Column #8, October 1995 by Scott Edwards:

Rotary Encoders Help You Program a Friendly Spin-and-Grin Interface

A Digital Dial Plus Header-Post Jumpers

A SAD consequence of the digitization of electronics has been the gradual disappearance of knobs. *Digital* equipment likes its input in terms of *digit*s, not twists of the wrist.

Knobs are elegant and intuitive. Twist clockwise and the parameter being controlled increases; counter-clockwise and it decreases. Natural as breathing. Thankfully, the user-interface used by the largest number of people and requiring the most precise and reflexive control, the steering wheel of a car, is a big knob.

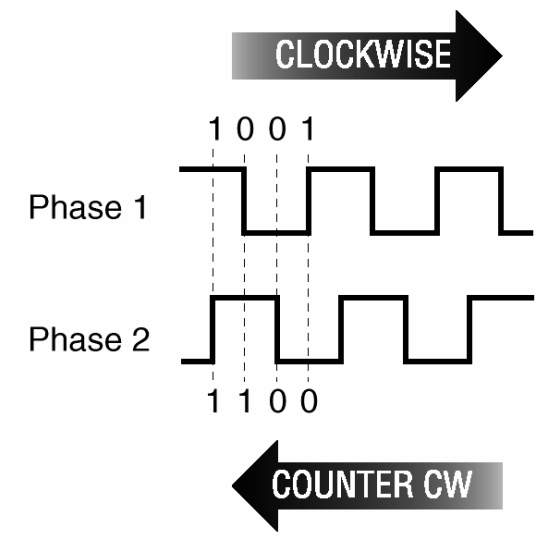
Knobs are battling back from the brink of extinction, though, now that digital versions called *rotary encoders* are being used in more designs. In this month's column, I'll show you a simple method for converting the output of a standard rotary encoder into data your Stamp can use. In the second half of the column, I'll introduce you to a technique for making tidy wiring harnesses that connect to those ubiquitous square header posts (like the pins on the Stamp board).

Rotary Encoders

The rotary encoders we're going to discuss today are properly known as "incremental" rotary encoders." There are others known as "absolute" encoders. An absolute encoder outputs a binary value that's proportional to the angle of the shaft, much the same as an analog pot's resistance depends on its shaft angle. The resolution of an absolute encoder is a function of the number of output bits: An eight-bit encoder breaks a full rotation (360 degrees) into 256 parts; 10 bits, 1024 parts; 12 bits, 4096 parts.

An incremental encoder is different. It has only two output bits, regardless of its angular resolution. As the shaft rotates, the bits change in the sequence shown in Figure 8.1. The encoder's resolution determines how far you must turn the encoder shaft before there is a change in one of the outputs.

Figure 8.1: The sequence of bits appearing at the outputs of a rotary encoder tells the direction of rotation



Given the output bits shown in the figure, it's fairly easy for a controller to figure out the direction of the encoder shaft's rotation. Let's say that the bits are now 01 and they change to 00. Figure 8.1 shows that as clockwise rotation. If the bits start at 01 and change to 11, that's counterclockwise.

So, one way to interpret the output from an encoder would be to look up the sequence in a pair of tables. However, there's an efficient shortcut method using the exclusive-OR (XOR) operator (expressed as ^ in PBASIC). XOR's effect on bits can be stated as, "If bit A or bit B (but not both) = 1, then bit C = 1." In table form:

A XOR B	= C
0 ^ 0	0
0 ^ 1	1
1 ^ 0	1
1 ^ 1	0

XOR has quite a few uses in programming. Any bit XORed with 1 is inverted; bits XORed with 0 are unchanged. If two bits are equal they XOR to 0; if not equal they XOR to 1.

In the case of the encoder sequence, it turns out that for any given sequence, XORing the righthand bit of the old value with the lefthand bit of the new value tells you the direction of rotation. For example, take the clockwise sequence 01 00: 1 XOR 0 = 1. Now the counter-clockwise sequence 01 11: 1 XOR 1 = 0. This relationship holds for any pair of numbers in either direction.

Figure 8.2 and Listing 8.1 demonstrate how a rotary encoder could be employed in a user-interface application. I got the idea for it from one of my customers. He used a Backpack-equipped LCD to simulate a radio "bandspread" tuning dial. Turn the encoder to the left and the display scrolls to the left; turn it right, the display scrolls right. In my customer's application, the Stamp also updated a frequency synthesizer chip. This gave him knob-controlled tuning with the advantages of digital control and display.

Figure 8.3 is a photo of my breadboard setup. I used an assembled Counterfeit kit (my Stamp-compatible controller) running at four times the normal Stamp speed (16 MHz). This allowed the program to keep up with all but the fastest spins of the dial. If the controller isn't fast enough to track every transition of the encoder's outputs, "slippage" errors occur; the display gets out of sync with the knob. So speed is important.

Because the Counterfeit was running at 16 MHz, all of its instructions were proportionately accelerated. Although the program specifies 2400 baud for serial output to the Backpack-equipped LCD, the actual baud rate is four times that, 9600 baud. I installed a jumper on the Backpack's baud-rate header to set it for 9600 baud too.

Figure 8.2: Connection diagram for the rotary-encoder demo

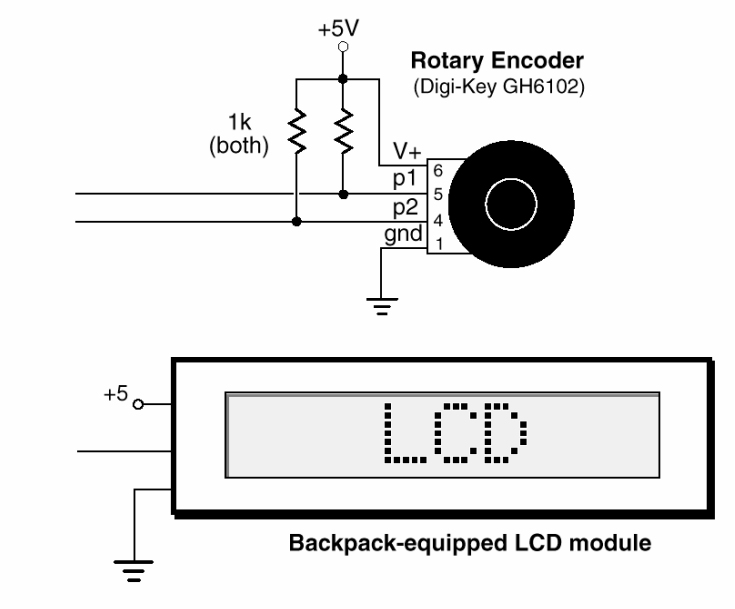


Figure 8.3: Photo of the Rotary-Encoder Setup

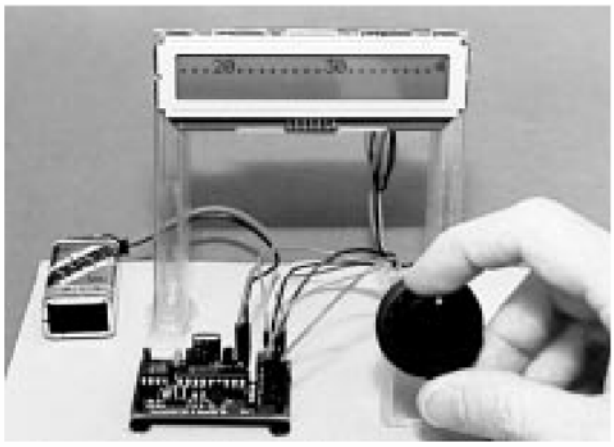


Figure 8.3 also provides a nice introduction to this column's second topic, making jumper wires for convenient breadboarding with the Stamp, Counterfeit and accessories. Everything in the picture is hooked up with the kind of jumpers described in the next section.

Making Connections

The Stamp, its cousin the Counterfeit, and accessories like the Stretcher and Backpack, provide 0.025-inch square metal posts for making connections to the outside world. I've slowly become aware that most users aren't quite sure what to *do* with these connectors, known as header stakes. They usually end up wire-wrapping to make their connections.

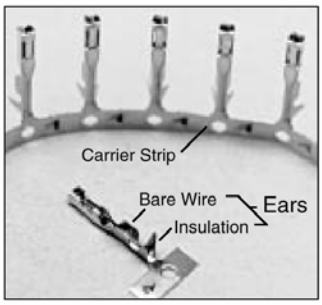
They also end up *u*nwrapping old connections. And fixing bad connections. There's a better way. If you're willing to invest $8 in a tool and a few bucks more in a supply of connectors, you can make slick, secure jumper wires and wiring harnesses. Jameco (Sources) carries "female crimp pins" that are designed to fit 0.025-inch header stakes. The pins are stock number 100765 and cost a dime apiece in quantities of 10 or more. These pins are designed to fit into plastic housings to make tidy, detachable wiring harnesses of the sort you find inside PCs.

You can crimp these pin/sockets onto the ends of 22, 24 or 26-gauge stranded hookup wire with the help of Jameco's tool, stock number 99442 ($7.95). Here's how:

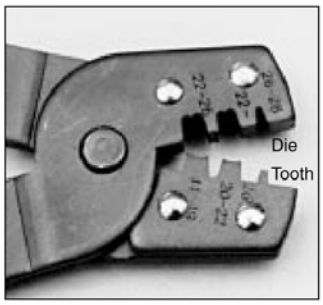
The crimp pins come attached to a ribbon of metal called a carrier strip. Cut a one-pin section from the end of this strip, leaving the pin attached to the piece of metal. This tab of carrier-strip metal makes a nice handle for the tiny pin. Strip 1/4 inch of insulation from the end of a piece of hookup wire. Hold the pin by the carrier tab and load the wire into the crimp pin as shown in Figure 8.4. See the two sets of ears on the crimp pin? Line up the insulation with the back set and the bare wire with the front set.

Using the very end of the crimping tool, pinch the ears gently inward toward the wire. This ensures that it will fit into the concave crimping die. Next, position the back end of the crimp pin against the smallest tooth with the ears pointing into the die. Line the ears up with the funnel-shaped opening of the die, and squeeze the handles of the tool. When you open the crimp tool, you'll see that it has neatly wrapped one set of ears around the insulation, and the other around the bare wire. Pinch the bare-wire part of the crimp with the corners of the tips of the tool as final insurance of good contact.

Figure 8.4: Making Stamp-compatible jumpers is easy with this procedure



0.025" Socket Crimp Pins



Crimping Tool

Grasp the wire and socket, using carrier-strip tab as a handle.

Use the end of the tool and gently pinch the ears to fit the die.

Put the connector into the tool with the ears facing into the die. Squeeze hard.

Done. Give the bare-wire ears a final crimp to ensure good contact, and break off the carrier-strip tab.

You can use this connector as-is, or you can cover it with a short piece of heat-shrink tubing. If your wiring scheme permits, you can even use one of those neat plastic housings listed on the same page of the Jameco catalog as the pins. Just slip the pins into the housing until you hear a click. A latch inside the housing prevents the pin from slipping out.

One more benefit of header-socket pins: they fit perfectly on 22-gauge solid hookup wire for making temporary connections to breadboards and other circuits. By the way, if you're a manufacturer of wire and cable goodies who could make the above-described jumper wires in the low thousands for a reasonable price, get in touch (see Sources for my contact information). I'd love to add prefabricated header-socket jumpers to my expanding line of Stamp goodies, but I haven't found anyone to make them for a decent price.

```
' Program Listing 8.1 Demonstrating Use Of a Rotary Encoder
' Program: Rotary.BAS (Read a rotary encoder and scroll a display)
' This program demonstrates the use of a digital rotary encoder
' as a unique user-interface control. Turning the knob scrolls a
' virtual tuning dial displayed on an LCD. To keep the code and
' hardware as simple as possible, the LCD is equipped with an
' LCD Serial Backpack, which interprets data and instructions sent
' to it serially.
SYMBOL old = b0               ' Previous state of encoder bits.
SYMBOL new = b1               ' Current state of encoder bits.
SYMBOL directn = bit0         ' Direction of encoder travel; 1=CW.
SYMBOL index1 = b2            ' For/Next counter variable.
SYMBOL index2 = b3            ' For/Next counter variable.
SYMBOL I = 254                ' Instruction-toggle for LCD.
SYMBOL LCD_cls = 1            ' Clear-screen instruction for LCD.
SYMBOL left = 24              ' Scroll-left instruction for LCD.
SYMBOL right = 28             ' Scroll-right instruction for LCD.
' The program starts by printing a scale on the LCD screen.
' The LCD's RAM can hold up to 80 characters and scroll them
' circularly across the display. The code below prints...
' 0........10........20........30........40........50..
' ...up to 70. Only the first 24 characters are initially
' visible on the display, but turning the encoder knob scrolls
' them into view, like an old-fashioned radio tuning dial.
pause 1000                    ' Let LCD initialize.
serout 0,n2400,(I,LCD_cls)    ' Clear LCD screen.
for index1 = 0 to 70 step 10  ' Scale: 0-70 (uses 80-char LCD RAM).
serout 0,n2400,(#index1)      ' Print number on the screen.
for index2 = 1 to 8
serout 0,n2400,(".")          ' Print "........" between numbers.
next
next
' Before entering the main loop, the program stores the beginning
' state of the encoder bits into the variable 'new.' It ANDs the
' pins with %11000000 in order to strip off all bits except for
' 6 and 7. (ANDing a bit with 0 always produces 0; ANDing with 1
' copies the state of the bit.)
let new = pins & %11000000    ' Mask off all but bits 6 & 7.
```

```
start:
  let old = new & %11000000   ' Mask bits and copy new into old.
again:
  let new = pins & %11000000  ' Copy encoder bits to new.
  if new = old then again     ' If no change, try again.
  let directn = bit6 ^ bit15  ' XOR right bit of new w/ left bit of old.
  if directn = 1 then CW      ' If result=1, encoder turned clockwise.
  serout 0,n2400,(I,left)     ' If result=0, counterclock (scroll left).
goto start                    ' Do it again.
CW:
  serout 0,n2400,(I,right)    ' Clockwise (scroll right).
goto start                    ' Do it again.
```