Chapter 8. A/D Conversion and Data Acquisition

A/D conversion (or analog-to-digital conversion) is to read analog values from, for example, temperature or voltage level, into the PIC chip in the form of digital value. PIC 16F877 has an internal built-in module for this A/D conversion. A/D conversion has many applications: reading from a temperature senor and displaying in on a PC screen of an LCD display; reading from a pressure sensor for blood pressure and acquiring the pressure data into a text file in a PC; reading a current value through an electric wire and alerting a circuit protection device for an above-normal power consumption, etc. In this chapter, we thoroughly study the built-in A/D module and practice of A/D conversion coding with a few practical applications.

1. A/D Conversion Module

A/D conversion is well utilized for external analog signal reading such as voltage, current, temperature, pressure, distance, or even color information. 16F877 has a A/D module. In this chapter, we will study the details of A/D converter module and its application. As need arises, some explanation on general A/D Conversion is discussed occasionally.
There is an 8 channel A-to-D (or A/D) converter module inside a 16F877: AN7 – AN0. These pins are not as well organized as other I/O pins. The lower four channels, AN0 – AN3, are arranged in the pin nos. 2 – 5, and AN4 – AN7 arrange with the pin Nos. of 7 through 10.
The A/D module allows conversion of an analog input signal to a corresponding 10-bit digital number. The output of the sample and hold is the input into the converter, which generates the result via successive approximation.
The analog reference voltages (positive and negative supply) are software selectable to either the device's supply voltages (AVDD, AVss) or the voltage level on the AN3/VREF+ and AN2/VREF-pins.

There are three types of registers we have to well control for A/D conversion. They are: A/D Result Registers (ADRESH and ADRESL), A/D Control Register0 (ADCON0), and A/D Control Register1 (ADCON1). The ADCON0 register controls the operation of the A/D module. The ADCON1 register configures the functions of the port pins. ADRESH and ADRESL registers contain the 10-bit results. Since each register is 8-bit register, we see that only one of the registers would be fully filled while the other would be partially filled by the A/D conversion result. Which register we configure to be fully filled, and which one to be partially filled is controlled by 'result justification': left- or right- justified.

Let's examine ADCON0 register first for the A/D operation.

The first two bits are assigned to select the A/D conversion clock. For correct A/D conversion, as the electrical specification of 16F877 states, the minimum A/D conversion clock must be selected to ensure a minimum of 1.6 μs. With 20MHz crystal oscillation, the pre-scaled clock of $F_{osc}/2$ would be 100 ns, while the pre-scaled clock of $F_{osc}/8$ would be 400ns. So either selection would violate the minimum conversion clock of 1.6 μs. The $F_{osc}/32$ with 1.6 μs would satisfy the minimum clock. However, the internal RC source has typical 6 μs of clock pulse. So, in 20MHz oscillator, selection of RC is safer and may be the only safe option for the A/D clock.

ADCON0 Register (1Fh)

ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	**0**	ADON
						Always 0	

ADCS1:ADCS0: A/D Conversion Clock Select bits
00 = Fosc/2
01 = Fosc/8
10 = Fosc/32
11 = FRC (clock derived from internal A/D RC oscillator)

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
1 = A/D conversion in progress **(Start A/D)**
0 = A/D conversion not in progress

ADON: A/D On bit
1 = A/D converter **On**
0 = A/D converter **Off**

The next three bits select which one channel we use to read analog signal from, external world. Similar channel selection is done in ADCON1 to determine which channels are for analog input and which are for digital I/Os. Anyway, for ADCON0, select one channel you want to read. If you have multiple analog signals, you still have to select one channel for reading and then select another for another reading, etc. CS2:CS0=(000) would select the AN0 for the analog signal reading channel. The second bit (GO/~DONE) indicates the A/D conversion status: 1 indicates the process is still going on and 0 for no process. By setting the bit, we can start the A/D process. This bit is automatically cleared, when a process is finished, there is no need to clear the bit in program code. The last bit ADON works as a switch to turn on/off the A/D module: setting would make the A/D module ready for a conversion process. However, the final say is reserved to the GO/~DONE bit for actually starting the conversion.

For the ADCON1 register, we use only five bits: ADFM and PCFG3:PCFG0.

ADFM is to decide how we store the 10-bit A/D conversion result to the two A/D result registers: ADRESH and ADRESL. When set, the "Right Justification" is selected which stores the 8 LSBs of the result are stored to ADRESL and the 2 MSBs of the results are stored to 2 LSB positions of ADRESH. On the other hand, with its bit cleared, the "Left Justification" is selected which stores the 8 MSBs of the result into ADRESH register and the 2 LSBs of the result to the 2 MSB positions of ADRESL register. See the diagram below for illustration.

ADCON 1Register (9Fh)

----	----	ADFM	----	PCFG3	PCFG2	PCFG1	PCFG0

Read as '0' Read as '0' **Read as '0'**

ADFM: A/D Result format select
1 = Right justified
0 = left justified

PCFG3:PCFG0: A/D Port Configuration Control bits

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

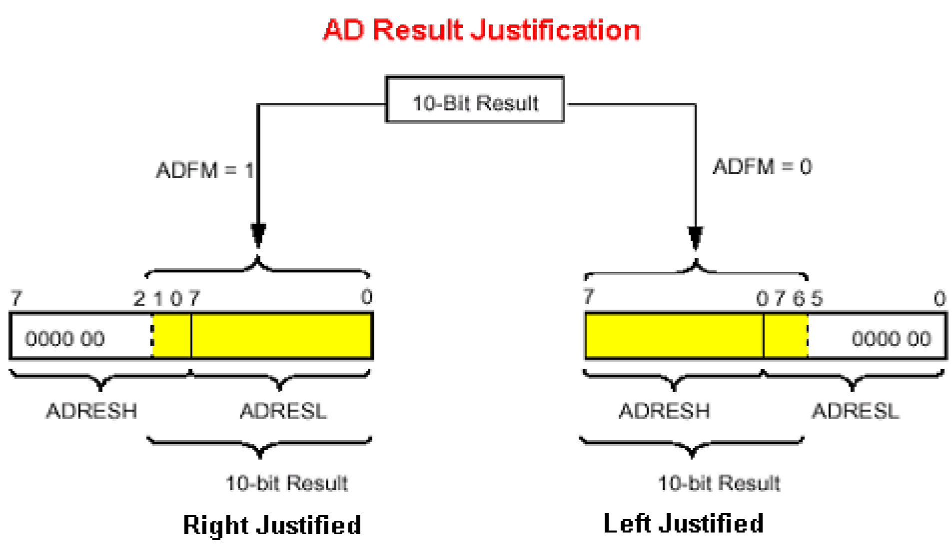


Fig. 58 AD Result Justification

By the way, how much difference do the different justifications make? For example, with left justification, let's assume that we ignore the ADRESL and use only the content of ADRESH. In other words, by this, we ignore the 2 LSBs of the result, and get only the 8 MSBs. Since the last two bits are not used, the resolution would be reduced by 4.

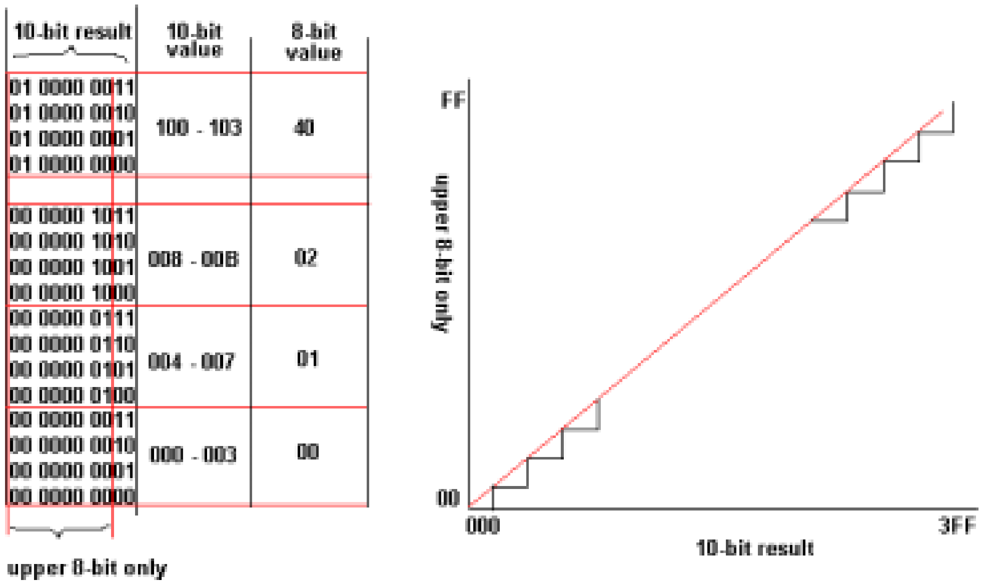


Fig. 59 Graph of FF against 3FF

However, except the resolution, it still can show some reasonable linear relationship of the result. As illustrated below, a value in the range 000 – 003h in 10-bit result would be just 00h in higher 8-bit only. Similarly, any value in the range of 008 – 00Bh in 10-bit result would be just 01h in the ADRESH only scheme with left justification. But coarse may be, the ADRESH only with the left justification still has the enough resolution power of analog value differentiation.

If you need very high resolution, you have to go with all 10-bit result. However, when high resolution is not needed, it is OK with the upper 8-bit result only. However, using all 10 bits is not difficult a matter in programming. It is only a matter of convenience or inconvenience. We will see the actual differences of the above two schemes with actual voltage reading.

The lowest 4 bits are allocated to decide the pin configuration, for analog pins or digital pins. For example, with PCFG3:PCFG0 = 0000, all the pins are assigned as analog input pins, i.e., A/D conversion pins. However, PCFG3:PCFG0=0110 or 0111 would make all the pins as digital I/O pins. Other combinations mix the analog and digital pins of the 8 channels. Another configuration included in these four bits is the selection of positive reference (V_{REF+}) and negative reference voltage (V_{REF-}) for A/D conversion. For example, PCFG3:PCFG0 = 0000 would select the logic power voltage (V_{DD}, or +5V, namely) as V_{REF+} and Ground, V_{SS}, as V_{REF-}. A reference voltage sets the maximum input voltage the A/D converter can convert. In other words, any voltage above the positive reference voltage, or any voltage below the negative reference voltage, would be saturated (or cut-off) to the reference voltage level. Therefore, with PCFG3:PCFG0 = 0000, any negative voltage would be treated as 0 volt or ground, and +5V is the maximum voltage can be converted. When we want to change the reference voltages, and expand or shrink the voltage range of the analog input, we have to select appropriate combination of the 4 bits of ADCON1. For example, PCFG3:PCFG0 = 1000 allocates AN3 and AN2 for V_{REF+} and V_{REF-}, respectively.

The reference voltage, along with the number of bits used for conversion result, determines the step size of the converter, i.e., converter's resolution. For example, with positive reference voltage +5V and negative reference at the ground, the conversion range is 5V. This 5V is now divided by 2^{10} =1024 (maximum binary value of a 10-bit number) steps. Therefore, the step size is 5/1024=0.0048V or 4.88mV. Therefore the weight of each bit of the 10 bit results, therefore, has the multiple of the step voltage: bit 0 represents 2^0x4.88mV=4.88mV; bit 1 represents 2^1x4.88mV=9.76mV; bit 2 for 2^2x4.88mV=19.52mV; and bit 9 for 2^9x4.88mV=2,5V. Therefore, a result of 1000100010 would be interpreted as: $(2^9 + 2^5 + 2^1) \times 0.00488 = 2.664$ V. As the equation shows, the maximum resolution we could get is 4.88mV. The resolution defines the smallest voltage change that can be measured. Every voltage below 4.88mV is read as 0 and any voltage above 4.88mV and below 9.76mV would be read as 4.88mV.

The last two registers involved in the A/D conversion are PIE1 (Peripheral Interrupt Enable 1) register and PIR1 (Peripheral Interrupt Request 1) register. PIE1 register is to grant or deny a peripheral interrupt and PIR1 register indicates the completion of a peripheral's process. From the both registers, we use only the 6^{th} bit (ADIE from PIE1 and ADIF from PIR1) for A/D conversion control. Setting ADIF would trigger an interrupt whenever an A/D conversion is completed. Interrupt will be discussed later. In this chapter, we disable the interrupt for the time being. Clearing ADIE would not trigger an interrupt. Therefore, a completion of the A/D conversion should be checked by a "completion flag" bit. ADIF bit indicates the status of an A/D conversion process: ADIF=1 for completion and ADIF=0 for incompletion. The completion flag bit must be cleared, after a completion of an A/D conversion, by software. Note that we do not use ADIF as an A/D conversion status bit, instead we use GO/~DONE bit of ADCON0 as the conversion status bit. ADIF bit is only to be cleared after GO/~DONE indicates the completion of A/D conversion.

PIE1 REGISTER (8Ch)

PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE

ADIE: A/D Converter Interrupt Enable bit
1 = Enables the A/D converter interrupt
0 = Disables the A/D converter interrupt

PIR1 REGISTER (0Ch)

PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

ADIF: A/D Converter Interrupt Flag bit
1 = An A/D conversion completed
0 = The A/D conversion is not complete

Here goes the A/D conversion procedure in software perspective:
1. Make PORTA as inputs by setting all bits of TRISA register.
2. Disable A/D interrupt by clearing ADIE bit of PIE1 register.
3. Configure ADCON0 register.
4. Configure ADCON1 register.
5. Start A/D conversion by setting GO/~DONE bit of ADCON0 register.
6. Monitor GO/~DONE bit for a completion of the conversion. If the bit is cleared go to 7.
7. Conversion completed. Clear ADIF bit.
8. Move the content of ADRESH to a temporary space.
9. Move the content of ADRESL to another temporary space.

2. First Example of A/D Conversion

Let's have a simple voltage reading example with 16F877 by connecting a variable resistor between the +5V voltage source and the ground. Then connect the wiper terminal, which changes the terminal resistance and the terminal voltage from the ground, to AN0 channel of A/D conversion. We will read the voltage while changing the wiper position and display the value on a PC monitor. The first example is to display with two decimal point value for the voltage at the wiper terminal like. 2.50 or 1.96V by using only 8 MSBs of the result. The second example will use all 10 bit results and display with 3 decimal points like 2.496 or 1.962V.

Since we use only one channel (AN0) with the positive reference voltage and the negative reference voltage as +5 V and the ground, respectively, the configuration of ADCON0 goes like ADCON0=11000001 for internal RC clock, channel 0 (AN0), with A/D switch on. However, no conversion is started yet.

```
banksel     ADCON0          ;KKCCCGXO
movlw       0xC1            ;11000001
movwf       ADCON0          ;initialize ADC (RA0 is ADC port)
```

ADCON0 Register (1Fh)

ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	0	ADON
1	1	0	0	0	0	0	1

RC clock **Channel 0 (AN0)** **A/D Stop** **A/D turned on**

For ADCON1, every bit is cleared to indicate "Left Justification" of the result and channel assignment along with reference voltage levels: ADCON1=00000000. We are going to ignore the 2 LSBs of the result stored in ADRESHL. Instead, we will take only ADRESH as if it comes from 8-bit A/D converter.

ADCON1 Register (9Fh)

		ADFM		PCFG3	PCFG2	PCFG1	PCFG0
1	1	0	0	0	0	0	1
		Left Justified		**All Analog Channel.**		**Vref+ = 5V**	**Vref- = 0V**

```
        movlw       0x00
        banksel     ADCON1
        movwf       ADCON1                  ;PORT A is for ADC channel
                                            ;With LEFT JUSTIFICATION
                                            ;We will ignore two least significant
                                            ;bits without much loss
```

The above lines are just a part of initialization. So now let's discuss about how to actually read and store the data, and then send to a PC for a display.

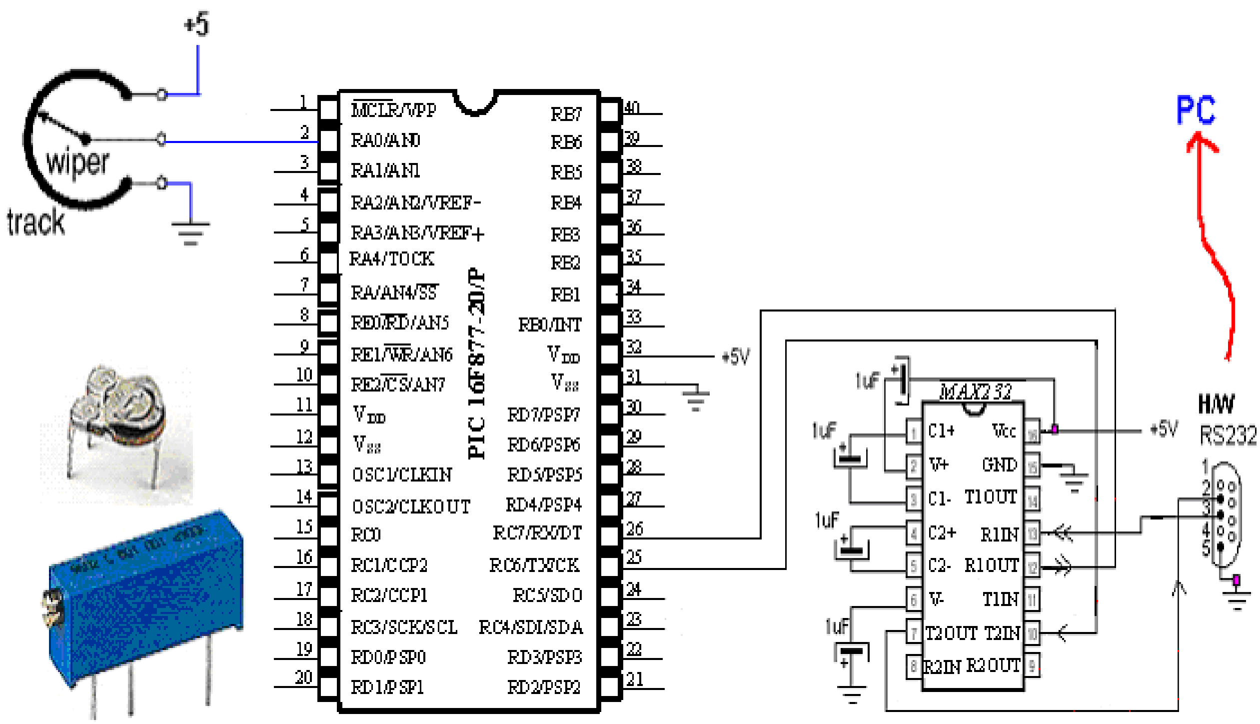


Fig. 60 A/D Conversion Example

Getting a conversion result would be much more convenient in a subroutine form since we are going to convert consecutively. The conversion routine starts from some delay to give the A/D module time to warm up. Then, we GO the conversion, and check the GO/~DONE bit is cleared indicating the completion of the conversion. When the conversion is finished, we clear ADIF bit, then, move the result in ADRESH to a temporary location.

```
;subroutine getADC
;The conversion result will be stored in W register
getADC
```

```
      call        delay10ms          ;warm up
      banksel     ADCON0
      bsf         ADCON0, GO         ;start conversion
ADCloop
      btfsc       ADCON0, GO         ;wait for completion
      goto        ADCloop
      bcf         PIR1, ADIF         ;clear conversion complete flag
      movf        ADRESH,0           ;store the result to W register
      return
```

Now, we have to determine the read value from AN0 by calling `getADC` subroutine. As we discussed in the section of A/D reference voltage, the 10-bit A/D conversion has, for [0, 5]V range, 4.88mV per conversion step. Therefore, the final measured voltage from the ADRESH can be simply formulated by:

$$V_{mea} = B_n \times 2^9 + B_n \times 2^8 + B_n \times 2^7 + B_n \times 2^6 + B_n \times 2^5 + B_n \times 2^4 + B_n \times 2^3 + B_n \times 2^2$$

where, Bn, n = 0, ...7, are the bit values of ADRESH register ignoring the 2 LSBs in ADRESL.

The equation above looks too simple for a high-level language programmer, but it's not that simple in16F877 programming. First, 2^9=512 and 2^8=256 are bigger than 1 byte value, which is the size of calculation and storage in 8-bit microcontroller. Of course, we can split the result into to registers, but still some burden we already feel. Second, after all the burdens we take for the bigger numbers, we still have problem to covert them into decimal point numbers. This problem is much bigger than the first one.

In 16F877 with Assembly language programming environment, it is much wiser to solve a problem by examining the bit pattern. Let's consider the 8-bit excepts as the 8-bit result. Then consider a value of the 8-bit result only when one bit is set.

1000 0000 $=2^7=128$
0100 0000 $= 2^6= 64$
0010 0000 $=2^5=32$
0001 0000 $=2^4=16$
0000 1000 $=2^3=8$
0000 0100 $=2^2=4$
0000 0010 $=2^1=2$
0000 0001 $=2^0=1$

Therefore, when all the bits are set, which occurs when the voltage reading is 5V, the numeric sum would be 255. Since our intention is to display in two decimal point format, we could say the value must correspond to 5.00. Let's name the single digit before the decimal point as D1 (digit 1), and two digits trailing the decimal point D2 (digit 2) and D3 (digit 3). But we can ignore the decimal point in the interpretation of the conversion result. Therefore, when we set the highest voltage as 500 instead of 5.00, there is no change or influence on the conversion result interpretation. The easiest way to convert the numerical values to 3 digit equivalent numbers so that the highest number corresponds to 500 is to double the numerical value for the 3-digit equivalent.

1000 0000 $=2^7=128$ $\rightarrow$ 256
0100 0000 $= 2^6= 64$ $\rightarrow$128
0010 0000 $=2^5=32$ $\rightarrow$064
0001 0000 $=2^4=16$ $\rightarrow$032
0000 1000 $=2^3=8$ $\rightarrow$016
0000 0100 $=2^2=4$ $\rightarrow$008
0000 0010 $=2^1=2$ $\rightarrow$004
0000 0001 $=2^0=1$ $\rightarrow$002

However the sum of the 3 digit equivalents does not add up to 500, because the numerical sum is 255, not 250. The sum reaches at 510. So we need some minor massage around the numbers. What can be acceptable is shown as follows for a conversion of an 8-bit result to a two decimal point voltage value.

1000 0000 $=2^7=128$ $\rightarrow$ 250
0100 0000 $= 2^6= 64$ $\rightarrow$125
0010 0000 $=2^5=32$ $\rightarrow$063
0001 0000 $=2^4=16$ $\rightarrow$032
0000 1000 $=2^3=8$ $\rightarrow$016
0000 0100 $=2^2=4$ $\rightarrow$008
0000 0010 $=2^1=2$ $\rightarrow$004
0000 0001 $=2^0=1$ $\rightarrow$002

Now, we check each bit of the 8-bit result stored in ADRESH. When the LSB is set, for example, the D3 must be increased by 2. If MSB is set, the D1 must be increased by 2. The highest digit D1 has its maximum at 5, so there is no reason to worry if the sum would be bigger 9. For example, an A/D conversion result is reached at ADRESH and its content is 00001110. Since we would have increased D3 by total 14 times, the value of D3 is 14 in decimal. This decimal number must be changed to 4 with carry 1 to the upper digit D2.

How do we automatically find if a sum is bigger than 9 and add the carry to the one upper digit? This procedure could be borrowed from BCD (Binary Coded Decimal) arithmetic. In BCD when sum of two BCD numbers (A digit BCD number occupies 4 binary bits) are bigger than 9, we add 6. For us, we would check the value of, say, D1, and if the value is equal to or greater than 10 (in decimal), we increase the one upper digit, D2, by one, subtract 10 from D1. If the value is 9 or less, we do not do anything at all. The same philosophy can be applied to D2. That means we can build a subroutine to check a value in a digit if it is below 10 or not. As we display 4 bit numbers from 7 – 15 as below, the numbers equal to or greater than 10 have: (bit3=1 and bit2=0) OR (bit3=1 and bit2=0 and bit1=1).

```
;subroutine to check >=10 or <10 =================
;W holds the value of a digit (D 1 or D2)
;HILO is a flag register to indicate the result
; >=10  ---> HILO=0
;<10    ---> HILO =0
TEN
      banksel     HILO
```

```
      clrf        HILO
      movwf       TEMPTEN           ;the content D1 (or D2) now to TEMPTEN
      btfss       TEMPTEN, 0x03     ;3rd bit check (8 or above)
      return                        ;if third bit is zero, it is <10
      btfss       TEMPTEN, 0x02     ;if third bit is 1, check bit 2
      goto        nextbit
      bsf         HILO,0x00         ;if bit3=1 & bit2=1 it is >10
      return
nextbit
      btfss       TEMPTEN,0x01      ;if bit3=1 & bit2=0, then we check bit 1
      return
      bsf         HILO, 0x00        ;if bit3=1 &bit2=0 &bit1=1, it is >10
      return
```

The example code shown below displays the three digit voltage values on a monitor of a PC. In the example, a complete listing is provided. But I warn you here that the initialization of serial communication is not included in the code. In other words, the subroutine `Asynic_mode` which was discussed in Chapter 5 must be included, and it must be called at the very first part of the code. Otherwise, you do not get anything on your screen.

```
;ADC-V1.asm
;
;This program is to read voltage output from a rheostat
; and display the value on a PC terminal (current is updated every 2 seconds)
;
; AN0 is connected to the rheostat wiper
;;
;USE ONLY most significant 8 bits stored in ADRESH
;Max 5.00 V
;min 0.00 V
;
;PC's Hyper Terminal Set-Up: 8N1 19200
;Baud:      19200
;Data Bit:  8
;Parity:    None
;Stop Bit:  1
;Control:   None

        list P = 16F877

STATUS      EQU   0x03
ZERO        EQU   0x02        ;Z flag
TXSTA       EQU   0x98        ;TX status and control
RCSTA       EQU   0x18        ;RX status and control
SPBRG       EQU   0x99        ;Baud Rate assignment
TXREG       EQU   0x19        ;USART TX Register
RCREG       EQU   0x1A        ;USART RX Register
PIR1        EQU   0x0C        ;USART RX/TX buffer status (empty or full)
RCIF        EQU   0x05        ;PIR1<5>: RX Buffer 1-Full  0-Empty
TXIF        EQU   0x04        ;PIR1<4>: TX Buffer 1-empty 0-full
TXMODE      EQU   0x20        ;TXSTA=00100000 : 8-bit, Async
RXMODE      EQU   0x90        ;RCSTA=10010000 : 8-bit, enable port, enable RX
BAUD        EQU   0x0F        ;0x0F (19200), 0x1F (9600)
PORTD       EQU   0x08
TRISD       EQU   0x88
```

```
PORTA       EQU   0x05
TRISA       EQU   0x85
ADCON0      EQU   0x1f
ADCON1      EQU   0x9f
ADRESH      EQU   0x1e        ;High Byte Result
ADRESL      EQU   0x9E        ;Low Byte Result
PIE1        EQU   0x8c
GO          EQU   0x02
ADIE        EQU   0x06
ADIF        EQU   0x06


;DISPLAY FORMAT (with two decimal points)
;
;-----  ---- ----
;|   |  |  | |  |
;|   |  |  | |  |
;-----. ---- ----
;

      CBLOCK      0x20
            temp
            tempten
            HIGHBYTE
            LOWBYTE
            HILO        ;flagging for 1(10 or bigger) or 0 (less than 10)
            DIGIT1
            DIGIT2
            DIGIT3            ;D1. D2 D3 (display format)
            ASCIIreg
            AD1
            AD2
            AD3               ;final 3 ASCII digits to be displayed
            Kount20us
            Kount120us        ;Delay count (number of instr cycles for delay)
            Kount100us
            Kount1ms
            Kount10ms
            Kount1s
            Kount10s
            Kount1m
      ENDC
;=
;=========================================================
      org         0x0000
      GOTO        START
      org         0x05
;=======================================================

START
      movlw       0xFF
      banksel     TRISA
      movwf       TRISA             ;PORTA all inputs

      banksel     PIE1
      bcf         PIE1, ADIE        ;disable ADC interrupt
```

```
      banksel     ADCON0            ;KKCCCGXO
      movlw       0xC1              ;11000001
      movwf       ADCON0            ;initialize ADC (RA0 is ADC port)


      movlw       0x00
      banksel     ADCON1
      movwf       ADCON1            ;PORT A is for ADC channel
                                    ;With LEFT JUSTIFICATION
                                    ;We will ignore two least significant
bits
                                    ;without much loss
AGAIN
      banksel     PIR1
      bcf         PIR1, ADIF  ;clear conversion complete flag

      banksel     TEMP
      clrf        TEMP
      clrf        DIGIT1
      clrf        DIGIT2
      clrf        DIGIT3
      clrf        AD1         ;ASCII code for Digit1
      clrf        AD2         ;ASCII code for Digit2
      clrf        AD3         ;ASCII code for Digit3
      clrf        HILO

      call        GetADC      ;get the AD conversion result
      banksel     TEMP
      movwf       TEMP        ;Now TEMP holds the 8-bit ADC result
;
; No conversion to Max +5.00  Min 0.00 value
;pattern check
; 1000 0000  -->128 --->250
; 0100 0000  -->64  --->125
; 0010 0000  -->32 ---->063
; 0001 0000  -->16 ---->032
; 0000 1000  -->8  ---->016
; 0000 0100  -->4 ----->008
; 0000 0010  -->2 ----->004
; 0000 0001  -->1 ----->002

      movlw       0x00
B0    btfss       TEMP,0x00         ;check the bit 0 of the ADC result
      goto        B1
      incf        DIGIT3            ;DIGIT3=DIGIT3+2
      incf        DIGIT3

B1    btfss       TEMP, 0x01  ;bit 1 check
      goto        B2
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3            ;Digit3=digit3+4

B2    btfss       TEMP, 0x02  ;bit 2 check
      goto        B3
      incf        DIGIT3
```

```
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3            ;
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3            ;Digit3=digit3+8

; check if it is bigger than 10
      movf        DIGIT3, 0         ; to W
      call        TEN
      btfss       HILO,0x00
      goto        B3                ;Less than 10
      movlw       0x0A
      subwf       DIGIT3            ;f - 10 -->f
      incf        DIGIT2            ;Digit2=Digit2+1
      clrf        HILO

B3    btfss       TEMP, 0x03        ;bit 3 check
      goto        B4
      incf        DIGIT3
      incf        DIGIT3            ;
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3            ;Digit3=digit3+6
      incf        DIGIT2            ;Digit2=Digit2+1
; check if it is bigger than 10
      movf        DIGIT3, 0   ; to W
      call        TEN
      btfss       HILO,0x00
      goto        B4                ;Less than 10
      movlw       0x0A
      subwf       DIGIT3
      incf        DIGIT2
      clrf        HILO

B4    btfss       TEMP, 0x04        ;bit 4 check
      goto        B5
      incf        DIGIT3
      incf        DIGIT3            ;Digit3=Digit3+2
      incf        DIGIT2
      incf        DIGIT2
      incf        DIGIT2            ;Digit2=Digit2+3

; check if it is bigger than 10
      movf        DIGIT3, 0         ; to W
      call        TEN
      btfss       HILO,0x00
      goto        B5                ;Less than 10
      movlw       0x0A
      subwf       DIGIT3
      incf        DIGIT2
      clrf        HILO

B5    btfss       TEMP, 0x05        ;bit 5 check
      goto        B6
```

```
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3              ;Digit3=Digit3+3
      incf        DIGIT2
      incf        DIGIT2
      incf        DIGIT2              ;
      incf        DIGIT2
      incf        DIGIT2
      incf        DIGIT2              ;Digit2=Digit2+6
; check if it is bigger than 10
      movf        DIGIT3, 0           ; to W
      call        TEN
      btfss       HILO,0x00
      goto        D2A                 ;Less than 10
      movlw       0x0A
      subwf       DIGIT3
      incf        DIGIT2
      clrf        HILO
; Check DIGIT2 for 10 or above
D2A   movf        DIGIT2, 0
      call        TEN
      btfss       HILO, 0x00
      goto        B6
      movlw       0x0A
      subwf       DIGIT2
      incf        DIGIT1
      clrf        HILO

B6    btfss       TEMP, 0x06          ;bit 6 check
      goto        B7
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3              ;Digit3=Digit3+5
      incf        DIGIT2
      incf        DIGIT2              ;Digit2=Digit2+2
      incf        DIGIT1              ;DIgit1=Digit1+1

; check if it is bigger than 10
      movf        DIGIT3, 0           ; to W
      call        TEN
      btfss       HILO,0x00
      goto        D2B                 ;Less than 10
      movlw       0x0A
      subwf       DIGIT3
      incf        DIGIT2
      clrf        HILO
; Check DIGIT2 for 10 or above
D2B   movf        DIGIT2, 0
      call        TEN
      btfss       HILO, 0x00
      goto        B7
      movlw       0x0A
      subwf       DIGIT2
      incf        DIGIT1
      clrf        HILO
```

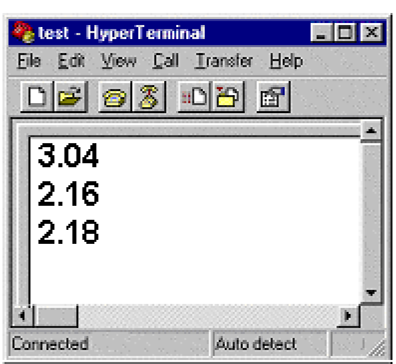
```
B7      btfss       TEMP, 0x07        ;bit 7 check
        goto        FINI
        incf        DIGIT2
        incf        DIGIT2
        incf        DIGIT2
        incf        DIGIT2
        incf        DIGIT2            ;Digit2=Digit2+5
        incf        DIGIT1
        incf        DIGIT1            ;DIgit1=Digit1+2

; check if it is bigger than 10

        movf        DIGIT2, 0
        call        TEN
        btfss       HILO, 0x00
        goto        FINI
        movlw       0x0A
        subwf       DIGIT2
        incf        DIGIT1
        clrf        HILO

FINI
        movf        DIGIT1,0
        call        HTOA        ;ASCII conversion of Digit1
        movwf       AD1         ;final digit to be displayed
        movf        DIGIT2,0
        call        HTOA        ;ASCII conversion of Digit2
        movwf       AD2
        movf        DIGIT3,0
        call        HTOA        ;ASCII conversion of Digit3
        movwf       AD3

;
;ready to display
        movf        AD1, 0      ;First Digit
        call        TXPOLL
        movlw       '.'         ;Decimal Point
        call        TXPOLL
        movf        AD2,0
        call        TXPOLL      ;Second Digit
        movf        AD3,0
        call        TXPOLL      ;Third Digit
        call        CRLF        ;Line Change and Carriage Return
;delay 2s
        call        delay1s
        call        delay1s


        goto  AGAIN

;================================================================
;subroutine to check >=10 or <10 ==================
; >=10  ---> HILO=0
;<10 --->HILO =0
```

```
TEN
      banksel     HILO
      clrf        HILO
      movwf       TEMPTEN
      btfss       TEMPTEN, 0x03     ;3rd bit
      return
      btfss       TEMPTEN, 0x02
      goto        nextbit
      bsf         HILO,0x00
      return
nextbit
      btfss       TEMPTEN,0x01
      return
      bsf         HILO, 0x00
      return
;---------------------------------------------
;Subroutine GetADC ========================
getADC
      call        delay10ms   ;warm up
      banksel     ADCON0
      bsf         ADCON0, GO  ;start conversion
ADCloop
      btfsc       ADCON0, GO  ;wait for conversion to finish
      goto        ADCloop
      bcf         PIR1, ADIF  ;clear conversion complete flag
      movf        ADRESH,0
      return
;----------------------------------------------------------
;RS232 TX subroutine ============
TXPOLL
      banksel     PIR1
      btfss       PIR1, TXIF  ; Check if TX buffer is empty
      goto        TXPOLL
      banksel     TXREG
      movwf       TXREG       ; Place the character to TX buffer
      return
;------------------------
;To send CR and LF ===============
CRLF
      movlw       H'0d'       ;CR
      call        TXPOLL
      movlw       H'0a'       ;LF
      call        TXPOLL
      return
;----------------------------
;; === hex to ascii conversion subroutine
;move the content to W before call this routine
;final result will be stored back to W
HTOA
      movwf       ASCIIreg
;check 0-9 or A-F
      btfsc       ASCIIreg, 0x03    ;0 - 7
      goto        RECHK
THIRTY
      movlw       0x30
      addwf       ASCIIreg
      movf        ASCIIreg,0
```

```
      return

RECHK andlw       0x06  ;
      btfsc       STATUS,ZERO
      goto        THIRTY
      movlw       0x37
      addwf       ASCIIreg
      movf        ASCIIreg,0
      return
;----------------------
;DELAY SUBROUTINES

Delay20us
      banksel     Kount20us
      movlw       H'1F'       ;D'31'
      movwf       Kount20us
R20us decfsz      Kount20us
      goto        R20us
      return
;
;
Delay120us
      banksel     Kount120us
      movlw       H'C5'       ;D'197'
      movwf       Kount120us
R120us
      decfsz      Kount120us
      goto        R120us
      return
;
Delay100us
      banksel     Kount100us
      movlw       H'A4'
      movwf       Kount100us
R100us
      decfsz      Kount100us
      goto        R100us
      return

;
;10ms delay
; call 100 times of 100 us delay  (with some time discrepancy)
Delay10ms
      banksel     Kount10ms
      movlw       H'64' ;100
      movwf       Kount10ms
R10ms call        delay100us
      decfsz      Kount10ms
      goto        R10ms
      return
;
;1 sec delay
;call 100 times of 10ms delay
Delay1s
      banksel     Kount1s
      movlw       H'64'
      movwf       Kount1s
```

```
R1s    call        Delay10ms
       decfsz      Kount1s
       goto        R1s
       return
;
;
;10 s delay
;call 10 times of 1 s delay
Delay10s
       banksel     Kount10s
       movlw       H'0A'       ;10
       movwf       Kount10s
R10s   call        Delay1s
       decfsz      Kount10s
       goto        R10s
       return
;
;1 min delay
;call 60 times of 1 sec delay
Delay1m
       banksel     Kount1m
       movlw       H'3C' ;60
       movwf       Kount1m
R1m    call        Delay1s
       decfsz      Kount1m
       goto        R1m
       return
;=======================================================
         END
;END OF PROGRAM
```

How do you feel about this rather a long line of code for just displaying a simple number? As we see most of the code are devoted to interpretation and display, rather than A/D conversion itself. If you do not have to display the measured voltage, but instead compare with a threshold value, code would be much shorter and simpler. Anyway, see of you have the result like illustrated below as you change the wiper position of the variable resistor.



Now let's use all 10-bit result for the same variable resistor setting we used for 8-bit result calculation. With a similar pattern observation and interpretation, we can relate the 10-bit result to three decimal digit voltage. The maximum voltage (without displaying the decimal point) would be, then, 5000.

10 0000 0000 = 2^9= 512 →2500
01 0000 0000 =2^8=256 →1250
00 1000 0000 =2^7=128 → 0625
00 0100 0000 =2^6= 64 →0312
00 0010 0000 =2^5=32 →0158
00 0001 0000 =2^4=16 →0080
00 0000 1000 =2^3=8 →0040
00 0000 0100 =2^2=4 →0020
00 0000 0010 =2^1=2 →0010
00 0000 0001 =2^0=1 →0005

The only difference in the 3 decimal digit case is that we have to have one more digit value D4 (or digit 4) and its ASCII equivalent (AD4). The subroutines for A/D conversion, check for a digit value if it is below 10 or not, hex to ASCII conversion, and time delay are all the same, except a minor change in the `getADC` subroutine, since we have to store the values of ADRESH and ADRESL. So slightly revised subroutine, `getADC2`, is shown below.

```
;Subroutine GetADC2 =========================
getADC2
      call        delay10ms   ;warm up
      banksel     ADCON0
      bsf         ADCON0, GO  ;start conversion
ADCloop           btfsc ADCON0, GO  ;wait for conversion to finish
      goto        ADCloop
      bcf         PIR1, ADIF  ;clear conversion complete flag
      movf        ADRESH,0
      banksel     tempHIGH
      movwf       tempHIGH          ;tempHIGH <--ADRESH
      banksel     ADRESL
      movf        ADRESL,0
      banksel     tempLOW
      movwf       tempLOW           ;tempLOW <--ADRESL
      return
```

Another slight change in the code is the justification of the A/D conversion result: we select this time "Right Justification" so that lower 8 bits are stored in ADRESL and the upper 2 bits of the results are stored at the lowest 2 LSB positions of ADRESH. The example code, without subroutines, is displayed below.

```
;ADC-V2.asm
;
;This program is to read voltage output from a rheostat
; and display the value on a PC terminal (current is updated every 2 seconds)
;
; AN0 is connected to the rheostat wiper
;
;USE whole 10 bits
;MAX 5.000 V
```

```
;Min 0.000 V
;PC's Hyper Terminal Set-Up: 8N1 19200
;Baud:       19200
;Data Bit:   8
;Parity:     None
;Stop Bit:   1
;Control:    None

        list P = 16F877

STATUS      EQU   0x03
ZERO        EQU   0x02        ;Z flag
TXSTA       EQU   0x98        ;TX status and control
RCSTA       EQU   0x18        ;RX status and control
SPBRG       EQU   0x99        ;Baud Rate assignment
TXREG       EQU   0x19        ;USART TX Register
RCREG       EQU   0x1A        ;USART RX Register
PIR1        EQU   0x0C        ;USART RX/TX buffer status (empty or full)
RCIF        EQU   0x05        ;PIR1<5>: RX Buffer 1-Full  0-Empty
TXIF        EQU   0x04        ;PIR1<4>: TX Buffer 1-empty 0-full
TXMODE      EQU   0x20        ;TXSTA=00100000 : 8-bit, Async
RXMODE      EQU   0x90        ;RCSTA=10010000 : 8-bit, enable port, enable RX
BAUD        EQU   0x0F        ;0x0F (19200), 0x1F (9600)
PORTD       EQU   0x08
TRISD       EQU   0x88
PORTA       EQU   0x05
TRISA       EQU   0x85
ADCON0      EQU   0x1f
ADCON1      EQU   0x9f
ADRESH      EQU   0x1e        ;High Byte Result
ADRESL      EQU   0x9E        ;Low Byte Result
PIE1        EQU   0x8c
GO          EQU   0x02
ADIE        EQU   0x06
ADIF        EQU   0x06


;DISPLAY FORMAT (with three decimal points)
;
; ----  ---- ---- ----
;|   |  |  | |  | |  |
;|   |  |  | |  | |  |
;-----. ---- ---- ----
;
;

      CBLOCK      0x20
            tempHIGH    ;storage space of ADC result
            tempLOW
            tempTEN
            HILO        ;flagging for 1(10 or bigger) or 0 (less than 10)
            DIGIT1
            DIGIT2
            DIGIT3
            DIGIT4      ;D1. D2 D3 (display format)  Double precision
            ASCIIreg
            AD1
```

```
            AD2
            AD3
            AD4          ;final 4 ASCII digits to be displayed
            Kount20us
            Kount120us   ;Delay count (number of instr cycles for delay)
            Kount100us
            Kount1ms
            Kount10ms
            Kount1s
            Kount10s
            Kount1m
      ENDC
;
;==========================================================
      org         0x0000
      GOTO        START
      org         0x05
;=======================================================


START
      movlw       0xFF
      banksel     TRISA
      movwf       TRISA       ;PORTA all inputs

      banksel     PIE1
      bcf         PIE1, ADIE  ;disable ADC interrupt

      banksel     ADCON0
      movlw       0xC1
      movwf       ADCON0      ;initialize ADC (RA0 is ADC port)


      movlw       0x80        ;
      banksel     ADCON1
      movwf       ADCON1      ;PORT A is for ADC channel
                              ;With RIGHT JUSTIFICATION
                              ;ADRESH(B9 and B8)
                              ;ADRESL (B7 - B0)
AGAIN
      banksel     PIR1
      bcf         PIR1, ADIF  ;clear conversion complete flag

      banksel     TEMPHIGH
      clrf        TEMPHIGH
      clrf        TEMPLOW
      clrf        DIGIT1
      clrf        DIGIT2
      clrf        DIGIT3
      clrf        DIGIT4
      clrf        AD1
      clrf        AD2
      clrf        AD3
      clrf        AD4
      clrf        HILO

      call        GetADC2
```

```
                              ;Now tempLOW holds the lower 8-bit ADC result
                              ;tempHIGH for the upper 2 bits
;
; Conversion to Max +5.000  Min 0.000 value
; 98 7654 3210 (bit)
; 10 0000 0000  -->512 --->2500
; 01 0000 0000  -->256 --->1250
; 00 1000 0000  -->128 --->0625
; 00 0100 0000  -->64  --->0312
; 00 0010 0000  -->32 ---->0158
; 00 0001 0000  -->16 ---->0080
; 00 0000 1000  -->8  ---->0040
; 00 0000 0100  -->4 ----->0020
; 00 0000 0010  -->2 ----->0010
; 00 0000 0001  -->1 ----->0005

      movlw       0x00
      banksel     tempLOW
B0    btfss       TEMPLOW,0x00      ;check the bit 0 of the ADC result
      goto        B1
      incf        DIGIT4            ;DIGIT4=DIGIT4+5
      incf        DIGIT4
      incf        DIGIT4
      incf        DIGIT4
      incf        DIGIT4

B1    btfss       TEMPLOW, 0x01     ;bit 1 check
      goto        B2
      incf        DIGIT3            ;Digit3=digit3+1

B2    btfss       TEMPLOW, 0x02     ;bit 2 check
      goto        B3
      incf        DIGIT3
      incf        DIGIT3            ;Digit3=digit3+2

B3    btfss       TEMPLOW, 0x03     ;bit 3 check
      goto        B4
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3            ;Digit3=digit3+4

B4    btfss       TEMPLOW, 0x04     ;bit 4 check
      goto        B5
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3            ;DIGIT3=DIGIT3+8

; check if it is bigger than 10
      movf        DIGIT3, 0   ; to W
      call        TEN
      btfss       HILO,0x00
      goto        B5          ;Less than 10
```

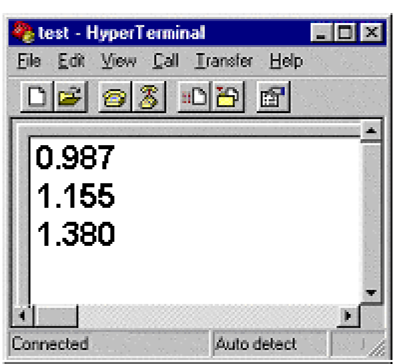
```
      movlw       0x0A
      subwf       DIGIT3
      incf        DIGIT2
      clrf        HILO

B5    btfss       TEMPLOW, 0x05     ;bit 5 check
      goto        B6
      incf        DIGIT4
      incf        DIGIT4
      incf        DIGIT4
      incf        DIGIT4
      incf        DIGIT4
      incf        DIGIT4
      incf        DIGIT4
      incf        DIGIT4            ;Digit4=Digit4+8
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3            ;Digit3=Digit3+5
      incf        DIGIT2            ;Digit2=Digit2+1
; check if it is bigger than 10
      movf        DIGIT4, 0   ; to W
      call        TEN
      btfss       HILO,0x00
      goto        D5A         ;Less than 10
      movlw       0x0A
      subwf       DIGIT4
      incf        DIGIT3
      clrf        HILO
; Check DIGIT2 for 10 or above
D5A   movf        DIGIT3, 0
      call        TEN
      btfss       HILO, 0x00
      goto        B6
      movlw       0x0A
      subwf       DIGIT3
      incf        DIGIT2
      clrf        HILO

B6    btfss       TEMPLOW, 0x06     ;bit 6 check
      goto        B7
      incf        DIGIT4
      incf        DIGIT4            ;digit4=digit4+2
      incf        DIGIT3            ;Digit3=Digit3+1
      incf        DIGIT2
      incf        DIGIT2
      incf        DIGIT2            ;Digit2=Digit2+3


; check if it is bigger than 10
      movf        DIGIT4, 0   ; to W
      call        TEN
      btfss       HILO,0x00
      goto        D6A         ;Less than 10
      movlw       0x0A
      subwf       DIGIT4
```

```
      incf        DIGIT3
      clrf        HILO
; Check DIGIT2 for 10 or above
D6A   movf        DIGIT3, 0
      call        TEN
      btfss       HILO, 0x00
      goto        B7
      movlw       0x0A
      subwf       DIGIT3
      incf        DIGIT2
      clrf        HILO


B7    btfss       TEMPLOW, 0x07      ;bit 7 check
      goto        B8
      incf        DIGIT4
      incf        DIGIT4
      incf        DIGIT4
      incf        DIGIT4
      incf        DIGIT4             ;digit4=digit4+5
      incf        DIGIT3
      incf        DIGIT3             ;digit3=digi3+2
      incf        DIGIT2
      incf        DIGIT2
      incf        DIGIT2
      incf        DIGIT2
      incf        DIGIT2
      incf        DIGIT2             ;Digit2=Digit2+6


; check if it is bigger than 10

      movf        DIGIT4, 0
      call        TEN
      btfss       HILO, 0x00
      goto        D7A
      movlw       0x0A
      subwf       DIGIT4
      incf        DIGIT3
      clrf        HILO

; check if it is bigger than 10

D7A   movf        DIGIT3, 0
      call        TEN
      btfss       HILO, 0x00
      goto        D7B
      movlw       0x0A
      subwf       DIGIT3
      incf        DIGIT2
      clrf        HILO
; check if it is bigger than 10

D7B   movf        DIGIT2, 0
      call        TEN
      btfss       HILO, 0x00
      goto        B8
```

```
      movlw       0x0A
      subwf       DIGIT2
      incf        DIGIT1
      clrf        HILO


B8    btfss       TEMPHIGH, 0x00    ;bit 8 check
      goto        B9
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3
      incf        DIGIT3            ;digit3=digi3+5
      incf        DIGIT2
      incf        DIGIT2            ;Digit2=Digit2+2
      incf        DIGIT1            ;digit1=digit1+1


; check if it is bigger than 10

      movf        DIGIT3, 0
      call        TEN
      btfss       HILO, 0x00
      goto        D8A
      movlw       0x0A
      subwf       DIGIT3
      incf        DIGIT2
      clrf        HILO

; check if it is bigger than 10

D8A   movf        DIGIT2, 0
      call        TEN
      btfss       HILO, 0x00
      goto        B9
      movlw       0x0A
      subwf       DIGIT2
      incf        DIGIT1
      clrf        HILO

B9    btfss       TEMPHIGH, 0x01    ;bit 9 check
      goto        FINI
      incf        DIGIT2
      incf        DIGIT2
      incf        DIGIT2
      incf        DIGIT2
      incf        DIGIT2            ;Digit2=Digit2+5
      incf        DIGIT1
      incf        DIGIT1            ;digit1=digit1+2


; check if it is bigger than 10

      movf        DIGIT2, 0
      call        TEN
      btfss       HILO, 0x00
      goto        FINI
```

```
      movlw       0x0A
      subwf       DIGIT2
      incf        DIGIT1
      clrf        HILO

FINI
      movf        DIGIT1,0
      call        HTOA
      movwf       AD1         ;final digit to be displayed
      movf        DIGIT2,0
      call        HTOA
      movwf       AD2
      movf        DIGIT3,0
      call        HTOA
      movwf       AD3
      movf        DIGIT4,0
      call        HTOA
      movwf       AD4

;
;ready to display
      movf        AD1, 0
      call        TXPOLL
      movlw       '.'
      call        TXPOLL
      movf        AD2,0
      call        TXPOLL
      movf        AD3,0
      call        TXPOLL
      movf        AD4,0
      call        TXPOLL
      call        CRLF  ;Line Change
;delay 2s
      call        delay1s
      call        delay1s

      goto  AGAIN
```

As your run the code while changing the wiper position of the variable resistor, we expect to see the following or similar display.



3. A/D Application to Infrared Ranger for Distance Measurement

IR ranger is a general purpose distance measuring sensor which usually consists of IR emitting dides, position sensitive detector, and signal processing circuit. Basically it measure the distance by the time elapsed between an IR transmission and IR reception.

SHARP's **GP2D12** sensor takes distance reading and reports the distance as an analog voltage with a distance range of 10cm (~4") to 80cm (~30"). The interface is 3-wire with power, ground and the output voltage and requires a JST 3-pin connector which is included with each detector package. This is a common, robust, inexpensive sensor.

As the Distance vs. Voltage curve shows the output voltage is gradually decreased as the distance increases. So even though the specification says that the maximum distance the GD2D12 can measure is 80cm, we could extend the range further, since the voltage further reduces as the distance increases. The big problem of this ranger is that there is one discontinuity point: below 10cm the voltage change is revered to decrease. Therefore, when you have, say, 2.8 V, you are not sure whether the distance is 15cm or 5 cm. When you use this ranger as many do, you have to be very careful that your application platform, robot or vehicle, should not approach an obstacle too close, less than 10 cm. Easiest solution is to give much more clearance from the obstacle, like 30cm, and if the output voltage from the range further increases, then you back off your robot or vehicle.

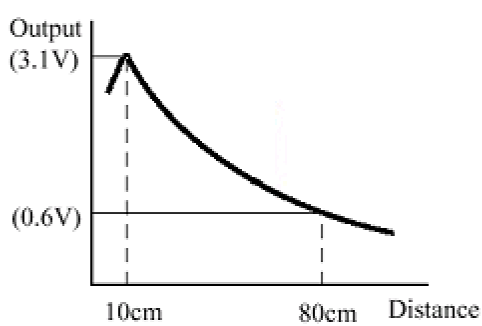


Fig. 61(a) Distance vs. Voltage curve

Fig. 61(b) SHARP GP2D12 Detector

As described, the ranger application is just another example of A/D conversion. Since the maximum voltage is less than +5V, we use the same configuration we used for the variable resistor. Except that we are going to connect the GD2D12 to AN1, instead. Since the voltage-distance relationship is nonlinear, we have to have a kind of table to interpret the voltage we get from A/D conversion to actual distance between the ranger and an obstacle. We will apply the same 8-bit result only approach for this example. Also we will display the distance on a PC monitor. The distance display format is with 3 digits.

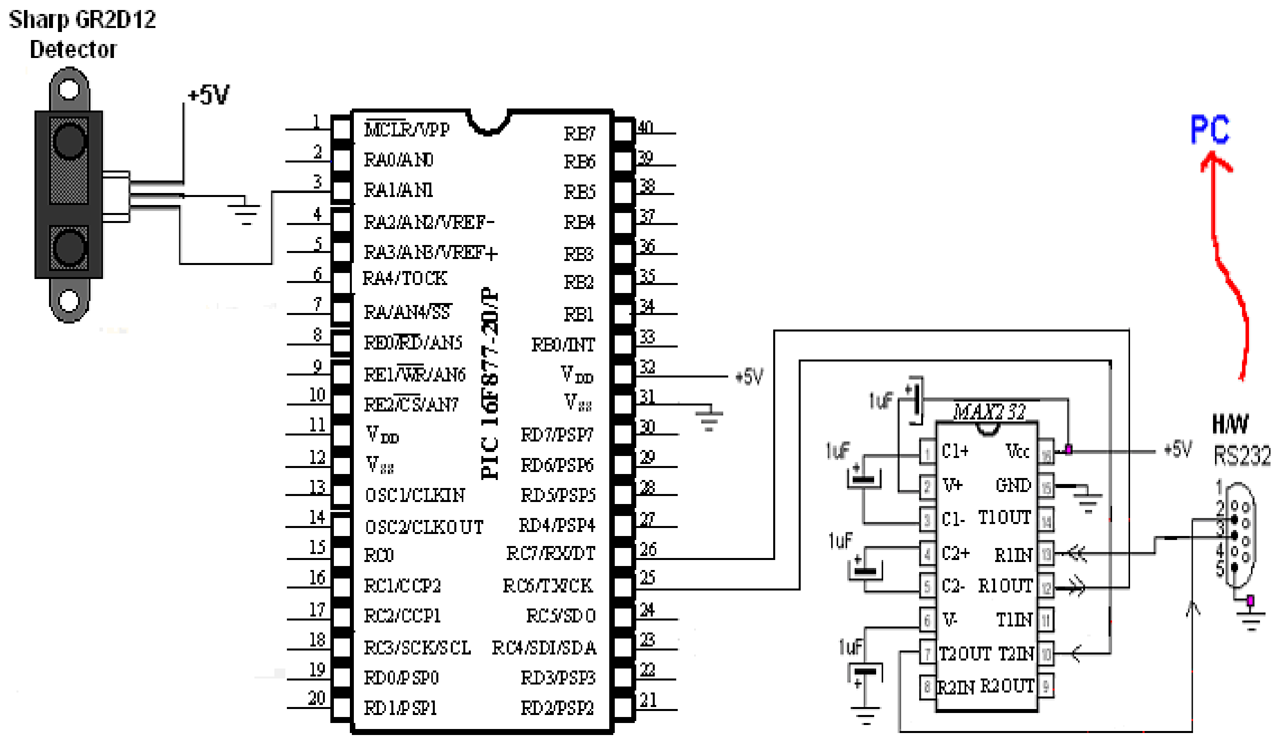


Fig. 62 Sharp GR2D12 Detector connection to PIC 16F877

For the conversion, we use the same logic we used for the variable resistance and the wiper voltage measurement. However the sum of the 3 digit equivalents does not add up to 500, because the numerical sum is 255, not 250. The sum reaches at 510. So we need some minor massage around the numbers. What can be acceptable is shown as follows for a conversion of an 8-bit result to a two decimal point voltage value.

1000 0000 $=2^7=128$ → 2.50 [V]
0100 0000 $= 2^6= 64$ →1.25 [V]
0010 0000 $=2^5=32$ →0.63 [V]
0001 0000 $=2^4=16$ →0.32 [V]
0000 1000 $=2^3=8$ →0.16 [V]
0000 0100 $=2^2=4$ →0.08 [V]
0000 0010 $=2^1=2$ →0.04 [V]
0000 0001 $=2^0=1$ →0.02 [V]

From the test of the output voltage vs. distance using an oscilloscope and an obstacle, we found that nonlinear relationship of voltage and current as follows.

Distance [cm]	10	20	40	50	60	80	100	>100
Output Voltage [V]	2.4	1.25	0.7	0.58	0.5	0.42	0.34	<0.33

For 2.4 V for 10cm, since we are not aiming for very accurate (actually we cannot do that with

the ranger), we use the value of 1.25V as an approximate value with the bit pattern of 10000000. Also, since the 10cm is the closest distance measurement, higher than this value can be ignore. For 1.25V for 20cm matches well with the bit pattern of 01000000. The 0.7 V for 40cm can also be approximated by 0.63V or bit pattern of 00100000. However, making 0.58V for 50cm is impossible by just one bit information. Instead, since 0.58 = 0.32+0.16+0.8, we can use the three bit information for 40 cm: 00011100 as a conclusion. In a similar manner, we can have the following simple distance interpretation pattern from the output voltage of the ranger.

```
1xxx xxxx  -->128 --->250----->010cm
01xx xxxx  -->64  --->125----->020cm
001x xxxx  -->32 ---->063----->040cm
0001 11xx  -->28 ---->056----->050cm
0001 10xx  -->24 ---->048----->060cm
0001 01xx  -->20 ---->040----->080cm
0001 00xx  -->16 ---->032----->100cm
0000 xxxx  -->8  ---->016 ---->Out of Range
```

The Pseudo-Code for voltage to distance interpretation is shown below. The label 'cm' followed by a three digit number indicates the place for displaying the distance of the number. The label 'cmqqq' is for displaying out of range distance situation.

```
            if B7=1 goto cm010, else goto B         ;<B7>=1 for 10cm
B:          if B6=1 goto cm020, else goto C         ;<B7:B6>=01 for 20cm
C:          if B5=1 goto cm040, else goto D         ;<B7:B5>=001 for 40cm
D:          if B4=1 goto D1, else goto E
      D1:   if B3=1 goto D2, else goto D3
      D2:   if B2=1 goto cm050, else goto cm060     ;<B7:B2>=000111 for 50cm
                                                    ;<B7:B2>=000110 for 60cm
      D3:   if B2=1 goto cm080, else goto cm100     ;<B7:B2>=000101 for 80cm
                                                    ;<B7:B2>=000100 for 100cm
E:          goto cmqqq                              ;<B7:B4>=0000 for Out of Range
```

The following example code is a full program for Sharp GP2D12 without listing subroutine. The subroutines needed this code are the same ones we used in the first example of A/D conversion reading the wiper voltage from the variable resistor.

```
;GP2D12.asm
;
;This program is to read voltage output from
; a Sharp Ranger GP2D12D
; and display the value on a PC terminal (updated every half second)
;
; AN1 is connected to the Ranger
;;
;USE ONLY most significant 8 bits
;Max 5.00 V
;min 0.00 V
;
```

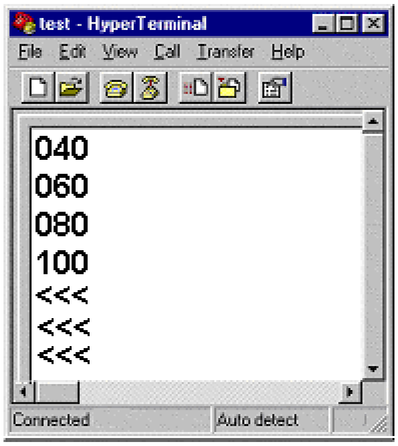
```
        list P = 16F877

STATUS      EQU   0x03
ZERO        EQU   0x02        ;Z flag
TXSTA       EQU   0x98        ;TX status and control
RCSTA       EQU   0x18        ;RX status and control
SPBRG       EQU   0x99        ;Baud Rate assignment
TXREG       EQU   0x19        ;USART TX Register
RCREG       EQU   0x1A        ;USART RX Register
PIR1        EQU   0x0C        ;USART RX/TX buffer status (empty or full)
RCIF        EQU   0x05        ;PIR1<5>: RX Buffer 1-Full  0-Empty
TXIF        EQU   0x04        ;PIR1<4>: TX Buffer 1-empty 0-full
TXMODE      EQU   0x20        ;TXSTA=00100000 : 8-bit, Async
RXMODE      EQU   0x90        ;RCSTA=10010000 : 8-bit, enable port, enable RX
BAUD        EQU   0x0F        ;0x0F (19200), 0x1F (9600)
PORTD       EQU   0x08
TRISD       EQU   0x88
PORTA       EQU   0x05
TRISA       EQU   0x85
ADCON0      EQU   0x1f
ADCON1      EQU   0x9f
ADRESH      EQU   0x1e        ;High Byte Result
ADRESL      EQU   0x9E        ;Low Byte Result
PIE1        EQU   0x8c
GO          EQU   0x02
ADIE        EQU   0x06
ADIF        EQU   0x06


;DISPLAY FORMAT
;
;XXX CM  (3 digit)
;
;
      CBLOCK      0x20
            temp
            tempten
            HIGHBYTE
            LOWBYTE
            HILO        ;flagging for 1(10 or bigger) or 0 (less than 10)
            DIGIT1
            DIGIT2
            DIGIT3            ;D1. D2 D3 (display format)  Double precision
            ASCIIreg
            AD1
            AD2
            AD3         ;final 3 ASCII digits to be displayed
            Kount20us
            Kount120us   ;Delay count (number of instr cycles for delay)
            Kount100us
            Kount1ms
            Kount10ms
            Kount100ms
            Kount500ms
            Kount1s
            Kount10s
```

```
            Kount1m
      ENDC
;
;The Next
;Bootloader first execute the first 4 addresses
;==========================================================
      org         0x0000
      goto        START
      org         0x05
;=======================================================
START
      movlw       0xFF
      banksel     TRISA
      movwf       TRISA       ;PORTA all Inputs

      banksel     PIE1
      bcf         PIE1, ADIE  ;disable ADC interrupt

      banksel     ADCON0            ;KKCCCGXO
      movlw       0xC9              ;11001001
      movwf       ADCON0            ;initialize ADC (AN1 is ADC port)


      movlw       0x00
      banksel     ADCON1
      movwf       ADCON1            ;PORT A is for ADC channel
                                    ;With LEFT JUSTIFICATION
                                    ;We will ignore two least significant
bits
                                    ;without much loss
AGAIN
      banksel     PIR1
      bcf         PIR1, ADIF  ;clear conversion complete flag

      banksel     TEMP
      clrf        TEMP
      clrf        AD1
      clrf        AD2
      clrf        AD3

      call        GetADC      ;voltage reading
      banksel     TEMP
      movwf       TEMP        ;Now TEMP holds the 8-bit ADC result
      movlw       '0'
      movwf       AD3         ;AD3=0
      movwf       AD1         ;AD2=0
      movwf       AD2         ;Ad1=0
                              ;Distance = 000 now
;
; Max +5.00  Min 0.00
;Conversion to Distance (See the nonlinear graph for GP2D12)
;Experimental results
;
; 2.70 [V]   9 cm
; 2.4       10
; 1.25            20
; 1.1       25
```

```
;0.9        30
;0.8        35
;0.7        40
;0.62       45
;0.58       50
;0.52       55
;0.5        60
;0.48       65
;0.46       70
;0.44       75
;0.42       80
;0.4        85
;0.38       90
;0.36       95
;0.34       100
;0.32       110
;0.3        140
;0.28       150

;FROM ADC with LEFT JUSTIFIED
; 1 0000 0000  -->256 --->500
; 0 1xxx xxxx  -->128 --->250----->010cm
; 0 01xx xxxx  -->64  --->125----->020cm (B)
; 0 001x xxxx  -->32 ---->063----->040cm (C)
; 0 0001 11xx  -->28 ---->056----->050cm (D, D1)
; 0 0001 10xx  -->24 ---->048----->060cm (D2)
; 0 0001 01xx  -->20 ---->040----->080cm (D3)
; 0 0001 00xx  -->16 ---->032----->100cm  (D3)
; 0 0000 xxxx  -->8  ---->016 ---->Out of Range (E)


;Pseudo-Code
;
; if B7=1 goto cm010, else goto B
; B: if B6=1 goto cm020, else goto C
; C: if B5=1 goto cm040, else goto D
; D: if B4=1 goto D1, else goto E
;   D1: if B3=1 goto D2, else goto D3
;   D2: if B2=1 goto cm050, else goto cm060
;   D3: if B2=1 goto cm080, else goto cm100
; E: goto cmqqq


      btfss       TEMP, 0x07
      goto        BB
      goto        cm010
BB    btfss       TEMP, 0x06
      goto        CC
      goto        cm020
CC    btfss       TEMP, 0x05
      goto        DD
      goto        cm040

DD    btfss       TEMP, 0x04
      goto        EE
D1    btfss       TEMP, 0x03
      goto        D3
```

```
      goto        D2

D2    btfss       TEMP, 0x02
      goto        cm060
      goto        cm050

D3    btfss       TEMP, 0x02
      goto        cm100
      goto        cm080

EE    goto        cmqqq             ;end of interpretation



cm010 movlw       '1'
      movwf       AD2         ;Distance = 010
      goto        FINI
cm020 movlw       '2'
      movwf       AD2         ;Distance = 020
      goto        FINI
cm040 movlw       '4'
      movwf       AD2         ;040
      goto        FINI
cm050 movlw       '5'
      mmovwf      AD2         ;050
      goto        FINI
cm060 movlw       '6'
      movwf       AD2         ;060
      goto        FINI
cm080 movlw       '8'
      movwf       AD2         ;080
      goto        FINI
cm100 movlw       '1'
      movwf       AD1         ;100
      goto        FINI
cmqqq movlw       '>'
      movwf       AD1
      movwf       AD2
      movwf       AD3         ;>>> to indicate out-of-range

FINI

;ready to display
      movf        AD1, 0
      call        TXPOLL
      movf        AD2,0
      call        TXPOLL
      movf        AD3,0
      call        TXPOLL
      call        CRLF  ;Line Change
;delay 2s
      call        delay100ms


      goto        AGAIN       ;repeat every 100ms
```

We expect to see the following display when we move an object away from the ranger.



4. Current Measurement Applications using A/D converter Module

As the last example of A/D conversion in data acquisition, a few current sensors are introduced here. Some are good for smaller current and others are better for rather a larger current. A simple thermister is also introduced for a simple temperature measurement using the A/D module.

CSA-1V hall effect current sensor measures a current through itself by the principle of the magnetic field generated by the current flow. The amount of the current will be translated by the chip and the corresponding value in voltage will be produced.

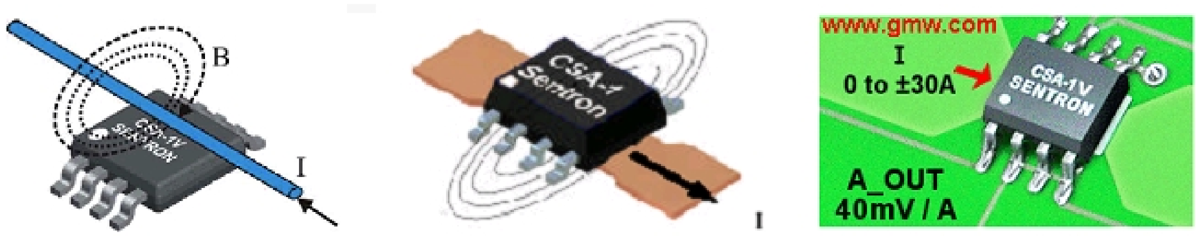


Fig. 63 How the CSA-1V Sentron works

According to the datasheet, when the current carrying conductor is about 0.1mm, 10A will produce 4V and the output terminal of the sensor chip. 5A would produce about 3.2 V.

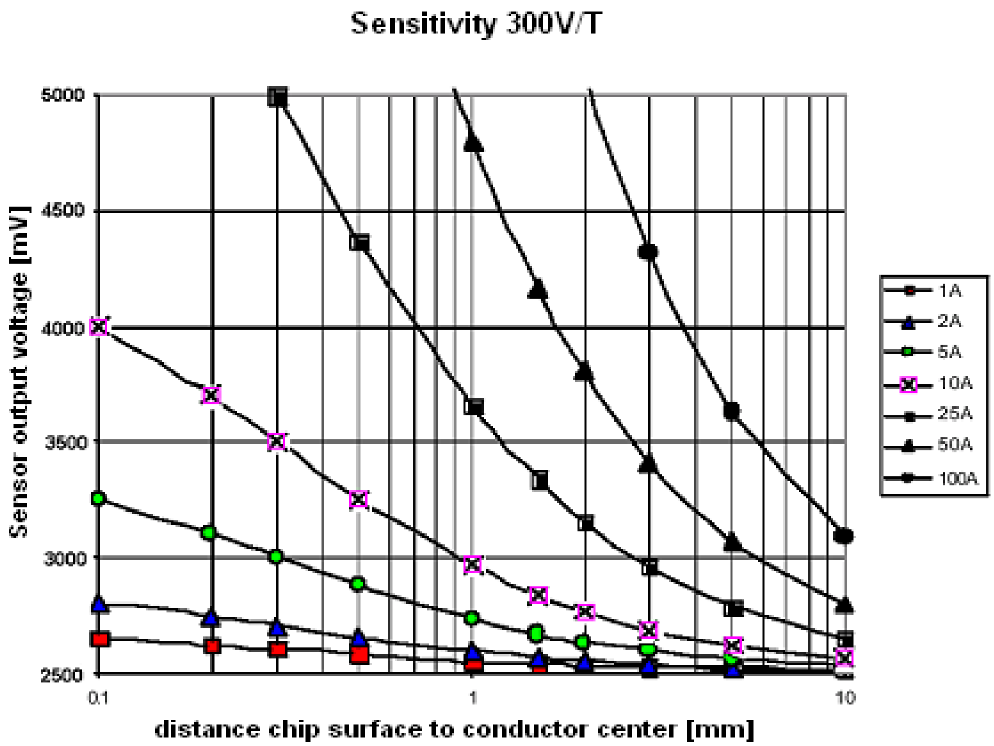


Fig. 64 Sensitivity graph

With the following connection, we can measure the current through the wire using the AN1 channel of 16F877. We apply the same logic and example code. The wire indicated with different color and different thickness (or AWG number) do not indicate the actual different in the conductor type on top of the sensor chip. We can use any wire, which can stand the maximum expected current flowing through, for the measurement. No special wire type is needed to place it on top (or bottom) of the sensor chip. However, you may want to use insulated wire to prevent an electric shock or electrocution by accidentally touching a bare wire carrying very high current.

There are two ways to get the output voltage for current measurement, if we get the output between pin#1 A_out and the ground, as connected in the drawing, the voltage range is [0, +5]V. However, if you use the pin#8 CO_out as internal reference, we can read a differential output voltage between pin #1 and pin#8, in which case the output range is [-2.5, +2.5]V.

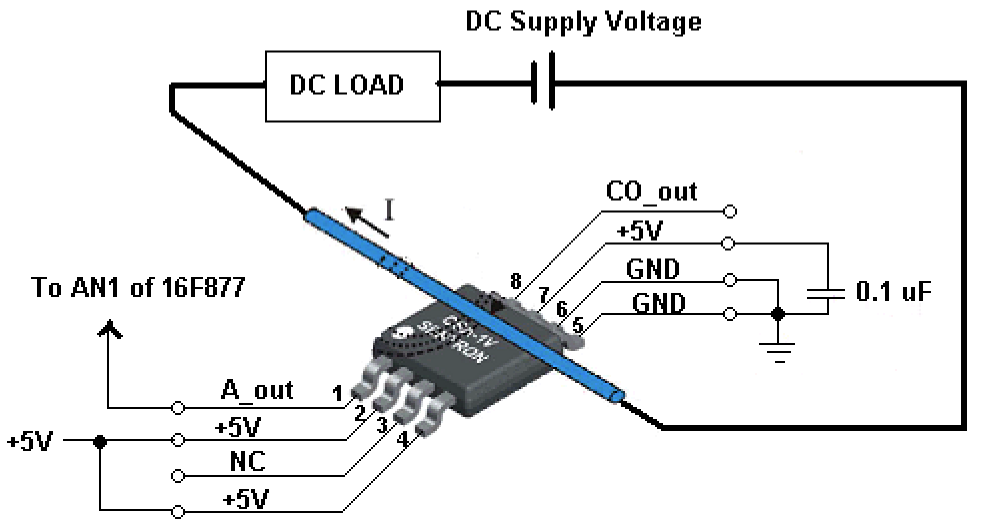


Fig. 65 CSA-1V Sentron connection to get output voltage

The next current sensing device is Maxim IC's MAX471/MAX472. MAX471 is a bidirectional, high-side current-sense amplifiers for portable PCs, telephones, and other systems where battery/DCpower-line monitoring is critical. High-side power-line monitoring is especially useful in battery-powered systems, since it does not interfere with the ground paths of the battery chargers or monitors often found in "smart" batteries.

The MAX471 has an internal 35mΩ current-sense resistor and measures battery currents up to ±3A. For applications requiring higher current or increased flexibility, the MAX472 functions with external sense and gain-setting resistors. Both devices have a current output that can be converted to a ground-referred voltage with a single resistor, allowing a wide range of battery voltages and currents.

An open-collector SIGN output indicates current-flow direction, so the user can monitor whether a battery is being charged or discharged. Both devices operate from 3V to 36V, draw less than 100µA over temperature, and include a 18µA max shutdown mode.

With the following connection, with 2KΩ resistor at the output side, the current-to-voltage conversion ration is 1V/A. Therefore, I V reading means 1 A current flow from the battery. The SIGN pin is to indicate the current flow direction. The Low level indicates that current flows from RS- to RS+.

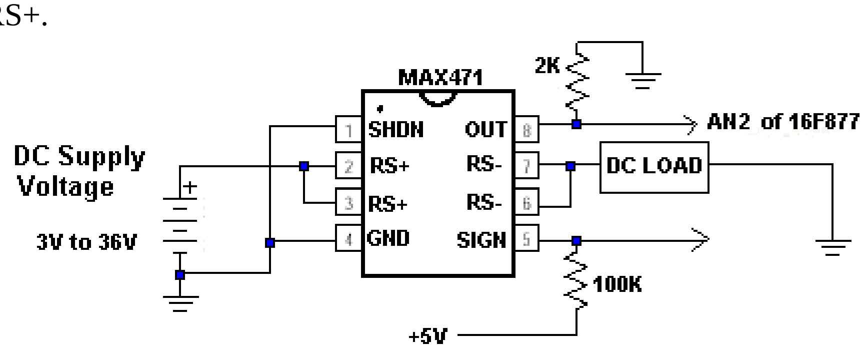


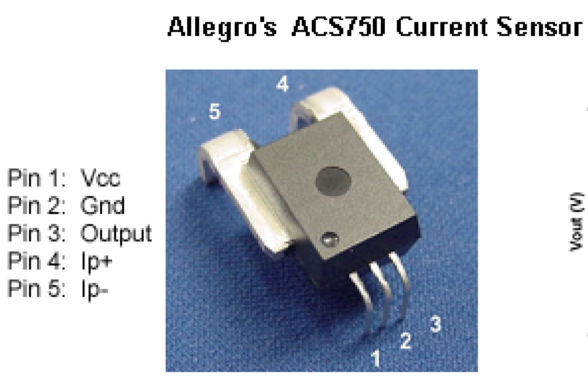
Fig. 66 Maxim IC's MAX471

In the connection diagram above, since the current flows, out of the DC supply voltage source, from RS+ to RD- to the LOAD to the ground, the SIGN output must be High. When we do not need the flow direction information, leave the pin open.

Another current sensor is Allegro MicroSystem's ACS750, a fully integrated current sensor. The Allegro ACS750 family of current sensors provides economical and precise solutions for current sensing in industrial, commercial, automotive, and communications systems. The device package allows for easy implementation by the customer. Typical applications include motor control, load detection and management, switched mode power supplies and over-current fault protection.

The sensor consists of a precision linear Hall IC optimized to an internal magnetic circuit to increase device sensitivity. The primary conductor used for current sensing (terminals 4 and 5) is designed for extremely low power loss. The power terminals are also electrically isolated from the sensor leads (pins 1 – 3). This allows the ACS750 family of sensors to be used in applications requiring electrical isolation without the use of opto-isolators or other costly isolation techniques.

As we see the current vs. output voltage curve, they are linearly related in the range of [0, +5]V for the current range of [-50, +50]A.



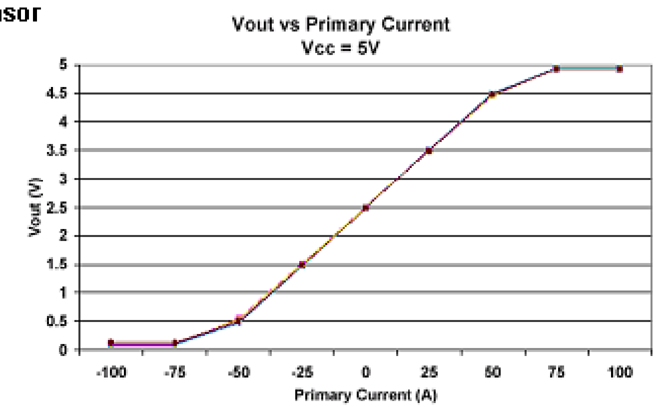


Fig. 67(a) Allegro MicroSystem's ACS750 Fig. 67(b) Graph of Volt vs. Primary Current

The following connection would read the current through the DC load in terms of voltage at the output pin #3, which is measured by the AN2 channel of the A/D conversion module. We use the same code we already examined for a wiper voltage measurement of a variable resistor.

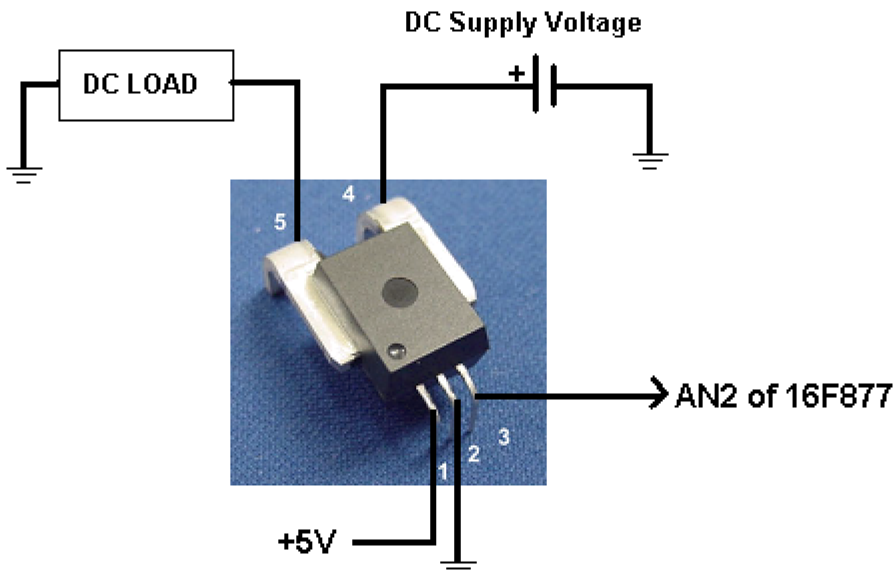


Fig. 68 ACS750 connection to read current through DC load

The last item we apply the same A/D conversion is a thermistor, or thermally sensitive resistor. The resistance value of the thermistor changes according to temperature. This part is used as a temperature sensor. There are three types of thermistors: negative temperature coefficient thermistor (NTC), positive temperature coefficient thermistor (PTC), and critical temperature resister thermistor (CTR). NTC decreases its resistance value continuously as temperature rises. PTC increases its resistance value suddenly when temperature rises above a specific point. On the other hand, CTR decreases its resistance value when temperature rises above a certain point.

The thermistor we examine is a very small NTC thermistor, A170. Cool resistance is about 1.5KΩ and hot resistance is about 25 Ω. We may need some type of calibration to accurately convert the resistance of the thermistor to the ambient temperature. The following connection would get about 2.5V for normal temperature. As the temperature rises, the output voltage would also rise. Test the voltage output with the example code we studied for the wiper voltage of a variable resistor.

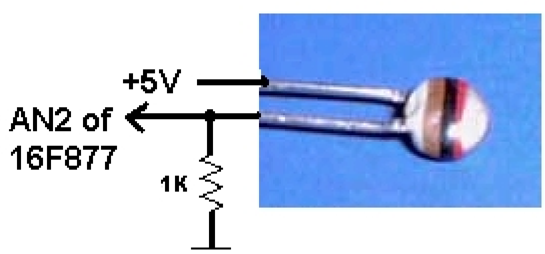


Fig. 69 NTC thermistor A170 connection to PIC 16F877