ADC, DAC, AND SENSOR INTERFACING

OBJECTIVES

Upon completion of this chapter, you will be able to:

- **Discuss the ADC (analog-to-digital converter) section of the AVR chip**
- **Interface temperature sensors to the AVR**
- **Explain the process of data acquisition using ADC**
- **Describe factors to consider in selecting an ADC chip**
- **Program the AVR's ADC in C and Assembly**
- **Describe the basic operation of a DAC (digital-to-analog converter) chip**
- **Interface a DAC chip to the AVR**
- **Program DAC chips in AVR C and Assembly**
- **Explain the function of precision IC temperature sensors**
- **Describe signal conditioning and its role in data acquisition**

This chapter explores more real-world devices such as ADCs (analog-to-digital converters), DACs (digital-to-analog converters), and sensors. We will also explain how to interface the AVR to these devices. In Section 1, we describe analog-to-digital converter (ADC) chips. We will program the ADC portion of the AVR chip in Section 2. In Section 3, we show the interfacing of sensors and discuss the issue of signal conditioning. The characteristics of DAC chips are discussed in Section 4.

SECTION 1: ADC CHARACTERISTICS

This section will explore ADC generally. First, we describe some general aspects of the ADC itself, then focus on the functionality of some important pins in ADC.

ADC devices

Analog-to-digital converters are among the most widely used devices for data acquisition. Digital computers use binary (discrete) values, but in the physical world everything is analog (continuous). Temperature, pressure (wind or liquid), humidity, and velocity are a few examples of physical quantities that we deal with every day. A physical quantity is converted to electrical (voltage, current) signals using a device called a *transducer*. Transducers are also referred to as *sensors*. Sensors for temperature, velocity, pressure, light, and many other natural quantities produce an output that is voltage (or current). Therefore, we need an analog-to-digital converter to translate the analog signals to digital numbers so that the microcontroller can read and process them. See Figures 1 and 2.

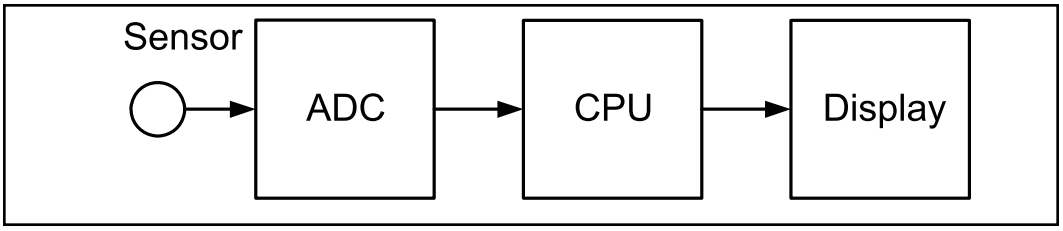


Figure 1. Microcontroller Connection to Sensor via ADC

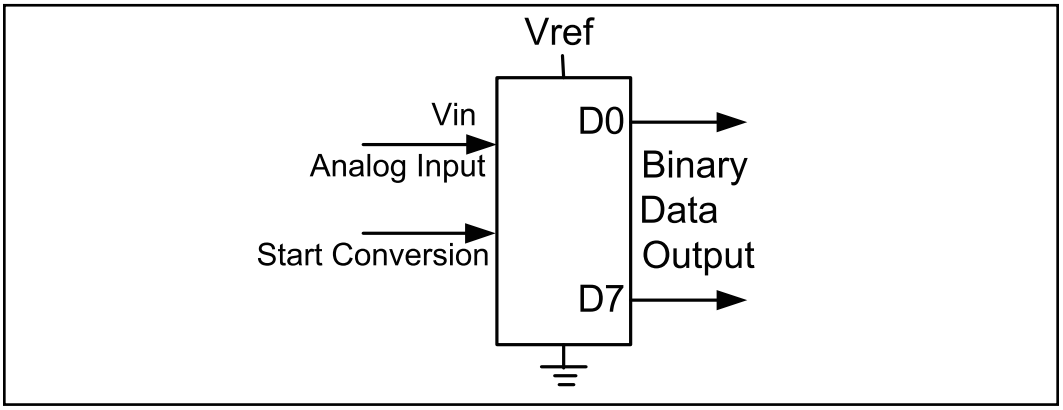


Figure 2. An 8-bit ADC Block Diagram

Table 1: Resolution versus Step Size for ADC (V_{ref} = 5 V)

n-bit	Number of steps	Step size (mV)
8	256	5/256 = 19.53
10	1024	5/1024 = 4.88
12	4096	5/4096 = 1.2
16	65,536	5/65,536 = 0.076

Notes: V_{CC} = 5 V
Step size (resolution) is the smallest change that can be discerned by an ADC.

Some of the major characteristics of the ADC

Resolution

The ADC has *n*-bit resolution, where *n* can be 8, 10, 12, 16, or even 24 bits. Higher-resolution ADCs provide a smaller step size, where *step size* is the smallest change that can be discerned by an ADC. Some widely used resolutions for ADCs are shown in Table 1. Although the resolution of an ADC chip is decided at the time of its design and cannot be changed, we can control the step size with the help of what is called V_{ref}. This is discussed below.

Conversion time

In addition to resolution, conversion time is another major factor in judging an ADC. *Conversion time* is defined as the time it takes the ADC to convert the analog input to a digital (binary) number. The conversion time is dictated by the clock source connected to the ADC in addition to the method used for data conversion and technology used in the fabrication of the ADC chip such as MOS or TTL technology.

V_{ref}

V_{ref} is an input voltage used for the reference voltage. The voltage connected to this pin, along with the resolution of the ADC chip, dictate the step size. For an 8-bit ADC, the step size is $V_{ref}/256$ because it is an 8-bit ADC, and 2 to the power of 8 gives us 256 steps. See Table 1. For example, if the analog input range needs to be 0 to 4 volts, V_{ref} is connected to 4 volts. That gives 4 V/256 = 15.62 mV for the step size of an 8-bit ADC. In another case, if we need a step size

Table 2: V_{ref} Relation to V_{in} Range for an 8-bit ADC

V_{ref} (V)	V_{in} Range (V)	Step Size (mV)
5.00	0 to 5	5/256 = 19.53
4.0	0 to 4	4/256 = 15.62
3.0	0 to 3	3/256 = 11.71
2.56	0 to 2.56	2.56/256 = 10
2.0	0 to 2	2/256 = 7.81
1.28	0 to 1.28	1.28/256 = 5
1	0 to 1	1/256 = 3.90

Step size is V_{ref} / 256

Table 3: V_{ref} Relation to V_{in} Range for an 10-bit ADC

V_{ref} (V)	V_{in} (V)	Step Size (mV)
5.00	0 to 5	5/1024 = 4.88
4.096	0 to 4.096	4.096/1024 = 4
3.0	0 to 3	3/1024 = 2.93
2.56	0 to 2.56	2.56/1024 = 2.5
2.048	0 to 2.048	2.048/1024 = 2
1.28	0 to 1.28	1/1024 = 1.25
1.024	0 to 1.024	1.024/1024 = 1

of 10 mV for an 8-bit ADC, then V_{ref} = 2.56 V, because 2.56 V/256 = 10 mV. For the 10-bit ADC, if the V_{ref} = 5V, then the step size is 4.88 mV as shown in Table 1. Tables 2 and 3 show the relationship between the V_{ref} and step size for the 8- and 10-bit ADCs, respectively. In some applications, we need the differential reference voltage where $V_{ref} = V_{ref}(+) - V_{ref}(-)$. Often the $V_{ref}(-)$ pin is connected to ground and the $V_{ref}(+)$ pin is used as the V_{ref}.

Digital data output

In an 8-bit ADC we have an 8-bit digital data output of D0–D7, while in the 10-bit ADC the data output is D0–D9. To calculate the output voltage, we use the following formula:

$$D_{out} = \frac{V_{in}}{step\ size}$$

where D_{out} = digital data output (in decimal), V_{in} = analog input voltage, and step size (resolution) is the smallest change, which is $V_{ref}/256$ for an 8-bit ADC. See Example 1. This data is brought out of the ADC chip either one bit at a time (serially), or in one chunk, using a parallel line of outputs. This is discussed next.

Example 1

For an 8-bit ADC, we have V_{ref} = 2.56 V. Calculate the D0–D7 output if the analog input is: (a) 1.7 V, and (b) 2.1 V.

Solution:

Because the step size is 2.56/256 = 10 mV, we have the following:
(a) D_{out} = 1.7 V/10 mV=170 in decimal, which gives us 10101010 in binary for D7–D0.
(b) D_{out}= 2.1 V/10 mV = 210 in decimal, which gives us 11010010 in binary for D7–D0.

Parallel versus serial ADC

The ADC chips are either parallel or serial. In parallel ADC, we have 8 or more pins dedicated to bringing out the binary data, but in serial ADC we have only one pin for data out. That means that inside the serial ADC, there is a parallel-in-serial-out shift register responsible for sending out the binary data one bit at a time. The D0–D7 data pins of the 8-bit ADC provide an 8-bit parallel data path between the ADC chip and the CPU. In the case of the 16-bit parallel ADC chip,

we need 16 pins for the data path. In order to save pins, many 12- and 16-bit ADCs use pins D0–D7 to send out the upper and lower bytes of the binary data. In recent years, for many applications where space is a critical issue, using such a large number of pins for data is not feasible. For this reason, serial devices such as the serial ADC are becoming widely used. While the serial ADCs use fewer pins and their smaller packages take much less space on the printed circuit board, more CPU time is needed to get the converted data from the ADC because the CPU must get data one bit at a time, instead of in one single read operation as with the parallel ADC. ADC848 is an example of a parallel ADC with 8 pins for the data output, while the MAX1112 is an example of a serial ADC with a single pin for D_{out}. Figures 3 and 4 show the block diagram for ADC848 and MAX1112.

Analog input channels

Many data acquisition applications need more than one ADC. For this reason, we see ADC chips with 2, 4, 8, or even 16 channels on a single chip. Multiplexing of analog inputs is widely used as shown in the ADC848 and MAX1112. In these chips, we have 8 channels of analog inputs, allowing us to monitor multiple quantities such as temperature, pressure, heat, and so on. AVR microcontroller chips come with up to 16 ADC channels.

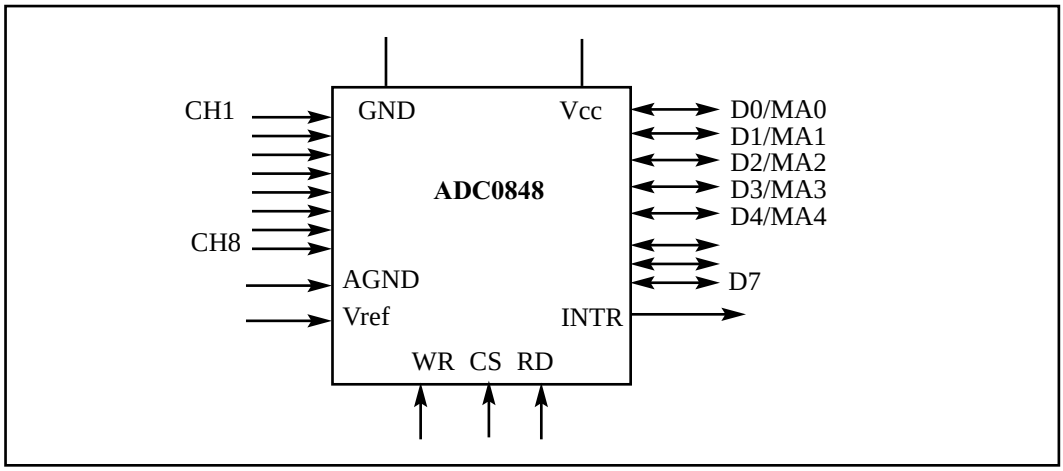


Figure 3. ADC0848 Parallel ADC Block Diagram

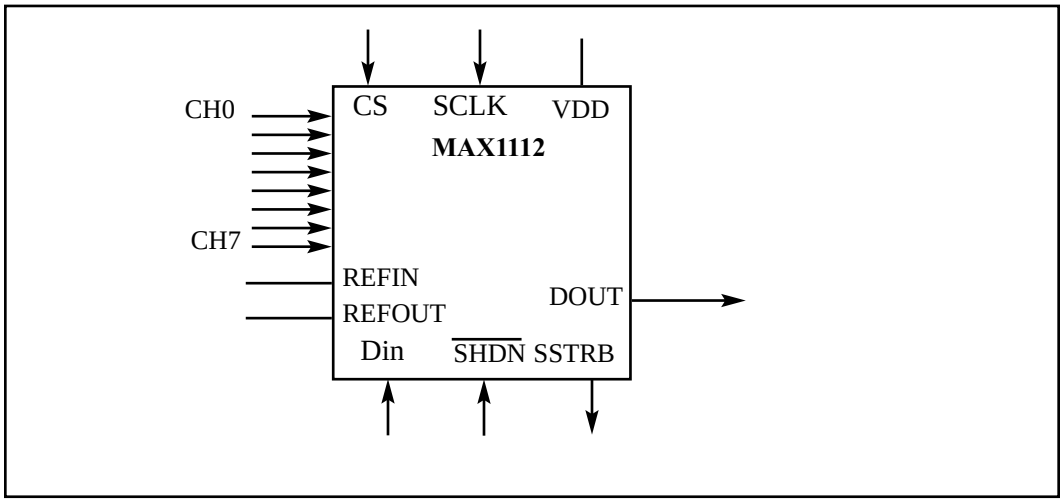


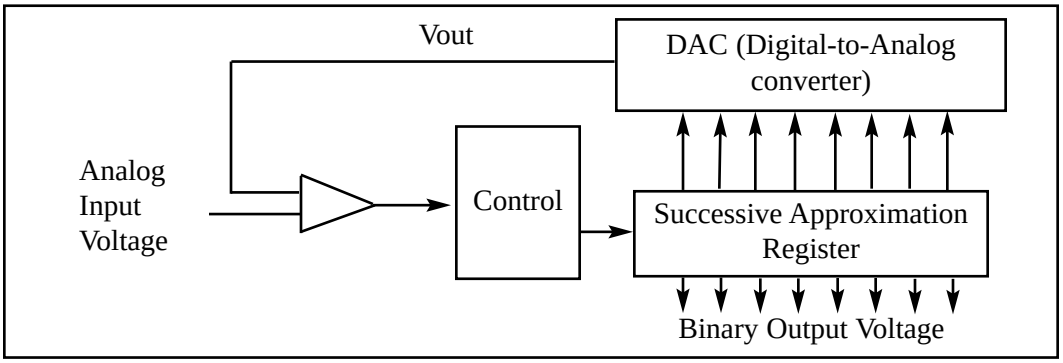
Figure 4. MAX1112 Serial ADC Block Diagram

Start conversion and end-of-conversion signals

The fact that we have multiple analog input channels and a single digital output register creats the need for start conversion (SC) and end-of-conversion (EOC) signals. When SC is activated, the ADC starts converting the analog input value of Vin to an *n*-bit digital number. The amount of time it takes to convert varies depending on the conversion method as was explained earlier. When the data conversion is complete, the end-of-conversion signal notifies the CPU that the converted data is ready to be picked up.

Successive Approximation ADC

Successive Approximation is a widely used method of converting an analog input to digital output. It has three main components: (a) successive approximation register (SAR), (b) comparator, and (c) control unit. See the figure below.



Assuming a step size of 10 mV, the 8-bit successive approximation ADC will go through the following steps to convert an input of 1 volt:

(1) It starts with binary 10000000. Since 128 × 10 mV = 1.28 V is greater than the 1 V input, bit 7 is cleared (dropped). (2) 01000000 gives us 64 × 10 mV = 640 mV and bit 6 is kept since it is smaller than the 1 V input. (3) 01100000 gives us 96 × 10 mV = 960 mV and bit 5 is kept since it is smaller than the 1 V input, (4) 01110000 gives us 112 × 10 mV = 1120 mv and bit 4 is dropped since it is greater than the 1 V input. (5) 01101000 gives us 108 × 10 mV = 1080 mV and bit 3 is dropped since it is greater than the 1 V input. (6) 01100100 gives us 100 × 10 mV = 1000 mV = 1 V and bit 2 is kept since it is equal to input. Even though the answer is found it does not stop. (7) 011000110 gives us 102 × 10 mV = 1020 mV and bit 1 is dropped since it is greater than the 1 V input. (8) 01100101 gives us 101 × 10 mV = 1010 mV and bit 0 is dropped since it is greater than the 1 V input.

Notice that the Successive Approximation method goes through all the steps even if the answer is found in one of the earlier steps. The advantage of the Successive Approximation method is that the conversion time is fixed since it has to go through all the steps.

Review Questions

1. Give two factors that affect the step size calculation.
2. The ADC0848 is a(n) _____-bit converter.
3. True or false. While the ADC0848 has 8 pins for D_{out}, the MAX1112 has only one D_{out} pin.
4. Find the step size for an 8-bit ADC, if Vref = 1.28 V.
5. For question 4, calculate the output if the analog input is: (a) 0.7 V, and (b) 1 V.

SECTION 2: ADC PROGRAMMING IN THE AVR

Because the ADC is widely used in data acquisition, in recent years an increasing number of microcontrollers have had an on-chip ADC peripheral, just like timers and USART. An on-chip ADC eliminates the need for an external ADC connection, which leaves more pins for other I/O activities. The vast majority of the AVR chips come with ADC. In this section we discuss the ADC feature of the ATmega32 and show how it is programmed in both Assembly and C.

ATmega32 ADC features

The ADC peripheral of the ATmega32 has the following characteristics:

(a) It is a 10-bit ADC.
(b) It has 8 analog input channels, 7 differential input channels, and 2 differential input channels with optional gain of 10x and 200x.
(c) The converted output binary data is held by two special function registers called ADCL (A/D Result Low) and ADCH (A/D Result High).
(d) Because the ADCH:ADCL registers give us 16 bits and the ADC data out is only 10 bits wide, 6 bits of the 16 are unused. We have the option of making either the upper 6 bits or the lower 6 bits unused.
(e) We have three options for V_{ref}. V_{ref} can be connected to AVCC (Analog V_{cc}), internal 2.56 V reference, or external AREF pin.
(f) The conversion time is dictated by the crystal frequency connected to the XTAL pins (Fosc) and ADPS0:2 bits.

AVR ADC hardware considerations

For digital logic signals a small variation in voltage level has no effect on the output. For example, 0.2 V is considered LOW, since in TTL logic, anything less than 0.5 V will be detected as LOW logic. That is not the case when we are dealing with analog voltage. See Example 2.

We can use many techniques to reduce the impact of ADC supply voltage and V_{ref} variation on the accuracy of ADC output. Next, we examine two of the most widely used techniques in the AVR.

Example 2

For an 10-bit ADC, we have V_{ref} = 2.56 V. Calculate the D0–D9 output if the analog input is: (a) 0.2 V, and (b) 0 V. How much is the variation between (a) and (b)?
Solution:

Because the step size is 2.56/1024 = 2.5 mV, we have the following:
(a) D_{out} = 0.2 V/2.5 mV = 80 in decimal, which gives us 1010000 in binary.

(b) D_{out} = 0 V/2.5 mV = 0 in decimal, which gives us 0 in binary.

The difference is 1010000, which is 7 bits!

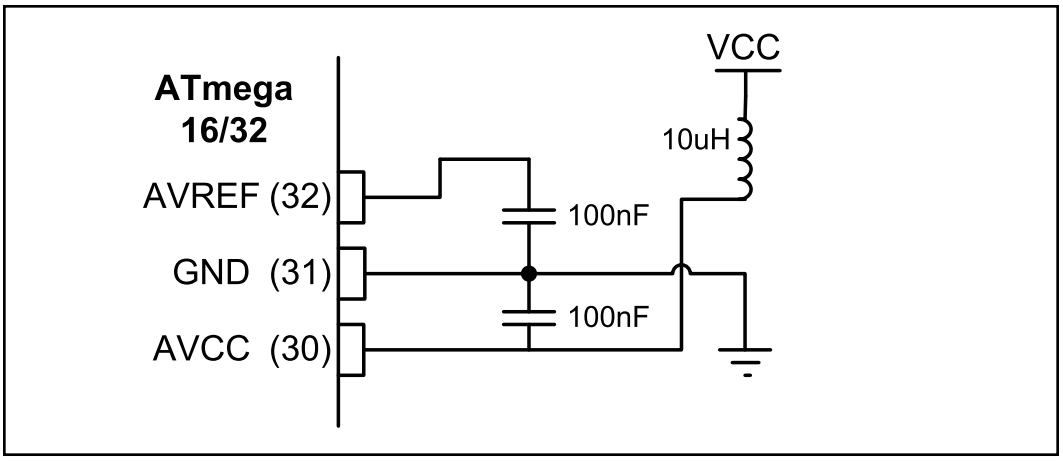


Figure 5. ADC Recommended Connection

Decoupling AVCC from VCC

The AVCC pin provides the supply for analog ADC circuitry. To get a better accuracy of AVR ADC we must provide a stable voltage source to the AVCC pin. Figure 5 shows how to use an inductor and a capacitor to achieve this.

Connecting a capacitor between V_{ref} and GND

By connecting a capacitor between the AVREF pin and GND you can make the V_{ref} voltage more stable and increase the precision of ADC. See Figure 5.

AVR programming in Assembly and C

In the AVR microcontroller five major registers are associated with the ADC that we deal with in this chapter. They are ADCH (high data), ADCL (low data), ADCSRA (ADC Control and Status Register), ADMUX (ADC multiplexer selection register), and SPIOR (Special Function I/O Register). We examine each of them in this section.

REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

REFS1:0 Bit 7:6 Reference Selection Bits
These bits select the reference voltage for the ADC.

ADLAR Bit 5 ADC Left Adjust Results
This bit dictates either the left bits or the right bits of the result registers ADCH:ADCL that are used to store the result. If we write a one to ADLAR, the result will be left adjusted; otherwise, the result is right adjusted.

MUX4:0 Bit 4:0 Analog Channel and Gain Selection Bits
The value of these bits selects the gain for the differential channels and also selects which combination of analog inputs are connected to the ADC.

Figure 6. ADMUX Register

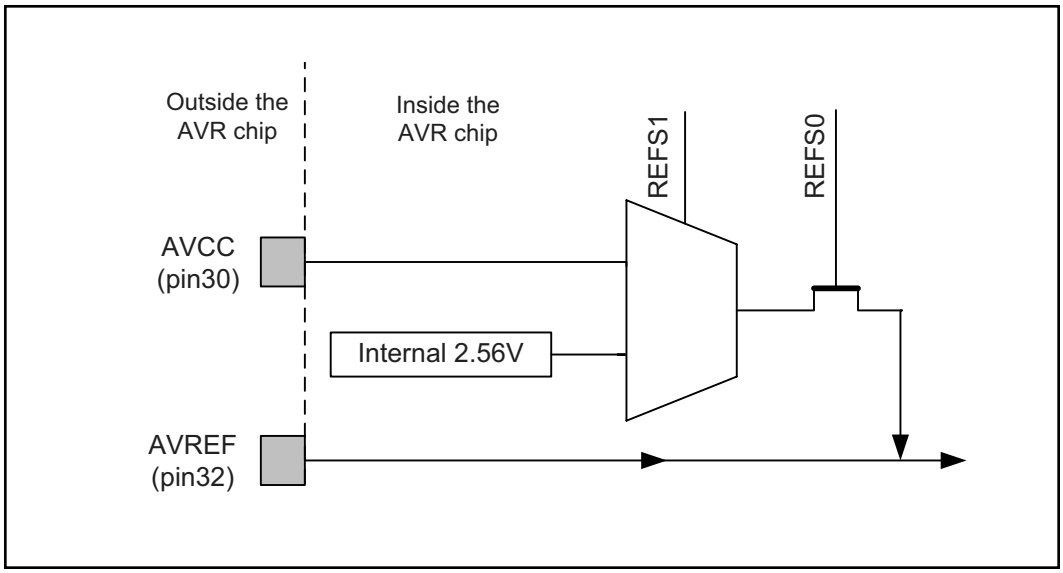


Figure 7. ADC Reference Source Selection

ADMUX register

Figure 6 shows the bits of ADMUX registers and their usage. In this section we will focus more on the function of these bits.

V_{ref} source

Figure 7 shows the block diagram of internal circuitry of V_{ref} selection. As you can see we have three options: (a) AREF pin, (b) AVCC pin, or (c) internal 2.56 V. Table 4 shows how the REFS1 and REFS0 bits of the ADMUX register can be used to select the V_{ref} source.

Table 4: V_{ref} Source Selection Table for AVR

REFS1	REFS0	V_{ref}	
0	0	AREF pin	Set externally
0	1	AVCC pin	Same as VCC
1	0	Reserved	----
1	1	Internal 2.56 V	Fixed regardless of VCC value

Notice that if you connect the VREF pin to an external fixed voltage you will not be able to use the other reference voltage options in the application, as they will be shorted with the external voltage.

Another important point to note is the fact that connecting a 100 nF external capacitor between the VREF pin and GND will increase the precision and stability of ADC, especially when you want to use internal 2.56 V. Refer to Figure 5 to see how to connect an external capacitor to the VREF pin of the ATmega32.

If you choose 2.56 V as the V_{ref}, the step size of ADC will be 2.56 / 1024 = 10/4 = 2.5 mV. Such a round step size will reduce the calculations in software.

ADC input channel source

Figure 8 shows the schematic of the internal circuitry of input channel selection. As you can see in the figure, either single-ended or the differential input can be selected to be converted to digital data. If you select single-ended input, you can choose the input channel among ADC0 to ACD7. In this case a single pin is used as the analog line, and GND of the AVR chip is used as common ground. Table 5 lists the values of MUX4–MUX0 bits for different single-ended inputs. As you see in Figure 8, if you choose differential input, you can also select the op-amp gain. You can choose the gain of the op-amp to be 1x, 10x,

Table 5: Single-ended Channels

MUX4...0	Single-ended Input
00000	ADC0
00001	ADC1
00010	ADC2
00011	ADC3
00100	ADC4
00101	ADC5
00110	ADC6
00111	ADC7

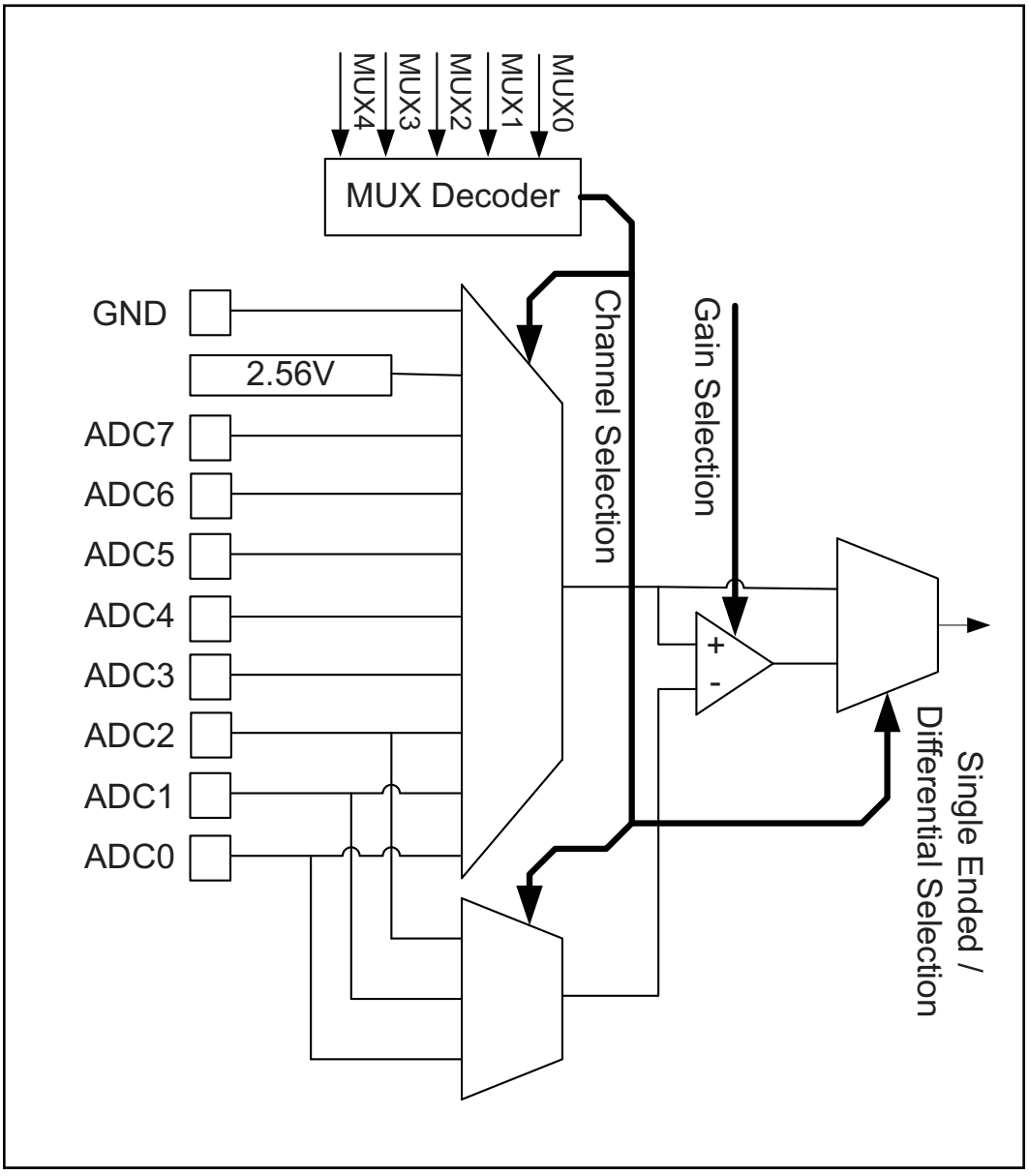


Figure 8. ADC Input Channel Selection

Table 6: V_{ref} Source Selection Table

MUX4...0	+ Differential Input	– Differential Input	Gain
01000 *	ADC0	ADC0	10x
01001	ADC1	ADC0	10x
01010 *	ADC0	ADC0	200x
01011	ADC1	ADC0	200x
01100 *	ADC2	ADC2	10x
01101	ADC3	ADC2	10x
01110 *	ADC2	ADC2	200x
01111	ADC3	ADC2	200x
10000	ADC0	ADC1	1x
10001 *	ADC1	ADC1	1x
10010	ADC2	ADC1	1x
10011	ADC3	ADC1	1x
10100	ADC4	ADC1	1x
10101	ADC5	ADC1	1x
10110	ADC6	ADC1	1x
10111	ADC7	ADC1	1x
11000	ADC0	ADC2	1x
11001	ADC1	ADC2	1x
11010 *	ADC2	ADC2	1x
11011	ADC3	ADC2	1x
11100	ADC4	ADC2	1x
11101	ADC5	ADC2	1x

*Note: The rows with * are not applicable.*

or 200x. You can select the positive input of the op-amp to be one of the pins ADC0 to ADC7, and the negative input of the op-amp can be any of ADC0, ADC1, or ADC2 pins. See Table 6.

ADLAR bit operation

The AVRs have a 10-bit ADC, which means that the result is 10 bits long and cannot be stored in a single byte. In AVR two 8-bit registers are dedicated to the ADC result, but only 10 of the 16 bits are used and 6 bits are unused. You can select the position of used bits in the bytes. If you set the ADLAR bit in ADMUX register, the result bits will be left-justified; otherwise, the result bits will be right-justified. See Figure 9. Notice that changing the ADLAR bit will affect the ADC data register immediately.

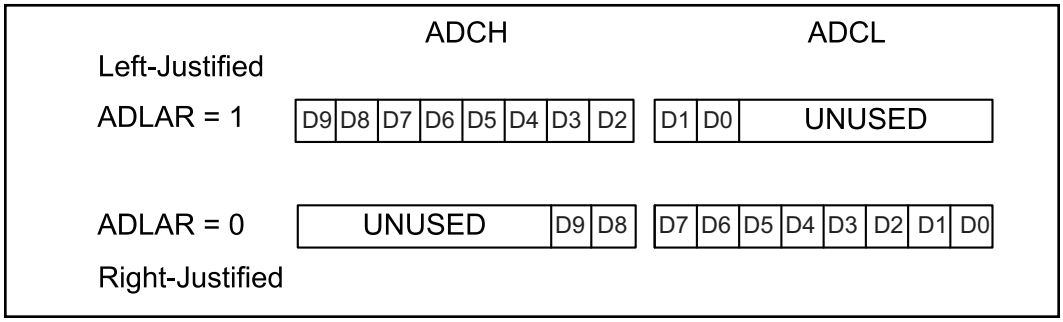


Figure 9. ADLAR Bit and ADCx Registers

ADCH: ADCL registers

After the A/D conversion is complete, the result sits in registers ADCL (A/D Result Low Byte) and ACDH (A/D Result High Byte). As we mentioned before, the ADLAR bit of the ADMUX is used for making it right-justified or left-justified because we need only 10 of the 16 bits.

ADCSRA register

The ADCSRA register is the status and control register of ADC. Bits of this register control or monitor the operation of the ADC. In Figure 10 you can see a description of each bit of the ADCSRA register. We will examine some of these bits in more detail.

ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADEN Bit 7 ADC Enable
This bit enables or disables the ADC. Setting this bit to one will enable the ADC, and clearing this bit to zero will disable it even while a conversion is in progress.

ADSC Bit 6 ADC Start Conversion
To start each conversion you have to set this bit to one.

ADATE Bit 5 ADC Auto Trigger Enable
Auto triggering of the ADC is enabled when you set this bit to one.

ADIF Bit 4 ADC Interrupt Flag
This bit is set when an ADC conversion completes and the data registers are updated.

ADIE Bit 3 ADC Interrupt Enable
Setting this bit to one enables the ADC conversion complete interrupt.

ADPS2:0 Bit 2:0 ADC Prescaler Select Bits
These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

Figure 10. ADCSRA (A/D Control and Status Register A)

ADC Start Conversion bit

As we stated before, an ADC has a Start Conversion input. The AVR chip has a special circuit to trigger start conversion. As you see in Figure 11, in addition to the ADCSC bit of ADCSRA there are other sources to trigger start of conversion. If you set the ADATE bit of ADCSRA to high, you can select auto trigger source by updating ADTS2:0 in the SFIOR register. If ADATE is cleared, the ADTS2:0 settings will have no effect. Notice that there are many considerations if you want to use auto trigger mode. We will not cover auto trigger mode in this text. If you want to use auto trigger mode we strongly recommend you to refer to the datasheet of the device that you want to use at www.atmel.com.

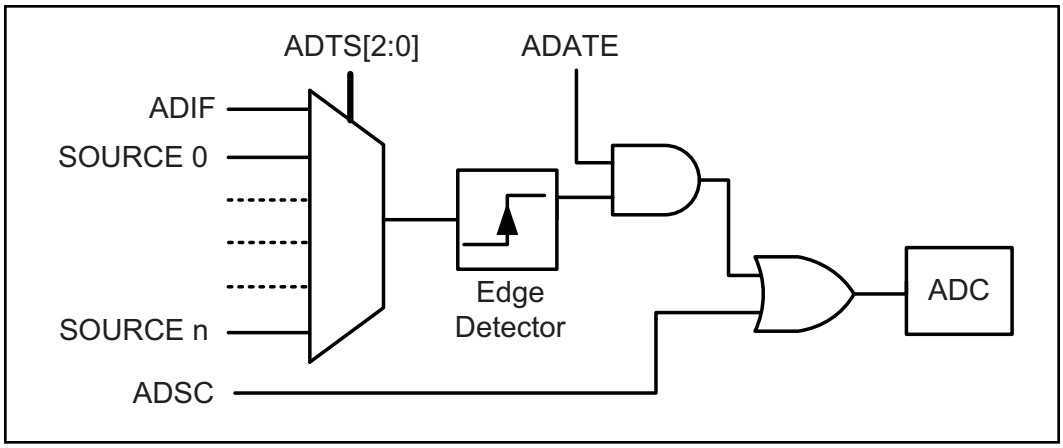
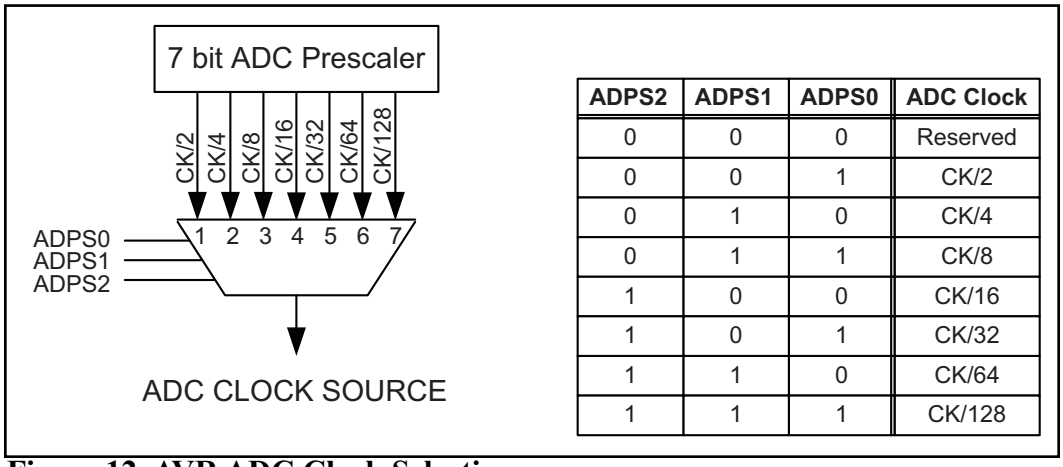


Figure 11. AVR ADC Trigger Source

A/D conversion time

As you see in Figure 12, by using the ADPS2:0 bits of the ADCSRA register we can set the A/D conversion time. To select the conversion time, we can select any of Fosc/2, Fosc/4, Fosc/8, Fosc/16, Fosc/32, Fosc/64, or Fosc/128 for ADC clock, where Fosc is the speed of the crystal frequency connected to the AVR chip. Notice that the multiplexer has 7 inputs since the option ADPS2:0 = 000 is reserved. For the AVR, the ADC requires an input clock frequency less than 200 kHz for the maximum accuracy. Look at Example 3 for clarification.



ADPS2	ADPS1	ADPS0	ADC Clock
0	0	0	Reserved
0	0	1	CK/2
0	1	0	CK/4
0	1	1	CK/8
1	0	0	CK/16
1	0	1	CK/32
1	1	0	CK/64
1	1	1	CK/128

Figure 12. AVR ADC Clock Selection

Example 3

An AVR is connected to the 8 MHz crystal oscillator. Calculate the ADC frequency for (a) ADPS2:0 = 001 (b) ADPS2:0 = 100 (c) ADPS2:0 = 111

Solution:

(a) Because ADPS2:0 = 001 (1 decimal), the ck/2 input will be activated; we have 8 MHz / 2 = 4 MHz (greater than 200 kHz and not valid)

(b) Because ADPS2:0 = 100 (4 decimal), the ck/8 input will be activated; we have 8 MHz / 16 = 500 kHz (greater than 200 kHz and not valid)

(c) Because ADPS2:0 = 111 (7 decimal), the ck/128 input will be activated; we have 8 MHz / = 62 kHz (a valid option since it is less than 200 kHz)

Sample-and-hold time in ADC

A timing factor that we should know about is the acquisition time. After an ADC channel is selected, the ADC allows some time for the sample-and-hold capacitor (C hold) to charge fully to the input voltage level present at the channel.

In the AVR, the first conversion takes 25 ADC clock cycles in order to initialize the analog circuitry and pass the sample-and-hold time. Then each consecutive conversion takes 13 ADC clock cycles.

Table 7 lists the conversion times for some different conditions. Notice that sample-and-hold time is the first part of each conversion.

Table 7: Conversion Time Table

Condition	Sample and Hold Time (Cycles)	Total Conversion Time (Cycles)
First Conversion	14.5	25
Normal Conversion, Single-ended	1.5	13
Normal Conversion, Differential	2	13.5
Auto trigger conversion	1.5 / 2.5	13/14

If the conversion time is not critical in your application and you do not want to deal with calculation of ADPS2:0 you can use ADPS2:0 = 111 to get the maximum accuracy of ADC.

Steps in programming the A/D converter using polling

To program the A/D converter of the AVR, the following steps must be taken:

1. Make the pin for the selected ADC channel an input pin.
2. Turn on the ADC module of the AVR because it is disabled upon power-on reset to save power.
3. Select the conversion speed. We use registers ADPS2:0 to select the conversion speed.
4. Select voltage reference and ADC input channels. We use the REFS0 and REFS1 bits in the ADMUX register to select voltage reference and the MUX4:0 bits in ADMUX to select the ADC input channel.
5. Activate the start conversion bit by writing a one to the ADSC bit of ADCSRA.
6. Wait for the conversion to be completed by polling the ADIF bit in the ADCSRA register.
7. After the ADIF bit has gone HIGH, read the ADCL and ADCH registers to get the digital data output. Notice that you have to read ADCL before ADCH; otherwise, the result will not be valid.
8. If you want to read the selected channel again, go back to step 5.
9. If you want to select another V_{ref} source or input channel, go back to step 4.

Programming AVR ADC in Assembly and C

The Assembly language Program 1 illustrates the steps for ADC conversion shown above. Figure 13 shows the hardware connection of Program 1.

```
;Program 1:    This program gets data from channel 0 (ADC0) of
;ADC and displays the result on Port C and Port D. This is done
;forever.
;****************** Program 1    ************************

.INCLUDE "M32DEF.INC"
     LDI   R16,0xFF
     OUT   DDRB, R16        ;make Port B an output
     OUT   DDRD, R16        ;make Port D an output
     LDI   R16,0
     OUT   DDRA, R16        ;make Port A an input for ADC
     LDI   R16,0x87         ;enable ADC and select ck/128
     OUT   ADCSRA, R16
     LDI   R16,0xC0         ;2.56V Vref, ADC0 single ended
     OUT   ADMUX, R16       ;input, right-justified data
READ_ADC:
     SBI   ADCSRA,ADSC      ;start conversion
KEEP_POLING:                ;wait for end of conversion
     SBIS  ADCSRA,ADIF      ;is it end of conversion yet?
     RJMP  KEEP_POLING      ;keep polling end of conversion
     SBI   ADCSRA,ADIF      ;write 1 to clear ADIF flag
     IN    R16,ADCL         ;YOU HAVE TO READ ADCL FIRST
     OUT   PORTD,R16        ;give the low byte to PORTD
     IN    R16,ADCH         ;READ ADCH AFTER ADCL
     OUT   PORTB,R16        ;give the high byte to PORTB
     RJMP  READ_ADC         ;keep repeating it
```

Program 1: Reading ADC Using Polling Method in Assembly

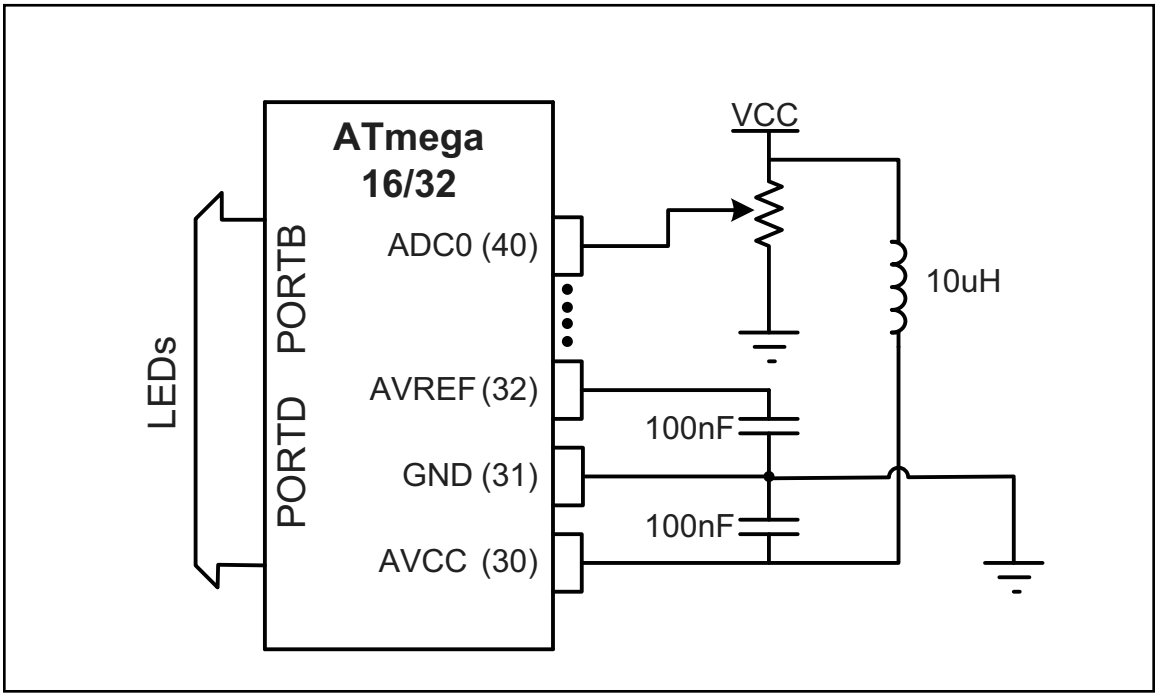


Figure 13. ADC Connection for Program 1

Program 1C is the C version of the ADC conversion for Program 1.

```
#include <avr/io.h>         //standard AVR header
int main (void)
{
  DDRB = 0xFF;              //make Port B an output
  DDRD = 0xFF;              //make Port D an output
  DDRA = 0;                 //make Port A an input for ADC input
  ADCSRA= 0x87;             //make ADC enable and select ck/128
  ADMUX= 0xC0;              //2.56V Vref, ADC0 single ended input
                            //data will be right-justified
  while (1){
    ADCSRA|=(1<<ADSC);      //start conversion
    while((ADCSRA&(1<<ADIF))==0);//wait for conversion to finish
    PORTD = ADCL;           //give the low byte to PORTD
    PORTB = ADCH;           //give the high byte to PORTB
  }
  return 0;
}
```

Program 1C: Reading ADC Using Polling Method in C

Programming A/D converter using interrupts

You should know how to use interrupts instead of polling to avoid tying down the microcontroller. To program the A/D using the interrupt method, we need to set HIGH the ADIE (A/D interrupt enable) flag. Upon completion of conversion, the ADIF (A/D interrupt flag) changes to HIGH; if ADIE = 1, it will force the CPU to jump to the ADC interrupt handler. Programs 2 and 2C show how to read ADC using interrupts.

```
.INCLUDE "M32DEF.INC"
.CSEG
     RJMP  MAIN
.ORG  ADCCaddr
     RJMP  ADC_INT_HANDLER
.ORG  40
;*****************************
MAIN: LDI   R16, HIGH(RAMEND)
     OUT   SPH, R16
     LDI   R16, LOW(RAMEND)
     OUT   SPL,R16
     SEI
     LDI   R16,0xFF
     OUT   DDRB, R16        ;make Port B an output
     OUT   DDRD, R16        ;make Port D an output
     LDI   R16,0
     OUT   DDRA, R16        ;make Port A an input for ADC
     LDI   R16,0x8F         ;enable ADC and select ck/128
     OUT   ADCSRA, R16
     LDI   R16,0xC0         ;2.56V Vref, ADC0 single ended
     OUT   ADMUX, R16       ;input right-justified data
     SBI   ADCSRA,ADSC      ;start conversion
```

Program 2: Reading ADC Using Interrupts in Assembly *(continued on next page)*

```
WAIT_HERE:
      RJMP  WAIT_HERE        ;keep repeating it
;***************************
ADC_INT_HANDLER:
      IN    R16,ADCL         ;YOU HAVE TO READ ADCL FIRST
      OUT   PORTD,R16        ;give the low byte to PORTD
      IN    R16,ADCH         ;READ ADCH AFTER ADCL
      OUT   PORTB,R16        ;give the high byte to PORTB
      SBI   ADCSRA,ADSC      ;start conversion again
      RETI
```

Program 2: Reading ADC Using Interrupts in Assembly *(continued from previous page)*

Program 2C is the C version of Program 2. Notice that this program is checked under WinAVR (20080610). If you use another compiler you may need to read the documentation of your compiler to know how to deal with interrupts in your compiler.

```
#include <avr\io.h>
#include <avr\interrupt.h>
ISR(ADC_vect){
  PORTD = ADCL;             //give the low byte to PORTD
  PORTB = ADCH;             //give the high byte to PORTB
  ADCSRA|=(1<<ADSC);        //start conversion
}
int main (void){
  DDRB = 0xFF;              //make Port B an output
  DDRD = 0xFF;              //make Port D an output
  DDRA = 0;                 //make Port A an input for ADC input
  sei();                    //enable interrupts
  ADCSRA= 0x8F;             //enable and interrupt select ck/128
  ADMUX= 0xC0;              //2.56V Vref and ADC0 single-ended
                            //input right-justified data
  ADCSRA|=(1<<ADSC);        //start conversion
  while (1);                //wait forever
  return 0;
}
```

Program 2C: Reading ADC Using Interrupts in C

Review Questions

1. What is the internal V_{ref} of the ATmega32?
2. The A/D of AVR is a(n) _____-bit converter.
3. True or false. The A/D of AVR has pins for D_{OUT}.
4. True or false. A/D in the AVR is an off-chip module.
5. Find the step size for an AVR ADC, if V_{ref} = 2.56 V.
6. For problem 5, calculate the D0–D9 output if the analog input is: (a) 0.7 V, and (b) 1 V.
7. How many single-ended inputs are available in the ATmega32 ADC?
8. Calculate the first conversion time for ADPS0–2 = 111 and Fosc = 4 MHz.
9. In AVR, the ADC requires an input clock frequency less than _____ .
10. Which bit is used to poll for the end of conversion?

SECTION 3: SENSOR INTERFACING AND SIGNAL CONDITIONING

This section will show how to interface sensors to the microcontroller. We examine some popular temperature sensors and then discuss the issue of signal conditioning. Although we concentrate on temperature sensors, the principles discussed in this section are the same for other types of sensors such as light and pressure sensors.

Temperature sensors

Transducers convert physical data such as temperature, light intensity, flow, and speed to electrical signals. Depending on the transducer, the output produced is in the form of voltage, current, resistance, or capacitance. For example, temperature is converted to electrical signals using a transducer called a *thermistor*. A thermistor responds to temperature change by changing resistance, but its response is not linear, as seen in Table 8.

The complexity associated with writing software for such nonlinear devices has led many manufacturers to market a linear temperature sensor. Simple and widely used linear temperature sensors include the LM34 and LM35 series from National Semiconductor Corp. They are discussed next.

Table 8: Thermistor Resistance vs. Temperature

Temperature (C)	Tf (K ohms)
0	29.490
25	10.000
50	3.893
75	1.700
100	0.817

From William Kleitz, Digital Electronics

LM34 and LM35 temperature sensors

The sensors of the LM34 series are precision integrated-circuit temperature sensors whose output voltage is linearly proportional to the Fahrenheit temperature. See Table 9. The LM34 requires no external calibration because it is inter nally calibrated. It outputs 10 mV for each degree of Fahrenheit temperature. Table 9 is a selection guide for the LM34.

Table 9: LM34 Temperature Sensor Series Selection Guide

Part	Temperature Range	Accuracy	Output Scale
LM34A	–50 F to +300 F	+2.0 F	10 mV/F
LM34	–50 F to +300 F	+3.0 F	10 mV/F
LM34CA	–40 F to +230 F	+2.0 F	10 mV/F
LM34C	–40 F to +230 F	+3.0 F	10 mV/F
LM34D	–32 F to +212 F	+4.0 F	10 mV/F

Note: Temperature range is in degrees Fahrenheit.

Table 10: LM35 Temperature Sensor Series Selection Guide

Part	Temperature Range	Accuracy	Output Scale
LM35A	–55 C to +150 C	+1.0 C	10 mV/C
LM35	–55 C to +150 C	+1.5 C	10 mV/C
LM35CA	–40 C to +110 C	+1.0 C	10 mV/C
LM35C	–40 C to +110 C	+1.5 C	10 mV/C
LM35D	0 C to +100 C	+2.0 C	10 mV/C

Note: Temperature range is in degrees Celsius.

The LM35 series sensors are precision integrated-circuit temperature sensors whose output voltage is linearly proportional to the Celsius (centigrade) temperature. The LM35 requires no external calibration because it is internally calibrated. It outputs 10 mV for each degree of centigrade temperature. Table 10 is the selection guide for the LM35. (For further information see http://www.national.com.)

Signal conditioning

Signal conditioning is widely used in the world of data acquisition. The most common transducers produce an output in the form of voltage, current, charge, capacitance, and resistance. We need to convert these signals to voltage, however, in order to send input to an A-to-D converter. This conversion (modification) is commonly called *signal conditioning*. See Figure 14. Signal conditioning can be current-to-voltage conversion or signal amplification. For example, the thermistor changes resistance with temperature. The change of resistance must be translated into voltages to be of any use to an ADC. We now look at the case of connecting an LM34 (or LM35) to an ADC of the ATmega32.

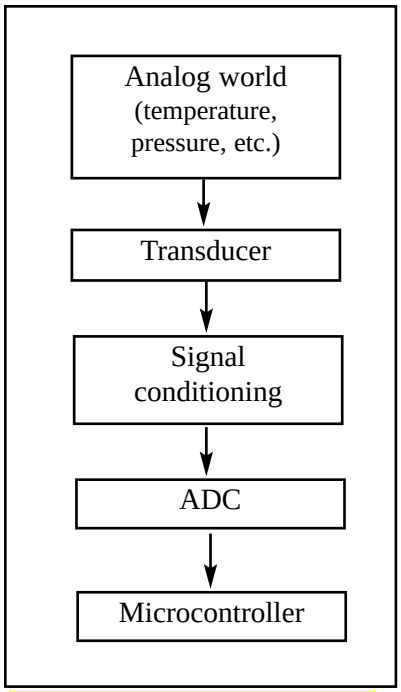


Figure 14. Getting Data from the Analog World

Interfacing the LM34 to the AVR

The A/D has 10-bit resolution with a maximum of 1024 steps, and the LM34 (or LM35) produces 10 mV for every degree of temperature change. Now, if we use the step size of 10 mV, the V_{out} will be 10,240 mV (10.24 V) for full-scale output. This is not acceptable even though the maximum temperature sensed by the LM34 is 300 degrees F, and the highest output we will get for the A/D is 3000 mV (3.00 V) .

Now if we use the internal 2.56 V reference voltage, the step size would be 2.56 V/1024 = 2.5 mV. This makes the binary output number for the ADC four times the real temperature because the sensor produces 10 mV for each degree of temperature change and the step size is 2.5 mV (10 mV/2.5 mV = 4). We can scale it by dividing it by 4 to get the real number for temperature. See Table 11.

Table 11: Temperature vs. V_{out} for AVR with V_{ref} = 2.56 V

Temp. (F)	V_{in} (mV)	# of steps	Binary V_{out} (b9–b0)	Temp. in Binary
0	0	0	00 00000000	00000000
1	10	4	00 00000100	00000001
2	20	8	00 00001000	00000010
3	30	12	00 00001100	00000011
10	100	20	00 00101000	00001010
20	200	80	00 01010000	00010100
30	300	120	00 01111000	00011110
40	400	160	00 10100000	00101000
50	500	200	00 11001000	00110010
60	600	240	00 11110000	00111100
70	700	300	01 00011000	01000110
80	800	320	01 01000000	01010000
90	900	360	01 01101000	01011010
100	1000	400	01 10010000	01100100

Figure 15 shows the pin configuration of the LM34/LM35 temperature sensor and the connection of the temperature sensor to the ATmega32.

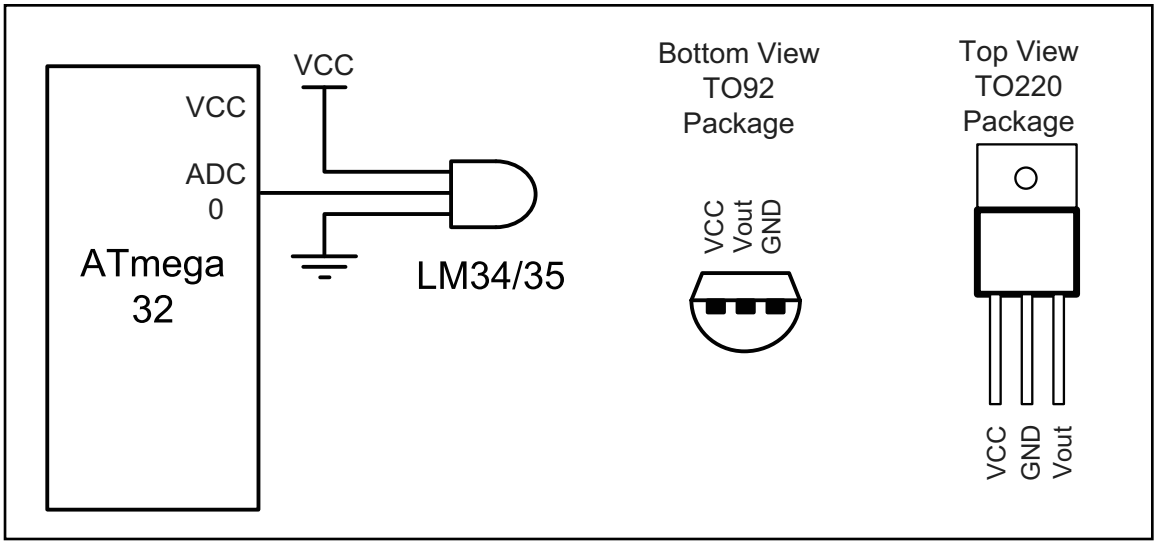


Figure 15. LM34/35 Connection to AVR and Its Pin Configuration

Reading and displaying temperature

Programs 4 and 4C show code for reading and displaying temperature in both Assembly and C, respectively.

The programs correspond to Figure 15. Regarding these two programs, the following points must be noted:

(1) The LM34 (or LM35) is connected to channel 0 (ADC0 pin).

(2) The 10-bit output of the A/D is divided by 4 to get the real temperature.

(3) To divide the 10-bit output of the A/D by 4 we choose the left-justified option and only read the ADCH register. It is same as shifting the result two bits right. See Example 4.

```
;this program reads the sensor and displays it on Port D
.INCLUDE "M32DEF.INC"
     LDI   R16,0xFF
     OUT   DDRD, R16        ;make Port D an output
     LDI   R16,0
     OUT   DDRA, R16        ;make Port A an input for ADC
     LDI   R16,0x87         ;enable ADC and select ck/128
     OUT   ADCSRA, R16
     LDI   R16,0xE0         ;2.56 V Vref, ADC0 single-ended
     OUT   ADMUX, R16       ;left-justified data
READ_ADC:
     SBI   ADCSRA,ADSC      ;start conversion
KEEP_POLING:                ;wait for end of conversion
     SBIS  ADCSRA,ADIF      ;is it end of conversion?
     RJMP  KEEP_POLING      ;keep polling end of conversion
     SBI   ADCSRA,ADIF      ;write 1 to clear ADIF flag
     IN    R16,ADCH         ;read only ADCH for 8 MSB of
     OUT   PORTD,R16        ;result and give it to PORTD
     RJMP  READ_ADC         ;keep repeating
```

Program 3: Reading Temperature Sensor in Assembly

```
;this program reads the sensor and displays it on Port D
#include <avr/io.h>         //standard AVR header
int main (void)
{
  DDRD = 0xFF;              //make Port D an output
  DDRA = 0;                 //make Port A an input for ADC input
  ADCSRA = 0x87;            //make ADC enable and select ck/128
  ADMUX = 0xE0;             //2.56 V Vref and ADC0 single-ended
                            //data will be left-justified
  while (1){
     ADCSRA |= (1<<ADSC);  //start conversion
     while((ADCSRA&(1<<ADIF))==0); //wait for end of conversion
     PORTB = ADCH;          //give the high byte to PORTB
  }
  return 0;
}
```

Program 3C: Reading Temperature Sensor in C

Example 4

In Table 11, verify the AVR output for a temperature of 70 degrees. Find values in the AVR A/D registers of ADCH and ADCL for left-justified.

Solution:

The step size is 2.56/1024 = 2.5 mV because Vref = 2.56 V.

For the 70 degrees temperature we have 700 mV output because the LM34 provides 10 mV output for every degree. Now, the number of steps are 700 mV/2.5 mV = 280 in decimal. Now 280 = 0100011000 in binary and the AVR A/D output registers have ADCH = 01000110 and ADCL = 00000000 for left-justified. To get the proper result we must divide the result by 4. To do that, we simply read the ADCH register, which has the value 70 (01000110) in it.

Review Questions

1. True or false. The transducer must be connected to signal conditioning circuitry before its signal is sent to the ADC.
2. The LM35 provides _____ mV for each degree of _____ (Fahrenheit, Celsius) temperature.
3. The LM34 provides ____ mV for each degree of _____ (Fahrenheit, Celsius) temperature.
4. Why do we set the V_{ref} of the AVR to 2.56 V if the analog input is connected to the LM35?
5. In Question 4, what is the temperature if the ADC output is 0011 1001?

SECTION 4: DAC INTERFACING

This section will show how to interface a DAC (digital-to-analog converter) to the AVR. Then we demonstrate how to generate a stair-step ramp on the scope using the DAC.

Digital-to-analog converter (DAC)

The digital-to-analog converter (DAC) is a device widely used to convert digital pulses to analog signals. In this section we discuss the basics of interfacing a DAC to the AVR.

Recall from your digital electronics course the two methods of creating a DAC: binary weighted and R/2R ladder. The vast majority of integrated circuit DACs, including the MC1408 (DAC0808) used in this section, use the R/2R method because it can achieve a much higher degree of precision. The first criterion for judging a DAC is its resolution, which is a function of the number of binary inputs. The common ones are 8, 10, and 12 bits. The number of data bit inputs decides the resolution of the DAC because the number of analog output levels is equal to 2^n, where n is the number of data bit inputs. Therefore, an 8-input DAC such as the DAC0808 provides 256 discrete voltage (or current) levels of output. See Figure 16. Similarly, the 12-bit DAC provides 4096 discrete voltage levels There are also 16-bit DACs, but they are more expensive.

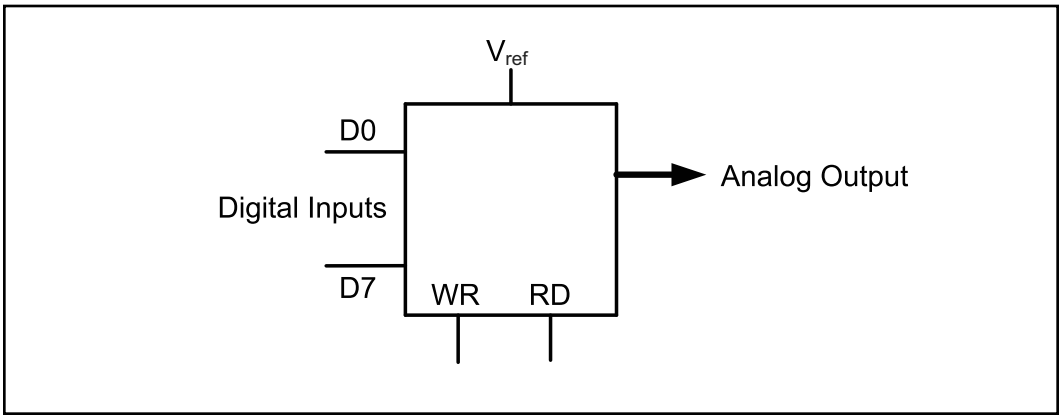


Figure 16. DAC Block Diagram

MC1408 DAC (or DAC0808)

In the MC1408 (DAC0808), the digital inputs are converted to current (I_{out}), and by connecting a resistor to the I_{out} pin, we convert the result to voltage. The total current provided by the I_{out} pin is a function of the binary numbers at the D0–D7 inputs of the DAC0808 and the reference current (I_{ref}), and is as follows:

$$I_{out} = I_{ref} \left(\frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256} \right)$$

where D0 is the LSB, D7 is the MSB for the inputs, and I_{ref} is the input current that must be applied to pin 14. The I_{ref} current is generally set to 2.0 mA. Figure 17 shows the generation of current reference (setting I_{ref} = 2 mA) by using the standard 5 V power supply. Now assuming that I_{ref} = 2 mA, if all the inputs to the DAC are high, the maximum output current is 1.99 mA (verify this for yourself).

Converting I_{out} to voltage in DAC0808

Ideally we connect the output pin I_{out} to a resistor, convert this current to voltage, and monitor the output on the scope. In real life, however, this can cause inaccuracy because the input resistance of the load where it is connected will also affect the output voltage. For this reason, the I_{ref} current output is isolated by connecting it to an op-amp such as the 741 with R_f = 5 kilohms for the feedback resistor. Assuming that R = 5 kilohms, by changing the binary input, the output voltage changes as shown in Example 5.

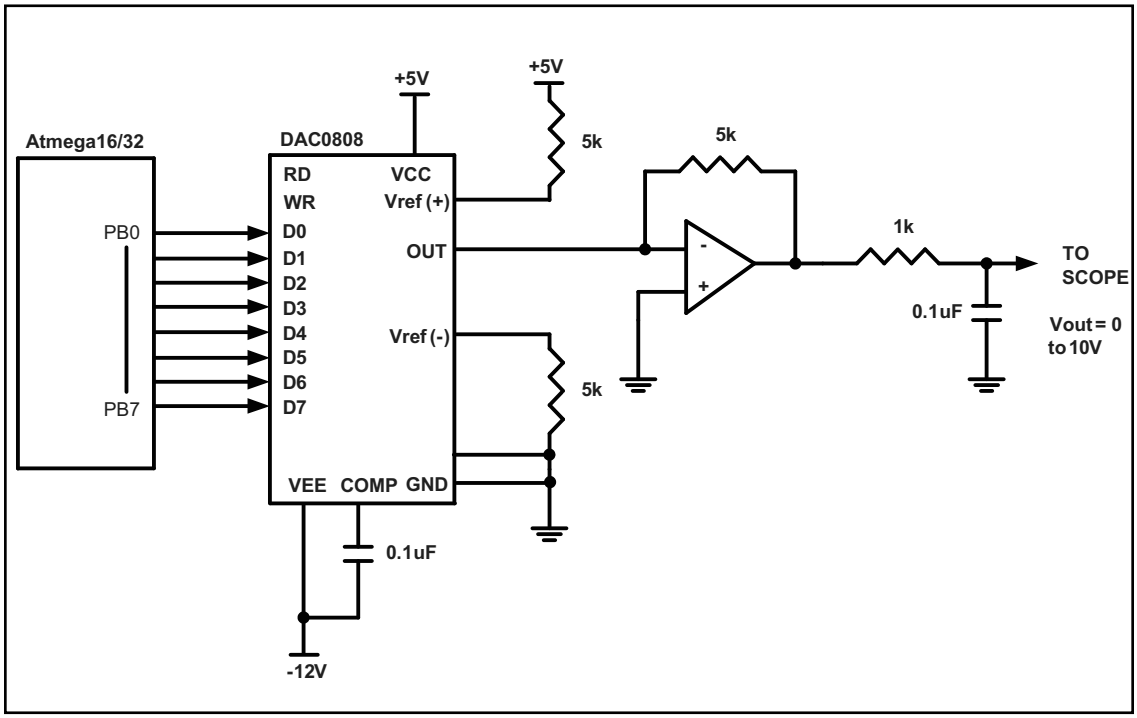


Figure 17. AVR Connection to DAC0808

Example 5

Assuming that R = 5 kilohms and I_{ref} = 2 mA, calculate V_{out} for the following binary inputs:

(a) 10011001 binary (99H)
(b) 11001000 (C8H)

Solution:

(a) I_{out} = 2 mA (153/256) = 1.195 mA and V_{out} = 1.195 mA × 5K = 5.975 V
(b) I_{out} = 2 mA (200/256) = 1.562 mA and V_{out} = 1.562 mA × 5K = 7.8125 V

Generating a stair-step ramp

In order to generate a stair-step ramp, you can set up the circuit in Figure 17 and load Program 4 on the AVR chip. To see the result wave, connect the output to an oscilloscope. Figure 18 shows the output.

```
LDI   R16,0xFF
      OUT   DDRB, R16        ;make Port B an output
AGAIN:
      INC   R16              ;increment R16
      OUT   PORTB,R16        ;sent R16 to PORTB
      NOP                    ;let DAC recover
      NOP
      RJMP  AGAIN
```

Program 4: DAC Programming

Programming DAC in C

Program 4C shows how to program the DAC in C.

```
#include <avr/io.h>         //standard AVR header

int main (void)
{
  unsigned char i = 0;      //define a counter
  DDRB = 0xFF;              //make Port B an output
  while (1){                //do forever
    PORTB = i;              //copy i into PORTB to be converted
    i++;                    //increment the counter
  }
  return 0;
}
```

Program 4C: DAC Programming in C

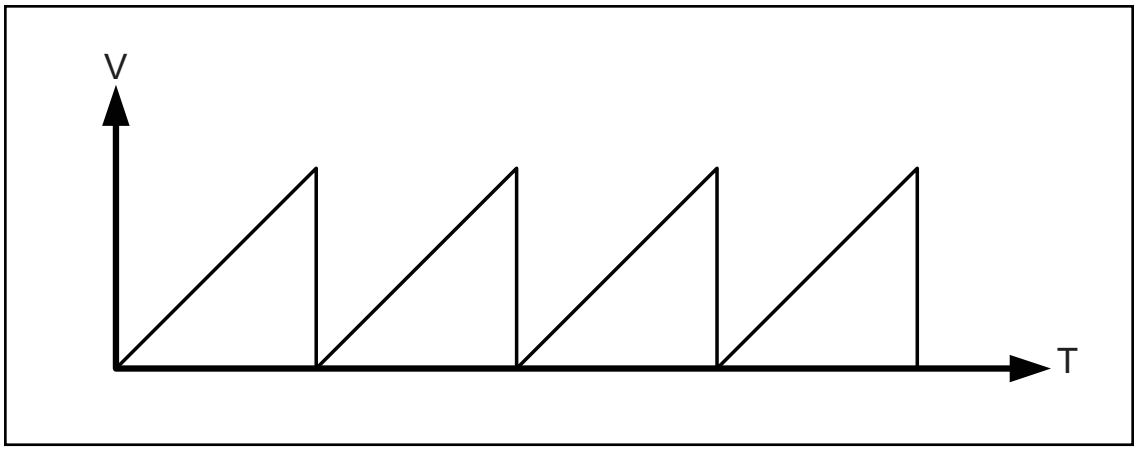


Figure 18. Stair Step Ramp Output

Review Questions

1. In a DAC, input is _____ (digital, analog) and output is _____ (digital, analog).
2. In an ADC, input is _____ (digital, analog) and output is _____ (digital, analog).
3. DAC0808 is a(n) ____-bit D-to-A converter.
4. (a) The output of DAC0808 is in _____ (current, voltage).
   (b) True or false. The output of DAC0808 is ideal to drive a motor.

SUMMARY

This chapter showed how to interface real-world devices such as DAC chips, ADC chips, and sensors to the AVR. First, we discussed both parallel and serial ADC chips, then described how the ADC module inside the AVR works and explained how to program it in both Assembly and C. Next we explored sensors. We also discussed the relation between the analog world and a digital device, and described signal conditioning, an essential feature of data acquisition systems. In the last section we studied the DAC chip, and showed how to interface it to the AVR.

PROBLEMS

SECTION 1: ADC CHARACTERISTICS

1. True or false. The output of most sensors is analog.
2. True or false. A 10-bit ADC has 10-bit digital output.
3. True or false. ADC0848 is an 8-bit ADC.
4. True or false. MAX1112 is a 10-bit ADC.
5. True or false. An ADC with 8 channels of analog input must have 8 pins, one for each analog input.

6. True or false. For a serial ADC, it takes a longer time to get the converted digital data out of the chip.
7. True or false. ADC0848 has 4 channels of analog input.
8. True or false. MAX1112 has 8 channels of analog input.
9. True or false. ADC0848 is a serial ADC.
10. True or false. MAX1112 is a parallel ADC.
11. Which of the following ADC sizes provides the best resolution?
    (a) 8-bit (b) 10-bit (c) 12-bit (d) 16-bit (e) They are all the same.
12. In Question 11, which provides the smallest step size?
13. Calculate the step size for the following ADCs, if V_{ref} is 5 V:
    (a) 8-bit (b) 10-bit (c) 12-bit (d) 16-bit
14. With V_{ref} = 1.28 V, find the V_{in} for the following outputs:
    (a) D7–D0 = 11111111 (b) D7–D0 = 10011001 (c) D7–D0 = 1101100
15. In the ADC0848, what should the V_{ref} value be if we want a step size of 5 mV?
16. With V_{ref} = 2.56 V, find the V_{in} for the following outputs:
    (a) D7–D0 = 11111111 (b) D7–D0 = 10011001 (c) D7–D0 = 01101100

SECTION 2: ADC PROGRAMMING IN THE AVR

17. True or false. The ATmega32 has an on-chip A/D converter.
18. True or false. A/D of the ATmega32 is an 8-bit ADC.
19. True or false. ATmega32 has 8 channels of analog input.
20. True or false. The unused analog pins of the ATmega32 can be used for I/O pins.
21. True or false. The A/D conversion speed in the ATmega32 depends on the crystal frequency.
22. True or false. Upon power-on reset, the A/D module of the ATmega32 is turned on and ready to go.
23. True or false. The A/D module of the ATmega32 has an external pin for the start-conversion signal.
24. True or false. The A/D module of the ATmega32 can convert only one channel at a time.
25. True or false. The A/D module of the ATmega32 can have multiple external V_{ref}+ at any given time.
26. True or false. The A/D module of the ATmega32 can use the V_{cc} for V_{ref}.
27. In the A/D of ATmega32, what happens to the converted analog data? How do we know that the ADC is ready to provide us the data?
28. In the A/D of ATmega32, what happens to the old data if we start conversion again before we pick up the last data?
29. For the A/D of ATmega32, find the step size for each of the following V_{ref}:
    (a) V_{ref} = 1.024 V (b) V_{ref} = 2.048 V (c) V_{ref} = 2.56 V
30. In the ATmega32, what should the V_{ref} value be if we want a step size of 2 mV?
31. In the ATmega32, what should the V_{ref} value be if we want a step size of 3 mV?

32. With a step size of 1 mV, what is the analog input voltage if all outputs are 1?
33. With V_{ref} = 1.024 V, find the V_{in} for the following outputs:
    (a) D9–D0 = 0011111111 (b) D9–D0 = 0010011000 (c) D9–D0 = 0011010000
34. In the A/D of ATmega32, what should the V_{ref} value be if we want a step size of 4 mV?
35. With V_{ref} = 2.56 V, find the V_{in} for the following outputs:
    (a) D9–D0 = 1111111111 (b) D9–D0 = 1000000001 (c) D9–D0 = 1100110000
36. Find the first conversion times for the following cases if XTAL = 8 MHz. Are they acceptable?
    (a) Fosc/2 (b) Fosc/4 (c) Fosc/8 (d) Fosc/16 (e) Fosc/32
37. Find the first conversion times for the following cases if XTAL = 4 MHz. Are they acceptable?
    (a) Fosc/8 (b) Fosc/16 (c) Fosc/32 (d) Fosc/64
38. How do we start conversion in the ATmega32?
39. How do we recognize the end of conversion in the ATmega32?
40. Which bits of which register of the ATmega32 are used to select the A/D's conversion speed?
41. Which bits of which register of the ATmega32 are used to select the analog channel to be converted?
42. Give the names of the interrupt flags for the A/D of the ATmega32. State to which register they belong.
43. Upon power-on reset, the A/D of the ATmega32 is given (on, off).

SECTION 3: SENSOR INTERFACING AND SIGNAL CONDITIONING

44. What does it mean when a given sensor is said to have a linear output?
45. The LM34 sensor produces _____ mV for each degree of temperature.
46. What is signal conditioning?

SECTION 4: DAC INTERFACING

47. True or false. DAC0808 is the same as DAC1408.
48. Find the number of discrete voltages provided by the *n*-bit DAC for the following:
    (a) $n = 8$ (b) $n = 10$ (c) $n = 12$
49. For DAC1408, if I_{ref} = 2 mA, show how to get the I_{out} of 1.99 when all inputs are HIGH.
50. Find the I_{out} for the following inputs. Assume I_{ref} = 2 mA for DAC0808.
    (a) 10011001 (b) 11001100 (c) 11101110
    (d) 00100010 (e) 00001001 (f) 10001000
51. To get a smaller step, we need a DAC with _____ (more, fewer) digital inputs.
52. To get full-scale output, what should be the inputs for DAC?

ANSWERS TO REVIEW QUESTIONS

SECTION 1: ADC CHARACTERISTICS

1. Number of steps and V_{ref} voltage
2. 8
3. True
4. 1.28 V/256 = 5 mV
5. (a) 0.7 V/ 5 mV = 140 in decimal and D7–D0 = 10001100 in binary.
   (b) 1 V/ 5 mV = 200 in decimal and D7–D0 = 11001000 in binary.

SECTION 2: ADC PROGRAMMING IN THE AVR

1. 2.56 V
2. 10
3. False
4. False
5. 2.56/1024 = 2.5 mV
6. (a) 700 mV/2.5 mV = 280 (100011000), (b) 1000 mV/ 2.5 mV = 400 (110010000)
7 8 channels
8. (1/(4 MHz/128)) × 25 = 800 microseconds
9. 200 kHz
10. ADIF bit of the ADCSRA register

SECTION 3: SENSOR INTERFACING AND SIGNAL CONDITIONING

1. True
2. 10, Celsius
3. 10, Fahrenheit
4. Using the 8-bit part of the 10-bit ADC, it gives us 256 steps, and 2.56 V/256 = 10 mV. The LM35 produces 10 mV for each degree of temperature, which matches the ADC's step size.
5. 00111001 = 57, which indicates it is 57 degrees.

SECTION 4: DAC INTERFACING

1. Digital, analog
2. Analog, digital
3. 8
4. (a) current (b) true