
**Column #115 November 2004 by Jon Williams:**

# I2C Again – And the Case for Continuos Improvement

*George Lucas says (and he may have been quoting someone else), "Movies are never 'done' – they're simply abandoned." Funny, that's how I feel about my BASIC Stamp programs; even the ones that work really well.*

I grew up, figuratively and literally, in a large corporation: the Toro Company. It was my first job out of the United States Air Force, and I ended up staying with Toro for about 14 (fantastic) years. I was lucky to have a lot of great mentors, and the lessons I learned at Toro stay with me today. One of Toro's core philosophies that I hold dear is that of continuous improvement. If something (a product, for example) can be made better, then the efforts to that end are well spent.

I get the idea that a lot of BASIC Stamp users have discovered the fun and utility of the myriad (over 1000) I2C devices available today, even those users that don't have the BS2p or BS2pe with the built-in I2COUT and I2CIN instructions. A couple years ago I wrote a column with manual (bit-banged) I2C code that would work on the BS2, BS2e, and BS2 sx. Well, that was a couple years ago and the PBASIC 2 and the BASIC Stamp compiler have

been upgraded since, so it just seems to make sense to revisit those programs to see if they could be improved. Indeed, they can, and that's just what we're going to work with this month.

Our purpose, then, is to do a very quick review of I2C and the implementation we can use easily with the BASIC Stamp 2 family, and then work through a few example chips so that we aren't fooled into thinking that our code doesn't stand up. I mention this because I get lots of "This just isn't possible..." e-mails when that is simply not the case (and I always send back proof of my position). Sometimes we have to look a bit beyond what we percieve are the "rules" and then bend them.

## Quickie I2C Review

Before I start, let me beg you, cojole you, plead with you – on my knees, if necessary – to download the I2C specification from Philips and at least give it a glance. And that goes for any I2C devices that you want to use as well. I think you'll find after working through a few parts here that any component you choose can be handled with just a tiny bit of custom code. You'll see in the examples.

Okay, now for the essentials. The I2C protocol is a 2-wire (synchronous) serial protocol that has a master and one or more slave devices. Yes, there is a provision in the specification for multiple masters, but that is beyond the scope of our experiments – and we're not likely to need multiple masters in a small microcontroller system anyway. The master generates the synchronous clock for all attached devices. Depending on what is happening at any given moment, any device on the bus might be a transmitter or a receiver. Data is sent back and forth a byte (eight bits) at a time, with the receiving device creating an acknowledge bit after each received byte.

The two wires used for the I2C bus are called SDA (serial data) and SCL (serial clock). These lines are pulled to Vdd through 4.7k resistors (typical). For a device to generate a "0" on either bus line, that line is pulled low. To create a "1" the bus pin is set to a Hi-Z (input) state and the pull-up takes care of the rest. We're going to cheat a bit, though, because the BS2 family has built-in commands for two-wire synchronous serial communication, SHIFTOUT and SHIFTIN, and these instructions nicely fullfill the byte and bit transmission and reception requirements of I2C. Both of these instructions drive the bus high to generate a "1" bit. In theory, this could create a problem if one of the other devices on the bus is shorted to ground. I've never had such a problem though, probably because the bit rate of SHIFTOUT and SHIFTIN is pretty swift, and the pin is left low when the function is finished. Even so, if

you're concerned you could always place 220-ohm resistors inline with the SDA and SCL pins.

Communication on the I2C bus begins with the master generating a "Start" condition. A Start is defined as bringing SDA low while SCL is high (see Figure 115.1). The master then transmits the slave address of the device it wishes to connect to. We'll be using 7-bit addressing (Figure 115.2) where the upper seven bits of the slave address byte contain the device type and address, and bit zero holds the data direction: "0" idicating a device write; "1" indicating a device read. What follows the slave address will vary, depending on the device and the type of request. On many devices we'll have one or two address bytes, followed by the data byte(s) to write to or read from the device. The transmission is terminated with a "Stop" condition; this is defined as bringing the SDA line from low to high while the SCL line remains high.

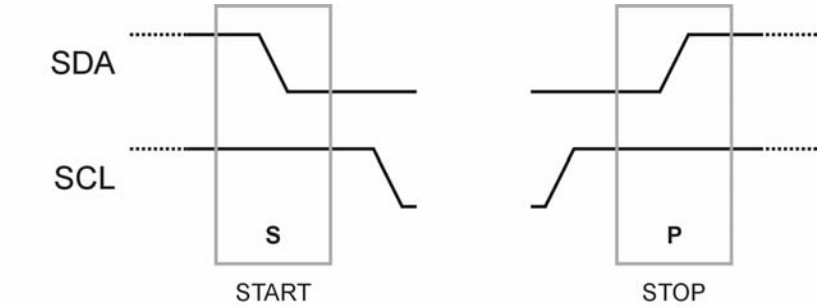


**Figure 115.1: Start Condition Brings SDA Low While SCL is High**

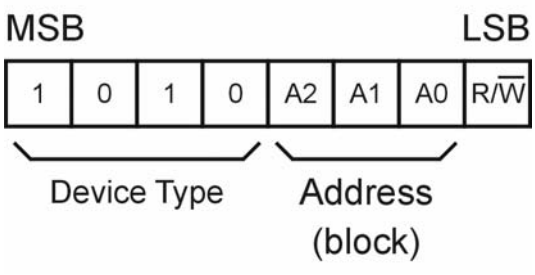


**Figure 115.2: 7-bit Addressing**

### Jump Right In, the Water's Warm

In my book demo code speaks louder than words, so let's just jump right in and discuss the low-level code for I2C communications. From these low-level routines we can communicate with any I2C device. What we'll do a bit later is create a useful set of higher-level routines that will handle most of our requirements. When those don't quite fit, we can build – from these same blocks – custom routines that will handle the special requirements of a given device.

We'll begin – logically – with the Start condition:

```
I2C_Start:
  INPUT SDA
  INPUT SCL
  LOW SDA

Clock_Hold:
  DO : LOOP UNTIL (SCL = 1)
  RETURN
```

The I2C_Start routine allows both bus lines to go high by making the control pins inputs and letting the pull-ups do their thing. Then the SDA line can be pulled low; a Start condition has been generated.

The I2C specification allows a slave device to indicate that it is not ready by holding the clock line low. This is called clock stretching. We can check for this at the section called Clock_Hold. If the SCL line is being held low, the (empty) DO-LOOP will run. The only possible danger here is a device that has shorted the SCL line to ground – this would cause the routine to hang indefitiely. We could increment a variable in the middle of the DO-LOOP and check for a timeout value if this becomes a problem but, again, this is something I've never experienced in any of my I2C experiments so I don't think it's worth doing except in a situation where a bus hang could create serious problems for the application.

After the Start condition, the master sends the address of the intended slave device to the bus. This is a single-byte transmission and is handled with the I2C_TX_Byte routine.

```
I2C_TX_Byte:
  SHIFTOUT SDA, SCL, MSBFIRST, [i2cWork\8]
  SHIFTIN SDA, SCL, MSBPRE, [i2cAck\1]
  RETURN
```

We can see how easy this is using SHIFTOUT to send out byte, MSB first. SHIFTIN handles picking the acknowledge bit from the bus. The \1 parameter is used with SHIFTIN so that we only produce one clock pulse for the acknowledge bit.

The complimentary routine, of course, is I2C_RX_Byte; its job is to receive a byte sent by the slave device.

```
I2C_RX_Byte_Nak:
  i2cAck = Nak
  GOTO I2C_RX

I2C_RX_Byte:
  i2cAck = Ack

I2C_RX:
  SHIFTIN SDA, SCL, MSBPRE, [i2cWork\8]
  SHIFTOUT SDA, SCL, LSBFIRST, [i2cAck\1]
  RETURN
```

This routine actually has two separate entry points: I2C_RX_Byte and I2C_RX_Byte_Nak. Why? The reason is that the master will indicate that it's requesting the last byte in a "package" by setting the ack bit high (Nak). The rest is as straightforward as with transmission. SHIFTIN handles the reception of the slave data byte, and SHIFTOUT transmits the acknowledge bit back.

To terminate a transmission the master generates a Stop condition.

```
I2C_Stop:
  LOW SDA
  INPUT SCL
  INPUT SDA
  RETURN
```

No magic here, either. The SDA line is held low while the SCL line is allowed to be pulled high by the bus pull-up; then the SDA line is released to its bus pull-up.

Okay, then, with these simple subroutines we can handle communication with any I2C device that uses 7-bit addressing. That said, we can save a lot of redundant code by constructing slightly higher level routines to write to or read from a device. Here's where we need to put in a bit of thought. You see, I2C devices can have zero, one, or two internal address bytes (called the word address) – depending on the device function. The PCF8574, for example, has no internal addresses; we simple write to or read from the device IO pins. The

MCP23016, though, has several configuration registers in addition to its IO, so it uses a single word address byte. If we look at an I2C EEPROM like the 24LC32 we'll see that it requires a two-byte word address to get to all of its memory locations.

The BS2p/pe I2COUT and I2CIN instructions handle these situations with a variable parameter list – we can specify no word address, one byte, or two bytes. In our code for the BS2/BS2e/BS2sx we'll have to be a bit verbose, but it's not tough and gives us the flexibility to handle multiple I2C devices of different configurations in the same project (a robot, for example).

Let's look at the code for writing a single byte to a given location within an I2C device.

```
Write_Byte:
  GOSUB I2C_Start
  i2cWork = slvAddr & %11111110
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Write_Byte
  IF (addrLen > 0) THEN
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0
    GOSUB I2C_TX_Byte
  ENDIF
  i2cWork = i2cData
  GOSUB I2C_TX_Byte
  GOSUB I2C_Stop
  RETURN
```

The routine begins by generating a Start condition, and then transmits the device slave address with bit zero of the slave address set to "0" to indicate a write operation. If the slave returns a Nak, the Start is resent – this called "Acknowledge Polling"; it causes the master to wait until the slave is actually ready for data before sending it.

Next, the routine will send the word address – if required by the device. The number of bytes required for the device word address is sent to the routine in the variable addrLen. For the PCF8574 the value of addrLen would be set to zero. If this was the case, the code would skip over sending the word address byte(s) and transmit the data byte, then generate the Stop condition.

If we were using an MCP23016 though, addrLen would be set to one and the word address (register we want to write to) would be placed in wrdAddr. The low byte (BYTE0) of wrdAddr is sent before the data byte and stop condition. For the 24LC32, addrLen would be set to two. In this case, both bytes of wrdAddr are transmitted; high byte (BYTE1), then low byte (BYTE0).

In an application program with multipe I2C devices, including a PCF8574A with its address bits set to %000, we could put the Write_Byte routine to use like this:

```
  devNum = %000
  slvAddr = PCF8574A | (devNum << 1)
  addrLen = 0
  i2cData = %00001111
  GOSUB Write_Byte
```

This would write %00001111 to the IO pins of the PCF8574A. Okay, now we can write to any location in an I2C device; let's build a routine that allows us to read data back.

```
Read_Byte:
  GOSUB I2C_Start
  IF (addrLen > 0) THEN
    i2cWork = slvAddr & %11111110
    GOSUB I2C_TX_Byte
    IF (i2cAck = Nak) THEN Read_Byte
    IF (addrLen = 2) THEN
      i2cWork = devAddr.BYTE1
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = devAddr.BYTE0
    GOSUB I2C_TX_Byte
    GOSUB I2C_Start
  ENDIF
  i2cWork = slvAddr | %00000001
  GOSUB I2C_TX_Byte
  GOSUB I2C_RX_Byte_Nak
  GOSUB I2C_Stop
  i2cData = i2cWork
  RETURN
```

You'll notice right off that the Read_Byte routine is a tad more involved than Write_Byte. The reason is this: at the time of use, we probably don't know what the internal address pointer of the device is sitting on, so this routine sets it manually. This is accomplished by starting what amounts to a write operation to the device, and then regenerating another Start

condition after the word address is transmitted. Of course, the word address is sent only for those devices that require it. After the address pointer is set (if required), the slave address is sent with bit 0 set to "1" to indicate a read operation. Since this routine only reads one byte, and that byte will be the last, the I2C_RX_Byte_Nak routine is used to retrieve the byte. With the data byte safely in hand, a Stop condition is generated and the work value is placed in i2cData for use by the main program code.

Let's say we wanted to read the value at location $200 in a 24LC32 (4K EEPROM). Our code would look something like this:

```
devNum = %000
slvAddr = EE24LC32 | (devNum << 1)
addrLen = 2
wrdAddr = $200
GOSUB Read_Byte
DEBUG "Location $200 holds: ", DEC i2cData
```

Again, this code is very verbose. If the only thing we had attached to our BASIC Stamp was a single 24LC32 we could set devNum, slvAddr, and addrLen as part of the initialization code and not have to worry about them after.

Hopefully this is all making sense now, and some of those data diagrams you find in I2C device data sheets are becoming easier to understand. Let's have a look at a couple more devices, and writing some additional routines to make data access simpler.

The first device we'll look at is the PCF8591. This is a nice little four-channel A2D converter with a single D2A output (all channels, in and out, have eight bits of resolution). When we look at its data sheet we'll see that writing to the D2A channel requires a control byte before the transmitting the analog output level. How do we handle this control byte ahead of our analog level byte? Well, the easiest way, in my opinion, is to tell the Write_Byte routine that we have a single-byte word address and put it in there. What this does for us is send two bytes to the same slave address without creating additional routines. Here's how simple it is to send a value to the analog output channel:

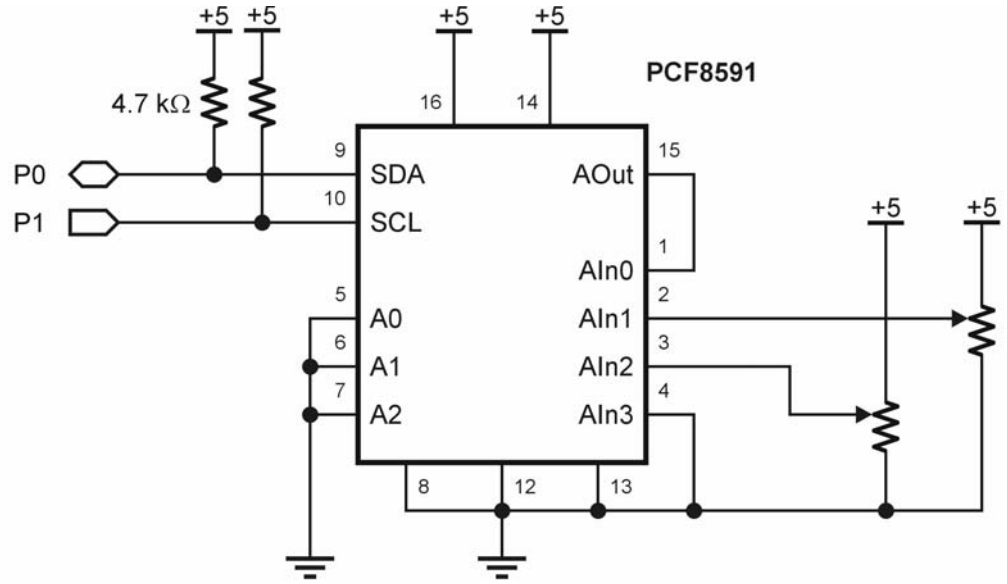


**Figure 115.3: PCF8591 4-channel A/D Converter**

```
addrLen = 1
wrdAddr = EnableD2A | AutoInc
i2cData = aOut
GOSUB Write_Byte
```

The control byte (which is placed in wrdAddr) is setup to enable the D2A output, configure all the analog inputs as single ended, and cause the PCF8591 to increment through them on each read.

Now things rev up a bit: we want to read all four analog input channels in a single operation. For this we're going to create a new high-level subroutine.

```
Read_Analog:
  GOSUB I2C_Start
  i2cWork = slvAddr | %00000001
  GOSUB I2C_TX_Byte
  GOSUB I2C_RX_Byte
  FOR idx = 0 TO 2
    GOSUB I2C_RX_Byte
    aIn(idx) = i2cWork
  NEXT
  GOSUB I2C_RX_Byte_Nak
  aIn(3) = i2cWork
  GOSUB I2C_Stop
  RETURN
```

After generating the Start condition and sending a read-mode slave address, we read back a byte and then throw it away? Why? Well, when you look at the PCF8591 data sheet (hint, hint) you'll see that a channel conversion is actually offset from a byte read. What this means is that the first byte read back is from a previous conversion and may no longer be valid.

Now that we've got fresh conversions, we can read channels 0 – 2 with a loop. This works because the I2C device will automatically increments the internal word address pointer after each read. The work byte used by our low-level I2C routines is transferred into the analog array used by the program. The final channel is read manually with I2C_RX_Byte_Nak as it is the final read in the group.

As you can see, our foundation routines are serving us well and we don't have to write a lot of code to get good use out of I2C devices. Let's look at one more example before wrapping up. In the previous example, reading all of the analog channels from the PCF8591 is called a "block read." What about a block write? Of course we can do that.

Let's say we want to add a real-time-clock to our project and we've already got other I2C devices. In this case, the DS1307 is a great solution. If we define the clock variables in the order they appear inside the DS1307, we can create a couple of very clean routines for setting or getting the clock data.

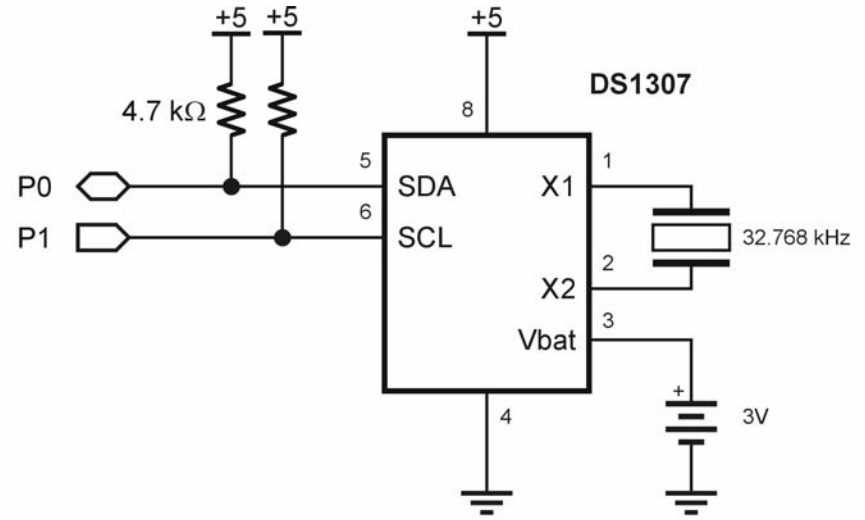


**Figure 115.5: DS1307**

First, here's how we would define the clock variables for the DS1307. Note that the order of these variables is critical for the proper operation of our block write and read routines.

```
secs            VAR     Byte
mins            VAR     Byte
hrs             VAR     Byte
day             VAR     Byte
date            VAR     Byte
month           VAR     Byte
year            VAR     Byte
control         VAR     Byte
```

Now, let's create a routine that sets all the clock variables in one fell swoop:

```
Set_Clock:
  GOSUB I2C_Start
  i2cWork = slvAddr & %11111110
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Set_Clock
  i2cWork = 0
  GOSUB I2C_TX_Byte
  FOR idx = 0 TO 7
    i2cWork = secs(idx)
    GOSUB I2C_TX_Byte
  NEXT
  GOSUB I2C_Stop
  RETURN
```

This should look pretty familiar by now. We generate a Start condition, send the slave address in write mode, and then send the word address. In this case, the word address is manually set to zero because this is the address of the seconds register. Since the internal word address will be incremented after each write, we can use a loop to write the clock variables, using secs as the root of an array.

Remember, the BASIC Stamp memory can be treated as an array even if we don't explicitly declare it as such – this can be very powerful when used carefully. And this is the reason that our variables must appear in the order that they do: the BASIC Stamp compiler assigns RAM space by variable size and in the order of declaration. Let's finish up with a block read of the DS1307:

```
Get_Clock:
  GOSUB I2C_Start
  i2cWork = slvAddr & %11111110
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Get_Clock
  i2cWork = 0
  GOSUB I2C_TX_Byte
  GOSUB I2C_Start
  i2cWork = slvAddr | %00000001
  GOSUB I2C_TX_Byte
  FOR idx = 0 TO 6
    GOSUB I2C_RX_Byte
    secs(idx) = i2cWork
  NEXT
  GOSUB I2C_RX_Byte_Nak
  control = i2cWork
  GOSUB I2C_Stop
  RETURN
```

Again, we begin with the Start condition, transmission of the slave address in write mode, followed by the register address; zero in this case to point to the seconds register. Then we resend the slave address in read mode, and use a loop to read the first seven clock variables (secs through year). The final variable, control, is read with I2C_RX_Byte_Nak because it is the last byte in the read sequence.

**More, More, More...**

Don't worry if this is all not perfectly clear yet – keep looking at the data sheets and the code examples and at some point you will have one of those "Aha!" moments of clarification. And be sure to download the example files because I've included more devices what we had room to discuss here, and I believe that by examining them you'll gain more insight into handling I2C devices with a BASIC Stamp microncontroller.

**What's Next?**

Those of you that have been around a while will remember that last December we created a one-wire serial slave device using the BS1 microcontroller. While simple, using a BS1 module is not the most cost-effective way to do this. Wouldn't it be nice if we could use a two dollar microcontroller without being forced to use assembly language? Of couse, and now we can. Next month we'll build a serial slave device using the SX micro and a free (can't beat that price) BASIC compiler from Parallax called SX/B.

Until then, happy Thanksgiving to you and your loved ones – and as always, happy Stamping.

```
' =============================================================================
'
'   File....... 24LC04.BS2
'   Purpose.... 24LC04 demo with a BS2/BS2e/BS2sx
'   Author..... Jon Williams, Parallax
'   E-mail..... jwilliams@parallax.com
'   Started....
'   Updated.... 10 SEP 2004
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =============================================================================


' -----[ Program Description ]---------------------------------------------


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


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

SDA             PIN     0                       ' I2C serial data line
SCL             PIN     1                       ' I2C serial clock line


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

Ack             CON     0                       ' acknowledge bit
Nak             CON     1                       ' no ack bit

EE24LC04        CON     %1010 << 4


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

slvAddr         VAR     Byte                    ' I2C slave address
devNum          VAR     Nib                     ' device number (0 - 7)
addrLen         VAR     Nib                     ' bytes in word addr (0 - 2)
wrdAddr         VAR     Word                    ' word address

i2cData         VAR     Byte                    ' data to/from device
i2cWork         VAR     Byte                    ' work byte for TX routine
i2cAck          VAR     Bit                     ' Ack bit from device

test            VAR     Nib
outVal          VAR     Byte
inVal           VAR     Byte
fails           VAR     Word
```

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


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

Check_Module:
  #IF ($STAMP >= BS2P) #THEN
    #ERROR "Use I2COUT and I2CIN!"
  #ENDIF

Setup:
  devNum = %00                                  ' chip select (%00 - %11)
  slvAddr = EE24LC04 | (devNum << 2)
  addrLen = 1                                   ' one word address byte

  DEBUG CLS
  DEBUG "24LC04 Demo   ", CR,
        "--------------", CR,
        "Address...    ", CR,
        "Output....    ", CR,
        "Input.....    ", CR,
        "Status....    ", CR,
        "Errors....    "


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

Main:
  fails = 0
  FOR wrdAddr = 0 TO 511                        ' test all locations
    slvAddr.BIT1 = wrdAddr.BIT8                 ' set page bit
    DEBUG CRSRXY, 11, 2, DEC3 wrdAddr
    FOR test = 0 TO 3                           ' use four patterns
      LOOKUP test, [$FF, $AA, $55, $00], outVal
      DEBUG CRSRXY, 11, 3, IHEX2 outVal
      i2cData = outVal
      GOSUB Write_Byte
      PAUSE 10
      GOSUB Read_Byte
      inVal = i2cData
      DEBUG CRSRXY, 11, 4, IHEX2 inVal,
            CRSRXY, 11, 5
      IF (inVal = outVal) THEN
        DEBUG "Pass "
      ELSE
        fails = fails + 1
        DEBUG "Fail ", CRSRXY, 11, 6, DEC fails
        EXIT                                    ' terminate location
      ENDIF
```

```
      PAUSE 10
    NEXT
  NEXT
  IF (fails = 0) THEN
    DEBUG CRSRXY, 11, 6, "None. All locations test good."
  ENDIF
  END


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




' =====[ High Level I2C Subroutines]=======================================

' Random location write
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte to be written is passed in "i2cData"

Write_Byte:
  GOSUB I2C_Start                               ' send Start
  i2cWork = slvAddr & %11111110                 ' send slave ID (write)
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Write_Byte             ' wait until not busy
  IF (addrLen > 0) THEN
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
  ENDIF
  i2cWork = i2cData                             ' send data
  GOSUB I2C_TX_Byte
  GOSUB I2C_Stop
  RETURN


' Random location read
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte read is returned in "i2cData"

Read_Byte:
  GOSUB I2C_Start                               ' send Start
  IF (addrLen > 0) THEN
    i2cWork = slvAddr & %11111110               ' send slave ID (write)
```

```
    GOSUB I2C_TX_Byte
    IF (i2cAck = Nak) THEN Read_Byte            ' wait until not busy
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
    GOSUB I2C_Start
  ENDIF
  i2cWork = slvAddr | %00000001                 ' send slave ID (read)
  GOSUB I2C_TX_Byte
  GOSUB I2C_RX_Byte_Nak
  GOSUB I2C_Stop
  i2cData = i2cWork
  RETURN


' -----[ Low Level I2C Subroutines ]-------------------------------------

' *** Start Sequence ***

I2C_Start:                                      ' I2C start bit sequence
  INPUT SDA
  INPUT SCL
  LOW SDA

Clock_Hold:
  DO : LOOP UNTIL (SCL = 1)                     ' wait for clock release
  RETURN


' *** Transmit Byte ***

I2C_TX_Byte:
  SHIFTOUT SDA, SCL, MSBFIRST, [i2cWork\8]      ' send byte to device
  SHIFTIN SDA, SCL, MSBPRE, [i2cAck\1]          ' get acknowledge bit
  RETURN


' *** Receive Byte ***

I2C_RX_Byte_Nak:
  i2cAck = Nak                                  ' no Ack = high
  GOTO I2C_RX

I2C_RX_Byte:
  i2cAck = Ack                                  ' Ack = low

I2C_RX:
  SHIFTIN SDA, SCL, MSBPRE, [i2cWork\8]         ' get byte from device
```

```
  SHIFTOUT SDA, SCL, LSBFIRST, [i2cAck\1]        ' send ack or nak
  RETURN


' *** Stop Sequence ***

I2C_Stop:                                        ' I2C stop bit sequence
  LOW SDA
  INPUT SCL
  INPUT SDA
  RETURN
```

```
' ================================================================================
'
'   File....... 24LC08.BS2
'   Purpose.... 24LC08 demo with a BS2/BS2e/BS2sx
'   Author..... Jon Williams, Parallax
'   E-mail..... jwilliams@parallax.com
'   Started....
'   Updated.... 10 SEP 2004
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' ================================================================================


' -----[ Program Description ]--------------------------------------------


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


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

SDA             PIN     0                       ' I2C serial data line
SCL             PIN     1                       ' I2C serial clock line


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

Ack             CON     0                       ' acknowledge bit
Nak             CON     1                       ' no ack bit

EE24LC08        CON     %1010 << 4


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

slvAddr         VAR     Byte                    ' I2C slave address
devNum          VAR     Nib                     ' device number (0 - 7)
addrLen         VAR     Nib                     ' bytes in word addr (0 - 2)
wrdAddr         VAR     Word                    ' word address

i2cData         VAR     Byte                    ' data to/from device
i2cWork         VAR     Byte                    ' work byte for TX routine
i2cAck          VAR     Bit                     ' Ack bit from device

test            VAR     Nib
outVal          VAR     Byte
inVal           VAR     Byte
fails           VAR     Word
```

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


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

Check_Module:
  #IF ($STAMP >= BS2P) #THEN
    #ERROR "Use I2COUT and I2CIN!"
  #ENDIF

Setup:
  addrLen = 1                                   ' one word address byte

  DEBUG CLS
  DEBUG "24LC08 Demo    ", CR,
        "---------------", CR,
        "Address...     ", CR,
        "Output....     ", CR,
        "Input.....     ", CR,
        "Status....     ", CR,
        "Errors....     "


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

Main:
  fails = 0
  FOR wrdAddr = 0 TO 1023                       ' test all locations
    slvAddr = EE24LC08 | (wrdAddr.BYTE1 << 1)   ' set block bits
    DEBUG CRSRXY, 11, 2, DEC4 wrdAddr
    FOR test = 0 TO 3                           ' use four patterns
      LOOKUP test, [$FF, $AA, $55, $00], outVal
      DEBUG CRSRXY, 11, 3, IHEX2 outVal
      i2cData = outVal
      GOSUB Write_Byte
      PAUSE 10
      GOSUB Read_Byte
      inVal = i2cData
      DEBUG CRSRXY, 11, 4, IHEX2 inVal,
            CRSRXY, 11, 5
      IF (inVal = outVal) THEN
        DEBUG "Pass "
      ELSE
        fails = fails + 1
        DEBUG "Fail ", CRSRXY, 11, 6, DEC fails
        EXIT                                    ' terminate location
      ENDIF
      PAUSE 10
    NEXT
  NEXT
```

```
  IF (fails = 0) THEN
    DEBUG CRSRXY, 11, 6, "None. All locations test good."
  ENDIF
  END


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




' =====[ High Level I2C Subroutines]=====================================

' Random location write
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte to be written is passed in "i2cData"

Write_Byte:
  GOSUB I2C_Start                               ' send Start
  i2cWork = slvAddr & %11111110                 ' send slave ID (write)
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Write_Byte             ' wait until not busy
  IF (addrLen > 0) THEN
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
  ENDIF
  i2cWork = i2cData                             ' send data
  GOSUB I2C_TX_Byte
  GOSUB I2C_Stop
  RETURN


' Random location read
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte read is returned in "i2cData"

Read_Byte:
  GOSUB I2C_Start                               ' send Start
  IF (addrLen > 0) THEN
    i2cWork = slvAddr & %11111110               ' send slave ID (write)
    GOSUB I2C_TX_Byte
    IF (i2cAck = Nak) THEN Read_Byte            ' wait until not busy
    IF (addrLen = 2) THEN
```

```
      i2cWork = wrdAddr.BYTE1                    ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                      ' send word address (0)
    GOSUB I2C_TX_Byte
    GOSUB I2C_Start
  ENDIF
  i2cWork = slvAddr | %00000001                  ' send slave ID (read)
  GOSUB I2C_TX_Byte
  GOSUB I2C_RX_Byte_Nak
  GOSUB I2C_Stop
  i2cData = i2cWork
  RETURN


' -----[ Low Level I2C Subroutines ]-------------------------------------

' *** Start Sequence ***

I2C_Start:                                       ' I2C start bit sequence
  INPUT SDA
  INPUT SCL
  LOW SDA

Clock_Hold:
  DO : LOOP UNTIL (SCL = 1)                      ' wait for clock release
  RETURN


' *** Transmit Byte ***

I2C_TX_Byte:
  SHIFTOUT SDA, SCL, MSBFIRST, [i2cWork\8]       ' send byte to device
  SHIFTIN SDA, SCL, MSBPRE, [i2cAck\1]           ' get acknowledge bit
  RETURN


' *** Receive Byte ***

I2C_RX_Byte_Nak:
  i2cAck = Nak                                   ' no Ack = high
  GOTO I2C_RX

I2C_RX_Byte:
  i2cAck = Ack                                   ' Ack = low

I2C_RX:
  SHIFTIN SDA, SCL, MSBPRE, [i2cWork\8]          ' get byte from device
  SHIFTOUT SDA, SCL, LSBFIRST, [i2cAck\1]        ' send ack or nak
  RETURN
```

```
' *** Stop Sequence ***

I2C_Stop:                                       ' I2C stop bit sequence
  LOW SDA
  INPUT SCL
  INPUT SDA
  RETURN
```

```
' ================================================================================
'
'   File....... 24LC16.BS2
'   Purpose.... 24LC16 demo with a BS2/BS2e/BS2sx
'   Author..... Jon Williams, Parallax
'   E-mail..... jwilliams@parallax.com
'   Started....
'   Updated.... 07 SEP 2004
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' ================================================================================


' -----[ Program Description ]--------------------------------------------


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


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

SDA             PIN     0                       ' I2C serial data line
SCL             PIN     1                       ' I2C serial clock line


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

Ack             CON     0                       ' acknowledge bit
Nak             CON     1                       ' no ack bit

EE24LC16        CON     %1010 << 4


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

slvAddr         VAR     Byte                    ' I2C slave address
devNum          VAR     Nib                     ' device number (0 - 7)
addrLen         VAR     Nib                     ' bytes in word addr (0 - 2)
wrdAddr         VAR     Word                    ' word address

i2cData         VAR     Byte                    ' data to/from device
i2cWork         VAR     Byte                    ' work byte for TX routine
i2cAck          VAR     Bit                     ' Ack bit from device

test            VAR     Nib
outVal          VAR     Byte
inVal           VAR     Byte
fails           VAR     Word
```

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


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

Check_Module:
  #IF ($STAMP >= BS2P) #THEN
    #ERROR "Use I2COUT and I2CIN!"
  #ENDIF

Setup:
  addrLen = 1                                   ' one word address byte

  DEBUG CLS
  DEBUG "24LC16 Demo    ", CR,
        "---------------", CR,
        "Address...     ", CR,
        "Output....     ", CR,
        "Input.....     ", CR,
        "Status....     ", CR,
        "Errors....     "


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

Main:
  fails = 0
  FOR wrdAddr = 0 TO 2047                       ' test all locations
    slvAddr = EE24LC16 | (wrdAddr.BYTE1 << 1)   ' set block bits
    DEBUG CRSRXY, 11, 2, DEC4 wrdAddr
    FOR test = 0 TO 3                           ' use four patterns
      LOOKUP test, [$FF, $AA, $55, $00], outVal
      DEBUG CRSRXY, 11, 3, IHEX2 outVal
      i2cData = outVal
      GOSUB Write_Byte
      PAUSE 10
      GOSUB Read_Byte
      inVal = i2cData
      DEBUG CRSRXY, 11, 4, IHEX2 inVal,
            CRSRXY, 11, 5
      IF (inVal = outVal) THEN
        DEBUG "Pass "
      ELSE
        fails = fails + 1
        DEBUG "Fail ", CRSRXY, 11, 6, DEC fails
        EXIT                                    ' terminate location
      ENDIF
      PAUSE 10
    NEXT
  NEXT
```

```
  IF (fails = 0) THEN
    DEBUG CRSRXY, 11, 6, "None. All locations test good."
  ENDIF
  END


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




' =====[ High Level I2C Subroutines]=======================================

' Random location write
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte to be written is passed in "i2cData"

Write_Byte:
  GOSUB I2C_Start                               ' send Start
  i2cWork = slvAddr & %11111110                 ' send slave ID (write)
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Write_Byte             ' wait until not busy
  IF (addrLen > 0) THEN
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
  ENDIF
  i2cWork = i2cData                             ' send data
  GOSUB I2C_TX_Byte
  GOSUB I2C_Stop
  RETURN


' Random location read
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte read is returned in "i2cData"

Read_Byte:
  GOSUB I2C_Start                               ' send Start
  IF (addrLen > 0) THEN
    i2cWork = slvAddr & %11111110               ' send slave ID (write)
    GOSUB I2C_TX_Byte
    IF (i2cAck = Nak) THEN Read_Byte            ' wait until not busy
    IF (addrLen = 2) THEN
```

```
      i2cWork = wrdAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
    GOSUB I2C_Start
  ENDIF
  i2cWork = slvAddr | %00000001                 ' send slave ID (read)
  GOSUB I2C_TX_Byte
  GOSUB I2C_RX_Byte_Nak
  GOSUB I2C_Stop
  i2cData = i2cWork
  RETURN


' -----[ Low Level I2C Subroutines ]-------------------------------------

' *** Start Sequence ***

I2C_Start:                                      ' I2C start bit sequence
  INPUT SDA
  INPUT SCL
  LOW SDA

Clock_Hold:
  DO : LOOP UNTIL (SCL = 1)                     ' wait for clock release
  RETURN


' *** Transmit Byte ***

I2C_TX_Byte:
  SHIFTOUT SDA, SCL, MSBFIRST, [i2cWork\8]      ' send byte to device
  SHIFTIN SDA, SCL, MSBPRE, [i2cAck\1]          ' get acknowledge bit
  RETURN


' *** Receive Byte ***

I2C_RX_Byte_Nak:
  i2cAck = Nak                                  ' no Ack = high
  GOTO I2C_RX

I2C_RX_Byte:
  i2cAck = Ack                                  ' Ack = low

I2C_RX:
  SHIFTIN SDA, SCL, MSBPRE, [i2cWork\8]         ' get byte from device
  SHIFTOUT SDA, SCL, LSBFIRST, [i2cAck\1]       ' send ack or nak
  RETURN
```

```
' *** Stop Sequence ***

I2C_Stop:                                          ' I2C stop bit sequence
  LOW SDA
  INPUT SCL
  INPUT SDA
  RETURN
```

```
' =============================================================================
'
'   File....... 24LC256.BS2
'   Purpose.... 24LC256 demo with a BS2/BS2e/BS2sx
'   Author..... Jon Williams, Parallax
'   E-mail..... jwilliams@parallax.com
'   Started....
'   Updated.... 07 SEP 2004
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =============================================================================


' -----[ Program Description ]---------------------------------------------


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


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

SDA             PIN     0                       ' I2C serial data line
SCL             PIN     1                       ' I2C serial clock line


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

Ack             CON     0                       ' acknowledge bit
Nak             CON     1                       ' no ack bit

EE24LC256       CON     %1010 << 4


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

slvAddr         VAR     Byte                    ' I2C slave address
devNum          VAR     Nib                     ' device number (0 - 7)
addrLen         VAR     Nib                     ' bytes in word addr (0 - 2)
wrdAddr         VAR     Word                    ' word address

i2cData         VAR     Byte                    ' data to/from device
i2cWork         VAR     Byte                    ' work byte for TX routine
i2cAck          VAR     Bit                     ' Ack bit from device

test            VAR     Nib
outVal          VAR     Byte
inVal           VAR     Byte
fails           VAR     Word
```

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


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

Check_Module:
  #IF ($STAMP >= BS2P) #THEN
    #ERROR "Use I2COUT and I2CIN!"
  #ENDIF

Setup:
  devNum = %000                                 ' chip select (%000 - %111)
  slvAddr = EE24LC256 | (devNum << 1)           ' setup slave ID
  addrLen = 2                                   ' 2 bytes in word address

  DEBUG CLS
  DEBUG "24LC256 Demo    ", CR,
        "----------------", CR,
        "Address...      ", CR,
        "Output....      ", CR,
        "Input.....      ", CR,
        "Status....      ", CR,
        "Errors....      "


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

Main:
  fails = 0
  FOR wrdAddr = $0000 TO $7FFF                  ' test all locations
    DEBUG CRSRXY, 11, 2, IHEX4 wrdAddr
    FOR test = 0 TO 3                           ' use four patterns
      LOOKUP test, [$FF, $AA, $55, $00], outVal
      DEBUG CRSRXY, 11, 3, IHEX2 outVal
      i2cData = outVal
      GOSUB Write_Byte
      PAUSE 10
      GOSUB Read_Byte
      inVal = i2cData
      DEBUG CRSRXY, 11, 4, IHEX2 inVal,
            CRSRXY, 11, 5
      IF (inVal = outVal) THEN
        DEBUG "Pass "
      ELSE
        fails = fails + 1
        DEBUG "Fail ", CRSRXY, 11, 6, DEC fails
        EXIT                                    ' terminate location
      ENDIF
      PAUSE 10
    NEXT
```

```
  NEXT
  IF (fails = 0) THEN
    DEBUG CRSRXY, 11, 6, "None. All locations test good."
  ENDIF
  END


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




' =====[ High Level I2C Subroutines]=====================================

' Random location write
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte to be written is passed in "i2cData"

Write_Byte:
  GOSUB I2C_Start                               ' send Start
  i2cWork = slvAddr & %11111110                 ' send slave ID (write)
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Write_Byte             ' wait until not busy
  IF (addrLen > 0) THEN
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
  ENDIF
  i2cWork = i2cData                             ' send data
  GOSUB I2C_TX_Byte
  GOSUB I2C_Stop
  RETURN


' Random location read
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte read is returned in "i2cData"

Read_Byte:
  GOSUB I2C_Start                               ' send Start
  IF (addrLen > 0) THEN
    i2cWork = slvAddr & %11111110               ' send slave ID (write)
    GOSUB I2C_TX_Byte
    IF (i2cAck = Nak) THEN Read_Byte            ' wait until not busy
```

```
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
    GOSUB I2C_Start
  ENDIF
  i2cWork = slvAddr | %00000001                 ' send slave ID (read)
  GOSUB I2C_TX_Byte
  GOSUB I2C_RX_Byte_Nak
  GOSUB I2C_Stop
  i2cData = i2cWork
  RETURN


' -----[ Low Level I2C Subroutines ]-------------------------------------

' *** Start Sequence ***

I2C_Start:                                      ' I2C start bit sequence
  INPUT SDA
  INPUT SCL
  LOW SDA

Clock_Hold:
  DO : LOOP UNTIL (SCL = 1)                     ' wait for clock release
  RETURN


' *** Transmit Byte ***

I2C_TX_Byte:
  SHIFTOUT SDA, SCL, MSBFIRST, [i2cWork\8]      ' send byte to device
  SHIFTIN SDA, SCL, MSBPRE, [i2cAck\1]          ' get acknowledge bit
  RETURN


' *** Receive Byte ***

I2C_RX_Byte_Nak:
  i2cAck = Nak                                  ' no Ack = high
  GOTO I2C_RX

I2C_RX_Byte:
  i2cAck = Ack                                  ' Ack = low

I2C_RX:
  SHIFTIN SDA, SCL, MSBPRE, [i2cWork\8]         ' get byte from device
  SHIFTOUT SDA, SCL, LSBFIRST, [i2cAck\1]       ' send ack or nak
  RETURN
```

```
' *** Stop Sequence ***

I2C_Stop:                                          ' I2C stop bit sequence
  LOW SDA
  INPUT SCL
  INPUT SDA
  RETURN
```

```
' ================================================================================
'
'   File....... 24LC32.BS2
'   Purpose.... 24LC32 demo with a BS2/BS2e/BS2sx
'   Author..... Jon Williams, Parallax
'   E-mail..... jwilliams@parallax.com
'   Started....
'   Updated.... 07 SEP 2004
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' ================================================================================


' -----[ Program Description ]--------------------------------------------


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


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

SDA             PIN     0                       ' I2C serial data line
SCL             PIN     1                       ' I2C serial clock line


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

Ack             CON     0                       ' acknowledge bit
Nak             CON     1                       ' no ack bit

EE24LC32        CON     %1010 << 4


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

slvAddr         VAR     Byte                    ' I2C slave address
devNum          VAR     Nib                     ' device number (0 - 7)
addrLen         VAR     Nib                     ' bytes in word addr (0 - 2)
wrdAddr         VAR     Word                    ' word address

i2cData         VAR     Byte                    ' data to/from device
i2cWork         VAR     Byte                    ' work byte for TX routine
i2cAck          VAR     Bit                     ' Ack bit from device

test            VAR     Nib
outVal          VAR     Byte
inVal           VAR     Byte
fails           VAR     Word
```

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


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

Check_Module:
  #IF ($STAMP >= BS2P) #THEN
    #ERROR "Use I2COUT and I2CIN!"
  #ENDIF

Setup:
  devNum = %000                                 ' chip select (%000 - %111)
  slvAddr = EE24LC32 | (devNum << 1)            ' setup slave ID
  addrLen = 2                                   ' 2 bytes in word address

  DEBUG CLS
  DEBUG "24LC32 Demo    ", CR,
        "---------------", CR,
        "Address...     ", CR,
        "Output....     ", CR,
        "Input.....     ", CR,
        "Status....     ", CR,
        "Errors....     "


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

Main:
  fails = 0
  FOR wrdAddr = 0 TO 4095                       ' test all locations
    DEBUG CRSRXY, 11, 2, DEC4 wrdAddr
    FOR test = 0 TO 3                           ' use four patterns
      LOOKUP test, [$FF, $AA, $55, $00], outVal
      DEBUG CRSRXY, 11, 3, IHEX2 outVal
      i2cData = outVal
      GOSUB Write_Byte
      PAUSE 10
      GOSUB Read_Byte
      inVal = i2cData
      DEBUG CRSRXY, 11, 4, IHEX2 inVal,
            CRSRXY, 11, 5
      IF (inVal = outVal) THEN
        DEBUG "Pass "
      ELSE
        fails = fails + 1
        DEBUG "Fail ", CRSRXY, 11, 6, DEC fails
        EXIT                                    ' terminate location
      ENDIF
      PAUSE 10
    NEXT
```

```
  NEXT
  IF (fails = 0) THEN
    DEBUG CRSRXY, 11, 6, "None. All locations test good."
  ENDIF
  END


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




' =====[ High Level I2C Subroutines]=====================================

' Random location write
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte to be written is passed in "i2cData"

Write_Byte:
  GOSUB I2C_Start                               ' send Start
  i2cWork = slvAddr & %11111110                 ' send slave ID (write)
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Write_Byte             ' wait until not busy
  IF (addrLen > 0) THEN
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
  ENDIF
  i2cWork = i2cData                             ' send data
  GOSUB I2C_TX_Byte
  GOSUB I2C_Stop
  RETURN


' Random location read
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte read is returned in "i2cData"

Read_Byte:
  GOSUB I2C_Start                               ' send Start
  IF (addrLen > 0) THEN
    i2cWork = slvAddr & %11111110               ' send slave ID (write)
    GOSUB I2C_TX_Byte
    IF (i2cAck = Nak) THEN Read_Byte            ' wait until not busy
```

```
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
    GOSUB I2C_Start
  ENDIF
  i2cWork = slvAddr | %00000001                 ' send slave ID (read)
  GOSUB I2C_TX_Byte
  GOSUB I2C_RX_Byte_Nak
  GOSUB I2C_Stop
  i2cData = i2cWork
  RETURN


' -----[ Low Level I2C Subroutines ]-------------------------------------

' *** Start Sequence ***

I2C_Start:                                      ' I2C start bit sequence
  INPUT SDA
  INPUT SCL
  LOW SDA

Clock_Hold:
  DO : LOOP UNTIL (SCL = 1)                     ' wait for clock release
  RETURN


' *** Transmit Byte ***

I2C_TX_Byte:
  SHIFTOUT SDA, SCL, MSBFIRST, [i2cWork\8]      ' send byte to device
  SHIFTIN SDA, SCL, MSBPRE, [i2cAck\1]          ' get acknowledge bit
  RETURN


' *** Receive Byte ***

I2C_RX_Byte_Nak:
  i2cAck = Nak                                  ' no Ack = high
  GOTO I2C_RX

I2C_RX_Byte:
  i2cAck = Ack                                  ' Ack = low

I2C_RX:
  SHIFTIN SDA, SCL, MSBPRE, [i2cWork\8]         ' get byte from device
  SHIFTOUT SDA, SCL, LSBFIRST, [i2cAck\1]       ' send ack or nak
  RETURN
```

```
' *** Stop Sequence ***

I2C_Stop:                                          ' I2C stop bit sequence
  LOW SDA
  INPUT SCL
  INPUT SDA
  RETURN
```

```
' =================================================================================
'
'   File....... 24LC515.BS2
'   Purpose.... 24LC515 demo with a BS2/BS2e/BS2sx
'   Author..... Jon Williams, Parallax
'   E-mail..... jwilliams@parallax.com
'   Started....
'   Updated.... 07 SEP 2004
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =================================================================================


' -----[ Program Description ]---------------------------------------------


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


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

SDA             PIN     0                       ' I2C serial data line
SCL             PIN     1                       ' I2C serial clock line


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

Ack             CON     0                       ' acknowledge bit
Nak             CON     1                       ' no ack bit

EE24LC515       CON     %1010 << 4


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

slvAddr         VAR     Byte                    ' I2C slave address
devNum          VAR     Nib                     ' device number (0 - 7)
addrLen         VAR     Nib                     ' bytes in word addr (0 - 2)
wrdAddr         VAR     Word                    ' word address

i2cData         VAR     Byte                    ' data to/from device
i2cWork         VAR     Byte                    ' work byte for TX routine
i2cAck          VAR     Bit                     ' Ack bit from device

test            VAR     Nib
outVal          VAR     Byte
inVal           VAR     Byte
fails           VAR     Word
```

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


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

Check_Module:
  #IF ($STAMP >= BS2P) #THEN
    #ERROR "Use I2COUT and I2CIN!"
  #ENDIF

Setup:
  devNum = %00                                  ' chip select (%00 - %11)
  slvAddr = EE24LC515 | (devNum << 1)           ' setup slave ID
  addrLen = 2                                   ' 2 bytes in word address

  DEBUG CLS
  DEBUG "24LC515 Demo    ", CR,
        "----------------", CR,
        "Address...      ", CR,
        "Output....      ", CR,
        "Input.....      ", CR,
        "Status....      ", CR,
        "Errors....      "


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

Main:
  fails = 0
  FOR wrdAddr = $0000 TO $FFFF                  ' test all locations
    slvAddr.BIT3 = wrdAddr.BIT15                ' set block bit
    DEBUG CRSRXY, 11, 2, IHEX4 wrdAddr
    FOR test = 0 TO 3                           ' use four patterns
      LOOKUP test, [$FF, $AA, $55, $00], outVal
      DEBUG CRSRXY, 11, 3, IHEX2 outVal
      i2cData = outVal
      GOSUB Write_Byte
      PAUSE 10
      GOSUB Read_Byte
      inVal = i2cData
      DEBUG CRSRXY, 11, 4, IHEX2 inVal,
            CRSRXY, 11, 5
      IF (inVal = outVal) THEN
        DEBUG "Pass "
      ELSE
        fails = fails + 1
        DEBUG "Fail ", CRSRXY, 11, 6, DEC fails
        EXIT                                    ' terminate location
      ENDIF
      PAUSE 10
```

```
    NEXT
  NEXT
  IF (fails = 0) THEN
    DEBUG CRSRXY, 11, 6, "None. All locations test good."
  ENDIF
  END


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




' =====[ High Level I2C Subroutines]=====================================

' Random location write
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte to be written is passed in "i2cData"

Write_Byte:
  GOSUB I2C_Start                               ' send Start
  i2cWork = slvAddr & %11111110                 ' send slave ID (write)
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Write_Byte             ' wait until not busy
  IF (addrLen > 0) THEN
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
  ENDIF
  i2cWork = i2cData                             ' send data
  GOSUB I2C_TX_Byte
  GOSUB I2C_Stop
  RETURN


' Random location read
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte read is returned in "i2cData"

Read_Byte:
  GOSUB I2C_Start                               ' send Start
  IF (addrLen > 0) THEN
    i2cWork = slvAddr & %11111110               ' send slave ID (write)
    GOSUB I2C_TX_Byte
```

```
    IF (i2cAck = Nak) THEN Read_Byte            ' wait until not busy
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
    GOSUB I2C_Start
  ENDIF
  i2cWork = slvAddr | %00000001                 ' send slave ID (read)
  GOSUB I2C_TX_Byte
  GOSUB I2C_RX_Byte_Nak
  GOSUB I2C_Stop
  i2cData = i2cWork
  RETURN


' -----[ Low Level I2C Subroutines ]-------------------------------------

' *** Start Sequence ***

I2C_Start:                                      ' I2C start bit sequence
  INPUT SDA
  INPUT SCL
  LOW SDA

Clock_Hold:
  DO : LOOP UNTIL (SCL = 1)                     ' wait for clock release
  RETURN


' *** Transmit Byte ***

I2C_TX_Byte:
  SHIFTOUT SDA, SCL, MSBFIRST, [i2cWork\8]      ' send byte to device
  SHIFTIN SDA, SCL, MSBPRE, [i2cAck\1]          ' get acknowledge bit
  RETURN


' *** Receive Byte ***

I2C_RX_Byte_Nak:
  i2cAck = Nak                                  ' no Ack = high
  GOTO I2C_RX

I2C_RX_Byte:
  i2cAck = Ack                                  ' Ack = low

I2C_RX:
  SHIFTIN SDA, SCL, MSBPRE, [i2cWork\8]         ' get byte from device
  SHIFTOUT SDA, SCL, LSBFIRST, [i2cAck\1]       ' send ack or nak
```

```
  RETURN


' *** Stop Sequence ***

I2C_Stop:                                          ' I2C stop bit sequence
  LOW SDA
  INPUT SCL
  INPUT SDA
  RETURN
```

```
' ================================================================================
'
'   File....... DS1307.BS2
'   Purpose.... DS1307 demo with a BS2/BS2e/BS2sx
'   Author..... Jon Williams, Parallax
'   E-mail..... jwilliams@parallax.com
'   Started....
'   Updated.... 08 SEP 2004
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' ================================================================================


' -----[ Program Description ]--------------------------------------------


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


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

SDA             PIN     0                       ' I2C serial data line
SCL             PIN     1                       ' I2C serial clock line
HrsIn           PIN     2                       ' adjust minutes button
MnsIn           PIN     3                       ' adjust hours button


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

Ack             CON     0                       ' acknowledge bit
Nak             CON     1                       ' no ack bit

DS1307          CON     %1101 << 4


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

slvAddr         VAR     Byte                    ' I2C slave address
devNum          VAR     Nib                     ' device number (0 - 7)
addrLen         VAR     Nib                     ' bytes in word addr (0 - 2)
wrdAddr         VAR     Word                    ' word address

i2cData         VAR     Byte                    ' data to/from device
i2cWork         VAR     Byte                    ' work byte for TX routine
i2cAck          VAR     Bit                     ' Ack bit from device

idx             VAR     Nib

secs            VAR     Byte                    ' DS1307 time registers
```

```
mins            VAR     Byte
hrs             VAR     Byte
day             VAR     Byte                    ' weekday
date            VAR     Byte                    ' day in month, 1 - 31
month           VAR     Byte
year            VAR     Byte
control         VAR     Byte                    ' SQWV I/O control

buttons         VAR     Nib                     ' debounced button inputs
btnHr           VAR     buttons.BIT1            ' advance hours
btnMn           VAR     buttons.BIT0            ' advance minutes


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


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

Check_Module:
  #IF ($stamp >= BS2P) #THEN
    #ERROR "Use I2COUT and I2CIN!"
  #ENDIF

Setup:
  slvAddr = DS1307                              ' 1 byte in word address
  addrLen = 1

  DEBUG CLS
  DEBUG "DS1307 Demo", CR,
        "-----------"

Reset_Clock:
  idx = HrsIn + MnsIn
  IF (idx = %00) THEN                           ' if both pressed, reset
    secs = $00
    mins = $00
    hrs = $06                                   ' 6:00 AM
    day = 5                                     ' Thur
    date = $01                                  ' 1st
    month = $01                                 ' January
    year = $04                                  ' 2004
    control = 0                                 ' disable SQW output
    GOSUB Set_Clock                             ' block write clock regs
  ENDIF


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

Main:
  DO
    GOSUB Get_Clock
```

```
    hrs = hrs & $3F
    DEBUG CRSRXY, 0, 2,
          HEX2 hrs, ":", HEX2 mins, ":", HEX2 secs
    PAUSE 100
    GOSUB Get_Buttons
    IF (buttons > %00) THEN
      hrs = hrs.NIB1 * 10 + hrs.NIB0            ' BCD to decimal
      hrs = hrs + btnHr // 24                   ' update
      hrs  = (hrs / 10 << 4) + (hrs // 10)      ' decimal to BCD
      mins = mins.NIB1 * 10 + mins.NIB0
      mins = mins + btnMn // 60
      mins = (mins / 10 << 4) + (mins // 10)
      secs = 0                                  ' reset
      GOSUB Set_Clock
    ENDIF
  LOOP
  END


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

Get_Buttons:
  buttons = %0011                               ' assume pressed
  FOR idx = 1 TO 5
    btnHr = btnHr & ~HrsIn                      ' validate inputs
    btnMn = btnMn & ~MnsIn
    PAUSE 5
  NEXT
  RETURN


' Do a block write to clock registers

Set_Clock:
  GOSUB I2C_Start                               ' send Start
  i2cWork = slvAddr & %11111110                 ' send slave ID (write)
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Set_Clock              ' wait until not busy
  i2cWork = 0                                   ' point at secs register
  GOSUB I2C_TX_Byte
  FOR idx = 0 TO 7                              ' write secs to control
    i2cWork = secs(idx)
    GOSUB I2C_TX_Byte
  NEXT
  GOSUB I2C_Stop
  RETURN


' Do a block read from clock registers

Get_Clock:
```

```
  GOSUB I2C_Start                                 ' send Start
  i2cWork = slvAddr & %11111110                   ' send slave ID (write)
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Get_Clock                ' wait until not busy
  i2cWork = 0                                     ' point at secs register
  GOSUB I2C_TX_Byte
  GOSUB I2C_Start
  i2cWork = slvAddr | %00000001                   ' send slave ID (read)
  GOSUB I2C_TX_Byte
  FOR idx = 0 TO 6                                ' read secs to year
    GOSUB I2C_RX_Byte
    secs(idx) = i2cWork
  NEXT
  GOSUB I2C_RX_Byte_Nak                           ' read control
  control = i2cWork
  GOSUB I2C_Stop
  RETURN


' =====[ High Level I2C Subroutines]=====================================

' Random location write
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte to be written is passed in "i2cData"

Write_Byte:
  GOSUB I2C_Start                                 ' send Start
  i2cWork = slvAddr & %11111110                   ' send slave ID (write)
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Write_Byte               ' wait until not busy
  IF (addrLen > 0) THEN
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                     ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                       ' send word address (0)
    GOSUB I2C_TX_Byte
  ENDIF
  i2cWork = i2cData                               ' send data
  GOSUB I2C_TX_Byte
  GOSUB I2C_Stop
  RETURN


' Random location read
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte read is returned in "i2cData"
```

```
Read_Byte:
  GOSUB I2C_Start                                ' send Start
  IF (addrLen > 0) THEN
    i2cWork = slvAddr & %11111110                ' send slave ID (write)
    GOSUB I2C_TX_Byte
    IF (i2cAck = Nak) THEN Read_Byte             ' wait until not busy
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                    ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                      ' send word address (0)
    GOSUB I2C_TX_Byte
    GOSUB I2C_Start
  ENDIF
  i2cWork = slvAddr | %00000001                  ' send slave ID (read)
  GOSUB I2C_TX_Byte
  GOSUB I2C_RX_Byte_Nak
  GOSUB I2C_Stop
  i2cData = i2cWork
  RETURN


' -----[ Low Level I2C Subroutines]---------------------------------------

' *** Start Sequence ***

I2C_Start:                                       ' I2C start bit sequence
  INPUT SDA
  INPUT SCL
  LOW SDA

Clock_Hold:
  DO : LOOP UNTIL (SCL = 1)                      ' wait for clock release
  RETURN


' *** Transmit Byte ***

I2C_TX_Byte:
  SHIFTOUT SDA, SCL, MSBFIRST, [i2cWork\8]       ' send byte to device
  SHIFTIN SDA, SCL, MSBPRE, [i2cAck\1]           ' get acknowledge bit
  RETURN


' *** Receive Byte ***

I2C_RX_Byte_Nak:
  i2cAck = Nak                                   ' no Ack = high
  GOTO I2C_RX
```

```
I2C_RX_Byte:
  i2cAck = Ack                                  ' Ack = low

I2C_RX:
  SHIFTIN SDA, SCL, MSBPRE, [i2cWork\8]         ' get byte from device
  SHIFTOUT SDA, SCL, LSBFIRST, [i2cAck\1]       ' send ack or nak
  RETURN


' *** Stop Sequence ***

I2C_Stop:                                       ' I2C stop bit sequence
  LOW SDA
  INPUT SCL
  INPUT SDA
  RETURN
```

```
' =============================================================================
'
'   File....... DS1621.BS2
'   Purpose.... DS1621 demo for BS2/BS2e/BS2sx
'   Author..... Jon Williams, Parallax
'   E-mail..... jwilliams@parallax.com
'   Started....
'   Updated.... 07 SEP 2004
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =============================================================================


' -----[ Program Description ]---------------------------------------------


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


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

SDA             PIN     0                       ' I2C serial data line
SCL             PIN     1                       ' I2C serial clock line


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

DS1621          CON     %1001 << 4              ' Device type

Ack             CON     0                       ' acknowledge bit
Nak             CON     1                       ' no ack bit

RdTemp          CON     $AA                     ' read temperature
RdCntr          CON     $A8                     ' read counter
RdSlope         CON     $A9                     ' read slope
StartC          CON     $EE                     ' start conversion
StopC           CON     $22                     ' stop conversion
AccTH           CON     $A1                     ' access high temp limit
AccTL           CON     $A2                     ' access low temp limit
AccCfg          CON     $AC                     ' access config register

TempHi          CON     25                      ' 25C = ~77F
TempLo          CON     22                      ' 25C = ~72F

DegSym          CON     176                     ' degrees symbol


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

```
slvAddr         VAR     Byte                    ' I2C slave address
devNum          VAR     Nib                     ' device number (0 - 7)
addrLen         VAR     Nib                     ' bytes in word addr (0 - 2)
wrdAddr         VAR     Word                    ' word address

i2cData         VAR     Byte                    ' data to/from device
i2cWork         VAR     Byte                    ' work byte for TX routine
i2cAck          VAR     Bit                     ' Ack bit from device

tempIn          VAR     Word                    ' raw temp from DS1621
sign            VAR     tempIn.BIT8             ' - sign (after alignment)
halfC           VAR     tempIn.BIT0             ' half-degree C bit
tempC           VAR     Word                    ' temp in Celsius
tempF           VAR     Word                    ' temp in Fahrenheit


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


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

Reset:
  #IF ($stamp >= BS2P) #THEN
    #ERROR "Use I2COUT and I2CIN!"
  #ENDIF

  devNum = %000                                 ' chip select (%000 - %111)
  slvAddr = DS1621 | (devNum << 1)              ' setup slave ID

Setup:
  addrLen = 1
  wrdAddr = AccCfg
  i2cData = %1010
  GOSUB Write_Byte                              ' set TOut = active high
  PAUSE 10                                      ' allow EE write
  addrLen = 0
  i2cData = StartC
  GOSUB Write_Byte                              ' start conversions

Set_Thermostat:
  addrLen = 1
  wrdAddr = AccTH
  i2cData = TempHi
  GOSUB Write_Byte                              ' set high threshold
  wrdAddr = AccTL
  i2cData = TempLo
  GOSUB Write_Byte                              ' set low threshold


Demo_Screen:
  DEBUG CLS,
```

```
        "DS1621 Demo", CR,
        "-----------", CR,
        DegSym, "C... ", CR,
        DegSym, "F... "


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

Main:
  DO
    PAUSE 1000                                  ' delay between reads
    GOSUB Get_Temp                              ' get current temperature
    DEBUG CRSRXY, 6, 2, SDEC tempC, CLREOL,     ' display
          CRSRXY, 6, 3, SDEC tempF, CLREOL
  LOOP
  END


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

Get_Temp:
  GOSUB I2C_Start                               ' send Start
  i2cWork = slvAddr & %11111110                 ' send slave ID (write)
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Get_Temp               ' wait until not busy
  i2cWork = RdTemp                              ' send read temp command
  GOSUB I2C_TX_Byte
  GOSUB I2C_Start
  i2cWork = slvAddr | %00000001                 ' send slave ID (read)
  GOSUB I2C_TX_Byte
  GOSUB I2C_RX_Byte
  tempIn.BYTE1 = i2cWork                        ' get temp MSB
  GOSUB I2C_RX_Byte_Nak
  tempIn.BYTE0 = i2cWork                        ' get temp LSB
  GOSUB I2C_Stop

  tempIn = tempIn >> 7                          ' correct bit alignment
  ' Celsius
  tempC = (tempIn / 2) | ($FF00 * sign)
  ' Fahrenheit
  tempF = (tempIn | ($FF00 * sign)) + 110       ' convert to absolute T
  tempF = tempF * 9 / 10 - 67                   ' convert to F
  RETURN


' =====[ High Level I2C Subroutines]=========================================

' Random location write
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
```

```
' -- data byte to be written is passed in "i2cData"

Write_Byte:
  GOSUB I2C_Start                                ' send Start
  i2cWork = slvAddr & %11111110                  ' send slave ID (write)
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Write_Byte              ' wait until not busy
  IF (addrLen > 0) THEN
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                    ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                      ' send word address (0)
    GOSUB I2C_TX_Byte
  ENDIF
  i2cWork = i2cData                              ' send data
  GOSUB I2C_TX_Byte
  GOSUB I2C_Stop
  RETURN


' Random location read
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte read is returned in "i2cData"

Read_Byte:
  GOSUB I2C_Start                                ' send Start
  IF (addrLen > 0) THEN
    i2cWork = slvAddr & %11111110                ' send slave ID (write)
    GOSUB I2C_TX_Byte
    IF (i2cAck = Nak) THEN Read_Byte             ' wait until not busy
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                    ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                      ' send word address (0)
    GOSUB I2C_TX_Byte
    GOSUB I2C_Start
  ENDIF
  i2cWork = slvAddr | %00000001                  ' send slave ID (read)
  GOSUB I2C_TX_Byte
  GOSUB I2C_RX_Byte_Nak
  GOSUB I2C_Stop
  i2cData = i2cWork
  RETURN


' -----[ Low Level I2C Subroutines ]-------------------------------------
```

```
' *** Start Sequence ***

I2C_Start:                                      ' I2C start bit sequence
  INPUT SDA
  INPUT SCL
  LOW SDA

Clock_Hold:
  DO : LOOP UNTIL (SCL = 1)                     ' wait for clock release
  RETURN


' *** Transmit Byte ***

I2C_TX_Byte:
  SHIFTOUT SDA, SCL, MSBFIRST, [i2cWork\8]      ' send byte to device
  SHIFTIN SDA, SCL, MSBPRE, [i2cAck\1]          ' get acknowledge bit
  RETURN


' *** Receive Byte ***

I2C_RX_Byte_Nak:
  i2cAck = Nak                                  ' no Ack = high
  GOTO I2C_RX

I2C_RX_Byte:
  i2cAck = Ack                                  ' Ack = low

I2C_RX:
  SHIFTIN SDA, SCL, MSBPRE, [i2cWork\8]         ' get byte from device
  SHIFTOUT SDA, SCL, LSBFIRST, [i2cAck\1]       ' send ack or nak
  RETURN


' *** Stop Sequence ***

I2C_Stop:                                       ' I2C stop bit sequence
  LOW SDA
  INPUT SCL
  INPUT SDA
  RETURN
```

```
' =============================================================================
'
'   File....... PCF8574A.BS2
'   Purpose.... PCF8574/PCF8574A demo with a BS2/BS2e/BS2sx
'   Author..... Jon Williams, Parallax
'   E-mail..... jwilliams@parallax.com
'   Started....
'   Updated.... 07 SEP 2004
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =============================================================================


' -----[ Program Description ]---------------------------------------------
'
' This program reads and displays the PCF8574A pins P4 - P7 while
' displaying a running counter on PCF8574A pins P0 - P3.
'
' Special Note: When reading inputs while using the PCF8574A 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.
'
' I/O Notes:
'
' The input bit is pulled up to Vdd (+5) through 10K.  This input is con-
' nected to Vss (ground) through a N.O. pushbutton switch.  The input will
' read 1 when the switch is open, 0 when pressed.
'
' PCF8574A can sink current, but provide almost no source current.  Outputs
' for this program are setup as active-low.  The tilde (~) in front of
' variables inverts the bits since the PCF8574A uses active-low I/O.


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


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

SDA             PIN     0                       ' I2C serial data line
SCL             PIN     1                       ' I2C serial clock line


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

Ack             CON     0                       ' acknowledge bit
Nak             CON     1                       ' no ack bit

PCF8574         CON     %0100 << 4
PCF8574A        CON     %0111 << 4
```

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


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

slvAddr         VAR     Byte                    ' slave address
devNum          VAR     Nib                     ' device number (0 - 7)
addrLen         VAR     Nib                     ' 0, 1 or 2
devAddr         VAR     Word                    ' address in device

i2cData         VAR     Byte                    ' data to/from device
i2cWork         VAR     Byte                    ' work byte for TX routine
i2cAck          VAR     Bit                     ' Ack bit from device

cntr            VAR     Nib                     ' counter


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


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

Reset:
  #IF ($stamp >= BS2P) #THEN
    #ERROR "Use I2COUT and I2CIN!"
  #ENDIF

  devNum = 0                                    ' device address %000
  slvAddr = PCF8574A | (devNum << 1)            ' setup slave ID
  addrLen = 0                                   ' no internal addresses

  DEBUG CLS,
        "PCF8574A Demo"                         ' setup output screen


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

Main:
  DO
    FOR cntr = 0 TO 15                          ' loop through 4-bit count
      i2cData = ~cntr << 4 | MixDDR             ' create output byte
      GOSUB Write_Byte                          ' update LEDs
      GOSUB Read_Byte                           ' read switches

      ' report
      DEBUG CRSRXY, 0, 2, "In.... ", BIN4 ~i2cData.LOWNIB
      DEBUG CRSRXY, 0, 3, "Out... ", BIN4 cntr
      PAUSE 100
    NEXT
  LOOP
```

```
  END


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


' -----[ High Level I2C Subroutines]-----------------------------------------

' Random location write
' -- pass device slave address in "slvAddr"
' -- pass address bytes (0, 1 or 2) in "addrLen"
' -- register address passed in "devAddr"
' -- data byte to be written is passed in "i2cData"

Write_Byte:
  GOSUB I2C_Start                               ' send Start
  i2cWork = slvAddr & %11111110                 ' send slave ID
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Write_Byte             ' wait until not busy
  IF (addrLen > 0) THEN
    IF (addrLen = 2) THEN
      i2cWork = devAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = devAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
  ENDIF
  i2cWork = i2cData                             ' send data
  GOSUB I2C_TX_Byte
  GOSUB I2C_Stop
  RETURN


' Random location read
' -- pass device slave address in "slvAddr"
' -- pass address bytes (0, 1 or 2) in "addrLen"
' -- register address passed in "devAddr"
' -- data byte read is returned in "i2cData"

Read_Byte:
  GOSUB I2C_Start                               ' send Start
  IF (addrLen > 0) THEN
    i2cWork = slvAddr & %11111110               ' send slave ID (write)
    GOSUB I2C_TX_Byte
    IF (i2cAck = Nak) THEN Read_Byte            ' wait until not busy
    IF (addrLen = 2) THEN
      i2cWork = devAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = devAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
```

```
    GOSUB I2C_Start
  ENDIF
  i2cWork = slvAddr | %00000001                 ' send slave ID (read)
  GOSUB I2C_TX_Byte
  GOSUB I2C_RX_Byte_Nak
  GOSUB I2C_Stop
  i2cData = i2cWork
  RETURN


' -----[ Low Level I2C Subroutines ]-------------------------------------

' *** Start Sequence ***

I2C_Start:                                      ' I2C start bit sequence
  INPUT SDA
  INPUT SCL
  LOW SDA

Clock_Hold:
  DO : LOOP UNTIL (SCL = 1)                     ' wait for clock release
  RETURN


' *** Transmit Byte ***

I2C_TX_Byte:
  SHIFTOUT SDA, SCL, MSBFIRST, [i2cWork\8]      ' send byte to device
  SHIFTIN SDA, SCL, MSBPRE, [i2cAck\1]          ' get acknowledge bit
  RETURN


' *** Receive Byte ***

I2C_RX_Byte_Nak:
  i2cAck = Nak                                  ' no Ack = high
  GOTO I2C_RX

I2C_RX_Byte:
  i2cAck = Ack                                  ' Ack = low

I2C_RX:
  SHIFTIN SDA, SCL, MSBPRE, [i2cWork\8]         ' get byte from device
  SHIFTOUT SDA, SCL, LSBFIRST, [i2cAck\1]       ' send ack or nak
  RETURN


' *** Stop Sequence ***

I2C_Stop:                                       ' I2C stop bit sequence
  LOW SDA
```

```
  INPUT SCL
  INPUT SDA
  RETURN
```

```
' ================================================================================
'
'   File....... PCF8591.BS2
'   Purpose.... PCF8591 demo for BS2/BS2e/BS2sx
'   Author..... Jon Williams, Parallax
'   E-mail..... jwilliams@parallax.com
'   Started....
'   Updated.... 10 SEP 2004
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' ================================================================================


' -----[ 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.  Channels 1
' and 2 are pots (0 - 5v input).  Channe 3 is tied to Vss.


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


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

SDA             PIN     0                       ' I2C serial data line
SCL             PIN     1                       ' I2C serial clock line


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

Ack             CON     0                       ' acknowledge bit
Nak             CON     1                       ' no ack bit

PCF8591         CON     %1001 << 4              ' device type
EnableD2A       CON     %01000000               ' enable analog output
AutoInc         CON     %00000100               ' auto inc a2d channels

MVPB            CON     $139C                   ' millivolts per bit factor


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

slvAddr         VAR     Byte                    ' I2C slave address
devNum          VAR     Nib                     ' device number (0 - 7)
addrLen         VAR     Nib                     ' bytes in word addr (0 - 2)
wrdAddr         VAR     Word                    ' word address

i2cData         VAR     Byte                    ' data to/from device
```

```
i2cWork         VAR     Byte                    ' work byte for TX routine
i2cAck          VAR     Bit                     ' Ack bit from device

aOut            VAR     Byte                    ' analog output
aIn             VAR     Byte(4)                 ' analog input channels

idx             VAR     Nib


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


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

Check_Module:
  #IF ($STAMP >= BS2P) #THEN
    #ERROR "Use I2COUT and I2CIN!"
  #ENDIF

  devNum = %000                                 ' chip select (%000 - %111)
  slvAddr = PCF8591 | (devNum << 1)             ' setup slave ID
  addrLen = 1                                   ' 1 byte in word address

Setup:
  DEBUG CLS
  DEBUG "PCF8591 Demo", CR,
        "------------", CR,
        "D2A:        ", CR,
        "Ch0:        ", CR,
        "Ch1:        ", CR,
        "Ch2:        ", CR,
        "Ch3:        "


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

Main:
  FOR aOut = 0 TO 255
    DEBUG CRSRXY, 5, 2, DEC3 aOut
    i2cData = aOut
    wrdAddr = EnableD2A | AutoInc
    GOSUB Write_Byte
    GOSUB Read_Analog
    FOR idx = 0 TO 3
      DEBUG CRSRXY, 5, (3 + idx), DEC3 aIn(idx)
    NEXT
    PAUSE 500
  NEXT
  GOTO Main
```

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

' Reads all four analog channels from PCF8591
' -- pass device slave address in "slvAddr"
' -- values returned in aIn() array

Read_Analog:
  GOSUB I2C_Start                               ' send Start
  i2cWork = slvAddr | %00000001                 ' send slave ID (read)
  GOSUB I2C_TX_Byte
  GOSUB I2C_RX_Byte                             ' clear previous conversion
  FOR idx = 0 TO 2
    GOSUB I2C_RX_Byte
    aIn(idx) = i2cWork                          ' read Ch0 - Ch2
  NEXT
  GOSUB I2C_RX_Byte_Nak
  aIn(3) = i2cWork                              ' read Ch3
  GOSUB I2C_Stop
  RETURN


' =====[ High Level I2C Subroutines]=====================================

' Random location write
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte to be written is passed in "i2cData"

Write_Byte:
  GOSUB I2C_Start                               ' send Start
  i2cWork = slvAddr & %11111110                 ' send slave ID (write)
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Write_Byte             ' wait until not busy
  IF (addrLen > 0) THEN
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
  ENDIF
  i2cWork = i2cData                             ' send data
  GOSUB I2C_TX_Byte
  GOSUB I2C_Stop
  RETURN


' Random location read
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
```

```
' -- word address to write passed in "wrdAddr"
' -- data byte read is returned in "i2cData"

Read_Byte:
  GOSUB I2C_Start                                ' send Start
  IF (addrLen > 0) THEN
    i2cWork = slvAddr & %11111110                ' send slave ID (write)
    GOSUB I2C_TX_Byte
    IF (i2cAck = Nak) THEN Read_Byte             ' wait until not busy
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                    ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                      ' send word address (0)
    GOSUB I2C_TX_Byte
    GOSUB I2C_Start
  ENDIF
  i2cWork = slvAddr | %00000001                  ' send slave ID (read)
  GOSUB I2C_TX_Byte
  GOSUB I2C_RX_Byte_Nak
  GOSUB I2C_Stop
  i2cData = i2cWork
  RETURN


' -----[ Low Level I2C Subroutines ]-------------------------------------

' *** Start Sequence ***

I2C_Start:                                       ' I2C start bit sequence
  INPUT SDA
  INPUT SCL
  LOW SDA

Clock_Hold:
  DO : LOOP UNTIL (SCL = 1)                      ' wait for clock release
  RETURN


' *** Transmit Byte ***

I2C_TX_Byte:
  SHIFTOUT SDA, SCL, MSBFIRST, [i2cWork\8]       ' send byte to device
  SHIFTIN SDA, SCL, MSBPRE, [i2cAck\1]           ' get acknowledge bit
  RETURN


' *** Receive Byte ***

I2C_RX_Byte_Nak:
  i2cAck = Nak                                   ' no Ack = high
```

```
  GOTO I2C_RX

I2C_RX_Byte:
  i2cAck = Ack                                  ' Ack = low

I2C_RX:
  SHIFTIN SDA, SCL, MSBPRE, [i2cWork\8]         ' get byte from device
  SHIFTOUT SDA, SCL, LSBFIRST, [i2cAck\1]       ' send ack or nak
  RETURN


' *** Stop Sequence ***

I2C_Stop:                                       ' I2C stop bit sequence
  LOW SDA
  INPUT SCL
  INPUT SDA
  RETURN
```

```
' =================================================================================
'
'   File....... I2C_TEMPLATE.BS2
'   Purpose.... Core I2C routines for BS2/BS2e/BS2sx
'   Author..... Jon Williams, Parallax
'   E-mail..... jwilliams@parallax.com
'   Started....
'   Updated.... 07 SEP 2004
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =================================================================================


' -----[ Program Description ]---------------------------------------------


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


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

SDA             PIN     0                       ' I2C serial data line
SCL             PIN     1                       ' I2C serial clock line


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

Ack             CON     0                       ' acknowledge bit
Nak             CON     1                       ' no ack bit


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

slvAddr         VAR     Byte                    ' I2C slave address
devNum          VAR     Nib                     ' device number (0 - 7)
addrLen         VAR     Nib                     ' bytes in word addr (0 - 2)
wrdAddr         VAR     Word                    ' word address

i2cData         VAR     Byte                    ' data to/from device
i2cWork         VAR     Byte                    ' work byte for TX routine
i2cAck          VAR     Bit                     ' Ack bit from device


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


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

Reset:
```

```
  #IF ($STAMP >= BS2P) #THEN
    #ERROR "Use I2COUT and I2CIN!"
  #ENDIF


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

Main:

  END


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




' =====[ High Level I2C Subroutines]=====================================

' Random location write
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte to be written is passed in "i2cData"

Write_Byte:
  GOSUB I2C_Start                               ' send Start
  i2cWork = slvAddr & %11111110                 ' send slave ID (write)
  GOSUB I2C_TX_Byte
  IF (i2cAck = Nak) THEN Write_Byte             ' wait until not busy
  IF (addrLen > 0) THEN
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
  ENDIF
  i2cWork = i2cData                             ' send data
  GOSUB I2C_TX_Byte
  GOSUB I2C_Stop
  RETURN


' Random location read
' -- pass device slave address in "slvAddr"
' -- pass bytes in word address (0, 1 or 2) in "addrLen"
' -- word address to write passed in "wrdAddr"
' -- data byte read is returned in "i2cData"

Read_Byte:
```

```
  GOSUB I2C_Start                               ' send Start
  IF (addrLen > 0) THEN
    i2cWork = slvAddr & %11111110               ' send slave ID (write)
    GOSUB I2C_TX_Byte
    IF (i2cAck = Nak) THEN Read_Byte            ' wait until not busy
    IF (addrLen = 2) THEN
      i2cWork = wrdAddr.BYTE1                   ' send word address (1)
      GOSUB I2C_TX_Byte
    ENDIF
    i2cWork = wrdAddr.BYTE0                     ' send word address (0)
    GOSUB I2C_TX_Byte
    GOSUB I2C_Start
  ENDIF
  i2cWork = slvAddr | %00000001                 ' send slave ID (read)
  GOSUB I2C_TX_Byte
  GOSUB I2C_RX_Byte_Nak
  GOSUB I2C_Stop
  i2cData = i2cWork
  RETURN


' -----[ Low Level I2C Subroutines ]-------------------------------------

' *** Start Sequence ***

I2C_Start:                                      ' I2C start bit sequence
  INPUT SDA
  INPUT SCL
  LOW SDA

Clock_Hold:
  DO : LOOP UNTIL (SCL = 1)                     ' wait for clock release
  RETURN


' *** Transmit Byte ***

I2C_TX_Byte:
  SHIFTOUT SDA, SCL, MSBFIRST, [i2cWork\8]      ' send byte to device
  SHIFTIN SDA, SCL, MSBPRE, [i2cAck\1]          ' get acknowledge bit
  RETURN


' *** Receive Byte ***

I2C_RX_Byte_Nak:
  i2cAck = Nak                                  ' no Ack = high
  GOTO I2C_RX

I2C_RX_Byte:
  i2cAck = Ack                                  ' Ack = low
```

```
I2C_RX:
  SHIFTIN SDA, SCL, MSBPRE, [i2cWork\8]          ' get byte from device
  SHIFTOUT SDA, SCL, LSBFIRST, [i2cAck\1]        ' send ack or nak
  RETURN


' *** Stop Sequence ***

I2C_Stop:                                        ' I2C stop bit sequence
  LOW SDA
  INPUT SCL
  INPUT SDA
  RETURN
```