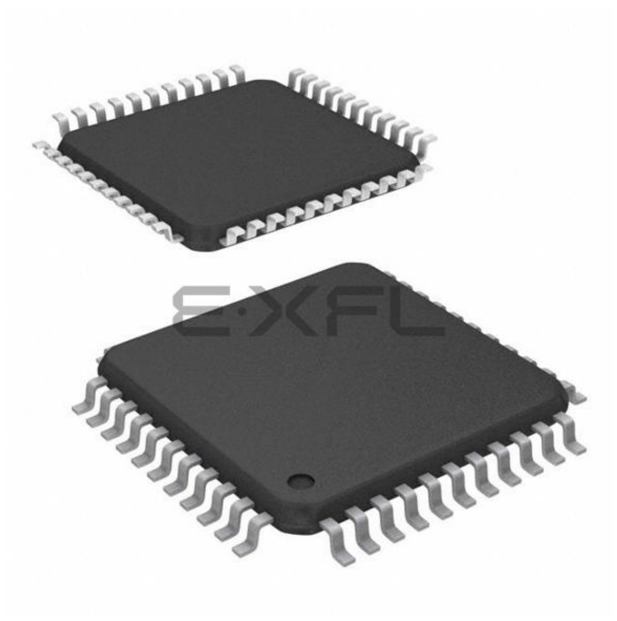
# Microchip Technology - ATMEGA644PV-10AUR Datasheet



**Welcome to E-XFL.COM**

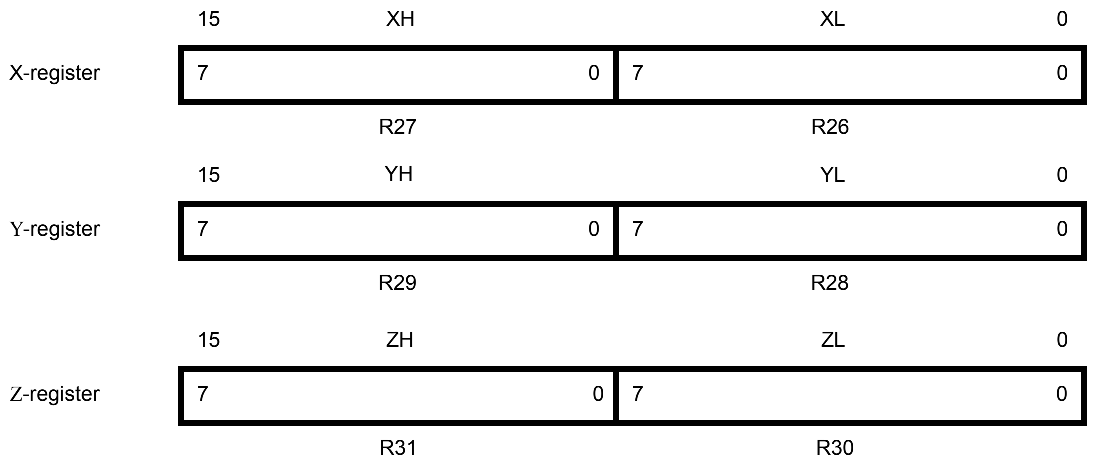
## What is "Embedded - Microcontrollers"?

"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 | 10MHz |
| Connectivity | I²C, SPI, UART/USART |
| Peripherals | Brown-out Detect/Reset, POR, PWM, WDT |
| Number of I/O | 32 |
| Program Memory Size | 64KB (32K x 16) |
| Program Memory Type | FLASH |
| EEPROM Size | 2K x 8 |
| RAM Size | 4K 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 | 44-TQFP |
| Supplier Device Package | 44-TQFP (10x10) |
| Purchase URL | https://www.e-xfl.com/product-detail/microchip-technology/atmega644pv-10aur |

**Figure 8-3. The X-, Y-, and Z-registers**



In the different addressing modes these address registers have functions as fixed displacement, automatic increment, and automatic decrement (see the instruction set reference for details).

**Related Links**

Instruction Set Summary on page 437

## 8.5. Stack Pointer

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack is implemented as growing from higher to lower memory locations. The Stack Pointer Register always points to the top of the Stack.

The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. A Stack PUSH command will decrease the Stack Pointer. The Stack in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. Initial Stack Pointer value equals the last address of the internal SRAM and the Stack Pointer must be set to point above start of the SRAM. See the table for Stack Pointer details.

**Table 8-1. Stack Pointer Instructions**

| Instruction | Stack pointer | Description |
|---|---|---|
| PUSH | Decremented by 1 | Data is pushed onto the stack |
| CALL<br>ICALL<br>RCALL | Decremented by 2 | Return address is pushed onto the stack with a subroutine call or interrupt |
| POP | Incremented by 1 | Data is popped from the stack |
| RET<br>RETI | Incremented by 2 | Return address is popped from the stack with return from subroutine or return from interrupt |

The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent. Note that the data space in some implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH Register will not be present.

When applying an external clock, it is required to avoid sudden changes in the applied clock frequency to ensure stable operation of the MCU. A variation in frequency of more than 2% from one clock cycle to the next can lead to unpredictable behavior. If changes of more than 2% is required, ensure that the MCU is kept in Reset during the changes.

The System Clock Prescaler can be used to implement run-time changes of the internal clock frequency while still ensuring stable operation.

**Related Links**

System Clock Prescaler on page 52

## 10.9. Timer/Counter Oscillator

The device uses the same crystal oscillator for Low-frequency Oscillator and Timer/Counter Oscillator. See Low Frequency Crystal Oscillator for details on the oscillator and crystal requirements.

On this device, the Timer/Counter Oscillator Pins (TOSC1 and TOSC2) are shared with XTAL1 and XTAL2. When using the Timer/Counter Oscillator, the system clock needs to be four times the oscillator frequency. Due to this and the pin sharing, the Timer/Counter Oscillator can only be used when the Calibrated Internal RC Oscillator is selected as system clock source.

Applying an external clock source to TOSC1 can be done if the Enable External Clock Input bit in the Asynchronous Status Register (ASSR.EXCLK) is written to '1'. See the description of the Asynchronous Operation of Timer/Counter2 for further description on selecting external clock as input instead of a 32.768kHz watch crystal.

**Related Links**

Low Frequency Crystal Oscillator on page 48

OCR2B on page 210

ASSR on page 213

## 10.10. Clock Output Buffer

The device can output the system clock on the CLKO pin. To enable the output, the CKOUT Fuse has to be programmed. This mode is suitable when the chip clock is used to drive other circuits on the system. The clock also will be output during reset, and the normal operation of I/O pin will be overridden when the fuse is programmed. Any clock source, including the internal RC Oscillator, can be selected when the clock is output on CLKO. If the System Clock Prescaler is used, it is the divided system clock that is output.

## 10.11. System Clock Prescaler

The device has a system clock prescaler, and the system clock can be divided by configuring the Clock Prescale Register (CLKPR). This feature can be used to decrease the system clock frequency and the power consumption when the requirement for processing power is low. This can be used with all clock source options, and it will affect the clock frequency of the CPU and all synchronous peripherals. $clk_{I/O}$, $clk_{ADC}$, $clk_{CPU}$, and $clk_{FLASH}$ are divided by a factor as shown in the CLKPR description.

When switching between prescaler settings, the System Clock Prescaler ensures that no glitches occurs in the clock system. It also ensures that no intermediate frequency is higher than neither the clock frequency corresponding to the previous setting, nor the clock frequency corresponding to the new setting. The ripple counter that implements the prescaler runs at the frequency of the undivided clock, which may be faster than the CPU's clock frequency. Hence, it is not possible to determine the state of

  - OC0A: Output Compare Match A output. The PB3 pin can serve as an external output for the Timer/Counter0 Output Compare. The pin has to be configured as an output (DDB3 set “1”) to serve this function. The OC0A pin is also the output pin for the PWM mode timer function.
  - PCINT11: Pin Change Interrupt source 11. The PB3 pin can serve as an external interrupt source.
- AIN0/INT2/PCINT10 – Port B, Bit 2
  - AIN0: Analog Comparator Positive input. This pin is directly connected to the positive input of the Analog Comparator.
  - INT2: External Interrupt source 2. The PB2 pin can serve as an External Interrupt source to the MCU.
  - PCINT10: Pin Change Interrupt source 10. The PB2 pin can serve as an external interrupt source.
- T1/CLKO/PCINT9 – Port B, Bit 1
  - T1: Timer/Counter1 counter source.
  - CLKO: Divided System Clock: The divided system clock can be output on the PB1 pin. The divided system clock will be output if the CKOUT Fuse is programmed, regardless of the PORTB1 and DDB1 settings. It will also be output during reset.
  - PCINT9: Pin Change Interrupt source 9. The PB1 pin can serve as an external interrupt source.
- T0/XCK0/PCINT8 – Port B, Bit 0
  - T0: Timer/Counter0 counter source.
  - XCK0: USART0 External clock. The Data Direction Register (DDB0) controls whether the clock is output (DDB0 set “1”) or input (DDB0 cleared). The XCK0 pin is active only when the USART0 operates in Synchronous mode.
  - PCINT8: Pin Change Interrupt source 8. The PB0 pin can serve as an external interrupt source.

Table 15-7 and Table 15-8 relate the alternate functions of Port B to the overriding signals shown in Figure 15-5. SPI MSTR INPUT and SPI SLAVE OUTPUT constitute the MISO signal, while MOSI is divided into SPI MSTR OUTPUT and SPI SLAVE INPUT.

**Table 15-7. Overriding Signals for Alternate Functions in PB[7:4]**

| Signal Name | PB7/SCK/PCINT15 | PB6/MISO/PCINT14 | PB5/MOSI/PCINT13 | PB4/$\overline{SS}$/OC0B/PCINT12 |
|---|---|---|---|---|
| PUOE | SPE • $\overline{MSTR}$ | SPE • MSTR | SPE • $\overline{MSTR}$ | SPE • $\overline{MSTR}$ |
| PUOV | PORTB7 • $\overline{PUD}$ | PORTB6 • $\overline{PUD}$ | PORTB5 • $\overline{PUD}$ | PORTB4 • $\overline{PUD}$ |
| DDOE | SPE • $\overline{MSTR}$ | SPE • MSTR | SPE • $\overline{MSTR}$ | SPE • $\overline{MSTR}$ |
| DDOV | 0 | 0 | 0 | 0 |
| PVOE | SPE • MSTR | SPE • $\overline{MSTR}$ | SPE • MSTR | OC0B ENABLE |
| PVOV | SCK OUTPUT | SPI SLAVE OUTPUT | SPI MSTR OUTPUT | OC0B |
| DIEOE | PCINT15 • PCIE1 | PCINT14 • PCIE1 | PCINT13 • PCIE1 | PCINT12 • PCIE1 |
| DIEOV | 1 | 1 | 1 | 1 |

| SPI Mode | Conditions | Leading Edge | Trailing Edge |
|---|---|---|---|
| 2 | CPOL=1, CPHA=0 | Sample (Falling) | Setup (Rising) |
| 3 | CPOL=1, CPHA=1 | Setup (Falling) | Sample (Rising) |

The SPI data transfer formats are shown in the following figure.

**Figure 20-3. SPI Transfer Format with CPHA = 0**

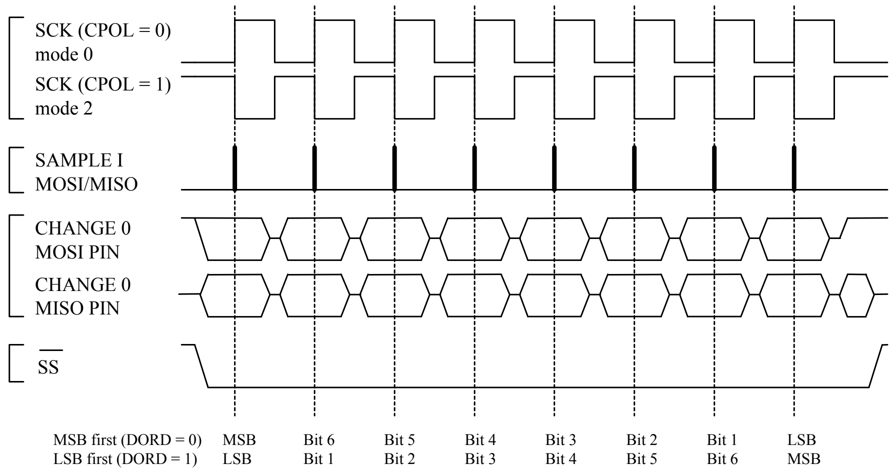
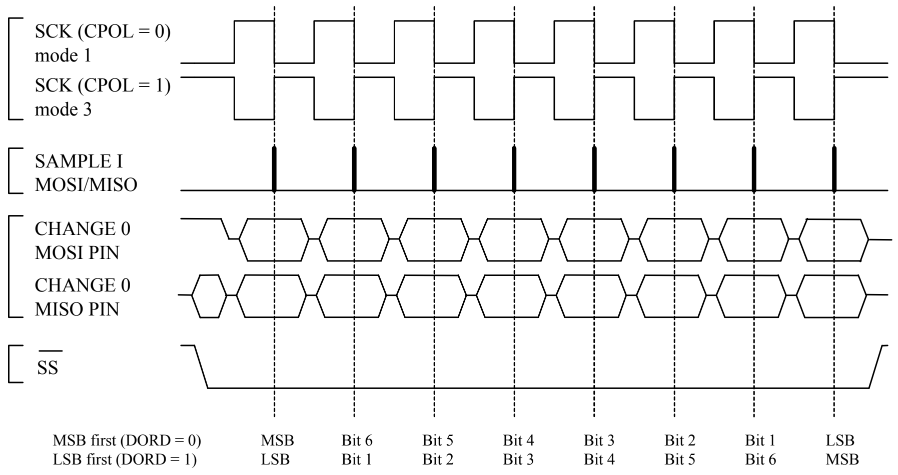

SCK (CPOL = 0) mode 0
SCK (CPOL = 1) mode 2
SAMPLE I MOSI/MISO
CHANGE 0 MOSI PIN
CHANGE 0 MISO PIN
SS
MSB first (DORD = 0) MSB Bit 6 Bit 5 Bit 4 Bit 3 Bit 2 Bit 1 LSB
LSB first (DORD = 1) LSB Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 MSB

**Figure 20-4. SPI Transfer Format with CPHA = 1**


SCK (CPOL = 0) mode 1
SCK (CPOL = 1) mode 3
SAMPLE I MOSI/MISO
CHANGE 0 MOSI PIN
CHANGE 0 MISO PIN
SS
MSB first (DORD = 0) MSB Bit 6 Bit 5 Bit 4 Bit 3 Bit 2 Bit 1 LSB
LSB first (DORD = 1) LSB Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 MSB

## 20.5. Register Description

### 20.5.3. SPI Data Register 0

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

The device is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

**Name:** SPDR0
**Offset:** 0x4E
**Reset:** 0xXX
**Property:** When addressing as I/O Register: address offset is 0x2E

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | SPID[7:0] | | | | | | | |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | x | x | x | x | x | x | x | x |

**Bits 7:0 – SPID[7:0]: SPI Data**
The SPI Data Register is a read/write register used for data transfer between the Register File and the SPI Shift Register. Writing to the register initiates data transmission. Reading the register causes the Shift Register Receive buffer to be read.

About Code Examples on page 20

## 21.7.2. 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 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.

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**

```
USART_Transmit:
   ; Wait for empty transmit buffer
   in      r18, UCSR0A
   sbrs    r18, UDRE
   rjmp    USART_Transmit
   ; Copy 9th bit from r17 to TXB8
   cbi     UCSR0B,TXB8
   sbrc    r17,0
   sbi     UCSR0B,TXB8
   ; Put LSB data (r16) into buffer, sends the data
   out    UDR0,r16
   ret
```

**C Code Example**

```
void USART_Transmit( unsigned int data )
{
   /* Wait for empty transmit buffer */
   while ( !( UCSR0A & (1<<UDRE))) )
      ;
   /* Copy 9th bit to TXB8 */
   UCSR0B &= ~(1<<TXB8);
   if ( data & 0x0100 )
      UCSR0B |= (1<<TXB8);
   /* Put data into buffer, sends the data */
   UDR0 = data;
}
```

**Note:** 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.

**Related Links**

About Code Examples on page 20

## 21.7.3. Transmitter Flags and Interrupts

The USART Transmitter has two flags that indicate its state: USART Data Register Empty (UDRE) and Transmit Complete (TXC). Both flags can be used for generating interrupts.

The Data Register Empty (UDRE) 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 (UDRIE) bit in UCSRnB is written to '1', the USART Data Register Empty Interrupt will be executed as long as UDRE is set (provided that global interrupts are enabled). UDRE 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 UDRE or disable the Data Register Empty interrupt - otherwise, a new interrupt will occur once the interrupt routine terminates.

The Transmit Complete (TXC) Flag bit is set 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 TXC Flag bit is either automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a '1' to its bit location. The TXC Flag is useful in half-duplex communication interfaces (like the RS-485 standard), where a transmitting application must enter receive mode and free the communication bus immediately after completing the transmission.

When the Transmit Compete Interrupt Enable (TXCIE) bit in UCSRnB is written to '1', the USART Transmit Complete Interrupt will be executed when the TXC Flag becomes set (provided that global interrupts are enabled). When the transmit complete interrupt is used, the interrupt handling routine does not have to clear the TXC Flag, this is done automatically when the interrupt is executed.

### 21.7.4. Parity Generator

The Parity Generator calculates the parity bit for the serial frame data. When parity bit is enabled (UCSRnC.UPM[1]=1), the transmitter control logic inserts the parity bit between the last data bit and the first stop bit of the frame that is sent.

### 21.7.5. Disabling the Transmitter

When writing the TX Enable bit in the USART Control and Status Register n B (UCSRnB.TXEN) to zero, the disabling of the Transmitter will not become effective until ongoing and pending transmissions are completed, i.e., when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted. When disabled, the Transmitter will no longer override the TxDn pin.

## 21.8. Data Reception – The USART Receiver

The USART Receiver is enabled by writing the Receive Enable (RXEN) bit in the UCSRnB Register to '1'. When the Receiver is enabled, the normal pin operation of the RxDn pin is overridden by the USART and given the function as the Receiver's serial input. The baud rate, mode of operation and frame format must be set up once before any serial reception can be done. If synchronous operation is used, the clock on the XCKn pin will be used as transfer clock.

### 21.8.1. Receiving Frames with 5 to 8 Data Bits

The Receiver starts data reception when it detects a valid start bit. Each bit that follows the start bit will be sampled at the baud rate or XCKn clock, and shifted into the Receive Shift Register until the first stop bit of a frame is received. A second stop bit will be ignored by the Receiver. When the first stop bit is received, i.e., a complete serial frame is present in the Receive Shift Register, the contents of the Shift Register will be moved into the receive buffer. The receive buffer can then be read by reading the UDRn I/O location.

The following code example shows a simple USART receive function based on polling of the Receive Complete (RXC) Flag. When using frames with less than eight bits the most significant bits of the data read from the UDR0 will be masked to zero. The USART 0 has to be initialized before the function can be used. For the assembly code, the received data will be stored in R16 after the code completes.

**Assembly Code Example**

```
USART_Receive:
   ; Wait for data to be received
   in    r17, UCSR0A
   sbrs  r17, RXC
```

## 21.12.2. USART Control and Status Register n A

**Name:** UCSR0A, UCSR1A
**Offset:** 0xC0 + n*0x08 [n=0..1]
**Reset:** 0x20
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RXC | TXC | UDRE | FE | DOR | UPE | U2X | MPCM |
| Access | R | R/W | R | R | R | R | R/W | R/W |
| Reset | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

**Bit 7 – RXC: USART Receive Complete**
This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). If the Receiver is disabled, the receive buffer will be flushed and consequently the RXC bit will become zero. The RXC Flag can be used to generate a Receive Complete interrupt (see description of the RXCIE bit).

**Bit 6 – TXC: USART Transmit Complete**
This flag bit is set 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 (UDRn). The TXC 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. The TXC Flag can generate a Transmit Complete interrupt (see description of the TXCIE bit).

**Bit 5 – UDRE: USART Data Register Empty**
The UDRE Flag indicates if the transmit buffer (UDRn) is ready to receive new data. If UDRE is one, the buffer is empty, and therefore ready to be written. The UDRE Flag can generate a Data Register Empty interrupt (see description of the UDRIE bit). UDRE is set after a reset to indicate that the Transmitter is ready.

**Bit 4 – FE: Frame Error**
This bit is set if the next character in the receive buffer had a Frame Error when received. I.e., when the first stop bit of the next character in the receive buffer is zero. This bit is valid until the receive buffer (UDRn) is read. The FEn bit is zero when the stop bit of received data is one. Always set this bit to zero when writing to UCSRnA.

This bit is reserved in Master SPI Mode (MSPIM).

**Bit 3 – DOR: Data OverRun**
This bit is set if a Data OverRun condition is detected. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the Receive Shift Register, and a new start bit is detected. This bit is valid until the receive buffer (UDRn) is read. Always set this bit to zero when writing to UCSRnA.

This bit is reserved in Master SPI Mode (MSPIM).

**Bit 2 – UPE: USART Parity Error**
This bit is set if the next character in the receive buffer had a Parity Error when received and the Parity Checking was enabled at that point (UCSRnC.UPM1 = 1). This bit is valid until the receive buffer (UDRn) is read. Always set this bit to zero when writing to UCSRnA.

This bit is reserved in Master SPI Mode (MSPIM).

**Table 23-1. TWI Terminology**

| Term | Description |
|---|---|
| Master | The device that initiates and terminates a transmission. The Master also generates the SCL clock. |
| Slave | The device addressed by a Master. |
| Transmitter | The device placing data on the bus. |
| Receiver | The device reading data from the bus. |

This device has one instance of TWI. For this reason, the instance index *n* is omitted.

The Power Reduction TWI bit in the Power Reduction Register (PRRn.PRTWI) must be written to '0' to enable the two-wire Serial Interface.

TWI0 is in 0.

**Related Links**

Power Management and Sleep Modes on page 57

### 23.2.2. Electrical Interconnection

As depicted in the TWI Bus Definition, both bus lines are connected to the positive supply voltage through pull-up resistors. The bus drivers of all TWI-compliant devices are open-drain or open-collector. This implements a wired-AND function which is essential to the operation of the interface. A low level on a TWI bus line is generated when one or more TWI devices output a zero. A high level is output when all TWI devices tri-state their outputs, allowing the pull-up resistors to pull the line high. Note that all AVR devices connected to the TWI bus must be powered in order to allow any bus operation.

The number of devices that can be connected to the bus is only limited by the bus capacitance limit of 400pF and the 7-bit slave address space. Two different sets of specifications are presented there, one relevant for bus speeds below 100kHz, and one valid for bus speeds up to 400kHz.

## 23.3. Data Transfer and Frame Format

### 23.3.1. Transferring Bits

Each data bit transferred on the TWI bus is accompanied by a pulse on the clock line. The level of the data line must be stable when the clock line is high. The only exception to this rule is for generating start and stop conditions.

**Figure 23-2. Data Validity**

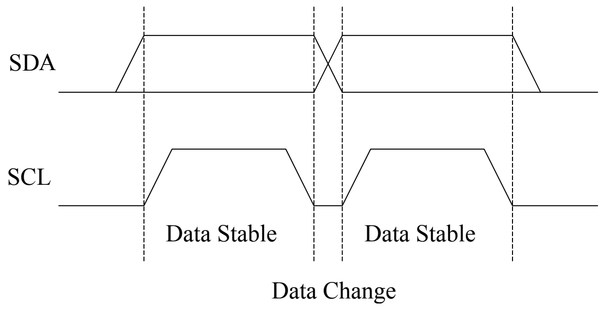
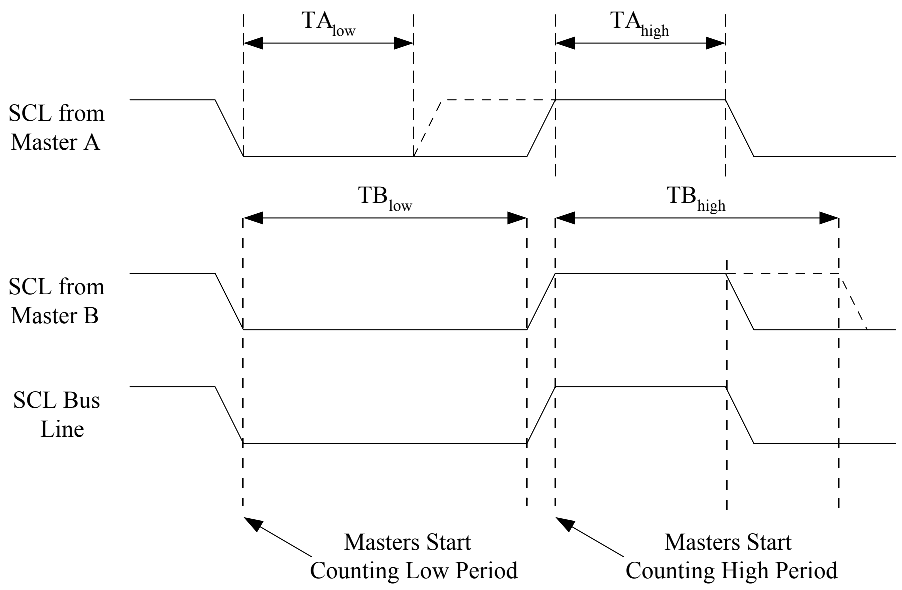
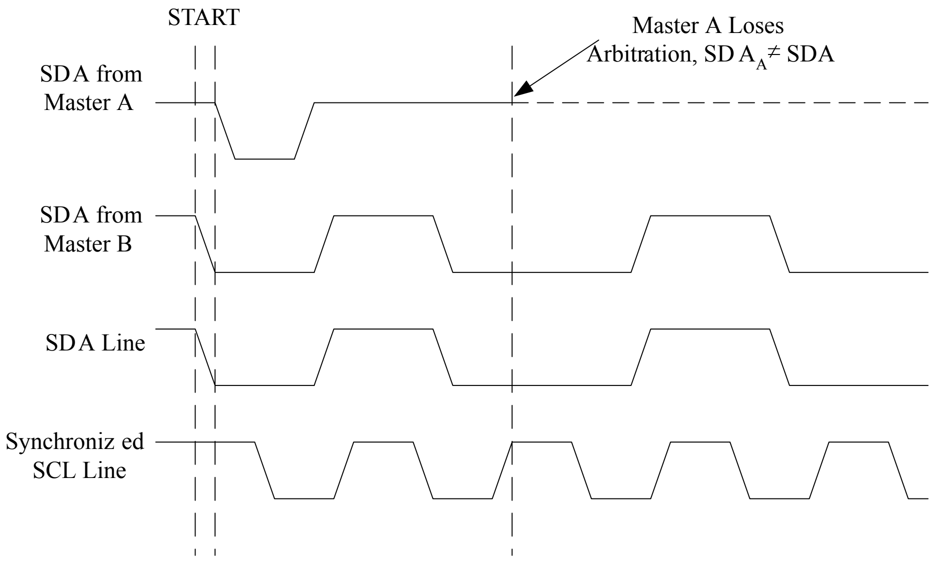


**Figure 23-7. SCL Synchronization Between Multiple Masters**



Arbitration is carried out by all masters continuously monitoring the SDA line after outputting data. If the value read from the SDA line does not match the value the Master had output, it has lost the arbitration. Note that a Master can only lose arbitration when it outputs a high SDA value while another Master outputs a low value. The losing Master should immediately go to Slave mode, checking if it is being addressed by the winning Master. The SDA line should be left high, but losing masters are allowed to generate a clock signal until the end of the current data or address packet. Arbitration will continue until only one Master remains, and this may take many bits. If several masters are trying to address the same Slave, arbitration will continue into the data packet.

**Figure 23-8. Arbitration Between Two Masters**



Note that arbitration is not allowed between:

- A REPEATED START condition and a data bit
- A STOP condition and a data bit
- A REPEATED START and a STOP condition

It is the user software's responsibility to ensure that these illegal arbitration conditions never occur. This implies that in multi-master systems, all data transfers must use the same composition of SLA+R/W and

TWINT bit in TWCRn is set. Immediately after the application has cleared TWINT, the TWI n will initiate transmission of the START condition.

2. When the START condition has been transmitted, the TWINT Flag in TWCRn is set, and TWSRn is updated with a status code indicating that the START condition has successfully been sent.
3. The application software should now examine the value of TWSRn, to make sure that the START condition was successfully transmitted. If TWSRn indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load SLA+W into TWDR. Remember that TWDRn is used both for address and data. After TWDRn has been loaded with the desired SLA+W, a specific value must be written to TWCRn, instructing the TWI n hardware to transmit the SLA+W present in TWDRn. 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 TWCRn is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the address packet.
4. When the address packet has been transmitted, the TWINT Flag in TWCRn is set, and TWSRn is updated with a status code indicating that the address packet has successfully been sent. The status code will also reflect whether a Slave acknowledged the packet or not.
5. The application software should now examine the value of TWSRn, to make sure that the address packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSRn 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 TWDRn. Subsequently, a specific value must be written to TWCRn, instructing the TWI n hardware to transmit the data packet present in TWDRn. 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 n will not start any operation as long as the TWINT bit in TWCRn 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 TWCRn is set, and TWSRn 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 TWSRn, 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 TWCRn, instructing the TWI n 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 n will not start any operation as long as the TWINT bit in TWCRn 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 n Registers with the value relevant for the next TWI n bus cycle. As an example, TWDRn must be loaded with the value to be transmitted in the next bus cycle.
- After all TWI n Register updates and other pending application software tasks have been completed, TWCRn is written. When writing TWCRn, the TWINT bit should be set. Writing a one to

# 25. ADC - Analog to Digital Converter

## 25.1. Features

- 10-bit Resolution
- 0.5 LSB Integral Non-Linearity
- ±2 LSB Absolute Accuracy
- 13 - 260µs Conversion Time
- Up to 15kSPS at Maximum Resolution
- 8 Multiplexed Single Ended Input Channels
- Differential mode with selectable gain at 1x, 10x or 200x(1)
- Optional Left Adjustment for ADC Result Readout
- 0 - $V_{CC}$ ADC Input Voltage Range
- 2.7V - $V_{CC}$ Differential ADC Voltage Range
- Selectable 2.56V or 1.1V ADC Reference Voltage
- Free Running or Single Conversion Mode
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceler

**Note:**

1. The differential input channels are not tested for devices in PDIP Package. This feature is only guaranteed to work for devices in TQFP and VQFN/QFN/MLF Packages.

## 25.2. Overview

The device features a 10-bit successive approximation ADC. The ADC is connected to an 8-channel Analog Multiplexer which allows 8 single-ended voltage inputs constructed from the pins of Port A. The single-ended voltage inputs refer to 0V (GND).

The device also supports 16 differential voltage input combinations. Two of the differential inputs (ADC1, ADC0 and ADC3, ADC2) are equipped with a programmable gain stage. This provides amplification steps of 0 dB (1x), 20 dB (10x), or 46 dB (200x) on the differential input voltage before the A/D conversion. Seven differential analog input channels share a common negative terminal (ADC1), while any other ADC input can be selected as the positive input terminal. If 1x or 10x gain is used, 8-bit resolution can be expected. If 200x gain is used, 6- bit resolution can be expected. Note that internal references of 1.1V should not be used on 10x and 200x gain.

The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion. A block diagram of the ADC is shown below.

The ADC has a separate analog supply voltage pin, $AV_{CC}$. $AV_{CC}$ must not differ more than ±0.3V from $V_{CC}$. See section ADC Noise Canceler on how to connect this pin.

The Power Reduction ADC bit in the Power Reduction Register (PRR.PRADC) must be written to '0' in order to be enable the ADC.

The ADC converts an analog input voltage to a 10-bit digital value through successive approximation. The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. Optionally, $AV_{CC}$ or an internal 2.56V reference voltage may be connected to the AREF pin by

## 25.8.5. ADC Control and Status Register B

**Name:** ADCSRB
**Offset:** 0x7B
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | ACME | | | | ADTS2 | ADTS1 | ADTS0 |
| Access | | R/W | | | | R/W | R/W | R/W |
| Reset | | 0 | | | | 0 | 0 | 0 |

**Bit 6 – ACME: Analog Comparator Multiplexer Enable**
When this bit is written logic one and the ADC is switched off (ADEN in ADCSRA is zero), the ADC multiplexer selects the negative input to the Analog Comparator. When this bit is written logic zero, AIN1 is applied to the negative input of the Analog Comparator. For a detailed description of this bit, see *Analog Comparator Multiplexed Input*..

**Bits 2:0 – ADTSn: ADC Auto Trigger Source [n = 2:0]**
If ADATE in ADCSRA is written to one, the value of these bits selects which source will trigger an ADC conversion. If ADATE is cleared, the ADTS[2:0] settings will have no effect. A conversion will be triggered by the rising edge of the selected Interrupt Flag. Note that switching from a trigger source that is cleared to a trigger source that is set, will generate a positive edge on the trigger signal. If ADEN in ADCSRA is set, this will start a conversion. Switching to Free Running mode (ADTS[2:0]=0) will not cause a trigger event, even if the ADC Interrupt Flag is set.

**Table 25-6. ADC Auto Trigger Source Selection**

| ADTS[2:0] | Trigger Source |
|---|---|
| 000 | Free Running mode |
| 001 | Analog Comparator |
| 010 | External Interrupt Request 0 |
| 011 | Timer/Counter0 Compare Match A |
| 100 | Timer/Counter0 Overflow |
| 101 | Timer/Counter1 Compare Match B |
| 110 | Timer/Counter1 Overflow |
| 111 | Timer/Counter1 Capture Event |

1. Write the Interrupt Vector Change Enable (IVCE) bit to one.
2. Within four cycles, write the desired value to IVSEL while writing a zero to IVCE.

Interrupts will automatically be disabled while this sequence is executed. Interrupts are disabled in the cycle IVCE is set, and they remain disabled until after the instruction following the write to IVSEL. If IVSEL is not written, interrupts remain disabled for four cycles. The I-bit in the Status Register is unaffected by the automatic disabling.

**Note:** If Interrupt Vectors are placed in the Boot Loader section and Boot Lock bit BLB02 is programmed, interrupts are disabled while executing from the Application section. If Interrupt Vectors are placed in the Application section and Boot Lock bit BLB12 is programed, interrupts are disabled while executing from the Boot Loader section.

**Bit 0 – IVCE: Interrupt Vector Change Enable**
The IVCE bit must be written to logic one to enable change of the IVSEL bit. IVCE is cleared by hardware four cycles after it is written or when IVSEL is written. Setting the IVCE bit will disable interrupts, as explained in the IVSEL description above. See Code Example below.

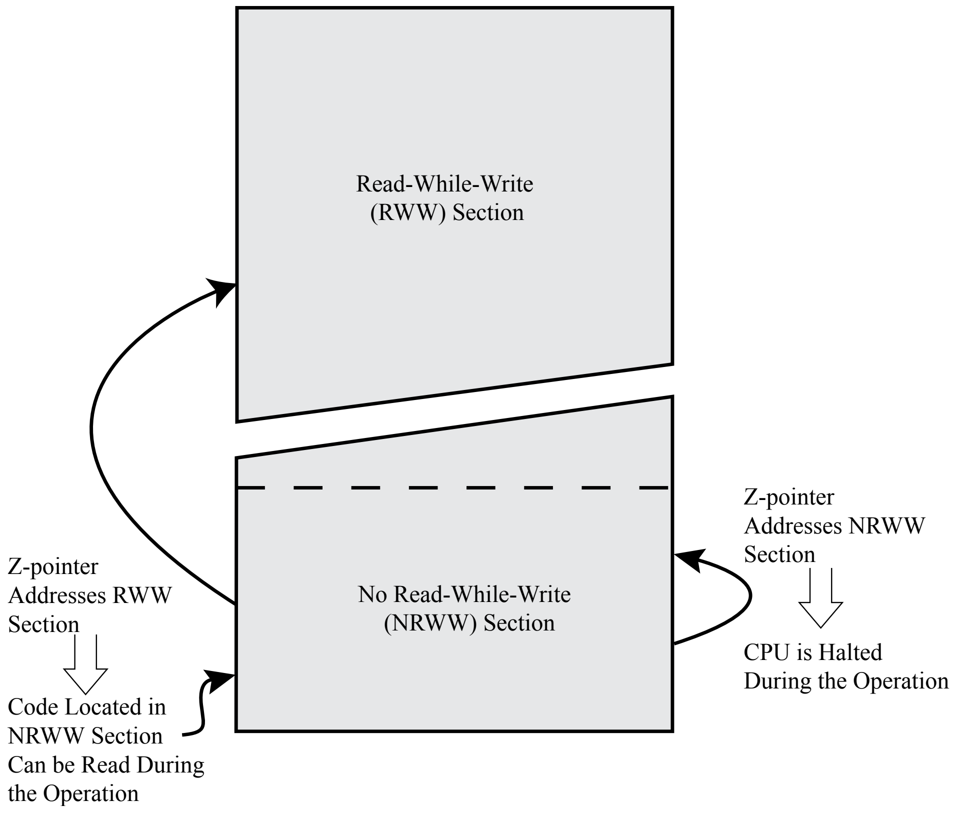
Assembly Code Example

```
Move_interrupts:
; Get MCUCR
in     r16, MCUCR
mov    r17, r16
; Enable change of Interrupt Vectors
ori    r16, (1<<IVCE)
out    MCUCR, r16
; Move interrupts to Boot Flash section
ori    r17, (1<<IVSEL)
out    MCUCR, r17
ret
```

C Code Example

```
void Move_interrupts(void)
{
uchar temp;
/* GET MCUCR*/
temp = MCUCR;
/* Enable change of Interrupt Vectors */
MCUCR = temp|(1<<IVCE);
/* Move interrupts to Boot Flash section */
MCUCR = temp|(1<<IVSEL);
}
```

**Figure 27-1. Read-While-Write vs. No Read-While-Write**



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

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

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Rd | – | – | – | – | – | EFB2 | EFB1 | EFB0 |

Fuse and Lock bits that are programmed read as '0'. Fuse and Lock bits that are unprogrammed, will read as '1'.

**Related Links**

Fuse Bits on page 365

## 27.8.10. Reading the Signature Row from Software

To read the Signature Row from software, load the Z-pointer with the signature byte address given in the following table and set the SIGRD and SPMEN bits in SPMCSR (SPMCSR.SIGRD and SPMCSR.SPMEN). When an LPM instruction is executed within three CPU cycles after the SPMCSR.SIGRD and SPMCSR.SPMEN are set, the signature byte value will be loaded in the destination register. The SPMCSR.SIGRD and SPMCSR.SPMEN will auto-clear upon completion of reading the Signature Row Lock bits or if no LPM instruction is executed within three CPU cycles. When SPMCSR.SIGRD and SPMCSR.SPMEN are cleared, LPM will work as described in the Instruction set Manual.

**Table 27-5. Signature Row Addressing**

| Signature Byte | Z-pointer Address |
|---|---|
| Device Signature Byte 1 | 0x0000 |
| Device Signature Byte 2 | 0x0002 |
| Device Signature Byte 3 | 0x0004 |
| RC Oscillator Calibration Byte | 0x0001 |
| Serial Number Byte 1 | 0x000E |
| Serial Number Byte 0 | 0x000F |
| Serial Number Byte 3 | 0x0010 |
| Serial Number Byte 2 | 0x0011 |
| Serial Number Byte 5 | 0x0012 |
| Serial Number Byte 4 | 0x0013 |
| Serial Number Byte 6 | 0x0015 |
| Serial Number Byte 7 | 0x0016 |
| Serial Number Byte 8 | 0x0017 |

**Note:** All other addresses are reserved for future use.

# 28. MEMPROG- Memory Programming

## 28.1. Program And Data Memory Lock Bits

The devices provides Lock bits. These can be left unprogrammed ('1') or can be programmed ('0') to obtain the additional features listed in Table. Lock Bit Protection Modes in this section. The Lock bits can only be erased to “1” with the Chip Erase command.

**Table 28-1. Lock Bit Byte**[(1)]

| Lock Bit Byte | Bit No. | Description | Default Value |
|---|---|---|---|
| | 7 | – | 1 (unprogrammed) |
| | 6 | – | 1 (unprogrammed) |
| BLB12 | 5 | Boot Lock bit | 1 (unprogrammed) |
| BLB11 | 4 | Boot Lock bit | 1 (unprogrammed) |
| BLB02 | 3 | Boot Lock bit | 1 (unprogrammed) |
| BLB01 | 2 | Boot Lock bit | 1 (unprogrammed) |
| LB2 | 1 | Lock bit | 1 (unprogrammed) |
| LB1 | 0 | Lock bit | 1 (unprogrammed) |

**Note:**

1. '1' means unprogrammed, '0' means programmed.

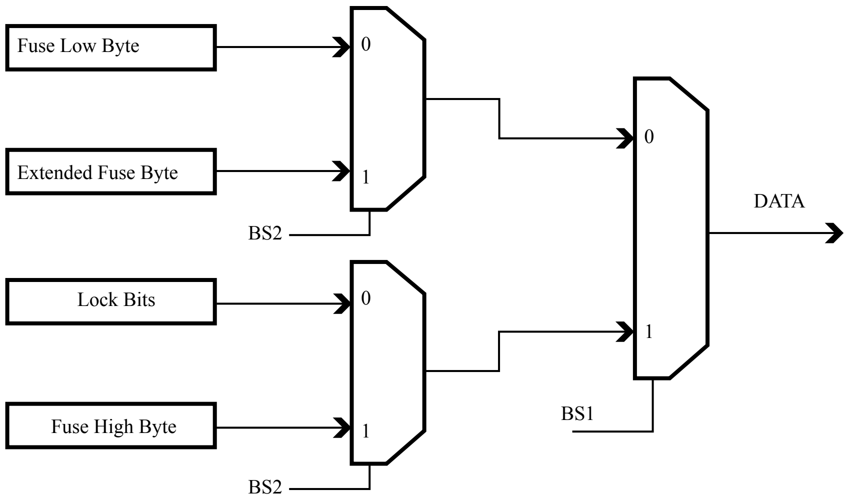
**Table 28-2. Lock Bit Protection Modes**[(1)(2)]

| Memory Lock Bits | | | Protection Type |
|---|---|---|---|
| LB Mode | LB2 | LB1 | |
| 1 | 1 | 1 | No memory lock features enabled. |
| 2 | 1 | 0 | Further programming of the Flash and EEPROM is disabled in Parallel and Serial Programming mode. The Fuse bits are locked in both Serial and Parallel Programming mode.(1) |
| 3 | 0 | 0 | Further programming and verification of the Flash and EEPROM is disabled in Parallel and Serial Programming mode. The Boot Lock bits and Fuse bits are locked in both Serial and Parallel Programming mode.(1) |

**Note:**

1. Program the Fuse bits and Boot Lock bits before programming the LB1 and LB2.
2. '1' means unprogrammed, '0' means programmed.

**Figure 28-5. Mapping Between BS1, BS2 and the Fuse and Lock Bits During Read**



### 28.8.13. Reading the Signature Bytes

The algorithm for reading the Signature bytes is as follows (Please refer to Programming the Flash for details on Command and Address loading):

1. Step A: Load Command “0000 1000”.
2. Step B: Load Address Low Byte (0x00 - 0x02).
3. Set $\overline{\text{OE}}$ to “0”, and BS1 to “0”. The selected Signature byte can now be read at DATA.
4. Set $\overline{\text{OE}}$ to “1”.

### 28.8.14. Reading the Calibration Byte

The algorithm for reading the Calibration byte is as follows (Please refer to Programming the Flash for details on Command and Address loading):

1. Step A: Load Command “0000 1000”.
2. Step B: Load Address Low Byte, 0x00.
3. Set $\overline{\text{OE}}$ to “0”, and BS1 to “1”. The Calibration byte can now be read at DATA.
4. Set $\overline{\text{OE}}$ to “1”.

### 28.8.15. Parallel Programming Characteristics

For characteristics of the Parallel Programming, please refer to *Parallel Programming Characteristics*.

## 28.9. Serial Downloading

Both the Flash and EEPROM memory arrays can be programmed using the serial SPI bus while $\overline{\text{RESET}}$ is pulled to GND. The serial interface consists of pins SCK, MOSI (input) and MISO (output). After $\overline{\text{RESET}}$ is set low, the Programming Enable instruction needs to be executed first before program/erase operations can be executed.

Figure 30-16. I/O Pin Pull-up Resistor Current vs. Input Voltage ($V_{CC}$ = 2.7V)

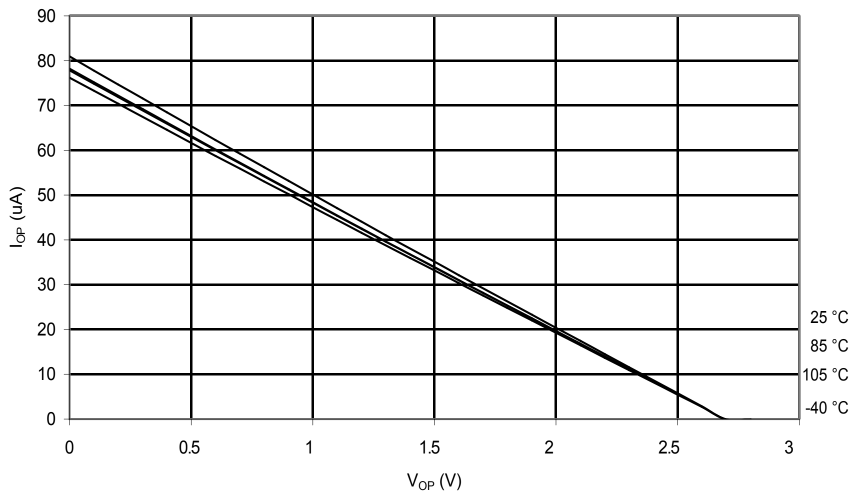


Figure 30-17. I/O Pin Pull-up Resistor Current vs. Input Voltage ($V_{CC}$ = 5V)

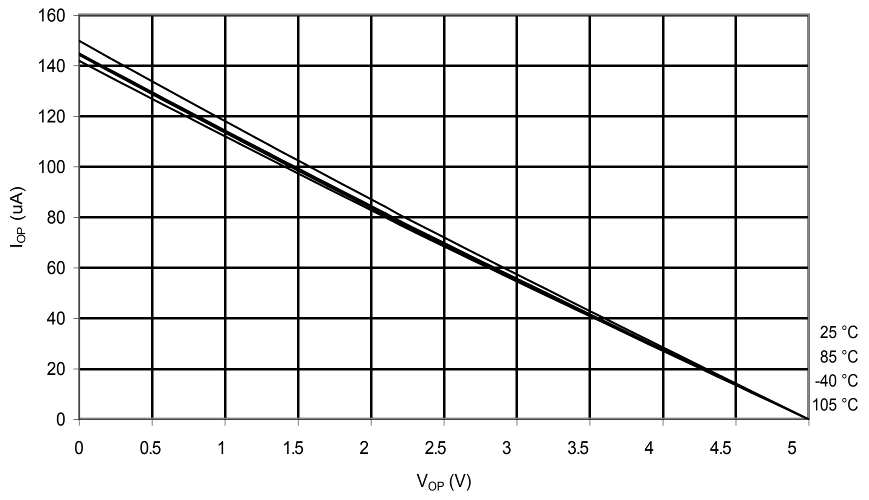
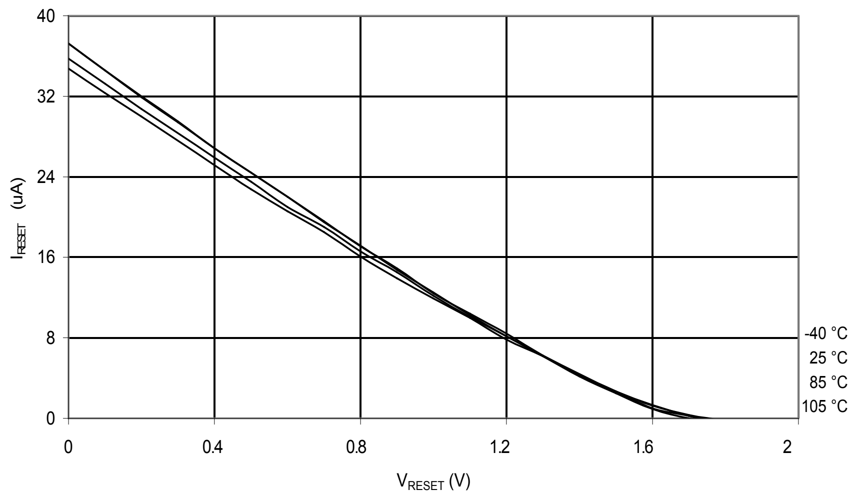


Figure 30-18. Reset Pull-up Resistor Current vs. Reset Pin Voltage ($V_{CC}$ = 1.8V)


$I_{RESET}$ (uA)
$V_{RESET}$ (V)
-40 °C
25 °C
85 °C
105 °C