Column #79, November 2001 by Jon Williams:

Expand Your Stamp's I/O With I²C

Unless you've been asleep for the past several years, computer networking has become the rage. Want proof? Just head down to your favorite book store and take a look at the computer section. It's certainly an important field -- many of us rely on LANs, WANs and, of course, the Internet as a normal part of our personal and technical lives.

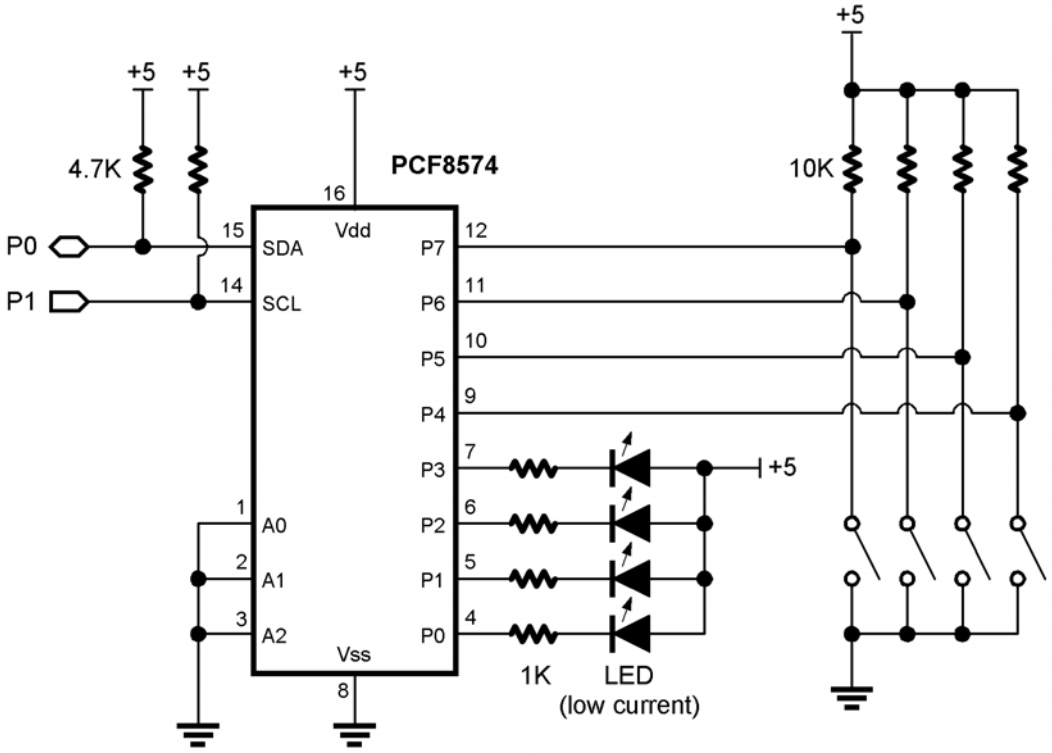
Now me personally ... I'm not that interested in computer networks beyond my connection to the Internet and getting e-mail. What does interest me, however, is microcontroller networking – especially with the BASIC Stamp.

Two Little Wires – Lots 'o Neat Devices

This month's "network" experiments will be conducted via the Philips I²C bus. Briefly, the I²C bus is a synchronous, two-wire bi-directional bus. A device (or devices) that control messages on the bus is called a master. Devices that respond to the messages are slaves. Note that with the BS2p, it can only serve as a master.

There are a lot of interesting I²C parts available. Two, in particular, are the Philips PCF8574 8-bit I/O expander and the Philips PCF8591 D2A/A2D. I selected these devices because they are very useful parts and because using them with the BS2p requires a slight adjustment with the BS2p syntax.

Figure 79.1: PCF8574 Circuit



What's The Address?

If you check the PBASIC manual, you'll find the BS2p I²C commands look like this:

I2CIN *SlaveID, Address {\LowAddress}, [InputData]*
I2COUT *SlaveID, Address {\LowAddress}, [OutputData]*

I point this out because most I²C devices have multiple internal addresses (i.e., memory devices). The PCF8574 and PCF8591 are configured differently though, and don't use the *Address* parameter. It's okay – this poses no problem for the BS2p.

Eight Bits At A Time

The PCF8574 gives us eight bits of additional I/O for the price of only two Stamp pins. And guess what? The device is addressable and we can drop up to eight of them on the same I²C bus, giving us up to 64 bits of I/O (if you use the PCF8574A you can have 16 for up to 128 I/O points!). Also remember that the BS2p will support two unique I²C bus systems (on pins 0 and 1; pins 8 and 9), doubling the possible number of I/O points.

Using the PCF8574 is very easy. The only thing we have to do is replace the *Address* parameter in the Stamp's I²C commands with a value that represents the desired direction (input = 1, output = 0) of the device's pins. Additionally, we have to refresh the state of output pins whenever we do a read (**I2CIN**). This is accomplished by ORing the output bits with the direction value.

Take a look at Listing 79.1. This is a very short, very simple program that will read four switches and then display an incrementing counter on four LEDs – remotely, of course, using the PCF8574. The count displayed on the LEDs is limited by the value input on the switches.

We start with an **I2CIN** statement to get the counter limit value from the switches. Notice that the value of the counter is ORed with our direction value since we're doing a read. This will keep any set output bits set. The tilde (~) is used to invert the counter bits since the LEDs are connected in an active-low configuration.

The same holds true for our inputs, so again we'll use the tilde. The inputs are connected to I/O bits 4 – 7 of the PCF8574 so we can use the HighNib modifier to pull them out. The program does a quick check to see if the count has reached the input limit and, if so, resets the counter to zero. After that, a **DEBUG** display lets us know what's going on and the PCF8574 outputs are refreshed (using **I2COUT**) with the counter value.

Next, the counter is incremented and a short delay is inserted so we can see the LEDs light. Then we start all over again at the top.

Easy, huh? You bet. I am particularly excited about using this device in my growing robotics experiments. By running an I²C buss through my robot chassis, I can drop I/O points wherever I need them and I only need bring two wires back to the BASIC Stamp.

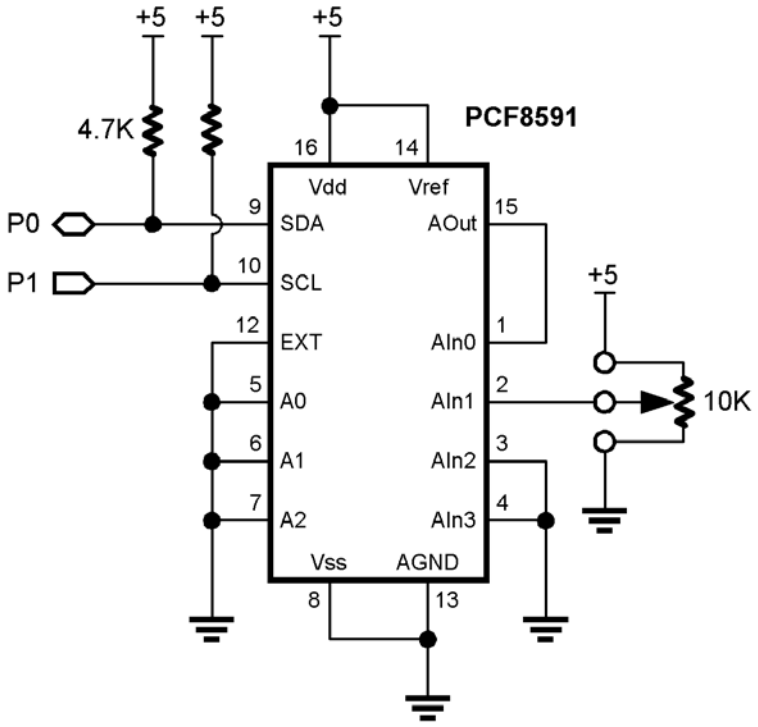
Oh ... one final note on the PCF8574. It may seem a bit counter-intuitive to connect the LEDs in an active low configuration. This is actually a requirement of the device. While it can sink (active low) up to 25 mA, it can only source (active high) 300 uA – even the lowest low-current LED will not light with only 300 uA.

Easy Analog I/O Expansion

We've just seen how easy it is to add extra digital I/O to the BS2p. Wouldn't it be nice if we could just as easily add analog I/O? Of course – and it is.

The device that makes this possible is the PCF8591. Like the PCF8574, this device is addressable and communicates to the host via I²C. It has four 8-bit analog input channels and one 8-bit analog output channel. The analog inputs can be configured as single-ended, differential or mixed.

Figure 79.2: PCF8591 Circuit



Let's keep things simple, shall we? For our demo program (Program Listing 79.2), we'll put the program in a loop and send the value of the *aOut* variable to the analog output pin. This output will be read back through analog input channel 0 on the PCF8591. Analog input channel 1 will read the voltage from the wiper of a 10K potentiometer. Channels 2 and 3 are not used and are tied to ground (do not leave unused inputs open).

When using the PCF8591 a control byte is sent after the *SlaveID*. When using the BS2p, we'll put the control byte in the *Address* parameter position. For this program, we've created a constant value for the control byte that enables the analog output pin, sets the analog inputs to single-ended and causes the analog channel number to be incremented after each conversion.

At the top of the program we use **I2COUT** to send the analog output value (held in *aOut*) to the PCF8591. The next thing we do is read back the four analog input channels. Notice the use of the dummy variable in the input data. This is necessary because the channel data lags by one byte. Using the dummy value aligns the analog data array (*aIn*) with the output from the PCF8591. The **STR** modifier and \4 parameter cause the **I2CIN** to retrieve four bytes. This is facilitated by the setting of the auto increment bit and setting the channel address to zero. You can, of course, read the channels individually. This requires a change in the control byte.

Okay, now that we have the values, let's display them. A simple loop iterates through the four channels and we use the */ (star-slash) operator to multiply each input value by 19.6 (each input bit equals 19.6 millivolts). **DEBUG** is used to display the current *aOut* value and the analog data for each channel.

After a slight delay the *aOut* variable is incremented and the loop starts again. Once again, very easy. Am I right?

For those of you who want to do more with the PCF8574 and the PCF8591, I've included the Philips technical documentation in this months file's. Have fun with them – they really are a breeze to use.

Parts Is Parts

If this article has sparked your interest in I²C components, and in particular with the BS2p, take a look at the Parallax "Plus Pack." This is a collection of I²C and Dallas 1-Wire components and code to help experimenters and engineers get started with the BS2p. Both the PCF8574 and PCF8591 are included in the Plus Pack. It also includes a 2x16 character LCD.

No BS2p? No Problem...

As you've seen, the BS2p makes I²C communications incredibly easy. If you're a BS2, BS2e or BS2sx user – take heart; you can still use I²C. Sure, it takes a bit of code, but it can be implemented on the non-BS2p Stamps. The best source of Stamp code for I²C communications is at Professor Peter Anderson's site. Here's the URL:

www.phanderson.com/stamp/

Remembering September 11th

It's difficult – as a proud American – to close without commenting on the events of September 11th. Like many, I was glued to the news for the hours and days that followed that horrific set of events. Even still, I find my TV spends more time tuned to CNN than to any other channel.

I bring all this up because of my own human nature. I went to California a few weeks after the attack and the normally-bustling DFW airport was like a ghost town. And I'll admit that I looked at the passengers I traveled with differently than I had ever done in the past. I wondered if I could have the courage to protect others like those heroes who brought down that plane in Pennsylvania instead of allowing it to bring harm to more innocents.

I've even caught myself wondering about the many pieces spontaneous e-mail I receive. Several questions about GPS (something I've been experimenting with) have caused me to wonder what the person is going to do with it – I've even gone so far as to ask. Of course, all have answered with interesting and peaceful applications. I was almost ashamed that I had even asked.

Technology is neither good nor bad – it's how we apply it that makes it so. I hope that you always use technology, including the BASIC Stamp, for applications that help people and never for anything that would do harm.

Happy Thanksgiving. Happy Stamping. Peace be with you all.

```
' -----[ Title ]-----------------------------------------------------------------
' Program Listing 79.1
' File...... PCF8574.BSP
' Purpose... Demonstrates remote I/O via the Philips PCF8574
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' Started...
' Updated... 7 OCT 2001

' {$STAMP BS2p}


' -----[ Program Description ]---------------------------------------------------
'
' This program reads for remote switches and uses this reading as a limit for
' a counter that is displayed on LEDs connected to the other four bits.
'
' Note: Most (not all) I2C devices have multiple internal addresses, so the
' I2CIN and I2COUT commands support this with an address parameter (this byte
' comes after the Slave Address byte).  With the PCF8574, replace the address
' byte with a value that reflects the desired state of the I/O pins, where
' 1 is an input, 0 is an output.  For example:
'
' %11110000 = Bits 0 - 3 are outputs, bits 4 - 7 are inputs
'
' For the PCF8574 the syntax becomes:
'
'    I2CIN  pin, ddr value, [in byte]
'    I2COUT pin, ddr value, [out byte]
'
' Special Note: When reading inputs while using the PCF8574 in mixed I/O mode,
' you must refresh the output bits during the read.  This is easily accomplished
' by ORing the state of the output pins with the DDR value.
'
'    I2CIN  pin, (ddr value | out bits), [io byte]
'
' This program uses the bits in mixed mode and will use the syntax described
' immediately above.
'
' I/O Notes:
'
' The input bits are pulled up to Vdd (+5) through 10K.  The inputs are
' connected to Vss (ground) through a N.O. switch.  The inputs will read 1 when
' the switches are open, 0 when closed.
'
' PCF8574 can sink current, but provide almost no source current.  Inputs and
' outputs for this program are setup as active-low.  The tilde (~) in front of
' variables inverts the bits since we're using active low inputs and outputs.
```

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


' -----[ I/O Definitions ]----------------------------------------------------
'
I2Cpin          CON     0                       ' SDA on 0; SCL on 1


' -----[ Constants ]----------------------------------------------------------
'
DevType         CON     %0100 << 4              ' Device type
DevAddr         CON     %000 << 1               ' address = %000 -> %111
Wr8574          CON     DevType | DevAddr       ' write to PCF8574
Rd8574          CON     Wr8574 | 1              ' read from PCF8574

MixDDR          CON     %00001111               ' 1 = input, 0 = output


' -----[ Variables ]----------------------------------------------------------
'
ioByte          VAR     Byte                    ' i/o byte for PCF8574
limit           VAR     Nib                     ' counter limit
cntr            VAR     Nib                     ' counter


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


' -----[ Initialization ]-----------------------------------------------------
'
Initialize:
  DEBUG CLS                                     ' let DEBUG open
  PAUSE 250
  DEBUG "PCF8574 Demo", CR


' -----[ Main Code ]----------------------------------------------------------
'
Get Inputs:
  I2CIN I2Cpin, Rd8574, (MixDDR | ~cntr), [ioByte]
  limit = ~ioByte.HighNib                       ' invert limit bits
  IF (cntr <= limit) THEN Update LEDs           ' check counter limit
  cntr = 0                                      ' clear counter if at limit

Update LEDs:
  DEBUG Home, 10, 10
  DEBUG "Limit..... ", BIN4 limit, " (", DEC limit, ") ", CR
  DEBUG "Counter... ", BIN4 cntr, " (", DEC cntr, ") "
  I2COUT I2Cpin, Wr8574, MixDDR, [~cntr]        ' send new value
```

```
  cntr = cntr + 1 // 16                         ' update counter
  PAUSE 250
  GOTO Get Inputs


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

```
' -----[ Title ]-----------------------------------------------------------------
' Program Listing 79.2
' File...... PCF8591.BSP
' Purpose... PCF8591 A2D/D2A Demo
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' Started...
' Updated... 07 OCT 2001

' {$STAMP BS2p}


' -----[ Program Description ]---------------------------------------------------
'
' This program demonstates the Philips PCF8591 4-channel A2D plus 1-channel
' D2A.  Channel 0 input is tied to the output of the D2A pin.  Channel 1 input
' is tied to the wiper of a pot.  Channes 2 and 3 are tied to Vss.
'
' The PCF85591 uses a control byte after the Slave Address.  The control byte
' data (see details in PCF8591 documentation) is used to enable the analog
' output bit and set the kind of analog inputs.  In this demo, the analog output
' bit is enabled and four single-ended analog inputs are used.
'
' Note that the first byte transmitted in a read cycle contains the conversion
' result code of the previous read cycle, so a dummy byte is placed ahead of
' the analog input array in the I2CIN command.


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


' -----[ I/O Definitions ]-------------------------------------------------------
'
I2Cpin          CON     0                       ' SDA on 0; SCL on 1


' -----[ Constants ]-------------------------------------------------------------
'
DevType         CON     %1001 << 4              ' device type
DevAddr         CON     %000 << 1               ' address = %000 -> %111
Wr8591          CON     DevType | DevAddr       ' write to PCF8591
Rd8591          CON     Wr8591 | 1              ' read from PCF8591

EnableD2A       CON     %1 << 6                 ' enable analog output
FourSngl        CON     %00 << 4                ' four single-ended inputs
ThreeDiff       CON     %01 << 4                ' three differential inputs
SnglDiff        CON     %10 << 4                ' two single; one differential
TwoDiff         CON     %11 << 4                ' two differential inputs
AutoInc         CON     %1 << 2                 ' auto increment a2d channels
```

```
Ctrl            CON     EnableD2A | FourSngl | AutoInc

MVPB            CON     $139C                   ' millivolts per bit factor


' -----[ Variables ]------------------------------------------------------------
'
aOut            VAR     Byte                    ' analog out value
aIn             VAR     Byte(4)                 ' analog input channels
mVolts          VAR     Word                    ' convert to millivolts
dummy           VAR     mVolts.LowByte          ' place holder
chan            VAR     Nib                     ' channel number


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


' -----[ Initialization ]--------------------------------------------------------
'
Initialize:
  DEBUG CLS                                     ' call DEBUG window
  PAUSE 250                                     ' let it open
  DEBUG "PCF8591 Demo"


' -----[ Main Code ]-------------------------------------------------------------
'
Set D2A:
  DEBUG Home, 10, 10, "D2A Out........ ", DEC aOut, "  ", CR
  I2COUT I2Cpin, Wr8591, Ctrl, [aOut]

Get A2D:
  I2CIN I2Cpin, Rd8591, Ctrl, [dummy, STR aIn\4]

  FOR chan = 0 TO 3
    DEBUG "Channel ", DEC1 chan, " In... ", DEC aIn(chan), "  ", Tab
    mVolts = aIn(chan) */ MVPB
    DEBUG "(", DEC mVolts DIG 3, ".", DEC3 mVolts, " volts)", CR
  NEXT

  PAUSE 250                                     ' delay between updates
  aOut = aOut + 1                               ' increment analog output
  GOTO Set D2A                                  ' go again

  END


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