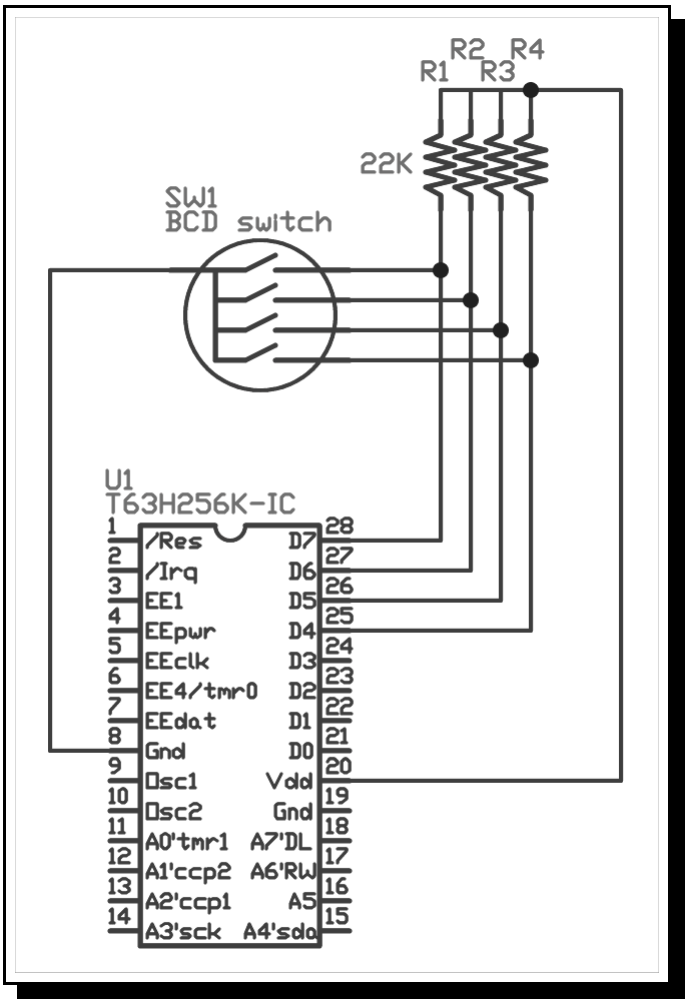
AN043 - Methods for using thumbwheel switches as numeric input devices.

Submitted by:

Glenn Clark
Protean Logic Inc.

In this application note common “thumbwheel” switches are used as input devices for the TICkit processor. Thumbwheel switches, or BCD switches are actually four SPST switches that are mechanically connected to generate a binary code that matches the mechanically selected digit. This approach offers the advantage of confirming a users selection without having some system to output numeric data. For user applications where there is no other requirement for output, thumbwheel switches are a logic choice for reliable numeric input. “Thumbwheel” switches come in a variety of packages. Some have up and down buttons to select different values, others have little handles on the wheel, still others have values that are selected with a screwdriver or small knob. The latter variety are used for setup values which do not change frequently.

The figure below shows a typical BCD thumbwheel switch connected to a TICkit processor. Obviously, many TICkit connections are not shown in this diagram. The common element of the single digit switch is connected to ground, while the four switched legs are connected to four ‘D’ port pins are pulled high. As different values are selected on the switch, the binary code for that value appears at the four ‘D’ pins. One thing to notice here is that the common is tied to ground while the switched legs are pulled high. This produces the inverse of the selected value. For example, a value showing on the switch of ‘3’ closes switches for D4 and D4 to produce a pattern of 1100. The correct value for 3 is 0011. So, the value will have to be inverted again in software. The reason the common pole of the switch was grounded will become clearer in later examples where binary decoders are used to select between many switches. For now lets just say it goes back to a very old TTL convention.

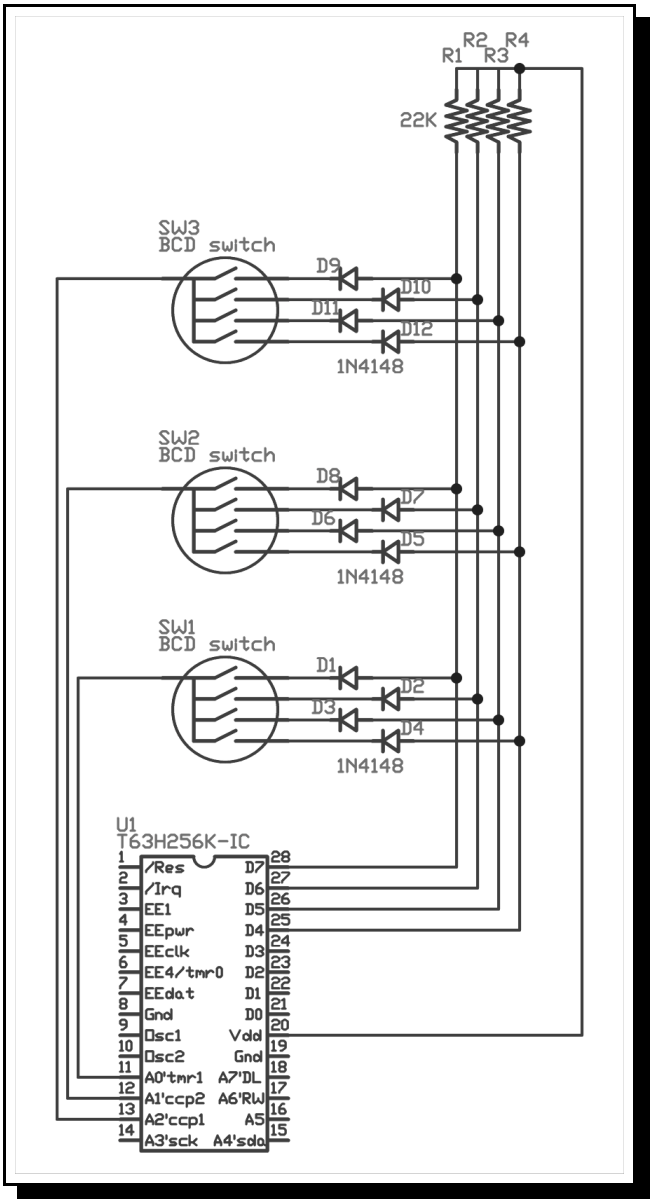


To read the value of the switches simply use the dport_get() function, mask out the pins D0-D3 and shift the resulting value into the right ordinal position by dividing by 16. The code fragment below shows how this is done.

```
FUNC none main
    LOCAL byte bcd_val
BEGIN
    dtris_set( 0y11110000b )
    ; make upper for pins inputs

    REP
        ; read the pins
        =( bcd_val, dport_get())
        =( bcd_val, b_and( 0xF0b, bcd_val )
        =( bcd_val, b_xor( 0xF0b, bcd_val )
        =( bcd_val, /( bcd_val, 16b ))
        con_out( bcd_val )
    LOOP
```

```
ENDFUN
```

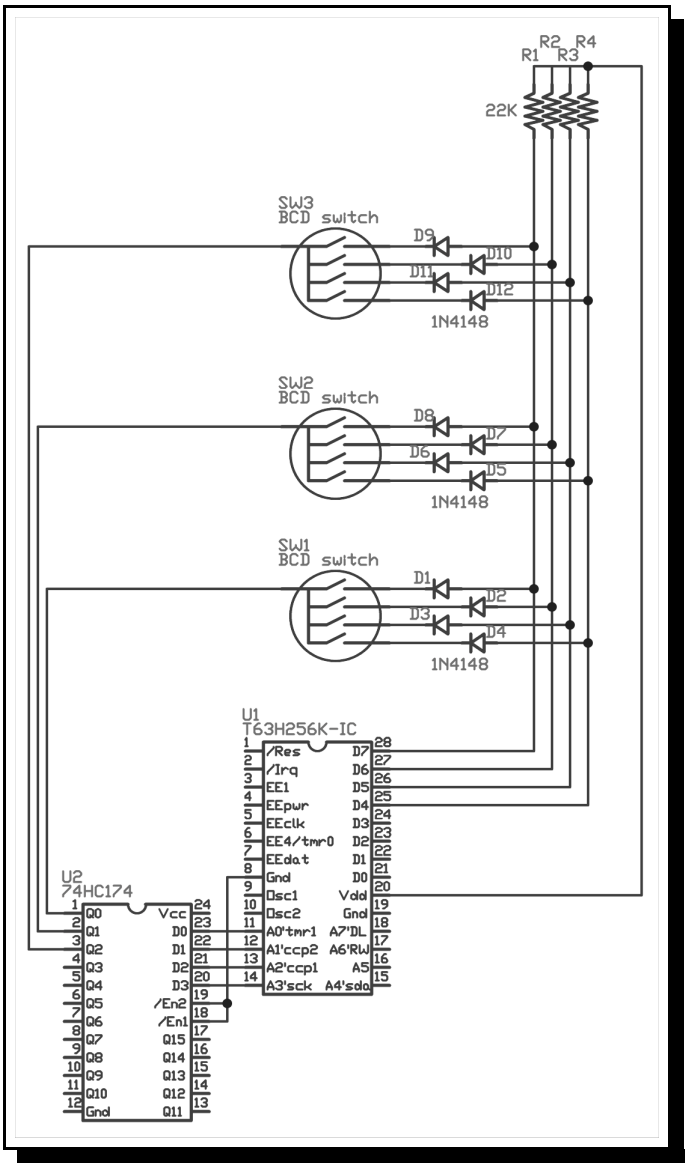
As you can see, reading the value is actually very easy. In practice, however, there are usually many switches required to create a usable numeric input. There are obviously a limited number of input pins on the TICkit, so a method must be employed which reuses pins for multiple switches. This method is a 'bus' method. By using some small signal diodes, the switch outputs of multiple switches can be combined on the same four I/O pins. The TICkit can control which switch values are on the I/O lines by controlling which switch common is low. The circuit to the right illustrates this method.



In this circuit, diodes have been added to prevent the switch settings of one switch from creating false readings on another switch. Also, the switch common for each decade switch has been tied to an I/O pin. By lowering the common of the switch we want to read while keeping all the other commons high, each decade switch can be read individually. This allows us to read 12 switches by only using 7 I/O lines. The code to read this arrangement follows:

```
; program to read 3 decades
; and assemble the corresponding
; word value

FUNC none main
    LOCAL byte sw_in
    LOCAL word val_in
BEGIN
    dtris_set( 0y11110000b )
    pin_high( pin_a0 )
    pin_high( pin_a1 )
    pin_high( pin_a2 )
    ; make the pins inputs

    REP
        =( val_in, 0w )
        ; read the pins

        ; read switch  one
        Pin_low( pin_a0 )
        =( sw_in, dport_get())
        =( sw_in, b_and( 0xF0b, sw_in )
        =( sw_in, b_xor( 0xF0b, sw_in )
        =( sw_in, /( sw_in, 16b ))
        Pin_high( pin_a0 )

        =( val_in, +( val_in, sw_in ))

        Pin_low( pin_a1 )
        =( sw_in, dport_get())
        =( sw_in, b_and( 0xF0b, sw_in )
        =( sw_in, b_xor( 0xF0b, sw_in )
        =( sw_in, /( sw_in, 16b ))
        Pin_high( pin_a1 )


        =( val_in, +( val_in, *( sw_in, 10w )))
```

```
            Pin_low( pin_a2 )
            =( sw_in, dport_get())
            =( sw_in, b_and( 0xF0b, sw_in )
            =( sw_in, b_xor( 0xF0b, sw_in )
            =( sw_in, /( sw_in, 16b ))
            Pin_high( pin_a2 )

            =( val_in, +( val_in, *( sw_in, 100w )))

            con_out( val_in )
        LOOP
    ENDFUN
```

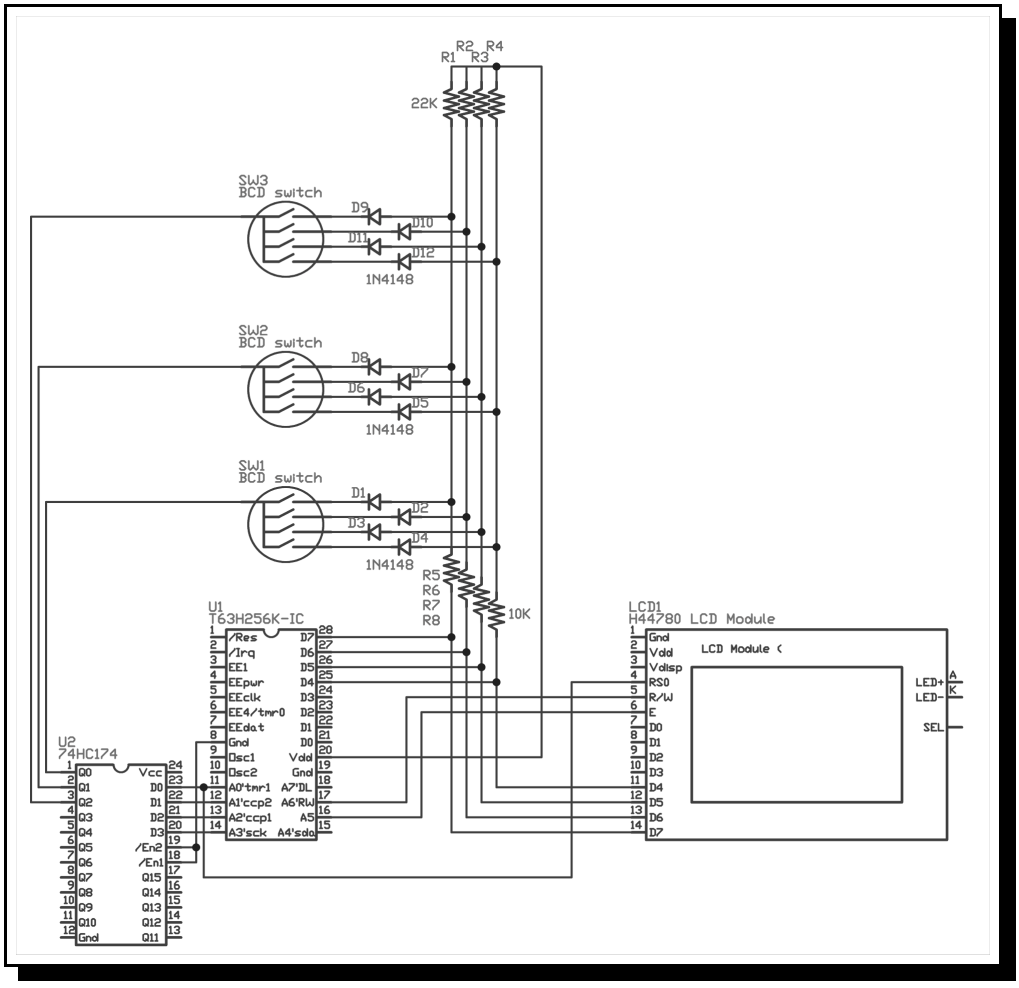


Controlling the common of each switch using an I/O line is effective and requires little additional circuitry, however it still may consume too many I/O lines when many switches are required. The same basic technique of busing the switches can be used in conjunction with a discrete decode IC like the 74HC154 or 74HC138. These devices bring a single output low upon the basis of an applied binary pattern. The diagram to the right shows a 74HC154 being used in this situation. Not all of the switches are shown to keep the drawing size down, but as many as 16 switches can be used in this circuit while still only requiring 8 I/O lines. The use of the decoder ICs demonstrates the convention of "low active" selecting methods where the common of the decade switches are pulled low when selected.

Usually, when many switches are used, there are several values that are being selected. The same decoder can be used to select switches from different "banks" to form several values. For example switches 0 through 4 might be used to select a starting value in and application, while switches 5 through 9 could be used for the stopping value. A program to read 10 switches to assemble two values like those mentioned above is shown here:

```
    FUNC word read switches
        PARAM byte first_sw
        PARAM byte last_sw
        LOCAL byte cur_sw
        LOCAL byte sw_in
        LOCAL word mult 1w
    BEGIN
        =( exit_value, 0w )
        =( cur_sw, first_sw )
        REP
            aport_set( b_or( cur_sw, b_and( aport_get(), 0xF8b )))
            =( sw_in, dport_get())
            =( sw_in, b_and( 0xF0b, sw_in )
            =( sw_in, b_xor( 0xF0b, sw_in )
            =( sw_in, /( sw_in, 16b ))
            =( exit_value, +( exit_value, *( sw_in, mult )))
        LOOP
    ENDFUN
```

```
FUNC none main
    LOCAL word start_val
    LOCAL word stop_val
BEGIN
    =( start_val, read_switches( 0b, 4b ))
    =( stop_val, read_switches( 5b, 9b ))
    Con_string( “Step From: “ )
    Con_out( start_val )
    Con_string( “ through: “ )
    Con_out( stop_val )
    Con_string( “/r/l” )
    REP
        debug_on()
    LOOP
ENDFUN
```



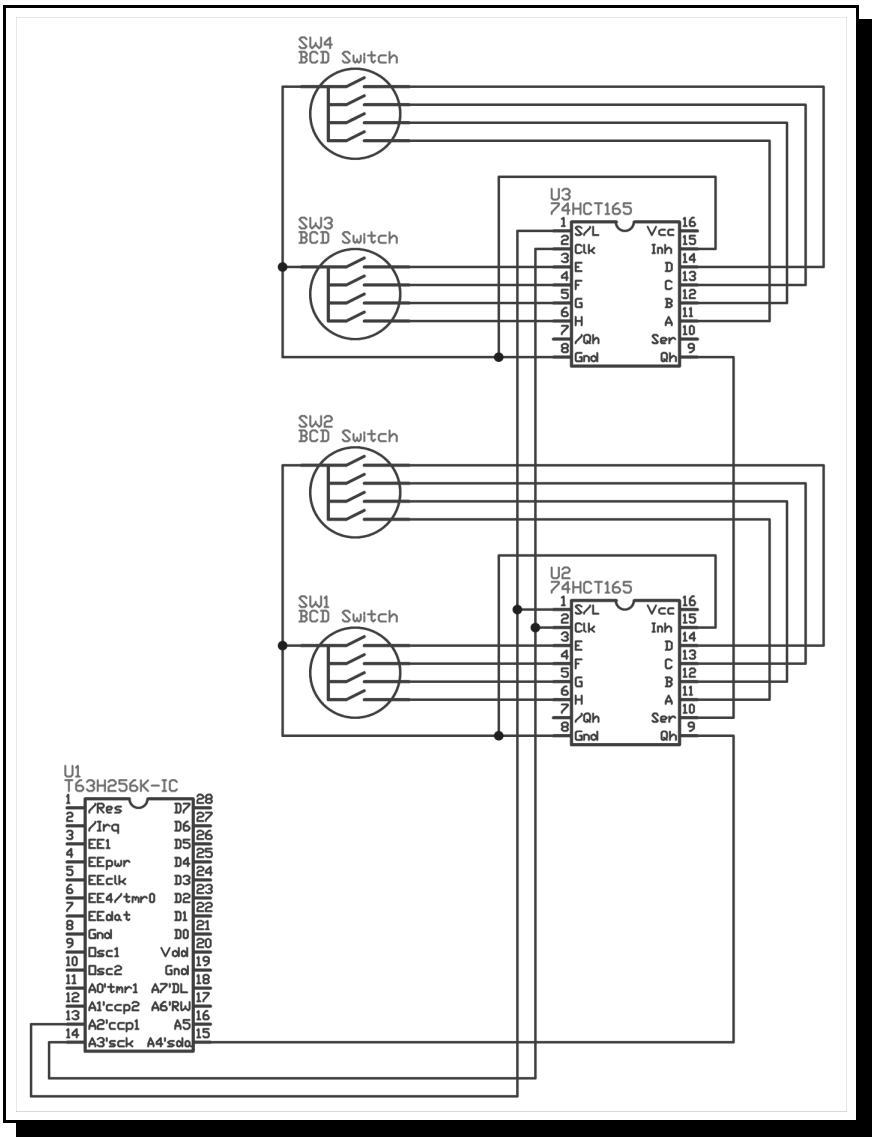
The previous circuit can be used with other bus oriented devices like character LCDs. The circuit shown to the right uses a common 44780 based character display which shares the four data lines and one of the select lines. This is a very nice way to access a lot of switches and output character data while still saving many I/O lines for application specific usage.

Notice that four 10K resistors are used between the switch bus and the LCD this is to ensure that no bus lines are low through the switches while the LCD is outputting. This prevents the possibility of damage should the program mistakenly enable LCD output or TICkit output while a switch is enabled.

The method for reading the switches in this circuit are identical to that used for the previous circuit with the additional requirement that the LCD may alter the selection value and thus the switch selection needs to be updated before any switch is read.

The final example of a interfacing to “thumbwheel” switches uses serial techniques and inexpensive latched shift registers to read as many switches as you want, all while using just three I/O pins on the TICkit processor. This method also had wiring advantages when the front panel has to be cabled to the processor board. It is much easier to cable three wires than

seven, eight or more. You can consult other Protean application notes on serial I/O techniques for higher performance methods of interfacing to shift registers.



The circuit utilizing 74HCT165 ICs is shown on the right. It is important to use the HCT series of integrated circuits as these have a TTL style input. This means that no connection is interpreted as high. This works well with SPST switches like those contained in the thumbwheel switches. If you use HC series or some other series of logic, you will need to put pull-up resistors on all of the switch outputs to ensure a high level when the switches is not closed. The program for interfacing to these devices is shown below. This program utilizes some common libraries included with the development system of the TICkit. You can also use the faster SPI based serial method, but the special considerations of this technique are beyond the scope of this application note.

Assuming that the notation order of the decades corresponds to the switch numbering. (SW1 is ones, SW2 is tens, and so on) Software will need to adjust the wiring wierdnesses. Also note that you can keep adding additional 165 ICs for more switches or digital input without requiring any more TICkit I/O lines. Simply daisy chain the Qh from the additional devices to the SER input of the existing device as shown in this diagram. Then the software generates the additional clocks to shift that data into the TICkit for interpretation.

```
; Sample program to read four BCD switches using 74HCT165 ICs

DEF tic63_i
LIB fbasic.lib


DEF u74165_rst pin_A2           ; pin A2 connects to 74HC165 RCLK select
DEF u74165_clk pin_A3           ; pin A3 connects to 74HC165 SRCLK line
DEF u74165_data pin_A4          ; pin A4 connects to 74HC165 D-IN line

LIB cn_str.lib                  ; console string routines

FUNC byte read_u74165
    LOCAL byte count_out 8b
BEGIN
    pin_low( u74165_clk )       ; make pin an output
```

```
        =( exit_value, 0b )
        REPEAT
            =( exit_value, <<( exit_value ))
            IF pin_in( u74165_data )
                ++( exit_value )
            ENDIF

            pulse_out_high( u74165_clk, 1w )
            --( count_out )
        UNTIL ==( count_out, 0b )
    ENDFUN

    FUNC none main
        LOCAL byte reg_byte
        LOCAL byte sw_in
        LOCAL word val_in
    BEGIN
        pin_high( u74165_rst )
        pin_low( u74165_clk )
        REP
            pin_low( u74165_rst )       ; latch enable chip
            pin_high( u74165_rst )      ; latches inputs into shift registers

            =( Val_in, 0w )

            =( reg_byte, read_u74165())
            =( sw_in, b_xor( 0xF0b, b_and( 0xF0b, reg_byte )))
            =( sw_in, /( sw_in, 16b ))
            =( val_in, +( val_in, sw_in ))

            =( sw_in, b_xor( 0x0Fb, b_and( 0x0Fb, reg_byte )))
            =( val_in, +( val_in, *( sw_in, 10w )))

            =( reg_byte, read_u74165())
            =( sw_in, b_xor( 0xF0b, b_and( 0xF0b, reg_byte )))
            =( sw_in, /( sw_in, 16b ))
            =( val_in, +( val_in, *( sw_in, 100w )))

            =( sw_in, b_xor( 0x0Fb, b_and( 0x0Fb, reg_byte )))
            =( val_in, +( val_in, *( sw_in, 1000w )))

            con_string( “The setting is: “ )
            con_out( val_in )
            con_string( "\r\l" )
        LOOP
    ENDFUN
```

This concludes this application note. BCD switches can make a very profession looking intuitive input interface for a variety of projects and interface to them is relatively simple.