Reading Rotary Encoders

Introduction. This application note covers the use of incremental rotary encoders with PIC microcontrollers. It presents an example program in TechTools assembly language for reading a typical encoder and displaying the results as an up/down count on a seven-segment LED display.

Background. Incremental rotary encoders provide a pair of digital signals that allow a microcontroller to determine the speed and direction of a shaft's rotation. They can be used to monitor motors and mechanisms, or to provide a control-knob user interface. The best-known application for rotary encoders is the mouse, which contains two encoders that track the x- and y-axis movements of a ball in the device's underside.

Rotary encoders generally contain a simple electro-optical mechanism

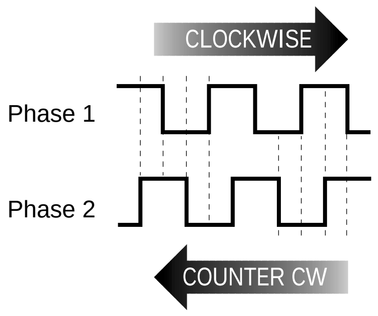


Figure 1. Quadrature waveforms from a rotary encoder contain directional information.

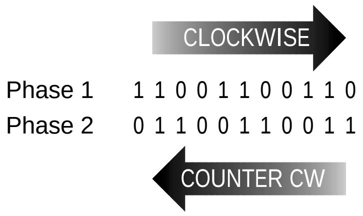
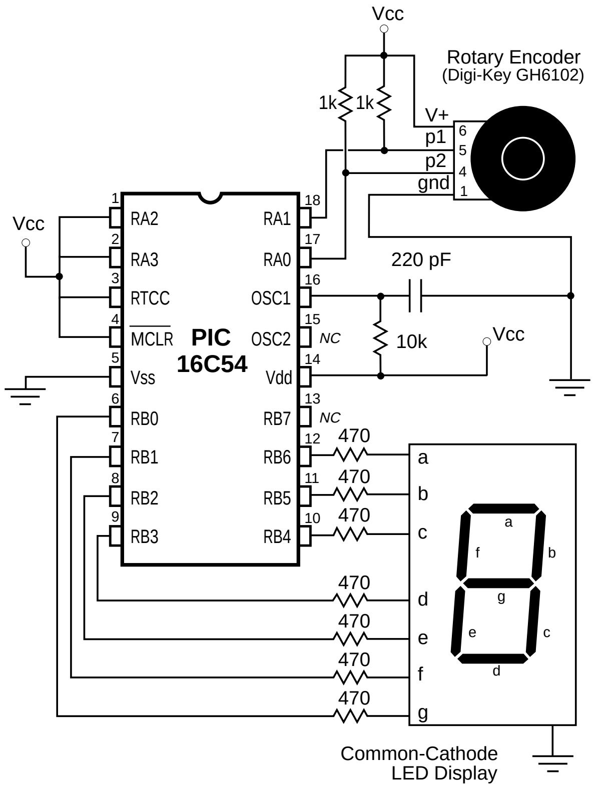


Figure 2. Sequence of two-bit numbers output by the phases of a rotary encoder.

consisting of a slotted wheel, two LEDs, and two light sensors. Each LED/sensor pair is arranged so that the devices face each other through the slots in the wheel. As the wheel turns, it alternately blocks and passes light, resulting in square wave outputs from the sensors.

The LED/sensor pairs are mounted offset relative to one another, so that the output square waves are offset 90 degrees in phase. This is known as quadrature, because 90 degrees amount to one-quarter of a full 360-degree cycle.



This phase relationship provides the information needed to determine the encoder’s direction of rotation (see figure 1).

The dotted lines in figure 1 indicate a common method of reading direction. For instance, if phase 1 is high and phase 2 is rising, the direction is clockwise (CW). If phase 1 is low and phase 2 is rising, the direction is counterclockwise (CCW).

For the sake of interpreting this output with a PIC or other microcontroller, it's probably more useful to look at the changing states of the phases as a series of two-bit numbers, as shown in figure 2 above.

When the encoder shaft is turning CW, you get a different sequence of numbers (01,00,10,11) than when it is turning CCW (01,11,10,00). You may recognize this sequence as Gray code. It is distinguished by the fact that only one bit changes in any transition. Gray code produces no incorrect intermediate values when the count rolls over. In normal binary counting, 11 rolls over to 00. If one bit changed slightly before the other, the intermediate number value could be incorrectly read as 01 or 10 before settling into the correct state of 00.

Interpreting this code amounts to comparing the incoming sequence to the known sequences for CW and CCW rotation. A lookup table would do the trick. However, this approach, while easy to understand, is inefficient. The shortcut method uses an interesting property of the two-bit Gray code sequence.

Pick any pair of two-bit numbers from the CW sequence shown in figure 2; for instance, the first two: 10, 11. Compute the exclusive-OR (XOR) of the righthand bit of the first number with the lefthand bit of the second. In this case, that would be 0 XOR 1 = 1. Try this for any CW pair of numbers from the table, and you'll always get 1.
Now reverse the order of the number pair: 11, 10. XOR the right bit of the first with the left of the second (1 XOR 1 = 0). Any CCW pair of numbers will produce a 0.

How it works. The schematic in figure 3 shows a typical rotary encoder connected to the lower two bits of port RA, and a seven-segment LED display to port RB. The circuit performs a simple task: the count displayed on the LED goes up when the control is turned CW, and down when it is turned CCW. The display is in hexadecimal, using seven-segment approximations of the letters: A, b, C, d, E, and F.

The program begins by setting up the I/O ports and clearing the variable *counter*. It gets an initial input from the encoder, which goes into the variable *old*, and strips off all but the two least-significant bits (LSBs).

The body of the program, starting with *:loop*, calls *check_encoder* and then displays the latest value of *counter* on the LED display.

Most of the interesting business happens in *check_encoder* itself. Here, the program gets the latest value at the encoder inputs, strips all but the two LSBs, and XORs the result into a copy of the old value. If the result is zero, the encoder hasn't moved since its last reading, and the routine returns without changing the value of *counter*.

If the value has changed, the routine moves the value in *old* one bit to the left in order to align its LSB with the high bit of the two-bit value in *new*. It XORs the variables together. It then examines the bit *old.1*. If the bit is 1, *counter* is incremented; if it's 0, *counter* is decremented.

Modifications. To avoid 'slippage' errors (where a change in encoder position does not change the counter, or results in the wrong change), *check_encoder* must be called at least once every 1/(encoder resolution * max revs per second). For instance, if the encoder might turn 300 rpm (5 revs per second) and its resolution is 32 transitions per turn, *check_encoder* must be called every 1/(32*5) seconds, or 6.25 milliseconds. For a user interface, bear in mind that generally the larger the knob, the slower the input. Substitution of a larger control knob may be all that's required to reduce the sampling rate.

In circumstances where electrical noise might be a problem, Microchip's PIC data sheet indicates that it might be wise to move the port I/O assignments to the beginning of *check_encoder*. Electrostatic discharge (ESD) from the user's fingertips, or some other electrical noise, could corrupt an I/O control register. This would prevent the routine from reading the encoder.

Program listing. This program may be downloaded from our Internet ftp site at ftp.tech-tools.com. The ftp site may be accessed directly or through our web site at http://www.tech-tools.com.

; PROGRAM: Read Rotary Encoder (RD_ENCDR.SRC)

```
; This program accepts input from a rotary encoder through bits RA.0 and RA.1,
; determines the direction of rotation, and increments or decrements a counter
; appropriately. It displays the hexadecimal contents of the four-bit counter on a
; seven-segment LED display connected to port RB.

; Remember to change device info when programming a different PIC.
            device      pic16c54,rc_osc,wdt_off,protect_off
            reset       start

encoder     =           ra
display     =           rb

; Variable storage above special-purpose registers.
            org         8

temp        ds          1
counter     ds          1
old         ds          1
new         ds          1

; Set starting point in program ROM to zero.
            org         0

start       mov         !rb, #0             ; Set rb to output.
            mov         !ra, #255           ; Set ra to input.
            clr         counter
            mov         old, encoder
            and         old, #00000011b
:loop       call        chk_encoder
            mov         w, counter
            call        sevenseg
            mov         display, w
            goto        :loop

chk_encoder mov         new, encoder        ; Get latest state of input bits.
            and         new, #00000011b     ; Strip off all but the encoder bits.
            mov         temp, new
            xor         temp, old           ; Is new = old?
            jz          :return             ; If so, return without changing
                                            ; counter.
            clc                             ; Clear carry in preparation for
                                            ; rotate-left instruction.
            rl          old                 ; Move old to the left to align old.0
                                            ; with new.1.
            xor         old, new
            jb          old.1, :up          ; If the XOR resut is 1, increment
                                            ; counter, otherwise decrement.
:down       dec         counter
```

```
                skip
:up             inc         counter
                and         counter, #00001111b
                mov         old,new
:return         ret

sevenseg        jmp         pc+w                    ; display lookup table
                retw        126, 48, 109, 121, 51, 91, 95, 112
                retw        127, 115, 119, 31, 78, 61, 79, 71
```