Chapter 12: Analog-to-Digital Converter

The PIC18 Microcontroller

Han-Way Huang

Minnesota State University, Mankato

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.

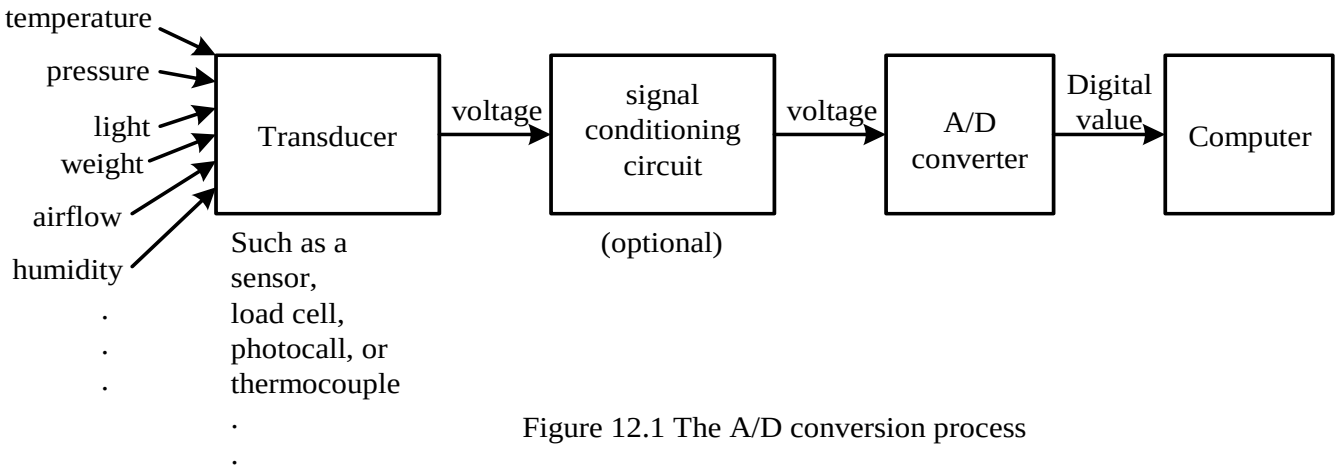


Figure 12.1 The A/D conversion process

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

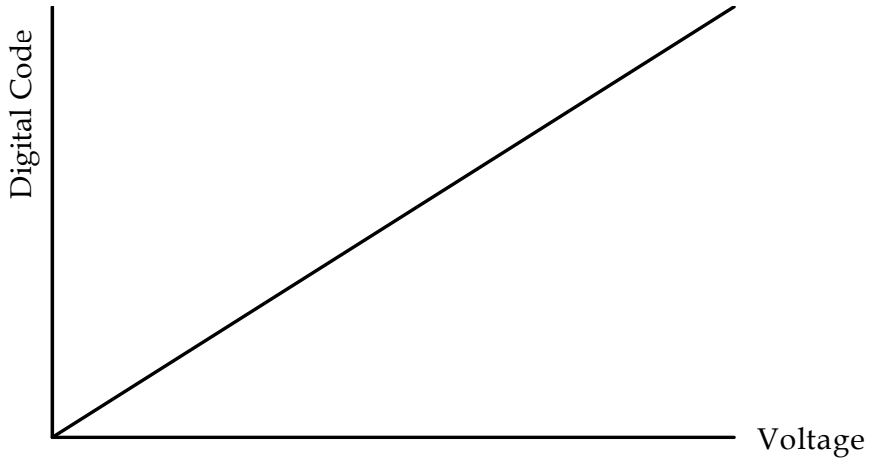


Figure 12.2 An ideal A/D converter output characteristic

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 $V_{DD}/2^n$.
- Average conversion error is $V_{DD}/2^{n+1}$.
- A real A/D converter has nonlinearity.

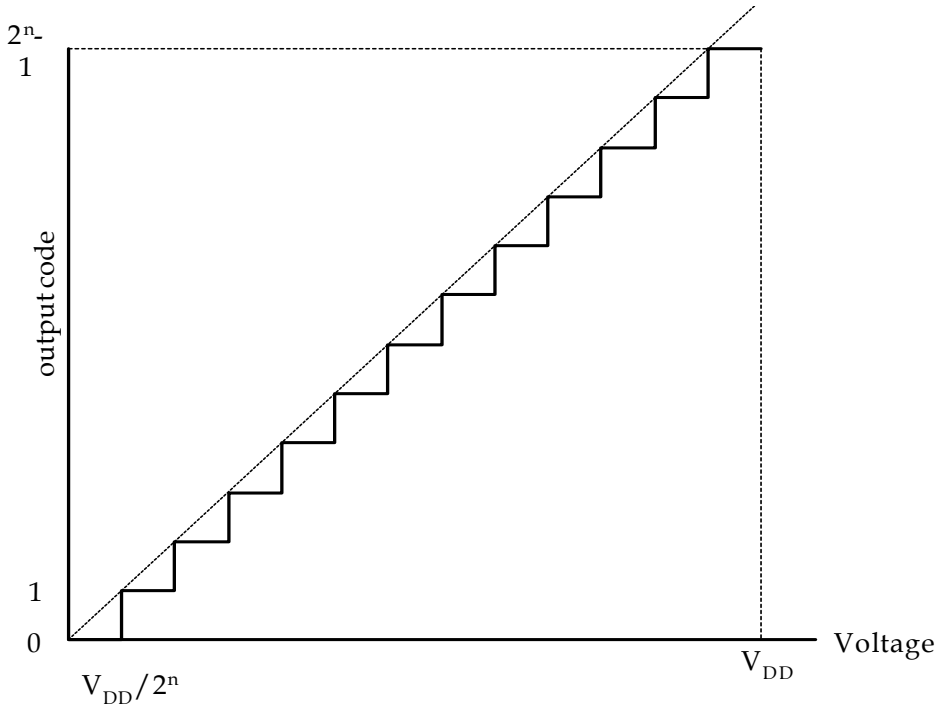


Figure 12.3 Output characteristic of an ideal n-bit A/D converter

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.

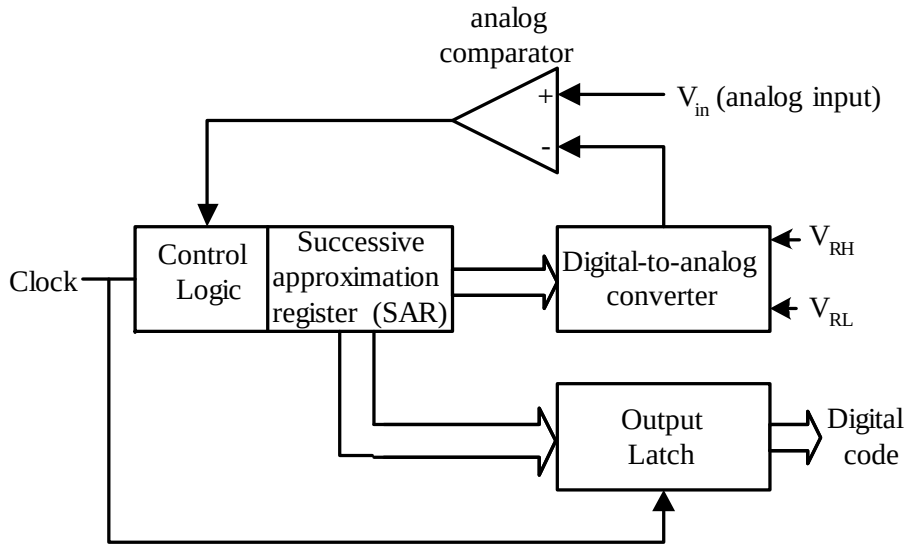


Figure 12.4 Block diagram of a successive approximation A/D converter

Successive Approximation Method

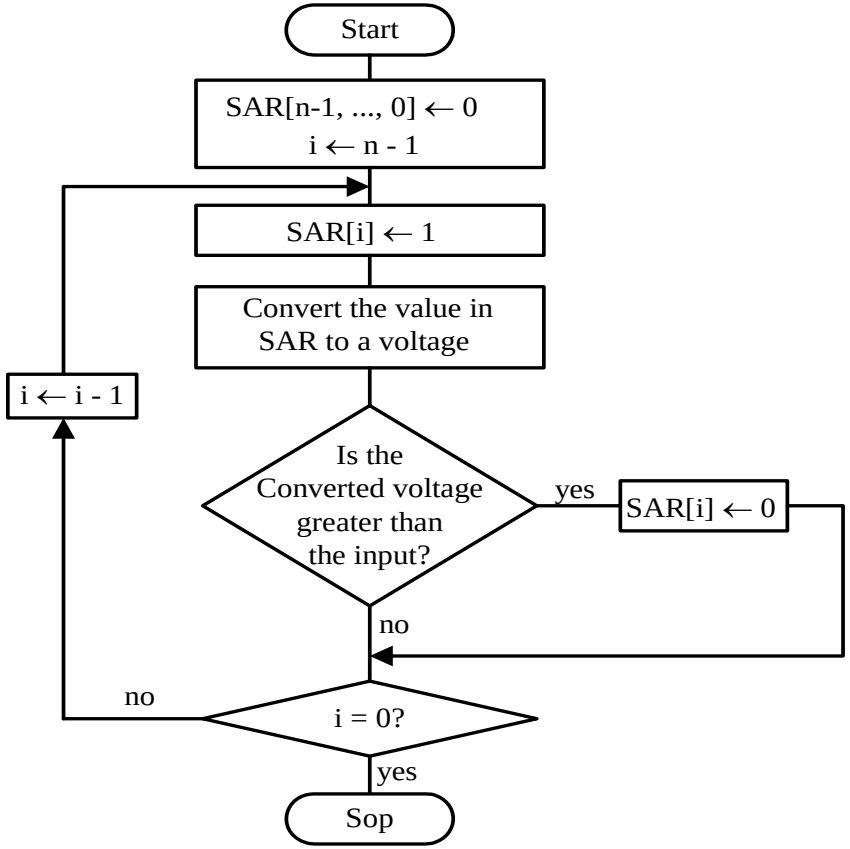


Figure 12.5 Successive approximation A/D conversion method

Optimal Voltage Range for A/D Conversion

- A/D converter requires a low reference voltage (V_{REF-}) and a high reference voltage (V_{REF+}) to perform conversion.
- Most A/D converters are ratiomertic:

  1. An analog input of V_{REF-} 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.
- The output of a transducer should be scaled and shifted to the range of 0V ~ V_{DD} in order to achieve the best accuracy

Example 12.1 Suppose that there is a 10-bit A/D converter with V_{REF-} = 1 V and V_{REF+} = 4V. 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:

$1V + (3 \times 25) \div (210 - 1) = 1.07$ V
$1V + (3 \times 80) \div (210 - 1) = 1.23$ V
$1V + (3 \times 240) \div (210 - 1) = 1.70$ 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

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

- Both R_1 and R_2 must be commercially available resistors.

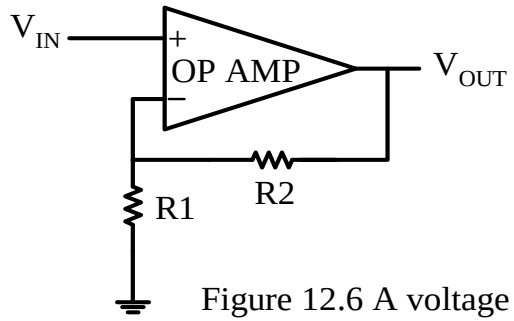


Figure 12.6 A voltage scaler

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 KΩ for R_2 and 10 KΩ for R_1.

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.

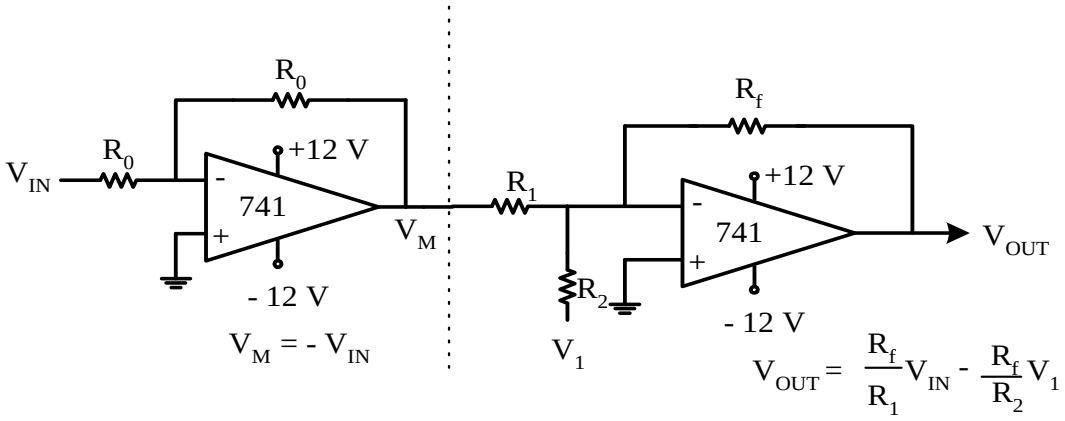


Figure 12T.7 Level shifting and scaling circuit

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 ~ 3.0V to the range of 0V ~ 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 KΩ and V_1 = -5V, solve R_2 = 50 KΩ, and R_f = 12KΩ

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

	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)

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
	--	--	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

Table 12.2 Voltage reference configuration bits

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}-

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

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
	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)

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	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	1/0

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
	--	--	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

	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)

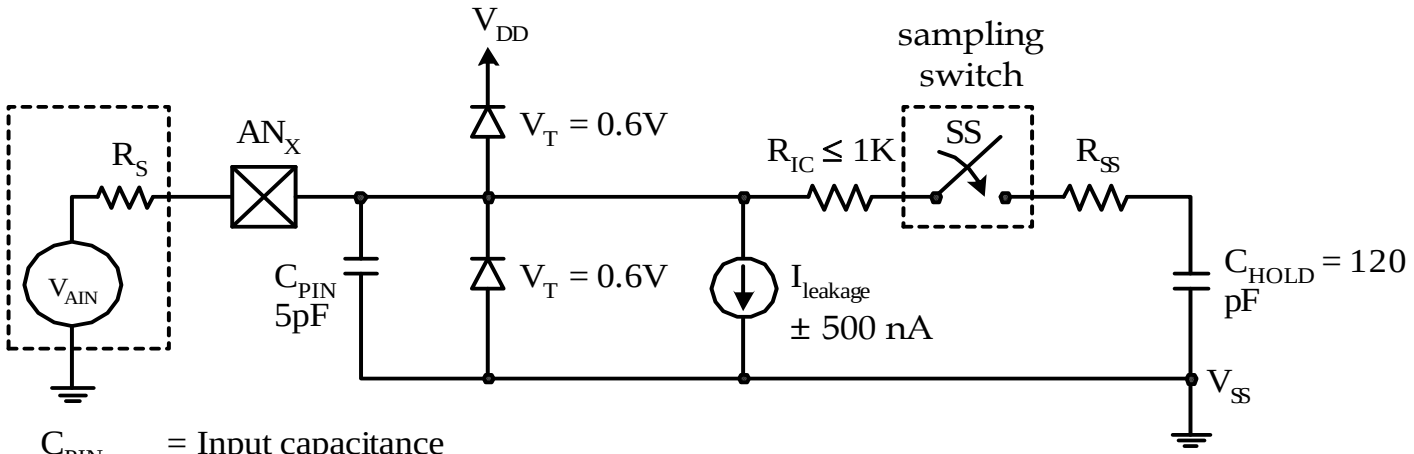
	7	6	5	4	3	2	1	0
	ADFM	--	--	--	--	ADCS2	ADCS1	ADCS0
value after reset	0	0	0	0	0	0	0	0

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)

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.



C_{PIN} = Input capacitance
V_T = threshold voltage
$I_{leakage}$ = leakage current at the pin due to various junctions.
R_{IC} = interconnect resistance
SS = sampling switch
C_{HOLD} = sample/hold capacitance (from DAC). This capacitor has a range from 25 pF to 120 pF.
RSS = sampling switch resistance

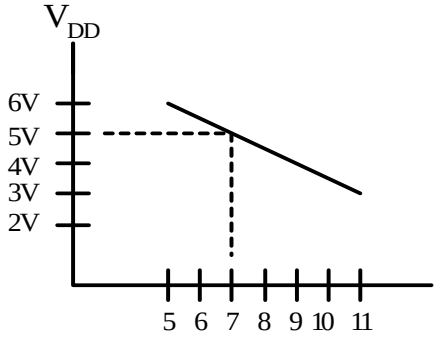


Figure 12.11 Analog input circuit model
(redraw with permission of Microchip)

- The capacitor C_{HOLD} holds the voltage to be converted.
- The required minimum acquisition time T_{ACQ} is computed as follows:

TACQ = amplifier settling time + holding capacitor charging time + temperature coefficient

$= T_{AMP} + T_C + T_{COFF}$

where,

$T_{AMP} = 2\ \mu s$

$T_{COFF} = (temp - 25^oC) \times (0.05\ \mu s/^oC)$, 0 for temp < 25^oC.

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

$= 120\ pF \times (1\ K\Omega + 7K\Omega + 2\ K\Omega) = 9.61\ \mu s$

Automatic Acquisition Time

- For earlier PIC18 members, when the GO/$\overline{\text{DONE}}$ 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/$\overline{\text{DONE}}$ 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 μs.
- 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}	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

A/D Conversion Operation

- For PIC18 members without the feature of automatic acquisition, A/D conversion will start immediately after setting the GO/$\overline{\text{DONE}}$ 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 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 μs.
- f_{OSC} = 32 MHz, the A/D clock source must be set to 64 f_{OSC}, which makes T_{AD} = 2μs.
- 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 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,

```
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
- 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:

```
movlw   0x81          ; select fOSC/64 as the conversion clock
movwf   ADCON0,A      ;   "
movlw   0xCE          ; A/D conversion result right justified
movwf   ADCON1,A      ; configure only AN0 pin as analog,
```

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/$\overline{\text{DONE}}$ 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>
            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
```

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_0REF	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_0REF	VREF+	= VDD, VREF- = VSS
ADC_5ANA_1REF	AN3	= VREF+, VREF- = VSS
ADC_5ANA_0REF	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	AN3	= VREF+
ADC_1ANA_2REF	AN3	= VREF+, AN2 = VREF-, 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	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

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

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_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	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);
      Delay10TCYx(20);          // provides 200 instruction cycles of acquisition time
      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.

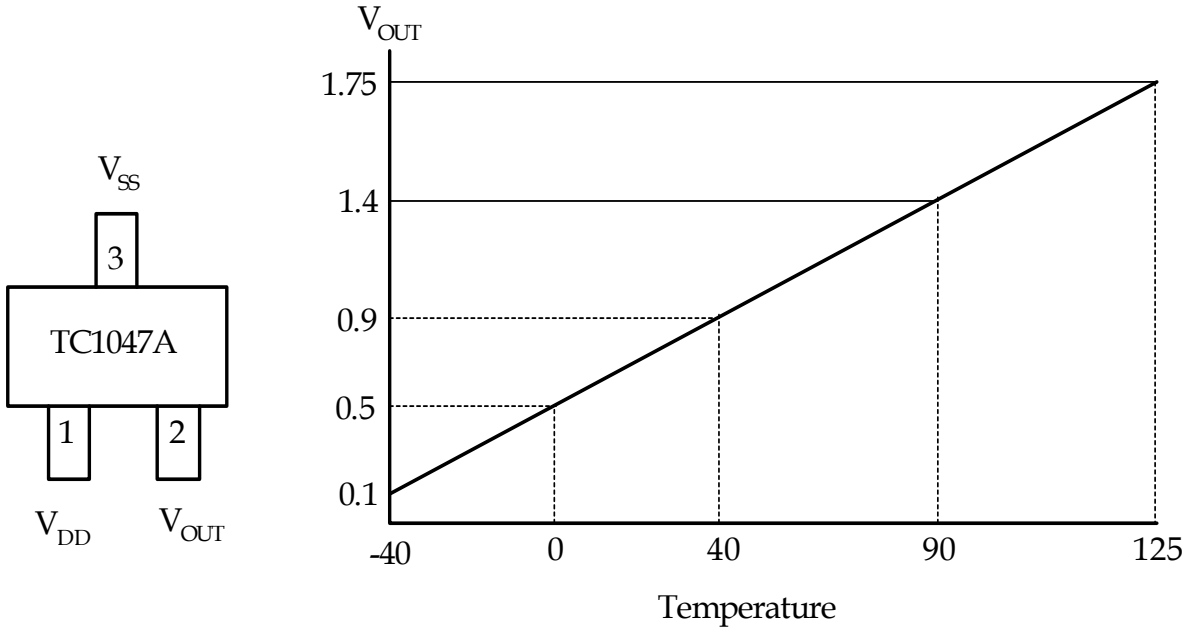


Figure 12.14 TC1047A V_{OUT} vs. temperature characteristic

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^{o}C to +125^{o}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 ~ 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.

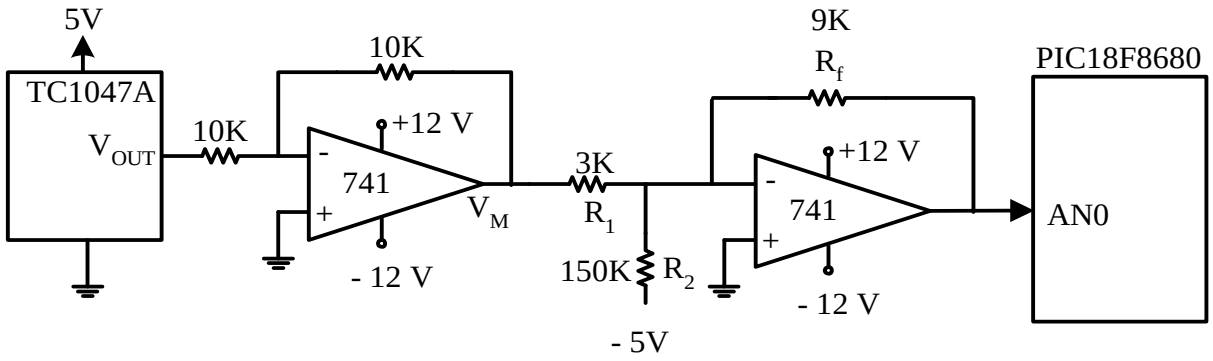


Figure 12.15 Circuit connection between the TC1047A and the PIC18

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
;*****************************************************************
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 space to hold the quotient
rem       set     0x04      ; memory space 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
```

```
            org     0x00            ; reset vector
            goto    start
            org     0x08
            retfie
            org     0x18
            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
forever     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
            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
            mulwf        rem
```

```
            pushr       PRODL           ; push (remainder x 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 initialize 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 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
        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
      }
}
```

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.

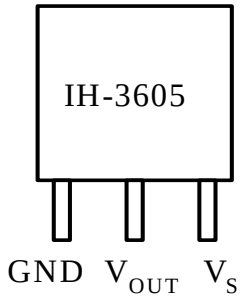


Figure 12.16 Honeywell IH-3605 humidity sensor

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	±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 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.

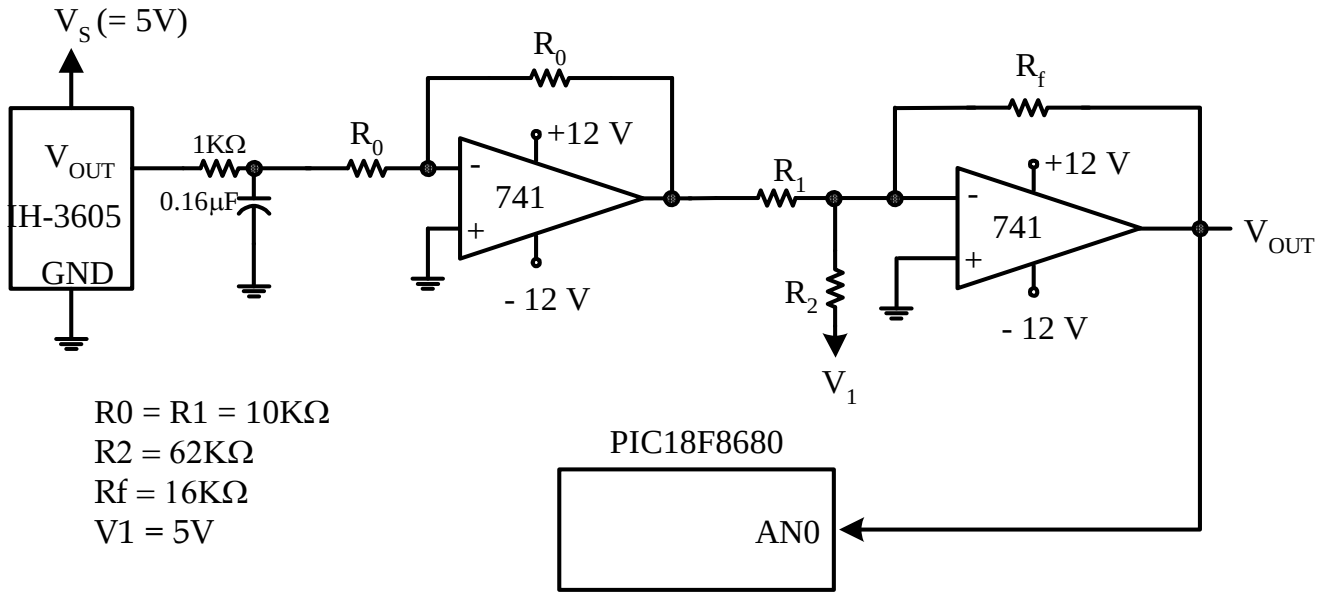


Figure 12.17 Relative humidity measurement circuit

```
#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
        }
}
```

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	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

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.

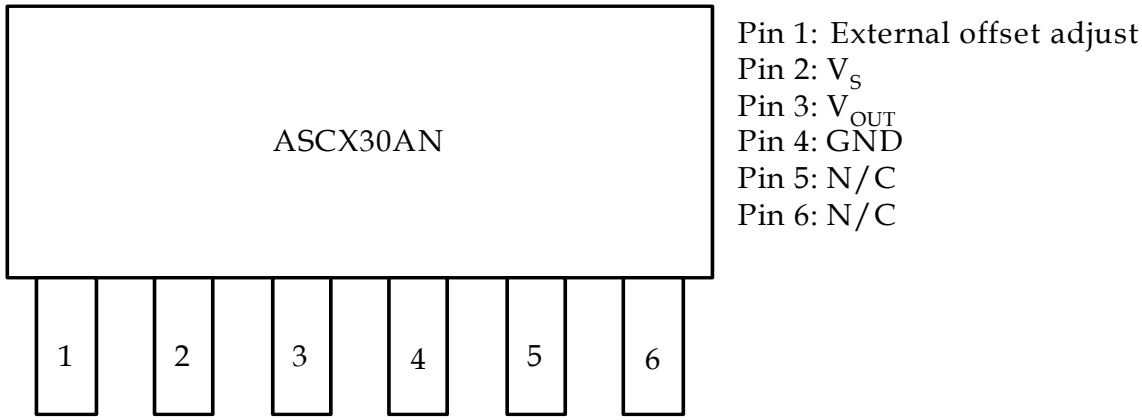


Figure 12.18 ASCX30AN pin assignment

Table 12.7 ASCX30AN performance characteristics (1)

Characteristic	min	typ	max
Pressure range	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.

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.

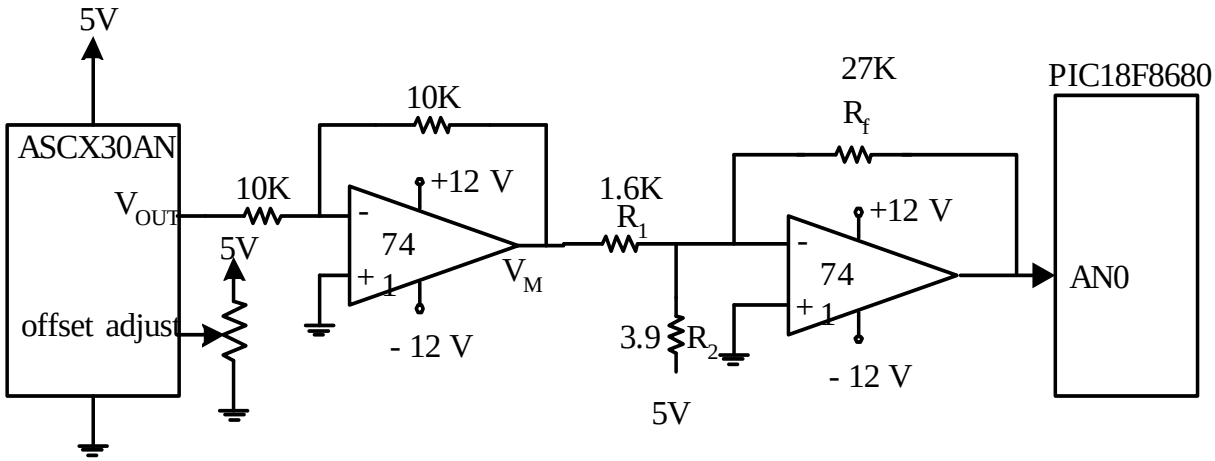


Figure 12.19 Barometric pressure sensor output scaling and shifting circuit.

- 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 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 = 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;
        }
}
```