

ANALOG-TO-DIGITAL CONVERTER

PIC Microcontroller: An Introduction to Software & Hardware Interfacing Han-Way Huang Thomson Delmar Learning, 2005

Chung-Ping Young 楊中平

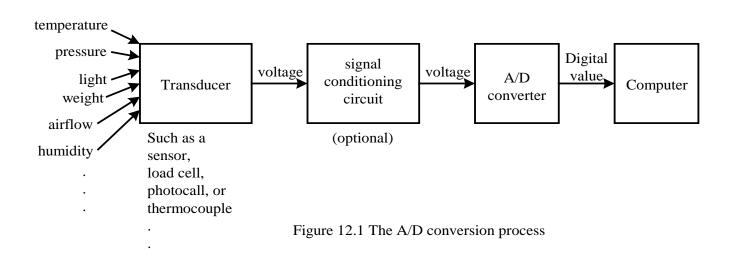


Networked Embedded Applications and Technologies Lab



Basics of A/D Conversion

- Can convert only electrical voltages to digital values
- A transducer is needed to convert a non-electric quantity into an electrical voltage
- Different names of transducers are used for different physical quantities
- A data acquisition system is used to referred to those systems that perform A/D conversions.







Analog Voltage and Digital Code Characteristic

- 1. Characteristic of an Ideal A/D Converter
 - Needs infinite number of bits to encode the A/D conversion result
 - Unachievable and impractical

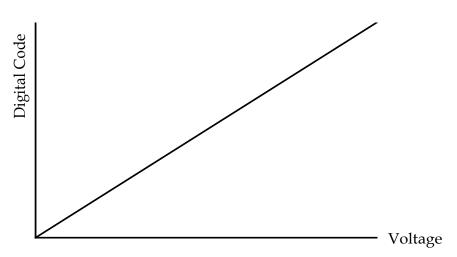


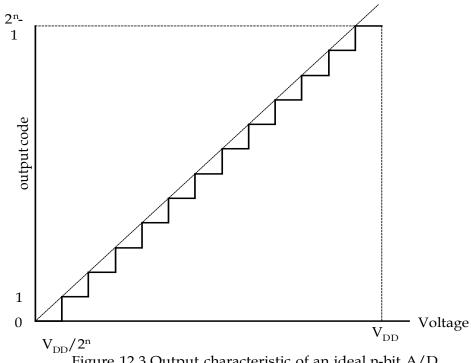
Figure 12.2 An ideal A/D converter output characteristic

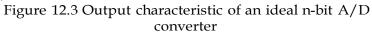




Characteristic of an Ideal n-bit A/D Converter

- The area between dotted line and staircase is called the **quantization error**.
- The **resolution** of this A/D converter is V_{DD}/2ⁿ.
- Average conversion error is V_{DD}/2ⁿ⁺¹.
- A real A/D converter has nonlinearity.









A/D Conversion Algorithms

- 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

Successive Approximation A/D Conversion Method

- Most commonly used A/D conversion method for 8-bit and 16-bit microcontrollers.

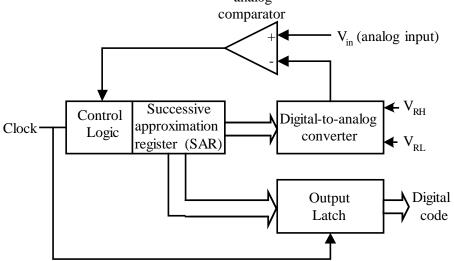


Figure 12.4 Block diagram of a successive approximation A/D converter





Successive Approximation Method

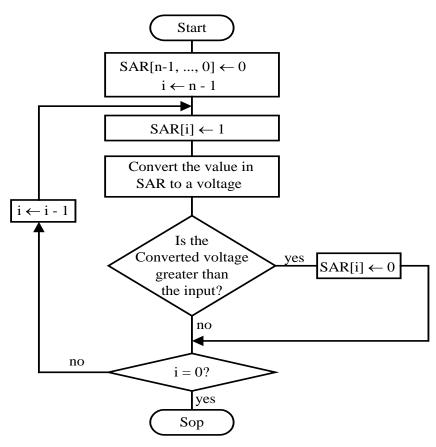


Figure 12.5 Successive approximation A/D conversion method





Optimal Voltage Range for A/D Conversion

- A/D converter requires a low reference voltage (V_{REF-}) and a high reference voltage (V_{RFF+}) to perform conversion.
- Most A/D converters are ratiomertic:
 - 1. An analog input of V_{RFF} is converted to digital code 0.
 - 2. An analog input of V_{REF+} is converted to digital code $2^n 1$.
 - 3. An analog input of k V is converted to digital code

$$(2^{n} - 1) \times (k - V_{REF-}) \div (V_{REF+} - V_{REF-})$$

- The A/D conversion result k corresponds to the following analog input:

$$V_{K} = V_{REF-} + (V_{REF+} - V_{REF-}) \times k \div (2^{n} - 1)$$

- Most systems use V_{DD} and 0V as V_{REF+} and V_{REF-}, respectively.





Example 12.1 Suppose that there is a 10-bit A/D converter with $V_{REF-} = 1 \text{ V}$ and $V_{REF+} = 4 \text{ V}$. Find the corresponding voltage values for the A/D conversion results of 25, 80, 240, 500, 720, 800, and 900.

Solution:

The corresponding voltages are as follows:

$$1V + (3 \times 25) \div (2^{10} - 1) = 1.07 V$$

 $1V + (3 \times 80) \div (2^{10} - 1) = 1.23 V$
 $1V + (3 \times 240) \div (2^{10} - 1) = 1.70 V$
 $1V + (3 \times 500) \div (210 - 1) = 2.47 V$
 $1V + (3 \times 720) \div (210 - 1) = 3.11 V$
 $1V + (3 \times 800) \div (210 - 1) = 3.35 V$
 $1V + (3 \times 900) \div (210 - 1) = 3.64 V$

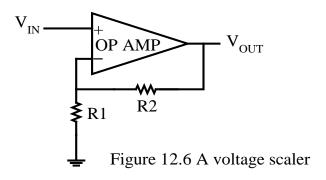


Scaling Circuit

- Used to amplify the transducer output from a range of 0V ~ V_Z to 0 ~ V_{DD}.
- Usually V_Z is smaller than V_{DD} .
- Voltage gain of the circuit in Figure 12.6 is

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

- Both R₁ and R₂ must be commercially available resistors.





Example 12.2 Suppose the transducer output voltage ranges from 0V to 200 mV. Choose the appropriate values for R1 and R2 to scale this range to 0~5V. **Solution:**

$$R2/R1 = (V_{OUT}/V_{IN}) - 1 = 24$$

Choose 240 K Ω for R₂ and 10 K Ω for R₁.

Voltage Translation Circuit

- Needed to shift and scale the transducer output in a range of $-V_X \sim V_Z$ to $0V \sim V_{DD}$.
- The circuit is shown in Figure 12.7.

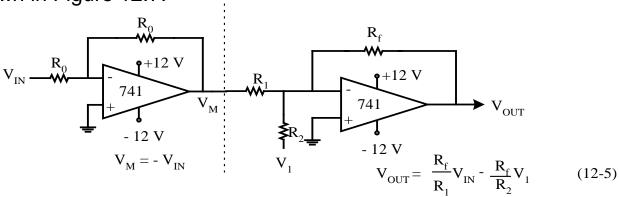


Figure 12T.7 Level shifting and scaling circuit



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 $-1.2V \sim 3.0V$ to the range of $0V \sim 5V$.

Solution: Applying Equation 12.5:

$$0 = -1.2 \times (Rf/R1) - (Rf/R2) \times V1$$

 $5 = 3.0 \times (Rf/R1) - (Rf/R2) \times V1$

Choose R0 = R1 = 10 K Ω and V₁ = -5V, solve R₂ = 50 K Ω , and R_f = 12K Ω



The PIC18 A/D Converter

- The PIC18 has a 10-bit A/D converter.
- The number of analog inputs varies among difference PIC18 devices.
- The A/D converter has the following registers:
- A/D Result High Register (ADRESH)
- A/D Result Low Register (ADRESL)
- A/D Control Register 0 (ADCON0)
- A/D Control Register 1 (ADCON1)
- A/D Control Register 2 (ADCON2)
- The contents of these registers vary with the PIC18 members.
- Early PIC18 (PIC18FXX2) members have only ADCON0 and ADCON1 registers.

ADCON0 Register

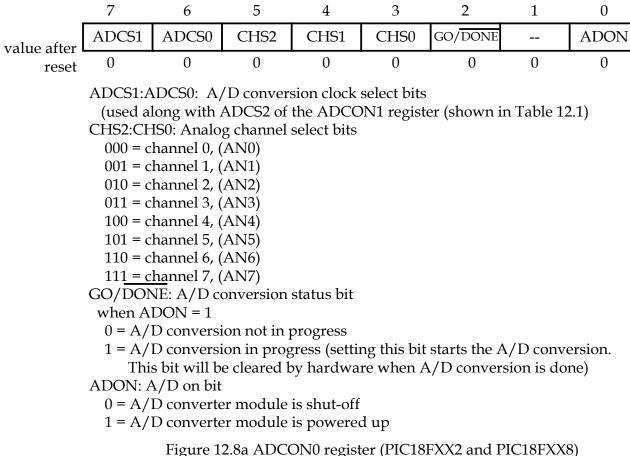


Figure 12.8a ADCON0 register (PIC18FXX2 and PIC18FXX8) (redraw with permission of Microchip)



Table 12.1 A/D conversion clock source select bits

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)

	7	6	5	4	3	2	1	0
value after			CHS3	CHS2	CHS1	CHS0	GO/DONE	ADON
reset	0	0	0	0	0	0	0	0
	0000 = 0001 = 0010 = 0011 = 0100 = 0111 = 1000 = 1011 = 1100 = 1101 = 1110 = 1111 = GO/DON when A 0 = A/This		(AN0) (AN1) (AN2) (AN3) (AN4) (AN5) (AN6) (AN7) (AN8) (AN9) 0, (AN10) 1, (AN11) 2, (AN12) 3, (AN13) 4, (AN14) 5, (AN15) conversion ion not in progeneous in	status bit progress gress (settii			A/D conve	
	-			is shut-off is powered				

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



7 6 5 4 3 2 1 0 VCFG0 CHS2 GO/DONE VCFG1 CHS1 CHS0 **ADON** value after 0 0 0 0 0 0 0 0 reset

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/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)

Table 12.2 Voltage reference configuration bits

	0	
VCFG1:VCFG0	A/DV_{REF} +	A/DV_{REF} -
00	AV_{DD}	$\mathrm{AV}_{\mathrm{SS}}$
01	External V _{REF} +	$\mathrm{AV}_{\mathrm{SS}}$
10	AV_{DD}	External ${ m V}_{ m REF}$ -
11	External V _{REF} +	External V_{REF}^{REF} -



ADCON1 Register

- Configure an input pin as analog or digit.
- An input to be converted must be an analog input.

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





Table 12.3 A/D port configuration control bits

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

A = analog input D = digital input

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

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

VCFG1:VCFG0: Voltage reference configuration bits (see Table 12.2)

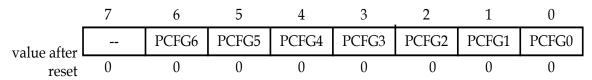
PCFG3:PCFG0: A/D port configuration control bits

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





PCFG6..PFCG0: AN6..AN0 A/D port configuration bit

0 = Pin configured as an analog channel -- digital input disabled and read as 0

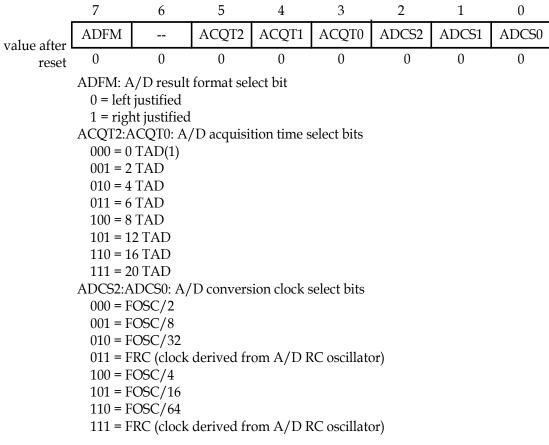
1 = Pin configured as a digital input

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





ADCON2 Register



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)



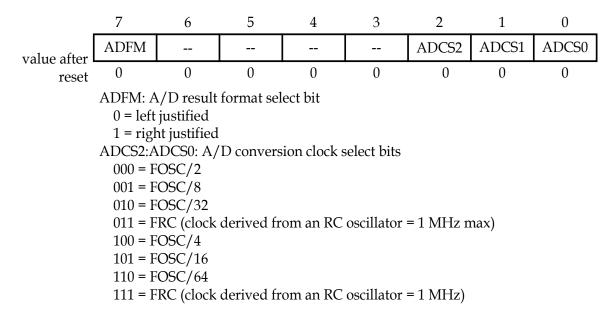
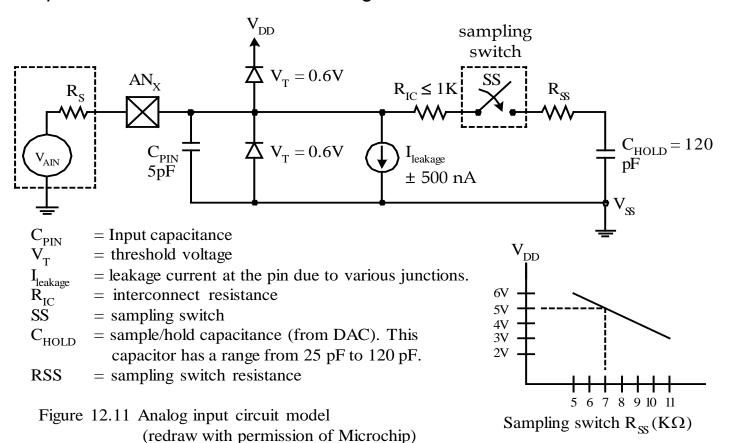


Figure 12.10b ADCON2 register (PIC18F8720/8620/8520/6720/6620/6520) (redraw with permission of Microchip)



A/D Acquisition Requirements

- The A/D converter has a sample-and-hold circuit for analog input.
- The sample-and-hold circuit keeps the voltage stable when it is converted.
- The sample-and-hold circuit is shown in Figure 12.11.





Automatic Acquisition Time

- For earlier PIC18 members, when the GO/DONE bit is set, sampling is stopped and conversion begins.
- The user is responsible for making sure enough acquisition time is provided.
- For newer PIC18 members, the A/D module will continue to sample after the GO/DONE bit is set for the selected acquisition time.
- The automatic acquisition time makes A/D programming a little easier.

Selecting the A/D Conversion Clock

- The per bit A/D conversion time is defined as T_{AD}.
- Each 10-bit A/D conversion takes 12 T_{AD} to complete.
- For some devices, the options for T_{AD} are defined in ADCON0. For others, the options for T_{AD} are defined in ADCON2.
- The length of T_{AD} must be at least 1.6 μ s.
- The options for T_{AD} versus crystal oscillator frequency are listed in Table 12.4





Table 12.4 T_{AD} vs. device operating frequencies

AD clock	source (T _{AD})	maximum device frequency			
Operation	ADCS2:ADCS0	normal MCU	Low power MCU		
2 T _{OSC} 4 T _{OSC} 8 T _{OSC} 16 T _{OSC} 32 T _{OSC} 64 T _{OSC} RC	000 100 001 101 010 110 x11	1.25 MHz 2.50 MHz 5.00 MHz 10.0 MHz 20.0 MHz 40.0 MHz 1.00 MHz	666 KHz 1.33 MHz 2.66 MHz 5.33 MHz 10.65 MHz 21.33 MHz 1.00 MHz		

Note. Lower power device has a letter L in its name. For example, PIC18LF8720

A/D Conversion Operation

- For PIC18 members without the feature of automatic acquisition, A/D conversion will start immediately after setting the GO/DONE bit.
- For devices with automatic acquisition, the device will continue to perform data acquisition until the preprogrammed time is up and then start the A/D conversion.

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
- $f_{OSC} = 32 \text{ MHz}$
- The 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:

- $T_{COFF} = (60 25) \times 0.05 = 1.75 \,\mu s$
- The required acquisition time = $(9.61 + 2 + 1.75) = 13.36 \mu s$.
- f_{OSC} = 32 MHz, the A/D clock source must be set to 64 f_{OSC} , which makes T_{AD} = 2 μs .
- PIC18F8680 supports automatic acquisition time, it must be set to at least 8 T_{AD} to make it greater than 13.36 μs

Assembly instruction sequence that achieve the desired setting:

```
movlw
          0x01
                             : select channel ANO and enable A/D
          ADCON0,A
movwf
         0x0E
movlw
                             ; configure only channel ANO as analog port,
         ADCON1,A
                             ; select V_{DD} and V_{SS} as reference voltage
movwf
movlw
          0xA6
                             ; set A/D result right justified, set acquisition
          ADCON2,A
                             ; time to 8 TAD, clock source F<sub>OSC</sub>/64
movwf
```

In C language,

```
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
- $f_{OSC} = 32 \text{ MHz}$
- The 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:



Procedure for Performing A/D Conversion

- Configure the A/D module
 - 1. Configure analog pins, reference voltages
 - 2. Select A/D input channel
 - 3. Select A/D acquisition time (if available)
 - 4. Select A/D conversion clock
 - 5. Enable A/D module
- Configure A/D interrupt
 - 1. Clear ADIF flag
 - 2. Set ADIE bit (if desired)
 - 3. Set GIE bit (if desired)
- Wait for the desired acquisition time (if required)
- Start conversion by setting the GO/DONE bit
- Wait for A/D conversion to complete
- Read the A/D result registers; clear the ADIF flag
- For next conversion, go to step 1 or step 2.





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 0V to 5V. Write a program to measure the voltage applied to the AN0 pin, convert it, and retrieved the conversion result and place it in PRODH:PRODL.

Solution:

```
#include <p18F8680.inc>
                   0x00
         org
         goto
                   start
                   80x0
         org
         retfie
                   0x18
         org
         retfie
         movlw
                   0x01
                                       : select channel ANO and enable A/D
start
         movwf
                   ADCON0,A
         movlw
                   0x0E
                                       ; use VDD & VSS as reference voltages &
                   ADCON1,A
                                       ; configure channel ANO as analog input
         movwf
                   0xA6
                                       ; select F<sub>OSC</sub>/64 as conversion clock,
         movlw
                                       ; 8 T<sub>AD</sub> for acquisition time, right-justified
                   ADCON2,A
         movwf
                   ADCON0,GO,A
                                      ; start A/D conversion
         bsf
         btfsc
                   ADCON0, DONE, A; wait until conversion is done
wait con
         bra
                   wait con
         movff
                   ADRESH, PRODH; save conversion result
                   ADRESL, PRODL ;
         movff
         end
```





A/D Converter C Library Functions

char BusyADC (void);

- Returns a 1 if the A/D module is busy. Returns a 0 if not.

void CloseADC (void);

- Disable the A/D module.

void ConvertADC (void);

- Start an A/D conversion

int ReadADC (void);

- Reads the A/D conversion result

void SetChanADC (unsigned char channel);

- Selects the pin used as input to the A/D converter. The channel value can be from ADC_CH0 through ADC_CH15.

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

- The setting of config and config2 depends on the device.
- The PIC18F1X20/2X20/3X20/4X20 have a third argument: **portconfig**.
- The values of these arguments are in the following slides





PIC18FXX2, PIC18FXX2, and PIC18FXX8

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

A/D clock source

ADC_FOSC_2 FOSC/2
ADC_FOSC_4 FOSC/4
ADC_FOSC_8 FOSC/8
ADC_FOSC_16 FOSC/16
ADC_FOSC_32 FOSC/32
ADC_FOSC_64 FOSC/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 OREF
                   VREF+ = VDD, VREF- = VSS, all analog channels
ADC 7ANA 1REF
                   AN3
                          = VREF+, all analog channels except AN3
ADC 6ANA 2REF
                   AN3
                          = VREF+, AN2 = VREF-
ADC 6ANA OREF
                   VREF+ = VDD, VREF- = VSS
ADC 5ANA 1REF
                   AN3
                          = VREF+. VREF- = VSS
ADC 5ANA OREF
                   VREF+ = VDD, VREF- = VSS
ADC 4ANA 2REF
                   AN3
                          = VREF+. AN2 = VREF-
ADC 4ANA 1REF
                   AN3
                          = VREF+
ADC 3ANA 2REF
                   AN3
                          = VREF+. AN2 = VREF-
ADC 3ANA 0REF
                   VREF+ = VDD. VREF- = VSS
ADC 2ANA 2REF
                   AN3
                          = VREF+, AN2 = VREF-
ADC 2ANA 1REF
                          = VRFF+
                   AN3
ADC 1ANA 2REF
                   AN3
                          = VREF+, AN2 = VREF-, AN0 = A
ADC 1ANA OREF
                   AN0 is analog input
ADC OANA OREF
                   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 FOSC/2
ADC_FOSC_4 FOSC/4
ADC_FOSC_8 FOSC/8
ADC_FOSC_16 FOSC/16
ADC_FOSC_32 FOSC/32
ADC_FOSC_64 FOSC/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
) Interrupts	

A/D

Interrupts enabled ADC_INT_ON ADC_INT_OFF Interrupts disabled





A/D voltage configuration

ADC_VREFPLUS_VDD VREF+ = AVDD ADC_VREFPLUS_EXT VREF+ = external ADC VREFMINUS VSSVREF- = AVSS

ADC_VREFMINUS_EXT VREF- = external

PIC18F6X8X, PIC18F8X8X, PIC18F1X20, PIC18F2X20, PIC18F4X20

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

A/D clock source

ADC_FOSC_2 FOSC/2
ADC_FOSC_4 FOSC/4
ADC_FOSC_8 FOSC/8
ADC_FOSC_16 FOSC/16
ADC_FOSC_32 FOSC/32
ADC FOSC 64 FOSC/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 Tad ADC 2 TAD 2 Tad ADC 4 TAD 4 Tad ADC 6 TAD 6 Tad ADC 8 TAD 8 Tad ADC 12 TAD 12 Tad ADC 16 TAD 16 Tad ADC_20_TAD 20 Tad

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

Channel

ADC CHO 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 VREF+ = AVDD ADC_VREFPLUS_EXT VREF+ = external ADC_VREFMINUS_VSS VREF- = AVSS ADC_VREFMINUS_EXT VREF- = external

- The third argument **portconfig** is any value from 0 to 127 for the PIC18F1220/1320.
- The argument **portconfig** is any value from 0 to 15 for the PIC2220/2320/4220 and PIC18F4320/6X8X/8X8X.





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 AN0 pin of port A 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:

```
#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);
                           // provides 200 instruction cycles of acquisition time
    Delay10TCYx(20);
    ConvertADC();
                                     // start A/D conversion
    while(BusyADC( )); // wait for completion
    result = ReadADC();
                         // read result
    CloseADC();
```





Using the Temperature Sensor TC1047A

- Can measure temperature from -40°C to 125°C
- The voltage output of this sensor is 0.1V at -40°C, 1.75V at 125°C.

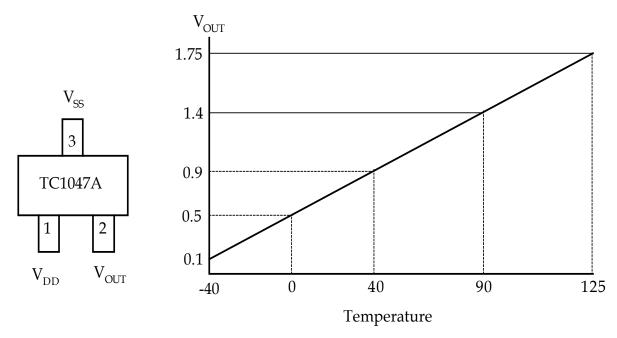


Figure 12.14 TC1047A V_{OUT} vs. temperature characteristic



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 ten times per second and assume that the PIC18F8680 operates with a 32 MHz crystal oscillator.

Solution:

- A signal conditioning circuit is needed to shift and scale the output of the TC1047A to $0 \sim 5V$.
- To convert the A/D conversion result back to the corresponding temperature, perform the following operations:
 - 1. Divide the conversion result by 6.2
 - 2. Subtract the quotient by 40
- The operation of "divide by 6.2" can be implemented by
 - 1. Multiplying the conversion result by 10
 - 2. Dividing the product by 62
- The digital thermometer circuit is shown in Figure 12.15.



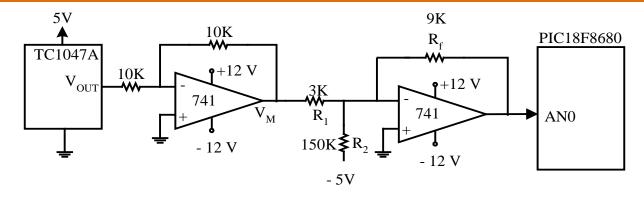


Figure 12.15 Circuit connection between the TC1047A and the PIC18

The Procedure

Step 1

Configure 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 temperature reading. Use 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; else replace **quo** with its two's complement.

Step 8

```
If rem \neq 0, then decrement quo by 1 rem \leftarrow 62 - rem.
```

Step 9

Compute the fractional digit by multiplying **rem** by 10 and then dividing the resulted 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.



```
#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
                                ; number of bytes used for local variable
loc varm equ
                                ; offset of PD3 from frame pointer
          equ
                 6
                                ; offset of PD2 from frame pointer
          equ
                 5
                                ; offset of PD1 from frame pointer
          equ
                  4
                                ; offset of PD0 from frame pointer
          equ
                  -8
                                ; offset of MD_lo from frame pointer
          equ
```

```
-3
buf hi
          equ
                  0x01
ptr hi
                                ; address of buffer to hold product
          equ
                  0x00
ptr lo
          equ
```

-7

-6

-5

-4

equ

equ

equ

equ



pd0

pd1

pd2

pd3

MD lo

MD hi

ND lo

ND hi

buf lo

; offset of MD_hi from frame pointer

; offset of ND lo from frame pointer

; offset of ND_hi from frame pointer

```
The following definitions are used by the 16-bit unsigned divide routine
                              : local variable size
loc var
         equ
lp cnt
                              : loop count
         equ
temp
        equ
                              ; temporary storage
                    -7
                              ; offset for quotient and dividend from frame
quo hi
        equ
                    -8
quo_lo
                              ; pointer
         equ
rem hi
        equ
                              ; offset for remainder from frame pointer
                    -6
rem lo
        equ
                              ; offset for divisor from frame pointer
dsr hi
         equ
                    -4
dsr lo
         equ
         set
                   0x02
                              ; memory space to hold the quotient
quo
                   0x04
                              : memory space to hold the remainder
          set
rem
 variables for holding temperature conversion result and string
                   0x06
                              ; space to hold integer part of the temperature
int pt
          set
                   80x0
temp_buf set
                              ; reserve 6 bytes to hold the temperature
                              ; digits, sign, period, and NULL
```



```
0x00
          org
                                     ; reset vector
          goto
                    start
                    80x0
          org
          retfie
                    0x18
          org
          retfie
start
          lfsr
                    FSR1,0xC00
                                     ; set up stack pointer
; initialize the temperature string to ^0.0
          call
                    a2d init
                                     ; configure and turn on A/D module
                    0x20
         movlw
forever
                                     ; store space character
          movwf
                   temp_buf
                    temp_buf+1
          movwf
                    0x30
          movlw
                                     ; store a 0 digit
          movwf
                   temp_buf+2
          movwf
                    temp_buf+4
          movlw
                    0x2E
                                     ; store a period
                    temp_buf+3
          movwf
          clrf
                    temp_buf+5
                                     ; terminate the string with a NULL character
          call
                    OpenTmr0
                                     : initialize and enable Timer0
          bsf
                    ADCON0,GO,A; start A/D conversion
```





```
wait a2d btfsc
                    ADCON0,DONE,A; wait until A/D conversion is complete
                    wait_a2d
         bra
                    ADRESL
                                      ; push the A/D conversion result in
         pushr
                    ADRESH
         pushr
                                      : 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
                                                ; deallocate space used in the stack
                                      6
                                      ; place the address of product in FSR0
         movlw
                    ptr lo
                    FSR0L,A
         movwf
                    ptr_hi
         movlw
                    FSR0H,A
         movwf
                    POSTINCO,W,A
                                      ; push (A/D result x 10)
         movf
                    WREG
         pushr
         movf
                    INDF0,W,A
                    WREG
         pushr
                    2
         alloc stk
         push_dat
                    0x3E
                                      ; push 62 into the stack as the divisor
         push dat
         call
                    div16u.FAST
```



```
dealloc stk
                                        ; retrieve remainder high byte (should be 0)
                     rem+1
          popr
                                        ; retrieve remainder low byte
          popr
                      rem
                                        ; retrieve integer part of the (should be 0)
                     int_pt+1
          popr
          popr
                     int_pt
                                        ; temperature
; subtract 40 from integer part to obtain the actual temperature
                                        ; calculate the actual temperature
          movlw
                      0x28
          subwf
                     int_pt,F,A
                                        ; if non-minus, no need for further check
          bnn
                      non minus
                     int_pt,A
                                        ; find the magnitude of temperature
          negf
          movlw
                     0x2D
          movlw
                     temp_buf
                                        ; store the minus sign
                      rem,W,A
                                        ; check the fractional part before divide
          movf
                     separate_dd
                                        ; branch to separate integer digits if rem = 0
          bz
          decf
                     int_pt,F,A
                                        ; fractional digit \neq 0, decrement integer part
                     0x3E
                                        ; need to find the complement of the fractional
          movlw
          subwf
                                        ; digit
                     rem.F.A
          negf
                     rem,A
; calculate the fractional digit
non minus
          movlw
                      0x0A
          mulwf
                      rem
```





```
PRODL
          pushr
                                        ; push (remainder x 10)
                     PRODH
          pushr
          alloc stk
                     2
          push_dat
                     0x3E
                                         ; push 62 into the stack as the divisor
          push_dat
          call
                     div16u,FAST
                                         2
          dealloc stk
                     rem+1
                                        ; retrieve remainder high byte (= 0)
          popr
                                        ; retrieve remainder low byte
          popr
                     rem
                                        ; retrieve quotient high byte (= 0)
                     quo+1
          popr
                                        ; retrieve quotient low byte
          popr
                     quo
; round the fractional digit
                     0x1F
                                        ; is remainder >= 31?
          movlw
          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
          alloc_stk
          push_dat
                     0x0A
                                        ; push 10 as the divisor
          push_dat
          call
                     div16u,FAST
          dealloc_stk
                                        2
                     rem+1
          popr
          popr
                     rem
          popr
                     quo+1
          popr
                     quo
          movlw
                     0x30
          addwf
                     rem,W,A
          movwf
                     temp_buf+2
                                        ; save the one's digit
                     quo,W,A
          movf
                                        ; check the quotient
                      next time
                                        ; wait to perform next conversion
          bz
; prepare to separate ten's digit
          movlw
                     0x0A
                                        ; is the quotient >= 10?
          cpfslt
                     quo,A
```



```
goto
                    yes_ge
         movlw
                   0x30
                   quo,W,A
         addwf
         movwf
                   temp_buf+1
                                       ; save the ten's digit
                   next_time
         goto
                   0x31
         movlw
                                       ; save "1" as the hundred's digit
yes ge
                   temp_buf,A
         movwf
                   0x0A
         movlw
                                       ; separate the ten's digit and place it
         subwf
                   quo,W,A
                                       ; in WREG
                   0x30
         addlw
                                       ; convert ten's digit to ASCII and
         movwf
                   temp_buf+1,A
                                       ; save ten's digit
                   INTCON, TMR0IF, A; wait until 200 ms is over
next_time btfss
                   next_time
         goto
; add instructions to update display here
                   forever
                                       ; prepare to perform next A/D conversion
         goto
```

```
; 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
                     TMR0H
                                       : so that it overflows in
           movwf
                    0xAF
                                       ; 200 ms
           movlw
                    TMR0L
           movwf
           movlw
                    0x84
                                       ; enable TMR0, select internal clock,
                    T0CON
           movwf
                                       ; set prescaler to 32
                     INTCON, TMR0IF; clear TMR0IF flag
           bcf
           return
: This routine initialize the A/D converter to select channel ANO 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
                    ADCON0
           movwf
                                       : and enable A/D module
                                       ; use VDD & VSS as A/D reference
           movlw
                     0x0E
voltage
           movwf
                     ADCON1
                                       ; & configure AN0 as analog input
```



```
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
```



```
#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
    OpenTimerO(TIMER INT OFF & TO 16BIT & TO 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
```



```
//wait until A/D conversion is done
while(BusyADC());
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;
```



Using the IH-3606 Humidity Sensor

- Voltage output is 0.8V to 3.9V for the relative humidity from 0 to 100%
- Can resist contaminant vapors such as organic solvent, chlorine, and ammonia.
- A 3-pin device
- Light sensitive and should be shielded from bright light
- Pin assignment is shown in Figure 12.16.
- Characteristics are listed in Table 12.5.

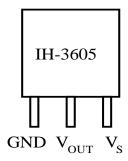


Figure 12.16 Honeywell IH-3605 humidity sensor

Table 12.5 Specifications of IH-3605

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	$\pm 0.007\%$ RH/°C (negligible)		
Temp. effect on 100% RH voltage	-0.22% RH/°C		
Output voltage	$V_{OUT} = (V_s)(0.16 \text{ to } 0.78)$ 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 1KHz 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		

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:

- It is beneficial to scale and shift the humidity sensor output to the range of 0~5V.
- A 1-KHz low pass filter is needed at the output of the humidity sensor.
- To convert to the relative humidity, divide the A/D conversion result by 10.23.

- The humidity data will be represented using an LCD or four seven-segment displays. $V_s (= 5V)$

 V_{OUT} V_{O

 $R0 = R1 = 10K\Omega$ $R2 = 62K\Omega$

 $R_2 = 02R\Omega_2$ $R_1 = 16K\Omega$

 $V1 = 10K\Omega$

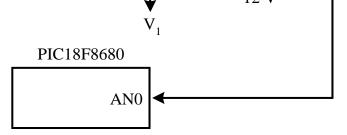


Figure 12.17 Relative humidity measurement circuit



 \boldsymbol{V}_{OUT}

```
#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
}</pre>
```





Measuring Barometric Pressure

- Barometric pressure is the pressure existing at any point in the earth atmosphere.
- The barometric pressure can be measured as an absolute pressure or can be referenced to some other value or scale.
- The meteorology and avionics industries measure the absolute pressure.
- The units used to represent the barometric pressure include **in-Hg**, **kPa**, **mbar**, and **psi**.
- A comparison of barometric pressures in different units is shown in Table 12.6.

Table 12.6 Altitude versus pressure data

	Altitude (ft)	Pressure (in-Hg)	Pressure (mbar)	Pressure (kPa)	Pressure (psi)
	0	29.92	29.92 1013.4 101.4		14.70
١	500	29.38	995.1	99.5	14.43
1	1000	28.85	977.2	97.7	14.17
١	6000	23.97	811.9	81.2	11.78
١	10000	20.57	696.7	69.7	10.11
	15000	16.86	571.1	57.1	8.28

The SenSym ASCX30AN Pressure Sensor

- Can measure barometric pressure from 0 to 30 psia
- The range of barometric pressure is between 28 to 32 **in-Hg** or 948 to 1083.8 **mbar**.
- The ASCX30AN output voltage would range from 2.06 V to 2.36 V.

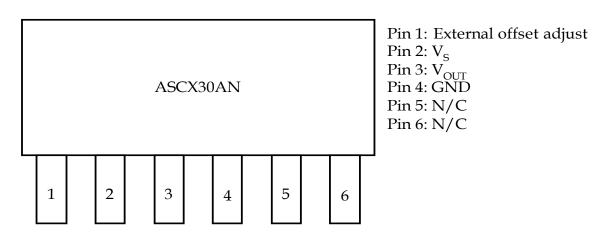


Figure 12.18 ASCX30AN pin assignment



Table 12.7 ASCX30AN performance characteristics (1)

Characteristic	min	typ	max
Pressurerange	0 psia		30 psia
Zeropressureoffset	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		±0.1	±0.5
pressure hysteresis ⁽³⁾			
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.





Example 12.10 Describe the circuit connection of the ASCX30AN, the voltage level shifting and scaling circuit, and write a program to measure and display the barometric pressure in units of **mbar**.

Solution:

- A shift and scale circuit is needed to convert the voltage to the range from 0 to 5V.
- The typical offset adjust is 0.25V and can be achieved by using a potentiometer.

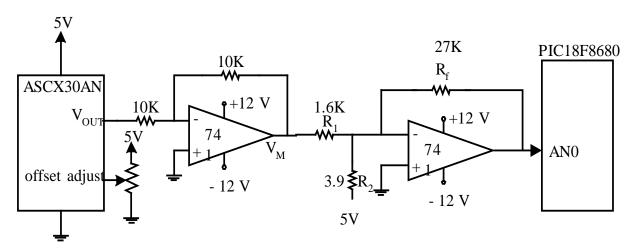


Figure 12.19 Barometric pressure sensor output scaling and shifting circuit.



- The barometric pressure (in mbar) can be derived by dividing the A/D conversion result by 7.53 and then add 948 to the quotient.

```
Barometric pressure
                          = 948 + A/D result / 7.53
                   = 948 + (A/D result * 100)/753
#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 ANO, 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 & TO 16BIT & TO SOURCE INT &
               TO 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
                                 //start an A/D conversion
     ConvertADC();
     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 = 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;
```