Column #125, September 2005 by Jon Williams:

Control from Your Favorite Terminal

Well, it's good to be home. Since the start of the EFX group my colleague, John Barrowman, and I have been doing a lot of travel and participation in group events, many having to do with Halloween and holiday decorating. It's interesting what folks will ask for, some of it odd, some of it quite straightforward. After the MIDI project we did with the SX28 a few months ago I got a lot of mail asking how to use a terminal program and an SX for device control. We can do that – and with some recent updates to the SX/B compiler it's even easier to do.

I have to admit that I'm having an absolute blast with the SX/B compiler. That may seem silly, especially since I'm "on the inside" and actually part of the development team. Still, I'm really having fun; SX/B is letting me build high-performance projects – both for work (several Parallax EFX products, for example) and for play – with relative ease. And when one needs to build lots of do-dads, the low cost of the SX controller line is certainly a big help.

I feel like my greatest strength for the SX/B team is coming from the ranks of BASIC Stamp users; like many of you, I'm just not patient enough to write full-blown assembly language programs (and I have tremendous admiration for those that do). What I like about SX/B is that I can get full-performance from the SX without having to go the assembly language

route. I frequently send a note to our compiler engineer that says, "Hey, I'd like to do this…" He's a pretty accommodating guy and with input from me and other devoted users, SX/B continues to grow.

The latest version of SX/B (as of this article) is 1.4, and it offers a couple of really nice new features that we'll exploit this month in our project. The first is the ability to allow a subroutine to return a value to the caller without having to explicitly declare the destination address in the call. We used to do this:

```
  RX_BYTE @char
```

Now we can do this:

```
  char = RX_BYTE
```

Why does this matter? Well, the latter version is easier to understand and we don't have to remember to add the pesky '@' (address of) symbol. It actually simplifies the subroutine code as well. Prior to version 1.3 (when return values were introduced) we would write the RX_BYTE subroutine like this:

```
RX_BYTE:
  temp1 = __PARAM1
  SERIN SIn, Baud, temp2
  __RAM(temp1) = temp2
  RETURN temp1
```

As you can see, the subroutine is expecting an address to be passed as a parameter (we can tell because the __RAM array expects an address). If we forgot to put the '@' symbol in front of the destination variable name the value received by the serial port would not go where it was intended – this could be frustrating to track down.

Let's see the same subroutine that returns a value:

```
RX_BYTE:
  SERIN SIn, Baud, temp1
  RETURN temp1
```

I think you'll agree that the second version is easier and it even uses one less variable.

The other neat feature recently introduced in SX/B is string (address) handling. It's a little more involved, so let's save that for our project.

Cheap PC Control

There's no denying that PCs are cheap – so much so that it's no longer out of the question to dedicate a PC to a control task. As I mentioned earlier, I got a lot of mail regarding the MIDI project. While many were interested in it, not every body wanted to invest in MIDI control software, especially when the control might be localized.

At about the same time I was getting that mail regarding the MIDI project my friend Rick was showed me a new product he was developing for the gas industry. It is a very modular system with components that are connected through a multi-drop RS-485 link. What was particularly interesting is that Rick chose to use a text interface between the devices. By using text to move data, Rick is able to monitor and control the system through a standard terminal program. Since the data moving back and forth is relatively sparse, the downside of having to convert to and from text is greatly outweighed by the simplicity of using a terminal program as a monitor and debugging tool.

Figure 125.1 shows the schematic for this month's project, which really doesn't get much simpler: an SX28 and a MAX232 level converter so we can connect to the PC. I haven't done anything with the outputs (RB and RC), as you'd have to decide what you're actually going to control before you connect to them. Start with LEDs to get the program working, and then connect whatever happens to be appropriate. It might be a ULN2803 for driving relays or solenoids, or an SSR (solid state relay) like the Crydom D2W203F-11 for controlling AC circuits.

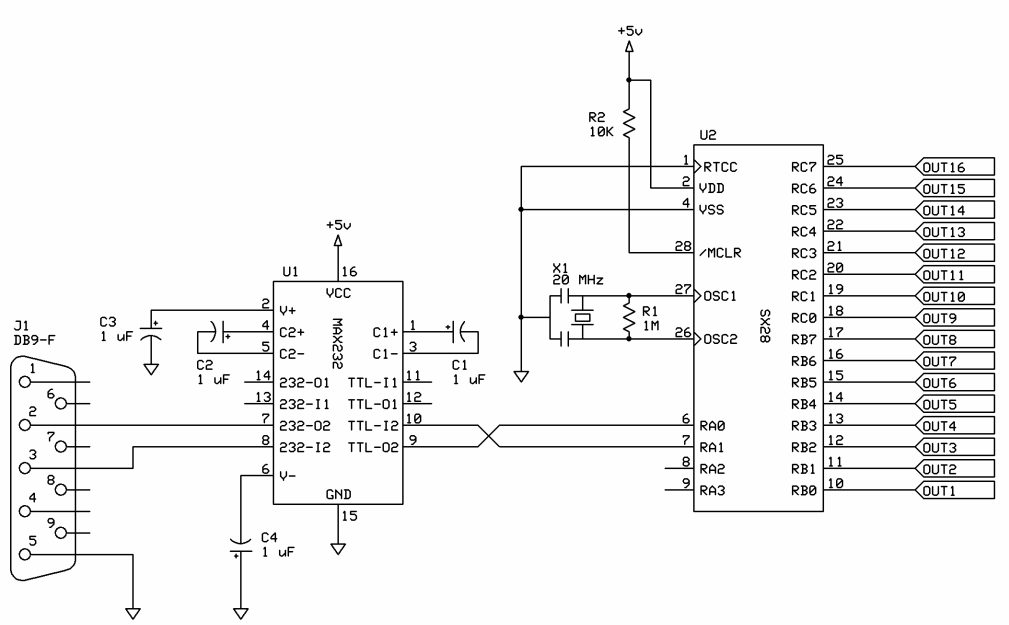

Figure 125.1 PC_Port 16 Schematic

Our goal this month is to create an interface between a generic terminal program and the SX – Figure 125.2 shows an example session using HyperTerminal. Once the [derivative] project is working through a terminal, it's a very simple matter to control the device from our favorite PC development tool: VB, C, Java, Python, Perl – you name it; the interface is just text.

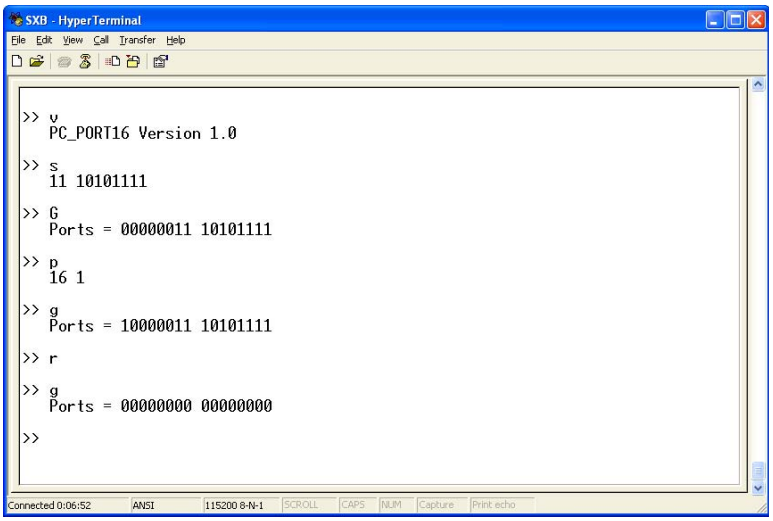

Figure 125.2 SX/B HyperTerminal

As I mentioned a second ago, SX/B 1.4 makes string handling easier for the programmer. We still have to write a subroutine to transmit the string to an external device, but the setup for sending a string is now a single-step process. At the top of our control program we'll define a bunch of z-strings (zero terminated strings) – in DATA statements much in the way we do it in the BASIC Stamp:

```
Prompt:
  DATA CR, LF, ">> ", 0

Version:
  DATA CR, LF, "   PC_PORT16 Version 1.0", CR, LF, 0

Pad:
  DATA CR, LF, "   ", 0

CRLF:
  DATA CR, LF, 0

PortStatus:
  DATA CR, LF, "   Ports = ", 0
```

Note that the strings also contain constant values for carriage return (CR) and line feed (LF) that are also defined in the program (i.e., they are not built in to SX/B).

It is our responsibility to write the subroutine that handles the string because SX/B has no idea where it's going to go. In this program, we'll send it to the PC using SEROUT. First, of course, we need to define the subroutine for the compiler:

```
TX_STR        SUB    2
```

As you can see, a subroutine that handles a string requires two bytes: the base address and character offset (these will be handled by the compiler when we make the call to TX_STR). The reason for this is that the SX's [native] IREAD instruction will be used to pull a character and it requires a 12-bit address. Here's the code for TX_STR:

```
TX_STR:
  temp3 = __PARAM1
  temp4 = __PARAM2
  DO
    READ temp3 + temp4, temp5
    IF temp5 = 0 THEN EXIT
    TX_BYTE temp5
    INC temp4
    temp3 = temp3 + Z
  LOOP
  RETURN
```

We start by saving the base address and character offset in variables temp1 and temp2. Then we enter a loop that uses READ to pull a character and, if the character value is not zero, we send it to the PC with TX_BYTE. By using variables for the base and offset, both can be updated allowing the string to cross SX page boundaries. This makes our life simple, and the 1.4 compiler even lets us do this:

```
  TX_STR "Hello, World!"
```

Yes, we can embed a string right into the program code. A note of caution, however: the string will be embedded in place (the terminating zero is added by the compiler) so if we're going to use the same string more than once then using this style is not the best choice. Just to clarify, when a string is going to be used in more than one place in the program then the best thing to do is put that string into a DATA statement.

As we just saw, TX_STR calls TX_BYTE to send the character to the PC at the specified baud rate (115.2 kBaud in this program). Let's have a look at that code:

```
TX_BYTE:
  temp1 = __PARAM1
  IF __PARAMCNT = 1 THEN
    temp2 = 1
  ELSE
    temp2 = __PARAM2
    IF temp2 = 0 THEN
      temp2 = 1
    ENDIF
  ENDIF
  DO WHILE temp2 > 0
    SEROUT SOut, Baud, temp1
    DEC temp2
  LOOP
  RETURN
```

This routine requires at least one parameter, and can take two. The second parameter (if provided) will be the number of times to transmit the character. So if we want to send a string of 20 asterisks, we can do this:

```
  TX_BYTE "*", 20
```

Working our way through TX_BYTE we start by saving the character to transmit in temp1. Then we check the number of parameters sent by looking at __PARAMCNT. This is an internal variable and set by the compiler based on the syntax we use (one parameter or two). If only one parameter was sent then temp2 will be set to one, otherwise we set it to the second parameter. Since I don't think it makes sense to send a zero in the count parameter, the subroutine traps this condition and changes it to one.

The actual transmission of the character is done in a DO-LOOP construct that uses the count (temp2) parameter for control. Each time through the loop the character gets sent and the count variable is decremented. When the count reaches zero, the loop terminates and the subroutine is finished.

Okay, then, let's get into the program. After initialization, the program sends a prompt to the terminal (or control application) and then waits for input. In this case, the input will be a command character followed by a carriage return.

```
Main:
  TX_STR Prompt
  cmd = RX_BYTE
  IF cmd = CR THEN
    TX_STR CRLF
    GOTO Main
  ENDIF
  char = RX_BYTE
  IF char <> CR THEN
    TX_STR CRLF
    GOTO Main
  ENDIF
```

The reason I decided to follow the command character with a forced CR is that it allows me an "Oops!" condition in the event I press the wrong command key (some keys are expecting data that will change the SX outputs). If I press the wrong key then all I have to do is hit Esc or any other key (except CR) to get back to the prompt without consequence.

The program uses RX_BYTE to get a byte from the terminal. One of the things that this program does is convert letters to uppercase – this simplifies our command letter processing.

```
RX_BYTE:
  SERIN SIn, Baud, temp1
  IF temp1 >= "a" THEN
    IF temp1 <= "z" THEN
      temp1 = temp1 - $20
    ENDIF
  ENDIF
  RETURN temp1
```

As you can see, this subroutine is quite simple; we wait for a character then examine it to see if it falls between "a" and "z" (inclusive). If it does, then we subtract $20 from the character (ASCII code) to convert it to uppercase before returning it to the caller.

With a command character in hand we can compare it against a know list of commands and jump to the code that handles that. In the BASIC Stamp we frequently use LOOKDOWN and BRANCH to handle this sort of processing, but in the SX I prefer to use straightforward IF-THEN statements; in SX/B – because the way code is compiled – it seems to result in more efficient assembly output (have a look at the compiled source using Ctrl-L to see what I mean).

```
  IF cmd = "V" THEN Show_Version
  IF cmd = "G" THEN Get_Ports
  IF cmd = "S" THEN Set_Ports
  IF cmd = "H" THEN Set_HiPort
  IF cmd = "L" THEN Set_LoPort
  IF cmd = "P" THEN Set_OnePort
  IF cmd = "R" THEN Reset_Ports
```

As you can see, it would be quite easy for us to add new commands to the list. Let's have a look at how each command is handled, shall we?

The first command is "V" for Version. This feature may be important if we develop a piece of control software that can work with multiple control devices – getting the version (hence available features) from the connected device will prevent possible incompatibility issues.

```
Show_Version:
  TX_STR Version
  GOTO Main
```

Boy, that was tough, wasn't it? Since we've already covered sending strings there's really nothing else to cover.

Next is "G" for Get Ports Status. This command will return the status of the 16 output ports in this form:

Status = 00000000 00000000

Note that what follows "Status =" are the actual states of the pins, where "1" is on and "0" is off, and the display is MSB to LSB. What we need to do here is create a subroutine that will transmit a value as a binary string, much the way the PBASIC BIN8 modifier does.

First, the Get_Ports code:

```
Get_Ports:
  TX_STR PortStatus
  TX_BIN8 PortHi
  TX_BYTE " "
  TX_BIN8 PortLo
  TX_STR CRLF
  GOTO Main
```

And now the TX_BIN8 subroutine that is used by Get_Ports:

```
TX_BIN8:
  temp3 = __PARAM1
  FOR temp4 = 1 TO 8
    IF temp3.7 = 1 THEN
      TX_BYTE "1"
    ELSE
      TX_BYTE "0"
    ENDIF
    temp3 = temp3 << 1
  NEXT
  RETURN
```

The TX_BIN8 subroutine, of course, expects a value to be sent; this will be saved in temp3. Using a FOR-NEXT loop, the bits are examined from MSB to LSB. If the bit is set then we use TX_BYTE to send "1" otherwise we send "0." Since temp3 is a work variable and doesn't need to be preserved, the code is simplified by looking only at the MSB. In order to examine all of the bits, temp3 is shifted left each time through the loop. This moves the next bit into the MSB.

Okay, now that we can see the outputs, how do we change them? The program supports three different methods of updating the outputs: all 16 at once, the high and low groups, or individual port bits. Let's start with all ports using the "S" (Set All Ports) command:

```
Set_Ports:
  TX_STR Pad
  PortHi = RX_BIN8
  TX_BYTE " "
  PortLo = RX_BIN8
  TX_STR CRLF
  GOTO Main
```

For the Set_Ports code we need a routine that is the complement of TX_BIN8 – in this case it's RX_BIN8. This will allow use to receive a value expressed in binary form, and is used to accept values for the high port (RC) and low port (RB) separately. A space is transmitted after the receipt of the PortHi value to indicate a new input (for PortLo).

```
RX_BIN8:
  temp3 = 0
  FOR temp4 = 1 TO 8
    temp5 = RX_BYTE
    IF temp5 >= "0" THEN
```

```
      IF temp5 <= "1" THEN
        temp3 = temp3 << 1
        IF temp5 = "1" THEN
          INC temp3
        ENDIF
      ELSE
        EXIT
      ENDIF
    ELSE
      EXIT
    ENDIF
  NEXT
  RETURN temp3
```

We start by clearing temp3 that will ultimately hold the return value. Then we setup a FOR-NEXT loop to get eight bits. A character is retrieved from the serial port and checked to see if it's a valid binary digit: "0" or "1." If it is, then the return value is shifted left and the new bit value is added to the return variable. Shifting left means that the routine is expecting the value to be transmitted MSB first.

The FOR-NEXT loop takes advantage of EXIT to terminate early if a non-binary character is sent before the end of the loop. This allows us to enter the minimum number of bits required to express the value. If, for example, we enter "1111" and then press space, the value 15 will be returned to the caller.

There are two additional commands, "H" and "L," that allow the user to set the high and low ports independently. Those routines are simply subsets of the Get_Ports code.

I think the trickiest aspect of this program is the code for "P" (Set Individual Port) that allows the user to specify a port number (1 to 16) and its condition (0 for off, 1 for on). For this code we'll need a routine that will accept a decimal value: RX_DEC2.

```
RX_DEC2:
  temp3 = 0
  FOR temp4 = 1 TO 2
    temp5 = RX_BYTE
    IF temp5 >= "0" THEN
      IF temp5 <= "9" THEN
        temp3 = temp3 * 10
        temp5 = temp5 - "0"
        temp3 = temp3 + temp5
      ELSE
        EXIT
```

```
      ENDIF
    ELSE
      EXIT
    ENDIF
  NEXT
  RETURN temp3
```

While it may not seem so at first, this code is identical to the RX_BIN8 subroutine. The difference, of course, is in the decimal base. To "shift" digits in this code we need to multiply by ten, and then add the new value (after it is converted from its ASCII code) to the result. Since we're dealing in decimal and don't want to overrun the limitations of a byte, the subroutine allows a maximum of two digits.

And now it gets a little hairy … but just a little.

```
Set_OnePort:
  TX_STR Pad
  idx = RX_DEC2
  TX_BYTE " "
  cmd = RX_BYTE
  IF idx >= 1 THEN
    IF idx <= 8 THEN
      DEC idx
      temp1 = 1 << idx
      IF cmd = "1" THEN
        PortLo = PortLo | temp1
      ENDIF
      IF cmd = "0" THEN
        temp1 = ~temp1
        PortLo = PortLo & temp1
      ENDIF
    ENDIF
  ENDIF
  IF idx >= 9 THEN
    IF idx <= 16 THEN
      idx = idx - 9
      temp1 = 1 << idx
      IF cmd = "1" THEN
        PortHi = PortHi | temp1
      ENDIF
      IF cmd = "0" THEN
        temp1 = ~temp1
        PortHi = PortHi & temp1
      ENDIF
```

```
    ENDIF
  ENDIF
  TX_STR CRLF
  GOTO Main
```

This code is not as bad as it looks at first blush. What we have to remember is that SX/B is very close to assembly language (many instructions are 1-for-1) so it gets a bit verbose – certainly more than PBASIC.

The code waits for the port number, prints a space pad, and then waits for a state value. The port value passed is compared against valid ranges: 1-to-8 for the low port, and 9-to-16 for the high port. If the value sent to the program falls outside of either range this section terminates and goes back to Main.

For analysis, let's assume that the user entered a port value of "4" and a state value of "1"; the user wants to turn output 4 on. First we zero-align the port value based on the group that will be updated, and then a mask is created from this value. In this case, the port 4 value gets converted to a pin-mask of %00001000. If the state is "1" then the mask is ORed with the appropriate SX port to enable the specified bit. If the state is "0" then the mask is inverted and ANDed with the SX port to clear the selected port bit.

Finally, we have the "R" command to reset (clear) the outputs.

```
Reset_Ports:
  PortHi = %00000000
  PortLo = %00000000
  TX_STR CRLF
  GOTO Main
```

Nothing magic here, simply clear the ports and go back to the top.

Okay, I think that about does it. I hope that you learned something from this project and that you can use it as the starting point for some neat PC-based control projects. And, by the way, if you need more ports remember that the SX48 and SX52 are available – and Parallax is selling fully-populated SX48 and SX52 proto boards for ten bucks! With this framework code and all those IO pins, there's no limit to what you could do.

Before I close, let me explain something. You may have noticed that I always use the variables temp1 through temp5 in my SX/B subroutines. There is a method to this apparent madness. What we haven't really discussed yet is that SX/B allows external files to be included in a listing, so by being consistent with subroutine variable names it's easier to

bundle common routines like RX_BYTE and TX_BYTE in a separate file. Then we can do this:

```
LOAD RXTX.SXB
```

Cool, huh? Yeah, I think so too.

Have fun with the SX and until next time, Happy Stamping!

```
' =============================================================================
'
'   File..... PC_PORT16.SXB
'   Purpose... Provides 16 outputs from PC serial port
'   Author.... Jon Williams, Parallax
'   E-mail.... jwilliams@parallax.com
'   Started...
'   Updated... 31 AUG 2005
'
' =============================================================================


' -----------------------------------------------------------------------------
' Program Description
' -----------------------------------------------------------------------------
'
' Allows the programmer to turn a serial port into 16 digital outputs.
' Control is through a simple text protocol that allows control from a
' terminal or any other program that can send commands.
'
' Note: Requires SX/B 1.41 or higher for proper string handling.


' -----------------------------------------------------------------------------
' Device Settings
' -----------------------------------------------------------------------------

DEVICE          SX28, OSCXT2, TURBO, STACKX, OPTIONX
FREQ            4_000_000


' -----------------------------------------------------------------------------
' IO Pins
' -----------------------------------------------------------------------------

SIn             VAR     RA.0                    ' input from master
SOut            VAR     RA.1                    ' output to master
PortLo          VAR     RB
PortHi          VAR     RC


' -----------------------------------------------------------------------------
' Constants
' -----------------------------------------------------------------------------

Baud            CON     "T9600"                 ' use with MAX232/USB2SER

CR              CON     13
LF              CON     10
```

```
' -----------------------------------------------------------------------------
' Variables
' -----------------------------------------------------------------------------

cmd             VAR     Byte                    ' command input
char            VAR     Byte                    ' character in/out
idx             VAR     Byte                    ' loop control

temp1           VAR     Byte                    ' subroutine work vars
temp2           VAR     Byte
temp3           VAR     Byte
temp4           VAR     Byte
temp5           VAR     Byte


' =============================================================================
  PROGRAM Start
' =============================================================================

Prompt:
  DATA CR, LF, ">> ", 0

Version:
  DATA CR, LF, "   PC_PORT16 Version 1.0", CR, LF, 0

Pad:
  DATA CR, LF, "   ", 0

CRLF:
  DATA CR, LF, 0

PortStatus:
  DATA CR, LF, "   Ports = ", 0


' -----------------------------------------------------------------------------
' Subroutine Declarations
' -----------------------------------------------------------------------------

WAIT_MS         SUB     1, 2                    ' delay in milliseconds
RX_BYTE         SUB                             ' rx a byte
RX_BIN8         SUB                             ' rx byte in BIN8 format
RX_DEC2         SUB                             ' rx byte in DEC2 format
TX_BYTE         SUB     1, 2                    ' tx a byte { x count }
TX_STR          SUB     2                       ' tx a string
TX_BIN8         SUB     1                       ' tx byte in BIN8 format


' -----------------------------------------------------------------------------
' Program Code
' -----------------------------------------------------------------------------
```

```
Start:
  PLP_A = %0011                                 ' pull-up unused pins
  TRIS_B = %00000000                            ' make outputs
  TRIS_C = %00000000

  SOut = 1
  WAIT_MS 250

Main:
  TX_STR Prompt                                 ' send prompt
  cmd = RX_BYTE                                 ' get command
  IF cmd = CR THEN                              ' clear early CR
    TX_STR CRLF
    GOTO Main
  ENDIF
  char = RX_BYTE
  IF char <> CR THEN                            ' wait for CR
    TX_STR CRLF
    GOTO Main
  ENDIF

  IF cmd = "V" THEN Show_Version                ' process command
  IF cmd = "G" THEN Get_Ports
  IF cmd = "S" THEN Set_Ports
  IF cmd = "H" THEN Set_HiPort
  IF cmd = "L" THEN Set_LoPort
  IF cmd = "P" THEN Set_OnePort
  IF cmd = "R" THEN Reset_Ports

  TX_STR CRLF                                   ' force whitespace
  GOTO Main

Show_Version:
  TX_STR Version                                ' send the version
  GOTO Main

Get_Ports:
  TX_STR PortStatus                             ' send header
  TX_BIN8 PortHi                                ' send port status
  TX_BYTE " "                                   ' separator
  TX_BIN8 PortLo
  TX_STR CRLF
  GOTO Main

Set_Ports:
  TX_STR Pad                                    ' send bad
  PortHi = RX_BIN8                              ' get high bits
  TX_BYTE " "
  PortLo = RX_BIN8                              ' get low bits
  TX_STR CRLF
```

```
  GOTO Main

Set_HiPort:
  TX_STR Pad
  PortHi = RX_BIN8                              ' get high bits
  TX_STR CRLF
  GOTO Main

Set_LoPort:
  TX_STR Pad
  PortLo = RX_BIN8                              ' get low bits
  TX_STR CRLF
  GOTO Main

Set_OnePort:
  TX_STR Pad
  idx = RX_DEC2                                 ' get port value, 1 - 16
  TX_BYTE " "
  cmd = RX_BYTE                                 ' get command, "0".."1"
  IF idx >= 1 THEN
    IF idx <= 8 THEN
      DEC idx                                   ' zero align
      temp1 = 1 << idx                          ' make mask
      IF cmd = "1" THEN
        PortLo = PortLo | temp1                 ' turn on port bit
      ENDIF
      IF cmd = "0" THEN
        temp1 = ~temp1                          ' invert mask
        PortLo = PortLo & temp1
      ENDIF
    ENDIF
  ENDIF
  IF idx >= 9 THEN
    IF idx <= 16 THEN
      idx = idx - 9                             ' zero align
      temp1 = 1 << idx                          ' make mask
      IF cmd = "1" THEN
        PortHi = PortHi | temp1                 ' turn on port bit
      ENDIF
      IF cmd = "0" THEN
        temp1 = ~temp1                          ' invert mask
        PortHi = PortHi & temp1
      ENDIF
    ENDIF
  ENDIF
  TX_STR CRLF
  GOTO Main

Reset_Ports:
  PortHi = %00000000                            ' clear high port
  PortLo = %00000000                            ' clear low port
```

```
  TX_STR CRLF
  GOTO Main


' -----------------------------------------------------------------------------
' Subroutine Code
' -----------------------------------------------------------------------------

' Use: WAIT_MS baseDelay {, multiplier }
' -- delays in milliseconds: baseDelay { x multiplier }
' -- multiplier is optional

WAIT_MS:
  temp1 = __PARAM1                          ' capture base delay
  IF __PARAMCNT = 2 THEN                    ' multiplier?
    temp2 = __PARAM2                        ' yes, capture
  ELSE
    temp2 = 1                               ' no, set to 1
  ENDIF
  IF temp1 > 0 THEN
    IF temp1 > 0 THEN
      PAUSE temp1 * temp2
    ENDIF
  ENDIF
  RETURN

' -----------------------------------------------------------------------------

' Use: theVar = RX_BYTE
' -- receives one byte on "SIn" at "Baud"
' -- converts "a".."z" to "A".."Z" (makes uppercase)

RX_BYTE:
  SERIN SIn, Baud, temp1                    ' rx the byte
  IF temp1 >= "a" THEN                      ' check for lowercase
    IF temp1 <= "z" THEN
      temp1 = temp1 - $20                   ' make uppercase if needed
    ENDIF
  ENDIF
  RETURN temp1

' -----------------------------------------------------------------------------

' Use: theVar = RX_BIN8
' -- receives number sent as text in binary format
' -- up to eight digits
' -- non "0" or "1" digit terminates input

RX_BIN8:
  temp3 = 0                                 ' clear return value
  FOR temp4 = 1 TO 8                        ' loop through 8 bits
```

```
    temp5 = RX_BYTE                         ' get character
    IF temp5 >= "0" THEN                    ' validate
      IF temp5 <= "1" THEN                  '
        temp3 = temp3 << 1                  ' shift bits
        IF temp5 = "1" THEN
          INC temp3                         ' add "1" bit
        ENDIF
      ELSE
        EXIT
      ENDIF
    ELSE
      EXIT                                  ' exit if not "0" or "1"
    ENDIF
  NEXT
  RETURN temp3

' -----------------------------------------------------------------------------

' Use: theVar = RX_DEC2
' -- receives number sent as text in decimal format
' -- up to two digits
' -- non "0".."9" digit terminates input

RX_DEC2:
  temp3 = 0                                 ' clear return value
  FOR temp4 = 1 TO 2                        ' loop through 2 digits
    temp5 = RX_BYTE                         ' get character
    IF temp5 >= "0" THEN                    ' validate
      IF temp5 <= "9" THEN
        temp3 = temp3 * 10                  ' shift digits
        temp5 = temp5 - "0"                 ' convert ASCII to value
        temp3 = temp3 + temp5               ' add to return var
      ELSE
        EXIT
      ENDIF
    ELSE
      EXIT
    ENDIF
  NEXT
  RETURN temp3

' -----------------------------------------------------------------------------

' Use: TX_BYTE theByte {, count}
' -- transmit "theByte" at "Baud" on "SOut"
' -- optional "count" may be specified (must be > 0)

TX_BYTE:
  temp1 = __PARAM1                          ' save byte
  IF __PARAMCNT = 1 THEN                    ' if no count
    temp2 = 1                               '   set to 1
```

```
  ELSE                                      ' otherwise
    temp2 = __PARAM2                        '   get count
    IF temp2 = 0 THEN                       ' do not allow 0
      temp2 = 1
    ENDIF
  ENDIF
  DO WHILE temp2 > 0                        ' loop through count
    SEROUT SOut, Baud, temp1                ' send the byte
    DEC temp2                               ' decrement count
  LOOP
  RETURN

' -----------------------------------------------------------------------------

' Use: TX_STR [string | label]
' -- "string" is an embedded string constant
' -- "label" is DATA statement label for stored z-String

TX_STR:
  temp3 = __PARAM1                          ' get string offset
  temp4 = __PARAM2                          ' get string base

  DO
    READ temp4 + temp3, temp5               ' read a character
    IF temp5 = 0 THEN EXIT                  ' if 0, string complete
    TX_BYTE temp5                           ' send character
    INC temp3                               ' point to next character
    temp4 = temp4 + Z                       ' update base on overflow
  LOOP
  RETURN

' -----------------------------------------------------------------------------

' Use: TX_BIN8 theByte
' -- transmits value of "theByte" in BIN8 format

TX_BIN8:
  temp3 = __PARAM1                          ' save the value
  FOR temp4 = 1 TO 8                        ' loop through eight bits
    IF temp3.7 = 1 THEN                     ' if MSB is set
      TX_BYTE "1"                           '   send "1"
    ELSE                                    ' else
      TX_BYTE "0"                           '   send "0"
    ENDIF
    temp3 = temp3 << 1                      ' shift next bit to MSB
  NEXT
  RETURN
```

```
' =============================================================================
'
'   File..... PC_PORT16_SX52.SXB
'   Purpose... Provides 16 outputs from PC serial port
'   Author.... Jon Williams, Parallax
'   E-mail.... jwilliams@parallax.com
'   Started...
'   Updated... 16 OCT 2005
'
' =============================================================================


' -----------------------------------------------------------------------------
' Program Description
' -----------------------------------------------------------------------------
'
' Allows the programmer to turn a serial port into 16 digital outputs.
' Control is through a simple text protocol that allows control from a
' terminal or any other program that can send commands.
'
' Note: Requires SX/B 1.41 or higher for proper string handling.


' -----------------------------------------------------------------------------
' Device Settings
' -----------------------------------------------------------------------------

DEVICE          SX52, OSCXT2
FREQ            4_000_000


' -----------------------------------------------------------------------------
' IO Pins
' -----------------------------------------------------------------------------

SIn             VAR     RA.0                    ' input from master
SOut            VAR     RA.1                    ' output to master
PortLo          VAR     RB
PortHi          VAR     RC


' -----------------------------------------------------------------------------
' Constants
' -----------------------------------------------------------------------------

Baud            CON     "T9600"                 ' use with MAX232/USB2SER

CR              CON     13
LF              CON     10
```

```
' -----------------------------------------------------------------------------
' Variables
' -----------------------------------------------------------------------------

cmd             VAR     Byte                    ' command input
char            VAR     Byte                    ' character in/out
idx             VAR     Byte                    ' loop control

temp1           VAR     Byte                    ' subroutine work vars
temp2           VAR     Byte
temp3           VAR     Byte
temp4           VAR     Byte
temp5           VAR     Byte


' =============================================================================
  PROGRAM Start
' =============================================================================

Prompt:
  DATA CR, LF, ">> ", 0

Version:
  DATA CR, LF, "   PC_PORT16 Version 1.0", CR, LF, 0

Pad:
  DATA CR, LF, "   ", 0

CRLF:
  DATA CR, LF, 0

PortStatus:
  DATA CR, LF, "   Ports = ", 0


' -----------------------------------------------------------------------------
' Subroutine Declarations
' -----------------------------------------------------------------------------

WAIT_MS         SUB     1, 2                    ' delay in milliseconds
RX_BYTE         SUB                             ' rx a byte
RX_BIN8         SUB                             ' rx byte in BIN8 format
RX_DEC2         SUB                             ' rx byte in DEC2 format
TX_BYTE         SUB     1, 2                    ' tx a byte { x count }
TX_STR          SUB     2                       ' tx a string
TX_BIN8         SUB     1                       ' tx byte in BIN8 format


' -----------------------------------------------------------------------------
' Program Code
' -----------------------------------------------------------------------------
```

```
Start:
  PLP_A = %0011                                 ' pull-up unused pins
  PLP_D = %00000000
  PLP_E = %00000000

  SOut = 1
  TRIS_A = %11111101
  TRIS_B = %00000000                            ' make outputs
  TRIS_C = %00000000

  WAIT_MS 250

Main:
  TX_BYTE "*", 13
  TX_BYTE CR
  TX_STR "SX/B Compiler"
  TX_BYTE CR
  WAIT_MS 250, 2
  GOTO Main




  TX_STR Prompt                                 ' send prompt
  cmd = RX_BYTE                                 ' get command
  IF cmd = CR THEN                              ' clear early CR
    TX_STR CRLF
    GOTO Main
  ENDIF
  char = RX_BYTE
  IF char <> CR THEN                            ' wait for CR
    TX_STR CRLF
    GOTO Main
  ENDIF

  IF cmd = "V" THEN Show_Version                ' process command
  IF cmd = "G" THEN Get_Ports
  IF cmd = "S" THEN Set_Ports
  IF cmd = "H" THEN Set_HiPort
  IF cmd = "L" THEN Set_LoPort
  IF cmd = "P" THEN Set_OnePort
  IF cmd = "R" THEN Reset_Ports

  TX_STR CRLF                                   ' force whitespace
  GOTO Main

Show_Version:
  TX_STR Version                                ' send the version
  GOTO Main
```

```
Get_Ports:
  TX_STR PortStatus                         ' send header
  TX_BIN8 PortHi                            ' send port status
  TX_BYTE " "                               ' separator
  TX_BIN8 PortLo
  TX_STR CRLF
  GOTO Main

Set_Ports:
  TX_STR Pad                                ' send bad
  PortHi = RX_BIN8                          ' get high bits
  TX_BYTE " "
  PortLo = RX_BIN8                          ' get low bits
  TX_STR CRLF
  GOTO Main

Set_HiPort:
  TX_STR Pad
  PortHi = RX_BIN8                          ' get high bits
  TX_STR CRLF
  GOTO Main

Set_LoPort:
  TX_STR Pad
  PortLo = RX_BIN8                          ' get low bits
  TX_STR CRLF
  GOTO Main

Set_OnePort:
  TX_STR Pad
  idx = RX_DEC2                             ' get port value, 1 - 16
  TX_BYTE " "
  cmd = RX_BYTE                             ' get command, "0".."1"
  IF idx >= 1 THEN
    IF idx <= 8 THEN
      DEC idx                               ' zero align
      temp1 = 1 << idx                      ' make mask
      IF cmd = "1" THEN
        PortLo = PortLo | temp1             ' turn on port bit
      ENDIF
      IF cmd = "0" THEN
        temp1 = ~temp1                      ' invert mask
        PortLo = PortLo & temp1
      ENDIF
    ENDIF
  ENDIF
  IF idx >= 9 THEN
    IF idx <= 16 THEN
      idx = idx - 9                         ' zero align
      temp1 = 1 << idx                      ' make mask
```

```
      IF cmd = "1" THEN
        PortHi = PortHi | temp1             ' turn on port bit
      ENDIF
      IF cmd = "0" THEN
        temp1 = ~temp1                      ' invert mask
        PortHi = PortHi & temp1
      ENDIF
    ENDIF
  ENDIF
  TX_STR CRLF
  GOTO Main

Reset_Ports:
  PortHi = %00000000                        ' clear high port
  PortLo = %00000000                        ' clear low port
  TX_STR CRLF
  GOTO Main


' -----------------------------------------------------------------------------
' Subroutine Code
' -----------------------------------------------------------------------------

' Use: WAIT_MS baseDelay {, multiplier }
' -- delays in milliseconds: baseDelay { x multiplier }
' -- multiplier is optional

WAIT_MS:
  temp1 = __PARAM1                          ' capture base delay
  IF __PARAMCNT = 2 THEN                    ' multiplier?
    temp2 = __PARAM2                        ' yes, capture
  ELSE
    temp2 = 1                               ' no, set to 1
  ENDIF
  IF temp1 > 0 THEN
    IF temp1 > 0 THEN
      PAUSE temp1 * temp2
    ENDIF
  ENDIF
  RETURN

' -----------------------------------------------------------------------------

' Use: theVar = RX_BYTE
' -- receives one byte on "SIn" at "Baud"
' -- converts "a".."z" to "A".."Z" (makes uppercase)

RX_BYTE:
  SERIN SIn, Baud, temp1                    ' rx the byte
  IF temp1 >= "a" THEN                      ' check for lowercase
    IF temp1 <= "z" THEN
```

```
      temp1 = temp1 - $20                   ' make uppercase if needed
    ENDIF
  ENDIF
  RETURN temp1

' -----------------------------------------------------------------------------

' Use: theVar = RX_BIN8
' -- receives number sent as text in binary format
' -- up to eight digits
' -- non "0" or "1" digit terminates input

RX_BIN8:
  temp3 = 0                                 ' clear return value
  FOR temp4 = 1 TO 8                        ' loop through 8 bits
    temp5 = RX_BYTE                         ' get character
    IF temp5 >= "0" THEN                    ' validate
      IF temp5 <= "1" THEN                  '
        temp3 = temp3 << 1                  ' shift bits
        IF temp5 = "1" THEN
          INC temp3                         ' add "1" bit
        ENDIF
      ELSE
        EXIT
      ENDIF
    ELSE
      EXIT                                  ' exit if not "0" or "1"
    ENDIF
  NEXT
  RETURN temp3

' -----------------------------------------------------------------------------

' Use: theVar = RX_DEC2
' -- receives number sent as text in decimal format
' -- up to two digits
' -- non "0".."9" digit terminates input

RX_DEC2:
  temp3 = 0                                 ' clear return value
  FOR temp4 = 1 TO 2                        ' loop through 2 digits
    temp5 = RX_BYTE                         ' get character
    IF temp5 >= "0" THEN                    ' validate
      IF temp5 <= "9" THEN
        temp3 = temp3 * 10                  ' shift digits
        temp5 = temp5 - "0"                 ' convert ASCII to value
        temp3 = temp3 + temp5               ' add to return var
      ELSE
        EXIT
      ENDIF
    ELSE
```

```
      EXIT
    ENDIF
  NEXT
  RETURN temp3

' -----------------------------------------------------------------------------

' Use: TX_BYTE theByte {, count}
' -- transmit "theByte" at "Baud" on "SOut"
' -- optional "count" may be specified (must be > 0)

TX_BYTE:
  temp1 = __PARAM1                          ' save byte
  IF __PARAMCNT = 1 THEN                    ' if no count
    temp2 = 1                               '   set to 1
  ELSE                                      ' otherwise
    temp2 = __PARAM2                        '   get count
    IF temp2 = 0 THEN                       ' do not allow 0
      temp2 = 1
    ENDIF
  ENDIF
  DO WHILE temp2 > 0                        ' loop through count
    SEROUT SOut, Baud, temp1                ' send the byte
    DEC temp2                               ' decrement count
  LOOP
  RETURN

' -----------------------------------------------------------------------------

' Use: TX_STR [string | label]
' -- "string" is an embedded string constant
' -- "label" is DATA statement label for stored z-String

TX_STR:
  temp3 = __PARAM1                          ' get string offset
  temp4 = __PARAM2                          ' get string base

  DO
    READ temp4 + temp3, temp5               ' read a character
    IF temp5 = 0 THEN EXIT                  ' if 0, string complete
    TX_BYTE temp5                           ' send character
    INC temp3                               ' point to next character
    temp4 = temp4 + Z                       ' update base on overflow
  LOOP
  RETURN

' -----------------------------------------------------------------------------

' Use: TX_BIN8 theByte
' -- transmits value of "theByte" in BIN8 format
```

```
TX_BIN8:
  temp3 = __PARAM1                          ' save the value
  FOR temp4 = 1 TO 8                        ' loop through eight bits
    IF temp3.7 = 1 THEN                     ' if MSB is set
      TX_BYTE "1"                           '   send "1"
    ELSE                                    ' else
      TX_BYTE "0"                           '   send "0"
    ENDIF
    temp3 = temp3 << 1                      ' shift next bit to MSB
  NEXT
  RETURN
```

```
' ================================================================================
'
'   File...... SX52_Serial_Test.SXB
'   Purpose...
'   Author....
'   E-mail....
'   Started... 10/15/2005
'   Updated...
'
' ================================================================================


' --------------------------------------------------------------------------------
' Program Description
' --------------------------------------------------------------------------------


' --------------------------------------------------------------------------------
' Device Settings
' --------------------------------------------------------------------------------

DEVICE    SX52, OSCHS2, BOR42
FREQ        50_000_000          ' Use a 50 MHz Ceramic Resonator w/Parallel
                                ' 10K resistor across OSC1 & OSC2

' --------------------------------------------------------------------------------
' IO Pins
' --------------------------------------------------------------------------------

RxD                         VAR           RA.0
TxD                         VAR           RA.1

' Used in original example
'RxD           VAR     RA.2
'TxD           VAR     RA.3

' --------------------------------------------------------------------------------
' Constants
' --------------------------------------------------------------------------------

PcBaud         CON           "T9600"
CrLf                  CON           1


' --------------------------------------------------------------------------------
' Variables
' --------------------------------------------------------------------------------

temp1                 VAR           Byte    ' subroutine work vars
temp2                 VAR           Byte
temp3                 VAR           Byte
```

```
temp4                   VAR             Byte
temp5                   VAR             Byte


' =============================================================================
  PROGRAM Start
' =============================================================================

Text:
  DATA  13, 10, "Test of the SX52 Proto Board", 13, 10, 0


' -----------------------------------------------------------------------------
' Subroutine Declarations
' -----------------------------------------------------------------------------

WAIT_MS                 SUB     1, 2
TX_BYTE                 SUB     1
TX_STRING               SUB     2               ' string pointer = 2 bytes


' -----------------------------------------------------------------------------
' Program Code
' -----------------------------------------------------------------------------

Start:
  TRIS_A=%00000000                              ' All outputs
  RA=2                                          ' Set RA.1 High (RS-232 Output)
  WAIT_MS 250, 4
  TX_BYTE 12                                    ' clear terminal screen

Main:
  WAIT_MS 250, 4
  TX_STRING Text
  GOTO Main


' -----------------------------------------------------------------------------
' Subroutine Code
' -----------------------------------------------------------------------------

' Use: TX_BYTE theByte
' -- sends "theByte" out TxD at Baud

TX_BYTE:
  temp3 = __PARAM1                              ' capture byte
  SEROUT TxD, PcBaud, temp3                     ' send it
  RETURN

' -----------------------------------------------------------------------------
```

```
' Use: TX_STRING [ string | label ]
' -- "string" is an embedded literal string
' -- "label" is DATA statement label for stored z-String

TX_STRING:
  temp1 = __PARAM1                          ' get string offset
  temp2 = __PARAM2
  DO
    READ temp2 + temp1, temp3               ' read a character
    IF temp3 = 0 THEN EXIT                  ' if 0, string complete
    TX_BYTE temp3                           ' send the byte
    INC temp1                               ' point to next character
    temp2 = temp2 + Z                       ' update base on overflow
  LOOP

  RETURN

' -----------------------------------------------------------------------------

' Use: WAIT_MS baseDelay {, multiplier }
' -- delays in milliseconds: baseDelay { x multiplier }
' -- multiplier is optional

WAIT_MS:
  temp4 = __PARAM1                          ' capture base delay
  IF __PARAMCNT = 2 THEN                    ' multiplier?
    temp5 = __PARAM2                        ' yes, capture
  ELSE
    temp5 = 1                               ' no, set to 1
  ENDIF
  IF temp4 > 0 THEN
    IF temp5 > 0 THEN
      PAUSE temp4 * temp5
    ENDIF
  ENDIF
  RETURN

' -----------------------------------------------------------------------------
```