Lesson 15
Encoders

Overview

Introduction	In this lesson we will examine the types of encoders available and how to deal with them programmatically.
In this section	Following is a list of topics in this section:

Encoder Hardware

Introduction

Earthlings are accustomed to doing a lot of things by turning a knob. Tuning a radio, for example, or adjusting code speed just seem like natural applications for selecting an input by rotating a knob.

Many PICs have analog inputs. Thus we could use a potentiometer to generate an adjustable voltage. However, as we all know from trying to get the right range on a VCO, the resolution is an issue. Also, the potentiometer followed by an A/D converter is a fairly complex and expensive approach to getting user input.

The encoder solves this problem. Encoders give us a simple, digital way to either sense the position of a knob, or, more commonly, to detect which way the user is turning the knob.

Encoding Schemes

Encoders generate data in two ways. The most obvious is to provide a digital indication of position with a binary output. Although this is an obvious approach, it has a significant problem; as the resolution goes up, more bits, and consequently more PIC pins, are needed to sense the position. As we have begun to recognize, these pins are the PICs most precious resource.

The second way, and by far the most common, is the *Gray Code*. Gray code uses two outputs to allow us to sense whether the user is turning the shaft, and if so, in which direction. The way it does this is quite simple. Each of the two outputs is a square wave dependent on the shaft position. The two square waves from the two outputs are out of phase. It turns out that this makes it quite simple to detect in which direction the shaft is moving.

Technologies

Besides the two encoding schemes, encoders can also be broken down by the technology they use. The least expensive encoders use wipers or brushes to sense the motion of a conductive disk which contains a pattern that allows generation of the Gray code. From the perspective of the PIC, the encoder appears to be two switches, so some sort of pullup is needed. Since the user may well turn the shaft quickly, and the wipers will have some contact bounce, the resolution of mechanical encoders is limited.

Disassembled Panasonic EVQ encoder

Optical encoders get around these issues by using a slotted wheel and a phototransistor. Optical encoders allow much higher resolution than mechanical encoders, however they can be significantly more expensive. While mechanical encoders can be had for between $1 and $5 U.S. in hobbyist quantities, optical encoders are typically $35 to $70. Still, they are frequently used in tuning VFOs where high resolution and a very smooth feel are desired.

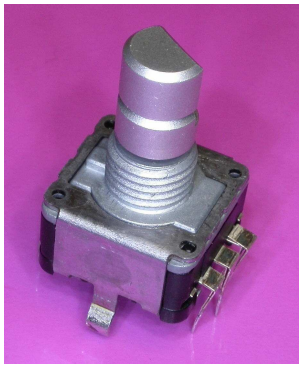
Some Example Encoders

Introduction

There are a dizzying array of encoders available, at a wide range of prices. Like so many things, there probably isn't a best encoder, but each has advantages and disadvantages depending on the particular application. Here we examine a number of encoders and talk about their differences.

Mechanical Encoders

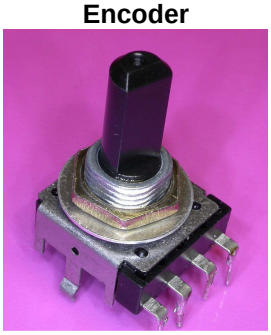
Panasonic EVGWEB encoder



For most applications, we are likely to choose a mechanical encoder, simply because the price is so much lower than the optical encoders.

The first PIC-ELs shipped with a Panasonic encoder. This is a low cost, low-resolution 12mm encoder with relatively soft detents. This encoder is unusual in that the pulses do not align exactly with the detents. On occasion, this encoder is available at a very low price, and with appropriate software adjustments can be a very nice encoder. Newer PIC-ELs shipped with an encoder which has the code stretched at the detent, so the same output always shows when the encoder is resting in the detent.

Panasonic EVQVEMF Encoder



The Panasonic EVQVEMF is a 14 mm encoder with very nice soft detents, quite similar to the EVGWEB but somewhat smoother. When used with a larger knob, this encoder has a very nice feel. This is also a fairly low cost encoder and offers reasonable resolution. Because of the detents, it may not be suitable for all applications.

This encoder is a little unique in that the two signals do not share a common connection, so there are four terminals instead of the usual three.

Bourns ECW1 encoder

The Bourns ECW1 encoder is a somewhat larger encoder with reasonable resolution and very solid detents. This encoder might be more suitable for making selections from a list or for selecting channels, rather than making adjustments in some continuous parameter as would be more common for other encoders. It's relatively large size (24x29 mm) might indicate against it's use in many QRP applications. Nevertheless, it's positive detents and reasonable price may make it attractive in some applications.

Continued on next page

Some Example Encoders, Continued

Mechanical Encoders (continued)

The CTS 288 encoder is available without detents, which makes it more attractive for tuning applications. The encoder also more closely resembles the more traditional potentiometers, and is available in a range of connection and mounting options. However, the relatively low resolution does limit its usefulness. At 20x27mm it is also relatively large.

CTS 288 encoder

Bourns 3315 encoder

The Bourns 3315 is a very small (9mm) encoder, also without detents. Its small size and low cost make it a very attractive encoder for QRP applications. But, like most mechanical encoders, its low resolution limits its usefulness.

This encoder also has a 3mm shaft, rather than the more common 6mm shaft, which can facilitate the use of smaller knobs which may be important where panel space is limited.

Most of the above encoders are available at a variety of resolutions, although the maximum resolution available varies between models, with maximum resolutions of 16 or 24 pulses per revolution being the most common.

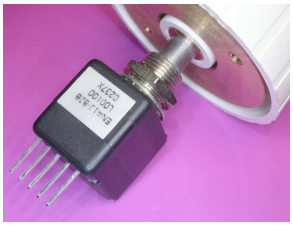
Optical Encoders

When resolutions higher than about 32 pulses per revolution are needed, one generally is forced to use an optical encoder. Optical encoders are far more expensive than mechanical encoders, but resolutions of 200 pulses per revolution or higher are available. Because of the higher cost, these can be provided with rather elaborate bearing mechanisms which, especially when combined with a weighted knob, can result in a very appealing “feel”.

Optical encoders also require power, and typically provide a transistor, rather than switch closure, output. This is generally not a concern in PIC applications.

The Bourns ENA encoder is typical of the better optical encoders. With up to 128 cycles per revolution, this encoder can provide a very nice tuning adjustment for a VFO. However, the cost is more than ten times the price of a typical mechanical encoder.

Bourns ENA Encoder



The Clarostat 600 encoder is another very nice optical encoder. Although it has a very high list price, sometimes it can be found on the web at a substantial discount.

Sensing Encoder Signals

Introduction

The encoders provide two switch closure outputs, although as mentioned earlier, for optical encoders, the "switch closure" is actually a transistor output. For the switch closure, the PIC's weak pullups could be used, but to help limit noise it is generally preferable to use lower resistance pullups. It can be tempting to bypass the outputs to reduce noise pickup, but the experimenter is cautioned against too much capacitance. Even though it is basically a mechanical device, the signals can be fairly fast, as we will see shortly.

A simple Experiment

To get a feel for the encoder output, we will do a simple program to reflect the status of the encoder outputs to the LEDs.

For this experiment (Lesson15a), first set the LED's to be outputs:

```
Start
        ; Set the LEDs to be outputs
                banksel         TRISB           ; Select bank 1
                errorlevel      -302            ; Yes, we know!
                bcf             TRISB,LED1      ; Clear the TRIS bits
                bcf             TRISB,LED2      ; for each of the LEDs
                bcf             TRISB,LED3      ; making them outputs
                errorlevel      +302            ; Back on just in case
                banksel         PORTB           ; Back to bank 0
```

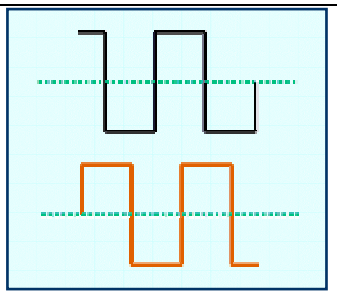
Then, loop, setting the LED state to reflect the encoder inputs:

```
Loop
        ; Turn off all LED's in ouptut word
                movlw           B'00001110'     ; LED outputs are HIGH
                movwf           Output          ; to turn LED off
        ; Set current status into Output
                btfsc           PORTA,ENC1      ; ENC1 low?
                bcf             Output,LED1     ; No, turn on LED1
                btfsc           PORTA,ENC2      ; ENC2 low?
                bcf             Output,LED2     ; No, turn on LED2
        ; Send the outputs
                movf            Output,W        ; Pick up the output word
                movwf           PORTB           ; and send it to PORTB
                goto            Loop            ; Do it again
```

When running this program, it can be helpful to place a large knob on the encoder, especially if you are using an encoder with detents. Notice that the LEDs go through several transitions between each detent. You will need to turn the knob very slowly to see the transitions. Depending on the particular encoder, you may find that the LED state is always the same when the shaft is resting in a detent.

The Gray Code

If the output of the encoder is plotted against rotation, the signals would look much like the figure. Notice that each output produces a square wave with rotation, but the two square waves are out of phase. The waveform might not be perfectly symmetrical; in particular, sometimes the waveform is "stretched" near the detent to ensure that the output at the detent is always predictable.



Decoding the Gray Code

Introduction

Now that it is clear how the outputs behave, there are a variety of way the programmer could choose to decode the outputs.

Building a Table

Were we to look at the two bits as a single number, we could see that there are four possible states, 0, 1, 2, or 3. The direction of rotation could be determined by examining all combinations of previous and current state of these values.

Since there are only four possible values, there are only sixteen possible combinations of previous and current state, so some sort of table lookup is clearly a feasible approach. Notice that there are a number of combinations that should be impossible.

	Current			
Previous	0	1	2	3
0	No motion	CW	CCW	
1	CCW	No motion		CW
2	CW		No motion	CCW
3	CCW		CW	No motion

Converting the table to code

If the previous and current states were combined we could use the combination as a lookup in a table similar to a previous lesson. Taking the previous state and multiplying it by four and then adding the current state would give us a value between zero and fifteen. Multiplying by four is convenient because we can do that simply by shifting left two bits.

```
; Move last reading over 2 and mask other bits
        rlf             Input,F         ; Rotate the input storage
        rlf             Input,F         ; over two bits
        movlw           B'00001100'     ; Keep 2 bits from last time
        andwf           Input,F         ; but clear all others
; Move current status into input word
        btfsc           PORTA,ENC1      ; ENC1 low?
        bsf             Input,ENC1      ; No, set it high on output
        btfsc           PORTA,ENC2      ; ENC2 low?
        bsf             Input,ENC2      ; No, set it high on output
```

In earlier lessons, we used a table lookup to come up with a value. In this case, however, we probably want to execute some code, but we can use the same kind of logic:

```
                addwf           PCL,F           ; Jump depending on value
                goto            SetNo           ; 0000 Same
                goto            SetUp           ; 0001 Up
                goto            SetDn           ; 0010 Down
                goto            SetErr          ; 0011 Error
                goto            SetDn           ; 0100 Down
                goto            SetNo           ; 0101 Same
                goto            SetErr          ; 0110 Error
                goto            SetUp           ; 0111 Up
                goto            SetUp           ; 1000 Up
                goto            SetErr          ; 1001 Error
                goto            SetNo           ; 1010 Same
                goto            SetDn           ; 1011 Down
                goto            SetErr          ; 1100 Error
                goto            SetDn           ; 1101 Down
                goto            SetUp           ; 1110 Up
                goto            SetNo           ; 1111 Same
```

Continued on next page

Decoding the Gray Code, Continued

Converting the table to code (continued)

In this example (Lesson15b) we would like to illuminate one of two LEDs to show which direction we are turning the shaft. Since we may possibly get invalid values, it would be good to use the center LED to indicate an error.

```
        ; Movement is clockwise, turn on LED1
SetUp
                bcf             Output,LED1     ; Clear=ON
                goto            Outputs
        ; Movement is counterclockwise, turn on LED3
SetDn
                bcf             Output,LED3     ; Clear=ON
                goto            Outputs
        ; Got a bad combination, turn on LED2
SetErr
                bcf             Output,LED2     ; Clear=ON
                goto            Outputs
        ; No change, don't change LEDs
SetNo
                goto            Loop
```

If the necessary initialization and setting the outputs is done, the program will illuminate LED 1 or 3 depending on which way the shaft is being turned. Once in a while, LED2 will light, indicating an error. When the shaft stops, the LEDs are left in their previous position. Since this is all happening in a tight loop, had we chosen to extinguish the LEDs when rotation stopped, the LEDs would be mostly dark, since even if the operator was turning the shaft quickly, most of the time there would be no change.

Depending on the speed of rotation and the particular encoder, the error LED will flash occasionally. However, without some method of latching the error, the flash will normally not be noticeable for the same reason mentioned above.

Contact Bounce

What is happening, of course, is contact bounce. For a mechanical encoder, the brushes that make the contact will have a certain amount of bounce. Since the brushes on the two channels cannot be expected to bounce in harmony, from time to time an impossible combination will appear at the outputs. What is needed is a way to debounce the encoder outputs.

Debouncing

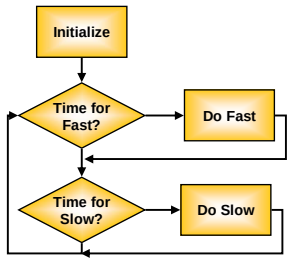
Introduction

The contact bounce on the encoder can be a significant issue if the application provides feedback that makes the little perturbations in the apparent rotation evident to the user.

Optical encoders typically don't suffer from contact bounce, but mechanical encoders do. The degree to which this is a problem varies from encoder to encoder. It creates a problem because the transitions can happen quite quickly if the user rotates the shaft quickly, especially with higher resolution encoders. The contact bounce, however, isn't considerate enough to take into account how quickly the shaft is being rotated, and sometimes the contact bounce can continue for some time. Bounce specifications of 60 ms. are not that uncommon, and even with relatively low resolutions, the user can often create transitions in that time frame.

Fortunately, not all examples of encoders are worst case, and the ability of the user to provide fast transitions can be managed to some degree with the shape and size of the knob provided. The actual timing chosen for debouncing is often best tuned to the particular encoder.

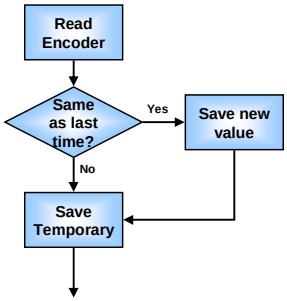
How to debounce



Since we would typically want to do something besides read the encoder, it will be helpful to use our timer routines from a previous lesson. In this case, we only have two tasks; reading the encoder and then displaying the result.

When reading the encoder, we will sample the inputs and not accept them until we get two consecutive readings the same.

Our "Fast" logic will simply read the encoder and remember the reading. If the reading is the same as the last time, the result will be calculated and provided to the "Slow" logic. The Slow logic will be simply display the result. The fast logic, by not accepting a value until it gets the same value twice, presumably has waited until any bounce has stopped. Because our error transitions may disappear quickly, it can be helpful to make the slow logic quite slow, to give a chance to see the error LED.



It would be better, or course, to read the encoder more times. The challenge is trading off the amount of time waiting for the bounce to subside with the frequency at which pulses are arriving. We don't want to read the encoder too frequently or we can read the same value several times while we are still within a single bounce.

Continued on next page

Debouncing, Continued

Debounce Code

In this example, we will have some code that runs about every 0.5 ms. The experimenter may want to adjust the time to match the particular encoder. The encoder will be read each time, and when two successive readings match, that reading will be used:

```
;-----------------------------------------------------------------------
;       2000 times per second code here
;-----------------------------------------------------------------------

        ; In order to debounce the input, we won't accept a reading
        ; until we have seen the same input on two successive
        ; reads of the encoder spaced 1/2 ms apart.  This is probably
        ; not as much time as we would like, but the encoders shiped
        ; with newer PIC-ELs generate their transitions in a very short
        ; space, so a slower read would likely miss transitions.
Hz2000
        ; Read the encoder bits
                movf            PORTA,W         ; Pick up the input word
                andlw           H'03'           ; Mask off all but encoder
                movwf           ThisRead        ; And save into current reading
        ; See if the same as last time
                movf            ThisRead,W      ; Pick up current
                xorwf           LastRead,W      ; Will be zero if same
                btfsc           STATUS,Z        ; Zero?
                goto            NewReading      ; Yes, will use reading
                movf            ThisRead,W      ; No, remember for
                movwf           LastRead        ; next time
                return                          ; Exit Hz2000
        ; We now have two successive readings that are identical.
        ; Or them into the input word after shifting the existing
        ; contents left two bits and masking off any excess.
NewReading
                movf            ThisRead,W      ; Remember for
                movwf           LastRead        ; next time
        ; Move last reading over 2 and mask other bits
                rlf             Input,F         ; Rotate the input storage
                rlf             Input,F         ; over two bits
                movlw           B'00001100'     ; Keep 2 bits from last time
                andwf           Input,F         ; but clear all others
        ; Move current status into input word
                movf            ThisRead,W      ; Pick up current reading
                iorwf           Input,F         ; And OR it into the input
```

At this point, we fall through to the same table we used in the previous example.

Initialization

The mainline for this example is very similar to the examples we used for the timer. Of course, we need to do the appropriate initialization of the timer in the `OPTION_REG` and we need to set the LEDs to be outputs. We also need to initialize the various storage locations.

Continued on next page

Debouncing, Continued

Initialization (continued)

```
Start
       ; Initialize GP register locations
              clrf          Input          ; Clear the input storage
              clrf          Output         ; and the output storage
              movlw         HZ2000T        ; Initialize the Hz2000
              movwf         Hz2000cnt      ; counter
              movlw         HZ10T          ; and the HZ10 counter
              movwf         Hz10Cnt        ;
       ; Set up timer
              errorlevel    -302           ; Yes, we know!
              banksel       INTCON
              bcf           INTCON,T0IE    ; Mask timer interrupt
              banksel       OPTION_REG
              bcf           OPTION_REG,T0CS; Select timer
              bcf           OPTION_REG,PSA ; Prescaler to timer
              bcf           OPTION_REG,PS2 ; \
              bcf           OPTION_REG,PS1 ;  >- 1:2 prescale
              bcf           OPTION_REG,PS0 ; /
       ; Set the LEDs to be outputs
              banksel       TRISB          ; Select bank 1
              bcf           TRISB,LED1     ; Clear the TRIS bits
              bcf           TRISB,LED2     ; for each of the LEDs
              bcf           TRISB,LED3     ; making them outputs
              errorlevel    +302           ; Back on just in case
              banksel       PORTB          ; Back to bank 0
```

Main program loop

The main program loop is virtually identical to that in the timer lesson, except for the times selected. The experimenter may wish to try different times. The 0.5 ms selected is a little fast for eliminating bounce on many encoders, but slowing down the time very much makes it easy to miss transitions should the user turn the knob quickly.

```
;-------------------------------------------------------------------
;       Main program loop here
;-------------------------------------------------------------------
main
              btfss         INTCON,T0IF    ; Did timer overflow?
              goto          main           ; No, hang around some more
              bcf           INTCON,T0IF    ; reset overflow flag

;-------------------------------------------------------------------
;       Check for 2000 times per second
;-------------------------------------------------------------------
              decfsz        Hz2000cnt,F    ; Count down until Hz2000
              goto          $+4            ; Not time yet
              movlw         HZ2000T        ; Reset the counter so
              movwf         Hz2000cnt      ; it's available next time
              call          Hz2000         ; Go do 2000X per second code
;-------------------------------------------------------------------
;       Check for 10 times per second
;-------------------------------------------------------------------
              decfsz        Hz10Cnt,F      ; Count down until Hz10
              goto          $+4            ; Not time yet
              movlw         HZ10T          ; Reset the counter so
              movwf         Hz10Cnt        ; it's available next time
              call          Hz10           ; Go do 10X per second code

              goto          main
              end
```

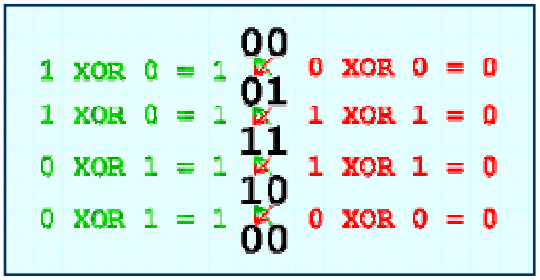
A Simpler Approach

Introduction

The table lookup approach used in the previous example is straightforward and easy to understand. However, it is somewhat lengthy. Since the Gray code is a very simple, repeatable code, it would seem that a simpler algorithm might be possible. In fact, there are a number of ways to shorten the code.

Examining the pattern

If we list how the transitions occur in the gray code, some patterns become evident. It is clear, for example, that on each transition, only one bit can change. If we study the list, a relationship between the low order bit of the previous transition and the high order bit of the current transition becomes evident.



If we look at this pair, the two bits are always the same going in one direction, and always different going in the opposite direction. This means if we XOR the two bits, the result will be one for one direction, and zero for the other.

Code for this example

In the earlier example, the table took care of the situation where there had been no change. In this case, however, we need to check to whether the input has changed before testing the left and right bits:

```
        ; Read the encoder.  If the reading has changed, go ahead
        ; and see what direction the change implies.
Hz2000
        ; Read the encoder bits
                movf            PORTA,W         ; Pick up the input word
                andlw           B'00000011'     ; Mask off all but encoder
                movwf           ThisRead        ; And save into current reading

        ; If the current reading is the same as the previous reading,
        ; we don't want to change the LEDs
                xorwf           LastRead,W      ; Previous reading
                btfsc           STATUS,Z        ; Same?
                return                          ; Yes, do nothing
```

Continued on next page

A Simpler Approach, Continued

Code for this example (continued)

As in the previous example, we will save the reading by rotating the previous reading to make room for the new. However, in this case, we will only rotate the old reading one bit, then XOR the new reading into the saved word. We can then test the middle bit (bit 1) to determine which way the shaft is turning:

```
; We now want to check the result or the right bit of the
; previous reading XORed with the left bit of the current
        movf            ThisRead,W      ; Remember for
        movwf           LastRead        ; next time

; Move last reading over 1 and mask other bits
        rlf             Input,F         ; Rotate the input storage
        movlw           B'00000110'     ; Keep 2 bits from last time
        andwf           Input,F         ; but clear all others

; XOR current status into input word
        movf            ThisRead,W      ; Pick up current reading
        xorwf           Input,F         ; And XOR it into the input

; Set LEDs
        movlw           B'00001110'     ; Initially set the outputs
        movwf           Output          ; to all LEDs off

        btfss           Input,1         ; Test bit 1
        goto            SetUp           ; Move up
        goto            SetDn           ; Move Down
```

This is quite a bit shorter than our table lookup, although it is somewhat less obvious. The astute student will note that we could save an instruction by clearing the carry before the rotate instead of performing and AND with a constant. The higher bits don't matter to us. While this shortens the routine yet again, it does so at the expense of readability. This is a common problem. The programmer is often faced with the tradeoff of efficiency versus readability.

Additional Approaches

Introduction	We have examined two ways of reading the encoder, and have looked at debouncing the encoder in much the same way we debounced switches. However, there are probably as many approaches as there are programmers. Here we will briefly review some other alternatives.
Capacitors	We mentioned that we frequently might not want to bypass the encoder outputs with capacitors. For some encoders, particularly those where the bounce is short-lived, smoothing the bounce with capacitors can simplify our code by eliminating the need to include debounce code. The experimenter is cautioned, however, that getting the capacitor size right is the same problem as getting the debounce code timing right. As the capacitor gets better at smoothing out the bounces, it also hides the transitions. This can cause problems when the user turns the shaft quickly, especially with higher resolution encoders. It should be noted that, other than bypassing RF, anything that could be done with capacitors can be done in code. However, too much capacitance can make it impossible to fix the problem in code. Nevertheless, some people are more comfortable with the hardware, and may find experimenting with components more attractive than experimenting with the code. In the best case, one may wish to study the encoder output with an oscilloscope to help identify the size of the capacitors to use. For most encoders, 0.01uF isn't a bad starting point. However, for encoders with long lived bounce this may be low by orders of magnitude. Similarly, for high resolution encoders this may be too high a value.
Backlash	Some encoders, especially higher resolution encoders, can exhibit a certain amount of backlash when the user releases the knob. Contact bounce can also masquerade as backlash. One solution sometimes employed is for the code to ignore a change in direction unless two transitions show the same direction. Often, this approach will be a satisfactory alternative to debouncing the transitions.
Reduced Resolution	On encoders with detents, the encoder must often undergo two or four transitions between detents. Since the user cannot stop between detents, many applications can benefit from only updating the application after these two or four transitions. When an encoder is available with resolution higher than is actually needed for the application, this approach can result in a smoother response as viewed by the user. Another reduced resolution approach is to only examine the encoder when one of the outputs is in a particular state. This is most commonly used if one of the encoder outputs is connected to the interrupt on change pin of the PIC. This approach can result in a more responsive application when the PIC's resources are heavily committed to other tasks and the developer does not wish to spend a lot of time polling the encoder.
Summary	We have presented a few ways to read the encoder, but certainly have not exhausted all the possibilities. The student is encouraged to experiment with encoder algorithms and timing to find the range of possibilities for different applications and encoders.

Wrap Up

Summary	In this lesson, we have examined various types of rotary encoders, and discussed some of the advantages and disadvantages of each. We have looked at how to decode the output of the encoder, and have reviewed a number of approaches to handling some of the issues with various encoders.
Coming Up	Up until now, all our code has been absolute. While this is simple, it makes it hard to re-use code from earlier projects. In the next lesson, we will examine relocatable code which allows us to build libraries of routines that we can exploit in later projects. This will provide a foundation for later lessons that build our skills in dealing with the LCD and the DDS.