Column #91 November 2002 by Jon Williams:

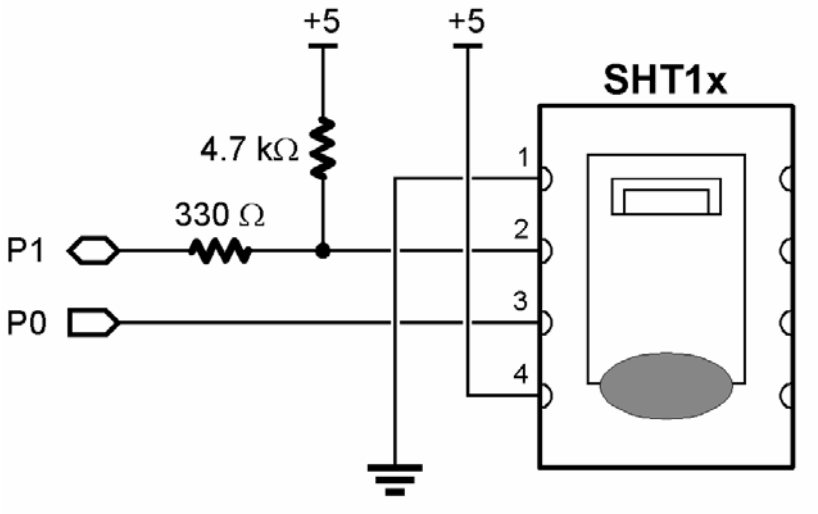
Environmental Sensing

If you're not a regular reader or you're fairly new to Nuts & Volts or my column, you may not know that I am an admitted temperature nut. I don't know why; I've just always been. I'll bet I tweak the thermostat setting in my apartment three or four times a day. And you can bet that I'm looking forward to the opportunity to build my own home – with lots of insulation, of course – and a lot of smart sensors to keep that home comfortable.

On that note of smart sensors ... my boss recently sent me a sample of a new temperature/humidity sensor from Sensirion: the SHT1x. The SHT1x is, indeed, smart and connecting to the BASIC Stamp is a breeze through a two-wire interface that is similar to I2C.

The SHT1x is factory calibrated so that it returns temperature with a resolution of 0.01 degrees Celsius and relative humidity with a resolution of 0.03 percent. The accuracy is better than most other sensors too. Worst-case temperature accuracy is +/- 2 degrees C – but in the "room temperature" range the accuracy better than +/- 1 degree C. The relative humidity sensor is similarly accurate: +/- 3.5% in the range 20% to 80%. This is quite good for a low-cost sensor.

Figure 91.1: SHT1x to BASIC Stamp Connection



Getting Connected

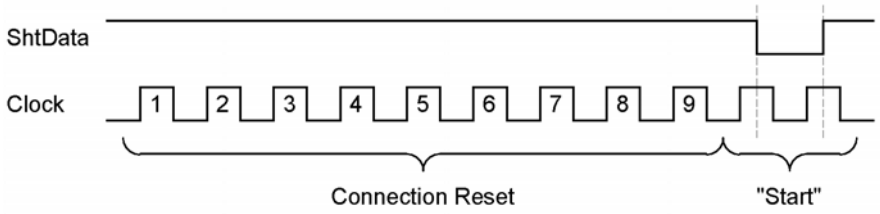
Connecting to the SHT1x straightforward, just a simple-two wire interface that is similar to I2C (see Figure 91.1). The difference is that only the bi-directional data line requires a pull-up. The clock line is directly driven as the SHT1x never affects it (is is the case with I2C). The 330 ohm resistor between the BASIC Stamp and the SHT1x protects both in the event the Stamp is making the data line go high and the SHT1x is attempting to pull the data line low.

SHT1x Protocol

Just as the physical connection is similar to I2C, the SHT1x communication protocol is also similar, but different enough that we probably can't share the SHT1x bus with I2C devices. You'll remember from the I2C code we developed a few months ago that I2C devices require a "start" sequence before addressing the bus. Well, so does SHT1x. Take a look at Figure 91.2.

Before attempting to communicate with the SHT1x for the fist time, it's a good idea to reset its serial interface by running the "connection reset sequence." The sequence consists of nine clock pulses while the data line is high, followed by the "start" sequence which is the data line being pulled low in the middle of one clock pulse and released in the middle of the next.

Figure 91.2: SHT1x Communication Protocol



Let's look at the code – called from the initialization section of our program – that handles the communication reset sequence.

```
SHT Connection Reset:
  SHIFTOUT ShtData, Clock, LSBFirst, [$FFF\9]

SHT_Start:
  INPUT ShtData
  LOW Clock
  HIGH Clock
  LOW ShtData
  LOW Clock
  HIGH Clock
  INPUT ShtData
  LOW Clock
  RETURN
```

The first part of the sequence is easily handled with SHIFTOUT. In this case we've specified $FFF to keep the data line high during the clock pulses and we've also use the \9 parameter to tell the Stamp how many clock pulses to generate.

The next section is also simple; just a bit clunky because of the requirements to lower and raise the data line in the middle of clock pulses. There's no way to do this with SHIFTOUT so we just do it all manually using INPUT, LOW and HIGH. INPUT is used with the data line since the pull-up will take the line high when we make that pin an input. The clock line is directly driven by the BASIC Stamp so we use HIGH and LOW to create the appropriate pulse levels.

You'll also notice that there's a label called SHT_Start in the middle of this code. The start sequence is required before every communication with the SHT1x, so this label provides a convenient entry point to that code.

Figure 91.3: Details of the SHT1x Communication Protocol

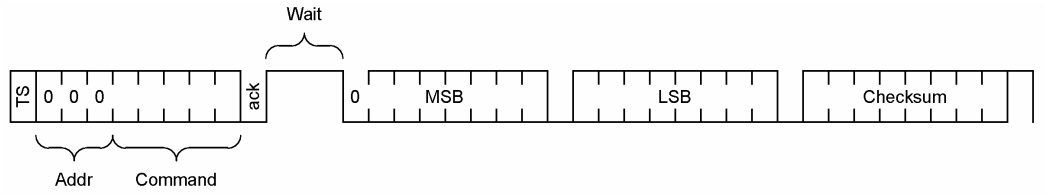


Figure 91.3 shows the details of the SHT1x protocol. After the transmission start (TS), a command is sent to the device. The SHT1x command byte allows for a three-bit address field. Currently, only address %000 is supported, but in the future we'll be able to connect up to eight sensors to the same SHT1x bus.

Commands and data are sent to the SHT1x a byte at a time with this subroutine:

```
SHT Write Byte:
  SHIFTOUT ShtData, Clock, MSBFirst, [ioByte]
  SHIFTIN  ShtData, Clock, LSBPre, [ackBit\1]
  RETURN
```

This is simple code: it clocks eight bits to the SHT1x and then clocks one acknowledge bit back in.

After the acknowledge bit, there will be a delay while the SHT1x processes a read (measurement) command. The end of this delay is signaled by the SHT1x pulling the data line low. Here's the code that monitors the data line:

```
SHT Wait:
  INPUT ShtData
  FOR toDelay = 1 TO 250
    timeOut = Ins.LowBit(ShtData)
    IF (timeOut = No) THEN SHT Wait Done
    PAUSE 1
  NEXT

SHT Wait Done:
  RETURN
```

Since the SHT1x spec says that a 14-bit measurement should be made in no more than 210 milliseconds (+15%), this code uses a FOR-NEXT loop with an embedded delay. If the entire loop runs, it would be something greater than 250 milliseconds since scanning the data line does add to the loop timing.

In the loop we will grab the state of the data line and store it in timeOut. If the data line goes low before the loop ends, the IF-THEN logic will break us out. The constant "No" has been defined as having a value of zero, "Yes" having a value of one. If the data line does not go low before the end of the loop, the code will exit with timeOut containing the value one ("Yes"). Unless something goes wrong, you probably won't see this happen.

Now that the SHT1x has taken a measurement, we can retrieve the result. Again, this is done one byte at a time.

```
SHT Read Byte:
  SHIFTIN  ShtData, Clock, MSBPre, [ioByte]
  SHIFTOUT ShtData, Clock, LSBFirst, [ackBit\1]
  INPUT ShtData
  RETURN
```

As you can see, SHT_Read_Byte is the compliment of SHT_Write_Byte. This time we will read eight bits from the SHT1x, then output an acknowledge bit. And, just like I2C, the acknowledge bit will be zero (low) until the last read; then it will be one (high). Note that we must manually release the data line since the SHIFTOUT for the acknowledge bit makes the data line an output.

Let's put it all together and read the temperature from the SHT1x.

```
SHT Measure Temp:
  GOSUB SHT Start
  ioByte = ShtTemp
  GOSUB SHT_Write_Byte
  GOSUB SHT Wait
  ackBit = Ack
  GOSUB SHT Read Byte
  soT.HighByte = ioByte
  ackBit = NoAck
  GOSUB SHT_Read_Byte
  soT.LowByte = ioByte
```

This subroutine begins by calling the "start" sequence, then sending the ShtTemp (read temperature; %00011) command. This is followed by the wait and finally reading two bytes from the SHT1x. The temperature data is returned as a 14-bit value, with the MSB sent first.

The value retuned by the SHT1x is converted to standard units with the following formulas:

Celsius = soT x 0.01 – 40
Fahrenheit = soT x 0.018 – 40

Uh oh ... here we go with those dastardly fractional numbers again – and the Stamp only does integer math. It's okay, we've got that neat ** operator in our bag of tricks. Remember, the ** operator allows us to multiply by a fractional value less than 1.0 in units of 1/65536.

Let's look at the end of the SHT_Measure_Temp subroutine and then go through it.

```
tC = soT / 10  - 400
tF = soT ** 11796 - 400
RETURN
```

Since the Stamp does do integer math, we'll start by multiplying the formula conversion factors by 10 so that our temperature values are expressed in tenths of a degree. Here are the formulas, converted for our PBASIC code:

Celsius (tenths) = soT * 0.1 – 400
Fahrenheit (tenths) = soT * 0.18 – 400

Multiplying by 0.1 is the same as dividing by ten, so we'll kept the Celsius equation simple. Converting the fractional value for Fahrenheit works like this:

0.18 x 65536 = 11796

As you can see, that was actually quite painless and we end up with an accurate temperature reading. Let's move on to humidity:

```
SHT_Measure_Humidity:
  GOSUB SHT Start
  ioByte = ShtHumi
  GOSUB SHT Write Byte
  GOSUB SHT Wait
  ackBit = Ack
  GOSUB SHT_Read_Byte
  soRH.HighByte = ioByte
  ackBit = NoAck
  GOSUB SHT Read Byte
  soRH.LowByte = ioByte
```

The process for reading humidity from the SHT1x is identical to reading the temperature. What will be returned is a 12-bit relative humidity value. The tricky part comes in the conversion to a usable value as the humidity output is slightly non-linear. Here's the formula for converting the sensor output to relative humidity:

RH = (soRH x 0.0405) – (soRH2 x 0.0000028) – 4

This formula presents a couple of problems in the second section. First, squaring the raw output value will almost always cause a roll-over (greater than 65535) error and the constant value in that section is very small.

The way we can work around this is to multiply our sensor output by the square root of the constant. Here's how we break-up that tricky middle part:

soRH2 x 0.0000028 à soRH x 0.0017 * soRH x 0.0017

If we multiply our constant factor by ten, then take its square root, we get this:

soRH x 0.00529 * soRH x 0.00529 à (soRH ** 346) * (soRH ** 347)

We can help ourselves even more by multiplying the conversion factors by ten (again), then applying a bit of round-off math like we used to do in school. This ends up giving us the best accuracy using this simplified approach:

```
rhLin = (soRH ** 26542)
rhLin = rhLin - ((soRH ** 3468) * (soRH ** 3468) + 50 / 100)
rhLin = rhLin – 40
```

The SHT1x specification states that for temperatures "significantly different" from 25ºC (77ºF) the temperature coefficient of the RH sensor should be considered. The formula for temperature compensating the sensor is given as:

RHTC = (TC – 25) x (SORH x 0.00008 + 0.01) + RHLIN

Again, we've got very small fractional numbers so what we'll do is multiply by 100. For the middle section:

0.00008 x 100 = 0.008 à 0.008 * 65536 = 524
0.01 x 100 = 1

The temperature is expressed in tenths so we'll end up dividing it down to whole units to prevent a roll-over error when multiplied by the middle section. Now that the first part of the equation is in 100ths, we have to multiply the rhLin by ten to match before we add the two values together. Finally, we do the rounding by adding five and dividing the final result by ten to keep the rtTrue value in tenths of a percent.

Here's what it looks like:

```
rhTrue = ((tC / 10 - 25) * (soRH ** 524 + 1)
rhTrue = rhTrue + (rhLin * 10)) + 5 / 10
RETURN
```

Yes, I know that was a little crazy, but in the end, it works – and works pretty well. Getting used to math like this will be helpful as we begin to explore other types of sensors, especially those with a nonlinear output. Okay, now that we can read the SHT1x and convert its output values to usable units, it's time to show it off.

Our little demo program uses a terminal window to display data so we can easily verify that the sensor is working. But what about an embedded application where the temperature and humidity may not change rapidly? How can our application check the sensor?

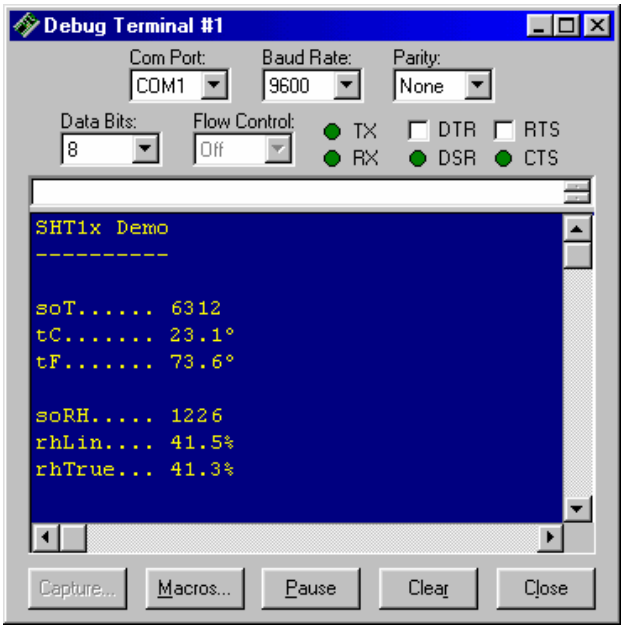
Well, the folks at Sesirion thought about that and actually built a small heater into the SHT1x. The device can be tested by taking readings, turning the heater on for a few seconds, then taking another set of readings. The temperature should go up and the humidity down. The heater is also useful in very moist environments to prevent condensation from forming on the sensing element.

To turn on the heater, we must change a bit in the SHT1x status register. The following code is from the top part of the demo:

```
Heater On:
  DEBUG "SHT1x heater on", CR
  status = %00000100
  GOSUB SHT Write Status
  DEBUG "Waiting 2 seconds", CR
  PAUSE 2000

Heater Off:
  DEBUG "SHT1x heater off", CR, CR
  status = %00000000
  GOSUB SHT_Write_Status
```

Figure 91.4: Example Output with DEBUG



As you can see by the code, when bit two of the status byte is set, the heater will be turned on. You want to be careful not to leave the heater on for very long, and it doesn't take long at all to see a significant change in the output values.

Here's the code that sends the status byte to the SHT1x:

```
SHT Write Status:
  GOSUB SHT Start
  ioByte = ShtStatW
  GOSUB SHT Write Byte
  ioByte = status
  GOSUB SHT_Write_Byte
  RETURN
```

Writing to the status register is a two-byte sequence: first the command, then the data to write. The command tells the SHT1x that the following byte is the status byte and it is written into the SHT1x. There is a complimentary subroutine in the code that reads the status byte. It's there for future use, but not actually required by the demo. Another bit in the status byte allows us to

change the SHT1x output resolution: eight bits for humidity and 12 bits for temperature. There's really no benefit to doing this in Stamp-based applications.

After the sensor check demonstration, the program continuously loops and displays the sensor values as shown in Figure 91.4.

It's In The Packaging

I know that a few of you are thinking, "Great, this is a cool sensor, but how am I going to connect a surface mount device to my BASIC Stamp?" Don't worry, it's being taken care of. Parallax likes this little sensor so much we're making an adaptor board for it that even includes the resistors. The package will end up in an eight-pin DIP format – about the same size as a 555 timer IC. Check the Parallax web site for details.

Credit Where It's Due

I really have to thank my pal, Tracy Allen, for his help with the SHT1x math and converting the equations to PBASIC. If you've never been to Tracy's web site, you owe yourself a visit as Tracy provides more solid technical information on PBASIC programming than at any other site I've seen.

Check it out at www.emesystems.com. If you thought the math we did here was tricky, wait until you see what Tracy has to offer. In a word ... Wow. He also has excellent explanations of those pesky ** and */ operators and how to get the most from them. When you visit Tracy's site, be sure to set a bookmark in your browser since his site is worthy of frequent visits.

Next month we're going to work with another sensor – an accelerometer. Until then, Happy Stamping.

```
' =============================================================================
'
'   Program Listing 91.1
'   File...... SHT1x.BS2
'   Purpose... Interface to Sensirion SHT1x temperature/humidity sensor
'   Author.... Jon Williams
'   E-mail.... jwilliams@parallaxinc.com
'   Started...
'   Updated... 28 SEP 2002
'
'   {$STAMP BS2}
'
' =============================================================================


' -----------------------------------------------------------------------------
' Program Description
' -----------------------------------------------------------------------------
'
' This program demonstrates the interface and conversion of SHT1x data to
' usable program values.  This program takes advantage of the PBASIC **
' operator to multiply by a fractional value.
'
' For detailed information on the use and application of the ** operator,
' see Tracy Allen's web page at this link:
'
' -- http://www.emesystems.com/BS2math1.htm
'
' For Tracy's SHT1x code [very advanced]:
'
' -- http://www.emesystems.com/OL2sht1x.htm
'
' For SHT11/15 documentation and app notes, visit:
'
' -- http://www.sensirion.com


' -----------------------------------------------------------------------------
' Revision History
' -----------------------------------------------------------------------------


' -----------------------------------------------------------------------------
' I/O Definitions
' -----------------------------------------------------------------------------

ShtData         CON     1                       ' bi-directional data
Clock           CON     0
```

```
' -----------------------------------------------------------------------------
' Constants
' -----------------------------------------------------------------------------

ShtTemp         CON     %00011                  ' read temperature
ShtHumi         CON     %00101                  ' read humidity
ShtStatW        CON     %00110                  ' status register write
ShtStatR        CON     %00111                  ' status register read
ShtReset        CON     %11110                  ' soft reset (wait 11 ms after)

Ack             CON     0
NoAck           CON     1

No              CON     0
Yes             CON     1

MoveTo          CON     2                       ' for DEBUG control
ClrRt           CON     11                      ' clear DEBUG line to right

DegSym          CON     186                     ' degrees symbol for DEBUG


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

ioByte          VAR     Byte                    ' data from/to SHT1x
ackBit          VAR     Bit                     ' ack/nak from/to SHT1x
toDelay         VAR     Byte                    ' timeout delay timer
timeOut         VAR     Bit                     ' timeout status

soT             VAR     Word                    ' temp counts from SHT1x
tC              VAR     Word                    ' temp - celcius
tF              VAR     Word                    ' temp - fahrenheit

soRH            VAR     Word                    ' humidity counts from SHT1x
rhLin           VAR     Word                    ' humidity; linearized
rhTrue          VAR     Word                    ' humidity; temp compensated

status          VAR     Byte                    ' SHT1x status byte


' -----------------------------------------------------------------------------
' EEPROM Data
' -----------------------------------------------------------------------------


' -----------------------------------------------------------------------------
' Initialization
' -----------------------------------------------------------------------------
```

```
Initialize:
  GOSUB SHT Connection Reset                    ' reset device connection

  PAUSE 250                                     ' let DEBUG window open
  DEBUG CLS
  DEBUG "SHT1x Demo", CR
  DEBUG "----------", CR

  ' GOTO Main                                   ' skip heater demo

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

Sensor Demo:
  GOSUB SHT Measure Temp
  DEBUG MoveTo, 0, 3
  DEBUG "tF..... "
  DEBUG DEC (tF / 10), ".", DEC1 tF, DegSym, ClrRt, CR

  GOSUB SHT Measure Humidity
  DEBUG "rhLin... "
  DEBUG DEC (rhLin / 10), ".", DEC1 rhLin, "%", ClrRt, CR, CR

Heater_On:
  DEBUG "SHT1x heater on", CR
  status = %00000100                            ' heater bit = On
  GOSUB SHT Write Status
  DEBUG "Waiting 2 seconds", CR
  PAUSE 2000

Heater Off:
  DEBUG "SHT1x heater off", CR, CR
  status = %00000000                            ' heater bit = Off
  GOSUB SHT Write Status

  GOSUB SHT_Measure_Temp
  DEBUG "tF..... "
  DEBUG DEC (tF / 10), ".", DEC1 tF, DegSym, ClrRt, CR

  GOSUB SHT Measure Humidity
  DEBUG "rhLin... "
  DEBUG DEC (rhLin / 10), ".", DEC1 rhLin, "%", ClrRt, CR, CR

  PAUSE 5000

Main:
  DEBUG CLS
  DEBUG "SHT1x Demo", CR
  DEBUG "----------", CR
```

```
Main2:
  GOSUB SHT Measure Temp
  DEBUG MoveTo, 0, 3
  DEBUG "soT..... "
  DEBUG DEC soT, ClrRt, CR
  DEBUG "tC..... "
  DEBUG DEC (tC / 10), ".", DEC1 tC, DegSym, ClrRt, CR
  DEBUG "tF..... "
  DEBUG DEC (tF / 10), ".", DEC1 tF, DegSym, ClrRt

  GOSUB SHT Measure Humidity
  DEBUG MoveTo, 0, 7
  DEBUG "soRH..... "
  DEBUG DEC soRH, ClrRt, CR
  DEBUG "rhLin.... "
  DEBUG DEC (rhLin / 10), ".", DEC1 rhLin, "%", ClrRt, CR
  DEBUG "rhTrue... "
  DEBUG DEC (rhTrue / 10), ".", DEC1 rhTrue, "%", ClrRt

  PAUSE 1000                                    ' minimum delay between readings
  GOTO Main2

  END


' -----------------------------------------------------------------------------
' Subroutines
' -----------------------------------------------------------------------------

' connection reset: 9 clock cyles with ShtData high, then start sequence
'
SHT Connection Reset:
  SHIFTOUT ShtData, Clock, LSBFirst, [$FFF\9]

' generates SHT1x "start" sequence
'
'          _____         _____
' ShtData       |_______|
'
' Clock       |   |   |   |
'
SHT Start:
  INPUT ShtData                                 ' let pull-up take line high
  LOW Clock
  HIGH Clock
  LOW ShtData
  LOW Clock
  HIGH Clock
  INPUT ShtData
  LOW Clock
  RETURN
```

```
' measure temperature
' -- celcius = soT * 0.01 - 40
' -- fahrenheit = soT * 0.018 - 40
'
SHT_Measure_Temp:
  GOSUB SHT_Start                               ' alert device
  ioByte = ShtTemp                              ' temperature command
  GOSUB SHT Write Byte                          ' send command
  GOSUB SHT Wait                                ' wait until measurement done
  ackBit = Ack                                  ' another read follows
  GOSUB SHT_Read_Byte                           ' get MSB
  soT.HighByte = ioByte
  ackBit = NoAck                                ' last read
  GOSUB SHT Read Byte                           ' get LSB
  soT.LowByte = ioByte

  ' Note: Conversion factors are multiplied by 10 to return the
  '       temperature values in tenths of degrees

  tC = soT / 10 - 400                           ' convert to tenths C
  tF = soT ** 11796 - 400                       ' convert to tenths F
  RETURN


' measure humidity
'
SHT Measure Humidity:
  GOSUB SHT Start                               ' alert device
  ioByte = ShtHumi                              ' humidity command
  GOSUB SHT_Write_Byte                          ' send command
  GOSUB SHT Wait                                ' wait until measurement done
  ackBit = Ack                                  ' another read follows
  GOSUB SHT Read Byte                           ' get MSB
  soRH.HighByte = ioByte
  ackBit = NoAck                                ' last read
  GOSUB SHT_Read_Byte                           ' get LSB
  soRH.LowByte = ioByte

  ' linearize humidity
  '   rhLin = (soRH * 0.0405) - (soRH^2 * 0.0000028) - 4
  '
  ' for the BASIC Stamp:
  '   rhLin = (soRH * 0.0405) - (soRH * 0.004 * soRH * 0.0007) - 4
  '
  ' Conversion factors are multiplied by 10 and then rounded to
  ' return tenths
  '
  rhLin = (soRH ** 26542)
  rhLin = rhLin - ((soRH ** 3468) * (soRH ** 3468) + 50 / 100)
  rhLin = rhLin - 40
```

```
  ' temperature compensated humidity
  '   rhTrue = (tC - 25) * (soRH * 0.00008 + 0.01) + rhLin
  '
  ' Conversion factors are multiplied by 100 to improve accuracy and then
  ' rounded off.
  '
  rhTrue = ((tC / 10 - 25) * (soRH ** 524 + 1) + (rhLin * 10)) + 5 / 10
  RETURN


' sends "status"
'
SHT Write Status:
  GOSUB SHT Start                               ' alert device
  ioByte = ShtStatW                             ' write to status reg command
  GOSUB SHT Write Byte                          ' send command
  ioByte = status
  GOSUB SHT_Write_Byte
  RETURN


' returns "status"
'
SHT_Read_Status:
  GOSUB SHT Start                               ' alert device
  ioByte = ShtStatW                             ' write to status reg command
  GOSUB SHT Read Byte                           ' send command
  ackBit = NoAck                                ' only one byte to read
  GOSUB SHT_Read_Byte
  RETURN


' sends "ioByte"
' returns "ackBit"
'
SHT_Write_Byte:
  SHIFTOUT ShtData, Clock, MSBFirst, [ioByte]   ' send byte
  SHIFTIN  ShtData, Clock, LSBPre, [ackBit\1]   ' get ack bit
  RETURN


' returns "ioByte"
' sends "ackBit"
'
SHT Read Byte:
  SHIFTIN  ShtData, Clock, MSBPre, [ioByte]     ' get byte
  SHIFTOUT ShtData, Clock, LSBFirst, [ackBit\1] ' send ack bit
  INPUT ShtData                                 ' release data line
  RETURN
```

```
' wait for device to finish measurement (pulls data line low)
' -- timeout after ~1/4 second
'
SHT Wait:
  INPUT ShtData                                 ' data line is input
  FOR toDelay = 1 TO 250                        ' give ~1/4 second to finish
    timeOut = Ins.LowBit(ShtData)               ' scan data line
    IF (timeOut = No) THEN SHT Wait Done        ' if low, we're done
    PAUSE 1
  NEXT

SHT_Wait_Done:
  RETURN


' reset SHT1x with soft reset
'
SHT_Soft_Reset:
  GOSUB SHT Connection Reset                    ' reset the connection
  ioByte = ShtReset                             ' reset command
  ackBit = NoAck                                ' only one byte to send
  GOSUB SHT Write Byte                          ' send it
  PAUSE 11                                      ' wait at least 11 ms
  RETURN
```