



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	SPI, UART/USART, USI
Peripherals	Brown-out Detect/Reset, LCD, POR, PWM, WDT
Number of I/O	54
Program Memory Size	16KB (8K x 16)
Program Memory Type	FLASH
EEPROM Size	512 x 8
RAM Size	1K x 8
Voltage - Supply (Vcc/Vdd)	2.7V ~ 5.5V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	64-VQFN Dual Rows, Exposed Pad
Supplier Device Package	64-QFN (7x7)
Purchase URL	<a href="https://www.e-xfl.com/product-detail/atmel/atmega169p-16mcu">https://www.e-xfl.com/product-detail/atmel/atmega169p-16mcu</a>

**2.2.14 AREF**

This is the analog reference pin for the A/D Converter.

**2.2.15 LCDCAP**

An external capacitor (typical > 470 nF) must be connected to the LCDCAP pin as shown in [Figure 23-2 on page 236](#). This capacitor acts as a reservoir for LCD power ( $V_{LCD}$ ). A large capacitance reduces ripple on  $V_{LCD}$  but increases the time until  $V_{LCD}$  reaches its target value.

## 7.3 EEPROM Data Memory

The ATmega169P contains 512 bytes of data EEPROM memory. It is organized as a separate data space, in which single bytes can be read and written. The EEPROM has an endurance of at least 100,000 write/erase cycles. This section describes the access between the EEPROM and the CPU, specifying the EEPROM Address Registers, the EEPROM Data Register, and the EEPROM Control Register.

For a detailed description of SPI, JTAG and Parallel data downloading to the EEPROM, see ["Serial Downloading" on page 310](#), ["Programming via the JTAG Interface" on page 316](#), and ["Parallel Programming Parameters, Pin Mapping, and Commands" on page 299](#) respectively.

### 7.3.1 EEPROM Read/Write Access

The EEPROM Access Registers are accessible in the I/O space.

The write access time for the EEPROM is given in [Table 7-1 on page 24](#). A self-timing function, however, lets the user software detect when the next byte can be written. If the user code contains instructions that write the EEPROM, some precautions must be taken. In heavily filtered power supplies,  $V_{CC}$  is likely to rise or fall slowly on power-up/down. This causes the device for some period of time to run at a voltage lower than specified as minimum for the clock frequency used. See ["Preventing EEPROM Corruption" on page 27](#) for details on how to avoid problems in these situations.

In order to prevent unintentional EEPROM writes, a specific write procedure must be followed.

When the EEPROM is read, the CPU is halted for four clock cycles before the next instruction is executed. When the EEPROM is written, the CPU is halted for two clock cycles before the next instruction is executed.

The following procedure should be followed when writing the EEPROM (the order of steps 3 and 4 is not essential). See ["EEPROM Register Description" on page 28](#) for supplementary description for each register bit:

1. Wait until EEWB becomes zero.
2. Wait until SPMEN 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 EEMWE bit while writing a zero to EEWB in EECR.
6. Within four clock cycles after setting EEMWE, write a logical one to EEWB.

The EEPROM can not be programmed during a CPU write to the Flash memory. The software must check that the Flash programming is completed before initiating a new EEPROM write. Step 2 is only relevant if the software contains a Boot Loader allowing the CPU to program the Flash. If the Flash is never being updated by the CPU, step 2 can be omitted. See ["Boot Loader Support – Read-While-Write Self-Programming" on page 280](#) 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.

## 7.3.2 EEPROM Write During Power-down Sleep Mode

When entering Power-down sleep mode while an EEPROM write operation is active, the EEPROM write operation will continue, and will complete before the Write Access time has passed. However, when the write operation is completed, the clock continues running, and as a consequence, the device does not enter Power-down entirely. It is therefore recommended to verify that the EEPROM write operation is completed before entering Power-down.

## 7.3.3 Preventing EEPROM Corruption

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

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

EEPROM data corruption can easily be avoided by following this design recommendation:

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 detection level of the internal BOD does not match the needed detection level, 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.

## 7.4 I/O Memory

The I/O space definition of the ATmega169P is shown in ["Register Summary" on page 373](#).

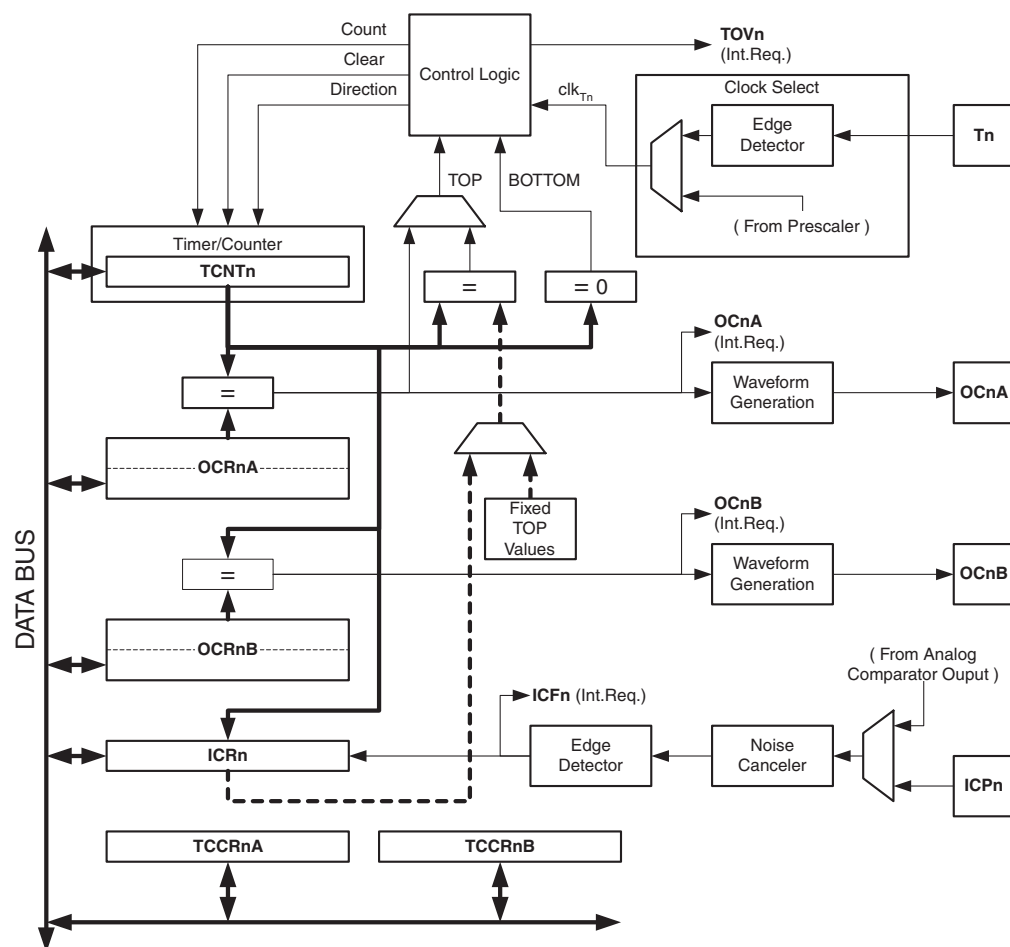
All ATmega169P I/Os and peripherals are placed in the I/O space. All I/O locations may be accessed by the LD/LDS/LDD and ST/STS/STD instructions, transferring data between the 32 general purpose working registers and the I/O space. I/O Registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions. Refer to the instruction set section for more details. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The ATmega169P is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

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

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

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

**Figure 15-1. 16-bit Timer/Counter Block Diagram<sup>(1)</sup>**



Note: 1. Refer to [Figure 1-1 on page 2](#), [Table 13-5 on page 74](#), and [Table 13-11 on page 78](#) for Timer/Counter1 pin placement and description.

## 15.2.1 Registers

The *Timer/Counter* (TCNT1), *Output Compare Registers* (OCR1A/B), and *Input Capture Register* (ICR1) are all 16-bit registers. Special procedures must be followed when accessing the 16-bit registers. These procedures are described in the section ["Accessing 16-bit Registers" on page 109](#). The *Timer/Counter Control Registers* (TCCR1A/B) are 8-bit registers and have no CPU access restrictions. Interrupt requests (abbreviated to Int.Req. in the figure) signals are all visible in the *Timer Interrupt Flag Register* (TIFR1). All interrupts are individually masked with the *Timer Interrupt Mask Register* (TIMSK1). TIFR1 and TIMSK1 are not shown in the figure.

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

The double buffered Output Compare Registers (OCR1A/B) are compared with the Timer/Counter value at all time. The result of the compare can be used by the Waveform Generator to generate a PWM or variable frequency output on the Output Compare pin (OC1A/B). See ["Out-](#)

the main code and the interrupt code update the temporary register, the main code must disable the interrupts during the 16-bit access.

The following code examples show how to do an atomic read of the TCNT1 Register contents. Reading any of the OCR1A/B or ICR1 Registers can be done by using the same principle.

## Assembly Code Example<sup>(1)</sup>

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

## C Code Example<sup>(1)</sup>

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

Note: 1. See "About Code Examples" on page 10.

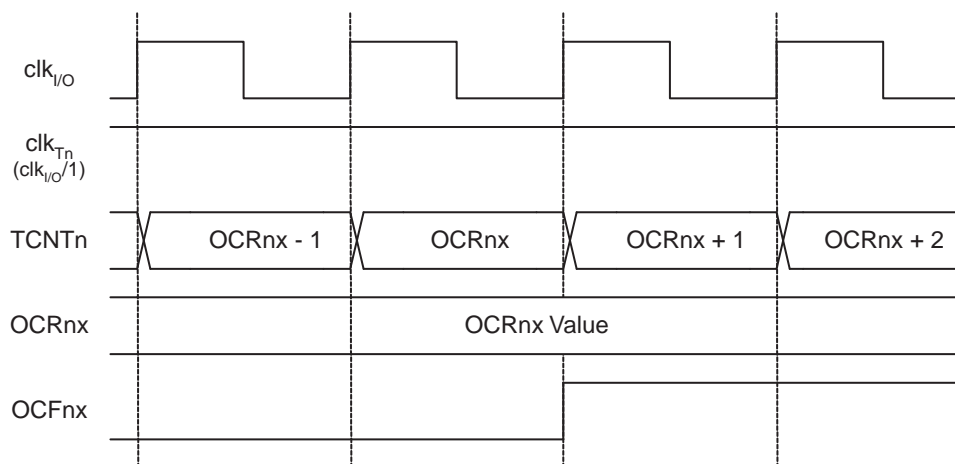
The assembly code example returns the TCNT1 value in the r17:r16 register pair.

The extreme values for the OCR1x Register represents special cases when generating a PWM waveform output in the phase and frequency 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 set to 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 = 9) and COM1A1:0 = 1, the OC1A output will toggle with a 50% duty cycle.

## 15.10 Timer/Counter Timing Diagrams

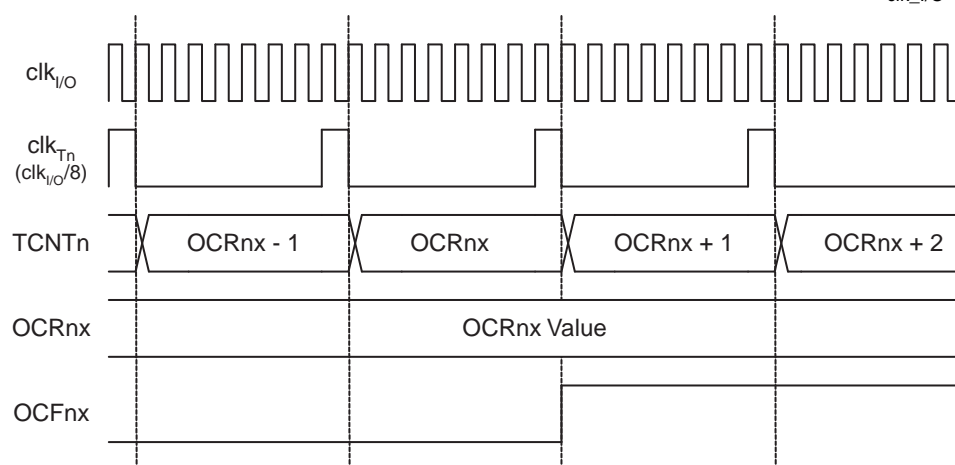
The Timer/Counter is a synchronous design and the timer clock ( $\text{clk}_{Tn}$ ) is therefore shown as a clock enable signal in the following figures. The figures include information on when Interrupt Flags are set, and when the OCR1x Register is updated with the OCR1x buffer value (only for modes utilizing double buffering). [Figure 15-10](#) shows a timing diagram for the setting of OCF1x.

**Figure 15-10.** Timer/Counter Timing Diagram, Setting of OCF1x, no Prescaling



[Figure 15-11](#) shows the same timing data, but with the prescaler enabled.

**Figure 15-11.** Timer/Counter Timing Diagram, Setting of OCF1x, with Prescaler ( $f_{\text{clk\_I/O}}/8$ )



[Figure 15-12 on page 127](#) shows the count sequence close to TOP in various modes. When using phase and frequency correct PWM mode the OCR1x Register is updated at BOTTOM. The timing diagrams will be the same, but TOP should be replaced by BOTTOM, TOP-1 by

FOC1A/FOC1B bits are implemented as strobes. Therefore it is the value present in the COM1x1:0 bits that determine the effect of the forced compare.

A FOC1A/FOC1B strobe will not generate any interrupt nor will it clear the timer in Clear Timer on Compare match (CTC) mode using OCR1A as TOP.

The FOC1A/FOC1B bits are always read as zero.

## 15.11.4 TCNT1H and TCNT1L – Timer/Counter1

Bit	7	6	5	4	3	2	1	0	
(0x85)	TCNT1[15:8]								TCNT1H
(0x84)	TCNT1[7:0]								TCNT1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The two *Timer/Counter* I/O locations (TCNT1H and TCNT1L, combined TCNT1) give direct access, both for read and for write operations, to the Timer/Counter unit 16-bit counter. To ensure that both the high and low bytes are read and written simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This temporary register is shared by all the other 16-bit registers. See ["Accessing 16-bit Registers" on page 109](#).

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

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

## 15.11.5 OCR1AH and OCR1AL – Output Compare Register 1 A

Bit	7	6	5	4	3	2	1	0	
(0x89)	OCR1A[15:8]								OCR1AH
(0x88)	OCR1A[7:0]								OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

## 15.11.6 OCR1BH and OCR1BL – Output Compare Register 1 B

Bit	7	6	5	4	3	2	1	0	
(0x8B)	OCR1B[15:8]								OCR1BH
(0x8A)	OCR1B[7:0]								OCR1BL
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	

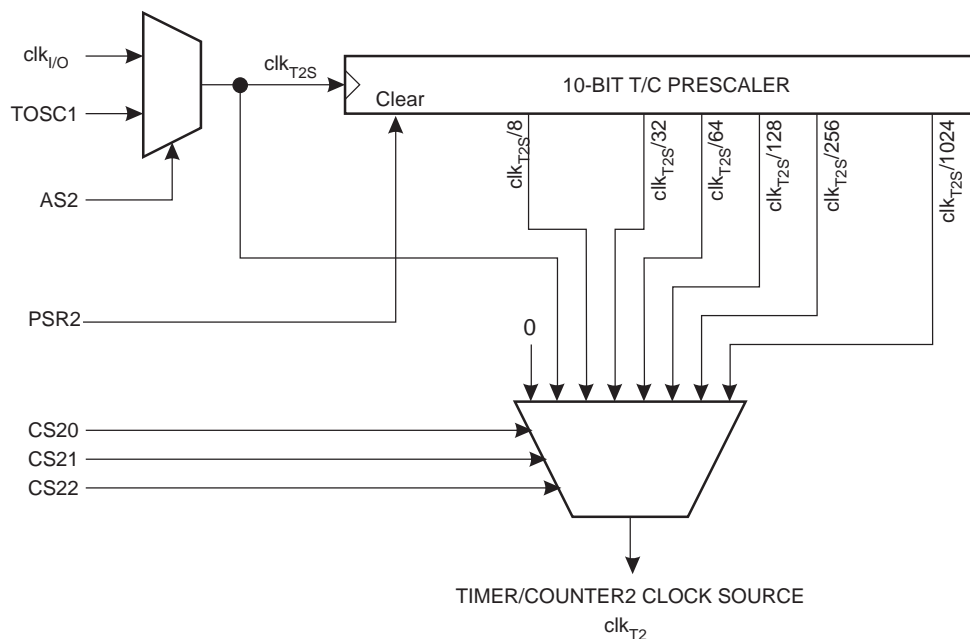
The Output Compare Registers contain a 16-bit value that is continuously compared with the counter value (TCNT1). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC1x pin.

The Output Compare Registers are 16-bit in size. To ensure that both the high and low bytes are written simultaneously when the CPU writes to these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This temporary register is shared by all the other 16-bit registers. See ["Accessing 16-bit Registers" on page 109](#).



## 17.9 Timer/Counter Prescaler

Figure 17-12. Prescaler for Timer/Counter2



The clock source for Timer/Counter2 is named  $clk_{T2S}$ .  $clk_{T2S}$  is by default connected to the main system I/O clock  $clk_{I/O}$ . By setting the AS2 bit in ASSR, Timer/Counter2 is asynchronously clocked from the TOSC1 pin. This enables use of Timer/Counter2 as a Real Time Counter (RTC). When AS2 is set, pins TOSC1 and TOSC2 are disconnected from Port C. A crystal can then be connected between the TOSC1 and TOSC2 pins to serve as an independent clock source for Timer/Counter2. The Oscillator is optimized for use with a 32.768 kHz crystal. If applying an external clock on TOSC1, the EXCLK bit in ASSR must be set.

For Timer/Counter2, the possible prescaled selections are:  $clk_{T2S}/8$ ,  $clk_{T2S}/32$ ,  $clk_{T2S}/64$ ,  $clk_{T2S}/128$ ,  $clk_{T2S}/256$ , and  $clk_{T2S}/1024$ . Additionally,  $clk_{T2S}$  as well as 0 (stop) may be selected. Setting the PSR2 bit in GTCCR resets the prescaler. This allows the user to operate with a predictable prescaler.

When the SPI is enabled, the data direction of the MOSI, MISO, SCK, and  $\overline{SS}$  pins is overridden according to [Table 18-1](#). For more details on automatic port overrides, refer to ["Alternate Port Functions" on page 71](#).

**Table 18-1.** SPI Pin Overrides<sup>(1)</sup>

Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User Defined	Input
MISO	Input	User Defined
SCK	User Defined	Input
$\overline{SS}$	User Defined	Input

Note: 1. See ["Alternate Functions of Port B" on page 74](#) for a detailed description of how to define the direction of the user defined SPI pins.

The following code examples show how to initialize the SPI as a Master and how to perform a simple transmission. DDR\_SPI in the examples must be replaced by the actual Data Direction Register controlling the SPI pins. DD\_MOSI, DD\_MISO and DD\_SCK must be replaced by the actual data direction bits for these pins. For example if MOSI is placed on pin PB5, replace DD\_MOSI with DDB5 and DDR\_SPI with DDRB.

For the assembly code, the baud rate parameter is assumed to be stored in the r17:r16 Registers.

## Assembly Code Example<sup>(1)</sup>

```

USART_Init:
    ; Set baud rate
    sts  UBRRH0, r17
    sts  UBRRLO, r16
    ; Enable receiver and transmitter
    ldi  r16, (1<<RXEN0) | (1<<TXEN0)
    sts  UCSR0B, r16
    ; Set frame format: 8data, 2stop bit
    ldi  r16, (1<<USBS0) | (3<<UCSZ00)
    sts  UCSR0C, r16
    ret

```

## C Code Example<sup>(1)</sup>

```

#define FOSC 1843200// Clock Speed
#define BAUD 9600
#define MYUBRR FOSC/16/BAUD-1
void main( void )
{
    ...
    USART_Init ( MYUBRR );
    ...
}

void USART_Init( unsigned int ubrr)
{
    /* Set baud rate */
    UBRRH0 = (unsigned char) (ubrr>>8);
    UBRRLO = (unsigned char)ubrr;
    /* Enable receiver and transmitter */
    UCSR0B = (1<<RXEN0) | (1<<TXEN0);
    /* Set frame format: 8data, 2stop bit */
    UCSRnC = (1<<USBS0) | (3<<UCSZ00);
}

```

Note: 1. See "About Code Examples" on page 10.

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.

The following equations can be used to calculate the ratio of the incoming data rate and internal receiver baud rate.

$$R_{slow} = \frac{(D+1)S}{S-1+D \cdot S+S_F} \qquad R_{fast} = \frac{(D+2)S}{(D+1)S+S_M}$$

- D** Sum of character size and parity size (D = 5 to 10 bit).
- S** Samples per bit. S = 16 for Normal Speed mode and S = 8 for Double Speed mode.
- S<sub>F</sub>** First sample number used for majority voting. S<sub>F</sub> = 8 for normal speed and S<sub>F</sub> = 4 for Double Speed mode.
- S<sub>M</sub>** Middle sample number used for majority voting. S<sub>M</sub> = 9 for normal speed and S<sub>M</sub> = 5 for Double Speed mode.
- R<sub>slow</sub>** is the ratio of the slowest incoming data rate that can be accepted in relation to the receiver baud rate. R<sub>fast</sub> is the ratio of the fastest incoming data rate that can be accepted in relation to the receiver baud rate.

Table 19-2 and Table 19-3 list the maximum receiver baud rate error that can be tolerated. Note that Normal Speed mode has higher toleration of baud rate variations.

**Table 19-2.** Recommended Maximum Receiver Baud Rate Error for Normal Speed Mode (U2Xn = 0)

D # (Data+Parity Bit)	R <sub>slow</sub> (%)	R <sub>fast</sub> (%)	Max Total Error (%)	Recommended Max Receiver Error (%)
5	93.20	106.67	+6.67/-6.8	±3.0
6	94.12	105.79	+5.79/-5.88	±2.5
7	94.81	105.11	+5.11/-5.19	±2.0
8	95.36	104.58	+4.58/-4.54	±2.0
9	95.81	104.14	+4.14/-4.19	±1.5
10	96.17	103.78	+3.78/-3.83	±1.5

**Table 19-3.** Recommended Maximum Receiver Baud Rate Error for Double Speed Mode (U2Xn = 1)

D # (Data+Parity Bit)	R <sub>slow</sub> (%)	R <sub>fast</sub> (%)	Max Total Error (%)	Recommended Max Receiver Error (%)
5	94.12	105.66	+5.66/-5.88	±2.5
6	94.92	104.92	+4.92/-5.08	±2.0
7	95.52	104.35	+4.35/-4.48	±1.5
8	96.00	103.90	+3.90/-4.00	±1.5
9	96.39	103.53	+3.53/-3.61	±1.5
10	96.70	103.23	+3.23/-3.30	±1.0

**Table 19-6.** Examples of UBRRn Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	$f_{osc} = 8.0000 \text{ MHz}$				$f_{osc} = 11.0592 \text{ MHz}$				$f_{osc} = 14.7456 \text{ MHz}$			
	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. <sup>(1)</sup>	0.5 Mbps		1 Mbps		691.2 Kbps		1.3824 Mbps		921.6 Kbps		1.8432 Mbps	

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

## 22.6.1 ADC Input Channels

When changing channel selections, the user should observe the following guidelines to ensure that the correct channel is selected:

In Single Conversion mode, always select the channel before starting the conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the conversion to complete before changing the channel selection.

In Free Running mode, always select the channel before starting the first conversion. The channel selection may be changed one ADC clock cycle after writing one to ADSC. However, the simplest method is to wait for the first conversion to complete, and then change the channel selection. Since the next conversion has already started automatically, the next result will reflect the previous channel selection. Subsequent conversions will reflect the new channel selection.

## 22.6.2 ADC Voltage Reference

The reference voltage for the ADC ( $V_{REF}$ ) indicates the conversion range for the ADC. Single ended channels that exceed  $V_{REF}$  will result in codes close to 0x3FF.  $V_{REF}$  can be selected as either AVCC, internal 1.1V reference, or external AREF pin.

AVCC is connected to the ADC through a passive switch. The internal 1.1V reference is generated from the internal bandgap reference ( $V_{BG}$ ) through an internal buffer. In either case, the external AREF pin is directly connected to the ADC, and the reference voltage can be made more immune to noise by connecting a capacitor between the AREF pin and ground.  $V_{REF}$  can also be measured at the AREF pin with a high impedant voltmeter. Note that  $V_{REF}$  is a high impedant source, and only a capacitive load should be connected in a system.

If the user has a fixed voltage source connected to the AREF pin, the user may not use the other reference voltage options in the application, as they will be shorted to the external voltage. If no external voltage is applied to the AREF pin, the user may switch between AVCC and 1.1V as reference selection. The first ADC conversion result after switching reference voltage source may be inaccurate, and the user is advised to discard this result.

## 22.7 ADC Noise Canceler

The ADC features a noise canceler that enables conversion during sleep mode to reduce noise induced from the CPU core and other I/O peripherals. The noise canceler can be used with ADC Noise Reduction and Idle mode. To make use of this feature, the following procedure should be used:

- a. Make sure that the ADC is enabled and is not busy converting. Single Conversion mode must be selected and the ADC conversion complete interrupt must be enabled.
- b. Enter ADC Noise Reduction mode (or Idle mode). The ADC will start a conversion once the CPU has been halted.
- c. If no other interrupts occur before the ADC conversion completes, the ADC interrupt will wake up the CPU and execute the ADC Conversion Complete interrupt routine. If another interrupt wakes up the CPU before the ADC conversion is complete, that interrupt will be executed, and an ADC Conversion Complete interrupt request will be generated when the ADC conversion completes. The CPU will remain in active mode until a new sleep command is executed.

Note that the ADC will not be automatically turned off when entering other sleep modes than Idle mode and ADC Noise Reduction mode. The user is advised to write zero to ADEN before entering such sleep modes to avoid excessive power consumption.

## Assembly Code Example<sup>(1)</sup>

```

LCD_Init:
    ; Use 32 kHz crystal oscillator
    ; 1/3 Bias and 1/3 duty, SEG21:SEG24 is used as port pins
    ldi r16, (1<<LCDCS) | (1<<LCDMUX1) | (1<<LCDPM2)
    sts LCDCRB, r16
    ; Using 16 as prescaler selection and 7 as LCD Clock Divide
    ; gives a frame rate of 49 Hz
    ldi r16, (1<<LCDCD2) | (1<<LCDCD1)
    sts LCDFRR, r16
    ; Set segment drive time to 125 µs and output voltage to 3.3 V
    ldi r16, (1<<LCDDC1) | (1<<LCDCC3) | (1<<LCDCC2) | (1<<LCDCC1)
    sts LCDCCR, r16
    ; Enable LCD, default waveform and no interrupt enabled
    ldi r16, (1<<LCDEN)
    sts LCDERA, r16
    ret

```

## C Code Example<sup>(1)</sup>

```

Void LCD_Init(void);
{
    /* Use 32 kHz crystal oscillator */
    /* 1/3 Bias and 1/3 duty, SEG21:SEG24 is used as port pins */
    LCDCRB = (1<<LCDCS) | (1<<LCDMUX1) | (1<<LCDPM2);
    /* Using 16 as prescaler selection and 7 as LCD Clock Divide */
    /* gives a frame rate of 49 Hz */
    LCDFRR = (1<<LCDCD2) | (1<<LCDCD1);
    /* Set segment drive time to 125 µs and output voltage to 3.3 V */
    LCDCCR = (1<<LCDDC1) | (1<<LCDCC3) | (1<<LCDCC2) | (1<<LCDCC1);

    /* Enable LCD, default waveform and no interrupt enabled */
    LCDERA = (1<<LCDEN);
}

```

Note: 1. See "About Code Examples" on page 10.

Before a re-initialization is done, the LCD controller/driver should be disabled.

### 23.4.2 Updating the LCD

Display memory (LCDDR0, LCDDR1, ..), LCD Blanking (LCDBL), Low power waveform (LCDAB) and contrast control (LCDCCR) are latched prior to every new frame. There are no restrictions on writing these LCD Register locations, but an LCD data update may be split between two frames if data are latched while an update is in progress. To avoid this, an interrupt routine can be used to update Display memory, LCD Blanking, Low power waveform, and contrast control, just after data are latched.

## 23.5.4 LCDCCR – LCD Contrast Control Register

Bit	7	6	5	4	3	2	1	0	
(0xE7)	LCDDC2	LCDDC1	LCDDC0	LCDMDT	LCDCC3	LCDCC2	LCDCC1	LCDCC0	LCDCCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

### • Bits 7:5 – LCDDC2:0: LDC Display Configuration

The LCDDC2:0 bits determine the amount of time the LCD drivers are turned on for each voltage transition on segment and common pins. A short drive time will lead to lower power consumption, but displays with high internal resistance may need longer drive time to achieve satisfactory contrast. Note that the drive time will never be longer than one half prescaled LCD clock period, even if the selected drive time is longer. When using static bias or blanking, drive time will always be one half prescaled LCD clock period.

**Table 23-7.** LCD Display Configuration

LCDDC2	LCDDC1	LCDDC0	Nominal drive time
0	0	0	300 $\mu$ s
0	0	1	70 $\mu$ s
0	1	0	150 $\mu$ s
0	1	1	450 $\mu$ s
1	0	0	575 $\mu$ s
1	0	1	850 $\mu$ s
1	1	0	1150 $\mu$ s
1	1	1	50% of $\text{clk}_{\text{LCD\_PS}}$

### • Bit 4 – LCDMDT: LCD Maximum Drive Time

Writing this bit to one turns the LCD drivers on 100% on the time, regardless of the drive time configured by LCDDC2:0.

### • Bits 3:0 – LCDCC3:0: LCD Contrast Control

The LCDCC3:0 bits determine the maximum voltage  $V_{\text{LCD}}$  on segment and common pins. The different selections are shown in [Table 23-8 on page 251](#). New values take effect every beginning of a new frame.



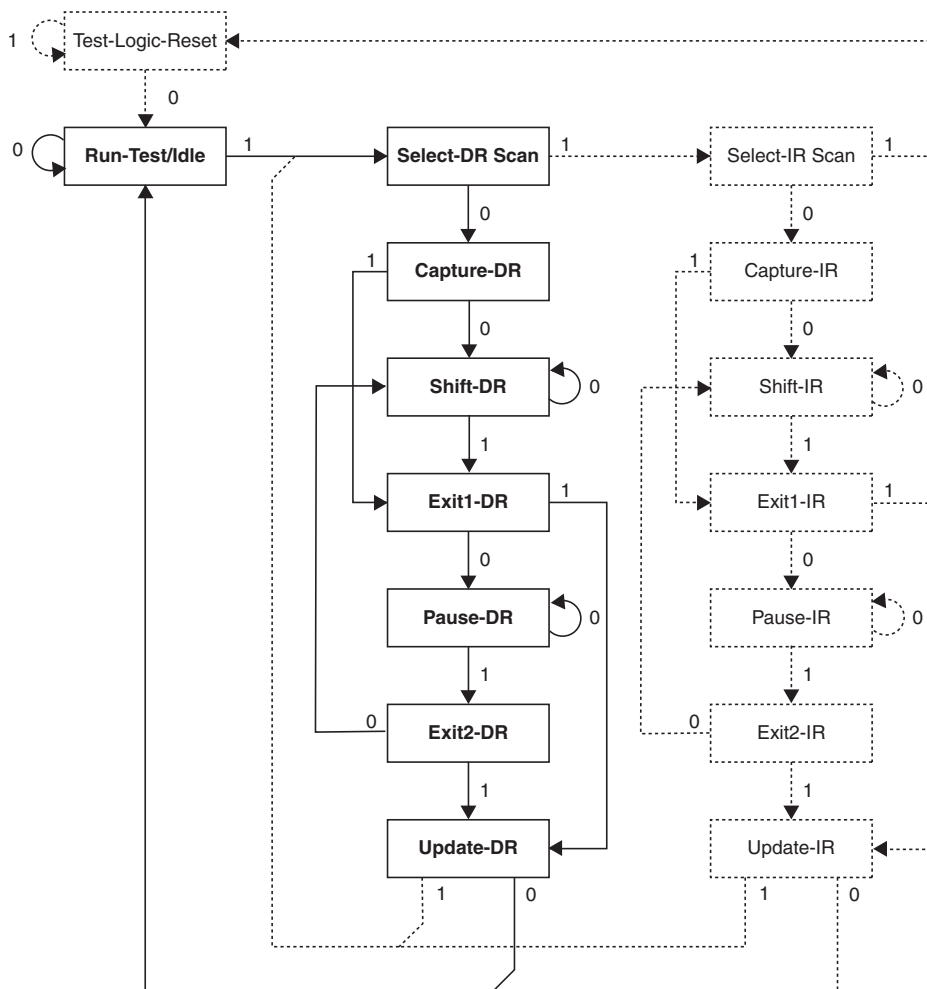
**Table 27-11.** XA1 and XA0 Coding

XA1	XA0	Action when XTAL1 is Pulsed
0	0	Load Flash or EEPROM Address (High or low address byte determined by BS1).
0	1	Load Data (High or Low data byte for Flash determined by BS1).
1	0	Load Command
1	1	No Action, Idle

**Table 27-12.** Command Byte Bit Coding

Command Byte	Command Executed
1000 0000	Chip Erase
0100 0000	Write Fuse bits
0010 0000	Write Lock bits
0001 0000	Write Flash
0001 0001	Write EEPROM
0000 1000	Read Signature Bytes and Calibration byte
0000 0100	Read Fuse and Lock bits
0000 0010	Read Flash
0000 0011	Read EEPROM

**Figure 27-16.** State Machine Sequence for Changing/Reading the Data Word



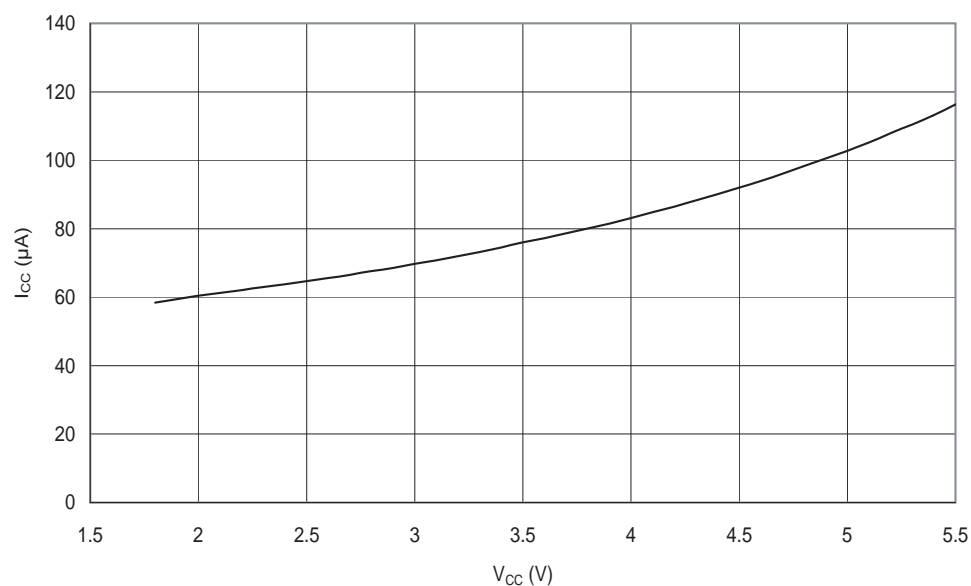
## 27.9.11 Flash Data Byte Register

The Flash Data Byte Register provides an efficient way to load the entire Flash page buffer before executing Page Write, or to read out/verify the content of the Flash. A state machine sets up the control signals to the Flash and senses the strobe signals from the Flash, thus only the data words need to be shifted in/out.

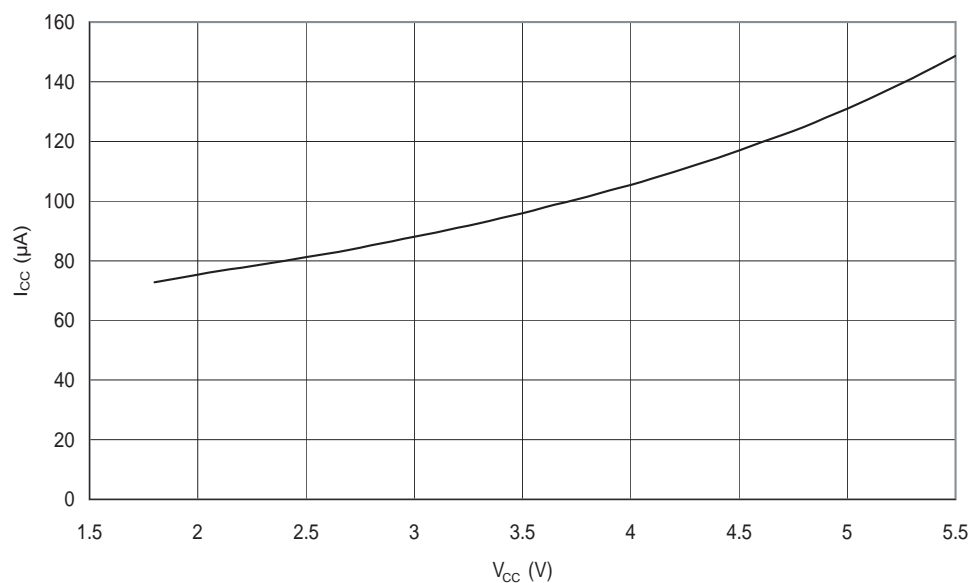
The Flash Data Byte Register actually consists of the 8-bit scan chain and a 8-bit temporary register. During page load, the Update-DR state copies the content of the scan chain over to the temporary register and initiates a write sequence that within 11 TCK cycles loads the content of the temporary register into the Flash page buffer. The AVR automatically alternates between writing the low and the high byte for each new Update-DR state, starting with the low byte for the first Update-DR encountered after entering the PROG\_PAGELOAD command. The Program Counter is pre-incremented before writing the low byte, except for the first written byte. This ensures that the first data is written to the address set up by PROG\_COMMANDS, and loading the last location in the page buffer does not make the Program Counter increment into the next page.

During Page Read, the content of the selected Flash byte is captured into the Flash Data Byte Register during the Capture-DR state. The AVR automatically alternates between reading the low and the high byte for each new Capture-DR state, starting with the low byte for the first Cap-

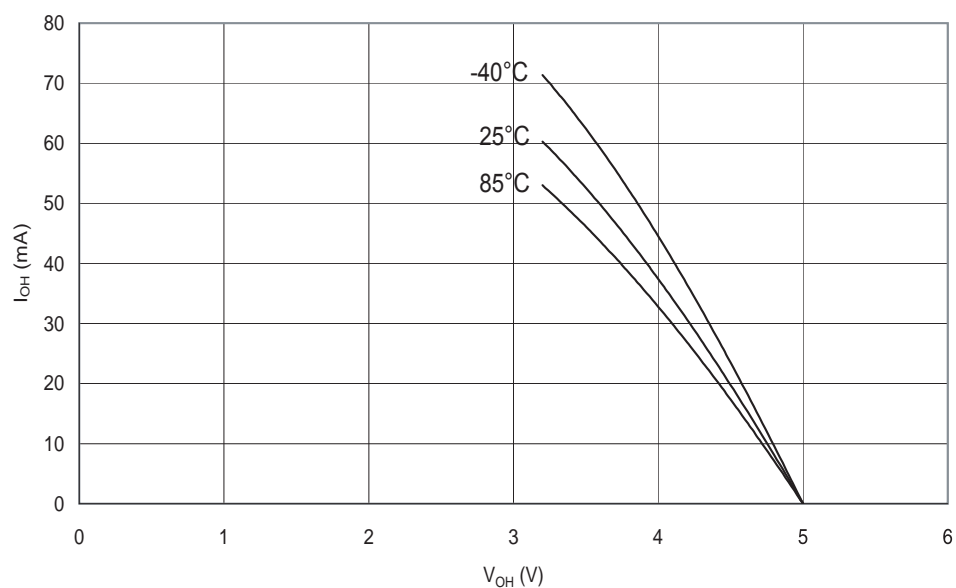
**Figure 29-20.** Standby Supply Current vs.  $V_{CC}$  (4 MHz Xtal, Watchdog Timer Disabled)



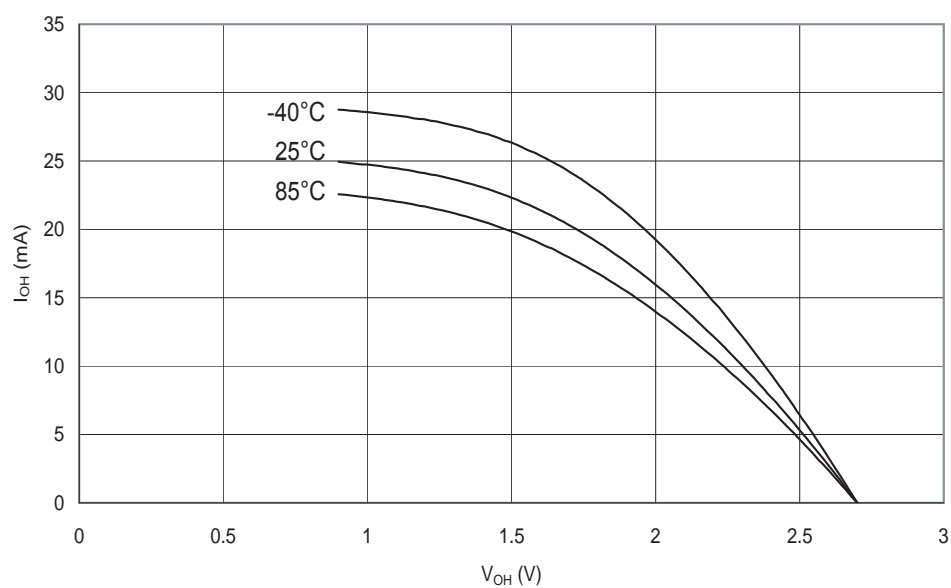
**Figure 29-21.** Standby Supply Current vs.  $V_{CC}$  (6 MHz Resonator, Watchdog Timer Disabled)



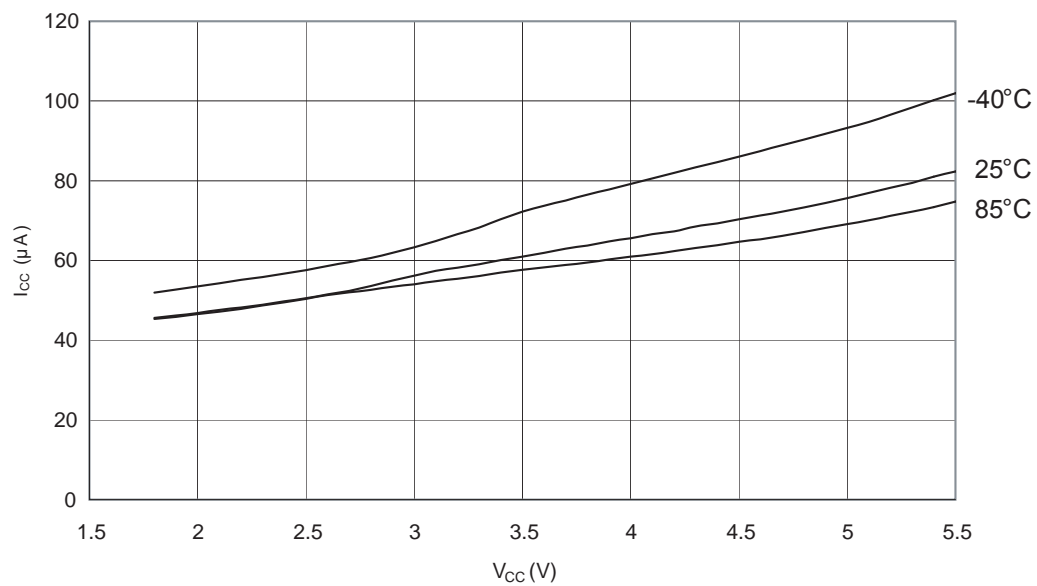
**Figure 29-32.** I/O Pin Source Current vs. Output Voltage, Port B ( $V_{CC}=5V$ )



**Figure 29-33.** I/O Pin Source Current vs. Output Voltage, Port B ( $V_{CC}=2.7V$ )



**Figure 29-62.** Analog Comparator Current vs.  $V_{CC}$



**Figure 29-63.** Programming Current vs.  $V_{CC}$

