CIRCUIT CELLAR®
THE MAGAZINE FOR COMPUTER APPLICATIONS

FROM THE BENCH

by Jeff Bachiochi

Light-to-Frequency Conversion (Part 2)

Pulse and Oxygen Content

Last month, Jeff showed you how to use a TAOS TSL230R light-to-frequency converter. In this column he describes a TSL230R-based device that allows you to monitor a heart rate.

Last month I left you with a graph of sampled data from a project that uses a TAOS TSL230R programmable light-to-frequency converter to monitor the light transmitted through living tissue. You learned that the heart's pumping action moves blood through the body via its distribution system of arteries, capillaries, and veins. The oxygen collected in your lungs passes through your entire body in this way. When your heart muscle contracts, it increases arterial pressure as blood cells are forced out into the arteries. The arterial system, which is strong and flexible, slightly expands and contracts during pressure changes. Although you can feel this pulsation best at specific points on the body, it takes place everywhere.

Your bones and tissue block (or absorb) much of the light passing through your body. For the most part, a constant (although small) portion of light will pass completely through. This light intensity will be modulated slightly by the arterial system as it pulsates and dynamically changes the amount of light that is able to pass through it. This project monitors light passing through the body in an effort to extract heartbeat information from the small dynamically changing portion of light.

As I explained last month, the TSL230R eliminates the need for designing high-gain analog circuitry to amplify the output of a photodiode. This analog circuitry is extremely susceptible to external noise and it requires an A/D converter to get the signal into a microcontroller for data analysis. The TSL230R's frequency output eliminates this messy front end and is easy to attach to any microcontroller thanks to the TTL compatible I/O.

PULSE

Figure 1 is similar to the graph I presented in Part 1. You can see the systolic and diastolic phases of each heartbeat. The former is the contraction, or working, phase of the heart when the pressure is highest. As the pressure increases, more blood squeezes into the arterial system. More blood absorbs more light. Less light getting through reduces the frequency output of the TSL230R. Lower frequencies require longer period counts. Thus, the data has higher counts during the systolic phase. During the diastolic phase, the heart rests and pressure drops as blood draws back to the heart in preparation for the next heartbeat.

You've had your blood pressure checked. Usually, a cuff is placed on your upper arm and pumped until it cuts off the flow of blood. A stethoscope is used to listen for returned blood flow as the pressure in the cuff is released. The numbers you receive from this test like 120/80 are actually your systolic pressure over your diastolic pressure. It's the systolic reading that identifies potential high blood pressure.

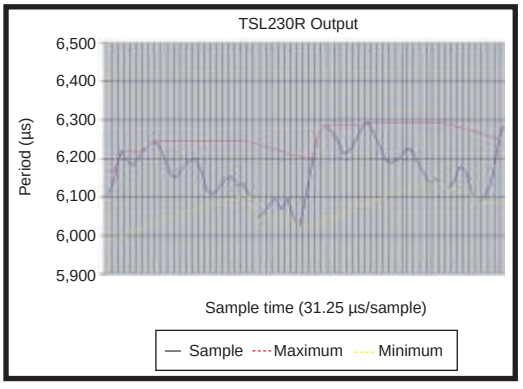

Figure 1—*The TSL230R sensor's sample period counts were taken every 31.25 ms. Code algorithms pick out the maximum and minimum peaks used to determine a heart rate. Peak values are allowed to leak off after a change in slope has been detected when a sample exceeds the opposing peak.*

What you need to extract from the TSL230R's data is the amount of time for a complete systolic-diastolic cycle. In this case I'll determine the systolic (maximum) peaks and calculate the heart rate based on the number of samples between consecutive peaks. Although the output data in Figure 1 might be typical, the actual wave shape might be quite different from person to person. (I'm sure a specialist could glean a lot of information about your body's performance from this wave shape. But I'm not a doctor, and I don't play one on TV.)

One of the biggest problems when looking at the sampled data is the level of constant absorption. It isn't really constant. This level varies because of source lighting changes or light path changes. You can control light source changes by creating a stable current supply for the LED used as a light source, but beyond creating a snug yet comfortable sensor, even the slightest twitch or movement of the measured appendage will change the path of light and look to your sensor like an AC component where you're considering it a (constant) DC level. This creates havoc when you're trying to measure the minimum/maximum excursions because of systolic and diastolic pressures.

PROJECT

Let's take a closer look at how you can use a TSL230R to monitor

a heart rate. The push buttons allow for manual changes to the system so you can determine the optimum settings even during program execution. I predefined PB1 as an LED toggle between red and IR LEDs. PB2 was predefined as a Sensitivity mode increment to the TSL230R. PB3 was predefined as a Divisor mode increment to the TSL230R. I used these inputs to manually change the TSL230R's settings and observe the frequency output of the device for each of the modes. Because these are all based on the light source, it was convenient to choose between the two sources for light, the red LED and the IR LED. These LED colors were chosen because they're within the TSL230R's bandwidth and they have different absorption characteristics based on the amount of oxygen in the blood. I tried choosing LED currents that would allow the LEDs to have similar intensities based on having no objects between the LEDs and the TSL230R sensor.

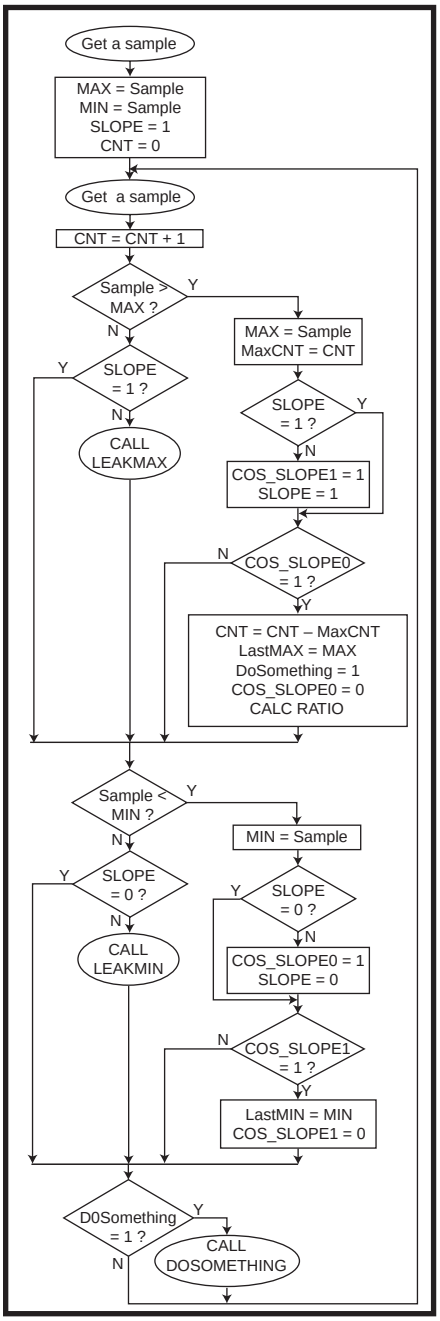

Figure 2—*Maximum and minimum samples are held for use when determining both the number of samples between maximum peaks (beats per minute) and the ratio of the AC count and the DC count.*

The sensor's frequency output is connected to the external interrupt input on a microprocessor. The frequency's period is measured by counting timer ticks between any two consecutive rising edges (interrupts) of the input signal. Last month I showed you that the best mode to use would be that which produced the highest count without overflowing the 16-bit (1 µs/count) counter.

The maximum count of Timer1 is set at 31.25 ms (or 1/32 s) by reloading with a constant each overflow, and it becomes the sampling rate. However, if the tick count exceeds half that value, there is a good chance a count during a successive sample period will overflow because the count period of the sensor and the sample times are asynchronous. So you can trap any count over this value and assign the 16-bit count an 0xFFFF value, which indicates a count that's over the maximum limit (or a timer overflow, which would also be over that limit).

Just as the count for a low-frequency output can be too long, a count can be too short for a high-frequency output. A short count is limited by how fast the count recovery routine (interrupt) can be serviced. For obvious reasons, counts never will be less than this value, but they could be illegal for counts up to twice this value.

With the establishment of maximum and minimum count limits, a sample can be taken while the TSL230R cycles through each mode to automatically determine the best one to use (while the sensor is connected to a patient). Mode selection must be done separately for each LED because there is no guarantee the LEDs will provide acceptable counts using the same mode for each.

A sample output similar to Figure 1 can be recorded at this point. To help identify the maximum and minimum sample counts, positive and negative peak detectors are implemented in code according to certain rules (see Figure 2). Like a hardware peak detector in which a voltage passes through a diode to charge a capacitor, anytime a sample is greater than the last saved maximum or less than the last saved minimum, it's saved as the new maximum or minimum peak (see Figure 3).

To prevent cases in which there's a shift downward (or upward in the case of minimums) because of a change in DC level or a change in the absorption constant, and a new maximum peak can't occur, you must leak off some charge or allow the maximum to work its way down (like a resistor discharging the capacitor). To determine the rate of leakage, find the mean between the maximum and minimum peaks and then reduce the former (or increase the latter) by some factor of it.

I temporarily programmed PB4 to manually cycle this value by a factor of two each time (1, 2, 4, … 64, 128, 1, 2, etc.). This leakage adjustment recalculates the maximum and minimum peaks each sample time, except when

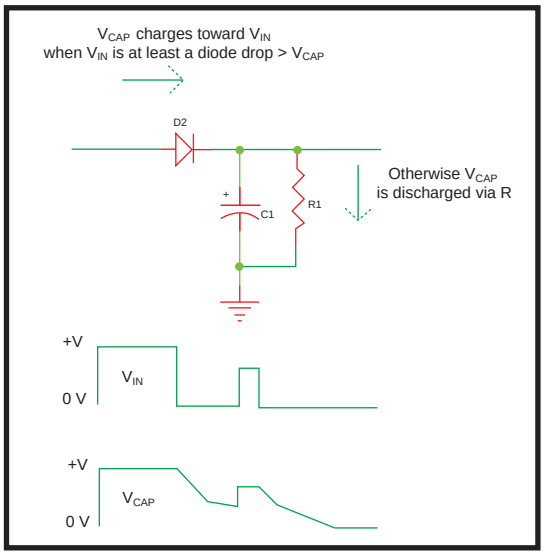

Figure 3—*Code algorithms in this project act like a hardware peak detector. The diode allows the capacitor to charge and hold the highest voltage applied to V_{IN}. The resister can be sized to slowly leak off the capacitor's charge (V_{CAP}).*

a new peak is detected and held until the slope changes state. I ultimately let the program automatically select the leakage factor by looking at sample data. Too little leakage can cause a maximum or minimum peak to be missed (if the DC level shifts), while too much leakage can cause the recognition of false peaks.

Referring to Figure 1, notice how a rising slope changes state when a sample falls below the leaking minimum value. The opposite is true for a falling slope: it changes state when a sample exceeds the leaking maximum value. Not only is the slope affected by this action, but a change-of-state (COS) flag is recorded for both a positive slope [SLOPE=1 (COS_SLOPE1)] and a negative slope [SLOPE=0 (COS_SLOPE0)]. As you can see in Figure 2, special events occur during the execution of the sampling loop. This is when the number of beats per minute (bpm) and O_2 can be displayed.

The DoSomething flag is set each time the slope equals one and the COS_SLOPE0 equals one, which means the slope has just changed to one. A fortune telling circuit would be handy because there's no way of knowing that any particular peak sample will be the maximum before the wave's slope changes. Wait until some point when you can be sure that this has happened. When a sample value falls lower than the leaking minimum peak (a slope change), you can be sure you have the last maximum peak. The CNT (number of samples since the last determined peak) is adjusted to reflect the real CNT where the new maximum peak has occurred, such that it will reference that new maximum peak the next time the CNT is saved during a new maximum detection.

The new maximum peak value is saved. A ratio is calculated using this new value along with the last minimum peak value. The ratio is a number that represents the AC value (the difference between maximum and minimum peak samples) divided by the DC value (the mean sample). The ratio is calculated and stored separately during red and IR LED operations. Because this is the point in time when a positive peak has indeed occurred, enabling the piezo beeper will give an audible indication of such an event. By simply enabling Timer0, background interrupts from this timer overflowing will allow the interrupt routine to toggle the piezo output once each overflow a number of times before disabling itself, self-completing a short beep. Although some calculations are performed in this portion of the code, most display routines are handled at the end of the loop. All display routines are serial datastreams. Although this project requires connecting to a PC running HyperTerminal to see the data, it very easily could be formatted for a small LCD.

SERIAL DATA

The sampled value is the main attraction here. By properly shielding the TSL230R from extraneous light (the biggest culprit being 60-Hz light sources), a nice waveform can be produced from the sampled data. Pay attention to the old adage: Garbage in, garbage out. Bogus data will surely cause frustration no mater how good your algorithm.

A number of flags enable various data output. Although the sample values are valuable, they aren't too helpful unless they're displayed in a format you can easily recognize. I found that using HyperTerminal to grab the data and Excel to view the imported data as a graph was extremely helpful in planning my algorithm without having to write an application in, say, Visual Basic to grab and display samples either as text or graphs. I'm sure National Instrument's LabView would've been a great tool to use here as well. By adding other values like MaxCNT and MinCNT to the output, I was able to see exactly how my algorithm was using the data and fine-tune it (see Figure 1).

Ultimately, the basis of this project comes down to displaying two variables, the beats per minute and oxygen saturation level. I've covered how a beep fires off each time a maximum peak is established. Text messages to display the beats per minute and O_2 are chosen using PB4 (after reprogramming its function).

BEATS PER MINUTE

Based on the sampling period of 31.25 ms, a heart beating at 60 bpm, or 1 bps, would have 32 samples between maximum peaks. So, the beats per minute equals 32 divided by the number of samples (CNT) times

60 bpm: 1,920/CNT. With a design scope between 50 and 250 bpm, that would be approximately 39 samples (at 50 bpm) and seven samples (at 250 bpm).

Converting the number of samples to beats per minute is adequate for the low end where the number of samples is larger. At the upper end where the number of samples is lower, the resolution is unacceptable. This can be improved by averaging a number of samples. By averaging eight samples, the calculated beats per minute error (because of resolution granularity) can be cut to less than 2% in the 250 bpm range. Empirically, the 50 to 250 bpm range is a bit skewed, where 40 bpm seems like a better low end. When I was resting, my rate was approximately 50 bpm. At the other end of the spectrum, 250 bpm seemed too high a value because a resting infant measures approximately 110 bpm. Even under stress rates of 250 bpm seem extreme.

Photo 1—*Joshua rejoices in having the nose tube and oxygen mask removed after surgery. The inset shows a pulse oximetry sensor applied to his big toe. The glow of a red LED triggered the technical questions I put to the nurse on duty.*

O_2 SATURATION

Before the use of pulse oximetry, the only way to monitor oxygen levels in blood was to take a blood sample. Discounting the invasiveness of the procedure, this required getting a laboratory involved. Needless to say, this wasn't a real-time process by any means.

My four-month-old grandson Joshua recently had surgery (see Photo 1). As my family huddled around him in the recovery room, I asked one of the nurses about the monitor they were using to display bpm and O_2. I have the sneaking suspicion she doesn't get asked technological questions too often because her eyes popped open and she immediately went into a sales pitch about how this tool improves patient care.

Calculating O_2 saturation is based on the fact that the light absorption is dependent on both the wavelength and material. Deoxygenated blood absorbs red light (600 to 700 nm) at a greater rate than IR light (800 to 940 nm). This means that by comparing the measurements made at the two wavelengths, you can calculate the amount of oxygen in the blood. The maximum and minimum excursion values (AC portion) of the sampled data are related to the mean value (DC portion). So, it's the ratio of these values that can be compared between each of the two wavelengths. Depending on which LED is on, either IR_Ratio or RED_Ratio is calculated once per heartbeat.

These ratios are used to calculate O_2 saturation when PB4 has requested O_2 mode. RED_Ratio divided by IR_Ratio should provide a value less than 10. This nonlinear value relates to an oxygen level where 0.4 corresponds to 100%, 1 corresponds to 85%, and 3.4 corresponds to 0%. Here I'm interested in values between 80% and 100%. For this project, I assumed this portion of the curve to be linear.

There are plenty of opportunities for error when trying to determine O_2 saturation. The most obvious sources of error are the actual LED wavelengths of the devices. Extraneous light, sensor movement, light-absorbing species in the blood (e.g., dyes and gases like carbon monoxide), and poor blood flow also can causes errors. A tool is only as good as the technician working with it.

FINISHED?

This project has potential. You could build a stand-alone sensor in which a processor is mounted along with the LEDs, the TSL230R light-to-frequency converter, and a piezo beeper to give audible feedback concerning the beats per minute rate. With an on-board 3-V battery, this low-power device could become a metronome for pointing out radical problems without being tied to bulky medical equipment. By adding a small LCD, both beats per minute and O_2 data can be determined with a highly portable system.

This project is a handy way to experiment with the TSL230R sensor. Although I didn't get around to squashing the code into an eight-pin microcontroller, you can easily put the stand-alone version with a piezo output in that device. Based on the jungle of wires and tubes I saw in Joshua's recovery room, I bet future medical gizmos will incorporate wireless networking.

TAOS has unique parts that are good alternatives to analog optical sensors. Their digital output provides a simple interface to all microcontrollers without the need for potentially noisy analog signal conditioning and ADCs external or internal to the microcontroller. Check out TAOS's array of optical products the next time you have a design that requires light detection.

Jeff Bachiochi (pronounced BAH-key-AH-key) has been writing for Circuit Cellar *since 1988. His background includes product design and manufacturing. He may be reached at jeff.bachiochi@circuitcellar.com.*

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/174.

RESOURCES

J. Mitchell, "Principles of Measurement in Anesthesia," http://jam.customer.netspace.net.au/anaesth/measurement.html#09.

Omimeter.org, "Principles of Pulse Oximetry Technology," 2002, www.oximeter.org/pulseox/principles.htm.

TAOS, Inc., "Pulse Oximetry," www.taosinc.com/downloads/pdf/pulse.pdf.

———"TSL230R, TSL230AR, TSL230BR: Programmable Light-to-Frequency Converters," TAOS048A, 2004.

SOURCE

TSL230R Light-to-frequency converter
TAOS, Inc.
www.taosinc.com