

Enhancing the User interface for the 16 Bit Micro Experimenter

This is the seventh in a series of articles describing the capabilities for the 16 Bit Micro Experimenter (Experimenter for short) and its associated hardware features (see Dec 2009, Feb 2010, Apr 2010, June 2010, August 2010, and Oct 2010). In this article we will look at enhancing the Experimenter user interface with a rotary encoder using the PIC24F timer peripheral set and its interrupt capability. The timer peripheral is a significant component in PIC24F peripheral arsenal. With the PIC24F timer we can count external pulses, or optionally count the internal 16 MHz PIC24F CPU clock as a timing source. All timer operations occur in hardware with minimal need for software intervention---- a really powerful capability to incorporate into Experimenter applications.

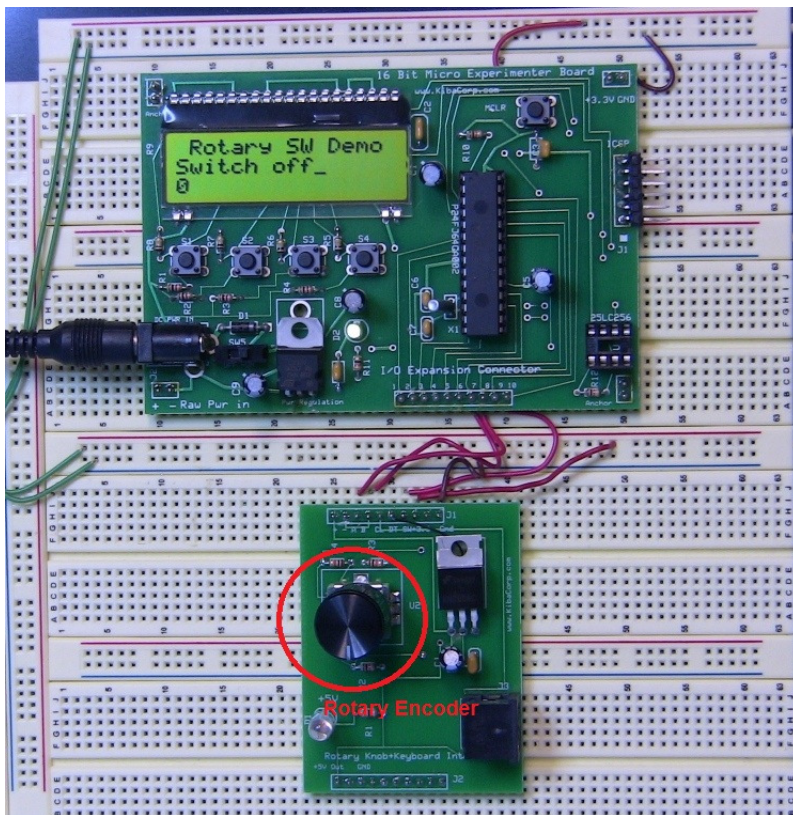


Figure 1 Experimenter with Rotary Encoder

There is a lot of ground to cover here, but no worries, as we supply code examples, electronics parts list, schematics, as well as a mini-kit to help with these examples. As stated in earlier articles, all software will be kept to general level functional calls however; some basic familiarity with C language syntax is required, as we will be incorporating some of the 'C' code libraries introduced in earlier Nuts and Volts articles that discussed the 16 bit Experimenter.

Understanding the Rotary Encoder

Let's look at the rotary encoder and understand its operation. Most modern home and car stereos use mechanical rotary encoders for volume. They are also used in a variety of home appliances, like ovens, for temperature and timing settings. The rotary encoder is different from a potentiometer in that the encoder can undergo full rotation without limits. It is also different from a potentiometer in that we have two outputs: A & B. Each output signal is the result of the opening and closing of switches internal to the rotary encoder as it turns. To insure that digital signals are generated the switch outputs are tied through a pull-up resistors to +3.3 VDC and the common is connected to ground. If a switch is open, a high voltage is generated; if closed, zero or low voltage is generated.

Now let's imagine that the encoder internal shaft has two cams, one for A, and one for B. Each cam has twelve "detent" or protrusions along its circumference that provides 360 degrees/12 or 30 degree resolution with each detent. As a cam turns, each detent opens or closes a switch to generate a waveform (see Figure 3 below). The A&B cams are placed 90 degrees from each other. Because of this placement, as the encoder shaft turns, the A & B output waveforms are 90 degrees out of phase (again look at Figure 3 below.)

The microcontroller samples any of these output waveforms and determines the amount of rotation. If time is also known, speed can also be determined. The final missing ingredient is direction of shaft rotation CW or CCW; this is where both A & B waveforms are used together.

Waveform A is the same no matter what the direction of shaft rotation is. Waveform B reverses its order as shaft turns from CW to CWW. By sampling both waveforms simultaneously and comparing them against one another, the microcontroller determines direction of rotation.

With a rotary encoder the microcontroller can determine direction, speed and amount of rotation. All three of these physical phenomena can be used for operator input. For one of our demos we will use the rotary encoder rotations and direction to cause up/down scrolling of text on the LCD display.

The specific encoder used in these demos is available from Spark Fun (part number COM-09117). This encoder also has a built-in select switch (by pushing in on the encoder knob). We will use all these features in our first experiment.

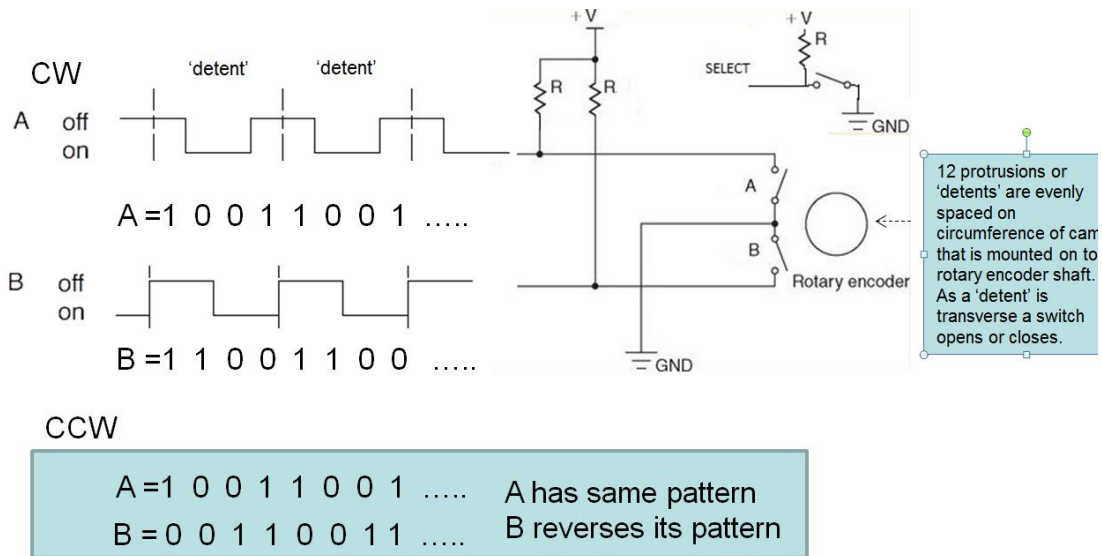


Figure 2 Rotary Encoder CW and CCW Waveforms

Let's review the PIC24F Peripheral Timer

We need the PIC24F timer peripheral and its interrupt capability to work with the rotary encoder. We have plenty of timers with the Experimenter –5 total (Timer 1, Timer 2, Timer 3, Timer 4, and Timer 5). A timer high level block diagram is shown in Figure 3.

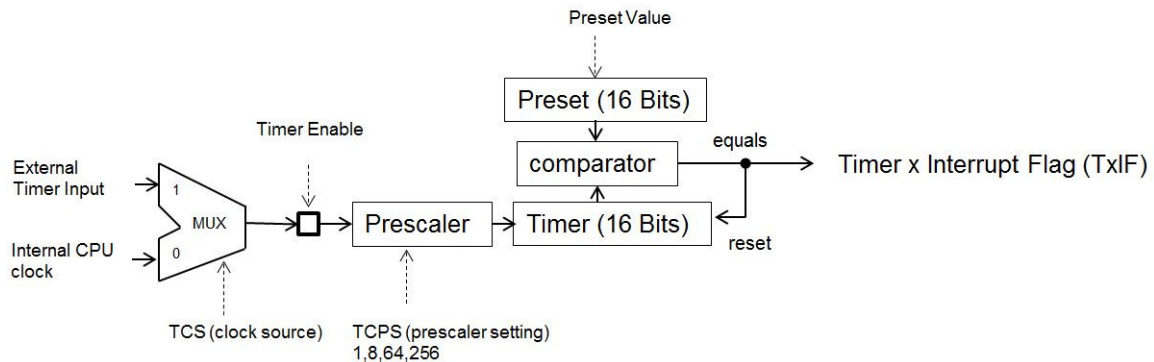


Figure 3 PIC24F Timer Generic Block Diagram

Each element within the timer has its own set of controls that the microcontroller needs to initialize. Starting on the left side of the block diagram is the input clock source selection for the timer controlled by TCS (timer clock select bit). The TCS setting allows for either external clock or the internal PIC24F CPU clock. The PIC24F internal clock is 16 MHz—which is the clock we will use for our demo. The next control is Timer Enable that turns on the timer. The next stage is the prescaler that allows the clock to be scaled down by factors of 1, 8, 64, and 256. The prescaler is set using control bits TCPS (Timer

Clock Prescaler set). All the control bits reside in an internal control register designated Timer Control (T1CON for Timer 1). The final element within the block diagram is the preset register and the timer register both of which are 16 bits wide. The microcontroller writes to the preset register to set the upper count range for the timer. During timer operation the preset value is constantly compared against the running count value of the timer for a match condition. If a match occurs a Timer Interrupt flag is latched to alert the microcontroller. Another action that occurs as a result of this latch is the reset of timer register--- This automatically starts the next timing cycle.

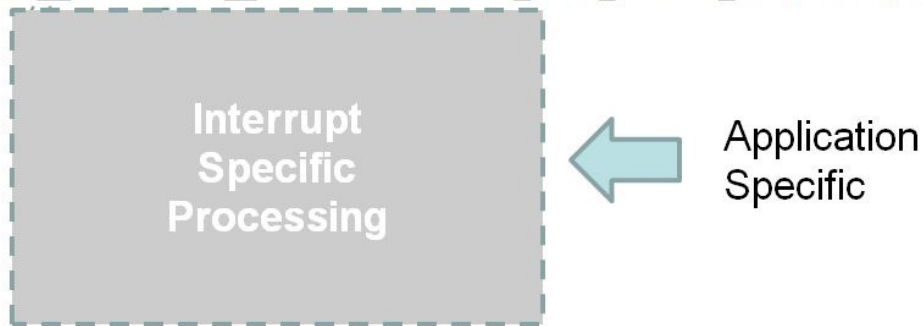
The Timer Interrupt flag can also cause an interrupt to PIC24F if interrupts have been enabled. This is the most efficient way for the microcontroller to perform periodic processing and this is the approach we will use in our demo. There is a code snippet which shows both how the Timer 1 is initialized, and how to enable Timer 1 interrupts. In this snippet the control bits for T1CON are set for internal, the prescaler to 8 giving us a final clock of (16MHz/8) or 2MHz. With preset set to 10,000, the timer will generate interrupts every 5 milliseconds ($1/2\text{MHz} \times 10,000$). The 5 milliseconds are sufficient for capturing any rotation activity and will work well to debounce the A&B switches within the encoder (including the push switch).

```
void TimerInit(void)
{
    //set for ev 5 msec
    PR1 = 10000;
    IPC0bits.T1IP = 5;
    T1CON = 0b1000000000010000; //prescaler =8  TON =1
    IFS0bits.T1IF = 0;
    _T1IE = 1; //enable interrupts
}
```

Figure 4 Timer Initialization

Interrupts appear in the PIC24F C code as unique function declarations as shown in the following template (see figure 5).

```
void __attribute__((interrupt, no_auto_psv)) _T1Interrupt(void) {
```



```
IFS0bits.T1IF = 0;
```

Figure 5 Timer Interrupt Service Template

Interrupt functions accept no parameters and returns no parameters (note the use of void), and each interrupt source has its own specific function call. The code that executes during an interrupt function is called the “interrupt service routine”. General recommendations for writing an interrupt service are; first, to perform the minimal essential processing needed for the service (that is-- get in and out quickly), and, secondly, do not call any other functions during the service. Finally, an interrupt service must reset the original interrupt flag that initiated the interrupt before exiting. If the interrupt service communicates with other parts of the program, then this communication must occur through the use of shared variables.

Interfacing to a Rotary Encoder Using Timer 1

For this demo we use Timer1 with interrupt service to process rotary encoder outputs. As the encoder knob is turned, the Experimenter LCD displays the direction and amount of rotation of the encoder. In addition, the display also shows the condition of the SELECT switch (that is part of the encoder) as ON or OFF, depending on whether you push in the knob. The demo uses the LCD library that was discussed in previous Nuts and Volts articles covering the 16 Bit Experimenter. Figure 6 shows the LCD display. The hook up diagram is shown in Figure 8. This demo is an excellent way to test your rotary encoder and verify its proper operation.

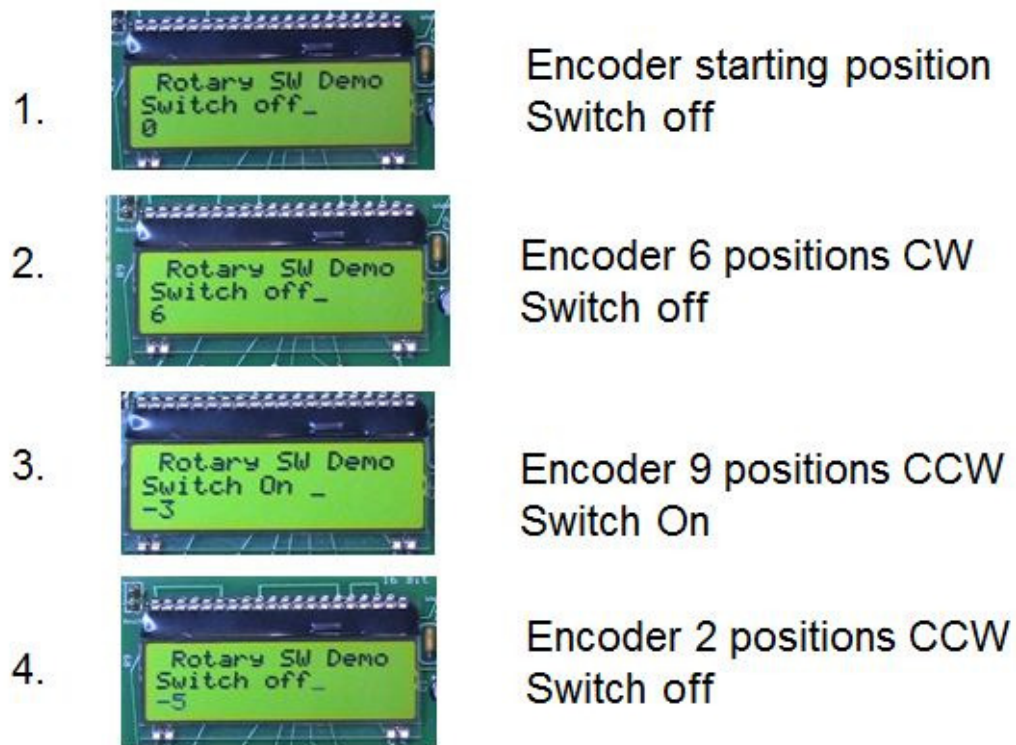


Figure 6 Rotary Encoder Display demo

Let's take a closer look at the interrupt service. With a continuous 5 millisecond interrupt, the interrupt service code samples Output A and determines if it is changing in value from previous sample or remaining the same. If A goes high to low, the code will then look at the Output B and make a determination on direction. Only during CCW operation when A goes high to low will the Output B be zero. If this CCW condition is not detected the rotation direction defaults to CW. In all cases where Output A changes from a low to a high, the condition will count as a "detent" movement.

The interrupt service reports the current encoder position and rotation using two variables (D for direction (1 =CW -1 =CCW), and RCOUNT (Rotational count). RCOUNT will increment or decrement in value based upon rotational direction. As a final function, the interrupt service routine determines knob switch state (ON, OFF) and provides debounce for this switch. The Timer 1 interrupt service code is shown in Figure 7.

```

//T1 interrupt -rotary encoder service
void __attribute__((interrupt, no_auto_psv)) _T1Interrupt(void) {

    static char d;          // direction variable
    switch (RState) {
        default:
            case R_IDLE:    // idle waiting on a change
                if (! ENCHA) //test if A output to go low from high
                {
                    RState =R_DETECT; //yes, change state
                    if (! ENCHB) //check if B output is low direction=CCW
                    d =-1;
                }
                else //else by default direction is CW
                d =1;
                break;
            case R_DETECT:
                if (ENCHA) //test if A output going high from low
                {
                    RState =R_IDLE; //yes, change state
                    RCount +=d; //increment count
                }
                break;
    } //switch
    //debounce switch knob push switch
    if (SW==1) // debounce for knob test switch
    debounce =0;
    else debounce++;
    if ((debounce>=10) && (SW==0) )
    SWITCH =1;
    IFS0bits.T1IF = 0;
}

```

Figure 7 Timer1 Interrupt Service Routine

Scrolling text display application using the Rotary Encoder

The first demo verifies operation of the rotary encoder. How about a demo that uses the encoder in a real application? In this second demo (Scrolling demo.MCP) we use the encoder to assist the user in scrolling through a large amount of text that is stored in the microcontroller flash memory and viewed on the LCD. Our Experimenter LCD is a 16X3 display, limiting how much text we can view at a time. In this demo we have hardcoded President Lincoln's entire Gettysburg Address in microcontroller flash. The speech is approximately 1,172 characters and 64 lines of text. The rotary encoder will be used to scroll backwards and forwards to facilitate viewing the entire speech one line at a time.

We use the same interrupt and timer set-ups as in the earlier demo and to this add two new functions:

- **Scroll Down()** –this function advances forward to next line of text in the speech, and for each rotary detent , retrieves this line for display, and halts when it reaches the last line of the speech.
- **Scroll Back()** –this function moves backwards to the previous line of text in the speech, and for each rotary detent , retrieves this line for display, and halts when it reaches the first line of the speech.

Figure 7 shows the scrolling demo in action. Enjoy!

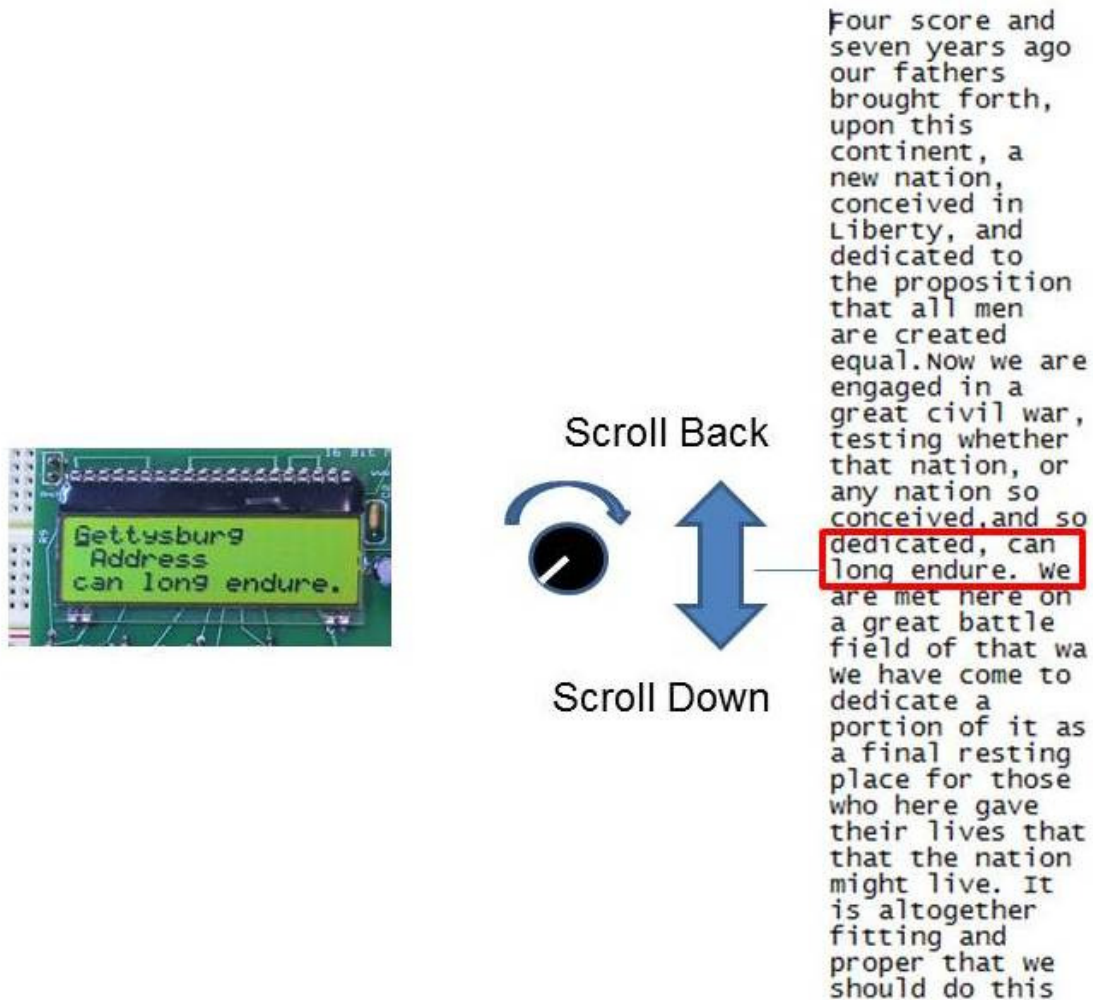


Figure 8 Scrolling demo

Where to go from here?

We have covered a new interface capability for the Experimenter. The rotary encoder really increases our “bag of tricks” and has wide applicability. We also discussed the PIC24F timer and its interrupt capability, applying timers to interface to these input devices. Timers are a broad subject and there are many uses for them. A good reference

book for this material and other applications is available on the Nuts and Volts web store entitled “Programming 16 bit Microcontroller in C” by Lucio Di Jasio. For your convenience there is mini-kit for this article containing all interface electronics, mechanical connectors and associated printed circuit board. This mini-kit is also available from Nuts and Volts. The mini-kit not only contains a rotary knob interface but also a full PS/2 keyboard interface. We cover more of this keyboard interface at another time.

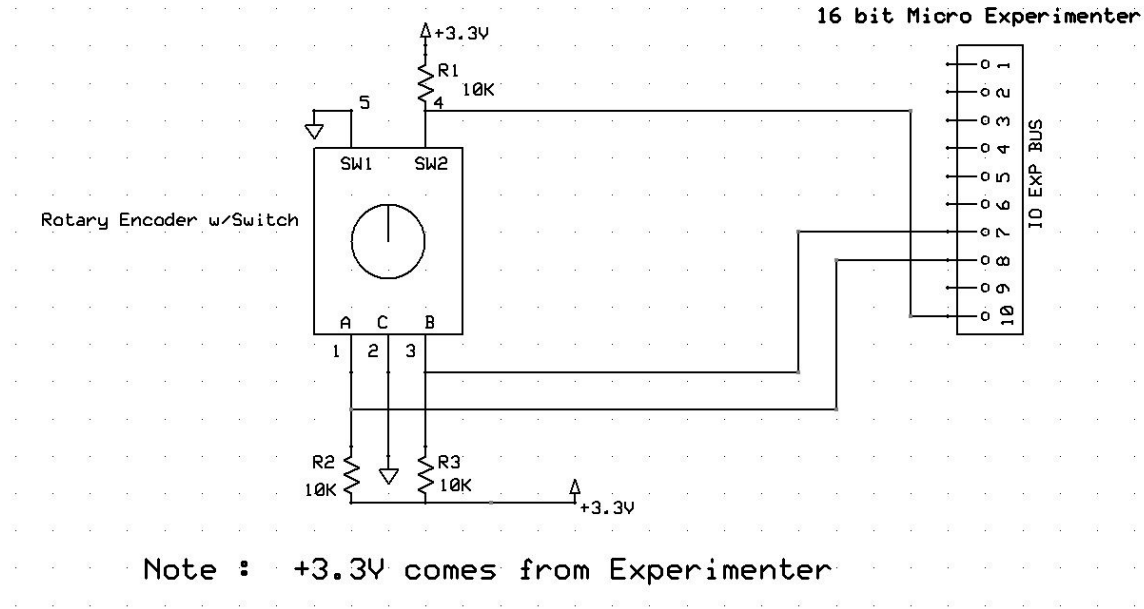


Figure 9 Prototype Hook up

Qty	Part List	Source
3	10K resistor	various
1	Rotary Encoder	Sparkfun COM-09117

Figure 10 Rotary Encoder Parts List