Most logic devices have a 5V supply rail. However, you often need to know the upper and lower switching thresholds for a device operating at supply levels other than 5V. In some cases, it is also important to know the variation of these thresholds with supply voltage. A simple test circuit allows you to make these measurements (Figure 1). The basic idea is to create an oscillator by feeding an output signal back to the control input and then monitor the control input with an oscilloscope. Then, you can easily monitor the upper and lower transition points on the scope. To slow the frequency of oscillation and reduce the effects of propagation delay, it’s sometimes desirable to add an RC network.

The circuit examines the threshold voltages at the control input of an analog switch, but the scheme applies to any simple logic device. Switching thresholds for the IC depend on the logic-supply voltage you apply to $V_L$. (The analog-switch channel COM-NO handles voltages of $\pm15V$.) The data sheet specifies IN thresholds for the most common usage ($V_L = 5V$), but the chip accommodates $V_L$ from 4.5 to the positive rail (15V in this case). The circuit characterizes threshold voltages over the full operating range of $V_L$ (Figure 2). At power-up, the switch is open. $V+$ charges $C_1$ through $R_1$ and $R_2$ until $IN$ reaches its upper threshold, closing the switch. $C_1$ then discharges to ground (through $R_2$ and the switch) until $IN$ reaches its lower threshold, thereby opening the switch and repeating the cycle.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/infoaccess.asp, enter No. 321 in the “Circle Number” field, and hit the Search bar. Put a tick in the “Select” box and hit “Submit.”

---

**Figure 1**

![Circuit Diagram](image1)

**Figure 2**

![Threshold Voltage Graph](image2)

---

For the circuit in Figure 1, the thresholds vary with supply voltage as shown.
CRC (cyclic-redundancy checking) allows you to verify the integrity of transmitted data frames. It finds use in many transmission protocols, both bit-oriented (High Level Data Link Control, Advanced Data Communications Control Procedure, CRC-CCITT, for example) and frame-oriented (asynchronous transfer mode, Ethernet, Fiber Distributed Data Interface, for example). CRC protection constitutes adding redundancy bits (n−k bits) to a frame (k bits) of data to transmit a coded frame of n bits. Express the data as a polynomial data frame of k bits:

$$X_k = \{x_{k-1}, x_{k-2}, \ldots, x_0\}, \quad (1)$$

which is written as:

$$X_k(D) = x_{k-1}D^{k-1} + x_{k-2}D^{k-2} + \ldots + x_1D + x_0, \quad (2)$$

where D is the variable and bits $x_i$ (1,0) are the coefficients of the polynomial. Starting from $X_k(D)$ to generate a code word of n bits (n=k), you use a so-called generator polynomial:

$$G_{n-k}(D) = g_{n-k-1}D^{n-k-1} + g_{n-k-2}D^{n-k-2} + \ldots + g_1D + g_0, \quad (3)$$

Multiplying $X_k(D)$ by $D^{n-k}$, you obtain a polynomial of grade n. This operation shifts the data $X_k(D)$ to the left by n−k bits, attaching n−k zeros to the least significant bits. Now, consider:

$$D^{n-k}X_k(D) = Q_k(D) \ast G_{n-k}(D) + R_{n-k-1}(D), \quad (4)$$

where the original data multiplied by $D^{n-k}$ is divided by the generator polynomial, yielding the quotient $Q_k(D)$ and the remainder $R_{n-k-1}(D)$, which always has the grade n−k−1 (n−k bits). The transmitted frame consists of the original data followed by n−k bits of remainder. The polynomial arithmetic, in this case, is cast in a binary algebraic field. Addition and subtraction are identical and equivalent to an XOR operation without the need of a carry. Thus, it is possible to rewrite Equation 4 as follows:

$$D^{n-k}X_k(D) + R_{n-k-1}(D) = Q_k(D) \ast G_{n-k}(D), \quad (5)$$

Equation 5 shows that the transmitted frame is divisible by the generator polynomial. At the receiving side it is divided by the same generator polynomial, and the quotient,
Qk(D) is discarded. If no errors exist in the transmitted data, the remainder should always be zero. Both the transmitter and the receiver side must perform the same division by Gk\(n-k\)(D). The linear-feedback shift registers in Figure 1 implement the division. When the kth bit loads in, the shift-register value is the remainder. For more information about how the CRC scheme works, see Reference 1. An AND function performs multiplication by one or zero, and an XOR function performs the sum (subtraction) operation. Listing 1 shows simple synthesizable VHDL code that generates the divider schematic.

The code includes constants that allow division by the generator polynomial. Note that Listing 1 includes common polynomials from Listing 2, used as examples (see commented Constants) and also includes the relative initial value of the CRC shift register. Some standards dictate reversing the incoming bit stream (LSB first), the calculated CRC, or both. The signal load_frame, active high, defines the dimension of the serially loaded data frame. For example, the simulation in Figure 2 has a data-frame size of 20 bits. At the 20th bit, load_frame goes low, reloading the initial value for the CRC shift registers in readiness for the next frame. The calculated CRC value is CC16. You can download listings 1 and 2 from EDN's Web site, www.ednmag.com. Click on “Search Databases” and then enter the Software Center to download the file for Design Idea #2553.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/infoaccess.asp, enter No. 322 in the “Circle Number” field, and hit the Search bar. Put a tick in the “Select” box and hit “Submit.”

Reference


---

LISTING 1—SERIAL CRC CHECKER/GENERATOR

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE work.crc_pkg.ALL;

ENTITY crc IS
  PORT (  
    mr  : IN std_logic;
    clk : IN std_logic;
    load_frame  : IN std_logic;
    data_in  : IN std_logic;
    crc_out  : OUT std_logic_vector((width_poly-1) DOWNTO 0)
  );
END crc;

ARCHITECTURE rtl OF crc IS
  SIGNAL crc_shift  : std_logic_vector((width_poly-1) DOWNTO 0);
  SIGNAL mr_and_load_frame : std_logic;
BEGIN
  mr_and_load_frame <= mr AND load_frame;

  shift_register : PROCESS(mr_and_load_frame,clk)
  VARIABLE xor0 : std_logic;
  BEGIN
    IF (mr_and_load_frame = '0') THEN
      xor0 := '0';
      crc_shift <= crc_init;
    ELSIF ((clk = '1') AND (clk'EVENT)) THEN
      xor0 := (data_in XOR crc_shift(width_poly-1));
      IF (load_frame = '1') THEN
        IF (xor0 = '0') THEN
          crc_shift <= crc_shift(width_poly-2 DOWNTO 0) & xor0;
        ELSE
          crc_shift <= crc_shift(width_poly-2 DOWNTO 0) XOR poly_gen(width_poly-1 DOWNTO 1) & xor0;
        END IF;
      END IF;
    END IF;
  END PROCESS shift_register;

  crc_out  <= crc_shift;
END rtl;
```

LISTING 2—POLYNOMIALS AND RELATIVE INITIAL VALUE OF CRC SHIFT REGISTER

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

PACKAGE crc_pkg IS
  CONSTANT width_poly : integer := 16;
  -- (comment) CRC-16 CITI X25 standard : G(D) = D'16 + D'12 + D'5 + D'0 = (1-0001-0000-0010-0001)
  CONSTANT poly_gen  : std_logic_vector(width_poly-1) DOWNTO 0) := X'1021' ;
  CONSTANT crc_init  : std_logic_vector(width_poly-1) DOWNTO 0) := X'8005' ;
  -- (comment) CRC-32 : G(D) = 1-0000-0100-1100-0001-1001-0111-0111
  CONSTANT poly_gen  : std_logic_vector(width_poly-1) DOWNTO 0) := X'80050000' ;
  CONSTANT crc_init  : std_logic_vector(width_poly-1) DOWNTO 0) := X'00000000' ;
  CONSTANT width_poly : integer := 32;
  -- (comment) CRC-32 : G(D) = 1-0000-0100-1100-0001-1001-0111-0111
  CONSTANT poly_gen  : std_logic_vector(width_poly-1) DOWNTO 0) := X'80050000' ;
  CONSTANT crc_init  : std_logic_vector(width_poly-1) DOWNTO 0) := X'00000000' ;
END;
```
Using a product originally developed for PC-motherboard environmental monitoring, you can configure a low-cost, general-purpose DAS (data-acquisition system). The DAS in Figure 1 can directly monitor multiple voltage channels, as well as temperature and frequency. It can also monitor digital sensors. Using only a few additional components, the system can accommodate other sensor and transducer elements. Figure 1 shows one possible configuration. You can expand the scheme to cover additional input channels if necessary. For voltage sensing, the ADM9240 contains a multichannel ADC that can directly monitor as many as six input channels. The original intent of the IC was to monitor power supplies on PC motherboards, but it is flexible enough for general-purpose use. The maximum input-voltage ranges on the channels are 3.3, 3.6, 4.4, 6.64, and 16V. Figure 1 shows the DAS monitoring two power supplies: PS1 and PS2. You can measure voltages greater than the channel range by using a simple voltage divider, as illustrated with PS3.

You can use the onboard DAC in the DAS, originally intended as a fan-speed controller, as a programmable, precision voltage reference. This configuration, for example, would facilitate measuring resistance-type sensors with the voltage-sensing channels. You could also use it as a bridge-excitation voltage source for accurate bridge-sensing applications. You can determine an unknown resistance value by setting the DAC’s output voltage to a known level and using a known fixed resistance (Figure 1). You can implement current sensing by placing an accurate series resistor, \( R_{\text{SENSE}} \), in the ground line and then monitoring the voltage drop across the resistor. An on-chip bandgap silicon sensor in the DAS provides temperature monitoring over the IC’s 24 to 85°C operating range. You can use two frequency-monitoring channels in the DAS to monitor the pulsed digital output from a tachometer. The channels can also serve as general-purpose frequency counters.

The original intent of the five digital-input lines in the DAS was to monitor digital voltage-identification lines. You can use these lines as general-purpose input lines. They can sense high- or low-level status signals from digital sensors or from alarm channels. Figure 1 shows a thermostatic sensor. You control and read the DAS using a simple two-wire SMBus or FC interface to a µC or µP. If a dedicated FC controller is unavailable, you can use a port “bit-banging” technique. Easy expansion is also possible by selecting a different device address. Using a different device-address bus needs no additional communication lines, because multiple devices may reside on the same bus.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/infoaccess.asp, enter No. 323 in the “Circle Number” field, and hit the Search bar. Put a tick in the “Select” box and “Submit.”

**Figure 1**

You can configure a DAS to simultaneously measure multichannel voltages, temperature, resistance, current, and frequency.
Algorithm tests for point location

Lawrence Arendt, Manitoba HVDC Research Centre, Winnipeg, Canada

A recent software project approximated the phase-space trajectory (also known as a strange attractor) of a certain dynamic system by using several nonoverlapping triangles. It became necessary to determine whether particular operating points were on or off that attractor. Determining whether a circle or a rectangle contains a point is trivial. A circle centered at point C and having a radius \( R \) contains a point \( P(x,y) \) only if \( ||CP|| < R \). A rectangle \( ABCD \) with opposite vertices \( B(x_1,y_1) \) and \( D(x_2,y_2) \) contains a point \( P(x,y) \) only if \( x_1 < x < x_2 \) and \( y_1 < y < y_2 \). However, it is not such a trivial task to determine whether triangle \( ABC \), which has vertices \( A(x_1,y_1), B(x_2,y_2), \) and \( C(x_3,y_3) \), contains point \( P(x,y) \) (Figure 1). You can simplify the task by calculating the triangle’s area using the formula:

\[
\text{Area}(P_1P_2P_3) = \frac{1}{2} | x_1y_2 - x_2y_1 + x_2y_3 - x_3y_2 + x_3y_1 - x_1y_3 |
\]

where \( P_1, P_2, \) and \( P_3 \) are the vertices. If you use the absolute value of the determinant, you need not be concerned whether \( P_1, P_2, P_3 \) are labeled in a counterclockwise fashion. To determine whether \( ABC \) contains \( P \), calculate the area of \( ABC \) using Equation 1 and then define three new triangles, each having as its vertices point \( P \) and two vertices of \( ABC \). This operation results in three unique triangles: \( ABP, BCP, \) and \( CAP \). You can then calculate the areas of triangles \( ABP, BCP, \) and \( CAP \). If \( ABC \) contains \( P \) (Figure 2), then Area (\( ABP \)) + Area (\( BCP \)) + Area (\( CAP \)) = Area (\( ABC \)). If \( ABC \) does not contain \( P \) (Figure 3), then Area (\( ABP \)) + Area (\( BCP \)) + Area (\( CAP \)) > Area (\( ABC \)).

The beauty of the algorithm in Listing 1 is that you need no squares, square roots, or trigonometric functions. You can extend the methodology to triangles in 3-D space. You can also adapt it for finding the area of irregular-shaped polygons. The C++ program in Listing 1 is for a \( \text{TTriangle} \) class having a \( \text{Contains(...)} \) function similar to that of the Windows \( \text{TRect} \) class. \( \text{TPoint} \) is the floating-point equivalent of the integer \( \text{TPoint} \) class. You can download Listing 1 from \( \text{EDN} \)'s Web site, www.ednmag.com. Click on "Search Databases" and enter the Software Center to download the file for Design Idea #2566.

Is this the best Idea in this issue? Vote at www.ednmag.com/ednmag/infoaccess.asp, enter No. 324 in the "Circle Number" field, and hit the Search bar. Put a tick in the "Select" box and hit "Submit."

**Figure 1** It is not a trivial task to determine whether a point \( P \) is within a triangle \( ABC \).

**Figure 2** If \( P \) is within triangle \( ABC \), the three new triangles will be within \( ABC \), too.

**Figure 3** If \( P \) is outside triangle \( ABC \), the three new triangles will extend beyond \( ABC \)'s borders.
A classic AGC circuit keeps the output-energy level constant.

When the energy of \( y(t) \) is lower than \( E_{\text{MIN}} \), if the output signal's energy is between \( E_{\text{MAX}} \) and \( E_{\text{MIN}} \), the control circuit ceases to alter the gain of the PGA, and the circuit operates in open-loop fashion. For the programming pulses (INC), use a fixed-frequency square-wave signal, which defines the adaptation rate of the PGA.

Figure 3 shows the experimental results. The input signal (Figure 3a) is a sinusoid with a frequency of 3840 Hz, amplitude-modulated by a 50-Hz square-wave signal. The maximum and minimum levels of the input signal are 1V and 200 mV, respectively. The cutoff frequency of the filter is 589 Hz. The programming pulses consist of a 7.68-kHz square-wave signal (readily available in the application at hand).

To keep the output signal's amplitude in the interval 0.9 to 1V, adjust the \( E_{\text{MAX}} \) and \( E_{\text{MIN}} \). If the output signal's energy is between \( E_{\text{MAX}} \) and \( E_{\text{MIN}} \), the control circuit ceases to alter the gain of the PGA, and the circuit operates in open-loop fashion. For the programming pulses (INC), use a fixed-frequency square-wave signal, which defines the adaptation rate of the PGA.

Figure 3 shows the experimental results. The input signal (Figure 3a) is a sinusoid with a frequency of 3840 Hz, amplitude-modulated by a 50-Hz square-wave signal. The maximum and minimum levels of the input signal are 1V and 200 mV, respectively. The cutoff frequency of the filter is 589 Hz. The programming pulses consist of a 7.68-kHz square-wave signal (readily available in the application at hand).

To keep the output signal's amplitude in the interval 0.9 to 1V, adjust the \( E_{\text{MAX}} \) and \( E_{\text{MIN}} \). If the output signal's energy is between \( E_{\text{MAX}} \) and \( E_{\text{MIN}} \), the control circuit ceases to alter the gain of the PGA, and the circuit operates in open-loop fashion. For the programming pulses (INC), use a fixed-frequency square-wave signal, which defines the adaptation rate of the PGA.

Figure 3 shows the experimental results. The input signal (Figure 3a) is a sinusoid with a frequency of 3840 Hz, amplitude-modulated by a 50-Hz square-wave signal. The maximum and minimum levels of the input signal are 1V and 200 mV, respectively. The cutoff frequency of the filter is 589 Hz. The programming pulses consist of a 7.68-kHz square-wave signal (readily available in the application at hand).

To keep the output signal's amplitude in the interval 0.9 to 1V, adjust the \( E_{\text{MAX}} \) and \( E_{\text{MIN}} \). If the output signal's energy is between \( E_{\text{MAX}} \) and \( E_{\text{MIN}} \), the control circuit ceases to alter the gain of the PGA, and the circuit operates in open-loop fashion. For the programming pulses (INC), use a fixed-frequency square-wave signal, which defines the adaptation rate of the PGA.

Figure 3 shows the experimental results. The input signal (Figure 3a) is a sinusoid with a frequency of 3840 Hz, amplitude-modulated by a 50-Hz square-wave signal. The maximum and minimum levels of the input signal are 1V and 200 mV, respectively. The cutoff frequency of the filter is 589 Hz. The programming pulses consist of a 7.68-kHz square-wave signal (readily available in the application at hand).

To keep the output signal's amplitude in the interval 0.9 to 1V, adjust the \( E_{\text{MAX}} \) and \( E_{\text{MIN}} \). If the output signal's energy is between \( E_{\text{MAX}} \) and \( E_{\text{MIN}} \), the control circuit ceases to alter the gain of the PGA, and the circuit operates in open-loop fashion. For the programming pulses (INC), use a fixed-frequency square-wave signal, which defines the adaptation rate of the PGA.

Figure 3 shows the experimental results. The input signal (Figure 3a) is a sinusoid with a frequency of 3840 Hz, amplitude-modulated by a 50-Hz square-wave signal. The maximum and minimum levels of the input signal are 1V and 200 mV, respectively. The cutoff frequency of the filter is 589 Hz. The programming pulses consist of a 7.68-kHz square-wave signal (readily available in the application at hand).

To keep the output signal's amplitude in the interval 0.9 to 1V, adjust the \( E_{\text{MAX}} \) and \( E_{\text{MIN}} \). If the output signal's energy is between \( E_{\text{MAX}} \) and \( E_{\text{MIN}} \), the control circuit ceases to alter the gain of the PGA, and the circuit operates in open-loop fashion. For the programming pulses (INC), use a fixed-frequency square-wave signal, which defines the adaptation rate of the PGA.

Figure 3 shows the experimental results. The input signal (Figure 3a) is a sinusoid with a frequency of 3840 Hz, amplitude-modulated by a 50-Hz square-wave signal. The maximum and minimum levels of the input signal are 1V and 200 mV, respectively. The cutoff frequency of the filter is 589 Hz. The programming pulses consist of a 7.68-kHz square-wave signal (readily available in the application at hand).

To keep the output signal's amplitude in the interval 0.9 to 1V, adjust the \( E_{\text{MAX}} \) and \( E_{\text{MIN}} \). If the output signal's energy is between \( E_{\text{MAX}} \) and \( E_{\text{MIN}} \), the control circuit ceases to alter the gain of the PGA, and the circuit operates in open-loop fashion. For the programming pulses (INC), use a fixed-frequency square-wave signal, which defines the adaptation rate of the PGA.

Figure 3 shows the experimental results. The input signal (Figure 3a) is a sinusoid with a frequency of 3840 Hz, amplitude-modulated by a 50-Hz square-wave signal. The maximum and minimum levels of the input signal are 1V and 200 mV, respectively. The cutoff frequency of the filter is 589 Hz. The programming pulses consist of a 7.68-kHz square-wave signal (readily available in the application at hand).

To keep the output signal's amplitude in the interval 0.9 to 1V, adjust the \( E_{\text{MAX}} \) and \( E_{\text{MIN}} \). If the output signal's energy is between \( E_{\text{MAX}} \) and \( E_{\text{MIN}} \), the control circuit ceases to alter the gain of the PGA, and the circuit operates in open-loop fashion. For the programming pulses (INC), use a fixed-frequency square-wave signal, which defines the adaptation rate of the PGA.

Figure 3 shows the experimental results. The input signal (Figure 3a) is a sinusoid with a frequency of 3840 Hz, amplitude-modulated by a 50-Hz square-wave signal. The maximum and minimum levels of the input signal are 1V and 200 mV, respectively. The cutoff frequency of the filter is 589 Hz. The programming pulses consist of a 7.68-kHz square-wave signal (readily available in the application at hand).

To keep the output signal's amplitude in the interval 0.9 to 1V, adjust the \( E_{\text{MAX}} \) and \( E_{\text{MIN}} \). If the output signal's energy is between \( E_{\text{MAX}} \) and \( E_{\text{MIN}} \), the control circuit ceases to alter the gain of the PGA, and the circuit operates in open-loop fashion. For the programming pulses (INC), use a fixed-frequency square-wave signal, which defines the adaptation rate of the PGA.
and $E_{\text{MIN}}$ levels to 0.64 and 0.57V, respectively. In a full-wave rectifier, with a sinusoidal input signal, the energy relates to the amplitude by $E = 2A/\pi$. Figure 3b, the output of the AGC circuit, shows that the circuit behaves as expected. Note that the settling time of the gain changes depends not only on the frequency of the adaptation signal (INC) but also on the energy level of the input signal. The circuit obtained good performance for an input-signal amplitude in the interval 0.05 to 2.2V. This interval represents approximately 33 dB of dynamic range. The circuit is well-suited for implementation in systems using a DSP. You insert an ADC after the PGA, and the DSP performs all computations on the right side of the dotted line in Figure 2—full-wave rectifier, lowpass filter, and control. The DSP provides the digital control signals (INC and U/D) to the PGA.

**Figure 3**

The input signal (a) to Figure 2’s circuit is a 3840-Hz sine wave modulated by a 50-Hz square wave. You can see the settling characteristics in (b).

---

**Speed C functions for C16x µCs**

**Hans-Herbert Kirste, WAGO, Minden, Germany**

The Infineon C16x Series of µCs use an internal 16-bit-register architecture. The architecture can also process byte-wide functions. The µC uses byte moves to implement library functions such as memset() or memcpy(). This process makes the library code independent of count values and address values; both may be odd or even. The disadvantage is the loss of speed because of the need to process loop counters for every byte. Memory access also cuts the speed, because two cycles are necessary to write a word in 2 bytes. The code in Listing 1 implements a fast_memset() function that takes care of the byte count as well as the start address. Both may be odd or even. The function uses as many word moves as possible to preset the memory area. You can easily change the routine to process the copy function, memcpy(), with the same advantages. You can download Listing 1 from EDN's Web site, www.ednmag.com. Click on “Search Databases” and then enter the Software Center to download the file for Design Idea #2569. **Is this the best Idea in this issue?** Vote at www.ednmag.com/ednmag/infoaccess.asp, enter No. 325 in the “Circle Number” field, and hit the Search bar. Put a tick in the “Select” box and hit “Submit.”

**LISTING 1—FUNCTION-ACCELERATION CODE FOR C16X µCS**

```c
void * fast_memset ( void * pvDest, UCHAR ucVal, UINT uiSize ) {
    UCHAR ucVal = ( UCHAR ) ucVal;  /* create a word to fill the memory */
    if ( _-zero ( p ) ) /* if the start address is odd */
        *(UCCHAR far *)p = ucVal;  /* write a byte to the start address */
    p = ( UCHAR far *) ( (UCCHAR far *) p + 1 );  /* move pointer to next address */
    --uiSize;    /* count one byte written*/
    while ( uiSize ) { /* count remaining byte count to word count */
        if ( _zero ( w ) ) /* if one byte is left*/
            *(UCCHAR far *)p = ucVal;  /* write to the last byte */
        p = ( UCHAR far *) ( ( UCHAR far *) p + 1 );  /* point to the next word */
        --uiSize;    /* decrement the word counter */
    }
    return ( pvDest );  /* return the pointer to start of memory area */
}
```

You can see the settling characteristics in (b).
Intelligent sensors and signal conditioners for plant-automation systems must produce outputs compatible with standard data-acquisition systems, such as PCs, programmable logic controllers, and remote terminals for supervisory and control systems. Standard analog inputs are 0 to 1V, ±10V, 0 to 1 mA, and 4 to 20 mA. The circuit in Figure 1 makes all these signals available, using only three packages in addition to the DAC. The voltage-output section requires a thin-film resistor package, Beckman Series 668/698 (which tracks with temperature), and two FET-input op amps. The current-output section requires the addition of a four-resistor package, Beckman Series 664/694, and two more op amps. A reference voltage of −1.000V yields voltage outputs of 0 to 1V and ±1V and either 0 to 1 mA with $R_x = 1 \, k\Omega$ and JP2 and JP3 connected or 4 to 20 mA with $R_x = 83.33 \, k\Omega$ and JP1 and JP4 connected.

A reference of −10.00V and appropriate values of $R_x$ yield corresponding voltage and current outputs. All $R_1$ and $R_2$ values in Figure 1 are 100 kΩ ± 0.1%. You should select $Q_1$ and possibly use a heat sink, taking into consideration the supply voltage and output-current range.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/infoaccess.asp, enter No. 327 in the “Circle Number” field, and hit the Search bar. Put a tick in the “Select” box and hit “Submit.”

**Figure 1**

This circuit simultaneously provides unipolar and bipolar voltage outputs and two output-current ranges.