



#### Welcome to E-XFL.COM

#### What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "<u>Embedded -</u> <u>Microcontrollers</u>"

#### Details

| Product Status             | Active  |
|----------------------------|---|
| Core Processor             | AVR   |
| Core Size                  | 8-Bit   |
| Speed                      | 8MHz  |
| Connectivity               | I <sup>2</sup> C, SPI, UART/USART                                       |
| Peripherals                | Brown-out Detect/Reset, POR, PWM, WDT                                   |
| Number of I/O              | 32  |
| Program Memory Size        | 16KB (8K x 16)  |
| Program Memory Type        | FLASH   |
| EEPROM Size                | 512 x 8   |
| RAM Size                   | 1K x 8  |
| Voltage - Supply (Vcc/Vdd) | 2.7V ~ 5.5V   |
| Data Converters            | A/D 8x10b   |
| Oscillator Type            | Internal  |
| Operating Temperature      | -40°C ~ 85°C (TA)   |
| Mounting Type              | Surface Mount   |
| Package / Case             | 44-TQFP   |
| Supplier Device Package    | 44-TQFP (10x10)   |
| Purchase URL               | https://www.e-xfl.com/product-detail/microchip-technology/atmega16l-8au |

Email: info@E-XFL.COM

Address: Room A, 16/F, Full Win Commercial Centre, 573 Nathan Road, Mongkok, Hong Kong

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

**Data Retention** Reliability Qualification results show that the projected data retention failure rate is much less than 1 PPM over 20 years at 85°C or 100 years at 25°C.



JTAG Interface and On-chip Debug System If the On-chip debug system is enabled by the OCDEN Fuse and the chip enter Power down or Power save sleep mode, the main clock source remains enabled. In these sleep modes, this will contribute significantly to the total current consumption. There are three alternative ways to avoid this:

- Disable OCDEN Fuse.
- Disable JTAGEN Fuse.
- Write one to the JTD bit in MCUCSR.

The TDO pin is left floating when the JTAG interface is enabled while the JTAG TAP controller is not shifting data. If the hardware connected to the TDO pin does not pull up the logic level, power consumption will increase. Note that the TDI pin for the next device in the scan chain contains a pull-up that avoids this problem. Writing the JTD bit in the MCUCSR register to one or leaving the JTAG fuse unprogrammed disables the JTAG interface.



#### Alternate Functions of Port C

f The Port C pins with alternate functions are shown in Table 28. If the JTAG interface is enabled, the pull-up resistors on pins PC5(TDI), PC3(TMS) and PC2(TCK) will be activated even if a reset occurs.

| Table 28. | Port C Pins | Alternate | <b>Functions</b> |
|-----------|-------------|-----------|------------------|
|-----------|-------------|-----------|------------------|

| Port Pin | Alternate Function                               |
|----------|--|
| PC7      | TOSC2 (Timer Oscillator Pin 2)                   |
| PC6      | TOSC1 (Timer Oscillator Pin 1)                   |
| PC5      | TDI (JTAG Test Data In)                          |
| PC4      | TDO (JTAG Test Data Out)                         |
| PC3      | TMS (JTAG Test Mode Select)                      |
| PC2      | TCK (JTAG Test Clock)                            |
| PC1      | SDA (Two-wire Serial Bus Data Input/Output Line) |
| PC0      | SCL (Two-wire Serial Bus Clock Line)             |

The alternate pin configuration is as follows:

#### • TOSC2 - Port C, Bit 7

TOSC2, Timer Oscillator pin 2: When the AS2 bit in ASSR is set (one) to enable asynchronous clocking of Timer/Counter2, pin PC7 is disconnected from the port, and becomes the inverting output of the Oscillator amplifier. In this mode, a Crystal Oscillator is connected to this pin, and the pin can not be used as an I/O pin.

### • TOSC1 - Port C, Bit 6

TOSC1, Timer Oscillator pin 1: When the AS2 bit in ASSR is set (one) to enable asynchronous clocking of Timer/Counter2, pin PC6 is disconnected from the port, and becomes the input of the inverting Oscillator amplifier. In this mode, a Crystal Oscillator is connected to this pin, and the pin can not be used as an I/O pin.

#### • TDI - Port C, Bit 5

TDI, JTAG Test Data In: Serial input data to be shifted in to the Instruction Register or Data Register (scan chains). When the JTAG interface is enabled, this pin can not be used as an I/O pin.

#### • TDO - Port C, Bit 4

TDO, JTAG Test Data Out: Serial output data from Instruction Register or Data Register. When the JTAG interface is enabled, this pin can not be used as an I/O pin.

The TD0 pin is tri-stated unless TAP states that shifts out data are entered.

#### • TMS – Port C, Bit 3

TMS, JTAG Test Mode Select: This pin is used for navigating through the TAP-controller state machine. When the JTAG interface is enabled, this pin can not be used as an I/O pin.

#### • TCK – Port C, Bit 2

TCK, JTAG Test Clock: JTAG operation is synchronous to TCK. When the JTAG interface is enabled, this pin can not be used as an I/O pin.



#### • SDA – Port C, Bit 1

SDA, Two-wire Serial Interface Data: When the TWEN bit in TWCR is set (one) to enable the Two-wire Serial Interface, pin PC1 is disconnected from the port and becomes the Serial Data I/O pin for the Two-wire Serial Interface. In this mode, there is a spike filter on the pin to suppress spikes shorter than 50 ns on the input signal, and the pin is driven by an open drain driver with slew-rate limitation. When this pin is used by the Two-wire Serial Interface, the pull-up can still be controlled by the PORTC1 bit.

#### • SCL - Port C, Bit 0

SCL, Two-wire Serial Interface Clock: When the TWEN bit in TWCR is set (one) to enable the Two-wire Serial Interface, pin PC0 is disconnected from the port and becomes the Serial Clock I/O pin for the Two-wire Serial Interface. In this mode, there is a spike filter on the pin to suppress spikes shorter than 50 ns on the input signal, and the pin is driven by an open drain driver with slew-rate limitation. When this pin is used by the Two-wire Serial Interface, the pull-up can still be controlled by the PORTC0 bit.

Table 29 and Table 30 relate the alternate functions of Port C to the overriding signals shown in Figure 26 on page 55.

| Signal<br>Name | PC7/TOSC2       | PC6/TOSC1      | PC5/TDI | PC4/TDO             |
|----------------|-----------------|----------------|---------|---------------------|
| PUOE           | AS2             | AS2            | JTAGEN  | JTAGEN              |
| PUOV           | 0               | 0              | 1       | 0                   |
| DDOE           | AS2             | AS2            | JTAGEN  | JTAGEN              |
| DDOV           | 0               | 0              | 0       | SHIFT_IR + SHIFT_DR |
| PVOE           | 0               | 0              | 0       | JTAGEN              |
| PVOV           | 0               | 0              | 0       | TDO                 |
| DIEOE          | AS2             | AS2            | JTAGEN  | JTAGEN              |
| DIEOV          | 0               | 0              | 0       | 0                   |
| DI             | _               | -              | _       | _                   |
| AIO            | T/C2 OSC OUTPUT | T/C2 OSC INPUT | TDI     | _                   |

Table 29. Overriding Signals for Alternate Functions in PC7..PC4



Unit" on page 73. for details. The compare match event will also set the Compare Flag (OCF0) which can be used to generate an output compare interrupt request.

**Definitions** Many register and bit references in this document are written in general form. A lower case "n" replaces the Timer/Counter number, in this case 0. However, when using the register or bit defines in a program, the precise form must be used, that is, TCNT0 for accessing Timer/Counter0 counter value and so on.

The definitions in Table 37 are also used extensively throughout the document. **Table 37.** Definitions

| BOTTOM | The counter reaches the BOTTOM when it becomes 0x00.   |
|--------|--|
| MAX    | The counter reaches its MAXimum when it becomes 0xFF (decimal 255).  |
| ТОР    | The counter reaches the TOP when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be the fixed value 0xFF (MAX) or the value stored in the OCR0 Register. The assignment is dependent on the mode of operation. |

**Timer/Counter Clock Sources** The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the clock select logic which is controlled by the clock select (CS02:0) bits located in the Timer/Counter Control Register (TCCR0). For details on clock sources and prescaler, see "Timer/Counter0 and Timer/Counter1 Prescalers" on page 87.

**Counter Unit** The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit. Figure 28 shows a block diagram of the counter and its surroundings.

Figure 28. Counter Unit Block Diagram



Signal description (internal signals):

| count | Increment or decrement TCNT0 by 1. |
|-------|------------------------------------|
|-------|------------------------------------|

direction Select between increment and decrement.

clear Clear TCNT0 (set all bits to zero).

**clk**<sub>Tn</sub> Timer/Counter clock, referred to as clk<sub>T0</sub> in the following.

**TOP** Signalize that TCNT0 has reached maximum value.

**BOTTOM** Signalize that TCNT0 has reached minimum value (zero).

Depending of the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock ( $clk_{T0}$ ).  $clk_{T0}$  can be generated from an external or internal clock source, selected by the Clock Select bits (CS02:0). When no clock source is selected (CS02:0 = 0) the timer is stopped. However, the TCNT0 value can be accessed by the CPU, regardless of



## Output Compare Units

The 16-bit comparator continuously compares TCNT1 with the *Output Compare Register* (OCR1x). If TCNT equals OCR1x the comparator signals a match. A match will set the *Output Compare Flag* (OCF1x) at the next timer clock cycle. If enabled (OCIE1x = 1), the Output Compare Flag generates an output compare interrupt. The OCF1x Flag is automatically cleared when the interrupt is executed. Alternatively the OCF1x Flag can be cleared by software by writing a logical one to its I/O bit location. The Waveform Generator uses the match signal to generate an output according to operating mode set by the *Waveform Generation mode* (WGM13:0) bits and *Compare Output mode* (COM1x1:0) bits. The TOP and BOTTOM signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation (See "Modes of Operation" on page 101.)

A special feature of output compare unit A allows it to define the Timer/Counter TOP value (that is, counter resolution). In addition to the counter resolution, the TOP value defines the period time for waveforms generated by the Waveform Generator.

Figure 43 shows a block diagram of the output compare unit. The small "n" in the register and bit names indicates the device number (n = 1 for Timer/Counter1), and the "x" indicates output compare unit (A/B). The elements of the block diagram that are not directly a part of the output compare unit are gray shaded.





The OCR1x Register is double buffered when using any of the twelve *Pulse Width Modulation* (PWM) modes. For the normal and *Clear Timer on Compare* (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCR1x Compare Register to either TOP or BOTTOM of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCR1x Register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCR1x Buffer Register, and if double buffering is disabled the CPU will access the OCR1x directly. The content of the OCR1x (Buffer or Compare) Register is only changed by a write operation (the Timer/Counter does not update this register automatically as the TCNT1 and ICR1 Register). Therefore OCR1x is not read via the High byte



will be the same, but TOP should be replaced by BOTTOM, TOP-1 by BOTTOM+1 and so on. The same renaming applies for modes that set the TOV1 Flag at BOTTOM.



Figure 51. Timer/Counter Timing Diagram, no Prescaling

Figure 52 shows the same timing data, but with the prescaler enabled.

Figure 52. Timer/Counter Timing Diagram, with Prescaler ( $f_{clk}$   $_{I/O}/8$ )





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

The FOC1A/FOC1B bits are always read as zero.

#### • Bit 1:0 - WGM11:0: Waveform Generation Mode

Combined with the WGM13:2 bits found in the TCCR1B Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of wave-form generation to be used, see Table 47. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare match (CTC) mode, and three types of Pulse Width Modulation (PWM) modes. (See "Modes of Operation" on page 101.)

| Mode | WGM13 | WGM12<br>(CTC1) | WGM11<br>(PWM11) | WGM10<br>(PWM10) | Timer/Counter Mode of Operation  | ТОР    | Update of OCR1X | TOV1 Flag Set<br>on |
|------|-------|-----------------|------------------|------------------|----------------------------------|--------|-----------------|---------------------|
| 0    | 0     | 0               | 0                | 0                | Normal                           | 0xFFFF | Immediate       | MAX                 |
| 1    | 0     | 0               | 0                | 1                | PWM, Phase Correct, 8-bit        | 0x00FF | TOP             | воттом              |
| 2    | 0     | 0               | 1                | 0                | PWM, Phase Correct, 9-bit        | 0x01FF | TOP             | воттом              |
| 3    | 0     | 0               | 1                | 1                | PWM, Phase Correct, 10-bit       | 0x03FF | TOP             | воттом              |
| 4    | 0     | 1               | 0                | 0                | СТС                              | OCR1A  | Immediate       | MAX                 |
| 5    | 0     | 1               | 0                | 1                | Fast PWM, 8-bit                  | 0x00FF | BOTTOM          | ТОР                 |
| 6    | 0     | 1               | 1                | 0                | Fast PWM, 9-bit                  | 0x01FF | BOTTOM          | ТОР                 |
| 7    | 0     | 1               | 1                | 1                | Fast PWM, 10-bit                 | 0x03FF | BOTTOM          | ТОР                 |
| 8    | 1     | 0               | 0                | 0                | PWM, Phase and Frequency Correct | ICR1   | BOTTOM          | воттом              |
| 9    | 1     | 0               | 0                | 1                | PWM, Phase and Frequency Correct | OCR1A  | BOTTOM          | воттом              |
| 10   | 1     | 0               | 1                | 0                | PWM, Phase Correct               | ICR1   | TOP             | воттом              |
| 11   | 1     | 0               | 1                | 1                | PWM, Phase Correct               | OCR1A  | TOP             | BOTTOM              |
| 12   | 1     | 1               | 0                | 0                | СТС                              | ICR1   | Immediate       | MAX                 |
| 13   | 1     | 1               | 0                | 1                | Reserved                         | -      | -               | _                   |
| 14   | 1     | 1               | 1                | 0                | Fast PWM                         | ICR1   | BOTTOM          | ТОР                 |
| 15   | 1     | 1               | 1                | 1                | Fast PWM                         | OCR1A  | BOTTOM          | ТОР                 |

**Table 47.** Waveform Generation Mode Bit Description<sup>(1)</sup>

Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.



Note: See "Alternate Functions of Port B" on page 58 for a detailed description of how to define the direction of the user defined SPI pins.

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



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

```
SPI MasterInit:
      ; Set MOSI and SCK output, all others input
     ldi r17,(1<<DD MOSI) | (1<<DD SCK)
     out DDR SPI,r17
      ; Enable SPI, Master, set clock rate fck/16
      ldi r17,(1<<SPE) | (1<<MSTR) | (1<<SPR0)
      out SPCR, r17
      ret
   SPI MasterTransmit:
      ; Start transmission of data (r16)
     out SPDR, r16
   Wait Transmit:
      ; Wait for transmission complete
      sbis SPSR, SPIF
     rjmp Wait Transmit
      ret
C Code Example<sup>(1)</sup>
   void SPI MasterInit(void)
    {
      /* Set MOSI and SCK output, all others input */
     DDR SPI = (1<<DD MOSI) | (1<<DD SCK);
     /* Enable SPI, Master, set clock rate fck/16 */
     SPCR = (1<<SPE) | (1<<MSTR) | (1<<SPR0);
   }
   void SPI_MasterTransmit(char cData)
    {
      /* Start transmission */
     SPDR = cData;
      /* Wait for transmission complete */
     while(!(SPSR & (1<<SPIF)))</pre>
        ;
    }
```

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



all single conversions, and the first free running conversion) when  $CK_{ADC2}$  is low will take the same amount of time as a single ended conversion (13 ADC clock cycles from the next prescaled clock cycle). A conversion initiated by the user when  $CK_{ADC2}$  is high will take 14 ADC clock cycles due to the synchronization mechanism. In Free Running mode, a new conversion is initiated immediately after the previous conversion completes, and since  $CK_{ADC2}$  is high at this time, all automatically started (that is, all but the first) free running conversions will take 14 ADC clock cycles.

The gain stage is optimized for a bandwidth of 4 kHz at all gain settings. Higher frequencies may be subjected to non-linear amplification. An external low-pass filter should be used if the input signal contains higher frequency components than the gain stage bandwidth. Note that the ADC clock frequency is independent of the gain stage bandwidth limitation. For example, the ADC clock period may be 6 µs, allowing a channel to be sampled at 12 kSPS, regardless of the bandwidth of this channel.

If differential gain channels are used and conversions are started by Auto Triggering, the ADC must be switched off between conversions. When Auto Triggering is used, the ADC prescaler is reset before the conversion is started. Since the gain stage is dependent of a stable ADC clock prior to the conversion, this conversion will not be valid. By disabling and then re-enabling the ADC between each conversion (writing ADEN in ADCSRA to "0" then to "1"), only extended conversions are performed. The result from the extended conversions will be valid. See "Prescaling and Conversion Timing" on page 207 for timing details.

**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.

Special care should be taken when changing differential channels. Once a differential channel has been selected, the gain stage may take as much as 125  $\mu$ s to stabilize to the new value. Thus conversions should not be started within the first 125  $\mu$ s after selecting a new differential channel. Alternatively, conversion results obtained within this period should be discarded.

The same settling time should be observed for the first differential conversion after changing ADC reference (by changing the REFS1:0 bits in ADMUX).



Table 85. 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     | 1     | 0     | 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. If differential channels are used, the result is presented in two's complement form.

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 216.



| Signal<br>Name | Direction as Seen from the ADC | Description  | Recommended<br>Input when Not<br>in Use | Output Values when Recommended<br>Inputs are used, and CPU is not<br>Using the ADC |
|----------------|--------------------------------|--|---|--|
| 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 gain stages.   | 1                                       | 1  |
| PRECH          | Input                          | Precharge output latch of comparator. (Active low)   | 1                                       | 1  |
| SCTEST         | Input                          | Switch-cap TEST enable. Output<br>from x10 gain stage send out to<br>Port Pin having ADC_4                             | 0                                       | 0  |
| ST             | Input                          | Output of gain stages will settle<br>faster if this signal is high first two<br>ACLK periods after AMPEN goes<br>high. | 0                                       | 0  |
| VCCREN         | Input                          | Selects Vcc as the ACC reference voltage.  | 0                                       | 0  |

#### **Table 92.** Boundary-scan Signals for the ADC (Continued)

Note: Incorrect setting of the switches in Figure 123 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 123. 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 92 should be used. The user is recommended **not** to use the Differential Gain stages during scan. Switch-cap based gain stages require fast operation and accurate timing which is difficult to obtain when used in a scan chain. Details concerning operations of the differential gain stage is therefore not provided.

The AVR ADC is based on the analog circuitry shown in Figure 123 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 200 ns after enabling the ADC before





Figure 125. Memory Sections<sup>(1)</sup>

Note: 1. The parameters in the figure above are given in Table 100 on page 257.

Boot Loader Lock 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 96 and Table 97 for further details. The Boot 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 3) does not control reading nor writing by LPM/SPM, if it is attempted.



Bits



**Figure 126.** Addressing the Flash during SPM<sup>(1)</sup>

Notes: 1. The different variables used in Figure 126 are listed in Table 102 on page 258.
2. PCPAGE and PCWORD are listed in Table 107 on page 262.

# Self-Programming the Flash

**ning** 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 256 for an assembly code example.





**Figure 129.** Programming the Flash Waveforms<sup>(1)</sup>



Programming the<br/>EEPROMThe EEPROM is organized in pages, see Table 108 on page 262. When programming the<br/>EEPROM, the program data is latched into a page buffer. This allows one page of data to be<br/>programmed simultaneously. The programming algorithm for the EEPROM data memory is as<br/>follows (refer to "Programming the Flash" on page 266 for details on Command, Address and<br/>Data loading):

- 1. A: Load Command "0001 0001".
- 2. G: Load Address High Byte (\$00 \$FF)
- 3. B: Load Address Low Byte (\$00 \$FF)
- 4. C: Load Data (\$00 \$FF)
- 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 BS1 to "0".
- 2. Give WR a negative pulse. This starts programming of the EEPROM page. RDY/BSY goes low.
- 3. Wait until to RDY/BSY goes high before programming the next page. (See Figure 130 for signal waveforms)



Table 117. JTAG Programming Instruction 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_ <b>bbbbbbb</b>                 | xxxxxxx_xxxxxxx          |                |
|  | 0110010_0000000                         | XXXXXXX_XXXXXXXX         |                |
|  | 0110011_00000000                        | XXXXXXX_00000000         |                |
| 6a. Enter Fuse Write                   | 0100011_01000000                        | xxxxxxx_xxxxxxx          |                |
| 6b. Load Data Low Byte <sup>(6)</sup>  | 0010011_iiiiiiiiiiiiiiiiiiiiiiiiiiiiiii | XXXXXXX_XXXXXXX          | (3)            |
| 6c. Write Fuse High byte               | 0110111_00000000                        | xxxxxxx_xxxxxxx          | (1)            |
|  | 0110101_00000000                        | XXXXXXX_XXXXXXX          |                |
|  | 0110111_00000000                        | XXXXXXX_XXXXXXXX         |                |
|  | 0110111_00000000                        | XXXXXXX_XXXXXXX          |                |
| 6d. Poll for Fuse Write complete       | 0110111_00000000                        | xxxxx <b>o</b> x_xxxxxxx | (2)            |
| 6e. Load Data Low Byte <sup>(7)</sup>  | 0010011_iiiiiiiiiiiiiiiiiiiiiiiiiiiiiii | XXXXXXX_XXXXXXX          | (3)            |
| 6f. Write Fuse Low byte                | 0110011_00000000                        | xxxxxxx_xxxxxxx          | (1)            |
|  | 0110001_0000000                         | XXXXXXX_XXXXXXXX         |                |
|  | 0110011_00000000                        | XXXXXXX_XXXXXXXX         |                |
|  | 0110011_00000000                        | XXXXXXX_XXXXXXX          |                |
| 6g. Poll for Fuse Write complete       | 0110011_00000000                        | xxxxx <b>o</b> x_xxxxxxx | (2)            |
| 7a. Enter Lock Bit Write               | 0100011_00100000                        | xxxxxxx_xxxxxxx          |                |
| 7b. Load Data Byte <sup>(8)</sup>      | 0010011_11 <b>iiiii</b>                 | xxxxxxx_xxxxxxx          | (4)            |
| 7c. Write Lock Bits                    | 0110011_00000000                        | XXXXXXX_XXXXXXX          | (1)            |
|  | 0110001_0000000                         | XXXXXXX_XXXXXXXX         |                |
|  | 0110011_00000000                        | XXXXXXX_XXXXXXXX         |                |
|  | 0110011_00000000                        | XXXXXXXX_XXXXXXXX        |                |
| 7d. Poll for Lock Bit Write complete   | 0110011_00000000                        | xxxxx <b>o</b> x_xxxxxxx | (2)            |
| 8a. Enter Fuse/Lock Bit Read           | 0100011_00000100                        | xxxxxxx_xxxxxxx          |                |
| 8b. Read Fuse High Byte <sup>(6)</sup> | 0111110_00000000                        | XXXXXXX_XXXXXXXX         |                |
|  | 0111111_00000000                        | xxxxxxx_00000000         |                |
| 8c. Read Fuse Low Byte <sup>(7)</sup>  | 0110010_0000000                         | xxxxxxx_xxxxxxx          |                |
|  | 0110011_00000000                        | xxxxxxx_00000000         |                |
| 8d. Read Lock Bits <sup>(8)</sup>      | 0110110_0000000                         | xxxxxxx_xxxxxxx          | (5)            |
|  | 0110111_00000000                        | xxxxxxx_xx <b>000000</b> |                |
| 8e. Read Fuses and Lock Bits           | 0111110_00000000                        | xxxxxxx_xxxxxxx          | (5)            |
|  | 0110010_0000000                         | xxxxxxx_ <b>00000000</b> | Fuse High Byte |
|  | 0110110_0000000                         | xxxxxxx_00000000         | Fuse Low Byte  |
|  | 0110111_00000000                        | XXXXXXX_00000000         | Lock bits      |
| 9a. Enter Signature Byte Read          | 0100011_00001000                        | XXXXXXX_XXXXXXX          |                |
| 9b. Load Address Byte                  | 0000011_ <b>bbbbbbb</b> b               | xxxxxxx_xxxxxxx          |                |
| 9c. Read Signature Byte                | 0110010_00000000                        | xxxxxxx_xxxxxxx          |                |
|  | 0110011_00000000                        | xxxxxxx_00000000         |                |
| 10a. Enter Calibration Byte Read       | 0100011_00001000                        | xxxxxxx_xxxxxxx          |                |



Idle Supply Current Figure 156. Idle Supply Current vs. Frequency (0.1 MHz - 1.0 MHz)



Figure 157. Idle Supply Current vs. Frequency (1 MHz - 20 MHz)









Figure 171. Standby Supply Current vs.  $V_{CC}$  (4 MHz Xtal, Watchdog Timer Disabled)







Prescaling and Conversion Timing 207 Changing Channel or Reference Selection 210 ADC Noise Canceler 211 ADC Conversion Result 216

### JTAG Interface and On-chip Debug System 222

Features 222 Overview 222 Test Access Port – TAP 222 TAP Controller 224 Using the Boundary-scan Chain 225 Using the On-chip Debug System 225 On-chip Debug Specific JTAG Instructions 226 On-chip Debug Related Register in I/O Memory 227 Using the JTAG Programming Capabilities 227 Bibliography 227

### IEEE 1149.1 (JTAG) Boundary-scan 228

Features 228 System Overview 228 Data Registers 228 Boundary-scan Specific JTAG Instructions 230 Boundary-scan Chain 232 ATmega16 Boundary-scan Order 241 Boundary-scan Description Language Files 245

### Boot Loader Support – Read-While-Write Self-Programming 246

Features 246 Application and Boot Loader Flash Sections 246 Read-While-Write and no Read-While-Write Flash Sections 246 Boot Loader Lock Bits 248 Entering the Boot Loader Program 249 Addressing the Flash during Self-Programming 251 Self-Programming the Flash 252

### Memory Programming 259

Program And Data Memory Lock Bits 259 Fuse Bits 260 Signature Bytes 261 Calibration Byte 261 Page Size 262 Parallel Programming Parameters, Pin Mapping, and Commands 262 Parallel Programming 265 Serial Downloading 273 Programming via the JTAG Interface 278

## iv ATmega16(L)