer to the AVR Studio User Guide. Only highlights are presented in this document.

All necessary execution commands are available in AVR Studio, both on source level and on disassembly level. The user can execute the program, single step through the code either by tracing into or stepping over functions, step out of functions, place the cursor on a statement and execute until the statement is reached, stop the execution, and reset the execution target. In addition, the user can have an unlimited number of code Break Points (using the BREAK instruction) and up to two data memory Break Points, alternatively combined as a mask (range) Break Point.

25.6 On-chip Debug Specific JTAG Instructions

The On-chip debug support is considered being private JTAG instructions, and distributed within ATMEL and to selected third party vendors only. Instruction opcodes are listed for reference.

25.6.1 PRIVATE0; 0x8

Private JTAG instruction for accessing On-chip debug system.

25.6.2 PRIVATE1; 0x9

Private JTAG instruction for accessing On-chip debug system.

25.6.3 PRIVATE2; 0xA

Private JTAG instruction for accessing On-chip debug system.

25.6.4 PRIVATE3; 0xB

Private JTAG instruction for accessing On-chip debug system.

25.7 On-chip Debug Related Register in I/O Memory

25.7.1 On-chip Debug Register – OCDR

Bit	7	6	5	4	3	2	1	0	
	On-Chip Debug Register								OCDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The OCDR Register provides a communication channel from the running program in the microcontroller to the debugger. The CPU can transfer a byte to the debugger by writing to this location. At the same time, an internal flag; I/O Debug Register Dirty – IDRD – is set to indicate to the debugger that the register has been written. When the CPU reads the OCDR Register the 7 LSB will be from the OCDR Register, while the MSB is the IDRD bit. The debugger clears the IDRD bit when it has read the information.

In some AVR devices, this register is shared with a standard I/O location. In this case, the OCDR Register can only be accessed if the OCDEN Fuse is programmed, and the debugger enables access to the OCDR Register. In all other cases, the standard I/O location is accessed.

Refer to the debugger documentation for further information on how to use this register.

25.8 Using the JTAG Programming Capabilities

Programming of AVR parts via JTAG is performed via the 4-pin JTAG port, TCK, TMS, TDI, and TDO. These are the only pins that need to be controlled/observed to perform JTAG programming (in addition to power pins). It is not required to apply 12V externally. The JTAGEN Fuse must be programmed and the JTD bit in the MCUCR Register must be cleared to enable the JTAG Test Access Port.

The JTAG programming capability supports:

- Flash programming and verifying.
- EEPROM programming and verifying.
- Fuse programming and verifying.
- Lock bit programming and verifying.

The Lock bit security is exactly as in parallel programming mode. If the Lock bits LB1 or LB2 are programmed, the OCDEN Fuse cannot be programmed unless first doing a chip erase. This is a security feature that ensures no back-door exists for reading out the content of a secured device.

The details on programming through the JTAG interface and programming specific JTAG instructions are given in the section ”Programming via the JTAG Interface” on page 198.

26. Boot Loader Support – Read-While-Write Self-Programming

The Boot Loader Support provides a real Read-While-Write Self-Programming mechanism for downloading and uploading program code by the MCU itself. This feature allows flexible application software updates controlled by the MCU using a Flash-resident Boot Loader program. The Boot Loader program can use any available data interface and associated protocol to read code and write (program) that code into the Flash memory, or read the code from the program memory. The program code within the Boot Loader section has the capability to write into the entire Flash, including the Boot Loader memory. The Boot Loader can thus even modify itself, and it can also erase itself from the code if the feature is not needed anymore. The size of the Boot Loader memory is configurable with fuses and the Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

26.1 Boot Loader Features

- **Read-While-Write Self-Programming**
- **Flexible Boot Memory Size**
- **High Security (Separate Boot Lock Bits for a Flexible Protection)**
- **Separate Fuse to Select Reset Vector**
- **Optimized Page[1] Size**
- **Code Efficient Algorithm**
- **Efficient Read-Modify-Write Support**

Note: 1. A page is a section in the Flash consisting of several bytes (see ”Page Size” on page 188) used during programming. The page organization does not affect normal operation.

26.2 Application and Boot Loader Flash Sections

The Flash memory is organized in two main sections, the Application section and the Boot Loader section (see Figure 26-2). The size of the different sections is configured by the BOOTSZ Fuses as shown in Table 26-7 on page 183 and Figure 26-2. These two sections can have different level of protection since they have different sets of Lock bits.

26.2.1 Application Section

The Application section is the section of the Flash that is used for storing the application code. The protection level for the Application section can be selected by the application Boot Lock bits (Boot Lock bits 0), see Table 26-2 on page 174. The Application section can never store any Boot Loader code since the SPM instruction is disabled when executed from the Application section.

26.2.2 BLS – Boot Loader Section

While the Application section is used for storing the application code, the The Boot Loader software must be located in the BLS since the SPM instruction can initiate a programming when executing from the BLS only. The SPM instruction can access the entire Flash, including the BLS itself. The protection level for the Boot Loader section can be selected by the Boot Loader Lock bits (Boot Lock bits 1), see Table 26-3 on page 174.

26.3 Read-While-Write and No Read-While-Write Flash Sections

Whether the CPU supports Read-While-Write or if the CPU is halted during a Boot Loader software update is dependent on which address that is being programmed. In addition to the two sections that are configurable by the BOOTSZ Fuses as described above, the Flash is also divided into two fixed sections, the Read-While-Write (RWW) section and the No Read-While-Write (NRWW) section. The limit between the RWW- and NRWW sections is given in Table 26-8 on page 184 and Figure 26-2 on page 173. The main difference between the two sections is:

- When erasing or writing a page located inside the RWW section, the NRWW section can be read during the operation.
- When erasing or writing a page located inside the NRWW section, the CPU is halted during the entire operation.

Note that the user software can never read any code that is located inside the RWW section during a Boot Loader software operation. The syntax “Read-While-Write section” refers to which section that is being programmed (erased or written), not which section that actually is being read during a Boot Loader software update.

26.3.1 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 a call/jmp/lpm or an interrupt) during programming, the software might end up in an unknown state. To avoid this, the interrupts should either be disabled or moved to the Boot Loader section. The Boot Loader section is always located in the NRWW section. The RWW Section Busy bit (RWWSB) in the Store Program Memory Control and Status Register (SPMCSR) will be read as logical one as long as the RWW section is blocked for reading. After a programming is completed, the RWWSB must be cleared by software before reading code located in the RWW section. See Section “26.5.1” on page 175. for details on how to clear RWWSB.

26.3.2 NRWW – No Read-While-Write Section

The code located in the NRWW section can be read when the Boot Loader software is updating a page in the RWW section. When the Boot Loader code updates the NRWW section, the CPU is halted during the entire Page Erase or Page Write operation.

Table 26-1. Read-While-Write Features

Which Section does the Z-pointer Address During the Programming?	Which Section Can be Read During Programming?	CPU Halted?	Read-While-Write Supported?
RWW Section	NRWW Section	No	Yes
NRWW Section	None	Yes	No

Figure 26-1. Read-While-Write vs. No Read-While-Write

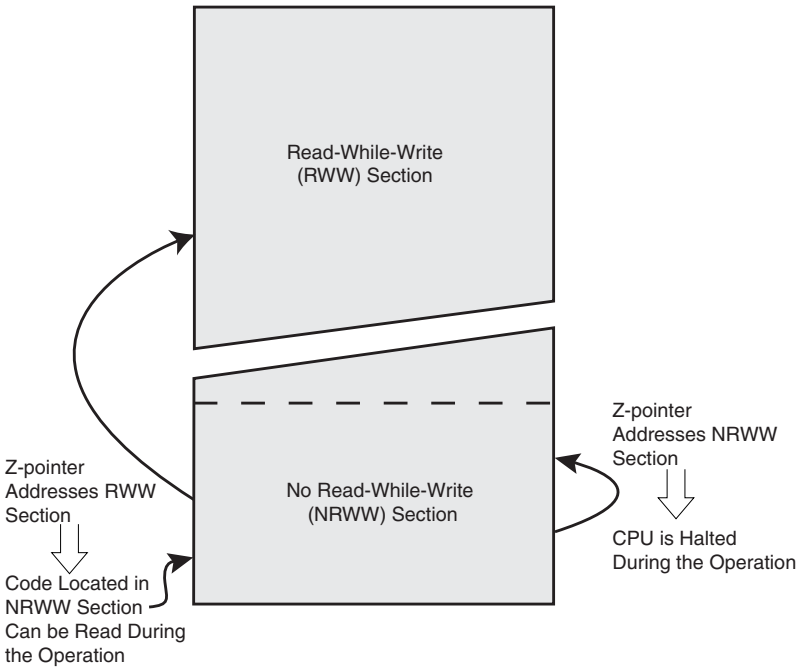
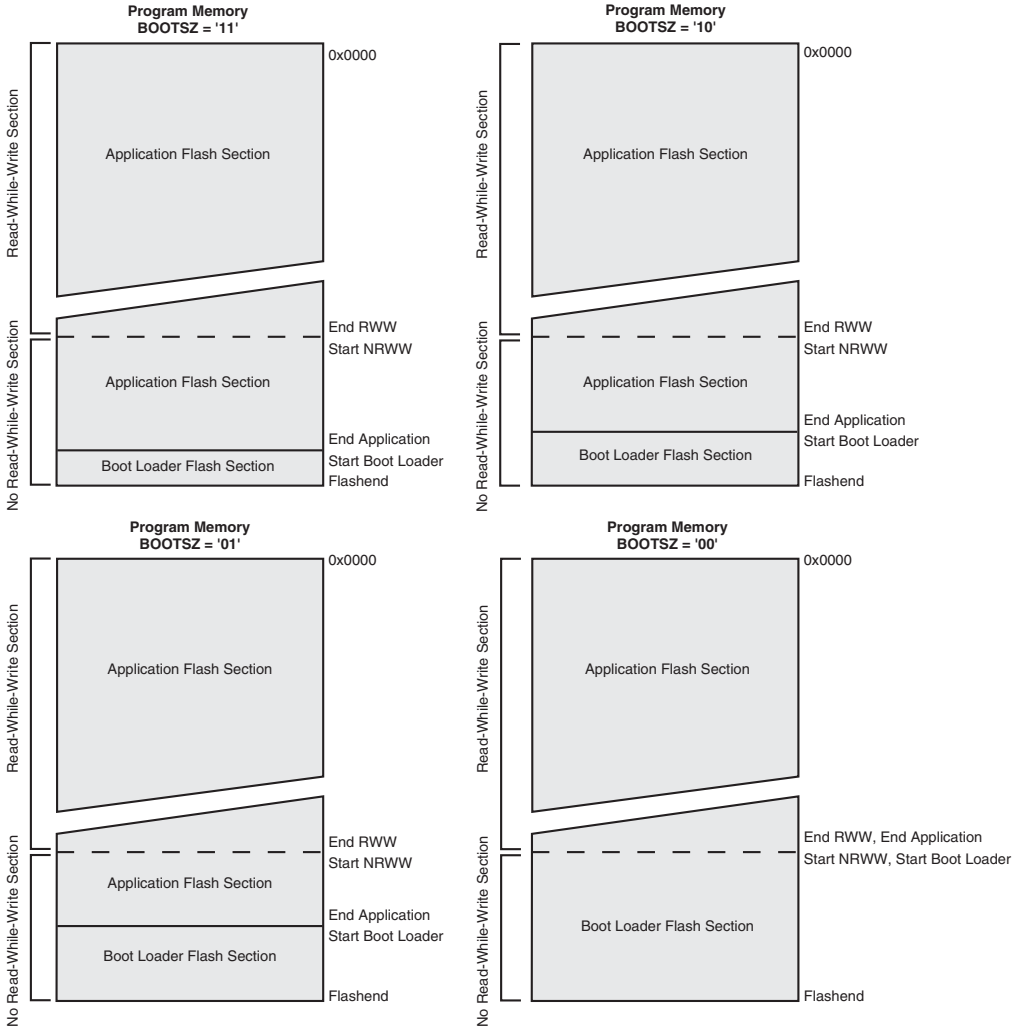


Figure 26-2. Memory Sections



Note: 1. The parameters in the figure above are given in Table 26-7 on page 183.

26.4 Boot Loader Lock Bits

If no Boot Loader capability is needed, the entire Flash is available for application code. The Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

The user can select:

- To protect the entire Flash from a software update by the MCU.
- To protect only the Boot Loader Flash section from a software update by the MCU.
- To protect only the Application Flash section from a software update by the MCU.
- Allow software update in the entire Flash.

See Table 26-2 and Table 26-3 for further details. The Boot Lock bits can be set in software and in Serial or Parallel Programming mode, but they can be cleared by a Chip Erase command only. The general Write Lock (Lock Bit mode 2) does not control the programming of the Flash memory by SPM instruction. Similarly, the general Read/Write Lock (Lock Bit mode 1) does not control reading nor writing by LPM/SPM, if it is attempted.

Table 26-2. Boot Lock Bit0 Protection Modes (Application Section)[1]

BLB0 Mode	BLB02	BLB01	Protection
1	1	1	No restrictions for SPM or LPM accessing the Application section.
2	1	0	SPM is not allowed to write to the Application section.
3	0	0	SPM is not allowed to write to the Application section, and LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.
4	0	1	LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.

Note: 1. "1" means unprogrammed, "0" means programmed

Table 26-3. Boot Lock Bit1 Protection Modes (Boot Loader Section)[1]

BLB1 Mode	BLB12	BLB11	Protection
1	1	1	No restrictions for SPM or LPM accessing the Boot Loader section.
2	1	0	SPM is not allowed to write to the Boot Loader section.
3	0	0	SPM is not allowed to write to the Boot Loader section, and LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.
4	0	1	LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.

Note: 1. "1" means unprogrammed, "0" means programmed

26.5 Entering the Boot Loader Program

Entering the Boot Loader takes place by a jump or call from the application program. This may be initiated by a trigger such as a command received via the TWI interface. Alternatively, the Boot Reset Fuse can be programmed so that the Reset Vector is pointing to the Boot Flash start address after a reset. In this case, the Boot Loader is started after a reset. After the application code is loaded, the program can start executing the application code. Note that the fuses cannot be changed by the MCU itself. This means that once the Boot Reset Fuse is programmed, the Reset Vector will always point to the Boot Loader Reset and the fuse can only be changed through the serial or parallel programming interface.

Table 26-4. Boot Reset Fuse(1)

BOOTRST	Reset Address
1	Reset Vector = Application Reset (address 0x0000)
0	Reset Vector = Boot Loader Reset (see Table 26-7 on page 183)

Note: 1. “1” means unprogrammed, “0” means programmed

26.5.1 Store Program Memory Control and Status Register – SPMCSR

The Store Program Memory Control and Status Register contains the control bits needed to control the Boot Loader operations.

Bit	7	6	5	4	3	2	1	0	
	SPMIE	RWWSB	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 180 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 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 180 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 “100001”, “010001”, “001001”, “000101”, “000011” or “000001” in the lower five bits will have no effect.

26.6 Addressing the Flash During Self-Programming

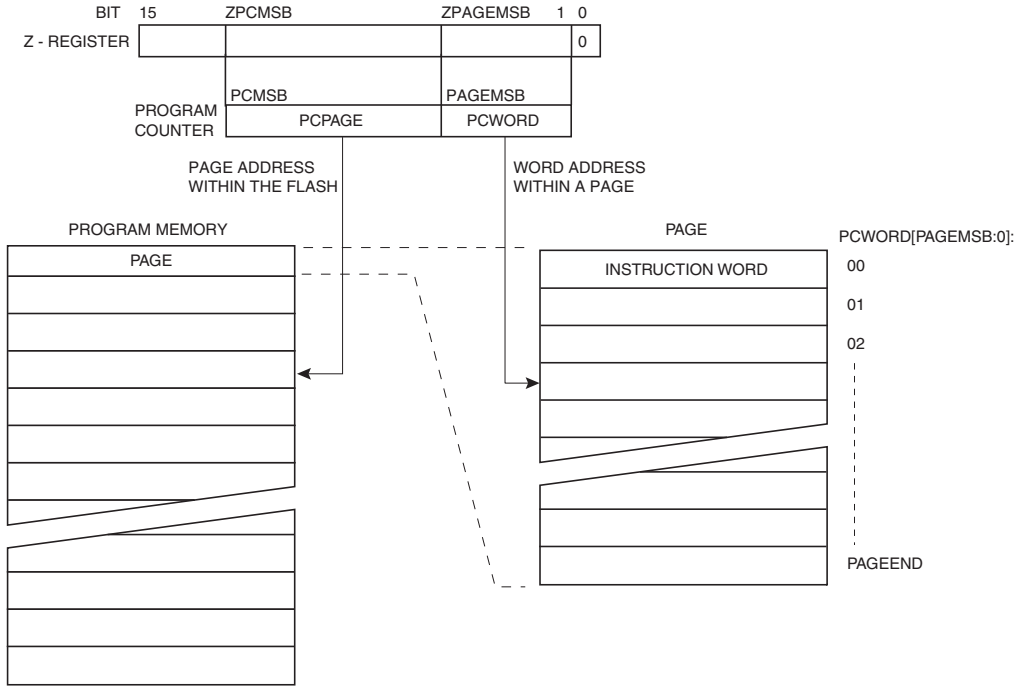
The Z-pointer is used to address the SPM commands.

Bit	15	14	13	12	11	10	9	8
ZH (R31)	Z15	Z14	Z13	Z12	Z11	Z10	Z9	Z8
ZL (R30)	Z7	Z6	Z5	Z4	Z3	Z2	Z1	Z0
	7	6	5	4	3	2	1	0

Since the Flash is organized in pages (see ”Fuse Bits” on page 186), the Program Counter can be treated as having two different sections. One section, consisting of the least significant bits, is addressing the words within a page, while the most significant bits are addressing the pages. This is shown in Figure 26-3. Note that the Page Erase and Page Write operations are addressed independently. Therefore it is of major importance that the Boot Loader software addresses the same page in both the Page Erase and Page Write operation. Once a programming operation is initiated, the address is latched and the Z-pointer can be used for other operations.

The only SPM operation that does not use the Z-pointer is Setting the Boot Loader Lock bits. The content of the Z-pointer is ignored and will have no effect on the operation. The LPM instruction does also use the Z-pointer to store the address. Since this instruction addresses the Flash byte-by-byte, also the LSB (bit Z0) of the Z-pointer is used.

Figure 26-3. Addressing the Flash During SPM[1]



Note: 1. The different variables used in Figure 26-3 are listed in Table 26-9 on page 184.

26.7 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 182 for an assembly code example.

26.7.1 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.

26.7.2 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.

26.7.3 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 will be ignored during this operation.

- Page Write to the RWW section: The NRWW section can be read during the Page Write.
- Page Write to the NRWW section: The CPU is halted during the operation.

26.7.4 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 49.

26.7.5 Consideration While Updating BLS

Special care must be taken if the user allows the Boot Loader section to be updated by leaving Boot Lock bit11 unprogrammed. An accidental write to the Boot Loader itself can corrupt the entire Boot Loader, and further software updates might be impossible. If it is not necessary to change the Boot Loader software itself, it is recommended to program the Boot Lock bit11 to protect the Boot Loader software from any internal software changes.

26.7.6 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 49, or the interrupts must be disabled. Before addressing the RWW section after the programming is completed, the user software must clear the RWWSB by writing the RWWSRE. See ”Simple Assembly Code Example for a Boot Loader” on page 182 for an example.

26.7.7 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 26-2 and Table 26-3 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 lO_{ck} 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.

26.7.8 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 (EEWE) in the EECR Register and verifies that the bit is cleared before writing to the SPMCSR Register.

26.7.9 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 LPM instruction is executed within three CPU cycles after the BLBSET and SPMEN bits are set in SPMCSR, the value of the Lock bits will be loaded in the destination register. The BLBSET and SPMEN bits will auto-clear upon completion of reading the Lock bits or if no LPM instruction is executed within three CPU cycles or no SPM instruction is executed within four CPU cycles. When BLBSET and SPMEN are cleared, LPM will work as described in the ”AVR Instruction Set” description.

Bit	7	6	5	4	3	2	1	0
Rd	–	–	BLB12	BLB11	BLB02	BLB01	LB2	LB1

The algorithm for reading the Fuse Low byte is similar to the one described above for reading the Lock bits. To read the Fuse Low byte, load the Z-pointer with 0x0000 and set the BLBSET and SPMEN bits in SPMCSR. When an LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in the SPMCSR, the value of the Fuse Low byte (FLB) will be loaded in the destination register as shown below. Refer to Table 27-4 on page 186 for a detailed description and mapping of the Fuse Low byte.

Bit	7	6	5	4	3	2	1	0
Rd	FLB7	FLB6	FLB5	FLB4	FLB3	FLB2	FLB1	FLB0

Similarly, when reading the Fuse High byte, load 0x0003 in the Z-pointer. When an LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in the SPMCSR, the value of the Fuse High byte (FHB) will be loaded in the destination register as shown below. Refer to Table 27-3 on page 186 for detailed description and mapping of the Fuse High byte.

Bit	7	6	5	4	3	2	1	0
Rd	FHB7	FHB6	FHB5	FHB4	FHB3	FHB2	FHB1	FHB0

Fuse and Lock bits that are programmed, will be read as zero. Fuse and Lock bits that are unprogrammed, will be read as one.

26.7.10 Reading the Signature Row from Software

To read the Signature Row from software, load the Z-pointer with the signature byte address given in Table 26-5 and set the SIGRD and SPMEN bits in SPMCSR. When an LPM instruction is executed within three CPU cycles after the SIGRD and SPMEN bits are set in SPMCSR, the signature byte value will be loaded in the destination register. The SIGRD and SPMEN bits will auto-clear upon completion of reading the Signature Row Lock bits or if no LPM instruction is executed within three CPU cycles. When SIGRD and SPMEN are cleared, LPM will work as described in the ”AVR Instruction Set” description.

Table 26-5. Signature Row Addressing

Signature Byte	Z-Pointer Address
Device Signature Byte 1	0x0000
Device Signature Byte 2	0x0002
Device Signature Byte 3	0x0004
Fast RC Oscillator Calibration Byte	0x0001
Slow RC Oscillator Calibration Word	High byte: 0x0007 Low byte: 0x0006
Bandgap PTAT Current Calibration Byte	0x0009
V-ADC Cell1 Gain Calibration Word	High byte: 0x0011 Low byte: 0x0010
V-ADC Cell2 Gain Calibration Word	High byte: 0x0013 Low byte: 0x0012
V-ADC Cell3 Gain Calibration Word	High byte: 0x0015 Low byte: 0x0014
V-ADC Cell4 Gain Calibration Word	High byte: 0x0017 Low byte: 0x0016
VPTAT Calibration High Word	High byte: 0x001B Low byte: 0x001A

All other addresses are reserved for future use.

26.7.11 Programming Time for Flash when Using SPM

The Fast RC Oscillator is used to time Flash accesses. Table 26-6 shows the typical programming time for Flash accesses from the CPU.

Table 26-6. SPM Programming Time

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

26.7.12 Simple Assembly Code Example for a Boot Loader

```
  ;-the routine writes one page of data from RAM to Flash
  ; the first data location in RAM is pointed to by the Y pointer
  ; the first data location in Flash is pointed to by the Z-pointer
  ;-error handling is not included
  ;-the routine must be placed inside the Boot space
  ; (at least the Do_spm sub routine). Only code inside NRWW section
  ; can be read during Self-Programming (Page Erase and Page Write).
  ;-registers used: r0, r1, temp1 (r16), temp2 (r17), looplo (r24),
  ; loophi (r25), spmcrval (r20)
  ; storing and restoring of registers is not included in the routine
  ; register usage can be optimized at the expense of code size
  ;-It is assumed that either the interrupt table is moved to the
  ; Boot loader section or that the interrupts are disabled.
.equ PAGESIZEB = PAGESIZE*2   ;PAGESIZEB is page size in BYTES, not words
.org SMALLBOOTSTART
Write_page:
  ; Page Erase
  ldi  spmcrval, (1<<PGERS) | (1<<SPMEN)
  call Do_spm

  ; re-enable the RWW section
  ldi  spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call Do_spm

  ; transfer data from RAM to Flash page buffer
  ldi  looplo, low(PAGESIZEB)   ;init loop variable
  ldi  loophi, high(PAGESIZEB)  ;not required for PAGESIZEB<=256
Wrloop:
  ld   r0, Y+
  ld   r1, Y+
  ldi  spmcrval, (1<<SPMEN)
  call Do_spm
  adiw ZH:ZL, 2
  sbiw loophi:looplo, 2         ;use subi for PAGESIZEB<=256
  brne Wrloop

  ; execute Page Write
  subi ZL, low(PAGESIZEB)       ;restore pointer
  sbci ZH, high(PAGESIZEB)      ;not required for PAGESIZEB<=256
  ldi  spmcrval, (1<<PGWRT) | (1<<SPMEN)
  call Do_spm

  ; re-enable the RWW section
  ldi  spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call Do_spm

  ; read back and check, optional
  ldi  looplo, low(PAGESIZEB)   ;init loop variable
  ldi  loophi, high(PAGESIZEB)  ;not required for PAGESIZEB<=256
  subi YL, low(PAGESIZEB)       ;restore pointer
  sbci YH, high(PAGESIZEB)
Rdloop:
  lpm  r0, Z+
  ld   r1, Y+
  cpse r0, r1
  jmp  Error
```

```
  sbiw loophi:looplo, 1          ;use subi for PAGESIZEB<=256
  brne Rdloop
  ; return to RWW section
  ; verify that RWW section is safe to read
Return:
  in   temp1, SPMCSR
  sbrs temp1, RWWSB     ; If RWWSB is set, the RWW section is not ready yet
  ret
  ; re-enable the RWW section
  ldi  spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call Do_spm
  rjmp Return

Do_spm:
  ; check for previous SPM complete
Wait_spm:
  in   temp1, SPMCSR
  sbrc temp1, SPMEN
  rjmp Wait_spm
  ; input: spmcrval determines SPM action
  ; disable interrupts if enabled, store status
  in   temp2, SREG
  cli
  ; check that no EEPROM write access is present
Wait_ee:
  sbic EECR, EEWE
  rjmp Wait_ee
  ; SPM timed sequence
  out  SPMCSR, spmcrval
  spm
  ; restore SREG (to enable interrupts if originally enabled)
  out  SREG, temp2
  ret
```

26.7.13 ATmega406 Boot Loader Parameters

In Table 26-7 through Table 26-9, the parameters used in the description of the Self-Programming are given

Table 26-7. Boot Size Configuration[1]

BOOTSZ1	BOOTSZ0	Boot Size	Pages	Application Flash Section	Boot Loader Flash Section	End Application Section	Boot Reset Address (Start Boot Loader Section)
1	1	256 words	4	0x0000 - 0x4EFF	0x4F00 - 0x4FFF	0x4EFF	0x4F00
1	0	512 words	8	0x0000 - 0x4DFF	0x4E00 - 0x4FFF	0x4DFF	0x4E00
0	1	1024 words	16	0x0000 - 0x4BFF	0x4C00 - 0x4FFF	0x4BFF	0x4C00
0	0	2048 words	32	0x0000 - 0x47FF	0x4800 - 0x4FFF	0x47FF	0x4800

Note: 1. The different BOOTSZ Fuse configurations are shown in Figure 26-2

Table 26-8. Read-While-Write Limit[1]

Section	Pages	Address
Read-While-Write section (RWW)	288	0x0000 - 0x47FF
No Read-While-Write section (NRWW)	32	0x4800 - 0x4FFF

Note: 1. For details about these two section, see "NRWW – No Read-While-Write Section" on page 171 and "RWW – Read-While-Write Section" on page 171.

Table 26-9. Explanation of different variables used in Figure 26-3 and the mapping to the Z-pointer[1]

Variable		Corresponding Z-value	Description
PCMSB	14		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		Z15	Bit in Z-register that is mapped to PCMSB. Because Z0 is not used, the ZPCMSB equals PCMSB + 1.
ZPAGEMSB		Z6	Bit in Z-register that is mapped to PCMSB. Because Z0 is not used, the ZPAGEMSB equals PAGEMSB + 1.
PCPAGE	PC[14:6]	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 LPM instruction. See "Addressing the Flash During Self-Programming" on page 177 for details about the use of Z-pointer during Self-Programming.

27. Memory Programming

27.1 Program And Data Memory Lock Bits

The ATmega406 provides six Lock bits which can be left unprogrammed (“1”) or can be programmed (“0”) to obtain the additional features listed in Table 27-2. The Lock bits can only be erased to “1” with the Chip Erase command.

Table 27-1. Lock Bit Byte[1]

Lock Bit Byte	Bit No	Description	Default Value
	7	–	1 (unprogrammed)
	6	–	1 (unprogrammed)
BLB12	5	Boot Lock bit	1 (unprogrammed)
BLB11	4	Boot Lock bit	1 (unprogrammed)
BLB02	3	Boot Lock bit	1 (unprogrammed)
BLB01	2	Boot Lock bit	1 (unprogrammed)
LB2	1	Lock bit	1 (unprogrammed)
LB1	0	Lock bit	1 (unprogrammed)

Note: 1. “1” means unprogrammed, “0” means programmed

Table 27-2. Lock Bit Protection Modes[1][2]

Memory Lock Bits			Protection Type
LB Mode	**LB2**	**LB1**	
1	1	1	No memory lock features enabled.
2	1	0	Further programming of the Flash and EEPROM is disabled in Parallel and Serial Programming mode. The Fuse bits are locked in both Serial and Parallel Programming mode.[1]
3	0	0	Further programming and verification of the Flash and EEPROM is disabled in Parallel and Serial Programming mode. The Boot Lock bits and Fuse bits are locked in both Serial and Parallel Programming mode.[1]
BLB0 Mode	**BLB02**	**BLB01**	
1	1	1	No restrictions for SPM or LPM accessing the Application section.
2	1	0	SPM is not allowed to write to the Application section.
3	0	0	SPM is not allowed to write to the Application section, and LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.
4	0	1	LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.

Table 27-2. Lock Bit Protection Modes[1][2] (Continued)

Memory Lock Bits			Protection Type
BLB1 Mode	BLB12	BLB11	
1	1	1	No restrictions for SPM or LPM accessing the Boot Loader section.
2	1	0	SPM is not allowed to write to the Boot Loader section.
3	0	0	SPM is not allowed to write to the Boot Loader section, and LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.
4	0	1	LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.

Notes: 1. Program the Fuse bits and Boot Lock bits before programming the LB1 and LB2.
2. "1" means unprogrammed, "0" means programmed

27.2 Fuse Bits

The ATmega406 has two Fuse bytes. Table 27-3 - Table 27-4 describe briefly the functionality of all the fuses and how they are mapped into the Fuse bytes. Note that the fuses are read as logical zero, "0", if they are programmed.

Table 27-3. Fuse High Byte

Fuse High Byte	Bit No	Description	Default Value
–	7	–	1
–	6	–	1
–	5	–	1
–	4	–	1
–	3	–	1
–	2	–	1
OCDEN[1]	1	Enable OCD	1 (unprogrammed, OCD disabled)
JTAGEN	0	Enable JTAG	0 (programmed, JTAG enabled)

Notes: 1. Never ship a product with the OCDEN Fuse programmed regardless of the setting of Lock bits and JTAGEN Fuse. A programmed OCDEN Fuse enables some parts of the clock system to be running in all sleep modes. This may increase the power consumption.

Table 27-4. Fuse Low Byte

Fuse Low Byte	Bit No	Description	Default Value
WDTON[3]	7	Watchdog Timer always on	1 (unprogrammed)
EESAVE	6	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed, EEPROM not preserved)
BOOTSZ1	5	Select Boot Size (see Table 26-7 on page 183 for details)	0 (programmed)[2]

Table 27-4. Fuse Low Byte (Continued)

Fuse Low Byte	Bit No	Description	Default Value
BOOTSZ0	4	Select Boot Size (see Table 26-7 on page 183 for details)	0 (programmed)(2)
BOOTRST	3	Select Reset Vector	1 (unprogrammed)
SUT1	2	Select start-up time	1 (unprogrammed)(1)
SUT0	1	Select start-up time	0 (programmed)(1)
CKSEL	0	Clock Selection	1 (unprogrammed)(4)

Notes:
1. The default value of SUT1:0 results in maximum start-up time for the default clock source. See Table 6-2 on page 28 for details.
2. The default value of BOOTSZ1:0 results in maximum Boot Size. See Table 26-7 on page 183 for details.
3. See "Watchdog Timer Control Register - WDTCSR" on page 45 for details.
4. When unpgrogrammed, Internal RC Oscillator is used. Programming this fuse is for test purpose only, and should not be used in application.

The status of the Fuse bits is not affected by Chip Erase. Note that the Fuse bits are locked if Lock bit1 (LB1) is programmed. Program the Fuse bits before programming the Lock bits.

27.2.1 Latching of Fuses

The fuse values are latched when the device enters programming mode and changes of the fuse values will have no effect until the part leaves Programming mode. This does not apply to the EESAVE Fuse which will take effect once it is programmed. The fuses are also latched on Power-up in Normal mode.

27.3 Signature Bytes

All Atmel microcontrollers have a three-byte signature code which identifies the device. This code can be read in both serial and parallel mode, also when the device is locked. The three bytes reside in a separate address space.

For the ATmega406 the signature bytes are:

1. 0x000: 0x1E (indicates manufactured by Atmel).
2. 0x001: 0x95 (indicates 40KB Flash memory).
3. 0x002: 0x07(indicates ATmega406 device when 0x001 is 0x95).

27.4 Calibration Bytes

The ATmega406 has calibration bytes for the Fast RC Oscillator, Slow RC Oscillator, internal voltage reference, internal temperature reference and each differential cell voltage input. These bytes reside in the high bytes in the signature address space. During Reset, the calibration byte for the Fast RC Oscillator is automatically written into the corresponding calibration register. The other calibration bytes should be handled by the application software. See "Reading the Signature Row from Software" on page 180 for details.

27.5 Page Size

Table 27-5. No. of Words in a Page and No. of Pages in the Flash

Flash Size	Page Size	PCWORD	No. of Pages	PCPAGE	PCMSB
20K words (40K bytes)	64 words	PC[5:0]	320	PC[14:6]	14

Table 27-6. No. of Words in a Page and No. of Pages in the EEPROM

EEPROM Size	Page Size	PCWORD	No. of Pages	PCPAGE	EEAMSB
512 bytes	4 bytes	EEA[1:0]	128	EEA[8:2]	8

27.6 Parallel Programming Parameters, Pin Mapping, and Commands

This section describes how to parallel program and verify Flash Program memory, EEPROM Data memory, Memory Lock bits, and Fuse bits in the ATmega406. Pulses are assumed to be at least 250 ns unless otherwise noted.

27.6.1 Signal Names

In this section, some pins of the ATmega406 are referenced by signal names describing their functionality during parallel programming, see Figure 27-1 and Table 27-7. Pins not described in the following table are referenced by pin names.

The XA1/XA0 pins determine the action executed when the XTAL1 pin is given a positive pulse. The bit coding is shown in Table 27-9.

When pulsing $\overline{WR}$ or $\overline{OE}$, the command loaded determines the action executed. The different Commands are shown in Table 27-10.

Figure 27-1. Parallel Programming

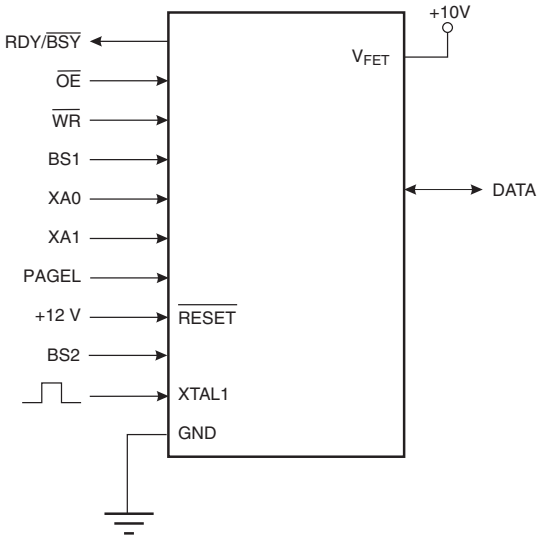


Table 27-7. Pin Name Mapping

Signal Name in Programming Mode	Pin Name	I/O	Function
BS2	PA0	I	Byte Select 2 (“0” selects low byte, “1” selects 2’nd high byte).
RDY/$\overline{\text{BSY}}$	PA1	O	0: Device is busy programming, 1: Device is ready for new command.
$\overline{\text{OE}}$	PA2	I	Output Enable (Active low).
$\overline{\text{WR}}$	PA3	I	Write Pulse (Active low).
BS1	PA4	I	Byte Select 1 (“0” selects low byte, “1” selects high byte).
XA0	PA5	I	XTAL Action Bit 0
XA1	PA6	I	XTAL Action Bit 1
PAGEL	PA7	I	Program Memory and EEPROM data Page Load.
DATA	PB7:0	I/O	Bi-directional Data bus (Output when $\overline{\text{OE}}$ is low).

Table 27-8. Pin Values Used to Enter Programming Mode

Pin	Symbol	Value
PAGEL	Prog_enable[3]	0
XA1	Prog_enable[2]	0
XA0	Prog_enable[1]	0
BS1	Prog_enable[0]	0

Table 27-9. XA1 and XA0 Coding

XA1	XA0	Action when XTAL1 is Pulsed
0	0	Load Flash or EEPROM Address (High or low address byte determined by BS1).
0	1	Load Data (High or Low data byte for Flash determined by BS1).
1	0	Load Command
1	1	No Action, Idle

Table 27-10. Command Byte Bit Coding

Command Byte	Command Executed
1000 0000	Chip Erase
0100 0000	Write Fuse bits
0010 0000	Write Lock bits

Table 27-10. Command Byte Bit Coding (Continued)

Command Byte	Command Executed
0001 0000	Write Flash
0001 0001	Write EEPROM
0000 1000	Read Signature Bytes and Calibration byte
0000 0100	Read Fuse and Lock bits
0000 0010	Read Flash
0000 0011	Read EEPROM

27.7 Parallel Programming

27.7.1 Enter Programming Mode

The following algorithm puts the device in parallel programming mode:

1. Make sure the chip is started as explained in Section 8.3 ”Power-on Reset and Charger Connect” on page 38.
2. Set $\overline{\text{RESET}}$ to “0” and toggle XTAL1 at least six times.
3. Set the Prog_enable pins listed in Table 27-8 on page 189 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.

27.7.2 Considerations for Efficient Programming

The loaded command and address are retained in the device during programming. For efficient programming, the following should be considered.

- The command needs only be loaded once when writing or reading multiple memory locations.
- Skip writing the data value 0xFF, that is the contents of the entire EEPROM (unless the EESAVE Fuse is programmed) and Flash after a Chip Erase.
- Address high byte needs only be loaded before programming or reading a new 256 word window in Flash or 256 byte EEPROM. This consideration also applies to Signature bytes reading.

27.7.3 Chip Erase

The Chip Erase will erase the Flash and EEPROM[1] memories plus Lock bits. The Lock bits are not reset until the program memory has been completely erased. The Fuse bits are not changed. A Chip Erase must be performed before the Flash and/or EEPROM are reprogrammed.

Note: 1. The EEPRPOM memory is preserved during Chip Erase if the EESAVE Fuse is programmed.

Load Command “Chip Erase”

1. Set XA1, XA0 to “10”. This enables command loading.
2. Set BS1 to “0”.
3. Set DATA to “1000 0000”. This is the command for Chip Erase.
4. Give XTAL1 a positive pulse. This loads the command.
5. Give $\overline{\text{WR}}$ a negative pulse. This starts the Chip Erase. RDY/$\overline{\text{BSY}}$ goes low.

6. Wait until RDY/$\overline{\text{BSY}}$ goes high before loading a new command.

27.7.4 Programming the Flash

The Flash is organized in pages, see Table 27-5 on page 188. When programming the Flash, the program data is latched into a page buffer. This allows one page of program data to be programmed simultaneously. The following procedure describes how to program the entire Flash memory:

A. Load Command “Write Flash”

1. Set XA1, XA0 to “10”. This enables command loading.
2. Set BS1 to “0”.
3. Set DATA to “0001 0000”. This is the command for Write Flash.
4. Give XTAL1 a positive pulse. This loads the command.

B. Load Address Low byte

1. Set XA1, XA0 to “00”. This enables address loading.
2. Set BS1 to “0”. This selects low address.
3. Set DATA = Address low byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the address low byte.

C. Load Data Low Byte

1. Set XA1, XA0 to “01”. This enables data loading.
2. Set DATA = Data low byte (0x00 - 0xFF).
3. Give XTAL1 a positive pulse. This loads the data byte.

D. Load Data High Byte

1. Set BS1 to “1”. This selects high data byte.
2. Set XA1, XA0 to “01”. This enables data loading.
3. Set DATA = Data high byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the data byte.

E. Latch Data

1. Set BS1 to “1”. This selects high data byte.
2. Give PAGEL a positive pulse. This latches the data bytes. (See Figure 27-3 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 27-2 on page 192. Note that if less than eight bits are required to address words in the page (pagesize < 256), the most significant bit(s) in the address low byte are used to address the page when performing a Page Write.

G. Load Address High byte

1. Set XA1, XA0 to “00”. This enables address loading.
2. Set BS1 to “1”. This selects high address.
3. Set DATA = Address high byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the address high byte.

H. Program Page

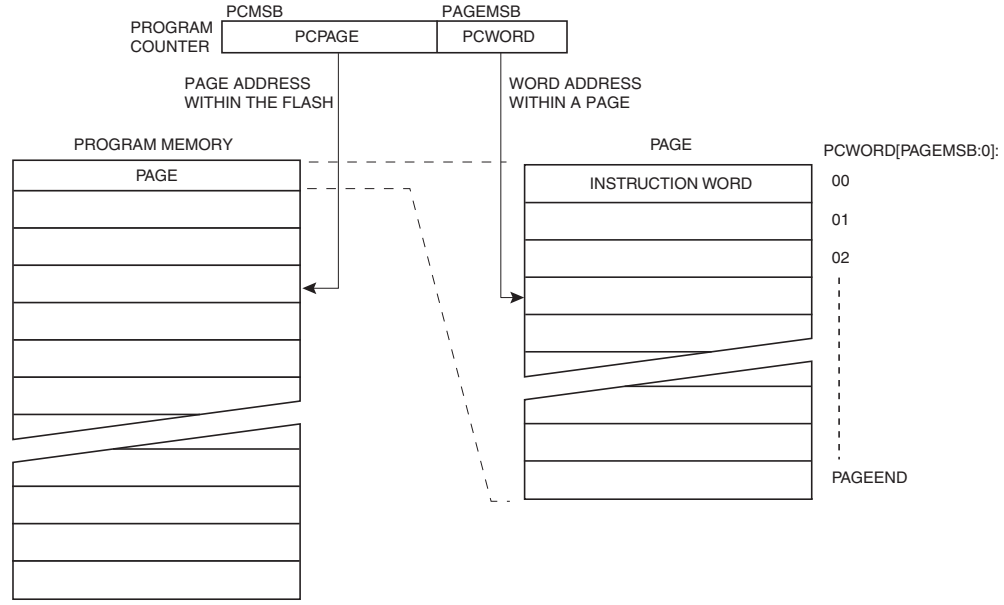
1. Give $\overline{WR}$ a negative pulse. This starts programming of the entire page of data. RDY/$\overline{BSY}$ goes low.
2. Wait until RDY/$\overline{BSY}$ goes high (See Figure 27-3 for signal waveforms).

I. Repeat B through H until the entire Flash is programmed or until all data has been programmed.

J. End Page Programming

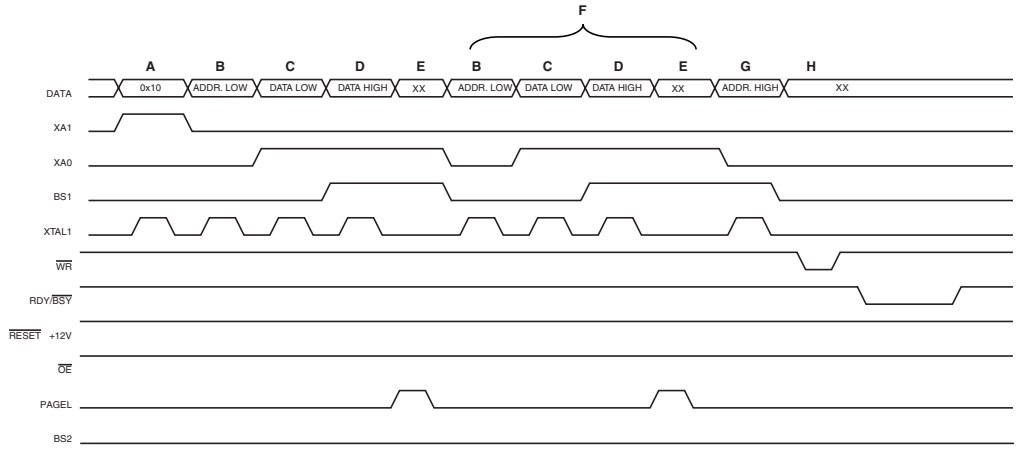
1. 1. Set XA1, XA0 to “10”. This enables command loading.
2. Set DATA to “0000 0000”. This is the command for No Operation.
3. Give XTAL1 a positive pulse. This loads the command, and the internal write signals are reset.

Figure 27-2. Addressing the Flash Which is Organized in Pages[1]



Note: 1. PCPAGE and PCWORD are listed in Table 27-5 on page 188.

Figure 27-3. Programming the Flash Waveforms[1]



Note: 1. “XX” is don’t care. The letters refer to the programming description above.

27.7.5 Programming the EEPROM

The EEPROM is organized in pages, see Table 27-6 on page 188. 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 191 for details on Command, Address and Data loading):

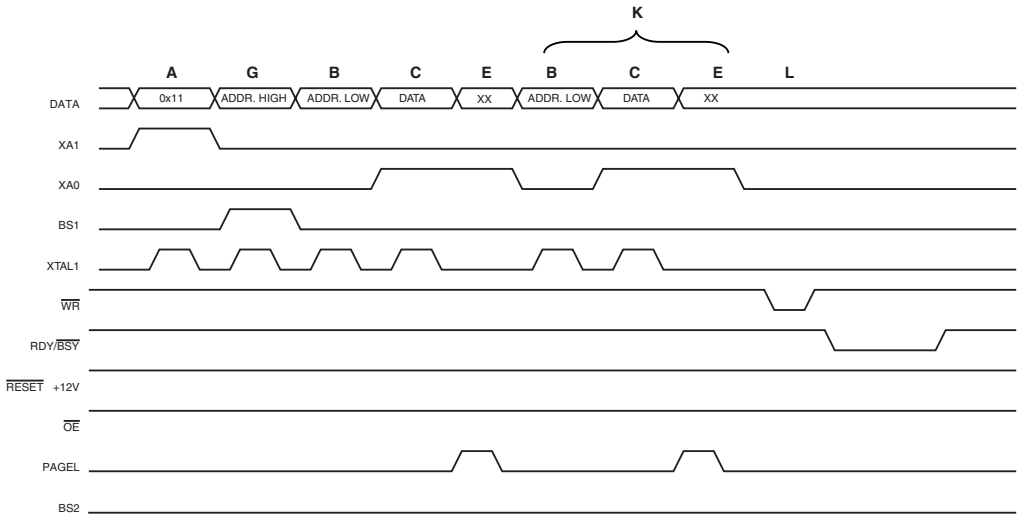
1. A: Load Command "0001 0001".
2. G: Load Address High Byte (0x00 - 0xFF).
3. B: Load Address Low Byte (0x00 - 0xFF).
4. C: Load Data (0x00 - 0xFF).
5. E: Latch data (give PAGEL a positive pulse).

K: Repeat 3 through 5 until the entire buffer is filled.

L: Program EEPROM page

1. Set BS to "0".
2. Give $\overline{WR}$ a negative pulse. This starts programming of the EEPROM page. RDY/$\overline{BSY}$ goes low.
3. Wait until to RDY/$\overline{BSY}$ goes high before programming the next page (See Figure 27-4 for signal waveforms).

Figure 27-4. Programming the EEPROM Waveforms



27.7.6 Reading the Flash

The algorithm for reading the Flash memory is as follows (refer to "Programming the Flash" on page 191 for details on Command and Address loading):

1. A: Load Command "0000 0010".
2. G: Load Address High Byte (0x00 - 0xFF).
3. B: Load Address Low Byte (0x00 - 0xFF).
4. Set $\overline{OE}$ to "0", and BS1 to "0". The Flash word low byte can now be read at DATA.
5. Set BS to "1". The Flash word high byte can now be read at DATA.
6. Set $\overline{OE}$ to "1".

27.7.7 Reading the EEPROM

The algorithm for reading the EEPROM memory is as follows (refer to ”Programming the Flash” on page 191 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”.

27.7.8 Programming the Fuse Low Bits

The algorithm for programming the Fuse Low bits is as follows (refer to ”Programming the Flash” on page 191 for details on Command and Data loading):

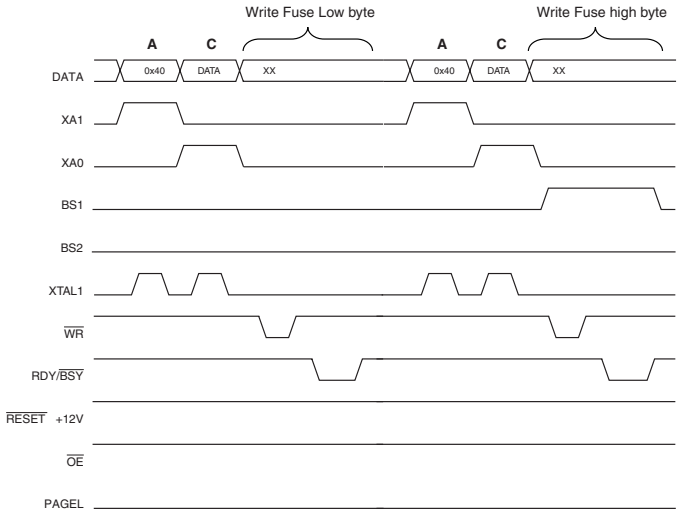
1. A: Load Command “0100 0000”.
2. C: Load Data Low Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.

27.7.9 Programming the Fuse High Bits

The algorithm for programming the Fuse High bits is as follows (refer to ”Programming the Flash” on page 191 for details on Command and Data loading):

1. A: Load Command “0100 0000”.
2. C: Load Data Low Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Set BS1 to “1” and BS2 to “0”. This selects high data byte.
4. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.
5. Set BS1 to “0”. This selects low data byte.

Figure 27-5. Programming the FUSES Waveforms



27.7.10 Programming the Lock Bits

The algorithm for programming the Lock bits is as follows (refer to "Programming the Flash" on page 191 for details on Command and Data loading):

1. A: Load Command "0010 0000".
2. C: Load Data Low Byte. Bit n = "0" programs the Lock bit. If LB mode 3 is programmed (LB1 and LB2 is programmed), it is not possible to program the Boot Lock bits by any External Programming mode.
3. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.

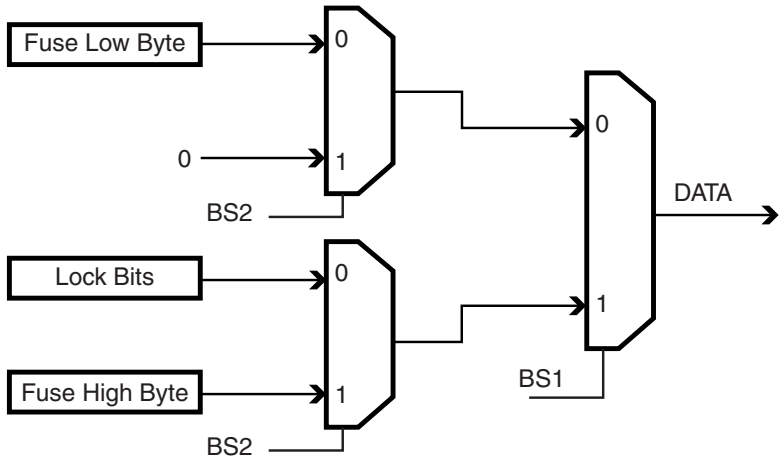
The Lock bits can only be cleared by executing Chip Erase.

27.7.11 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 191 for details on Command loading):

1. A: Load Command "0000 0100".
2. Set $\overline{OE}$ to "0", BS2 to "0" and BS1 to "0". The status of the Fuse Low bits can now be read at DATA ("0" means programmed).
3. Set $\overline{OE}$ to "0", BS2 to "1" and BS1 to "1". The status of the Fuse High bits can now be read at DATA ("0" means programmed).
4. Set $\overline{OE}$ to "0", BS2 to "0" and BS1 to "1". The status of the Lock bits can now be read at DATA ("0" means programmed).
5. Set $\overline{OE}$ to "1".

Figure 27-6. Mapping Between BS1, BS2 and the Fuse and Lock Bits During Read



27.7.12 Reading the Signature Bytes

The algorithm for reading the Signature bytes is as follows (refer to "Programming the Flash" on page 191 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".

27.7.13 Reading the Calibration Byte

The algorithm for reading the Calibration byte is as follows (refer to "Programming the Flash" on page 191 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".

27.7.14 Parallel Programming Characteristics

Figure 27-7. Parallel Programming Timing, Including some General Timing Requirements

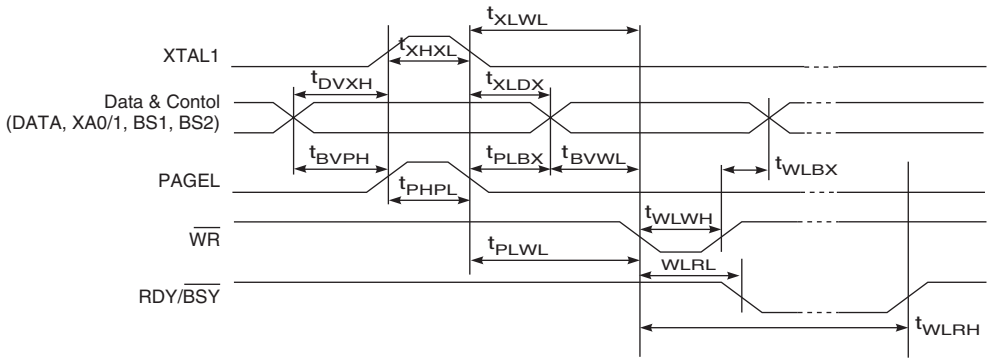
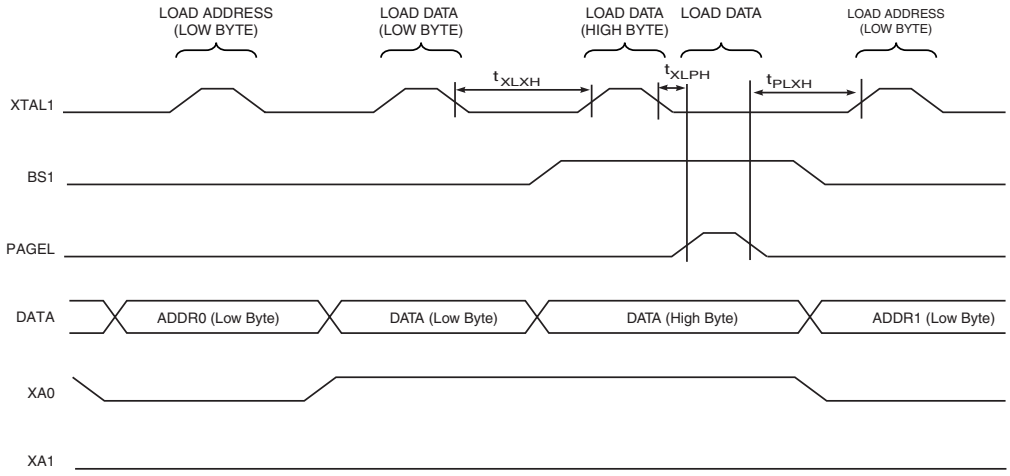
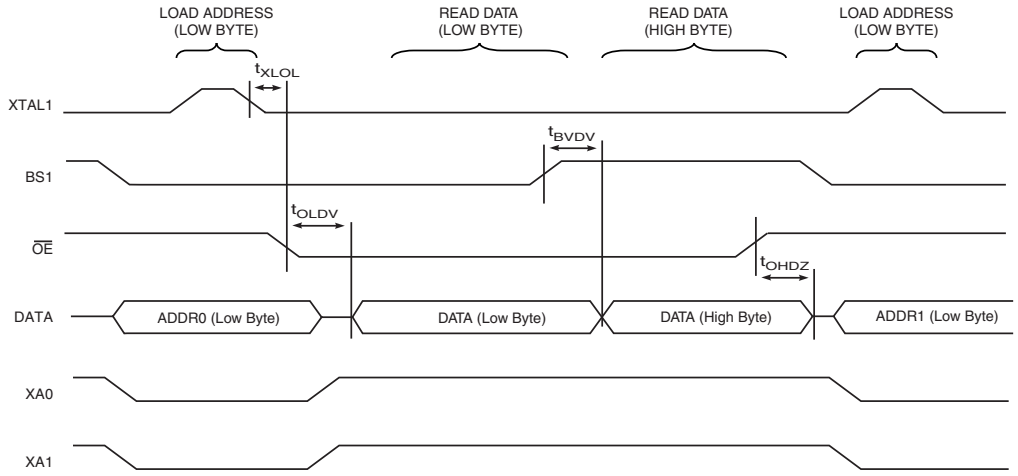


Figure 27-8. Parallel Programming Timing, Loading Sequence with Timing Requirements[1]



Note: 1. The timing requirements shown in Figure 27-7 (i.e., t_{DVXH}, t_{XHXL}, and t_{XLDX}) also apply to loading operation.

Figure 27-9. Parallel Programming Timing, Reading Sequence (within the Same Page) with Timing Requirements[1]



Note: 1. The timing requirements shown in Figure 27-7 (i.e., t_{DVXH}, t_{XHXL}, and t_{XLDX}) also apply to reading operation.

Table 27-11. Parallel Programming Characteristics, V_{FET} = 5V ± 10%

Symbol	Parameter	Min	Typ	Max	Units
V_{PP}	Programming Enable Voltage	11.5		12.5	V
I_{PP}	Programming Enable Current			250	µA
t_{DVXH}	Data and Control Valid before XTAL1 High	67			ns
t_{XLXH}	XTAL1 Low to XTAL1 High	200			ns
t_{XHXL}	XTAL1 Pulse Width High	150			ns
t_{XLDX}	Data and Control Hold after XTAL1 Low	67			ns
t_{XLWL}	XTAL1 Low to $\overline{WR}$ Low	0			ns
t_{XLPH}	XTAL1 Low to PAGEL high	0			ns
t_{PLXH}	PAGEL low to XTAL1 high	150			ns
t_{BVPH}	BS1 Valid before PAGEL High	67			ns
t_{PHPL}	PAGEL Pulse Width High	150			ns
t_{PLBX}	BS1 Hold after PAGEL Low	67			ns
t_{WLBX}	BS2/1 Hold after $\overline{WR}$ Low	67			ns
t_{PLWL}	PAGEL Low to $\overline{WR}$ Low	67			ns
t_{BVWL}	BS1 Valid to $\overline{WR}$ Low	67			ns
t_{WLWH}	$\overline{WR}$ Pulse Width Low	150			ns
t_{WLRL}	$\overline{WR}$ Low to RDY/$\overline{BSY}$ Low	0		1	µs
t_{WLRH}	$\overline{WR}$ Low to RDY/$\overline{BSY}$ High[1]	3.7		4.5	ms
t_{WLRH_CE}	$\overline{WR}$ Low to RDY/$\overline{BSY}$ High for Chip Erase[2]	7.5		9	ms
t_{XLOL}	XTAL1 Low to $\overline{OE}$ Low	0			ns

Table 27-11. Parallel Programming Characteristics, V_{FET} = 5V ± 10% (Continued)

Symbol	Parameter	Min	Typ	Max	Units
t_{BVDV}	BS1 Valid to DATA valid	0		250	ns
t_{OLDV}	$\overline{OE}$ Low to DATA Valid			250	ns
t_{OHDZ}	$\overline{OE}$ High to DATA Tri-stated			250	ns

Notes: 1. t_{WLRH} is valid for the Write Flash, Write EEPROM, Write Fuse bits and Write Lock bits commands.
2. t_{WLRH_CE} is valid for the Chip Erase command.

27.8 Programming via the JTAG Interface

Programming through the JTAG interface requires control of the four JTAG specific pins: TCK, TMS, TDI, and TDO. Control of the reset and clock pins is not required.

To be able to use the JTAG interface, the JTAGEN Fuse must be programmed. The device is default shipped with the fuse programmed. In addition, the JTD bit in MCUCSR must be cleared. Alternatively, if the JTD bit is set, the external reset can be forced low. Then, the JTD bit will be cleared after two chip clocks, and the JTAG pins are available for programming. This provides a means of using the JTAG pins as normal port pins in Running mode while still allowing In-System Programming via the JTAG interface. Note that this technique can not be used when using the JTAG pins for Boundary-scan or On-chip Debug. In these cases the JTAG pins must be dedicated for this purpose.

As a definition in this datasheet, the LSB is shifted in and out first of all Shift Registers.

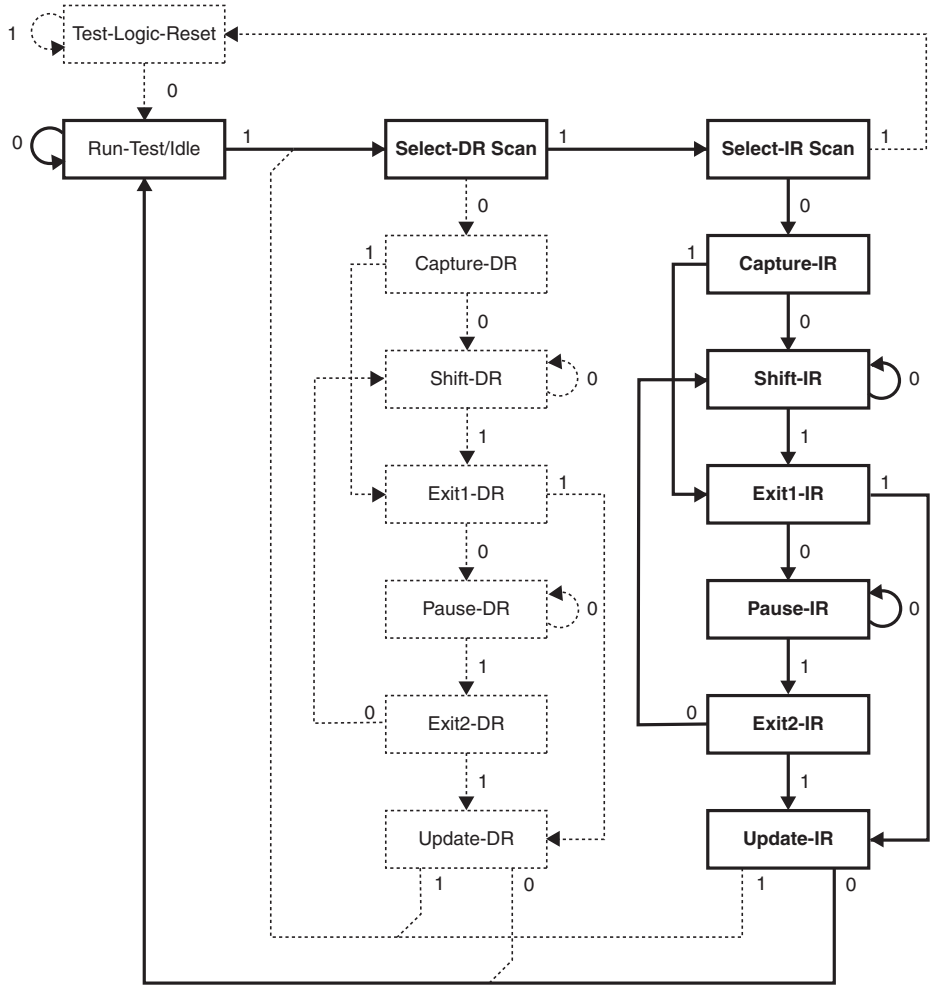
27.8.1 Programming Specific JTAG Instructions

The instruction register is 4-bit wide, supporting up to 16 instructions. The JTAG instructions useful for programming are listed below.

The OPCODE for each instruction is shown behind the instruction name in hex format. The text describes which data register is selected as path between TDI and TDO for each instruction.

The Run-Test/Idle state of the TAP controller is used to generate internal clocks. It can also be used as an idle state between JTAG sequences. The state machine sequence for changing the instruction word is shown in Figure 27-10.

Figure 27-10. State Machine Sequence for Changing the Instruction Word



27.8.2 AVR_RESET (0xC)

The AVR specific public JTAG instruction for setting the AVR device in the Reset mode or taking the device out from the Reset mode. The TAP controller is not reset by this instruction. The one bit Reset Register is selected as data register. Note that the reset will be active as long as there is a logic “one” in the Reset Chain. The output from this chain is not latched.

The active states are:

- Shift-DR: The Reset Register is shifted by the TCK input.

27.8.3 PROG_ENABLE (0x4)

The AVR specific public JTAG instruction for enabling programming via the JTAG port. The 16-bit Programming Enable Register is selected as data register. The active states are the following:

- Shift-DR: The programming enable signature is shifted into the data register.
- Update-DR: The programming enable signature is compared to the correct value, and Programming mode is entered if the signature is valid.

27.8.4 PROG_COMMANDS (0x5)

The AVR specific public JTAG instruction for entering programming commands via the JTAG port. The 15-bit Programming Command Register is selected as data register. The active states are the following:

- Capture-DR: The result of the previous command is loaded into the data register.
- Shift-DR: The data register is shifted by the TCK input, shifting out the result of the previous command and shifting in the new command.
- Update-DR: The programming command is applied to the Flash inputs
- Run-Test/Idle: One clock cycle is generated, executing the applied command (not always required, see Table 27-12 below).

27.8.5 PROG_PAGELOAD (0x6)

The AVR specific public JTAG instruction to directly load the Flash data page via the JTAG port. An 8-bit Flash Data Byte Register is selected as the data register. This is physically the 8 LSBs of the Programming Command Register. The active states are the following:

- Shift-DR: The Flash Data Byte Register is shifted by the TCK input.
- Update-DR: The content of the Flash Data Byte Register is copied into a temporary register. A write sequence is initiated that within 11 TCK cycles loads the content of the temporary register into the Flash page buffer. The AVR automatically alternates between writing the low and the high byte for each new Update-DR state, starting with the low byte for the first Update-DR encountered after entering the PROG_PAGELOAD command. The Program Counter is pre-incremented before writing the low byte, except for the first written byte. This ensures that the first data is written to the address set up by PROG_COMMANDS, and loading the last location in the page buffer does not make the program counter increment into the next page.

27.8.6 PROG_PAGEREAD (0x7)

The AVR specific public JTAG instruction to directly capture the Flash content via the JTAG port. An 8-bit Flash Data Byte Register is selected as the data register. This is physically the 8 LSBs of the Programming Command Register. The active states are the following:

- Capture-DR: The content of the selected Flash byte is captured into the Flash Data Byte Register. The AVR automatically alternates between reading the low and the high byte for each new Capture-DR state, starting with the low byte for the first Capture-DR encountered after entering the PROG_PAGEREAD command. The Program Counter is post-incremented after reading each high byte, including the first read byte. This ensures that the first data is captured from the first address set up by PROG_COMMANDS, and reading the last location in the page makes the program counter increment into the next page.
- Shift-DR: The Flash Data Byte Register is shifted by the TCK input.

27.8.7 Data Registers

The data registers are selected by the JTAG instruction registers described in section ”Programming Specific JTAG Instructions” on page 198. The data registers relevant for programming operations are:

- Reset Register
- Programming Enable Register
- Programming Command Register
- Flash Data Byte Register

27.8.8 Reset Register

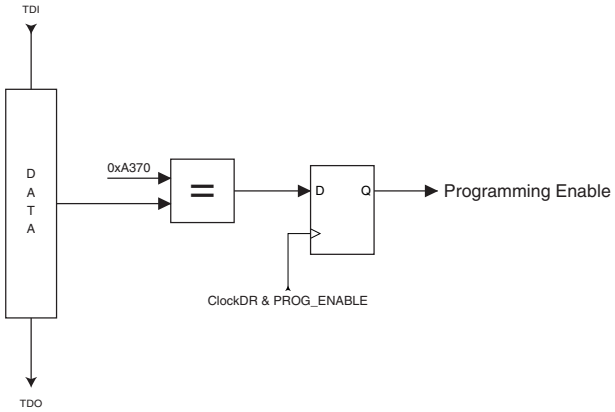
The Reset Register is a Test Data Register used to reset the part during programming. It is required to reset the part before entering Programming mode.

A high value in the Reset Register corresponds to pulling the external reset low. The part is reset as long as there is a high value present in the Reset Register. Depending on the Fuse settings for the clock options, the part will remain reset for a Reset Time-out period (refer to ”Clock Sources” on page 26) after releasing the Reset Register. The output from this data register is not latched, so the reset will take place immediately, as shown in Figure 8-1 on page 38.

27.8.9 Programming Enable Register

The Programming Enable Register is a 16-bit register. The contents of this register is compared to the programming enable signature, binary code 0b1010_0011_0111_0000. When the contents of the register is equal to the programming enable signature, programming via the JTAG port is enabled. The register is reset to 0 on Power-on Reset, and should always be reset when leaving Programming mode.

Figure 27-11. Programming Enable Register



27.8.10 Programming Command Register

The Programming Command Register is a 15-bit register. This register is used to serially shift in programming commands, and to serially shift out the result of the previous command, if any. The JTAG Programming Instruction Set is shown in Table 27-12. The state sequence when shifting in the programming commands is illustrated in Figure 27-13.

Figure 27-12. Programming Command Register

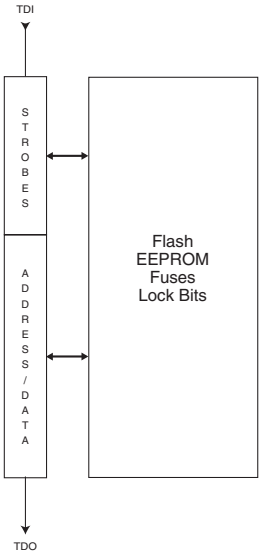


Table 27-12. JTAG Programming Instruction

Set **a** = address high bits, **b** = address low bits, **H** = 0 - Low byte, 1 - High Byte, **o** = data out, **i** = data in, x = don't care

Instruction	TDI Sequence	TDO Sequence	Notes
1a. Chip Erase	0100011_10000000 0110001_10000000 0110011_10000000 0110011_10000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	
1b. Poll for Chip Erase Complete	0110011_10000000	xxxxx**o**x_xxxxxxxx	(2)
2a. Enter Flash Write	0100011_00010000	xxxxxxx_xxxxxxxx	
2b. Load Address High Byte	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)
2c. Load Address Low Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
2d. Load Data Low Byte	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	
2e. Load Data High Byte	0010111_**iiiiiiii**	xxxxxxx_xxxxxxxx	
2f. Latch Data	0110111_00000000 1110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
2g. Write Flash Page	0110111_00000000 0110101_00000000 0110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
2h. Poll for Page Write Complete	0110111_00000000	xxxxx**o**x_xxxxxxxx	(2)
3a. Enter Flash Read	0100011_00000010	xxxxxxx_xxxxxxxx	
3b. Load Address High Byte	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)
3c. Load Address Low Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	

Table 27-12. JTAG Programming Instruction (Continued)
Set (Continued) **a** = address high bits, **b** = address low bits, **H** = 0 - Low byte, 1 - High Byte, **o** = data out, **i** = data in, x

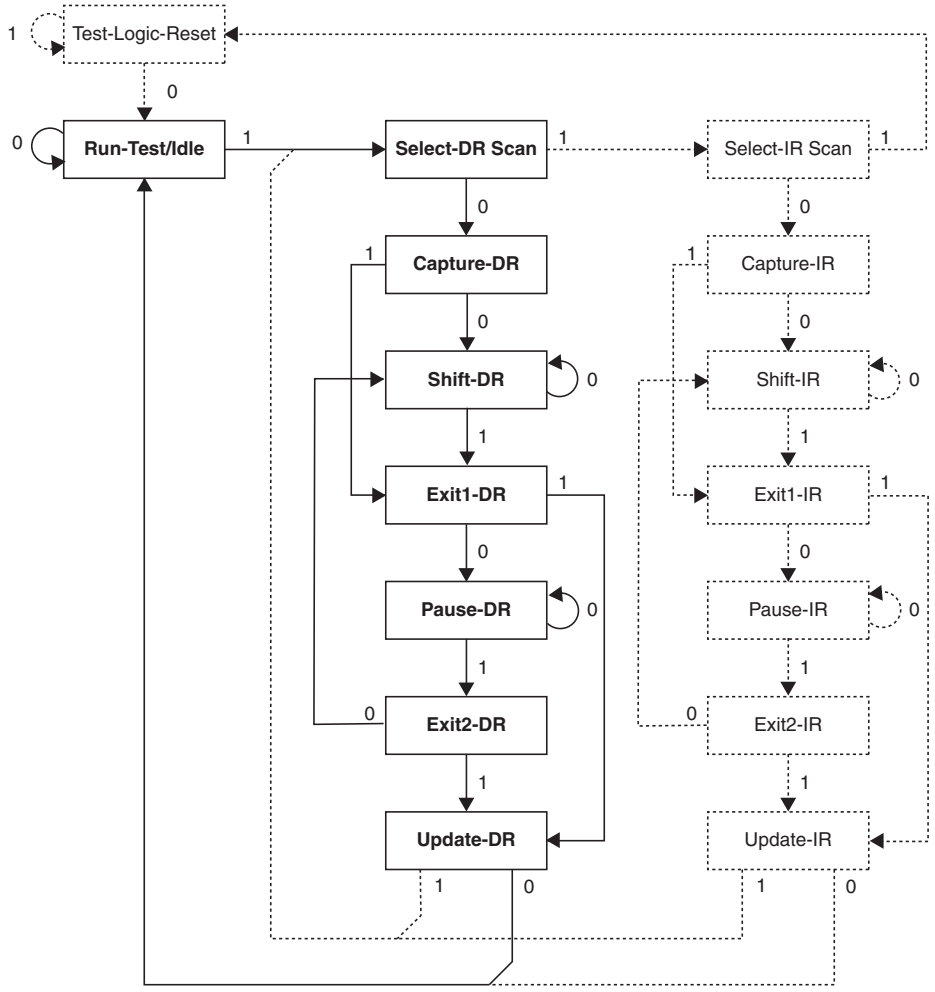
Instruction	TDI Sequence	TDO Sequence	Notes
3d. Read Data Low and High Byte	0110010_00000000 0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo** xxxxxxx_**oooooooo**	 Low byte High byte
4a. Enter EEPROM Write	0100011_00010001	xxxxxxx_xxxxxxxx	
4b. Load Address High Byte	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)
4c. Load Address Low Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
4d. Load Data Byte	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	
4e. Latch Data	0110111_00000000 1110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
4f. Write EEPROM Page	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
4g. Poll for Page Write Complete	0110011_00000000	xxxxx**o**x_xxxxxxxx	(2)
5a. Enter EEPROM Read	0100011_00000011	xxxxxxx_xxxxxxxx	
5b. Load Address High Byte	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)
5c. Load Address Low Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
5d. Read Data Byte	0110011_**bbbbbbbb** 0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
6a. Enter Fuse Write	0100011_01000000	xxxxxxx_xxxxxxxx	
6b. Load Data High Byte[6]	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	(3)
6c. Write Fuse High Byte	0110111_00000000 0110101_00000000 0110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
6d. Poll for Fuse Write Complete	0110111_00000000	xxxxx**o**x_xxxxxxxx	(2)
6e. Load Data Low Byte[7]	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	(3)
6f. Write Fuse Low Byte	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
6g. Poll for Fuse Write Complete	0110011_00000000	xxxxx**o**x_xxxxxxxx	(2)
7a. Enter Lock Bit Write	0100011_00100000	xxxxxxx_xxxxxxxx	
7b. Load Data Byte[8]	0010011_11**iiiiii**	xxxxxxx_xxxxxxxx	(4)

Table 27-12. JTAG Programming Instruction (Continued)
Set (Continued) **a** = address high bits, **b** = address low bits, **H** = 0 - Low byte, 1 - High Byte, **o** = data out, **i** = data in, x

Instruction	TDI Sequence	TDO Sequence	Notes
7c. Write Lock Bits	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
7d. Poll for Lock Bit Write complete	0110011_00000000	xxxxx**o**x_xxxxxxxx	(2)
8a. Enter Fuse/Lock Bit Read	0100011_00000100	xxxxxxx_xxxxxxxx	
8b. Read Fuse High Byte[6]	0111110_00000000 0111111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
8c. Read Fuse Low Byte[7]	0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
8d. Read Lock Bits[8]	0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xx**oooooo**	(5)
8e. Read Fuses and Lock Bits	0111010_00000000 0110010_00000000 0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo** xxxxxxx_**oooooooo** xxxxxxx_**oooooooo**	(5) Fuse High byte Fuse Low byte Lock bits
9a. Enter Signature Byte Read	0100011_00001000	xxxxxxx_xxxxxxxx	
9b. Load Address Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
9c. Read Signature Byte	0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
10a. Enter Calibration Byte Read	0100011_00001000	xxxxxxx_xxxxxxxx	
10b. Load Address Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
10c. Read Calibration Byte	0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
11a. Load No Operation Command	0100011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	

Notes:
1. This command sequence is not required if the seven MSB are correctly set by the previous command sequence (which is normally the case).
2. Repeat until **o** = “1”.
3. Set bits to “0” to program the corresponding Fuse, “1” to unprogram the Fuse.
4. Set bits to “0” to program the corresponding Lock bit, “1” to leave the Lock bit unchanged.
5. “0” = programmed, “1” = unprogrammed.
6. The bit mapping for Fuses High byte is listed in Table 27-3 on page 186
7. The bit mapping for Fuses Low byte is listed in Table 27-4 on page 186
8. The bit mapping for Lock bits byte is listed in Table 27-1 on page 185
9. Address bits exceeding PCMSB and EEAMSB (Table 27-5 and Table 27-6) are don’t care
10. All TDI and TDO sequences are represented by binary digits (0b...).

Figure 27-13. State Machine Sequence for Changing/Reading the Data Word



27.8.11 Flash Data Byte Register

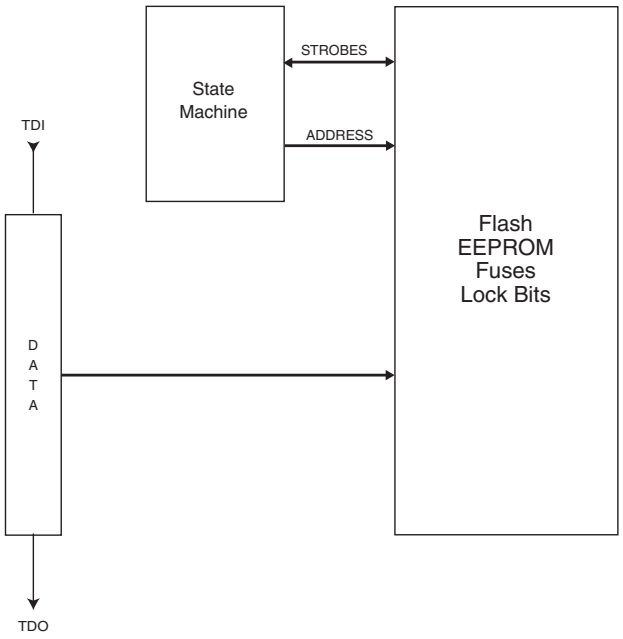
The Flash Data Byte Register provides an efficient way to load the entire Flash page buffer before executing Page Write, or to read out/verify the content of the Flash. A state machine sets up the control signals to the Flash and senses the strobe signals from the Flash, thus only the data words need to be shifted in/out.

The Flash Data Byte Register actually consists of the 8-bit scan chain and a 8-bit temporary register. During page load, the Update-DR state copies the content of the scan chain over to the temporary register and initiates a write sequence that within 11 TCK cycles loads the content of the temporary register into the Flash page buffer. The AVR automatically alternates between writing the low and the high byte for each new Update-DR state, starting with the low byte for the first Update-DR encountered after entering the PROG_PAGELOAD command. The Program Counter is pre-incremented before writing the low byte, except for the first written byte. This ensures that the first data is written to the address set up by PROG_COMMANDS, and loading the last location in the page buffer does not make the Program Counter increment into the next page.

During Page Read, the content of the selected Flash byte is captured into the Flash Data Byte Register during the Capture-DR state. The AVR automatically alternates between reading the low and the high byte for each new Capture-DR state, starting with the low byte for the first Cap-

ture-DR encountered after entering the PROG_PAGEREAD command. The Program Counter is post-incremented after reading each high byte, including the first read byte. This ensures that the first data is captured from the first address set up by PROG_COMMANDS, and reading the last location in the page makes the program counter increment into the next page.

Figure 27-14. Flash Data Byte Register



The state machine controlling the Flash Data Byte Register is clocked by TCK. During normal operation in which eight bits are shifted for each Flash byte, the clock cycles needed to navigate through the TAP controller automatically feeds the state machine for the Flash Data Byte Register with sufficient number of clock pulses to complete its operation transparently for the user. However, if too few bits are shifted between each Update-DR state during page load, the TAP controller should stay in the Run-Test/Idle state for some TCK cycles to ensure that there are at least 11 TCK cycles between each Update-DR state.

27.8.12 Programming Algorithm

All references below of type “1a”, “1b”, and so on, refer to Table 27-12.

27.8.13 Entering Programming Mode

1. Enter JTAG instruction AVR_RESET and shift 1 in the Reset Register.
2. Enter instruction PROG_ENABLE and shift 0b1010_0011_0111_0000 in the Programming Enable Register.

27.8.14 Leaving Programming Mode

1. Enter JTAG instruction PROG_COMMANDS.
2. Disable all programming instructions by using no operation instruction 11a.
3. Enter instruction PROG_ENABLE and shift 0b0000_0000_0000_0000 in the programming Enable Register.
4. Enter JTAG instruction AVR_RESET and shift 0 in the Reset Register.

27.8.15 Performing Chip Erase

1. Enter JTAG instruction PROG_COMMANDS.
2. Start Chip Erase using programming instruction 1a.
3. Poll for Chip Erase complete using programming instruction 1b, or wait for t_{WLRH_CE} (refer to Table 27-11 on page 197).

27.8.16 Programming the Flash

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash write using programming instruction 2a.
3. Load address High byte using programming instruction 2b.
4. Load address Low byte using programming instruction 2c.
5. Load data using programming instructions 2d, 2e and 2f.
6. Repeat steps 4 and 5 for all instruction words in the page.
7. Write the page using programming instruction 2g.
8. Poll for Flash write complete using programming instruction 2h, or wait for t_{WLRH} (refer to Table 27-11 on page 197).
9. Repeat steps 3 to 7 until all data have been programmed.

A more efficient data transfer can be achieved using the PROG_PAGELOAD instruction:

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash write using programming instruction 2a.
3. Load the page address using programming instructions 2b and 2c. PCWORD (refer to Table 27-5 on page 188) is used to address within one page and must be written as 0.
4. Enter JTAG instruction PROG_PAGELOAD.
5. Load the entire page by shifting in all instruction words in the page byte-by-byte, starting with the LSB of the first instruction in the page and ending with the MSB of the last instruction in the page. Use Update-DR to copy the contents of the Flash Data Byte Register into the Flash page location and to auto-increment the Program Counter before each new word.
6. Enter JTAG instruction PROG_COMMANDS.
7. Write the page using programming instruction 2g.
8. Poll for Flash write complete using programming instruction 2h, or wait for t_{WLRH} (refer to Table 27-11 on page 197).
9. Repeat steps 3 to 8 until all data have been programmed.

27.8.17 Reading the Flash

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash read using programming instruction 3a.
3. Load address using programming instructions 3b and 3c.
4. Read data using programming instruction 3d.
5. Repeat steps 3 and 4 until all data have been read.

A more efficient data transfer can be achieved using the PROG_PAGEREAD instruction:

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash read using programming instruction 3a.
3. Load the page address using programming instructions 3b and 3c. PCWORD (refer to Table 27-5 on page 188) is used to address within one page and must be written as 0.

4. Enter JTAG instruction PROG_PAGEREAD.
5. Read the entire page (or Flash) by shifting out all instruction words in the page (or Flash), starting with the LSB of the first instruction in the page (Flash) and ending with the MSB of the last instruction in the page (Flash). The Capture-DR state both captures the data from the Flash, and also auto-increments the program counter after each word is read. Note that Capture-DR comes before the shift-DR state. Hence, the first byte which is shifted out contains valid data.
6. Enter JTAG instruction PROG_COMMANDS.
7. Repeat steps 3 to 6 until all data have been read.

27.8.18 Programming the EEPROM

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable EEPROM write using programming instruction 4a.
3. Load address High byte using programming instruction 4b.
4. Load address Low byte using programming instruction 4c.
5. Load data using programming instructions 4d and 4e.
6. Repeat steps 4 and 5 for all data bytes in the page.
7. Write the data using programming instruction 4f.
8. Poll for EEPROM write complete using programming instruction 4g, or wait for t_{WLRH} (refer to Table 27-11 on page 197).
9. Repeat steps 3 to 8 until all data have been programmed.

Note that the PROG_PAGELOAD instruction can not be used when programming the EEPROM.

27.8.19 Reading the EEPROM

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable EEPROM read using programming instruction 5a.
3. Load address using programming instructions 5b and 5c.
4. Read data using programming instruction 5d.
5. Repeat steps 3 and 4 until all data have been read.

Note that the PROG_PAGEREAD instruction can not be used when reading the EEPROM.

27.8.20 Programming the Fuses

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Fuse write using programming instruction 6a.
3. Load data high byte using programming instructions 6b. A bit value of “0” will program the corresponding fuse, a “1” will unprogram the fuse.
4. Write Fuse High byte using programming instruction 6c.
5. Poll for Fuse write complete using programming instruction 6d, or wait for t_{WLRH} (refer to Table 27-11 on page 197).
6. Load data low byte using programming instructions 6e. A “0” will program the fuse, a “1” will unprogram the fuse.
7. Write Fuse low byte using programming instruction 6f.
8. Poll for Fuse write complete using programming instruction 6g, or wait for t_{WLRH} (refer to Table 27-11 on page 197).

27.8.21 Programming the Lock Bits

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Lock bit write using programming instruction 7a.
3. Load data using programming instructions 7b. A bit value of “0” will program the corresponding lock bit, a “1” will leave the lock bit unchanged.
4. Write Lock bits using programming instruction 7c.
5. Poll for Lock bit write complete using programming instruction 7d, or wait for t_{WLRH} (refer to Table 27-11 on page 197).

27.8.22 Reading the Fuses and Lock Bits

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Fuse/Lock bit read using programming instruction 8a.
3. To read all Fuses and Lock bits, use programming instruction 8e.
   To only read Fuse High byte, use programming instruction 8b.
   To only read Fuse Low byte, use programming instruction 8c.
   To only read Lock bits, use programming instruction 8d.

27.8.23 Reading the Signature Bytes

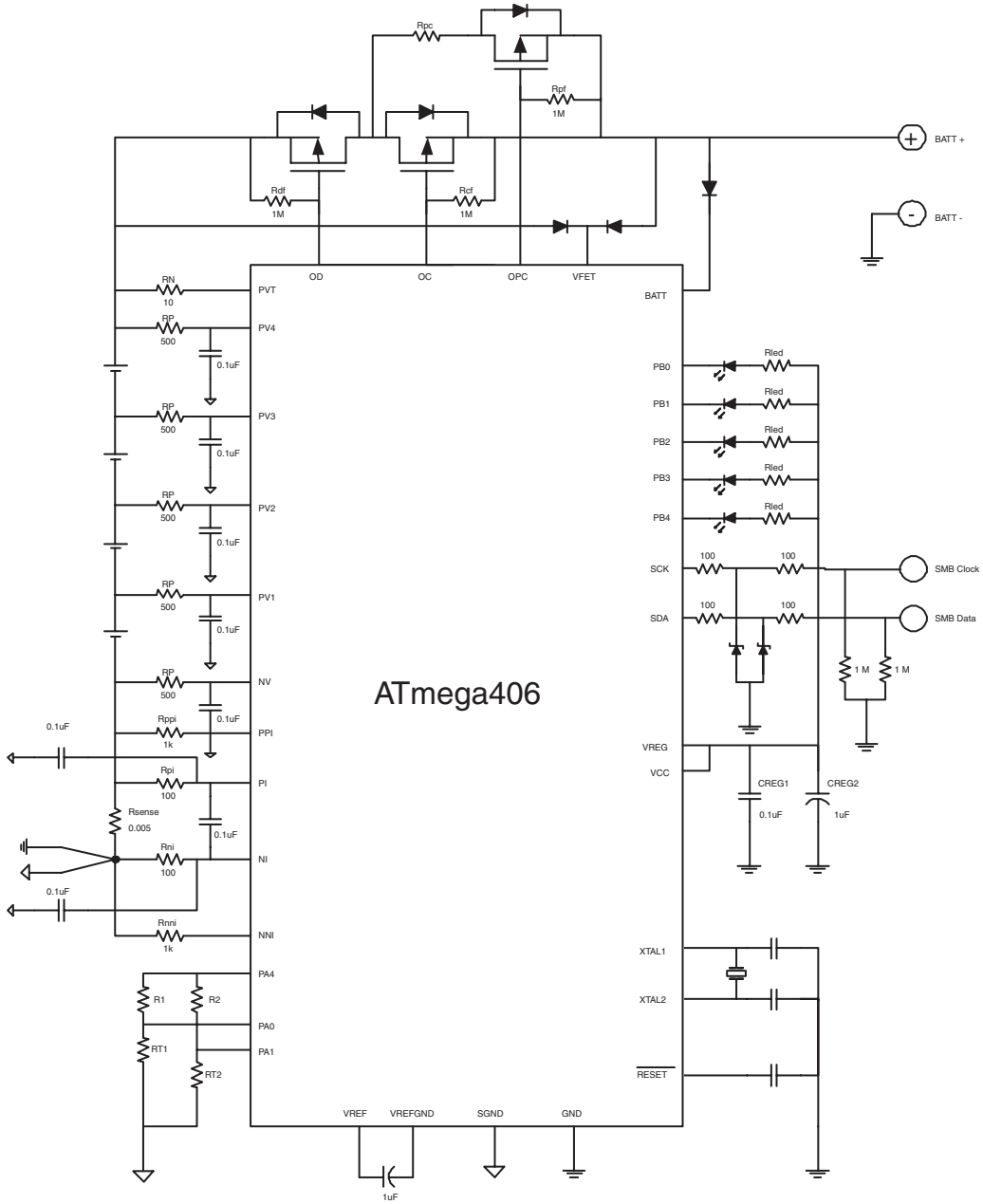
1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Signature byte read using programming instruction 9a.
3. Load address 0x00 using programming instruction 9b.
4. Read first signature byte using programming instruction 9c.
5. Repeat steps 3 and 4 with address 0x01 and address 0x02 to read the second and third signature bytes, respectively.

27.8.24 Reading the Calibration Byte

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Calibration byte read using programming instruction 10a.
3. Load address 0x00 using programming instruction 10b.
4. Read the calibration byte using programming instruction 10c.

28. Operating Circuit

Figure 28-1. Operating Circuit Diagram



29. Electrical Characteristics

29.1 Absolute Maximum Ratings*

Operating Temperature ..... -30°C to +85°C
Storage Temperature ..... -65°C to +150°C
Voltage on PA0 - PA7, PB0 - PB7, PD0 - PD1, VCC, PI, PPI, NI, NNI, XTAL1, and XTAL2 with respect to Ground ..... -0.5V to V_{REG} +0.5V
Voltage on SCL, SDA, NV, PV1 and $\overline{\text{RESET}}$ with respect to Ground .....-0.5V to + 6.0V
Voltage on PVT and VFET with respect to Ground .....-0.5V to + 35V
Voltage on PC0, OPC, OC and BATT with respect to Ground .....-0.5V to VFET + 0.5V
Voltage on OD, PV2 - PV4 with respect to Ground .....-0.5V to PVT + 0.5V
Maximum Operating Voltage 25V

*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.

29.2 DC Characteristics

DC Characteristics, T_A = -30°C to 85°C, V_{CC} = 1.8V to 5.5V

	Parameter	Condition	Min	Typ	Max	Unit
Supply Current	Power Supply Current	Active 1 MHz		TBD	1.2	mA
		Idle 1 MHz		TBD	0.5	mA
		ADC Noise Reduction		TBD	0.6	mA
		Power-down		TBD	25	µA
		Power-off		TBD	2	µA
		Power-save		TBD	85	µA
Voltage Regulator	Regulated Output Voltage(1)	I_{OUT} = 5 mA	3.25	3.3	3.35	V
	Temperature Stability(1)	I_{OUT} = 5 mA T_A = 0 - 60 °C		± 5	± 15	mV
		I_{OUT} = 5 mA T_A = -30 – 85 °C		± 20	± 70	mV
	Load Regulation	0.1 mA < I_{OUT} < 5 mA		± 20	± 60	mV
	Line Regulation	4V < V_{FET} < 25V, I_{OUT} = 1 mA		± 2	± 10	mV

DC Characteristics, T_A = -30°C to 85°C, V_{CC} = 1.8V to 5.5V

	Parameter	Condition	Min	Typ	Max	Unit
V-ADC	Reference Voltage			1.100		V
	Conversion Time			512		µs
	Effective Resolution		12			Bits
	1 LSB Un-scaled Inputs			269		µV
	1 LSB Scaled Inputs (x 0.2)			1.34		mV
	INL				± 1	LSB
	Offset			± 100		µV
	Absolute Accuracy(1) Cell * Inputs	V_{CELL} = 4.0V T_A = 0 - 60°C			± 0.25	%
	Absolute Accuracy(1) ADC * Inputs	0.1V < V_{ADC} < 0.9V T_A = 0 - 60°C			± 0.8	%
Coulomb Counter	Reference Voltage			± 220		mV
	1 LSB Instantaneous Current			53.7		µV
	1 LSB Accumulate Current			1.68		µV
	INL	250 ms conversion time			± 0.003	% FSR
	CC-ADC Offset(2)	T_A = 0 - 60°C			± 10	µV
	Total error after 3.9 ms conversion time(1)	$\|V_{PI-NI}\|$ < 20 mV T_A = 0 - 60°C		±50	± 200	µV
		20 mV < $\|V_{PI-NI}\|$ < 150 mV T_A = 0 - 60°C		± 0.4	± 1.0	%
	Total error after 250 ms conversion time(1)	$\|V_{PI-NI}\|$ < 2 mV T_A = 0 - 60°C		± 2.5	± 15	µV
		2 mV < $\|V_{PI-NI}\|$ < 150 mV T_A = 0 - 60°C		± 0.4	± 0.8	%
Temperature Sensor	V_{PTAT}, Voltage Proportional to Absolute Temperature			0.6		mV/K
	Absolute Accuracy(3)				± 4	K

Notes:
1. After calibration at a second temperature. By default the first calibration is performed at 85°C in Atmel factory test. The second calibration step can easily be implemented in a standard test flow at room temperature.
2. After system offset compensation in software.
3. The measured V_{PTAT} voltage must be scaled with the calibration value stored in the VPTAT Calibration Register to get the absolute temperature.

Table 29-1. Characteristics for the General I/O Lines

Symbol	Parameter	Condition	Min	Typ	Max	Unit
V_{OL}	Output Low Voltage	IOL = 0 - 1.0mA	0		0.5	V
V_{OH}	Output High Voltage	IOH = -1.0 - 0.0mA(1)	1.7		2.4	V
V_{IL}	Input Low Voltage		0		TBD	V
V_{IH}	Input High Voltage		TBD		TBD	V

Note: 1. Maximum total source current is 2mA

29.3 2-wire Serial Interface Characteristics

Table 29-2 describes the requirements for devices connected to the Two-wire Serial Bus. The ATmega406 Two-wire Serial Interface meets or exceeds these requirements under the noted conditions.

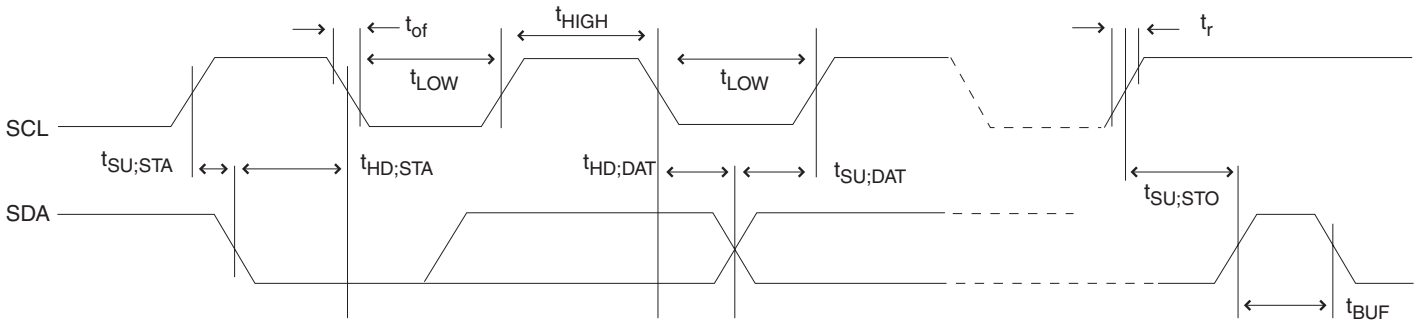
Timing symbols refer to Figure 29-1.

Table 29-2. Two-wire Serial Bus Requirements

Symbol	Parameter	Condition	Min	Max	Units
VIL	Input Low-voltage		-0.5	0.8	V
VIH	Input High-voltage		2.1	5.5	V
VOL(1)	Output Low-voltage	350 µA sink current	0	0.4	V
tr(1)	Rise Time for both SDA and SCL			300	ns
tof(1)	Output Fall Time from V_{IHmin} to V_{ILmax}	C_b < 400 pF(2)		250	ns
tSP(1)	Spikes Suppressed by Input Filter		0	50	ns
I_i	Input Current each I/O Pin	$0.1V_{BUS} < V_i < 0.9V_{BUS}$	-5	5	µA
C_i(1)	Capacitance for each I/O Pin		–	10	pF
f_{SCL}	SCL Clock Frequency	f_{CK}(3) > max(16f_{SCL}, 450 kHz)(4)	0	100	kHz
Rp	Value of Pull-up resistor	$f_{SCL} \leq$ 100 kHz	$\frac{V_{BUS} - 0{,}4V}{350\mu A}$	$\frac{V_{BUS} - 0{,}4V}{100\mu A}$	Ω
$t_{HD;STA}$	Hold Time (repeated) START Condition	$f_{SCL} \leq$ 100 kHz	4.0	–	µs
t_{LOW}	Low Period of the SCL Clock	$f_{SCL} \leq$ 100 kHz	4.7	–	µs
t_{HIGH}	High period of the SCL clock	$f_{SCL} \leq$ 100 kHz	4.0	–	µs
$t_{SU;STA}$	Set-up time for a repeated START condition	$f_{SCL} \leq$ 100 kHz	4.7	–	µs
$t_{HD;DAT}$	Data hold time	$f_{SCL} \leq$ 100 kHz	0.3	3.45	µs
$t_{SU;DAT}$	Data setup time	$f_{SCL} \leq$ 100 kHz	250	–	ns
$t_{SU;STO}$	Setup time for STOP condition	$f_{SCL} \leq$ 100 kHz	4.0	–	µs
t_{BUF}	Bus free time between a STOP and START condition	$f_{SCL} \leq$ 100 kHz	4.7	–	µs

Notes:
1. In ATmega406, this parameter is characterized and not tested.
2. C_b = capacitance of one bus line in pF.
3. f_{CK} = CPU clock frequency
4. This requirement applies to all ATmega406 Two-wire Serial Interface operation. Other devices connected to the Two-wire Serial Bus need only obey the general f_{SCL} requirement.

Figure 29-1. Two-wire Serial Bus Timing



30. ATmega406 Typical Characteristics – TBD

31. Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xFF)	Reserved	–	–	–	–	–	–	–	–	
(0xFE)	Reserved	–	–	–	–	–	–	–	–	
(0xFD)	Reserved	–	–	–	–	–	–	–	–	
(0xFC)	Reserved	–	–	–	–	–	–	–	–	
(0xFB)	Reserved	–	–	–	–	–	–	–	–	
(0xFA)	Reserved	–	–	–	–	–	–	–	–	
(0xF9)	Reserved	–	–	–	–	–	–	–	–	
(0xF8)	BPPLR	–	–	–	–	–	–	BPPLE	BPPL	121
(0xF7)	BPCR	–	–	–	–	DUVD	SCD	DCD	CCD	121
(0xF6)	CBPTR	SCPT[3:0]				OCPT[3:0]				122
(0xF5)	BPOCD	DCDL[3:0]				CCDL[3:0]				123
(0xF4)	BPSCD	–	–	–	–	SCDL[3:0]				123
(0xF3)	BPDUV	–	–	DUVT1	DUVT0	DUDL[3:0]				124
(0xF2)	BPIR	DUVIF	COCIF	DOCIF	SCIF	DUVIE	COCIE	DOCIE	SCIE	125
(0xF1)	CBCR	–	–	–	–	CBE4	CBE3	CBE2	CBE1	130
(0xF0)	FCSR	–	–	PWMOC	PWMOPC	CPS	DFE	CFE	PFD	127
(0xEF)	Reserved	–	–	–	–	–	–	–	–	
(0xEE)	Reserved	–	–	–	–	–	–	–	–	
(0xED)	Reserved	–	–	–	–	–	–	–	–	
(0xEC)	Reserved	–	–	–	–	–	–	–	–	
(0xEB)	Reserved	–	–	–	–	–	–	–	–	
(0xEA)	Reserved	–	–	–	–	–	–	–	–	
(0xE9)	CADICH	CADIC[15:8]								108
(0xE8)	CADICL	CADIC[7:0]								108
(0xE7)	CADRDC	CADRDC[7:0]								109
(0xE6)	CADRCC	CADRCC[7:0]								108
(0xE5)	CADCSRB	–	CADACIE	CADRCIE	CADICIE	–	CADACIF	CADRCIF	CADICIF	107
(0xE4)	CADCSRA	CADEN	–	CADUB	CADAS1	CADAS0	CADSI1	CADSI0	CADSE	105
(0xE3)	CADAC3	CADAC[31:24]								108
(0xE2)	CADAC2	CADAC[23:16]								108
(0xE1)	CADAC1	CADAC[15:8]								108
(0xE0)	CADAC0	CADAC[7:0]								108
(0xDF)	Reserved	–	–	–	–	–	–	–	–	
(0xDE)	Reserved	–	–	–	–	–	–	–	–	
(0xDD)	Reserved	–	–	–	–	–	–	–	–	
(0xDC)	Reserved	–	–	–	–	–	–	–	–	
(0xDB)	Reserved	–	–	–	–	–	–	–	–	
(0xDA)	Reserved	–	–	–	–	–	–	–	–	
(0xD9)	Reserved	–	–	–	–	–	–	–	–	
(0xD8)	Reserved	–	–	–	–	–	–	–	–	
(0xD7)	Reserved	–	–	–	–	–	–	–	–	
(0xD6)	Reserved	–	–	–	–	–	–	–	–	
(0xD5)	Reserved	–	–	–	–	–	–	–	–	
(0xD4)	Reserved	–	–	–	–	–	–	–	–	
(0xD3)	Reserved	–	–	–	–	–	–	–	–	
(0xD2)	Reserved	–	–	–	–	–	–	–	–	
(0xD1)	BGCRR	BGCR7	BGCR6	BGCR5	BGCR4	BGCR3	BGCR2	BGCR1	BGCR0	116
(0xD0)	BGCCR	BGEN	–	BGCC5	BGCC4	BGCC3	BGCC2	BGCC1	BGCC0	116
(0xCF)	Reserved	–	–	–	–	–	–	–	–	
(0xCE)	Reserved	–	–	–	–	–	–	–	–	
(0xCD)	Reserved	–	–	–	–	–	–	–	–	
(0xCC)	Reserved	–	–	–	–	–	–	–	–	
(0xCB)	Reserved	–	–	–	–	–	–	–	–	
(0xCA)	Reserved	–	–	–	–	–	–	–	–	
(0xC9)	Reserved	–	–	–	–	–	–	–	–	
(0xC8)	Reserved	–	–	–	–	–	–	–	–	
(0xC7)	Reserved	–	–	–	–	–	–	–	–	
(0xC6)	Reserved	–	–	–	–	–	–	–	–	
(0xC5)	Reserved	–	–	–	–	–	–	–	–	
(0xC4)	Reserved	–	–	–	–	–	–	–	–	
(0xC3)	Reserved	–	–	–	–	–	–	–	–	
(0xC2)	Reserved	–	–	–	–	–	–	–	–	
(0xC1)	Reserved	–	–	–	–	–	–	–	–	
(0xC0)	CCSR	–	–	–	–	–	–	XOE	ACS	29

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xBF)	Reserved	–	–	–	–	–	–	–	–	
(0xBE)	TWBCSR	TWBCIF	TWBCIE	–	–	–	TWBDT1	TWBDT0	TWBCIP	162
(0xBD)	TWAMR	TWAM[6:0]							–	143
(0xBC)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	140
(0xBB)	TWDR	2–wire Serial Interface Data Register								142
(0xBA)	TWAR	TWA[6:0]							TWGCE	142
(0xB9)	TWSR	TWS[7:3]					–	TWPS1	TWPS0	141
(0xB8)	TWBR	2–wire Serial Interface Bit Rate Register								140
(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)	Reserved	–	–	–	–	–	–	–	–	
(0x8C)	Reserved	–	–	–	–	–	–	–	–	
(0x8B)	Reserved	–	–	–	–	–	–	–	–	
(0x8A)	Reserved	–	–	–	–	–	–	–	–	
(0x89)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte								99
(0x88)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte								99
(0x87)	Reserved	–	–	–	–	–	–	–	–	
(0x86)	Reserved	–	–	–	–	–	–	–	–	
(0x85)	TCNT1H	Timer/Counter1 – Counter Register High Byte								99
(0x84)	TCNT1L	Timer/Counter1 – Counter Register Low Byte								99
(0x83)	Reserved	–	–	–	–	–	–	–	–	
(0x82)	Reserved	–	–	–	–	–	–	–	–	
(0x81)	TCCR1B	–	–	–	–	CTC1	CS12	CS11	CS10	98
(0x80)	Reserved	–	–	–	–	–	–	–	–	
(0x7F)	Reserved	–	–	–	–	–	–	–	–	
(0x7E)	DIDR0	–	–	–	–	VADC3D	VADC2D	VADC1D	VADC0D	114

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0x7D)	Reserved	–	–	–	–	–	–	–	–	
(0x7C)	VADMUX	–	–	–	–	VADMUX3	VADMUX2	VADMUX1	VADMUX0	112
(0x7B)	Reserved	–	–	–	–	–	–	–	–	
(0x7A)	VADCSR	–	–	–	–	VADEN	VADSC	VADCCIF	VADCCIE	113
(0x79)	VADCH	–	–	–	–	VADC Data Register High byte				113
(0x78)	VADCL	VADC Data Register Low byte								113
(0x77)	Reserved	–	–	–	–	–	–	–	–	
(0x76)	Reserved	–	–	–	–	–	–	–	–	
(0x75)	Reserved	–	–	–	–	–	–	–	–	
(0x74)	Reserved	–	–	–	–	–	–	–	–	
(0x73)	Reserved	–	–	–	–	–	–	–	–	
(0x72)	Reserved	–	–	–	–	–	–	–	–	
(0x71)	Reserved	–	–	–	–	–	–	–	–	
(0x70)	Reserved	–	–	–	–	–	–	–	–	
(0x6F)	TIMSK1	–	–	–	–	–	–	OCIE1A	TOIE1	100
(0x6E)	TIMSK0	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	91
(0x6D)	Reserved	–	–	–	–	–	–	–	–	
(0x6C)	PCMSK1	PCINT[15:8]								57
(0x6B)	PCMSK0	PCINT[7:0]								57
(0x6A)	Reserved	–	–	–	–	–	–	–	–	
(0x69)	EICRA	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	54
(0x68)	PCICR	–	–	–	–	–	–	PCIE1	PCIE0	56
(0x67)	Reserved	–	–	–	–	–	–	–	–	
(0x66)	FOSCCAL	Fast Oscillator Calibration Register								27
(0x65)	Reserved	–	–	–	–	–	–	–	–	
(0x64)	PRR0	–	–	–	–	PRTWI	PRTIM1	PRTIM0	PRVADC	35
(0x63)	Reserved	–	–	–	–	–	–	–	–	
(0x62)	WUTCSR	WUTIF	WUTIE	WUTCF	WUTR	WUTE	WUTP2	WUTP1	WUTP0	47
(0x61)	Reserved	–	–	–	–	–	–	–	–	
(0x60)	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	45
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	10
0x3E (0x5E)	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	12
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	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	175
0x36 (0x56)	Reserved	–	–	–	–	–	–	–	–	
0x35 (0x55)	MCUCR	JTD	–	–	PUD	–	–	IVSEL	IVCE	52/66
0x34 (0x54)	MCUSR	–	–	–	JTRF	WDRF	BODRF	EXTRF	PORF	41
0x33 (0x53)	SMCR	–	–	–	–	SM2	SM1	SM0	SE	31
0x32 (0x52)	Reserved	–	–	–	–	–	–	–	–	
0x31 (0x51)	OCDR	On-Chip Debug Register								169
0x30 (0x50)	Reserved	–	–	–	–	–	–	–	–	
0x2F (0x4F)	Reserved	–	–	–	–	–	–	–	–	
0x2E (0x4E)	Reserved	–	–	–	–	–	–	–	–	
0x2D (0x4D)	Reserved	–	–	–	–	–	–	–	–	
0x2C (0x4C)	Reserved	–	–	–	–	–	–	–	–	
0x2B (0x4B)	GPIOR2	General Purpose I/O Register 2								24
0x2A (0x4A)	GPIOR1	General Purpose I/O Register 1								24
0x29 (0x49)	Reserved	–	–	–	–	–	–	–	–	
0x28 (0x48)	OCR0B	Timer/Counter0 Output Compare Register B								90
0x27 (0x47)	OCR0A	Timer/Counter0 Output Compare Register A								90
0x26 (0x46)	TCNT0	Timer/Counter0 (8 Bit)								90
0x25 (0x45)	TCCR0B	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	89
0x24 (0x44)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	86
0x23 (0x43)	GTCCR	TSM	–	–	–	–	–	–	PSRSYNC	102
0x22 (0x42)	EEARH	–	–	–	–	–	–	–	High Byte	19
0x21 (0x41)	EEARL	EEPROM Address Register Low Byte								19
0x20 (0x40)	EEDR	EEPROM Data Register								19
0x1F (0x3F)	EECR	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	19
0x1E (0x3E)	GPIOR0	General Purpose I/O Register 0								24
0x1D (0x3D)	EIMSK	–	–	–	–	INT3	INT2	INT1	INT0	55
0x1C (0x3C)	EIFR	–	–	–	–	INTF3	INTF2	INTF1	INTF0	55

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x1B (0x3B)	PCIFR	–	–	–	–	–	–	PCIF1	PCIF0	
0x1A (0x3A)	Reserved	–	–	–	–	–	–	–	–	
0x19 (0x39)	Reserved	–	–	–	–	–	–	–	–	
0x18 (0x38)	Reserved	–	–	–	–	–	–	–	–	
0x17 (0x37)	Reserved	–	–	–	–	–	–	–	–	
0x16 (0x36)	TIFR1	–	–	–	–	–	–	OCF1A	TOV1	100
0x15 (0x35)	TIFR0	–	–	–	–	–	OCF0B	OCF0A	TOV0	91
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	–	–	–	–	–	–	PORTD1	PORTD0	72
0x0A (0x2A)	DDRD	–	–	–	–	–	–	DDD1	DDD0	72
0x09 (0x29)	PIND	–	–	–	–	–	–	PIND1	PIND0	72
0x08 (0x28)	PORTC	–	–	–	–	–	–	–	PORTC0	74
0x07 (0x27)	Reserved	–	–	–	–	–	–	–	–	
0x06 (0x26)	Reserved	–	–	–	–	–	–	–	–	
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	71
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	71
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	72
0x02 (0x22)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTB3	PORTA2	PORTA1	PORTA0	71
0x01 (0x21)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	71
0x00 (0x20)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	71

Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O registers within the address range $00 - $1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the status flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses $00 - $3F must be used. When addressing I/O registers as data space using LD and ST instructions, $20 must be added to these addresses. The ATmega406 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from $60 - $FF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

32. Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF − Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 − Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd − 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL	k	Direct Subroutine Call	PC ← k	None	4
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd − Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd − Rr − C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd − K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1/2

32. Instruction Set Summary (Continued)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1

32. Instruction Set Summary (Continued)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
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

33. Ordering Information

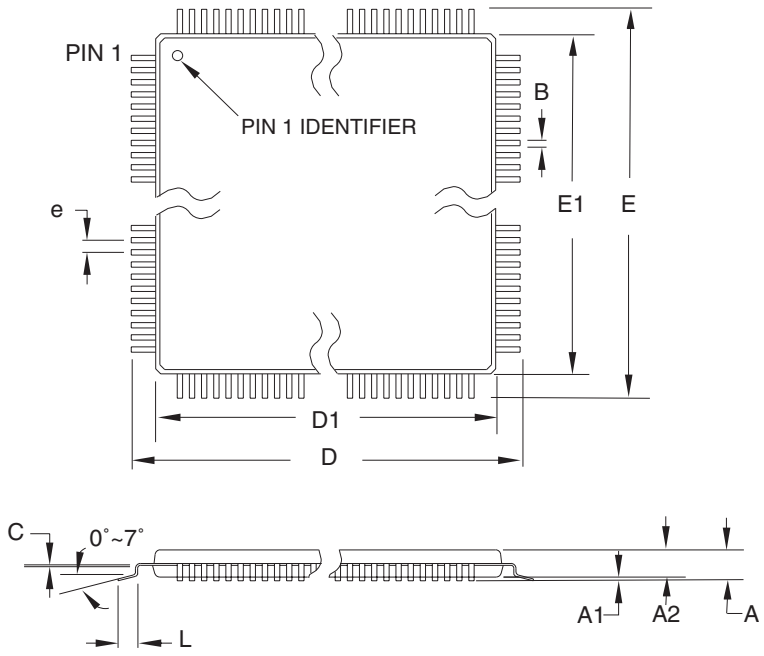
Speed (MHz)	Power Supply	Ordering Code	Package[1]	Operation Range
1	4.0 - 25V	ATmega406-1AAU[2]	48AA	Industrial (-30°C to 85°C)

Notes: 1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging alternative, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.

Package Type	
48AA	48-lead, 7 x 7 x 1.44 mm body, 0.5 mm lead pitch, Low Profile Plastic Quad Flat Package (LQFP)

34. Packaging Information

34.1 48AA



COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	1.60	
A1	0.05	–	0.15	
A2	1.35	1.40	1.45	
D	8.75	9.00	9.25	
D1	6.90	7.00	7.10	Note 2
E	8.75	9.00	9.25	
E1	6.90	7.00	7.10	Note 2
B	0.17	–	0.27	
C	0.09	–	0.20	
L	0.45	–	0.75	
e	0.50 TYP			

Notes:
1. This package conforms to JEDEC reference MS-026, Variation BBC.
2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25 mm per side. Dimensions D1 and E1 are maximum plastic body size dimensions including mold mismatch.
3. Lead coplanarity is 0.08 mm maximum.

10/5/2001

	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	**48AA,** 48-lead, 7 x 7 mm Body Size, 1.4 mm Body Thickness, 0.5 mm Lead Pitch, Low Profile Plastic Quad Flat Package (LQFP)	48AA	C

35. Errata

35.1 Rev. D

- **Voltage Regulator Start-up sequence**
- **V_{REF} influenced by MCU state**
- **EEPROM read from application code does not work in Lock Bit Mode 3**

1. Voltage Regulator Start-up sequence

When powering up ATmega406 some precautions are necessary to ensure proper start-up of the Voltage Regulator.

Problem Fix/Workaround

The three steps below are needed to ensure proper start-up of the voltage regulator.

a. Do NOT connect a capacitor larger than 100 nF on the VFET pin. This is to ensure fast rise time on the VFET pin when a supply voltage is connected.

b. During assembly, always connect Cell1 first, then Cell2 and so on until the top cell is connected to PVT. If the cell voltages are about 2 volts or larger, the Voltage Regulator will normally start up properly in Power-off mode (VREG appr. 2.8 volts).

c. After all cells have been assembled as described in step 2, a charger source must be connected at the BATT+ terminal to initialize the chip, see Section 8.3 ”Power-on Reset and Charger Connect” on page 38 in the datasheet.

If the Voltage Regulator started up in Power-off during assembly of the cells, the chip will initialize when the charger source makes the voltage at the BATT pin exceed 7 - 8 Volts.

If the Voltage Regulator did not start up properly, the charger source has one additional requirement to ensure proper start up and initialization. In this case the charger source must ensure that the voltage at the VFET pin increases quickly at least 3 Volts above the voltage at the PVT pin, and that the voltage at the BATT pin exceeds 7 - 8 Volts. This will start up and initialize the chip directly.

2. V_{REF} influenced by MCU state

The reference voltage at the V_{REF} pin depends on the following conditions of the device:

a. Charger Over-current and/or Discharge Over-current Protection active but Short-circuit inactive. This will increase V_{REF} voltage with typical 1 mV compared to a condition were all Current Protections are disabled.

b. Short-circuit Protection active. Short-circuit measurements are activated when SCD in BPCR is zero (default) and DFE in FET Control and Status Register (FCSR) is set. This will increase V_{REF} voltage with typical 8 mV compared to a condition with short-circuit measurements inactive.

c. V-ADC conversion of the internal VTEMP voltage. This will increase V_{REF} voltage with typical 15 mV compared to a condition with short-circuit measurements inactive.

Problem Fix/Work around

To ensure the highest accuracy, set the Bandgap Calibration Register (BGCC) to get 1.100 V at V_{REF} after the chip is configured with the actual Battery Protection settings and the Discharge FET is enabled.

3. EEPROM read from application code does not work in Lock Bit Mode 3

When the Memory Lock Bits LB2 and LB1 are programmed to mode 3, EEPROM read does not work from the application code.

Problem Fix/Work around

Do not set Lock Bit Protection Mode 3 when the application code needs to read from EEPROM.

36. Datasheet Revision History

36.1 Rev 2548C - 05/05

1. Updated Section 35. ”Errata” on page 225.

36.2 Rev 2548B - 04/05

1. Typos updated, bit “PSRASY” removed, CS12:0 renamed CS1[2:0].
2. Removed “BGEN” bit in BGCCR register. The bandgap voltage reference is always enabled in ATmega406 revision E.
3. Updated Figure 2-1 on page 3, Figure 6-1 on page 25, Figure 24-9 on page 137, Figure 21-1 on page 120.
4. Updated Table 7-2 on page 33, Table 7-3 on page 34, Table 8-1 on page 38, Table 26-5 on page 181, Figure 27-1 on page 188.
5. Updated Section 12.3.2 ”Alternate Functions of Port A” on page 66 and Section 21. ”Battery Protection” on page 118 description.
6. Updated registers ”External Interrupt Flag Register – EIFR” on page 55 and ”Timer/Counter Control Register B – TCCR0B” on page 89.
7. Updated Section 17.1 ”Features” on page 103 and Section 17.2 ”Operation” on page 103.
   Updated Section 19.1 ”Features” on page 111.
   Updated Section 20.2 ”Register Description for Voltage Reference and Temperature Sensor” on page 116.
8. Updated Section 29. ”Electrical Characteristics” on page 211.
9. Updated Section 35. ”Errata” on page 225.

Table of Contents

Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

Regional Headquarters

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature

2548C–AVR–05/05