12

Analog-to-Digital Converter

12.1 Objectives

After completing this chapter, you should be able to

- Explain the A/D conversion process
- Describe the resolution, the various channels, and the operation modes of the PIC18 A/D converter
- Interpret the A/D conversion results
- Describe the procedure for using the PIC18 A/D converter
- Configure the A/D converter for the application
- Use the temperature sensor TC1047A
- Use the humidity sensor IH-3605
- Use the barometric pressure sensor BPT

12.2 Basics of A/D Conversion

Many embedded applications deal with nonelectric quantities, such as weight, humidity, pressure, mass or airflow, temperature, light intensity, and speed. These quantities are *analog* in nature because they have a continuous set of values over a given range, in contrast to the discrete values of digital signals. To enable the microcontroller to process these quantities, they need to be represented in digital form; thus, an *analog-to-digital* (A/D) converter is required.

12.2.1 A Data Acquisition System

An A/D converter can deal only with electrical voltage. A nonelectric quantity must be converted into a voltage before A/D conversion can be performed. The conversion of a nonelectric quantity to a voltage requires the use of a *transducer*. In general, a transducer is a device that converts the quantity from one form to another. For example, a temperature sensor is a transducer that can convert the temperature into a voltage. A *load cell* is the transducer that can convert a weight into a voltage.

A transducer may not generate an output voltage in the range suitable for A/D conversion. A voltage *scaler* (or *amplifier*) is often needed to amplify the transducer output voltage into a range that can be handled by the A/D converter. The circuit that performs the scaling and shifting of the transducer output is called a *signal-conditioning circuit*. The overall A/D process is illustrated in Figure 12.1.

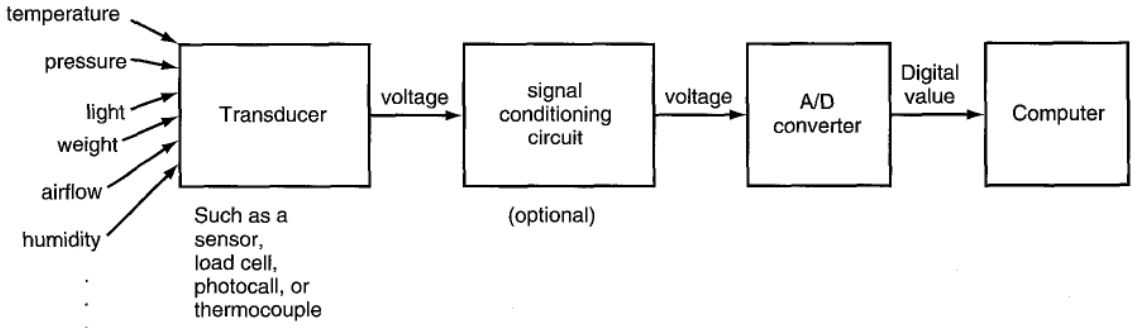

Figure 12.1 ■ The A/D conversion process

12.2.2 Analog Voltage and Digital Code Characteristic

An ideal A/D converter should demonstrate the linear input/output relationship as shown in Figure 12.2.

The output characteristic shown in Figure 12.2 is unrealistic because it requires the A/D converter to use infinite number of bits to represent the conversion result. The output characteristic of an ideal A/D converter using n bits to represent the conversion result is shown in Figure 12.3. An n-bit A/D converter has 2^n possible output code values. The area between the dotted line and the staircase is called the *quantization error*. The value of $V_{DD}/2^n$ is the resolution of this A/D converter. Using n bits to represent the conversion result, the average *conversion error* is $V_{DD}/2^{n+1}$ if the converter is perfectly linear. For a real A/D converter, the output characteristic may have *nonlinearity* (the staircase may have unequal steps in some values) and *nonmonotonicity* (higher voltage may have smaller code).

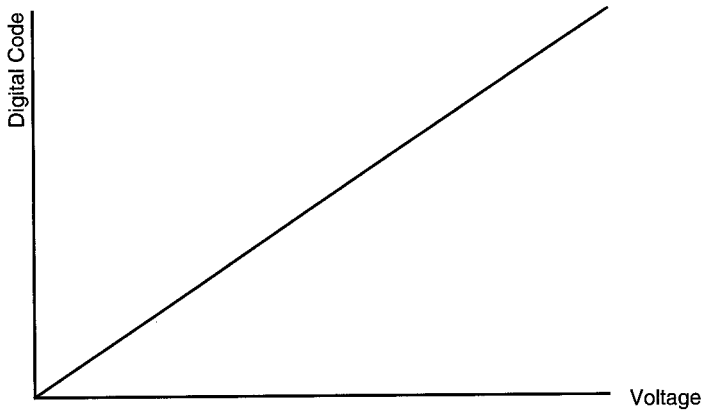

Figure 12.2 ■ An ideal A/D converter output characteristic

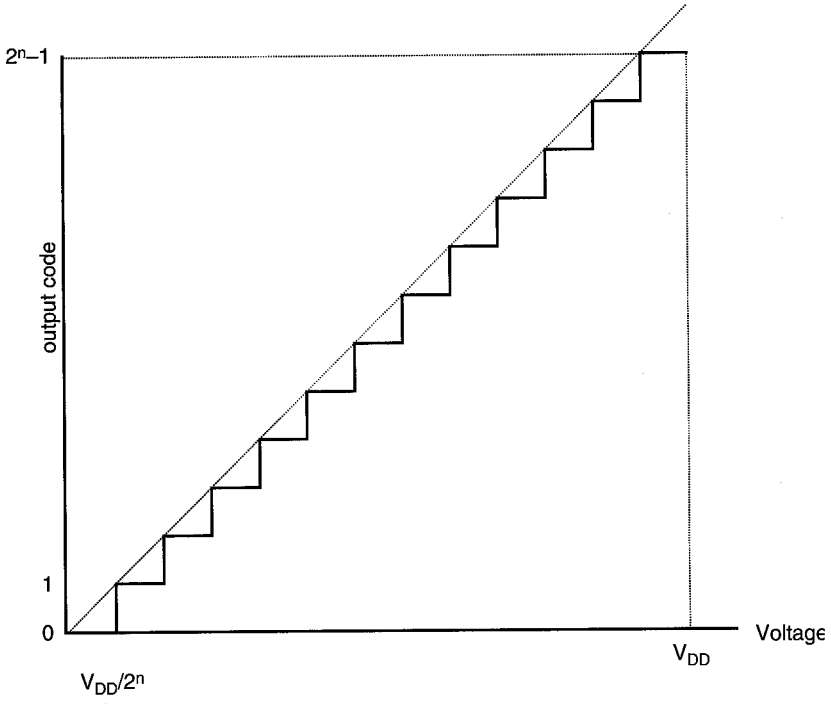

Figure 12.3 ■ Output characteristic of an ideal n-bit A/D converter

Obviously, the more bits used in representing the A/D conversion result, the smaller the conversion error will be. Most microcontrollers used 8 bits, 10 bits, or 12 bits to represent the conversion result. Some microcontrollers (mainly 8051 variants from Silicon Laboratory, TI, and Analog Devices) use 16 bits or even 24 bits to represent conversion results. Whenever the on-chip A/D converter cannot provide the required accuracy, an external A/D converter should be considered.

12.2.3 A/D Conversion Algorithms

Many A/D conversion algorithms have been invented in the past. These algorithms can be divided into four categories:

1. Parallel (flash) A/D converter
2. Slope and double-slope A/D converter
3. Sigma-delta A/D converter
4. Successive approximation A/D converter

Parallel (Flash) A/D Converters

In this type of A/D converter, 2^n comparators are used. One of the inputs to each comparator is the input voltage to be converted, whereas the other input corresponds to the voltage that represents one of the 2^n combinations of n-bit values. The comparator output will be high whenever the analog input (to be converted) is higher than the voltage that represents one of the 2^n combinations of the n-bit value. The largest n-bit value that causes the comparator output to become true is selected as the A/D conversion value. It is obvious that this type of A/D converter will be very fast. However, they require a lot of hardware resources to implement and therefore are not suitable for implementing high-resolution A/D converters. Over the years, several variations to this approach have been proposed to produce high-speed A/D converters. The most commonly used technique is to pipeline a flash A/D converter, which will reduce the amount of hardware required while still achieving high conversion speed.

Slope and Double-Slope A/D Converters

This type of A/D converter is used in PIC14000 microcontrollers in which the charging and discharging of a capacitor is used to perform the A/D conversion. This type of A/D converter requires relatively simple hardware and is popular in low-speed applications. In addition, high resolution (10-bit to 16-bit) can be achieved.

Sigma-Delta A/D Converters

This type of A/D converter uses the *oversampling* technique to perform A/D conversion. This type of converter has good noise immunity and can achieve high resolution. Sigma-delta A/D converters are becoming more popular in implementing high-resolution A/D converters. The only disadvantage is its conversion speed. However, this weakness is improving because of the improvement in CMOS technology.

Successive-Approximation A/D Converters

This method was first detailed in the December 1975 issue of the IEEE Journal of Solid State. This method utilizes charge redistribution over an array of ratioed capacitors to perform A/D conversion. The block diagram of this method is shown in Figure 12.4.

The successive-approximation method approximates the analog signal to n-bit code in n steps. It first initializes the successive-approximation register (SAR) to 0 and then performs a series of guessing, starting with the most significant bit and proceeding toward the least significant bit. The algorithm of the successive-approximation method is illustrated in Figure 12.5. For every bit of the SAR, the algorithm does the following:

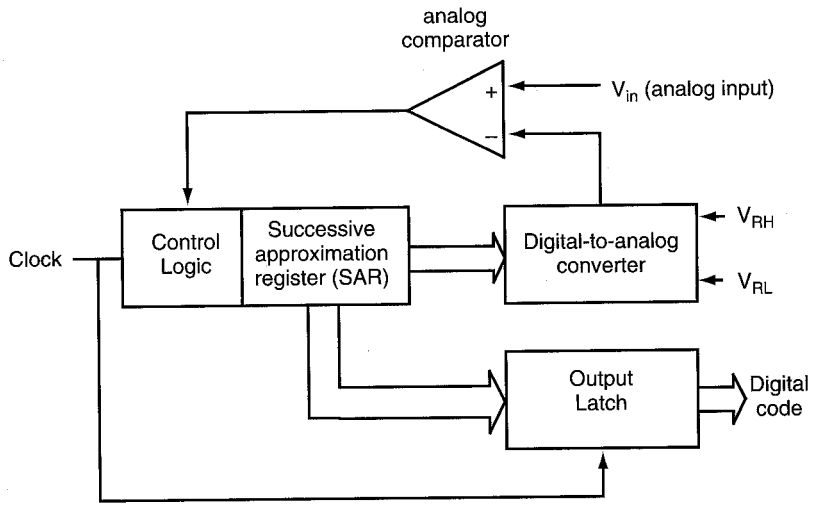

Figure 12.4 ■ Block diagram of a successive approximation A/D converter

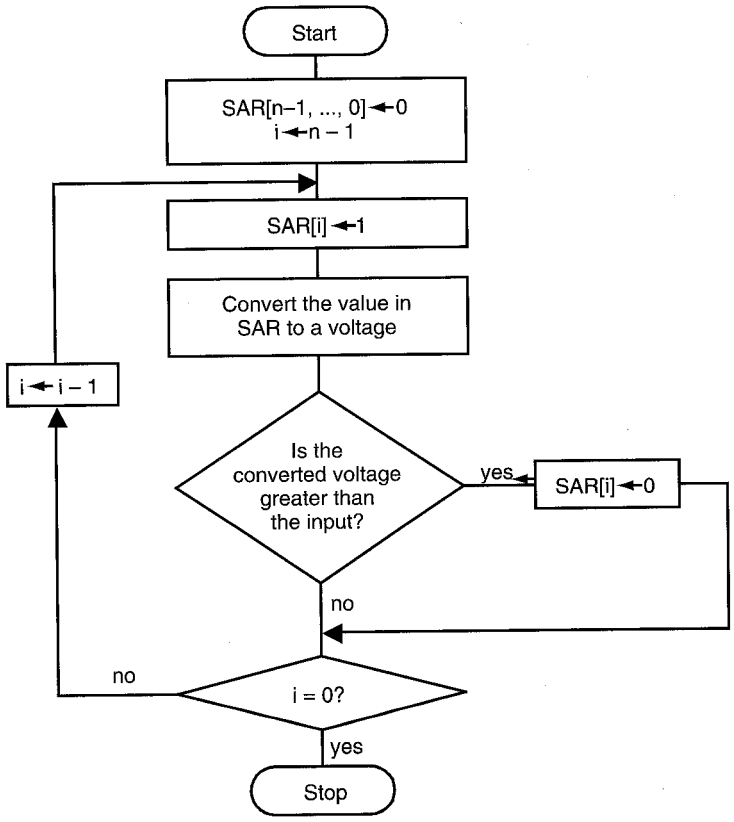

Figure 12.5 ■ Successive approximation A/D conversion method

- Guesses the bit to be a 1
- Converts the value of the SAR register to an analog voltage
- Compares the D/A output with the analog input and clears the bit to 0 if the D/A output is larger (which indicates that the guess is wrong)

Because of its balanced speed and precision, this method has become one of the most popular A/D conversion methods. Most microcontrollers use this method to implement the A/D converter. The PIC18 microcontrollers also use this technique to implement A/D converters.

12.2.4 Optimal Voltage Range for A/D Conversion

An A/D converter needs a *negative reference voltage* (V_{REF-}) and a *positive reference voltage* (V_{REF+}) to perform the conversion. The V_{REF-} voltage is often set to ground, whereas the V_{REF+} voltage is often set to V_{DD}. Some microcontrollers simply tie V_{REF-} to the ground voltage and leave only the high-reference voltage programmable. Most A/D converters are *ratiometric* because of the following:

- A 0-V analog input is converted to the digital code of n 0s.
- A V_{DD} (or V_{REF+}) analog input is converted to the digital code of $2^n - 1$.
- A k-V input will be converted to the digital code of $k \times (2^n - 1) \div V_{DD}$.

Here, n is the number of bits used to represent the A/D conversion result.

By common sense, the A/D conversion result would be most accurate if the value of analog signal covers the whole voltage range from V_{REF-} to V_{REF+}. The A/D conversion result k corresponds to an analog voltage V_k given by the following equation:

$$V_k = V_{REF-} + (\text{range} \times k) \div (2^n - 1) \tag{12.1}$$

where range = $V_{REF-} - V_{REF+}$.

Example 12.1

Suppose that there is a 10-bit A/D converter with $V_{REF-} = 1$ V and $V_{REF+} = 4$ V. Find the corresponding voltage values for the A/D conversion results of 25, 80, 240, 500, 720, 800, and 900.

Solution: Range = $V_{REF-} - V_{REF+} = 4\text{ V} - 1\text{ V} = 3\text{ V}$

The voltage corresponding to the A/D conversion results of 25, 80, 240, 500, 720, 800, and 900 are as follows:

$1\text{ V} + (3 \times 25) \div (2^{10} - 1) = 1.07\text{ V}$
$1\text{ V} + (3 \times 80) \div (2^{10} - 1) = 1.23\text{ V}$
$1\text{ V} + (3 \times 240) \div (2^{10} - 1) = 1.70\text{ V}$
$1\text{ V} + (3 \times 500) \div (2^{10} - 1) = 2.47\text{ V}$
$1\text{ V} + (3 \times 720) \div (2^{10} - 1) = 3.11\text{ V}$
$1\text{ V} + (3 \times 800) \div (2^{10} - 1) = 3.35\text{ V}$
$1\text{ V} + (3 \times 900) \div (2^{10} - 1) = 3.64\text{ V}$

12.2.5 Scaling Circuit

Some of the transducer output voltages are in the range of 0 ~ V_Z, where $V_Z < V_{DD}$. Because V_Z sometimes can be much smaller than V_{DD}, the A/D converter cannot take advantage of the available full dynamic range, and therefore conversion results can be very inaccurate. The voltage scaling (amplifying) circuit can be used to improve the accuracy because it allows the A/D converter to utilize its full dynamic range. The diagram of a voltage scaling circuit is shown in Figure 12.6. Because the OP AMP has an infinite input impedance, the current that flows through the resistor R_2 will be the same as the current that flows through R_1. In addition, the voltage at the inverting input terminal (same as the voltage drop across R_1) would be the same as that at the noninverting terminal (V_{IN}). Therefore, the voltage gain of this circuit is given by the following equation:

$$A_V = V_{OUT} \div V_{IN} = (R_1 + R_2) \div R_1 = 1 + R_2/R_1 \tag{12.2}$$

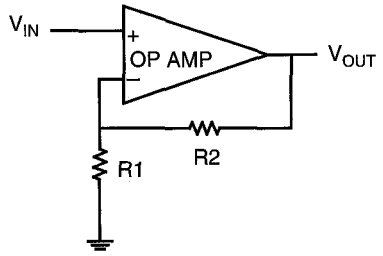

Figure 12.6 ■ A voltage scaler

Example 12.2

Suppose that the transducer output voltage ranges from 0 V to 200 mV. Choose the appropriate values for R_1 and R_2 to scale this range to 0 V ~ 5 V.

Solution: 5 V ÷ 200 mV = 25

$\therefore R_2/R_1 = 24$

By choosing 240 KΩ for R_2 and 10 KΩ for R_1, we obtain a R_2/R_1 ratio of 24 and achieve the desired scaling goal.

12.2.6 Voltage Translation Circuit

Some transducers have output voltage in the range of V_1 ~ V_2 (V_1 can be negative and V_2 can be unequal to V_{DD}) instead of 0 V ~ V_{DD}. The accuracy of A/D conversion can be improved by using a circuit that shifts and scales the transducer output so that it falls in the full range of 0 V ~ V_{DD}.

An OP AMP circuit that can shift and scale the transducer output is shown in Figure 12.7c. This circuit consists of a summing circuit (Figure 12.7a) and an inverting circuit (Figure 12.7b). The voltage V_{IN} comes from the transducer output, whereas V_1 is an adjusting voltage. By choosing appropriate values for V_1 and resistors R_1, R_2, and R_f, the desired voltage shifting and scaling can be achieved.

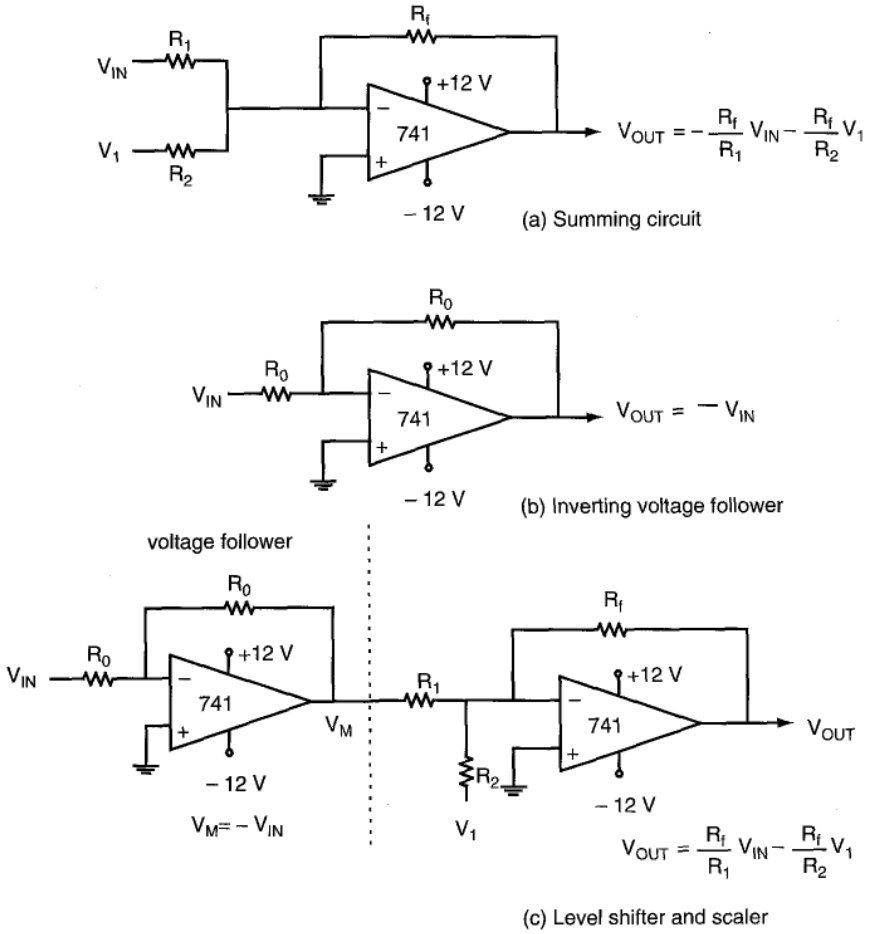

Figure 12.7 ■ Level shifting and scaling circuit

Example 12.3

Choose appropriate resistor values and the adjusting voltage so that the circuit shown in Figure 12.7c can shift the voltage from the range of –1.2 V ~ 3.0 V to the range of 0 V ~ 5 V.

Solution: Applying Equation 12.5,

$0 = -1.2 \times (R_f/R_1) - (R_f/R_2) \times V_1$

$5 = 3.0 \times (R_f/R_1) - (R_f/R_2) \times V_1$

By choosing $R_0 = R_1 = 10$ K, $R_2 = 50$ K, $R_f = 12$ K, and $V_1 = -5$ V, one can translate and scale the voltage to the desired range. This example tells us that the selection of resistors and the voltage V_1 is a trial-and-error process at best.

12.3 The PIC18 A/D Converter

A PIC18 microcontroller has a 10-bit A/D converter and may have from 5 to 16 analog inputs. The assignment of analog inputs and pins can be found in the datasheet of the appropriate PIC18 microcontroller.

12.3.1 Registers Associated with A/D Converter

The A/D converter has five registers that control the overall operation of the module:

- A/D control register 0 (ADCON0)
- A/D control register 1 (ADCON1)
- A/D control register 2 (ADCON2)
- A/D result high register (ADRESH)
- A/D result low register (ADRESL)

ADCON0 Register

The contents of the ADCON0 register are shown in Figures 12.8a, 12.8b, and 12.8c. Earlier PIC18 members (e.g., PIC18FXX2 and PIC18FXX8) have only two control registers, whereas later PIC18 members (e.g., PIC18FXX20, PIC18F8X2X, PIC18F6X2X, PIC18F6X8X, and PIC18F8X8X) have three control registers.

The contents of ADCON0 for the PIC1320/1220 are shown in Figure 12.8c.

	7	6	5	4	3	2	1	0
	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	--	ADON
value after reset	0	0	0	0	0	0	0	0

ADCS1:ADCS0: A/D conversion clock select bits
(used along with ADCS2 of the ADCON1 register (shown in Table 12.1)
CHS2:CHS0: Analog channel select bits
000 = channel 0, (AN0)
001 = channel 1, (AN1)
010 = channel 2, (AN2)
011 = channel 3, (AN3)
100 = channel 4, (AN4)
101 = channel 5, (AN5)
110 = channel 6, (AN6)
111 = channel 7, (AN7)
GO/$\overline{\text{DONE}}$: A/D conversion status bit
when ADON = 1
0 = A/D conversion not in progress
1 = A/D conversion in progress (setting this bit starts the A/D conversion.
This bit will be cleared by hardware when A/D conversion is done)
ADON: A/D on bit
0 = A/D converter module is shut-off
1 = A/D converter module is powered up

Figure 12.8a ■ ADCON0 register (PIC18FXX2 and PIC18FXX8) (redraw with permission of Microchip)

ADCS2: ADCS0	Clock conversion
000	FOSC/2
001	FOSC/8
010	FOSC/32
011	FRC (clock derived from RC oscillator)
100	FOSC/4
101	FOSC/16
110	FOSC/64
111	FRC (clock derived from RC oscillator)

Table 12.1 ■ A/D conversion clock source select bits

	7	6	5	4	3	2	1	0
	--	--	CHS3	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
value after reset	0	0	0	0	0	0	0	0

CHS3:CHS0: Analog channel select bits
- 0000 = channel 0, (AN0)
- 0001 = channel 1, (AN1)
- 0010 = channel 2, (AN2)
- 0011 = channel 3, (AN3)
- 0100 = channel 4, (AN4)
- 0101 = channel 5, (AN5)
- 0110 = channel 6, (AN6)
- 0111 = channel 7, (AN7)
- 1000 = channel 8, (AN8)
- 1001 = channel 9, (AN9)
- 1010 = channel 10, (AN10)
- 1011 = channel 11, (AN11)
- 1100 = channel 12, (AN12)
- 1101 = channel 13, (AN13)
- 1110 = channel 14, (AN14)
- 1111 = channel 15, (AN15)

GO/$\overline{\text{DONE}}$: A/D conversion status bit

when ADON = 1
- 0 = A/D conversion not in progress
- 1 = A/D conversion in progress (setting this bit starts the A/D conversion. This bit will be cleared by hardware when A/D conversion is done)

ADON: A/D on bit
- 0 = A/D converter module is shut-off
- 1 = A/D converter module is powered up

Figure 12.8b ■ ADCON0 register (PIC18FXX20/PIC18FXX80/PIC18FXX85) (redraw with permission of Microchip)

	7	6	5	4	3	2	1	0
	VCFG1	VCFG0	--	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
value after reset	0	0	0	0	0	0	0	0

VCFG1:VCFG0: Voltage reference configuration bits
(See Table 12.2)
CHS2:CHS0: Analog channel select bits
000 = channel 0, (AN0)
001 = channel 1, (AN1)
010 = channel 2, (AN2)
011 = channel 3, (AN3)
100 = channel 4, (AN4)
101 = channel 5, (AN5)
110 = channel 6, (AN6)
111 = channel 7, (AN7)
GO/$\overline{\text{DONE}}$: A/D conversion status bit
when ADON = 1
0 = A/D conversion not in progress
1 = A/D conversion in progress (setting this bit starts the A/D conversion.
This bit will be cleared by hardware when A/D conversion is done)
ADON: A/D on bit
0 = A/D converter module is shut-off
1 = A/D converter module is powered up

Figure 12.8c ■ ADCON0 register (PIC18F1220/1320) (redraw with permission of Microchip)

VCFG1:VCFG0	A/D V_{REF}+	A/D V_{REF}−
00	AV_{DD}	AV_{SS}
01	External V_{REF}+	AV_{SS}
10	AV_{DD}	External V_{REF}−
11	External V_{REF}+	External V_{REF}−

Table 12.2 ■ Voltage reference configuration bits

The user selects the analog input channel to be converted and the clock source to control the conversion process by programming the ADCON0 register. The ADON bit must be set to 1 in order to enable the A/D module, whereas the GO/$\overline{\text{DONE}}$ bit actually starts the conversion. The GO/$\overline{\text{DONE}}$ bit will be cleared automatically once the A/D conversion is done. The user has the option to use the power supply or the external supplied voltages as the high reference voltage and the low reference voltage.

ADCON1 Register

This register is used mainly to configure I/O pins as analog or digit port pins. The contents of this register are shown in Figures 12.9a, 12.9b, and 12.9c.

When a pin is connected to a signal to be converted, the pin must be configured as an analog input. Otherwise, the pin will be read as 0 and cannot serve the purpose of analog signal.

The ADFM bit in Figure 12.9a allows the user to select the result format to be left or right justified to better match the requirement of a specific application.

	7	6	5	4	3	2	1	0
	ADFM	ADCS2	--	--	PCFG3	PCFG2	PCFG1	PCFG0
value after reset	0	0	0	0	0	0	0	0

ADFM: A/D result format select bit

0 = left justified. Six least significant bits of ADRESL are 0s.

1 = right justified. Most significant bits of ADRESH are 0s.

ADCS2: A/D conversion clock select.

This bit along with the ADCS1:ADCS0 bits of ADCON0 are used to select clock source for A/D conversion.

PCFG3:PCFG0: A/D port configuration control bits.

(see Table 12.3)

Figure 12.9a ■ ADCON1 register (PIC18FXX2 and PIC18FXX8) (redraw with permission of Microchip)

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	V_{REF+}	V_{REF-}	C/R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2/1
011x	D	D	D	D	D	D	D	D	–	–	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

A = analog input D = digital input

C/R = # of analog input channels/# of A/D voltage references

Table 12.3 ■ A/D port configuration control bits

	7	6	5	4	3	2	1	0
	--	--	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
value after reset	0	0	0	0	0	0	0	0

VCFG1:VCFG0: Voltage reference configuration bits
(see Table 12.2)
PCFG3:PCFG0: A/D port configuration control bits

	AN15	AN14	AN13	AN12	AN11	AN10	AN9	AN8	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0
0000	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	D	D	A	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	D	D	D	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	D	D	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	D	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	D	D	D	A	A	A	A	A	A	A	A	A
0111	D	D	D	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D

1. AN15:AN12 are available only in PIC18F8X8X devices
2. AN12 is also available in PIC18F2X20/PIC18F4X20 devices
3. AN5 through AN7 are not available in PIC18F2X20 devices

Figure 12.9b ■ ADCON1 register (PIC18FXX20/PIC18FXX80/PIC18FXX85) (excluding PIC18F1320/1220) (redraw with permission of Microchip)

	7	6	5	4	3	2	1	0
	--	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0
value after reset	0	0	0	0	0	0	0	0

PCFG6..PFCG0: AN6..AN0 A/D port configuration bit
0 = Pin configured as an analog channel -- digital input disabled and read as 0
1 = Pin configured as a digital input

Figure 12.9c ■ ADCON1 register (PIC18F1220/1320) (redraw with permission of Microchip)

ADCON2 Register

This register is not available in earlier devices, such as PIC18FXX2 or PIC18FXX8. The main function of this register is to allow the user to program the A/D conversion clock source and A/D acquisition time. The newer PIC18 devices, such as PIC18F8X8X and PIC18F2X20/4X20, allow the user to program the data acquisition time to reduce the software overhead. For other devices that do not allow the user to program the data acquisition time, the user will need to use software delay to provide the required data acquisition time. To select an appropriate value for the data acquisition time, one needs to take clock source into consideration at the same time. The contents of this register are shown in Figures 12.10a and 12.10b.

	7	6	5	4	3	2	1	0
	ADFM	--	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
value after reset	0	0	0	0	0	0	0	0

ADFM: A/D result format select bit
- 0 = left justified
- 1 = right justified

ACQT2:ACQT0: A/D acquisition time select bits
- 000 = 0 TAD(1)
- 001 = 2 TAD
- 010 = 4 TAD
- 011 = 6 TAD
- 100 = 8 TAD
- 101 = 12 TAD
- 110 = 16 TAD
- 111 = 20 TAD

ADCS2:ADCS0: A/D conversion clock select bits
- 000 = FOSC/2
- 001 = FOSC/8
- 010 = FOSC/32
- 011 = FRC (clock derived from A/D RC oscillator)
- 100 = FOSC/4
- 101 = FOSC/16
- 110 = FOSC/64
- 111 = FRC (clock derived from A/D RC oscillator)

Note 1: If the A/D FRC clock source is selected, a delay of one TCY (instruction cycle) is added before the A/D clock starts. This allows the SLEEP instruction to be executed before starting a conversion.

Figure 12.10a ■ ADCON2 register (PIC18F8X8X/8X2X/6X2X/2X20/4x20/1220/1320) (redraw with permission of Microchip)

	7	6	5	4	3	2	1	0
	ADFM	--	--	--	--	ADCS2	ADCS1	ADCS0
value after reset	0	0	0	0	0	0	0	0

ADFM: A/D result format select bit
0 = left justified
1 = right justified
ADCS2:ADCS0: A/D conversion clock select bits
000 = FOSC/2
001 = FOSC/8
010 = FOSC/32
011 = FRC (clock derived from an RC oscillator = 1 MHz max)
100 = FOSC/4
101 = FOSC/16
110 = FOSC/64
111 = FRC (clock derived from an RC oscillator = 1 MHz)

Figure 12.10b ■ ADCON2 register (PIC18F8720/8620/8520/6720/6620/6520) (redraw with permission of Microchip)

12.3.2 A/D Acquisition Requirements

In order to ensure that the analog input voltage is stable during the A/D conversion process, the PIC18 microcontroller has a sample-and-hold circuit at the input. The analog input model is shown in Figure 12.11. The capacitor C_{HOLD} holds the voltage to be converted into digital code. For the A/D converter to meet its specified accuracy, the charge-holding capacitor (C_{HOLD}) must be allowed to fully charge to the input channel voltage level. The source impedance (R_S) and the internal sampling switch impedance (R_{SS}) directly affect the time required to charge the capacitor C_{HOLD}. The sampling switch impedance varies over the device voltage (V_{DD}). The source impedance affects the offset voltage at the analog input (because of pin leakage current). The maximum recommended impedance for analog sources is 2.5 KΩ. The voltage acquisition must be done before the conversion is started.

The minimum *acquisition time* (T_{ACQ}) is computed by the following equation:

T_{ACQ} = amplifier settling time + holding capacitor charging time + temperature coefficient
$= T_{AMP} + T_C + T_{COFF}$

The *amplifier settling time* (T_{AMP}) is about 2 μs. The *temperature coefficient* is given by the following equation:

$T_{COFF} = (\text{temp} - 25°\text{C}) \times (0.05\ \mu\text{s}/°\text{C})$.

Temperature coefficient is equal to 0 at temperatures below 25° C.

The *holding capacitor charging time* is given by the following equation:

$T_C = -120\ \text{pF} \times (1\ \text{K} + R_{SS} + R_S) \log(1/2047)$
$= 120\ \text{pF} \times (1\ \text{K}\Omega + R_{SS} + R_S) \log(2047)$

Since R_S = 2.5 K and R_{SS} is 7 K in Figure 12.11, T_C is computed to be 9.61 μs. The acquisition time at 50° C is calculated to be 12.86 μs.

For earlier PIC18 members, when the GO/$\overline{\text{DONE}}$ bit is set, sampling is stopped, and a conversion begins. The user is responsible for ensuring that the required acquisition time has passed between selecting the desired input channel and setting the GO/$\overline{\text{DONE}}$ bit.

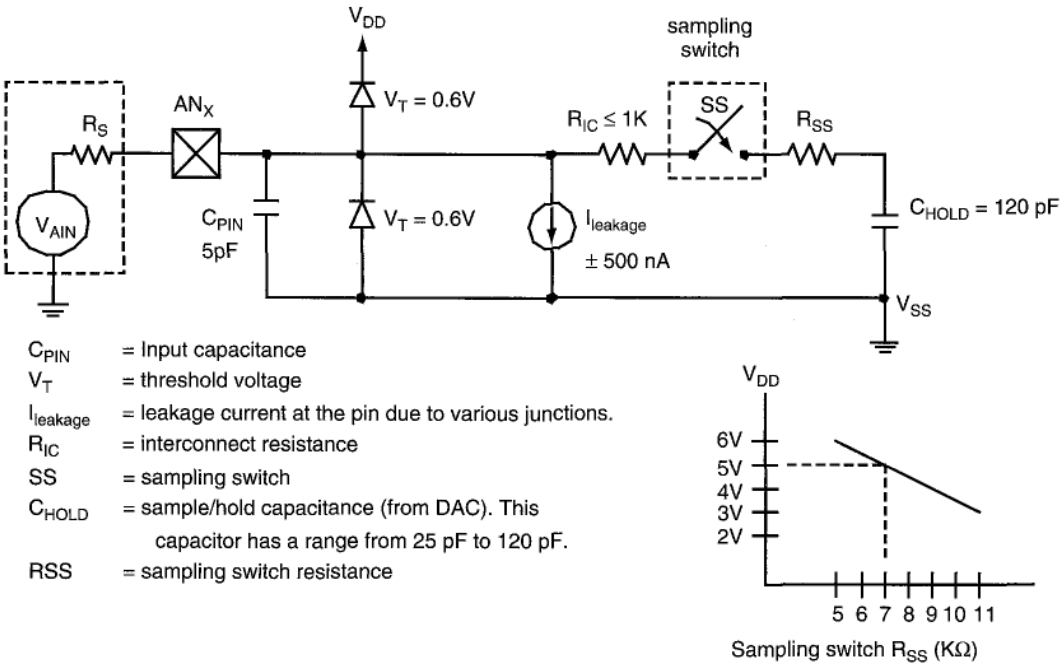

Figure 12.11 ■ Analog input circuit model (redraw with permission of Microchip)

For those devices that have automatic acquisition time, if ACQT2:ACQT0 bits (ADCON2<5:3>) are programmed to be a nonzero value, the A/D module will continue to sample after the GO/$\overline{\text{DONE}}$ bit is set for the selected acquisition time and then automatically begins a conversion. Once the conversion is started, the holding capacitor (C_{HOLD} in Figure 12.11) will be disconnected from the analog input. The holding capacitor will be reconnected to the analog input at the end of conversion. This feature will be very useful if the user wants to collect samples continuously. Since the acquisition time is programmed, there is no need to wait for an acquisition time between selecting a channel and setting the GO/$\overline{\text{DONE}}$ bit.

12.3.3 Selecting the A/D Conversion Clock

The A/D conversion time per bit is defined as T_{AD}. Each A/D conversion takes 12 T_{AD} for 10-bit resolution. There are seven possible options for T_{AD}:

- 2 T_{OSC}
- 4 T_{OSC}
- 8 T_{OSC}
- 16 T_{OSC}
- 32 T_{OSC}
- 64 T_{OSC}
- Internal RC oscillator period

For correct A/D conversion, the A/D conversion clock (T_{AD}) must be selected to be no smaller than 1.6 μs.

Table 12.4 shows the resultant T_{AD} times derived from the device operating frequencies and the A/D clock source selected. The RC clock source is used mainly in sleep mode. When the device is in sleep mode, all clock signals are stopped. Therefore, RC clock source is the only available clock source for the A/D converter.

AD clock source (T_{AD})		maximum device frequency	
Operation	**ADCS2:ADCS0**	**Normal MCU**	**Low power MCU**
2 T_{OSC}	000	1.25 MHz	666 KHz
4 T_{OSC}	100	2.50 MHz	1.33 MHz
8 T_{OSC}	001	5.00 MHz	2.66 MHz
16 T_{OSC}	101	10.0 MHz	5.33 MHz
32 T_{OSC}	010	20.0 MHz	10.65 MHz
64 T_{OSC}	110	40.0 MHz	21.33 MHz
RC	x11	1.00 MHz	1.00 MHz
Note: Lower power device has a letter L in its name. For example, PIC18LF8720			

Table 12.4 ■ T_{AD} vs. device operating frequencies

12.3.4 A/D Conversion Process

Figure 12.12 shows the operation of the A/D converter after the GO bit has been set and the ACQT2:ACQT0 bits are cleared. A conversion is started after the following instruction to allow entry into sleep mode before the conversion begins.

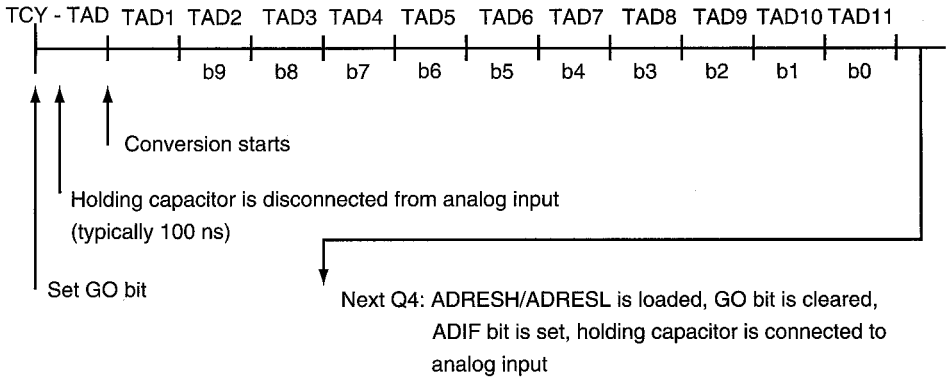

Figure 12.12 ■ A/D conversion T_{AD} cycles (ACQT<2:0> = 000, T_{ACQ} = 0)
(redraw with permission of Microchip)

Figure 12.13 shows the operation of the A/D converter after the GO bit has been set, the ACQT2:ACQT0 bits have been set to 010, and a 4-T_{AD} acquisition time has been set before the conversion starts.

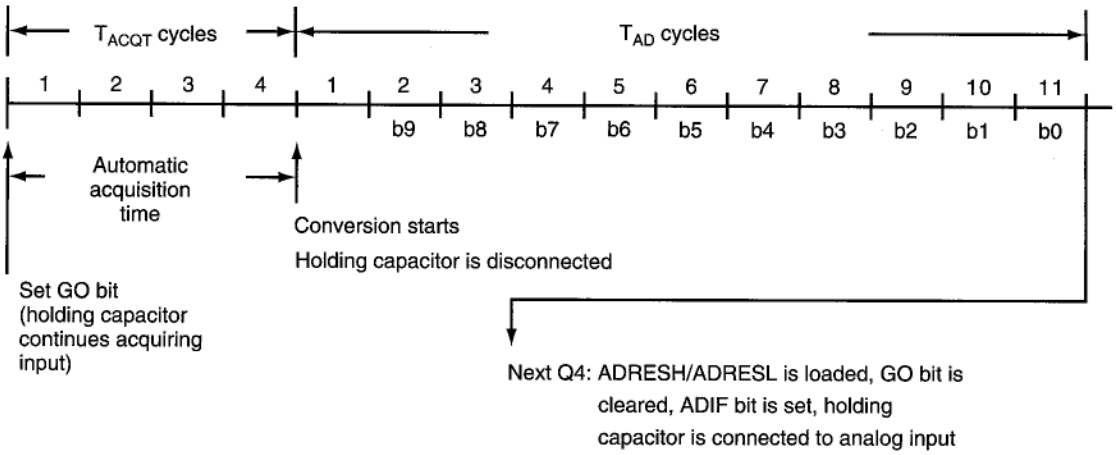

Figure 12.13 ■ A/D conversion T_{AD} cycles (ACQT<2:0> = 010, $T_{ACQ} = 4T_{AD}$) (redraw with permission of Microchip)

Clearing the GO/$\overline{\text{DONE}}$ bit during a conversion will abort the current conversion. The A/D result register pair will not be updated with the partially completed A/D conversion sample. This means that the ADRESH:ADRESL registers will continue to contain the value of the last completed conversion.

After the A/D conversion is completed or aboted, a minimum wait time of 2 T_{AD} is required before the next acquisition can be started. After this wait, acquisition on the selected channel is automatically started. The GO/$\overline{\text{DONE}}$ bit can then be set to start the conversion.

12.3.5 Use of the CCP2 Register

An A/D conversion can be started by the special event trigger of the CCP2 module. This requires that the CCP2M3:CCP2M0 bits (CCP2CON<3:0>) be programmed to "1011" and that the A/D module is enabled. When the trigger occurs, the GO/$\overline{\text{DONE}}$ bit will be set, starting the A/D conversion, and the Timer1 (or Timer3) counter will be reset to zero. Timer1 (or Timer3) is reset to automatically repeat the A/D acquisition period with minimal software overhead. The appropriate analog input channel must be selected and the minimum acquisition done before the special event trigger sets the GO/$\overline{\text{DONE}}$ bit.

If the A/D module is not enabled (ADON bit is cleared), the special event trigger will be ignored by the A/D module but will still reset the Timer1 (or Timer3) counter.

Example 12.4

Write an instruction sequence to configure the A/D converter of the PIC18F8680 to operate with the following parameters:

- Conversion result right justified
- f_{OSC} = 32 MHz
- Highest ambient temperature may reach 60° C
- Use V_{DD} and V_{SS} as the high and low reference voltages
- Convert channel AN0
- Enable A/D module

Solution: The temperature coefficient for the input circuit is (60 – 25) × 0.05 = 1.75 μs. The required acquisition time is (9.61 + 2 + 1.75) μs = 13.36 μs.

By setting the clock source to 64 f_{OSC}, T_{AD} is calculated to be 2 μs. The data acquisition time must be at least 8 T_{AD} to make it greater than 13.36 μs. Since channel AN0 is the channel to be converted, it must be configured for analog input.

The following instruction sequence will configure the A/D module of the PIC18F8680 as specified:

```
movlw 0x01          ; select channel AN0 and enable A/D
movwf ADCON0,A      ; "
movlw 0x0E          ; configure only channel AN0 as analog port,
movwf ADCON1,A      ; select VDD and VSS as reference voltage
movlw 0xA6          ; set A/D result right justified, set acquisition
movwf ADCON2,A      ; time to 8 TAD, clock source FOSC/64
```

In C language, the following statements will achieve the same configuration:

```
ADCON0 = 0x01;
ADCON1 = 0x0E;
ADCON2 = 0xA6;
```

Example 12.5

Write an instruction sequence to configure the A/D converter of the PIC18F452 to operate with the following parameters:

- Conversion result right justified
- f_{OSC} = 32 MHz
- Highest ambient temperature may reach 60° C
- Use V_{DD} and V_{SS} as the high and low reference voltages
- Convert channel AN0
- Enable A/D module

Solution: There are only two control registers to be programmed for the PIC18F452:

```
movlw 0x81          ; select FOSC/64 as conversion clock source,
movwf ADCON0,A      ; select channel AN0, enable A/D module
movlw 0xCE          ; A/D conversion result right justified, configure
movwf ADCON1,A      ; only channel AN0 as analog, VDD & VSS as
                    ; reference voltages for conversion
```

12.4 Procedure for Performing A/D Conversion

Before the PIC18 can perform A/D conversion, the A/D module must be connected and configured properly. The procedure for performing an A/D conversion is as follows:

1. Configure the A/D module:
   - Configure the analog pins, reference voltages, and digital/analog I/O pins
   - Select the A/D input channel
   - Select the A/D acquisition time (if available)

- Select the A/D conversion clock source
- Enable the A/D module

2. Configure the A/D interrupt (if desired):
   - Clear the ADIF bit
   - Set the ADIE bit
   - Set the GIE bit
3. Wait for the desired acquisition time (if required).
4. Start the conversion by setting the GO/$\overline{\text{DONE}}$ bit.
5. Wait for the A/D conversion to complete by either waiting for the GO/$\overline{\text{DONE}}$ (DONE) bit to be cleared or waiting for the A/D interrupt.
6. Read the A/D result registers; clear the ADIF flag.
7. For the next conversion, go to Step 1 or Step 2 as required. The A/D conversion time per bit is defined as T_{AD}. A minimum waiting time of 2 T_{AD} is required before next acquisition starts.

Example 12.6

Assume that the AN0 pin of a PIC18F8680 running with a 32-MHz crystal oscillator is connected to a potentiometer. The voltage range of the potentiometer is from 0 V to 5 V. Write a program to measure the voltage applied to the AN0 pin, convert it, retrieve the conversion result, and place it in PRODH:PRODL.

Solution: The program that follows the procedure described earlier is as follows:

```
            #include    <p18F8680.inc>
            org         0x00
            goto        start
            org         0x08
            retfie
            org         0x18
            retfie
start       movlw       0x01            ; select channel AN0 and enable A/D
            movwf       ADCON0,A        ; "
            movlw       0x0E            ; use VDD & VSS as reference voltages &
            movwf       ADCON1,A        ; configure channel AN0 as analog input
            movlw       0xA6            ; select FOSC/64 as conversion clock,
            movwf       ADCON2,A        ; 8 TAD for acquisition time, right-justified
            bsf         ADCON0,GO,A     ; start A/D conversion
wait_con    btfsc       ADCON0,DONE,A   ; wait until conversion is done
            bra         wait_con
            movff       ADRESH,PRODH    ; save conversion result
            movff       ADRESL,PRODL    ; "
            end
```

12.5 Microchip A/D Converter C Library Functions

The A/D converter is supported with six library functions. One needs to include the **adc.h** header file in order to use these library functions. The prototype declarations of these functions are as follows:

char **BusyADC** (void);

This function checks whether the A/D module is currently performing a conversion. The return value is 1 if the A/D module is busy performing a conversion. Otherwise, the return value is 0.

void **CloseADC** (void);

This function disables the A/D converter and A/D interrupt mechanism.

void **ConvertADC** (void);

This function starts an A/D conversion. The **BusyADC( )** function may be used to detect completion of the conversion.

void **OpenADC** (unsigned char **config,** unsigned char **config2**);

For the PIC18F1X20/2X20/4X20, the **OpenADC** function has a third argument **portconfig.** Because different PIC18 members may have different number of control registers, the setting of the parameters **config** and **config2** may be different:

PIC18FXX2, PIC18FXX2, and **PIC18FXX8**

The first argument, **config,** may have the following values:

A/D clock source	
ADC_FOSC_2	$F_{OSC}/2$
ADC_FOSC_4	$F_{OSC}/4$
DC_FOSC_8	$F_{OSC}/8$
ADC_FOSC_16	$F_{OSC}/16$
ADC_FOSC_32	$F_{OSC}/32$
ADC_FOSC_64	$F_{OSC}/64$
ADC_FOSC_RC	Internal RC oscillator
A/D result justification	
ADC_RIGHT_JUST	Result in least significant bits
ADC_LEFT_JUST	Result in most significant bits
A/D voltage reference source	
ADC_8ANA_0REF	$V_{REF+} = V_{DD}$, $V_{REF-} = V_{SS}$, all analog channels
ADC_7ANA_1REF	AN3 = V_{REF+}, all analog channels except AN3
ADC_6ANA_2REF	AN3 = V_{REF+}, AN2 = V_{REF-}
ADC_6ANA_0REF	$V_{REF+} = V_{DD}$, $V_{REF-} = V_{SS}$
ADC_5ANA_1REF	AN3 = V_{REF+}, $V_{REF-} = V_{SS}$
ADC_5ANA_0REF	$V_{REF+} = V_{DD}$, $V_{REF-} = V_{SS}$
ADC_4ANA_2REF	AN3 = V_{REF+}, AN2 = V_{REF-}
ADC_4ANA_1REF	AN3 = V_{REF+}

ADC_3ANA_2REF	AN3 = V_{REF+}, AN2 = V_{REF-}
ADC_3ANA_0REF	V_{REF+} = V_{DD}, V_{REF-} = V_{SS}
ADC_2ANA_2REF	AN3 = V_{REF+}, AN2 = V_{REF-}
ADC_2ANA_1REF	AN3 = V_{REF+}
ADC_1ANA_2REF	AN3 = V_{REF+}, AN2 = V_{REF-}, AN0 = A
ADC_1ANA_0REF	AN0 is analog input
ADC_0ANA_0REF	All digital I/O

The second argument, **config2,** may have the following values:

Channel	
ADC_CH0	Channel 0
ADC_CH1	Channel 1
ADC_CH2	Channel 2
ADC_CH3	Channel 3
ADC_CH4	Channel 4
ADC_CH5	Channel 5
ADC_CH6	Channel 6
ADC_CH7	Channel 7
A/D Interrupts	
ADC_INT_ON	Interrupts enabled
ADC_INT_OFF	Interrupts disabled

PIC18C658/858, PIC18C601/801, PIC18F6X20, PIC18F8X20

The first argument, **config,** may have the following values:

A/D clock source		
ADC_FOSC_2	$F_{OSC}/2$	
ADC_FOSC_4	$F_{OSC}/4$	
ADC_FOSC_8	$F_{OSC}/8$	
ADC_FOSC_16	$F_{OSC}/16$	
ADC_FOSC_32	$F_{OSC}/32$	
ADC_FOSC_64	$F_{OSC}/64$	
ADC_FOSC_RC	Internal RC oscillator	
A/D result justification		
ADC_RIGHT_JUST	Result in least significant bits	
ADC_LEFT_JUST	Result in most significant bits	
A/D port configuration		
ADC_0ANA	All digital	
ADC_1ANA	Analog AN0	Digital AN1-AN15
ADC_2ANA	Analog AN0-AN1	Digital AN2-AN15
ADC_3ANA	Analog AN0-AN2	Digital AN3-AN15
ADC_4ANA	Analog AN0-AN3	Digital AN4-AN15

ADC_5ANA	Analog AN0-AN4	Digital AN5-AN15
ADC_6ANA	Analog AN0-AN5	Digital AN6-AN15
ADC_7ANA	Analog AN0-AN6	Digital AN7-AN15
ADC_8ANA	Analog AN0-AN7	Digital AN8-AN15
ADC_9ANA	Analog AN0-AN8	Digital AN9-AN15
ADC_10ANA	Analog AN0-AN9	Digital AN10-AN15
ADC_11ANA	Analog AN0-AN10	Digital AN11-AN15
ADC_12ANA	Analog AN0-AN11	Digital AN12-AN15
ADC_13ANA	Analog AN0-AN12	Digital AN13-AN15
ADC_14ANA	Analog AN0-AN13	Digital AN14-AN15
ADC_15ANA	All analog	

The second argument, **config2,** has the following values:

Channel	
ADC_CH0	Channel 0
ADC_CH1	Channel 1
ADC_CH2	Channel 2
ADC_CH3	Channel 3
ADC_CH4	Channel 4
ADC_CH5	Channel 5
ADC_CH6	Channel 6
ADC_CH7	Channel 7
ADC_CH8	Channel 8
ADC_CH9	Channel 9
ADC_CH10	Channel 10
ADC_CH11	Channel 11
ADC_CH12	Channel 12
ADC_CH13	Channel 13
ADC_CH14	Channel 14
ADC_CH15	Channel 15
A/D Interrupts	
ADC_INT_ON	Interrupts enabled
ADC_INT_OFF	Interrupts disabled
A/D voltage configuration	
ADC_VREFPLUS_VDD	$V_{REF+} = AV_{DD}$
ADC_VREFPLUS_EXT	V_{REF+} = external
ADC_VREFMINUS_VSS	$V_{REF-} = AV_{SS}$
ADC_VREFMINUS_EXT	V_{REF-} = external

PIC18F6X8X, PIC18F8X8X, PIC18F1X20, PIC18F2X20, PIC18F4X20

The first argument, **config,** may have the following values:

A/D clock source	
ADC_FOSC_2	$F_{OSC}/2$
ADC_FOSC_4	$F_{OSC}/4$
ADC_FOSC_8	$F_{OSC}/8$
ADC_FOSC_16	$F_{OSC}/16$
ADC_FOSC_32	$F_{OSC}/32$
ADC_FOSC_64	$F_{OSC}/64$
ADC_FOSC_RC	Internal RC oscillator
A/D result justification	
ADC_RIGHT_JUST	Result in least significant bits
ADC_LEFT_JUST	Result in most significant bits
A/D acquisition time select	
ADC_0_TAD	0 T_{AD}
ADC_2_TAD	2 T_{AD}
ADC_4_TAD	4 T_{AD}
ADC_6_TAD	6 T_{AD}
ADC_8_TAD	8 T_{AD}
ADC_12_TAD	12 T_{AD}
ADC_16_TAD	16 T_{AD}
ADC_20_TAD	20 T_{AD}

The second argument, **config2,** has the following values:

Channel	
ADC_CH0	Channel 0
ADC_CH1	Channel 1
ADC_CH2	Channel 2
ADC_CH3	Channel 3
ADC_CH4	Channel 4
ADC_CH5	Channel 5
ADC_CH6	Channel 6
ADC_CH7	Channel 7
ADC_CH8	Channel 8
ADC_CH9	Channel 9
ADC_CH10	Channel 10
ADC_CH11	Channel 11
ADC_CH12	Channel 12
ADC_CH13	Channel 13
ADC_CH14	Channel 14
ADC_CH15	Channel 15

A/D Interrupts	
ADC_INT_ON	Interrupts enabled
ADC_INT_OFF	Interrupts disabled
A/D voltage configuration	
ADC_VREFPLUS_VDD	$V_{REF+} = AV_{DD}$
ADC_VREFPLUS_EXT	V_{REF+} = external
ADC_VREFMINUS_VSS	$V_{REF-} = AV_{SS}$
ADC_VREFMINUS_EXT	V_{REF-} = external

The third argument, **Portconfig,** is any value from 0 to 127 for the PIC18F1220/1320 and 0 to 15 for the PIC18F2220/2320/4220/4320/6X8X/8X8X, inclusive. This is the value of bits 0 through 6 or bits 0 through 3 of the ADCON1 register, which are the port configuration bits.

int **ReadADC** (void);

This function reads the result of an A/D conversion and returns it as a 16-bit signed value. Depending on the configuration, the conversion result may be right or left justified.

void **SetChanADC** (unsigned char **channel**);

This function selects the pin used as input to the A/D converter. The channel value can be ADC_CH0 through ADC_CH15.

Example 12.7

Write a C program to configure the A/D module of the PIC18F452 with the following characteristics and take one sample, convert it, and store the result in a memory location:

- Clock source set to $F_{OSC}/64$
- Result right justified
- Set port A AN0 pin for analog input, others for digital
- Use V_{DD} and V_{SS} as high and low reference voltages
- Select AN0 to convert
- Disable interrupt

Solution: The C program that performs the configuration, takes one sample, and performs the conversion is as follows:

```
#include <p18F452.h>
#include <adc.h>
#include <stdlib.h>
#include <delays.h>
int result;
void main (void)
{
   OpenADC(ADC_FOSC_64 & ADC_RIGHT_JUST & ADC_1ANA_0REF, ADC_CH0 & ADC_INT_OFF);
   Delay10TCYx(20);          // provides 200 instruction cycles of acquisition time
   ConvertADC( );            // start A/D conversion
   while(BusyADC( ));        // wait for completion
   result = ReadADC( );      // read result
   CloseADC( );
}
```

The statement **Delay10TCYx(20)** is not needed for those devices with programmable acquisition time.

12.6 Using the Temperature Sensor TC1047A

The TC1047A from Microchip is a three-pin temperature sensor whose voltage output is directly proportional to the measured temperature. The TC1047A can accurately measure temperatures from –40° C to 125° C with a power supply from 2.7 V to 5.5 V.

The output voltage range for these devices is typically 100 mV at –40° C, 500 mV at 0° C, 750 mV at +25° C, and +1.75 V at +125° C. As shown in Figure 12.14, the TC1047A has a 10 mV/°C voltage slope output response.

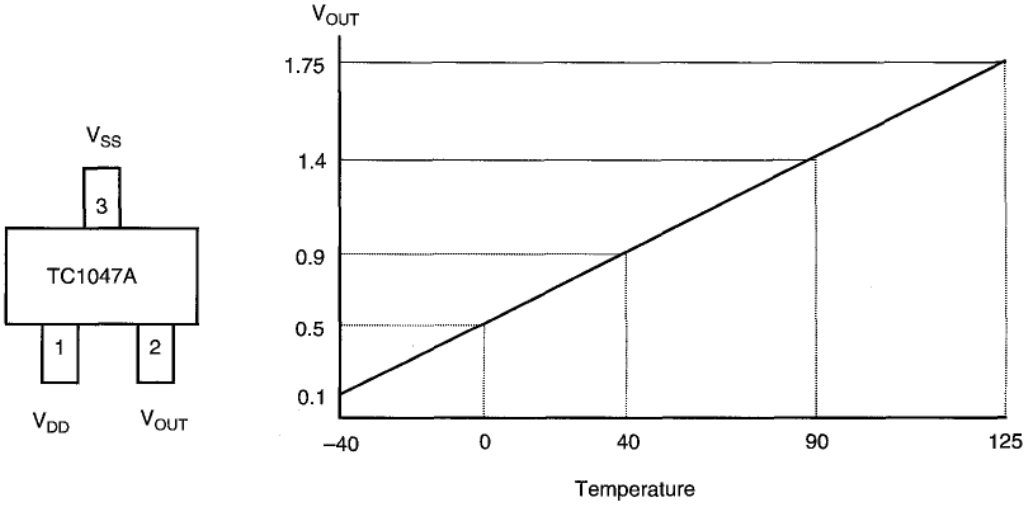

Figure 12.14 ■ TC1047A V_{OUT} vs. temperature characteristic

Example 12.8

Describe a circuit connection and the required program to build a digital thermometer. Display the temperature in three integral and one fractional digits using the LCD. Measure and display the temperature over the whole range of TC1047A, that is, –40° C to +125° C. Update the display data 10 times per second and assume that the PIC18F8680 operates with a 32-MHz crystal oscillator.

Solution: Since the voltage output of the TC1047A is from 0.1 V to 1.75 V for the temperature range of –40° C through 125° C, we will need to use the circuit shown in Figure 12.7c to perform the translation and scaling for V_{OUT}. The circuit connection is shown in Figure 12.15. This circuit will use V_{DD} and V_{SS} as the reference voltages required in the process of A/D conversion. The range of the voltage connected to the AN0 pin will be between 0 V and 5V for the given temperature range.

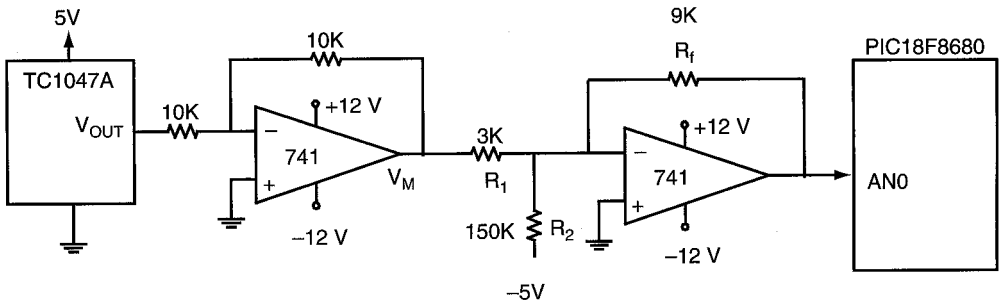

Figure 12.15 ■ Circuit connection between the TC1047A and the PIC18 MCU

To convert the A/D conversion result to the corresponding temperature, we need to do the following:

1. Divide the conversion result by 6.2
2. Subtract the quotient by 40

Since the PIC18 microcontroller cannot handle floating-point numbers directly, the operation of "dividing by 6.2" can be implemented by multiplying the conversion result by 10 and then dividing the product by 62.

The procedure for starting an A/D conversion, reading the conversion result, calculating the corresponding temperature, and converting the temperature value into an ASCII string every 200 ms is as follows:

Step 1
Configure the A/D converter and Timer0 properly. Timer0 is configured to overflow every 200 ms.

Step 2
Start an A/D conversion.

Step 3
Wait until the A/D conversion is complete.

Step 4
Multiply the conversion result by 10.

Step 5
Divide the product resulted in Step 4 by 62 to obtain the temperature reading. Use the variables **quo** and **rem** to hold the quotient and remainder.

Step 6
Subtract 40 from the variable **quo** to obtain the actual temperature.

Step 7

If **quo** > 0, go to Step 9; otherwise, replace quo with its two's complement.

Step 8
If **rem** ≠ 0, then

1. decrement **quo** by 1
2. rem ← 62 – rem

Step 9
Compute the fractional digit by multiplying **rem** by 10 and then dividing the resulting product by 62.

Step 10
Compute the integral digits by performing repeated division by 10 to **quo.**

Step 11
Wait until Timer0 overflows and then go to Step 2.
The assembly program that implements this algorithm is as follows:

```
#include <p18F8680.inc>
; ----
; include macro definitions for pushr, push_dat, popr, alloc_stk, dealloc_stk here.
; ----
;*****************************************************************
; The following definitions are used by the 16-bit multiplication routine
;*****************************************************************
loc_varm    equ     4           ; number of bytes used for local variable
pd0         equ     7           ; offset of PD3 from frame pointer
pd1         equ     6           ; offset of PD2 from frame pointer
pd2         equ     5           ; offset of PD1 from frame pointer
pd3         equ     4           ; offset of PD0 from frame pointer
MD_lo       equ     -8          ; offset of MD_lo from frame pointer
MD_hi       equ     -7          ; offset of MD_hi from frame pointer
ND_lo       equ     -6          ; offset of ND_lo from frame pointer
ND_hi       equ     -5          ; offset of ND_hi from frame pointer
buf_lo      equ     -4
buf_hi      equ     -3
ptr_hi      equ     0x01        ; address of buffer to hold product
ptr_lo      equ     0x00        ; "
;*****************************************************************
; The following definitions are used by the 16-bit unsigned divide routine
;*****************************************************************
loc_var     equ     2           ; local variable size
lp_cnt      equ     1           ; loop count
temp        equ     2           ; temporary storage
quo_hi      equ     -7          ; offset for quotient and dividend from frame
quo_lo      equ     -8          ; pointer
rem_hi      equ     -5          ; offset for remainder from frame pointer
rem_lo      equ     -6          ; "
dsr_hi      equ     -3          ; offset for divisor from frame pointer
dsr_lo      equ     -4          ; "
quo         set     0x02        ; memory location to hold the quotient
rem         set     0x04        ; memory location to hold the remainder
;*****************************************************************
; variables for holding temperature conversion result and string
;*****************************************************************
int_pt      set     0x06        ; space to hold integer part of the temperature
temp_buf    set     0x08        ; reserve 6 bytes to hold the temperature
                                ; digits, sign, period, and NULL
```

```
;************************************************************
; Reset vector is here.
;************************************************************
                org       0x00          ; reset vector
                goto      start
                org       0x08
                retfie
                org       0x18
                retfie
start           lfsr      FSR1,0xC00    ; set up stack pointer
                call      a2d_init      ; configure and turn on A/D module
; initialize the temperature string to ^^0.0
                movlw     0x20          ; store space character
                movwf     temp_buf      ; "
                movwf     temp_buf+1    ; "
                movlw     0x30          ; store a 0 digit
                movwf     temp_buf+2    ; "
                movwf     temp_buf+4    ; "
                movlw     0x2E          ; store a period
                movwf     temp_buf+3    ; "
                clrf      temp_buf+5    ; terminate the string with a NULL character
                call      OpenTmr0      ; initialize and enable Timer0
forever         bsf       ADCON0,GO,A ; start A/D conversion
wait_a2d        btfsc     ADCON0,DONE,A     ; wait until A/D conversion is complete
                bra       wait_a2d      ; "
                pushr     ADRESL        ; push the A/D conversion result in
                pushr     ADRESH        ; stack
                push_dat  0x0A          ; push 10 in the stack for multiplier
                push_dat  0x00          ; "
                push_dat  ptr_lo        ; pass buffer pointer to the subroutine
                push_dat  ptr_hi        ; "
                call      mul_16U,FAST; multiply A/D conversion result by 10
                dealloc_stk 6           ; deallocate space used in the stack
                movlw     ptr_lo        ; place the address of product in FSR0
                movwf     FSR0L,A       ; "
                movlw     ptr_hi        ; "
                movwf     FSR0H,A       ; "
                movf      POSTINC0,W,A; push (A/D result x 10)
                pushr     WREG          ; "
                movf      INDF0,W,A     ; "
                pushr     WREG          ; "
                alloc_stk 2
                push_dat  0x3E          ; push 62 into the stack as the divisor
                push_dat  0             ; "
                call      div16u,FAST   ;
                dealloc_stk 2
                popr      rem+1         ; retrieve remainder high byte (should be 0)
                popr      rem           ; retrieve remainder low byte
                popr      int_pt+1      ; retrieve integer part of the (should be 0)
                popr      int_pt        ; temperature
```

```
; subtract 40 from integer part to obtain the actual temperature
                movlw     0x28          ; calculate the actual temperature
                subwf     int_pt,F,A    ; "
                bnn       non_minus     ; if non-minus, no need for further check
                negf      int_pt,A      ; find the magnitude of temperature
                movlw     0x2D
                movlw     temp_buf      ; store the minus sign
                movf      rem,W,A       ; check the fractional part before divide
                bz        separate_dd   ; branch to separate integer digits if rem = 0
                decf      int_pt,F,A    ; fractional digit ≠ 0, decrement integer part
                movlw     0x3E          ; need to find the complement of the fractional
                subwf     rem,F,A       ; digit
                negf      rem,A         ; "
; calculate the fractional digit
non_minus
                movlw     0x0A          ; multiply the remainder by 10
                mulwf     rem           ; "
                pushr     PRODL         ; push (remainder × 10)
                pushr     PRODH         ; "
                alloc_stk 2
                push_dat  0x3E          ; push 62 into the stack as the divisor
                push_dat  0             ; "
                call      div16u,FAST   ;
                dealloc_stk 2
                popr      rem+1         ; retrieve remainder high byte (= 0)
                popr      rem           ; retrieve remainder low byte
                popr      quo+1         ; retrieve quotient high byte (= 0)
                popr      quo           ; retrieve quotient low byte
; round the fractional digit
                movlw     0x1F          ; is remainder >= 31?
                cpfslt    rem,A         ; smaller, then skip
                incf      quo,A
                movlw     0x0A          ; is quo equal to 10?
                cpfseq    quo,F,A
                goto      no_round
                clrf      quo,A
                incf      int_pt,A
no_round        movlw     0x30          ; convert to ASCII of BCD digit
                addwf     quo,F,A
                movwf     temp_buf+4    ; save the ASCII of the fractional digit
; separate the integral digits using repeated division by 10
separate_dd
                pushr     int_pt        ; push integer part
                push_dat  0             ; "
                alloc_stk 2
                push_dat  0x0A          ; push 10 as the divisor
                push_dat  0             ; "
                call      div16u,FAST
                dealloc_stk 2
```

```
                popr    rem+1
                popr    rem
                popr    quo+1
                popr    quo
                movlw   0x30
                addwf   rem,W,A
                movwf   temp_buf+2  ; save the one's digit
                movf    quo,W,A     ; check the quotient
                bz      next_time   ; wait to perform next conversion
; prepare to separate ten's digit
                movlw   0x0A        ; is the quotient >= 10?
                cpfslt  quo,A       ; "
                goto    yes_ge
                movlw   0x30
                addwf   quo,W,A
                movwf   temp_buf+1  ; save the ten's digit
                goto    next_time   ; "
yes_ge          movlw   0x31        ; save "1" as the hundred's digit
                movwf   temp_buf,A  ; "
                movlw   0x0A        ; separate the ten's digit and place it
                subwf   quo,W,A     ; in WREG
                addlw   0x30        ; convert ten's digit to ASCII and
                movwf   temp_buf+1,A; save ten's digit
next_time       btfss   INTCON,TMR0IF,A   ; wait until 200 ms is over
                goto    next_time
; ----
; add instructions to update display here
; ----
                goto    forever     ; prepare to perform next A/D conversion
; *****************************************************************
; This routine will place 15535 in TMR0 so that it overflows in 50000 count.
; When prescaler is set to 32 with fosc = 32MHz, it will overflow in 200 ms.
; *****************************************************************
OpenTmr0        movlw   0x3C        ; place 15535 in TMR0
                movwf   TMR0H       ; so that it overflows in
                movlw   0xAF        ; 200 ms
                movwf   TMR0L       ; "
                movlw   0x84        ; enable TMR0, select internal clock,
                movwf   T0CON       ; set prescaler to 32
                bcf     INTCON,TMR0IF     ; clear TMR0IF flag
                return
; *****************************************************************
; This routine initializes the A/D converter to select channel AN0 as
; analog input other pins for digital pin. Select VDD and VSS as A/D
; conversion reference voltages, result right justified, FOSC/64 as
; A/D clock source, 8 TAD for acquisition time.
; *****************************************************************
a2d_init        movlw   0x01        ; select channel AN0
                movwf   ADCON0      ; and enable A/D module
                movlw   0x0E        ; use VDD & VSS as A/D reference voltage
```

```
                movwf   ADCON1      ; & configure AN0 as analog others for digital
                movlw   0xA6        ; result right justified, FOSC/64 as A/D clock
                movwf   ADCON2      ; source and set acquisition time to 8 TAD
                return
;----
; include subroutines div16u and mul_16U here.
;----
                end
```

The C language version of the program is as follows:

```
#include    <p18F8680.h>
#include    <timers.h>
#include    <adc.h>
unsigned    char temp_buf[6];
void main (void)
{
    int a2d_val;
    unsigned int quo, rem;
    char fd1, fdr;
    ADCON0 = 0x01;          //select channel AN0, enable A/D module
    ADCON1 = 0x0E;          //use VDD, VSS as reference and configure AN0 for analog
    ADCON2 = 0xA6;          //result right justified, 8TAD acquisition time, FOSC/64
    OpenTimer0 (TIMER_INT_OFF & T0_16BIT & T0_SOURCE_INT & T0_PS_1_32); /*start Timer0 and make it
                overflow in 200 ms*/
    while (1) {
        temp_buf[5] = '\0';
        temp_buf[0] = 0x20; //set to space
        temp_buf[1] = 0x20; //set to space
        temp_buf[2] = 0x30; //set to digit 0
        temp_buf[3] = 0x2E; //store the decimal point
        temp_buf[4] = 0x30; //set to digit 0
        ConvertADC( );      //start an A/D conversion
        while(BusyADC());   //wait until A/D conversion is done
        a2d_val = 10 * ReadADC();
        quo = a2d_val / 62; //convert to temperature
        rem = a2d_val % 62;
        if (quo < 40)       //is temperature minus?
        {
            quo = 40 - quo;
            temp_buf[0] = 0x2D;             //set sign to minus
            if (rem != 0)
            {
                quo--;
                rem = 62 - rem;
            }
        }
        fd1 = (rem * 10) / 62;              //fd1 will be between 0 and 9
        fdr = (rem * 10) % 62;
        if (fdr >= 31)
            fd1 ++;
```

```
            if (fd1 == 10) {        //fractional digit can only be between 0 and 9
                quo++;
                fd1 = 0;
            }
            temp_buf[4] = 0x30 + fd1;          //store the ASCII code of fractional digit
            temp_buf[2] = quo % 10 + 0x30;     //store ASCII code of one's digit
            quo = quo / 10;
            if (quo != 0)
            {
                temp_buf[1] = (quo - 10) + 0x30;      //ten's digit of temperature
                quo /= 10;
            }
            if (quo == 1)
                temp_buf[0] = 0x31; //hundred's digit of temperature
            while(!INTCONbits.TMR0IF); //wait until Timer0 overflows
            INTCONbits.TMR0IF = 0; //clear the TMR0IF flag
        }
    }
```

12.7 Using the IH-3606 Humidity Sensor

The IH-3605 is a humidity sensor made by Honeywell. The IH-3605 humidity sensor provides a linear output from 0.8 V to 3.9 V in the full range of 0% to 100% *relative humidity* (RH) when powered by a 5-V power supply. It can operate in a range of 0% to 100% RH, –40° to 185° F.

The pins of the IH-3605 are shown in Figure 12.16. The specification of the IH-3605 is listed in Table 12.5. The IH-3605 is light sensitive and should be shielded from bright light for best results.

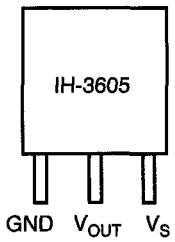

Figure 12.16 ■ Honeywell IH-3605 humidity sensor

Specification	Description
Total accuracy	± 2% RH, 0–100% TH @25° C
Interchangeability	± 5% RH up to 60% RH, ±8% RH at 90% RH
Operating temperature	-40 to 85° C (-40 to 185° F)
Storage temperature	-51 to 110° C (-60 to 223° F)
Linearity	±0.5% RH typical
Repeatability	±0.5% RH
Humidity Stability	±1% RH typical at 50% RH in 5 years
Temp. effect on 0% RH voltage	±0.007% RH° C (negligible)
Temp. effect on 100% RH voltage	-0.22% RH/° C
Output voltage	V_{OUT} = (V_S)(0.16 to 0.78) nominal relative to supply voltage for 0–100% RH; i.e., 1–4.9V for 6.3V supply; 0.8 - 3.9V for 5V supply; Sink capability 50 microamp; drive capability 5 microamps typical; low pass 1 KHz filter required. Turn on time < 0.1 sec to full output.
VS Supply requirement	4 to 9V, regulated or use output/supply ratio; calibrated at 5V
Current requirement	200 microamps typical @5V, increased to 2mA at 9V

Table 12.5 ■ Specifications of IH-3605

The IH-3605 can resist contaminant vapors, such as organic solvents, chlorine, and ammonia. It is unaffected by water condensate as well. Because of this capability, the IH-3605 has been used in refrigeration, drying, instrumentation, meteorology, and many other applications.

Example 12.9

Construct a humidity measurement system that consists of the PIC18F8680, an IH-3605 humidity sensor, and an LCD. The PIC18F8680 is running with a 32-MHz crystal oscillator.

Solution: Since the output of the IH-3605 is between 0.8 V and 3.9 V with a 5-V power supply, it would be beneficial to use a circuit to translate and then scale it to between 0 V and 5 V so that the best accuracy can be achieved. The circuit connection is shown in Figure 12.17. A set of resistor values and V1 voltage are given in Figure 12.17. A low-pass filter that consists of a 1-KΩ resistor and a 0.16-μF capacitor is added to meet the requirement. The user can use an LCD or four seven-segment displays to display the relative humidity.

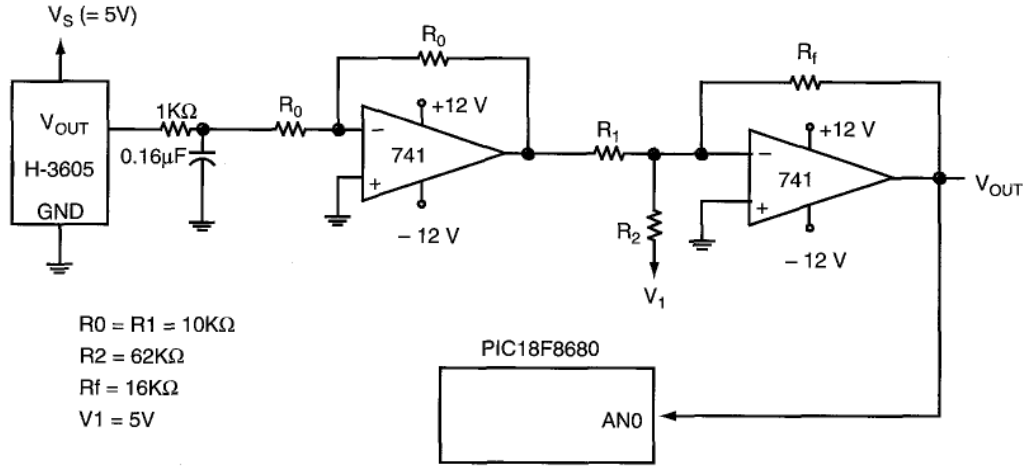

Figure 12.17 ■ Relative humidity measurement circuit

To translate from the conversion result to the relative humidity, divide the conversion result by 10.23. The following C program will configure the A/D module, start the A/D conversion, translate the conversion result to the relative humidity, and convert the humidity into an ASCII string to be output on the LCD.

The procedure for performing the measurement and display of relative humidity is similar to Example 12.8, and the C program that implements it is shown here:

```
#include    <p18F8680.h>
#include    <timers.h>
#include    <adc.h>
unsigned char hum_buf[6]; //buffer to hold relative humidity
void main (void)
{
   unsigned short long a2d_val;
```

```
    unsigned short long quo, rem, temp1;
    char fd1, i;
    ADCON0 = 0x01;                  //select channel AN0, enable A/D module
    ADCON1 = 0x0E;                  //use VDD, VSS as reference and configure AN0 for analog
    ADCON2 = 0xA6;                  //result right justified, acquisition time = 8 TAD, Fosc/64
    OpenTimer0 (TIMER_INT_OFF & T0_16BIT & T0_SOURCE_INT & T0_PS_1_32);
                //start Timer0 and make it overflow in 200 ms
    while (1) {
        hum_buf[0] = 0x20;          //set to space
        hum_buf[1] = 0x20;          //set to space
        hum_buf[2] = 0x30;          //set to digit 0
        hum_buf[3] = 0x2E;          //store the decimal point
        hum_buf[4] = 0x30;          //set to digit 0
        hum_buf[5] = '\0';          //terminate with a NULL character
        ConvertADC( );              //start an A/D conversion
        while (BusyADC( ));         //wait until A/D conversion is done
        a2d_val = 100 * ReadADC();
        quo = a2d_val / 1023;       //convert to relative humidity
        rem = a2d_val % 1023;       // "
        fd1 = (rem * 10) / 1023;    //compute the fractional digit
        temp1 = (rem * 10) % 1023;
        if (temp1 > 511)            //should round up the fractional digit
            fd1 ++;
        if (fd1 == 10) {            //if fractional digit becomes 10, zero it
            fd1 = 0;                // and add 1 to integer part
            quo++;
        }
        hum_buf[4] = 0x30 + fd1;    //ASCII code of fractional digit
        hum_buf[2] = quo % 10 + 0x30; //ASCII code of one's digit
        quo = quo / 10;
        if (quo != 0)
        {
            hum_buf[1] = (quo % 10) + 0x30;
            quo /= 10;
        }
        if (quo == 1)
            hum_buf[0] = 0x31;
        while (!INTCONbits.TMR0IF);  //wait until Timer0 overflows
        for (i = 0; i < 4; i++) {    //wait for Timer0 to overflow four more times
            INTCONbits.TMR0IF = 0;   //clear the TMR0IF flag
            while(!INTCONbits.TMR0IF); //wait for Timer0 to overflow
        }
        INTCONbits.TMR0IF = 0
    }
}
```

This C program performs the A/D conversion and translates the result into an ASCII string so that it can be displayed in an LCD or a terminal.

12.8 Measuring Barometric Pressure

Barometric pressure refers to the air pressure existing at any point within the earth's atmosphere. This pressure can be measured as an absolute pressure (with reference to absolute vacuum) or can be referenced to some other value or scale. The meteorology and avionics industries traditionally measure the absolute pressure and then reference it to a sea-level pressure value. This complicated process is used in generating maps of weather systems.

Mathematically, atmosphere pressure is exponentially related to altitude. Once the pressure at a particular location and altitude is measured, the pressure at any other altitude can be calculated.

Several units have been used to measure the barometric pressure: in-Hg, kPa, mbar, or psi. A comparison of barometric pressure using four different units at sea level up to 15000 feet is shown in Table 12.6.

Altitude (ft)	Pressure (in-Hg)	Pressure (mbar)	Pressure (kPa)	Pressure (psi)
0	29.92	1013.4	101.4	14.70
500	29.38	995.1	99.5	14.43
1000	28.85	977.2	97.7	14.17
6000	23.97	811.9	81.2	11.78
10000	20.57	696.7	69.7	10.11
15000	16.86	571.1	57.1	8.28

Table 12.6 ■ Altitude versus pressure data

There are three forms of pressure transducer: *gauge, differential,* and *absolute.* Both the gauge pressure (psig) and differential (psid) transducers measure pressure differentially. The acronym "psi" stands for "pounds per square inch," whereas the letters "g" and "d" stand for "gauge" and "differential," respectively. A gauge pressure transducer measures pressure against ambient air, whereas the differential transducer measures against a reference pressure. An absolute pressure transducer measures the pressure against a vacuum (0 psia), and hence it measures the barometric pressure.

The SenSym ASCX30AN is a 0- to 30-psia (psi absolute) pressure transducer. The range of barometric pressure is between 28 to 32 inches of mercury (in-Hg), or 13.75 to 15.72 psia or 948 to 1083.8 mbar. The transducer output is about 0.15 V/psi, which would translate to an output voltage from 2.06 V to 2.36 V. The complete specification of the SenSym ASCX30AN is shown in Table 12.7. Since the range of V_{OUT} is very narrow, the user will need to use a level shifting and scaling circuit in order to take advantage of the available dynamic range.

The pin assignment of the SenSym ASCX30AN is shown in Figure 12.18.

Characteristic	min	typ	max
Pressure	0 psia	–	30 psia
Zero pressure offset	0.205	0.250	0.295
Full-scale span(2)	4.455	4.500	4.545
Output at FS pressure	4.660	4.750	4.840
Combined pressure non-linearity and pressure hysteresis(3)	–	±0.1	±0.5
Temperature effect on span(4)	–	±0.2	±1.0
Temperature effect on offset(4)	–	±0.2	±1.0
Response time (10% - 90%)(5)	–	0.1	–
Repeatability	–	±0.05	–

Note
1. *Reference conditions:* TA = 25° C, supply voltage V_S = 5 V
2. Full scale span is the algebraic difference between the output voltage at full-scale pressure and the output at zero pressure. Full-scale span is ratiometric to the supply voltage.
3. Pressure non-linearity is based on the best-fit straight line. Pressure hysteresis is the maximum output difference at any point within the operating pressure range for increasing and decreasing pressure.
4. Maximum error band of the offset voltage or span over the compensated temperature range relative to the 25° C reading.
5. Response time for 0 psi to full-scale pressure step response.
6. If maximum pressure is exceeded, even momentarily, the package may leak or burst, or the pressure-sensing die may burst.

Table 12.7 ■ ASCX30AN performance characteristics(1)

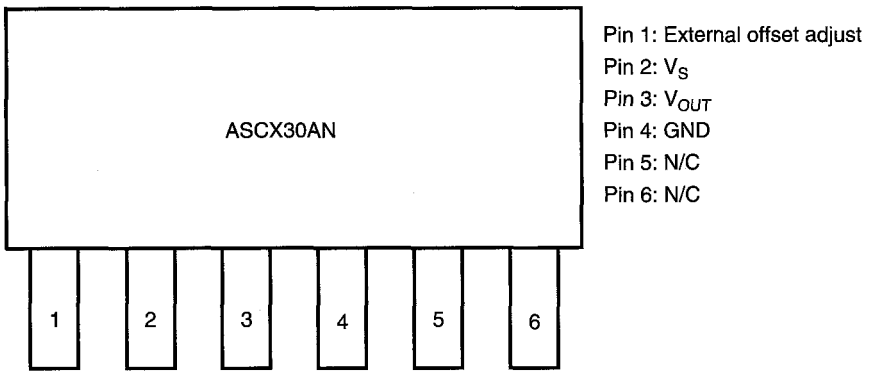

Figure 12.18 ■ ASCX30AN pin assignment

Example 12.10

Describe the circuit connection of the ASCX30AN and the voltage level shifting and scaling circuit and write a program to measure and display the barometric pressure in units of mbar.

Solution: The circuit connection is shown in Figure 12.19. The circuit in Figure 12.19 will shift and scale V_{OUT} to the range of 0 V ~ 5 V. According to Table 12.7, the typical value for the offset adjustment is around 0.25 V and can be achieved by using a potentiometer.

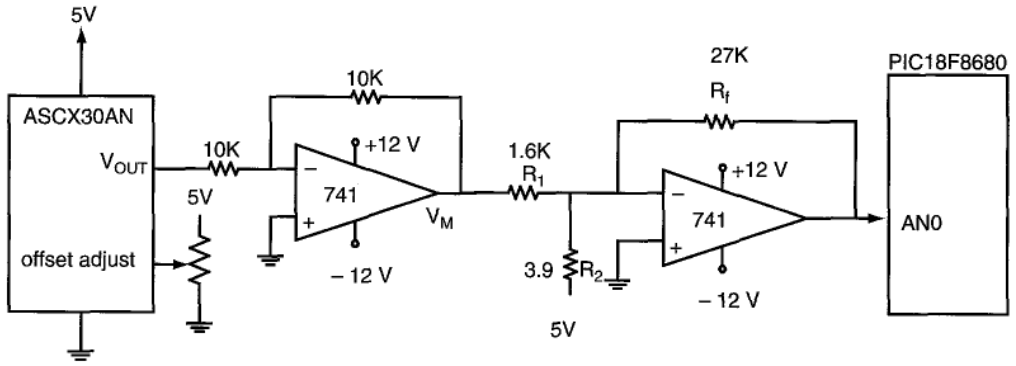

Figure 12.19 ■ Barometric pressure sensor output scaling and shifting circuit

The barometric pressure range is from 948 mbar to 1083.8 mbar. The A/D conversion result will be 0 and 1023 for these two values. Therefore, the user needs to divide the A/D conversion result by 7.53 in order to find the corresponding barometric pressure. The following equation shows the conversion:

Barometric pressure = 948 + A/D result/7.53
= 948 + (A/D result * 100)/753

The C program that performs A/D conversion, translates the result into barometric pressure, and then converts the barometric pressure into an ASCII string is as follows:

```
#include <p18F8680.h>
#include <timers.h>
#include <adc.h>
unsigned char bp_buf[7];
void main (void)
{
    unsigned short long a2d_val;
    unsigned short long quo, rem, temp1;
    char fd1, i;
    ADCON0 = 0x01;                          //select channel AN0, enable A/D module
    ADCON1 = 0x0E;                          //use VDD, VSS as reference and configure AN0 for analog
    ADCON2 = 0xA6;                          //result right justified, acquisition time = 8 TAD, FOSC/64
    OpenTimer0 (TIMER_INT_OFF & T0_16BIT & T0_SOURCE_INT & T0_PS_1_32);
              //start Timer0 and make it overflow in 200 ms
    while (1) {
        bp_buf[0] = 0x20;                   //set to space
        bp_buf[1] = 0x20;                   //set to space
        bp_buf[2] = 0x20;                   //set to space
        bp_buf[3] = 0x30;                   //set to digit 0
        bp_buf[4] = 0x2E;                   //store the decimal point
        bp_buf[5] = 0x30;                   //set to digit 0
        bp_buf[6] = '\0';                   //terminate the string with a NULL character
        ConvertADC( );                      //start an A/D conversion
        while(BusyADC());                   //wait until A/D conversion is done
```

```
            a2d_val = 100 * ReadADC();
            quo = a2d_val/753;              //convert to barometric pressure
            rem = a2d_val%753;              // "
            quo += 948;                     //add the barometric pressure at A/D
                                            //conversion result 0 to obtain the
                                            //actual barometric pressure
            fd1 = (rem * 10)/753;           //compute the fractional digit
            temp1 = (rem * 10)%753;
            if (temp1 > 376)                //should we round up the fractional digit?
               fd1 ++;
            if (fd1 == 10) {                //if fractional digit becomes 10, zero it
               fd1 = 0;                     //and add 1 to integer part
               quo++;
            }
            bp_buf[5] = 0x30 + fd1;         //ASCII code of fractional digit
            bp_buf[3] = quo % 10 + 0x30;    //ASCII code of one's digit
            quo /= 10;
            bp_buf[2] = quo % 10 + 0x30;    //ten's digit
            quo /= 10;
            bp_buf[1] = quo % 10 + 0x30;    //hundred's digit
            quo /= 10;
            bp_buf[0] = quo + 0x30;         //thousand's digit
            while(!INTCONbits.TMR0IF);      //wait until Timer0 overflows
            for (i = 0; i < 4; i++) {       //wait for Timer0 to overflow four more times
               INTCONbits.TMR0IF = 0;       //clear the TMR0IF flag
               while(!INTCONbits.TMR0IF);   //wait for Timer0 to overflow
            }
            INTCONbits.TMR0IF = 0;
         }
      }
```

12.9 Summary

A data acquisition system consists of four major components: a transducer, a signal conditioning circuit, an A/D converter, and a computer. The transducer converts a nonelectric quantity into a voltage. The transducer output may not be appropriate for processing by the A/D converter. The signal conditioning circuit shifts and scales the output from a transducer to a range that can take advantage of the full capability of the A/D converter. The A/D converter converts an electric voltage into a digital value that will be further processed by the computer.

Because of the discrete nature of a digital system, the A/D conversion result has a quantization error. The accuracy of an A/D converter is dictated by the number of bits used to represent the analog quantity. The more bits are used, the smaller the quantization error will be.

There are four major A/D conversion algorithms:

- Parallel (flash) A/D converter
- Slope and double-slope A/D converters
- Sigma-delta A/D converters
- Successive-approximation A/D converters

The PIC18 microcontroller uses the successive-approximation algorithm to perform the A/D conversion. All A/D conversion parameters are configured via three A/D control registers: ADCON0, ADCON1, and ADCON2. Some PIC18 members have only the first two controller registers (ADCON0 and ADCON1). The following A/D conversion parameters need to be configured:

- Analog channel to be converted
- High and low reference voltages
- Analog or digital nature of the A/D port pins
- A/D result format (right or left justified)
- A/D acquisition time
- A/D clock source

For some earlier PIC18 members, the A/D acquisition time may not be programmable and must be provided by using software delay.

The TC1047A temperature sensor, the IH-3606 humidity sensor, and the ASCX30AN pressure sensor are used as examples to illustrate the A/D conversion process. The TC1047A can measure a temperature in the range from –40° C to 125° C. The IH-3606 can measure relative humidity from 0% to 100%. The ASCX30AN can measure a pressure in the range from 0 to 30 psi absolute. These three examples demonstrate the need for a good voltage shifting and scaling circuit.

There are applications that require A/D accuracy higher than that provided by the PIC18 microcontrollers. In this situation, the user has the option of using an external A/D converter with higher precision or of selecting a different microcontroller with higher A/D resolution. Many 8051 variants from Silicon Laboratory, TI, and Analog Devices have much higher A/D resolutions.

12.10 Exercises

E12.1 Design a circuit that can scale the voltage from the range of 0 mV ~ 100 mV to the range of 0 V ~ 5 V.

E12.2 Design a circuit that can shift and scale the voltage from the range of –80 mV ~ 160 mV to the range of 0 V ~ 5 V.

E12.3 Design a circuit that can shift and scale the voltage from the range of –50 mV ~ 75 mV to the range of 0 V ~ 5 V.

E12.4 Design a circuit that can shift and scale the voltage from the range of 2 V ~ 2.5 V to the range of 0 V ~ 5 V.

E12.5 Suppose that there is a 10-bit A/D converter with V_{REF-} = 2 V and V_{REF+} = 4 V. Find the corresponding voltage values for the A/D conversion results of 40, 100, 240, 500, 720, 800, and 1000.

E12.6 Suppose that there is a 12-bit A/D converter with V_{REF-} = 1 V and V_{REF+} = 4 V. Find the corresponding voltage values for the A/D conversion results of 80, 180, 480, 640, 960, 1600, 2048, 3200, and 4000.

E12.7 Write a few instructions to configure the PIC18F452 A/D converter with the following parameters:

- f_{OSC} = 16 MHz (you need to choose appropriate clock source)
- Channel AN3
- A/D result left justified

- Configure A/D channel AN3, AN1, and AN0 for analog input, other A/D port pins for digital I/O
- Select V_{DD} and V_{SS} as reference voltages

E12.8 Write an instruction sequence to configure the PIC18F4320 A/D converter with the following characteristics:

- f_{OSC} = 8 MHz
- Channel AN3
- A/D result right justified
- Configure A/D channel AN3 through AN0 for analog input, other A/D port pins for digital I/O
- Select V_{DD} and V_{SS} as reference voltages

E12.9 Write a few instructions to configure the PIC18F8720 A/D converter with the following parameters:

- f_{OSC} = 25 MHz (you need to choose appropriate clock source)
- Channel AN15
- A/D result right justified
- Configure A/D channel AN15 for analog input
- Select V_{DD} and V_{SS} as reference voltages

E12.10 The LM35 from National Semiconductor is a Centigrade temperature sensor with three external connection pins. The pin assignment and circuit connection for converting temperature are shown in Figure 12E.1 Use this device to construct a circuit to display the room temperature in Celsius. Assume that the temperature range is –27° C through 100° C. Use an LCD to display the temperature.

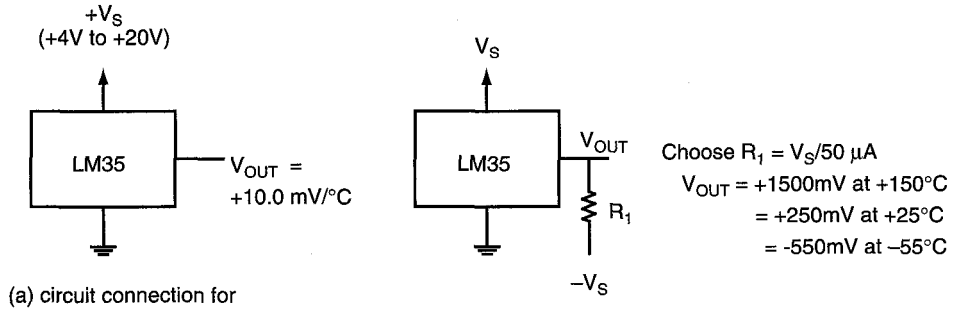

Figure 12E.1 ■ Circuit connection for the LM35

E12.11 The LM34 from National Semiconductor is a Fahrenheit temperature sensor with three external connection pins. The pin assignment and circuit connection for converting temperature are shown in Figure 12E.2. Use this device to construct a circuit to display the room temperature in Fahrenheit. Assume that the temperature range of interest is from –40° C to 215° F. Use an LCD to display the temperature.

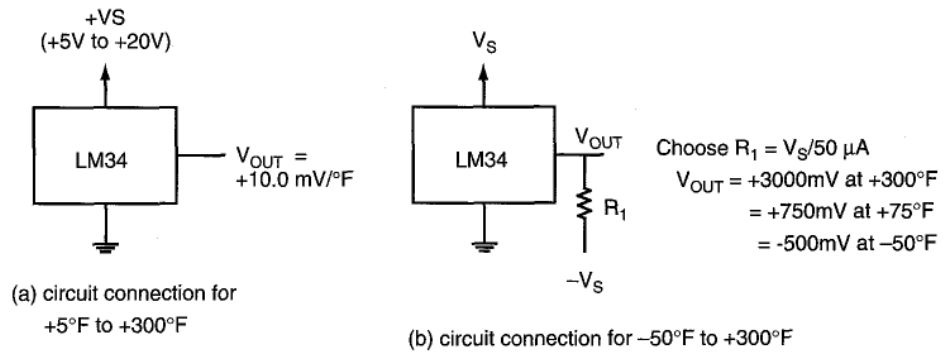

Figure 12E.2 ■ Circuit connection for the LM34

E12.12 The Microbridge AWM3300V is a mass airflow sensor manufactured by Honeywell. The block diagram of the AWM3300V is shown in Figure 12E.3. The Microbridge mass airflow sensor AWM3300V is designed to measure the airflow. Its applications include air-conditioning, medical ventilation/anesthesia control, gas analyzers, gas metering, fume cabinets, and process control. The AWM3300V operates on a single 10-V ± 10-mV power supply. The sensor output (from V_{OUT}) corresponding to the airflow rate of 0 ~ 1.0 liters per minute is 1.0 V to 5.0 V. The AWM3300V can operate in the temperature range of –25° C to 85° C. It takes 3 ms for the output voltage to settle after power-up. Design a circuit to measure and display the mass airflow using the AWM3300V. Write a program to configure the PIC18F8680 A/D module, start the A/D conversion, and display the mass airflow in an LCD display. Update the display 10 times per second.

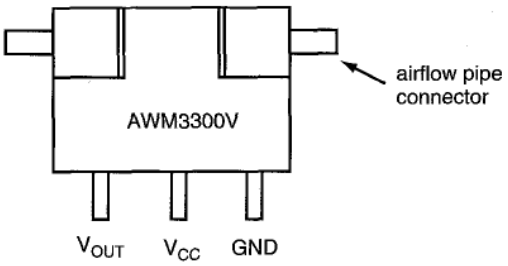

Figure 12E.3 ■ Microbridge AWM3300V

12.11 Lab Exercises and Assignments

L12.1 *A/D converter testing.* Perform the following steps:

Step 1
Set the function generator output (in the lab) to between 0 V and 5 V.

Step 2
Connect the AN1 pin to the functional generator output. Set the frequency to about 10 KHz.

Step 3
Write a program to perform the following operations:

- Configure the A/D module properly.
- Start the A/D conversion.
- Take 64 samples, convert them, and store the conversion results at appropriate SRAM locations.
- Compute the average value and store it in PRODL.

If your demo board has square waveform output, then select an output with frequency close to 16 KHz. Connect it to the AN1 pin.

L12.2 *Continuous voltage measurement.* Perform the following operations:

Step 1
Connect the AN0 pin to a potentiometer that can vary voltage from 0 V to 5 V.

Step 2
Configure the A/D module properly.

Step 3
Take one sample of the voltage from the potentiometer, convert the voltage, convert the A/D conversion result to the corresponding voltage, and display the voltage on the LCD.

Step 4
Repeat Step 3 every 100 ms.

L12.3 *Digital thermometer.* Use the LM34 temperature sensor, the 741 OP AMP, and the required resistors to construct a digital thermometer. The temperature should be displayed in three integral and one fractional digits. Use the LCD on your demo board to display the temperature. Update the temperature once every 100 ms.

L12.4 *Barometric pressure measurement.* The Motorola MPX4115A is a pressure sensor that can measure a pressure ranging from 15 kPa to 115 kPa (2.2–16.7 psi) and has a corresponding voltage output from 0.2 V to 4.8 V. The small outline package of this device has eight pins. Among these eight pins, only three pins carry useful signals:

Pin 2: V_S

Pin 3: Ground

Pin 4: V_{OUT}

This pressure sensor is often used in aviation altimeters, industrial controls, engine control, and weather stations and weather reporting devices.

Connect an MPX4115AC6U (or MPX4115A6U, case 482) device to your PIC18 demo board to measure the current ambient barometric pressure once every second and display the pressure on the LCD display. The datasheet of the MPX4115A can be found in the complimentary CD. Since the barometric pressure is in a narrow range (use the range mentioned in Example 12.10), one may need to construct a signal conditioning circuit to translate and scale the voltage output of the pressure sensor to the range of 0 V to 5 V.