Column #109 May 2004 by Jon Williams:

Expansion Made Easy

One of the great things about my job – aside from the fact that I get to work with BASIC Stamps all day long (and get paid for it!) – is the customer contact I have. Almost all are very friendly and find what I get to do here with Nuts & Volts useful, and sometimes even a bit entertaining. Often, I get messages that are a cry for help, and I always enjoy helping when I can. And, from time-to-time, a customer will alert me to a part that I hadn't previously worked with. That's always an adventure, and sometimes those adventures result in a real gem of a find.

Case in point: A couple months ago I got a note from a customer who was trying to connect his BS2p to a new part from Microchip called the MCP23016. As the part wasn't yet in production, he sent me one of his samples and it turned out that he had made a simple coding error – I was able to get the BS2p to control the MCP23016 without breaking a sweat.

Let me just say that this part rocks! Oh ... I'll bet by now that you're wondering what it is. The MCP23016 is a 16-bit (two ports) IO expander; you can think of it as a much better version of the PCF8574. Why is it better? Well, for one, there's none of that quasi-bidirectional silliness of the PCF8574 (which is a full-on pain in the backside...), its ports

behave like those on a microcontroller, each having a DDR (data direction register) to specify what an output port does. And (this is the best part) ... it can sink and source 25 mA per pin. Honestly, the MCP23016 makes the PCF8574 look like a schoolyard sissy.

Not long after I worked with the MCP23016 another customer contacted me about creating an LCD interface with the PCF8574. I gave him some guidance, but I think you know where I stand with that device. Still, the idea is a good one: why not create a 2-wire LCD interface with a two-dollar part? So that's what I did – using the MCP23016, of course. And that's what I'm going to share with you here.

Easy IO – Easy LCD Terminal

What I like most about the MCP23016 is that it's easy to deal with; we simply set the DDRs for the ports as we need, then write to them or read from them. Nothing could be simpler. As an added bonus, each port (8-bits) has a register that sets the polarity of the inputs. I like this because it lets us read active-low inputs to the MCP23016 as 1 (high) when they are active.

So let's get to it. What we're going to do this month is create an LCD terminal with the MCP23016, an LED, and a four active-low buttons. The demo program will test all of the features of the terminal; later we can strip out the demo stuff and use the subroutines in other applications.

Figure 109.1 shows the schematic of the MCP23016 connections to the LCD. Notice that we're using all eight bits of GP0 (port 0) to connect to the LCD data buss. This will simplify the code a bit versus the 4-bit interface that we typically use. We need three bits from GP1 (port 1) for LCD control, and the other bits are used to control the LED (an output), and the four buttons (inputs).

The MCP23016 has an internal clock circuit that is driven by an external resistor/capacitor combination. This clock determines how quickly the MCP23016 can respond to changes on its input pins to generate an interrupt output. We're not using that here, but we still need the RC circuit for the MPC23016 to run. The values shown are recommended by Microchip. Just be aware that the clock speed affects the MCP23016's stand-by current consumption. Be sure to download the MCP23016 docs from Microchip for details on clock RC values and using the interrupt output and capture registers.

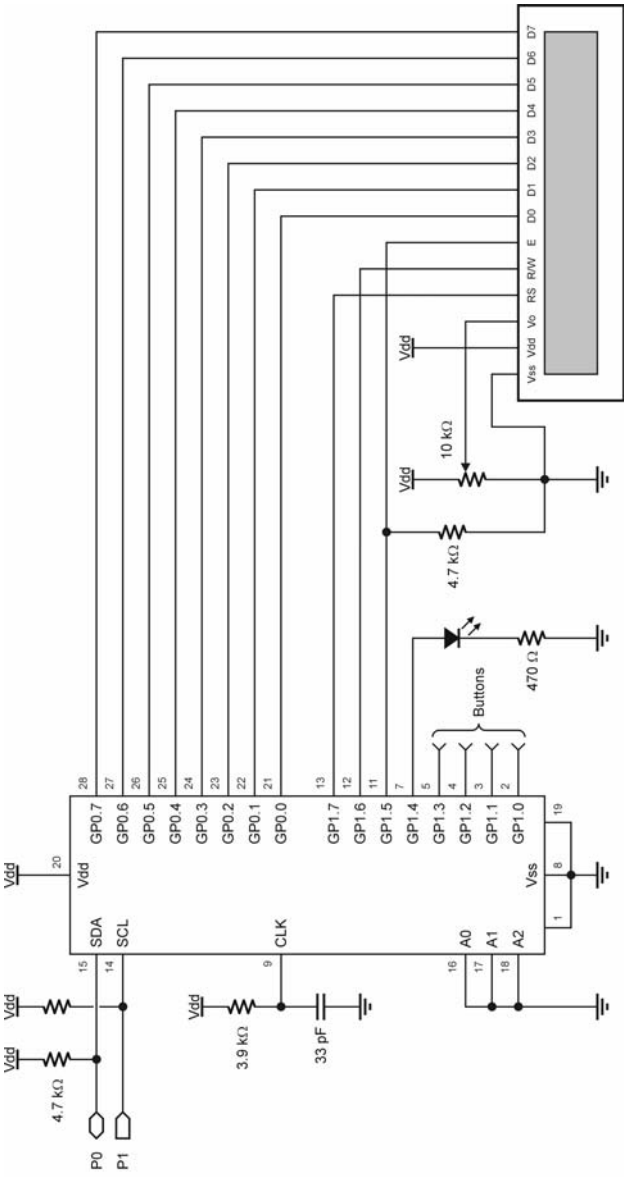

Figure 109.1: MCP23016 LCD Terminal Schematic

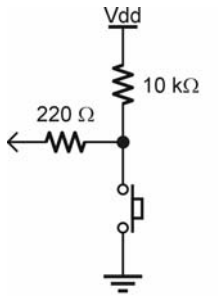

Figure 109.2: Standard Active-Low Button Circuit

Okay, the connections are simple enough, let's jump into the code. As with any IO port, we have to initialize the IO direction bits inputs or outputs.

```
Setup:
  PAUSE 500
  I2COUT SDA, Wr23016, IODIR0, [%00000000]
  I2COUT SDA, Wr23016, IODIR1, [%00001111]
  I2COUT SDA, Wr23016, IPOL1, [%00001111]
```

We start with a PAUSE so that the LCD and the MCP23016 can get through their internal reset operations. The next step is to set the pin directions. Let me point out a difference here between the MCP23016 and the BASIC Stamp. In the MCP23016, an output bit is specified with zero (0 looks like the letter O for output), and an input is specified with one (1 looks like I for input).

This code is written to be obvious, and after you get used to the device you can take advantage of automatic address indexing on writes and reads by writing to both IODIR registers with one line of code:

```
  I2COUT SDA, Wr23016, IODIR0, [%00000000, %00001111]
```

The final step in the setup process is to set the polarity of the input bits on GP1.0 – GP1.3. When writing to a polarity register, a one bit inverts the input. Since we are using active-low button circuits, we do want them inverted, hence the ones in bits zero through three.

Now that the ports on the MCP23016 are setup, it's time to initialize the LCD. For those of you that have worked with LCDs previously, this code will look quite familiar.

```
LCD_Init:
  lcdIO = %00110000
  GOSUB LCD_Command
  PAUSE 5
  GOSUB LCD_Command
  GOSUB LCD_Command
  lcdIO = %00111000
  GOSUB LCD_Command
  lcdIO = %00001100
  GOSUB LCD_Command
  lcdIO = %00000110
  GOSUB LCD_Command
```

This section follows the standard Hitachi initialization sequence to put the display into 8-bit mode, using multiple lines and the 5x7 font. It turns the underline cursor off and causes the cursor address pointer to automatically increment after a write or a read. Of course, we're not writing directly to the LCD, we're doing it through the MCP23016 – so let's have a look at how that's done.

```
LCD_Command:
  I2CIN SDA, Rd23016, GP1, [lcdCtrl]
  lcdRS = 0
  I2COUT SDA, Wr23016, GP1, [lcdCtrl]

LCD_Write:
  I2COUT SDA, Wr23016, GP0, [lcdIO]
  I2CIN SDA, Rd23016, GP1, [lcdCtrl]
  lcdE = 1
  I2COUT SDA, Wr23016, GP1, [lcdCtrl]
  lcdE = 0
  lcdRS = 1
  I2COUT SDA, Wr23016, GP1, [lcdCtrl]
  RETURN
```

You'll recall from our previous work with LCDs that we can a write can be either a command or a data byte for the display. The LCD disguises between a command and data by the state of the RS line; when we set the RS line low, the byte written is interpreted as a command, and when RS is high, the byte is interpreted as data to be written to the current cursor position.

What you can see, then, is that LCD_Command is just an entry point to the LCD_Write subroutine that takes care of setting the RS line low. In order to make the code work in other applications, we won't assume anything about the current state of RS, we'll read it back, modify it (make it 0), then send it to the MCP23016. This must look like a lot of work, especially compared to direct bit access we have on BASIC Stamp IO pins. The truth of the matter is that PBASIC shelters us from this kind of detail, underneath the hood of the BASIC Stamp the same kind of process is happening when we manipulate a single pin.

With RS setup properly, we can write the command that was passed in the variable lcdIO. The command is written to the LCD buss pins, and then the LCD E pin is "blipped" high momentarily. The process is the same as with manipulating the RS pin: we read the current state, set it the way we want, and then send it back. Notice that when we take the E pin back low we return the RS line to a high (data mode). This lets us enter LCD_Write with RS in the proper state.

One of the reasons I like LCDs so much is the ability to have custom characters. Ten years ago I created this little animation that is a chomping mouth – and I've brought it back as part of the LCD interface test. The codes for custom characters are stored in DATA statements, and then downloaded with a simple loop:

```
Download_Chars:
  lcdIO = LcdCGRam
  GOSUB LCD_Command
  FOR addr = CC0 TO (CC2 + 7)
    READ addr, lcdIO
    GOSUB LCD_Write
  NEXT
```

The process is straightforward: we is set the cursor position to the Character Generator RAM, and then write the data bytes that build those characters.

Okay, we've got what we need to get information to the LCD, so let's give it a try.

```
Main:
  lcdIO = LcdCls
  GOSUB LCD_Command
  addr = Msg1
  GOSUB Put_String
  PAUSE 2000
```

The top of our demo starts by clearing the LCD. This has a double-purpose in that it also returns the LCD cursor to the Home position. Then we write a string to the display with

another subroutine. Like the custom character data, strings are stored in DATA statements so they can be re-used without consuming additional program space. Here's the code that writes a string to the LCD:

```
Put_String:
  DO
    READ addr, lcdIO
    addr = addr + 1
    IF (lcdIO = 0) THEN EXIT
    GOSUB LCD_Write
   LOOP
  RETURN
```

The string display works by reading characters from a DATA statement. The start of the string (and current character in the loop) is pointed to by the variable addr. After reading a character the address gets updated, then is tested for zero. If it is zero, then we terminate the loop and return to the caller. If not, the character is sent to the LCD. You may be wondering why we update the address pointer right after the READ, when the value could be zero. Well, by doing this, we can write two strings back-to-back without having to set the address of the second string. The only condition is that the strings must be stored in DATA statements in the order desired, otherwise setting the starting address for subsequent strings is required.

With data in the LCD, let's see if we can read it back. This code is a little more involved, but not really difficult. What we have to do is set the GP0 pins as inputs, and then put the LCD in write mode by making the RW line high. When we do that and set the E pin high, the LCD will output the data at the cursor location to its buss pins. At that point we read the byte from the LCD, set E and RW to their normal states, and make the GP0 pins outputs.

```
LCD_Read:
  I2COUT SDA, Wr23016, IODIR0, [%11111111]
  I2CIN SDA, Rd23016, GP1, [lcdCtrl]
  lcdRW = 1
  I2COUT SDA, Wr23016, GP1, [lcdCtrl]
  lcdE = 1
  I2COUT SDA, Wr23016, GP1, [lcdCtrl]
  I2CIN SDA, Rd23016, GP0, [lcdIO]
  lcdE = 0
  lcdRW = 0
  I2COUT SDA, Wr23016, GP1, [lcdCtrl]
  I2COUT SDA, Wr23016, IODIR0, [%00000000]
  RETURN
```

A test loop will read back the LCD characters and display them in the Debug window. Remember that the LCD cursor is set to auto-increment after any write or read, so we have to set it to the Home position (line 1, column 0) before starting to read.

```
Read_Demo:
  DEBUG CLS, "Reading from LCD: "
  PAUSE 500
  lcdIO = LcdHome
  GOSUB LCD_Command
  FOR column = 0 TO 15
    GOSUB LCD_Read
    DEBUG lcdIO
  NEXT
```

The final test of the LCD is the use of the custom characters that we downloaded during the setup process.

```
Animation:
  FOR column = 0 TO LastCol
    FOR idx = 0 TO 4
      lcdIO = LcdLine1 + column
      GOSUB LCD_Command
      LOOKUP idx, [2, 1, 0, 1, " "], lcdIO
      GOSUB LCD_Write
      PAUSE 50
    NEXT
  NEXT
```

The animation process requires two loops: the outer loop is used to set the cursor position, and the inner loop sets the character to be displayed. Note that we have to reset the cursor position before each write because the LCD has been initialized to auto-increment the cursor. There's really no harm since we need a bit of a delay between animation "cells" anyway – the time required for the write helps in that regard.

A LOOKUP table is used to set the current animation character; in this case we're using the custom character values zero, one, and two. The final character in the sequence is a space and what we end up with is a "mouth" chomping its way across the LCD and removes our initial message.

Well, the LCD certainly seems to be working. Since our design is for a terminal with button inputs and an auxiliary LED output, let's go ahead and test them.

```
Button_Demo:
  lcdIO = LcdCls
  GOSUB LCD_Command
  addr = Msg2
  GOSUB Put_String

Show_Buttons:
  GOSUB Get_Buttons
  lcdIO = LcdLine2
  GOSUB LCD_Command
  FOR idx = 3 TO 0
    LOOKUP btns.LOWBIT(idx), ["-*"], lcdIO
    GOSUB LCD_Write
  NEXT
```

After clearing the LCD and writing "BUTTONS" on the first line, we'll put the program into a loop that reads and displays the button status. Reading the buttons is simply a matter of reading port GP1 from the MCP23016 and grabbing the lower four bits.

```
Get_Buttons:
  I2CIN SDA, Rd23016, GP1, [lcdCtrl]
  btns = lcdCtrl
  RETURN
```

Remember that we don't have to invert the active-low button inputs as the MCP23016 has been setup to do that for us (I love this feature). We could save a bit of variable space by aliasing btns to the NIB0 of lcdIO, but I decided not to so that things don't get mixed up as lcdIO is used in so many other places.

Back to the button demo code ... now that we have the current status we can display each on the second line using LOOKUP again. Here we'll use LOOKUP to select the character; as it stands, a dash means the button is not pressed, an asterisk when a button is pressed. I decided to do this because 0 and 1 are boring – but if that's what you want to need for one of your apps, you can change the LOOKUP line to this:

```
    lcdIO = "0" + btns.LOWBIT(idx)
```

Alright, we're almost home – the last thing we need to test is the LED. Just for fun, let's make it light when all the buttons are pressed.

```
Update_LED:
  IF (btns = %1111) THEN
    GOSUB LED_On
  ELSE
    GOSUB LED_Off
  ENDIF
```

And – finally – a couple subroutines to update the LED as required by the demo code:

```
LED_On:
  I2CIN SDA, Rd23016, GP1, [lcdCtrl]
  IF (lcdLED = Is_Off) THEN
    lcdLED = Is_On
    I2COUT SDA, Wr23016, GP1, [lcdCtrl]
  ENDIF
  RETURN

LED_Off:
  I2CIN SDA, Rd23016, GP1, [lcdCtrl]
  IF (lcdLED = Is_On) THEN
    lcdLED = Is_Off
    I2COUT SDA, Wr23016, GP1, [lcdCtrl]
  ENDIF
  RETURN

Set_LED:
  I2CIN SDA, Rd23016, GP1, [lcdCtrl]
  IF (lcdLED <> ledStatus) THEN
    lcdLED = ledStatus
    I2COUT SDA, Wr23016, GP1, [lcdCtrl]
  ENDIF
  RETURN
```

All of these routines are quite simple; we read the status of GP1, check the LED control bit, then update it if required and send the port data back. While our demo doesn't actually use the Set_LED routine, it's included because it will be useful when we want the terminal LED to follow a status bit from elsewhere in out application (that we've aliased as ledStatus). Do note that none of these routines write the LED status bit to the MCP23016 unless a change is actually required.

Okay, we're done. How about that for a simple, yet abundantly useful project? I think so. As I suggested some time back, I've become a very big fan of the I2C buss and the MCP23016 is a great part to use with it.

A couple final notes: Yes, you can control the MCP23016 with the BS2, BS2e, and BS2sx. You'll need to use manual I2C code since those BASIC Stamps don't have the I2CIN and I2COUT instructions. We did that back in the May 2002 (you can find that article online as a PDF at the Nuts & Volts web site). And, finally (I promise), the MCP23016 uses the same device address as the PCF8574A – so you can't mixed them on the same SDA buss pin (but you can mix the MCP23016 with the PCF8574AP).

Have fun with the MCP23016, it's a great part – and until next time, Happy Stamping.

```
' =============================================================================
'
'   File....... 2-Wire LCD.BPE
'   Purpose.... 2-Wire LCD interface using the Microchip MCP23016
'   Author..... Jon Williams
'   E-mail..... jwilliams@parallax.com
'   Started....
'   Updated.... 13 MAR 2004
'
'   {$STAMP BS2pe}
'   {$PBASIC 2.5}
'
' =============================================================================


' -----[ Program Description ]---------------------------------------------
'
' This program uses the Microchip MCP23016 IO port expander to create a
' 2-wire (I2C) LCD terminal.  The code can be used with any 1-, 2-, or
' 4-line character LCD (note that this program is configured for a multi-
' line LCD) with the Hitachi HD44780A driver (or equivalent).  The terminal
' includes a status LED and four input buttons.
'
' Connections:
'
'   GP0.0 -> GP0.7      LCD Buss (D0 -> D7)
'   GP1.0 -> GP1.3      active-low pushbuttons
'   GP1.4               active high LED (via 470 ohms to Vss)
'   GP1.5               LCD Enable
'   GP1.6               LCD R/W
'   GP1.7               LCD RS
'   Clock               3.9K to Vdd; 33 pF to Vss


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


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

SDA             PIN     0                       ' I2C data pin
SCL             PIN     SDA + 1                 ' I2C clock pin


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

' MCP23016

DevType         CON     %0100 << 4              ' Device type
DevAddr         CON     %000 << 1               ' address = %000 -> %111
Wr23016         CON     DevType | DevAddr       ' write to MCP23016
```

```
Rd23016         CON     Wr23016 | 1             ' read from MCP23016

GP0             CON     $00                     ' register addresses
GP1             CON     $01
OLAT0           CON     $02
OLAT1           CON     $03
IPOL0           CON     $04
IPOL1           CON     $05
IODIR0          CON     $06
IODIR1          CON     $07
INTCAP0         CON     $08
INTCAP1         CON     $09
IOCON0          CON     $0A
IOCON1          CON     $0B

' LCD

LcdCls          CON     $01                     ' clear the LCD
LcdHome         CON     $02                     ' move cursor home
LcdCrsrL        CON     $10                     ' move cursor left
LcdCrsrR        CON     $14                     ' move cursor right
LcdDispL        CON     $18                     ' shift chars left
LcdDispR        CON     $1C                     ' shift chars right

LcdDDRam        CON     $80                     ' Display Data RAM control
LcdCGRam        CON     $40                     ' Character Generator RAM
LcdLine1        CON     $80                     ' DDRAM address of line 1
LcdLine2        CON     $C0                     ' DDRAM address of line 2

LastCol         CON     15                      ' for 16-column LCD

Is_On           CON     1
Is_Off          CON     0


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

lcdIO           VAR     Byte                    ' data to/from LCD
lcdCtrl         VAR     Byte                    ' LCD control byte
lcdLED          VAR     lcdCtrl.BIT4            ' LED output
lcdE            VAR     lcdCtrl.BIT5            ' enable
lcdRW           VAR     lcdCtrl.BIT6            ' read or write
lcdRS           VAR     lcdCtrl.BIT7            ' data or command

addr            VAR     Word                    ' ee address for strings
column          VAR     Byte
idx             VAR     Nib

btns            VAR     Nib
btn0            VAR     btns.BIT0
btn1            VAR     btns.BIT1
```

```
btn2            VAR     btns.BIT2
btn3            VAR     btns.BIT3

ledStatus       VAR     Bit


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

CC0             DATA    $0E, $1F, $1C, $18, $1C, $1F, $0E, $00
CC1             DATA    $0E, $1F, $1F, $18, $1F, $1F, $0E, $00
CC2             DATA    $0E, $1F, $1F, $1F, $1F, $1F, $0E, $00

Msg1            DATA    "2-WIRE TERMINAL", 0
Msg2            DATA    "BUTTONS", 0


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

Validate:
  #IF ($STAMP < BS2P) #THEN                     ' check BASIC Stamp type
    #ERROR "Requires BS2p/BS2pe"                ' stop if not BS2p/BS2pe
  #ENDIF

Setup:
  PAUSE 500
  I2COUT SDA, Wr23016, IODIR0, [%00000000]      ' GP0 pins are outputs
  I2COUT SDA, Wr23016, IODIR1, [%00001111]      ' GP1.0 - GP1.3 are inputs
  I2COUT SDA, Wr23016, IPOL1, [%00001111]       ' invert button inputs

Lcd_Init:
  lcdIO = %00110000                             ' wake-up
  GOSUB LCD_Command
  PAUSE 5
  GOSUB LCD_Command
  GOSUB LCD_Command
  lcdIO = %00111000                             ' 8-bit, 2-line, 5x7 font
  GOSUB LCD_Command
  lcdIO = %00001100                             ' no crsr, no blink
  GOSUB LCD_Command
  lcdIO = %00000110                             ' inc crsr, no disp shift
  GOSUB LCD_Command

Download_Chars:
  lcdIO = LcdCGRam                              ' point to CCGRAM
  GOSUB LCD_Command
  FOR addr = CC0 TO (CC2 + 7)                   ' write custom chars
    READ addr, lcdIO
    GOSUB LCD_Write
  NEXT
```

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

Main:
  lcdIO = LcdCls                                ' clear the LCD
  GOSUB LCD_Command
  addr = Msg1                                   ' write "2-WIRE TERMINAL"
  GOSUB Put_String
  PAUSE 2000

Read_Demo:
  DEBUG CLS, "Reading from LCD: "
  PAUSE 500
  lcdIO = LcdHome
  GOSUB LCD_Command                             ' move cursor to L1, C0
  FOR column = 0 TO 15
    GOSUB LCD_Read                              ' read byte at cursor
    DEBUG lcdIO                                 ' show in Debug window
  NEXT

Animation:
  FOR column = 0 TO LastCol                     ' loop through colums
    FOR idx = 0 TO 4                            ' loop through "cells"
      lcdIO = LcdLine1 + column
      GOSUB LCD_Command                         ' move to current position
      LOOKUP idx, [2, 1, 0, 1, " "], lcdIO      ' get "cell"
      GOSUB LCD_Write                           ' write it
      PAUSE 50
    NEXT
  NEXT

Button_Demo:
  lcdIO = LcdCls                                ' clear LCD
  GOSUB LCD_Command
  addr = Msg2                                   ' write "BUTTONS"
  GOSUB Put_String

Show_Buttons:
  GOSUB Get_Buttons                             ' get buttons status
  lcdIO = LcdLine2                              ' move to line 2, col 1
  GOSUB LCD_Command
  FOR idx = 3 TO 0                              ' loop through buttons
    LOOKUP btns.LOWBIT(idx), ["-*"], lcdIO
    GOSUB LCD_Write
  NEXT

Update_LED:
  IF (btns = %1111) THEN                        ' all buttons pressed?
    GOSUB LED_On                                ' yes, LED on
  ELSE
    GOSUB LED_Off                               ' no, LED off
  ENDIF
```

```
  GOTO Show_Buttons
  END


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

' Send command to LCD
' -- put command byte in 'lcdIO'

LCD_Command:                                    ' write command to LCD
  I2CIN SDA, Rd23016, GP1, [lcdCtrl]            ' get control bits
  lcdRS = 0                                     ' set to command mode
  I2COUT SDA, Wr23016, GP1, [lcdCtrl]           ' update control bits


' Write character to current cursor position
' -- but byte to write in 'lcdIO'

LCD_Write:
  I2COUT SDA, Wr23016, GP0, [lcdIO]             ' put byte on LCD buss
  I2CIN SDA, Rd23016, GP1, [lcdCtrl]            ' get control bits
  lcdE = 1
  I2COUT SDA, Wr23016, GP1, [lcdCtrl]           ' blip E line
  lcdE = 0
  lcdRS = 1                                     ' return to char mode
  I2COUT SDA, Wr23016, GP1, [lcdCtrl]
  RETURN


' Reads byte from LCD
' -- set cursor position before calling
' -- returns byte read in 'lcdIO'

LCD_Read:
  I2COUT SDA, Wr23016, IODIR0, [%11111111]      ' make GP0 pins inputs
  I2CIN SDA, Rd23016, GP1, [lcdCtrl]            ' get control bits
  lcdRW = 1                                     ' set to read mode
  I2COUT SDA, Wr23016, GP1, [lcdCtrl]
  lcdE = 1
  I2COUT SDA, Wr23016, GP1, [lcdCtrl]
  I2CIN SDA, Rd23016, GP0, [lcdIO]              ' get byte from LCD
  lcdE = 0                                      ' restore E
  lcdRW = 0                                     ' back to write mode
  I2COUT SDA, Wr23016, GP1, [lcdCtrl]           ' update control bits
  I2COUT SDA, Wr23016, IODIR0, [%00000000]      ' return buss to outputs
  RETURN


' Writes stored (in DATA statement) zero-terminated string to LCD
' -- position LCD cursor
```

```
' -- point to 0-terminated string (first location in 'addr')

Put_String:
  DO
    READ addr, lcdIO                            ' read char from addr
    addr = addr + 1                             ' update addr
    IF (lcdIO = 0) THEN EXIT                    ' terminate loop if 0
    GOSUB LCD_Write                             ' else write char
   LOOP
  RETURN


' Get buttons status (1 = pressed)

Get_Buttons:
  I2CIN SDA, Rd23016, GP1, [lcdCtrl]            ' read port 1 bits
  btns = lcdCtrl                                ' get low nib
  RETURN


' Turn Terminal LED On

LED_On:
  I2CIN SDA, Rd23016, GP1, [lcdCtrl]            ' read port 1 bits
  IF (lcdLED = Is_Off) THEN                     ' check LED bit
    lcdLED = Is_On                              ' turn on if off now
    I2COUT SDA, Wr23016, GP1, [lcdCtrl]         ' update port 1 bits
  ENDIF
  RETURN


' Turn Terminal LED Off

LED_Off:
  I2CIN SDA, Rd23016, GP1, [lcdCtrl]            ' read port 1 bits
  IF (lcdLED = Is_On) THEN                      ' check LED bit
    lcdLED = Is_Off                             ' turn off if on now
    I2COUT SDA, Wr23016, GP1, [lcdCtrl]         ' update port 1 bits
  ENDIF
  RETURN


' Set LED output to value of ledStatus

Set_LED:
  I2CIN SDA, Rd23016, GP1, [lcdCtrl]            ' read port 1 bits
  IF (lcdLED <> ledStatus) THEN                 ' check LED bit
    lcdLED = ledStatus                          ' set LED to new status
    I2COUT SDA, Wr23016, GP1, [lcdCtrl]         ' update port 1 bits
  ENDIF
  RETURN
```