Column #114 October 2004 by Jon Williams:

Measuring Up – Up to 80 Centimeters, That Is

Add a bit of intelligence to your Halloween displays with IR distance measuring.

The night is drawing closer ... my favorite night of the whole year: Halloween. I love Halloween; the costumes, haunted houses, parties, friendly exchanges with trick-or-treaters – Halloween is the best. When I have the chance something I like to do is build Halloween-oriented props and decorations, and when I do you can bet that many of those props get some sort of automation via the BASIC Stamp microcontroller.

Good Halloween props add an element of surprise which, of course, intensifies the fright – and that's the most fun about Halloween, right? The only problem is that as a society we are far more sophisticated than in the past (especially the teenagers). We can easily see through a cheesy effect and find the trigger, which ruins the effect for those that immediately follow.

Instead of using a fixed-point trigger for an automated prop, what if we used a distance measuring device so that we could select a random trigger point? That would keep 'em guessing, wouldn't it? You bet. We've used sonic measuring devices in the past (SRF-04 and SRF-08), this time we'll do it with infrared. The device we're going to use is the low-cost Sharp GP2D12.

Read Volts, Get Distance

There is no great mystery to using the GP2D12: we simply connect it to an appropriate analog-to-digital converter and read the output voltage. The voltage is then converted to distance.

The first part is very easy. For this project we'll use the ADC0831 analog-to-digital converter, a part we've used before and should have no troubles with. In order to simplify the project code, we'll connect the wiper of a mutli-turn pot to the Vref input of the ADC0831 and set this to 2.55 volts. What this does for us is set each output count to be equal to 0.01 volts (255 [max count] divided by 2.55 [Vref] = 0.01 volts / count). Figure 114.1 shows the schematic for the project.

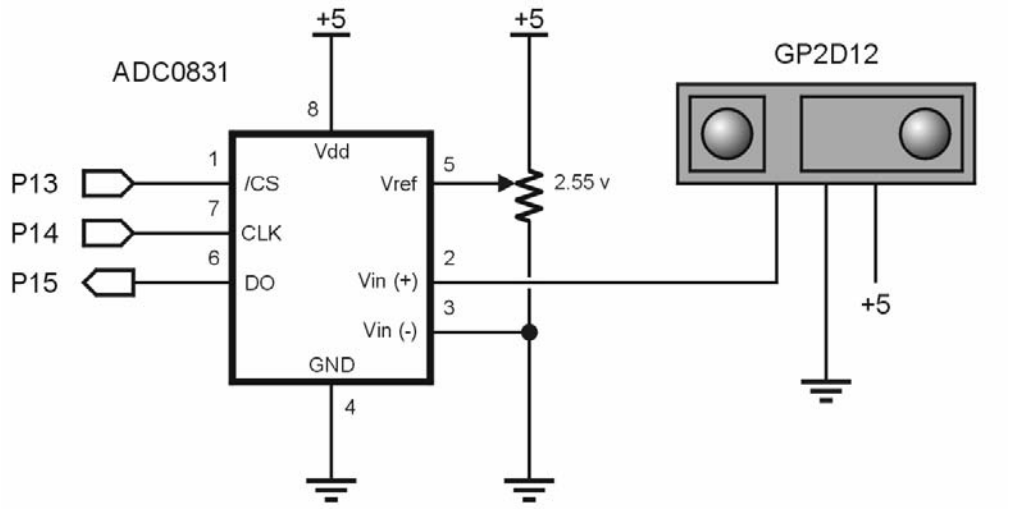

Figure 114.1: ADC0831 to GP2D12 Schematic

Let's have a look at the code that reads the voltage from the ADC0831:

```
Read_0831:
  LOW AdcCS
  SHIFTIN AdcDta, AdcClk, MSBPOST, [result\9]
  HIGH AdcCS
  RETURN
```

This code is straightforward, but if you haven't used the ADC0831 before you may be wondering why we need nine clocks for an eight-bit value. As always, you should download the documentation for any part you're working with, and when you look at the ADC0831 timing chart you'll see that the ADC conversion is started by bringing the CS (chip select) line low, then putting a pulse on the clock line – here's where we get the extra clock pulse. The value bits are clocked out, MSB to LSB, with the following eight clock pulses. Each ADC bit is valid after the falling edge of the clock, so we use MSBPOST to read the bits. Once all the bits are clocked in the device is deselected by bringing the CS line back high.

Okay, that's done, but what we're likely to run into is a bit of jitter in actual application. An easy way to smooth this jitter is to take the average of multiple readings. Let's do it:

```
Read_GP2D12:
  cVolts = 0
  FOR idx = 1 TO 3
    GOSUB Read_0831
    cVolts = cVolts + result
    PAUSE 30
  NEXT
  cVolts = cVolts / 3
  RETURN
```

We start by clearing the old cVolts value and then with a loop, take three readings of the ADC0831 and accumulate them. Keep in mind that we will need to use a Word-sized variable for cVolts, otherwise we'd likely get a roll-over error after the second reading. At the end of the loop we divide the accumulation by the number of loop iterations to get the average value.

What happens, though, when we're in a pinch for variable space? One way around this – though likely to be slightly less accurate than the method above, is to divide each reading before accumuating. Keep in mind that the lower the reading and the larger the divisor means a greater likelihood for error. If you keep the divisor small, this shouldn't become too much of a problem. Here's the code for the alternate version:

```
Read_GP2D12_Alternate:
  cVolts = 0
  FOR idx = 1 TO 3
    GOSUB Read_0831
    cVolts = cVolts + (result / 3)
    PAUSE 30
  NEXT
  RETURN
```

Straightening the Curve

Now comes the tricky part – converting the voltage output of the GP2D12 to a distance value. Have a look at Figure 2 and you'll see why I say this is tricky. Over the entire measurement range, the output from GP2D12 is not at all linear in respect to distance, so a simple mx + b equation is just not going to work. I plugged the data into a curve fitting program and found that it takes a fourth-order equation to get anywhere close to the data set. Applying a fourth-order equation with 16-bit integer-only math is just not very practical.

There are are interesting solutions to this dilema, but most of them were more than I wanted to wrap my brain around so I decided simple is better than interesting (my middle name, afterall, is "Simple"). Looking at the graph again we can see that the segments between data points are not far from the curve that would fit between those same points. What I decided to do then, is to calculate the slope between data points and interpolate from there. I felt like this was an acceptable solution given the slighty loose specifications of the GP2D12 (it is a low-cost device).

First things first – that curve in Figure 114.2 actually came from my sensor bounced of an 18% gray card (something phographers use). Using some cardboard and foam blocks, I setup and marked a test jig at five centimeter intervals, then measured the voltage at each interval from 10 to 80 centimeters using the code we've developed thus far.

Now what? As I just mentioned, the segments between data points can be treated as a line, so what we can do is find the data points that surround our current reading, calculate the slope of the line between them, and then interpolate the distance. Let's have a look at the code that does this then work our way through.

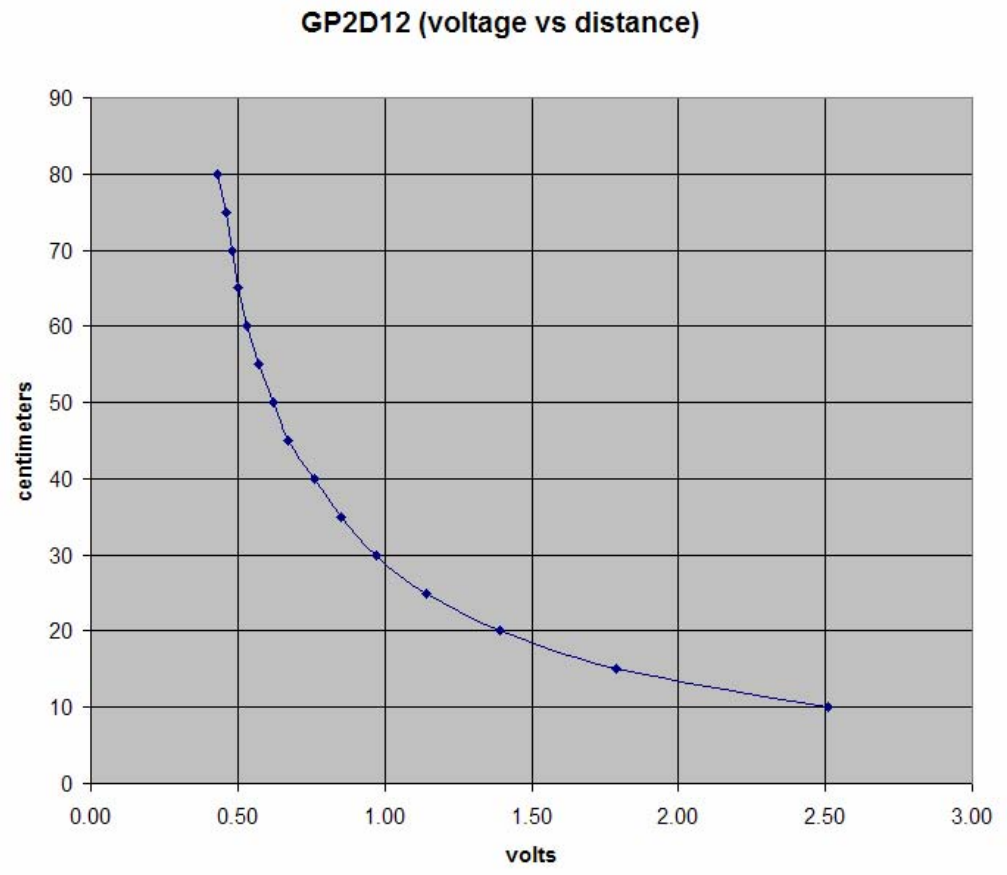

Figure 114.2: GPD212 Distance v. Voltage

Here's the table of distance readings from my test setup:

```
Vout            DATA    251, 179, 139, 114,  97
                DATA     85,  76,  67,  62,  57
                DATA     53,  50,  48,  46,  43
                DATA      0
```

And now the code that uses the table and the current voltage reading:

```
Estimate_Cm:
  FOR idx = 0 TO 15
    READ (Vout + idx), test2
    IF (test2 <= cVolts) THEN EXIT
  NEXT

  SELECT idx
    CASE 0
      cm = 10

    CASE 1 TO 14
      cm = 10 + (5 * idx)
      IF (test2 <> cVolts) THEN
        READ (Vout + idx - 1), test1
        slope = (test1 - test2) * 10 / Xspan
        cm = cm - ((cvolts - test2) * 10 / slope)
      ENDIF

    CASE 15
      cm = 80
  ENDSELECT
  RETURN
```

The first part of the process is locating the position of the current reading vis-à-vis the table values from our test setup. Since the table is very small the simplest method is to loop through the possible values until we find the test point that is less than or equal to our current voltage reading. We can use **EXIT** to terminate the loop early when we find a match.

On the extremes – when idx is either 0 or 15 – we simply set the distance reading to the minimum or maximum values. When I first stared working with the code I tried to provide an "out of range" calculation but the way the output falls on the data points, this just didn't work out very well. So keep this in mind when using the GP2D12 with this code: a reading of 10 cm actually means ten centimeters or less; a reading of 80 cm means 80 centimeters or greater.

Things get interesting when idx is between one and 14. The first step is to calculate the rough distance using idx. Next we check to see if the value of test2 is not equal to cVolts, because if it is we're done and have the distance value in hand. Most of the time, test2 will be less than cVolts so we'll find the other value that borders (is greater than) our current reading and interpolate from there.

At this point we already have the table value lower than cVolts, what we do next is subtract one from idx and read the value that is greater than cVolts – we'll put this value in test1. Now that we have the table values surrounding our input from the GP2D12, we can calculate the slope between them be taking the difference and dividing by the span between these points (5 centimeters in our test data). Since we're doing a division and the values on the outer end of the range get very small, we'll multiply the difference by 10 before dividing. This will prevent us from getting a slope value of zero.

We're almost done. The final step is to divide the difference between our current reading (cVolts) and test2 by the slope, then subtract that from the rough calculation of distance. Again we'll multiply the difference value by 10 – this time to remove the offset introduced by the way we calculated the slope.

Just to make things crystal clear, let's work through a set of numbers. We'll start with an input voltage of 2.10 volts. The table search will set idx to one as this entry (179) is the fist value less the current value of cVolts. Our rough calculation of distance, then, is 15 centimeters. At this point test2 is indeed less than cVolts so we have to read the next lower table value (251) and place this into test1. Using 251 and 179 for test1 and test2 we get a slope value of 144 (at this point slope is in millivolts per cm). Using the BASIC Stamp's integer math, the difference from our rough distance calculation works out like this:

((210 – 179) * 10 / 144 = 2

When we subtract two from our rough calculation we end up with a distance reading of 13 centimeters.

Okay, so much for the theory, how does it work in practice? I marked up my test rig at one centimeter intervals and found that it worked pretty well – the readings across the range were within a centimeter of the actual distance to my target. I found this perfectly acceptable given the [slightly loose] specifications of the GP2D12.

The reason I developed the code that I did is that it's very easy to plug in different sensor values. And I elected to use a **DATA** table instead of **LOOKUP** so that the program can be more easily expanded with more table entries (**LOOKUP** tables beyond a few values can get unwieldy). If you'd like to find a way to plug the voltage value into a formula in order to get the distance value, I encourage you to visit the Acroname web site and look at their application note on the GP2D12. That note goes into a very detailed discussion of finding slope and offset points to linearize the output from the GP2D12. It's a little bit complicated and requires some experimentation, but you may find this method valuable.

Scare 'Em, Danno

Before we head out, let's chat a bit about using the sensor as I suggested at the beginning of the article. As I've frequently mentioned in the past, we can learn a lot by mimicing what pros have already done. I was in a public washroom a few days ago and the sinks had automated faucets. When one places their hands about six inches from the nozzle the water starts running.

How would you program the BASIC Stamp to mimic the faucet control (to apply it to a Halloween display)? This would be my strategy:

1. Measure distance to target.
2. Is distance less than threshold?
3. If no, go back to Step 1.
4. If yes, check several more times with a delay in between.
5. If target stays in range, trigger the device.
6. Add a [random] delay allow the prop to run and reset.
7. Go back to Step 1.

Can you do it? Of course you can – you're a BASIC Stamp programmer!

Have a safe and happy Halloween. Until next time, Happy Stamping.

```
' =============================================================================
'
'   File..... GP2D12.BS2
'   Purpose... Read voltage from GP2D12 and estimate object distance
'   Author.... Jon Williams
'   E-mail.... jwilliams@parallax.com
'   Started...
'   Updated... 16 AUG 2004
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =============================================================================


' -----[ Program Description ]---------------------------------------------
'
' Uses ADC0831 ADC to read voltage output from GP2D12 range sensor.  Note
' that the Vref input of the ADC0831 is set to 2.55 vdc, giving 0.01 volts
' per count.
'
' Output is to a SEETRON serial LCD set at 9600 baud.


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


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

Lcd             PIN     0                       ' serial out to LCD

AdcDta          PIN     15                      ' ADC data line
AdcClk          PIN     14                      ' ADC clock
AdcCS           PIN     13                      ' ADC chip select


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

Xspan           CON     5                       ' 5 cm per data point

LcdI            CON     $FE                     ' lcd command instruction
LcdCls          CON     $01                     ' clear the LCD
LcdHome         CON     $02                     ' move cursor home
LcdDDRam        CON     $80                     ' Display Data RAM control
LcdCGRam        CON     $40                     ' Character Generator RAM
LcdLine1        CON     $80                     ' DDRAM address of line 1
LcdLine2        CON     $C0                     ' DDRAM address of line 2


#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
```

```
    T1200       CON     813
    T2400       CON     396
    T4800       CON     188
    T9600       CON     84
    T19K2       CON     32
    T38K4       CON     6
  #CASE BS2SX, BS2P
    T1200       CON     2063
    T2400       CON     1021
    T4800       CON     500
    T9600       CON     240
    T19K2       CON     110
    T38K4       CON     45
#ENDSELECT

Inverted        CON     $4000
Open            CON     $8000

LcdBaud         CON     T9600 + Inverted        ' for SEETRON LCD

#DEFINE __Has_LCD = 1                           ' set to 0 for DEBUG



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

result          VAR     Byte                    ' adc result
cVolts          VAR     Word                    ' 0.01 volts
cm              VAR     Byte                    ' centimeters
idx             VAR     Nib

test1           VAR     Byte                    ' test values for
test2           VAR     Byte                    '   interpolation
slope           VAR     Word                    ' mV/cm between test points



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

Vout            DATA    251, 179, 139, 114,  97
                DATA     85,  76,  67,  62,  57
                DATA     53,  50,  48,  46,  43
                DATA      0

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

Reset:
  HIGH AdcCS
  #IF __Has_LCD #THEN
    PAUSE 500
    SEROUT Lcd, LcdBaud, [LcdI, LcdCls]
```

```
    PAUSE 1
    SEROUT Lcd, LcdBaud, [LcdI, LcdLine1+3, "* GP2D12 *"]
    PAUSE 3000
    SEROUT Lcd, LcdBaud, [LcdI, LcdCls]
    PAUSE 1
  #ELSE
    DEBUG CLS, "GP2D12 Demo"
  #ENDIF


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

Main:
  GOSUB Read_GP2D12                             ' read sensor
  GOSUB Estimate_Cm                             ' estimate distance

  #IF __Has_LCD #THEN
    SEROUT Lcd, LcdBaud, [LcdI, LcdHome]
    PAUSE 1
    SEROUT Lcd, LcdBaud, [DEC cVolts / 100, ".", DEC2 cVolts]
    SEROUT Lcd, LcdBaud, [LcdI, LcdLine2, DEC cm, " cm"]
  #ELSE
    DEBUG CRSRXY, 0, 2,
          DEC cVolts / 100, ".", DEC2 cVolts,
          TAB, "volts", CR,
          DEC cm, TAB, "cm"
  #ENDIF

  PAUSE 100
  GOTO Main
  END


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

Read_0831:
  LOW AdcCS                                     ' enable ADC0831
  SHIFTIN AdcDta, AdcClk, MSBPOST, [result\9]   ' read the voltage
  HIGH AdcCS                                    ' disconnect ADC0831
  RETURN


Read_GP2D12:
  cVolts = 0                                    ' reset reading
  FOR idx = 0 TO 2                              ' three reads (to filter)
    GOSUB Read_0831                             ' get the voltage
    cVolts = cVolts + result                    ' accumulate
    PAUSE 30
  NEXT
  cVolts = cVolts / 3                           ' average the readings
  RETURN
```

```
Estimate_Cm:
  FOR idx = 0 TO 15                              ' search table for location
    READ (Vout + idx), test2                     ' read table value
    IF (test2 <= cVolts) THEN EXIT               ' found position
  NEXT

  SELECT idx
    CASE 0
      cm = 10                                    ' fix to minimum range

    CASE 1 TO 14                                 ' calculate range
      cm = 10 + (5 * idx)
      IF (test2 < cVolts) THEN                           ' estimate through
interpolation
        READ (Vout + idx - 1), test1             ' get other border value
        slope = (test1 - test2) * 10 / Xspan     ' determine slope between
points
        cm = cm - ((cvolts - test2) * 10 / slope)
      ENDIF

    CASE 15
      cm = 80                                    ' fix to maximum range
  ENDSELECT
  RETURN
```