



Column #127, November 2005 by Jon Williams:

Let There Be LEDs – Again!

We seem to be on an LED trend here; last month faux candles, this month a digital display. Okay, perhaps it's just me, but I get the feeling that with the ever-increasing variety of LED options – size, shape, and color – that there is renewed interest in LED control. I frequently find neat 7-segment displays at Tanners, and need a simple way to control them when using a BASIC Stamp. I found one and am going to share it with you.

The control of 7-segment LED displays can be tricky, and is downright impractical to do in high-level BASIC Stamp code – the best thing to do is enlist a “helper” chip. In the recent past we created our own multiplexer with the SX micro, and both Scott Edwards and I have written about the MAX7219. Well, not everybody has SX tools, and the MAX7219 is not cheap at \$11 apiece. Is there an alternative? Yes, as a matter of fact there is.

The device we'll work with this month is the Motorola MC14489 display multiplexer. It's certainly cheaper than the MAX7219 – the average online price is around four bucks. That said, it doesn't do quite as much (only five digits versus eight for the MAX7219), and it does have a couple quirks that take a bit of time to get used to. I've worked through those quirks and hope that I can steer you around my initial frustrations with this chip.

Gimme Three Pins

Connection to the MC14489 is simple, taking just three I/O pins (Clock, Data In, and Enable). The Clock and Data In lines can be shared, and the MC14489 has a Data Out connection to provide for daisy-chaining if we want to control more than five digits. Having a clock line you've probably guessed by now that we're going to use SHIFTOUT to send data. You got it.

The only other component required is a 10K resistor to control display brightness. You could actually drop a pot in place of the 10K if you want brightness control, and if you're really ambitious you could use a CdS photocell (not alone, though) to automatically adjust brightness for changes in ambient light. Let's keep things simple, shall we? A 10K resistor on the MC14489.Rx pin (8) works fine. Figure 127.1 shows the connections for using the MC14489 with up to five 7-segment, common-cathode displays. And if you want to try this circuit on the PDB, don't worry about the 470-ohm resistors that are inline with the 7-segment displays; the LEDs are very bright and work just fine with the circuit in Figure 127.1.

Now if you want to jump into the MC14489 quickly and can live with four digits, you have a ready-made option: the SLED4C Serial LED display from Reynolds Electronics. At 29 bucks it's a tad pricey, but it's nicely built and very convenient – it has four bright red digits (0.4" tall) with decimal points, a colon, and a third annunciator LED that can be used as you see fit.

Figure 127.2 shows the SLED4C and the connectors that are included. Note that I put the BS2p module in the picture to give you an idea of the module's size – it is not included with the SLED4C, so don't ask Bruce for it!

What I do like about the SLED4C is that it has mounting holes that make it easy to install into a permanent project; in fact, we did that at Parallax. One of my new colleagues, Chris Savage, was asked by our production group to make a solder-pot controller. Chris used our thermocouple kit, a rotary encoder, and the SLED4C to make the controller. At the time, he hadn't actually worked with the MC14489 and asked me if I knew anything about it. Well, I had just written an update to StampWorks that included the MC14489 so I was able to show Chris the ropes very quickly with the SLED4C. In the end, we have a nice new solder-pot controller in our shop.

What I really liked about Chris's project was the use of the rotary encoder to enter the temperature set-point. I've used a rotary encoder before, but not with a BASIC Stamp. My pal Scott wrote about using rotary encoders with the BS1 about 10 years ago (Yikes, this column has been around a while...), so we'll just adapt that strategy for our program. For details on encoder use, be sure to download Column #8 (October 1995) from Parallax or from the Nuts & Volts web site – that way we can focus on the MC14489.

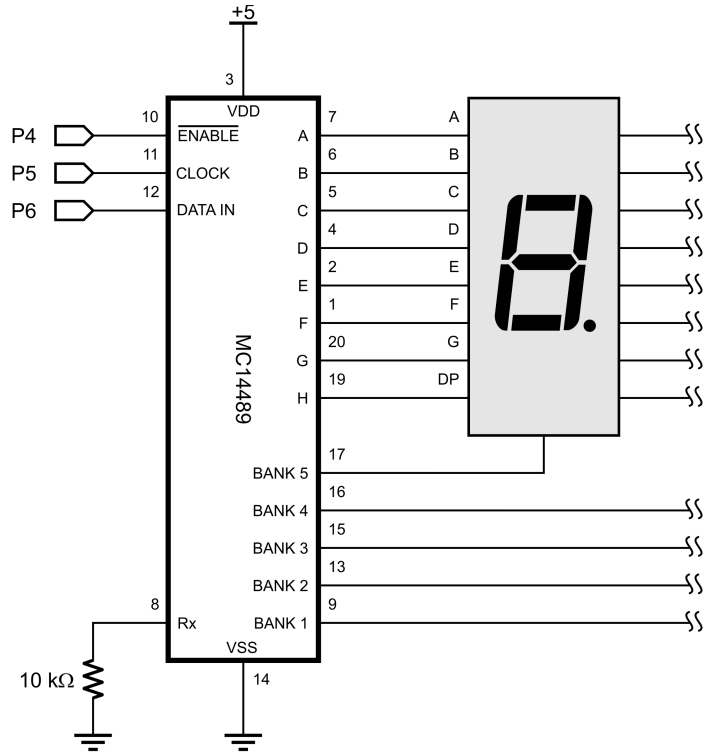


Figure 127.1: MC14489 Connections

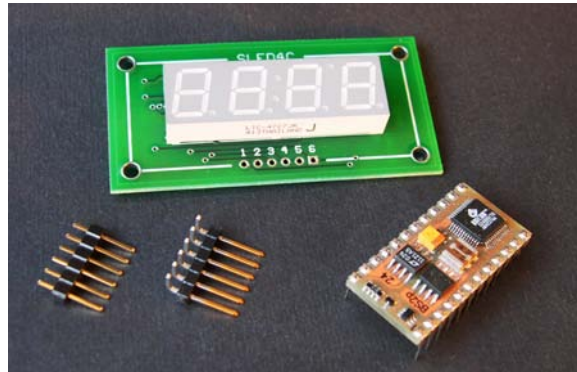


Figure 127.2: SLED4C Serial LED Display, Connectors, and BS2p24

Lock It Up

I've been asked a couple times about creating a digital lock, but wasn't really sure what to do it with – until I saw Chris's use of a rotary encoder on the solder-pot project. So, what we're going to do this month is create a digital version of the old combination lock. Figure 127.3 shows the connection for the encoder and a push-button (we'll use to select our digit), and Figure 127.4 shows the connections to the SLED4C display.

In Figure 127.2 you'll see two headers that are included with the SLED4C – this let's you decide how you're going to mount the unit. I soldered in the right-angle header so I could pop it into a breadboard, but if you're going to mount the display in a tight space, you may want to use the straight header (solder it with the long pins facing the back side of the display).

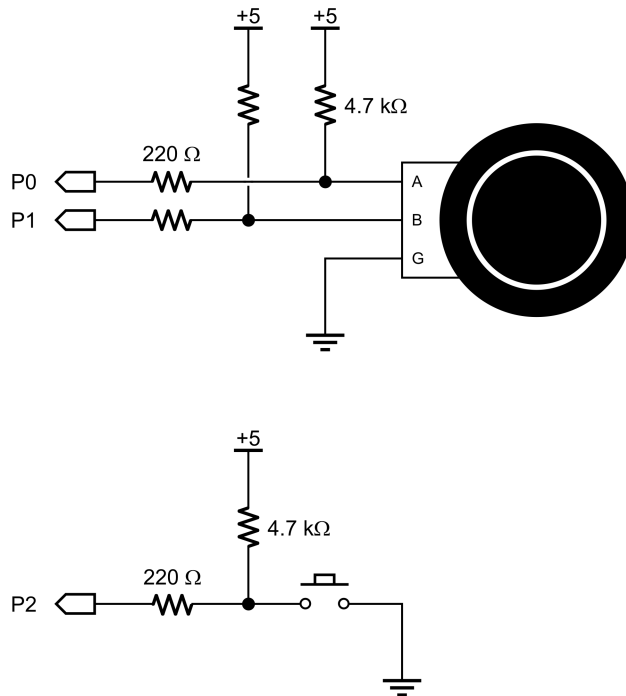


Figure 127.3: Encoder Connections

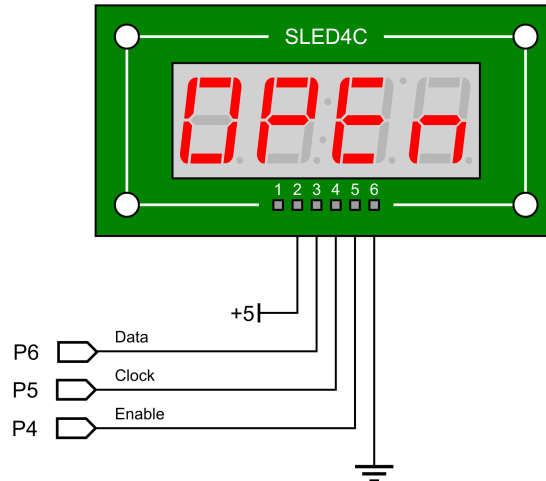


Figure 127.4: SLED4C Serial LED Connections

The MC14489

Okay, before we get into the lock code, let's talk about the MC14489. As I previously stated, it's pretty easy to use but does have a couple quirks, and the key to succeeding with the MC14489 is understanding those quirks. Communication to the MC14489 comes in the form of two packets: we either send a one-byte configuration value, or six nibbles that control the digits of the display (the sixth nibble handles display brightness and the decimal point position).

Quirk number one is that we can only send a nibble to each digit. Initially, this seems like no big deal – until we want to do some custom display that doesn't involve a standard number or letter that one might see on a 7-segment LED. Since each digit gets only four bits, when we enter what is called "no decode" mode, we only have control of segments A – D, and segments E – G get disabled in this mode.

Before I forget, let's discuss the display modes. Hex decode displays digits, 0 – F, based on the value of sent to the bank register. Special decode displays other letters and characters, things like "P" or an equal sign. No decode mode is just that – we have control of the individual outputs that control segments A – D.

Column #127: Let There Be LEDs – Again!

Quirk number two is that the display configuration is packed into one byte, so we don't have absolute control over every digit for every possible mode. Here's how the configuration byte is structured:

Bit0 – Display control; 0 = off, 1 = on
Bit1 – Bank 1 mode; 0 = Hex decode, 1 = Set by Bit6
Bit2 – Bank 2 mode; 0 = Hex decode, 1 = Set by Bit6
Bit3 – Bank 3 mode; 0 = Hex decode, 1 = Set by Bit6
Bit4 – Bank 4 mode; 0 = Hex decode, 1 = Set by Bit7
Bit5 – Bank 5 mode; 0 = Hex decode, 1 = Set by Bit7
Bit6 – 0 = No decode, 1 = Special decode (for Banks 1 – 3)
Bit7 – 0 = No decode, 1 = Special decode (for Banks 4 – 5)

Notice how banks 1 – 3 are grouped together (using Bit6), and banks 4 and 5 are grouped together (using Bit7)? Cutting through everything, it means this: any digit can be set to hex decode mode by making its control bit (1 – 5) zero; any digit within the same group can be in no decode or special decode, but within the same group you cannot mix no decode and special decode modes.

Okay, I know that's a mouthful. Once you understand that you can't mix no decode and special decode modes in the same group, it's easier to deal with the MC14489. Again, communicating with the chip is quite easy, can be handled with two subroutines.

```
Set_Cfg:
  Enable = 0
  SHIFTOUT DataIO, Clock, MSBFIRST, [config]
  Enable = 1
  RETURN
```

```
Set_Banks:
  Enable = 0
  SHIFTOUT DataIO, Clock, MSBFIRST,
    [dpCtrl\4,
     bank5\4,
     bank4\4,
     bank3\4,
     bank2\4,
     bank1\4]
  Enable = 1
  RETURN
```

As you can see, the Enable input of the MC14489 is active-low, so communication with the chip starts by bringing this pin low. Then we'll send (via SHIFTOUT) either one byte or six nibbles. The MC14489 will automatically put everything in the right place based on how many bits are sent. Note that we are using individual nibbles for bank control, so we have to use \4 in SHIFTOUT to specify four bits for each variable (the default bit count is eight). By using individual nibbles and this subroutine we get the most flexibility for our programs.

Okay, let's have a look at the lock code. Structurally, it's pretty simple. What we want to do is display the current dial value as the encoder is turned. When the button is pressed (I have an encoder with a built-in button, which is very convenient for this application), the value in the display will be compared against the combination value for the current state. If they match, the state value is incremented, otherwise the state is reset to zero (locked). When the state counter hits three, the lock is opened (an electric solenoid, for example, could be energized) and "OPEN" is displayed and flashed. At this point the program will wait for the encoder to be turned again, and when this happens the lock will reenergize.

First things first – how do we set the lock code? Keeping things easy, we can embed the combination into a DATA statement.

```
Combo          DATA    07, 25, 62
```

This works well for a couple reasons: we can read the current combo value using the state variable as an index, and being stored in EEPROM, we could update the program later to allow the combination to be set externally (without reprogramming the BASIC Stamp). Note that this program is using two digits for the dial value, so we have to keep the combination numbers under 100.

The first part of the program initializes everything. Normally, not much happens in this section but that's not the case here so we'll go through it.

```
Reset:
  LOW Solenoid
  HIGH Enable
  encOld = Encoder & %0011
  state = Locked
  GOSUB Update_Display
```

We start by engaging the lock, making the Enable pin an output and high to disable communication to the SLEC4C, and then we read the current encoder position so movement can be detected later. The lock state variable gets initialized (note the alias "Locked" that makes the program easier to understand), and finally, we update the SLED4C display.

Column #127: Let There Be LEDs – Again!

The display update is state dependent and really just a router to the code that handles the display for entering numbers or showing “OPEN.”

```
Update_Display:
  IF (state < Open) THEN
    GOTO Show_Dial
  ELSE
    GOTO Show_Open
  ENDIF
```

Let’s look at the module we’ll use most. It’s called Show_Dial and it does just that: it shows the current dial position, 0 to 99. Just to dress up the display, I centered the value in the middle of the display and surrounded it by underline characters (more on this in a bit).

```
Show_Dial:
  dpCtrl = %1000
  bank5 = ULine
  bank4 = dial DIG 1
  bank3 = dial DIG 0
  bank2 = ULine
  #IF (ShowState = 1) #THEN
    ' update state LEDs (colon dots)
    LOOKUP state, [%0000, %0001, %0011], bank1
  #ELSE
    bank1 = %0000
  #ENDIF
  display = IsOff
  GOSUB Set_Cfg
  GOSUB Set_Banks
  config = %00100111
  GOSUB Set_Cfg
  RETURN
```

The subroutine starts by updating the values of the bank control registers. In banks 5 and 2 we will show an underline character; this value comes from a program constant. The DIG operator gets used to pull the tens and ones digits from the dial value for banks 4 and 3.

There is a conditional constant in the program called ShowState that is used for testing. Its purpose is to show the current state of the lock code using the LEDs that form the colon in the SLED4C display. These LEDs can be individually controlled and are connected to bank 1 bits zero and one. If ShowState is set to one then LOOKUP gets used to enable the correct number of LEDs for the current state, otherwise they are all blanked (for normal operational mode).

Now that the register variables are updated, we will turn the display off. Why? Well, we don't have to, but what can happen is a bit of "ghosting" that just doesn't look very nice. Even with a stock BS2, the blanking and update happens so fast that we can't see the blanking period.

After the new values are sent to the bank control registers, we update the configuration register. Bit0 will always be set to enable the display. Bits 1, 2, and 5 are set to one which means they will either be in no decode or special decode mode; which depends on bits 6 and 7. In this case, both are set to zero so banks 1, 2, and 5 will be in no decode mode (remember that this mode gives us control over segments A – D only). Banks 3 and 4 are set to hex mode (0 in their configuration control bits) so we can display the dial digits.

Here's where that configuration byte quirk affected the program. Initially, I wanted the display to show "-XX-" (dash-digit-digit-dash) but when using the SLED4C the bank 1 digit needs to be set to no decode mode to enable control of the colon LEDs. This forces the use of no decode mode for banks 2 and 5 as well (so they match), and in that mode we cannot control segment G (which would make the dash we wanted). So, we live with the underline.

At this point, the display should show "_00_" – until we turn the dial. Here's the code that handles the encoder input.

```
Main:
DO WHILE (NumSelect = NotPressed)
  encNew = Encoder & %0011
  IF (encNew <> encOld) THEN
    dial = dial + 1 + (98 * (encA ^ encB)) // 100
    encOld = encNew
    GOSUB Update_Display
  ENDIF
LOOP
DO : LOOP WHILE (NumSelect = Pressed)
```

As you can see, we're simply monitoring the encoder while the button is not pressed. If it moves, the dial value gets updated and we save the new encoder position for the next check. The only tricky bit here is this:

```
dial = dial + 1 + (98 * (encA ^ encB)) // 100
```

You may remember from all of my discussions about the modulus operator (//) that if you want to increment a value between 0 and 99 with rollover, you can do this:

```
value = value + 1 // 100
```

Column #127: Let There Be LEDs – Again!

And if you want to decrement a value between 99 and 0 with roll-under you can do it like this:

```
value = value + 99 // 100
```

If you look closely at the program we've cleverly incorporated these two lines of code – how it operates is based on the encoder test. By XORing the encoder bits we can determine direction, so what this code does is multiply the direction result (0 or 1) by 98 and add that into the rest of the equation. As you can see, we'll end up adding one (clockwise) or 99 (counter-clockwise) when the encoder moves. By using modulus we're able to keep the dial value within our specified range with rollover for clockwise rotation and roll-under with counter-clockwise rotation. This is a neat trick and a good one to have in your bag.

When the button does get pressed the encoder test loop will get skipped. A small inline DO-LOOP is used to force the release of the button before moving on – this keeps the program from accidentally accepting the current dial value more than once when that is not the intention. After the button is released we can test the dial input against the current combination value.

```
Check_Digit:
  READ (Combo + state), testVal
  IF (dial = testVal) THEN
    state = state + 1 // 4
  ELSE
    state = Locked
  ENDIF
  GOSUB Update_Display
```

This is pretty easy. We start by reading the current state combination value from EEPROM, and the comparing it to the dial value. If they match, the state value gets incremented, otherwise we set the state back to Locked.

The last thing to do is check the current state to see if the lock is to be opened. First, let's look at the code that displays "OPEN" on the SLED4C:

```
Show_Open:
  dpCtrl = %1000
  bank5 = Ltr_O
  bank4 = Ltr_P
  bank3 = Ltr_E
  bank2 = Ltr_n
  bank1 = $0000
  display = IsOff
```

```

GOSUB Set_Cfg
GOSUB Set_Banks
config = %11010111
GOSUB Set_Cfg
RETURN

```

By now, this should make sense. Note that we've switched from no decode to special decode mode for banks 4 ("P"), 2 ("n"), and 1 (blank colon). And, finally, here's the code that activates the lock when we've reached the open state:

```

Check_Activation:
  IF (state = Open) THEN
    Solenoid = IsOn
    DO
      PAUSE 100
      display = display ^ 1
      GOSUB Set_Cfg
      encNew = Encoder & %0011
      LOOP UNTIL (encNew <> encOld)
      GOTO Reset
    ENDIF
  GOTO Main

```

Of course, we start by opening the physical lock with solenoid activation, and then we're going to make sure that we know the lock is open by flashing this display. This is easily accomplished by toggling the display control bit (config.BIT0), and we can do this using XOR (^). Through each iteration of the loop we check the current encoder inputs so that we can reset the lock when the encoder moves.

Well, that's all there is to it. Pretty simple, yes, but I think this code is useful and I will be adding more encoders to my projects. Just remember that the BASIC Stamp is not as fast as the SX, so you can "over spin" the encoder and get bogus values. That said, I found the program to be quite responsive with normal operation of the encoder.

Well, I hope this helps you get going with the MC14489. It may not be perfect, but it's inexpensive and easily connects to the BASIC Stamp to give us 7-segment LED control. Remember that the key is the configuration register – when you understand how it works the rest is pretty simple.

And just to help you in that regard, I made a chart that shows different configuration values for several useful display setups (see Figure 127.5). The bottom half of this chart is

Column #127: Let There Be LEDs – Again!

specifically for the SLED4C that we used in this project, hence the shaded bank 1 box, and you'll note that hex mode is not ever used in this position.

Config.7		Config.6				
5	4	3	2	1		
H	H	H	H	H	%xx000001	e.g., "72562", "A2b4E"
H	H	H	S	H	%x1000101	e.g., "123°F"
S	H	H	S	H	%11100101	e.g., "-25°C"
S	H	S	S	S	%11101111	e.g., "HELP "
N	H	H	H	N	%00100011	e.g., " OFF "
S	H	S	S	S	%11101111	e.g., " On "
H	S	H	S	S	%11010111	e.g., "OPEn "

Figure 127.5: Configuration Register Options

Happy Thanksgiving to you and yours, and until next time, Happy Stamping too!

Additional Resources:

Reynolds Electronics
www.rentron.com

```

' =====
'
' File..... SLED4C.BS2
' Purpose....
' Author..... Jon Williams -- Parallax, Inc.
' E-mail..... jwilliams@parallax.com
' Started....
' Updated.... 15 SEP 2005
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =====

' ----[ Program Description ]-----
'
' Test program the Rentrion SLED4C display
' -- See www.rentron.com/Products/SLED4C.htm

' ----[ Revision History ]-----

' ----[ I/O Definitions ]-----

Enable          PIN    4           ' SLED4C.5
Clock           PIN    5           ' SLED4C.4
DataIO          PIN    6           ' SLED4C.3

' ----[ Constants ]-----

IsOn            CON    1           ' for config.BIT0
IsOff           CON    0

LedsBright     CON    1           ' for dPnts.BIT3
LedsDim        CON    0

' MC14489 characters           ' decode mode

Ltr_A          CON    $0A        ' Hex
Ltr_b          CON    $0B        ' Hex
Ltr_C          CON    $0C        ' Hex
Ltr_c2         CON    $01        ' Special
Ltr_d          CON    $0D        ' Hex
Ltr_E          CON    $0E        ' Hex
Ltr_F          CON    $0F        ' Hex
Ltr_H          CON    $02        ' Special
Ltr_h2        CON    $03        ' Special

```

Column #127: Let There Be LEDs – Again!

```

Ltr_J          CON      $04          ' Special
Ltr_L          CON      $05          ' Special
Ltr_n          CON      $06          ' Special
Ltr_O          CON      $00          ' Hex
Ltr_o2         CON      $07          ' Special
Ltr_P          CON      $08          ' Special
Ltr_r          CON      $09          ' Special
Ltr_U          CON      $0A          ' Special
Ltr_u2         CON      $0B          ' Special
Ltr_Y          CON      $0C          ' Special

Blank          CON      $00          ' Special or No
ULine          CON      $08          ' No
Dash           CON      $0D          ' special
Equal          CON      $0E          ' Special
DegSym         CON      $0F          ' Special

Colon          CON      %011         ' No (SLED4C)
DegSym2        CON      %100         ' No (SLED4C)

' -----[ Variables ]-----

config         VAR      Byte          ' configuration register
display        VAR      config.BIT0  ' 0 = off, 1 = on

dPnts         VAR      Nib           ' decimal point control
segs5         VAR      Nib           ' segs - bank 5 (left)
segs4         VAR      Nib           '
segs3         VAR      Nib           '
segs2         VAR      Nib           ' segs - bank 2 (right)
segs1         VAR      Nib           ' used for colon/deg pnt

value         VAR      Word          '
valMode       VAR      Nib           ' 1, 0.1, 0.01, 0.001
temp          VAR      Word          '
tMode         VAR      Bit           ' 0 = C, 1 = F
hrs           VAR      Byte          '
mns           VAR      Byte          '

' -----[ EEPROM Data ]-----

' -----[ Initialization ]-----

Reset:
HIGH Enable          ' deselect SLED4C

' -----[ Program Code ]-----

```

```

Main:
  FOR valMode = 0 TO 3
    RANDOM value
    GOSUB Show_Pos_Value
    PAUSE 2000
  NEXT

  GOTO Main

' -----[ Subroutines ]-----

' Update MC14489/SLED4C configuration register

Set_Cfg:
  Enable = 0                                ' enable SLED4C
  SHIFTOUT DataIO, Clock, MSBFIRST, [config] ' send config register
  Enable = 1                                ' disable SLED4C
  RETURN

' -----

Set_SLED4C:
  Enable = 0                                ' enable SLED4C
  SHIFTOUT DataIO, Clock, MSBFIRST,         ' update LED registers
    [dPnts\4, segs5\4, segs3\4,
     segs3\4, segs2\4, segs1\4]
  Enable = 1                                ' disable SLED4C
  RETURN

' -----

Show_Pos_Value:

  IF (valMode < 4) THEN
    LOOKUP valMode, [%1000, %1011, %1100, %1101], dPnts
  ELSE
    dPnts = %1000
  ENDIF

  segs5 = value DIG 4
  segs4 = value DIG 3
  segs3 = value DIG 2
  segs2 = value DIG 1
  segs1 = Blank

  display = IsOff
  GOSUB Set_Cfg
  GOSUB Set_SLED4C

```

Column #127: Let There Be LEDs – Again!

```
config = %00000011
GOSUB Set_Cfg

RETURN

' -----

Show_Temp:

RETURN

' -----

Show_Time:

RETURN
```

```

' =====
'
' File..... Digi-Lock.BS2
' Purpose.... Digital combination lock
' Author..... Jon Williams -- Parallax, Inc.
' E-mail..... jwilliams@parallax.com
' Started....
' Updated.... 15 SEP 2005
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
' =====

' -----[ Program Description ]-----
'
' Digital lock using an encoder and the Rentron SLED4C display
' -- See www.rentron.com/Products/SLED4C.htm

' -----[ Revision History ]-----

' -----[ I/O Definitions ]-----

Encoder          VAR      INA          ' encoder bits (0 & 1)
NumSelect        PIN      2            ' button input
Solenoid         PIN      3            ' latch control

Enable           PIN      4            ' SLED4C.5 / MC14489.10
Clock            PIN      5            ' SLED4C.4 / MC14489.11
DataIO           PIN      6            ' SLED4C.3 / MC14489.12

' -----[ Constants ]-----

Pressed          CON      0            ' for active-low button
NotPressed       CON      1

IsOn             CON      1            ' display control
IsOff            CON      0

Locked           CON      0            ' lock states
Has1             CON      1
Has2             CON      2
Open             CON      3

' MC14489 characters          ' decode mode

Ltr_A           CON      $0A          ' Hex

```

Column #127: Let There Be LEDs – Again!

```

Ltr_b          CON    $0B          ' Hex
Ltr_C          CON    $0C          ' Hex
Ltr_c2         CON    $01          ' Special
Ltr_d          CON    $0D          ' Hex
Ltr_E          CON    $0E          ' Hex
Ltr_F          CON    $0F          ' Hex
Ltr_H          CON    $02          ' Special
Ltr_h2         CON    $03          ' Special
Ltr_J          CON    $04          ' Special
Ltr_L          CON    $05          ' Special
Ltr_n          CON    $06          ' Special
Ltr_O          CON    $00          ' Hex
Ltr_o2         CON    $07          ' Special
Ltr_P          CON    $08          ' Special
Ltr_r          CON    $09          ' Special
Ltr_U          CON    $0A          ' Special
Ltr_u2         CON    $0B          ' Special
Ltr_Y          CON    $0C          ' Special

Blank          CON    $00          ' Special or No
ULine          CON    $08          ' No
Dash           CON    $0D          ' special
Equal          CON    $0E          ' Special
DegSym         CON    $0F          ' Special

Colon          CON    %011         ' No (SLED4C)
DegSym2        CON    %100         ' No (SLED4C)

#DEFINE ShowState = 1          ' 1 = show lock state

' -----[ Variables ]-----

state          VAR    Nib          ' lock state

config         VAR    Byte         ' configuration register
dpCtrl        VAR    Nib          ' decimal point control
bank5          VAR    Nib
bank4          VAR    Nib
bank3          VAR    Nib
bank2          VAR    Nib
bank1          VAR    Nib          ' used for colon/deg pnt

display        VAR    config.BIT0  ' 0 = off, 1 = on
brightness     VAR    dpCtrl.BIT3  ' 1 = bright, 0 = dim

encOld         VAR    Nib          ' encoder readings
encNew         VAR    Nib
encA           VAR    encOld.BIT0
encB           VAR    encNew.BIT1

```

```

testVal      VAR      Byte      ' for combination test
dial         VAR      Byte

' -----[ EEPROM Data ]-----

Combo        DATA      07, 25, 62

' -----[ Initialization ]-----

Reset:
  LOW Solenoid      ' lock it up
  HIGH Enable      ' deselect SLED4C
  encOld = Encoder & %0011
  state = Locked
  dpCtrl = %1000   ' on bright, no DPs
  GOSUB Update_Display

' -----[ Program Code ]-----

Main:
  DO WHILE (NumSelect = NotPressed)
    encNew = Encoder & %0011      ' read encoder inputs
    IF (encNew <> encOld) THEN    ' changed?
      dial = dial + 1 + (98 * (encA ^ encB)) // 100
      encOld = encNew            ' save current scan
      GOSUB Update_Display      ' display new spinner value
    ENDIF
  LOOP

  DO : LOOP WHILE (NumSelect = Pressed)      ' force release

Check_Digit:
  READ (Combo + state), testVal      ' get current combo num
  IF (dial = testVal) THEN           ' test against spinner
    state = state + 1 // 4           ' update state if match
  ELSE
    state = Locked                   ' otherwise reset
  ENDIF
  GOSUB Update_Display

Check_Activation:
  IF (state = Open) THEN
    Solenoid = IsOn                 ' activate solenoid
    DO
      PAUSE 100                      ' flash timing
      display = display ^ 1          ' toggle display bit
    GOSUB Set_Cfg
  
```

Column #127: Let There Be LEDs – Again!

```
    encNew = Encoder & %0011           ' scan encoder
    LOOP UNTIL (encNew <> encOld)      ' wait until it moves
    GOTO Reset                          ' lock it up
ENDIF

GOTO Main

' -----[ Subroutines ]-----
' Update MC14489/SLED4C configuration register

Set_Cfg:
    Enable = 0                          ' enable SLED4C
    SHIFTOUT DataIO, Clock, MSBFIRST, [config] ' send config register
    Enable = 1                          ' disable SLED4C
    RETURN

' Update MC14489/SLED4C bank values

Set_Banks:
    Enable = 0                          ' enable SLED4C
    SHIFTOUT DataIO, Clock, MSBFIRST,    ' send control nibbles
        [dpCtrl\4,
        bank5\4,
        bank4\4,
        bank3\4,
        bank2\4,
        bank1\4]
    Enable = 1                          ' disable SLED4C
    RETURN

' Shows current spinner value or "OPEN"

Update_Display:
    IF (state < Open) THEN
        GOTO Show_Dial
    ELSE
        GOTO Show_Open
    ENDIF

' Show current dial value "_XX_"

Show_Dial:
    dpCtrl = %1000                      ' bright, no DPs
    bank5 = ULine
    bank4 = dial DIG 1                  ' tens
    bank3 = dial DIG 0                  ' ones
```

```

bank2 = ULine

#IF (ShowState = 1) #THEN
  ' update state LEDs (colon dots)
  LOOKUP state, [%0000, %0001, %0011], bank1
#ELSE
  bank1 = %0000
#ENDIF

display = IsOff                ' turn off display
GOSUB Set_Cfg
GOSUB Set_Banks
config = %00100111            ' show "_XX_"
GOSUB Set_Cfg
RETURN

' Show "OPEn"

Show_Open:
dpCtrl = %1000                ' bright, no DPs
bank5 = Ltr_O
bank4 = Ltr_P
bank3 = Ltr_E
bank2 = Ltr_n
bank1 = $0000                  ' clear extra LEDs
display = IsOff                ' turn off display
GOSUB Set_Cfg
GOSUB Set_Banks
config = %11010111            ' show "OPEn"
GOSUB Set_Cfg
RETURN

```

Column #127: Let There Be LEDs – Again!