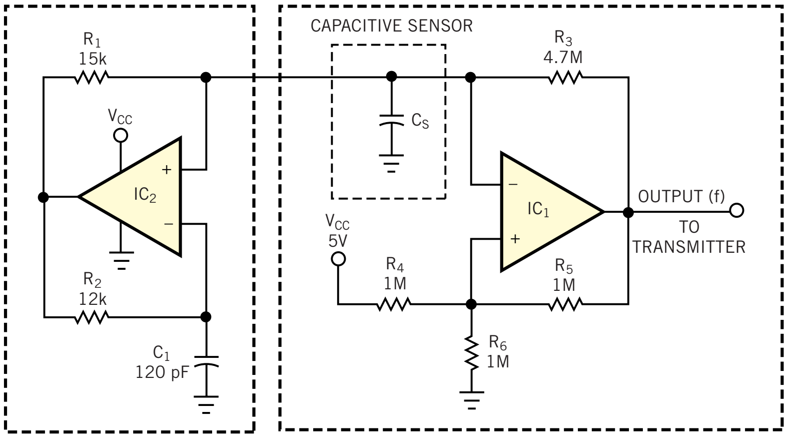
Edited by Bill Travis

Improved frequency modulator uses "negatron"

Alexander Bell, Infosoft International, Rego Park, NY

YOU CAN IMPROVE a capacitive-sensor circuit with a modulator and an RF transmitter (**Reference 1**) by modifying the modulator portion to obtain better accuracy. More improvements result from adding a "negatron" circuit, a configuration that uses equivalent negative capacitance. **Reference 2** gives some insight into the uses of negative impedance. The circuit in **Figure 1** works with the RF transmitter of **Reference 1** (**Figure 2**) as follows: The modulator uses op amp IC_1 in a standard flip-flop configuration (with the addition of the bias resistor R_4, because of single-supply operation). The negatron portion of the circuit uses op amp IC_2. The output frequency, f, of the modulator without the negatron is a function of the time constant $\tau_M=R_3C_S$. Thus, the frequency is a function of the physical value of the input (for example, pressure or humidity). The potential accuracy and stability of the modulator in **Figure 1** are greater than those of the modulator in **Reference 1** because of the low input capacitance and temperature coefficient of the LTC1124 op amp and the high stability of the R_4-R_6 voltage threshold.

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

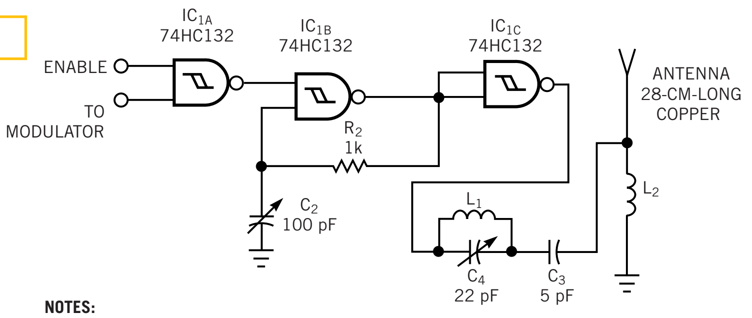


NOTE: OP AMPS=LTC1124s.

Figure 1

The modulator portion provides improved performance over the circuit in Reference 1; the negatron increases sensitivity and resolution.

Figure 2



NOTES:
L_1 IS SIX TURNS OF 22-GAUGE WIRE WITH 5-MM DIAMETER.
L_2 IS 18 TURNS OF 22-GAUGE WIRE WITH 5-MM INNER DIAMETER.

This simple transmitter from Reference 1 allows you to remotely measure physical phenomena.

Adding the negatron portion in **Figure 1** increases the relative sensitivity of the frequency modulator. The equivalent capacitance of the negatron is $C_N=-C_1(R_2/R_1)$. Assuming the physical value of the input causes the change ΔC (compared with the sensor's initial capacitance, C_0), the relative output-frequency change without the negatron (for small ΔC) is $\Delta f/f_0=\Delta C/C_0$. With the negatron added, the equation is: $\Delta f/f_0=\Delta C/(C_0-C_1(R_2/R_1))$. The result is higher relative sensitivity because of the reduction in the denominator value. The value of C_N is approximately −100 pF for the circuit in **Figure 1**. In other words, a given change in the input value causes a greater relative-frequency deviation. Note that adding the negatron changes the equivalent time constant: $\tau=R_3(C_S-C_1(R_2/R_1))$. You can make value adjustments in R_3 to obtain the desired initial frequency, f_0. Also, note that the measured values of f_0 and Δf may differ from the calculated ones because of parasitics and the input capacitance of the op amp.

References

1. Tiwari, Shyam, "Low-cost relative-humidity transmitter uses single logic IC," *EDN*, Nov 8, 2001, pg 116.
2. Belousov, Alexander, "Negative impedance improves capacitive sensors," *EDN*, March 30, 1995, pg 82.

Is this the best Design Idea in this issue? Select at www.ednmag.com.

Get buck-boost performance from a boost regulator

Tom Gross, Linear Technology Corp, Milpitas, CA

The SEPIC (single-ended, primary-inductance-converter) topology is generally a good choice for voltage regulators that must produce an on output voltage that falls in the middle of the input-voltage range, such as a 5V output from a 2.7 to 6V input, The topology has some disadvantages, however. The efficiency of a SEPIC circuit fares worse than that of buck and boost regulators, and SEPIC designs often involve the use of large, complex magnetic components, thereby complicating the design task. **Figure 1** shows a simple and efficient alternative topology. When the input voltage is lower than the output voltage, the circuit operates as a normal boost regulator. The inductor, L_1, stores energy when switch Q_1 is on and the boost diode, D_1, is reverse-biased. While D_1 is off, the output capacitor, C_1, delivers the load current. When Q_1 turns off, L_1 reverses its polarity, thereby forward-biasing D_1. L_1 then charges C_1 and delivers current to the load. The inductor voltage adds to the input voltage to generate the output voltage.

The low-battery comparator in IC_1, which usually checks battery levels, monitors the output voltage through its LBI pin. IC_1's internal comparator output switches low (sinking current). This action turns the p-channel Q_1 fully on, cre-

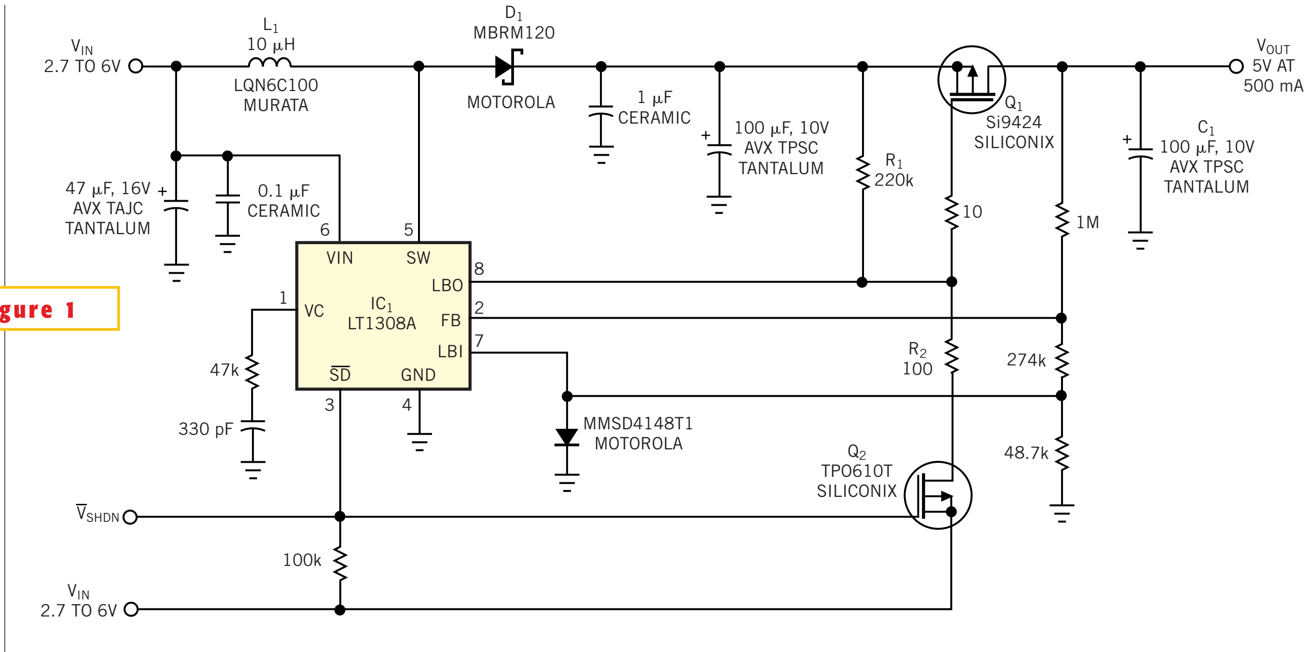


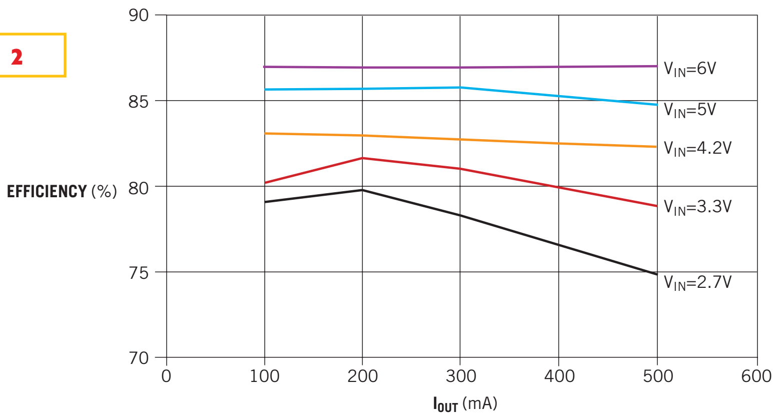
Figure 1

This circuit can both boost and step down the output voltage, depending on whether the input voltage is lower or higher than the output voltage.

ating a low-impedance path to the output. When the input voltage is the same value as or greater than the output voltage, the circuit functions like a linear regulator. In this case, the internal comparator's output assumes a high-impedance state. The voltage at the gate of Q_1 begins to pull up through R_1, a 220-kΩ resistor, so Q_1 begins to turn off. This action forces the output voltage to decrease, and the comparator eventually again switches states from high to low (sinking current). The comparator's low state causes the output voltage to rise again, and the cycle repeats. Thus, the circuit begins to operate as a linear regulator, with Q_1 acting as the pass transistor.

The circuit can also disconnect the input-to-output current path, unlike a conventional boost regulator. The shutdown signal connects to the gate of Q_2, a logic-level, p-channel MOSFET, as well as to IC_1's shutdown ($\overline{SD}$) pin. When the shutdown signal goes low, it turns off IC_1 and turns on Q_2. This action delivers V_{IN} (via R_2, a 100Ω resistor) to the gate of Q_1, thereby turning off the transistor. Hence, the shutdown operation disconnects the input-to-output current path. **Figure 2** shows how the efficiency of this linear-boost regulator depends on its mode of operation. When the input voltage is lower than the output voltage, the efficiency of the regulator is that of a boost regulator. When the input voltage exceeds the output voltage, the circuit operates as a linear regulator in which efficiency is approximately V_{OUT}/V_{IN}.

Figure 2



The efficiency of the circuit in Figure 1 depends on whether the circuit acts as a boost converter or a linear regulator.

Is this the best Design Idea in this issue? Select at www.ednmag.com.

Circuit transmits ARINC 429 data

Steve Woodward, University of North Carolina, Chapel Hill, NC

The ARINC (Aeronautical Radio Inc) 429 specification defines the air transport industry's hardware and protocol standards for the transfer of digital data between avionics systems. Circuitry that can implement elements of the 429 spec is often an essential part of control and sensor electronics intended for the aviation environment. ASIC chips for this purpose are commercially available, but they typically require nonstandard power supplies (for example, ±15V) and wide parallel interfaces. Therefore, it's sometimes inconvenient to accommodate them in 5V microcontroller-based designs. The circuit in **Figure 1** serves the Tx (transmit/output) portion of the 429 spec. The design implements a high-speed ARINC-429-compliant transmit function using a single 5V supply rail and 74HC series chips.

The physical transmission medium for the 429 standard is 78Ω shielded, twisted-pair cable that uses a complementary, differential bipolar RZ (return-to-zero) waveform (**Figure 2**). The voltages are the net differentials that the biphase drive develops: For example, the differential is 10V when you drive the Data A signal in **Figure 1** to 5V and the Data B signal to −5V. In addition to the signal levels, a 429 system must closely control the rise and fall times to conform to the specification. This control limits both in-

TABLE 1—FORMAT OF SPI BITS

		SPI bit							
Byte	ARINC field	7	6	5	4	3	2	1	0
1	Label	L	L	L	L	L	L	L	L
2	LS data	N	N	N	N	N	N	SDI	SDI
3	Data	N	N	N	N	N	N	N	N
4	Parity+MS data	OPB	SSM	SSM	N	N	N	N	N
5	Spacer byte	0	0	0	0	0	0	0	0

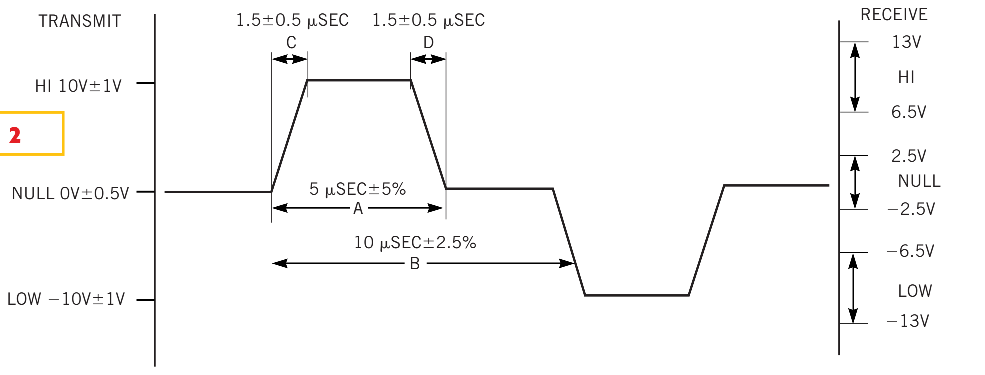
L…L = ARINC label (most-significant bit transmitted first)
SDI = ARINC source/destination identifier
N…N = 19 data bits (may include SDI and SSM fields; transmit least-significant bit first)
SSM = ARINC sign/status matrix
OPB = Odd parity bit (makes total count of "1" bits in the 32-bit word odd)

tra-cable signal crosstalk and EMI radiation that might interfere with sensitive aircraft communication and navigation systems. The operation of the transmitter in **Figure 1** centers around IC_1, a 4-bit×16-word FIFO memory. The serial ARINC bit stream comes from the microcontroller's synchronous serial-peripheral interface (SPI); the C input of IC_1 buffers the data stream. In addition, the D input of IC_1 serves as a buffered ARINC-enabled bit (the microcontroller's J port, bit 2). When low, this bit disables the ARINC transmitter logic and permits other system peripherals to use the SPI hardware.

Bits A and B of IC_1 go unused. The 100-kHz ARINC high-speed baud rate comes from the 1-MHz reference supplied to the IC_{3A} divide-by-10 circuit. The 100-kHz signal drives IC_1's shift-out (SO) pin and the IC_2 pulse gate. The presence of bits in the IC_1 FIFO (indicated by QR=1) resets the ARINC RDY bit in IC_{3B} and enables IC_{3A}. If IC_1's D bit (Pin 10) is also high, it gates the 100-kHz square wave to the IC_4 multiplexer. This action causes the sequential gating of −5, 0, and 5V onto the A/B data-output signals in ARINC-compatible waveforms. The LRC network at the output ensures compliance with the 429 requirements for rise and fall times. The circuit must process five 8-bit SPI bytes to generate each 32-bit, 429-compliant output word. **Table 1** shows the format of the SPI bits. The first four bytes in **Table 1** combine to form a 32-bit ARINC 429 word. The 32-bit word, reading from right to left, starts with byte 1 (again, reading from right to left), then tacks on byte 2, and so on.

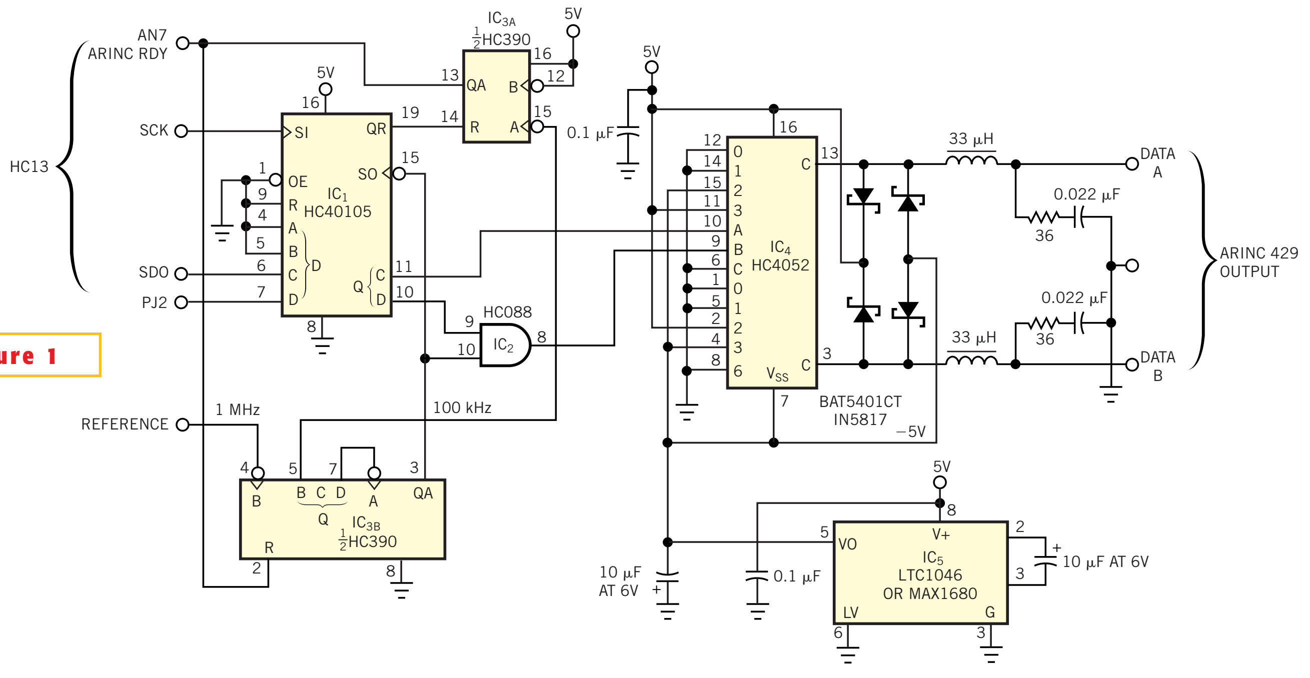
Is this the best Design Idea in this issue? Select at www.ednmag.com.

Figure 2



The ARINC spec imposes rigid requirements on waveforms.

Figure 1



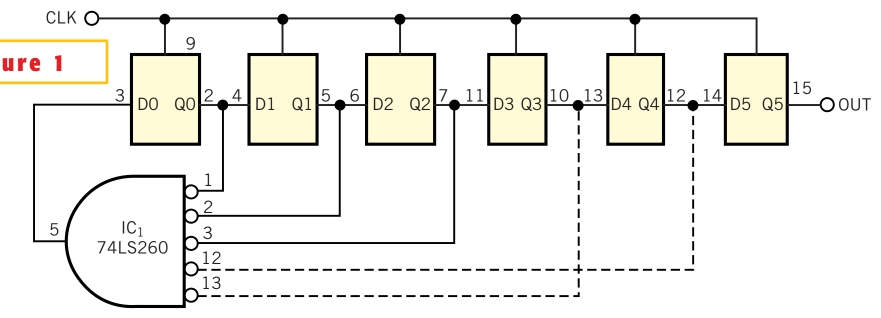
This ARINC 429 transmitter operates from a single 5V supply and meets all timing requirements of the 429 spec.

Circuit divides frequency by N+1

Bert Erickson, Fayetteville, NY

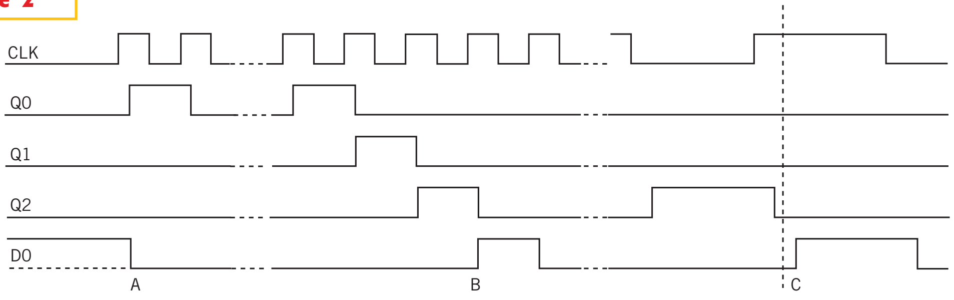
Digital frequency dividers usually use flip-flop stages that connect the $\overline{Q}$ pin to the D data-input pin of the following stage. This configuration creates a binary waveform that you can feed back to the input. You can divide any integer lower than 2^N with minimal stages, where N is the number of stages. These dividers can easily select one frequency from 100 for a receiver. However, as the applied clock rate approaches the ratings of the devices, decoding spikes appear. As a result, you'd be ill-advised to use the dominant pulse in such a waveform for clocks or strobes. The divider in **Figure 1** uses a ring configuration, and the stages connect Q-to-D, without using the $\overline{Q}$ output, to provide a binary sequence. Consequently, the circuit can divide only by N+1, but it produces a clean waveform at an applied clock rate that's substantially higher than you can apply to conventional binary dividers using flip-flops from the same process families.

Figure 1



This N+1 divider displays no decoding spikes, even at high clock rates. The added NOR circuit ensures proper start-up.

Figure 2



These waveforms illustrate the operation of the N+1 divider; note the propagation delay from the NOR gate in Section C.

If the last Q in a cascade of 74xx174 flip-flops connects to the D input, the loop becomes a shift-register ring counter. Moreover, the circuit can operate at a clock rate higher than that of any other configuration. Unfortunately, when you turn on the power or ground the reset pin, all Q outputs go low and remain low when you apply the clock. To circumvent installing a preset circuit that must operate at a high clock rate, you can place one NOR gate with its propagation delay in the loop. This addition ensures that the divider always starts and continues to function properly. Because this gate receives inputs from all stages at once, it features parallel carry and has the properties of a parallel-carry counter. For simplicity, **Figure 1** does not show the reset line, and you can delete the broken feedback lines, providing that you ground the corresponding input pins. You can take the output from any Q pin and use an additional stage, such as Q5, for a buffer. Although the output of the NOR gate resembles one of the Qs, you should not use it as an output.

Figure 2 shows the timing diagram for the divider. Section A of the diagram shows the state of D0 and the Q outputs in the start-up condition before the first and second clock pulses arrive. Note that when the power turns on or when the reset pin connects to ground, the Q outputs are all low, and D0 is high. D0 then transfers to Q0 on the rising edge of the first clock pulse. Section B of the diagram shows the output for a repetitive sequence starting with Q0. Section C is an expanded representation of the end of the sequence. Note that Q2 falls after the clock pulse rises. Then, with all Q outputs low, D0 rises a short time later to allow the sequence to repeat. D0's transfer can take place only after the next clock pulse rises. This factor creates an additional time slot to make the total N+1. Section C shows the propagation delay attributable to the NOR gate—approximately 10 nsec for most logic-circuit families and less than 5 nsec for the F and S series. This propagation delay is the only additional delay in the loop. A comparison of the output waveforms with those from a 74xx90 divide-by-5 counter shows prominent decoding spikes from this counter. The N+1 divider had no spikes and operates at a much faster clock rate.

Is this the best Design Idea in this issue? Select at www.ednmag.com.

Maintain precise timing with PC's speaker logic

DS Oberoi, CEDTI, and Harinder Dhingra, GCET, Jammu, India

Precise timing control is paramount in data acquisition and analysis and especially in digital-signal processing. The easiest way of maintaining timing control in a PC is to use delay loops. The disadvantage of this implementation is that the delay loop's elapsed time depends on the system's operating frequency. Hence, a program works accurately with a single operating frequency, but you must modify it for other frequencies. You can use add-on cards to achieve timing control, but they're costly, and, for simple operation, they remain underused. This Design Idea explores another way to obtain timing control. You can implement a stable delay loop with the aid of the PC's speaker logic. In PC/AT architecture, a 16-bit 8255 timer/counter IC is available, and the IC's operating frequency of 1.1931817 MHz is fixed across the entire range of PC families. You can use this fixed-frequency feature to implement a delay loop.

In PC/AT architecture, Counter 0 serves as a system timer and to maintain the time of day. Counter 1 generates pulses and serves as the DRAM refresh-rate generator. The PC uses Counter 2 for sound generation through the PC's speaker. Only Counter 2 is available for an implementation of the delay loop. You can enable or disable Counter 2 by setting bit 0 of the 8255's Port B to 1 or 0, respectively. For our purposes, Counter 2 operates as a rate generator (in Mode 2 operation). In this mode, the counter automatically decrements by one count with every counter clock pulse (1/1.1931817 MHz=0.838095 µsec= counter_count_time). Hence, the total period that Counter 2 can measure is approximately 54.92 msec (65,536×0.838905 µsec). **Listing 1** gives the details of implementing the delay loop. You can download **Listing 1** from the Web version of this Design Idea at www.ednmag.com.

The software sets Counter 2 to operate in Mode 2 by setting the control-word value to 0b4h and by writing this control-word data to the control-register port (043h). A routine called delay_loop() actually implements the delay loop. The routine first configures Counter 2 by writing the control word in the control register. Then, the program loads Counter 2 with starting (ffffh) data at address 042h in a 2-byte operation. The routine then enables the counter by setting bit 0 of the 8255's port B (in other words, port 61h) to one. Once enabled, Counter 2 automatically decrements by one count every 0.8380958 µsec. Before calling the delay routine, the software computes the total of Counter 2's count required for the desired delay (desired_delay_time). A "while" loop repeatedly reads back Counter 2's data from its input port (042h) in a 2-byte operation.

The two bytes combine to produce 16-bit data (counter_data). This data helps to compute the elapsed counts (count_elapsed); as soon as the required count is reached, the software exits this loop. The routine disables Counter 2 by setting bit 0 of the 8255's port B to 0 before returning to the calling program. Thus, it achieves the desired delay. Because the count depends on a fixed operating frequency across the entire PC family, you can use this method to implement a precise and stable timing loop. You can achieve delays of approximately 20 µsec to 54.9 msec. The software is in Turbo C++ and assembly language and was tested it in MS-DOS mode on a 450-MHz Pentium II computer.

LISTING 1—DELAY-LOOP ROUTINE

```
#include <conio.h>
#include <dos.h>
#define counter_count_time 0.0008380958  /* Counter2's single count period(mS) */
#define read_port 0x378                   /* Address of Read Port        */

unsigned counter_data, count_elapsed, required_count;
float desired_delay_time;

void delay_loop(unsigned delay_count);
main()
{
        desired_delay_time=0.9;     /* Delay Time in milli second */
  /* Find how many  counts are required for the  desired delay */
        required_count = desired_delay_time/counter_count_time;
        for (;;)
        {
         ; /*  Place the desired statements here, before delay */
         ;

         delay_loop(required_count); /* Call The Delay Routine */

         ; /*  Place the desired processing statements here, after delay */
         ;
        }
}

/* Delay Routine starts here */
void delay_loop(unsigned delay_count)
{
        count_elapsed = 0;
        asm mov al, 0b4h    /* Set the counter2 operation */
        asm out 043h,al
        asm mov al, 0ffh    /* Load the intial count as ffffh           */
        asm out 042h, al    /* completed in two cycles         */
        asm out 042h,al
        asm mov al, 01h     /* Enable the  counter and Start the ON period */
        asm out 061h, al

/* Counter data is read till,  desired counts are not  obtained           */
        while (count_elapsed < required_count)
        {
          asm in al, 042h        /*  Get LSB      */
          asm mov cl, al
          asm in al, 042h              /*  Get the MSB    */
          asm mov ah, al
          asm mov al, cl               /* Two bytes, combined to get 16 bit data*/
          asm mov counter_data , ax
          count_elapsed = (0xffff-  counter_data);  /* Calculate the elapsed counts  */
        }
/* Disable the counter2 */
          asm mov al, 00h
          asm out 061h, al
}
```

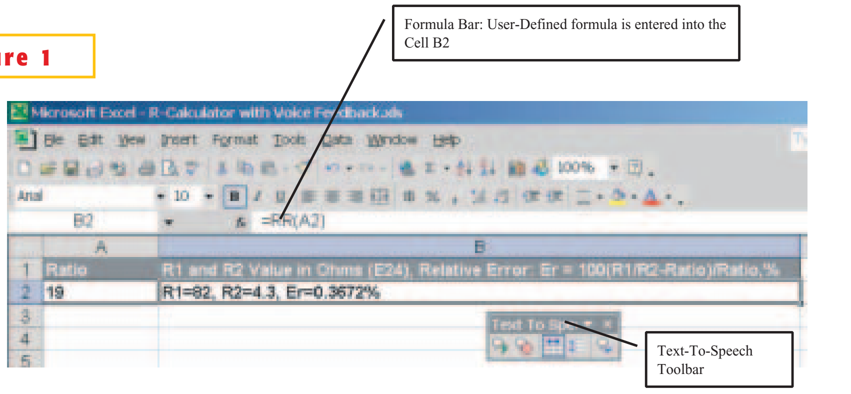
Is this the best Design Idea in this issue? Select at www.ednmag.com.

Voice feedback enhances engineering calculator

Alexander Bell, Infosoft International Inc, Rego Park, NY

The screen shot in **Figure 1** represents a Microsoft Excel 2002 worksheet designed to implement VFI (voice-feedback interface) for an engineering calculator. The voice-interface technique has both practical and educational aspects. It automates the common task of finding the values of two resistors for a given ratio. It also demonstrates the latest advances in natural-language programming technology with an example of the technology's actual implementation in CAD/CAE systems. A single user-defined function RR (resistor ratio) encapsulates both the computation engine and the VFI. The function uses VBA (Visual Basic for Applications) with the code placed in the standard code module of Excel File (**Listing 1**). You can download **Listing 1** and the Excel worksheet from the Web version of this Design Idea at www.ednmag.com.

Figure 1


Formula Bar: User-Defined formula is entered into the Cell B2

Text-To-Speech Toolbar

Microsoft Excel - R-Calculator with Voice Feedback.xls

B2 =RR(A2)

	A	B
1	Ratio	R1 and R2 Value in Ohms (E24), Relative Error: Er = 100(R1/R2-Ratio)/Ratio,%
2	19	R1=82, R2=4.3, Er=0.3672%

Cell A2 serves for data (ratio) entry; cell B2 contains the user-defined formula 5RR(A2). In automatic calculation mode, every time you enter a new ratio value in A2, the system recalculates R_1, R_2, and Relative Error.

The core search algorithm, which contains outer and inner loops, sequentially

LISTING 1—CODE MODULE FOR VOICE FEEDBACK

```
Option Explicit

Function RR(ByVal Ratio As Variant) As String
'** AUTHOR: ALEXANDER BELL
'** DATE  : 02/02/02
'** USAGE : FIND THE RESISTORS' VALUE R1 AND R2 FROM E24 SERIES WHICH MAKES THE
BEST
'**       : APPROXIMATION FOR GIVEN RATIO, BASED ON CRITERIA: MIN(ABS(RATIO-R1/R2)).
'**       : FORMULA LIMITATIONS: 1=< RATIO < 100 000 000.
'**       : NOTE: RETURN RESISTORS' VALUES ARE SHOWN IN OHMS
On Error GoTo ErrorHandle
   Const MAX_ORDER As Integer = 8  '** CONST USED TO SET LIMIT FOR RATIO (<100 000 000)
   Dim R1 As Double, R2 As Double  '** RESISTORS' VALUE VARIABLES
   Dim dblError As Double         '** ERROR OF APPROXIMATION VARIABLE
   Dim I As Integer, J As Integer  '** LOOP COUNTERS
   Dim dblMantissa As Double       '** MANTISSA VARIABLE
   Dim intOrder As Integer        '** ORDER VARIABLE
   Dim arrE24 As Variant          '** VARIABLE REFERENCING TO E24 VALUES ARRAY
   Dim str_MSG As String           '** VARIABLE CONTAINING THE VOICE NOTIFICATION TO
USER

'** DATA VALIDATION WITH VOICE FEEDBACK
*****************************************************
If CStr(Ratio) = "" Then
   str_MSG = "PLEASE, ENTER THE NUMERIC VALUE."
   Application.Speech.Speak str_MSG, True
   Exit Function
End If
If Not IsNumeric(Ratio) Then
   str_MSG = "WRONG ENTRY: PLEASE, ENTER THE NUMERIC VALUE."
   Application.Speech.Speak str_MSG, True
   Exit Function
End If
If Ratio < 1 Then
   str_MSG = "WRONG ENTRY: RATIO MUST BE MORE OR EQUAL 1, OTHERWISE USE THE
INVERSE VALUE."
   Application.Speech.Speak str_MSG, True
   Exit Function
End If
If Ratio >= 10 ^ MAX_ORDER Then
   str_MSG = "WRONG ENTRY: RATIO NUST BE LESS THAN " & CStr(10 ^ MAX_ORDER)
   Application.Speech.Speak str_MSG, True
   Exit Function
End If

'** E24 SERIES ARRAY. LAST VALUE (10) IS INCLUDED TO SIMPLIFY THE SEARCH
ALGORITHM
arrE24 = Array(1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.8, 2, 2.2, 2.4, 2.7, _
        3, 3.3, 3.9, 4.3, 4.7, 5.1, 5.6, 6.2, 6.8, 7.5, 8.2, 9.1, 10)

'** CALCULATE ORDER AND MANTISSA FOR GIVEN RATIO
intOrder = Int(Log(Ratio) / Log(10))
dblMantissa = Ratio / (10 ^ intOrder)
'** SET RESISTORS INITIAL VALUES
R1 = 1: R2 = 1
dblError = Abs(R1 / R2 - dblMantissa)

'** LOOP THROUGH E24 ARRAY TO FIND R2 AND R1 MANTISSA RESULTING IN A MIN ABS
ERROR
'** NOTE: THIS ALGORITHM RETURNS R1 MANTISSA WHICH IS ALWAYS BIGGER OR
EQUAL R2
For I = LBound(arrE24) To UBound(arrE24)
  For J = LBound(arrE24) To I
    If Abs(arrE24(I) / arrE24(J) - dblMantissa) < dblError Then
      R1 = arrE24(I)
      R2 = arrE24(J)
      dblError = Abs(arrE24(I) / arrE24(J) - dblMantissa)
    End If
  Next J
Next I

'** CALCULATE R1 VALUE IN OHMS AND RELATIVE ERROR IN (%)
R1 = R1 * 10 ^ intOrder
dblError = (R1 / R2 - Ratio) / Ratio

'** RETURN RESULT STRING: RESISTORS' VALUES IN OHMS, RELATIVE ERROR IN %
RR = "R1=" & CStr(R1) _
       & ", R2=" & CStr(R2) _
       & ", Er=" & Format(dblError, "0.0000%")
'** SEND "OK" VOICE NOTIFICATION TO USER AND EXIT FUNCTION
str_MSG = "OK. "
Application.Speech.Speak str_MSG, True
Exit Function
ErrorHandle:
'** IN CASE OF ERROR RETURN EMPTY STRING AND SEND THE ERROR VOICE
NOTIFICATION TO USER
  RR = ""
  str_MSG = "UNANTICIPATED ERROR IN CALCULATIONS."
  Application.Speech.Speak str_MSG, True
End Function
```

tests each pair of values, R_1 and R_2, to find the best approximation of the target ratio. In other words, the algorithm tries to minimize the absolute error: (ABS ($R_1/R_2-\text{Ratio}$)). The values of R_1 and R_2 come from the E24 EIA-standard series, but you can apply the same algorithm to any other standard series, such as E48, E96, or E192. The VFI sends the status-notification message in a verbal form instead of showing the Message Box. The VFI uses the built-in Excel 2002 Speech Object with the following syntax:

Application.Speech.Speak
<String Variable or Constant>, True,

where <String Variable or Constant> contains the actual spoken text, and the second property is set to True for asynchronous mode. **Listing 2** is an example of voice-error notification in the case that you enter non-numeric data as the ratio (data-validation error message):

LISTING 2—VOICE-ERROR NOTIFICATION

```
IfNot IsNumeric(Ratio) Then
  Str_MSG5"Wrong ENTRY: PLEASE, ENTER THE NUMERIC VALUE."
  Application.Speech.Speak str_MSG, True
  Exit Function
End If
```

Using the technique is simple. Open Excel File, switch to the Visual Basic Editor window (press Alt-F11), add the standard module, and paste the code from **Listing 1**. From the Debug menu item, choose Compile VBA Project, save the file with any name, and close the Visual Basic Editor window. Make sure you activate the text-to-speech tools (in the menu: Tools—Speech—Show Text To Speech Toolbar). Check whether you are in automatic- or manual-calculation mode (in the menu: Tools—Options—Calculation). Set the mode to automatic; otherwise, you'd have to use the F9 key to force a new calculation every time you enter a new ratio value. Choose any cell (for example, A2 in **Figure 1**) for the ratio (data-entry cell) and another cell (for example, B2) to display the results. Enter the formula =RR(A2) into cell B2. Now, every time you enter a new ratio value into cell A2, the system automatically calculates and displays R_1, R_2, and Relative Error, and sends the status voice notification. The notification is either "OK" to confirm the successful completion of the calculation or an error notification in the case of a data-validation or computation error. Note that some macros in Microsoft Office applications could result in potentially dangerous and harmful actions, and some may contain viruses. You use the macros at your own risk without warranties of any kind.

Is this the best Design Idea in this issue? Select at www.ednmag.com.