Welcome to 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	I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	23
Program Memory Size	8KB (4K 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 ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	32-TQFP
Supplier Device Package	32-TQFP (7x7)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atmega88v-15at

5. AVR ATmega48/88/168 Memories

This section describes the different memories in the Atmel® ATmega48/88/168. The AVR® architecture has two main memory spaces, the data memory and the program memory space. In addition, the Atmel ATmega48/88/168 features an EEPROM memory for data storage. All three memory spaces are linear and regular.

5.1 In-System Reprogrammable Flash Program Memory

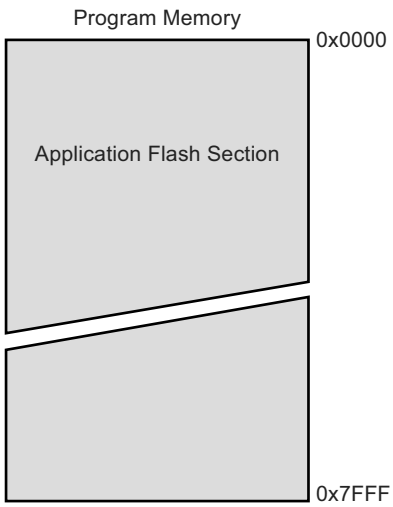
The Atmel ATmega48/88/168 contains 4/8/16K bytes on-chip in-system reprogrammable flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the flash is organized as 2/4/8K x 16. For software security, the flash program memory space is divided into two sections, boot loader section and application program section in Atmel ATmega88 and ATmega168. ATmega48 does not have separate boot loader and application program sections, and the SPM instruction can be executed from the entire flash. See SELFPRGEN description in Section 23.4.1 “Store Program Memory Control and Status Register – SPMCSR” on page 225 and Section 24.5.1 “Store Program Memory Control and Status Register – SPMCSR” on page 233 for more details.

The flash memory has an endurance of at least 75,000 write/erase cycles. The Atmel ATmega48/88/168 program counter (PC) is 11/12/13 bits wide, thus addressing the 2/4/8K program memory locations. The operation of boot program section and associated boot lock bits for software protection are described in detail in Section 23. “Self-Programming the Flash, ATmega48” on page 223 and Section 24. “Boot Loader Support – Read-While-Write Self-Programming, ATmega88 and ATmega168” on page 229. Section 25. “Memory Programming” on page 242 contains a detailed description on flash programming in SPI- or parallel programming mode.

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

Timing diagrams for instruction fetch and execution are presented in Section 4.7 “Instruction Execution Timing” on page 13.

Figure 5-1. Program Memory Map, ATmega48



- **Bit 2 – EEMPE: EEPROM Master Write 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 Write 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 Section 24. “Boot Loader Support – Read-While-Write Self-Programming, ATmega88 and ATmega168” on page 229 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	Typical Programming Time
EEPROM write (from CPU)	26,368	3.3ms

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

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

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

This bit is reserved in Atmel® ATmega48/88/168 and will always read as zero.

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

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

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

If using debugWIRE on-chip debug system, this bit should not be written to one. Writing a logic one to this bit shuts down the serial peripheral interface by stopping the clock to the module. When waking up the SPI again, the SPI should be re-initialized to ensure proper operation.

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

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

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

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

7.8 Minimizing Power Consumption

There are several possibilities 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.

7.8.1 Analog to Digital Converter

If enabled, the ADC will be enabled in all sleep modes. To save power, the ADC should be disabled before entering any sleep mode. When the ADC is turned off and on again, the next conversion will be an extended conversion. Refer to Section 21. “Analog-to-Digital Converter” on page 206 for details on ADC operation.

7.8.2 Analog Comparator

When entering idle mode, the analog comparator should be disabled if not used. When entering ADC noise reduction mode, the analog comparator should be disabled. In other sleep modes, the analog comparator is automatically disabled. However, if the analog comparator is set up to use the internal voltage reference as input, the analog comparator should be disabled in all sleep modes. Otherwise, the internal voltage reference will be enabled, independent of sleep mode. Refer to Section 20. “Analog Comparator” on page 203 for details on how to configure the analog comparator.

7.8.3 Brown-out Detector

If the brown-out detector is not needed by the application, this module should be turned off. If the brown-out detector is enabled by the BODLEVEL fuses, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to Section 8.5 “Brown-out Detection” on page 41 for details on how to configure the brown-out detector.

The most typical and general program setup for the reset and interrupt vector addresses in Atmel® ATmega48 is:

```
Address      Labels Code                 Comments
0x000               rjmp   RESET        ; Reset Handler
0x001               rjmp   EXT_INT0     ; IRQ0 Handler
0x002               rjmp   EXT_INT1     ; IRQ1 Handler
0x003               rjmp   PCINT0       ; PCINT0 Handler
0x004               rjmp   PCINT1       ; PCINT1 Handler
0x005               rjmp   PCINT2       ; PCINT2 Handler
0x006               rjmp   WDT          ; Watchdog Timer Handler
0x007               rjmp   TIM2_COMPA   ; Timer2 Compare A Handler
0x008               rjmp   TIM2_COMPB   ; Timer2 Compare B Handler
0x009               rjmp   TIM2_OVF     ; Timer2 Overflow Handler
0x00A               rjmp   TIM1_CAPT    ; Timer1 Capture Handler
0x00B               rjmp   TIM1_COMPA   ; Timer1 Compare A Handler
0x00C               rjmp   TIM1_COMPB   ; Timer1 Compare B Handler
0x00D               rjmp   TIM1_OVF     ; Timer1 Overflow Handler
0x00E               rjmp   TIM0_COMPA   ; Timer0 Compare A Handler
0x00F               rjmp   TIM0_COMPB   ; Timer0 Compare B Handler
0x010               rjmp   TIM0_OVF     ; Timer0 Overflow Handler
0x011               rjmp   SPI_STC      ; SPI Transfer Complete Handler
0x012               rjmp   USART_RXC    ; USART, RX Complete Handler
0x013               rjmp   USART_UDRE   ; USART, UDR Empty Handler
0x014               rjmp   USART_TXC    ; USART, TX Complete Handler
0x015               rjmp   ADC          ; ADC Conversion Complete Handler
0x016               rjmp   EE_RDY       ; EEPROM Ready Handler
0x017               rjmp   ANA_COMP     ; Analog Comparator Handler
0x018               rjmp   TWI          ; 2-wire Serial Interface Handler
0x019               rjmp   SPM_RDY      ; Store Program Memory Ready Handler
;
0x01A  RESET:       ldi    r16, high(RAMEND); Main program start
0x01B               out    SPH,r16      ; Set Stack Pointer to top of RAM
0x01C               ldi    r16, low(RAMEND)
0x01D               out    SPL,r16
0x01E               sei                 ; Enable interrupts
0x01F               <instr>  xxx
  ...   ...    ...    ...
```

9.2 Interrupt Vectors in ATmega88

Table 9-2. Reset and Interrupt Vectors in ATmega88

Vector No.	Program Address(2)	Source	Interrupt Definition
1	0x000(1)	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x001	INT0	External interrupt request 0
3	0x002	INT1	External interrupt request 1
4	0x003	PCINT0	Pin change interrupt request 0
5	0x004	PCINT1	Pin change interrupt request 1
6	0x005	PCINT2	Pin change interrupt request 2
7	0x006	WDT	Watchdog time-out interrupt

Notes:
1. When the BOOTRST fuse is programmed, the device will jump to the boot loader address at reset, see Section 24. “Boot Loader Support – Read-While-Write Self-Programming, ATmega88 and ATmega168” on page 229.
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.

12. 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:

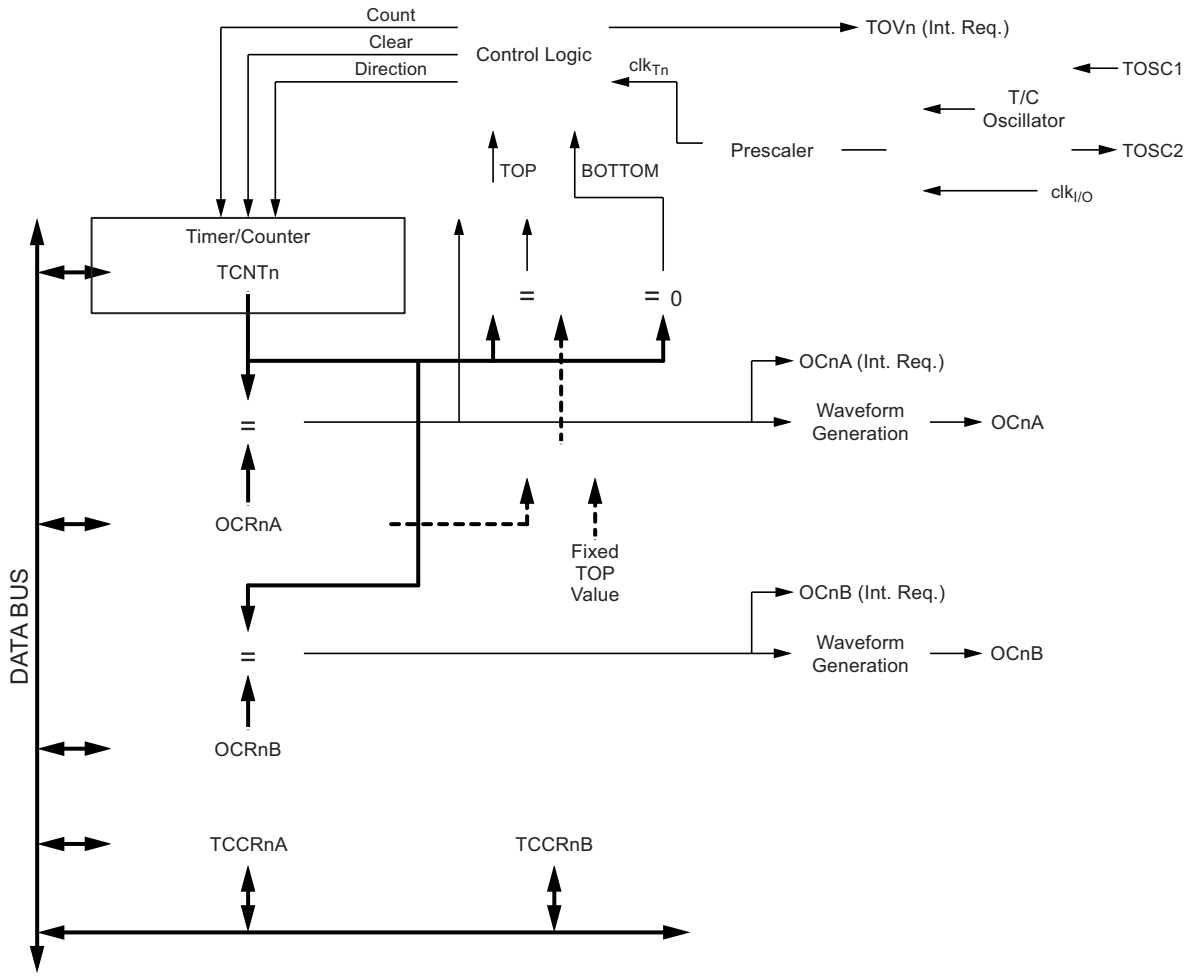
- Two independent output compare units
- Double buffered output compare registers
- Clear timer on compare match (auto reload)
- Glitch free, phase correct pulse width modulator (PWM)
- Variable PWM period
- Frequency generator
- Three independent interrupt sources (TOV0, OCF0A, and OCF0B)

12.1 Overview

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 12-1. For the actual placement of I/O pins, refer to Section 1-1 “Pinout ATmega48/88/168” on page 3. CPU accessible I/O registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O register and bit locations are listed in Section 12.8 “8-bit Timer/Counter Register Description” on page 87.

The PRTIM0 bit in Section 7.7.1 “Power Reduction Register - PRR” on page 35 must be written to zero to enable Timer/Counter0 module.

Figure 12-1. 8-bit Timer/Counter 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.

12.4.1 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).

12.4.2 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.

12.4.3 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 compare (FOC0x) strobe bits in normal mode. The OC0x registers keep their values even when changing between waveform generation modes.

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

12.5 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 12-4 on page 81 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 12-9 shows the same timing data, but with the prescaler enabled.

Figure 12-9. Timer/Counter Timing Diagram, with Prescaler ($f_{clk\_I/O}/8$)

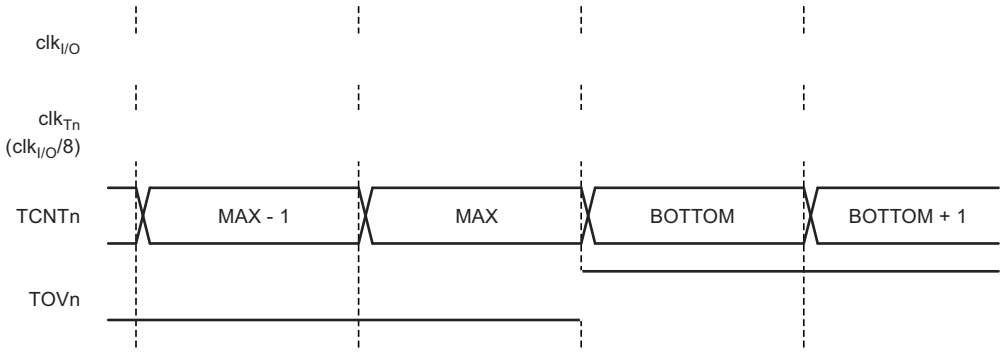


Figure 12-10 shows the setting of OCF0B in all modes and OCF0A in all modes except CTC mode and PWM mode, where OCR0A is TOP.

Figure 12-10.Timer/Counter Timing Diagram, Setting of OCF0x, with Prescaler ($f_{clk\_I/O}/8$)

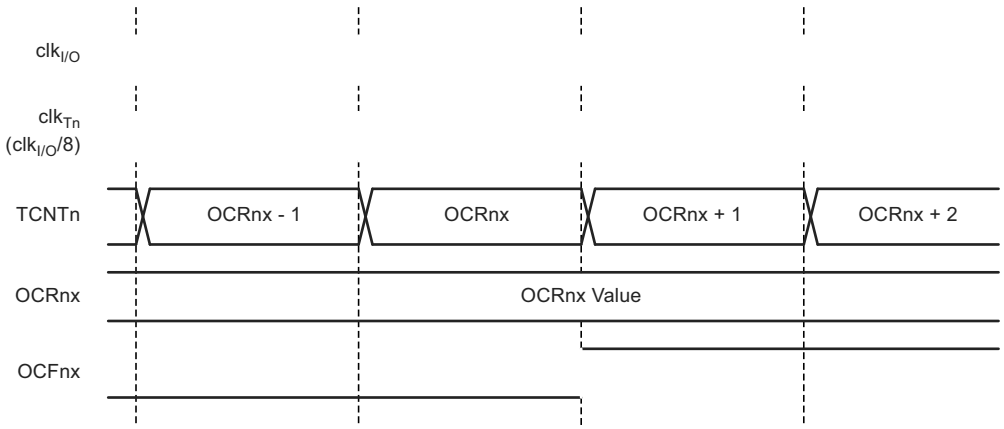


Figure 12-11 shows the setting of OCF0A and the clearing of TCNT0 in CTC mode and fast PWM mode where OCR0A is TOP.

Figure 12-11.Timer/Counter Timing Diagram, Clear Timer on Compare Match mode, with Prescaler ($f_{clk\_I/O}/8$)

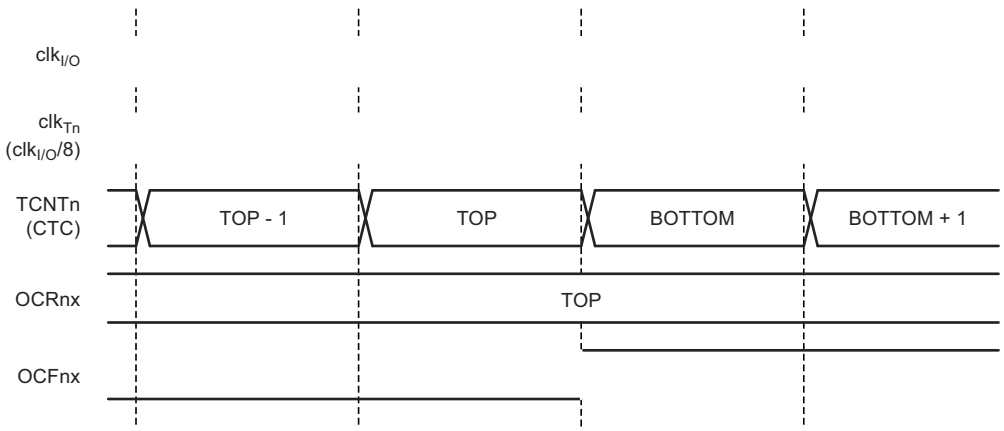
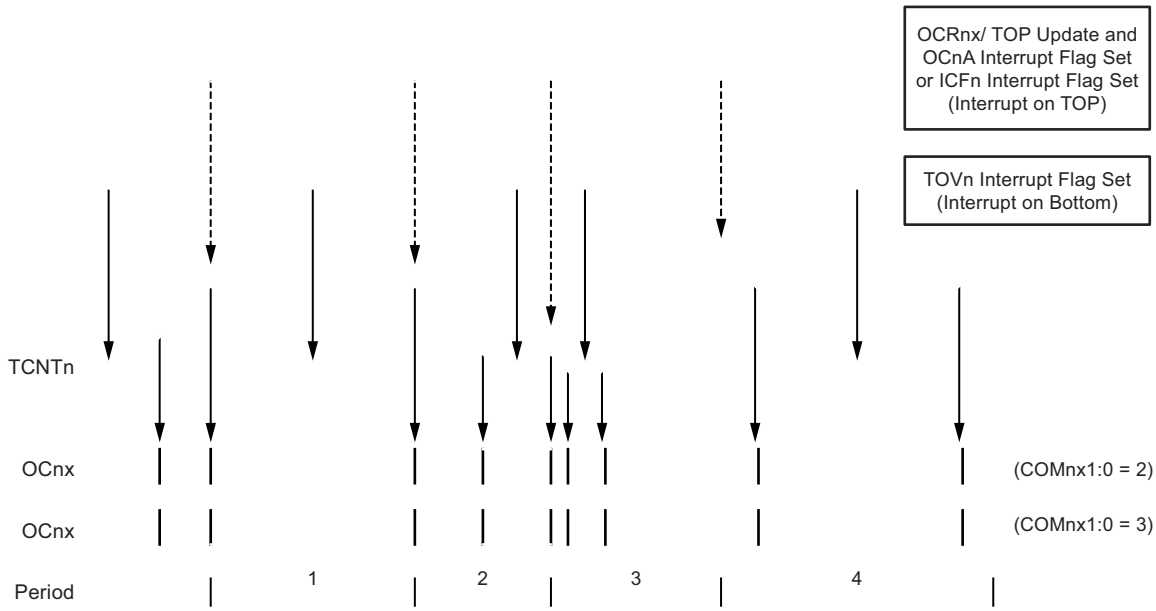


Figure 14-8. Phase Correct PWM Mode, Timing Diagram



The Timer/Counter overflow flag (TOV1) is set each time the counter reaches BOTTOM. When either OCR1A or ICR1 is used for defining the TOP value, the OC1A or ICF1 flag is set accordingly at the same timer clock cycle as the OCR1x registers are updated with the double buffer value (at TOP). The interrupt flags can be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the compare registers. If the TOP value is lower than any of the compare registers, a compare match will never occur between the TCNT1 and the OCR1x. Note that when using fixed TOP values, the unused bits are masked to zero when any of the OCR1x registers are written. As the third period shown in Figure 14-8 illustrates, changing the TOP actively while the Timer/Counter is running in the phase correct mode can result in an unsymmetrical output. The reason for this can be found in the time of update of the OCR1x register. Since the OCR1x update occurs at TOP, the PWM period starts and ends at TOP. This implies that the length of the falling slope is determined by the previous TOP value, while the length of the rising slope is determined by the new TOP value. When these two values differ the two slopes of the period will differ in length. The difference in length gives the unsymmetrical result on the output.

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

In phase correct PWM mode, the compare units allow generation of PWM waveforms on the OC1x pins. Setting the COM1x1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COM1x1:0 to three (See Table on page 114). The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OC1x). The PWM waveform is generated by setting (or clearing) the OC1x register at the compare match between OCR1x and TCNT1 when the counter increments, and clearing (or setting) the OC1x register at compare match between OCR1x and TCNT1 when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

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

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

The extreme values for the OCR1x register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCR1x is set equal to BOTTOM the output will be continuously low and if set equal to TOP the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values. If OCR1A is used to define the TOP value (WGM13:0 = 11) and COM1A1:0 = 1, the OC1A output will toggle with a 50% duty cycle.

The following code examples show how to initialize the SPI as a Slave and how to perform a simple reception.

Assembly Code Example[1]

```
SPI_SlaveInit:
       ; Set MISO output, all others input
       ldi    r17,(1<<DD_MISO)
       out    DDR_SPI,r17
       ; Enable SPI
       ldi    r17,(1<<SPE)
       out    SPCR,r17
       ret

SPI_SlaveReceive:
       ; Wait for reception complete
       sbis   SPSR,SPIF
       rjmp   SPI_SlaveReceive
       ; Read received data and return
       in     r16,SPDR
       ret
```

C Code Example[1]

```
void SPI_SlaveInit(void)
{
       /* Set MISO output, all others input */
       DDR_SPI = (1<<DD_MISO);
       /* Enable SPI */
       SPCR = (1<<SPE);
}

char SPI_SlaveReceive(void)
{
       /* Wait for reception complete */
       while(!(SPSR & (1<<SPIF)))
              ;
       /* Return Data Register */
       return SPDR;
}
```

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

16.1 $\overline{SS}$ Pin Functionality

16.1.1 Slave Mode

When the SPI is configured as a slave, the slave select ($\overline{SS}$) pin is always input. When $\overline{SS}$ is held low, the SPI is activated, and MISO becomes an output if configured so by the user. All other pins are inputs. When $\overline{SS}$ is driven high, all pins are inputs, and the SPI is passive, which means that it will not receive incoming data. Note that the SPI logic will be reset once the $\overline{SS}$ pin is driven high.

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

More advanced initialization routines can be made that include frame format as parameters, disable interrupts and so on. However, many applications use a fixed setting of the baud and control registers, and for these types of applications the initialization code can be placed directly in the main routine, or be combined with initialization code for other I/O modules.

17.5 Data Transmission – The USART Transmitter

The USART transmitter is enabled by setting the transmit enable (TXEN) bit in the UCSRnB register. When the transmitter is enabled, the normal port operation of the TxDn pin is overridden by the USART and given the function as the transmitter's serial output. The baud rate, mode of operation and frame format must be set up once before doing any transmissions. If synchronous operation is used, the clock on the XCKn pin will be overridden and used as transmission clock.

17.5.1 Sending Frames with 5 to 8 Data Bit

A data transmission is initiated by loading the transmit buffer with the data to be transmitted. The CPU can load the transmit buffer by writing to the UDRn I/O location. The buffered data in the transmit buffer will be moved to the shift register when the shift register is ready to send a new frame. The shift register is loaded with new data if it is in idle state (no ongoing transmission) or immediately after the last stop bit of the previous frame is transmitted. When the shift register is loaded with new data, it will transfer one complete frame at the rate given by the baud register, U2Xn bit or by XCKn depending on mode of operation.

The following code examples show a simple USART transmit function based on polling of the data register empty (UDREn) flag. When using frames with less than eight bits, the most significant bits written to the UDRn are ignored. The USART has to be initialized before the function can be used. For the assembly code, the data to be sent is assumed to be stored in register R16.

Assembly Code Example[1]

```
USART_Transmit:
       ; Wait for empty transmit buffer
       sbis   UCSRnA,UDREn
       rjmp   USART_Transmit
       ; Put data (r16) into buffer, sends the data
       out    UDRn,r16
       ret
```

C Code Example[1]

```
void USART_Transmit( unsigned char data )
{
       /* Wait for empty transmit buffer */
       while ( !( UCSRnA & (1<<UDREn)) )
                              ;
       /* Put data into buffer, sends the data */
       UDRn = data;
}
```

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

The function simply waits for the transmit buffer to be empty by checking the UDREn flag, before loading it with new data to be transmitted. If the data register empty interrupt is utilized, the interrupt routine writes the data into the buffer.

17.5.2 Sending Frames with 9 Data Bit

If 9-bit characters are used (UCSZn = 7), the ninth bit must be written to the TXB8 bit in UCSRnB before the low byte of the character is written to UDRn. The following code examples show a transmit function that handles 9-bit characters. For the assembly code, the data to be sent is assumed to be stored in registers R17:R16.

Assembly Code Example[1][2]

```
USART_Transmit:
       ; Wait for empty transmit buffer
       sbis   UCSRnA,UDREn
       rjmp   USART_Transmit
       ; Copy 9th bit from r17 to TXB8
       cbi    UCSRnB,TXB8
       sbrc   r17,0
       sbi    UCSRnB,TXB8
       ; Put LSB data (r16) into buffer, sends the data
       out    UDRn,r16
       ret
```

C Code Example[1][2

```
void USART_Transmit( unsigned int data )
{
       /* Wait for empty transmit buffer */
       while ( !( UCSRnA & (1<<UDREn))) )
                        ;
       /* Copy 9th bit to TXB8 */
       UCSRnB &= ~(1<<TXB8);
       if ( data & 0x0100 )
              UCSRnB |= (1<<TXB8);
       /* Put data into buffer, sends the data */
       UDRn = data;
}
```

Notes: 1. These transmit functions are written to be general functions. They can be optimized if the contents of the UCSRnB is static. For example, only the TXB8 bit of the UCSRnB register is used after initialization.

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

The ninth bit can be used for indicating an address frame when using multi processor communication mode or for other protocol handling as for example synchronization.

17.5.3 Transmitter Flags and Interrupts

The USART transmitter has two flags that indicate its state: USART data register empty (UDREn) and transmit complete (TXCn). Both flags can be used for generating interrupts.

The data register empty (UDREn) flag indicates whether the transmit buffer is ready to receive new data. This bit is set when the transmit buffer is empty, and cleared when the transmit buffer contains data to be transmitted that has not yet been moved into the shift register. For compatibility with future devices, always write this bit to zero when writing the UCSRnA register.

When the data register empty interrupt enable (UDRIEn) bit in UCSRnB is written to one, the USART data register empty interrupt will be executed as long as UDREn is set (provided that global interrupts are enabled). UDREn is cleared by writing UDRn. When interrupt-driven data transmission is used, the data register empty interrupt routine must either write new data to UDRn in order to clear UDREn or disable the data register empty interrupt, otherwise a new interrupt will occur once the interrupt routine terminates.

The transmit complete (TXCn) flag bit is set one when the entire frame in the transmit shift register has been shifted out and there are no new data currently present in the transmit buffer. The TXCn flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location.

Table 17-10. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies (Continued)

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

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

Table 17-11. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies (Continued)

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

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

Table 17-12. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies (Continued)

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

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

7. The application software should now examine the value of TWSR, to make sure that the data packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must write a specific value to TWCR, instructing the TWI hardware to transmit a STOP condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the STOP condition. Note that TWINT is NOT set after a STOP condition has been sent.

Even though this example is simple, it shows the principles involved in all TWI transmissions. These can be summarized as follows:

- When the TWI has finished an operation and expects application response, the TWINT flag is set. The SCL line is pulled low until TWINT is cleared.
- When the TWINT flag is set, the user must update all TWI registers with the value relevant for the next TWI bus cycle. As an example, TWDR must be loaded with the value to be transmitted in the next bus cycle.
- After all TWI register updates and other pending application software tasks have been completed, TWCR is written. When writing TWCR, the TWINT bit should be set. Writing a one to TWINT clears the flag. The TWI will then commence executing whatever operation was specified by the TWCR setting.

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

Table 19-3. Code Example

No.	Assembly Code Example	C Example	Comments
1	**ldi** r16, (1<<TWINT)|(1<<TWSTA)|(1<<TWEN) **out** TWCR, r16	TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)	Send START condition
2	wait1: **in** r16,TWCR **sbrs** r16,TWINT **rjmp** wait1	**while** (!(TWCR & (1<<TWINT)));	Wait for TWINT flag set. This indicates that the START condition has been transmitted
3	**in** r16,TWSR **andi** r16, 0xF8 **cpi** r16, START **brne** ERROR	**if** ((TWSR & 0xF8) != START ERROR();	Check value of TWI status register. Mask prescaler bits. If status different from START go to ERROR
	ldi r16, SLA_W **out** TWDR, r16 **ldi** r16, (1<<TWINT) | (1<<TWEN) **out** TWCR, r16	TWDR = SLA_W; TWCR = (1<<TWINT) | (1<<TWEN);	Load SLA_W into TWDR register. Clear TWINT bit in TWCR to start transmission of address
4	wait2: **in** r16,TWCR **sbrs** r16,TWINT **rjmp** wait2	**while** (!(TWCR & (1<<TWINT))) ;	Wait for TWINT flag set. This indicates that the SLA+W has been transmitted, and ACK/NACK has been received.
5	**in** r16,TWSR **andi** r16, 0xF8 **cpi** r16, MT_SLA_ACK **brne** ERROR	**if** ((TWSR & 0xF8) != MT_SLA_ACK) ERROR();	Check value of TWI status register. Mask prescaler bits. If status different from MT_SLA_ACK go to ERROR
	ldi r16, DATA **out** TWDR, r16 **ldi** r16, (1<<TWINT) | (1<<TWEN) **out** TWCR, r16	TWDR = DATA; TWCR = (1<<TWINT) | (1<<TWEN);	Load DATA into TWDR register. Clear TWINT bit in TWCR to start transmission of data

Figure 21-6. ADC Timing Diagram, Auto Triggered Conversion

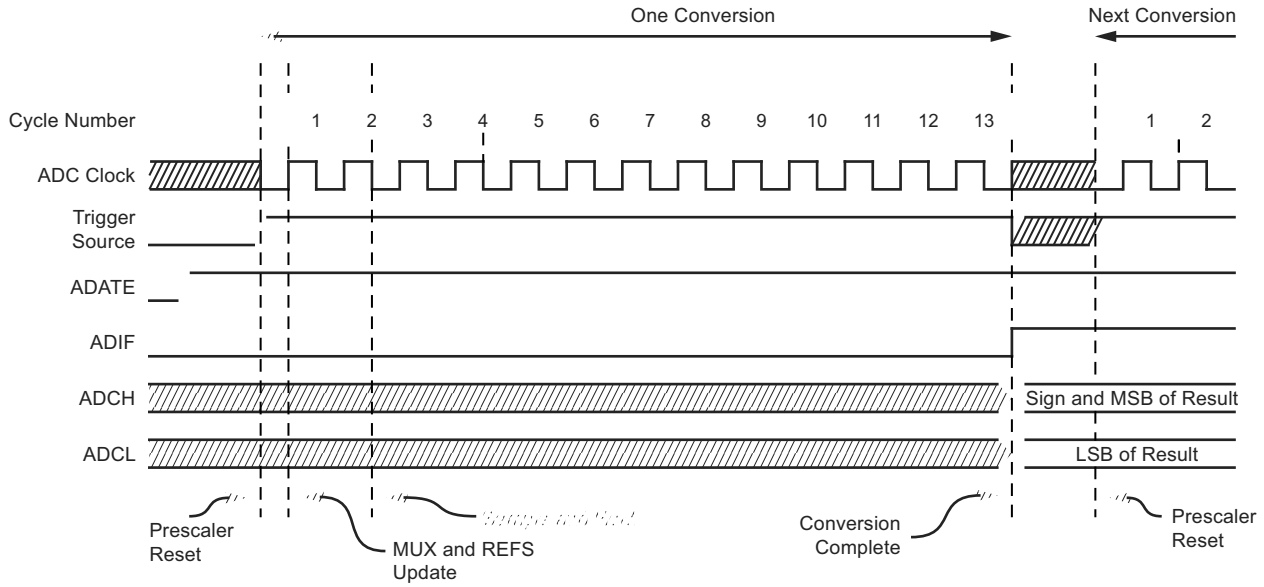


Figure 21-7. ADC Timing Diagram, Free Running Conversion

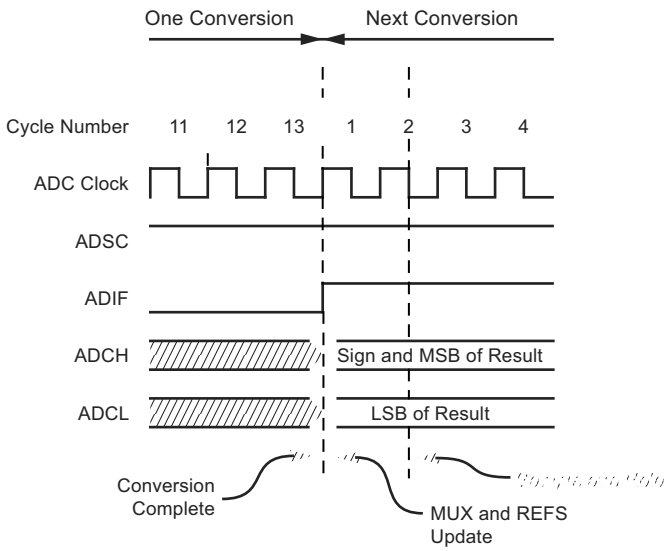


Table 21-1. ADC Conversion Time

Condition	Sample & Hold (Cycles from Start of Conversion)	Conversion Time (Cycles)
First conversion	13.5	25
Normal conversions, single ended	1.5	13
Auto triggered conversions	2	13.5

23.4.1 Store Program Memory Control and Status Register – SPMCSR

The store program memory control and status register contains the control bits needed to control the program memory operations.

Bit	7	6	5	4	3	2	1	0	
	SPMIE	**RWWSB**	–	RWWSRE	**BLBSET**	**PGWRT**	**PGERS**	**SELFPRGEN**	**SPMCSR**
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 – 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 SELFPRGEN bit in the SPMCSR register is cleared.

- **Bit 6 – RWWSB: Read-While-Write Section Busy**

This bit is for compatibility with devices supporting read-while-write. It will always read as zero in Atmel® ATmega48.

- **Bit 5 – Res: Reserved Bit**

This bit is a reserved bit in the Atmel ATmega48/88/168 and will always read as zero.

- **Bit 4 – RWWSRE: Read-While-Write Section Read Enable**

The functionality of this bit in ATmega48 is a subset of the functionality in ATmega88/168. If the RWWSRE bit is written while filling the temporary page buffer, the temporary page buffer will be cleared and the data will be lost.

- **Bit 3 – BLBSET: Boot Lock Bit Set**

The functionality of this bit in ATmega48 is a subset of the functionality in ATmega88/168. An LPM instruction within three cycles after BLBSET and SELFPRGEN are set in the SPMCSR register, will read either the lock bits or the fuse bits (depending on Z0 in the Z-pointer) into the destination register.
See Section 23.4.3 “Reading the Fuse and Lock Bits from Software” on page 226 for details.

- **Bit 2 – PGWRT: Page Write**

If this bit is written to one at the same time as SELFPRGEN, 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.

- **Bit 1 – PGERS: Page Erase**

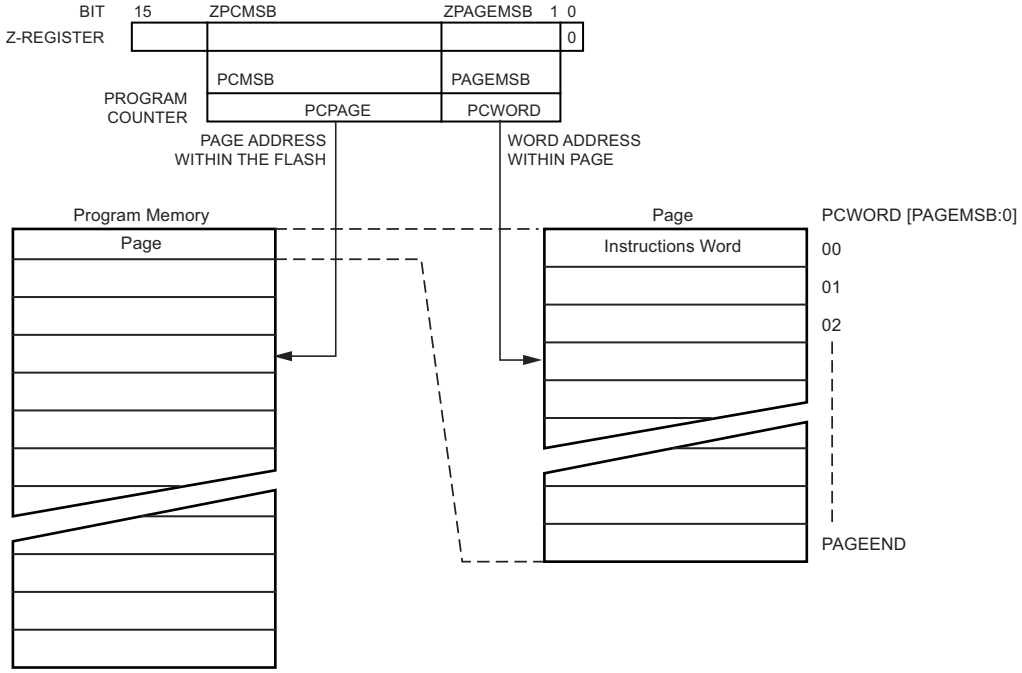
If this bit is written to one at the same time as SELFPRGEN, the next SPM instruction within four clock cycles executes page erase. The page address is taken from the high part of the Z-pointer. The data in R1 and R0 are ignored. The PGERS bit will auto-clear upon completion of a page erase, or if no SPM instruction is executed within four clock cycles. The CPU is halted during the entire page write operation.

- **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.

Figure 24-3. Addressing the Flash During SPM(1)



Note: 1. The different variables used in Figure 24-3 are listed in Table 24-8 on page 240.

24.7 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 Section 24.7.12 “Simple Assembly Code Example for a Boot Loader” on page 238 for an assembly code example.

24.7.1 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.

Table 25-3. Lock Bit Protection Modes[1][2]. Only ATmega88/168.

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.
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

25.2 Fuse Bits

The Atmel® ATmega48/88/168 has three fuse bytes. Table 25-4 - Table 25-7 on page 244 describe briefly the functionality of all the fuses and how they are mapped into the fuse bytes. Note that the fuses are read as logical zero, “0”, if they are programmed.

Table 25-4. Extended Fuse Byte for ATmega48

Extended Fuse Byte	Bit No	Description	Default Value
–	7	–	1
–	6	–	1
–	5	–	1
–	4	–	1
–	3	–	1
–	2	–	1
–	1	–	1
SELFPRGEN	0	Self programming enable	1 (unprogrammed)

25.2.1 Latching of Fuses

The fuse values are latched when the device enters programming mode and changes of the fuse values will have no effect until the part leaves programming mode. This does not apply to the EESAVE fuse which will take effect once it is programmed. The fuses are also latched on power-up in normal mode.

25.3 Signature Bytes

All Atmel® microcontrollers have a three-byte signature code which identifies the device. This code can be read in both serial and parallel mode, also when the device is locked. The three bytes reside in a separate address space.

25.3.1 ATmega48 Signature Bytes

1. 0x000: 0x1E (indicates manufactured by Atmel).
2. 0x001: 0x92 (indicates 4KB flash memory).
3. 0x002: 0x05 (indicates Atmel ATmega48 device when 0x001 is 0x92).

25.3.2 ATmega88 Signature Bytes

1. 0x000: 0x1E (indicates manufactured by Atmel).
2. 0x001: 0x93 (indicates 8KB flash memory).
3. 0x002: 0x0A (indicates ATmega88 device when 0x001 is 0x93).

25.3.3 ATmega168 Signature Bytes

1. 0x000: 0x1E (indicates manufactured by Atmel).
2. 0x001: 0x94 (indicates 16KB flash memory).
3. 0x002: 0x06 (indicates Atmel ATmega168 device when 0x001 is 0x94).

25.4 Calibration Byte

The Atmel ATmega48/88/168 has a byte calibration value for the internal RC oscillator. This byte resides in the high byte of address 0x000 in the signature address space. During reset, this byte is automatically written into the OSCCAL register to ensure correct frequency of the calibrated RC oscillator.

25.5 Parallel Programming Parameters, Pin Mapping, and Commands

This section describes how to parallel program and verify flash program memory, EEPROM data memory, memory lock bits, and fuse bits in the Atmel ATmega48/88/168. Pulses are assumed to be at least 250ns unless otherwise noted.

25.5.1 Signal Names

In this section, some pins of the ATmega48/88/168 are referenced by signal names describing their functionality during parallel programming, see Figure 25-1 on page 246 and Table 25-8 on page 246. Pins not described in the following table are referenced by pin names.

The XA1/XA0 pins determine the action executed when the XTAL1 pin is given a positive pulse. The bit coding is shown in Table 25-10 on page 246.

When pulsing $\overline{WR}$ or $\overline{OE}$, the command loaded determines the action executed. The different Commands are shown in Table 25-11 on page 247.