Stamp Applications no. 21 (November '96):

Look Into 'The Eye from TI' For Precision Light Readings

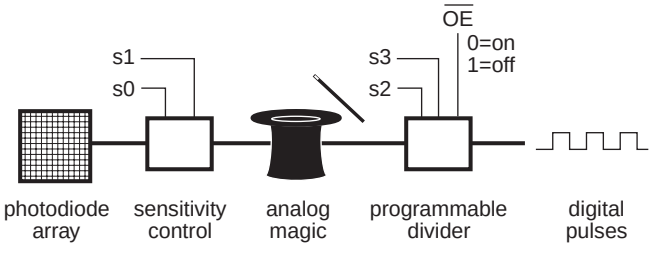
TSL230 light-to-frequency chip
plus beginner's race-timer project
by Scott Edwards

PHOTODIODES are excellent light sensors—better in most applications than cadmium-sulfide photocells, phototransistors, photovoltaic cells, etc. What those other devices have going for them is that their outputs are easier to interface than the tiny variations in current through a photodiode. Nobody wants to fool around with the analog electronics required to use photodiodes.

Texas Instruments (TI) looked at this situation and, instead of saying, "ain't it a shame" said, "let's make a product!" The product—actually one of a family of products—is the TSL230 programmable light-to-frequency converter. The TSL230 measures light to the nearest gnat's whisker using an array of photodiodes, and outputs Stamp-friendly digital square waves. We'll look at two strategies for using it with the Stamps, one for the BS1 and another for the BS2.

In BASIC for Beginners, we'll lay the foundations of a fairly typical Stamp project: a three-lane race timer.

The Eye from TI. Figure 1 is a block diagram of the TSL230 programmable light-to-frequency converter. The IC package (an 8-pin DIP) is transparent so that light shining on it reaches an array of photodiodes. An electronic sensitivity control connects part or all of this array to a current-to-frequency converter that I've labeled "analog magic." The frequency ranges from near 0 Hz in darkness to around 1.4 MHz in bright light.



s1	s0	sensitivity
0	0	power down
0	1	x1
1	0	x10
1	1	x100

s3	s2	divide by
0	0	1
0	1	2
1	0	10
1	1	100

Figure 1. Simplified block diagram of the TSL230 light sensor.

The output is fed into a programmable divider that can pass the signal as-is, or divide it by 2, 10, or 100.

The TSL230 has a couple of especially microcontroller-friendly features: Its OE (output enable) pin lets you disconnect the output so that it can share a data bus with other devices. And the sensitivity control may be used to power down the chip to reduce its current draw from 2mA active to 10µA idle.

Since the TSL230's output is basically a train of pulses, it's easy to measure with the Stamps using the Pulsin instruction available on the BS1 and BS2, or the new Count instruction on the BS2 only. Figure 2 shows the hookup used in the example programs of listings 1 and 2.

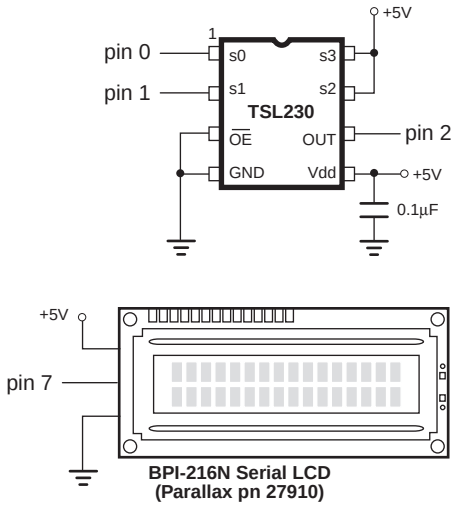


Figure 2. Hookup for listings 1 and 2.

Using Pulsin, the Stamp measures the width of a single pulse. This provides a quick snapshot of the frequency. In order to get decent resolution out of a single pulse measurement, we want the pulse to be as long as possible without exceeding the max pulse width (655 ms for BS1 or 130 ms for BS2). To maximize pulse width, I set the output divider to 100, meaning that each output pulse represents 100 cycles from the light-to-frequency converter.

With the Count instruction, you can get high resolution by either ensuring that the frequency is close to the maximum that Count will handle (125 kHz), or by setting the instruction to count for a long period of time (up to 65 seconds).

Since bright light drives the light-to-frequency converter as high as 1.4 MHz, we have to set the programmable divider to ensure that the output to the BS2 never exceeds 125 kHz. The only setting that will do the trick is divide-by-100.

With that setting, the max output to the BS2 is 14 kHz. Count has a 16-bit range (65535 max), so it could be set to count as long as 4.6 seconds without overflowing. In the example program, I used a 1-second count.

Both of the example programs display the effects of cycling the sensitivity settings through x1, x10, and x100. The sensitivity control is possibly the neatest feature of the chip. TI refers to it as an "electronic iris," since it works by controlling the surface area of the photodiode array used to drive the light-to-frequency converter. Want more sensitivity? Switch in all the photodiodes. Less? Switch in fewer photodiodes.

Controlling sensitivity can be important to a light-sensing application. For example, I found that in bright sunlight the x10 and x100 setting returned the same value. The light-to-frequency converter was *saturated*—driven as high as it would go regardless of further input. Only the x1 setting returned usable results in bright sunlight.

In normal room light, I saw the advantage of the higher sensitivity settings. I tried covering the TSL230 with a piece of transparent plastic cut from a plastic bag. At the x100 setting, the plastic made a large difference in the light measurement. At x1, it didn't make a difference at all.

Applications for the TSL230? TI uses a smart washing machine as an example. The TSL230 monitors the amount of light passing through the rinse water to determine when to end the cycle. A similar arrangement might keep tabs on the quality of water in an aquarium or beer in a vat. Photographic applications, like light metering, exposure control, color matching, densitometry, etc. seem obvious. Or how about an improved version of those heart-rate monitors that pass light through the skin? The frequency-modulated output of the TSL230 ought to be less vulnerable to noise than amplitude-modulated sensors.

BASIC for Beginners. Programming is about problem-solving, so this month we'll conduct a brainstorming session aimed at finding a suitable solution to a typical Stamp programming problem.

The problem is this: We want to use a BS1 as a three-lane race timer for a Pinewood Derby race. This kind of race involves small (6" or so) gravity-powered cars made from blocks of wood. The cars start simultaneously, and the winner is the car that reaches the end of the straight, downward-sloping track in the shortest time.

From an electronic standpoint, let's assume that we have four switches arranged as shown in figure 3. The start switch pulses low when the cars leave the starting gate, and the three finish switches pulse low as their respective cars cross the finish line. The switches could be simple mechanical devices, or fancier light-beam electronic switches.

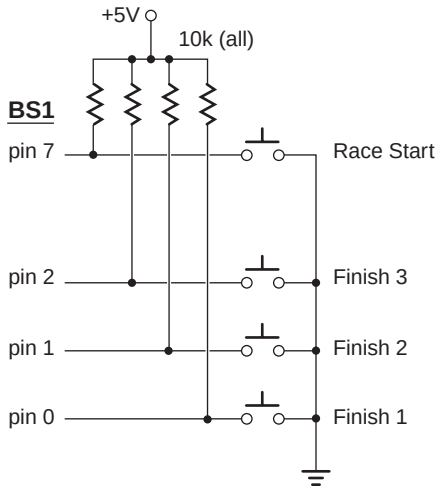


Figure 3. Arrangement of race-timer switches.

The most important function of our program will be to keep an accurate record of the time from start to finish for each of the cars. A secondary goal will be to have as much timing precision as possible, since a race may be won by just milliseconds.

The first duty of our program will be to watch the start switch until it goes low, signaling the start of the race. We have two choices: to read the start pin directly, or to use the Button instruction, which offers such amenities as switch debouncing and autorepeat.

How to choose? Let's try to state the function of the start switch as specifically as possible. The contacts of the start switch will close at the instant the cars are released to start the race. The switch will bounce (flutter on/off for a few milliseconds) after initially closing, then stay closed for some indefinite amount of time. Once the cars are on their way, the switch may reopen after a while.

Hmm, switch bounce. Maybe we ought to go for the debouncing feature of Button. Or should we? After all, it's the initial switch *closure* that signals the start of the race, not what happens after that. Once the race starts, we can completely ignore the start switch. To put it into computerese, it's the high-to-low *transition* that matters. So Button is unnecessary; we can just watch the pin. A bit of initial coding and we have a test program:

```
SYMBOL starter = pin7
hold:
if starter =1 then hold
debug "Start!", cr
end
```

As long as the start switch is open, the Stamp sees a 1 on pin7 (which we have renamed "starter"). So the program keeps executing the line after hold: over and over. When the switch closes, starter is equal to 0, so the if...then condition is no longer true. The program drops down to the next line and prints the "Start!" message on the debug screen.

Now comes the harder part—keeping track of time for each of the racers until they cross the finish line. The Stamp doesn't have a timer function, so 'keeping time' really means adding 1 to a counter in a loop that ends when the race is over. For instance, if we had only one car to keep track of, we could use a variation of the hold loop above:

```
SYMBOL finish = pin0
SYMBOL time = w2
'... (hold loop here) ...
timing:
  if finish = 0 then raceOver
  time = time + 1
```

```
goto timing
raceOver:
debug "Finish time =", #time, cr
end
```

While this approach would be fine for one lane, it wouldn't work at all for three lanes. The race ends only when all three cars have finished, and a finish switch only pulses briefly when its car crosses the finish line. Thinking about adding a second lane to that program, you might be tempted to write:

```
if finish1 = 0 then raceOver
if finish2 = 0 then raceOver
```

Then all timing would stop when either car crossed the finish line. We want race times for all cars, and for the race to finish only when all cars are done. Another thought might be:

```
if finish1 = 0 and finish2 = 0 ...
     ... then raceOver
```

No good. This code would only work if car 1 and car 2 finished simultaneously.

The point is that there's really no way to patch up the one-lane approach to work for three lanes. Something critical is missing, and we have to figure out what it is before we can make a working program.

Why did I show you this blind alley? I wanted to talk about courage. That's right, courage; it takes guts to throw away an idea that initially seems promising.

The cowardly thing to do would be to attempt to save that first-draft code by adding more and more instructions to counter each of its deficiencies.

We're brave and bold enough to say "phooey" to a bad idea. Let's start over with another run at the problem in light of that false start.

The primary problem with the one-lane code is that it can't deal with the fact that the finish switches only pulse when the cars cross the finish line. If these switches came on and stayed on, that code would be a lot more viable.

Ah, the snake rears its ugly head. Faced with difficulty, you begin to think about messing with the hardware to salvage bad software. Resist temptation; hardware mods are a last resort. Keep thinking.

OK, suppose we store the status of each car as something like "racing" and "finished." Then we have a clear-cut condition for keeping time, as in 'if racing1 = 1 then time1 = time1+1.' (Assuming that 'racing1' is the status of car1, where 1=in-the-race and 0=finished.) We also have a clear condition for stopping a car's timer; 'if finish1 = 0 then racing1 = 0.' And—hallelujah—we have a condition for ending the whole race; 'if racing1 = 0 and racing2 = 0 and racing3 = 0 then all_finished.'

It appears that our racing timer is now on the right track.

Next month, we'll prototype the racing timer application and look at ways to streamline the code. We'll see how bit variables can be used as flags, and how to convert If...Then instructions into compact Boolean logic. I think you will be surprised and pleased to see how simple the final code is.

NOTE:

Texas Instruments has licensed the TSL230 to another company, TAOS, www.taosinc.com. See them for availability of the TSL230 sensor.

This article was originally published in 1996. The Stamp Applications column continues with a changing roster of writers. See www.nutsvolts.com or www.parallaxinc.com for current Stamp-oriented information.

Listing 1. BS1 Program to Demonstrate TSL230

```
' Program: TSL230.BAS (Interface with TSL230 light sensor)
' This program demonstrates the light-to-frequency conversion
' capability of the TSL230 sensor from Texas Instruments.
' BS1 pins 0 and 1 control the sensitivity of the '230 through
' its "electronic aperture" feature. The higher the sensitivity,
' the higher the frequency output for a given light intensity,
' as shown below:
'                bit1     bit0      Sensitivity
'                ----     ----      -----------
'                 0        0        sensor OFF
'                 0        1        x1
'                 1        0        x10
'                 1        1        x100
' Since the BS1 measures pulse width rather than frequency, its
' response is reciprocal; larger numbers mean less light. The program
' reverses this by dividing the light-dependent value into 65535.
' The program displays its readings on a 2x16 serial LCD module.

SYMBOL sens = b2        ' Sensitivity setting.
SYMBOL mult = b1        ' Multiplier for a given sensitivity.
SYMBOL light = w2       ' Light-intensity reading.
SYMBOL I = 254          ' Instruction prefix for LCD.
SYMBOL one = 128        ' Address of 1st LCD line.
SYMBOL two = 192        ' Address of 2nd LCD line.
dirs = %00000011        ' Make pins 0 and 1 outputs.
again:                  ' Main program loop.
for sens = 1 to 3       ' Walk through sensitivity settings.
  pins = sens           ' Write sensitivity setting to pins.
  lookup sens,(0,1,10,100),mult ' Get the sensitivity multiplier.
  pulsin 2,1,light      ' Take a light reading.
  light = 65535/light   ' Compute reciprocal.
  serout 7,n2400,(I,one,"multiplier: x",#mult,"   ")     ' Display.
  serout 7,n2400,(I,two,"light: ",#light,"    ")
  pause 1000            ' Wait a second between readings.
next                    ' Next sensitivity.
goto again              ' Repeat forever.
```

Listing 2. BS2 Program to Demonstrate TSL230

```
' Program: TSL230.BS2 (Interface with TSL230 light sensor)
' This program demonstrates the light-to-frequency conversion
' capability of the TSL230 sensor from Texas Instruments.
' BS1 pins 0 and 1 control the sensitivity of the '230 through
' its "electronic aperture" feature. The higher the sensitivity,
' the higher the frequency output for a given light intensity,
' as shown below:
'                bit1     bit0      Sensitivity
'                ----     ----      -----------
'                 0        0        sensor OFF
'                 0        1        x1
'                 1        0        x10
'                 1        1        x100
' This BS2 program uses the COUNT instruction to count the number
' of TSL230 output cycles over a period of 1 second. This approach
' trades a relatively long measurement period for excellent
' resolution and accuracy. For a quick-and-dirty measurement,
' the PULSIN approach used with the BS1 could be employed instead.
' The program displays its readings on a 2x16 serial LCD module.

sens    var byte        ' Sensitivity setting.
mult    var byte        ' Multiplier for a given sensitivity.
light   var word        ' Light-intensity reading.
I       con 254         ' Instruction prefix for LCD.
one     con 128         ' Address of 1st LCD line.
two     con 192         ' Address of 2nd LCD line.
n2400   con $418d       ' Serial constant for 2400 bps

dirs = %00000011        ' Make pins 0 and 1 outputs.
again:                  ' Main program loop.
for sens = 1 to 3       ' Walk through sensitivity settings.
  OUTS = sens           ' Write sensitivity setting to pins.
  lookup sens,[0,1,10,100],mult ' Get the sensitivity multiplier.
  count 2,1000,light    ' Count pulses for 1 second.
  serout 7,n2400,[I,one,"multiplier: x",DEC mult,"   "]  ' Display.
  serout 7,n2400,[I,two,"light: ",DEC light,"    "]
next                    ' Next sensitivity.
goto again              ' Repeat forever.
```