



Welcome to [E-XFL.COM](https://www.e-xfl.com)

What is "[Embedded - Microcontrollers](#)"?

"[Embedded - Microcontrollers](#)" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "[Embedded - Microcontrollers](#)"

Details

Product Status	Obsolete
Core Processor	AVR
Core Size	8-Bit
Speed	8MHz
Connectivity	I ² C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	32
Program Memory Size	16KB (8K x 16)
Program Memory Type	FLASH
EEPROM Size	512 x 8
RAM Size	1K x 8
Voltage - Supply (Vcc/Vdd)	2.7V ~ 5.5V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	0°C ~ 70°C (TA)
Mounting Type	Through Hole
Package / Case	40-DIP (0.600", 15.24mm)
Supplier Device Package	40-PDIP
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atmega16l-8pc

General Purpose Register File

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

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

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

Figure 4. AVR CPU General Purpose Working Registers

	7	0	Addr.	
General Purpose Working Registers	R0		\$00	
	R1		\$01	
	R2		\$02	
	...			
	R13		\$0D	
	R14		\$0E	
	R15		\$0F	
	R16		\$10	
	R17		\$11	
	...			
	R26		\$1A	X-register Low Byte
	R27		\$1B	X-register High Byte
	R28		\$1C	Y-register Low Byte
	R29		\$1D	Y-register High Byte
	R30		\$1E	Z-register Low Byte
	R31		\$1F	Z-register High Byte

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

As shown in Figure 4, 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.

AVR ATmega16 Memories

This section describes the different memories in the ATmega16. The AVR architecture has two main memory spaces, the Data Memory and the Program Memory space. In addition, the ATmega16 features an EEPROM Memory for data storage. All three memory spaces are linear and regular.

In-System Reprogrammable Flash Program Memory

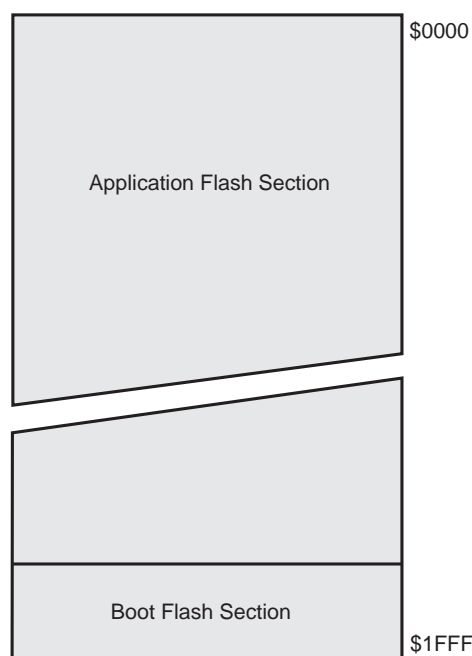
The ATmega16 contains 16 Kbytes On-chip In-System Reprogrammable Flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 8K x 16. For software security, the Flash Program memory space is divided into two sections, Boot Program section and Application Program section.

The Flash memory has an endurance of at least 10,000 write/erase cycles. The ATmega16 Program Counter (PC) is 13 bits wide, thus addressing the 8K 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 246](#). [“Memory Programming” on page 259](#) contains a detailed description on Flash data serial downloading using the SPI pins or the JTAG interface.

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

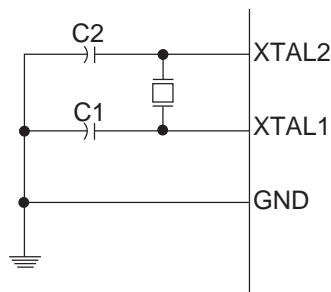
Timing diagrams for instruction fetch and execution are presented in [“Instruction Execution Timing” on page 13](#).

Figure 8. Program Memory Map



choosing capacitors for use with crystals are given in [Table 4](#). For ceramic resonators, the capacitor values given by the manufacturer should be used.

Figure 12. Crystal Oscillator Connections



The Oscillator can operate in three different modes, each optimized for a specific frequency range. The operating mode is selected by the fuses CKSEL3..1 as shown in [Table 4](#).

Table 4. Crystal Oscillator Operating Modes

CKOPT	CKSEL3..1	Frequency Range (MHz)	Recommended Range for Capacitors C1 and C2 for Use with Crystals (pF)
1	101 ⁽¹⁾	0.4 - 0.9	—
1	110	0.9 - 3.0	12 - 22
1	111	3.0 - 8.0	12 - 22
0	101, 110, 111	1.0 ≤	12 - 22

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

the double buffering feature. If the new value written to OCR0 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 OC0 output can be set to toggle its logical level on each compare match by setting the Compare Output mode bits to toggle mode (COM01:0 = 1). The OC0 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 OCR0 is set to zero (0x00). The waveform frequency is defined by the following equation:

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

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

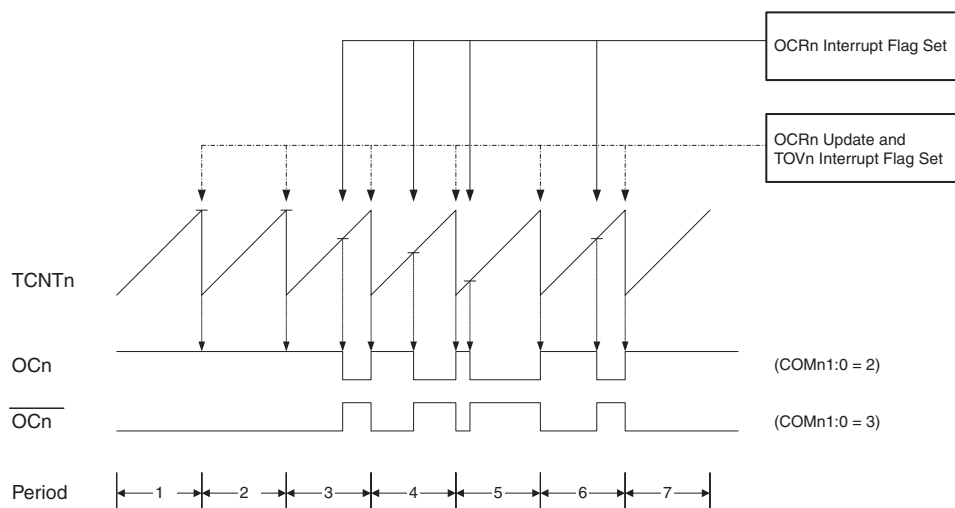
As for the Normal mode of operation, the TOV0 Flag is set in the same timer clock cycle that the counter counts from MAX to 0x00.

Fast PWM Mode

The fast Pulse Width Modulation or fast PWM mode (WGM01:0 = 3) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM option by its single-slope operation. The counter counts from BOTTOM to MAX then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC0) is cleared on the compare match between TCNT0 and OCR0, and set at BOTTOM. In inverting Compare Output mode, the output is set on compare match and cleared at BOTTOM. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct PWM mode that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), and therefore reduces total system cost.

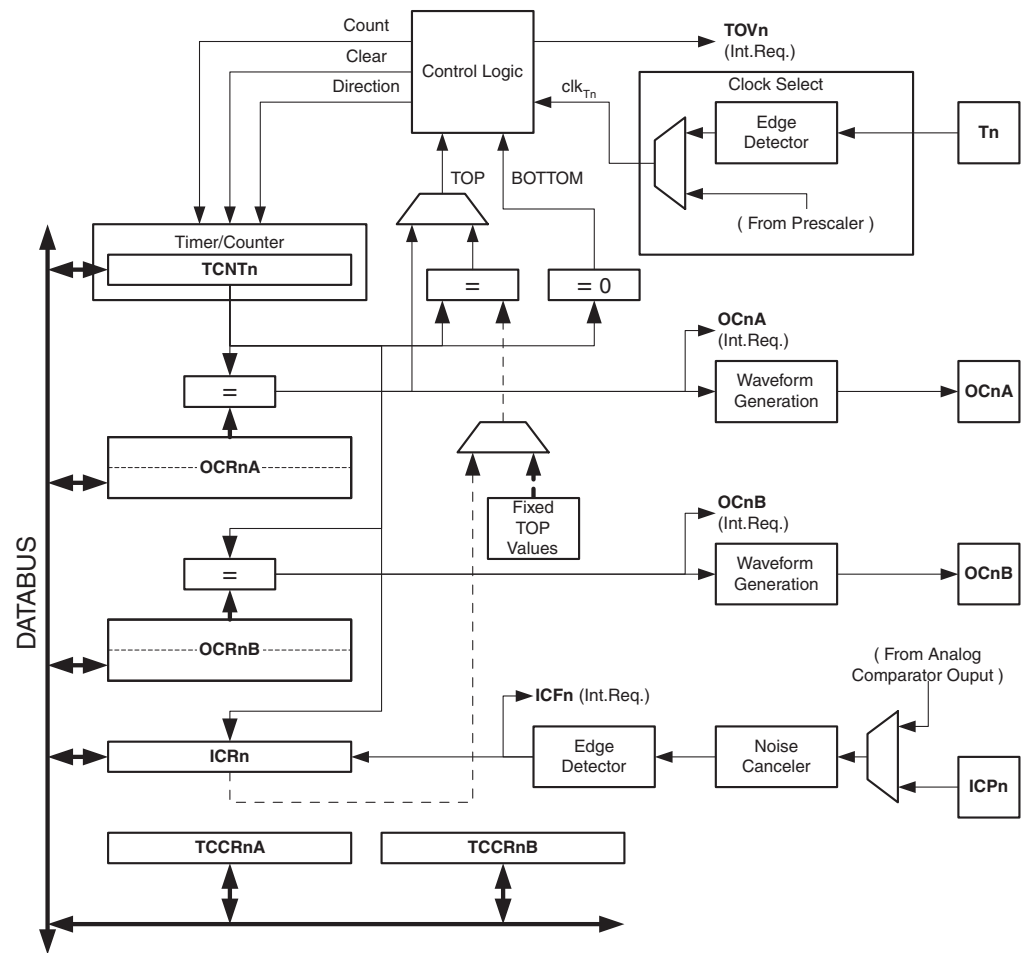
In fast PWM mode, the counter is incremented until the counter value matches the MAX value. The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 32. The TCNT0 value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNT0 slopes represent compare matches between OCR0 and TCNT0.

Figure 32. Fast PWM Mode, Timing Diagram



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

Figure 40. 16-bit Timer/Counter Block Diagram⁽¹⁾



Note: 1. Refer to [Figure 1 on page 2](#), [Table 25 on page 58](#), and [Table 31 on page 63](#) for Timer/Counter1 pin placement and description.

Registers

The *Timer/Counter* (TCNT1), *Output Compare Registers* (OCR1A/B), and *Input Capture Register* (ICR1) are all 16-bit registers. Special procedures must be followed when accessing the 16-bit registers. These procedures are described in the section [“Accessing 16-bit Registers” on page 92](#). The *Timer/Counter Control Registers* (TCCR1A/B) are 8-bit registers and have no CPU access restrictions. Interrupt requests (abbreviated to Int.Req. in the figure) signals are all visible in the *Timer Interrupt Flag Register* (TIFR). All interrupts are individually masked with the *Timer Interrupt Mask Register* (TIMSK). TIFR and TIMSK are not shown in the figure since these registers are shared by other timer units.

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

The double buffered Output Compare Registers (OCR1A/B) are compared with the Timer/Counter value at all time. The result of the compare can be used by the Waveform Generator to generate a PWM or variable frequency output on the Output Compare pin (OC1A/B). See [“Output Compare Units” on page 98](#). The compare match event will also set the Compare Match Flag (OCF1A/B) which can be used to generate an output compare interrupt request.

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

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

Timer/Counter Interrupt Mask Register – TIMSK⁽¹⁾

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Note: 1. This register contains interrupt control bits for several Timer/Counters, but only Timer1 bits are described in this section. The remaining bits are described in their respective timer sections.

• Bit 5 – TICIE1: Timer/Counter1, Input Capture Interrupt Enable

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Input Capture Interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 45.) is executed when the ICF1 Flag, located in TIFR, is set.

• Bit 4 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare A match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 45.) is executed when the OCF1A Flag, located in TIFR, is set.

• Bit 3 – OCIE1B: Timer/Counter1, Output Compare B Match Interrupt Enable

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare B match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 45.) is executed when the OCF1B Flag, located in TIFR, is set.

• Bit 2 – TOIE1: Timer/Counter1, Overflow Interrupt Enable

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Overflow Interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 45.) is executed when the TOV1 Flag, located in TIFR, is set.

Timer/Counter Interrupt Flag Register – TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Note: This register contains flag bits for several Timer/Counters, but only Timer1 bits are described in this section. The remaining bits are described in their respective timer sections.

8-bit Timer/Counter2 with PWM and Asynchronous Operation

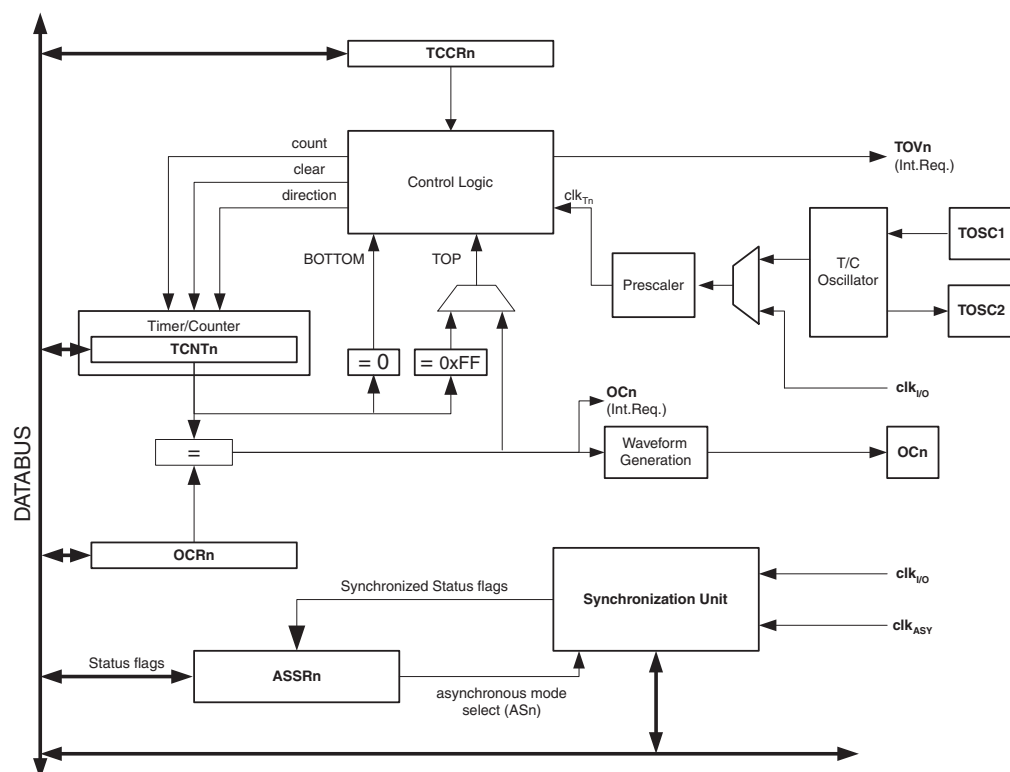
Timer/Counter2 is a general purpose, single compare unit, 8-bit Timer/Counter module. The main features are:

- Single Compare unit Counter
- Clear Timer on Compare Match (Auto Reload)
- Glitch-free, Phase Correct Pulse Width Modulator (PWM)
- Frequency Generator
- 10-bit Clock Prescaler
- Overflow and Compare Match Interrupt Sources (TOV2 and OCF2)
- Allows clocking from External 32 kHz Watch Crystal Independent of the I/O Clock

Overview

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 53. For the actual placement of I/O pins, refer to “Pinout ATmega16” 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 128.

Figure 53. 8-bit Timer/Counter Block Diagram



Registers

The Timer/Counter (TCNT2) and Output Compare Register (OCR2) are 8-bit registers. Interrupt request (shortened as Int. Req.) signals are all visible in the Timer Interrupt Flag Register (TIFR). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK). TIFR and TIMSK are not shown in the figure since these registers are shared by other timer units.

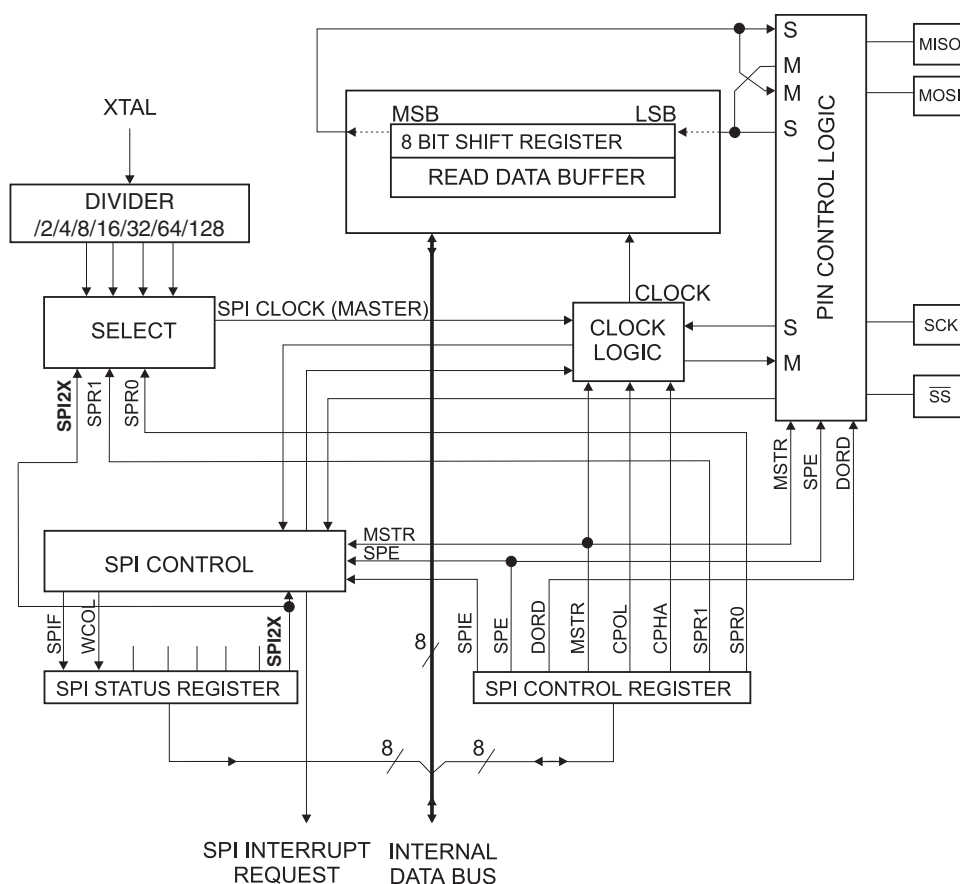
The Timer/Counter can be clocked internally, via the prescaler, or asynchronously clocked from the TOSC1/2 pins, as detailed later in this section. The asynchronous operation is controlled by the Asynchronous Status Register (ASSR). The Clock Select logic block controls which clock source the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the Clock Select logic is referred to as the timer clock (clk_{T2}).

Serial Peripheral Interface – SPI

The Serial Peripheral Interface (SPI) allows high-speed synchronous data transfer between the ATmega16 and peripheral devices or between several AVR devices. The ATmega16 SPI includes the following features:

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

Figure 65. SPI Block Diagram⁽¹⁾



Note: 1. Refer to [Figure 1 on page 2](#), and [Table 25 on page 58](#) for SPI pin placement.

The interconnection between Master and Slave CPUs with SPI is shown in [Figure 66](#). The system consists of two Shift Registers, and a Master clock generator. The SPI Master initiates the communication cycle when pulling low the Slave Select \overline{SS} pin of the desired Slave. Master and Slave prepare the data to be sent in their respective Shift Registers, and the Master generates the required clock pulses on the SCK line to interchange data. Data is always shifted from Master to Slave on the Master Out – Slave In, MOSI, line, and from Slave to Master on the Master In – Slave Out, MISO, line. After each data packet, the Master will synchronize the Slave by pulling high the Slave Select, \overline{SS} , line.

When configured as a Master, the SPI interface has no automatic control of the \overline{SS} line. This must be handled by user software before communication can start. When this is done, writing a

The following code example shows how to read the UCSRC Register contents.

Assembly Code Example⁽¹⁾

```
USART_ReadUCSRC:
    ; Read UCSRC
    in r16,UBRRH
    in r16,UCSRC
    ret
```

C Code Example⁽¹⁾

```
unsigned char USART_ReadUCSRC( void )
{
    unsigned char ucsrc;
    /* Read UCSRC */
    ucsrc = UBRRH;
    ucsrc = UCSRC;
    return ucsrc;
}
```

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

The assembly code example returns the UCSRC value in r16.

Reading the UBRRH contents is not an atomic operation and therefore it can be read as an ordinary register, as long as the previous instruction did not access the register location.

USART Register Description

USART I/O Data Register – UDR

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or UDR. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDR Register location. Reading the UDR Register location will return the contents of the Receive Data Buffer Register (RXB).

For 5-bit, 6-bit, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver.

The transmit buffer can only be written when the UDRE Flag in the UCSRA Register is set. Data written to UDR when the UDRE Flag is not set, will be ignored by the USART Transmitter. When data is written to the transmit buffer, and the Transmitter is enabled, the Transmitter will load the data into the transmit Shift Register when the Shift Register is empty. Then the data will be serially transmitted on the TxD pin.

The receive buffer consists of a two level FIFO. The FIFO will change its state whenever the receive buffer is accessed. Due to this behavior of the receive buffer, do not use read modify write instructions (SBI and CBI) on this location. Be careful when using bit test instructions (SBIC and SBIS), since these also will change the state of the FIFO.

Electrical Interconnection

As depicted in [Figure 76](#), 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.

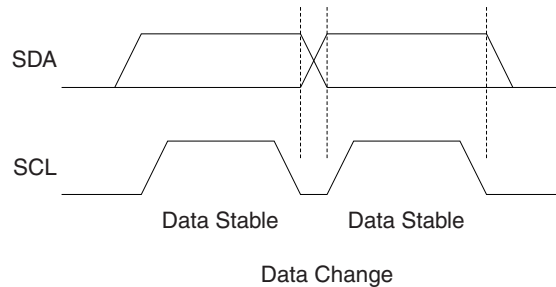
The number of devices that can be connected to the bus is only limited by the bus capacitance limit of 400 pF and the 7-bit Slave address space. A detailed specification of the electrical characteristics of the TWI is given in [“Two-wire Serial Interface Characteristics” on page 294](#). Two different sets of specifications are presented there, one relevant for bus speeds below 100 kHz, and one valid for bus speeds up to 400 kHz.

Data Transfer and Frame Format

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 77. Data Validity



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 releasing 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.

TWI Register Description

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 178](#) for calculating bit rates.

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 condition on the bus if it is free. However, if the bus is not free, the TWI waits until a STOP condition

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.

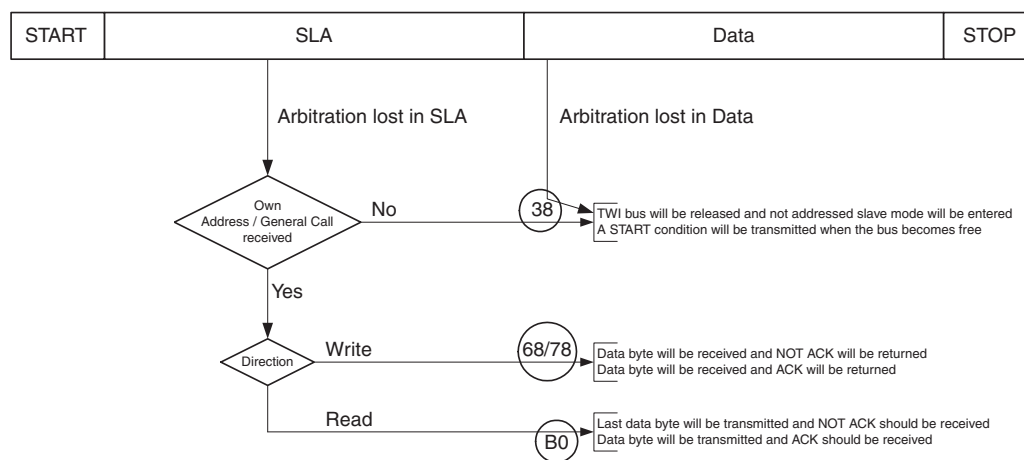
In the following an assembly and C implementation of the example is given. Note that the code below assumes that several definitions have been made, for example by using include-files.

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 96](#). Possible status values are given in circles.

Figure 96. Possible Status Codes Caused by Arbitration



JTAG Interface and On-chip Debug System

Features

- JTAG (IEEE std. 1149.1 Compliant) Interface
- Boundary-scan Capabilities According to the IEEE std. 1149.1 (JTAG) Standard
- Debugger Access to:
 - All Internal Peripheral Units
 - Internal and External RAM
 - The Internal Register File
 - Program Counter
 - EEPROM and Flash Memories
 - Extensive On-chip Debug Support for Break Conditions, Including
 - AVR *Break* Instruction
 - Break on Change of Program Memory Flow
 - Single Step Break
 - Program Memory Breakpoints on Single Address or Address Range
 - Data Memory Breakpoints on Single Address or Address Range
- Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- On-chip Debugging Supported by AVR Studio®

Overview

The AVR IEEE std. 1149.1 compliant JTAG interface can be used for

- Testing PCBs by using the JTAG Boundary-scan capability
- Programming the non-volatile memories, Fuses and Lock bits
- On-chip Debugging

A brief description is given in the following sections. Detailed descriptions for Programming via the JTAG interface, and using the Boundary-scan Chain can be found in the sections [“Programming via the JTAG Interface” on page 278](#) and [“IEEE 1149.1 \(JTAG\) Boundary-scan” on page 228](#), respectively. The On-chip Debug support is considered being private JTAG instructions, and distributed within ATMEL and to selected third party vendors only.

[Figure 112](#) shows a block diagram of the JTAG interface and the On-chip Debug system. The TAP Controller is a state machine controlled by the TCK and TMS signals. The TAP Controller selects either the JTAG Instruction Register or one of several Data Registers as the scan chain (Shift Register) between the TDI input and TDO output. The Instruction Register holds JTAG instructions controlling the behavior of a Data Register.

The ID-Register, Bypass Register, and the Boundary-scan Chain are the Data Registers used for board-level testing. The JTAG Programming Interface (actually consisting of several physical and virtual Data Registers) is used for JTAG Serial Programming via the JTAG interface. The Internal Scan Chain and Break Point Scan Chain are used for On-chip Debugging only.

Test Access Port – TAP

The JTAG interface is accessed through four of the AVR’s pins. In JTAG terminology, these pins constitute the Test Access Port – TAP. These pins are:

- TMS: Test Mode Select. This pin is used for navigating through the TAP-controller state machine.
- TCK: Test Clock. JTAG operation is synchronous to TCK.
- TDI: Test Data In. Serial input data to be shifted in to the Instruction Register or Data Register (Scan Chains).
- TDO: Test Data Out. Serial output data from Instruction register or Data Register.

Figure 120. Boundary-scan Cells for Oscillators and Clock Options

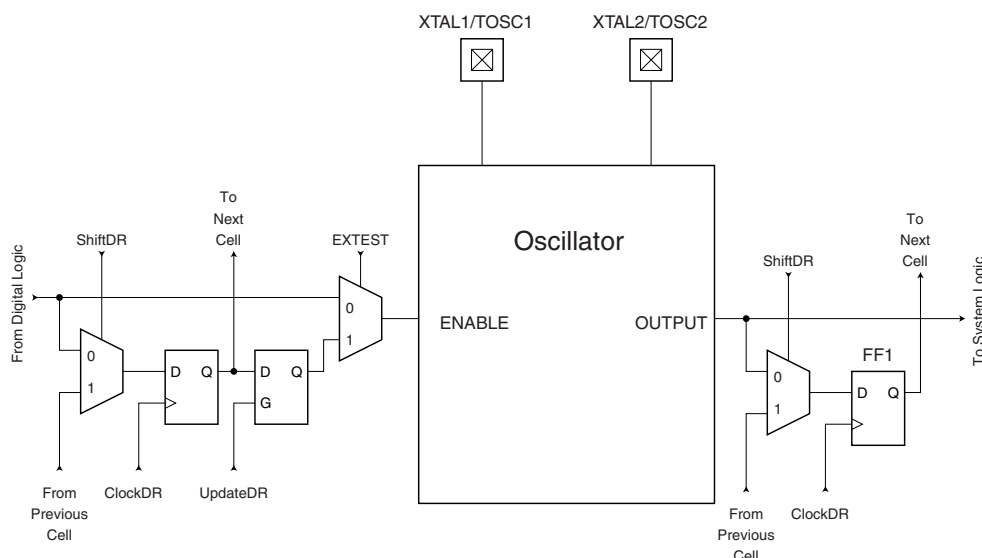


Table 90 summarizes the scan registers for the external clock pin XTAL1, Oscillators with XTAL1/XTAL2 connections as well as 32 kHz Timer Oscillator.

Table 90. Scan Signals for the Oscillators⁽¹⁾⁽²⁾⁽³⁾

Enable Signal	Scanned Clock Line	Clock Option	Scanned Clock Line when not Used
EXTCLKEN	EXTCLK (XTAL1)	External Clock	0
OSCON	OSCCK	External Crystal External Ceramic Resonator	0
RCOSCEN	RCCK	External RC	1
OSC32EN	OSC32CK	Low Freq. External Crystal	0
TOSKON	TOSCK	32 kHz Timer Oscillator	0

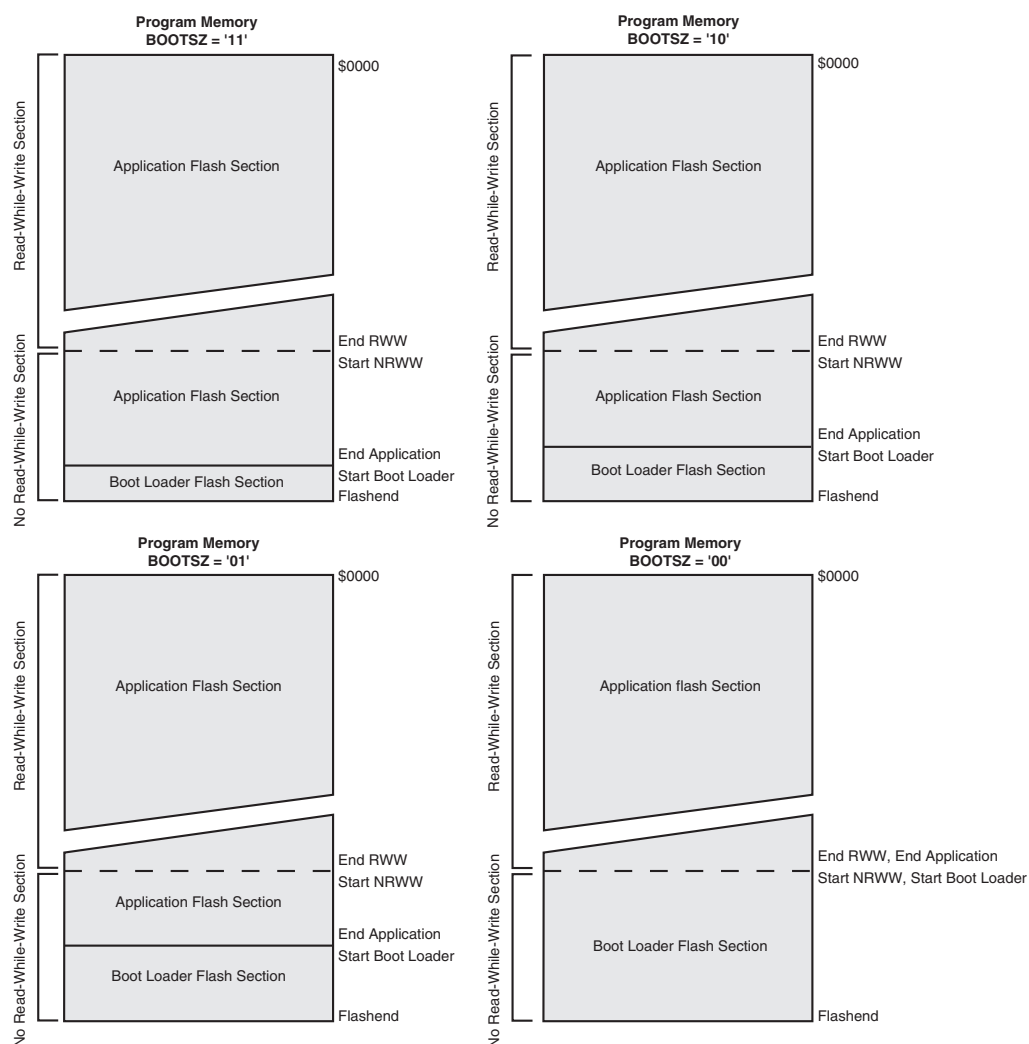
- Notes:
1. Do not enable more than one clock source as main clock at a time.
 2. Scanning an Oscillator output gives unpredictable results as there is a frequency drift between the Internal Oscillator and the JTAG TCK clock. If possible, scanning an external clock is preferred.
 3. The clock configuration is programmed by fuses. As a fuse is not changed run-time, the clock configuration is considered fixed for a given application. The user is advised to scan the same clock option as to be used in the final system. The enable signals are supported in the scan chain because the system logic can disable clock options in sleep modes, thereby disconnecting the Oscillator pins from the scan path if not provided. The INTCAP Fuses are not supported in the scan-chain, so the boundary scan chain can not make a XTAL Oscillator requiring internal capacitors to run unless the fuse is correctly programmed.

Scanning the Analog Comparator

The relevant Comparator signals regarding Boundary-scan are shown in Figure 121. The Boundary-scan cell from Figure 122 is attached to each of these signals. The signals are described in Table 91.

The Comparator need not be used for pure connectivity testing, since all analog inputs are shared with a digital port pin as well.

Figure 125. Memory Sections⁽¹⁾



Note: 1. The parameters in the figure above are given in [Table 100 on page 257](#).

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 96](#) and [Table 97](#) 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 3) does not control reading nor writing by LPM/SPM, if it is attempted.

Performing Page Erase by SPM

To execute Page Erase, set up the address in the Z-pointer, write “X0000011” to SPMCR and execute SPM within four clock cycles after writing SPMCR. 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 must be written zero during this operation.

- Page Erase to the RWW section: The NRWW section can be read during the page erase.
- Page Erase to the NRWW section: The CPU is halted during the operation.

Filling the Temporary Buffer (Page Loading)

To write an instruction word, set up the address in the Z-pointer and data in R1:R0, write “00000001” to SPMCR and execute SPM within four clock cycles after writing SPMCR. 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 SPMCR. 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.

Note: If the EEPROM is written in the middle of an SPM Page Load operation, all data loaded will be lost.

Performing a Page Write

To execute Page Write, set up the address in the Z-pointer, write “X0000101” to SPMCR and execute SPM within four clock cycles after writing SPMCR. The data in R1 and R0 is ignored. The page address must be written to PCPAGE. Other bits in the Z-pointer must be written zero during this operation.

- Page Write to the RWW section: The NRWW section can be read during the Page Write.
- Page Write to the NRWW section: The CPU is halted during the operation.

Using the SPM Interrupt

If the SPM interrupt is enabled, the SPM interrupt will generate a constant interrupt when the SPMEN bit in SPMCR is cleared. This means that the interrupt can be used instead of polling the SPMCR 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 45](#).

Consideration while Updating BLS

Special care must be taken if the user allows the Boot Loader section to be updated by leaving Boot Lock bit11 unprogrammed. An accidental write to the Boot Loader itself can corrupt the entire Boot Loader, and further software updates might be impossible. If it is not necessary to change the Boot Loader software itself, it is recommended to program the Boot Lock bit11 to protect the Boot Loader software from any internal software changes.

Prevent Reading the RWW Section during Self-Programming

During Self-Programming (either Page Erase or Page Write), the RWW section is always blocked for reading. The user software itself must prevent that this section is addressed during the Self-Programming operation. The RWWSB in the SPMCR 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 45](#), 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 256](#) for an example.

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

Flash corruption can easily be avoided by following these design recommendations (one is sufficient):

1. If there is no need for a Boot Loader update in the system, program the Boot Loader Lock bits to prevent any Boot Loader software updates.
2. Keep the AVR RESET active (low) during periods of insufficient power supply voltage. This can be done by enabling the internal Brown-out Detector (BOD) if the operating voltage matches the detection level. If not, an external low V_{CC} Reset Protection circuit can be used. If a reset occurs while a write operation is in progress, the write operation will be completed provided that the power supply voltage is sufficient.
3. Keep the AVR core in Power-down Sleep mode during periods of low V_{CC} . This will prevent the CPU from attempting to decode and execute instructions, effectively protecting the SPMCR Register and thus the Flash from unintentional writes.

Programming Time for Flash when using SPM

The Calibrated RC Oscillator is used to time Flash accesses. [Table 99](#) shows the typical programming time for Flash accesses from the CPU.

Table 99. 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

Programming the Flash

The Flash is organized in pages, see [Table 107 on page 262](#). When programming the Flash, the program data is latched into a page buffer. This allows one page of program data to be programmed simultaneously. The following procedure describes how to program the entire Flash memory:

A. Load Command "Write Flash"

1. Set XA1, XA0 to "10". This enables command loading.
2. Set BS1 to "0".
3. Set DATA to "0001 0000". This is the command for Write Flash.
4. Give XTAL1 a positive pulse. This loads the command.

B. Load Address Low byte

1. Set XA1, XA0 to "00". This enables address loading.
2. Set BS1 to "0". This selects low address.
3. Set DATA = Address Low byte (\$00 - \$FF).
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 (\$00 - \$FF).
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 (\$00 - \$FF).
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 PAGES a positive pulse. This latches the data bytes. (See [Figure 129](#) 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 128 on page 267](#). Note that if less than 8 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 (\$00 - \$FF).
4. Give XTAL1 a positive pulse. This loads the address High byte.

H. Program Page

1. Set BS1 = "0"
2. Give \overline{WR} a negative pulse. This starts programming of the entire page of data. RDY/ \overline{BSY} goes low.
3. Wait until RDY/ \overline{BSY} goes high. (See [Figure 129](#) for signal waveforms)

9. Added [Table 73, “TWI Bit Rate Prescaler,” on page 182](#) to describe the TWPS bits in the [“TWI Status Register – TWSR” on page 181](#).
10. Added section [“Default Clock Source” on page 25](#).
11. Added note about frequency variation when using an external clock. Note added in [“External Clock” on page 31](#). An extra row and a note added in [Table 118 on page 293](#).
12. Various minor TWI corrections.
13. Added [“Power Consumption” data in “Features” on page 1](#).
14. Added section [“EEPROM Write During Power-down Sleep Mode” on page 22](#).
15. Added note about Differential Mode with Auto Triggering in [“Prescaling and Conversion Timing” on page 207](#).
16. Added updated [“Packaging Information” on page 337](#).

Rev. 2466E-10/02

1. Updated [“DC Characteristics” on page 291](#).

Rev. 2466D-09/02

1. Changed all Flash write/erase cycles from 1,000 to 10,000.
2. Updated the following tables: [Table 4 on page 26](#), [Table 15 on page 38](#), [Table 42 on page 85](#), [Table 45 on page 111](#), [Table 46 on page 111](#), [Table 59 on page 143](#), [Table 67 on page 167](#), [Table 90 on page 235](#), [Table 102 on page 258](#), [“DC Characteristics” on page 291](#), [Table 119 on page 293](#), [Table 121 on page 295](#), and [Table 122 on page 297](#).
3. Updated [“Errata” on page 340](#).

Rev. 2466C-03/02

1. Updated typical EEPROM programming time, [Table 1 on page 20](#).
2. Updated typical start-up time in the following tables:
[Table 3 on page 25](#), [Table 5 on page 27](#), [Table 6 on page 28](#), [Table 8 on page 29](#), [Table 9 on page 29](#), and [Table 10 on page 29](#).
3. Updated [Table 17 on page 43](#) with typical WDT Time-out.
4. **Added Some Preliminary Test Limits and Characterization Data.**
Removed some of the TBD's in the following tables and pages:
[Table 15 on page 38](#), [Table 16 on page 42](#), [Table 116 on page 272](#) (table removed in document review #D), [“Electrical Characteristics” on page 291](#), [Table 119 on page 293](#), [Table 121 on page 295](#), and [Table 122 on page 297](#).
5. **Updated TWI Chapter.**
Added the note at the end of the [“Bit Rate Generator Unit” on page 178](#).
6. **Corrected description of ADSC bit in “ADC Control and Status Register A – ADCSRA” on page 219.**
7. **Improved description on how to do a polarity check of the ADC doff results in “ADC Conversion Result” on page 216.**