


# designideas

READERS SOLVE DESIGN PROBLEMS

## Create a swept-sine function in LabView with just one virtual instrument

Sean McPeak, University of California—San Diego

 Swept sine waves are useful when you want to test a product over a wide frequency range. A large research project included the requirement to determine wave propagation in the open ocean. This application required the generation of a swept sine wave to drive an acoustic transducer. Although many waveform generators have a built-in function for this requirement, you must program it yourself if you want to implement a swept sine with a multifunction data-acquisition card. You can create a swept-sine function in National Instruments' (www.ni.com) LabView with just one VI (virtual instrument). Using this function, you can control start and stop frequencies, sample rate, and the overall duration of the sweep (Figure 1).

The LabView software calculates an array of numbers that represent the swept-sine-wave time series at each sample point as the frequency either increases or decreases, depending on the direction of the sweep. You must handle the frequency change of the output on a point-by-point basis. The basic form of the equation is  $Y(I) = V \times \sin((A \times I^2)/2 + B \times I)$ , where  $Y(I)$  is the amplitude of the swept sine wave as a function of the sample point,  $I$  is the integer that steps through the time series,  $V$  is the peak voltage, and  $A$  and  $B$  are variables. You define  $A$  as  $2 \times \pi(f_{\text{STOP}} - f_{\text{START}})/N$ , and you define  $B$  as  $2 \times \pi f_{\text{START}}$ , where  $N$  is the number of samples,  $f_{\text{START}}$  is the normalized start frequency, and  $f_{\text{STOP}}$  is the normalized stop frequency. To normalize the start

### DIs Inside

44 Charlieplexing at high duty cycle

48 Serial port tests digital circuits

51 DAC calibrates 4- to 20-mA output current

51 Alarm tells you to close the refrigerator door

▶ To see all of EDN's Design Ideas, visit [www.edn.com/designideas](http://www.edn.com/designideas).

and stop frequencies, you must change the unit to cycles per sample. You accomplish this task by dividing the  $f_1$  and  $f_2$  frequencies in hertz by the sample rate. You determine the sample rate by deciding how smooth of a transition you want to represent your swept sine wave. A good rule of thumb is to have at least 10 samples/cycle at the high-

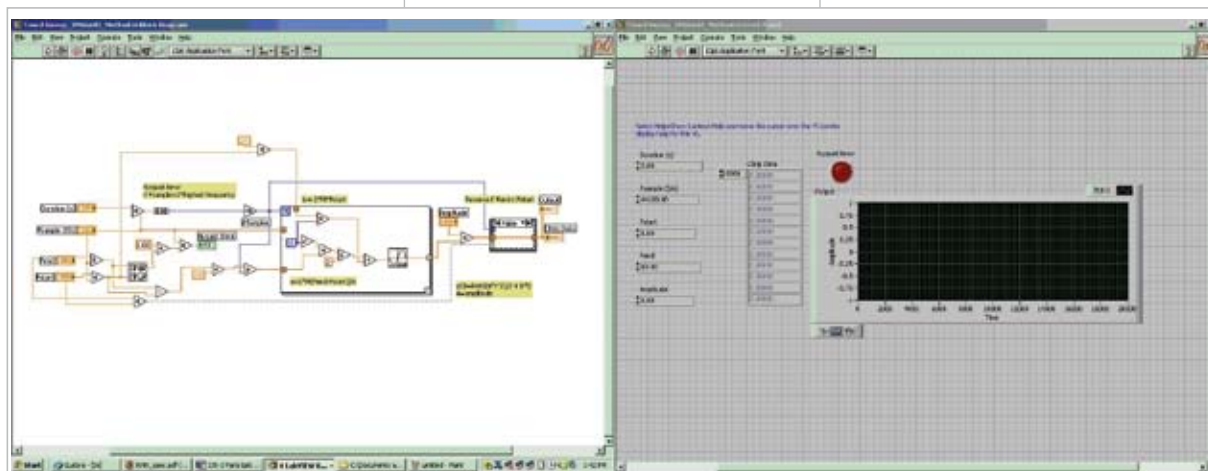


Figure 1 With just one LabView virtual instrument, you can control start and stop frequencies, sample rate, and the overall duration of the sweep.

est frequency. When setting the sample rate, you need to take into account the overall frequency span you are sweeping and the duration of the sweep itself. It is also helpful to compare the results and performance of the LabView data-acquisition-system implementation of the swept sine wave with those of an AWG (arbitrary-waveform generator).

You use two methods of comparison. First, you compare the output of both the data-acquisition and the AWG swept sine wave on a spectrum analyzer. Second, you run them both through an audio-amplifier/speaker system and simply listen to the output. This method is useful in determining sweep rate, duration, and stop and start frequencies. This type of comparison is valid only if the frequencies involved are in the audible range. The LabView VI employs simple array manipulation and uses a “for” loop. The input duration is in seconds, the sample frequency is in samples per second, and the starting and ending frequencies are in hertz. Dividing the sample rate immediately converts the start and end frequencies to cycles per sample. A maximum/minimum block takes the normalized ending and starting frequencies as its inputs and uses the maximum output of the input pair. You use this method to determine whether your design meets the

Nyquist criteria, given the sample rate and highest frequency you require.

This approach drives a simple Boolean variable to alert the user about whether the design meets the Nyquist criteria. You set the “for” loop to run for the total number of samples you want to calculate. You determine this value by multiplying the duration in seconds by the sample rate in samples per second. To guarantee that the loop processes all of the generated samples, you must add one, because the loop stops at  $N-1$ .

You implement the output function in the “for” loop with simple algebraic operators and the sine block. The output is an array that reaches the perimeter of the “for” loop. It is important to enable indexing at this node. This action allows the circuit to individually handle each element in the array at the output of the “for” loop. You can also add a simple gain stage to set the peak-to-peak value to whatever point you want. Finally, you use the “rotate-1D-array-block” case structure to flip the array if the ending frequency is lower than the starting frequency. This approach handles cases in which you want a frequency sweep that starts out in a higher frequency and descends to a lower frequency.

You can easily modify and expand

this simple program. One idea would be to use the output array, which is nothing more than the time series representing a predescribed frequency sweep, to feed a loop that would drive a data-acquisition module. The output of the module should accurately represent the frequency sweep, as long as the module’s output sample rate is the same as the sample rate you use for generating the frequency-sweep time series. You should then be able to track the output samples and, when they are complete, reverse the frequency-sweep array. You then again feed this new flipped array to the data-acquisition module. Depending on the maximum and minimum frequencies, sweep duration, and sample rate you use, it may be difficult to flip the array and configure the module quickly enough to not miss a sample. In that case, you can prefill a frequency-sweep array for a set number of passes.

These modifications allow the sweep to continue back and forth for a set period. Another improvement would be to add some real-time FFT (fast-Fourier-transform) capability so that the user can see the sweep in the frequency domain. This approach also adds an increased level of insurance that the circuit properly meets the sweep definition.**EDN**

# Charlieplexing at high duty cycle

Luke Sangalli, Digital Designs, Melbourne, Australia


 A few articles have recently appeared describing novel ways to increase the number of LEDs a microprocessor can drive with a limited number of pins (**Reference 1**). The standard multiplexing technique made popular by multidigit seven-segment displays has, in pin-scarce designs, given way to “Charlieplexing.” Charlie Allen devised this technique while working at Maxim (www.maxim-ic.com), which has since introduced

TABLE 1 NO. OF LEDs AND DUTY CYCLES

No. of pins	No. of LEDs	Charlieplexing duty cycle (%)	Standard multiplexing duty cycle (%)/no. of pins
Three	Six	33.3	50/five
Four	12	25	33/seven
Five	20	20	25/nine
Six	30	16.6	20/11
Seven	42	14.2	16.6/13
Eight	56	12.5	14.2/15
Nine	72	11.1	12.5/17
10	90	10	11.1/19

ICs using the technique (**Reference 2**). Allen used the high-impedance third input state available to most microprocessors for turning off LEDs in a matrix; the LEDs’ respective microprocessor pins’ high or low states individually turn on these LEDs. Using this method, you can drive nine seven-segment LED displays using only nine microprocessor pins rather than the usual 17. For  $N$  pins, you can individually address  $N \times (N-1)$  LEDs using Charlieplexing. One of the gripes people often level at Charlieplexing regards its poor duty cycle. A previous Design Idea com-

compares the standard multiplexing method with Charlieplexing (Reference 3). Using Charlieplexing, the maximum duty cycle for a 20-LED display is only 5%. The poor duty-cycle figure is due not to the method, however, but rather to the driving capability of the microprocessor and the parasitic-leakage paths. A single pin cannot usually sink the current a number of LEDs require to effectively light up, so these designs often require one source pin and one sink pin to light only one LED at any time. However, adding a transistor or two resistors allows you to circumvent these issues.

If you rearrange the LEDs in the familiar cross-point array and add a transistor to each column to carry the common current, you'll see the duty cycle of the Charlieplexing method does not differ much from standard multiplexing (Figure 1). For a 20-LED, five-column matrix, each LED remains on for 20% of the time compared with 25% for standard multiplexing, but now using only

## BY THE TIME YOU GET TO 90 LEDs, THE PCB REAL ESTATE AND COST OF THE 10 TRANSISTOR/RESISTOR SETS PALE IN COMPARISON TO THE DISPLAY ITSELF.

five pins instead of nine (Table 1).

One of the drawbacks of adding the transistor and resistors to each column is that you need additional components to achieve a reasonable LED brightness when a large number of LEDs is involved. This approach, however, is a better alternative to using a costly IC and no worse than standard multiplexing or "Gugaplexing," which also requires additional transistors and resistors. From a cost and benefits point of

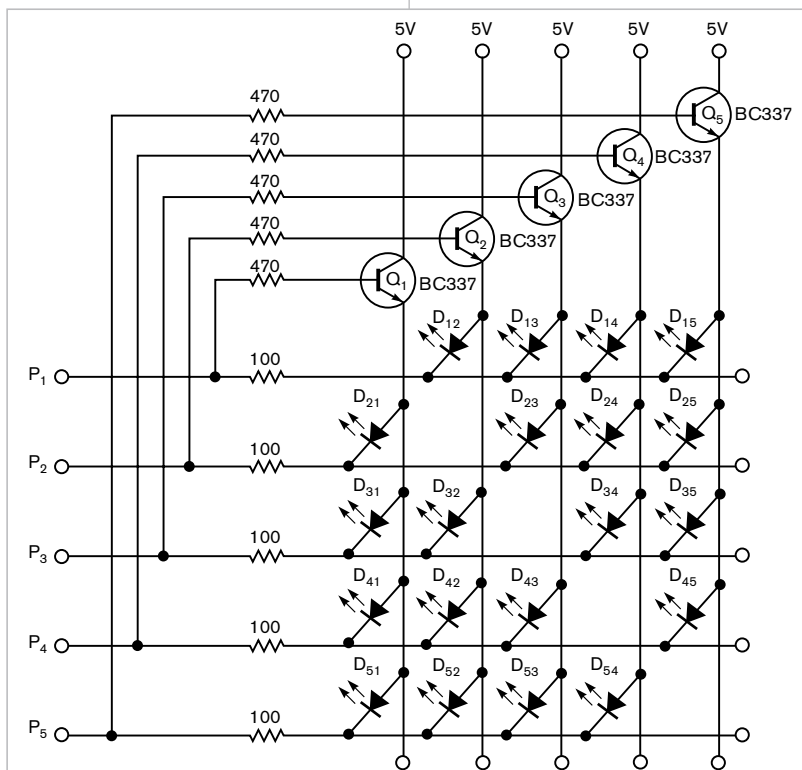
view, consider that, by the time you get to 90 LEDs, the PCB (printed-circuit-board) real estate and cost of the additional 10 transistor/resistor sets pale in comparison to the display itself.

Examining the circuit in detail, you'll notice that it has five microprocessor pins,  $P_1$  through  $P_5$ , available, for a total of  $N \times (N-1) = 20$  LEDs. When  $P_3$ , for example, is high, the emitter of  $Q_3$  is at approximately 4.4V, and you can turn off  $D_{13}$ ,  $D_{23}$ ,  $D_{43}$ , or  $D_{53}$  if you make  $P_1$ ,  $P_2$ ,  $P_4$ , or  $P_5$  low. Any pin that you set to input, or high impedance, alternatively turns off the corresponding LED. When  $P_1$  and  $P_4$  are low,  $P_3$  is high, and  $P_2$  and  $P_5$  are in high-impedance states. With  $P_3$  high, transistor  $Q_3$  biases on, all the other transistor bases are either low, which ensures that no current will flow, or high-impedance, which supplies no current into the base to allow the transistor to conduct. All the diodes in the third column can turn on, but only  $D_{13}$  and  $D_{43}$  have a path directly to ground through  $P_1$  and  $P_4$ , which are low and through the 100 $\Omega$  current-limiting resistors.

$D_{23}$  and  $D_{53}$  connect to the high-impedance input pins and can conduct only through the 100 $\Omega$  resistors attempting to turn on  $Q_2$  and  $Q_5$ . Because of their forward-voltage drop—typically, 2.2V—the emitters of  $Q_2$  and  $Q_5$  will be less than 1.6V, as the following equation shows:  $5V_{CC} - 0.6V(Q_3) - 2.2V(D_{23} \text{ or } D_{53}) - 0.6V(Q_2 \text{ or } Q_5) - I_{LED} \times 100\Omega < 1.6V$ , where  $I_{LED}$  is the current of the LEDs. This scenario does not allow any LED in Column 2 or Column 5 to light up to any level that would have an undesirable effect. EDN

## REFERENCES

- 1 Lancaster, Don, *Tech Musings*, August 2001, [www.tinaja.com/glib/muse152.pdf](http://www.tinaja.com/glib/muse152.pdf).
- 2 "Charlieplexing—Reduced Pin-Count LED Display Multiplexing," Application Note 1880, Maxim, Feb 10, 2003, <http://pdfserv.maxim-ic.com/en/an/AN1880.pdf>.
- 3 Gupta, Saurabh, and Dhananjay V Gadre, "Multiplexing technique yields a reduced-pin-count LED display," *EDN*, Oct 16, 2008, pg 68, [www.edn.com/article/CA6602447](http://www.edn.com/article/CA6602447).



**Figure 1** Arranging LEDs in a cross-point array and adding a transistor to each column show that the duty cycle of Charlieplexing is similar to that of standard multiplexing.

# Serial port tests digital circuits

Yury Magda, Cherkassy, Ukraine

A PC's serial port provides signal lines that you can use to read voltage levels of digital circuits. You can use the port to test digital TTL (transistor-to-transistor-logic)-level circuits. You just need to convert the TTL levels to RS-232 voltages, and you can add a multiplexer to increase the number of signals that the serial port can sense.

The circuit in **Figure 1** uses a MAX232 IC from Maxim (www.maxim-ic.com) to convert RS-232 voltage levels to TTL levels (**Reference 1**). A 74HC4051 from Texas Instruments (www.ti.com) lets you select any of four digital inputs and route them to the serial port (**Reference 2**). **Listing 1**, which is available with the online version of this Design Idea at www.edn.com/090625dia, lets you control the RTS (ready-to-send) and DTR (data-terminal-ready) pins in the serial port that selects the signal under test. The CTS (clear-to-send) pin then reads the signal under test into the PC.

The four digital-input signals, A0 through A3, from your device under test connect to the first four inputs, X0 through X3, of the multiplexer. Only one of those signals can pass through to the X output, Pin 3, at a time. By setting the appropriate binary code on the serial port's RTS and DTR lines, you can select the signal to pass through the multiplexer (**Table 1**).

The PC software, running on Windows XP, sequentially sets those binary combinations on the port's RTS and DTR lines and reads the digital signal on the CTS line. The software then reads the status of the selected bit and displays it when you press the "check-status" button (**Figure 2**). The code is written in Microsoft C# 2008, but it

will run on the 2005 version, as well.

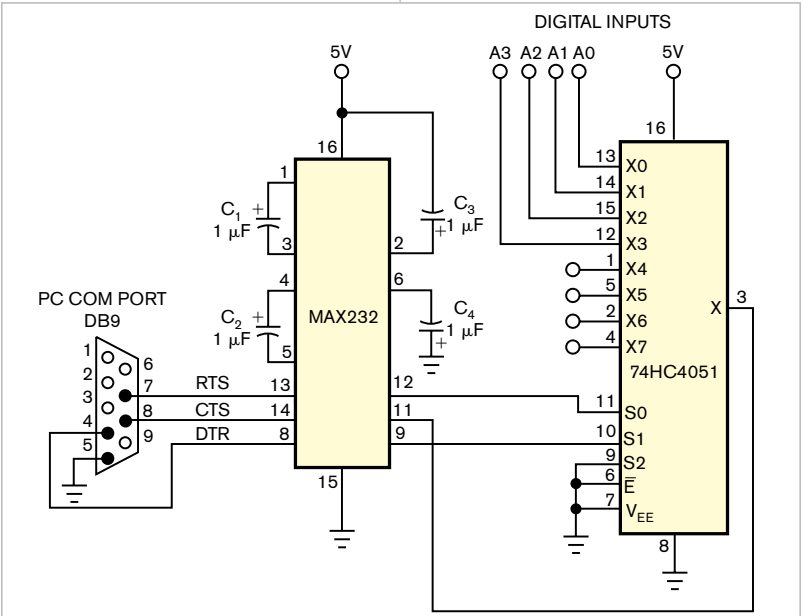
To create the application, select the "Windows Form Application" from the templates in the project wizard. Place the text-box, label, and button components on the project's main form and assign titles for them. You should place the serial-port component on the design area of the project. Then, set the appropriate parameters for the serial-port component, including the port number, baud rate, data bits, parity, and stop bits.

When you build the circuit, follow all precautions concerning the MAX232 and 74HC4051 wiring according to

the manufacturers' data sheets. Place bypass capacitors as close as possible to the IC's power and ground. You can replace the MAX232 with a MAX225 or MAX233.**EDN**

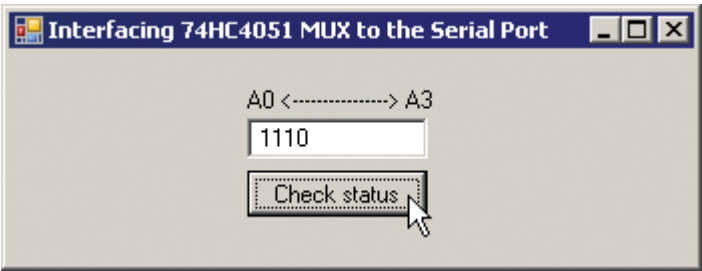
## REFERENCES

- 1 "MAX220-MAX249 +5V-Powered, Multichannel RS-232 Drivers/Receivers," Maxim, January 2006, <http://datasheets.maxim-ic.com/en/ds/MAX220-MAX249.pdf>.
- 2 "CD54/74HC4051, CD54/74HCT4051, CD54/74HC4052, CD74HCT4052, CD54/74HC4053, CD74HCT4053 High-Speed CMOS Logic Analog Multiplexers/Demultiplexers," Texas Instruments, 2004, <http://focus.ti.com/lit/ds/symlink/cd74hct4053.pdf>.



**Figure 1** This circuit lets you pass up to four TTL-level signals to an RS-232 port to read their status.

TABLE 1 INPUT SELECTION		
Signal to X pin	RTS bit	DTR bit
A0	0	0
A1	1	0
A2	0	1
A3	1	1



**Figure 2** A main window of the running application shows that input lines A0–A2 have high logic levels and A3 has a low logic level.

## DAC calibrates 4- to 20-mA output current

Ronald Moradkhan and Steven Lau,  
Maxim Integrated Products, Sunnyvale, CA

Industrial controls make heavy use of 4- to 20-mA current loops to transmit process measurements because current loops retain information in the presence of noise and changes in loop voltage. The loop circuit requires proper calibration to ensure accurate readings. The circuit in **Figure 1** calibrates the loop by generating a current in response to a control voltage:

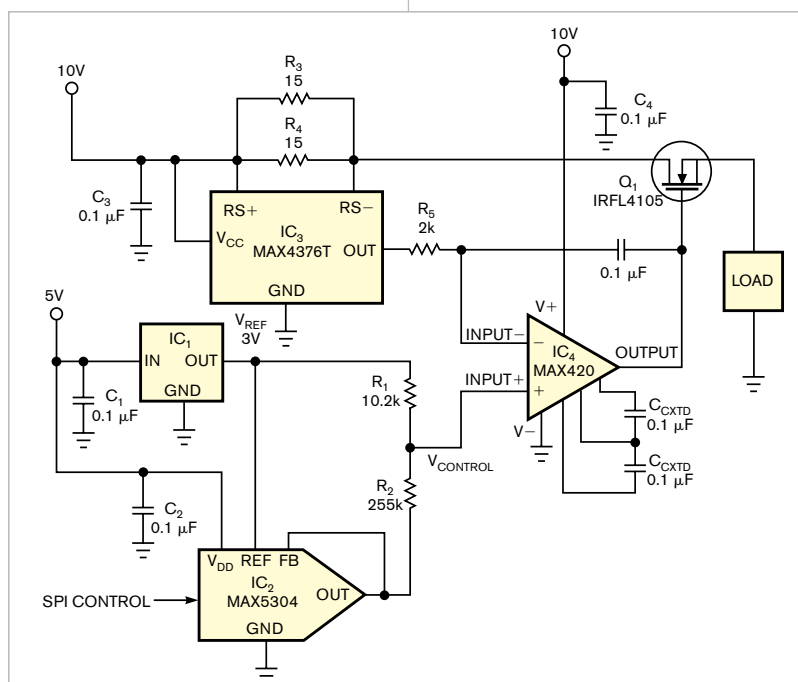
$$I_{OUT} = \frac{V_{CONTROL}}{R_{SENSE} \times K_{CSA}}$$

where  $I_{OUT}$  is the output current,  $V_{CONTROL}$  is the control voltage,  $R_{SENSE}$  is the sense resistance, and  $K_{CSA}$  is the gain of the current-sense amplifier—20 in this case. The circuit comprises  $IC_2$ , a Maxim (www.maxim-ic.com) MAX5304 DAC;  $IC_3$ , a MAX4376T current-sense amplifier;  $IC_4$ , a MAX420 op amp; and  $Q_1$ , an N-channel IRFL4105 MOSFET. The op amp lets the control voltage set the output current because it forces the voltage on

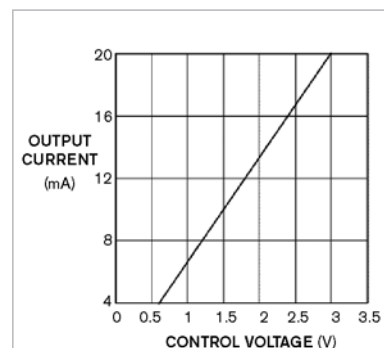
the negative input equal to that on its positive input. The output current depends on the value of the sense resistor, the gain of the current-sense amplifier, and the control voltage.

The DAC provides the control voltage that lets you automate the calibration procedure. By selecting the right value for the sense resistor and by using a suitable resistor divider for  $R_1$  and  $R_2$  at the output of the DAC, you can adjust the circuit's output to 4 mA when the DAC's digital input is zero-scale and 20 mA when the digital input is full-scale. **Figure 1** shows the component values you need to achieve that condition.

With a zero-scale digital input, the DAC output is 0V and the resistor divider produces 0.6V at the op amp's positive input, forcing the output current to 4 mA. With a full-scale digital input, both the DAC output and the midpoint of the resistor divider are at the 3V reference voltage, forcing the output current to 20 mA. A transfer curve relates the output current to the control voltage (**Figure 2**). **EDN**



**Figure 1** This DAC-controlled 4- to 20-mA transmitter allows digital control of the loop current.



**Figure 2** The circuit in **Figure 1** produces a linear output current versus digital control voltage.

## Alarm tells you to close the refrigerator door

Boris Khaykin, TRW Automotive, Livonia, MI

The circuit in **Figure 1** is a simpler and safer device than a sim-

ilar one I recently read about (**Reference 1**). A few years ago, I built the

circuit that this Design Idea describes, and the gadget still operates with the original 9V battery. The circuit operates by sensing a decrease in resistance of photocell  $PC_1$  that results from light in the refrigerator when its door is open. A counter is in a reset state when  $PC_1$  is in the dark, and its resis-

tance is greater than  $30\text{ k}\Omega$ . Usually, the dark resistance is greater than  $200\text{ k}\Omega$ , and current consumption at this state is less than  $40\text{ }\mu\text{A}$ . Oscillator-counter  $\text{IC}_1$  starts counting when  $\text{PC}_1$ 's

resistance is lower than  $15\text{ k}\Omega$ —that is, when the light bulb in the refrigerator is on. Then, in 20 seconds, it turns on a buzzer for 20 seconds or until someone closes the door. The current at this

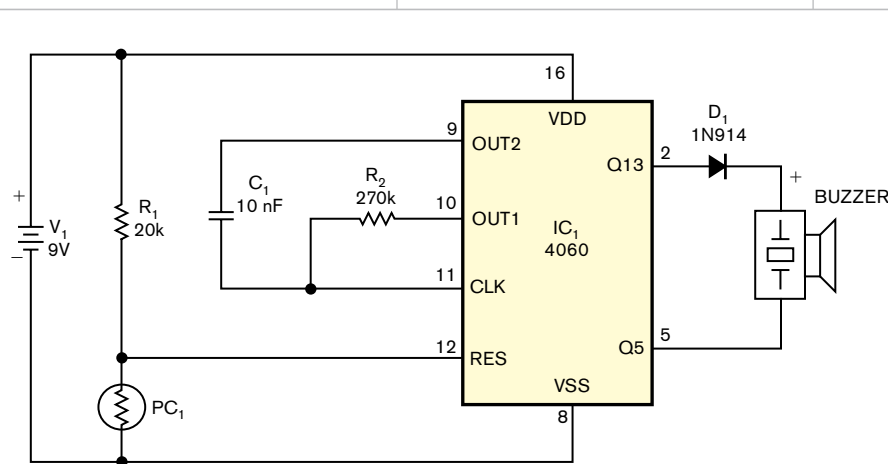
state is approximately  $2.5\text{ mA}$ .

You can use almost any photocell, such as the Jameco ([www.jameco.com](http://www.jameco.com)) 202403 CDS0018001 with  $200\text{-k}\Omega$  dark and  $3\text{-k}\Omega$  light resistance.

This circuit uses a RadioShack ([www.radioshack.com](http://www.radioshack.com)) 273-074 buzzer. You can use any similar piezoelectric buzzer with an operating dc voltage of 1.5 to 15V.  $V_1$  can be as low as 3V. The trade-off is that using a voltage this low gives you longer battery life but lower volume of sound. **EDN**

## REFERENCE

1 Babu, TA, "Alarm Sounds When Refrigerator Door Remains Open Too Long," *Electronic Design*, March 26, 2009, pg 46, <http://electronicdesign.com/Articles/ArticleD/20806/20806.html>.



**Figure 1** This gadget, placed inside a refrigerator, sounds an alarm when the refrigerator door is open for more than 20 seconds.