design ideas

Edited by Bill Travis

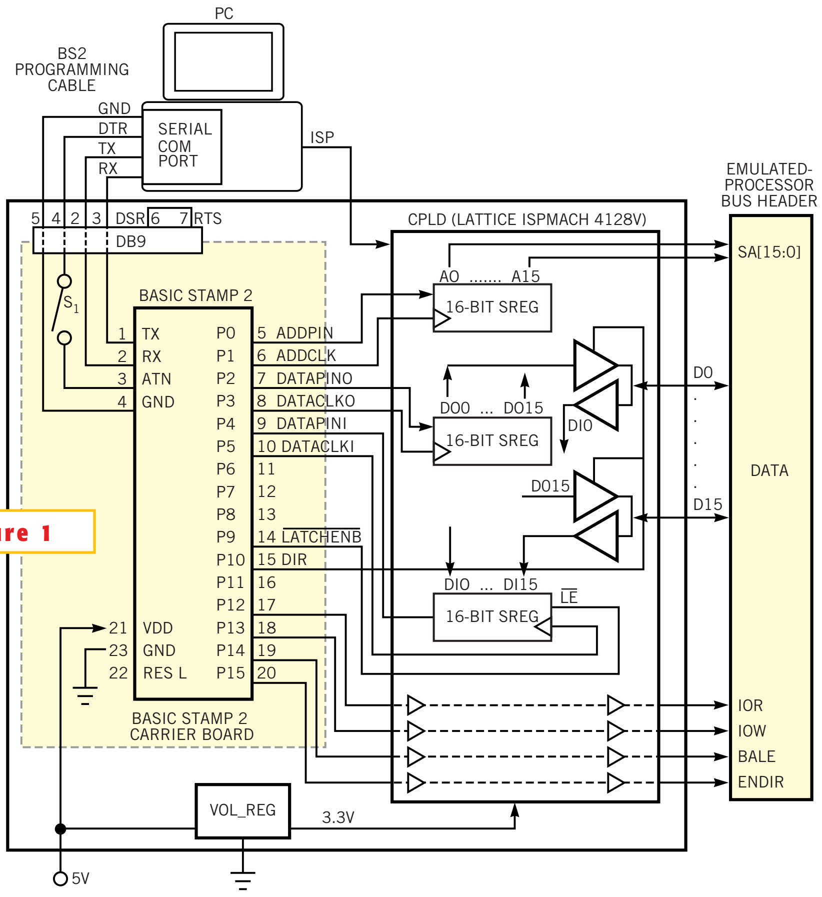
Simple emulator speeds testing

Navdhish Gupta, Chicago, IL

Designing and testing embedded hardware can be frustrating if you have to rely on somebody else's perhaps-unready firmware to test your hardware. Often, hardware is ready for testing before debugging and system firmware are available from software developers. Microprocessor emulators are solutions but are often expensive, hard to use, and sometimes inappropriate for use by hardware developers, who often just need to read and configure registers on devices on their boards. If you are not primarily concerned with your processor's bus performance and need simple read and write access to registers on chips on your board to configure and test hardware, then building a simple processor-bus emulator, such as the one in **Figure 1,** may be an attractive option. The simple emulator shown uses an easy-to-use Basic Stamp microcontroller (www.parallax.com) on a carrier board and a small CPLD to emulate a 16-bit "ISA-like" I/O bus. **Figure 2** shows the timing parameters for the bus (**Figure 2a** shows write-cycle timing; **Figure 2b** shows read-cycle timing). Note that the bus emulator runs much slower than a "normal" processor bus but is useful to read and configure registers that you need to test hardware. This design uses the 16-bit ISA-like bus mainly for illustrative purposes; you could emulate almost any processor bus with a similar setup.

Publish your Design Idea in *EDN*. See the What's Up section at www.edn.com.

Figure 1



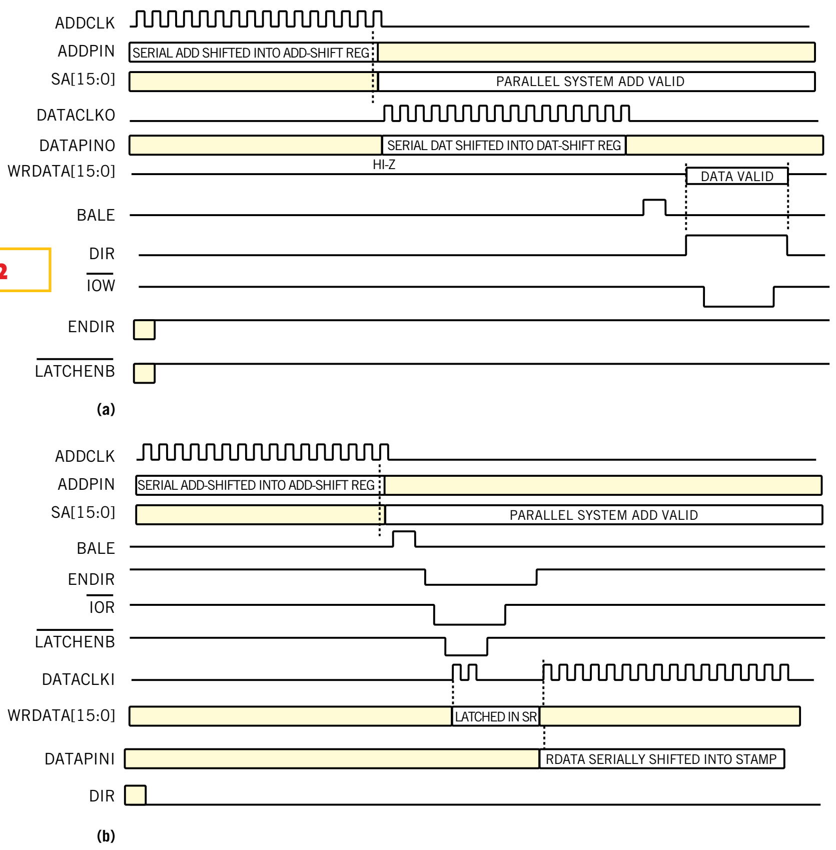
A simple emulator uses a Basic Stamp microcontroller and a small CPLD.

The simple emulator consists of two parts, the hardware (**Figure 1**) and the Basic Stamp firmware (**Listing 1**, which you can download from the Web version of this Design Idea at www.edn.com). You must connect the emulator itself to a 5V power supply and a PC with a keyboard, monitor, and serial port, and you load any simple terminal-emulator software, such as Hyper Term. The serial programming cable you use to program the Basic Stamp also communicates with the terminal-emulator software by inserting a switch, S_1, that lets you disconnect or connect the DTR/ATN connection. When the switch is closed, you can program the Basic Stamp, and, when it is open, you can use the terminal-emulator software to communicate with the Basic Stamp. You enter commands on the keyboard, and the results appear on the monitor. The emulator connects to the board and devices to test and configure, such as a custom FPGA, through the board's normal processor bus. (You need to either socket the original processor or provide a test port on the board.) You tristate the original processor or remove it from the board to use the emulator.

The Basic Stamp firmware emulates processor-bus cycles by changing the appropriate control signals on its pins. The

CPLD uses shift registers to interface the serial address and data into and out of the Basic Stamp to the emulated 16-bit parallel address- and data-bus signals. The CPLD also "conditions" the control signals from the Basic Stamp to the emulator header by performing logic-threshold conversion through CPLD I/O buffers. **Listing 2**, also available from the Web version of this Design Idea at www.edn.com, shows the Verilog code for the CPLD. Note that, although the control signals IOW, IOR, BALE, and ENDIR come from the basic code firmware and the Basic Stamp pins and are conditioned through the CPLD; they could come directly from a simple finite-state machine in the CPLD if your design requires more realistic bus timing. The Lattice (www.latticesemi.com) ispMACH 4128V CPLD is a 5V-tolerant device whose inputs you can safely drive with voltages as high as 5.5V. It also supports as many as 64 I/O lines and is in-system-programmable through the IEEE-standard 1532 interface. These features make the CPLD a good choice for use in the hardware implementation of the logic needed in the emulator.□

Figure 2


ADDCLK
ADDPIN
SERIAL ADD SHIFTED INTO ADD-SHIFT REG
SA[15:0]
PARALLEL SYSTEM ADD VALID
DATACLKO
DATAPINO
SERIAL DAT SHIFTED INTO DAT-SHIFT REG
WRDATA[15:0]
HI-Z
DATA VALID
BALE
DIR
IOW
ENDIR
LATCHENB
(a)
ADDCLK
ADDPIN
SERIAL ADD-SHIFTED INTO ADD-SHIFT REG
SA[15:0]
PARALLEL SYSTEM ADD VALID
BALE
ENDIR
IOR
LATCHENB
DATACLKI
WRDATA[15:0]
LATCHED IN SR
DATAPINI
RDATA SERIALLY SHIFTED INTO STAMP
DIR
(b)

The simple emulator uses Basic Stamp signals for write-cycle timing (a) and read-cycle timing (b).

Software snippet provides improved subset-sum algorithm

Ivan Basov, Brandeis University, Waltham, MA

The subset-sum problem is one of the most frequently occurring NP (nondeterministic, polynomial-time)-complete) problems. It asks whether a subset of numbers in a set of positive integers adds up exactly to a given value. A relaxed version of the problem tries to identify a subset of numbers that adds up to a maximum value no greater than a given value. This problem arises in transportation, network design, scheduling, logistics systems, robotics, and many other areas. The problem permits you to develop and illustrate the power of different algorithmic tools. The problem is as follows:

Given a set of positive integer values W[1], W[2], ...W[m] and an integer n.0, does a subset of the values add up to exactly n?

A well-known pseudo-polynomial algorithm (**Reference 1**) defines a table: T[ij], 1≤i≤m and 1≤j≤n, to be T[ij]=true if and only if a subset of W[1],...W[i] sums to exactly j. The algorithm uses O(mn) time to fill in a table that uses O(mn) space. **Table 1** with n=13 shows the true entries and leaves the false ones blank. This Design Idea proposes an improved algorithm that uses O(mn) time to fill in an array that uses only O(n) space:

```
SubsetSum(W, m, n)
1. Define a bit array A[j], 1 ≤ j ≤ n.
2. Initialize the array to zeros.
3. A[0]:=1.
4. for i:=1 to m do.
5. for j:=n to 0 do.
6. if A[j]=1 then A[j1W[i]]:=1.
7. return A[n].
```

You can easily prove that the returned value is 1 if and only if a subset of the weights adds up to exactly n. The proof is analogous to the one of the original O(mn)-space algorithm. The following routine implements the above algorithm in C++. It just shifts a bit map m times by W[j] bits and applies a bitwise OR operation with the bit map from the previous step.

```
int SubsetSum(int W[], int m, int n)
{
bit_vector x=1;
for(i=1; i <=m; i++) x |=x<<W[i];
```

```
    return (x>>n) &1;
}
```

The bit_vector class overloads bitwise operators and behaves as an (n+1) bits integer (with bits ranging from 0 to n). Now, consider a low-density subset-sum problem, the case in which the above algorithms produce a bunch of zeros and only a few ones in the bit array. You use a dynamically growing linked list and waste no space for "empty" elements:

```
SubsetSumLD(W, m, n)
1. Define, a linked list with only one element with value 0 and with the
   Head being equal to the Tail.
2. for i:=1 to m do.
3. for Element :=Head to Tail do.
4. if Element.value+W[i]≤ n.
5. Insert(Element.value+W[i]).
6. if Head.value=n.
7. return 1.
8. else.
9. return 0.
```

TABLE 1—SUBSET-SUM ARRAY

i/j	0	1	2	3	4	5	6	7	8	9	10	11	12	13
W[1]=1	T	T												
W[2]=9	T	T								T	T			
W[3]=5	T	T				T	T			T	T			
W[4]=3	T	T		T	T	T	T		T	T	T		T	T
W[5]=8	T	T		T	T	T	T		T	T	T	T	T	T

The function Insert(value) inserts the value into the list in descending sorted order. The function does nothing if an element with the same value already exists. The disadvantage of the subset-sum algorithm is that it solves only a decision—a yes-or-no problem—and doesn't allow restoring the partition itself. To overcome this disadvantage, you can use an array of integers instead of an array of bits and store the number of ones in the corresponding element. This solution requires O(n log(m)) space, and the array represents the sum of the rows of the table T[ij] of the original O(mn)-space algorithm.□

REFERENCE

1. Garey, Michael R and Johnson, David S, *A Guide to the Theory of NP Completeness*, Freeman, San Francisco, CA, 1979.

IC removes nonlinear temperature effects

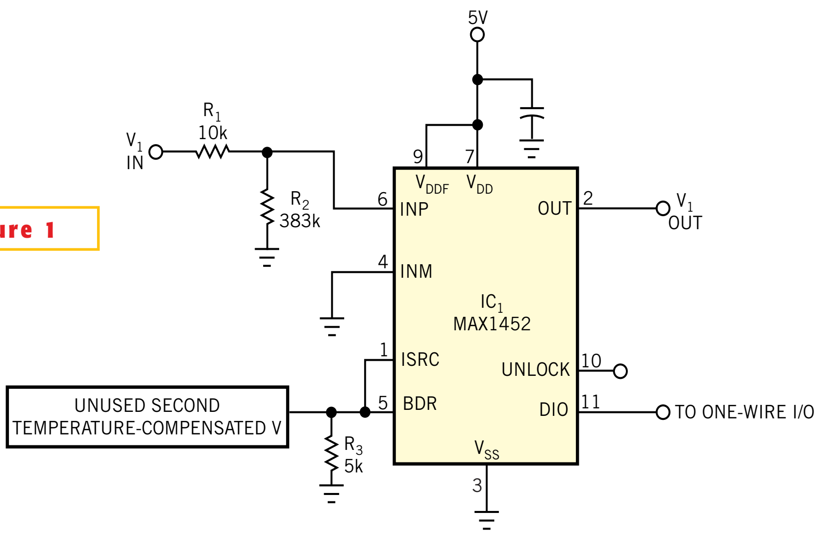
Kevin Schemansky, Maxim Integrated Products, Howell, MI

A COMMON TECHNIQUE for removing nonlinear temperature-related effects from a dc voltage is to incorporate a temperature sensor that a microprocessor samples via an A/D converter. The processor determines—by mathematical calculation or a temperature-indexed look-up table—the amount of adjustment is necessary at a given temperature and delivers the resulting value via a D/A converter. Though effective, that approach incurs design effort, cost, and multiple ICs. An alternative technique uses a single IC in an unorthodox fashion (**Figure 1**). The chip, intended as a low-cost interface to a Wheatstone bridge, normally provides precision analog-signal processing along with digitally programmable compensation of gain, offset, and temperature effects. Much of its capability remains unused in this application, but its size and pre-engineered internal circuitry can readily add nonlinear temperature compensation to a design.

IC_1 contains a bandgap temperature sensor whose output is digitized by an 8-bit A/D converter, producing a digital representation of die temperature with 1.45°/bit resolution. An internal 768-byte EEPROM, read/write accessible via the one-wire asynchronous DIO pin, provides the user with two independent temperature-indexed look-up tables. (This design uses only one.) For the table in use, an 8-bit temperature register indexes which of 176 16-bit stored values is applied to an internal 16-bit D/A converter. Because the temperature-indexing boundaries of −69 to +184°C are outside the IC's operating range of −40 to +125°C, the IC can compensate for brief temperature excursions outside its normal operating range.

The **Figure 1** circuit removes temperature-variable nonlinear offsets from the analog voltage, V_1IN. First, you program IC_1 for its minimum gain, 39V/V, so the

Figure 1



IC_1 introduces nonlinear temperature compensation to the analog voltage, V_1.

R_1/R_2 attenuation provides an overall gain of unity. (The internal PGA provides a gain range of 39 to 264V/V.) Next, program the internal coarse-offset DAC with 0000 binary and the offset-TC DAC with 0000hex, to ensure that they have no effect on the V_1 input voltage. To generate a compensated output (Pin 2), the output stage adds in the temperature-indexed value of a third DAC (the offset DAC), whose 16-bit resolution achieves adjustments as fine as 74 μV. After calculating coefficients for the offset DAC, you program them into the IC via the DIO pin. The equation for output voltage is $V_1OUT=39(V_1IN/39)+V_{DD}$(offset-DAC value/65,536)(offset-DAC sign).□

Motor controller uses fleapower

Anthony Smith, Scitech, Biddenham, Bedfordshire, UK

A simple, permanent-magnet dc motor is an essential element in a variety of products, such as toys, servo mechanisms, valve actuators, robots, and automotive electronics. In many of these applications, the motor must rotate in a given direction until the mechanism reaches the end of travel, at which point the motor must automatically stop. Although you can use microswitches to stop the motor at the end of travel, their size, weight, and cost can be prohibitive, particularly in low-cost, portable items. The circuit in **Figure 1** implements a low-cost, micropower, latching motor controller that uses current sensing rather than switches to stop the motor. The design is optimized for a supply voltage of 3 to 9V, making it well-suited to battery-powered applications. To understand how the circuit works, assume that cross-coupled flip-flops IC_{1A} and IC_{1B} are both in a reset state, such that the D input of each one is high. Because both Q outputs are low, the H-bridge transistors, Q_1 to Q_4, are all off, and the motor is idle.

A momentary closure of the Forward switch latches the Q output of IC_{1A} high and turns on MOSFET Q_2. This action, in turn, provides bias for Q_3, causing the motor to run in the forward direction. The circuit is now latched, and, because IC_{1B}'s D input is now low, any closure of the Reverse pushbutton has no effect on the H-bridge, and the motor continues running in the clockwise direction. IC_2 contains a comparator and a bandgap reference-voltage (nominally 1.182V) output at the Reference terminal (Pin 6). The motor's armature current generates a sense voltage, V_{SENSE} , across current-sense resistor R_{SENSE}. The comparator compares V_{SENSE} at the IN+ input to a reference voltage, which R_5 and R_6 set, at the IN− pin. Under normal running conditions, V_{SENSE} is lower than the voltage at the IN− input, and the comparator's output at Pin 8 is low. However, when the mechanism hits an end stop at the limit of travel, the motor's armature current increases rapidly, causing a corresponding rise in V_{SENSE}. The comparator detects this increase, and the comparator's output goes high, thus resetting

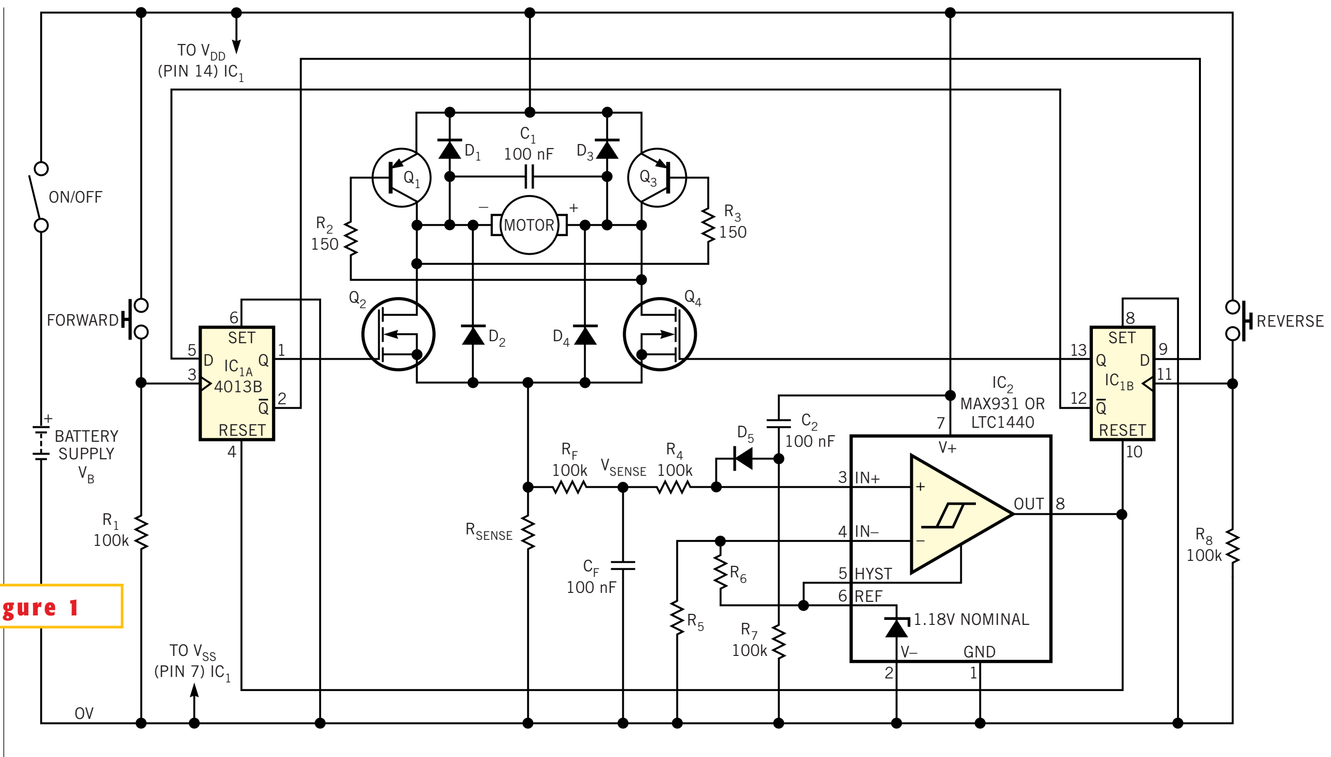
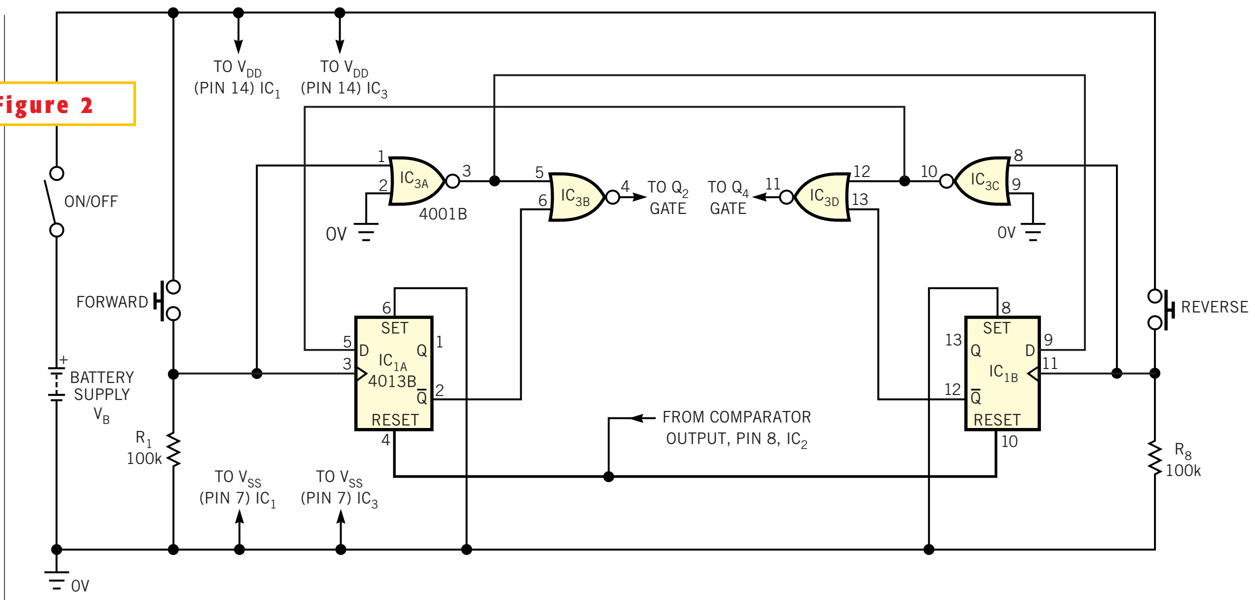


Figure 1

This latching motor controller uses current sensing, rather than switches, to stop the motor.

Figure 2



This addition to the circuit in Figure 1 yields a nonlatching controller, in which the motor runs only when the associated switch is closed.

flip-flop IC_{1A}. Transistors Q_2 and Q_3 now turn off, and the motor stops running.

Because IC_{1B}'s D input is now high, closing the Reverse switch latches IC_{1B}'s Q output high. This action turns on Q_4 and Q_1 in the opposite arm of the H-bridge, such that the motor now runs in the counterclockwise, reverse direction. IC_{1A}'s D input is now low, thereby disabling the Forward pushbutton, and the motor continues to run in the reverse direction until the mechanism hits the opposite end stop. Once again, the comparator detects the increase in armature current, and the comparator resets IC_{1B} and turns off the motor. The motor's cutoff threshold is a function of R_{SENSE}, R_5, and R_6, and the bandgap voltage at the REF pin. R_{SENSE} should be in the region of a few ohms to ensure that V_{SENSE} is also small when the motor is running under normal loading conditions. Keeping V_{SENSE} low makes practically all of the supply voltage available to the motor and ensures that the gate-source voltage of Q_2 and Q_4 is as large as possible. You should choose values greater than 300 kΩ for R_5 and R_6 to minimize the loading on the REF pin. In noisy environments, it may be necessary to decouple the IN− pin.

Filter components R_F and C_F serve a dual purpose. They are necessary to smooth the relatively "lumpy" voltage appearing across R_{SENSE} and are also essential to prevent the motor's in-rush current at turn-on from tripping the comparator. Values of 100 kΩ and 100 nF should be adequate, but some experimentation may be necessary to determine the best time constant for your application. Reset components C_2, D_5, and R_7 ensure that the comparator's output is high at power-up. You should select all transistors in the H-bridge to produce minimal saturation voltage when the devices conduct the maximum motor current. Choose low-$V_{CE(SAT)}$ devices for Q_1 and Q_3, and make sure that MOSFETs Q_2 and Q_4 can receive full enhancement at the minimum operating voltage. Depending on the type of MOSFETs you use, you may need to add a resistor in series with each gate to prevent spurious oscillation.

You should select resistors R_2 and R_3 to provide adequate base drive for Q_1 and Q_3 at the lowest supply voltage. Depending on the application, it may be possible to replace Q_1 and Q_3 with p-channel MOSFETs, in which case R_2 and R_3 can be fairly large. Freewheeling diodes D_1 to D_4 are necessary to commutate the currents generated by the motor's back-EMF at turn-off. You may require capacitor C_1 to suppress any noise that the motor's brushes produce. The circuit's quiescent, motor-idle current drain is extremely low, making it ideally suited to battery-powered applications. Measurements on a breadboard circuit operating from a 9V supply voltage reveal a quiescent current of just 9.5 μA. You can adapt the circuit to produce a nonlatching version in which the motor runs in the forward or the reverse direction only while the associated switch is closed. However, this approach is not simply a matter of gating the comparator's output with each switch; that action merely causes the motor to hiccup at the cutoff point. Instead, you must again latch the comparator's output and add some extra logic in the form of a quad NOR gate. **Figure 2** shows the arrangement, in which NOR gates IC_{3B} and IC_{3D} control the H-bridge, and the comparator controls the flip-flop's reset pins as in **Figure 1**.

The motor runs in the forward direction only while you depress the Forward button; releasing the pushbutton stops the motor. While the switch is closed, IC_{3A} holds IC_{1B}'s D input low, thereby ensuring that the Reverse switch can have no effect on the motor while the Forward switch is closed. Once you release the Forward switch, you can use the Reverse switch to reverse the motor's rotation. Again, the motor runs only while you depress the pushbutton. You can see from the symmetry of the circuit that closing the Reverse switch locks out any operation of the Forward switch. When the mechanism hits an end stop in either direction, the comparator resets the relevant flip-flop and shuts off the motor.□