

Synchronizing controller detects baud rate

WILLIAM GRILL, RIVERHEAD SYSTEMS, LITTLETON, CO

A simple and inexpensive implementation using an eight-pin 12C508 controller (Microchip Technology, Chandler, AZ) provides both bit-rate detection and a synchronous, appended-clock output from an asynchronous input-data stream (Figure 1a)

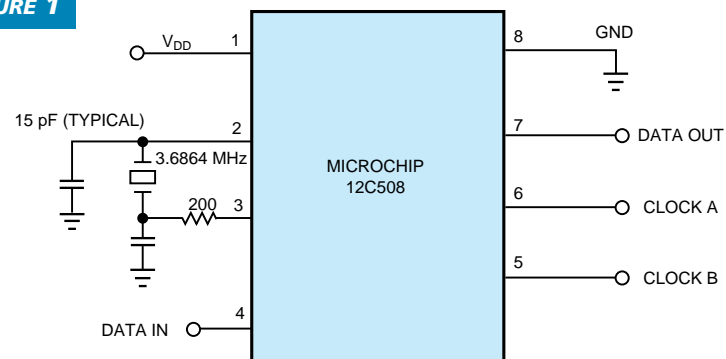
This implementation relies on two code sequences (Figure 1b). A training sequence first identifies the bit rate. This sequence begins with register and counter initialization followed by a gap detection. Following the detected, 0.325-sec, high-true gap, the controller monitors eight transitions. Between each transition, a local loop counter counts the number of loops necessary to arrive at the next transition. The controller tests the accumulated number of loops between transitions to identify the count period that resulted in the fewest loops. After processing all eight transitions of the training sequence, the controller evaluates the shortest maintained count according to a table index based on the training loop's code length. Because the controller's timing is crystal-based and the loop's path lengths are equal, the processed count corresponds to the bit rate that the controller then uses to define a delay necessary for the second code sequence.

The second output-code sequence begins by locating a fixed, second gap of approximately 40 msec. This time allows the controller to completely terminate the training sequence before beginning the controller's main iterative loop. The sequence then takes the delay, identified earlier, and processes the serial input data on Pin 4 to the output at Pin 7 while maintaining complementary clocks on pins 5 and 6. Equalizing the instruction paths for this code sequence is a key requirement of this application.

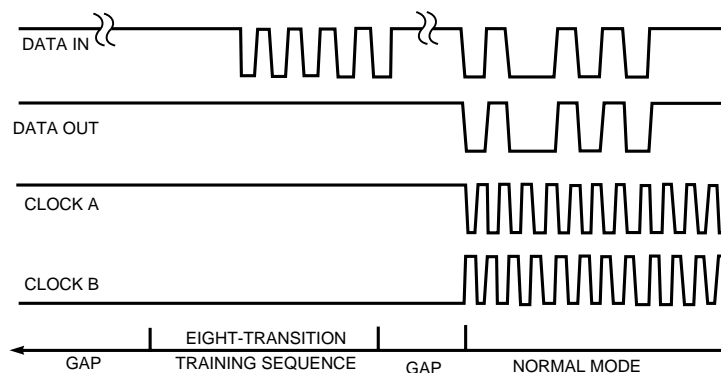
The controller uses a 3.6864-MHz crystal to provide precise baud-rate timing. This scheme allows the combined code applications to support 300- to 9600-bps asynchronous inputs. An 8-bit training pattern of 55 (hex) allows multiple single-bit isolated transitions to qualify the detected bit rate (Figure 1b)

The output-sequence code provides resynchronization of the clock-related delay counters at the input data-transition

FIGURE 1



(a)



(b)

A simple controller (a) provides baud-rate information and a synchronous output clock by using two code sequences (b).

edges on Pin 4. This resynchronization allows flexibility in supporting data-bit edge delays and distortion and variations in the controller's crystal or external timing reference's accuracy. You can download applicable code from EDN's Web site, www.edn-mag.com. At the registered-user area, go into the Software Center to download the file from DI-SIG, #2158. (DI #2158)

Emissions killers trap common-mode currents

GLEN CHENIER, FUJITSU NETWORK COMMUNICATIONS, RICHARDSON, TX

An unshielded twisted-pair cable that is transformer-coupled to a digital system can easily act as a radiating antenna, not because of the differential analog signal the cable carries, but because of common-mode currents induced by unwanted stray coupling from the digital portions of the system. These currents from fast digital transitions contain harmonics in the hundreds of megahertz and can be a nightmare to design engineers who have to make systems conform to radiated-emissions limits.

If the coupling transformer has a center tap on the winding to which the cable attaches, you can use this tap to reduce the level of these nasty common-mode currents on the cable. Connecting the tap to a quiet earth ground provides a path to shunt these currents harmlessly to earth before they can sneak out the cable and radiate (Figure 1). A capacitor in the connection provides the same RF grounding function but presents a high impedance to any 60-Hz ground loop currents if the far end of the cable also connects to a ground-referenced transformer winding. This capacitor should be only a few hundred picofarads, must have short leads and circuit traces between transformer tap and good earth ground, and must have a sufficiently high voltage rating to withstand high-voltage transients as the end market requires.

The technique works as follows: The opposite ends of the transformer winding are balanced with respect to ground; that is, the windings push and pull with equal amplitude but opposite polarity on each and every transmitted data symbol. The center of the transformer is the pivot on which the winding balances. As such, this pivot point is neutral relative to ground; an actual connection to ground makes no difference to the differ-

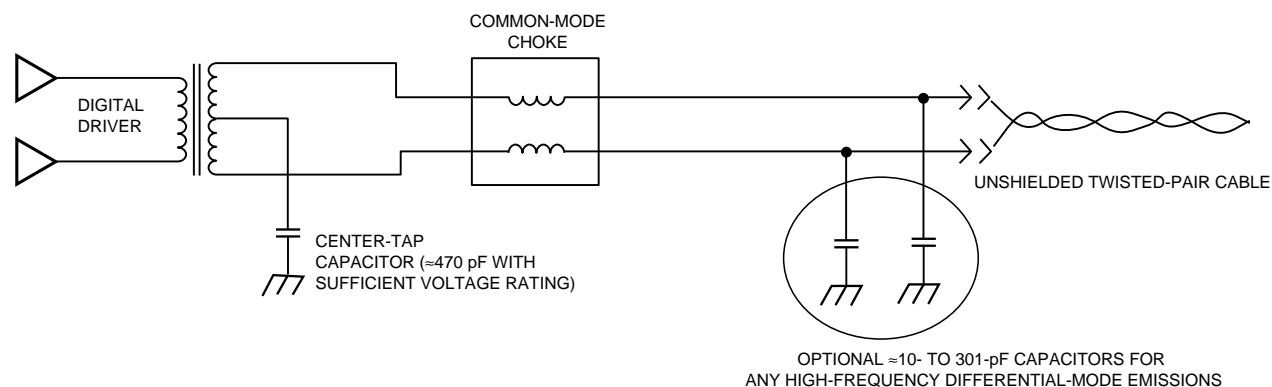
ential information signal.

If a common-mode signal impresses both conductors, the resulting currents at opposite ends of the winding flow both toward and away from the center tap in the same phase. This flow causes magnetic cancellation between the two halves of the winding, and the resulting inductance is very low, resulting only in the residual leakage inductance. In this way, both conductors have a low-impedance path to earth ground without affecting the wanted differential signal. Note that filtering each conductor with an RC network also provides the low-impedance path to ground; unfortunately, this filter also destroys the differential signal in high-bit-rate applications.

The technique in Figure 1 also helps reduce susceptibility to common-mode currents that external fields induce; the unwanted currents pass harmlessly through each half of the transformer winding and cancel each other out. Interwinding capacitance the usual mechanism by which common-mode voltages can affect transformer-coupled receiver inputs is less critical because both conductors have a low-impedance path to ground, resulting in minimum common-mode voltage on each conductor.

Using a common-mode choke in addition to the center-tap trap results in a real common-mode killer. The two techniques complement each other, and it can be helpful to use both together in stubborn cases. As Figure 1 indicates, you can place the common-mode choke virtually a transformer on its side in line with the cable, preferably at a point just before the cable exits the (ideally) shielded enclosure to avoid stray-noise pickup on the cable after the choke. A similar but opposite mag-

FIGURE 1



A center-tap capacitor connected to earth ground implements a common-mode current trap and reduces RF emissions. A common-mode choke in addition to this center-tap trap results in a common-mode killer.

netic magic takes place in the common-mode choke, which must present a high series impedance instead of a low shunt impedance to common-mode currents. The winding turns ratio is 1-to-1, and the polarity is such that the magnetic fields from the differential signal now cancel, resulting in almost zero attenuation other than that resulting from the leakage inductance. On the other hand, the common-mode currents cause magnetic addition, which results in high impedance and reduces the level of unwanted currents.

You can also make a common-mode choke by slipping a large ferrite sleeve over the two conductors of the twisted pairs or by winding one or more turns of the twisted pairs through a large toroid doughnut. Many ferrite suppliers make these

sleeves and toroids just for this purpose. Also, well-balanced common-mode chokes of the more conventional transformer-like construction are also readily available from datacomm-transformer suppliers.

Two capacitors following the common-mode choke can reduce high-frequency differential-mode emissions caused by non-common-mode currents. (DI #2160)

To Vote For This Design, Circle No. 345

Isolated driver forms solid-state circuit breaker

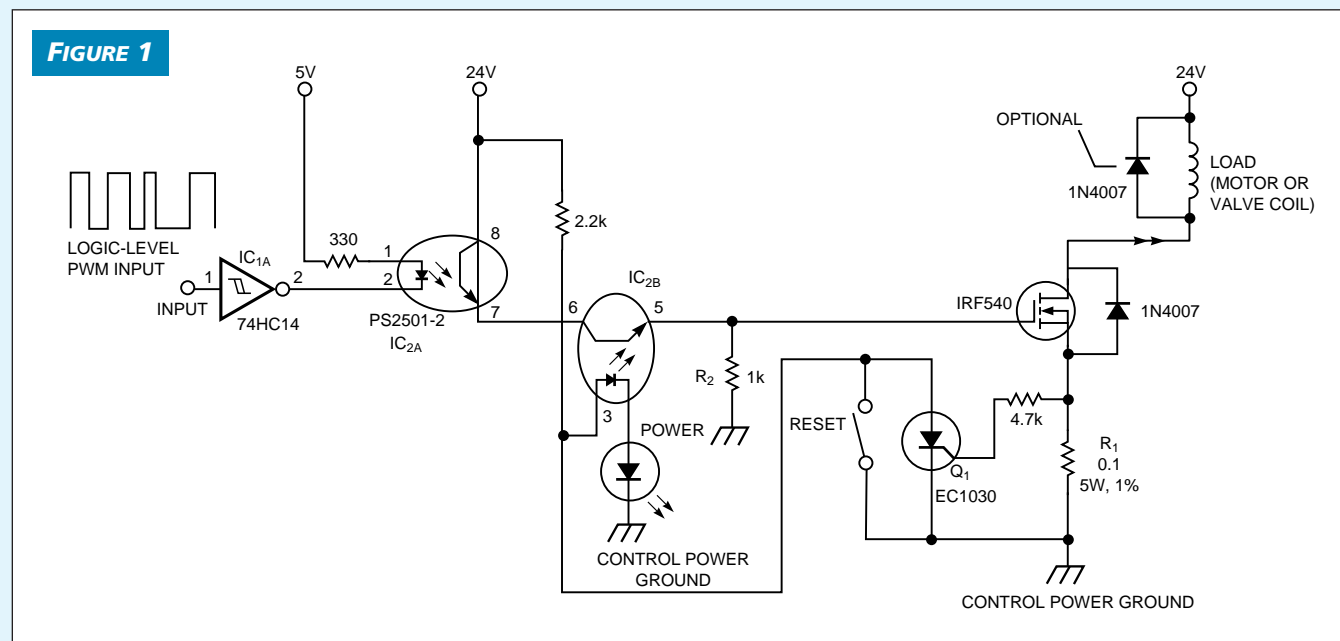
BOB WATSON, CORLEY MANUFACTURING CO, CHATTANOOGA, TN

The circuit in Figure 1 allows standard TTL logic levels to safely drive a high-power dc load. The circuit provides for both signal and ground isolation as well as a solid-state circuit breaker.

The input signal drives IC_{1A}, which in turn provides drive current for optoisolator IC_{2A}. In the absence of an overcurrent condition, IC_{2B} conducts the signal to the gate of the MOSFET.

R₁, to cause a voltage drop of approximately 0.7V, SCR Q₁ latches on. When Q₁ is on, the circuit pulls Pin 3 of IC_{2B} low, which stops the transistor side of IC_{2B} from conducting. R₂ then holds the gate of the MOSFET low, which prevents it from conducting until you reset the SCR. (DI #2163)

To Vote For This Design, Circle No. 346



An overcurrent condition in this isolated PWM driver turns on SCR Q₁, which stops IC_{2B} from conducting.

μ C measures high-frequency signals

STAN D'SOUZA, MICROCHIP TECHNOLOGY INC, CHANDLER, AZ

To measure a high-frequency signal using an 8-bit μ C, the time period of the measured frequency must be relatively close to the internal clock of the μ C. For example, if the internal clock period is 1 μ sec, then the maximum frequency that you can measure is 1 MHz in most μ C applications.

However, you can use the circuit in Figure 1a to measure a frequency much higher than the internal clock frequency of the μ C. This circuit uses an external binary ripple counter operating in asynchronous mode. The circuit gates the incoming frequency to the counter using NAND gates and control lines from the μ C. To enable counting, the μ C sets CNTL₁ and CNTL₂ to 1. Once the measurement time is complete, CNTL₁ resets to 0, which stops further inputs to the counter.

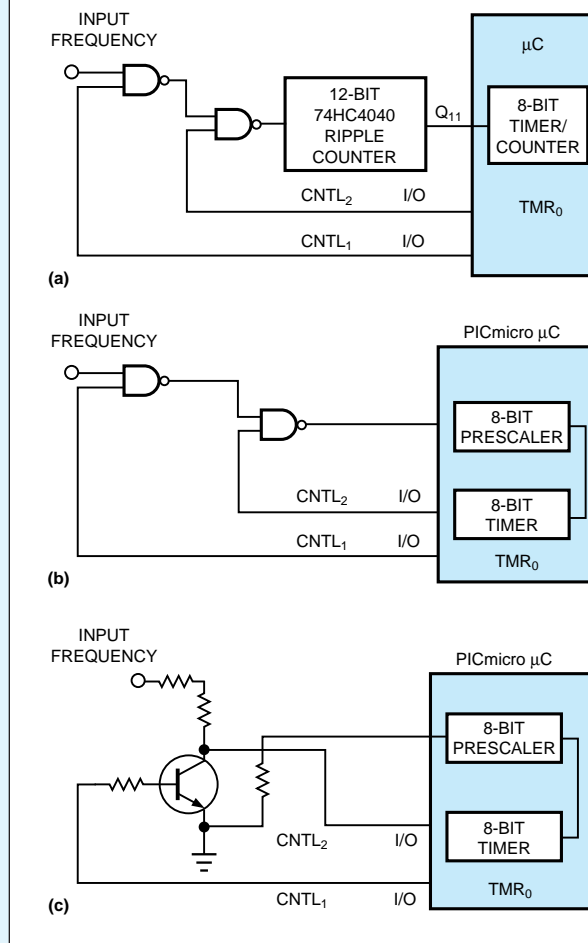
To read the value of the low 12 bits of the frequency, the μ C toggles CNTL₂ N times until the internal 8-bit timer increments by one. The low 12-bit value of the frequency is then equal to 4096 N. Reading the value of the counter in this manner requires only one I/O line as opposed to 12 lines to read the counter's 12 bits. In combination with the internal 8-bit counter in the μ C, a 20-bit frequency measurement is possible. The limiting factor of the measurement is the maximum frequency input to the NAND gates and the ripple counter.

Although the circuit in Figure 1a is straightforward, the circuit does involve the additional cost of a 14-pin NAND gate and a 16-pin counter, which may be undesirable in many applications. In PICmicro 8-bit μ Cs, the internal 8-bit counter/timer, TMR₀, has an associated 8-bit divider, or prescaler. The counter has read/write capability, but you can't read the prescaler value. You can modify Figure 1a's circuit using a PICmicro μ C to implement a 16-bit frequency counter (Figure 1b).

This circuit uses the internal 8-bit prescaler to divide the incoming frequency. The circuit feeds the output of the prescaler to the 8-bit timer, TMR₀, for measurement. As with Figure 1a's circuit, once the gate time is over, CNTL₁ blocks additional clocks from the input signal. Then, CNTL₂ pulses the 8-bit prescaler N times until the 8-bit timer/counter, TMR₀, increments by 1. In this case, the lower 8-bit value of the measured frequency equals 256 N. The μ C then concatenates the value of the counter with the 8-bit timer's value to give a 16-bit value of the measured frequency.

Figure 1c shows a further simplification of the circuit in Figure 1b by replacing the NAND gates with two transistors and four resistors. To start the counter, the μ C configures CNTL₁ and CNTL₂ as inputs or in a high-impedance mode. Thus, the circuit directs the incoming signal to the prescaler and in turn to the 8-bit timer/counter, TMR₀. When the gate time is complete, the μ C makes CNTL₁ an output going low. A low CNTL₁ deactivates the transistor, whose output becomes an open collector. The μ C can now make CNTL₂ an output normally high and going low to pulse the input to the prescaler. The μ C pulses CNTL₂ low N times until the value of TMR₀ increments by 1. The low 8-bit value of the frequency is equal to 256 N. To begin counting again, the μ C reads the value of TMR₀, which clears the prescaler, and again configures CNTL₁ and CNTL₂ as

FIGURE 1



A binary ripple counter and NAND gate team with a μ C (a) to measure input frequencies higher than the μ C clock. Using a PICmicro μ C reduces the number of necessary components (b). A further simplification (c) replaces the NAND gate with one transistor and four resistors.

inputs.

Note that the architecture of the PICmicro μ C allows accurate timekeeping for the gating pulse because the timing of a software loop is predictable and accurate to within one instruction cycle; a PICmicro μ C executes each instruction in one cycle, except for branch instructions, which take two cycles. (DI #2164)

To Vote For This Design, Circle No. 347

Microamps monitor dual-supply batteries

BRUCE ANDERSON, UNIVERSITY OF WISCONSIN—MADISON

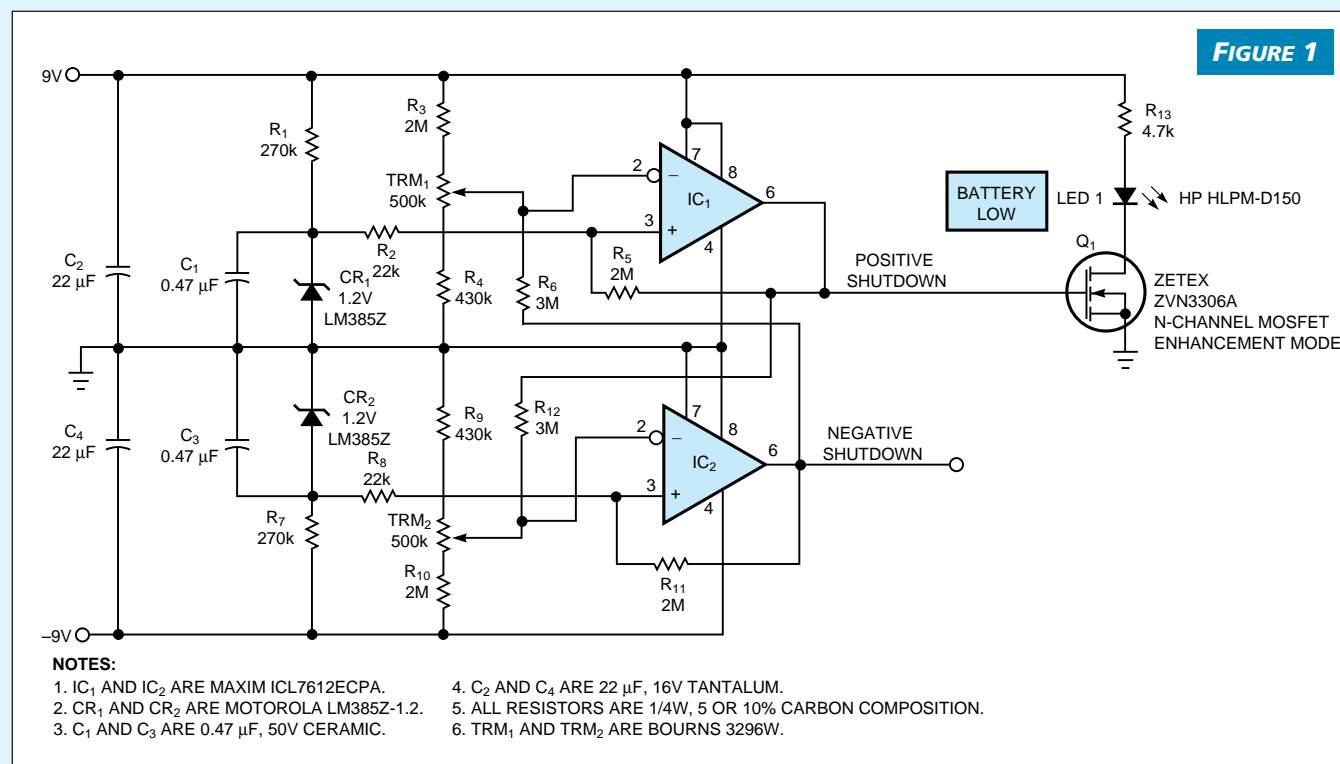
The low-power circuit in Figure 1 monitors two 9V batteries in a dual-supply configuration and turns on the Battery Low LED if either battery voltage drops below its limit. It also provides two shutdown signals you can use to turn off voltage regulators, such as Maxim's MAX663/664 positive and negative regulators. By using low-power voltage references and op amps, the circuit holds the current drain to approximately 45 μ A from each battery, with the positive drain rising to approximately 1 mA when the LED turns on.

Each battery voltage undergoes comparison with a Motorola LM385Z 1.2V reference, using a Maxim 7612 op amp with hysteresis via a 2-M Ω resistor (R_5 and R_{11}). Cross-coupling via the 3-M Ω resistors (R_6 and R_{12}) ensures that if either shutdown signal goes true, both do, and the circuit locks up in the shutdown state with the Battery Low LED illuminated. C_1 and C_3 delay the reference voltages so that when the batteries switch on, the circuit comes up in the proper state. Positive Shutdown is at the positive rail when true. Negative Shutdown is at the negative rail when true. The values shown allow you to adjust the battery-low limits over a range of approximately 3.8 to 8.1V. For our applications with Eveready EN22 alkaline batteries, we typically set the limits at 6.5V. This setting uses a good portion

of the battery life, yet allows some reserve for continued operation after the LED comes on.

The circuit has two convenient features that were unforeseen before testing. One is that when the batteries switch off the LED flashes briefly as the decoupling capacitors discharge. The flashing indicates that the batteries are not so totally dead that they cannot light the LED. The other is that the LM385 has an initial turn-on voltage about 10% higher than the steady-state 1.2V reference. Thus, when you switch the batteries on, they must have a voltage about 10% higher than the steady-state threshold to be considered good. So if the Battery Low LED stays off when the device turns on, the batteries will remain good for a while. Of course, if you are not using the shutdown signals to turn off regulators, you can set the threshold so that your device will continue to operate for a period after the LED comes on. Using the battery-discharge characteristics and your circuit's voltage and current requirements, you can select a threshold that gives appropriate reserve for your application. (DI #2169)

To Vote For This Design, Circle No. 348



If either battery voltage in a two-battery supply drops below a preset limit, a Battery Low LED turns on.

Program provides integer-to-binary conversion

BERT ERICKSON, FAYETTEVILLE, NY

Binary numbers rarely appear in applications of C or C++ programs, so any reference to converting from an integer to a binary number is usually relegated to a few simple examples in the appendix. However, when you're working with codes for communication systems, terms such as parity, checksum, distance, weight, and block codes are much easier to verify with a check solution when they are in binary form. C and C++ statements do use integers for manipulations that have binary implications. However, when the analysis gets down to the binary-number level, the conversion from integers is hard to find in the libraries supplied with the compiler. The `cintbin` and `classicC` functions in Listings 1 and 2 convert an integer in the main function to a binary number that remains available in the main function.

The ones and zeros in the elements of the array correspond to the location of bits in the customary binary number. You can compile the C++ `cintbin` version as listed. Readers who have an ANSI C compiler can use the program preceded by `//` in Listing 2. For long integers, refer to the revised edition of *Microsoft C Programming for the PC* by Robert Lafore. The first part of the listing is only a driver that has a call to the function and a printout for the binary number. You can use the bits in the binary number in any additional statements.

The first argument in the call should be 31 or less to provide some leading zeros but large enough to make sure the most significant bit is included. The temporary variable `z` and the return value provide some assurance that the result is valid. The statements in both functions are self-explanatory, so the only thing left to do is to compile one of the programs and enter 31 4,294,967,295 with a space after 31 to verify the 32-bit binary number 1111 1111 1111 1111 1111 1111 1111 1111. You can download the listings and the executable `cintbin` file from EDN's Web site, www.ednmag.com. At the registered-user area, go into the Software Center to download the files from DI-SIG, #2156. (DI #2156)

To Vote For This Design, Circle No. 349

LISTING 1—C++ INTEGER-TO-BINARY CONVERSION ROUTINE

```
// cintbin.cpp      a C++ function
// Converts an integer to a binary number
#include <iostream.h>
#include <math.h>
int c[32];
main() {
int n; unsigned long int x;
unsigned long int cintbin(int, unsigned long int); //Declare

cout << "\n\tEnter max power of 2 desired (31 or less) and "
<< "the integer number\n" << "\t(4,294,967,295 or less) "
<< "with a space between them ";
cin >> n >> x;
cout << "\tReturn x = " << cintbin(n, x)
<< " should equal the input value shown above\n Bin # = ";
for (int k = n; k >= 0; k--) cout << ' ' << c[k];
return 0;
}

unsigned long int cintbin(int n, unsigned long int x) //Define
{
unsigned long int y, z; z = x;
for (int i = 0; i < n+1; i++) c[i] = 0;

for (int j = n; j >= 0; j--)
{ y = int(pow(2,j)); if (x >= y)
{ c[j] = 1; x = x - y; }
}
return z;
}
```

LISTING 2—CLASSIC C INTEGER-TO-BINARY-CONVERSION ROUTINE

```
//// classicC.cpp a C function
//// Converts an integer to a binary number
#include <stdio.h>
#include <math.h>
int c[32];
//main() {
// int n; unsigned long int x;
// unsigned long int classicC(int, unsigned long int); //Declare
//
// printf("\n\n\tEnter max power of 2 desired (31 or less) and ");
// printf("\tthe integer number\n");
// printf("\t(4,294,967,295 or less) with a space between them ");
// scanf("%d %lu",&n,&x);
// printf("\tReturn x = %lu",classicC(n,x));
// printf(" should equal the input value shown above\n Bin # = ");
// for (int k = n; k >= 0; k--)
// printf(" %d",c[k]);
// return 0;
// }
//
// unsigned long classicC(int n, unsigned long x) //Define classicC
// {
// unsigned long y, z;
// int i, j;
// z = x;
// for (i = 0; i < n+1; i++)
// c[i] = 0;
// for (j = n; j >= 0; j--)
// {
// y = pow(2,j);
// if (x >= y)
// {
// c[j] = 1;
// x = x - y;
// }
// }
// return z;
// }
```