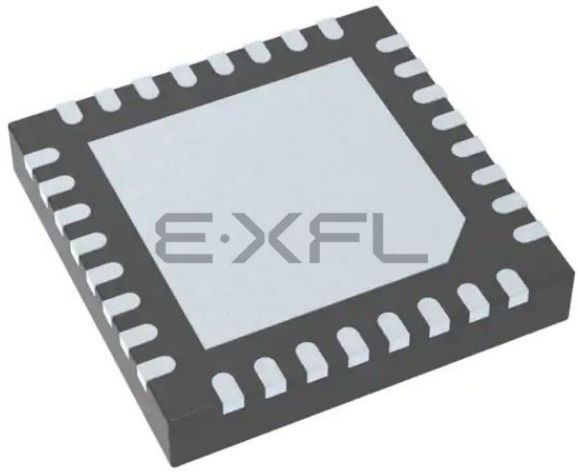


Welcome to E-XFL.COM

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	Active
Core Processor	AVR
Core Size	8-Bit
Speed	16MHz
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 ~ 125°C (TA)
Mounting Type	Surface Mount
Package / Case	32-VFQFN Exposed Pad
Supplier Device Package	32-QFN (5x5)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atmega88pa-15mz

3. Automotive Quality Grade

The Atmel® ATmega48PA/88PA/168PA have been developed and manufactured according to the most stringent requirements of the international standard ISO-TS-16949. This data sheet contains limit values extracted from the results of extensive characterization (temperature and voltage).

The quality and reliability of the Atmel ATmega48PA/88PA/168PA have been verified during regular product qualification as per AEC-Q100 grade 1 (–40°C to +125°C).

Table 3-1. Temperature Grade Identification for Automotive Products

Temperature (°C)	Temperature Identifier	Comments
–40; +125	Z	Full automotive temperature range

4. Resources

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

Note: 1.

5. Data Retention

Reliability qualification results show that the projected data retention failure rate is much less than 1 PPM over 20 years at 85°C.

6. About Code Examples

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

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

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

Assembly Code Example

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

C Code Example

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

8.6.4 GPIOR2 – General Purpose I/O Register 2

Bit	7	6	5	4	3	2	1	0	
0x2B (0x4B)	MSB							LSB	GPIOR2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

8.6.5 GPIOR1 – General Purpose I/O Register 1

Bit	7	6	5	4	3	2	1	0	
0x2A (0x4A)	MSB							LSB	GPIOR1
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

8.6.6 GPIOR0 – General Purpose I/O Register 0

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

12. Interrupts

This section describes the specifics of the interrupt handling as performed in the Atmel® ATmega48PA/88PA/168PA. For a general explanation of the AVR® interrupt handling, refer to Section 7.7 “Reset and Interrupt Handling” on page 14.

The interrupt vectors in the Atmel ATmega48PA, Atmel ATmega88PA, and ATmega168PA are generally the same, with the following differences:

- Each interrupt vector occupies two instruction words in Atmel ATmega168PA and one instruction word in the Atmel ATmega48PA and Atmel ATmega88PA.
- Atmel ATmega48PA does not have a separate boot loader section. In the Atmel ATmega88PA, and Atmel ATmega168PA, the reset vector is affected by the BOOTRST fuse, and the interrupt vector start address is affected by the IVSEL bit in MCUCR.

12.1 Interrupt Vectors in Atmel ATmega48PA

Table 12-1. Reset and Interrupt Vectors in ATmega48PA

Vector No.	Program Address	Source	Interrupt Definition
1	0x000	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
8	0x007	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x008	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x009	TIMER2 OVF	Timer/Counter2 overflow
11	0x00A	TIMER1 CAPT	Timer/Counter1 capture event
12	0x00B	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x00C	TIMER1 COMPB	Timer/coutner1 compare match B
14	0x00D	TIMER1 OVF	Timer/Counter1 overflow
15	0x00E	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x00F	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x010	TIMER0 OVF	Timer/Counter0 overflow
18	0x011	SPI, STC	SPI serial transfer complete
19	0x012	USART, RX	USART Rx complete
20	0x013	USART, UDRE	USART, data register empty
21	0x014	USART, TX	USART, Tx complete
22	0x015	ADC	ADC conversion complete
23	0x016	EE READY	EEPROM ready
24	0x017	ANALOG COMP	Analog comparator
25	0x018	TWI	2-wire serial interface
26	0x019	SPM READY	Store program memory ready

14.2.2 Toggling the Pin

Writing a logic one to PINxn toggles the value of PORTxn, independent on the value of DDRxn. Note that the SBI instruction can be used to toggle one single bit in a port.

14.2.3 Switching Between Input and Output

When switching between tri-state ({DDxn, PORTxn} = 0b00) and output high ({DDxn, PORTxn} = 0b11), an intermediate state with either pull-up enabled {DDxn, PORTxn} = 0b01) or output low ({DDxn, PORTxn} = 0b10) must occur. Normally, the pull-up enabled state is fully acceptable, as a high-impedance environment will not notice the difference between a strong high driver and a pull-up. If this is not the case, the PUD bit in the MCUCR register can be set to disable all pull-ups in all ports.

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

Table 14-1 summarizes the control signals for the pin value.

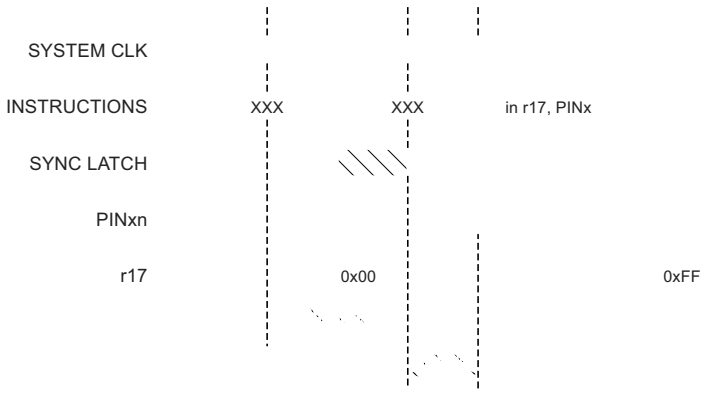
Table 14-1. Port Pin Configurations

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output low (sink)
1	1	X	Output	No	Output high (source)

14.2.4 Reading the Pin Value

Independent of the setting of data direction bit DDxn, the port pin can be read through the PINxn register bit. As shown in Figure 14-2 on page 65, the PINxn register bit and the preceding latch constitute a synchronizer. This is needed to avoid metastability if the physical pin changes value near the edge of the internal clock, but it also introduces a delay. Figure 14-3 shows a timing diagram of the synchronization when reading an externally applied pin value. The maximum and minimum propagation delays are denoted $t_{pd,max}$ and $t_{pd,min}$ respectively.

Figure 14-3. Synchronization when Reading an Externally Applied Pin value



Consider the clock period starting shortly after the first falling edge of the system clock. The latch is closed when the clock is low, and goes transparent when the clock is high, as indicated by the shaded region of the “SYNC LATCH” signal. The signal value is latched when the system clock goes low. It is clocked into the PINxn register at the succeeding positive clock edge. As indicated by the two arrows tpd,max and tpd,min, a single signal transition on the pin will be delayed between ½ and 1½ system clock period depending upon the time of assertion.

Table 14-8. Overriding Signals for Alternate Functions in PC3...PC0

Signal Name	PC3/ADC3/ PCINT11	PC2/ADC2/ PCINT10	PC1/ADC1/ PCINT9	PC0/ADC0/ PCINT8
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
DIEOE	PCINT11 × PCIE1 + ADC3D	PCINT10 × PCIE1 + ADC2D	PCINT9 × PCIE1 + ADC1D	PCINT8 × PCIE1 + ADC0D
DIEOV	PCINT11 × PCIE1	PCINT10 × PCIE1	PCINT9 × PCIE1	PCINT8 × PCIE1
DI	PCINT11 INPUT	PCINT10 INPUT	PCINT9 INPUT	PCINT8 INPUT
AIO	ADC3 INPUT	ADC2 INPUT	ADC1 INPUT	ADC0 INPUT

14.3.3 Alternate Functions of Port D

The port D pins with alternate functions are shown in Table 14-9.

Table 14-9. Port D Pins Alternate Functions

Port Pin	Alternate Function
PD7	AIN1 (analog comparator negative input) PCINT23 (pin change interrupt 23)
PD6	AIN0 (analog comparator positive input) OC0A (Timer/Counter0 output compare match A output) PCINT22 (pin change interrupt 22)
PD5	T1 (Timer/Counter 1 external counter input) OC0B (Timer/Counter0 output compare match B output) PCINT21 (pin change interrupt 21)
PD4	XCK (USART external clock input/output) T0 (Timer/Counter 0 external counter input) PCINT20 (pin change interrupt 20)
PD3	INT1 (external interrupt 1 input) OC2B (Timer/Counter2 output compare match B output) PCINT19 (pin change interrupt 19)
PD2	INT0 (external interrupt 0 input) PCINT18 (pin change interrupt 18)
PD1	TXD (USART output pin) PCINT17 (pin change interrupt 17)
PD0	RXD (USART input pin) PCINT16 (pin change interrupt 16)

The alternate pin configuration is as follows:

- **AIN1/OC2B/PCINT23 – Port D, Bit 7**

AIN1, analog comparator negative input. Configure the port pin as input with the internal pull-up switched off to avoid the digital port function from interfering with the function of the analog comparator.

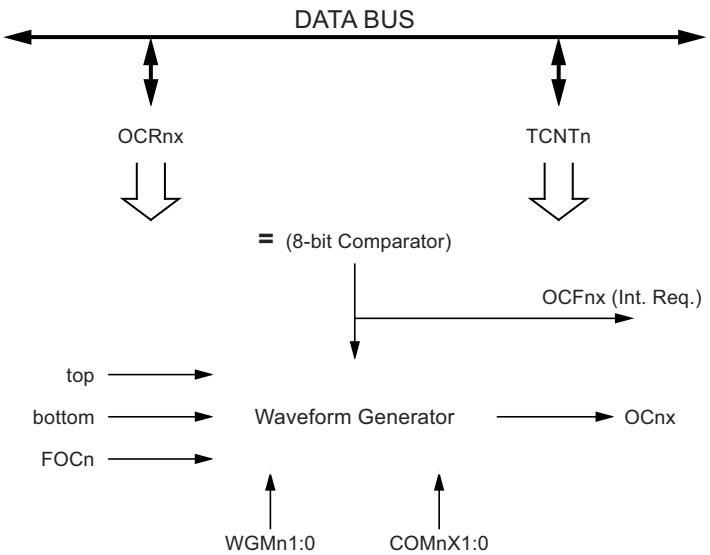
PCINT23: Pin change interrupt source 23. The PD7 pin can serve as an external interrupt source.

15.5 Output Compare Unit

The 8-bit comparator continuously compares TCNT0 with the output compare registers (OCR0A and OCR0B). Whenever TCNT0 equals OCR0A or OCR0B, the comparator signals a match. A match will set the output compare flag (OCF0A or OCF0B) at the next timer clock cycle. If the corresponding interrupt is enabled, the output compare flag generates an output compare interrupt. The output compare flag is automatically cleared when the interrupt is executed. Alternatively, the flag can be cleared by software by writing a logical one to its I/O bit location. The waveform generator uses the match signal to generate an output according to operating mode set by the WGM02:0 bits and compare output mode (COM0x1:0) bits. The max and bottom signals are used by the waveform generator for handling the special cases of the extreme values in some modes of operation (Section 15.7 “Modes of Operation” on page 86).

Figure 15-3 shows a block diagram of the output compare unit.

Figure 15-3. Output Compare Unit, Block Diagram



The OCR0x registers are double buffered when using any of the pulse width modulation (PWM) modes. For the normal and clear timer on compare (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR0x compare registers to either top or bottom of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCR0x register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR0x buffer register, and if double buffering is disabled the CPU will access the OCR0x directly.

15.5.1 Force Output Compare

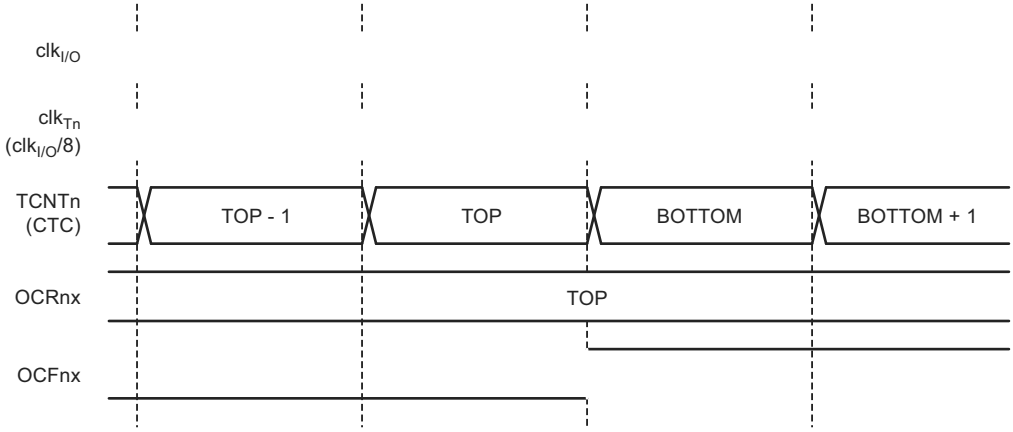
In non-PWM waveform generation modes, the match output of the comparator can be forced by writing a one to the force output compare (FOC0x) bit. Forcing compare match will not set the OCF0x flag or reload/clear the timer, but the OC0x pin will be updated as if a real compare match had occurred (the COM0x1:0 bits settings define whether the OC0x pin is set, cleared or toggled).

15.5.2 Compare Match Blocking by TCNT0 Write

All CPU write operations to the TCNT0 register will block any compare match that occur in the next timer clock cycle, even when the timer is stopped. This feature allows OCR0x to be initialized to the same value as TCNT0 without triggering an interrupt when the Timer/Counter clock is enabled.

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

Figure 15-11.Timer/Counter Timing Diagram, Clear Timer on Compare Match mode, with Prescaler ($f_{clk\_I/O}$/8)



15.9 Register Description

15.9.1 TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A**1**	COM0A**0**	COM0B**1**	COM0B**0**	–	–	**WGM01**	**WGM00**	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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

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

When OC0A is connected to the pin, the function of the COM0A1:0 bits depends on the WGM02:0 bit setting. Table 15-2 shows the COM0A1:0 bit functionality when the WGM02:0 bits are set to a normal or CTC mode (non-PWM).

Table 15-2. Compare Output Mode, non-PWM Mode

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC0A on compare match
1	0	Clear OC0A on compare match
1	1	Set OC0A on compare match

15.9.2 TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	**WGM02**	**CS02**	**CS01**	**CS00**	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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

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

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

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

The FOC0A bit is always read as zero.

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

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

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

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

The FOC0B bit is always read as zero.

- **Bits 5:4 – Reserved**

These bits are reserved bits in the Atmel® ATmega48PA/88PA/168PA and will always read as zero.

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

See the description in the Section 15.9.1 “TCCR0A – Timer/Counter Control Register A” on page 91.

- **Bits 2:0 – CS02:0: Clock Select**

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

Table 15-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (timer/ccounter stopped)
0	0	1	$clk_{I/O}$/(no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

If external pin modes are used for the Timer/Counter0, transitions on the T0 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

15.9.3 TCNT0 – Timer/Counter Register

Bit	7	6	5	4	3	2	1	0	
0x26 (0x46)	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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

15.9.4 OCR0A – Output Compare Register A

Bit	7	6	5	4	3	2	1	0	
0x27 (0x47)	OCR0A[7:0]								OCR0A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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

15.9.5 OCR0B – Output Compare Register B

Bit	7	6	5	4	3	2	1	0	
0x28 (0x48)	OCR0B[7:0]								OCR0B
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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

15.9.6 TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:3 – Reserved**

These bits are reserved bits in the Atmel® ATmega48PA/88PA/168PA and will always read as zero.

- **Bit 2 – OCIE0B: Timer/Counter Output Compare Match B Interrupt Enable**

When the OCIE0B bit is written to one, and the I-bit in the status register is set, the Timer/Counter compare match B interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter occurs, i.e., when the OCF0B bit is set in the Timer/Counter interrupt flag register – TIFR0.

- **Bit 1 – OCIE0A: Timer/Counter0 Output Compare Match A Interrupt Enable**

When the OCIE0A bit is written to one, and the I-bit in the status register is set, the Timer/Counter0 compare match A interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter0 occurs, i.e., when the OCF0A bit is set in the Timer/Counter 0 interrupt flag register – TIFR0.

- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

When the TOIE0 bit is written to one, and the I-bit in the status register is set, the Timer/Counter0 overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in the Timer/Counter 0 interrupt flag register – TIFR0.

The general I/O port function is overridden by the output compare (OC1x) from the waveform generator if either of the COM1x1:0 bits are set. However, the OC1x pin direction (input or output) is still controlled by the *data direction register* (DDR) for the port pin. The data direction register bit for the OC1x pin (DDR_OC1x) must be set as output before the OC1x value is visible on the pin. The port override function is generally independent of the waveform generation mode, but there are some exceptions. Refer to Table 16-2 on page 116, Table 16-3 on page 117 and Table 16-4 on page 117 for details.

The design of the output compare pin logic allows initialization of the OC1x state before the output is enabled. Note that some COM1x1:0 bit settings are reserved for certain modes of operation. See
Section 16.11 “Register Description” on page 116

The COM1x1:0 bits have no effect on the input capture unit.

16.8.1 Compare Output Mode and Waveform Generation

The waveform generator uses the COM1x1:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COM1x1:0 = 0 tells the waveform generator that no action on the OC1x register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to Table 16-2 on page 116. For fast PWM mode refer to Table 16-3 on page 117, and for phase correct and phase and frequency correct PWM refer to Table 16-4 on page 117.

A change of the COM1x1:0 bits state will have effect at the first compare match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOC1x strobe bits.

16.9 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the output compare pins, is defined by the combination of the *waveform generation mode* (WGM13:0) and *compare output mode* (COM1x1:0) bits. The compare output mode bits do not affect the counting sequence, while the waveform generation mode bits do. The COM1x1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM1x1:0 bits control whether the output should be set, cleared or toggle at a compare match (See
Section 16.8 “Compare Match Output Unit” on page 107)

For detailed timing information refer to Section 16.10 “Timer/Counter Timing Diagrams” on page 114.

16.9.1 Normal Mode

The simplest mode of operation is the *normal mode* (WGM13:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 16-bit value (MAX = 0xFFFF) and then restarts from the BOTTOM (0x0000). In normal operation the *Timer/Counter overflow flag* (TOV1) will be set in the same timer clock cycle as the TCNT1 becomes zero. The TOV1 flag in this case behaves like a 17th bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV1 flag, the timer resolution can be increased by software. There are no special cases to consider in the normal mode, a new counter value can be written anytime.

The input capture unit is easy to use in normal mode. However, observe that the maximum interval between the external events must not exceed the resolution of the counter. If the interval between events are too long, the timer overflow interrupt or the prescaler must be used to extend the resolution for the capture unit.

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

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
        in r16, SPSR
        sbrs r16, SPIF
        rjmp   SPI_SlaveReceive
        ; Read received data and return
        in     r16,SPDR
        ret
```

C Code Example[1]

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

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

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

20.6.2 Sending Frames with 9 Data Bit

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

Assembly Code Example[1][2]

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

C Code Example[1][2]

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

Notes:
1. These transmit functions are written to be general functions. They can be optimized if the contents of the UCSRnB is static. For example, only the TXB8 bit of the UCSRnB Register is used after initialization.
2. See Section 6. “About Code Examples” on page 7.

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.

Table 20-6. 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 20-7. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 16.0000MHz			
	U2Xn = 0		U2Xn = 1	
	UBRRn	Error	UBRRn	Error
2400	416	–0.1%	832	0.0%
4800	207	0.2%	416	–0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	–0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	–0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	–0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	–3.5%	16	2.1%
230.4k	3	8.5%	8	–3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max.(1)	1Mbps		2Mbps	

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

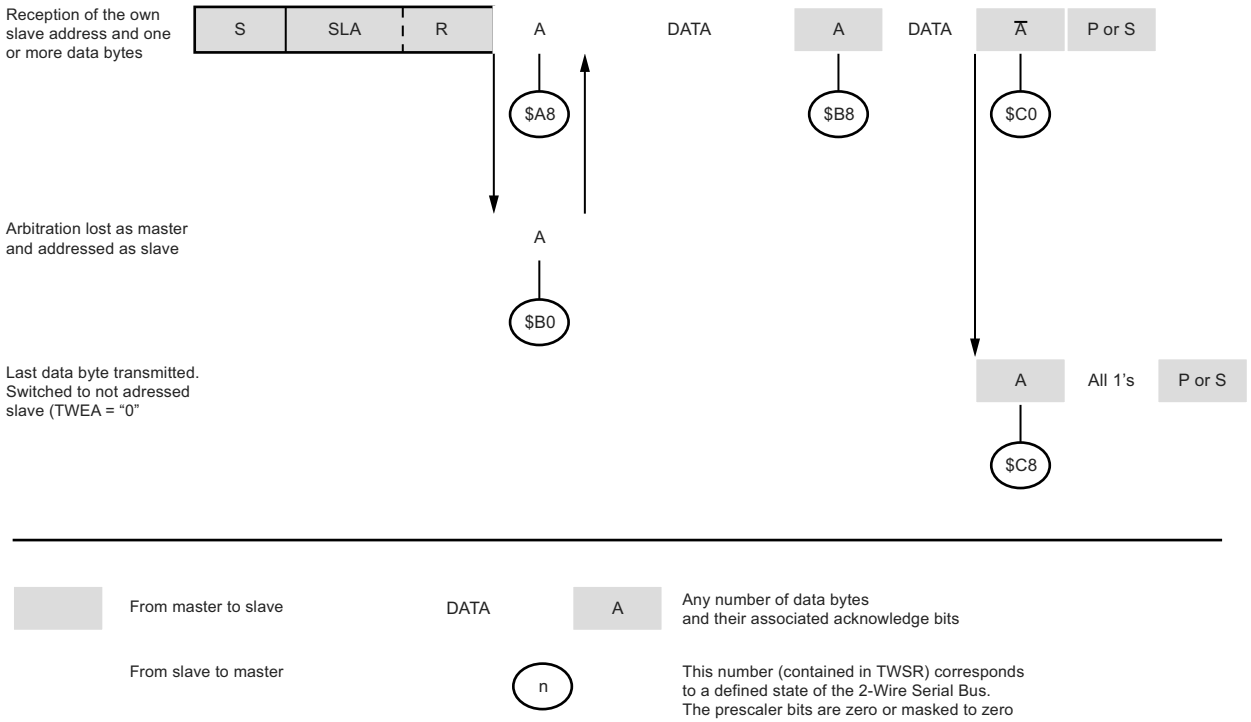
5. The application software should now examine the value of TWSR, to make sure that the address packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. 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.

Figure 22-18. Formats and States in the Slave Transmitter Mode



22.7.5 Miscellaneous States

There are two status codes that do not correspond to a defined TWI state, see Table 22-7.

Status 0xF8 indicates that no relevant information is available because the TWINT flag is not set. This occurs between other states, and when the TWI is not involved in a serial transfer.

Status 0x00 indicates that a bus error has occurred during a 2-wire serial bus transfer. A bus error occurs when a START or STOP condition occurs at an illegal position in the format frame. Examples of such illegal positions are during the serial transfer of an address byte, a data byte, or an acknowledge bit. When a bus error occurs, TWINT is set. To recover from a bus error, the TWSTO flag must set and TWINT must be cleared by writing a logic one to it. This causes the TWI to enter the not addressed Slave mode and to clear the TWSTO Flag (no other bits in TWCR are affected). The SDA and SCL lines are released, and no STOP condition is transmitted.

Table 22-7. Miscellaneous States

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWINT	TWEA	
0xF8	No relevant state information available; TWINT = "0"	No TWDR action	No TWCR action				Wait or proceed current transfer
0x00	Bus error due to an illegal START or STOP condition	No TWDR action	0	1	1	X	Only the internal hardware is affected, no STOP condition is sent on the bus. In all cases, the bus is released and TWSTO is cleared.

26.2.2 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 28-5 on page 253 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 28-5 on page 253 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 shown below. See Table 28-5 on page 253 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**

Fuse and Lock bits that are programmed, will be read as zero. Fuse and lock bits that are unprogrammed, will be read as one.

26.2.3 Preventing Flash Corruption

During periods of low V_{CC}, the flash program can be corrupted because the supply voltage is too low for the CPU and the Flash to operate properly. These issues are the same as for board level systems using the Flash, and the same design solutions should be applied.

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

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

1. 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.
2. Keep the AVR core in power-down sleep mode during periods of low V_{CC}. This will prevent the CPU from attempting to decode and execute instructions, effectively protecting the SPMCSR register and thus the flash from unintentional writes.

```
        ;       re-enable the RWW section
        ldi     spmcrval, (1<<RWWSRE) | (1<<SELFPRGEN)
        rcall   Do_spm

        ;       read back and check, optional
        ldi     looplo, low(PAGESIZEB)      ;init loop variable
        ldi     loophi, high(PAGESIZEB)     ;not required for PAGESIZEB<=256
        subi    YL, low(PAGESIZEB)          ;restore pointer
        sbci    YH, high(PAGESIZEB)
Rdloop:
        lpm     r0, Z+
        ld      r1, Y+
        cpse    r0, r1
        rjmp    Error
        sbiw    loophi:looplo, 1            ;use subi for PAGESIZEB<=256
        brne    Rdloop

        ;       return to RWW section
        ;       verify that RWW section is safe to read
Return:
        in      temp1, SPMCSR
        sbrs    temp1, RWWSB                ; If RWWSB is set, the RWW section is not
ready yet
        ret
        ;       re-enable the RWW section
        ldi     spmcrval, (1<<RWWSRE) | (1<<SELFPRGEN)
        rcall   Do_spm
        rjmp    Return

Do_spm:
        ;       check for previous SPM complete
Wait_spm:
        in      temp1, SPMCSR
        sbrc    temp1, SELFPRGEN
        rjmp    Wait_spm
        ;       input: spmcrval determines SPM action
        ;       disable interrupts if enabled, store status
        in      temp2, SREG
        cli
        ;       check that no EEPROM write access is present
Wait_ee:
        sbic    EECR, EEPE
        rjmp    Wait_ee
        ;       SPM timed sequence
        out     SPMCSR, spmcrval
        spm
        ;       restore SREG (to enable interrupts if originally enabled)
        out     SREG, temp2
        ret
```

Table 27-4. Boot Reset Fuse(1)

BOOTRST	Reset Address
1	Reset vector = Application reset (address 0x0000)
0	Reset vector = Boot loader reset (see Table 27-7 on page 247)

Note: 1. “1” means unprogrammed, “0” means programmed

27.7 Addressing the Flash during Self-programming

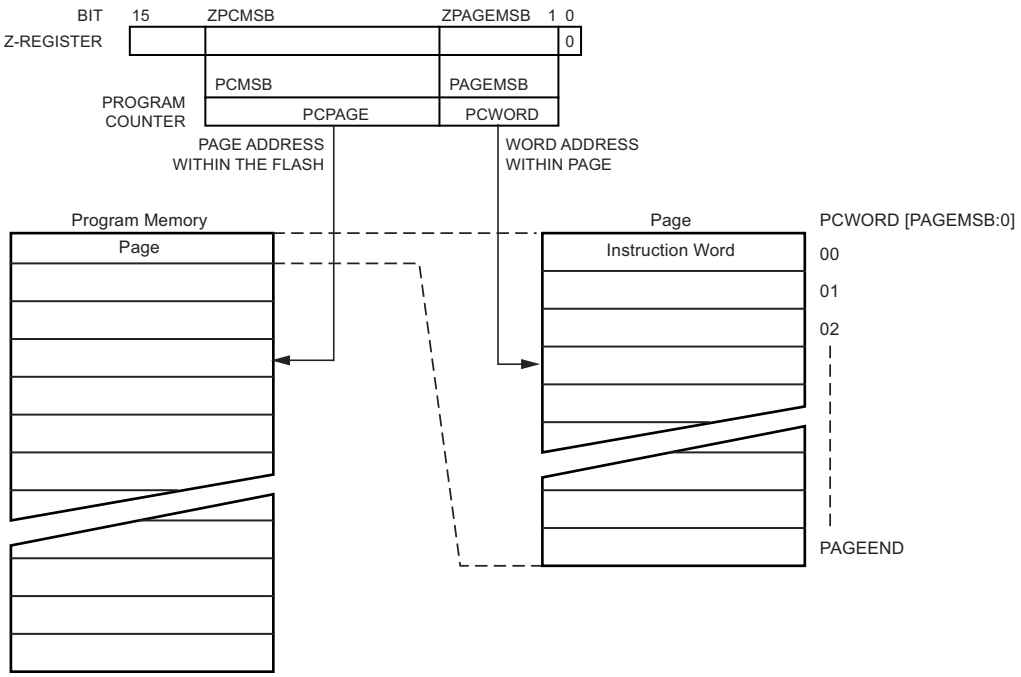
The Z-pointer is used to address the SPM commands.

Bit	15	14	13	12	11	10	9	8
ZH (R31)	Z15	Z14	Z13	Z12	Z11	Z10	Z9	Z8
ZL (R30)	Z7	Z6	Z5	Z4	Z3	Z2	Z1	Z0
	7	6	5	4	3	2	1	0

Since the flash is organized in pages (see Table 28-9 on page 255), the program counter can be treated as having two different sections. One section, consisting of the least significant bits, is addressing the words within a page, while the most significant bits are addressing the pages. This is1 shown in Figure 27-3. Note that the page erase and page write operations are addressed independently. Therefore it is of major importance that the boot loader software addresses the same page in both the page erase and page write operation. Once a programming operation is initiated, the address is latched and the Z-pointer can be used for other operations.

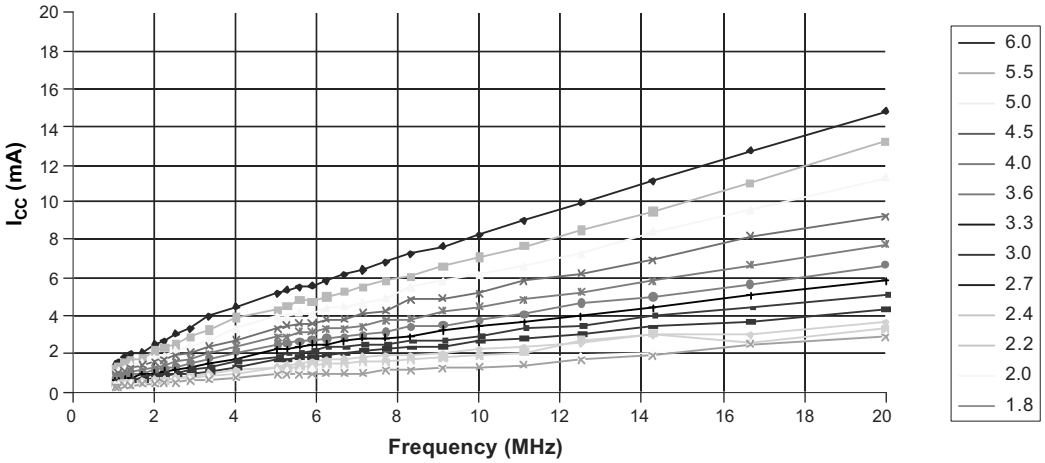
The only SPM operation that does not use the Z-pointer is setting the boot loader lock bits. The content of the Z-pointer is ignored and will have no effect on the operation. The LPM instruction does also use the Z-pointer to store the address. Since this instruction addresses the Flash byte-by-byte, also the LSB (bit Z0) of the Z-pointer is used.

Figure 27-3. Addressing the Flash During SPM(1)



Note: 1. The different variables used in Figure 27-3 are listed in Table 27-9 on page 247.

Figure 30-54. Active Supply Current versus Frequency (1-16MHz)



30.3.2 Idle Supply Current

Figure 30-55. Idle Supply Current versus Low Frequency (0.1-1.0MHz)

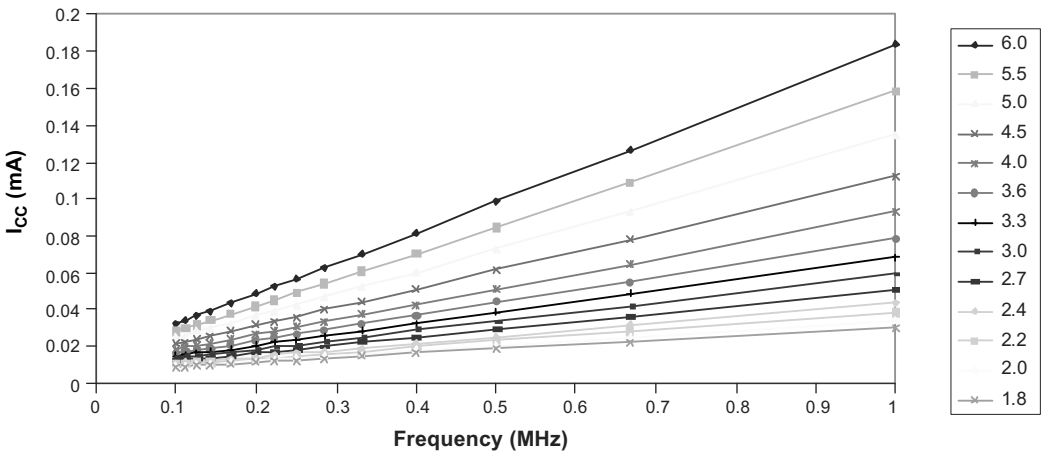
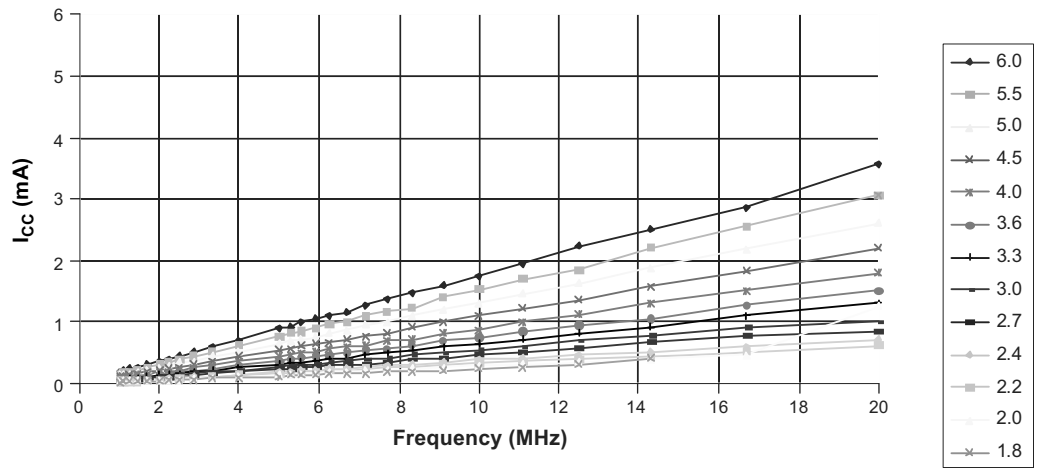


Figure 30-56. Idle Supply Current versus Frequency (1-16MHz)



31. Register Summary (Continued)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xBA)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	209
(0xB9)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	208
(0xB8)	TWBR	2-wire Serial Interface Bit Rate Register								206
(0xB7)	Reserved	–	–	–	–	–	–	–	–	
(0xB6)	ASSR	–	EXCLK	AS2	TCN2UB	OCR2AUB	OCR2BUB	TCR2AUB	TCR2BUB	142
(0xB5)	Reserved	–	–	–	–	–	–	–	–	
(0xB4)	OCR2B	Timer/Counter2 Output Compare Register B								141
(0xB3)	OCR2A	Timer/Counter2 Output Compare Register A								141
(0xB2)	TCNT2	Timer/Counter2 (8-bit)								141
(0xB1)	TCCR2B	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	140
(0xB0)	TCCR2A	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	137
(0xAF)	Reserved	–	–	–	–	–	–	–	–	
(0xAE)	Reserved	–	–	–	–	–	–	–	–	
(0xAD)	Reserved	–	–	–	–	–	–	–	–	
(0xAC)	Reserved	–	–	–	–	–	–	–	–	
(0xAB)	Reserved	–	–	–	–	–	–	–	–	
(0xAA)	Reserved	–	–	–	–	–	–	–	–	
(0xA9)	Reserved	–	–	–	–	–	–	–	–	
(0xA8)	Reserved	–	–	–	–	–	–	–	–	
(0xA7)	Reserved	–	–	–	–	–	–	–	–	
(0xA6)	Reserved	–	–	–	–	–	–	–	–	
(0xA5)	Reserved	–	–	–	–	–	–	–	–	
(0xA4)	Reserved	–	–	–	–	–	–	–	–	
(0xA3)	Reserved	–	–	–	–	–	–	–	–	
(0xA2)	Reserved	–	–	–	–	–	–	–	–	
(0xA1)	Reserved	–	–	–	–	–	–	–	–	
(0xA0)	Reserved	–	–	–	–	–	–	–	–	
(0x9F)	Reserved	–	–	–	–	–	–	–	–	
(0x9E)	Reserved	–	–	–	–	–	–	–	–	
(0x9D)	Reserved	–	–	–	–	–	–	–	–	
(0x9C)	Reserved	–	–	–	–	–	–	–	–	
(0x9B)	Reserved	–	–	–	–	–	–	–	–	
(0x9A)	Reserved	–	–	–	–	–	–	–	–	
(0x99)	Reserved	–	–	–	–	–	–	–	–	
(0x98)	Reserved	–	–	–	–	–	–	–	–	

Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the status flags are cleared by writing a logical one to them. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such status flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The Atmel ATmega48PA/88PA/168PA is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.
5. Only valid for the Atmel ATmega48PA/88PA/168PA.
6. BODS and BODSE only available for picoPower devices ATmega48PA/88PA/168PA