A Simple Voltmeter

CDR Charles B. Cameron, USN

15 March 2004

Abstract

This article describes the implementation of a rudimentary voltmeter using a PIC16F874 microcontroller. The input voltage must be in the range 0 V to 5 V. The program displays the voltage either in raw form using the 10 bits provided by the analog-to-digital conversion withint the PIC16F874 or in binary-coded decimal form.

Contents

1 Introduction **3**

2 Analog-to-digital Conversion **4**

3 The `main` Program **7**
3.1 Initialization 7
3.2 Updating the Display 7

4 Interrupt Service Routines **8**
4.1 Timer 2 Interrupt Handler 8
4.2 Analog-to-digital Converter Interrupt Handler 8

Index **19**

List of Figures

1 PIC16F874 as a voltmeter 3
2 Multiplication by 5 and 10 6

List of Tables

1 Multiplexing the LED display 4

Listings

1 Voltmeter 10

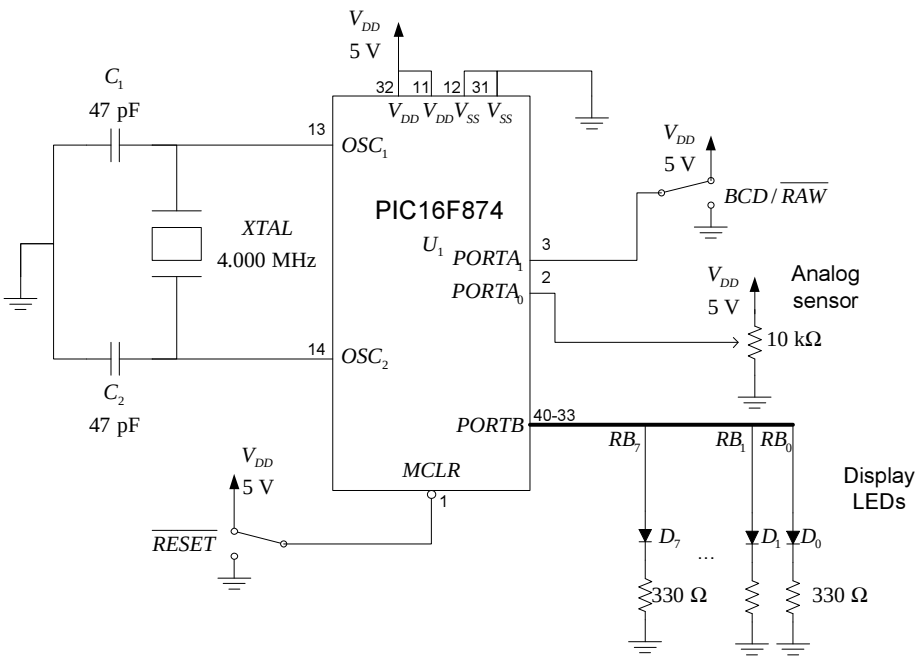

Figure 1: Schematic drawing of a PIC16F874 programmed to function like a voltmeter.

1 Introduction

This project describes the use of a PIC16F874 implementation to implement a simple voltmeter. Figure 1 shows the schematic drawing for this project and Listing 1 on page 10 contains the source code of the program.

The schematic drawing shows that a potentiometer provides analog voltage signals in the range 0 V to 5 V. The wiper is connected to the PIC16F874's AN0 pin, analog channel 0. Internally the PIC16F874 converts the sampled voltage to a 10-bit digital representation. The digital value can be displayed on the eight-bit LED display in either of two ways:

- In raw binary form or
- Using three binary-coded decimal (BCD) digits, representing volts, tenths of volts, and hundredths of volts.

Because eight bits suffices for neither of these directly, the display is multiplexed. There are three phases in the display of each converted value. Table 1 shows the manner and order in which values are displayed. The processor displays a new step every 500 ms, giving the display a blinking appearance with a pause between successive displays. A new sample is taken at the start of every 1.5 s sequence.

Step #	Raw Data Display	BCD Display
0	Blank	Blank
1	8 most significant bits	Right-justified BCD digit for volts
2	2 least significant bits, left justified	BCD digits for tenths and hundredths of volts

Table 1: Multiplexing the LED display

2 Analog-to-digital Conversion

The internal voltage references are used for this project. It is assumed that the lower reference is $V_{\text{REF}}^{-} = 0$ V and the upper reference is $V_{\text{REF}}^{+} = 5$ V exactly. If this assumption is violated, the voltmeter will not yield accurate results. To get more accurate results, a precision voltmeter could be used.

The general expression for converting from an analog voltage x to a 10-bit digital representation n using the PIC16F874 is given by

$$n = \left\lfloor \left(\frac{x - V_{\text{REF}}^{-}}{V_{\text{REF}}^{+} - V_{\text{REF}}^{-}} \right) 2^{10} \right\rfloor \tag{1}$$

$$n = \left\lfloor \left(\frac{x}{5\ \text{V}} \right) 2^{10} \right\rfloor. \tag{2}$$

This is correct when the converted value is right justified in a 16-bit field, spread across the two registers **ADRESH** and **ADRESL**. However, the value can also be left justified in a 16-bit field, spread across the same two registers. In this format, the representation is

$$n = \left\lfloor \left(\frac{x}{5\ \text{V}} \right) 2^{10} \right\rfloor 2^{6}. \tag{3}$$

Our program needs to reverse the calculation in (1), however, whenever it attempts to convert the raw binary data into BCD format. This can be done mathematically by

$$x = V_{\text{REF}}^{-} + \left(\frac{n}{2^{10}} \right) \left(V_{\text{REF}}^{+} - V_{\text{REF}}^{-} \right) \tag{4}$$

$$x = \left(\frac{n}{2^{10}} \right) (5\ \text{V}). \tag{5}$$

Applying this to (3) we get

$$x = \left(\frac{n}{2^{16}} \right) (5\ \text{V}) = \left(\frac{5\ \text{V}}{2^{16}} \right) (n). \tag{6}$$

The first of these forms expresses x as some fraction of the distance from 0 V to 5 V; the second expresses it as some multiple of the basic stepsize, $\frac{5\text{ V}}{2^{16}} = 76.29\ \mu\text{V}$.[1] The first therefore offers us the opportunity to perform integer multiplication of n by 5 V. The latter requires we multiply n by a rather more involved quantity, 76.29 μV.

Integer multiplication by $5_{10} = 101_2$ is easy in a computer. All it requires is to double the number, double it again, and add it to the original quantity. Multiplication by 76.29_{10} is more complicated and requires the use of a generalized multiplication routine. The value can either be regarded as a real number—necessitating the use of floating-point arithmetic—or it can be regarded as an integral number of, say, hundreds of nanovolts:

$$763_{10} \times 100\text{ nV} = 10\,1111\,1011_2 \times 100\text{ nV},$$

permitting use of integer arithmetic. For greater accuracy, it can instead be regarded as an integral number of tens of nanovolts:

$$7629_{10} \times 10\text{ nV} = 1\,1101\,1100\,1101_2 \times 100\text{ nV}.$$

In this project we choose to do integer multiplication by 5 for simplicity.

A binary number $n = b_{m-1} \ldots b_2 b_1 b_0$ means

$$n = b_{m-1} \ldots b_2 b_1 b_0 \tag{7}$$

$$= (b_{m-1} 2^{m-1}) + \cdots + (b_2 2^2) + (b_1 2^1) + (b_0 2^0). \tag{8}$$

When this is divided by 2^m, the result is

$$\frac{n}{2^m} = \frac{b_{m-1} \ldots b_2 b_1 b_0}{2^m} = 0.b_{m-1} \ldots b_2 b_1 b_0. \tag{9}$$

We can place the multiplier 5 in one byte and the multiplier n in a two-byte field with the binary point implicitly placed at its left end. The result of the multiplication will be three bytes with the binary point coming after the leftmost byte.

Figure 2 shows pictorially how multiplication of a 16-bit quantity by the constants 5 and 10 should proceed.

To get the decimal equivalent of the converted number of volts, therefore, all we need to do is the following:

1. Multiply n by 5. The most significant digit is found in the leftmost byte of the result.

2. Zero the most significant byte. This amounts to removing the integer portion and leaving the fractional portion.

[1] The PIC16F874 uses 10 bits to represent the converted voltage. Because they are left-justified in a 16-bit field, though, it is appropriate to use 2^{16} rather than 2^{10} in the denominator.

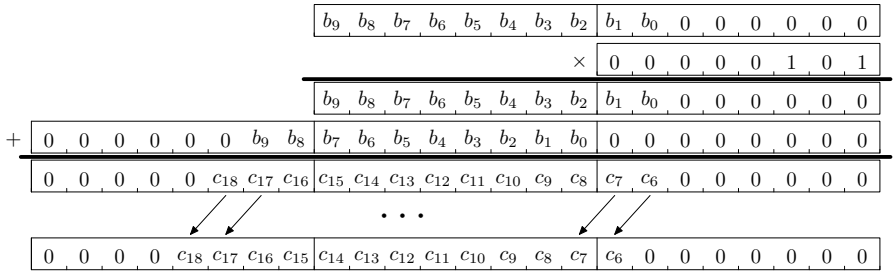

Figure 2: Multiplication of a 10-bit quantity by the constant 5 when left justified in a 16-bit field. An additional left shift converts this into the result of multiplication by the constant 10 instead.

3. Multiply this by 10. The most significant digit is found in the leftmost byte of the result.

4. Continue zeroing the most significant byte and multiplying by 10 to extract as many significant digits as you wish. Since the PIC16F874 only provides 10 bits of accuracy, there is no point in extracting more than 3 digits in all.

Note that unless the PIC16F874 encounters a whole voltage step, the digital output will not change to the next step. Also, to get better results requires rounding off, not rounding down. Rounding off can be accomplished by adding half of the least significant amount of interest before doing all the manipulations. For example, if we elect to extract three significant figures, we should add the equivalent of 0.005 V first. To illustrate that this is right, consider how to round 2.647 V to the nearest 10 mV. We should round up:

$$\begin{aligned}\mathrm{Round}(2.647, 2) &= \frac{\lfloor 10^2(2.647 + 0.005)\rfloor}{10^2} \\ &= 2.65.\end{aligned}$$

The 16-bit digital equivalent of 0.005 V is

$$\frac{0.005\ \mathrm{V}}{5\ \mathrm{V}} 2^{16} = 65.5 \approx 66_{10} = 42_{16}. \tag{10}$$

So here is the revised algorithm:

1. Sample the analog voltage and left justify the result.

2. Add 0042_{16} in preparation for later rounding.

3. Multiply by 5. The leftmost of the three resultant bytes contains the number of whole volts.

4. Zero the most significant byte.

5. Multiply by 10. The leftmost of the three resultant bytes contains the number of tenths of volts.

6. Zero the most significant byte.

7. Multiply by 10. The leftmost of the three resultant bytes contains the number of hundredths of volts.

3 The `main` Program

The main program starts at line 417 of Listing 1. It initializes all data values in lines 417–469 . Once it has accomplished this, it commences an endless task of keeping the display up to date in lines 472–469.

3.1 Initialization

In lines 420–426 Port B is initialized for output by placing **`TRISB_Init`** (eight 0 bits) into the **`TRISB`** register.

In lines 428–434 Port A is initialized for output by placing **`TRISA_Init`** into the **`TRISA`** register. This value is established in line 126 to ensure that bit 0 provides the analog input and bit 1 provides the user's choice for the display format.

The program initializes Timer 2 in lines 437–453. The initialization constants are created in lines 106, 114, and 115. Timer 2 will interrupt the processor every 10 ms.

The program initializes the analog-to-digital converter in lines 455–463 with constants created in lines 119 and 123. Analog values from input bit AN0 will be converted to 10-bit values left-justified in registers **`ADRESH`** AND **`ADRESL`**.

Finally the display is blanked in lines 465–469.

3.2 Updating the Display

There are two display schemes, as mentioned in the Introduction. Bit 1 of Port A is examined in line 475.

- If it is a 0, the display uses raw data.
- If it is a 1, the display uses binary-coded decimal (BCD).

Irrespective of the display format, the display is presented in three phases:

1. It is blank for 500 ms;
2. It shows the most significant 8 bits of the 10-bit converted value or the number of whole volts in BCD for 500 ms;
3. It shows the least significant 2 bits of the 10-bit converted value or the number of tenths and hundredths of volts in BCD for 500 ms;

To effect this, the main program relies upon the use of the variable **DisplayNum**, maintained by the Timer 2 interrupt handler. It uses this variable in one of two different table look-up schemes:

- The main program calls subroutine **GetDisplayValue** to display raw data, starting at line 369 of Listing 1.
- It calls subroutine **GetDisplayBCDValue** to display BCD data, starting at line 387 of Listing 1.

Both of these routines allocate two programs instructions for each table entry. The first of these transfers the data to be displayed into register **W**; the second returns it to the main program.

4 Interrupt Service Routines

4.1 Timer 2 Interrupt Handler

The Timer 2 interrupt handler only needs to do three things:

1. Count the number of occurrences of the Timer 2 interrupt. Since these occur every 10 ms, the handler counts down from 50 to 0, after which 500 ms has elapsed.
2. Count the number of times 500 ms has passed. This entails counting down from 3 to 0, after which 1.5 s has passed. At the end of this interval, the handler initiates an analog-to-digital conversion.
3. Change the variable **DisplayNum** every 500 ms. This variable must stay within the range $[0, \texttt{Displaymax} - 1]$. This permits the main program to know what phase of the three-phase output should be displayed at any time.

4.2 Analog-to-digital Converter Interrupt Handler

The analog-to-digital converter interrupts the processor whenever a conversion is complete. The interrupt handler for these interrupts is in lines 212–367 of Listing 1 and constitutes most of the code in the entire program.

Broadly speaking, the handler is responsible for putting the converted value into a form suitable for output. If the desired output format is raw data, the task is trivial. Registers **ADRESH** and **ADRESL** are copied to registers **HighWord** and **LowWord**, respectively. The task is more involved when the desired output format is BCD and details are in Section 2.

First, 42_{16} is added to the raw data in lines 231–243.

Then the number of whole volts is determined by multiplying the adjusted data by 5. This is done in lines 245–280. The only peculiar aspect of the code is the propagation of the carry bits. The **incf** instruction cannot be used because it does not calculate the carry bit, whereas the **addwf** instruction does. But

when calculating the most significant byte, there is no possibility of an output carry, as reference to Figure 2 on page 6 makes clear.

Next the number of tenths of volts is determined by zeroing the most significant byte—the whole volts—and multiplying the rightmost two bytes by 10. This is done in lines 282–322. The algorithm calls for multiplying by 5 and then shifting the result one bit further to the left.

Finally the number of hundredths of volts is deterimined by once again zeroing the most significant byte—the tenths of volts—and multiplying the rightmost two bytes by 10. This is done in lines 324–364.

Listing 1: Voltmeter

```
;*********************************************************************
;   This program provides a simple voltmeter to display the voltage   *
;   available at AN0. There are two displays, either one of which can*
;   be selected by a switch input:                                    *
;     (1) if RA1 is a 1, the analog value is converted to BCD format. *
;     (2) if RA1 is a 0, the analog value is displayed in raw format. *
;   The analog data is provided at analog input AN0. It is sampled    *
;   once every 1.5 s.  Eight LEDs provide the output.                 *
;                                                                     *
;   When output is in BCD format, it is presented in three            *
;   phases:                                                           *
;     (1) Blank for 0.5 s.                                            *
;     (2) A right-justified BCD representation of the number of volts *
;         is displayed for 0.5 s.                                     *
;     (3) Two BCD digits are displayed for 0.5 s.  The leftmost of    *
;         these digits represents the number of tenths of volts;      *
;         the rightmost represents the number of hundredths of volts. *
;                                                                     *
;   When output is in raw format, it is still presented in three      *
;   phases:                                                           *
;     (1) Blank for 0.5 s.                                            *
;     (2) Eight bits representing the most significant eight bits of  *
;         the 10-bit converted value.                                 *
;     (3) Two bits, left justified, representing the least significant*
;         two bits of the 10-bit converted value.                     *
;*********************************************************************
;                                                                     *
;    Filename:      voltmeter.asm                                     *
;    Date:          11 March 2004                                     *
;    File Version:  1                                                 *
;                                                                     *
;    Author:        CDR Charles B. Cameron, USN                       *
;    Company:       United States Naval Academy                       *
;                                                                     *
;                                                                     *
;*********************************************************************
;                                                                     *
;    Files required:                                                  *
;                                                                     *
;                    p16f874.inc                                      *
;                                                                     *
;*********************************************************************
;                                                                     *
;    Notes:                                                           *
;    It is assumed that the processor is driven by a 4 MHz crystal.   *
;    It is also assumed that Vdd = 5 V exactly. To the extent that    *
;    this assumption is in error, the converted voltages also will be *
;    inaccurate.                                                      *
;                                                                     *
;*********************************************************************


    list        p=16f874            ; list directive to define processor
   #include <p16f874.inc>           ; processor specific variable definitions

;  The following configuration choices turn code protection off, watch
```

```
;   dog timer off,brown-out reset disabled, power-up timer enabled, HS
;    oscillator  mode selected,flash  program memory write disabled,
;   low-voltage in-circuit serial programming disabled, and data EE
;   memory code protrection off.
ConfigBits set _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON
ConfigBits set   ConfigBits & _HS_OSC & _WRT_ENABLE_ON
ConfigBits set   ConfigBits   & _LVP_OFF & _CPD_OFF
    __CONFIG ConfigBits


;***** VARIABLE DEFINITIONS
w_temp        EQU   0x20   ; variable used for context saving
                           ; Addresses A0, 120, and 1A0 also
                           ; are set aside for this purpose.
status_temp   EQU   0x21   ; variable used for context saving

    cblock      0x22
temp                     ; A location for temporary values
count                    ; Used to count 10 ms interrupts.
ADC_count                ; Used to counts 500 ms intervals.
HighWord                 ; High word of the ADC conversion.
LowWord                  ; Low word of the ADC conversion.
DisplayNum               ; Indicates which display is active.
                         ; Valid values are:
                         ;        0 = blank display
                         ;        1 = display most significant byte
                         ;        2 = display least  significant byte
y0                       ; y = y2:y1:y0, a three-byte number
y1
y2
z0                       ; z = z2:z1:z0, a three-byte number
z1
z2
OnesDigit                ; BCD representation of analog volts
TenthsDigit              ; BCD representation of analog deci volts
HundredthsDigit          ; BCD representation of analog centi volts
            endc

DisplayBlank    equ     0
DisplayHigh     equ     1
DisplayLow      equ     2
DisplayMax      equ     3   ; If the DisplayNum reaches this value, reset it.

; Port A bit definitions
; Bit 0:   AN0  Input from a potentiometer
; Bit 1:   RA1  1 if output is to be in BCD form
;               0 if output is to be in raw 10-bit binary form
DisplayPreference   equ     1 ; points to bit 1 of Port A

count_init        equ D'50'   ; Count 50 x 10 ms interrupts for 500 ms.
ADC_count_init equ D'3'     ; Count 3 x 500 ms for 1.5 s

INTCON_Init equ B'11000000' ; Bit 7 = GIE, Bit 6 = PEIE
; Assume f_osc = 4 MHz, so T_inst = 1 us
; If x = prescaler, y = PR2, and z = postscaler, we want
; x (y+1) (z+1) = 10,000 us = 10 ms = Timer 2 interrupt interval
; Choose x = 4, y = 249, z=9
```

```
PR2_Init     equ D'250'-1    ; y = 249
T2CON_Init equ B'01001001' ; Bits 6..3 => z=1001=9; Bits 1..0=01 => x=4

TRISB_Init equ B'00000000' ; All Port B bits are for output.

ADCON0_Init equ B'01000001' ; Bits 7,6=01=> T_AD = 8 T_OSC = 2 us
                            ; Bits 5..3 = 000 => Channel 0
                            ; Bit 2 = 0 => Don't start a conversion
                            ; Bit 0 = 1 => Turn the ADC on & start acquistion
ADCON1_Init equ B'00001110' ; Bit 7 = 0 => left justified result
                            ; Bit 6..4 = xxx
                            ; Bit 3..0 = 1110 => AN0, other Port A bits digital
TRISA_Init equ B'00000011' ; Let all Port A pins be used for output,
                            ; except bit 0 is for analog input.
                            ; Also, bit 1 is used for digital input
                            ; DisplayPreference

; Macro Bankset n takes a parameter n in the range 0 to 3, inclusive
; It sets the STATUS bits RP1 and RP0 to provide access to the specified
; bank, n.
BankSet macro n
        if  n==0            ; Bank 0
            bcf STATUS,RP0
            bcf STATUS,RP1
        endif
        if  n==1            ; Bank 1
            bsf STATUS,RP0
            bcf STATUS,RP1
        endif
        if  n==2            ; Bank 2
            bcf STATUS,RP0
            bsf STATUS,RP1
        endif
        if  n==3            ; Bank 3
            bsf STATUS,RP0
            bsf STATUS,RP1
        endif
        endm



;*********************************************************************
        ORG     0x000           ; processor reset vector
        clrf    PCLATH          ; ensure page bits are cleared
        goto    main            ; go to beginning of program


        ORG     0x004           ; interrupt vector location
        movwf   w_temp          ; save off current W register contents
        movf    STATUS,w        ; move status register into W register
        bcf     STATUS,RP0      ; ensure file register bank set to 0
        movwf   status_temp     ; save off contents of STATUS register


; Two ISR handlers are provided, one for Timer 2 interrupts and the
; other for A/D converter interrupts.
        btfsc   PIR1,TMR2IF
```

```
        call    HandleTimer2
        btfsc   PIR1,ADIF
        call    HandleADC

        bcf     STATUS,RP0      ; ensure file register bank set to 0
        movf    status_temp,w   ; retrieve copy of STATUS register
        movwf   STATUS          ; restore pre-isr STATUS register contents
        swapf   w_temp,f
        swapf   w_temp,w        ; restore pre-isr W register contents
        retfie                  ; return from interrupt

HandleTimer2
        bcf     PIR1,TMR2IF  ; Clear the interrupt flag


        ; Decrement the Timer 2 interrupt counter. When it
        ; reaches zero, 500 ms has passed. Decrement the
        ; 500 ms counter.
        decfsz  count,F
        goto    HandleTimer2End
        movlw   count_init      ; Re-initialize the 10 ms counter.
        movwf   count
        decfsz  ADC_count,F
        goto    UpdateDisplay
        ; When the ADC_count reaches zero, 1.5 s has passed so
        ; start an A/D conversion.
        BSF     ADCON0,GO_DONE
        movlw   ADC_count_init ; Re-initialize the 500 ms counter.
        movwf   ADC_count

UpdateDisplay
        ; Update the display every 500 ms
        incf    DisplayNum,F
        movlw   DisplayMax ; If DisplayNum = DisplayMax, zeroize it.
        subwf   DisplayNum,W
        btfsc   STATUS,Z
        clrf    DisplayNum

HandleTimer2End
        return

HandleADC
        bcf     PIR1,ADIF ; Clear the interrupt
        ; Save the converted values
        movf    ADRESH,W
        movwf   HighWord
        BankSet     1
        movf    ADRESL,W
        BankSet 0
        movwf   LowWord

        ; If we're displaying BCD digits,
        ; then we want to round the result to the
        ; nearest decivolt. To do this,
        ; add 0.005 V, then truncate. In 16-bit binary,
        ; this is (0.005 V / 5 V)*2^16 = 66 = 42h.
        ; If we're displaying raw data, skip
```

```
        ; to the end of the interrupt handler.
        btfss   PORTA,DisplayPreference
        goto    HandleADCEnd
RoundValues
        clrf    y2
        clrf    z2
        movlw   H'42'
        addwf   LowWord,F
        btfss   STATUS,C  ; Add the carry into HighWord if C=1.
        goto    ConvertToBCD
        movlw   1
        addwf   HighWord,F
        btfsc   STATUS,C
        incf    y2          ; Add the carry into y2 if C=1.
        movf    y2,W        ; Since y2 = 0 initially, it's safe to use incf
        movwf   z2

ConvertToBCD
        ; Calculate volts, decivolts, and centivolts
        ; First calculate the whole volts digit by multiplying
        ; the left-justified A/D result by 5 since the
        ; voltage references are assumed to be 5 V and 0 V.
        movf    LowWord,W ; Let z = y = y2:y1:y0 <- x = 0:HighWord:LowWord
        movwf   y0
        movwf   z0
        movf    HighWord,W
        movwf   y1
        movwf   z1

        bcf     STATUS,C
        rlf     y0,F    ; Shift y2:y1:y0 left 1 bit
        rlf     y1,F    ; y = 2x
        rlf     y2,F
        bcf     STATUS,C
        rlf     y0,F    ; Shift y2:y1:y0 left 1 bit
        rlf     y1,F    ; y = 4x
        rlf     y2,F
        movf    z0,W
        addwf   y0,F
        btfss   STATUS,C
        goto    ComputeUnitsDigit_y1
        movlw   1       ; Propagate the carry into y1
        addwf   y1,F
        btfsc   STATUS,C
        incf    y2,F    ; Propagate the carry into y2
ComputeUnitsDigit_y1
        movf    z1,W
        addwf   y1,F    ; y <- y + x = 4x + x = 5 x
        btfsc   STATUS,C
        incf    y2,F    ; Propagate the carry into y2
        movf    z2,W
        addwf   y2,W
        movwf   OnesDigit

        ; Now compute the tenths digit.
        clrf    y2      ; Eliminate the whole digits, first
        ; At this point, y1:y0 contains fractional volts
```

```
        ; Multipy this by 10 and it will contain tenths of volts plus fractional
        ; tenths of volts. Accomplish this by multiplication by 5, then doubling.
        movf    y0,W
        movwf z0        ; z <- y
        movf    y1,W
        movwf z1
        movf    y2,W
        movwf z2
        bcf     STATUS,C
        rlf     z0,F
        rlf     z1,F
        rlf     z2,F    ; z <- 2 y
        bcf     STATUS,C
        rlf     z0,F
        rlf     z1,F
        rlf     z2,F    ; z <- 4 y
        movf    y0,W    ; z <- z + 4y = 5y
        addwf z0,F
        btfss   STATUS,C
        goto    ComputeTenthsDigit_y1
        movlw 1         ; Propagate the carry into y1
        addwf z1,F
        btfsc   STATUS,C
        incf    z2,F    ; Propagate the carry into y2
ComputeTenthsDigit_y1
        movf    y1,W
        addwf z1,F
        btfsc   STATUS,C
        incf    z2,F
        movf    y2,W
        addwf z2,F
        movf    z2,W
        bcf     STATUS,C ; z <- 2 z = 10 y
        rlf     z0,F
        rlf     z1,F
        rlf     z2,F
        movf    z2,W
        movwf TenthsDigit

        ; Now compute the hundredths digit
        clrf    z2      ; Eliminate the tenths, first.
        ; At this point, z1:z0 contains 10 x fractional volts.
        ; Multiply it by ten to isolate the hundredths of volts.
        movf    z0,W
        movwf y0        ; z <- y
        movf    z1,W
        movwf y1
        movf    z2,W
        movwf y2
        bcf     STATUS,C
        rlf     z0,F
        rlf     z1,F
        rlf     z2,F    ; z <- 2 y
        bcf     STATUS,C
        rlf     z0,F
        rlf     z1,F
        rlf     z2,F    ; z <- 4 y
```

```
        movf    y0,W    ; z <- z + 4y = 5y
        addwf   z0,F
        btfss   STATUS,C
        goto    ComputeHundredthsDigit_y1
        movlw   1
        addwf   z1,F        ; Propagate the carry into y1
        btfsc   STATUS,C
        incf    z2,F        ; Propagate the carry into y2

ComputeHundredthsDigit_y1
        movf    y1,W
        addwf   z1,F
        btfsc   STATUS,C
        incf    z2,F
        movf    y2,W
        addwf   z2,F
        movf    z2,W
        bcf     STATUS,C ; z <- 2 z = 10 y
        rlf     z0,F
        rlf     z1,F
        rlf     z2,F
        movf    z2,W
        movwf   HundredthsDigit

HandleADCEnd
        return

GetDisplayValue
        ; This subroutine returns one of three things:
        ;   (1) Zero, if W=0
        ;   (2) The most significant 8 converted bits if W=1
        ;   (3) The least significant 8 converted bits if W=0
        bcf     STATUS,C
        movwf   temp
        ; Doubling the index lets us put two instructions in
        ; each table location.
        rlf     temp,W
        addwf   PCL,F
        clrw                ; Index 0
        return
        movf    HighWord,W ; Index 1
        return
        movf    LowWord,W ; Index 2
        return

GetDisplayBCDValue
        ; This subroutine returns one of three things:
        ;   (1) Zero, if W=0
        ;   (2) The right-justified BCD units digit if W=1
        ;   (3) Two BCD digits, for tenths and hundredths, if W=2
        bcf     STATUS,C
        movwf   temp
        ; Doubling the index lets us put two instructions in
        ; each table location.
        rlf     temp,W
        addwf   PCL,F
        clrw                    ; Index 0
```

```
        return
        movf   OnesDigit,W     ; Index 1
        return
        ; Well, note that the index 1 code is more than just 2
        ; instructions. This is ok as long as we don't need
        ; another index.
        movf   TenthsDigit,W   ; Index 2
        movwf  temp
        bcf    STATUS,C
        rlf    temp,F                ; Shift the BCD digit to
        rlf    temp,F                ; positions 7 down to 4
        rlf    temp,F
        rlf    temp,F
        movf   HundredthsDigit,W ; and fold in the hundredths
                                 ; in bits 3 down to 0.
        addwf  temp,W
        return

main

; ****************************************************************************
; Here is the code to test a PIC16F874.
; Configure Port B for output. It will be used to drive 8 LED indicators.
    clrf    PORTB
    movlw   TRISB_Init
    BankSet    1
    movwf   TRISB
    BankSet    0

; Configure Port A.
; a test signal.
    clrf    PORTA     ; Initialize Port A by clearing the output latches.
    BankSet    1
    movlw   TRISA_Init
    movwf   TRISA
    BankSet    0


; Configure Timer 2 to generate an interrupt every 10 ms.
    movlw   INTCON_Init
    movwf   INTCON
    bcf     PIR1,TMR2IF
    BankSet    1
    bsf     PIE1,TMR2IE
    movlw   PR2_Init
    movwf   PR2
    BankSet    0
    clrf    TMR2
    movlw   T2CON_Init
    movwf   T2CON
    movlw   count_init
    movwf   count
    movlw   ADC_count_init
    movwf   ADC_count
    bsf     T2CON,TMR2ON ; Turn Timer 2 on now.

; Configure the ADC
```

```
    bcf     PIR1,ADIF
    movlw   ADCON0_Init
    movwf   ADCON0
    BankSet 1
    bsf     PIE1,ADIE
    movlw   ADCON1_Init
    movwf   ADCON1
    BankSet 0

; Initialize the display with all blanks
    clrf    HighWord
    clrf    LowWord
    movlw   DisplayBlank
    movwf   DisplayNum


loop
    ; Check to see whether raw ADC data or BCD equivalents
    ; are needed
    btfsc   PORTA,DisplayPreference
    goto    DisplayBCDValues

DisplayBinaryValues
    ; Retrieve the correct binary display values and output them.
    movf    DisplayNum,W
    call    GetDisplayValue
    movwf   PORTB
    goto    loop        ; Continue indefinitely.

DisplayBCDValues
    ; Retrieve the correct BCD display values and output them.
    movf    DisplayNum,W
    call    GetDisplayBCDValue
    movwf   PORTB
    goto    loop

    END
```

Index

ADC, *see* analog-to-digital conversion
analog, 3
analog-to-digital conversion, 8, 9

BCD, *see* binary-coded decimal
binary-coded decimal, 3, 4, 8

display, 3, 7

initialization, 7
interrupt, 8

LED, *see* light-emitting diode
light-emitting diode, 3

multiplication, 5

potentiometer, 3

table look-up, 8

voltage reference, 4
voltmeter, 3