Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

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

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Obsolete
Core Processor	AVR
Core Size	8-Bit
Speed	16MHz
Connectivity	I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	23
Program Memory Size	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 ~ 105°C (TA)
Mounting Type	Surface Mount
Package / Case	32-TQFP
Supplier Device Package	32-TQFP (7x7)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atmega168-15at1

- **Bit 2 – EEMPE: EEPROM Master Write Enable**

The EEMPE bit determines whether setting EEPE to one causes the EEPROM to be written. When EEMPE is set, setting EEPE within four clock cycles will write data to the EEPROM at the selected address If EEMPE is zero, setting EEPE will have no effect. When EEMPE has been written to one by software, hardware clears the bit to zero after four clock cycles. See the description of the EEPE bit for an EEPROM write procedure.

- **Bit 1 – EEPE: EEPROM Write Enable**

The EEPROM write enable signal EEPE is the write strobe to the EEPROM. When address and data are correctly set up, the EEPE bit must be written to one to write the value into the EEPROM. The EEMPE bit must be written to one before a logical one is written to EEPE, otherwise no EEPROM write takes place. The following procedure should be followed when writing the EEPROM (the order of steps 3 and 4 is not essential):

1. Wait until EEPE becomes zero.
2. Wait until SELFPRGEN in SPMCSR becomes zero.
3. Write new EEPROM address to EEAR (optional).
4. Write new EEPROM data to EEDR (optional).
5. Write a logical one to the EEMPE bit while writing a zero to EEPE in EECR.
6. Within four clock cycles after setting EEMPE, write a logical one to EEPE.

The EEPROM can not be programmed during a CPU write to the flash memory. The software must check that the flash programming is completed before initiating a new EEPROM write. Step 2 is only relevant if the software contains a boot loader allowing the CPU to program the flash. If the flash is never being updated by the CPU, step 2 can be omitted. See Section 24. “Boot Loader Support – Read-While-Write Self-Programming, ATmega88 and ATmega168” on page 229 for details about boot programming.

Caution: An interrupt between step 5 and step 6 will make the write cycle fail, since the EEPROM master write enable will time-out. If an interrupt routine accessing the EEPROM is interrupting another EEPROM access, the EEAR or EEDR register will be modified, causing the interrupted EEPROM access to fail. It is recommended to have the global interrupt flag cleared during all the steps to avoid these problems.

When the write access time has elapsed, the EEPE bit is cleared by hardware. The user software can poll this bit and wait for a zero before writing the next byte. When EEPE has been set, the CPU is halted for two cycles before the next instruction is executed.

- **Bit 0 – EERE: EEPROM Read Enable**

The EEPROM read enable signal EERE is the read strobe to the EEPROM. When the correct address is set up in the EEAR register, the EERE bit must be written to a logic one to trigger the EEPROM read. The EEPROM read access takes one instruction, and the requested data is available immediately. When the EEPROM is read, the CPU is halted for four cycles before the next instruction is executed.

The user should poll the EEPE bit before starting the read operation. If a write operation is in progress, it is neither possible to read the EEPROM, nor to change the EEAR register.

The calibrated oscillator is used to time the EEPROM accesses. Table 5-2 lists the typical programming time for EEPROM access from the CPU.

Table 5-2. EEPROM Programming Time

Symbol	Number of Calibrated RC Oscillator Cycles	Typical Programming Time
EEPROM write (from CPU)	26,368	3.3ms

6.5 Low Frequency Crystal Oscillator

The device can utilize a 32.768kHz watch crystal as clock source by a dedicated low frequency crystal Oscillator. The crystal should be connected as shown in Figure 6-2 on page 25. When this oscillator is selected, start-up times are determined by the SUT fuses and CKSEL0 as shown in Table 6-7.

Table 6-7. Start-up Times for the Low Frequency Crystal Oscillator Clock Selection

Power Conditions	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	CKSEL0	SUT1..0
BOD enabled	1KCK	14CK[1]	0	00
Fast rising power	1KCK	14CK + 4.1ms[1]	0	01
Slowly rising power	1KCK	14CK + 65ms[1]	0	10
Reserved			0	11
BOD enabled	32KCK	14CK	1	00
Fast rising power	32KCK	14CK + 4.1ms	1	01
Slowly rising power	32KCK	14CK + 65ms	1	10
Reserved			1	11

Note: 1. These options should only be used if frequency stability at start-up is not important for the application.

6.6 Calibrated Internal RC Oscillator

The calibrated internal RC oscillator by default provides a 8.0MHz clock. The frequency is nominal value at 3V and 25°C. The device is shipped with the CKDIV8 fuse programmed. See Section 6.11 “System Clock Prescaler” on page 31 for more details. This clock may be selected as the system clock by programming the CKSEL fuses as shown in Table 6-8. If selected, it will operate with no external components. During reset, hardware loads the calibration byte into the OSCCAL register and thereby automatically calibrates the RC oscillator. At 3V and 25°C, this calibration gives a frequency of 8MHz ±1%. The tolerance of the internal RC oscillator remains better than ±10% within the whole automotive temperature and voltage ranges (2.7V to 5.5V, –40°C to +125°C). The oscillator can be calibrated to any frequency in the range 7.3 - 8.1MHz within ±1% accuracy, by changing the OSCCAL register. When this Oscillator is used as the chip clock, the Watchdog Oscillator will still be used for the watchdog timer and for the reset time-out. For more information on the pre-programmed calibration value, see Section 25.4 “Calibration Byte” on page 245.

Table 6-8. Internal Calibrated RC Oscillator Operating Modes[1][3]

Frequency Range[2] (MHz)	CKSEL3..0
7.3 - 8.1	0010

Notes: 1. The device is shipped with this option selected.
2. The frequency ranges are preliminary values. Actual values are TBD.
3. If 8MHz frequency exceeds the specification of the device (depends on V_{CC}), the CKDIV8 fuse can be programmed in order to divide the internal frequency by 8.

When this oscillator is selected, start-up times are determined by the SUT fuses as shown in Table 6-9.

Table 6-9. Start-up Times for the Internal Calibrated RC Oscillator Clock Selection

Power Conditions	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	SUT1..0
BOD enabled	6CK	14CK[1]	00
Fast rising power	6CK	14CK + 4.1ms	01
Slowly rising power	6CK	14CK + 65ms[2]	10
Reserved			11

Notes: 1. If the RSTDISBL fuse is programmed, this start-up time will be increased to 14CK + 4.1ms to ensure programming mode can be entered.
2. The device is shipped with this option selected.

14. 16-bit Timer/Counter1 with PWM

The 16-bit Timer/Counter unit allows accurate program execution timing (event management), wave generation, and signal timing measurement. The main features are:

- True 16-bit Design (i.e., allows 16-bit PWM)
- Two independent output compare units
- Double buffered output compare registers
- One input capture unit
- Input capture noise canceler
- Clear timer on compare match (auto reload)
- Glitch-free, phase correct pulse width modulator (PWM)
- Variable PWM period
- Frequency generator
- External event counter
- Four independent interrupt sources (TOV1, OCF1A, OCF1B, and ICF1)

14.1 Overview

Most register and bit references in this section are written in general form. A lower case “n” replaces the Timer/Counter number, and a lower case “x” replaces the output compare unit channel. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT1 for accessing Timer/Counter1 counter value and so on.

A simplified block diagram of the 16-bit Timer/Counter is shown in Figure 14-1 on page 95. For the actual placement of I/O pins, refer to Section 1-1 “Pinout ATmega48/88/168” on page 3. 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 Section 14.10 “16-bit Timer/Counter Register Description” on page 113.

The PRTIM1 bit in Section 7.7.1 “Power Reduction Register - PRR” on page 35 must be written to zero to enable Timer/Counter1 module.

15.8 8-bit Timer/Counter Register Description

15.8.1 Timer/Counter Control Register A – TCCR2A

Bit	7	6	5	4	3	2	1	0	
	COM2A**1**	COM2A**0**	COM2B**1**	COM2B**0**	–	–	**WGM21**	**WGM20**	TCCR2A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:6 – COM2A1:0: Compare Match Output A Mode**

These bits control the output compare pin (OC2A) behavior. If one or both of the COM2A1:0 bits are set, the OC2A output overrides the normal port functionality of the I/O pin it is connected to. However, note that the data direction register (DDR) bit corresponding to the OC2A pin must be set in order to enable the output driver.

When OC2A is connected to the pin, the function of the COM2A1:0 bits depends on the WGM22:0 bit setting. Table 15-2 shows the COM2A1:0 bit functionality when the WGM22:0 bits are set to a normal or CTC mode (non-PWM).

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

COM2A1	COM2A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC2A on compare match
1	0	Clear OC2A on compare match
1	1	Set OC2A on compare match

Table 15-3 shows the COM2A1:0 bit functionality when the WGM21:0 bits are set to fast PWM mode.

Table 15-3. Compare Output Mode, Fast PWM Mode(1)

COM2A1	COM2A0	Description
0	0	Normal port operation, OC2A disconnected.
0	1	WGM22 = 0: normal port operation, OC0A disconnected. WGM22 = 1: toggle OC2A on compare match.
1	0	Clear OC2A on compare match, set OC2A at TOP
1	1	Set OC2A on compare match, clear OC2A at TOP

Note: 1. A special case occurs when OCR2A equals TOP and COM2A1 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See Section 15.6.3 “Fast PWM Mode” on page 125 for more details.

Table 15-4 shows the COM2A1:0 bit functionality when the WGM22:0 bits are set to phase correct PWM mode.

Table 15-4. Compare Output Mode, Phase Correct PWM Mode(1)

COM2A1	COM2A0	Description
0	0	Normal port operation, OC2A disconnected.
0	1	WGM22 = 0: normal port operation, OC2A disconnected. WGM22 = 1: toggle OC2A on compare match.
1	0	Clear OC2A on compare match when up-counting. Set OC2A on compare match when down-counting.
1	1	Set OC2A on compare match when up-counting. Clear OC2A on compare match when down-counting.

Note: 1. A special case occurs when OCR2A equals TOP and COM2A1 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See Section 15.6.4 “Phase Correct PWM Mode” on page 126 for more details.

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

17.6.2 Receiving Frames with 9 Data Bits

If 9-bit characters are used (UCSZn=7) the ninth bit must be read from the RXB8n bit in UCSRnB before reading the low bits from the UDRn. This rule applies to the FEn, DORn and UPEn status flags as well. Read status from UCSRnA, then data from UDRn. Reading the UDRn I/O location will change the state of the receive buffer FIFO and consequently the TXB8n, FEn, DORn and UPEn 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   UCSRnA, RXCn
        rjmp   USART_Receive
        ; Get status and 9th bit, then data from buffer
        in     r18, UCSRnA
        in     r17, UCSRnB
        in     r16, UDRn
        ; If error, return -1
        andi   r18,(1<<FEn)|(1<<DORn)|(1<<UPEn)
        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 (!(UCSRnA & (1<<RXCn)))
        ;
        /* Get status and 9th bit, then data */
        /* from buffer */
        status = UCSRnA;
        resh = UCSRnB;
        resl = UDRn;
        /* If error, return -1 */
        if (status & (1<<FEn)|(1<<DORn)|(1<<UPEn))
                return -1;
        /* Filter the 9th bit, then return */
        resh = (resh >> 1) & 0x01;
        return ((resh << 8) | resl);
}
```

Note: 1. The example code assumes that the part specific header file is included. For I/O registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

The 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.

17.6.7 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 UDRn I/O location until the RXCn flag is cleared. The following code example shows how to flush the receive buffer.

Assembly Code Example[1]

```
USART_Flush:
        sbis   UCSRnA, RXCn
        ret
        in     r16, UDRn
        rjmp   USART_Flush
```

C Code Example[1]

```
void USART_Flush(void)
{
        unsigned char dummy;
        while (UCSRnA & (1<<RXCn)) dummy = UDRn;
}
```

Note: 1. The example code assumes that the part specific header file is included. For I/O registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

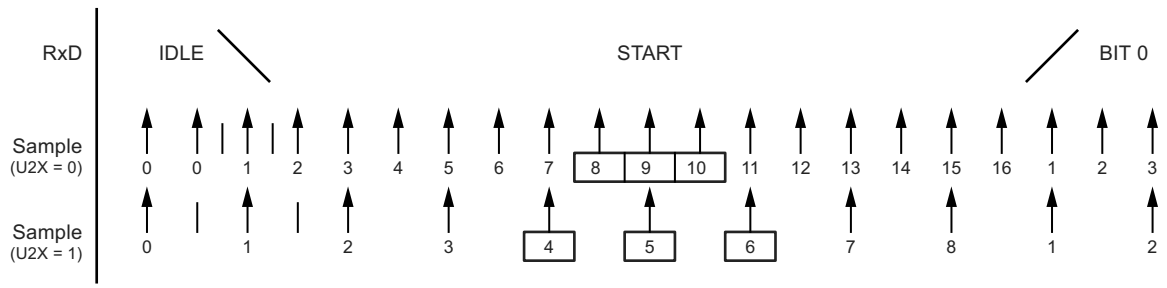
17.7 Asynchronous Data Reception

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 RxDn 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.

17.7.1 Asynchronous Clock Recovery

The clock recovery logic synchronizes internal clock to the incoming serial frames. Figure 17-5 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 (U2Xn = 1) of operation. Samples denoted zero are samples done when the RxDn line is idle (i.e., no communication activity).

Figure 17-5. Start Bit Sampling



When the clock recovery logic detects a high (idle) to low (start) transition on the RxDn 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.

20.2 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.

See Section 8.8 “Internal Voltage Reference” on page 43

- **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 20-1.

Table 20-1. 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.

20.3 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 20-2. If ACME is cleared or ADEN is set, AIN1 is applied to the negative input to the analog comparator.

Table 20-2. 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

20.3.1 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 7..2 – Res: Reserved Bits**

These bits are unused bits in the Atmel® ATmega48/88/168, and will always read as zero.

- **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.

21.5.2 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:

a. 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.

b. The AV_{CC} pin on the device should be connected to the digital V_{CC} supply voltage via an LC network as shown in Figure 21-9.

c. Use the ADC noise canceler function to reduce induced noise from the CPU.

d. If any ADC [3..0] port pins are used as digital outputs, it is essential that these do not switch while a conversion is in progress. However, using the 2-wire interface (ADC4 and ADC5) will only affect the conversion on ADC4 and ADC5 and not the other ADC channels.

Figure 21-9. ADC Power Connections

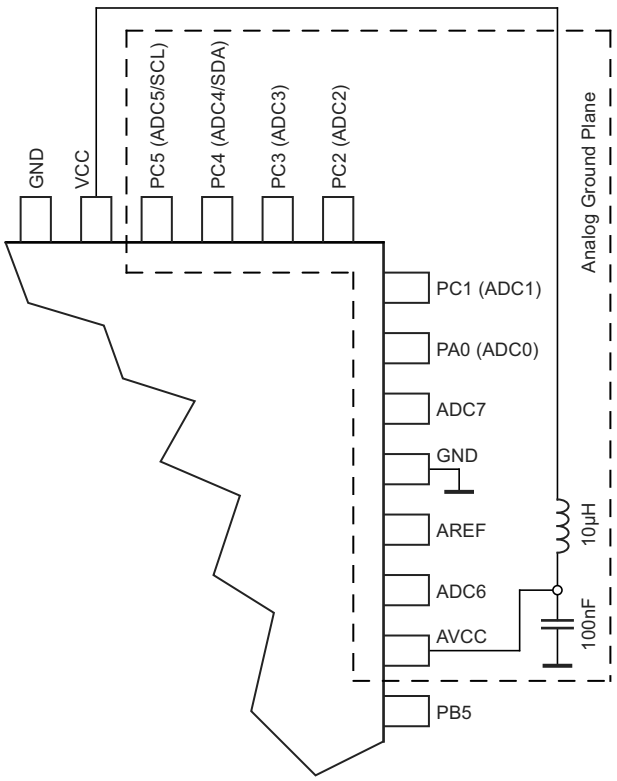


Table 21-3. Input Channel Selections

MUX3..0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	(reserved)
1001	(reserved)
1010	(reserved)
1011	(reserved)
1100	(reserved)
1101	(reserved)
1110	1.1V (V_{BG})
1111	0V (GND)

21.6.2 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 system clock frequency and the input clock to the ADC.

Table 21-4. 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

21.6.3 The ADC Data Register – ADCL and ADCH

21.6.3.1 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	

21.6.3.2 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 Section 21.6 “ADC Conversion Result” on page 217.

34. Errata ATmega88

The revision letter in this section refers to the revision of the ATmega88 device.

34.1 Rev. G

- Interrupts may be lost when writing the timer registers in the asynchronous timer

1. Interrupts may be lost when writing the timer registers in the asynchronous timer

If one of the timer registers which is synchronized to the asynchronous Timer2 clock is written in the cycle before an overflow interrupt occurs, the interrupt may be lost.

Problem Fix/Workaround

Always check that the Timer2 Timer/Counter register, TCNT2, does not have the value 0xFF before writing the Timer2 control register, TCCR2, or output compare register, OCR2.

34.2 Rev. E

- Interrupts may be lost when writing the timer registers in the asynchronous timer
- POR sensitivity with Vcc ramp up from a very low supply voltage

1. Interrupts may be lost when writing the timer registers in the asynchronous timer

If one of the timer registers which is synchronized to the asynchronous Timer2 clock is written in the cycle before an overflow interrupt occurs, the interrupt may be lost.

Problem Fix/Workaround

Always check that the Timer2 Timer/Counter register, TCNT2, does not have the value 0xFF before writing the Timer2 control register, TCCR2, or output compare register, OCR2.

2. POR sensitivity with Vcc ramp up from a very low supply voltage

If Vcc ramp up from a stable 150mV to 300mV plateau, the power on reset (POR) may not reset the device properly.

Problem Fix/Workaround

None.

Note: Please note from datasheet 7530F-AVR-09/07 we introduce a new errata numbering scheme (Errata Rev F of datasheet 7530E-AVR-03/07 is equivalent to Errata Rev E of datasheet 7530F-AVR-09/07)