Data Modes

There are four combinations of SCK phase and polarity with respect to serial data, which are determined by control bits CPHA and CPOL. The SPI data transfer formats are shown in Figure 67 and Figure 68. Data bits are shifted out and latched in on opposite edges of the SCK signal, ensuring sufficient time for data signals to stabilize. This is clearly seen by summarizing Table 67 and Table 68, as done below:

Table 70. CPOL Functionality

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

Figure 67. SPI Transfer Format with CPHA = 0

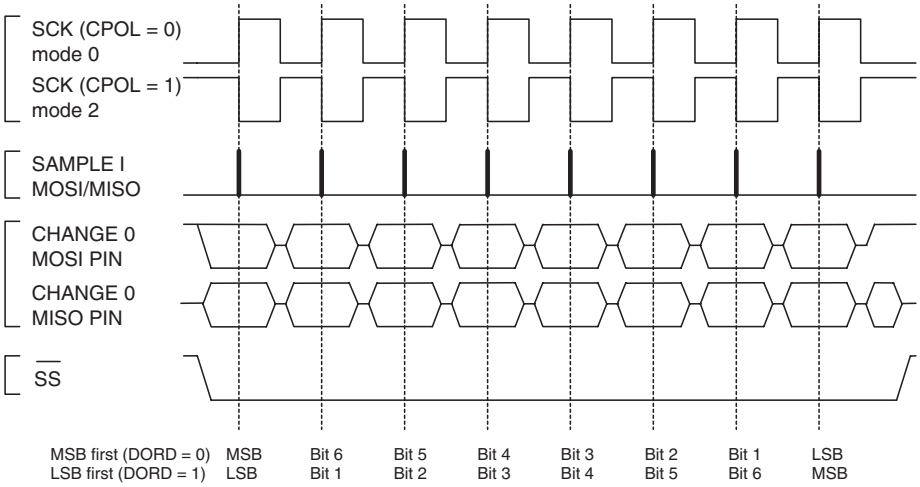
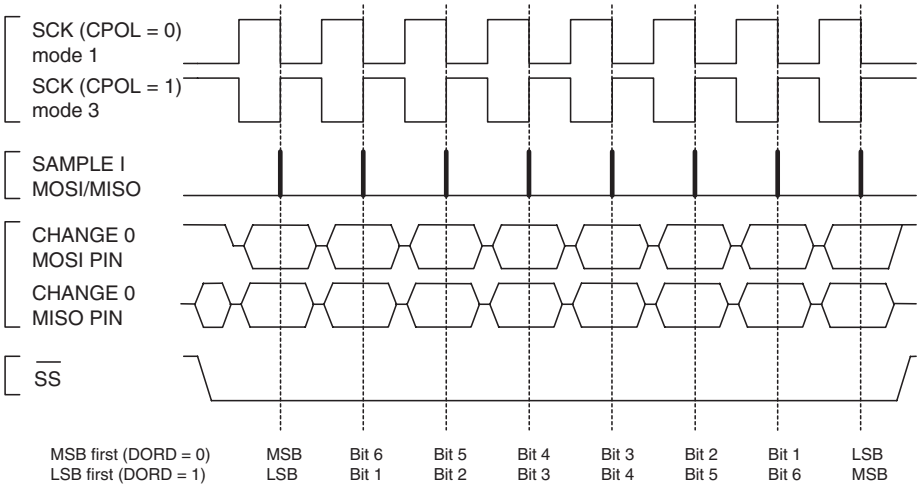


Figure 68. SPI Transfer Format with CPHA = 1



USART

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device. The main features are:

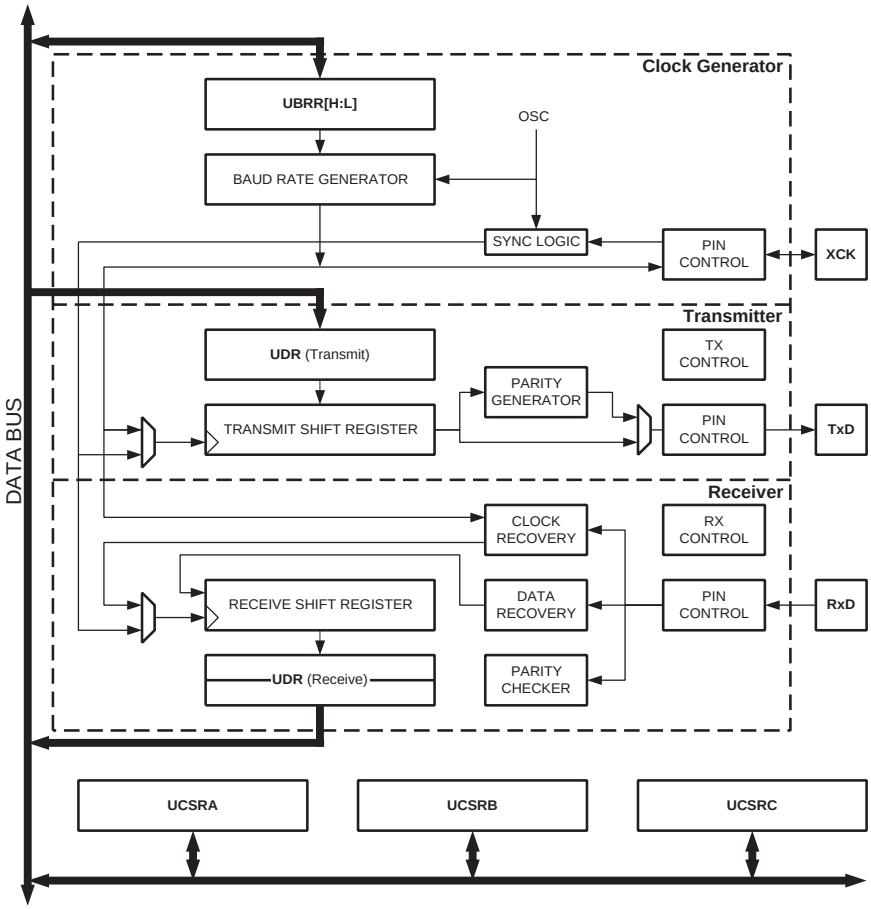
- **Full Duplex Operation (Independent Serial Receive and Transmit Registers)**
- **Asynchronous or Synchronous Operation**
- **Master or Slave Clocked Synchronous Operation**
- **High Resolution Baud Rate Generator**
- **Supports Serial Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Stop Bits**
- **Odd or Even Parity Generation and Parity Check Supported by Hardware**
- **Data OverRun Detection**
- **Framing Error Detection**
- **Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter**
- **Three Separate Interrupts on TX Complete, TX Data Register Empty and RX Complete**
- **Multi-processor Communication Mode**
- **Double Speed Asynchronous Communication Mode**

The PRUSART0 bit in “Power Reduction Register - PRR” on page 34 must be written to zero to enable USART module.

Overview

A simplified block diagram of the USART Transmitter is shown in Figure 69. CPU accessible I/O Registers and I/O pins are shown in bold.

Figure 69. USART Block Diagram(1)



Note: 1. Refer to Figure 1 on page 2, Table 37 on page 69, and Table 31 on page 65 for USART pin placement.

The dashed boxes in the block diagram separate the three main parts of the USART (listed from the top): Clock Generator, Transmitter and Receiver. Control Registers are shared by all units. The Clock Generation logic consists of synchronization logic for external clock input used by synchronous slave operation, and the baud rate generator. The XCK (Transfer Clock) pin is only used by synchronous transfer mode. The Transmitter consists of a single write buffer, a serial Shift Register, Parity Generator and Control logic for handling different serial frame formats. The write buffer allows a continuous transfer of data without any delay between frames. The Receiver is the most complex part of the USART module due to its clock and data recovery units. The recovery units are used for asynchronous data reception. In addition to the recovery units, the Receiver includes a Parity Checker, Control logic, a Shift Register and a two level receive buffer (UDR). The Receiver supports the same frame formats as the Transmitter, and can detect Frame Error, Data OverRun and Parity Errors.

AVR USART vs. AVR UART – Compatibility

The USART is fully compatible with the AVR UART regarding:

- Bit locations inside all USART Registers.
- Baud Rate Generation.
- Transmitter Operation.
- Transmit Buffer Functionality.
- Receiver Operation.

However, the receive buffering has two improvements that will affect the compatibility in some special cases:

- A second Buffer Register has been added. The two Buffer Registers operate as a circular FIFO buffer. Therefore the UDR must only be read once for each incoming data! More important is the fact that the Error Flags (FE and DOR) and the ninth data bit (RXB8) are buffered with the data in the receive buffer. Therefore the status bits must always be read before the UDR Register is read. Otherwise the error status will be lost since the buffer state is lost.
- The Receiver Shift Register can now act as a third buffer level. This is done by allowing the received data to remain in the serial Shift Register (see Figure 69) if the Buffer Registers are full, until a new start bit is detected. The USART is therefore more resistant to Data OverRun (DOR) error conditions.

The following control bits have changed name, but have same functionality and register location:

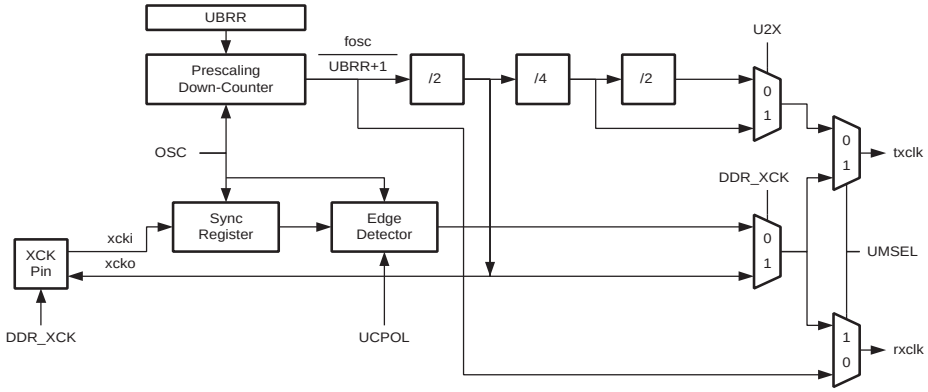
- CHR9 is changed to UCSZ2.
- OR is changed to DOR.

Clock Generation

The Clock Generation logic generates the base clock for the Transmitter and Receiver. The USART supports four modes of clock operation: Normal asynchronous, Double Speed asynchronous, Master synchronous and Slave synchronous mode. The UMSEL bit in USART Control and Status Register C (UCSRC) selects between asynchronous and synchronous operation. Double Speed (asynchronous mode only) is controlled by the U2X found in the UCSRA Register. When using synchronous mode (UMSEL = 1), the Data Direction Register for the XCK pin (DDR_XCK) controls whether the clock source is internal (Master mode) or external (Slave mode). The XCK pin is only active when using synchronous mode.

Figure 70 shows a block diagram of the clock generation logic.

Figure 70. Clock Generation Logic, Block Diagram



Signal description:

- **txclk** Transmitter clock (Internal Signal).
- **rxclk** Receiver base clock (Internal Signal).
- **xcki** Input from XCK pin (internal Signal). Used for synchronous slave operation.
- **xcko** Clock output to XCK pin (Internal Signal). Used for synchronous master operation.
- **fosc** XTAL pin frequency (System Clock).

Internal Clock Generation – The Baud Rate Generator

Internal clock generation is used for the asynchronous and the synchronous master modes of operation. The description in this section refers to Figure 70.

The USART Baud Rate Register (UBRR) and the down-counter connected to it function as a programmable prescaler or baud rate generator. The down-counter, running at system clock (f_{osc}), is loaded with the UBRR value each time the counter has counted down to zero or when the UBRRL Register is written. A clock is generated each time the counter reaches zero. This clock is the baud rate generator clock output (= $f_{osc}/(UBRR+1)$). The Transmitter divides the baud rate generator clock output by 2, 8 or 16 depending on mode. The baud rate generator output is used directly by the Receiver's clock and data recovery units. However, the recovery units use a state machine that uses 2, 8 or 16 states depending on mode set by the state of the UMSEL, U2X and DDR_XCK bits.

Table 71 contains equations for calculating the baud rate (in bits per second) and for calculating the UBRR value for each mode of operation using an internally generated clock source.

Table 71. Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(\boldsymbol{UBRR}+1)}$	$\boldsymbol{UBRR} = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(\boldsymbol{UBRR}+1)}$	$\boldsymbol{UBRR} = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master mode	$BAUD = \frac{f_{OSC}}{2(\boldsymbol{UBRR}+1)}$	$\boldsymbol{UBRR} = \frac{f_{OSC}}{2BAUD} - 1$

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps)

BAUD Baud rate (in bits per second, bps)

f_{osc} System Oscillator clock frequency

UBRR Contents of the UBRRH and UBRRL Registers, (0-4095)

Some examples of UBRR values for some system clock frequencies are found in Table 79 (see page 175).

Double Speed Operation (U2X)

The transfer rate can be doubled by setting the U2X bit in UCSRA. Setting this bit only has effect for the asynchronous operation. Set this bit to zero when using synchronous operation.

Setting this bit will reduce the divisor of the baud rate divider from 16 to 8, effectively doubling the transfer rate for asynchronous communication. Note however that the Receiver will in this case only use half the number of samples (reduced from 16 to 8) for data sampling and clock recovery, and therefore a more accurate baud rate setting and system clock are required when this mode is used. For the Transmitter, there are no downsides.

External Clock

External clocking is used by the synchronous slave modes of operation. The description in this section refers to Figure 70 for details.

External clock input from the XCK pin is sampled by a synchronization register to minimize the chance of meta-stability. The output from the synchronization register must then pass through an edge detector before it can be used by the Transmitter and Receiver. This process introduces a two CPU clock period delay and therefore the maximum external XCK clock frequency is limited by the following equation:

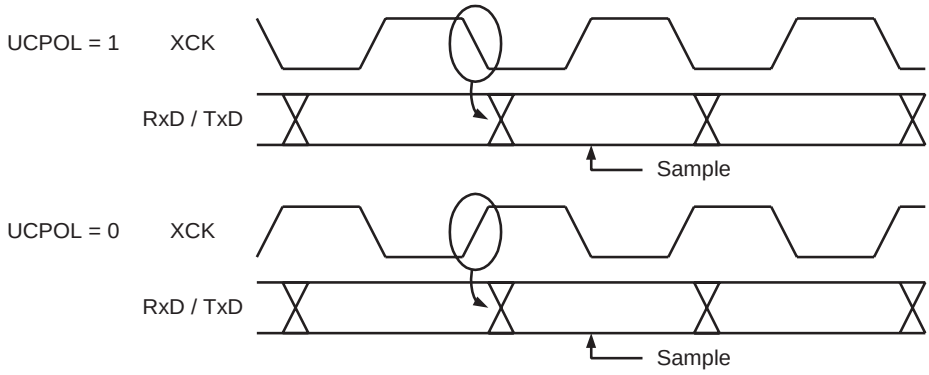
$$f_{XCK} < \frac{f_{OSC}}{4}$$

Note that f_{osc} depends on the stability of the system clock source. It is therefore recommended to add some margin to avoid possible loss of data due to frequency variations.

Synchronous Clock Operation

When synchronous mode is used (UMSEL = 1), the XCK pin will be used as either clock input (Slave) or clock output (Master). The dependency between the clock edges and data sampling or data change is the same. The basic principle is that data input (on RxD) is sampled at the opposite XCK clock edge of the edge the data output (TxD) is changed.

Figure 71. Synchronous Mode XCK Timing.



The UCPOL bit UCRSC selects which XCK clock edge is used for data sampling and which is used for data change. As Figure 71 shows, when UCPOL is zero the data will be changed at rising XCK edge and sampled at falling XCK edge. If UCPOL is set, the data will be changed at falling XCK edge and sampled at rising XCK edge.

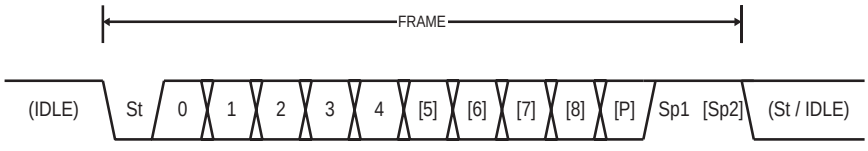
Frame Formats

A serial frame is defined to be one character of data bits with synchronization bits (start and stop bits), and optionally a parity bit for error checking. The USART accepts all 30 combinations of the following as valid frame formats:

- 1 start bit
- 5, 6, 7, 8, or 9 data bits
- no, even or odd parity bit
- 1 or 2 stop bits

A frame starts with the start bit followed by the least significant data bit. Then the next data bits, up to a total of nine, are succeeding, ending with the most significant bit. If enabled, the parity bit is inserted after the data bits, before the stop bits. When a complete frame is transmitted, it can be directly followed by a new frame, or the communication line can be set to an idle (high) state. Figure 72 illustrates the possible combinations of the frame formats. Bits inside brackets are optional.

Figure 72. Frame Formats



St Start bit, always low.

(n) Data bits (0 to 8).

P Parity bit. Can be odd or even.

Sp Stop bit, always high.

IDLE No transfers on the communication line (RxD or TxD). An IDLE line must be high.

The frame format used by the USART is set by the UCSZ2:0, UPM1:0 and USBS bits in UCSRB and UCSRC. The Receiver and Transmitter use the same setting. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the Receiver and Transmitter.

The USART Character SiZe (UCSZ2:0) bits select the number of data bits in the frame. The USART Parity mode (UPM1:0) bits enable and set the type of parity bit. The selection between one or two stop bits is done by the USART Stop Bit Select (USBS) bit. The Receiver ignores the second stop bit. An FE (Frame Error) will therefore only be detected in the cases where the first stop bit is zero.

Parity Bit Calculation

The parity bit is calculated by doing an exclusive-or of all the data bits. If odd parity is used, the result of the exclusive or is inverted. The relation between the parity bit and data bits is as follows:

$$P_{even} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

P_{even} Parity bit using even parity

P_{odd} Parity bit using odd parity

d_n Data bit n of the character

If used, the parity bit is located between the last data bit and first stop bit of a serial frame.

USART Initialization

The USART has to be initialized before any communication can take place. The initialization process normally consists of setting the baud rate, setting frame format and enabling the Transmitter or the Receiver depending on the usage. For interrupt driven USART operation, the Global Interrupt Flag should be cleared (and interrupts globally disabled) when doing the initialization.

Before doing a re-initialization with changed baud rate or frame format, be sure that there are no ongoing transmissions during the period the registers are changed. The TXC Flag can be used to check that the Transmitter has completed all transfers, and the RXC Flag can be used to check that there are no unread data in the receive buffer. Note that the TXC Flag must be cleared before each transmission (before UDR is written) if it is used for this purpose.

The following simple USART initialization code examples show one assembly and one C function that are equal in functionality. The examples assume asynchronous operation using polling (no interrupts enabled) and a fixed frame format. The baud rate is given as a function parameter. For the assembly code, the baud rate parameter is assumed to be stored in the r17:r16 Registers.

Assembly Code Example[1]

```
USART_Init:
  ; Set baud rate
  sts   UBRRH, r17
  sts   UBRRL, r16
  ; Enable receiver and transmitter
  ldi   r16, (1<<RXEN)|(1<<TXEN)
  sts   UCSRB,r16
  ; Set frame format: 8data, 2stop bit
  ldi   r16, (1<<USBS)|(3<<UCSZ0)
  sts   UCSRC,r16
  ret
```

C Code Example[1]

```
#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 */
  UBRRH = (unsigned char)(ubrr>>8);
  UBRRL = (unsigned char)ubrr;
  /* Enable receiver and transmitter */
  UCSRB = (1<<RXEN)|(1<<TXEN);
  /* Set frame format: 8data, 2stop bit */
  UCSRC = (1<<USBS)|(3<<UCSZ0);
}
```

Note: 1. See “About Code Examples” on page 6.

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.

Data Transmission – The USART Transmitter

The USART Transmitter is enabled by setting the *Transmit Enable* (TXEN) bit in the UCSRB Register. When the Transmitter is enabled, the normal port operation of the TxD pin is overridden by the USART and given the function as the Transmitter's serial output. The baud rate, mode of operation and frame format must be set up once before doing any transmissions. If synchronous operation is used, the clock on the XCK pin will be overridden and used as transmission clock.

Sending Frames with 5 to 8 Data Bit

A data transmission is initiated by loading the transmit buffer with the data to be transmitted. The CPU can load the transmit buffer by writing to the UDR I/O location. The buffered data in the transmit buffer will be moved to the Shift Register when the Shift Register is ready to send a new frame. The Shift Register is loaded with new data if it is in idle state (no ongoing transmission) or immediately after the last stop bit of the previous frame is transmitted. When the Shift Register is loaded with new data, it will transfer one complete frame at the rate given by the Baud Register, U2X bit or by XCK depending on mode of operation.

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

Assembly Code Example[1]

```
USART_Transmit:
  ; Wait for empty transmit buffer
  sbis  UCSRA,UDRE
  rjmp  USART_Transmit
  ; Put data (r16) into buffer, sends the data
  sts   UDR,r16
  ret
```

C Code Example[1]

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

Note: 1. See "About Code Examples" on page 6.

The function simply waits for the transmit buffer to be empty by checking the UDRE Flag, before loading it with new data to be transmitted. If the Data Register Empty interrupt is utilized, the interrupt routine writes the data into the buffer.

Sending Frames with 9 Data Bit

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

Assembly Code Example[1][2]

```
USART_Transmit:
  ; Wait for empty transmit buffer
  sbis  UCSRA,UDRE
  rjmp  USART_Transmit
  ; Copy 9th bit from r17 to TXB8
  cbi   UCSRB,TXB8
  sbrc  r17,0
  sbi   UCSRB,TXB8
  ; Put LSB data (r16) into buffer, sends the data
  sts   UDR,r16
  ret
```

C Code Example[1][2]

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

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

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.

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 UCSRA Register.

When the Data Register Empty Interrupt Enable (UDRIE) bit in UCSRB is written to one, 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 UDR. When interrupt-driven data transmission is used, the Data Register Empty interrupt routine must either write new data to UDR 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 one when the entire frame in the Transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer. The 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 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 UCSRB is set, 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.

Parity Generator

The Parity Generator calculates the parity bit for the serial frame data. When parity bit is enabled (UPM1 = 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.

Disabling the Transmitter

The disabling of the Transmitter (setting the TXEN to zero) 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 TxD pin.

Data Reception – The USART Receiver

The USART Receiver is enabled by writing the Receive Enable (RXEN) bit in the UCSRB Register to one. When the Receiver is enabled, the normal pin operation of the RxD 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 XCK pin will be used as transfer clock.

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 XCK 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 UDR 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 UDR will be masked to zero. The USART has to be initialized before the function can be used.

Assembly Code Example[1]

```
USART_Receive:
  ; Wait for data to be received
  sbis  UCSRA, RXC
  rjmp  USART_Receive
  ; Get and return received data from buffer
  in    r16, UDR
  ret
```

C Code Example[1]

```
unsigned char USART_Receive( void )
{
  /* Wait for data to be received */
  while ( !(UCSRA & (1<<RXC)) )
        ;
  /* Get and return received data from buffer */
  return UDR;
}
```

Note: 1. See "About Code Examples" on page 6.

The function simply waits for data to be present in the receive buffer by checking the RXC Flag, before reading the buffer and returning the value.

Receiving Frames with 9 Data Bits

If 9-bit characters are used (UCSZ=7) the ninth bit must be read from the RXB8 bit in UCSRB **before** reading the low bits from the UDR. This rule applies to the FE, DOR and UPE Status Flags as well. Read status from UCSRA, then data from UDR. Reading the UDR I/O location will change the state of the receive buffer FIFO and consequently the TXB8, FE, DOR and UPE bits, which all are stored in the FIFO, will change.

The following code example shows a simple USART receive function that handles both nine bit characters and the status bits.

Assembly Code Example[1]

```
USART_Receive:
  ; Wait for data to be received
  sbis  UCSRA, RXC
  rjmp  USART_Receive
  ; Get status and 9th bit, then data from buffer
  in    r18, UCSRA
  in    r17, UCSRB
  in    r16, UDR
  ; If error, return -1
  andi  r18,(1<<FE)|(1<<DOR)|(1<<UPE)
  breq  USART_ReceiveNoError
  ldi   r17, HIGH(-1)
  ldi   r16, LOW(-1)
USART_ReceiveNoError:
  ; Filter the 9th bit, then return
  lsr   r17
  andi  r17, 0x01
  ret
```

C Code Example[1]

```
unsigned int USART_Receive( void )
{
  unsigned char status, resh, resl;
  /* Wait for data to be received */
  while ( !(UCSRA & (1<<RXC)) )
        ;
  /* Get status and 9th bit, then data */
  /* from buffer */
  status = UCSRA;
  resh = UCSRB;
  resl = UDR;
  /* If error, return -1 */
  if ( status & (1<<FE)|(1<<DOR)|(1<<UPE) )
    return -1;
  /* Filter the 9th bit, then return */
  resh = (resh >> 1) & 0x01;
  return ((resh << 8) | resl);
}
```

Note: 1. See “About Code Examples” on page 6.

The receive function example reads all the I/O Registers into the Register File before any computation is done. This gives an optimal receive buffer utilization since the buffer location read will be free to accept new data as early as possible.

Receive Compete Flag and Interrupt

The USART Receiver has one flag that indicates the Receiver state.

The Receive Complete (RXC) Flag indicates if there are unread data present in the receive buffer. This flag is one when unread data exist in the receive buffer, and zero when the receive buffer is empty (i.e., does not contain any unread data). If the Receiver is disabled (RXEN = 0), the receive buffer will be flushed and consequently the RXC bit will become zero.

When the Receive Complete Interrupt Enable (RXCIE) in UCSRB is set, the USART Receive Complete interrupt will be executed as long as the RXC Flag is set (provided that global interrupts are enabled). When interrupt-driven data reception is used, the receive complete routine must read the received data from UDR in order to clear the RXC Flag, otherwise a new interrupt will occur once the interrupt routine terminates.

Receiver Error Flags

The USART Receiver has three Error Flags: Frame Error (FE), Data OverRun (DOR) and Parity Error (UPE). All can be accessed by reading UCSRA. Common for the Error Flags is that they are located in the receive buffer together with the frame for which they indicate the error status. Due to the buffering of the Error Flags, the UCSRA must be read before the receive buffer (UDR), since reading the UDR I/O location changes the buffer read location. Another equality for the Error Flags is that they can not be altered by software doing a write to the flag location. However, all flags must be set to zero when the UCSRA is written for upward compatibility of future USART implementations. None of the Error Flags can generate interrupts.

The Frame Error (FE) Flag indicates the state of the first stop bit of the next readable frame stored in the receive buffer. The FE Flag is zero when the stop bit was correctly read (as one), and the FE Flag will be one when the stop bit was incorrect (zero). This flag can be used for detecting out-of-sync conditions, detecting break conditions and protocol handling. The FE Flag is not affected by the setting of the USBS bit in UCSRC since the Receiver ignores all, except for the first, stop bits. For compatibility with future devices, always set this bit to zero when writing to UCSRA.

The Data OverRun (DOR) Flag indicates data loss due to a receiver buffer full condition. 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. If the DOR Flag is set there was one or more serial frame lost between the frame last read from UDR, and the next frame read from UDR. For compatibility with future devices, always write this bit to zero when writing to UCSRA. The DOR Flag is cleared when the frame received was successfully moved from the Shift Register to the receive buffer.

The Parity Error (UPE) Flag indicates that the next frame in the receive buffer had a Parity Error when received. If Parity Check is not enabled the UPE bit will always be read zero. For compatibility with future devices, always set this bit to zero when writing to UCSRA. For more details see “Parity Bit Calculation” on page 157 and “Parity Checker” on page 165.

Parity Checker

The Parity Checker is active when the high USART Parity mode (UPM1) bit is set. Type of Parity Check to be performed (odd or even) is selected by the UPM0 bit. When enabled, the Parity Checker calculates the parity of the data bits in incoming frames and compares the result with the parity bit from the serial frame. The result of the check is stored in the receive buffer together with the received data and stop bits. The Parity Error (UPE) Flag can then be read by software to check if the frame had a Parity Error.

The UPE bit is set if the next character that can be read from the receive buffer had a Parity Error when received and the Parity Checking was enabled at that point (UPM1 = 1). This bit is valid until the receive buffer (UDR) is read.

Disabling the Receiver

In contrast to the Transmitter, disabling of the Receiver will be immediate. Data from ongoing receptions will therefore be lost. When disabled (i.e., the RXEN is set to zero) the Receiver will no longer override the normal function of the RxD port pin. The Receiver buffer FIFO will be flushed when the Receiver is disabled. Remaining data in the buffer will be lost

Flushing the Receive Buffer

The receiver buffer FIFO will be flushed when the Receiver is disabled, i.e., the buffer will be emptied of its contents. Unread data will be lost. If the buffer has to be flushed during normal operation, due to for instance an error condition, read the UDR I/O location until the RXC Flag is cleared. The following code example shows how to flush the receive buffer.

Assembly Code Example[1]

```
USART_Flush:
  sbis  UCSRA, RXC
  ret
  in    r16, UDR
  rjmp  USART_Flush
```

C Code Example[1]

```
void USART_Flush( void )
{
  unsigned char dummy;
  while ( UCSRA & (1<<RXC) ) dummy = UDR;
}
```

Note: 1. See “About Code Examples” on page 6.

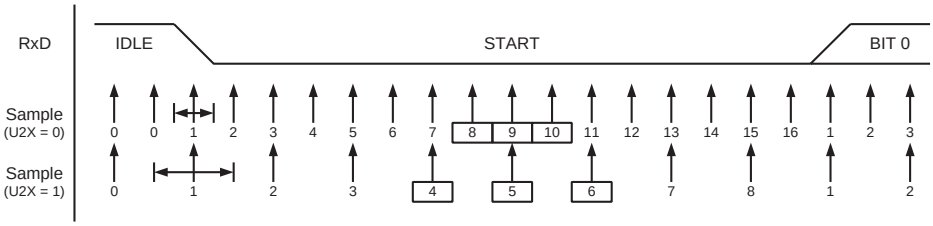
Asynchronous Data Reception

The USART includes a clock recovery and a data recovery unit for handling asynchronous data reception. The clock recovery logic is used for synchronizing the internally generated baud rate clock to the incoming asynchronous serial frames at the RxD pin. The data recovery logic samples and low pass filters each incoming bit, thereby improving the noise immunity of the Receiver. The asynchronous reception operational range depends on the accuracy of the internal baud rate clock, the rate of the incoming frames, and the frame size in number of bits.

Asynchronous Clock Recovery

The clock recovery logic synchronizes internal clock to the incoming serial frames. Figure 73 illustrates the sampling process of the start bit of an incoming frame. The sample rate is 16 times the baud rate for Normal mode, and eight times the baud rate for Double Speed mode. The horizontal arrows illustrate the synchronization variation due to the sampling process. Note the larger time variation when using the Double Speed mode (U2X = 1) of operation. Samples denoted zero are samples done when the RxD line is idle (i.e., no communication activity).

Figure 73. Start Bit Sampling

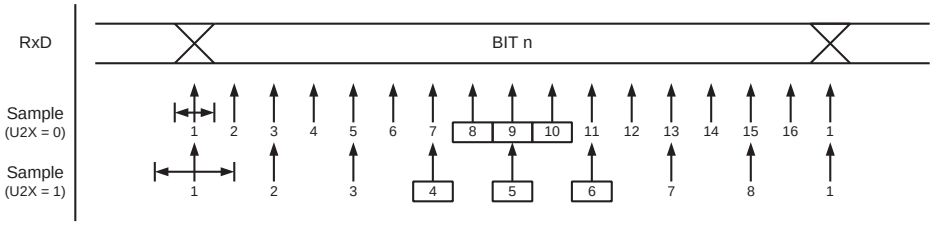


When the clock recovery logic detects a high (idle) to low (start) transition on the RxD line, the start bit detection sequence is initiated. Let sample 1 denote the first zero-sample as shown in the figure. The clock recovery logic then uses samples 8, 9, and 10 for Normal mode, and samples 4, 5, and 6 for Double Speed mode (indicated with sample numbers inside boxes on the figure), to decide if a valid start bit is received. If two or more of these three samples have logical high levels (the majority wins), the start bit is rejected as a noise spike and the Receiver starts looking for the next high to low-transition. If however, a valid start bit is detected, the clock recovery logic is synchronized and the data recovery can begin. The synchronization process is repeated for each start bit.

Asynchronous Data Recovery

When the receiver clock is synchronized to the start bit, the data recovery can begin. The data recovery unit uses a state machine that has 16 states for each bit in Normal mode and eight states for each bit in Double Speed mode. Figure 74 shows the sampling of the data bits and the parity bit. Each of the samples is given a number that is equal to the state of the recovery unit.

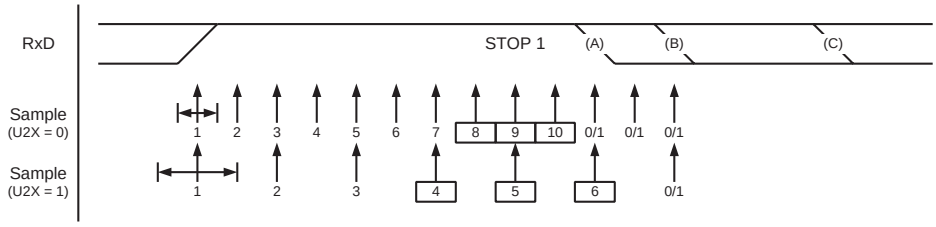
Figure 74. Sampling of Data and Parity Bit



The decision of the logic level of the received bit is taken by doing a majority voting of the logic value to the three samples in the center of the received bit. The center samples are emphasized on the figure by having the sample number inside boxes. The majority voting process is done as follows: If two or all three samples have high levels, the received bit is registered to be a logic 1. If two or all three samples have low levels, the received bit is registered to be a logic 0. This majority voting process acts as a low pass filter for the incoming signal on the RxD pin. The recovery process is then repeated until a complete frame is received. Including the first stop bit. Note that the Receiver only uses the first stop bit of a frame.

Figure 75 shows the sampling of the stop bit and the earliest possible beginning of the start bit of the next frame.

Figure 75. Stop Bit Sampling and Next Start Bit Sampling



The same majority voting is done to the stop bit as done for the other bits in the frame. If the stop bit is registered to have a logic 0 value, the Frame Error (FE) Flag will be set.

A new high to low transition indicating the start bit of a new frame can come right after the last of the bits used for majority voting. For Normal Speed mode, the first low level sample can be at point marked (A) in Figure 75. For Double Speed mode the first low level must be delayed to (B). (C) marks a stop bit of full length. The early start bit detection influences the operational range of the Receiver.

Asynchronous Operational Range

The operational range of the Receiver is dependent on the mismatch between the received bit rate and the internally generated baud rate. If the Transmitter is sending frames at too fast or too slow bit rates, or the internally generated baud rate of the Receiver does not have a similar (see Table 72) base frequency, the Receiver will not be able to synchronize the frames to the start bit.

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_F First sample number used for majority voting. S_F = 8 for normal speed and S_F = 4 for Double Speed mode.

S_M Middle sample number used for majority voting. S_M = 9 for normal speed and S_M = 5 for Double Speed mode.

R_{slow} is the ratio of the slowest incoming data rate that can be accepted in relation to the receiver baud rate. R_{fast} is the ratio of the fastest incoming data rate that can be accepted in relation to the receiver baud rate.

Table 72 and Table 73 list the maximum receiver baud rate error that can be tolerated. Note that Normal Speed mode has higher toleration of baud rate variations.

Table 72. Recommended Maximum Receiver Baud Rate Error for Normal Speed Mode (U2X = 0)

D # (Data+Parity Bit)	R_{slow} (%)	R_{fast} (%)	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 73. Recommended Maximum Receiver Baud Rate Error for Double Speed Mode (U2X = 1)

D # (Data+Parity Bit)	R_{slow} (%)	R_{fast} (%)	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

The recommendations of the maximum receiver baud rate error was made under the assumption that the Receiver and Transmitter equally divides the maximum total error.

There are two possible sources for the receivers baud rate error. The Receiver's system clock (XTAL) will always have some minor instability over the supply voltage range and the temperature range. When using a crystal to generate the system clock, this is rarely a problem, but for a resonator the system clock may differ more than 2% depending of the resonators tolerance. The second source for the error is more controllable. The baud rate generator can not always do an exact division of the system frequency to get the baud rate wanted. In this case an UBRR value that gives an acceptable low error can be used if possible.

Multi-processor Communication Mode

Setting the Multi-processor Communication mode (MPCM) bit in UCSRA enables a filtering function of incoming frames received by the USART Receiver. Frames that do not contain address information will be ignored and not put into the receive buffer. This effectively reduces the number of incoming frames that has to be handled by the CPU, in a system with multiple MCUs that communicate via the same serial bus. The Transmitter is unaffected by the MPCM setting, but has to be used differently when it is a part of a system utilizing the Multi-processor Communication mode.

If the Receiver is set up to receive frames that contain 5 to 8 data bits, then the first stop bit indicates if the frame contains data or address information. If the Receiver is set up for frames with nine data bits, then the ninth bit (RXB8) is used for identifying address and data frames. When the frame type bit (the first stop or the ninth bit) is one, the frame contains an address. When the frame type bit is zero the frame is a data frame.

The Multi-processor Communication mode enables several slave MCUs to receive data from a master MCU. This is done by first decoding an address frame to find out which MCU has been addressed. If a particular slave MCU has been addressed, it will receive the following data frames as normal, while the other slave MCUs will ignore the received frames until another address frame is received.

Using MPCM

For an MCU to act as a master MCU, it can use a 9-bit character frame format (UCSZ = 7). The ninth bit (TXB8) must be set when an address frame (TXB8 = 1) or cleared when a data frame (TXB = 0) is being transmitted. The slave MCUs must in this case be set to use a 9-bit character frame format.

The following procedure should be used to exchange data in Multi-processor Communication mode:

1. All Slave MCUs are in Multi-processor Communication mode (MPCM in UCSRA is set).
2. The Master MCU sends an address frame, and all slaves receive and read this frame. In the Slave MCUs, the RXC Flag in UCSRA will be set as normal.
3. Each Slave MCU reads the UDR Register and determines if it has been selected. If so, it clears the MPCM bit in UCSRA, otherwise it waits for the next address byte and keeps the MPCM setting.
4. The addressed MCU will receive all data frames until a new address frame is received. The other Slave MCUs, which still have the MPCM bit set, will ignore the data frames.
5. When the last data frame is received by the addressed MCU, the addressed MCU sets the MPCM bit and waits for a new address frame from master. The process then repeats from 2.

Using any of the 5- to 8-bit character frame formats is possible, but impractical since the Receiver must change between using n and n+1 character frame formats. This makes full-duplex operation difficult since the Transmitter and Receiver uses the same character size setting. If 5- to 8-bit character frames are used, the Transmitter must be set to use two stop bit (USBS = 1) since the first stop bit is used for indicating the frame type.

Do not use Read-Modify-Write instructions (SBI and CBI) to set or clear the MPCM bit. The MPCM bit shares the same I/O location as the TXC Flag and this might accidentally be cleared when using SBI or CBI instructions.

USART Register Description

USART I/O Data Register – UDR

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
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 USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or UDR. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDR Register location. Reading the UDR Register location will return the contents of the Receive Data Buffer Register (RXB).

For 5-, 6-, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver.

The transmit buffer can only be written when the UDRE Flag in the UCSRA Register is set. Data written to UDR when the UDRE Flag is not set, will be ignored by the USART Transmitter. When data is written to the transmit buffer, and the Transmitter is enabled, the Transmitter will load the data into the Transmit Shift Register when the Shift Register is empty. Then the data will be serially transmitted on the TxD pin.

The receive buffer consists of a two level FIFO. The FIFO will change its state whenever the receive buffer is accessed. Due to this behavior of the receive buffer, do not use Read-Modify-Write instructions (SBI and CBI) on this location. Be careful when using bit test instructions (SBIC and SBIS), since these also will change the state of the FIFO.

USART Control and Status Register A – UCSRA

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	UPE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	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 (UDR). 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 (UDR) 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 (UDR) is read. The FE bit is zero when the stop bit of received data is one. Always set this bit to zero when writing to UCSRA.

- **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 (UDR) is read. Always set this bit to zero when writing to UCSRA.

- **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 (UPM1 = 1). This bit is valid until the receive buffer (UDR) is read. Always set this bit to zero when writing to UCSRA.

- **Bit 1 – U2X: Double the USART Transmission Speed**

This bit only has effect for the asynchronous operation. Write this bit to zero when using synchronous operation.

Writing this bit to one will reduce the divisor of the baud rate divider from 16 to 8 effectively doubling the transfer rate for asynchronous communication.

- **Bit 0 – MPCM: Multi-processor Communication Mode**

This bit enables the Multi-processor Communication mode. When the MPCM bit is written to one, all the incoming frames received by the USART Receiver that do not contain address information will be ignored. The Transmitter is unaffected by the MPCM setting. For more detailed information see “Multi-processor Communication Mode” on page 169.

USART Control and Status Register B – UCSRB

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – RXCIE: RX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the RXC Flag. A USART Receive Complete interrupt will be generated only if the RXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXC bit in UCSRA is set.

- **Bit 6 – TXCIE: TX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the TXC Flag. A USART Transmit Complete interrupt will be generated only if the TXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXC bit in UCSRA is set.

- **Bit 5 – UDRIE: USART Data Register Empty Interrupt Enable**

Writing this bit to one enables interrupt on the UDRE Flag. A Data Register Empty interrupt will be generated only if the UDRIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDRE bit in UCSRA is set.

- **Bit 4 – RXEN: Receiver Enable**

Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxD pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FE, DOR, and UPE Flags.

- **Bit 3 – TXEN: Transmitter Enable**

Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxD pin when enabled. The disabling of the Transmitter (writing TXEN to zero) 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 TxD port.

- **Bit 2 – UCSZ2: Character Size**

The UCSZ2 bits combined with the UCSZ1:0 bit in UCSRC sets the number of data bits (Character SiZe) in a frame the Receiver and Transmitter use.

- **Bit 1 – RXB8: Receive Data Bit 8**

RXB8 is the ninth data bit of the received character when operating with serial frames with nine data bits. Must be read before reading the low bits from UDR.

- **Bit 0 – TXB8: Transmit Data Bit 8**

TXB8 is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. Must be written before writing the low bits to UDR.

USART Control and Status Register C – UCSRC

Bit	7	6	5	4	3	2	1	0	
	–	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

- **Bit 6 – UMSEL: USART Mode Select**

This bit selects between asynchronous and synchronous mode of operation.

Table 74. UMSEL Bit Settings

UMSEL	Mode
0	Asynchronous Operation
1	Synchronous Operation

- **Bit 5:4 – UPM1:0: Parity Mode**

These bits enable and set type of parity generation and check. If enabled, the Transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The Receiver will generate a parity value for the incoming data and compare it to the UPM0 setting. If a mismatch is detected, the UPE Flag in UCSRA will be set.

Table 75. UPM Bits Settings

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

- **Bit 3 – USBS: Stop Bit Select**

This bit selects the number of stop bits to be inserted by the Transmitter. The Receiver ignores this setting.

Table 76. USBS Bit Settings

USBS	Stop Bit(s)
0	1-bit
1	2-bit

- **Bit 2:1 – UCSZ1:0: Character Size**

The UCSZ1:0 bits combined with the UCSZ2 bit in UCSRB sets the number of data bits (Character SiZe) in a frame the Receiver and Transmitter use.

Table 77. UCSZ Bits Settings

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

- **Bit 0 – UCPOL: Clock Polarity**

This bit is used for synchronous mode only. Write this bit to zero when asynchronous mode is used. The UCPOL bit sets the relationship between data output change and data input sample, and the synchronous clock (XCK).

Table 78. UCPOL Bit Settings

UCPOL	Transmitted Data Changed (Output of TxD Pin)	Received Data Sampled (Input on RxD Pin)
0	Rising XCK Edge	Falling XCK Edge
1	Falling XCK Edge	Rising XCK Edge

USART Baud Rate Registers – UBRRL and UBRRH

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	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	
	0	0	0	0	0	0	0	0	

- **Bit 15:12 – Reserved Bits**

These bits are reserved for future use. For compatibility with future devices, these bit must be written to zero when UBRRH is written.

- **Bit 11:0 – UBRR11:0: USART Baud Rate Register**

This is a 12-bit register which contains the USART baud rate. The UBRRH contains the four most significant bits, and the UBRRL contains the eight least significant bits of the USART baud rate. Ongoing transmissions by the Transmitter and Receiver will be corrupted if the baud rate is changed. Writing UBRRL will trigger an immediate update of the baud rate prescaler.

Examples of Baud Rate Setting

For standard crystal and resonator frequencies, the most commonly used baud rates for asynchronous operation can be generated by using the UBRR settings in Table 79. UBRR values which yield an actual baud rate differing less than 0.5% from the target baud rate, are bold in the table. Higher error ratings are acceptable, but the Receiver will have less noise resistance when the error ratings are high, especially for large serial frames (see "Asynchronous Operational Range" on page 167). The error values are calculated using the following equation:

$$\text{Error}[\%] = \left(\frac{\text{BaudRate}_{\text{Closest Match}}}{\text{BaudRate}} - 1\right) \bullet 100\%$$

Table 79. Examples of UBRR Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	f_{osc} = 1.0000 MHz				f_{osc} = 1.8432 MHz				f_{osc} = 2.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max. (1)	62.5 kbps		125 kbps		115.2 kbps		230.4 kbps		125 kbps		250 kbps	

1. UBRR = 0, Error = 0.0%

Table 80. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 3.6864 MHz				f_{osc} = 4.0000 MHz				f_{osc} = 7.3728 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	95	0.0%	191	0.0%	103	0.2%	207	0.2%	191	0.0%	383	0.0%
4800	47	0.0%	95	0.0%	51	0.2%	103	0.2%	95	0.0%	191	0.0%
9600	23	0.0%	47	0.0%	25	0.2%	51	0.2%	47	0.0%	95	0.0%
14.4k	15	0.0%	31	0.0%	16	2.1%	34	-0.8%	31	0.0%	63	0.0%
19.2k	11	0.0%	23	0.0%	12	0.2%	25	0.2%	23	0.0%	47	0.0%
28.8k	7	0.0%	15	0.0%	8	-3.5%	16	2.1%	15	0.0%	31	0.0%
38.4k	5	0.0%	11	0.0%	6	-7.0%	12	0.2%	11	0.0%	23	0.0%
57.6k	3	0.0%	7	0.0%	3	8.5%	8	-3.5%	7	0.0%	15	0.0%
76.8k	2	0.0%	5	0.0%	2	8.5%	6	-7.0%	5	0.0%	11	0.0%
115.2k	1	0.0%	3	0.0%	1	8.5%	3	8.5%	3	0.0%	7	0.0%
230.4k	0	0.0%	1	0.0%	0	8.5%	1	8.5%	1	0.0%	3	0.0%
250k	0	-7.8%	1	-7.8%	0	0.0%	1	0.0%	1	-7.8%	3	-7.8%
0.5M	–	–	0	-7.8%	–	–	0	0.0%	0	-7.8%	1	-7.8%
1M	–	–	–	–	–	–	–	–	–	–	0	-7.8%
Max. (1)	230.4 kbps		460.8 kbps		250 kbps		0.5 Mbps		460.8 kbps		921.6 kbps	

1. UBRR = 0, Error = 0.0%

Table 81. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 8.0000 MHz				f_{osc} = 11.0592 MHz				f_{osc} = 14.7456 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	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)	0.5 Mbps		1 Mbps		691.2 kbps		1.3824 Mbps		921.6 kbps		1.8432 Mbps	

1. UBRR = 0, Error = 0.0%

Table 82. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 16.0000 MHz				f_{osc} = 18.4320 MHz				f_{osc} = 20.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250k	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	–	–	4	-7.8%	–	–	4	0.0%
1M	0	0.0%	1	0.0%	–	–	–	–	–	–	–	–
Max. (1)	1 Mbps		2 Mbps		1.152 Mbps		2.304 Mbps		1.25 Mbps		2.5 Mbps	

1. UBRR = 0, Error = 0.0%

USI – Universal Serial Interface

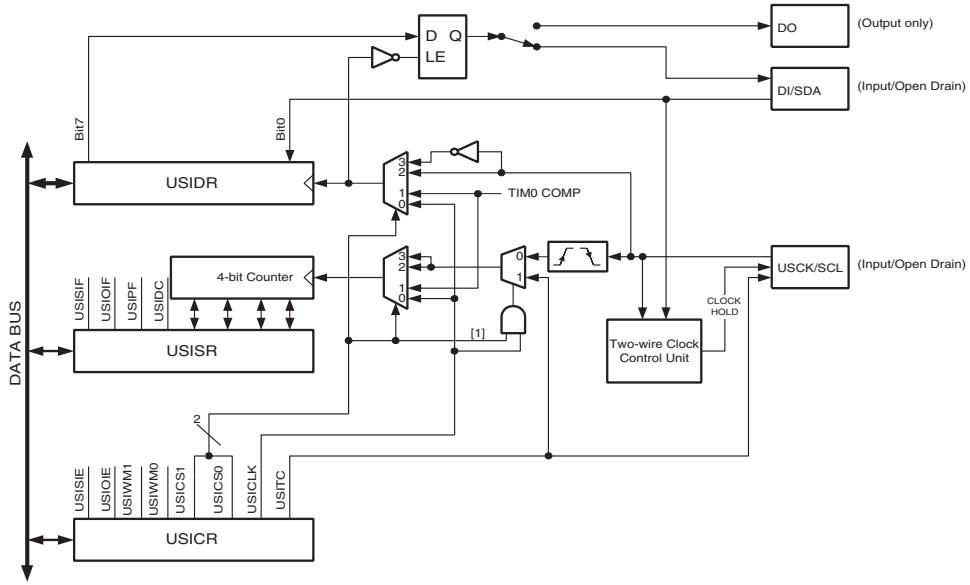
The Universal Serial Interface, or USI, provides the basic hardware resources needed for serial communication. Combined with a minimum of control software, the USI allows significantly higher transfer rates and uses less code space than solutions based on software only. Interrupts are included to minimize the processor load. The main features of the USI are:

- **Two-wire Synchronous Data Transfer (Master or Slave)**
- **Three-wire Synchronous Data Transfer (Master or Slave)**
- **Data Received Interrupt**
- **Wakeup from Idle Mode**
- **In Two-wire Mode: Wake-up from All Sleep Modes, Including Power-down Mode**
- **Two-wire Start Condition Detector with Interrupt Capability**

Overview

A simplified block diagram of the USI is shown on Figure 76. For the actual placement of I/O pins, refer to “Pinout ATmega169” on page 2. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the “USI Register Descriptions” on page 185.

Figure 76. Universal Serial Interface, Block Diagram



The 8-bit Shift Register is directly accessible via the data bus and contains the incoming and outgoing data. The register has no buffering so the data must be read as quickly as possible to ensure that no data is lost. The most significant bit is connected to one of two output pins depending of the wire mode configuration. A transparent latch is inserted between the Serial Register Output and output pin, which delays the change of data output to the opposite clock edge of the data input sampling. The serial input is always sampled from the Data Input (DI) pin independent of the configuration.

The 4-bit counter can be both read and written via the data bus, and can generate an overflow interrupt. Both the Serial Register and the counter are clocked simultaneously by the same clock source. This allows the counter to count the number of bits received or transmitted and generate an interrupt when the transfer is complete. Note that when an external clock source is selected the counter counts both clock edges. In this case the counter counts the number of edges, and not the number of bits. The clock can be selected from three different sources: The USCK pin, Timer/Counter0 Compare Match or from software.

The Two-wire clock control unit can generate an interrupt when a start condition is detected on the Two-wire bus. It can also generate wait states by holding the clock pin low after a start condition is detected, or after the counter overflows.

Functional Descriptions

Three-wire Mode

The USI Three-wire mode is compliant to the Serial Peripheral Interface (SPI) mode 0 and 1, but does not have the slave select (SS) pin functionality. However, this feature can be implemented in software if necessary. Pin names used by this mode are: DI, DO, and USCK.

Figure 77. Three-wire Mode Operation, Simplified Diagram

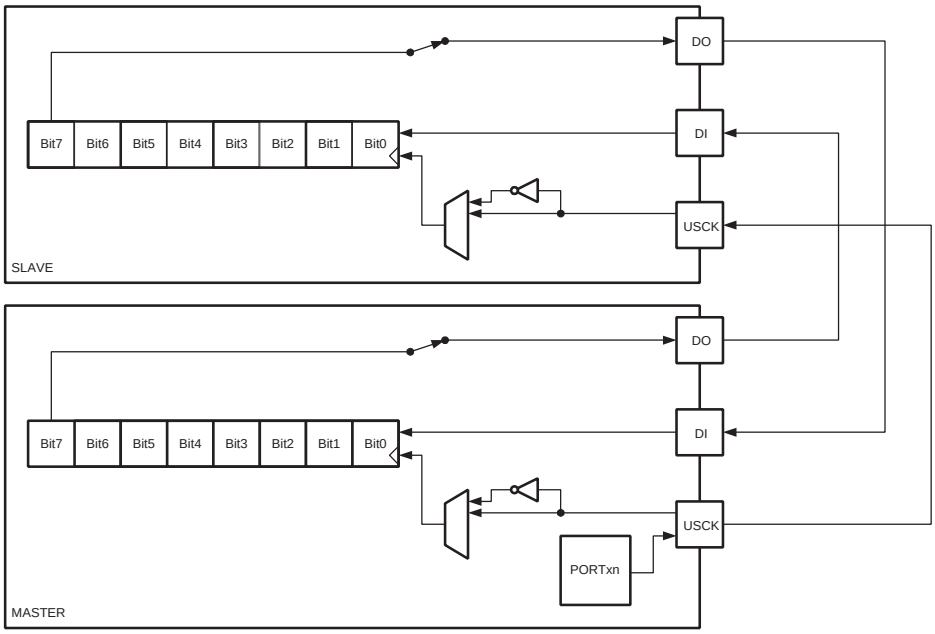
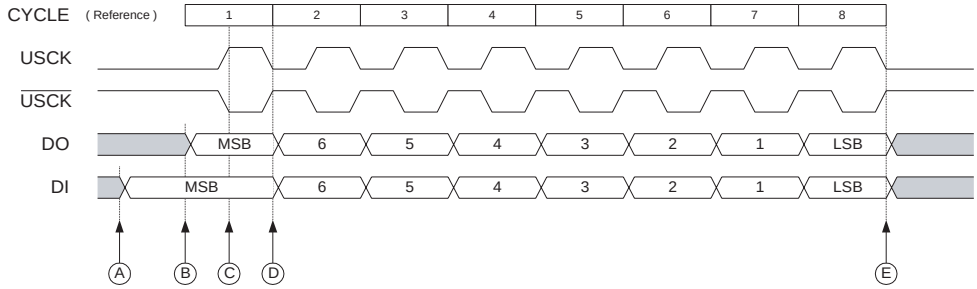


Figure 77 shows two USI units operating in Three-wire mode, one as Master and one as Slave. The two Shift Registers are interconnected in such way that after eight USCK clocks, the data in each register are interchanged. The same clock also increments the USI's 4-bit counter. The Counter Overflow (interrupt) Flag, or USIOIF, can therefore be used to determine when a transfer is completed. The clock is generated by the Master device software by toggling the USCK pin via the PORT Register or by writing a one to the USITC bit in USICR.

Figure 78. Three-wire Mode, Timing Diagram



The Three-wire mode timing is shown in Figure 78. At the top of the figure is a USCK cycle reference. One bit is shifted into the USI Shift Register (USIDR) for each of these cycles. The USCK timing is shown for both external clock modes. In External Clock mode 0 (USICS0 = 0), DI is sampled at positive edges, and DO is changed (Data Register is shifted by one) at negative edges. External Clock mode 1 (USICS0 = 1) uses the opposite edges versus mode 0, i.e., samples data at negative and changes the output at positive edges. The USI clock modes corresponds to the SPI data mode 0 and 1.

Referring to the timing diagram (Figure 78.), a bus transfer involves the following steps:

1. The Slave device and Master device sets up its data output and, depending on the protocol used, enables its output driver (mark A and B). The output is set up by writing the data to be transmitted to the Serial Data Register. Enabling of the output is done by setting the corresponding bit in the port Data Direction Register. Note that point A and B does not have any specific order, but both must be at least one half USCK cycle before point C where the data is sampled. This must be done to ensure that the data setup requirement is satisfied. The 4-bit counter is reset to zero.
2. The Master generates a clock pulse by software toggling the USCK line twice (C and D). The bit value on the slave and master's data input (DI) pin is sampled by the USI on the first edge (C), and the data output is changed on the opposite edge (D). The 4-bit counter will count both edges.
3. Step 2. is repeated eight times for a complete register (byte) transfer.
4. After eight clock pulses (i.e., 16 clock edges) the counter will overflow and indicate that the transfer is completed. The data bytes transferred must now be processed before a new transfer can be initiated. The overflow interrupt will wake up the processor if it is set to Idle mode. Depending of the protocol used the slave device can now set its output to high impedance.

SPI Master Operation Example

The following code demonstrates how to use the USI module as a SPI Master:

```
SPITransfer:
    sts     USIDR,r16
    ldi     r16,(1<<USIOIF)
    sts     USISR,r16
    ldi     r16,(1<<USIWM0)|(1<<USICS1)|(1<<USICLK)|(1<<USITC)
SPITransfer_loop:
    sts     USICR,r16
    lds     r16, USISR
    sbrs    r16, USIOIF
    rjmp    SPITransfer_loop
    lds     r16,USIDR
    ret
```

The code is size optimized using only eight instructions (+ ret). The code example assumes that the DO and USCK pins are enabled as output in the DDRE Register. The value stored in register r16 prior to the function is called is transferred to the Slave device, and when the transfer is completed the data received from the Slave is stored back into the r16 Register.

The second and third instructions clears the USI Counter Overflow Flag and the USI counter value. The fourth and fifth instruction set Three-wire mode, positive edge Shift Register clock, count at USITC strobe, and toggle USCK. The loop is repeated 16 times.

The following code demonstrates how to use the USI module as a SPI Master with maximum speed (fsck = fck/4):

```
SPITransfer_Fast:

    sts     USIDR,r16
    ldi     r16,(1<<USIWM0)|(0<<USICS0)|(1<<USITC)
    ldi     r17,(1<<USIWM0)|(0<<USICS0)|(1<<USITC)|(1<<USICLK)

    sts     USICR,r16 ; MSB
    sts     USICR,r17
    sts     USICR,r16
    sts     USICR,r17
    sts     USICR,r16
    sts     USICR,r17
    sts     USICR,r16
    sts     USICR,r17
    sts     USICR,r16
    sts     USICR,r17
    sts     USICR,r16
    sts     USICR,r17
    sts     USICR,r16
    sts     USICR,r17
    sts     USICR,r16 ; LSB
    sts     USICR,r17

    lds     r16,USIDR
ret
```

SPI Slave Operation Example

The following code demonstrates how to use the USI module as a SPI Slave:

```
init:
    ldi     r16,(1<<USIWM0)|(1<<USICS1)
    sts     USICR,r16
...
SlaveSPITransfer:
    sts     USIDR,r16
    ldi     r16,(1<<USIOIF)
    sts     USISR,r16
SlaveSPITransfer_loop:
    lds     r16, USISR
    sbrs    r16, USIOIF
    rjmp    SlaveSPITransfer_loop
    lds     r16,USIDR
    ret
```

The code is size optimized using only eight instructions (+ ret). The code example assumes that the DO is configured as output and USCK pin is configured as input in the DDR Register. The value stored in register r16 prior to the function is called is transferred to the master device, and when the transfer is completed the data received from the Master is stored back into the r16 Register.

Note that the first two instructions is for initialization only and needs only to be executed once.These instructions sets Three-wire mode and positive edge Shift Register clock. The loop is repeated until the USI Counter Overflow Flag is set.

Two-wire Mode

The USI Two-wire mode is compliant to the Inter IC (TWI) bus protocol, but without slew rate limiting on outputs and input noise filtering. Pin names used by this mode are SCL and SDA.

Figure 79. Two-wire Mode Operation, Simplified Diagram

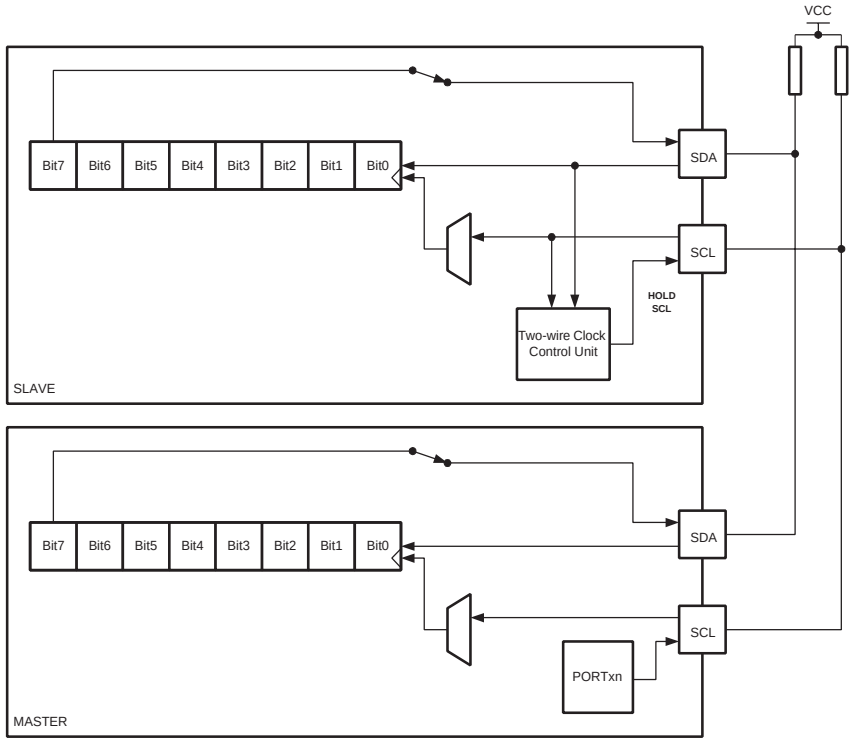
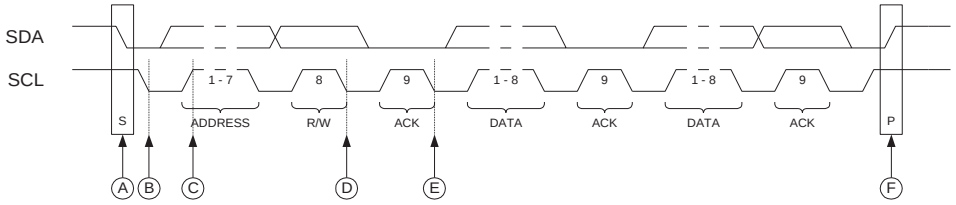


Figure 79 shows two USI units operating in Two-wire mode, one as Master and one as Slave. It is only the physical layer that is shown since the system operation is highly dependent of the communication scheme used. The main differences between the Master and Slave operation at this level, is the serial clock generation which is always done by the Master, and only the Slave uses the clock control unit. Clock generation must be implemented in software, but the shift operation is done automatically by both devices. Note that only clocking on negative edge for shifting data is of practical use in this mode. The slave can insert wait states at start or end of transfer by forcing the SCL clock low. This means that the Master must always check if the SCL line was actually released after it has generated a positive edge.

Since the clock also increments the counter, a counter overflow can be used to indicate that the transfer is completed. The clock is generated by the master by toggling the USCK pin via the PORT Register.

The data direction is not given by the physical layer. A protocol, like the one used by the TWI-bus, must be implemented to control the data flow.

Figure 80. Two-wire Mode, Typical Timing Diagram

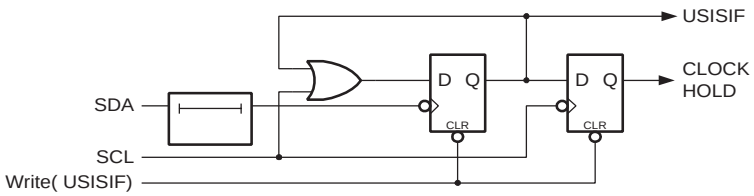


Referring to the timing diagram (Figure 80.), a bus transfer involves the following steps:

1. The a start condition is generated by the Master by forcing the SDA low line while the SCL line is high (A). SDA can be forced low either by writing a zero to bit 7 of the Shift Register, or by setting the corresponding bit in the PORT Register to zero. Note that the Data Direction Register bit must be set to one for the output to be enabled. The slave device's start detector logic (Figure 81.) detects the start condition and sets the USISIF Flag. The flag can generate an interrupt if necessary.
2. In addition, the start detector will hold the SCL line low after the Master has forced an negative edge on this line (B). This allows the Slave to wake up from sleep or complete its other tasks before setting up the Shift Register to receive the address. This is done by clearing the start condition flag and reset the counter.
3. The Master set the first bit to be transferred and releases the SCL line (C). The Slave samples the data and shift it into the Serial Register at the positive edge of the SCL clock.
4. After eight bits are transferred containing slave address and data direction (read or write), the Slave counter overflows and the SCL line is forced low (D). If the slave is not the one the Master has addressed, it releases the SCL line and waits for a new start condition.
5. If the Slave is addressed it holds the SDA line low during the acknowledgment cycle before holding the SCL line low again (i.e., the Counter Register must be set to 14 before releasing SCL at (D)). Depending of the R/W bit the Master or Slave enables its output. If the bit is set, a master read operation is in progress (i.e., the slave drives the SDA line) The slave can hold the SCL line low after the acknowledge (E).
6. Multiple bytes can now be transmitted, all in same direction, until a stop condition is given by the Master (F). Or a new start condition is given.

If the Slave is not able to receive more data it does not acknowledge the data byte it has last received. When the Master does a read operation it must terminate the operation by force the acknowledge bit low after the last byte transmitted.

Figure 81. Start Condition Detector, Logic Diagram



Start Condition Detector

The start condition detector is shown in Figure 81. The SDA line is delayed (in the range of 50 to 300 ns) to ensure valid sampling of the SCL line. The start condition detector is only enabled in Two-wire mode.

The start condition detector is working asynchronously and can therefore wake up the processor from the Power-down sleep mode. However, the protocol used might have restrictions on the SCL hold time. Therefore, when using this feature in this case the Oscillator start-up time set by the CKSEL Fuses (see “Clock Systems and their Distribution” on page 23) must also be taken into the consideration. Refer to the USISIF bit description on page 186 for further details.

Clock speed considerations.

Maximum frequency for SCL and SCK is f_{CK} /4. This is also the maximum data transmit and receieve rate in both two- and three-wire mode. In two-wire slave mode the Two-wire Clock Control Unit will hold the SCL low until the slave is ready to receive more data. This may reduce the actual data rate in two-wire mode.

Alternative USI Usage

When the USI unit is not used for serial communication, it can be set up to do alternative tasks due to its flexible design.

Half-duplex Asynchronous Data Transfer

By utilizing the Shift Register in Three-wire mode, it is possible to implement a more compact and higher performance UART than by software only.

4-bit Counter

The 4-bit counter can be used as a stand-alone counter with overflow interrupt. Note that if the counter is clocked externally, both clock edges will generate an increment.

12-bit Timer/Counter

Combining the USI 4-bit counter and Timer/Counter0 allows them to be used as a 12-bit counter.

Edge Triggered External Interrupt

By setting the counter to maximum value (F) it can function as an additional external interrupt. The Overflow Flag and Interrupt Enable bit are then used for the external interrupt. This feature is selected by the USICS1 bit.

Software Interrupt

The counter overflow interrupt can be used as a software interrupt triggered by a clock strobe.

USI Register Descriptions

USI Data Register – USIDR

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	USIDR
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 USI uses no buffering of the Serial Register, i.e., when accessing the Data Register (USIDR) the Serial Register is accessed directly. If a serial clock occurs at the same cycle the register is written, the register will contain the value written and no shift is performed. A (left) shift operation is performed depending of the USICS1..0 bits setting. The shift operation can be controlled by an external clock edge, by a Timer/Counter0 Compare Match, or directly by software using the USICLK strobe bit. Note that even when no wire mode is selected (USIWM1..0 = 0) both the external data input (DI/SDA) and the external clock input (USCK/SCL) can still be used by the Shift Register.

The output pin in use, DO or SDA depending on the wire mode, is connected via the output latch to the most significant bit (bit 7) of the Data Register. The output latch is open (transparent) during the first half of a serial clock cycle when an external clock source is selected (USICS1 = 1), and constantly open when an internal clock source is used (USICS1 = 0). The output will be changed immediately when a new MSB written as long as the latch is open. The latch ensures that data input is sampled and data output is changed on opposite clock edges.

Note that the corresponding Data Direction Register to the pin must be set to one for enabling data output from the Shift Register.

USI Status Register – USISR

Bit	7	6	5	4	3	2	1	0	
	USISIF	USIOIF	USIPF	USIDC	USICNT3	USICNT2	USICNT1	USICNT0	USISR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Status Register contains Interrupt Flags, line Status Flags and the counter value.

- **Bit 7 – USISIF: Start Condition Interrupt Flag**

When Two-wire mode is selected, the USISIF Flag is set (to one) when a start condition is detected. When output disable mode or Three-wire mode is selected and (USICSx = 0b11 & USICLK = 0) or (USICS = 0b10 & USICLK = 0), any edge on the SCK pin sets the flag.

An interrupt will be generated when the flag is set while the USISIE bit in USICR and the Global Interrupt Enable Flag are set. The flag will only be cleared by writing a logical one to the USISIF bit. Clearing this bit will release the start detection hold of USCL in Two-wire mode.

A start condition interrupt will wakeup the processor from all sleep modes.

- **Bit 6 – USIOIF: Counter Overflow Interrupt Flag**

This flag is set (one) when the 4-bit counter overflows (i.e., at the transition from 15 to 0). An interrupt will be generated when the flag is set while the USIOIE bit in USICR and the Global Interrupt Enable Flag are set. The flag will only be cleared if a one is written to the USIOIF bit. Clearing this bit will release the counter overflow hold of SCL in Two-wire mode.

A counter overflow interrupt will wakeup the processor from Idle sleep mode.

- **Bit 5 – USIPF: Stop Condition Flag**

When Two-wire mode is selected, the USIPF Flag is set (one) when a stop condition is detected. The flag is cleared by writing a one to this bit. Note that this is not an Interrupt Flag. This signal is useful when implementing Two-wire bus master arbitration.

- **Bit 4 – USIDC: Data Output Collision**

This bit is logical one when bit 7 in the Shift Register differs from the physical pin value. The flag is only valid when Two-wire mode is used. This signal is useful when implementing Two-wire bus master arbitration.

- **Bits 3..0 – USICNT3..0: Counter Value**

These bits reflect the current 4-bit counter value. The 4-bit counter value can directly be read or written by the CPU.

The 4-bit counter increments by one for each clock generated either by the external clock edge detector, by a Timer/Counter0 Compare Match, or by software using USICLK or USITC strobe bits. The clock source depends of the setting of the USICS1..0 bits. For external clock operation a special feature is added that allows the clock to be generated by writing to the USITC strobe bit. This feature is enabled by write a one to the USICLK bit while setting an external clock source (USICS1 = 1).

Note that even when no wire mode is selected (USIWM1..0 = 0) the external clock input (USCK/SCL) are can still be used by the counter.

USI Control Register – USICR

Bit	7	6	5	4	3	2	1	0	
	USISIE	USIOIE	USIWM1	USIWM0	USICS1	USICS0	USICLK	USITC	USICR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	W	W	
Initial Value	0	0	0	0	0	0	0	0	

The Control Register includes interrupt enable control, wire mode setting, Clock Select setting, and clock strobe.

- **Bit 7 – USISIE: Start Condition Interrupt Enable**

Setting this bit to one enables the Start Condition detector interrupt. If there is a pending interrupt when the USISIE and the Global Interrupt Enable Flag is set to one, this will immediately be executed. Refer to the USISIF bit description on page 186 for further details.

- **Bit 6 – USIOIE: Counter Overflow Interrupt Enable**

Setting this bit to one enables the Counter Overflow interrupt. If there is a pending interrupt when the USIOIE and the Global Interrupt Enable Flag is set to one, this will immediately be executed. Refer to the USIOIF bit description on page 186 for further details.

- **Bit 5..4 – USIWM1..0: Wire Mode**

These bits set the type of wire mode to be used. Basically only the function of the outputs are affected by these bits. Data and clock inputs are not affected by the mode selected and will always have the same function. The counter and Shift Register can therefore be clocked externally, and data input sampled, even when outputs are disabled. The relations between USIWM1..0 and the USI operation is summarized in Table 83.

Table 83. Relations between USIWM1..0 and the USI Operation

USIWM1	USIWM0	Description
0	0	Outputs, clock hold, and start detector disabled. Port pins operates as normal.
0	1	Three-wire mode. Uses DO, DI, and USCK pins. The *Data Output* (DO) pin overrides the corresponding bit in the PORT Register in this mode. However, the corresponding DDR bit still controls the data direction. When the port pin is set as input the pins pull-up is controlled by the PORT bit. The *Data Input* (DI) and *Serial Clock* (USCK) pins do not affect the normal port operation. When operating as master, clock pulses are software generated by toggling the PORT Register, while the data direction is set to output. The USITC bit in the USICR Register can be used for this purpose.
1	0	Two-wire mode. Uses SDA (DI) and SCL (USCK) pins(1). The *Serial Data* (SDA) and the *Serial Clock* (SCL) pins are bi-directional and uses open-collector output drives. The output drivers are enabled by setting the corresponding bit for SDA and SCL in the DDR Register. When the output driver is enabled for the SDA pin, the output driver will force the line SDA low if the output of the Shift Register or the corresponding bit in the PORT Register is zero. Otherwise the SDA line will not be driven (i.e., it is released). When the SCL pin output driver is enabled the SCL line will be forced low if the corresponding bit in the PORT Register is zero, or by the start detector. Otherwise the SCL line will not be driven. The SCL line is held low when a start detector detects a start condition and the output is enabled. Clearing the Start Condition Flag (USISIF) releases the line. The SDA and SCL pin inputs is not affected by enabling this mode. Pull-ups on the SDA and SCL port pin are disabled in Two-wire mode.
1	1	Two-wire mode. Uses SDA and SCL pins. Same operation as for the Two-wire mode described above, except that the SCL line is also held low when a counter overflow occurs, and is held low until the Counter Overflow Flag (USIOIF) is cleared.

Note: 1. The DI and USCK pins are renamed to *Serial Data* (SDA) and *Serial Clock* (SCL) respectively to avoid confusion between the modes of operation.

- **Bit 3..2 – USICS1..0: Clock Source Select**

These bits set the clock source for the Shift Register and counter. The data output latch ensures that the output is changed at the opposite edge of the sampling of the data input (DI/SDA) when using external clock source (USCK/SCL). When software strobe or Timer/Counter0 Compare Match clock option is selected, the output latch is transparent and therefore the output is changed immediately. Clearing the USICS1..0 bits enables software strobe option. When using this option, writing a one to the USICLK bit clocks both the Shift Register and the counter. For external clock source (USICS1 = 1), the USICLK bit is no longer used as a strobe, but selects between external clocking and software clocking by the USITC strobe bit.

Table 84 shows the relationship between the USICS1..0 and USICLK setting and clock source used for the Shift Register and the 4-bit counter.

Table 84. Relations between the USICS1..0 and USICLK Setting

USICS1	USICS0	USICLK	Shift Register Clock Source	4-bit Counter Clock Source
0	0	0	No Clock	No Clock
0	0	1	Software clock strobe (USICLK)	Software clock strobe (USICLK)
0	1	X	Timer/Counter0 Compare Match	Timer/Counter0 Compare Match
1	0	0	External, positive edge	External, both edges
1	1	0	External, negative edge	External, both edges
1	0	1	External, positive edge	Software clock strobe (USITC)
1	1	1	External, negative edge	Software clock strobe (USITC)

- **Bit 1 – USICLK: Clock Strobe**

Writing a one to this bit location strobes the Shift Register to shift one step and the counter to increment by one, provided that the USICS1..0 bits are set to zero and by doing so the software clock strobe option is selected. The output will change immediately when the clock strobe is executed, i.e., in the same instruction cycle. The value shifted into the Shift Register is sampled the previous instruction cycle. The bit will be read as zero.

When an external clock source is selected (USICS1 = 1), the USICLK function is changed from a clock strobe to a Clock Select Register. Setting the USICLK bit in this case will select the USITC strobe bit as clock source for the 4-bit counter (see Table 84).

- **Bit 0 – USITC: Toggle Clock Port Pin**

Writing a one to this bit location toggles the USCK/SCL value either from 0 to 1, or from 1 to 0. The toggling is independent of the setting in the Data Direction Register, but if the PORT value is to be shown on the pin the DDRE4 must be set as output (to one). This feature allows easy clock generation when implementing master devices. The bit will be read as zero.

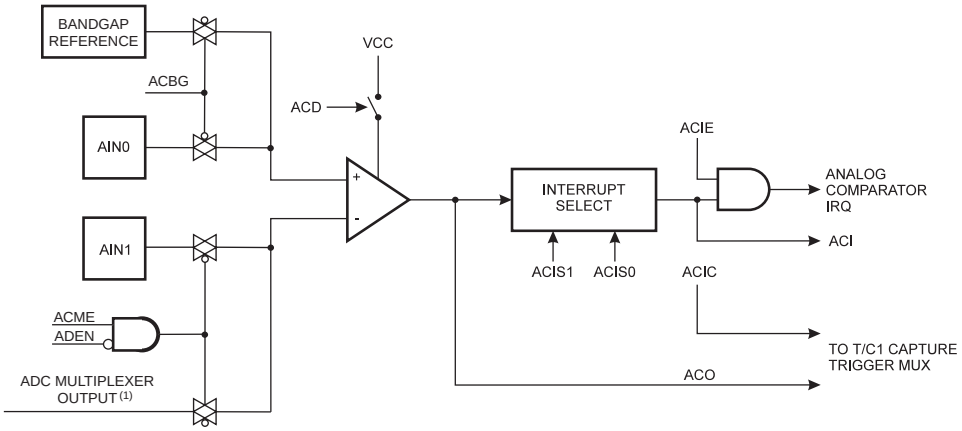
When an external clock source is selected (USICS1 = 1) and the USICLK bit is set to one, writing to the USITC strobe bit will directly clock the 4-bit counter. This allows an early detection of when the transfer is done when operating as a master device.

Analog Comparator

The Analog Comparator compares the input values on the positive pin AIN0 and negative pin AIN1. When the voltage on the positive pin AIN0 is higher than the voltage on the negative pin AIN1, the Analog Comparator output, ACO, is set. The comparator's output can be set to trigger the Timer/Counter1 Input Capture function. In addition, the comparator can trigger a separate interrupt, exclusive to the Analog Comparator. The user can select Interrupt triggering on comparator output rise, fall or toggle. A block diagram of the comparator and its surrounding logic is shown in Figure 82.

The Power Reduction ADC bit, PRADC, in "Power Reduction Register - PRR" on page 34 must be disabled by writing a logical zero to be able to use the ADC input MUX.

Figure 82. Analog Comparator Block Diagram(2)



Notes: 1. See Table 86 on page 192.
2. Refer to Figure 1 on page 2 and Table 29 on page 63 for Analog Comparator pin placement.

ADC Control and Status Register B – ADCSRB

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

- **Bit 6 – ACME: Analog Comparator Multiplexer Enable**

When this bit is written logic one and the ADC is switched off (ADEN in ADCSRA is zero), the ADC multiplexer selects the negative input to the Analog Comparator. When this bit is written logic zero, AIN1 is applied to the negative input of the Analog Comparator. For a detailed description of this bit, see "Analog Comparator Multiplexed Input" on page 192.

Analog Comparator Control and Status Register – ACSR

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

- **Bit 7 – ACD: Analog Comparator Disable**

When this bit is written logic one, the power to the Analog Comparator is switched off. This bit can be set at any time to turn off the Analog Comparator. This will reduce power

consumption in Active and Idle mode. When changing the ACD bit, the Analog Comparator Interrupt must be disabled by clearing the ACIE bit in ACSR. Otherwise an interrupt can occur when the bit is changed.

- **Bit 6 – ACBG: Analog Comparator Bandgap Select**

When this bit is set, a fixed bandgap reference voltage replaces the positive input to the Analog Comparator. When this bit is cleared, AIN0 is applied to the positive input of the Analog Comparator. When the bandgap reference is used as input to the Analog Comparator, it will take a certain time for the voltage to stabilize. If not stabilized, the first conversion may give a wrong value. See “Internal Voltage Reference” on page 42.

- **Bit 5 – ACO: Analog Comparator Output**

The output of the Analog Comparator is synchronized and then directly connected to ACO. The synchronization introduces a delay of 1 - 2 clock cycles.

- **Bit 4 – ACI: Analog Comparator Interrupt Flag**

This bit is set by hardware when a comparator output event triggers the interrupt mode defined by ACIS1 and ACIS0. The Analog Comparator interrupt routine is executed if the ACIE bit is set and the I-bit in SREG is set. ACI is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ACI is cleared by writing a logic one to the flag.

- **Bit 3 – ACIE: Analog Comparator Interrupt Enable**

When the ACIE bit is written logic one and the I-bit in the Status Register is set, the Analog Comparator interrupt is activated. When written logic zero, the interrupt is disabled.

- **Bit 2 – ACIC: Analog Comparator Input Capture Enable**

When written logic one, this bit enables the Input Capture function in Timer/Counter1 to be triggered by the Analog Comparator. The comparator output is in this case directly connected to the Input Capture front-end logic, making the comparator utilize the noise canceler and edge select features of the Timer/Counter1 Input Capture interrupt. When written logic zero, no connection between the Analog Comparator and the Input Capture function exists. To make the comparator trigger the Timer/Counter1 Input Capture interrupt, the ICIE1 bit in the Timer Interrupt Mask Register (TIMSK1) must be set.

- **Bits 1, 0 – ACIS1, ACIS0: Analog Comparator Interrupt Mode Select**

These bits determine which comparator events that trigger the Analog Comparator interrupt. The different settings are shown in Table 85.

Table 85. ACIS1/ACIS0 Settings

ACIS1	ACIS0	Interrupt Mode
0	0	Comparator Interrupt on Output Toggle.
0	1	Reserved
1	0	Comparator Interrupt on Falling Output Edge.
1	1	Comparator Interrupt on Rising Output Edge.

When changing the ACIS1/ACIS0 bits, the Analog Comparator Interrupt must be disabled by clearing its Interrupt Enable bit in the ACSR Register. Otherwise an interrupt can occur when the bits are changed.

Analog Comparator Multiplexed Input

It is possible to select any of the ADC7..0 pins to replace the negative input to the Analog Comparator. The ADC multiplexer is used to select this input, and consequently, the ADC must be switched off to utilize this feature. If the Analog Comparator Multiplexer Enable bit (ACME in ADCSRB) is set and the ADC is switched off (ADEN in ADCSRA is zero), MUX2..0 in ADMUX select the input pin to replace the negative input to the Analog Comparator, as shown in Table 86. If ACME is cleared or ADEN is set, AIN1 is applied to the negative input to the Analog Comparator.

Table 86. Analog Comparator Multiplexed Input

ACME	ADEN	MUX2..0	Analog Comparator Negative Input
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6
1	0	111	ADC7

Digital Input Disable Register 1 – DIDR1

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	AIN1D	AIN0D	DIDR1
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 1, 0 – AIN1D, AIN0D: AIN1, AIN0 Digital Input Disable**

When this bit is written logic one, the digital input buffer on the AIN1/0 pin is disabled. The corresponding PIN Register bit will always read as zero when this bit is set. When an analog signal is applied to the AIN1/0 pin and the digital input from this pin is not needed, this bit should be written logic one to reduce power consumption in the digital input buffer.

Analog to Digital Converter

Features

- **10-bit Resolution**
- **0.5 LSB Integral Non-linearity**
- **± 2 LSB Absolute Accuracy**
- **13 µs - 260 µs Conversion Time (50 kHz to 1 MHz ADC clock)**
- **Up to 15 kSPS at Maximum Resolution (200 kHz ADC clock)**
- **Eight Multiplexed Single Ended Input Channels**
- **Optional Left Adjustment for ADC Result Readout**
- **0 - V_{CC} ADC Input Voltage Range**
- **Selectable 1.1V ADC Reference Voltage**
- **Free Running or Single Conversion Mode**
- **ADC Start Conversion by Auto Triggering on Interrupt Sources**
- **Interrupt on ADC Conversion Complete**
- **Sleep Mode Noise Canceler**

The ATmega169 features a 10-bit successive approximation ADC. The ADC is connected to an 8-channel Analog Multiplexer which allows eight single-ended voltage inputs constructed from the pins of Port F. The single-ended voltage inputs refer to 0V (GND).

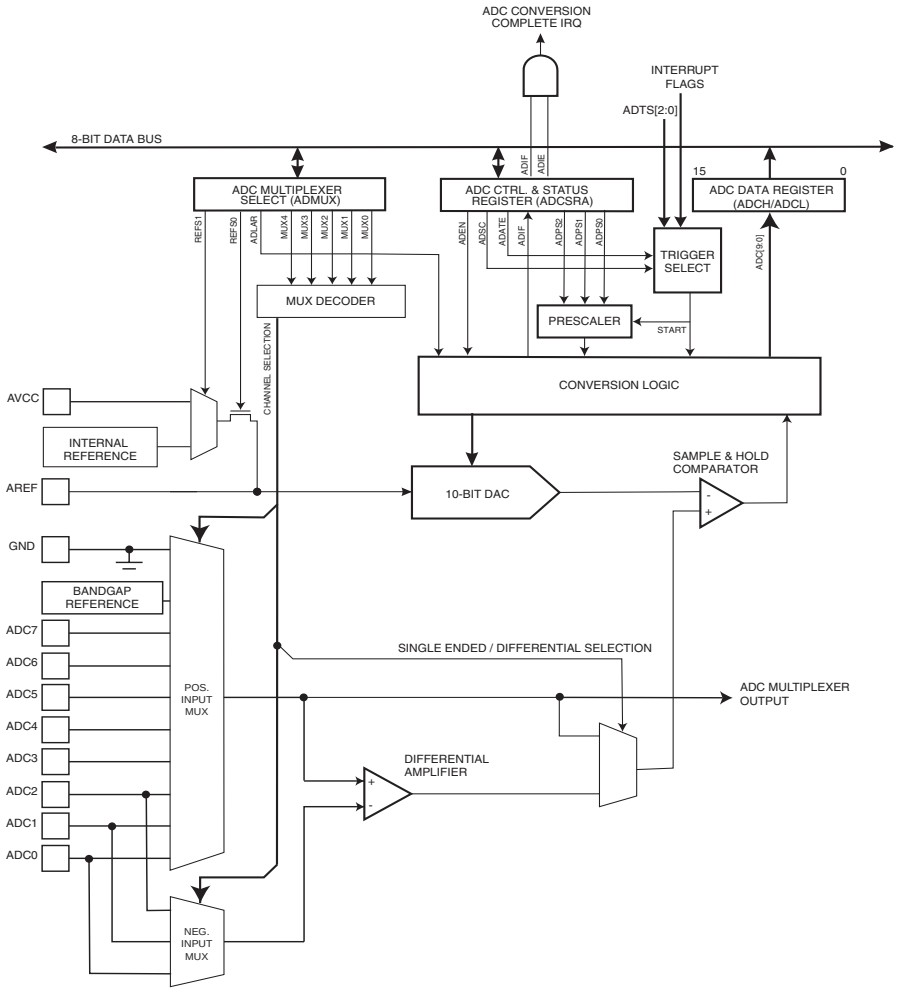
The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion. A block diagram of the ADC is shown in Figure 83.

The ADC has a separate analog supply voltage pin, AVCC. AVCC must not differ more than ± 0.3V from V_{CC}. See the paragraph “ADC Noise Canceler” on page 200 on how to connect this pin.

Internal reference voltages of nominally 1.1V or AVCC are provided On-chip. The voltage reference may be externally decoupled at the AREF pin by a capacitor for better noise performance.

The Power Reduction ADC bit, PRADC, in “Power Reduction Register - PRR” on page 34 must be written to zero to enable the ADC module.

Figure 83. Analog to Digital Converter Block Schematic



Operation

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, AVCC or an internal 1.1V reference voltage may be connected to the AREF pin by writing to the REFSn bits in the ADMUX Register. The internal voltage reference may thus be decoupled by an external capacitor at the AREF pin to improve noise immunity.

The analog input channel is selected by writing to the MUX bits in ADMUX. Any of the ADC input pins, as well as GND and a fixed bandgap voltage reference, can be selected as single ended inputs to the ADC. The ADC is enabled by setting the ADC Enable bit, ADEN in ADCSRA. Voltage reference and input channel selections will not go into effect until ADEN is set. The ADC does not consume power when ADEN is cleared, so it is recommended to switch off the ADC before entering power saving sleep modes.

The ADC generates a 10-bit result which is presented in the ADC Data Registers, ADCH and ADCL. By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the ADLAR bit in ADMUX.

If the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH, to ensure that the content of the Data Registers belongs to the same conversion. Once ADCL is read, ADC access

to Data Registers is blocked. This means that if ADCL has been read, and a conversion completes before ADCH is read, neither register is updated and the result from the conversion is lost. When ADCH is read, ADC access to the ADCH and ADCL Registers is re-enabled.

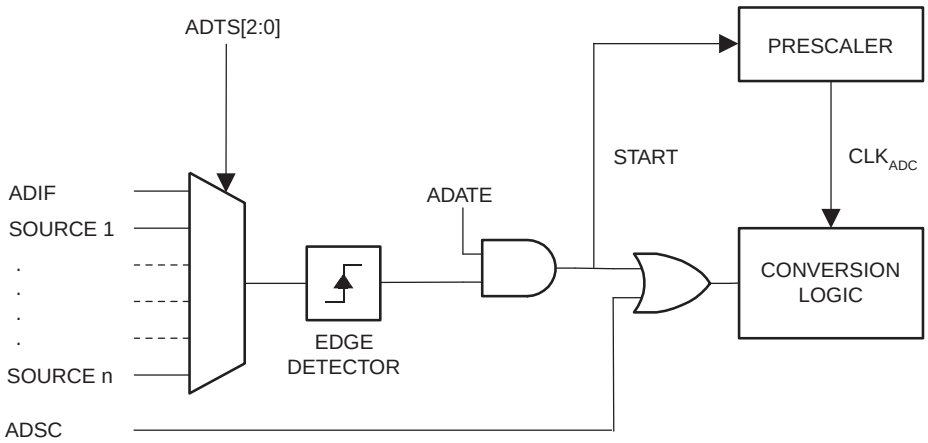
The ADC has its own interrupt which can be triggered when a conversion completes. When ADC access to the Data Registers is prohibited between reading of ADCH and ADCL, the interrupt will trigger even if the result is lost.

Starting a Conversion

A single conversion is started by writing a logical one to the ADC Start Conversion bit, ADSC. This bit stays high as long as the conversion is in progress and will be cleared by hardware when the conversion is completed. If a different data channel is selected while a conversion is in progress, the ADC will finish the current conversion before performing the channel change.

Alternatively, a conversion can be triggered automatically by various sources. Auto Triggering is enabled by setting the ADC Auto Trigger Enable bit, ADATE in ADCSRA. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in ADCSRB (See description of the ADTS bits for a list of the trigger sources). When a positive edge occurs on the selected trigger signal, the ADC prescaler is reset and a conversion is started. This provides a method of starting conversions at fixed intervals. If the trigger signal still is set when the conversion completes, a new conversion will not be started. If another positive edge occurs on the trigger signal during conversion, the edge will be ignored. Note that an Interrupt Flag will be set even if the specific interrupt is disabled or the Global Interrupt Enable bit in SREG is cleared. A conversion can thus be triggered without causing an interrupt. However, the Interrupt Flag must be cleared in order to trigger a new conversion at the next interrupt event.

Figure 84. ADC Auto Trigger Logic



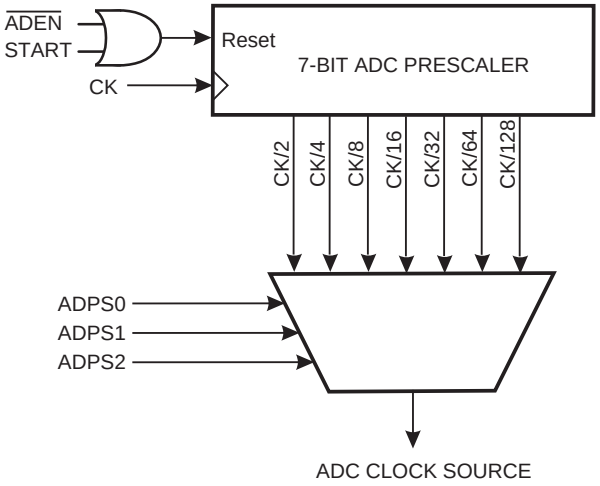
Using the ADC Interrupt Flag as a trigger source makes the ADC start a new conversion as soon as the ongoing conversion has finished. The ADC then operates in Free Running mode, constantly sampling and updating the ADC Data Register. The first conversion must be started by writing a logical one to the ADSC bit in ADCSRA. In this mode the ADC will perform successive conversions independently of whether the ADC Interrupt Flag, ADIF is cleared or not.

If Auto Triggering is enabled, single conversions can be started by writing ADSC in ADCSRA to one. ADSC can also be used to determine if a conversion is in progress.

The ADSC bit will be read as one during a conversion, independently of how the conversion was started.

Prescaling and Conversion Timing

Figure 85. ADC Prescaler



By default, the successive approximation circuitry requires an input clock frequency between 50 kHz and 200 kHz to get maximum resolution. If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200 kHz to get a higher sample rate.

The ADC module contains a prescaler, which generates an acceptable ADC clock frequency from any CPU frequency above 100 kHz. The prescaling is set by the ADPS bits in ADCSRA. The prescaler starts counting from the moment the ADC is switched on by setting the ADEN bit in ADCSRA. The prescaler keeps running for as long as the ADEN bit is set, and is continuously reset when ADEN is low.

When initiating a single ended conversion by setting the ADSC bit in ADCSRA, the conversion starts at the following rising edge of the ADC clock cycle.

A normal conversion takes 13 ADC clock cycles. The first conversion after the ADC is switched on (ADEN in ADCSRA is set) takes 25 ADC clock cycles in order to initialize the analog circuitry.

When the bandgap reference voltage is used as input to the ADC, it will take a certain time for the voltage to stabilize. If not stabilized, the first value read after the first conversion may be wrong.

The actual sample-and-hold takes place 1.5 ADC clock cycles after the start of a normal conversion and 13.5 ADC clock cycles after the start of an first conversion. When a conversion is complete, the result is written to the ADC Data Registers, and ADIF is set. In Single Conversion mode, ADSC is cleared simultaneously. The software may then set ADSC again, and a new conversion will be initiated on the first rising ADC clock edge.

When Auto Triggering is used, the prescaler is reset when the trigger event occurs. This assures a fixed delay from the trigger event to the start of conversion. In this mode, the sample-and-hold takes place two ADC clock cycles after the rising edge on the trigger source signal. Three additional CPU clock cycles are used for synchronization logic. When using Differential mode, along with Auto triggering from a source other than the ADC Conversion Complete, each conversion will require 25 ADC clocks. This is because the ADC must be disabled and re-enabled after every conversion.

In Free Running mode, a new conversion will be started immediately after the conversion completes, while ADSC remains high. For a summary of conversion times, see Table 87.

Figure 86. ADC Timing Diagram, First Conversion (Single Conversion Mode)

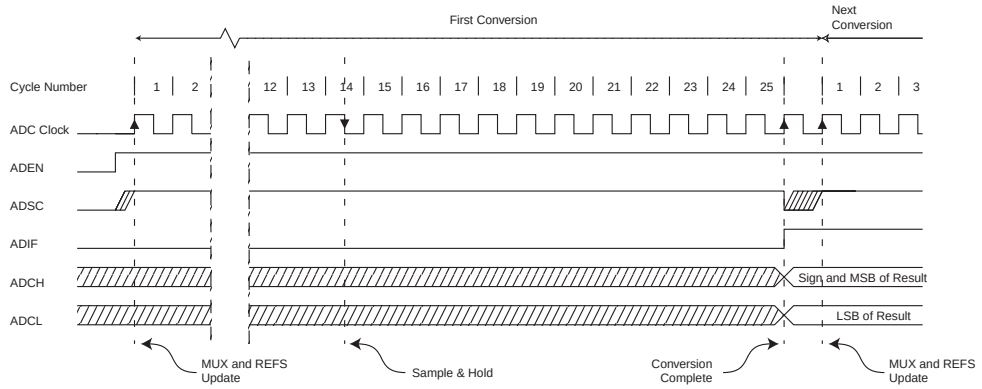


Figure 87. ADC Timing Diagram, Single Conversion

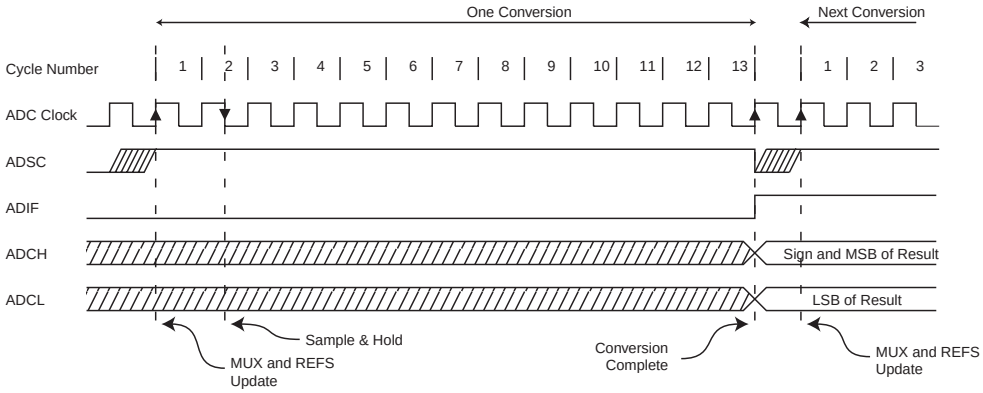


Figure 88. ADC Timing Diagram, Auto Triggered Conversion

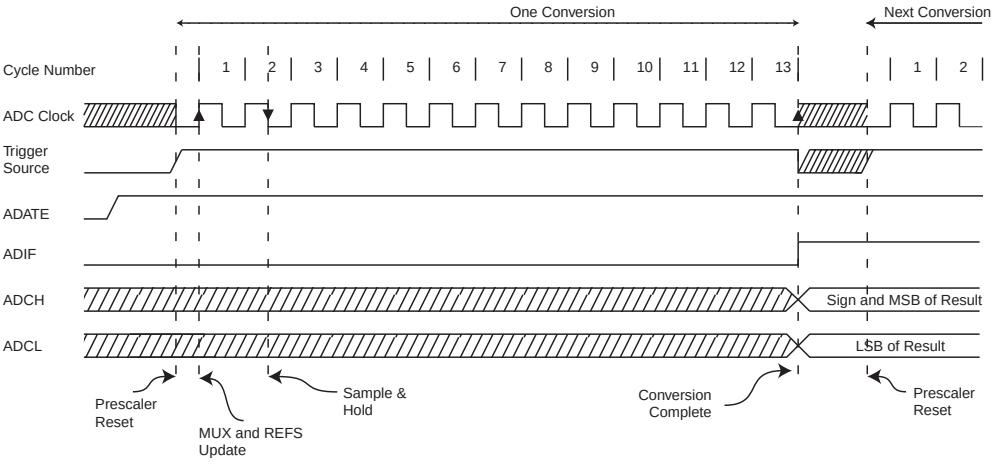


Figure 89. ADC Timing Diagram, Free Running Conversion

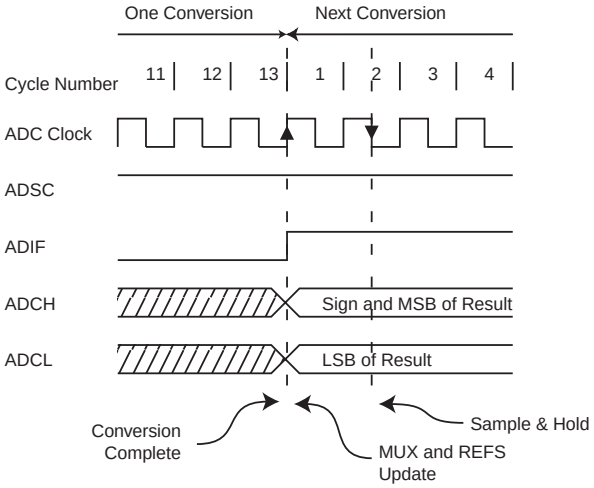


Table 87. ADC Conversion Time

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

Changing Channel or Reference Selection

The MUXn and REFS1:0 bits in the ADMUX Register are single buffered through a temporary register to which the CPU has random access. This ensures that the channels and reference selection only takes place at a safe point during the conversion. The channel and reference selection is continuously updated until a conversion is started. Once the conversion starts, the channel and reference selection is locked to ensure a sufficient sampling time for the ADC. Continuous updating resumes in the last ADC clock cycle before the conversion completes (ADIF in ADCSRA is set). Note that the conversion starts on the following rising ADC clock edge after ADSC is written. The user is thus advised not to write new channel or reference selection values to ADMUX until one ADC clock cycle after ADSC is written.

If Auto Triggering is used, the exact time of the triggering event can be indeterministic. Special care must be taken when updating the ADMUX Register, in order to control which conversion will be affected by the new settings.

If both ADATE and ADEN is written to one, an interrupt event can occur at any time. If the ADMUX Register is changed in this period, the user cannot tell if the next conversion is based on the old or the new settings. ADMUX can be safely updated in the following ways:

1. When ADATE or ADEN is cleared.
2. During conversion, minimum one ADC clock cycle after the trigger event.
3. After a conversion, before the Interrupt Flag used as trigger source is cleared.

When updating ADMUX in one of these conditions, the new settings will affect the next ADC conversion.

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.

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.

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:

1. 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.
2. Enter ADC Noise Reduction mode (or Idle mode). The ADC will start a conversion once the CPU has been halted.
3. 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.

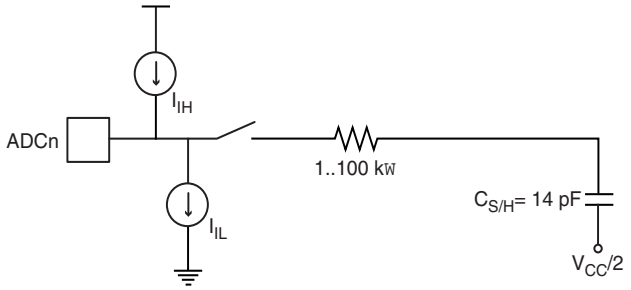
Analog Input Circuitry

The analog input circuitry for single ended channels is illustrated in Figure 90. An analog source applied to ADCn is subjected to the pin capacitance and input leakage of that pin, regardless of whether that channel is selected as input for the ADC. When the channel is selected, the source must drive the S/H capacitor through the series resistance (combined resistance in the input path).

The ADC is optimized for analog signals with an output impedance of approximately 10 kΩ or less. If such a source is used, the sampling time will be negligible. If a source with higher impedance is used, the sampling time will depend on how long time the source needs to charge the S/H capacitor, with can vary widely. The user is recommended to only use low impedant sources with slowly varying signals, since this minimizes the required charge transfer to the S/H capacitor.

Signal components higher than the Nyquist frequency ($f_{ADC}/2$) should not be present for either kind of channels, to avoid distortion from unpredictable signal convolution. The user is advised to remove high frequency components with a low-pass filter before applying the signals as inputs to the ADC.

Figure 90. Analog Input Circuitry

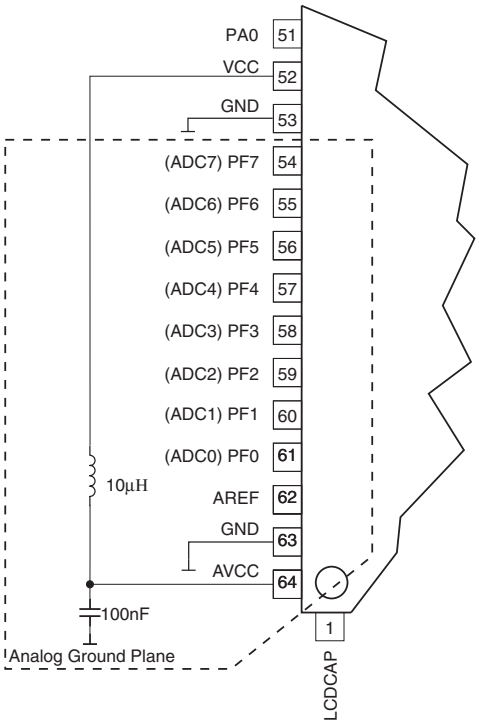


Analog Noise Canceling Techniques

Digital circuitry inside and outside the device generates EMI which might affect the accuracy of analog measurements. If conversion accuracy is critical, the noise level can be reduced by applying the following techniques:

1. Keep analog signal paths as short as possible. Make sure analog tracks run over the analog ground plane, and keep them well away from high-speed switching digital tracks.
2. The AVCC pin on the device should be connected to the digital V_{CC} supply voltage via an LC network as shown in Figure 91.
3. Use the ADC noise canceler function to reduce induced noise from the CPU.
4. If any ADC port pins are used as digital outputs, it is essential that these do not switch while a conversion is in progress.

Figure 91. ADC Power Connections



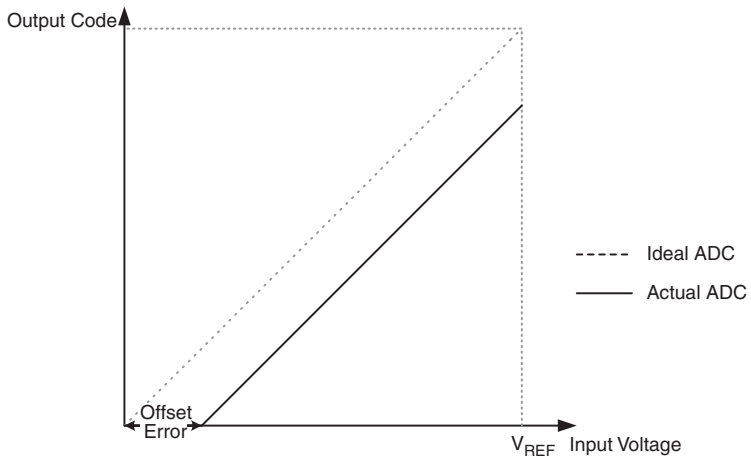
ADC Accuracy Definitions

An n-bit single-ended ADC converts a voltage linearly between GND and V_{REF} in 2^n steps (LSBs). The lowest code is read as 0, and the highest code is read as 2^n-1.

Several parameters describe the deviation from the ideal behavior:

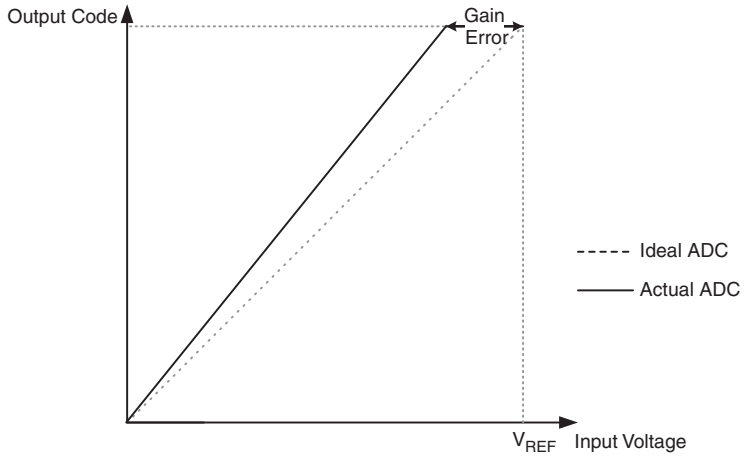
- Offset: The deviation of the first transition (0x000 to 0x001) compared to the ideal transition (at 0.5 LSB). Ideal value: 0 LSB.

Figure 92. Offset Error



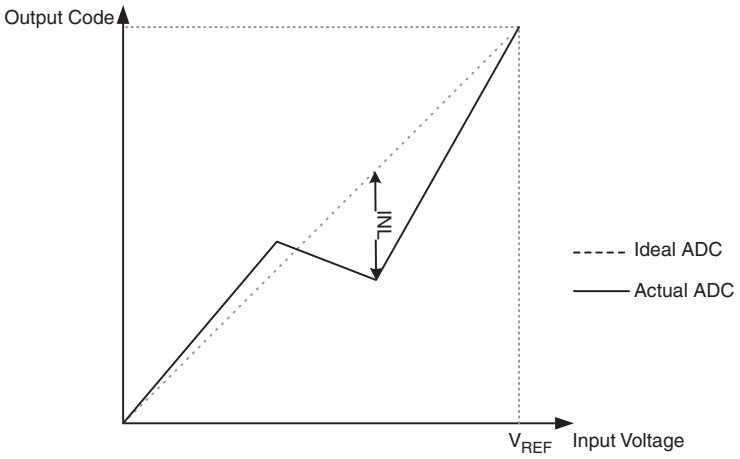
- Gain Error: After adjusting for offset, the Gain Error is found as the deviation of the last transition (0x3FE to 0x3FF) compared to the ideal transition (at 1.5 LSB below maximum). Ideal value: 0 LSB

Figure 93. Gain Error



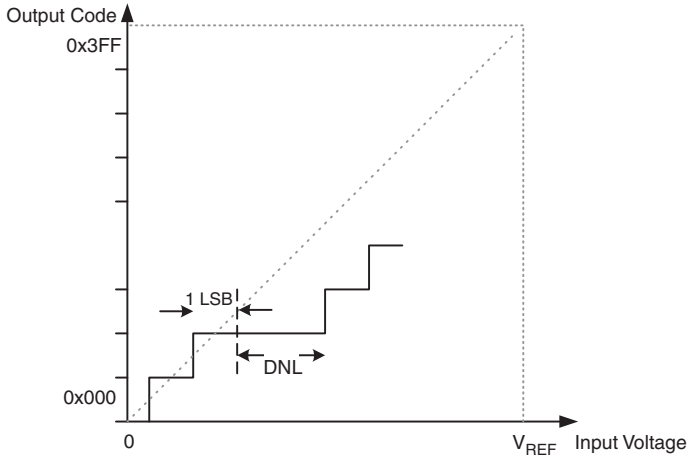
- Integral Non-linearity (INL): After adjusting for offset and gain error, the INL is the maximum deviation of an actual transition compared to an ideal transition for any code. Ideal value: 0 LSB.

Figure 94. Integral Non-linearity (INL)



- Differential Non-linearity (DNL): The maximum deviation of the actual code width (the interval between two adjacent transitions) from the ideal code width (1 LSB). Ideal value: 0 LSB.

Figure 95. Differential Non-linearity (DNL)



- Quantization Error: Due to the quantization of the input voltage into a finite number of codes, a range of input voltages (1 LSB wide) will code to the same value. Always ± 0.5 LSB.
- Absolute Accuracy: The maximum deviation of an actual (unadjusted) transition compared to an ideal transition for any code. This is the compound effect of offset, gain error, differential error, non-linearity, and quantization error. Ideal value: ± 0.5 LSB.

ADC Conversion Result

After the conversion is complete (ADIF is high), the conversion result can be found in the ADC Result Registers (ADCL, ADCH).

For single ended conversion, the result is

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

where V_{IN} is the voltage on the selected input pin and V_{REF} the selected voltage reference (see Table 89 on page 205 and Table 90 on page 206). 0x000 represents analog ground, and 0x3FF represents the selected reference voltage minus one LSB.

$$ADC = \frac{(V_{POS} - V_{NEG}) \cdot 512}{V_{REF}}$$

Figure 96. Differential Measurement Range

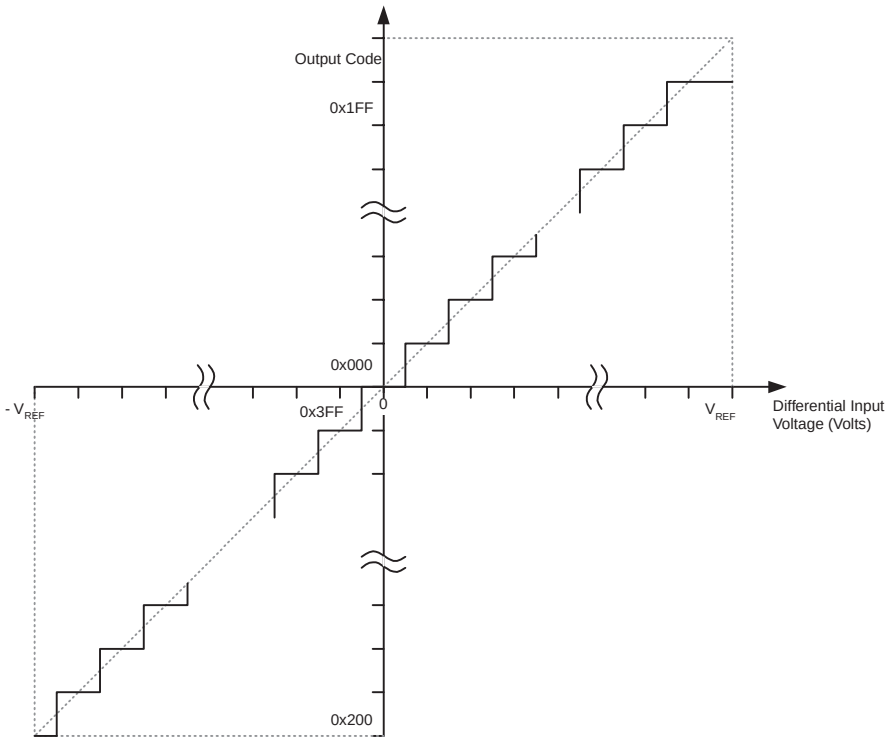


Table 88. Correlation Between Input Voltage and Output Codes

V_{ADCn}	Read Code	Corresponding Decimal Value
$V_{ADCm} + V_{REF}$	0x1FF	511
$V_{ADCm} + {}^{511}/_{512}\ V_{REF}$	0x1FF	511
$V_{ADCm} + {}^{510}/_{512}\ V_{REF}$	0x1FE	510
...	...	...
$V_{ADCm} + {}^{1}/_{512}\ V_{REF}$	0x001	1
V_{ADCm}	0x000	0
$V_{ADCm} - {}^{1}/_{512}\ V_{REF}$	0x3FF	-1
...	...	...
$V_{ADCm} - {}^{511}/_{512}\ V_{REF}$	0x201	-511
$V_{ADCm} - V_{REF}$	0x200	-512

ADMUX = 0xFB (ADC3 - ADC2, 1.1V reference, left adjusted result)

Voltage on ADC3 is 300 mV, voltage on ADC2 is 500 mV.

ADCR = 512 * (300 - 500) / 1100 = -93 = 0x3A3.

ADCL will thus read 0xC0, and ADCH will read 0xD8. Writing zero to ADLAR right adjusts the result: ADCL = 0xA3, ADCH = 0x03.

ADC Multiplexer Selection Register – ADMUX

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
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	

- **Bit 7:6 – REFS1:0: Reference Selection Bits**

These bits select the voltage reference for the ADC, as shown in Table 89. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

Table 89. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V Voltage Reference with external capacitor at AREF pin

- **Bit 5 – ADLAR: ADC Left Adjust Result**

The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC Data Register immediately, regardless of any ongoing conversions. For a complete description of this bit, see “The ADC Data Register – ADCL and ADCH” on page 208.

- **Bits 4:0 – MUX4:0: Analog Channel Selection Bits**

The value of these bits selects which combination of analog inputs are connected to the ADC. See Table 90 for details. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set).

Table 90. Input Channel Selections

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input
00000	ADC0	N/A	
00001	ADC1		
00010	ADC2		
00011	ADC3		
00100	ADC4		
00101	ADC5		
00110	ADC6		
00111	ADC7		
01000	N/A		
01001			
01010			
01011			
01100			
01101			
01110			
01111			
10000		ADC0	ADC1
10001		ADC1	ADC1
10010		ADC2	ADC1
10011		ADC3	ADC1
10100		ADC4	ADC1
10101		ADC5	ADC1
10110		ADC6	ADC1
10111		ADC7	ADC1
11000		ADC0	ADC2
11001		ADC1	ADC2
11010		ADC2	ADC2
11011		ADC3	ADC2
11100		ADC4	ADC2
11101		ADC5	ADC2
11110	1.1V (V_{BG})	N/A	
11111	0V (GND)		

ADC Control and Status Register A – ADCSRA

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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	

- **Bit 7 – ADEN: ADC Enable**

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

- **Bit 6 – ADSC: ADC Start Conversion**

In Single Conversion mode, write this bit to one to start each conversion. In Free Running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

- **Bit 5 – ADATE: ADC Auto Trigger Enable**

When this bit is written to one, Auto Triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in ADCSRB.

- **Bit 4 – ADIF: ADC Interrupt Flag**

This bit is set when an ADC conversion completes and the Data Registers are updated. The ADC Conversion Complete Interrupt is executed if the ADIE bit and the I-bit in SREG are set. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag. Beware that if doing a Read-Modify-Write on ADCSRA, a pending interrupt can be disabled. This also applies if the SBI and CBI instructions are used.

- **Bit 3 – ADIE: ADC Interrupt Enable**

When this bit is written to one and the I-bit in SREG is set, the ADC Conversion Complete Interrupt is activated.

- **Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits**

These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

Table 91. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	0	1	64
1	1	1	128

The ADC Data Register – ADCL and ADCH

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

When an ADC conversion is complete, the result is found in these two registers.When ADCL is read, the ADC Data Register is not updated until ADCH is read. Consequently, if the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH.

The ADLAR bit in ADMUX, and the MUXn bits in ADMUX affect the way the result is read from the registers. If ADLAR is set, the result is left adjusted. If ADLAR is cleared (default), the result is right adjusted.

- **ADC9:0: ADC Conversion Result**

These bits represent the result from the conversion, as detailed in “ADC Conversion Result” on page 204.

ADC Control and Status Register B – ADCSRB

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

- **Bit 7 – Res: Reserved Bit**

This bit is reserved for future use. To ensure compatibility with future devices, this bit must be written to zero when ADCSRB is written.

- **Bit 2:0 – ADTS2:0: ADC Auto Trigger Source**

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 ADTS2:0 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 92. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

Digital Input Disable Register 0 – DIDR0

Bit	7	6	5	4	3	2	1	0	
	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	DIDR0
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	

- **Bit 7..0 – ADC7D..ADC0D: ADC7..0 Digital Input Disable**

When this bit is written logic one, the digital input buffer on the corresponding ADC pin is disabled. The corresponding PIN Register bit will always read as zero when this bit is set. When an analog signal is applied to the ADC7..0 pin and the digital input from this pin is not needed, this bit should be written logic one to reduce power consumption in the digital input buffer.

LCD Controller

The LCD Controller/driver is intended for monochrome passive liquid crystal display (LCD) with up to four common terminals and up to 25 segment terminals.

Features

- **Display Capacity of 25 Segments and Four Common Terminals**
- **Support Static, 1/2, 1/3 and 1/4 Duty**
- **Support Static, 1/2, 1/3 Bias**
- **On-chip LCD Power Supply, only One External Capacitor needed**
- **Display Possible in Power-save Mode for Low Power Consumption**
- **Software Selectable Low Power Waveform Capability**
- **Flexible Selection of Frame Frequency**
- **Software Selection between System Clock or an External Asynchronous Clock Source**
- **Equal Source and Sink Capability to maximize LCD Life Time**
- **LCD Interrupt Can be Used for Display Data Update or Wake-up from Sleep Mode**
- **Segment and Common Pins not Needed for Driving the Display Can be Used as Ordinary I/O Pins**
- **Latching of Display Data gives Full Freedom in Register Update**

Overview

A simplified block diagram of the LCD Controller/Driver is shown in Figure 97. For the actual placement of I/O pins, see “Pinout ATmega169” on page 2.

An LCD consists of several segments (pixels or complete symbols) which can be visible or non visible. A segment has two electrodes with liquid crystal between them. When a voltage above a threshold voltage is applied across the liquid crystal, the segment becomes visible.

The voltage must alternate to avoid an electrophoresis effect in the liquid crystal, which degrades the display. Hence the waveform across a segment must not have a DC-component.

The PRLCD bit in “Power Reduction Register - PRR” on page 34 must be written to zero to enable the LCD module.

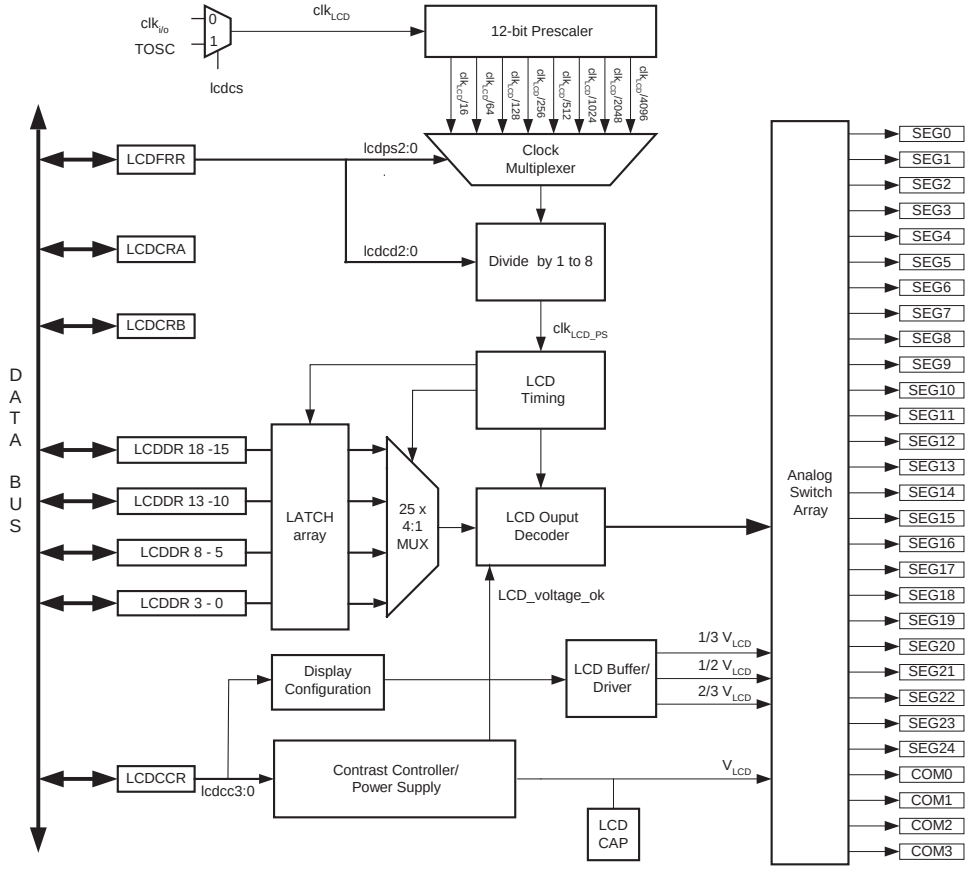
Definitions

Several terms are used when describing LCD. The definitions in Table 93 are used throughout this document.

Table 93. Definitions

LCD	A passive display panel with terminals leading directly to a segment
Segment	The least viewing element (pixel) which can be on or off
Common	Denotes how many segments are connected to a segment terminal
Duty	1/(Number of common terminals on a actual LCD display)
Bias	1/(Number of voltage levels used driving a LCD display -1)
Frame Rate	Number of times the LCD segments is energized per second.

Figure 97. LCD Module Block Diagram



LCD Clock Sources

The LCD Controller can be clocked by an internal synchronous or an external asynchronous clock source. The clock source clk_{LCD} is by default equal to the system clock, $clk_{I/O}$. When the LCDCS bit in the LCDCRB Register is written to logic one, the clock source is taken from the TOSC1 pin.

The clock source must be stable to obtain accurate LCD timing and hence minimize DC voltage offset across LCD segments.

LCD Prescaler

The prescaler consist of a 12-bit ripple counter and a 1- to 8-clock divider. The LCDPS2:0 bits selects clk_{LCD} divided by 16, 64, 128, 256, 512, 1024, 2048, or 4096.

If a finer resolution rate is required, the LCDCD2:0 bits can be used to divide the clock further by 1 to 8.

Output from the clock divider clk_{LCD_PS} is used as clock source for the LCD timing.

LCD Memory

The display memory is available through I/O Registers grouped for each common terminal. When a bit in the display memory is written to one, the corresponding segment is energized (on), and non-energized when a bit in the display memory is written to zero.

To energize a segment, an absolute voltage above a certain threshold must be applied. This is done by letting the output voltage on corresponding COM pin and SEG pin have opposite phase. For display with more than one common, one (1/2 bias) or two (1/3 bias) additional voltage levels must be applied. Otherwise, non-energized segments on COM0 would be energized for all non-selected common.

Addressing COM0 starts a frame by driving opposite phase with large amplitude out on COM0 compared to none addressed COM lines. Non-energized segments are in phase with the addressed COM0, and energized segments have opposite phase and large amplitude. For waveform figures refer to “Mode of Operation” on page 212. Latched data from LCDDR4 - LCDDR0 is multiplexed into the decoder. The decoder is controlled from the LCD timing and sets up signals controlling the analog switches to produce an output waveform. Next, COM1 is addressed, and latched data from LCDDR9 - LCDDR5 is input to decoder. Addressing continuous until all COM lines are addressed according to number of common (duty). The display data are latched before a new frame start.

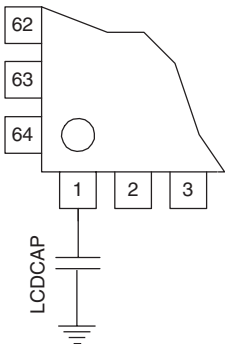
LCD Contrast Controller/Power Supply

The peak value (V_{LCD}) on the output waveform determines the LCD Contrast. V_{LCD} is controlled by software from 2.6V to 3.35V independent of V_{CC}. An internal signal inhibits output to the LCD until V_{LCD} has reached its target value.

LCDCAP

An external capacitor (typical > 470 nF) must be connected to the LCDCAP pin as shown in Figure 98. This capacitor acts as a reservoir for LCD power (V_{LCD}). A large capacitance reduces ripple on V_{LCD} but increases the time until VLCD reaches its target value.

Figure 98. LCDCAP Connection



LCD Buffer Driver

Intermediate voltage levels are generated from buffers/drivers. The buffers are active the amount of time specified by LCDDC[2:0] in “LCD Contrast Control Register – LCDCCR” on page 223. Then LCD output pins are tri-stated and buffers are switched off. Shortening the drive time will reduce power consumption, but displays with high internal resistance or capacitance may need longer drive time to achieve sufficient contrast.

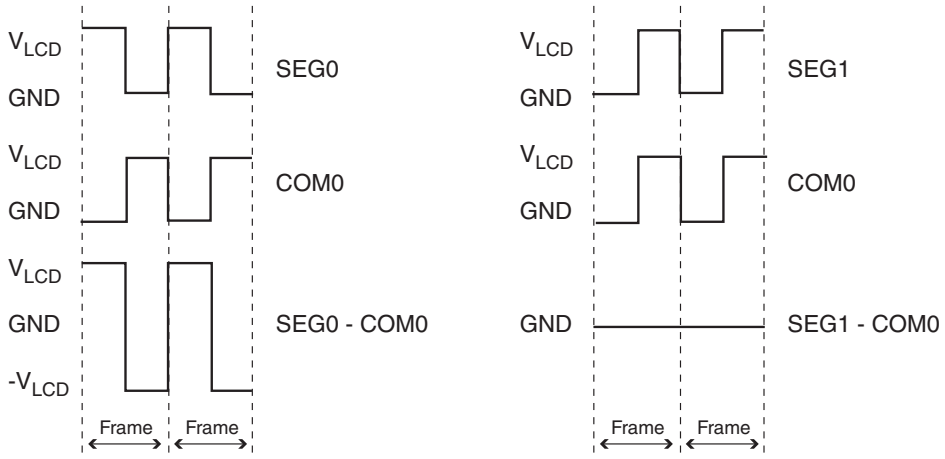
Mode of Operation

Static Duty and Bias

If all segments on a LCD have one electrode common, then each segment must have a unique terminal.

This kind of display is driven with the waveform shown in Figure 99. SEG0 - COM0 is the voltage across a segment that is on, and SEG1 - COM0 is the voltage across a segment that is off.

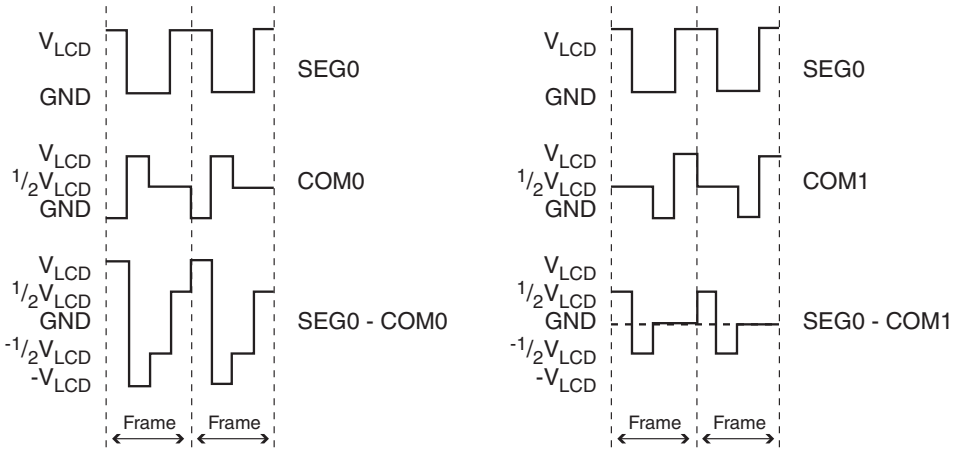
Figure 99. Driving a LCD with One Common Terminal



1/2 Duty and 1/2 Bias

For LCD with two common terminals (1/2 duty) a more complex waveform must be used to individually control segments. Although 1/3 bias can be selected 1/2 bias is most common for these displays. Waveform is shown in Figure 100. SEG0 - COM0 is the voltage across a segment that is on, and SEG0 - COM1 is the voltage across a segment that is off.

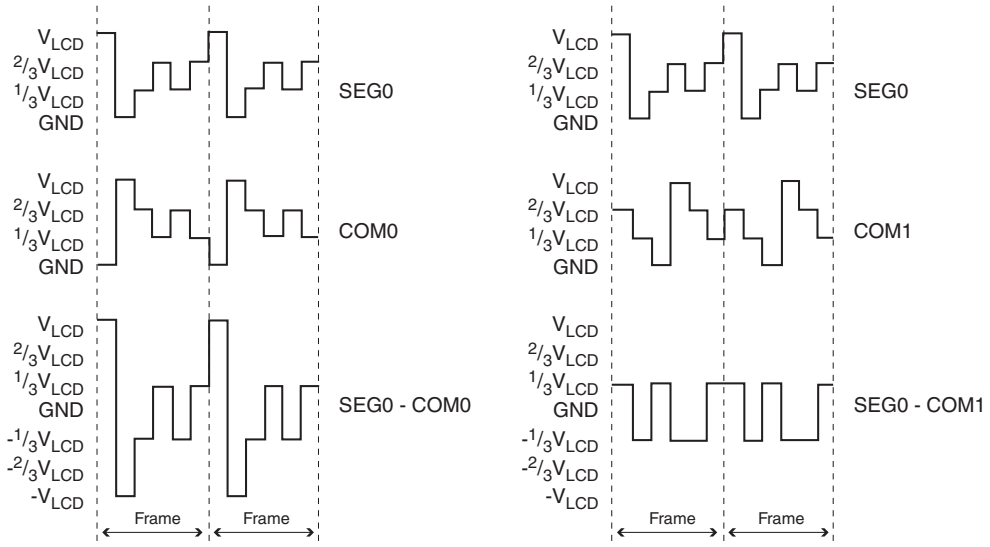
Figure 100. Driving a LCD with Two Common Terminals



1/3 Duty and 1/3 Bias

1/3 bias is usually recommended for LCD with three common terminals (1/3 duty). Waveform is shown in Figure 101. SEG0 - COM0 is the voltage across a segment that is on and SEG0-COM1 is the voltage across a segment that is off.

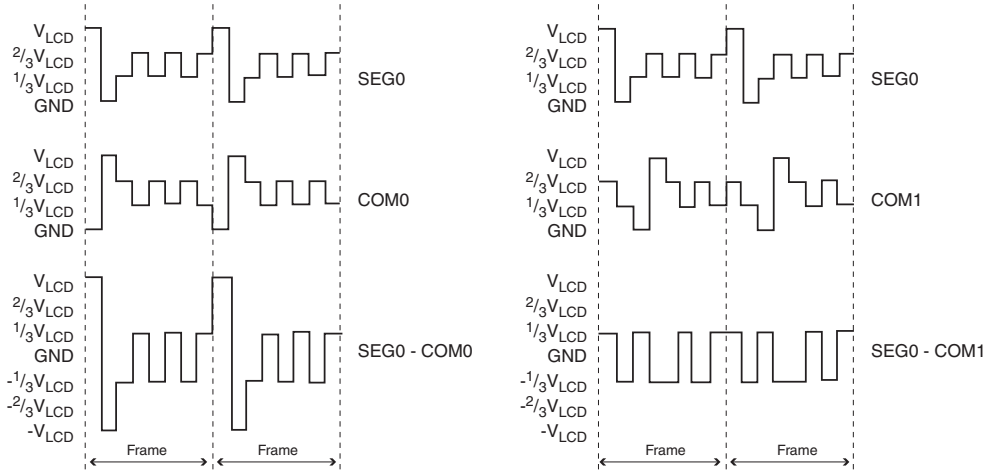
Figure 101. Driving a LCD with Three Common Terminals



1/4 Duty and 1/3 Bias

1/3 bias is optimal for LCD displays with four common terminals (1/4 duty). Waveform is shown in Figure 102. SEG0 - COM0 is the voltage across a segment that is on and SEG0 - COM1 is the voltage across a segment that is off.

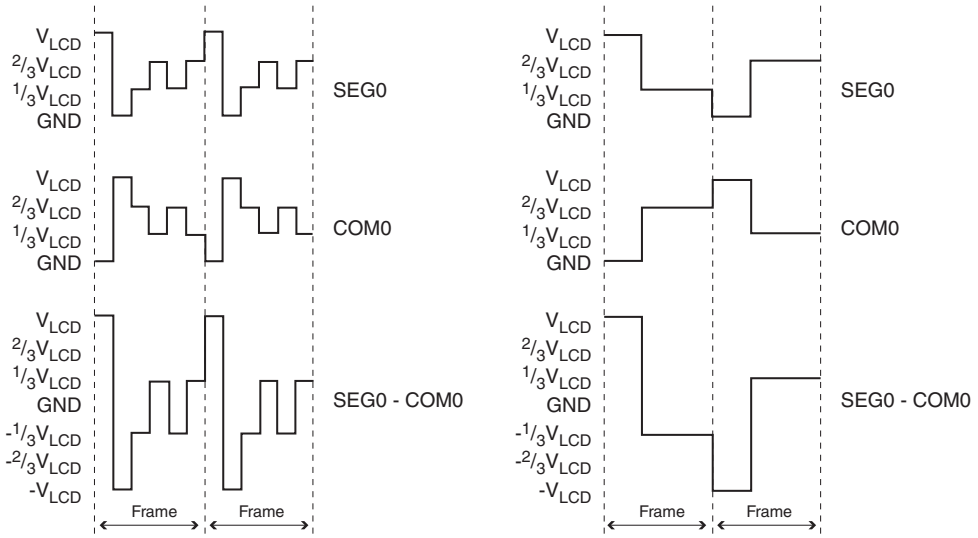
Figure 102. Driving a LCD with Four Common Terminals



Low Power Waveform

To reduce toggle activity and hence power consumption a low power waveform can be selected by writing LCDAB to one. Low power waveform requires two subsequent frames with the same display data to obtain zero DC voltage. Consequently data latching and Interrupt Flag is only set every second frame. Default and low power waveform is shown in Figure 103 for 1/3 duty and 1/3 bias. For other selections of duty and bias, the effect is similar.

Figure 103. Default and Low Power Waveform



Operation in Sleep Mode

When synchronous LCD clock is selected (LCDCS = 0) the LCD display will operate in Idle mode and Power-save mode with any clock source.

An asynchronous clock from TOSC1 can be selected as LCD clock by writing the LCDCS bit to one when Calibrated Internal RC Oscillator is selected as system clock source. The LCD will then operate in Idle mode, ADC Noise Reduction mode and Power-save mode.

When EXCLK in ASSR Register is written to one, and asynchronous clock is selected, the external clock input buffer is enabled and an external clock can be input on Timer Oscillator 1 (TOSC1) pin instead of a 32 kHz crystal. See “Asynchronous operation of the Timer/Counter” on page 138 for further details.

Before entering Power-down mode, Standby mode or ADC Noise Reduction mode with synchronous LCD clock selected, the user have to disable the LCD. Refer to “Disabling the LCD” on page 218.

Display Blanking

When LCDBL is written to one, the LCD is blanked after completing the current frame. All segments and common pins are connected to GND, discharging the LCD. Display memory is preserved. Display blanking should be used before disabling the LCD to avoid DC voltage across segments, and a slowly fading image.

Port Mask

For LCD with less than 25 segment terminals, it is possible to mask some of the unused pins and use them as ordinary port pins instead. Refer to Table 95 for details. Unused common pins are automatically configured as port pins.

LCD Usage

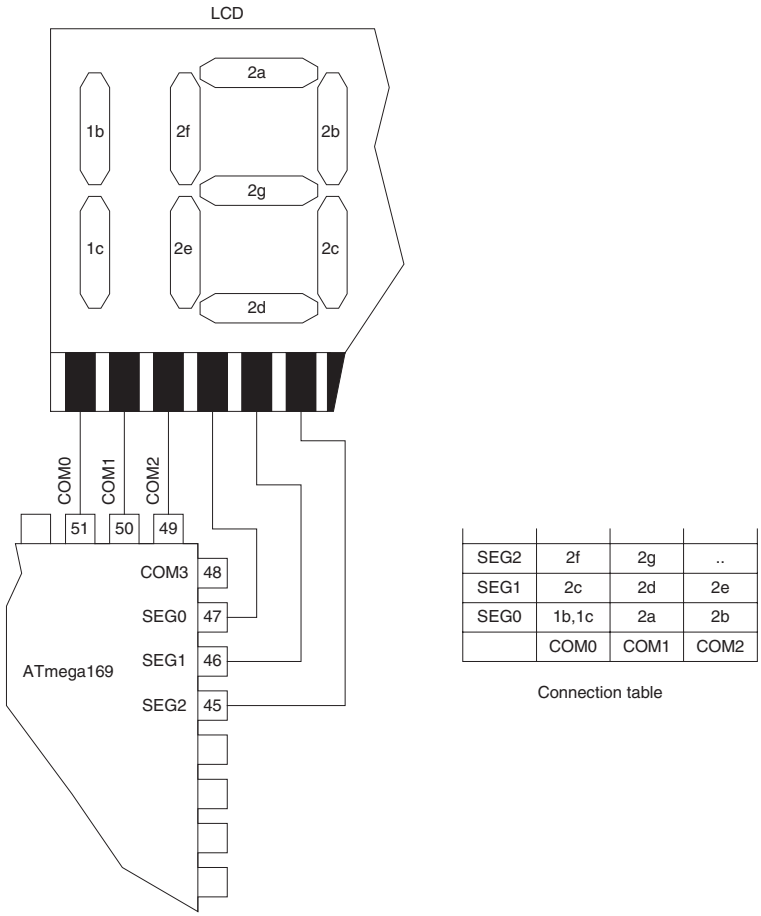
The following section describes how to use the LCD.

LCD Initialization

Prior to enabling the LCD some initialization must be preformed. The initialization process normally consists of setting the frame rate, duty, bias and port mask. LCD contrast is set initially, but can also be adjusted during operation.

Consider the following LCD as an example:

Figure 104. LCD usage example.



SEG2	2f	2g	..
SEG1	2c	2d	2e
SEG0	1b,1c	2a	2b
	COM0	COM1	COM2

Connection table

Display:	TN Positive, Reflective
Number of common terminals:	3
Number of segment terminals:	21
Bias system:	1/3 Bias
Drive system:	1/3 Duty
Operating voltage:	3.0 ± 0.3 V

Assembly Code Example[1]

```
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  LCDCRA, r16
  ret
```

C Code Example[1]

```
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 */
  LCDCRA = (1<<LCDEN);
}
```

Note: 1. See “About Code Examples” on page 6.

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

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.

In the example below we assume SEG10 and COM1 and SEG4 in COM0 are the only segments changed from frame to frame. Data are stored in r20 and r21 for simplicity

Assembly Code Example[1]

```
LCD_update:
  ; LCD Blanking and Low power waveform are unchanged.
  ; Update Display memory.
  sts  LCDDR0, r20
  sts  LCDDR6, r21
  ret
```

C Code Example[1]

```
Void LCD_update(unsigned char data1, data2);
{
  /* LCD Blanking and Low power waveform are unchanged. */
  /* Update Display memory. */
  LCDDR0 = data1;
  LCDDR6 = data2;
}
```

Note: 1. See “About Code Examples” on page 6.

Disabling the LCD

In some application it may be necessary to disable the LCD. This is the case if the MCU enters Power-down mode where no clock source is present.

The LCD should be completely discharged before being disabled. No DC voltage should be left across any segment. The best way to achieve this is to use the LCD Blanking feature that drives all segment pins and common pins to GND.

When the LCD is disabled, port function is activated again. Therefore, the user must check that port pins connected to a LCD terminal are either tri-state or output low (sink).

Assembly Code Example[1]

```
LCD_disable:
  ; Wait until a new frame is started.
Wait_1:
  lds  r16, LCDCRA
  sbrs r16, LCDIF
  rjmp Wait_1
  ; Set LCD Blanking and clear interrupt flag
  ; by writing a logical one to the flag.
  ldi  r16, (1<<LCDEN)|(1<<LCDIF)|(1<<LCDBL)
  sts  LCDCRA, r16
  ; Wait until LCD Blanking is effective.
Wait_2:
  lds  r16, LCDCRA
  sbrs r16, LCDIF
  rjmp Wait_2
  ; Disable LCD.
  ldi  r16, (0<<LCDEN)
  sts  LCDCRA, r16
  ret
```

C Code Example[1]

```
Void LCD_disable(void);
{
  /* Wait until a new frame is started. */
while ( !(LCDCRA & (1<<LCDIF)) )
  ;
  /* Set LCD Blanking and clear interrupt flag */
  /* by writing a logical one to the flag. */
  LCDCRA = (1<<LCDEN)|(1<<LCDIF)|(1<<LCDBL);
  /* Wait until LCD Blanking is effective. */
while ( !(LCDCRA & (1<<LCDIF)) )
  ;
  /* Disable LCD */
  LCDCRA = (0<<LCDEN);
}
```

Note: 1. See “About Code Examples” on page 6.

LCD Control and Status Register A – LCDCRA

Bit	7	6	5	4	3	2	1	0	
	LCDEN	LCDAB	–	LCDIF	LCDIE	–	–	LCDBL	LCDCRA
Read/Write	R/W	R/W	R	R/W	R/W	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – LCDEN: LCD Enable**

Writing this bit to one enables the LCD Controller/Driver. By writing it to zero, the LCD is turned off immediately. Turning the LCD Controller/Driver off while driving a display,

enables ordinary port function, and DC voltage can be applied to the display if ports are configured as output. It is recommended to drive output to ground if the LCD Controller/Driver is disabled to discharge the display.

- **Bit 6 – LCDAB: LCD Low Power Waveform**

When LCDAB is written logic zero, the default waveform is output on the LCD pins. When LCDAB is written logic one, the Low Power Waveform is output on the LCD pins. If this bit is modified during display operation the change takes place at the beginning of a new frame.

- **Bit 5 – Res: Reserved Bit**

This bit is reserved bit in the ATmega169 and will always read as zero.

- **Bit 4 – LCDIF: LCD Interrupt Flag**

This bit is set by hardware at the beginning of a new frame, at the same time as the display data is updated. The LCD Start of Frame Interrupt is executed if the LCDIE bit and the I-bit in SREG are set. LCDIF is cleared by hardware when executing the corresponding Interrupt Handling Vector. Alternatively, writing a logical one to the flag clears LCDIF. Beware that if doing a Read-Modify-Write on LCDCRA, a pending interrupt can be disabled. If Low Power Waveform is selected the Interrupt Flag is set every second frame.

- **Bit 3 – LCDIE: LCD Interrupt Enable**

When this bit is written to one and the I-bit in SREG is set, the LCD Start of Frame Interrupt is enabled.

- **Bits 2:1 – Res: Reserved Bits**

These bits are reserved bits in the ATmega169 and will always read as zero.

- **Bit 0 – LCDBL: LCD Blanking**

When this bit is written to one, the display will be blanked after completion of a frame. All segment and common pins will be driven to ground.

LCD Control and Status Register B – LCDCRB

Bit	7	6	5	4	3	2	1	0	
	LCDCS	LCD2B	LCDMUX1	LCDMUX0	–	LCDPM2	LCDPM1	LCDPM0	LCDCRB
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – LCDCS: LCD Clock Select**

When this bit is written to zero, the system clock is used. When this bit is written to one, the external asynchronous clock source is used. The asynchronous clock source is either Timer/Counter Oscillator or external clock, depending on EXCLK in ASSR. See “Asynchronous operation of the Timer/Counter” on page 138 for further details.

- **Bit 6 – LCD2B: LCD 1/2 Bias Select**

When this bit is written to zero, 1/3 bias is used. When this bit is written to one, ½ bias is used. Refer to the LCD Manufacture for recommended bias selection.

- **Bit 5:4 – LCDMUX1:0: LCD Mux Select**

The LCDMUX1:0 bits determine the duty cycle. Common pins that are not used are ordinary port pins. The different duty selections are shown in Table 94.

Table 94. LCD Duty Select

LCDMUX1	LCDMUX0	Duty	Bias	COM Pin	I/O Port Pin
0	0	Static	Static	COM0	COM1:3
0	1	1/2	1/2 or 1/3(1)	COM0:1	COM2:3
1	0	1/3	1/2 or 1/3(1)	COM0:2	COM3
1	1	1/4	1/2 or 1/3(1)	COM0:3	None

Note: 1. 1/2 bias when LCD2B is written to one and 1/3 otherwise.

- **Bit3 – Res: Reserved Bit**

This bit is reserved bit in the ATmega169 and will always read as zero.

- **Bits 2:0 – LCDPM2:0: LCD Port Mask**

The LCDPM2:0 bits determine the number of port pins to be used as segment drivers. The different selections are shown in Table 95. Unused pins can be used as ordinary port pins.

Table 95. LCD Port Mask

LCDPM2	LCDPM1	LCDPM0	I/O Port in Use as Segment Driver	Maximum Number of Segments
0	0	0	SEG0:12	13
0	0	1	SEG0:14	15
0	1	0	SEG0:16	17
0	1	1	SEG0:18	19
1	0	0	SEG0:20	21
1	0	1	SEG0:22	23
1	1	0	SEG0:23	24
1	1	1	SEG0:24	25

LCD Frame Rate Register – LCDFRR

Bit	7	6	5	4	3	2	1	0	
	–	LCDPS2	LCDPS1	LCDPS0	–	LCDCD2	LCDCD1	LCDCD0	LCDFRR
Read/Write	R	R/W	R/W	R/W	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – Res: Reserved Bit**

This bit is reserved bit in the ATmega169 and will always read as zero.

- **Bits 6:4 – LCDPS2:0: LCD Prescaler Select**

The LCDPS2:0 bits selects tap point from a prescaler. The prescaled output can be further divided by setting the clock divide bits (LCDCD2:0). The different selections are shown in Table 96. Together they determine the prescaled LCD clock (clk_{LCD_PS}), which is clocking the LCD module.

Table 96. LCD Prescaler Select

LCDPS2	LCDPS1	LCDPS0	Output from Prescaler clk_{LCD}/N	Applied Prescaled LCD Clock Frequency when LCDCD2:0 = 0, Duty = 1/4, and Frame Rate = 64 Hz
0	0	0	$clk_{LCD}/16$	8.1 kHz
0	0	1	$clk_{LCD}/64$	33 kHz
0	1	0	$clk_{LCD}/128$	66 kHz
0	1	1	$clk_{LCD}/256$	130 kHz
1	0	0	$clk_{LCD}/512$	260 kHz
1	0	1	$clk_{LCD}/1024$	520 kHz
1	1	0	$clk_{LCD}/2048$	1 MHz
1	1	1	$clk_{LCD}/4096$	2 MHz

- **Bit 3 – Res: Reserved Bit**

This bit is reserved bit in the ATmega169 and will always read as zero.

- **Bits 2:0 – LCDCD2:0: LCD Clock Divide 2, 1, and 0**

The LCDCD2:0 bits determine division ratio in the clock divider. The various selections are shown in Table 97. This Clock Divider gives extra flexibility in frame rate selection.

Table 97. LCD Clock Divide

LCDCD2	LCDCD1	LCDCD0	Output from Prescaler divided by (D):	clk_{LCD} = 32.768 kHz, N = 16, and Duty = 1/4, gives a frame rate of:
0	0	0	1	256 Hz
0	0	1	2	128 Hz
0	1	0	3	85.3 Hz
0	1	1	4	64 Hz
1	0	0	5	51.2 Hz
1	0	1	6	42.7 Hz
1	1	0	7	36.6 Hz
1	1	1	8	32 Hz

The frame frequency can be calculated by the following equation:

$$f_{frame} = \frac{f_{clk_{LCD}}}{(K \cdot N \cdot D)}$$

Where:

N = prescaler divider (16, 64, 128, 256, 512, 1024, 2048, or 4096).

K = 8 for duty = 1/4, 1/2, and static.

K = 6 for duty = 1/3.

D = Division factor (see Table 97).

This is a very flexible scheme, and users are encouraged to calculate their own table to investigate the possible frame rates from the formula above. Note when using 1/3 duty the frame rate is increased with 33% when Frame Rate Register is constant. Example of frame rate calculation is shown in Table 98.

Table 98. Example of frame rate calculation

clk_{LCD}	duty	K	N	LCDCD2:0	D	Frame Rate
4 MHz	1/4	8	2048	011	4	4000000/(8*2048*4) = 61 Hz
4 MHz	1/3	6	2048	011	4	4000000/(6*2048*4) = 81 Hz
32.768 kHz	Static	8	16	000	1	32768/(8*16*1) = 256 Hz
32.768 kHz	1/2	8	16	100	5	32768/(8*16*5) = 51 Hz

LCD Contrast Control Register – LCDCCR

Bit	7	6	5	4	3	2	1	0	
	LCDDC2	LCDDC1	LCDDC0	–	LCDCC3	LCDCC2	LCDCC1	LCDCC0	LCDCCR
Read/Write	R/W	R/W	R/W	R	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.

Note: These bits are not available in ATmega169 revisions A to D

Table 99. LCD Display Configuration

LCDDC2	LCDDC1	LCDDC0	Nominal drive time
0	0	0	300 µs
0	0	1	70 µs
0	1	0	150 µs
0	1	1	450 µs
1	0	0	575 µs
1	0	1	850 µs
1	1	0	1150 µs
1	1	1	50% of clk_{LCD_PS}

- **Bit 4 – Res: Reserved Bit**

This bit is reserved in the ATmega169 and will always read as zero.

- **Bits 3:0 – LCDCC3:0: LCD Contrast Control**

The LCDCC3:0 bits determine the maximum voltage V_{LCD} on segment and common pins. The different selections are shown in Table 100. New values take effect every beginning of a new frame.

Table 100. LCD Contrast Control

LCDCC3	LCDCC2	LCDCC1	LCDCC0	Maximum Voltage V_{LCD}
0	0	0	0	2.60 V
0	0	0	1	2.65 V
0	0	1	0	2.70 V
0	0	1	1	2.75 V
0	1	0	0	2.80 V
0	1	0	1	2.85 V
0	1	1	0	2.90 V
0	1	1	1	2.95 V
1	0	0	0	3.00 V
1	0	0	1	3.05 V
1	0	1	0	3.10 V
1	0	1	1	3.15 V
1	1	0	0	3.20 V
1	1	0	1	3.25 V
1	1	1	0	3.30 V
1	1	1	1	3.35 V

LCD Memory Mapping

Write a LCD memory bit to one and the corresponding segment will be energized (visible). Unused LCD Memory bits for the actual display can be used freely as storage.

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	–	–	LCDDR19
COM3	–	–	–	–	–	–	–	SEG324	LCDDR18
COM3	SEG323	SEG322	SEG321	SEG320	SEG319	SEG318	SEG317	SEG316	LCDDR17
COM3	SEG315	SEG314	SEG313	SEG312	SEG311	SEG310	SEG309	SEG308	LCDDR16
COM3	SEG307	SEG306	SEG305	SEG304	SEG303	SEG302	SEG301	SEG300	LCDDR15
	–	–	–	–	–	–	–	–	LCDDR14
COM2	–	–	–	–	–	–	–	SEG224	LCDDR13
COM2	SEG223	SEG222	SEG221	SEG220	SEG219	SEG218	SEG217	SEG216	LCDDR12
COM2	SEG215	SEG214	SEG213	SEG212	SEG211	SEG210	SEG209	SEG208	LCDDR11
COM2	SEG207	SEG206	SEG205	SEG204	SEG203	SEG202	SEG201	SEG200	LCDDR10
	–	–	–	–	–	–	–	–	LCDDR9
COM1	–	–	–	–	–	–	–	SEG124	LCDDR8
COM1	SEG123	SEG122	SEG121	SEG120	SEG119	SEG118	SEG117	SEG116	LCDDR7
COM1	SEG115	SEG114	SEG113	SEG112	SEG111	SEG110	SEG109	SEG108	LCDDR6
COM1	SEG107	SEG106	SEG105	SEG104	SEG103	SEG102	SEG101	SEG100	LCDDR5
	–	–	–	–	–	–	–	–	LCDDR4
COM0	–	–	–	–	–	–	–	SEG024	LCDDR3
COM0	SEG023	SEG022	SEG021	SEG020	SEG019	SEG018	SEG017	SEG016	LCDDR2
COM0	SEG015	SEG014	SEG013	SEG012	SEG011	SEG010	SEG009	SEG008	LCDDR1
COM0	SEG007	SEG006	SEG005	SEG004	SEG003	SEG002	SEG001	SEG000	LCDDR0
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	

JTAG Interface and On-chip Debug System

Features

- **JTAG (IEEE std. 1149.1 Compliant) Interface**
- **Boundary-scan Capabilities According to the IEEE std. 1149.1 (JTAG) Standard**
- **Debugger Access to:**
  - **All Internal Peripheral Units**
  - **Internal and External RAM**
  - **The Internal Register File**
  - **Program Counter**
  - **EEPROM and Flash Memories**
- **Extensive On-chip Debug Support for Break Conditions, Including**
  - **AVR Break Instruction**
  - **Break on Change of Program Memory Flow**
  - **Single Step Break**
  - **Program Memory Break Points on Single Address or Address Range**
  - **Data Memory Break Points on Single Address or Address Range**
- **Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface**
- **On-chip Debugging Supported by AVR Studio®**

Overview

The AVR IEEE std. 1149.1 compliant JTAG interface can be used for

- Testing PCBs by using the JTAG Boundary-scan capability
- Programming the non-volatile memories, Fuses and Lock bits
- On-chip debugging

A brief description is given in the following sections. Detailed descriptions for Programming via the JTAG interface, and using the Boundary-scan Chain can be found in the sections “Programming via the JTAG Interface” on page 285 and “IEEE 1149.1 (JTAG) Boundary-scan” on page 232**,** respectively. The On-chip Debug support is considered being private JTAG instructions, and distributed within ATMEL and to selected third party vendors only.

Figure 105 shows a block diagram of the JTAG interface and the On-chip Debug system. The TAP Controller is a state machine controlled by the TCK and TMS signals. The TAP Controller selects either the JTAG Instruction Register or one of several Data Registers as the scan chain (Shift Register) between the TDI – input and TDO – output. The Instruction Register holds JTAG instructions controlling the behavior of a Data Register.

The ID-Register, Bypass Register, and the Boundary-scan Chain are the Data Registers used for board-level testing. The JTAG Programming Interface (actually consisting of several physical and virtual Data Registers) is used for serial programming via the JTAG interface. The Internal Scan Chain and Break Point Scan Chain are used for On-chip debugging only.

Test Access Port – TAP

The JTAG interface is accessed through four of the AVR’s pins. In JTAG terminology, these pins constitute the Test Access Port – TAP. These pins are:

- TMS: Test mode select. This pin is used for navigating through the TAP-controller state machine.
- TCK: Test Clock. JTAG operation is synchronous to TCK.
- TDI: Test Data In. Serial input data to be shifted in to the Instruction Register or Data Register (Scan Chains).
- TDO: Test Data Out. Serial output data from Instruction Register or Data Register.

The IEEE std. 1149.1 also specifies an optional TAP signal; TRST – Test ReSeT – which is not provided.

When the JTAGEN fuse is unprogrammed, these four TAP pins are normal port pins and the TAP controller is in reset. When programmed and the JTD bit in MCUCSR is cleared, the TAP pins are internally pulled high and the JTAG is enabled for Boundary-scan and programming. The device is shipped with this fuse programmed.

For the On-chip Debug system, in addition to the JTAG interface pins, the $\overline{\text{RESET}}$ pin is monitored by the debugger to be able to detect external reset sources. The debugger can also pull the $\overline{\text{RESET}}$ pin low to reset the whole system, assuming only open collectors on the reset line are used in the application.

Figure 105. Block Diagram

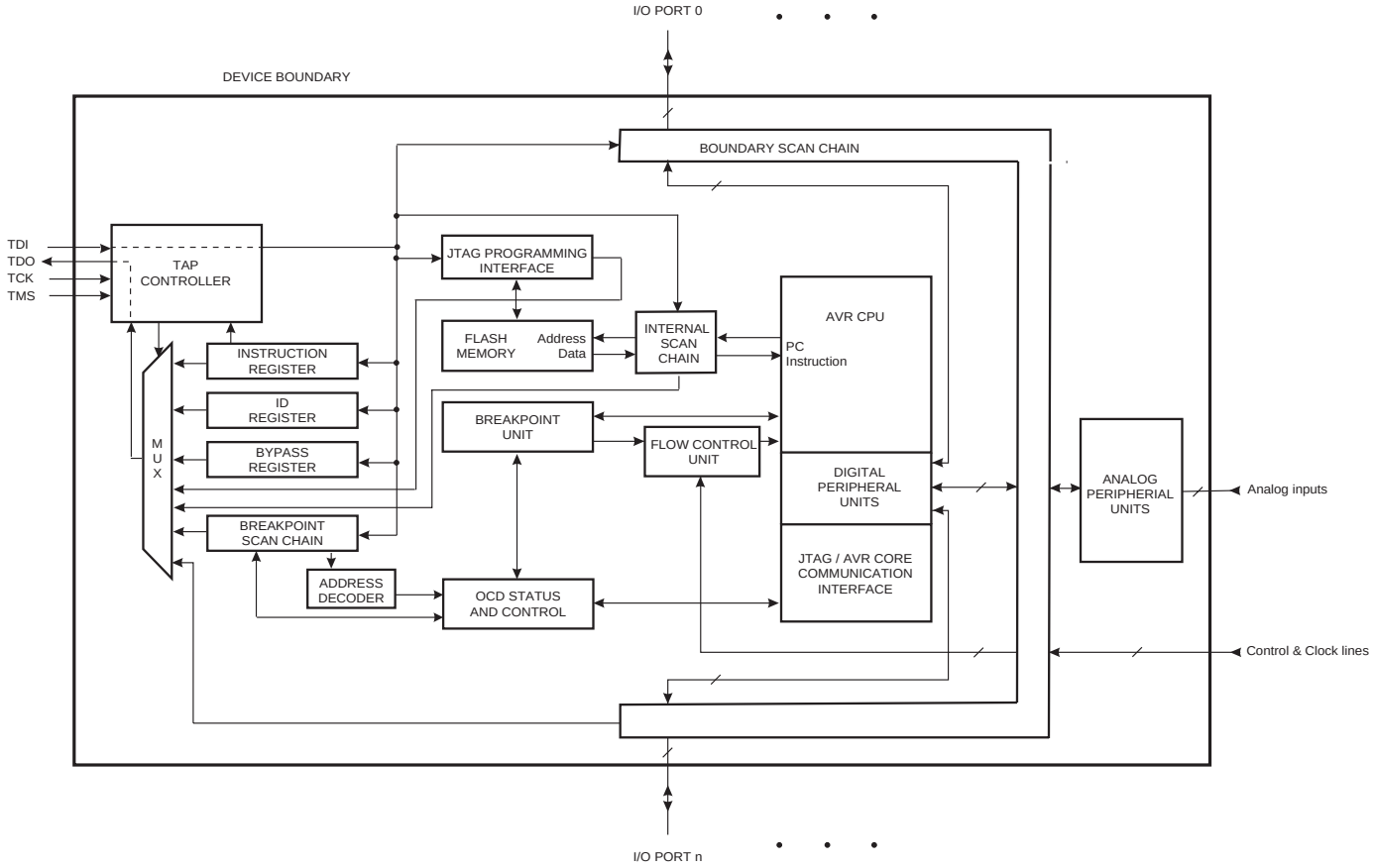
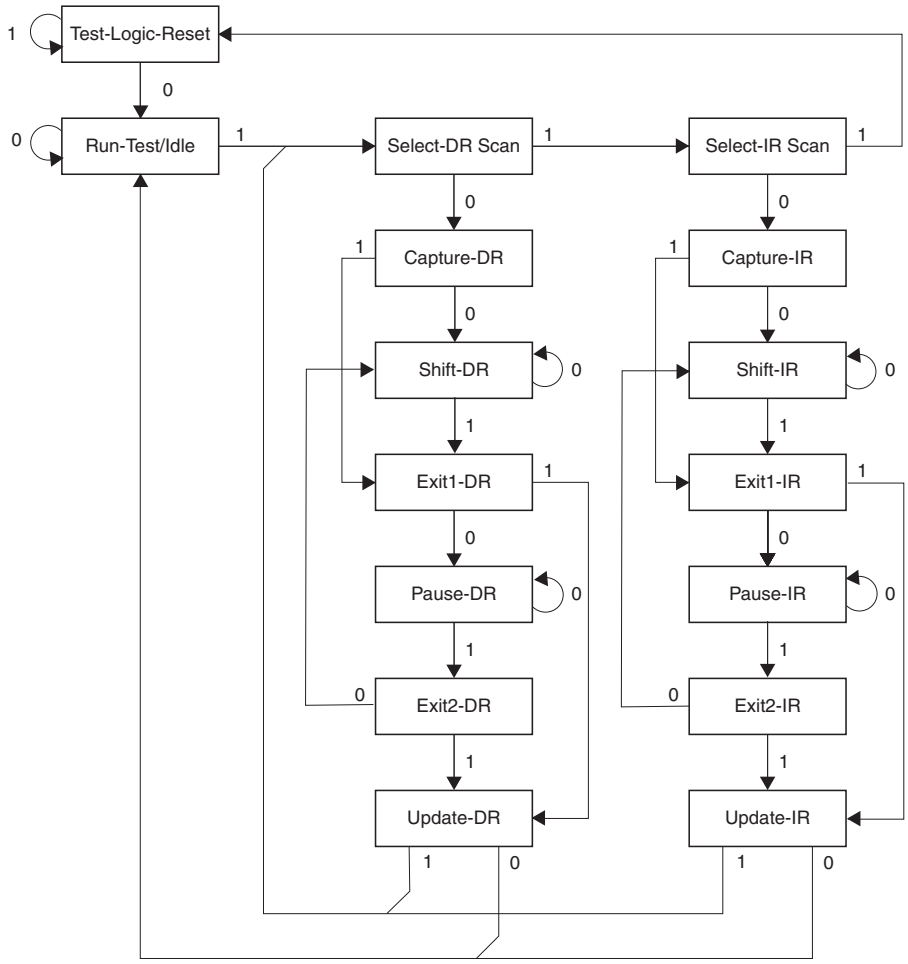


Figure 106. TAP Controller State Diagram



TAP Controller

The TAP controller is a 16-state finite state machine that controls the operation of the Boundary-scan circuitry, JTAG programming circuitry, or On-chip Debug system. The state transitions depicted in Figure 106 depend on the signal present on TMS (shown adjacent to each state transition) at the time of the rising edge at TCK. The initial state after a Power-on Reset is Test-Logic-Reset.

As a definition in this document, the LSB is shifted in and out first for all Shift Registers.

Assuming Run-Test/Idle is the present state, a typical scenario for using the JTAG interface is:

- At the TMS input, apply the sequence 1, 1, 0, 0 at the rising edges of TCK to enter the Shift Instruction Register – Shift-IR state. While in this state, shift the four bits of the JTAG instructions into the JTAG Instruction Register from the TDI input at the rising edge of TCK. The TMS input must be held low during input of the 3 LSBs in order to remain in the Shift-IR state. The MSB of the instruction is shifted in when this state is left by setting TMS high. While the instruction is shifted in from the TDI pin, the captured IR-state 0x01 is shifted out on the TDO pin. The JTAG Instruction selects a particular Data Register as path between TDI and TDO and controls the circuitry surrounding the selected Data Register.
- Apply the TMS sequence 1, 1, 0 to re-enter the Run-Test/Idle state. The instruction is latched onto the parallel output from the Shift Register path in the Update-IR

state. The Exit-IR, Pause-IR, and Exit2-IR states are only used for navigating the state machine.

- At the TMS input, apply the sequence 1, 0, 0 at the rising edges of TCK to enter the Shift Data Register – Shift-DR state. While in this state, upload the selected Data Register (selected by the present JTAG instruction in the JTAG Instruction Register) from the TDI input at the rising edge of TCK. In order to remain in the Shift-DR state, the TMS input must be held low during input of all bits except the MSB. The MSB of the data is shifted in when this state is left by setting TMS high. While the Data Register is shifted in from the TDI pin, the parallel inputs to the Data Register captured in the Capture-DR state is shifted out on the TDO pin.
- Apply the TMS sequence 1, 1, 0 to re-enter the Run-Test/Idle state. If the selected Data Register has a latched parallel-output, the latching takes place in the Update-DR state. The Exit-DR, Pause-DR, and Exit2-DR states are only used for navigating the state machine.

As shown in the state diagram, the Run-Test/Idle state need not be entered between selecting JTAG instruction and using Data Registers, and some JTAG instructions may select certain functions to be performed in the Run-Test/Idle, making it unsuitable as an Idle state.

Note: Independent of the initial state of the TAP Controller, the Test-Logic-Reset state can always be entered by holding TMS high for five TCK clock periods.

For detailed information on the JTAG specification, refer to the literature listed in “Bibliography” on page 231.

Using the Boundary-scan Chain

A complete description of the Boundary-scan capabilities are given in the section “IEEE 1149.1 (JTAG) Boundary-scan” on page 232.

Using the On-chip Debug System

As shown in Figure 105, the hardware support for On-chip Debugging consists mainly of

- A scan chain on the interface between the internal AVR CPU and the internal peripheral units.
- Break Point unit.
- Communication interface between the CPU and JTAG system.

All read or modify/write operations needed for implementing the Debugger are done by applying AVR instructions via the internal AVR CPU Scan Chain. The CPU sends the result to an I/O memory mapped location which is part of the communication interface between the CPU and the JTAG system.

The Break Point Unit implements Break on Change of Program Flow, Single Step Break, two Program Memory Break Points, and two combined Break Points. Together, the four Break Points can be configured as either:

- 4 single Program Memory Break Points.
- 3 Single Program Memory Break Point + 1 single Data Memory Break Point.
- 2 single Program Memory Break Points + 2 single Data Memory Break Points.
- 2 single Program Memory Break Points + 1 Program Memory Break Point with mask (“range Break Point”).
- 2 single Program Memory Break Points + 1 Data Memory Break Point with mask (“range Break Point”).

A debugger, like the AVR Studio, may however use one or more of these resources for its internal purpose, leaving less flexibility to the end-user.

A list of the On-chip Debug specific JTAG instructions is given in “On-chip Debug Specific JTAG Instructions” on page 230.

The JTAGEN Fuse must be programmed to enable the JTAG Test Access Port. In addition, the OCDEN Fuse must be programmed and no Lock bits must be set for the On-chip debug system to work. As a security feature, the On-chip debug system is disabled when either of the LB1 or LB2 Lock bits are set. Otherwise, the On-chip debug system would have provided a back-door into a secured device.

The AVR Studio enables the user to fully control execution of programs on an AVR device with On-chip Debug capability, AVR In-Circuit Emulator, or the built-in AVR Instruction Set Simulator. AVR Studio® supports source level execution of Assembly programs assembled with Atmel Corporation’s AVR Assembler and C programs compiled with third party vendors’ compilers.

AVR Studio runs under Microsoft® Windows® 95/98/2000, Windows NT® and Windows XP®.

For a full description of the AVR Studio, please refer to the AVR Studio User Guide. Only highlights are presented in this document.

All necessary execution commands are available in AVR Studio, both on source level and on disassembly level. The user can execute the program, single step through the code either by tracing into or stepping over functions, step out of functions, place the cursor on a statement and execute until the statement is reached, stop the execution, and reset the execution target. In addition, the user can have an unlimited number of code Break Points (using the BREAK instruction) and up to two data memory Break Points, alternatively combined as a mask (range) Break Point.

On-chip Debug Specific JTAG Instructions

The On-chip debug support is considered being private JTAG instructions, and distributed within ATMEL and to selected third party vendors only. Instruction opcodes are listed for reference.

PRIVATE0; 0x8 Private JTAG instruction for accessing On-chip debug system.

PRIVATE1; 0x9 Private JTAG instruction for accessing On-chip debug system.

PRIVATE2; 0xA Private JTAG instruction for accessing On-chip debug system.

PRIVATE3; 0xB Private JTAG instruction for accessing On-chip debug system.

On-chip Debug Related Register in I/O Memory

On-chip Debug Register – OCDR

Bit	7	6	5	4	3	2	1	0	
	MSB/IDRD							LSB	OCDR
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 OCDR Register provides a communication channel from the running program in the microcontroller to the debugger. The CPU can transfer a byte to the debugger by writing to this location. At the same time, an internal flag; I/O Debug Register Dirty – IDRD – is set to indicate to the debugger that the register has been written. When the CPU reads the OCDR Register the 7 LSB will be from the OCDR Register, while the MSB is the IDRD bit. The debugger clears the IDRD bit when it has read the information.

In some AVR devices, this register is shared with a standard I/O location. In this case, the OCDR Register can only be accessed if the OCDEN Fuse is programmed, and the debugger enables access to the OCDR Register. In all other cases, the standard I/O location is accessed.

Refer to the debugger documentation for further information on how to use this register.

Using the JTAG Programming Capabilities

Programming of AVR parts via JTAG is performed via the 4-pin JTAG port, TCK, TMS, TDI, and TDO. These are the only pins that need to be controlled/observed to perform JTAG programming (in addition to power pins). It is not required to apply 12V externally. The JTAGEN Fuse must be programmed and the JTD bit in the MCUCR Register must be cleared to enable the JTAG Test Access Port.

The JTAG programming capability supports:

- Flash programming and verifying.
- EEPROM programming and verifying.
- Fuse programming and verifying.
- Lock bit programming and verifying.

The Lock bit security is exactly as in parallel programming mode. If the Lock bits LB1 or LB2 are programmed, the OCDEN Fuse cannot be programmed unless first doing a chip erase. This is a security feature that ensures no back-door exists for reading out the content of a secured device.

The details on programming through the JTAG interface and programming specific JTAG instructions are given in the section “Programming via the JTAG Interface” on page 285.

Bibliography

For more information about general Boundary-scan, the following literature can be consulted:

- IEEE: IEEE Std. 1149.1-1990. IEEE Standard Test Access Port and Boundary-scan Architecture, IEEE, 1993.
- Colin Maunder: The Board Designers Guide to Testable Logic Circuits, Addison-Wesley, 1992.

IEEE 1149.1 (JTAG) Boundary-scan

Features

- **JTAG (IEEE std. 1149.1 compliant) Interface**
- **Boundary-scan Capabilities According to the JTAG Standard**
- **Full Scan of all Port Functions as well as Analog Circuitry having Off-chip Connections**
- **Supports the Optional IDCODE Instruction**
- **Additional Public AVR_RESET Instruction to Reset the AVR**

System Overview

The Boundary-scan chain has the capability of driving and observing the logic levels on the digital I/O pins, as well as the boundary between digital and analog logic for analog circuitry having off-chip connections. At system level, all ICs having JTAG capabilities are connected serially by the TDI/TDO signals to form a long Shift Register. An external controller sets up the devices to drive values at their output pins, and observe the input values received from other devices. The controller compares the received data with the expected result. In this way, Boundary-scan provides a mechanism for testing interconnections and integrity of components on Printed Circuits Boards by using the four TAP signals only.

The four IEEE 1149.1 defined mandatory JTAG instructions IDCODE, BYPASS, SAMPLE/PRELOAD, and EXTEST, as well as the AVR specific public JTAG instruction AVR_RESET can be used for testing the Printed Circuit Board. Initial scanning of the Data Register path will show the ID-Code of the device, since IDCODE is the default JTAG instruction. It may be desirable to have the AVR device in reset during test mode. If not reset, inputs to the device may be determined by the scan operations, and the internal software may be in an undetermined state when exiting the test mode. Entering reset, the outputs of any port pin will instantly enter the high impedance state, making the HIGHZ instruction redundant. If needed, the BYPASS instruction can be issued to make the shortest possible scan chain through the device. The device can be set in the reset state either by pulling the external $\overline{\text{RESET}}$ pin low, or issuing the AVR_RESET instruction with appropriate setting of the Reset Data Register.

The EXTEST instruction is used for sampling external pins and loading output pins with data. The data from the output latch will be driven out on the pins as soon as the EXTEST instruction is loaded into the JTAG IR-Register. Therefore, the SAMPLE/PRELOAD should also be used for setting initial values to the scan ring, to avoid damaging the board when issuing the EXTEST instruction for the first time. SAMPLE/PRELOAD can also be used for taking a snapshot of the external pins during normal operation of the part.

The JTAGEN Fuse must be programmed and the JTD bit in the I/O Register MCUCR must be cleared to enable the JTAG Test Access Port.

When using the JTAG interface for Boundary-scan, using a JTAG TCK clock frequency higher than the internal chip frequency is possible. The chip clock is not required to run.

Data Registers

The Data Registers relevant for Boundary-scan operations are:

- Bypass Register
- Device Identification Register
- Reset Register
- Boundary-scan Chain

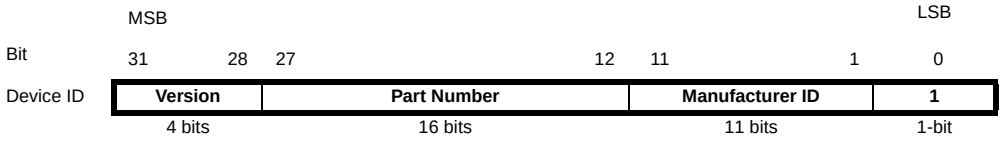
Bypass Register

The Bypass Register consists of a single Shift Register stage. When the Bypass Register is selected as path between TDI and TDO, the register is reset to 0 when leaving the Capture-DR controller state. The Bypass Register can be used to shorten the scan chain on a system when the other devices are to be tested.

Device Identification Register

Figure 107 shows the structure of the Device Identification Register.

Figure 107. The Format of the Device Identification Register



Version

Version is a 4-bit number identifying the revision of the component. The JTAG version number follows the revision of the device. Revision A is 0x0, revision B is 0x1 and so on.

Part Number

The part number is a 16-bit code identifying the component. The JTAG Part Number for ATmega169 is listed in Table 101.

Table 101. AVR JTAG Part Number

Part Number	JTAG Part Number (Hex)
ATmega169	0x9405

Manufacturer ID

The Manufacturer ID is a 11-bit code identifying the manufacturer. The JTAG manufacturer ID for ATMEL is listed in Table 102.

Table 102. Manufacturer ID

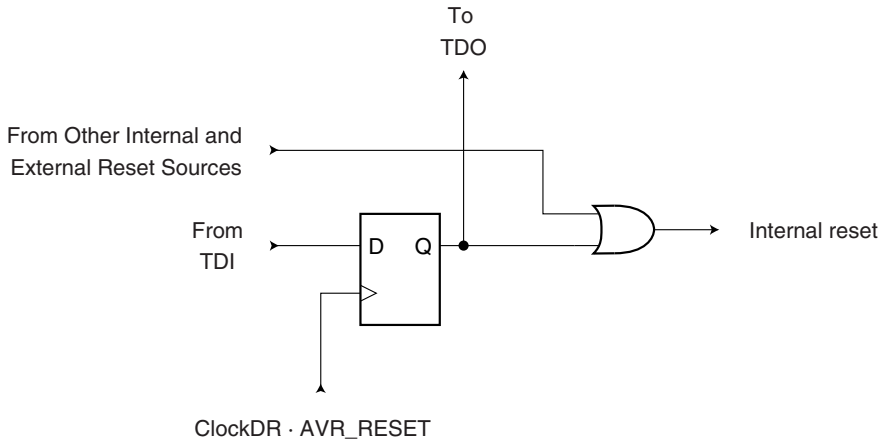
Manufacturer	JTAG Manufactor ID (Hex)
ATMEL	0x01F

Reset Register

The Reset Register is a test Data Register used to reset the part. Since the AVR tri-states Port Pins when reset, the Reset Register can also replace the function of the unimplemented optional JTAG instruction HIGHZ.

A high value in the Reset Register corresponds to pulling the external Reset low. The part is reset as long as there is a high value present in the Reset Register. Depending on the fuse settings for the clock options, the part will remain reset for a reset time-out period (refer to “Clock Sources” on page 24) after releasing the Reset Register. The output from this Data Register is not latched, so the reset will take place immediately, as shown in Figure 108.

Figure 108. Reset Register



Boundary-scan Chain

The Boundary-scan Chain has the capability of driving and observing the logic levels on the digital I/O pins, as well as the boundary between digital and analog logic for analog circuitry having off-chip connections.

See “Boundary-scan Chain” on page 236 for a complete description.

Boundary-scan Specific JTAG Instructions

The Instruction Register is 4-bit wide, supporting up to 16 instructions. Listed below are the JTAG instructions useful for Boundary-scan operation. Note that the optional HIGHZ instruction is not implemented, but all outputs with tri-state capability can be set in high-impedant state by using the AVR_RESET instruction, since the initial state for all port pins is tri-state.

As a definition in this datasheet, the LSB is shifted in and out first for all Shift Registers.

The OPCODE for each instruction is shown behind the instruction name in hex format. The text describes which Data Register is selected as path between TDI and TDO for each instruction.

EXTEST; 0x0

Mandatory JTAG instruction for selecting the Boundary-scan Chain as Data Register for testing circuitry external to the AVR package. For port-pins, Pull-up Disable, Output Control, Output Data, and Input Data are all accessible in the scan chain. For Analog circuits having off-chip connections, the interface between the analog and the digital logic is in the scan chain. The contents of the latched outputs of the Boundary-scan chain is driven out as soon as the JTAG IR-Register is loaded with the EXTEST instruction.

The active states are:

- Capture-DR: Data on the external pins are sampled into the Boundary-scan Chain.
- Shift-DR: The Internal Scan Chain is shifted by the TCK input.
- Update-DR: Data from the scan chain is applied to output pins.

IDCODE; 0x1

Optional JTAG instruction selecting the 32 bit ID-Register as Data Register. The ID-Register consists of a version number, a device number and the manufacturer code chosen by JEDEC. This is the default instruction after power-up.

The active states are:

- Capture-DR: Data in the IDCODE Register is sampled into the Boundary-scan Chain.
- Shift-DR: The IDCODE scan chain is shifted by the TCK input.

SAMPLE_PRELOAD; 0x2

Mandatory JTAG instruction for pre-loading the output latches and taking a snap-shot of the input/output pins without affecting the system operation. However, the output latches are not connected to the pins. The Boundary-scan Chain is selected as Data Register.

The active states are:

- Capture-DR: Data on the external pins are sampled into the Boundary-scan Chain.
- Shift-DR: The Boundary-scan Chain is shifted by the TCK input.
- Update-DR: Data from the Boundary-scan chain is applied to the output latches. However, the output latches are not connected to the pins.

AVR_RESET; 0xC

The AVR specific public JTAG instruction for forcing the AVR device into the Reset mode or releasing the JTAG reset source. The TAP controller is not reset by this instruction. The one bit Reset Register is selected as Data Register. Note that the reset will be active as long as there is a logic “one” in the Reset Chain. The output from this chain is not latched.

The active states are:

- Shift-DR: The Reset Register is shifted by the TCK input.

BYPASS; 0xF

Mandatory JTAG instruction selecting the Bypass Register for Data Register.

The active states are:

- Capture-DR: Loads a logic “0” into the Bypass Register.
- Shift-DR: The Bypass Register cell between TDI and TDO is shifted.

Boundary-scan Related Register in I/O Memory

MCU Control Register – MCUCR

The MCU Control Register contains control bits for general MCU functions.

Bit	7	6	5	4	3	2	1	0	
	JTD	–	–	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R/W	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – JTD: JTAG Interface Disable**

When this bit is zero, the JTAG interface is enabled if the JTAGEN Fuse is programmed. If this bit is one, the JTAG interface is disabled. In order to avoid unintentional disabling or enabling of the JTAG interface, a timed sequence must be followed when changing this bit: The application software must write this bit to the desired value twice within four cycles to change its value. Note that this bit must not be altered when using the On-chip Debug system.

If the JTAG interface is left unconnected to other JTAG circuitry, the JTD bit should be set to one. The reason for this is to avoid static current at the TDO pin in the JTAG interface.

MCU Status Register – MCUSR

The MCU Status Register provides information on which reset source caused an MCU reset.

Bit	7	6	5	4	3	2	1	0	
	–	–	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUSR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

- **Bit 4 – JTRF: JTAG Reset Flag**

This bit is set if a reset is being caused by a logic one in the JTAG Reset Register selected by the JTAG instruction AVR_RESET. This bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

Boundary-scan Chain

The Boundary-scan chain has the capability of driving and observing the logic levels on the digital I/O pins, as well as the boundary between digital and analog logic for analog circuitry having off-chip connection.

Scanning the Digital Port Pins

Figure 109 shows the Boundary-scan Cell for a bi-directional port pin with pull-up function. The cell consists of a standard Boundary-scan cell for the Pull-up Enable – PUExn – function, and a bi-directional pin cell that combines the three signals Output Control – OCxn, Output Data – ODxn, and Input Data – IDxn, into only a two-stage Shift Register. The port and pin indexes are not used in the following description

The Boundary-scan logic is not included in the figures in the datasheet. Figure 110 shows a simple digital port pin as described in the section “I/O-Ports” on page 55. The Boundary-scan details from Figure 109 replaces the dashed box in Figure 110.

When no alternate port function is present, the Input Data – ID – corresponds to the PINxn Register value (but ID has no synchronizer), Output Data corresponds to the PORT Register, Output Control corresponds to the Data Direction – DD Register, and the Pull-up Enable – PUExn – corresponds to logic expression $\overline{PUD} \cdot \overline{DDxn} \cdot PORTxn$.

Digital alternate port functions are connected outside the dotted box in Figure 110 to make the scan chain read the actual pin value. For Analog function, there is a direct connection from the external pin to the analog circuit, and a scan chain is inserted on the interface between the digital logic and the analog circuitry.

Figure 109. Boundary-scan Cell for Bi-directional Port Pin with Pull-up Function.

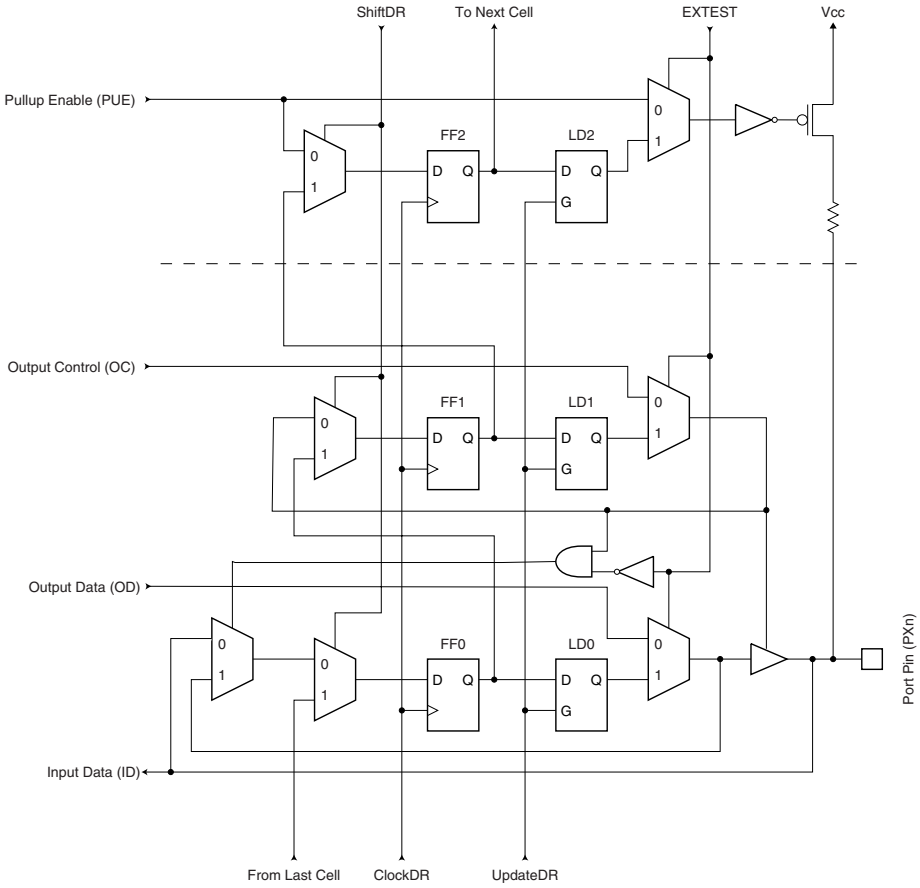
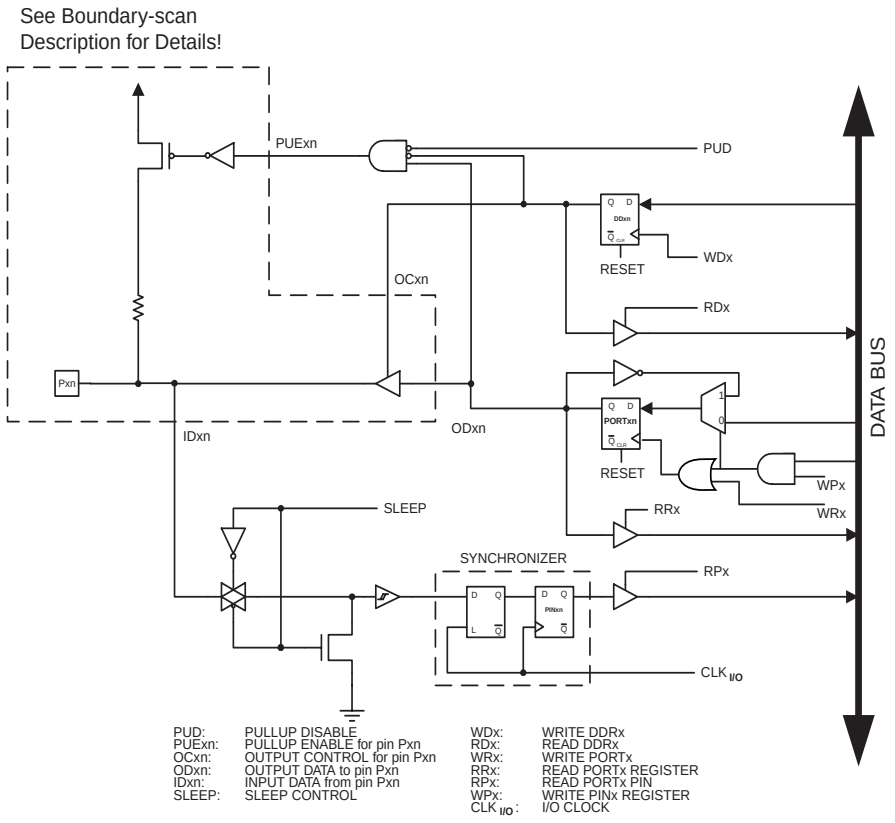


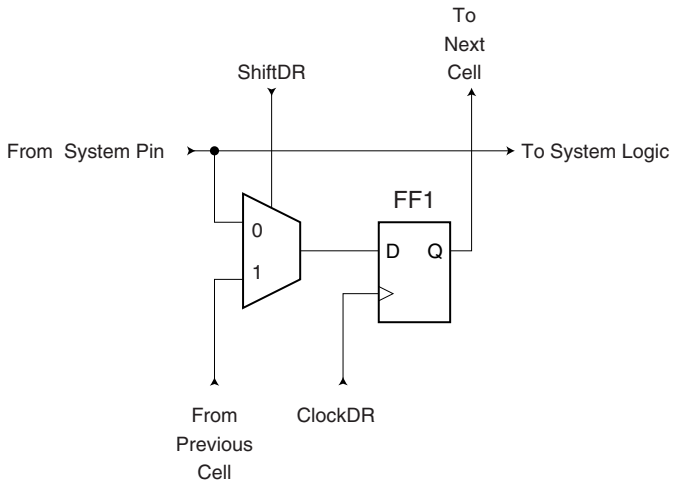
Figure 110. General Port Pin Schematic Diagram



Scanning the RESET Pin

The RESET pin accepts 5V active low logic for standard reset operation, and 12V active high logic for High Voltage Parallel programming. An observe-only cell as shown in Figure 111 is inserted both for the 5V reset signal; RSTT, and the 12V reset signal; RSTHV.

Figure 111. Observe-only Cell



Scanning the Clock Pins

The AVR devices have many clock options selectable by fuses. These are: Internal RC Oscillator, External Clock, (High Frequency) Crystal Oscillator, Low-frequency Crystal Oscillator, and Ceramic Resonator.

Figure 112 shows how each Oscillator with external connection is supported in the scan chain. The Enable signal is supported with a general Boundary-scan cell, while the Oscillator/clock output is attached to an observe-only cell. In addition to the main clock, the timer Oscillator is scanned in the same way. The output from the internal RC Oscillator is not scanned, as this Oscillator does not have external connections.

Figure 112. Boundary-scan Cells for Oscillators and Clock Options

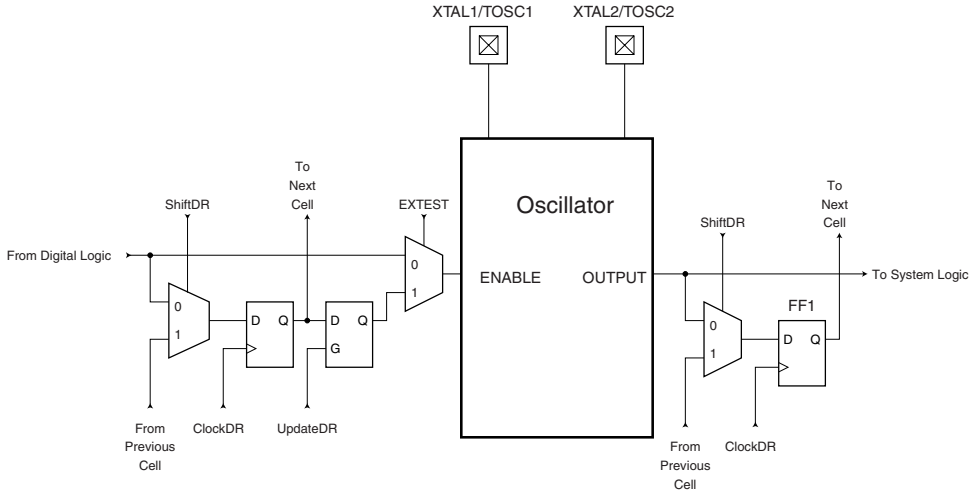


Table 103 summaries the scan registers for the external clock pin XTAL1, oscillators with XTAL1/XTAL2 connections as well as 32kHz Timer Oscillator.

Table 103. Scan Signals for the Oscillator(1)(2)(3)

Enable Signal	Scanned Clock Line	Clock Option	Scanned Clock Line when not Used
EXTCLKEN	EXTCLK (XTAL1)	External Clock	0
OSCON	OSCCK	External Crystal External Ceramic Resonator	1
OSC32EN	OSC32CK	Low Freq. External Crystal	1

Notes:
1. Do not enable more than one clock source as main clock at a time.
2. Scanning an Oscillator output gives unpredictable results as there is a frequency drift between the internal Oscillator and the JTAG TCK clock. If possible, scanning an external clock is preferred.
3. The clock configuration is programmed by fuses. As a fuse is not changed run-time, the clock configuration is considered fixed for a given application. The user is advised to scan the same clock option as to be used in the final system. The enable signals are supported in the scan chain because the system logic can disable clock options in sleep modes, thereby disconnecting the Oscillator pins from the scan path if not provided.

Scanning the Analog Comparator

The relevant Comparator signals regarding Boundary-scan are shown in Figure 113. The Boundary-scan cell from Figure 114 is attached to each of these signals. The signals are described in Table 104.

The Comparator need not be used for pure connectivity testing, since all analog inputs are shared with a digital port pin as well.

Figure 113. Analog Comparator

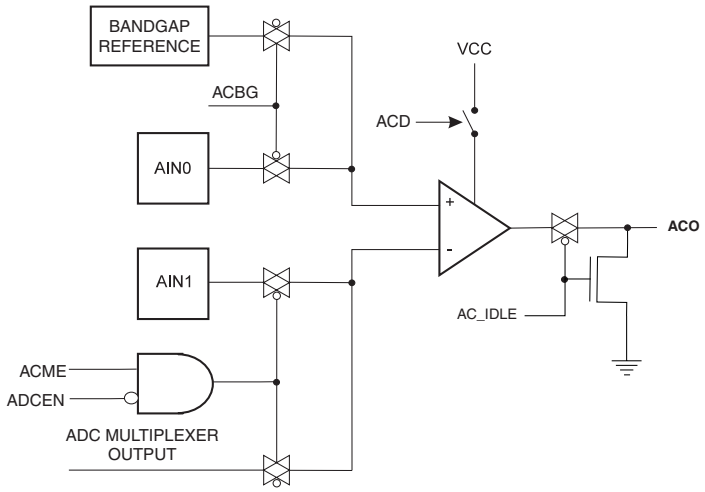


Figure 114. General Boundary-scan cell Used for Signals for Comparator and ADC

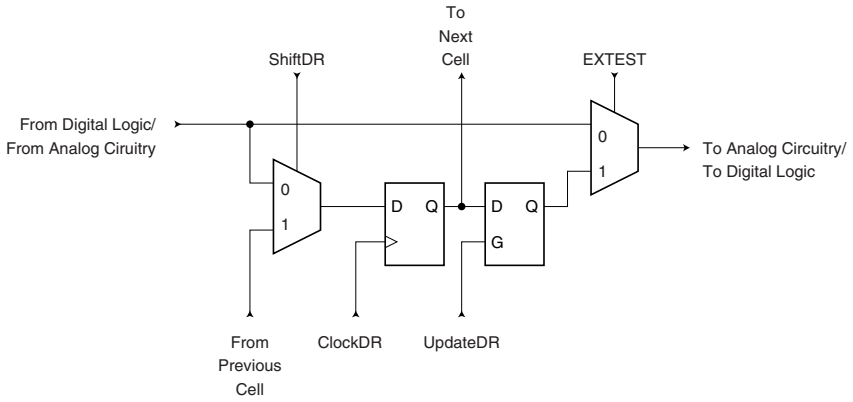


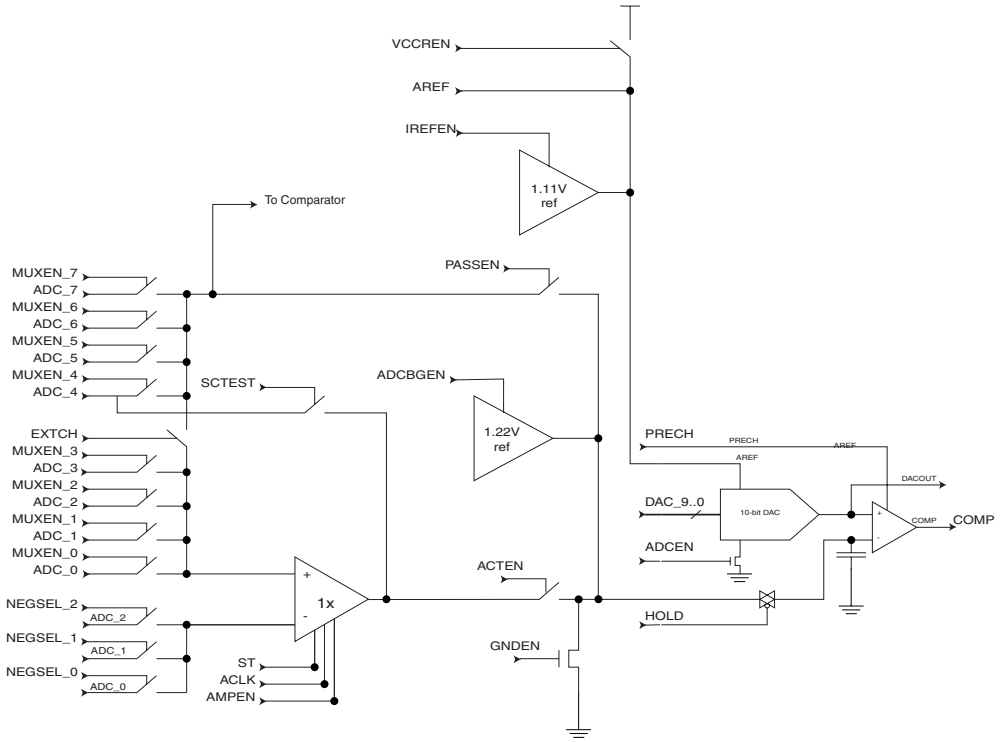
Table 104. Boundary-scan Signals for the Analog Comparator

Signal Name	Direction as Seen from the Comparator	Description	Recommended Input when Not in Use	Output Values when Recommended Inputs are Used
AC_IDLE	input	Turns off Analog Comparator when true	1	Depends upon µC code being executed
ACO	output	Analog Comparator Output	Will become input to µC code being executed	0
ACME	input	Uses output signal from ADC mux when true	0	Depends upon µC code being executed
ACBG	input	Bandgap Reference enable	0	Depends upon µC code being executed

Scanning the ADC

Figure 115 shows a block diagram of the ADC with all relevant control and observe signals. The Boundary-scan cell from Figure 111 is attached to each of these signals. The ADC need not be used for pure connectivity testing, since all analog inputs are shared with a digital port pin as well.

Figure 115. Analog to Digital Converter



The signals are described briefly in Table 105.

Table 105. Boundary-scan Signals for the ADC[1]

Signal Name	Direction as Seen from the ADC	Description	Recommended Input when not in Use	Output Values when Recommended Inputs are Used, and CPU is not Using the ADC
COMP	Output	Comparator Output	0	0
ACLK	Input	Clock signal to differential amplifier implemented as Switch-cap filters	0	0
ACTEN	Input	Enable path from differential amplifier to the comparator	0	0
ADCBGEN	Input	Enable Band-gap reference as negative input to comparator	0	0
ADCEN	Input	Power-on signal to the ADC	0	0
AMPEN	Input	Power-on signal to the differential amplifier	0	0
DAC_9	Input	Bit 9 of digital value to DAC	1	1
DAC_8	Input	Bit 8 of digital value to DAC	0	0
DAC_7	Input	Bit 7 of digital value to DAC	0	0
DAC_6	Input	Bit 6 of digital value to DAC	0	0
DAC_5	Input	Bit 5 of digital value to DAC	0	0
DAC_4	Input	Bit 4 of digital value to DAC	0	0
DAC_3	Input	Bit 3 of digital value to DAC	0	0
DAC_2	Input	Bit 2 of digital value to DAC	0	0
DAC_1	Input	Bit 1 of digital value to DAC	0	0
DAC_0	Input	Bit 0 of digital value to DAC	0	0
EXTCH	Input	Connect ADC channels 0 - 3 to by-pass path around differential amplifier	1	1
GNDEN	Input	Ground the negative input to comparator when true	0	0

Table 105. Boundary-scan Signals for the ADC[1] (Continued)

Signal Name	Direction as Seen from the ADC	Description	Recommended Input when not in Use	Output Values when Recommended Inputs are Used, and CPU is not Using the ADC
HOLD	Input	Sample & Hold signal. Sample analog signal when low. Hold signal when high. If differential amplifier is used, this signal must go active when ACLK is high.	1	1
IREFEN	Input	Enables Band-gap reference as AREF signal to DAC	0	0
MUXEN_7	Input	Input Mux bit 7	0	0
MUXEN_6	Input	Input Mux bit 6	0	0
MUXEN_5	Input	Input Mux bit 5	0	0
MUXEN_4	Input	Input Mux bit 4	0	0
MUXEN_3	Input	Input Mux bit 3	0	0
MUXEN_2	Input	Input Mux bit 2	0	0
MUXEN_1	Input	Input Mux bit 1	0	0
MUXEN_0	Input	Input Mux bit 0	1	1
NEGSEL_2	Input	Input Mux for negative input for differential signal, bit 2	0	0
NEGSEL_1	Input	Input Mux for negative input for differential signal, bit 1	0	0
NEGSEL_0	Input	Input Mux for negative input for differential signal, bit 0	0	0
PASSEN	Input	Enable pass-gate of differential amplifier.	1	1
PRECH	Input	Precharge output latch of comparator. (Active low)	1	1

Table 105. Boundary-scan Signals for the ADC[1] (Continued)

Signal Name	Direction as Seen from the ADC	Description	Recommended Input when not in Use	Output Values when Recommended Inputs are Used, and CPU is not Using the ADC
SCTEST	Input	Switch-cap TEST enable. Output from differential amplifier is sent out to Port Pin having ADC_4	0	0
ST	Input	Output of differential amplifier will settle faster if this signal is high first two ACLK periods after AMPEN goes high.	0	0
VCCREN	Input	Selects Vcc as the ACC reference voltage.	0	0

Note: 1. Incorrect setting of the switches in Figure 115 will make signal contention and may damage the part. There are several input choices to the S&H circuitry on the negative input of the output comparator in Figure 115. Make sure only one path is selected from either one ADC pin, Bandgap reference source, or Ground.

If the ADC is not to be used during scan, the recommended input values from Table 105 should be used. The user is recommended **not** to use the Differential Amplifier during scan. Switch-Cap based differential amplifier requires fast operation and accurate timing which is difficult to obtain when used in a scan chain. Details concerning operations of the differential amplifier is therefore not provided.

The AVR ADC is based on the analog circuitry shown in Figure 115 with a successive approximation algorithm implemented in the digital logic. When used in Boundary-scan, the problem is usually to ensure that an applied analog voltage is measured within some limits. This can easily be done without running a successive approximation algorithm: apply the lower limit on the digital DAC[9:0] lines, make sure the output from the comparator is low, then apply the upper limit on the digital DAC[9:0] lines, and verify the output from the comparator to be high.

The ADC need not be used for pure connectivity testing, since all analog inputs are shared with a digital port pin as well.

When using the ADC, remember the following

- The port pin for the ADC channel in use must be configured to be an input with pull-up disabled to avoid signal contention.
- In Normal mode, a dummy conversion (consisting of 10 comparisons) is performed when enabling the ADC. The user is advised to wait at least 200ns after enabling the ADC before controlling/observing any ADC signal, or perform a dummy conversion before using the first result.
- The DAC values must be stable at the midpoint value 0x200 when having the HOLD signal low (Sample mode).

As an example, consider the task of verifying a 1.5V ± 5% input signal at ADC channel 3 when the power supply is 5.0V and AREF is externally connected to V_{CC}.

The lower limit is: $\lceil 1024 \cdot 1.5V \cdot 0{,}95 / 5V \rceil = 291 = 0x123$
The upper limit is: $\lceil 1024 \cdot 1.5V \cdot 1.05 / 5V \rceil = 323 = 0x143$

The recommended values from Table 105 are used unless other values are given in the algorithm in Table 106. Only the DAC and port pin values of the Scan Chain are shown. The column “Actions” describes what JTAG instruction to be used before filling the Boundary-scan Register with the succeeding columns. The verification should be done on the data scanned out when scanning in the data on the same row in the table.

Table 106. Algorithm for Using the ADC

Step	Actions	ADCEN	DAC	MUXEN	HOLD	PRECH	PA3. Data	PA3. Control	PA3. Pull-up_ Enable
1	SAMPLE_ PRELOAD	1	0x200	0x08	1	1	0	0	0
2	EXTEST	1	0x200	0x08	0	1	0	0	0
3		1	0x200	0x08	1	1	0	0	0
4		1	0x123	0x08	1	1	0	0	0
5		1	0x123	0x08	1	0	0	0	0
6	Verify the COMP bit scanned out to be 0	1	0x200	0x08	1	1	0	0	0
7		1	0x200	0x08	0	1	0	0	0
8		1	0x200	0x08	1	1	0	0	0
9		1	0x143	0x08	1	1	0	0	0
10		1	0x143	0x08	1	0	0	0	0
11	Verify the COMP bit scanned out to be 1	1	0x200	0x08	1	1	0	0	0

Using this algorithm, the timing constraint on the HOLD signal constrains the TCK clock frequency. As the algorithm keeps HOLD high for five steps, the TCK clock frequency has to be at least five times the number of scan bits divided by the maximum hold time, $t_{hold,max}$

ATmega169 Boundary-scan Order

Table 107 shows the Scan order between TDI and TDO when the Boundary-scan chain is selected as data path. Bit 0 is the LSB; the first bit scanned in, and the first bit scanned out. The scan order follows the pin-out order as far as possible. Therefore, the bits of Port A is scanned in the opposite bit order of the other ports. Exceptions from the rules are the Scan chains for the analog circuits, which constitute the most significant bits of the scan chain regardless of which physical pin they are connected to. In Figure 109, PXn. Data corresponds to FF0, PXn. Control corresponds to FF1, and PXn. Pull-up_enable corresponds to FF2. Bit 4, 5, 6, and 7of Port F is not in the scan chain, since these pins constitute the TAP pins when the JTAG is enabled.

Table 107. ATmega169 Boundary-scan Order

Bit Number	Signal Name	Module
197	AC_IDLE	Comparator
196	ACO	
195	ACME	
194	AINBG	
193	COMP	ADC
192	ACLK	
191	ACTEN	
190	PRIVATE_SIGNAL1[1]	
189	ADCBGEN	
188	ADCEN	
187	AMPEN	
186	DAC_9	
185	DAC_8	
184	DAC_7	
183	DAC_6	
182	DAC_5	
181	DAC_4	
180	DAC_3	
179	DAC_2	
178	DAC_1	
177	DAC_0	
176	EXTCH	
175	GNDEN	
174	HOLD	
173	IREFEN	
172	MUXEN_7	
171	MUXEN_6	
170	MUXEN_5	
169	MUXEN_4	

Table 107. ATmega169 Boundary-scan Order (Continued)

Bit Number	Signal Name	Module
168	MUXEN_3	ADC
167	MUXEN_2	
166	MUXEN_1	
165	MUXEN_0	
164	NEGSEL_2	
163	NEGSEL_1	
162	NEGSEL_0	
161	PASSEN	
160	PRECH	
159	ST	
158	VCCREN	
157	PE0.Data	Port E
156	PE0.Control	
155	PE0.Pull-up_Enable	
154	PE1.Data	
153	PE1.Control	
152	PE1.Pull-up_Enable	
151	PE2.Data	
150	PE2.Control	
149	PE2.Pull-up_Enable	
148	PE3.Data	
147	PE3.Control	
146	PE3.Pull-up_Enable	
145	PE4.Data	
144	PE4.Control	
143	PE4.Pull-up_Enable	
142	PE5.Data	
141	PE5.Control	
140	PE5.Pull-up_Enable	
139	PE6.Data	
138	PE6.Control	
137	PE6.Pull-up_Enable	
136	PE7.Data	
135	PE7.Control	
134	PE7.Pull-up_Enable	
133	PB0.Data	Port B

Table 107. ATmega169 Boundary-scan Order (Continued)

Bit Number	Signal Name	Module
132	PB0.Control	Port B
131	PB0.Pull-up_Enable	
130	PB1.Data	
129	PB1.Control	
128	PB1.Pull-up_Enable	
127	PB2.Data	
126	PB2.Control	
125	PB2.Pull-up_Enable	
124	PB3.Data	
123	PB3.Control	
122	PB3.Pull-up_Enable	
121	PB4.Data	
120	PB4.Control	
119	PB4.Pull-up_Enable	
118	PB5.Data	
117	PB5.Control	
116	PB5.Pull-up_Enable	
115	PB6.Data	
114	PB6.Control	
113	PB6.Pull-up_Enable	
112	PB7.Data	
111	PB7.Control	
110	PB7.Pull-up_Enable	
109	PG3.Data	Port G
108	PG3.Control	
107	PG3.Pull-up_Enable	
106	PG4.Data	
105	PG4.Control	
104	PG4.Pull-up_Enable	
103	PG5	(Observe Only)
102	RSTT	Reset Logic (Observe-only)
101	RSTHV	
100	EXTCLKEN	Enable signals for main Clock/Oscillators
99	OSCON	
98	RCOSCEN	
97	OSC32EN	

Table 107. ATmega169 Boundary-scan Order (Continued)

Bit Number	Signal Name	Module
96	EXTCLK (XTAL1)	Clock input and Oscillators for the main clock (Observe-only)
95	OSCCK	
94	RCCK	
93	OSC32CK	
92	PD0.Data	Port D
91	PD0.Control	
90	PD0.Pull-up_Enable	
89	PD1.Data	
88	PD1.Control	
87	PD1.Pull-up_Enable	
86	PD2.Data	
85	PD2.Control	
84	PD2.Pull-up_Enable	
83	PD3.Data	
82	PD3.Control	
81	PD3.Pull-up_Enable	
80	PD4.Data	
79	PD4.Control	
78	PD4.Pull-up_Enable	
77	PD5.Data	
76	PD5.Control	
75	PD5.Pull-up_Enable	
74	PD6.Data	
73	PD6.Control	
72	PD6.Pull-up_Enable	
71	PD7.Data	
70	PD7.Control	
69	PD7.Pull-up_Enable	
68	PG0.Data	Port G
67	PG0.Control	
66	PG0.Pull-up_Enable	
65	PG1.Data	
64	PG1.Control	
63	PG1.Pull-up_Enable	
62	PC0.Data	Port C
61	PC0.Control	

Table 107. ATmega169 Boundary-scan Order (Continued)

Bit Number	Signal Name	Module
60	PC0.Pull-up_Enable	Port C
59	PC1.Data	
58	PC1.Control	
57	PC1.Pull-up_Enable	
56	PC2.Data	
55	PC2.Control	
54	PC2.Pull-up_Enable	
53	PC3.Data	
52	PC3.Control	
51	PC3.Pull-up_Enable	
50	PC4.Data	
49	PC4.Control	
48	PC4.Pull-up_Enable	
47	PC5.Data	
46	PC5.Control	
45	PC5.Pull-up_Enable	
44	PC6.Data	
43	PC6.Control	
42	PC6.Pull-up_Enable	
41	PC7.Data	
40	PC7.Control	
39	PC7.Pull-up_Enable	
38	PG2.Data	Port G
37	PG2.Control	
36	PG2.Pull-up_Enable	
35	PA7.Data	Port A
34	PA7.Control	
33	PA7.Pull-up_Enable	
32	PA6.Data	
31	PA6.Control	
30	PA6.Pull-up_Enable	
29	PA5.Data	
28	PA5.Control	
27	PA5.Pull-up_Enable	
26	PA4.Data	
25	PA4.Control	

Table 107. ATmega169 Boundary-scan Order (Continued)

Bit Number	Signal Name	Module
24	PA4.Pull-up_Enable	Port A
23	PA3.Data	
22	PA3.Control	
21	PA3.Pull-up_Enable	
20	PA2.Data	
19	PA2.Control	
18	PA2.Pull-up_Enable	
17	PA1.Data	
16	PA1.Control	
15	PA1.Pull-up_Enable	
14	PA0.Data	
13	PA0.Control	
12	PA0.Pull-up_Enable	
11	PF3.Data	Port F
10	PF3.Control	
9	PF3.Pull-up_Enable	
8	PF2.Data	
7	PF2.Control	
6	PF2.Pull-up_Enable	
5	PF1.Data	
4	PF1.Control	
3	PF1.Pull-up_Enable	
2	PF0.Data	
1	PF0.Control	
0	PF0.Pull-up_Enable	

Note: 1. PRIVATE_SIGNAL1 should always be scanned in as zero.

Boundary-scan Description Language Files

Boundary-scan Description Language (BSDL) files describe Boundary-scan capable devices in a standard format used by automated test-generation software. The order and function of bits in the Boundary-scan Data Register are included in this description. A BSDL file for ATmega169 is available.

Boot Loader Support – Read-While-Write Self-Programming

The Boot Loader Support provides a real Read-While-Write Self-Programming mechanism for downloading and uploading program code by the MCU itself. This feature allows flexible application software updates controlled by the MCU using a Flash-resident Boot Loader program. The Boot Loader program can use any available data interface and associated protocol to read code and write (program) that code into the Flash memory, or read the code from the program memory. The program code within the Boot Loader section has the capability to write into the entire Flash, including the Boot Loader memory. The Boot Loader can thus even modify itself, and it can also erase itself from the code if the feature is not needed anymore. The size of the Boot Loader memory is configurable with fuses and the Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

Boot Loader Features

- **Read-While-Write Self-Programming**
- **Flexible Boot Memory Size**
- **High Security (Separate Boot Lock Bits for a Flexible Protection)**
- **Separate Fuse to Select Reset Vector**
- **Optimized Page(1) Size**
- **Code Efficient Algorithm**
- **Efficient Read-Modify-Write Support**

Note: 1. A page is a section in the Flash consisting of several bytes (see Table 121 on page 269) used during programming. The page organization does not affect normal operation.

Application and Boot Loader Flash Sections

The Flash memory is organized in two main sections, the Application section and the Boot Loader section (see Figure 117). The size of the different sections is configured by the BOOTSZ Fuses as shown in Table 113 on page 264 and Figure 117. These two sections can have different level of protection since they have different sets of Lock bits.

Application Section

The Application section is the section of the Flash that is used for storing the application code. The protection level for the Application section can be selected by the application Boot Lock bits (Boot Lock bits 0), see Table 109 on page 256. The Application section can never store any Boot Loader code since the SPM instruction is disabled when executed from the Application section.

BLS – Boot Loader Section

While the Application section is used for storing the application code, the The Boot Loader software must be located in the BLS since the SPM instruction can initiate a programming when executing from the BLS only. The SPM instruction can access the entire Flash, including the BLS itself. The protection level for the Boot Loader section can be selected by the Boot Loader Lock bits (Boot Lock bits 1), see Table 110 on page 256.

Read-While-Write and No Read-While-Write Flash Sections

Whether the CPU supports Read-While-Write or if the CPU is halted during a Boot Loader software update is dependent on which address that is being programmed. In addition to the two sections that are configurable by the BOOTSZ Fuses as described above, the Flash is also divided into two fixed sections, the Read-While-Write (RWW) section and the No Read-While-Write (NRWW) section. The limit between the RWW- and NRWW sections is given in Table 114 on page 264 and Figure 117 on page 255. The main difference between the two sections is:

- When erasing or writing a page located inside the RWW section, the NRWW section can be read during the operation.
- When erasing or writing a page located inside the NRWW section, the CPU is halted during the entire operation.

Note that the user software can never read any code that is located inside the RWW section during a Boot Loader software operation. The syntax “Read-While-Write section” refers to which section that is being programmed (erased or written), not which section that actually is being read during a Boot Loader software update.

RWW – Read-While-Write Section

If a Boot Loader software update is programming a page inside the RWW section, it is possible to read code from the Flash, but only code that is located in the NRWW section. During an on-going programming, the software must ensure that the RWW section never is being read. If the user software is trying to read code that is located inside the RWW section (i.e., by a call/jmp/lpm or an interrupt) during programming, the software might end up in an unknown state. To avoid this, the interrupts should either be disabled or moved to the Boot Loader section. The Boot Loader section is always located in the NRWW section. The RWW Section Busy bit (RWWSB) in the Store Program Memory Control and Status Register (SPMCSR) will be read as logical one as long as the RWW section is blocked for reading. After a programming is completed, the RWWSB must be cleared by software before reading code located in the RWW section. See “Store Program Memory Control and Status Register – SPMCSR” on page 257. for details on how to clear RWWSB.

NRWW – No Read-While-Write Section

The code located in the NRWW section can be read when the Boot Loader software is updating a page in the RWW section. When the Boot Loader code updates the NRWW section, the CPU is halted during the entire Page Erase or Page Write operation.

Table 108. Read-While-Write Features

Which Section does the Z-pointer Address During the Programming?	Which Section Can be Read During Programming?	Is the CPU Halted?	Read-While-Write Supported?
RWW Section	NRWW Section	No	Yes
NRWW Section	None	Yes	No

Figure 116. Read-While-Write vs. No Read-While-Write

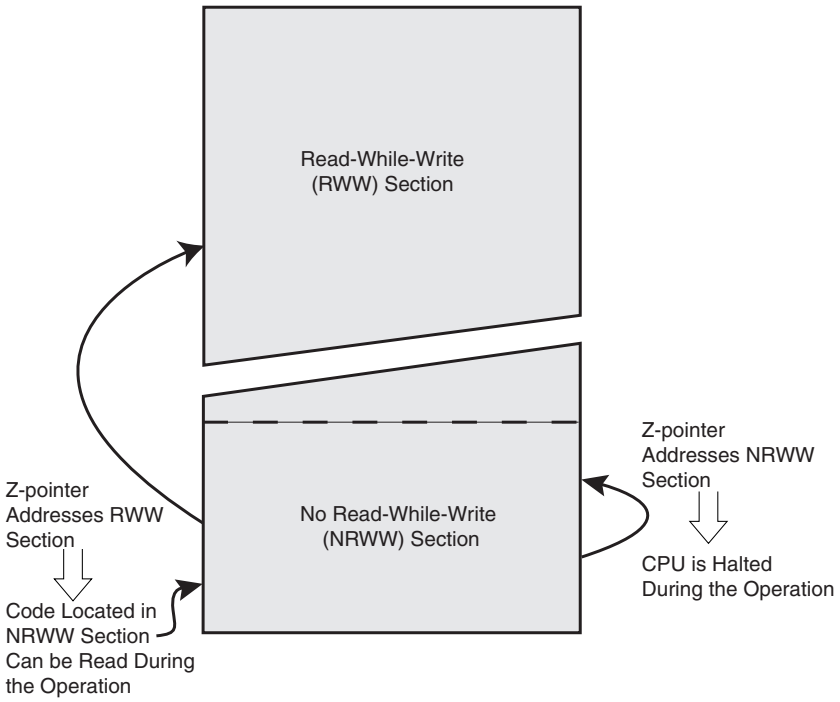
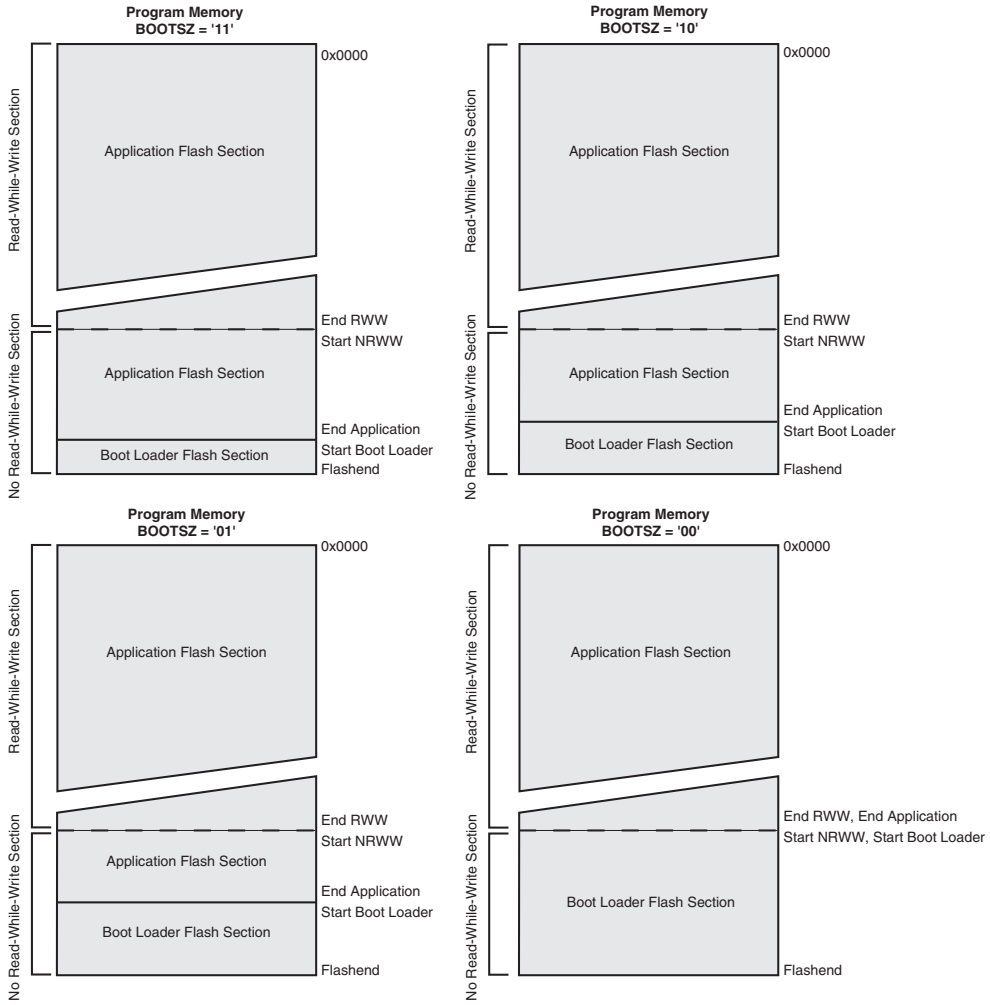


Figure 117. Memory Sections



Note: 1. The parameters in the figure above are given in Table 113 on page 264.

Boot Loader Lock Bits

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

The user can select:

- To protect the entire Flash from a software update by the MCU.
- To protect only the Boot Loader Flash section from a software update by the MCU.
- To protect only the Application Flash section from a software update by the MCU.
- Allow software update in the entire Flash.

See Table 109 and Table 110 for further details. The Boot Lock bits and general Lock bits can be set in software and in Serial or Parallel Programming mode, but they can be cleared by a Chip Erase command only. The general Write Lock (Lock Bit mode 2) does not control the programming of the Flash memory by SPM instruction. Similarly, the general Read/Write Lock (Lock Bit mode 1) does not control reading nor writing by LPM/SPM, if it is attempted.

Table 109. Boot Lock Bit0 Protection Modes (Application Section)[1]

BLB0 Mode	BLB02	BLB01	Protection
1	1	1	No restrictions for SPM or LPM accessing the Application section.
2	1	0	SPM is not allowed to write to the Application section.
3	0	0	SPM is not allowed to write to the Application section, and LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.
4	0	1	LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.

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

Table 110. Boot Lock Bit1 Protection Modes (Boot Loader Section)[1]

BLB1 Mode	BLB12	BLB11	Protection
1	1	1	No restrictions for SPM or LPM accessing the Boot Loader section.
2	1	0	SPM is not allowed to write to the Boot Loader section.
3	0	0	SPM is not allowed to write to the Boot Loader section, and LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.
4	0	1	LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.

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

Entering the Boot Loader Program

Entering the Boot Loader takes place by a jump or call from the application program. This may be initiated by a trigger such as a command received via USART, or SPI interface. Alternatively, the Boot Reset Fuse can be programmed so that the Reset Vector is pointing to the Boot Flash start address after a reset. In this case, the Boot Loader is started after a reset. After the application code is loaded, the program can start executing the application code. Note that the fuses cannot be changed by the MCU itself. This means that once the Boot Reset Fuse is programmed, the Reset Vector will always point to the Boot Loader Reset and the fuse can only be changed through the serial or parallel programming interface.

Table 111. Boot Reset Fuse[1]

BOOTRST	Reset Address
1	Reset Vector = Application Reset (address 0x0000)
0	Reset Vector = Boot Loader Reset (see Table 113 on page 264)

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

Store Program Memory Control and Status Register – SPMCSR

The Store Program Memory Control and Status Register contains the control bits needed to control the Boot Loader operations.

Bit	7	6	5	4	3	2	1	0	
	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	SPMCSR
Read/Write	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – SPMIE: SPM Interrupt Enable**

When the SPMIE bit is written to one, and the I-bit in the Status Register is set (one), the SPM ready interrupt will be enabled. The SPM ready Interrupt will be executed as long as the SPMEN bit in the SPMCSR Register is cleared.

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

When a Self-Programming (Page Erase or Page Write) operation to the RWW section is initiated, the RWWSB will be set (one) by hardware. When the RWWSB bit is set, the RWW section cannot be accessed. The RWWSB bit will be cleared if the RWWSRE bit is written to one after a Self-Programming operation is completed. Alternatively the RWWSB bit will automatically be cleared if a page load operation is initiated.

- **Bit 5 – Res: Reserved Bit**

This bit is a reserved bit in the ATmega169 and always read as zero.

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

When programming (Page Erase or Page Write) to the RWW section, the RWW section is blocked for reading (the RWWSB will be set by hardware). To re-enable the RWW section, the user software must wait until the programming is completed (SPMEN will be cleared). Then, if the RWWSRE bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles re-enables the RWW section. The RWW section cannot be re-enabled while the Flash is busy with a Page Erase or a Page Write (SPMEN is set). If the RWWSRE bit is written while the Flash is being loaded, the Flash load operation will abort and the data loaded will be lost.

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

If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles sets Boot Lock bits and general Lock bits, according to the data in R0. The data in R1 and the address in the Z-pointer are ignored. The BLBSET bit will automatically be cleared upon completion of the Lock bit set, or if no SPM instruction is executed within four clock cycles.

An LPM instruction within three cycles after BLBSET and SPMEN are set in the SPMCSR Register, will read either the Lock bits or the Fuse bits (depending on Z0 in the Z-pointer) into the destination register. See “Reading the Fuse and Lock Bits from Software” on page 261 for details.

- **Bit 2 – PGWRT: Page Write**

If this bit is written to one at the same time as SPMEN, the next SPM instruction within four clock cycles executes Page Write, with the data stored in the temporary buffer. The page address is taken from the high part of the Z-pointer. The data in R1 and R0 are ignored. The PGWRT bit will auto-clear upon completion of a Page Write, or if no SPM instruction is executed within four clock cycles. The CPU is halted during the entire Page Write operation if the NRWW section is addressed.

- **Bit 1 – PGERS: Page Erase**

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

- **Bit 0 – SPMEN: Store Program Memory Enable**

This bit enables the SPM instruction for the next four clock cycles. If written to one together with either RWWSRE, BLBSET, PGWRT' or PGERS, the following SPM instruction will have a special meaning, see description above. If only SPMEN is written, the following SPM instruction will store the value in R1:R0 in the temporary page buffer addressed by the Z-pointer. The LSB of the Z-pointer is ignored. The SPMEN bit will auto-clear upon completion of an SPM instruction, or if no SPM instruction is executed within four clock cycles. During Page Erase and Page Write, the SPMEN bit remains high until the operation is completed.

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

Addressing the Flash During Self-Programming

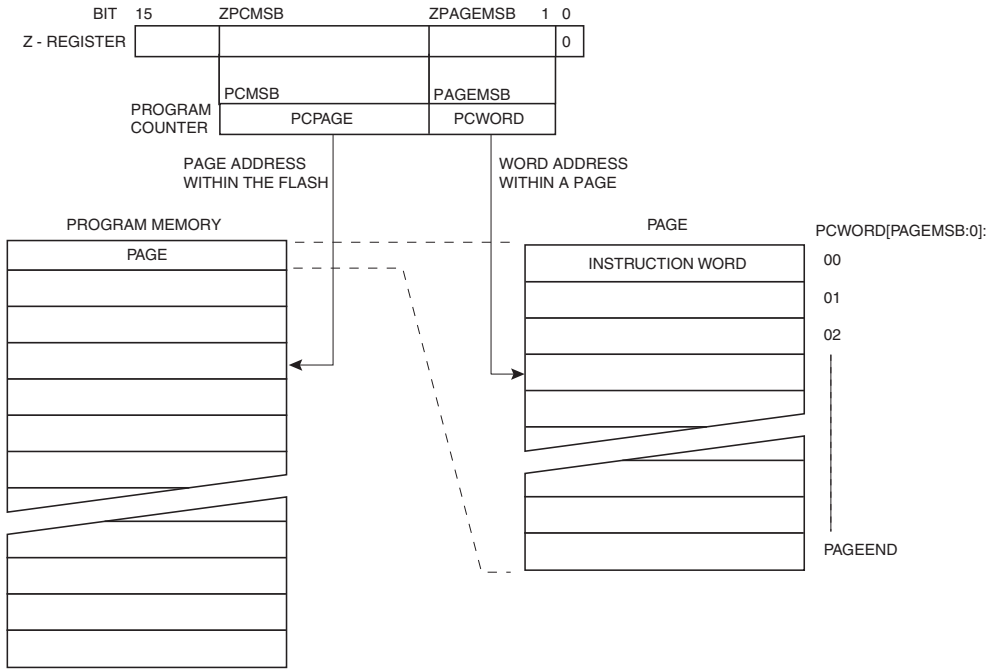
The Z-pointer is used to address the SPM commands.

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 121 on page 269), 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 is shown in Figure 118. 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 118. Addressing the Flash During SPM[1]



Note: 1. The different variables used in Figure 118 are listed in Table 115 on page 265.
2. PCPAGE and PCWORD are listed in Table 121 on page 269.

Self-Programming the Flash

The program memory is updated in a page by page fashion. Before programming a page with the data stored in the temporary page buffer, the page must be erased. The temporary page buffer is filled one word at a time using SPM and the buffer can be filled either before the Page Erase command or between a Page Erase and a Page Write operation:

Alternative 1, fill the buffer before a Page Erase

- Fill temporary page buffer
- Perform a Page Erase
- Perform a Page Write

Alternative 2, fill the buffer after Page Erase

- Perform a Page Erase
- Fill temporary page buffer
- Perform a Page Write

If only a part of the page needs to be changed, the rest of the page must be stored (for example in the temporary page buffer) before the erase, and then be rewritten. When using alternative 1, the Boot Loader provides an effective Read-Modify-Write feature which allows the user software to first read the page, do the necessary changes, and then write back the modified data. If alternative 2 is used, it is not possible to read the old data while loading since the page is already erased. The temporary page buffer can be accessed in a random sequence. It is essential that the page address used in both the Page Erase and Page Write operation is addressing the same page. See “Simple Assembly Code Example for a Boot Loader” on page 263 for an assembly code example.

Performing Page Erase by SPM

To execute Page Erase, set up the address in the Z-pointer, write “X0000011” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The data in R1 and R0 is ignored. The page address must be written to PCPAGE in the Z-register. Other bits in the Z-pointer will be ignored during this operation.

- Page Erase to the RWW section: The NRWW section can be read during the Page Erase.
- Page Erase to the NRWW section: The CPU is halted during the operation.

Filling the Temporary Buffer (Page Loading)

To write an instruction word, set up the address in the Z-pointer and data in R1:R0, write “00000001” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The content of PCWORD in the Z-register is used to address the data in the temporary buffer. The temporary buffer will auto-erase after a Page Write operation or by writing the RWWSRE bit in SPMCSR. It is also erased after a system reset. Note that it is not possible to write more than one time to each address without erasing the temporary buffer.

If the EEPROM is written in the middle of an SPM Page Load operation, all data loaded will be lost.

Performing a Page Write

To execute Page Write, set up the address in the Z-pointer, write “X0000101” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR. The data in R1 and R0 is ignored. The page address must be written to PCPAGE. Other bits in the Z-pointer must be written to zero during this operation.

- Page Write to the RWW section: The NRWW section can be read during the Page Write.
- Page Write to the NRWW section: The CPU is halted during the operation.

Using the SPM Interrupt

If the SPM interrupt is enabled, the SPM interrupt will generate a constant interrupt when the SPMEN bit in SPMCSR is cleared. This means that the interrupt can be used instead of polling the SPMCSR Register in software. When using the SPM interrupt, the Interrupt Vectors should be moved to the BLS section to avoid that an interrupt is accessing the RWW section when it is blocked for reading. How to move the interrupts is described in “Interrupts” on page 46.

Consideration While Updating BLS

Special care must be taken if the user allows the Boot Loader section to be updated by leaving Boot Lock bit11 unprogrammed. An accidental write to the Boot Loader itself can corrupt the entire Boot Loader, and further software updates might be impossible. If it is not necessary to change the Boot Loader software itself, it is recommended to program the Boot Lock bit11 to protect the Boot Loader software from any internal software changes.

Prevent Reading the RWW Section During Self-Programming

During Self-Programming (either Page Erase or Page Write), the RWW section is always blocked for reading. The user software itself must prevent that this section is addressed during the self programming operation. The RWWSB in the SPMCSR will be set as long as the RWW section is busy. During Self-Programming the Interrupt Vector table should be moved to the BLS as described in “Interrupts” on page 46, or the interrupts must be disabled. Before addressing the RWW section after the programming is completed, the user software must clear the RWWSB by writing the RWWSRE. See “Simple Assembly Code Example for a Boot Loader” on page 263 for an example.

Setting the Boot Loader Lock Bits by SPM

To set the Boot Loader Lock bits and general Lock bits, write the desired data to R0, write “X0001001” to SPMCSR and execute SPM within four clock cycles after writing SPMCSR.

Bit	7	6	5	4	3	2	1	0
R0	1	1	BLB12	BLB11	BLB02	BLB01	LB2	LB1

See Table 109 and Table 110 for how the different settings of the Boot Loader bits affect the Flash access.

If bits 5..2 in R0 are cleared (zero), the corresponding Boot Lock bit will be programmed if an SPM instruction is executed within four cycles after BLBSET and SPMEN are set in SPMCSR. The Z-pointer is don’t care during this operation, but for future compatibility it is recommended to load the Z-pointer with 0x0001 (same as used for reading the Lock bits). For future compatibility it is also recommended to set bits 7 and 6 in R0 to “1” when writing the Lock bits. When programming the Lock bits the entire Flash can be read during the operation.

EEPROM Write Prevents Writing to SPMCSR

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

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

Bit	7	6	5	4	3	2	1	0
Rd	–	–	BLB12	BLB11	BLB02	BLB01	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 SPMEN bits in SPMCSR. When an LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in the SPMCSR, the value of the Fuse Low byte (FLB) will be loaded in the destination register as shown below. Refer to Table 120 on page 268 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, load 0x0003 in the Z-pointer. When an LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in the SPMCSR, the value of the Fuse High byte (FHB) will be loaded in the destination register as shown below. Refer to Table 119 on page 268 for detailed description and mapping of the Fuse High byte.

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

When reading the Extended Fuse byte, load 0x0002 in the Z-pointer. When an LPM instruction is executed within three cycles after the BLBSET and SPMEN bits are set in

the SPMCSR, the value of the Extended Fuse byte (EFB) will be loaded in the destination register as shown below. Refer to Table 118 on page 267 for detailed description and mapping of the Extended Fuse byte.

Bit	7	6	5	4	3	2	1	0
Rd	–	–	–	–	EFB3	EFB2	EFB1	EFB0

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

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. If there is no need for a Boot Loader update in the system, program the Boot Loader Lock bits to prevent any Boot Loader software updates.
2. 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.
3. 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.

Programming Time for Flash when Using SPM

The calibrated RC Oscillator is used to time Flash accesses. Table 112 shows the typical programming time for Flash accesses from the CPU.

Table 112. SPM Programming Time

Symbol	Min Programming Time	Max Programming Time
Flash write (Page Erase, Page Write, and write Lock bits by SPM)	3.7 ms	4.5 ms

Simple Assembly Code Example for a Boot Loader

```
  ;-the routine writes one page of data from RAM to Flash
  ; the first data location in RAM is pointed to by the Y pointer
  ; the first data location in Flash is pointed to by the Z-pointer
  ;-error handling is not included
  ;-the routine must be placed inside the Boot space
  ; (at least the Do_spm sub routine). Only code inside NRWW section can
  ; be read during Self-Programming (Page Erase and Page Write).
  ;-registers used: r0, r1, temp1 (r16), temp2 (r17), looplo (r24),
  ; loophi (r25), spmcrval (r20)
  ; storing and restoring of registers is not included in the routine
  ; register usage can be optimized at the expense of code size
  ;-It is assumed that either the interrupt table is moved to the Boot
  ; loader section or that the interrupts are disabled.
.equ  PAGESIZEB = PAGESIZE*2      ;PAGESIZEB is page size in BYTES, not words
.org SMALLBOOTSTART
Write_page:
  ; Page Erase
  ldi   spmcrval, (1<<PGERS) | (1<<SPMEN)
  call  Do_spm

  ; re-enable the RWW section
  ldi   spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call  Do_spm

  ; transfer data from RAM to Flash page buffer
  ldi   looplo, low(PAGESIZEB)      ;init loop variable
  ldi   loophi, high(PAGESIZEB)     ;not required for PAGESIZEB<=256
Wrloop:
  ld    r0, Y+
  ld    r1, Y+
  ldi   spmcrval, (1<<SPMEN)
  call  Do_spm
  adiw  ZH:ZL, 2
  sbiw  loophi:looplo, 2            ;use subi for PAGESIZEB<=256
  brne  Wrloop

  ; execute Page Write
  subi  ZL, low(PAGESIZEB)          ;restore pointer
  sbci  ZH, high(PAGESIZEB)         ;not required for PAGESIZEB<=256
  ldi   spmcrval, (1<<PGWRT) | (1<<SPMEN)
  call  Do_spm

  ; re-enable the RWW section
  ldi   spmcrval, (1<<RWWSRE) | (1<<SPMEN)
  call  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
  jmp   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<<SPMEN)
  call  Do_spm
  rjmp  Return

Do_spm:
  ; check for previous SPM complete
Wait_spm:
  in    temp1, SPMCSR
  sbrc  temp1, SPMEN
  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, EEWE
  rjmp  Wait_ee
  ; SPM timed sequence
  out   SPMCSR, spmcrval
  spm
  ; restore SREG (to enable interrupts if originally enabled)
  out   SREG, temp2
  ret
```

ATmega169 Boot Loader Parameters

In Table 113 through Table 115, the parameters used in the description of the Self-Programming are given.

Table 113. Boot Size Configuration[1]

BOOTSZ1	BOOTSZ0	Boot Size	Pages	Application Flash Section	Boot Loader Flash Section	End Application Section	Boot Reset Address (Start Boot Loader Section)
1	1	128 words	2	0x0000 - 0x1F7F	0x1F80 - 0x1FFF	0x1F7F	0x1F80
1	0	256 words	4	0x0000 - 0x1EFF	0x1F00 - 0x1FFF	0x1EFF	0x1F00
0	1	512 words	8	0x0000 - 0x1DFF	0x1E00 - 0x1FFF	0x1DFF	0x1E00
0	0	1024 words	16	0x0000 - 0x1BFF	0x1C00 - 0x1FFF	0x1BFF	0x1C00

Note: 1. The different BOOTSZ Fuse configurations are shown in Figure 117

Table 114. Read-While-Write Limit[1]

Section	Pages	Address
Read-While-Write section (RWW)	112	0x0000 - 0x1BFF
No Read-While-Write section (NRWW)	16	0x1C00 - 0x1FFF

Note: 1. For details about these two section, see “NRWW – No Read-While-Write Section” on page 253 and “RWW – Read-While-Write Section” on page 253.

Table 115. Explanation of different variables used in Figure 118 and the mapping to the Z-pointer(1)

Variable		Corresponding Z-value	Description
PCMSB	12		Most significant bit in the Program Counter. (The Program Counter is 13 bits PC[12:0])
PAGEMSB	5		Most significant bit which is used to address the words within one page (64 words in a page requires six bits PC [5:0]).
ZPCMSB		Z13	Bit in Z-register that is mapped to PCMSB. Because Z0 is not used, the ZPCMSB equals PCMSB + 1.
ZPAGEMSB		Z6	Bit in Z-register that is mapped to PAGEMSB. Because Z0 is not used, the ZPAGEMSB equals PAGEMSB + 1.
PCPAGE	PC[12:6]	Z13:Z7	Program Counter page address: Page select, for Page Erase and Page Write
PCWORD	PC[5:0]	Z6:Z1	Program Counter word address: Word select, for filling temporary buffer (must be zero during Page Write operation)

Note: 1. Z15:Z14: always ignored
Z0: should be zero for all SPM commands, byte select for the LPM instruction.
See “Addressing the Flash During Self-Programming” on page 258 for details about the use of Z-pointer during Self-Programming.

Memory Programming

Program And Data Memory Lock Bits

The ATmega169 provides six Lock bits which can be left unprogrammed (“1”) or can be programmed (“0”) to obtain the additional features listed in Table 117. The Lock bits can only be erased to “1” with the Chip Erase command.

Table 116. 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 117. 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]
BLB0 Mode	**BLB02**	**BLB01**	
1	1	1	No restrictions for SPM or LPM accessing the Application section.
2	1	0	SPM is not allowed to write to the Application section.
3	0	0	SPM is not allowed to write to the Application section, and LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.
4	0	1	LPM executing from the Boot Loader section is not allowed to read from the Application section. If Interrupt Vectors are placed in the Boot Loader section, interrupts are disabled while executing from the Application section.
BLB1 Mode	**BLB12**	**BLB11**	

Table 117. Lock Bit Protection Modes[1][2] (Continued)

Memory Lock Bits			Protection Type
1	1	1	No restrictions for SPM or LPM accessing the Boot Loader section.
2	1	0	SPM is not allowed to write to the Boot Loader section.
3	0	0	SPM is not allowed to write to the Boot Loader section, and LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.
4	0	1	LPM executing from the Application section is not allowed to read from the Boot Loader section. If Interrupt Vectors are placed in the Application section, interrupts are disabled while executing from the Boot Loader section.

Notes: 1. Program the Fuse bits and Boot Lock bits before programming the LB1 and LB2.
2. “1” means unprogrammed, “0” means programmed

Fuse Bits

The ATmega169 has three Fuse bytes. Table 118 - Table 120 describe briefly the functionality of all the fuses and how they are mapped into the Fuse bytes. Note that the fuses are read as logical zero, “0”, if they are programmed.

Table 118. Extended Fuse Byte

Fuse Low Byte	Bit No	Description	Default Value
–	7	–	1
–	6	–	1
–	5	–	1
–	4	–	1
BODLEVEL2[1]	3	Brown-out Detector trigger level	1 (unprogrammed)
BODLEVEL1[1]	2	Brown-out Detector trigger level	1 (unprogrammed)
BODLEVEL0[1]	1	Brown-out Detector trigger level	1 (unprogrammed)
RESERVED[2]	0		1 (unprogrammed)

Notes: 1. See Table 17 on page 40 for BODLEVEL Fuse decoding.
2. This bit should never be programmed.

Table 119. Fuse High Byte

Fuse High Byte	Bit No	Description	Default Value
OCDEN[4]	7	Enable OCD	1 (unprogrammed, OCD disabled)
JTAGEN[5]	6	Enable JTAG	0 (programmed, JTAG enabled)
SPIEN[1]	5	Enable Serial Program and Data Downloading	0 (programmed, SPI prog. enabled)
WDTON[3]	4	Watchdog Timer always on	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed, EEPROM not preserved)
BOOTSZ1	2	Select Boot Size (see Table 113 for details)	0 (programmed)[2]
BOOTSZ0	1	Select Boot Size (see Table 113 for details)	0 (programmed)[2]
BOOTRST	0	Select Reset Vector	1 (unprogrammed)

Note:
1. The SPIEN Fuse is not accessible in serial programming mode.
2. The default value of BOOTSZ1..0 results in maximum Boot Size. See Table 113 on page 264 for details.
3. See “Watchdog Timer Control Register – WDTCR” on page 43 for details.
4. Never ship a product with the OCDEN Fuse programmed regardless of the setting of Lock bits and JTAGEN Fuse. A programmed OCDEN Fuse enables some parts of the clock system to be running in all sleep modes. This may increase the power consumption.
5. If the JTAG interface is left unconnected, the JTAGEN fuse should if possible be disabled. This to avoid static current at the TDO pin in the JTAG interface.

Table 120. Fuse Low Byte

Fuse Low Byte	Bit No	Description	Default Value
CKDIV8[4]	7	Divide clock by 8	0 (programmed)
CKOUT[3]	6	Clock output	1 (unprogrammed)
SUT1	5	Select start-up time	1 (unprogrammed)[1]
SUT0	4	Select start-up time	0 (programmed)[1]
CKSEL3	3	Select Clock source	0 (programmed)[2]
CKSEL2	2	Select Clock source	0 (programmed)[2]
CKSEL1	1	Select Clock source	1 (unprogrammed)[2]
CKSEL0	0	Select Clock source	0 (programmed)[2]

Note:
1. The default value of SUT1..0 results in maximum start-up time for the default clock source. See Table 16 on page 38 for details.
2. The default setting of CKSEL3..0 results in internal RC Oscillator @ 8 MHz. See Table 6 on page 26 for details.
3. The CKOUT Fuse allow the system clock to be output on PORTE7. See “Clock Output Buffer” on page 29 for details.
4. See “System Clock Prescaler” on page 29 for details.

The status of the Fuse bits is not affected by Chip Erase. Note that the Fuse bits are locked if Lock bit1 (LB1) is programmed. Program the Fuse bits before programming the Lock bits.

Latching of Fuses

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

Signature Bytes

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

For the ATmega169 the signature bytes are:

1. 0x000: 0x1E (indicates manufactured by Atmel).
2. 0x001: 0x94 (indicates 16KB Flash memory).
3. 0x002: 0x05 (indicates ATmega169 device when 0x001 is 0x94).

Calibration Byte

The ATmega169 has a byte calibration value for the internal RC Oscillator. This byte resides in the high byte of address 0x000 in the signature address space. During reset, this byte is automatically written into the OSCCAL Register to ensure correct frequency of the calibrated RC Oscillator.

Page Size

Table 121. No. of Words in a Page and No. of Pages in the Flash

Flash Size	Page Size	PCWORD	No. of Pages	PCPAGE	PCMSB
8K words (16K bytes)	64 words	PC[5:0]	128	PC[12:6]	12

Table 122. No. of Words in a Page and No. of Pages in the EEPROM

EEPROM Size	Page Size	PCWORD	No. of Pages	PCPAGE	EEAMSB
512 bytes	4 bytes	EEA[1:0]	128	EEA[8:2]	8

Parallel Programming Parameters, Pin Mapping, and Commands

This section describes how to parallel program and verify Flash Program memory, EEPROM Data memory, Memory Lock bits, and Fuse bits in the ATmega169. Pulses are assumed to be at least 250 ns unless otherwise noted.

Signal Names

In this section, some pins of the ATmega169 are referenced by signal names describing their functionality during parallel programming, see Figure 119 and Table 123. Pins not described in the following table are referenced by pin names.

The XA1/XA0 pins determine the action executed when the XTAL1 pin is given a positive pulse. The bit coding is shown in Table 125.

When pulsing $\overline{WR}$ or $\overline{OE}$, the command loaded determines the action executed. The different Commands are shown in Table 126.

Figure 119. Parallel Programming

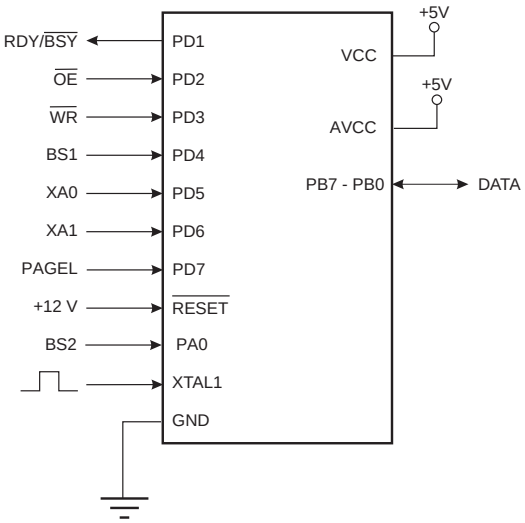


Table 123. Pin Name Mapping

Signal Name in Programming Mode	Pin Name	I/O	Function
RDY/$\overline{BSY}$	PD1	O	0: Device is busy programming, 1: Device is ready for new command.
$\overline{OE}$	PD2	I	Output Enable (Active low).
$\overline{WR}$	PD3	I	Write Pulse (Active low).
BS1	PD4	I	Byte Select 1 (“0” selects low byte, “1” selects high byte).
XA0	PD5	I	XTAL Action Bit 0
XA1	PD6	I	XTAL Action Bit 1
PAGEL	PD7	I	Program Memory and EEPROM data Page Load.
BS2	PA0	I	Byte Select 2 (“0” selects low byte, “1” selects 2’nd high byte).
DATA	PB7-0	I/O	Bi-directional Data bus (Output when $\overline{OE}$ is low).

Table 124. Pin Values Used to Enter Programming Mode

Pin	Symbol	Value
PAGEL	Prog_enable[3]	0
XA1	Prog_enable[2]	0
XA0	Prog_enable[1]	0
BS1	Prog_enable[0]	0

Table 125. 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 126. 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

Serial Programming Pin Mapping

Table 127. Pin Mapping Serial Programming

Symbol	Pins	I/O	Description
MOSI	PB2	I	Serial Data in
MISO	PB3	O	Serial Data out
SCK	PB1	I	Serial Clock

Parallel Programming

Enter Programming Mode

The following algorithm puts the device in parallel programming mode:

1. Apply 4.5 - 5.5V between V_{CC} and GND.
2. Set $\overline{\text{RESET}}$ to “0” and toggle XTAL1 at least six times.
3. Set the Prog_enable pins listed in Table 124 on page 271 to “0000” and wait at least 100 ns.
4. Apply 11.5 - 12.5V to $\overline{\text{RESET}}$. Any activity on Prog_enable pins within 100 ns after +12V has been applied to $\overline{\text{RESET}}$, will cause the device to fail entering programming mode.
5. Wait at least 50 µs before sending a new command.

Considerations for Efficient Programming

The loaded command and address are retained in the device during programming. For efficient programming, the following should be considered.

- The command needs only be loaded once when writing or reading multiple memory locations.
- Skip writing the data value 0xFF, that is the contents of the entire EEPROM (unless the EESAVE Fuse is programmed) and Flash after a Chip Erase.
- Address high byte needs only be loaded before programming or reading a new 256 word window in Flash or 256 byte EEPROM. This consideration also applies to Signature bytes reading.

Chip Erase

The Chip Erase will erase the Flash and EEPROM[1] memories plus Lock bits. The Lock bits are not reset until the program memory has been completely erased. The Fuse bits are not changed. A Chip Erase must be performed before the Flash and/or EEPROM are reprogrammed.

Note: 1. The EEPRPOM memory is preserved during Chip Erase if the EESAVE Fuse is programmed.

Load Command “Chip Erase”

1. Set XA1, XA0 to “10”. This enables command loading.
2. Set BS1 to “0”.
3. Set DATA to “1000 0000”. This is the command for Chip Erase.
4. Give XTAL1 a positive pulse. This loads the command.
5. Give $\overline{\text{WR}}$ a negative pulse. This starts the Chip Erase. RDY/$\overline{\text{BSY}}$ goes low.
6. Wait until RDY/$\overline{\text{BSY}}$ goes high before loading a new command.

Programming the Flash

The Flash is organized in pages, see Table 121 on page 269. When programming the Flash, the program data is latched into a page buffer. This allows one page of program data to be programmed simultaneously. The following procedure describes how to program the entire Flash memory:

A. Load Command “Write Flash”

1. Set XA1, XA0 to “10”. This enables command loading.
2. Set BS1 to “0”.
3. Set DATA to “0001 0000”. This is the command for Write Flash.
4. Give XTAL1 a positive pulse. This loads the command.

B. Load Address Low byte

1. Set XA1, XA0 to “00”. This enables address loading.
2. Set BS1 to “0”. This selects low address.
3. Set DATA = Address low byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the address low byte.

C. Load Data Low Byte

1. Set XA1, XA0 to “01”. This enables data loading.
2. Set DATA = Data low byte (0x00 - 0xFF).
3. Give XTAL1 a positive pulse. This loads the data byte.

D. Load Data High Byte

1. Set BS1 to “1”. This selects high data byte.
2. Set XA1, XA0 to “01”. This enables data loading.
3. Set DATA = Data high byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the data byte.

E. Latch Data

1. Set BS1 to “1”. This selects high data byte.
2. Give PAGEL a positive pulse. This latches the data bytes. (See Figure 121 for signal waveforms)

F. Repeat B through E until the entire buffer is filled or until all data within the page is loaded.

While the lower bits in the address are mapped to words within the page, the higher bits address the pages within the FLASH. This is illustrated in Figure 120 on page 275. Note that if less than eight bits are required to address words in the page (pagesize < 256), the most significant bit(s) in the address low byte are used to address the page when performing a Page Write.

G. Load Address High byte

1. Set XA1, XA0 to “00”. This enables address loading.
2. Set BS1 to “1”. This selects high address.
3. Set DATA = Address high byte (0x00 - 0xFF).
4. Give XTAL1 a positive pulse. This loads the address high byte.

H. Program Page

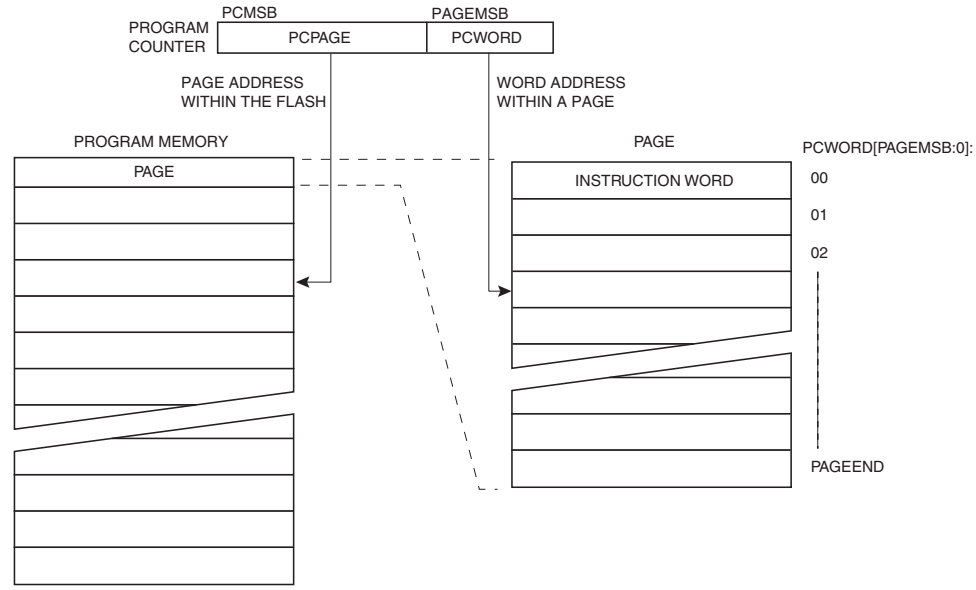
1. Give $\overline{\text{WR}}$ a negative pulse. This starts programming of the entire page of data. RDY/$\overline{\text{BSY}}$ goes low.
2. Wait until RDY/$\overline{\text{BSY}}$ goes high (See Figure 121 for signal waveforms).

I. Repeat B through H until the entire Flash is programmed or until all data has been programmed.

J. End Page Programming

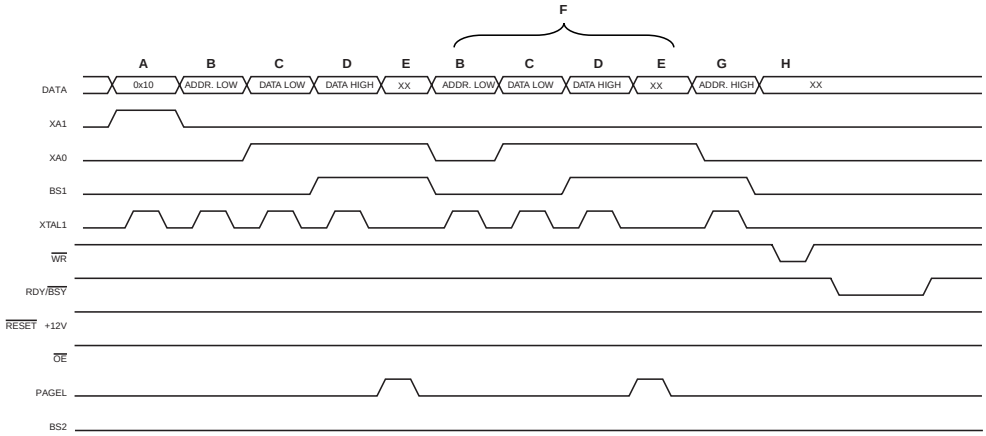
1. 1. Set XA1, XA0 to “10”. This enables command loading.
2. Set DATA to “0000 0000”. This is the command for No Operation.
3. Give XTAL1 a positive pulse. This loads the command, and the internal write signals are reset.

Figure 120. Addressing the Flash Which is Organized in Pages(1)



Note: 1. PCPAGE and PCWORD are listed in Table 121 on page 269.

Figure 121. Programming the Flash Waveforms(1)



Note: 1. "XX" is don't care. The letters refer to the programming description above.

Programming the EEPROM

The EEPROM is organized in pages, see Table 122 on page 269. When programming the EEPROM, the program data is latched into a page buffer. This allows one page of data to be programmed simultaneously. The programming algorithm for the EEPROM data memory is as follows (refer to "Programming the Flash" on page 273 for details on Command, Address and Data loading):

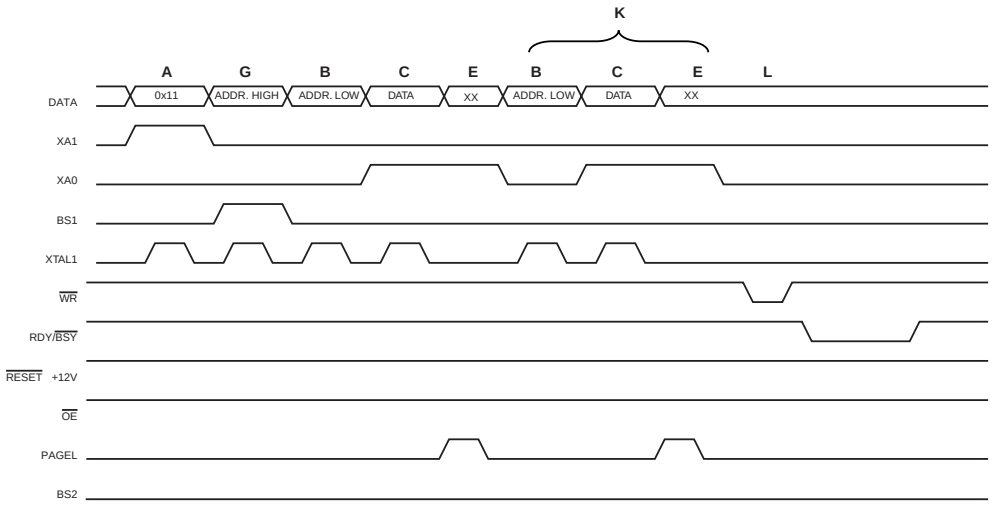
1. A: Load Command "0001 0001".
2. G: Load Address High Byte (0x00 - 0xFF).
3. B: Load Address Low Byte (0x00 - 0xFF).
4. C: Load Data (0x00 - 0xFF).
5. E: Latch data (give PAGEL a positive pulse).

K: Repeat 3 through 5 until the entire buffer is filled.

L: Program EEPROM page

1. Set BS to “0”.
2. Give $\overline{WR}$ a negative pulse. This starts programming of the EEPROM page. RDY/$\overline{BSY}$ goes low.
3. Wait until to RDY/$\overline{BSY}$ goes high before programming the next page (See Figure 122 for signal waveforms).

Figure 122. Programming the EEPROM Waveforms



Reading the Flash

The algorithm for reading the Flash memory is as follows (refer to “Programming the Flash” on page 273 for details on Command and Address loading):

1. A: Load Command “0000 0010”.
2. G: Load Address High Byte (0x00 - 0xFF).
3. B: Load Address Low Byte (0x00 - 0xFF).
4. Set $\overline{OE}$ to “0”, and BS1 to “0”. The Flash word low byte can now be read at DATA.
5. Set BS to “1”. The Flash word high byte can now be read at DATA.
6. Set $\overline{OE}$ to “1”.

Reading the EEPROM

The algorithm for reading the EEPROM memory is as follows (refer to “Programming the Flash” on page 273 for details on Command and Address loading):

1. A: Load Command “0000 0011”.
2. G: Load Address High Byte (0x00 - 0xFF).
3. B: Load Address Low Byte (0x00 - 0xFF).
4. Set $\overline{OE}$ to “0”, and BS1 to “0”. The EEPROM Data byte can now be read at DATA.
5. Set $\overline{OE}$ to “1”.

Programming the Fuse Low Bits

The algorithm for programming the Fuse Low bits is as follows (refer to “Programming the Flash” on page 273 for details on Command and Data loading):

1. A: Load Command “0100 0000”.
2. C: Load Data Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.

Programming the Fuse High Bits

The algorithm for programming the Fuse High bits is as follows (refer to “Programming the Flash” on page 273 for details on Command and Data loading):

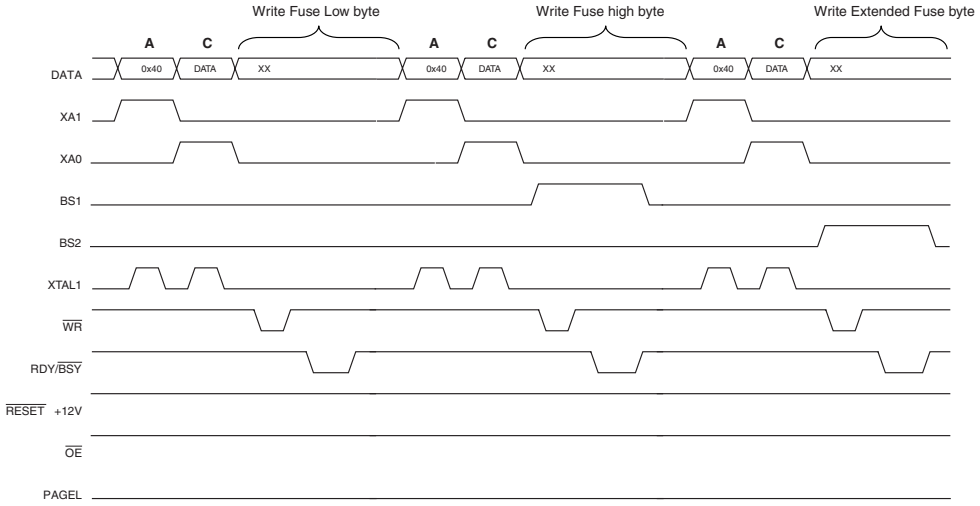
1. A: Load Command “0100 0000”.
2. C: Load Data Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. Set BS1 to “1” and BS2 to “0”. This selects high fuse byte.
4. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.
5. Set BS1 to “0”. This selects low data byte.

Programming the Extended Fuse Bits

The algorithm for programming the Extended Fuse bits is as follows (refer to “Programming the Flash” on page 273 for details on Command and Data loading):

1. 1. A: Load Command “0100 0000”.
2. 2. C: Load Data Byte. Bit n = “0” programs and bit n = “1” erases the Fuse bit.
3. 3. Set BS1 to “0” and BS2 to “1”. This selects extended fuse byte.
4. 4. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.
5. 5. Set BS2 to “0”. This selects low data byte.

Figure 123. Programming the FUSES Waveforms



Programming the Lock Bits

The algorithm for programming the Lock bits is as follows (refer to “Programming the Flash” on page 273 for details on Command and Data loading):

1. A: Load Command “0010 0000”.
2. C: Load Data Low Byte. Bit n = “0” programs the Lock bit. If LB mode 3 is programmed (LB1 and LB2 is programmed), it is not possible to program the Boot Lock bits by any External Programming mode.
3. Give $\overline{WR}$ a negative pulse and wait for RDY/$\overline{BSY}$ to go high.

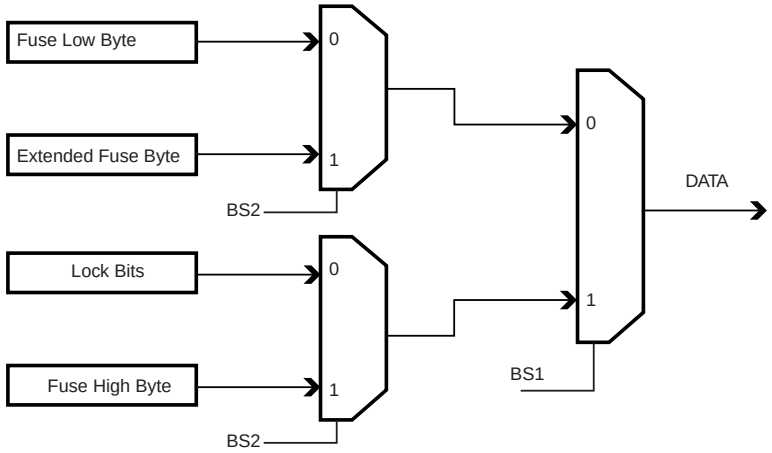
The Lock bits can only be cleared by executing Chip Erase.

Reading the Fuse and Lock Bits

The algorithm for reading the Fuse and Lock bits is as follows (refer to “Programming the Flash” on page 273 for details on Command loading):

1. A: Load Command “0000 0100”.
2. Set $\overline{OE}$ to “0”, BS2 to “0” and BS1 to “0”. The status of the Fuse Low bits can now be read at DATA (“0” means programmed).
3. Set $\overline{OE}$ to “0”, BS2 to “1” and BS1 to “1”. The status of the Fuse High bits can now be read at DATA (“0” means programmed).
4. Set OE to “0”, BS2 to “1”, and BS1 to “0”. The status of the Extended Fuse bits can now be read at DATA (“0” means programmed).
5. Set $\overline{OE}$ to “0”, BS2 to “0” and BS1 to “1”. The status of the Lock bits can now be read at DATA (“0” means programmed).
6. Set $\overline{OE}$ to “1”.

Figure 124. Mapping Between BS1, BS2 and the Fuse and Lock Bits During Read



Reading the Signature Bytes

The algorithm for reading the Signature bytes is as follows (refer to “Programming the Flash” on page 273 for details on Command and Address loading):

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

Reading the Calibration Byte

The algorithm for reading the Calibration byte is as follows (refer to “Programming the Flash” on page 273 for details on Command and Address loading):

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

Parallel Programming Characteristics

Figure 125. Parallel Programming Timing, Including some General Timing Requirements

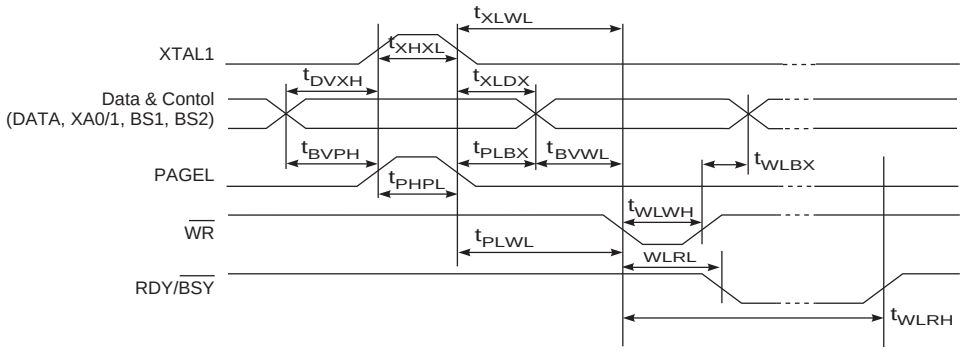
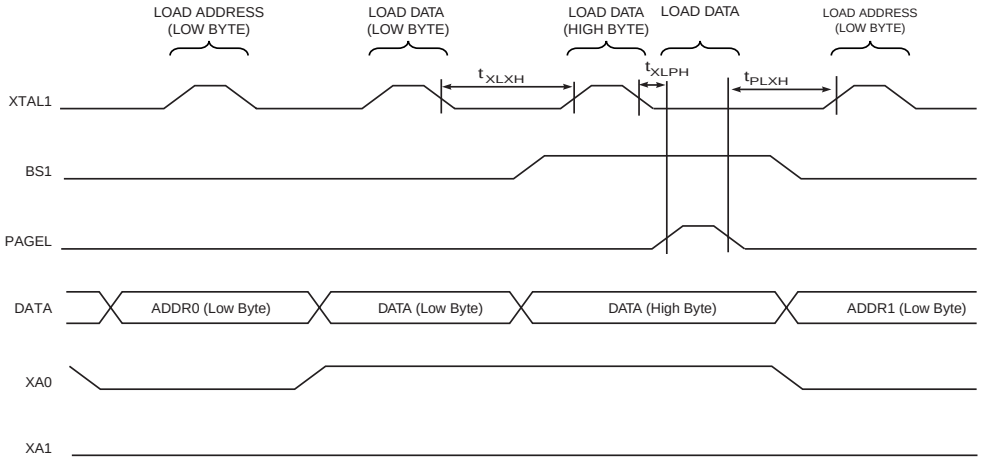
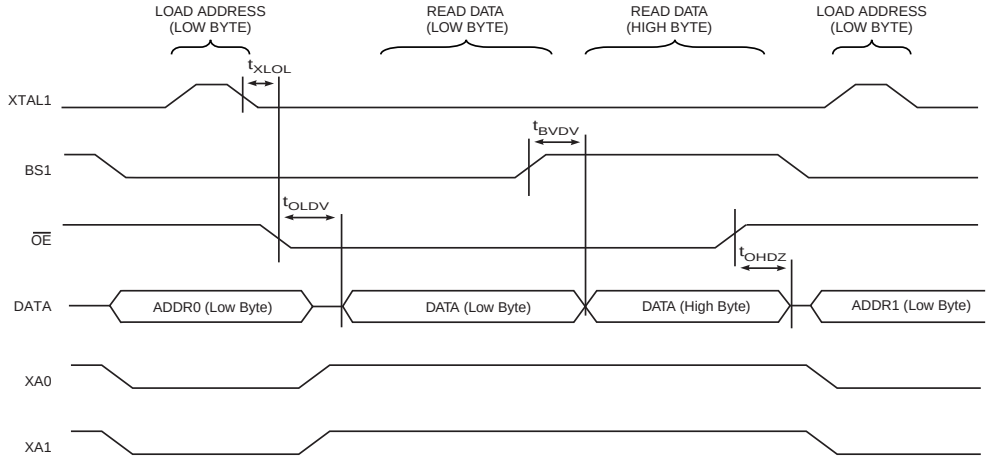


Figure 126. Parallel Programming Timing, Loading Sequence with Timing Requirements(1)



Note: 1. The timing requirements shown in Figure 125 (i.e., t_{DVXH}, t_{XHXL}, and t_{XLDX}) also apply to loading operation.

Figure 127. Parallel Programming Timing, Reading Sequence (within the Same Page) with Timing Requirements(1)



Note: 1. The timing requirements shown in Figure 125 (i.e., t_{DVXH}, t_{XHXL}, and t_{XLDX}) also apply to reading operation.

Table 128. Parallel Programming Characteristics, V_{CC} = 5V ± 10%

Symbol	Parameter	Min	Typ	Max	Units
V_{PP}	Programming Enable Voltage	11.5		12.5	V
I_{PP}	Programming Enable Current			250	µA
t_{DVXH}	Data and Control Valid before XTAL1 High	67			ns
t_{XLXH}	XTAL1 Low to XTAL1 High	200			ns
t_{XHXL}	XTAL1 Pulse Width High	150			ns
t_{XLDX}	Data and Control Hold after XTAL1 Low	67			ns
t_{XLWL}	XTAL1 Low to $\overline{WR}$ Low	0			ns
t_{XLPH}	XTAL1 Low to PAGEL high	0			ns
t_{PLXH}	PAGEL low to XTAL1 high	150			ns
t_{BVPH}	BS1 Valid before PAGEL High	67			ns
t_{PHPL}	PAGEL Pulse Width High	150			ns
t_{PLBX}	BS1 Hold after PAGEL Low	67			ns
t_{WLBX}	BS2/1 Hold after $\overline{WR}$ Low	67			ns
t_{PLWL}	PAGEL Low to $\overline{WR}$ Low	67			ns
t_{BVWL}	BS1 Valid to $\overline{WR}$ Low	67			ns
t_{WLWH}	$\overline{WR}$ Pulse Width Low	150			ns
t_{WLRL}	$\overline{WR}$ Low to RDY/$\overline{BSY}$ Low	0		1	µs
t_{WLRH}	$\overline{WR}$ Low to RDY/$\overline{BSY}$ High(1)	3.7		4.5	ms
t_{WLRH_CE}	$\overline{WR}$ Low to RDY/$\overline{BSY}$ High for Chip Erase(2)	7.5		9	ms
t_{XLOL}	XTAL1 Low to $\overline{OE}$ Low	0			ns

Table 128. Parallel Programming Characteristics, V_{CC} = 5V ± 10% (Continued)

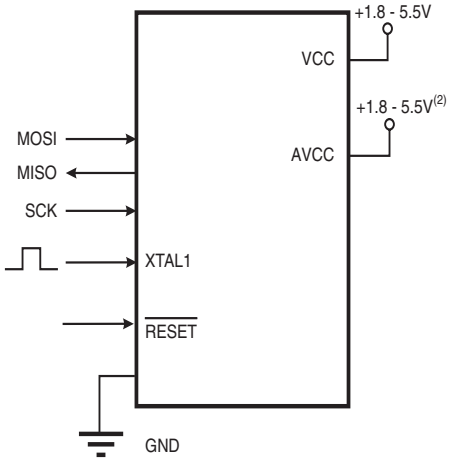
Symbol	Parameter	Min	Typ	Max	Units
t_{BVDV}	BS1 Valid to DATA valid	0		250	ns
t_{OLDV}	$\overline{OE}$ Low to DATA Valid			250	ns
t_{OHDZ}	$\overline{OE}$ High to DATA Tri-stated			250	ns

Notes: 1. t_{WLRH} is valid for the Write Flash, Write EEPROM, Write Fuse bits and Write Lock bits commands.
2. t_{WLRH_CE} is valid for the Chip Erase command.

Serial Downloading

Both the Flash and EEPROM memory arrays can be programmed using the serial SPI bus while $\overline{RESET}$ is pulled to GND. The serial interface consists of pins SCK, MOSI (input) and MISO (output). After $\overline{RESET}$ is set low, the Programming Enable instruction needs to be executed first before program/erase operations can be executed. NOTE, in Table 127 on page 272, the pin mapping for SPI programming is listed. Not all parts use the SPI pins dedicated for the internal SPI interface.

Figure 128. Serial Programming and Verify(1)



Notes: 1. If the device is clocked by the internal Oscillator, it is no need to connect a clock source to the XTAL1 pin.
2. V_{CC} - 0.3V < AVCC < V_{CC} + 0.3V, however, AVCC should always be within 1.8 - 5.5V

When programming the EEPROM, an auto-erase cycle is built into the self-timed programming operation (in the Serial mode ONLY) and there is no need to first execute the Chip Erase instruction. The Chip Erase operation turns the content of every memory location in both the Program and EEPROM arrays into 0xFF.

Depending on CKSEL Fuses, a valid clock must be present. The minimum low and high periods for the serial clock (SCK) input are defined as follows:

Low:> 2 CPU clock cycles for f_{ck} < 12 MHz, 3 CPU clock cycles for f_{ck} >= 12 MHz

High:> 2 CPU clock cycles for f_{ck} < 12 MHz, 3 CPU clock cycles for f_{ck} >= 12 MHz

Serial Programming Algorithm

When writing serial data to the ATmega169, data is clocked on the rising edge of SCK.

When reading data from the ATmega169, data is clocked on the falling edge of SCK. See Figure 129 for timing details.

To program and verify the ATmega169 in the serial programming mode, the following sequence is recommended (See four byte instruction formats in Table 130):

1. Power-up sequence:
   Apply power between V_{CC} and GND while $\overline{\text{RESET}}$ and SCK are set to “0”. In some systems, the programmer can not guarantee that SCK is held low during power-up. In this case, $\overline{\text{RESET}}$ must be given a positive pulse of at least two CPU clock cycles duration after SCK has been set to “0”.
2. Wait for at least 20 ms and enable serial programming by sending the Programming Enable serial instruction to pin MOSI.
3. The serial programming instructions will not work if the communication is out of synchronization. When in sync. the second byte (0x53), will echo back when issuing the third byte of the Programming Enable instruction. Whether the echo is correct or not, all four bytes of the instruction must be transmitted. If the 0x53 did not echo back, give $\overline{\text{RESET}}$ a positive pulse and issue a new Programming Enable command.
4. The Flash is programmed one page at a time. The page size is found in Table 121 on page 269. The memory page is loaded one byte at a time by supplying the 6 LSB of the address and data together with the Load Program Memory Page instruction. To ensure correct loading of the page, the data low byte must be loaded before data high byte is applied for a given address. The Program Memory Page is stored by loading the Write Program Memory Page instruction with the 7 MSB of the address. If polling (RDY/$\overline{\text{BSY}}$) is not used, the user must wait at least t_{WD_FLASH} before issuing the next page. (See Table 129.) Accessing the serial programming interface before the Flash write operation completes can result in incorrect programming.
5. **A:** The EEPROM array is programmed one byte at a time by supplying the address and data together with the appropriate Write instruction. An EEPROM memory location is first automatically erased before new data is written. If polling (RDY/$\overline{\text{BSY}}$) is not used, the user must wait at least t_{WD_EEPROM} before issuing the next byte (See Table 129). In a chip erased device, no 0xFFs in the data file(s) need to be programmed.
   B: The EEPROM array is programmed one page at a time. The Memory page is loaded one byte at a time by supplying the 2 LSB of the address and data together with the Load EEPROM Memory Page instruction. The EEPROM Memory Page is stored by loading the Write EEPROM Memory Page Instruction with the 4 MSB of the address. When using EEPROM page access only byte locations loaded with the Load EEPROM Memory Page instruction is altered. The remaining locations remain unchanged. If polling (RDY/$\overline{\text{BSY}}$) is not used, the user must wait at least t_{WD_EEPROM} before issuing the next page (See Table 129). In a chip erased device, no 0xFF in the data file(s) need to be programmed.
6. Any memory location can be verified by using the Read instruction which returns the content at the selected address at serial output MISO.
7. At the end of the programming session, $\overline{\text{RESET}}$ can be set high to commence normal operation.
8. Power-off sequence (if needed):
   Set $\overline{\text{RESET}}$ to “1”.
   Turn V_{CC} power off

Table 129. Minimum Wait Delay Before Writing the Next Flash or EEPROM Location

Symbol	Minimum Wait Delay
t_{WD_FUSE}	4.5 ms
t_{WD_FLASH}	4.5 ms
t_{WD_EEPROM}	9.0 ms
t_{WD_ERASE}	9.0 ms

Figure 129. Serial Programming Waveforms

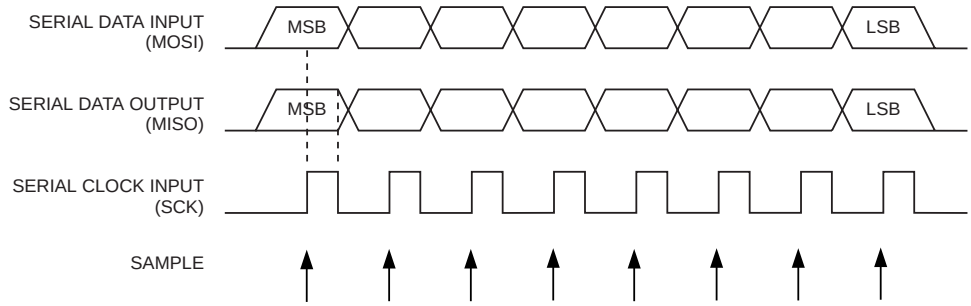


Table 130. Serial Programming Instruction Set

Instruction	Instruction Format				Operation
	Byte 1	Byte 2	Byte 3	Byte4	
Programming Enable	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	Enable Serial Programming after $\overline{\text{RESET}}$ goes low.
Chip Erase	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	Chip Erase EEPROM and Flash.
Read Program Memory	0010 **H**000	000**a aaaa**	**bbbb bbbb**	**oooo oooo**	Read **H** (high or low) data **o** from Program memory at word address **a**:**b**.
Load Program Memory Page	0100 **H**000	000x xxxx	xx**bb bbbb**	**iiii iiii**	Write **H** (high or low) data **i** to Program Memory page at word address **b**. Data low byte must be loaded before Data high byte is applied within the same address.
Write Program Memory Page	0100 1100	000**a aaaa**	**bb**xx xxxx	xxxx xxxx	Write Program Memory Page at address **a**:**b**.
Read EEPROM Memory	1010 0000	000x xx**aa**	**bbbb bbbb**	**oooo oooo**	Read data **o** from EEPROM memory at address **a**:**b**.
Write EEPROM Memory	1100 0000	000x xx**aa**	**bbbb bbbb**	**iiii iiii**	Write data **i** to EEPROM memory at address **a**:**b**.
Load EEPROM Memory Page (page access)	1100 0001	0000 0000	0000 00**bb**	**iiii iiii**	Load data **i** to EEPROM memory page buffer. After data is loaded, program EEPROM page.
Write EEPROM Memory Page (page access)	1100 0010	00xx xx**aa**	**bbbb bb**00	xxxx xxxx	Write EEPROM page at address **a**:**b**.
Read Lock bits	0101 1000	0000 0000	xxxx xxxx	xx**oo oooo**	Read Lock bits. “0” = programmed, “1” = unprogrammed. See Table 116 on page 266 for details.
Write Lock bits	1010 1100	111x xxxx	xxxx xxxx	11**ii iiii**	Write Lock bits. Set bits = “0” to program Lock bits. See Table 116 on page 266 for details.
Read Signature Byte	0011 0000	000x xxxx	xxxx xx**bb**	**oooo oooo**	Read Signature Byte **o** at address **b**.
Write Fuse bits	1010 1100	1010 0000	xxxx xxxx	**iiii iiii**	Set bits = “0” to program, “1” to unprogram. See Table 88 on page 205 for details.
Write Fuse High bits	1010 1100	1010 1000	xxxx xxxx	**iiii iiii**	Set bits = “0” to program, “1” to unprogram. See Table 87 on page 198 for details.
Write Extended Fuse Bits	1010 1100	1010 0100	xxxx xxxx	**xxxx iii**1	Set bits = “0” to program, “1” to unprogram. See Table 118 on page 267 for details.
Read Fuse bits	0101 0000	0000 0000	xxxx xxxx	**oooo oooo**	Read Fuse bits. “0” = programmed, “1” = unprogrammed. See Table 88 on page 205 for details.
Read Fuse High bits	0101 1000	0000 1000	xxxx xxxx	**oooo oooo**	Read Fuse High bits. “0” = programmed, “1” = unprogrammed. See Table 87 on page 198 for details.

Table 130. Serial Programming Instruction Set (Continued)

Instruction	Instruction Format				Operation
	Byte 1	Byte 2	Byte 3	Byte4	
Read Extended Fuse Bits	0101 0000	0000 1000	xxxx xxxx	**oooo oooo**	Read Extended Fuse bits. “0” = programmed, “1” = unprogrammed. See Table 118 on page 267 for details.
Read Calibration Byte	0011 1000	000x xxxx	0000 0000	**oooo oooo**	Read Calibration Byte
Poll RDY/$\overline{\text{BSY}}$	1111 0000	0000 0000	xxxx xxxx	xxxx xxx**o**	If **o** = “1”, a programming operation is still busy. Wait until this bit returns to “0” before applying another command.

Note: **a** = address high bits, **b** = address low bits, **H** = 0 - Low byte, 1 - High Byte, **o** = data out, **i** = data in, x = don’t care

SPI Serial Programming Characteristics

For characteristics of the SPI module see “SPI Timing Characteristics” on page 301.

Programming via the JTAG Interface

Programming through the JTAG interface requires control of the four JTAG specific pins: TCK, TMS, TDI, and TDO. Control of the reset and clock pins is not required.

To be able to use the JTAG interface, the JTAGEN Fuse must be programmed. The device is default shipped with the fuse programmed. In addition, the JTD bit in MCUCSR must be cleared. Alternatively, if the JTD bit is set, the external reset can be forced low. Then, the JTD bit will be cleared after two chip clocks, and the JTAG pins are available for programming. This provides a means of using the JTAG pins as normal port pins in Running mode while still allowing In-System Programming via the JTAG interface. Note that this technique can not be used when using the JTAG pins for Boundary-scan or On-chip Debug. In these cases the JTAG pins must be dedicated for this purpose.

During programming the clock frequency of the TCK Input must be less than the maximum frequency of the chip. The System Clock Prescaler can not be used to divide the TCK Clock Input into a sufficiently low frequency.

As a definition in this datasheet, the LSB is shifted in and out first of all Shift Registers.

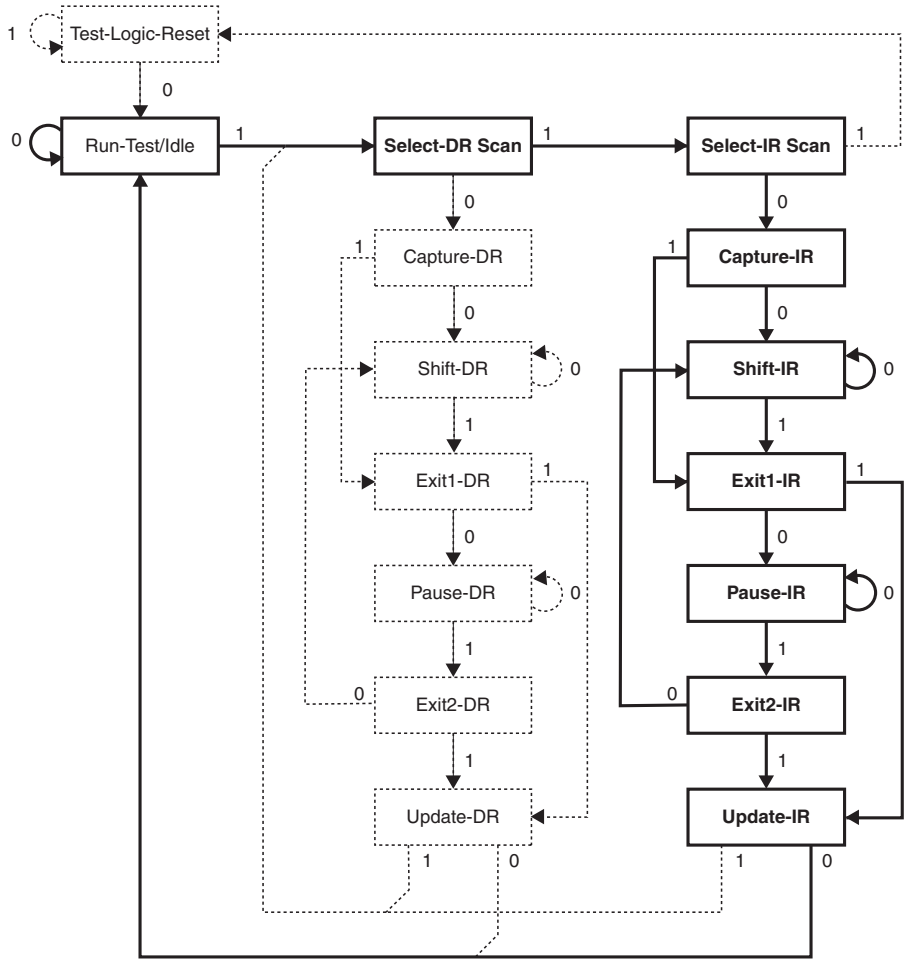
Programming Specific JTAG Instructions

The Instruction Register is 4-bit wide, supporting up to 16 instructions. The JTAG instructions useful for programming are listed below.

The OPCODE for each instruction is shown behind the instruction name in hex format. The text describes which Data Register is selected as path between TDI and TDO for each instruction.

The Run-Test/Idle state of the TAP controller is used to generate internal clocks. It can also be used as an idle state between JTAG sequences. The state machine sequence for changing the instruction word is shown in Figure 130.

Figure 130. State Machine Sequence for Changing the Instruction Word



AVR_RESET (0xC)

The AVR specific public JTAG instruction for setting the AVR device in the Reset mode or taking the device out from the Reset mode. The TAP controller is not reset by this instruction. The one bit Reset Register is selected as Data Register. Note that the reset will be active as long as there is a logic “one” in the Reset Chain. The output from this chain is not latched.

The active states are:

- Shift-DR: The Reset Register is shifted by the TCK input.

PROG_ENABLE (0x4)

The AVR specific public JTAG instruction for enabling programming via the JTAG port. The 16-bit Programming Enable Register is selected as Data Register. The active states are the following:

- Shift-DR: The programming enable signature is shifted into the Data Register.
- Update-DR: The programming enable signature is compared to the correct value, and Programming mode is entered if the signature is valid.

PROG_COMMANDS (0x5)

The AVR specific public JTAG instruction for entering programming commands via the JTAG port. The 15-bit Programming Command Register is selected as Data Register. The active states are the following:

- Capture-DR: The result of the previous command is loaded into the Data Register.
- Shift-DR: The Data Register is shifted by the TCK input, shifting out the result of the previous command and shifting in the new command.
- Update-DR: The programming command is applied to the Flash inputs
- Run-Test/Idle: One clock cycle is generated, executing the applied command (not always required, see Table 131 below).

PROG_PAGELOAD (0x6)

The AVR specific public JTAG instruction to directly load the Flash data page via the JTAG port. An 8-bit Flash Data Byte Register is selected as the Data Register. This is physically the 8 LSBs of the Programming Command Register. The active states are the following:

- Shift-DR: The Flash Data Byte Register is shifted by the TCK input.
- Update-DR: The content of the Flash Data Byte Register is copied into a temporary register. A write sequence is initiated 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.

PROG_PAGEREAD (0x7)

The AVR specific public JTAG instruction to directly capture the Flash content via the JTAG port. An 8-bit Flash Data Byte Register is selected as the Data Register. This is physically the 8 LSBs of the Programming Command Register. The active states are the following:

- Capture-DR: The content of the selected Flash byte is captured into the Flash Data Byte Register. 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 Capture-DR encountered after entering the PROG_PAGEREAD command. The Program Counter is post-incremented after reading each high byte, including the first read byte. This ensures that the first data is captured from the first address set up by PROG_COMMANDS, and reading the last location in the page makes the program counter increment into the next page.
- Shift-DR: The Flash Data Byte Register is shifted by the TCK input.

Data Registers

The Data Registers are selected by the JTAG instruction registers described in section “Programming Specific JTAG Instructions” on page 285. The Data Registers relevant for programming operations are:

- Reset Register
- Programming Enable Register
- Programming Command Register
- Flash Data Byte Register

Reset Register

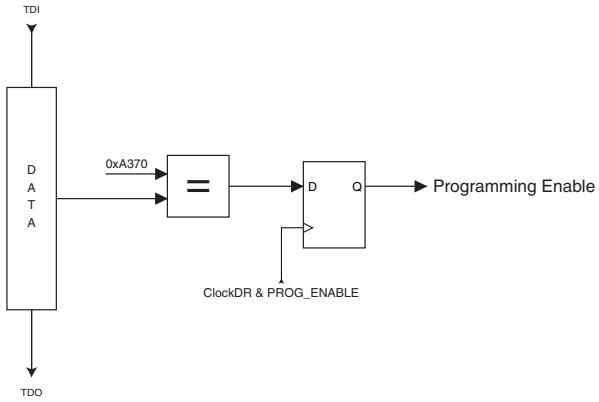
The Reset Register is a Test Data Register used to reset the part during programming. It is required to reset the part before entering Programming mode.

A high value in the Reset Register corresponds to pulling the external reset low. The part is reset as long as there is a high value present in the Reset Register. Depending on the Fuse settings for the clock options, the part will remain reset for a Reset Time-out period (refer to “Clock Sources” on page 24) after releasing the Reset Register. The output from this Data Register is not latched, so the reset will take place immediately, as shown in Figure 108 on page 234.

Programming Enable Register

The Programming Enable Register is a 16-bit register. The contents of this register is compared to the programming enable signature, binary code 0b1010_0011_0111_0000. When the contents of the register is equal to the programming enable signature, programming via the JTAG port is enabled. The register is reset to 0 on Power-on Reset, and should always be reset when leaving Programming mode.

Figure 131. Programming Enable Register



Programming Command Register

The Programming Command Register is a 15-bit register. This register is used to serially shift in programming commands, and to serially shift out the result of the previous command, if any. The JTAG Programming Instruction Set is shown in Table 131. The state sequence when shifting in the programming commands is illustrated in Figure 133.

Figure 132. Programming Command Register

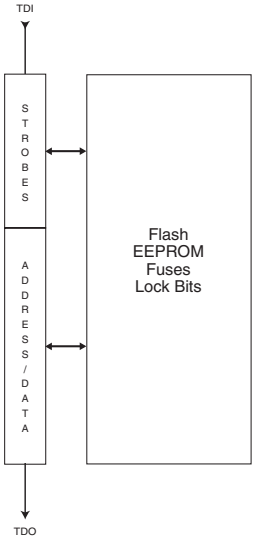


Table 131. JTAG Programming Instruction

Set **a** = address high bits, **b** = address low bits, **H** = 0 - Low byte, 1 - High Byte, **o** = data out, **i** = data in, x = don't care

Instruction	TDI Sequence	TDO Sequence	Notes
1a. Chip Erase	0100011_10000000 0110001_10000000 0110011_10000000 0110011_10000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	
1b. Poll for Chip Erase Complete	0110011_10000000	xxxxx**o**x_xxxxxxxx	(2)
2a. Enter Flash Write	0100011_00010000	xxxxxxx_xxxxxxxx	
2b. Load Address High Byte	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)
2c. Load Address Low Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
2d. Load Data Low Byte	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	
2e. Load Data High Byte	0010111_**iiiiiiii**	xxxxxxx_xxxxxxxx	
2f. Latch Data	0110111_00000000 1110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
2g. Write Flash Page	0110111_00000000 0110101_00000000 0110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
2h. Poll for Page Write Complete	0110111_00000000	xxxxx**o**x_xxxxxxxx	(2)
3a. Enter Flash Read	0100011_00000010	xxxxxxx_xxxxxxxx	
3b. Load Address High Byte	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)
3c. Load Address Low Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
3d. Read Data Low and High Byte	0110010_00000000 0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo** xxxxxxx_**oooooooo**	 Low byte High byte
4a. Enter EEPROM Write	0100011_00010001	xxxxxxx_xxxxxxxx	
4b. Load Address High Byte	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)
4c. Load Address Low Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
4d. Load Data Byte	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	
4e. Latch Data	0110111_00000000 1110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
4f. Write EEPROM Page	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
4g. Poll for Page Write Complete	0110011_00000000	xxxxx**o**x_xxxxxxxx	(2)
5a. Enter EEPROM Read	0100011_00000011	xxxxxxx_xxxxxxxx	
5b. Load Address High Byte	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)
5c. Load Address Low Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	

Table 131. JTAG Programming Instruction (Continued)
Set (Continued) **a** = address high bits, **b** = address low bits, **H** = 0 - Low byte, 1 - High Byte, **o** = data out, **i** = data in, x = don't care

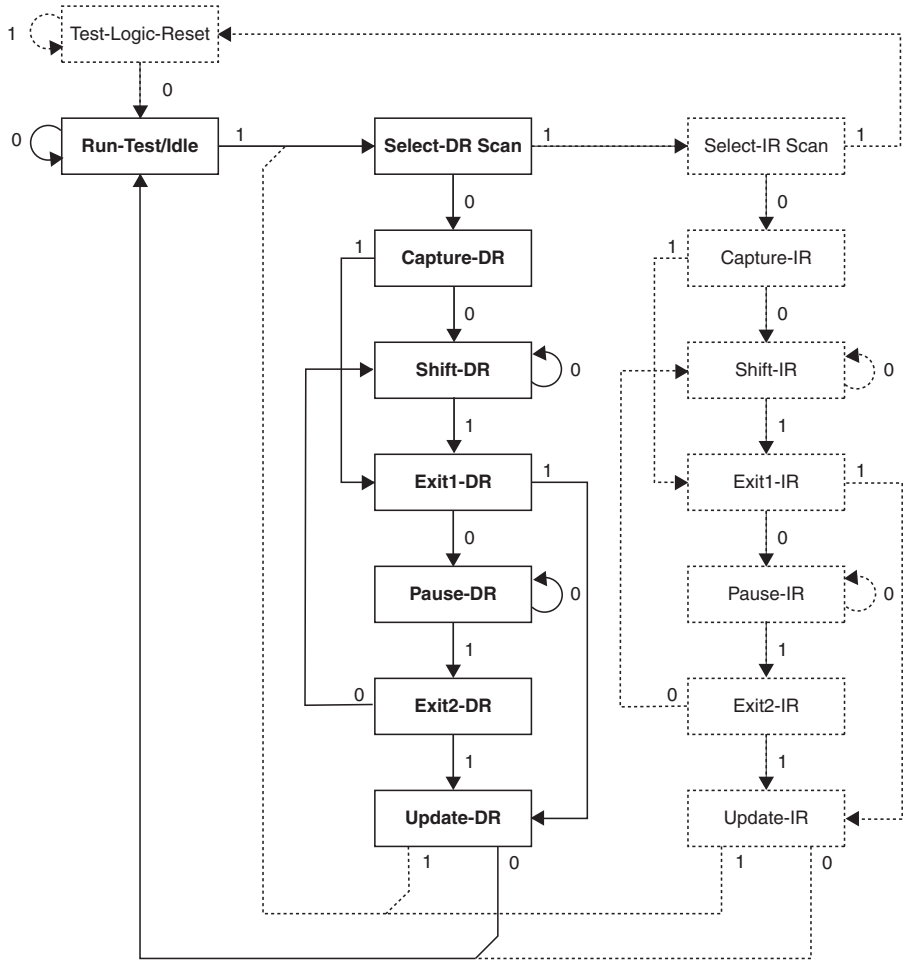
Instruction	TDI Sequence	TDO Sequence	Notes
5d. Read Data Byte	0110011_**bbbbbbbb** 0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
6a. Enter Fuse Write	0100011_01000000	xxxxxxx_xxxxxxxx	
6b. Load Data Low Byte(6)	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	(3)
6c. Write Fuse Extended Byte	0111011_00000000 0111001_00000000 0111011_00000000 0111011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
6d. Poll for Fuse Write Complete	0110111_00000000	xxxxx**o**x_xxxxxxxx	(2)
6e. Load Data Low Byte(7)	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	(3)
6f. Write Fuse High Byte	0110111_00000000 0110101_00000000 0110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
6g. Poll for Fuse Write Complete	0110111_00000000	xxxxx**o**x_xxxxxxxx	(2)
6h. Load Data Low Byte(7)	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	(3)
6i. Write Fuse Low Byte	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
6j. Poll for Fuse Write Complete	0110011_00000000	xxxxx**o**x_xxxxxxxx	(2)
7a. Enter Lock Bit Write	0100011_00100000	xxxxxxx_xxxxxxxx	
7b. Load Data Byte(9)	0010011_11**iiiiii**	xxxxxxx_xxxxxxxx	(4)
7c. Write Lock Bits	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
7d. Poll for Lock Bit Write complete	0110011_00000000	xxxxx**o**x_xxxxxxxx	(2)
8a. Enter Fuse/Lock Bit Read	0100011_00000100	xxxxxxx_xxxxxxxx	
8b. Read Extended Fuse Byte(6)	0111010_00000000 0111011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
8c. Read Fuse High Byte(7)	0111110_00000000 0111111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
8d. Read Fuse Low Byte(8)	0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
8e. Read Lock Bits(9)	0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xx**oooooo**	(5)

Table 131. JTAG Programming Instruction (Continued)
Set (Continued) **a** = address high bits, **b** = address low bits, **H** = 0 - Low byte, 1 - High Byte, **o** = data out, **i** = data in, x = don't care

Instruction	TDI Sequence	TDO Sequence	Notes
8f. Read Fuses and Lock Bits	0111010_00000000 0111110_00000000 0110010_00000000 0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo** xxxxxxx_**oooooooo** xxxxxxx_**oooooooo** xxxxxxx_**oooooooo**	(5) Fuse Ext. byte Fuse High byte Fuse Low byte Lock bits
9a. Enter Signature Byte Read	0100011_00001000	xxxxxxx_xxxxxxxx	
9b. Load Address Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
9c. Read Signature Byte	0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
10a. Enter Calibration Byte Read	0100011_00001000	xxxxxxx_xxxxxxxx	
10b. Load Address Byte	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
10c. Read Calibration Byte	0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
11a. Load No Operation Command	0100011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	

Notes:
1. This command sequence is not required if the seven MSB are correctly set by the previous command sequence (which is normally the case).
2. Repeat until **o** = "1".
3. Set bits to "0" to program the corresponding Fuse, "1" to unprogram the Fuse.
4. Set bits to "0" to program the corresponding Lock bit, "1" to leave the Lock bit unchanged.
5. "0" = programmed, "1" = unprogrammed.
6. The bit mapping for Fuses Extended byte is listed in Table 118 on page 267
7. The bit mapping for Fuses High byte is listed in Table 119 on page 268
8. The bit mapping for Fuses Low byte is listed in Table 120 on page 268
9. The bit mapping for Lock bits byte is listed in Table 116 on page 266
10. Address bits exceeding PCMSB and EEAMSB (Table 121 and Table 122) are don't care
11. All TDI and TDO sequences are represented by binary digits (0b...).

Figure 133. State Machine Sequence for Changing/Reading the Data Word



Flash Data Byte Register

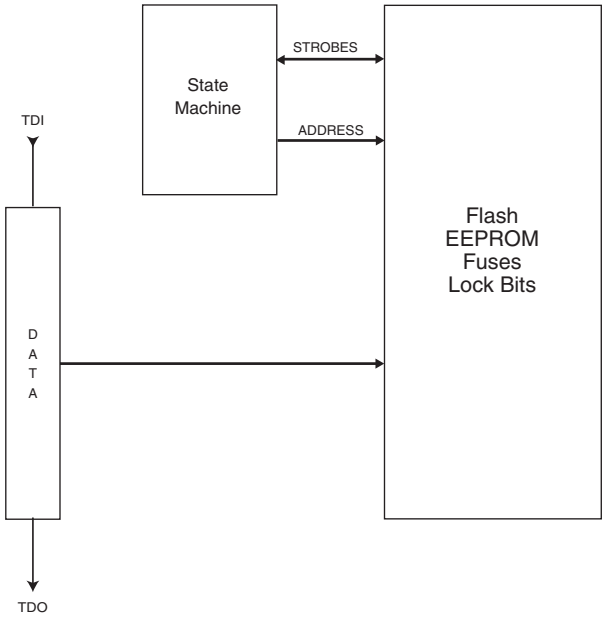
The Flash Data Byte Register provides an efficient way to load the entire Flash page buffer before executing Page Write, or to read out/verify the content of the Flash. A state machine sets up the control signals to the Flash and senses the strobe signals from the Flash, thus only the data words need 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 Capture-DR encountered after entering the PROG_PAGEREAD command. The Program Counter is post-incremented after reading each high byte,

including the first read byte. This ensures that the first data is captured from the first address set up by PROG_COMMANDS, and reading the last location in the page makes the program counter increment into the next page.

Figure 134. Flash Data Byte Register



The state machine controlling the Flash Data Byte Register is clocked by TCK. During normal operation in which eight bits are shifted for each Flash byte, the clock cycles needed to navigate through the TAP controller automatically feeds the state machine for the Flash Data Byte Register with sufficient number of clock pulses to complete its operation transparently for the user. However, if too few bits are shifted between each Update-DR state during page load, the TAP controller should stay in the Run-Test/Idle state for some TCK cycles to ensure that there are at least 11 TCK cycles between each Update-DR state.

Programming Algorithm

All references below of type “1a”, “1b”, and so on, refer to Table 131.

Entering Programming Mode

1. Enter JTAG instruction AVR_RESET and shift 1 in the Reset Register.
2. Enter instruction PROG_ENABLE and shift 0b1010_0011_0111_0000 in the Programming Enable Register.

Leaving Programming Mode

1. Enter JTAG instruction PROG_COMMANDS.
2. Disable all programming instructions by using no operation instruction 11a.
3. Enter instruction PROG_ENABLE and shift 0b0000_0000_0000_0000 in the programming Enable Register.
4. Enter JTAG instruction AVR_RESET and shift 0 in the Reset Register.

Performing Chip Erase

1. Enter JTAG instruction PROG_COMMANDS.
2. Start Chip Erase using programming instruction 1a.
3. Poll for Chip Erase complete using programming instruction 1b, or wait for t_{WLRH_CE} (refer to Table 128 on page 280).

Programming the Flash

Before programming the Flash a Chip Erase must be performed, see “Performing Chip Erase” on page 294.

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash write using programming instruction 2a.
3. Load address High byte using programming instruction 2b.
4. Load address Low byte using programming instruction 2c.
5. Load data using programming instructions 2d, 2e and 2f.
6. Repeat steps 4 and 5 for all instruction words in the page.
7. Write the page using programming instruction 2g.
8. Poll for Flash write complete using programming instruction 2h, or wait for t_{WLRH} (refer to Table 128 on page 280).
9. Repeat steps 3 to 7 until all data have been programmed.

A more efficient data transfer can be achieved using the PROG_PAGELOAD instruction:

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash write using programming instruction 2a.
3. Load the page address using programming instructions 2b and 2c. PCWORD (refer to Table 121 on page 269) is used to address within one page and must be written as 0.
4. Enter JTAG instruction PROG_PAGELOAD.
5. Load the entire page by shifting in all instruction words in the page byte-by-byte, starting with the LSB of the first instruction in the page and ending with the MSB of the last instruction in the page. Use Update-DR to copy the contents of the Flash Data Byte Register into the Flash page location and to auto-increment the Program Counter before each new word.
6. Enter JTAG instruction PROG_COMMANDS.
7. Write the page using programming instruction 2g.
8. Poll for Flash write complete using programming instruction 2h, or wait for t_{WLRH} (refer to Table 128 on page 280).
9. Repeat steps 3 to 8 until all data have been programmed.

Reading the Flash

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash read using programming instruction 3a.
3. Load address using programming instructions 3b and 3c.
4. Read data using programming instruction 3d.
5. Repeat steps 3 and 4 until all data have been read.

A more efficient data transfer can be achieved using the PROG_PAGEREAD instruction:

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Flash read using programming instruction 3a.
3. Load the page address using programming instructions 3b and 3c. PCWORD (refer to Table 121 on page 269) is used to address within one page and must be written as 0.
4. Enter JTAG instruction PROG_PAGEREAD.
5. Read the entire page (or Flash) by shifting out all instruction words in the page (or Flash), starting with the LSB of the first instruction in the page (Flash) and

ending with the MSB of the last instruction in the page (Flash). The Capture-DR state both captures the data from the Flash, and also auto-increments the program counter after each word is read. Note that Capture-DR comes before the shift-DR state. Hence, the first byte which is shifted out contains valid data.

6. Enter JTAG instruction PROG_COMMANDS.
7. Repeat steps 3 to 6 until all data have been read.

Programming the EEPROM

Before programming the EEPROM a Chip Erase must be performed, see "Performing Chip Erase" on page 294.

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable EEPROM write using programming instruction 4a.
3. Load address High byte using programming instruction 4b.
4. Load address Low byte using programming instruction 4c.
5. Load data using programming instructions 4d and 4e.
6. Repeat steps 4 and 5 for all data bytes in the page.
7. Write the data using programming instruction 4f.
8. Poll for EEPROM write complete using programming instruction 4g, or wait for t_{WLRH} (refer to Table 128 on page 280).
9. Repeat steps 3 to 8 until all data have been programmed.

Note that the PROG_PAGELOAD instruction can not be used when programming the EEPROM.

Reading the EEPROM

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable EEPROM read using programming instruction 5a.
3. Load address using programming instructions 5b and 5c.
4. Read data using programming instruction 5d.
5. Repeat steps 3 and 4 until all data have been read.

Note that the PROG_PAGEREAD instruction can not be used when reading the EEPROM.

Programming the Fuses

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Fuse write using programming instruction 6a.
3. Load data high byte using programming instructions 6b. A bit value of "0" will program the corresponding fuse, a "1" will unprogram the fuse.
4. Write Fuse High byte using programming instruction 6c.
5. Poll for Fuse write complete using programming instruction 6d, or wait for t_{WLRH} (refer to Table 128 on page 280).
6. Load data low byte using programming instructions 6e. A "0" will program the fuse, a "1" will unprogram the fuse.
7. Write Fuse low byte using programming instruction 6f.
8. Poll for Fuse write complete using programming instruction 6g, or wait for t_{WLRH} (refer to Table 128 on page 280).

Programming the Lock Bits

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Lock bit write using programming instruction 7a.
3. Load data using programming instructions 7b. A bit value of “0” will program the corresponding lock bit, a “1” will leave the lock bit unchanged.
4. Write Lock bits using programming instruction 7c.
5. Poll for Lock bit write complete using programming instruction 7d, or wait for t_{WLRH} (refer to Table 128 on page 280).

Reading the Fuses and Lock Bits

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Fuse/Lock bit read using programming instruction 8a.
3. To read all Fuses and Lock bits, use programming instruction 8e.
   To only read Fuse High byte, use programming instruction 8b.
   To only read Fuse Low byte, use programming instruction 8c.
   To only read Lock bits, use programming instruction 8d.

Reading the Signature Bytes

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Signature byte read using programming instruction 9a.
3. Load address 0x00 using programming instruction 9b.
4. Read first signature byte using programming instruction 9c.
5. Repeat steps 3 and 4 with address 0x01 and address 0x02 to read the second and third signature bytes, respectively.

Reading the Calibration Byte

1. Enter JTAG instruction PROG_COMMANDS.
2. Enable Calibration byte read using programming instruction 10a.
3. Load address 0x00 using programming instruction 10b.
4. Read the calibration byte using programming instruction 10c.

Electrical Characteristics

Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except $\overline{\text{RESET}}$ with respect to Ground	-0.5V to V_{CC}+0.5V
Voltage on $\overline{\text{RESET}}$ with respect to Ground	-0.5V to +13.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0 mA
DC Current V_{CC} and GND Pins	400.0 mA

*NOTICE: Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC Characteristics

T_A = -40°C to 85°C, V_{CC} = 1.8V to 5.5V (unless otherwise noted)

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
V_{IL}	Input Low Voltage except XTAL1 and $\overline{\text{RESET}}$ pins	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	-0.5 -0.5		$0.2V_{CC}$(1) $0.3V_{CC}$(1)	V
V_{IH}	Input High Voltage except XTAL1 and $\overline{\text{RESET}}$ pins	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.7V_{CC}$(2) $0.6V_{CC}$(2)		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{IL1}	Input Low Voltage XTAL1 pins	V_{CC} = 1.8V - 5.5V	-0.5		$0.1V_{CC}$(1)	V
V_{IH1}	Input High Voltage, XTAL1 pin	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.8V_{CC}$(2) $0.7V_{CC}$(2)		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{IL2}	Input Low Voltage, $\overline{\text{RESET}}$ pins	V_{CC} = 1.8V - 5.5V	-0.5		$0.1V_{CC}$(1) $0.2V_{CC}$(1)	V
V_{IH2}	Input High Voltage, $\overline{\text{RESET}}$ pins	V_{CC} = 1.8V - 5.5V	$0.9V_{CC}$(2)		V_{CC} + 0.5	V
V_{OL}	Output Low Voltage(3), Port A, C, D, E, F, G	I_{OL} = 10mA, V_{CC} = 5V I_{OL} = 5mA, V_{CC} = 3V			0.7 0.5	V
V_{OL1}	Output Low Voltage(3), Port B	I_{OL} = 20mA, V_{CC} = 5V I_{OL} = 10mA, V_{CC} = 3V			0.7 0.5	V
V_{OH}	Output High Voltage(4), Port A, C, D, E, F, G	I_{OH} = -10mA, V_{CC} = 5V I_{OH} = -5mA, V_{CC} = 3V	4.2 2.3			V
V_{OH1}	Output High Voltage(4), Port B	I_{OH} = -20mA, V_{CC} = 5V I_{OH} = -10mA, V_{CC} = 3V	4.2 2.3			V
I_{IL}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin low (absolute value)			1	µA
I_{IH}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin high (absolute value)			1	µA
R_{RST}	Reset Pull-up Resistor		30		60	kΩ
R_{PU}	I/O Pin Pull-up Resistor		20		50	kΩ

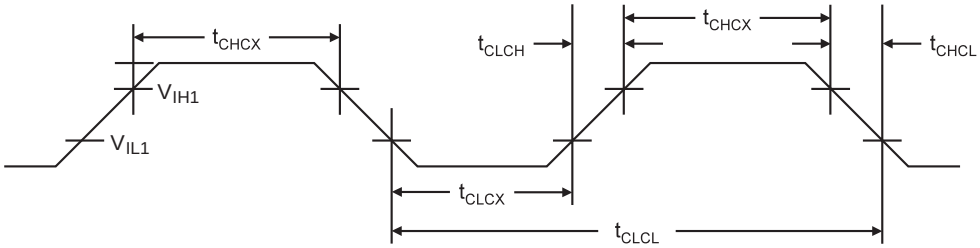
T_A = -40°C to 85°C, V_{CC} = 1.8V to 5.5V (unless otherwise noted) (Continued)

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
I_{CC}	Power Supply Current (All bits set in the "Power Reduction Register" on page 34)	Active 1MHz, V_{CC} = 2V			0.44	mA
		Active 4MHz, V_{CC} = 3V			2.5	mA
		Active 8MHz, V_{CC} = 5V			9.5	mA
		Idle 1MHz, V_{CC} = 2V			0.2	mA
		Idle 4MHz, V_{CC} = 3V			0.8	mA
		Idle 8MHz, V_{CC} = 5V			3.3	mA
	Power-down mode	WDT enabled, V_{CC} = 3V		<8	10	µA
		WDT disabled, V_{CC} = 3V		<1	2	µA
V_{ACIO}	Analog Comparator Input Offset Voltage	V_{CC} = 5V V_{in} = V_{CC}/2		<10	40	mV
I_{ACLK}	Analog Comparator Input Leakage Current	V_{CC} = 5V V_{in} = V_{CC}/2	-50		50	nA
t_{ACPD}	Analog Comparator Propagation Delay	V_{CC} = 2.7V V_{CC} = 4.0V		750 500		ns

Note:
1. "Max" means the highest value where the pin is guaranteed to be read as low
2. "Min" means the lowest value where the pin is guaranteed to be read as high
3. Although each I/O port can sink more than the test conditions (20 mA at V_{CC} = 5V, 10 mA at V_{CC} = 3V for Port B and 10 mA at V_{CC} = 5V, 5 mA at V_{CC} = 3V for all other ports) under steady state conditions (non-transient), the following must be observed:
   TQFP and QFN/MLF Package:
   1] The sum of all IOL, for all ports, should not exceed 400 mA.
   2] The sum of all IOL, for ports A0 - A7, C4 - C7, G2 should not exceed 100 mA.
   3] The sum of all IOL, for ports B0 - B7, E0 - E7, G3 - G5 should not exceed 100 mA.
   4] The sum of all IOL, for ports D0 - D7, C0 - C3, G0 - G1 should not exceed 100 mA.
   5] The sum of all IOL, for ports F0 - F7, should not exceed 100 mA.
   If IOL exceeds the test condition, VOL may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test condition.
4. Although each I/O port can source more than the test conditions (20 mA at V_{CC} = 5V, 10 mA at V_{CC} = 3V for Port B and 10mA at V_{CC} = 5V, 5 mA at V_{CC} = 3V for all other ports) under steady state conditions (non-transient), the following must be observed:
   TQFP and QFN/MLF Package:
   1] The sum of all IOH, for all ports, should not exceed 400 mA.
   2] The sum of all IOH, for ports A0 - A7, C4 - C7, G2 should not exceed 100 mA.
   3] The sum of all IOH, for ports B0 - B7, E0 - E7, G3 - G5 should not exceed 100 mA.
   4] The sum of all IOH, for ports D0 - D7, C0 - C3, G0 - G1 should not exceed 100 mA.
   5] The sum of all IOH, for ports F0 - F7, should not exceed 100 mA.
   If IOH exceeds the test condition, VOH may exceed the related specification. Pins are not guaranteed to source current greater than the listed test condition.

External Clock Drive Waveforms

Figure 135. External Clock Drive Waveforms



External Clock Drive

Table 132. External Clock Drive

Symbol	Parameter	V_{CC}=1.8-5.5V		V_{CC}=2.7-5.5V		V_{CC}=4.5-5.5V		Units
		Min.	Max.	Min.	Max.	Min.	Max.	
$1/t_{CLCL}$	Oscillator Frequency	0	1	0	8	0	16	MHz
t_{CLCL}	Clock Period	1000		125		62.5		ns
t_{CHCX}	High Time	400		50		25		ns
t_{CLCX}	Low Time	400		50		25		ns
t_{CLCH}	Rise Time		2.0		1.6		0.5	µs
t_{CHCL}	Fall Time		2.0		1.6		0.5	µs
Δt_{CLCL}	Change in period from one clock cycle to the next		2		2		2	%

Maximum Speed vs. V_{CC}

Maximum frequency is depending on $V_{CC.}$ As shown in Figure 136 and Figure 137, the Maximum Frequency vs. V_{CC} curve is linear between 1.8V < V_{CC} < 4.5V. To calculate the maximum frequency at a given voltage in this interval, use this equation:

$$Frequency = a \bullet (V - Vx) + Fy$$

To calculate required voltage for a given frequency, use this equation::

$$Voltage = b \bullet (F - Fy) + Vx$$

Table 133. Constants used to calculate maximum speed vs. V_{CC}

Voltage and Frequency range	*a*	*b*	*Vx*	*Fy*
2.7 < V_{CC} < 4.5 or 8 < Frq < 16	8/1.8	1.8/8	2.7	8
1.8 < V_{CC} < 2.7 or 4 < Frq < 8			1.8	4

At 3 Volt, this gives: $Frequency = \frac{8}{1.8} \bullet (3 - 2.7) + 8 = 9.33$

Thus, when V_{CC} = 3V, maximum frequency will be 9.33 MHz.

At 6 MHz this gives: $Voltage = \frac{1.8}{8} \bullet (6 - 4) + 1.8 = 2.25$

Thus, a maximum frequency of 6 MHz requires V_{CC} = 2.25V.