

# designideas

READERS SOLVE DESIGN PROBLEMS

## Program “excelerates” microcomputer-I/O allocation

Aubrey Kagan, Emphatec, Markham, ON, Canada

When I designed a system employing the LPC2138 ARM (www.arm.com)-based microcontroller, I quickly abandoned a pencil-and-paper approach to allocating the I/O. That method is tedious and error-prone because of the large number of pins on the microcontroller. Instead, I entered the data into Microsoft (www.microsoft.com) Excel (Reference 1). This approach let me assess the initial amount of I/O and any additional I/O that I would have to add. With the spreadsheet, I could create a rough bill of materials for a quote. Thereaf-

ter, it helped with the functional allocation and is an elegant and practical approach for almost any project. The online version of this article, at www.edn.com/081215dia, provides a sample spreadsheet that you can download.

First, you enter all the pins in ascending order (Column A in Figure 1). The LPC2138 can have as many as four functions per pin. Columns C to F show the functions and their corresponding pin numbers. Next, you insert the data-validation feature in each concomitant cell in Column B. When you click on a cell with this setup, a

### DIs Inside

54 Microcontroller measures resistance without an ADC

54 Five- to 10-LED flashlight circuit runs at 3V

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

drop-down arrow appears, and a selection of the cells appears to the arrow's right. You click on any cell in Column B and then click on the “data” menu and then the “validation” menu to see the setup of data validation. You format cells for which no options are available, such as  $V_{SS}$ , with a black background because, at start-up, you can delete the whole column to initialize, but the color formatting will remain.

You enter the project's I/O in the I/O-allocation table (columns J to O). You classify each pin as I (input), O (output), I/O (input/output), AI (analog input), or AO (analog output). You must allocate those pins to the microcontroller. Any I/O device that is not green is a direct user decision and not a function of anything else on the worksheet. Note that the information that appears in the pin column (Column N) is not the pin number but a reference to the pin number in Column A, so that, if you were allocating the function to Pin 8, the entry is “=A11,” as it is in Cell N6. Column 12 contains a look-up formula that fetches the function name that appears in Column B to the right of the selected pin.

The bottom of each table (cells A69 to B73 and K94 to N101) includes

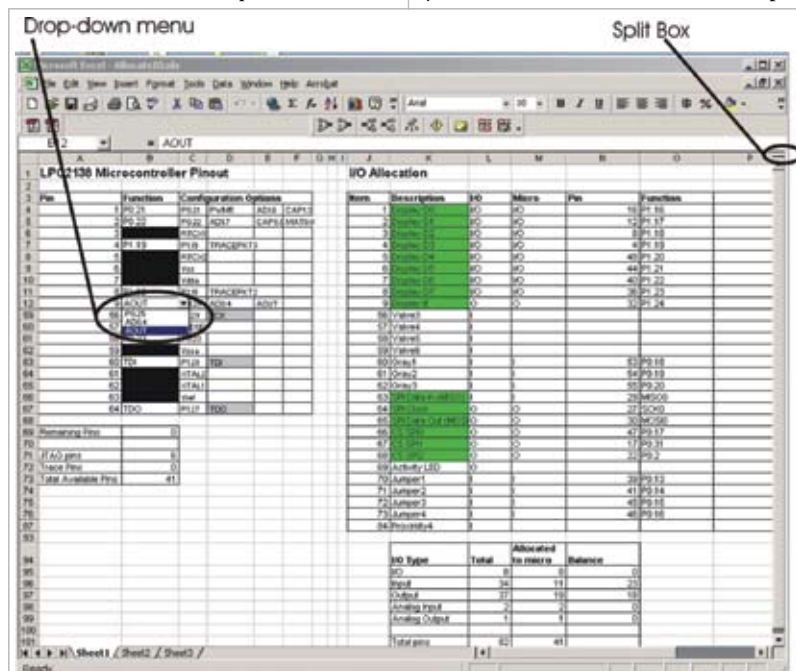


Figure 1 The completed worksheet has a large number of hidden rows to show the top and the bottom of the range.

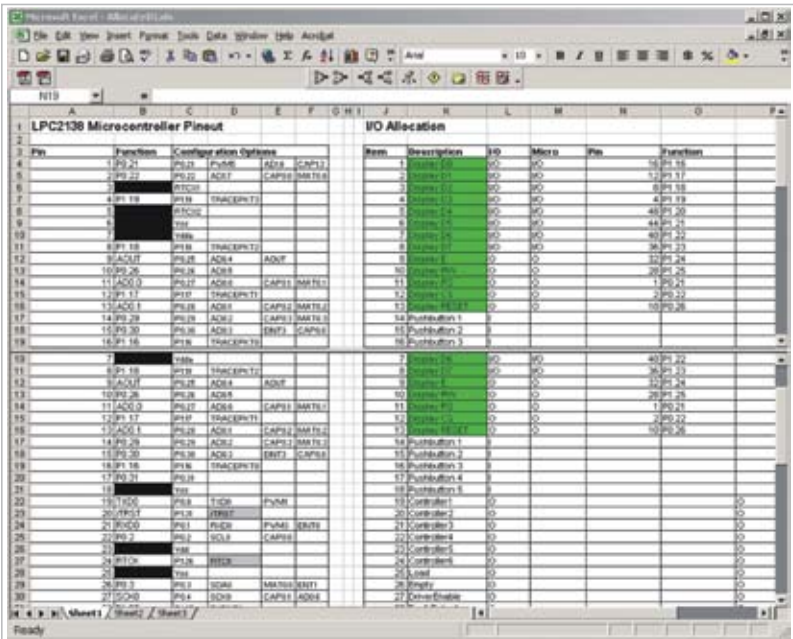
some statistics on the usage and availability of pins based on the allocation to allow you to keep tabs on the allocation as it progresses. Cell M101 has conditional formatting, so it turns red if the pins you allocate to the microcontroller exceed the total number of pins available on the microcontroller as calculated in cell B73. You can add hardware I/O to the right of the table to ensure that you include all I/O.

The usage of the spreadsheet takes place as follows:

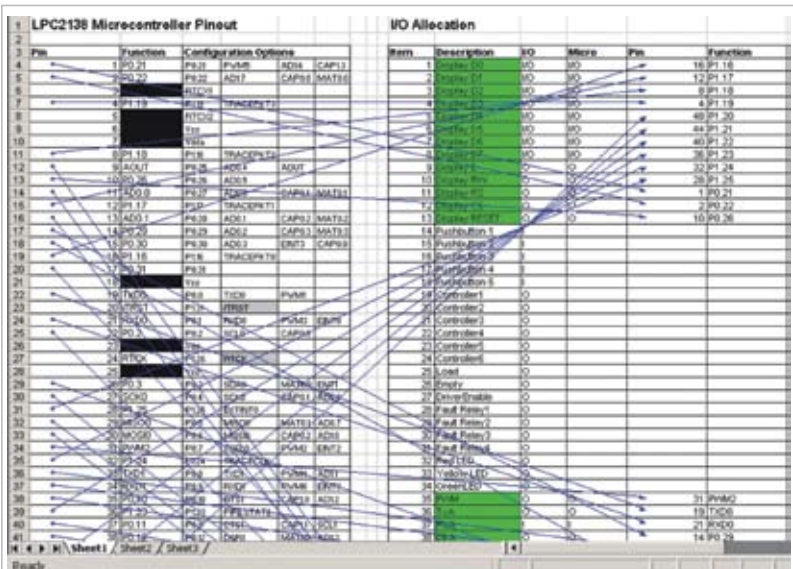
1. Delete cells B4 to B67.
2. Delete cells K4 to N87.
3. Create a list of project I/Os and fill in the I/O-allocation table. Insert rows for additional pins, remembering to update the entries in columns K and O.
4. Allocate those pins on the microcontroller that you cannot use for general I/O, such as the JTAG pins for emulation.
5. Drag down the split-box indicator so that the worksheet appears something like that in **Figure 2**.
6. In the upper pane, select the cell in Column B associated with the desired pin. Select the configuration from the drop-down box.
7. Go to the project-I/O function in Column N in the lower pane. Enter an equals sign and then click on the desired pin in Column A in the upper pane, scrolling up or down if necessary. The selected cell reference then fills into the formula. Complete the entry with the “enter” key.
8. Repeat for all the I/O.
9. Drag the split-box indicator back to the top to remove the screen split.

Some of the features in Excel can really make this model shine. For instance, the pin allocation of the LPC2138 does not follow the logical ordering of the pins. Perhaps it would help to see Port 0 listed in ascending order. You can use Excel’s sort feature to group like functions together.

To see where the information comes from, click on any entry in Column N, select the “tools” menu item, then select “auditing” and “trace precedents.” If you use this procedure with all the



**Figure 2** Two panes with the split box allow for easy pin allocation.



**Figure 3** The precedent feature lets you verify that you have allocated all the pins and that each pin has a unique assignment.

cells, you can visually trace unallocated or twice-allocated pins. A macro, “find all precedents,” which is available in the Web version of this Design Idea at [www.edn.com/081215dia](http://www.edn.com/081215dia), results in the screen in **Figure 3**. Another macro, “clear arrows,” also available on the Web site, clears all these indicators. Unfortunately, because the look-up table in Column O includes

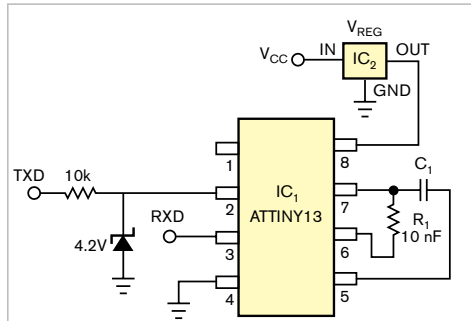
a reference to Column A, you cannot use the antecedents’ trace in the same manner.**EDN**

## REFERENCE

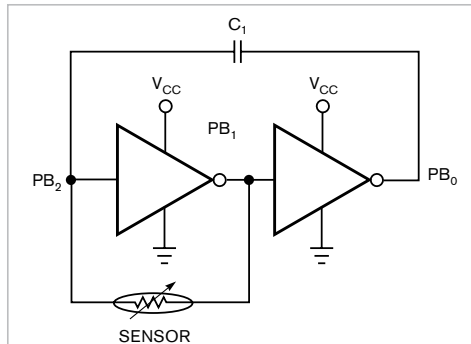
1 Kagan, Aubrey, *Excel by Example: A Microsoft Excel Cookbook for Electronics Engineers*, Elsevier/Newnes, May 2004, ISBN: 0750677562.

## Microcontroller measures resistance without an ADC

Ashish Aggarwal, Netaji Subash Institute of Technology, Dwarka, India



**Figure 1** This circuit can measure resistance by measuring the frequency of a microcontroller configured as an astable multivibrator.



**Figure 2** This design implements an equivalent oscillator based on the principle of an astable multivibrator in the Tiny13.

Sensors automate most of the processes in industry. Most of these sensors, such as those for ammonia gas, temperature, and the like, are resistive devices in which electrical resistance changes—mostly nonlinearly—as the surrounding conditions change. The sensors' resistances may vary from 1 mΩ to 10 MΩ. **Figure 1** illustrates a circuit for resistance measurement. The circuit uses an eight-pin AVR microcontroller, a Tiny13V from Atmel ([www.atmel.com](http://www.atmel.com)), for the controller. The Tiny13V works over a supply-voltage range of 1.8 to 5.5V.

This design implements an equivalent oscillator based on the principle of an astable multivibrator in the Tiny13 (**Figure 2**). The oscillator has no stable states, and the signal keeps oscillating between two quasistable states. This oscillator produces a frequency that depends on the value of the resistor. As resistance increases, frequency decreases, and you

can easily measure this frequency to yield the value of the resistance.

The resistance you want to measure connects between any two general-purpose I/O pins of the microcontroller, and a capacitor,  $C_1$ , of known value connects across the other general-purpose I/O pin. Note that  $PB_0$  and  $PB_1$  are always in different states to implement a NOT gate.  $PB_2$  measures a high or a low across resistor  $R_1$ .

Initially,  $PB_0$  is high,  $PB_1$  is low, and there is a high-impedance state at  $PB_2$ . As a result, the capacitor starts charging with time-constant  $RC$ . Note that the capacitor initially acts as a short, and  $PB_2$  senses a high. As the capacitor charges, the voltage across the resistor decreases, and, when  $PB_2$  detects a low,  $PB_1$  goes high and  $PB_0$  goes low.

Next, as the capacitor discharges, the potential across the resistor builds up, and, when  $PB_2$  detects a high,  $PB_0$  goes high and  $PB_1$  goes low. In this fashion, measuring the frequency or half the number of toggles of  $PB_0$  in a second gives an inverse relation of resistance,  $R_1$  (in **Figure 1**), with frequency,  $f$ :  $R_1 = k/f$ , where  $k$  is a proportionality constant. The result travels to a PC through a serial RS-232 interface. Because the Tiny13 has no UART, a software UART program and the program for measuring resistance are available with the Web version of this Design Idea at [www.edn.com/081215dib](http://www.edn.com/081215dib). **EDN**

## Five- to 10-LED flashlight circuit runs at 3V

GY Xu, XuMicro, Houston, TX

Almost all inexpensive commercial LED flashlights use a 4.5V power supply—three AA or AAA batteries—because white LEDs require 3.3 to 3.5V to fully turn on. Thus, there is a voltage gap between LEDs and traditional 3V incandescent-flashlight bulbs. The voltage difference makes for a difficult—but not impossible—transition from the old flashlight to an LED flashlight. The simple circuit in **Figure 1** solves this problem.

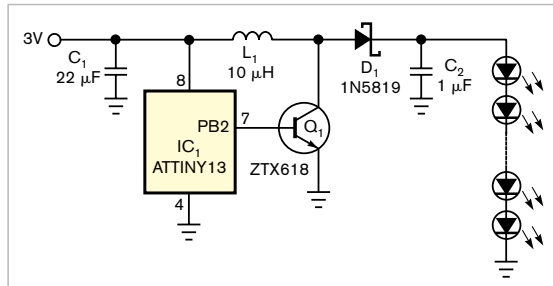
The circuit is just a typical voltage booster comprising six components that you can mount on a small PCB (printed-circuit board) measuring less than 1 in<sup>2</sup>. Component selection and their values are, however, important.  $IC_1$ , an Atmel ([www.atmel.com](http://www.atmel.com)) ATtiny13 microcontroller, works as a charge pump for boost control. Its internal oscillator frequency is 1.2 MHz at 3.5V, and it can operate with voltages as low as 1.8V with low power con-

sumption. The ATtiny13 has a small, eight-pin footprint.

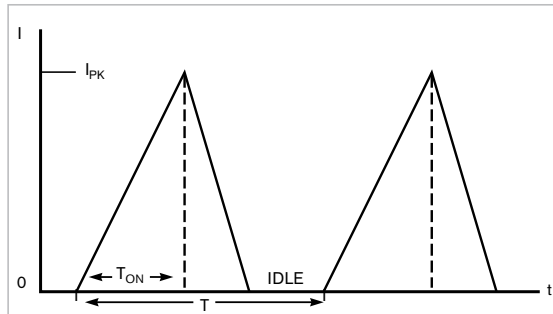
$Q_1$  is a low-saturation-voltage ZTX618 NPN transistor that can handle more than 3A of collector current.  $D_1$  is a Schottky diode with low forward-voltage drop to achieve high efficiency. When you apply the 3V supply-voltage power to  $IC_1$ ,  $IC_1$  outputs a high pulse that turns on  $Q_1$ . Its collector is effectively grounded. Inductor  $L_1$  charges linearly from 0A to some peak current until  $IC_1$  outputs a logic low, and  $Q_1$  then turns off (**Figure 2**). This circuit works only when the inductor is not saturated, so choosing the right inductor is important. At that mo-

ment, the established magnetic field in  $L_1$  collapses, causing a reverse induced voltage that makes  $D_1$  conduct. The energy in  $L_1$  transfers to  $C_2$ , which stores the energy until it is sufficient to light up the LEDs. The relationship between the supply voltage ( $V_{IN}$ ), the inductor ( $L$ ), its peak current ( $I_{PK}$ ), and the microcontroller's on time ( $T_{ON}$ ) is  $V_{IN} = L \times I_{PK} / T_{ON}$ .

For a supply voltage of 3V, you should select an inductor with a nominal value of  $10 \mu\text{H}$  and a saturation current larger than 1.5A. You can calculate the microcontroller's on time as  $5 \mu\text{sec}$ . **Listing 1**, which is available in the Web version of this Design Idea, at [www.edn.com/081215dic](http://www.edn.com/081215dic), uses this value for the charge pump's on time. The program in **Listing 1** is so simple that it takes only 22 bytes of the 1-kbyte



**Figure 1** A charge-pump circuit creates the boosted voltage to light LEDs for a flashlight.



**Figure 2** During the on time, current flows through the inductor and then charges the capacitor.

program memory. The charge-pump-control function is easy to understand. The instruction Sbi portb, 2 tells the microcontroller to output a logic high to turn on the charge pump. Because the microcontroller works at 1.2 MHz by its internal oscillator, each NOP (non-operation) takes one clock cycle, or  $0.83 \mu\text{sec}$ , to execute, so the on time is  $5 \mu\text{sec}$ . Similarly, Cbi portb, 2 tells the microcontroller to output a logic low that turns off the charge pump.

Measurement shows that the circuit works at a 100-kHz switching frequency and that the actual output is 17V/35 mA for five LEDs and 32V/20 mA for 10 LEDs. Unlike the usual voltage-booster circuit, this circuit needs no resistor, which wastes energy and generates useless heat, as a voltage divider or a sensor. **EDN**