

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	Active
Core Processor	AVR
Core Size	8-Bit
Speed	10MHz
Connectivity	I <sup>2</sup> C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	23
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)	1.8V ~ 5.5V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 105°C (TA)
Mounting Type	Surface Mount
Package / Case	32-VFQFN Exposed Pad
Supplier Device Package	32-VQFN (5x5)
Purchase URL	<a href="https://www.e-xfl.com/product-detail/atmel/atmega168v-10mq">https://www.e-xfl.com/product-detail/atmel/atmega168v-10mq</a>

priority level. RESET has the highest priority, and next is INT0 – the External Interrupt Request 0. The Interrupt Vectors can be moved to the start of the Boot Flash section by setting the IVSEL bit in the MCU Control Register (MCUCR). Refer to ["Interrupts" on page 54](#) 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, ATmega88 and ATmega168" on page 264](#).

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.

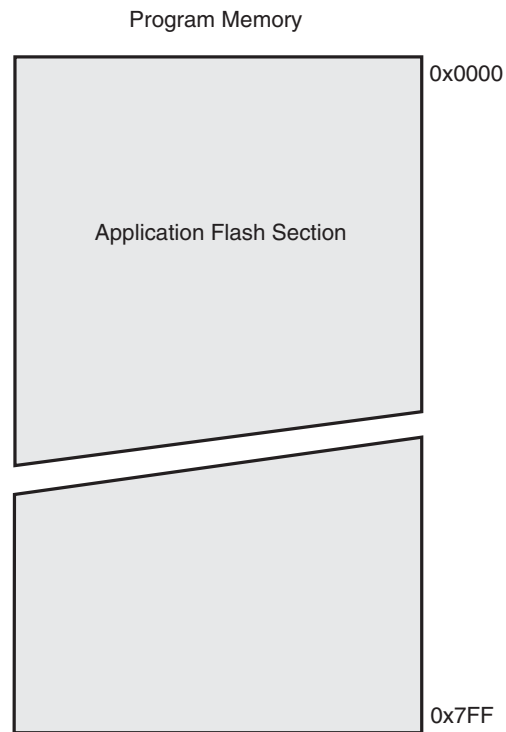
The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have Interrupt Flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered.

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.

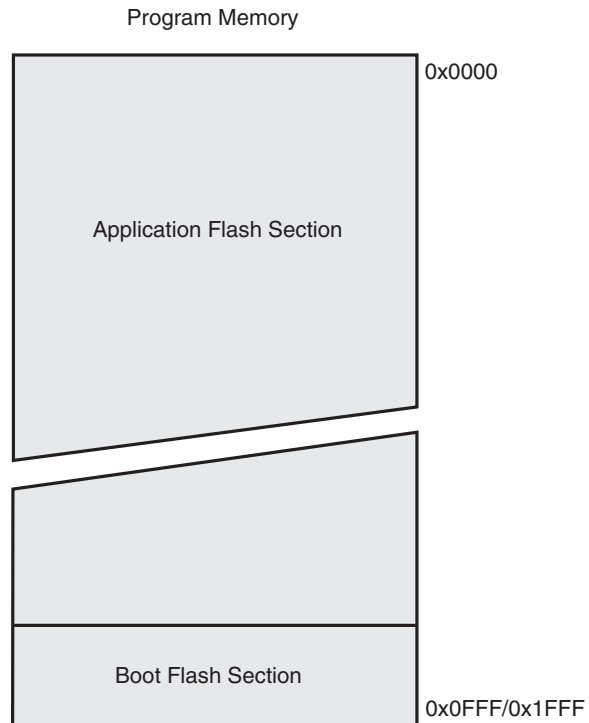
Note that the Status Register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software.

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.

**Figure 5-1.** Program Memory Map, ATmega48



**Figure 5-2.** Program Memory Map, ATmega88 and ATmega168



## 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
Read/Write	R	R	R	R	R	R	R	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 ATmega48/88/168 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 256/512/512 bytes EEPROM space. The EEPROM data bytes are addressed linearly between 0 and 255/511/511. The initial value of EEAR is undefined. A proper value must be written before the EEPROM may be accessed.

EEAR8 is an unused bit in ATmega48 and must always be written to zero.

## 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 ATmega48/88/168 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

## 6.7 128 kHz Internal Oscillator

The 128 kHz internal Oscillator is a low power Oscillator providing a clock of 128 kHz. The frequency is nominal at 3V and 25°C. This clock may be select as the system clock by programming the CKSEL Fuses to “11” as shown in [Table 6-10](#).

**Table 6-10.** 128 kHz Internal Oscillator Operating Modes

Nominal Frequency	CKSEL3..0
128 kHz	0011

Note: 1. The frequency is preliminary value. Actual value is TBD.

When this clock source is selected, start-up times are determined by the SUT Fuses as shown in [Table 6-11](#).

**Table 6-11.** Start-up Times for the 128 kHz Internal Oscillator

Power Conditions	Start-up Time from Power-down and Power-save	Additional Delay from Reset	SUT1..0
BOD enabled	6 CK	14CK <sup>(1)</sup>	00
Fast rising power	6 CK	14CK + 4 ms	01
Slowly rising power	6 CK	14CK + 64 ms	10
Reserved			11

Note: 1. If the RSTDISBL fuse is programmed, this start-up time will be increased to 14CK + 4.1 ms to ensure programming mode can be entered.

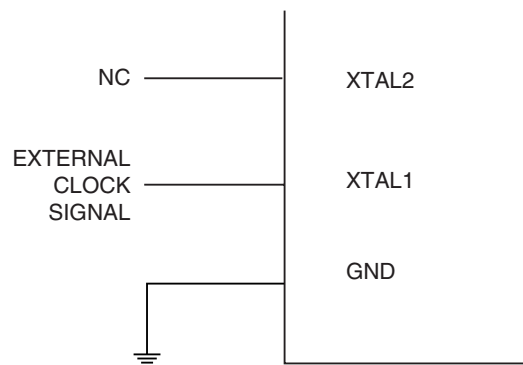
## 6.8 External Clock

To drive the device from an external clock source, XTAL1 should be driven as shown in [Figure 6-4 on page 33](#). To run the device on an external clock, the CKSEL Fuses must be programmed to “0000” (see [Table 6-12](#)).

**Table 6-12.** Crystal Oscillator Clock Frequency

Frequency	CKSEL3..0
0 - 20 MHz	0011

**Figure 6-4.** External Clock Drive Configuration



When this clock source is selected, start-up times are determined by the SUT Fuses as shown in [Table 6-13](#).

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

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

1. Write the Clock Prescaler Change Enable (CLKPCE) bit to one and all other bits in CLKPR to zero.
2. Within four cycles, write the desired value to CLKPS while writing a zero to CLKPCE.

Interrupts must be disabled when changing prescaler setting to make sure the write procedure is not interrupted.

## 6.11.1 Clock Prescale Register – CLKPR

Bit	7	6	5	4	3	2	1	0	
	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0	CLKPR
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	See Bit Description				

### • Bit 7 – CLKPCE: Clock Prescaler Change Enable

The CLKPCE bit must be written to logic one to enable change of the CLKPS bits. The CLKPCE bit is only updated when the other bits in CLKPR are simultaneously written to zero. CLKPCE is cleared by hardware four cycles after it is written or when CLKPS bits are written. Rewriting the CLKPCE bit within this time-out period does neither extend the time-out period, nor clear the CLKPCE bit.

### • Bits 3..0 – CLKPS3..0: Clock Prescaler Select Bits 3 - 0

These bits define the division factor between the selected clock source and the internal system clock. These bits can be written run-time to vary the clock frequency to suit the application requirements. As the divider divides the master clock input to the MCU, the speed of all synchronous peripherals is reduced when a division factor is used. The division factors are given in [Table 6-14](#).

The CKDIV8 Fuse determines the initial value of the CLKPS bits. If CKDIV8 is unprogrammed, the CLKPS bits will be reset to “0000”. If CKDIV8 is programmed, CLKPS bits are reset to “0011”, giving a division factor of 8 at start up. This feature should be used if the selected clock source has a higher frequency than the maximum frequency of the device at the present operating conditions. Note that any value can be written to the CLKPS bits regardless of the CKDIV8 Fuse setting. The Application software must ensure that a sufficient division factor is chosen if the selected clock source has a higher frequency than the maximum frequency of the device at the present operating conditions. The device is shipped with the CKDIV8 Fuse programmed.

### 11.1.1 External Interrupt Control Register A – EICRA

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

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
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 unused bits in the ATmega48/88/168, and will always read as zero.

- **Bit 3, 2 – ISC11, ISC10: Interrupt Sense Control 1 Bit 1 and Bit 0**

The External Interrupt 1 is activated by the external pin INT1 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT1 pin that activate the interrupt are defined in [Table 11-1](#). The value on the INT1 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

**Table 11-1.** Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

- **Bit 1, 0 – ISC01, ISC00: Interrupt Sense Control 0 Bit 1 and Bit 0**

The External Interrupt 0 is activated by the external pin INT0 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT0 pin that activate the interrupt are defined in [Table 11-2](#). The value on the INT0 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

**Table 11-2.** Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

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 12-7 on page 101](#)). 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.

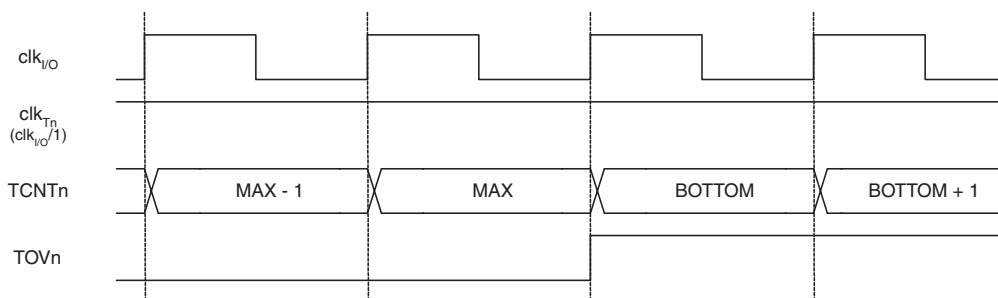
At the very start of period 2 in [Figure 12-7](#) OCnx has a transition from high to low even though there is no Compare Match. The point of this transition is to guarantee symmetry around BOTTOM. There are two cases that give a transition without Compare Match.

- OCRnx changes its value from MAX, like in [Figure 12-7](#). 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.

## 12.7 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 12-8](#) 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 12-8.** Timer/Counter Timing Diagram, no Prescaling



[Figure 12-9](#) shows the same timing data, but with the prescaler enabled.



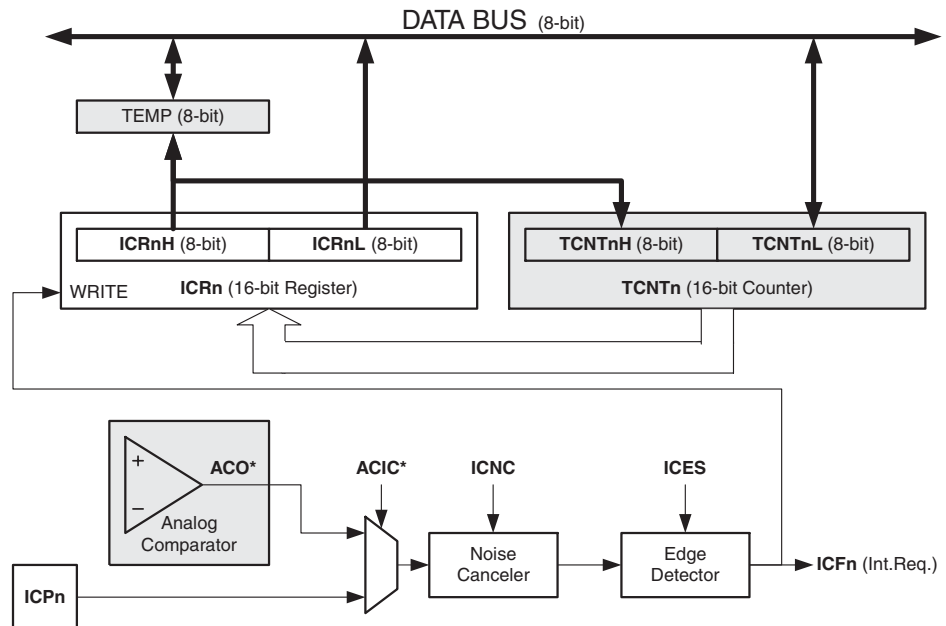
The Timer/Counter Overflow Flag (TOV1) is set according to the mode of operation selected by the WGM13:0 bits. TOV1 can be used for generating a CPU interrupt.

### 13.5 Input Capture Unit

The Timer/Counter incorporates an Input Capture unit that can capture external events and give them a time-stamp indicating time of occurrence. The external signal indicating an event, or multiple events, can be applied via the ICP1 pin or alternatively, via the analog-comparator unit. The time-stamps can then be used to calculate frequency, duty-cycle, and other features of the signal applied. Alternatively the time-stamps can be used for creating a log of the events.

The Input Capture unit is illustrated by the block diagram shown in Figure 13-3. The elements of the block diagram that are not directly a part of the Input Capture unit are gray shaded. The small “n” in register and bit names indicates the Timer/Counter number.

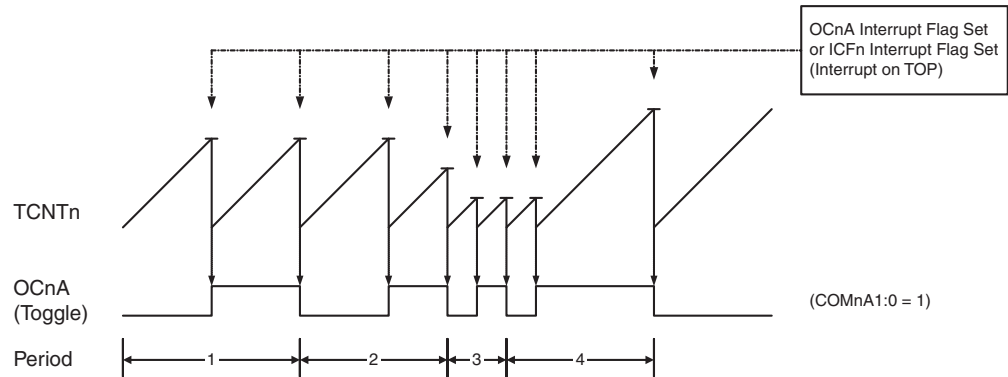
**Figure 13-3.** Input Capture Unit Block Diagram



When a change of the logic level (an event) occurs on the *Input Capture pin* (ICP1), alternatively on the *Analog Comparator output* (ACO), and this change confirms to the setting of the edge detector, a capture will be triggered. When a capture is triggered, the 16-bit value of the counter (TCNT1) is written to the *Input Capture Register* (ICR1). The *Input Capture Flag* (ICF1) is set at the same system clock as the TCNT1 value is copied into ICR1 Register. If enabled (ICIE1 = 1), the Input Capture Flag generates an Input Capture interrupt. The ICF1 Flag is automatically cleared when the interrupt is executed. Alternatively the ICF1 Flag can be cleared by software by writing a logical one to its I/O bit location.

Reading the 16-bit value in the *Input Capture Register* (ICR1) is done by first reading the low byte (ICR1L) and then the high byte (ICR1H). When the low byte is read the high byte is copied into the high byte temporary register (TEMP). When the CPU reads the ICR1H I/O location it will access the TEMP Register.

The ICR1 Register can only be written when using a Waveform Generation mode that utilizes the ICR1 Register for defining the counter’s TOP value. In these cases the *Waveform Genera-*

**Figure 13-6. CTC Mode, Timing Diagram**


An interrupt can be generated at each time the counter value reaches the TOP value by either using the OCF1A or ICF1 Flag according to the register used to define the TOP value. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing the TOP to a value close to BOTTOM when the counter is running with none or a low prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written to OCR1A or ICR1 is lower than the current value of TCNT1, the counter will miss the compare match. The counter will then have to count to its maximum value (0xFFFF) and wrap around starting at 0x0000 before the compare match can occur. In many cases this feature is not desirable. An alternative will then be to use the fast PWM mode using OCR1A for defining TOP (WGM13:0 = 15) since the OCR1A then will be double buffered.

For generating a waveform output in CTC mode, the OC1A output can be set to toggle its logical level on each compare match by setting the Compare Output mode bits to toggle mode (COM1A1:0 = 1). The OC1A value will not be visible on the port pin unless the data direction for the pin is set to output (DDR\_OC1A = 1). The waveform generated will have a maximum frequency of  $f_{OC1A} = f_{clk\_I/O}/2$  when OCR1A is set to zero (0x0000). The waveform frequency is defined by the following equation:

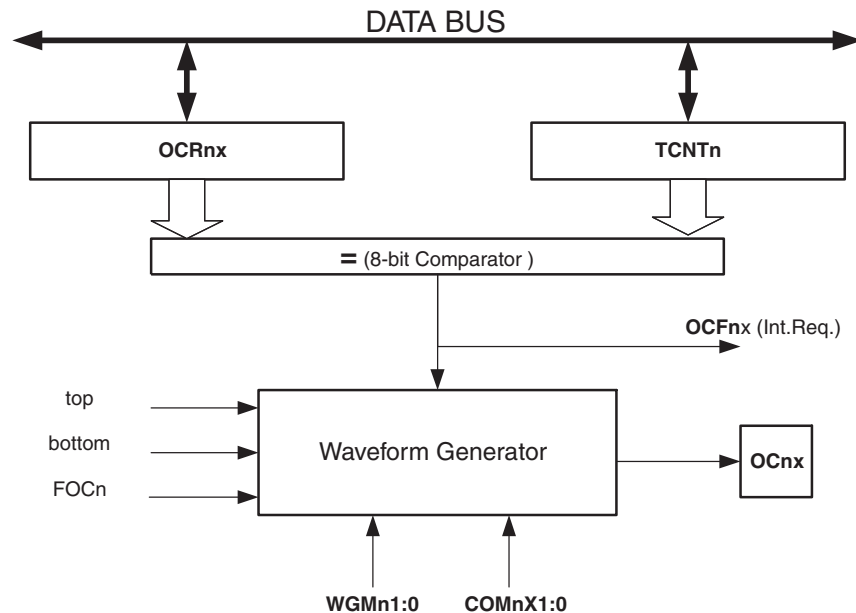
$$f_{OCnA} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCRnA)}$$

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

As for the Normal mode of operation, the TOV1 Flag is set in the same timer clock cycle that the counter counts from MAX to 0x0000.

### 13.8.3 Fast PWM Mode

The *fast Pulse Width Modulation* or fast PWM mode (WGM13:0 = 5, 6, 7, 14, or 15) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OC1x) is set on the compare match between TCNT1 and OCR1x, and cleared at TOP. In inverting Compare Output mode output is cleared on compare match and set at TOP. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct and phase and frequency correct PWM modes that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), hence reduces total system cost.

**Figure 15-3. Output Compare Unit, Block Diagram**


The OCR2x Register is double buffered when using any of the Pulse Width Modulation (PWM) modes. For the Normal and Clear Timer on Compare (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR2x Compare Register to either top or bottom of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCR2x Register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR2x Buffer Register, and if double buffering is disabled the CPU will access the OCR2x directly.

#### 15.4.1 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 (FOC2x) bit. Forcing compare match will not set the OCF2x Flag or reload/clear the timer, but the OC2x pin will be updated as if a real compare match had occurred (the COM2x1:0 bits settings define whether the OC2x pin is set, cleared or toggled).

#### 15.4.2 Compare Match Blocking by TCNT2 Write

All CPU write operations to the TCNT2 Register will block any compare match that occurs in the next timer clock cycle, even when the timer is stopped. This feature allows OCR2x to be initialized to the same value as TCNT2 without triggering an interrupt when the Timer/Counter clock is enabled.

#### 15.4.3 Using the Output Compare Unit

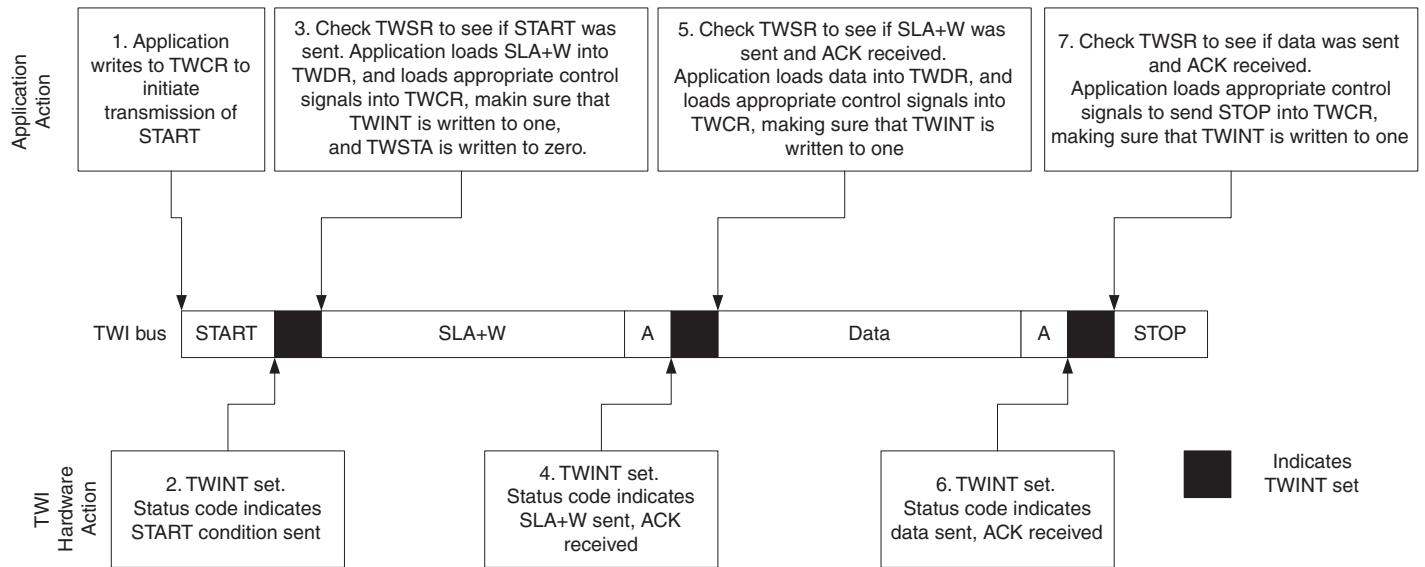
Since writing TCNT2 in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNT2 when using the Output Compare channel, independently of whether the Timer/Counter is running or not. If the value written to TCNT2 equals the OCR2x value, the compare match will be missed, resulting in incorrect waveform generation. Similarly, do not write the TCNT2 value equal to BOTTOM when the counter is downcounting.

**Table 17-9.** Examples of UBRRn Settings for Commonly Used Oscillator Frequencies

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

Note: 1. UBRRn = 0, Error = 0.0%

**Figure 19-11. 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

value	Device's Own Slave Address
-------	----------------------------

The upper seven bits are the address to which the 2-wire Serial Interface will respond when addressed by a Master. If the LSB is set, the TWI will respond to the general call address (0x00), otherwise it will ignore the general call address.

<b>TWCR</b>	<b>TWINT</b>	<b>TWEA</b>	<b>TWSTA</b>	<b>TWSTO</b>	<b>TWWC</b>	<b>TWEN</b>	<b>–</b>	<b>TWIE</b>
value	0	1	0	0	0	1	0	X

TWEN must be written to one to enable the TWI. The TWEA bit must be written to one to enable the acknowledgement of the device's own slave address or the general call address. TWSTA and TWSTO must be written to zero.

When TWAR and TWCR have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address if enabled) followed by the data direction bit. If the direction bit is "1" (read), the TWI will operate in ST mode, otherwise SR mode is entered. After its own slave address and the write bit have been received, the TWINT Flag is set and a valid status code can be read from TWSR. The status code is used to determine the appropriate software action. The appropriate action to be taken for each status code is detailed in [Table 19-6](#). The Slave Transmitter mode may also be entered if arbitration is lost while the TWI is in the Master mode (see state 0xB0).

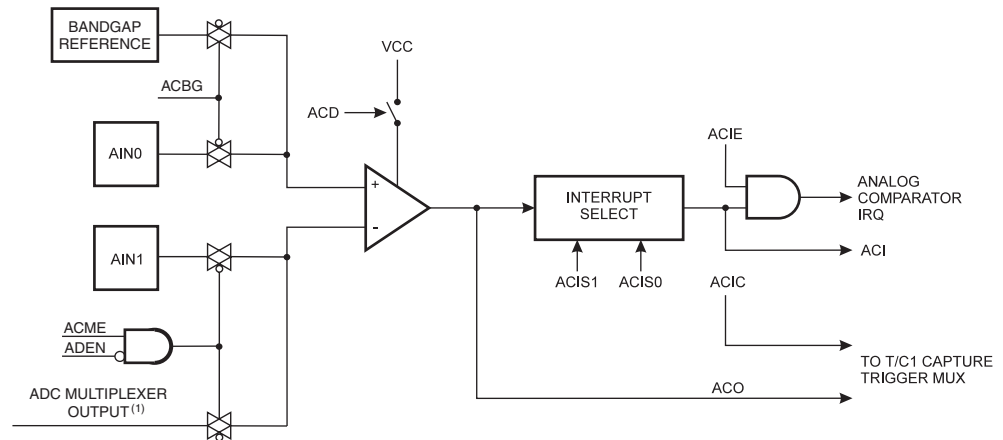
If the TWEA bit is written to zero during a transfer, the TWI will transmit the last byte of the transfer. State 0xC0 or state 0xC8 will be entered, depending on whether the Master Receiver transmits a NACK or ACK after the final byte. The TWI is switched to the not addressed Slave mode, and will ignore 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 2-wire Serial Bus is still monitored and address recognition may resume at any time by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the 2-wire Serial Bus.

In all sleep modes other than Idle mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the 2-wire Serial Bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock low during the wake up and until the TWINT Flag is cleared (by writing it to one). Further data transmission will be carried out as normal, with the AVR clocks running as normal. Observe that if the AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions.

Note that the 2-wire Serial Interface Data Register – TWDR does not reflect the last byte present on the bus when waking up from these sleep modes.

**Figure 20-1. Analog Comparator Block Diagram<sup>(2)</sup>**



- Notes:
1. See [Table 20-2 on page 238](#).
  2. Refer to [Figure 1-1 on page 2](#) and [Table 10-9 on page 78](#) for Analog Comparator pin placement.

#### 20.0.1 ADC Control and Status Register B – ADCSRB

Bit	7	6	5	4	3	2	1	0	
	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

##### • Bit 6 – ACME: Analog Comparator Multiplexer Enable

When this bit is written logic one and the ADC is switched off (ADEN in ADCSRA is zero), the ADC multiplexer selects the negative input to the Analog Comparator. When this bit is written logic zero, AIN1 is applied to the negative input of the Analog Comparator. For a detailed description of this bit, see ["Analog Comparator Multiplexed Input" on page 237](#).

#### 20.0.2 Analog Comparator Control and Status Register – ACSR

Bit	7	6	5	4	3	2	1	0	
	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

##### • Bit 7 – ACD: Analog Comparator Disable

When this bit is written logic one, the power to the Analog Comparator is switched off. This bit can be set at any time to turn off the Analog Comparator. This will reduce power consumption in Active and Idle mode. When changing the ACD bit, the Analog Comparator Interrupt must be disabled by clearing the ACIE bit in ACSR. Otherwise an interrupt can occur when the bit is changed.

##### • Bit 6 – ACBG: Analog Comparator Bandgap Select

When this bit is set, a fixed bandgap reference voltage replaces the positive input to the Analog Comparator. When this bit is cleared, AIN0 is applied to the positive input of the Analog Comparator. See [Section "8.1" on page 48](#).

- **Bit 0 – SELFPRGEN: Self Programming 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 SELFPRGEN 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 SELFPRGEN 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 SELFPRGEN bit remains high until the operation is completed.

Writing any other combination than “10001”, “01001”, “00101”, “00011” or “00001” in the lower five bits will have no effect.

### 23.1.2 EEPROM Write Prevents Writing to SPMCSR

Note that an EEPROM write operation will block all software programming to Flash. Reading the Fuses and Lock bits from software will also be prevented during the EEPROM write operation. It is recommended that the user checks the status bit (EEPE) in the EECR Register and verifies that the bit is cleared before writing to the SPMCSR Register.

### 23.1.3 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 SELFPRGEN bits in SPMCSR. When an LPM instruction is executed within three CPU cycles after the BLBSET and SELFPRGEN bits are set in SPMCSR, the value of the Lock bits will be loaded in the destination register. The BLBSET and SELFPRGEN 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 SELFPRGEN are cleared, LPM will work as described in the Instruction set Manual.

Bit	7	6	5	4	3	2	1	0
Rd	–	–	–	–	–	–	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 SELFPRGEN bits in SPMCSR. When an LPM instruction is executed within three cycles after the BLBSET and SELFPRGEN bits are set in the SPMCSR, the value of the Fuse Low byte (FLB) will be loaded in the destination register as shown below. See [Table 25-5 on page 282](#) 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

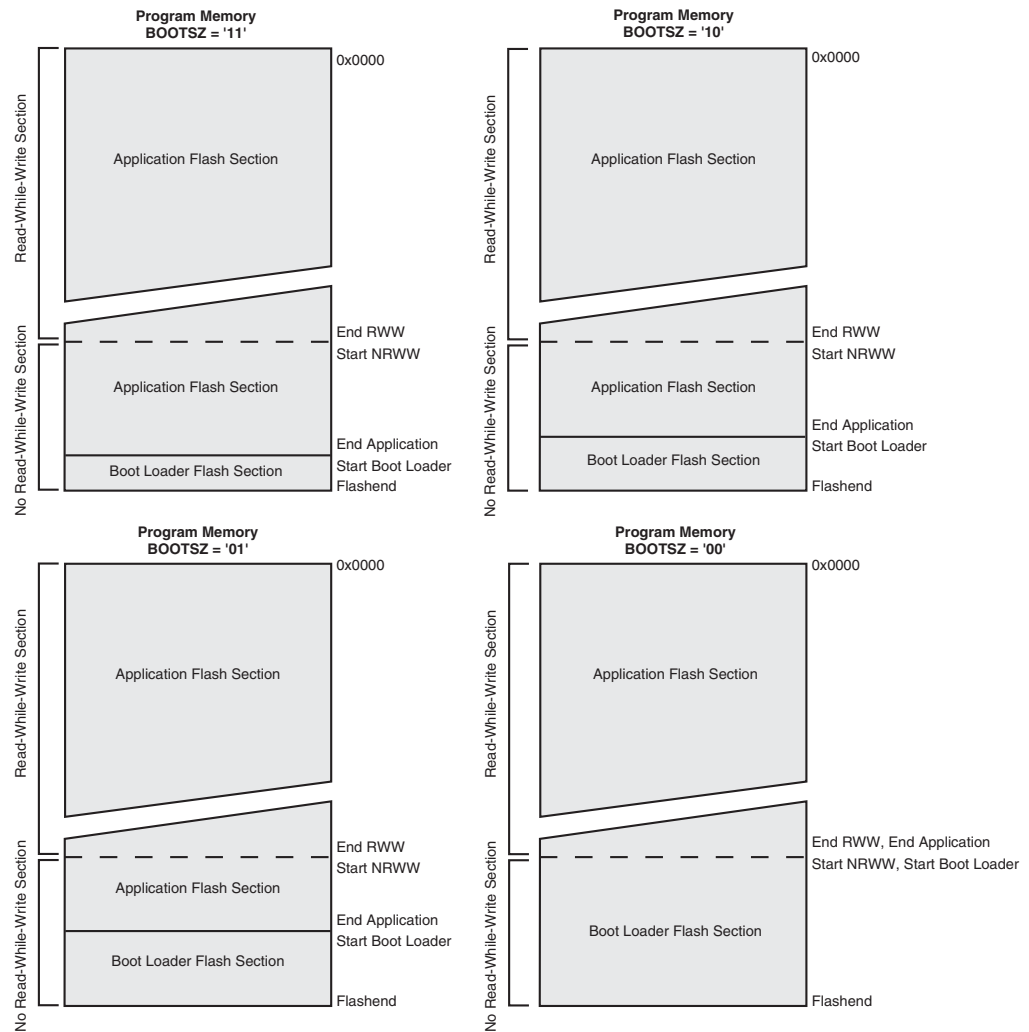
Similarly, when reading the Fuse High byte (FHB), load 0x0003 in the Z-pointer. When an LPM instruction is executed within three cycles after the BLBSET and SELFPRGEN bits are set in the SPMCSR, the value of the Fuse High byte will be loaded in the destination register as shown below. See [Table 25-4 on page 281](#) for detailed description and mapping of the Extended Fuse byte.

Bit	7	6	5	4	3	2	1	0
Rd	FHB7	FHB6	FHB5	FHB4	FHB3	FHB2	FHB1	FHB0

Similarly, when reading the Extended Fuse byte (EFB), load 0x0002 in the Z-pointer. When an LPM instruction is executed within three cycles after the BLBSET and SELFPRGEN bits are set in the SPMCSR, the value of the Extended Fuse byte will be loaded in the destination register as



**Figure 24-2. Memory Sections**



Note: 1. The parameters in the figure above are given in [Table 24-6 on page 276](#).

## 24.4 Boot Loader Lock Bits

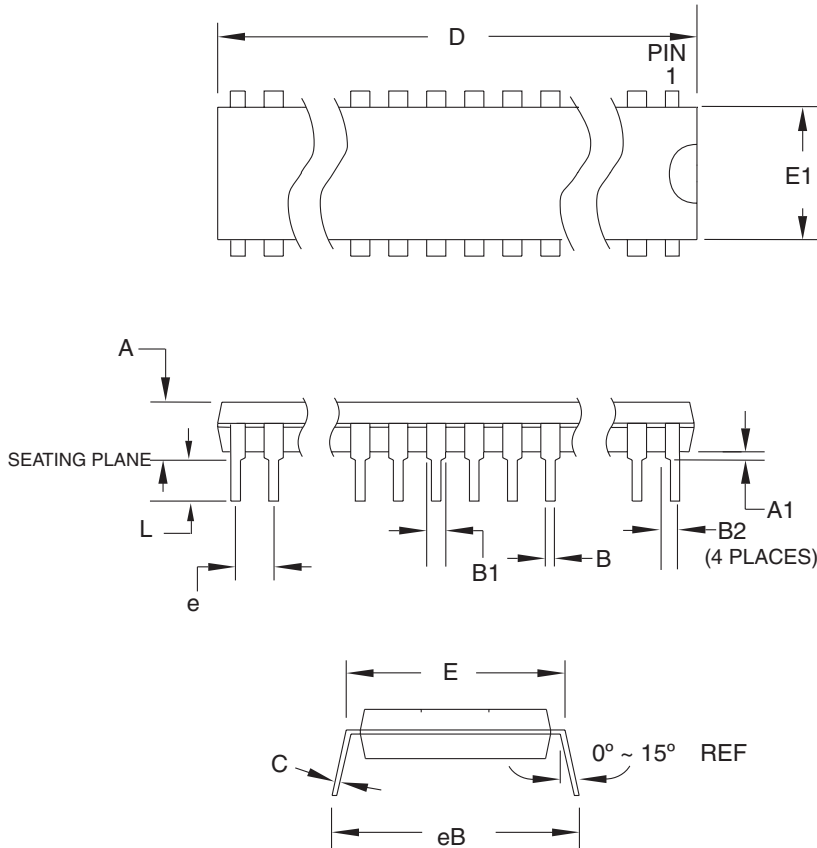
If no Boot Loader capability is needed, the entire Flash is available for application code. The Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

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 24-2](#) and [Table 24-3](#) 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.

31.2 28P3



COMMON DIMENSIONS  
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	—	—	4.5724	
A1	0.508	—	—	
D	34.544	—	34.798	Note 1
E	7.620	—	8.255	
E1	7.112	—	7.493	Note 1
B	0.381	—	0.533	
B1	1.143	—	1.397	
B2	0.762	—	1.143	
L	3.175	—	3.429	
C	0.203	—	0.356	
eB	—	—	10.160	
e	2.540 TYP			

Note: 1. Dimensions D and E1 do not include mold Flash or Protrusion.  
Mold Flash or Protrusion shall not exceed 0.25 mm (0.010").

09/28/01



2325 Orchard Parkway  
San Jose, CA 95131

TITLE

**28P3**, 28-lead (0.300"/7.62 mm Wide) Plastic Dual  
Inline Package (PDIP)

DRAWING NO.

28P3

REV.

B



**33.4 Rev. 2545B-01/04**

1. Added PDIP to "I/O and Packages", updated "Speed Grade" and Power Consumption Estimates in 33."Features" on page 1.
2. Updated "Stack Pointer" on page 11 with RAMEND as recommended Stack Pointer value.
3. Added section "Power Reduction Register" on page 39 and a note regarding the use of the PRR bits to 2-wire, Timer/Counters, USART, Analog Comparator and ADC sections.
4. Updated "Watchdog Timer" on page 49.
5. Updated Figure 13-2 on page 128 and Table 13-3 on page 129.
6. Extra Compare Match Interrupt OCF2B added to features in section "8-bit Timer/Counter2 with PWM and Asynchronous Operation" on page 138
7. Updated Table 7-2 on page 39, Table 21-5 on page 254, Table 25-4 to Table 25-7 on page 281 to 283 and Table 21-1 on page 244. Added note 2 to Table 25-1 on page 280. Fixed typo in Table 11-1 on page 84.
8. Updated whole "ATmega48/88/168 Typical Characteristics – Preliminary Data" on page 307.
9. Added item 2 to 5 in "Errata ATmega48" on page 347.
10. Renamed the following bits:
  - SPMEN to SELFPRGEN
  - PSR2 to PSRASY
  - PSR10 to PSRSYNC
  - Watchdog Reset to Watchdog System Reset
11. Updated C code examples containing old IAR syntax.
12. Updated BLBSET description in "Store Program Memory Control and Status Register – SPMCSR" on page 269.





6.10	Timer/Counter Oscillator .....	34
6.11	System Clock Prescaler .....	34
<b>7</b>	<b><i>Power Management and Sleep Modes .....</i></b>	<b>37</b>
7.1	Idle Mode .....	38
7.2	ADC Noise Reduction Mode .....	38
7.3	Power-down Mode .....	38
7.4	Power-save Mode .....	38
7.5	Standby Mode .....	39
7.6	Power Reduction Register .....	39
7.7	Minimizing Power Consumption .....	41
<b>8</b>	<b><i>System Control and Reset .....</i></b>	<b>43</b>
8.1	Internal Voltage Reference .....	48
8.2	Watchdog Timer .....	49
<b>9</b>	<b><i>Interrupts .....</i></b>	<b>54</b>
9.1	Interrupt Vectors in ATmega48 .....	54
9.2	Interrupt Vectors in ATmega88 .....	56
9.3	Interrupt Vectors in ATmega168 .....	59
<b>10</b>	<b><i>I/O-Ports .....</i></b>	<b>64</b>
10.1	Introduction .....	64
10.2	Ports as General Digital I/O .....	65
10.3	Alternate Port Functions .....	69
10.4	Register Description for I/O Ports .....	81
<b>11</b>	<b><i>External Interrupts .....</i></b>	<b>83</b>
11.1	Pin Change Interrupt Timing .....	83
<b>12</b>	<b><i>8-bit Timer/Counter0 with PWM .....</i></b>	<b>88</b>
12.1	Overview .....	88
12.2	Timer/Counter Clock Sources .....	89
12.3	Counter Unit .....	89
12.4	Output Compare Unit .....	90
12.5	Compare Match Output Unit .....	92
12.6	Modes of Operation .....	93
12.7	Timer/Counter Timing Diagrams .....	97
12.8	8-bit Timer/Counter Register Description .....	99
<b>13</b>	<b><i>16-bit Timer/Counter1 with PWM .....</i></b>	<b>106</b>