Edited by Bill Travis and Anne Watson Swager

Free-line indicator stops interruptions

JM Terrade, Clermont-Ferrand, France

When only one phone line is available for two phones, each time you want to make a call, someone may be using the second phone. A simple circuit lights an LED, which indicates whether the line is free (**Figure 1a**). Batteries are unnecessary; the phone line powers the circuit, and an accumulator saves energy for an "in use" indication. A rectifier bridge ensures that the voltage is positive for the circuit. You can safely use this circuit on a private phone line, but you may need authorization before connecting it to your operator line.

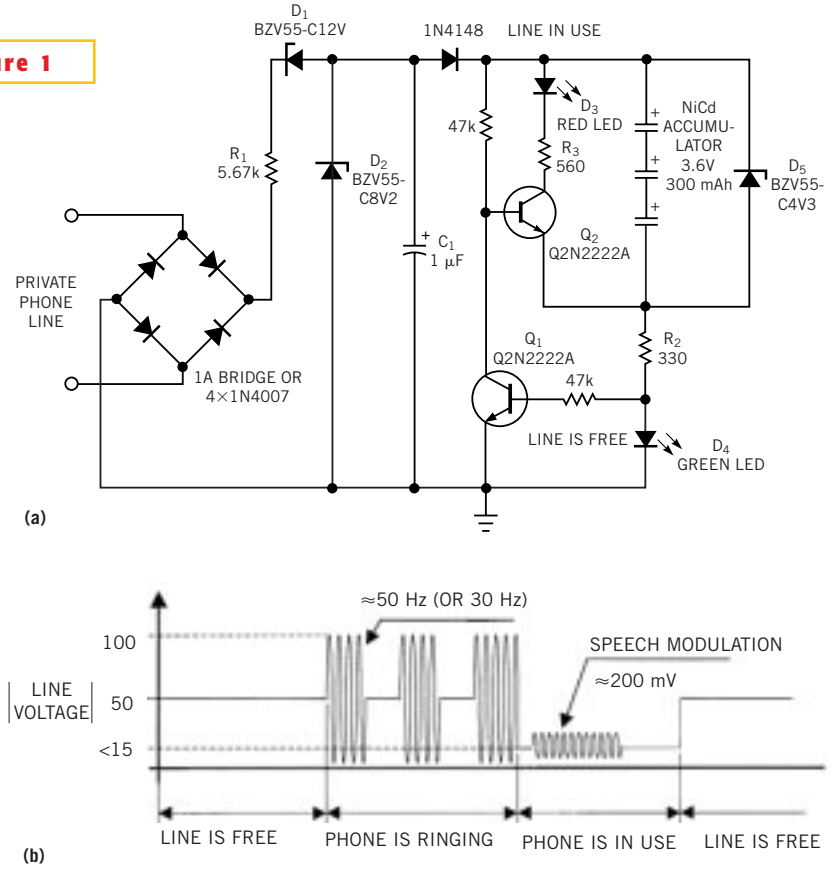
A phone line has different voltages between terminals depending on the line's availability. Three possible states exist (**Figure 1b**). **Figure 1b** shows the absolute value of the line voltage, because you can switch the line terminals.

In phase 1, the line is free, and the voltage is a continuous 50V dc voltage. The series zener diode, D_1, decreases the voltage by 12V, and R_1 and D_2 further limit the voltage to 8V. The current now flows through the NiCd accumulator, R_2, and D_4. The green LED, D_4, turns on, and the voltage across D_4 turns on Q_1. Q_2 is off, and there is no current for D_3. R_2 limits the current to 3 mA, which is enough to charge the accumulator. The green LED is a low-current model. D_5 protects the accumulator against overvoltage. Total line consumption is 5 mA because an extra 2 mA of current flows through D_2.

In phase 2, when the line is ringing, an extra ac voltage with an amplitude of 50V adds to the 50V dc voltage. In this case, the value of C_1 is critical. If C_1 is 1 µF, both LEDs will turn on because the 15V voltage value will vary. If C_1 is 47 µF, the voltage remains greater than 15V, and D_4 turns on.

In phase 3, when answering the call, the voltage falls to a value of about 10V. Voice modulation adds to this continuous voltage. The operator considers a phone line as "in use" if a current in the phone draws close to 30 mA through its 300Ω equivalent impedance. These current and impedance values are not criti-

Figure 1



D_4 lights to indicate that the phone line is free, and D_3 lights to indicate the line is in use (a), depending on the three possible phone-line voltage states (b).

cal. The line voltage, which is less than 15V, blocks D_1, and the voltage across D_2 is almost zero. Current discontinues its flow through R_2, and D_4 turns off. Q_1 is also off, and Q_2 conducts. Current travels from the accumulator through R_3, and D_3 turns on. R_3 limits the current to 3 mA, which is enough for a low-current LED. The 300 mAh, 3.6V accumulator is a phone type. If you unplug the circuit, D_3 remains on until the accumulator discharges.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

Generate CID/CIDCW analog signals

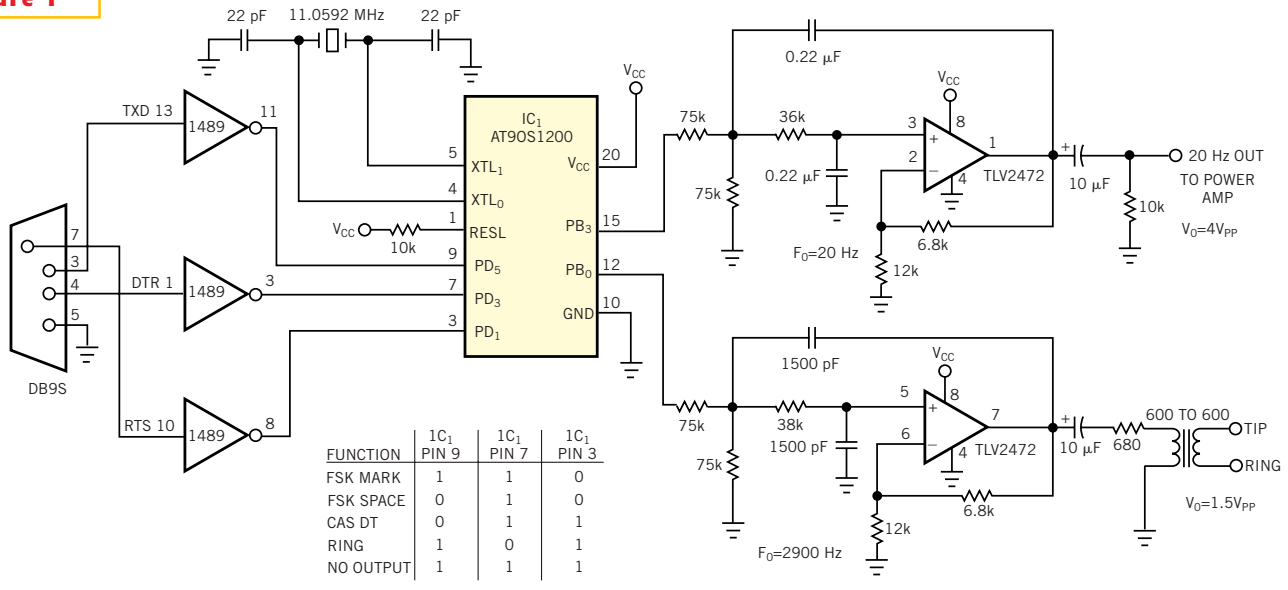
Hans Krobath, EEC, Nesconset, NY

A HIGH-SPEED µP and active lowpass filters can generate CID (caller-ID) and CIDCW (caller-ID-on-call-waiting) analog signals (**Figure 1**). CID data transmits at 1200 baud FSK between the first and second 20-Hz ring of an incoming call. CIDCW uses a CAS (CPE Alert Signal) dual tone, which consists of 2130 and 2750 Hz to initiate the FSK data transfer. You can produce these analog signals using software-generated PWM (pulse-width-modulated) outputs from a high-speed µP, such as an Atmel AVR or Scenix SX. You create these PWM outputs using a constant-sampling frequen-

LISTING 1—TABLE GENERATION FOR PWM MARKING AND SPACING SEQUENCE

```
PI = 3.1415926#
KM = 182                ' MAXIMUM RANGE OF PULSE WIDTH
DT = 576 / 11059200     ' TIME FOR EACH SAMPLE
DTI = DT
F1 = 19200 / 7          ' F1 = 2742.9 (2750)
F2 = 19200 / 9          ' F2 = 2133.3 (2130)
DTI = DT
FOR I = 1 TO 63         ' 63 SAMPLES THEN REPEAT
A = SIN(2 * PI * F1 * DT)   ' GENERATE F1 SINE WAVE
B = SIN(2 * PI * F2 * DT)   ' GENERATE F2 SINE WAVE
V = KM / 2 + (KM / 4) * A + (KM / 4) * B  'GENERATE F1 + F2 SINE WAVE
PRINT I, DT, V
DT = DT + DTI           ' NEXT SAMPLE
NEXT I
```

Figure 1



FUNCTION	$1C_1$ PIN 9	$1C_1$ PIN 7	$1C_1$ PIN 3
FSK MARK	1	1	0
FSK SPACE	0	1	0
CAS DT	0	1	1
RING	1	0	1
NO OUTPUT	1	1	1

Based on RS-232 inputs from a PC, pin 12 of IC_1 produces a PWM output proportional to FSK 1200-baud serial data or a dual-tone CAS signal. The output at pin 15 is a 20-Hz-ring signal. Subsequent lowpass filtering produces sine-wave outputs.

cy, and you select this frequency so that small, repetitive integer-number samples define the desired output. The 11.0592-MHz µP clock frequency and single-instruction cycle time are the same for IC_1. The design produces the 1200-baud FSK marking frequency of 1200 Hz by using 22 cycles of a constant-sampling frequency of 11.0592 MHz/419. The design produces the 2200-Hz spacing frequency by using 12 cycles of the same sampling frequency. At each sample, the µP produces a PWM TTL output proportional to the analog voltage. The µP obtains successive pulse widths from a look-up table that also provides pointers to allow phase-coherent transitions when switching between marking and spacing. A Basic program produces the tables for the marking and spacing PWM sequence (**Listing 1**).

A sampling frequency of 11.0592 MHz/576, or 19,200 Hz, produces the dual-tone 2750- and 2130-Hz CAS signal. Seven samples define 2742.9 Hz, and nine samples define 2133.3 Hz. These samples are well within the Bellcore specification. The sum of these waveforms repeats 7×9=63 times when you use the 19,200-Hz sampling frequency. Another Basic program produces the pulse-width outputs that generate the dual tone (**Listing 2**).

The design produces the 20-Hz-ring sine wave, which you can amplify and step up to the appropriate level in a similar manner using values to produce pulse widths that **Listing 3** provides.

An assembly-language program written for IC_1 uses the table-generating programs in **Listings 1**, **2**, and **3**. The program fits within the 512-word memory space to generate three PWM outputs, which comprise the analog signals for CID/CIDCW after lowpass filtering. You can download the program 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 #2622.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

LISTING 2—TABLE GENERATION FOR DUAL-TONE PWM OUTPUTS

```
PI = 3.1415926#
KM = 182                ' MAXIMUM RANGE OF PULSE WIDTH
DT = 576 / 11059200     ' TIME FOR EACH SAMPLE
DTI = DT
F1 = 19200 / 7          ' F1 = 2742.9 (2750)
F2 = 19200 / 9          ' F2 = 2133.3 (2130)
DTI = DT
FOR I = 1 TO 63         ' 63 SAMPLES THEN REPEAT
A = SIN(2 * PI * F1 * DT)   ' GENERATE F1 SINE WAVE
B = SIN(2 * PI * F2 * DT)   ' GENERATE F2 SINE WAVE
V = KM / 2 + (KM / 4) * A + (KM / 4) * B  'GENERATE F1 + F2 SINE WAVE
PRINT I, DT, V
DT = DT + DTI           ' NEXT SAMPLE
NEXT I
```

LISTING 3—TABLE GENERATION FOR A 20-Hz-RING SINE WAVE

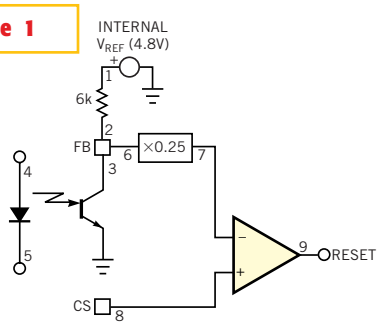
```
PI = 3.1415926#
KM = 246                ' MAXIMUM RANGE OF PULSE WIDTH
DT = 27648 / 11059200   ' TIME FOR EACH SAMPLE
DTI = DT
F = (11059200 / 27648) / 20 ' OUTPUT FREQUENCY
FOR I = 1 TO 20             ' 20 SAMPLES THEN REPEAT
V = KM / 2 + (KM / 2) * (SIN(2 * PI * F * DT))
PRINT I, DT, V
DT = DT + DTI           ' NEXT SAMPLE
NEXT I
```

Bipolars provide safe latch-off against opto failures

Christophe Basso, ON Semiconductor, Toulouse Cedex, France

A typical SMPS (switch-mode power supply) uses a reference voltage and an optocoupler to regulate the output voltage. The optocoupler carries the isolated information from the secondary side to the primary nonisolated side and ensures proper output regulation. Current-mode architectures obtain the final feedback level by implementing the optocoupler as a variable resistor that internally connects to a pullup element (**Figure 1**). The voltage on the feedback pin (FB) directly fixes the power supply's peak-current setpoint. When the output power demand is low, V_{FB} and the peak current are low. When the output requires power, V_{FB} increases and authorizes a higher peak current.

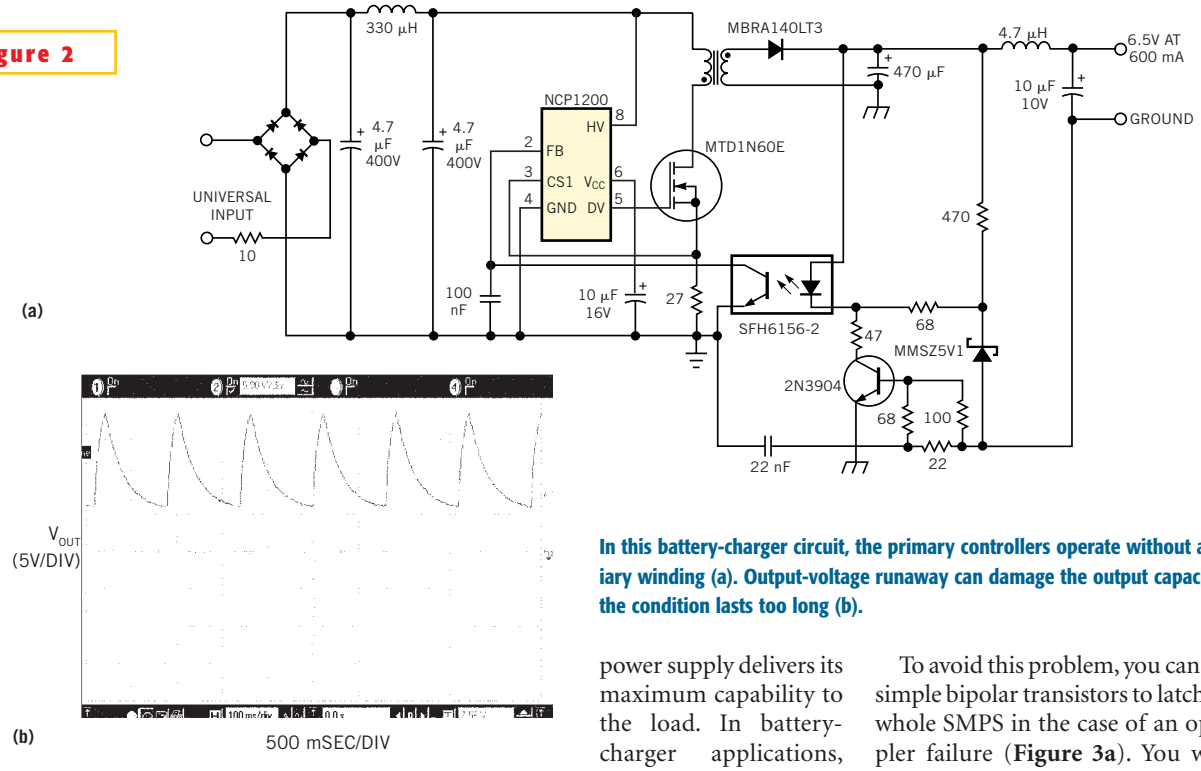
Figure 1



A typical current-mode architecture uses an optocoupler as a variable resistor.

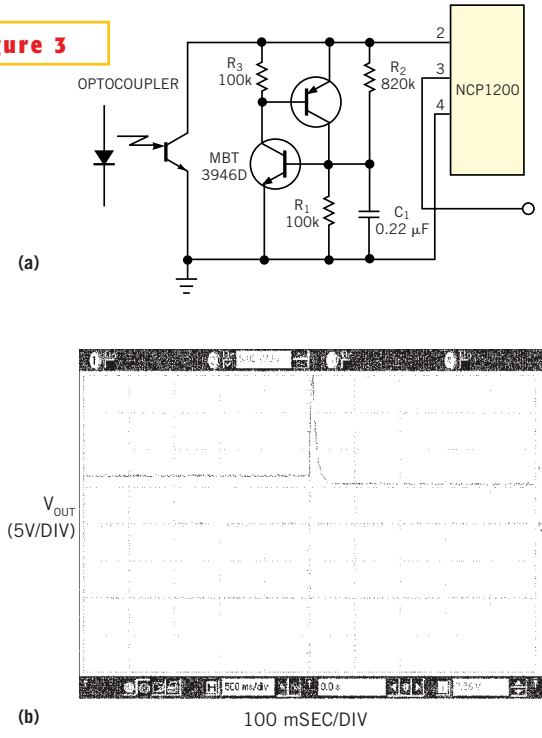
When an output short circuit is present, the LED optocoupler loses its bias, and the variable-resistor action disappears. The internal 6-kΩ pullup resistor forces V_{FB} to its maximum value, and the

Figure 2



In this battery-charger circuit, the primary controllers operate without any auxiliary winding (a). Output-voltage runaway can damage the output capacitors if the condition lasts too long (b).

Figure 3



You can use a dual npn+pnp to build a cheap thyristor (a). When the thyristor fires, the pulses permanently stop, leaving no voltage runaway (b).

power supply delivers its maximum capability to the load. In battery-charger applications, however, short-circuit conditions do not cause optocoupler loss because the output permanently monitors the delivered current and forces the current to be constant (**Figure 2a**). In this case, the primary implementation is simple because of the lack of the auxiliary winding. If the optocoupler fails to open, the peak-current setpoint increases to its maximum for a given time until the burst-protection feature takes over. This situation repeats until the user switches off the SMPS. The worst case arises in unloaded situations: The output voltage runs away until the burst sequence ends (**Figure 2b**). As a result, this condition can quickly damage output capacitors if this situation lasts too long.

To avoid this problem, you can use two simple bipolar transistors to latch-off the whole SMPS in the case of an optocoupler failure (**Figure 3a**). You wire the bipolars in a thyristor manner using a dual-transistor device, such as the MBT3946D.

In normal operation, R_1 through R_3 ensure that neither the pnp nor the npn can start conducting. Furthermore, R_1 and R_2 form a voltage divider that monitors V_{FB}. When V_{FB} increases, the voltage over C_1 begins to rise until the npn transistor starts to pull the pnp transistor's base to ground. This action immediately fires the SCR, which locks V_{FB} to nearly zero. When V_{FB} is less than 1.4V, the NCP1200 IC stops delivering pulses until the SCR resets. You can reset the SCR by unplugging the charger from the main outlet. **Figure 3b** shows the results of this operation and that the operation is safe with an open optocoupler. When the optocoupler fails, the output voltage grows until the SCR stops the IC operation. V_{OUT} then slowly discharges toward ground. C_1 filters out any spurious noise that appears at power-on that could adversely fire the SCR.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

Simple logic analyzer pushes μC to its limit

Tom Lyons Fisher, Inexpensive Systems, Huntingdon, PA, and Michael Deskevich, Juniata College, Huntingdon, PA

A simple logic-analyzer design is compatible with all versions of Windows and pushes the PIC 18C252 chip to its speed limit to achieve a 1-MHz sampling rate (**Figure 1**). The circuit can examine three channels of relatively low-speed logic signals that have infrequent, or sparse, transitions. The analyzer can record only 510 transitions per run, but a run can last as long as several minutes, if necessary. Applications include monitoring the I/O of an IBM keyboard or printer port, TI-calculator intercommunication, and serial (RS-232) signals.

Although the ability of the PIC 18C252 μC to use a 40-MHz clock input suggests that sampling rates in the megahertz region are easy to achieve, careful programming and unorthodox use of some of the PIC's features are necessary even to achieve a rate of 1 MHz. Nevertheless, the 18CXXX series works well for this application because it is the fastest PIC available, with 1536 bytes of RAM and an RS-232 port on-chip. Thus, this logic analyzer requires minimal external circuitry.

LISTING 1– CRITICAL CODE FOR THE LOOP IN MACRO FORM

```
Scan      MACRO
          LOCAL  Scn2,Scn3
;Has there been a transition on any channel?
  Scn3    movf   PORTB,W,A
          xorwf  Last,F,A
          movwf  Last,A
          bnz    Scn2
;If no transition, just increment the interval timer.
          tblrd  +*
          movff  TBLPTRU,INDF1
          goto   Scn3
;If transition, record the new logic states and the current value of interval timer.
  Scn2    iorwf  POSTINC1,F,A
          movff  TBLPTRH,POSTIC1
          movff  TBLPTRL,POSTINC1
          ENDM
```

Although the signal-input channels use the Schmitt-trigger inputs, which are available only on Port C, each channel has different capabilities. The red and yellow channels detect 5V logic signals. You can set the red channel to trigger on either a positive- or a negative-going transition. The black channel monitors only bipolar "external" RS-232 signals, which the MAX231 level translator converts into standard 5V logic (5V=logic 1).

With a 40-MHz clock, the PIC has a machine cycle of 10 MHz, or 100 nsec, so the sampling-code loop must use no more than 10 cycles for a sampling rate of 1 MHz. Because of stringent time constraints, a summary of the software strategy is "save fast and pick up the pieces later." This device records the time between transitions and the logic state of the three channels after the transition. **Listing 1** shows the critical code for the loop in macro form, before expansion. It takes exactly 10 μsec to determine whether there is a transition to record. Notice that this design uses the external address register, TBLPTR, as a timer/counter because a two-cycle command can increment the register's 21 bits. Because there is no time to store 3 bytes of a transition event within 10 cycles, the design stores the upper timer/counter byte when there is no transition, leaving only the logic states and the other 2 timer/counter bytes for you to store when there is a transition. The compiler copies the entire loop into program ROM 510 times to avoid a goto command, which requires two cycles.

When the loop is active, it records the current state of the timer/counter, which

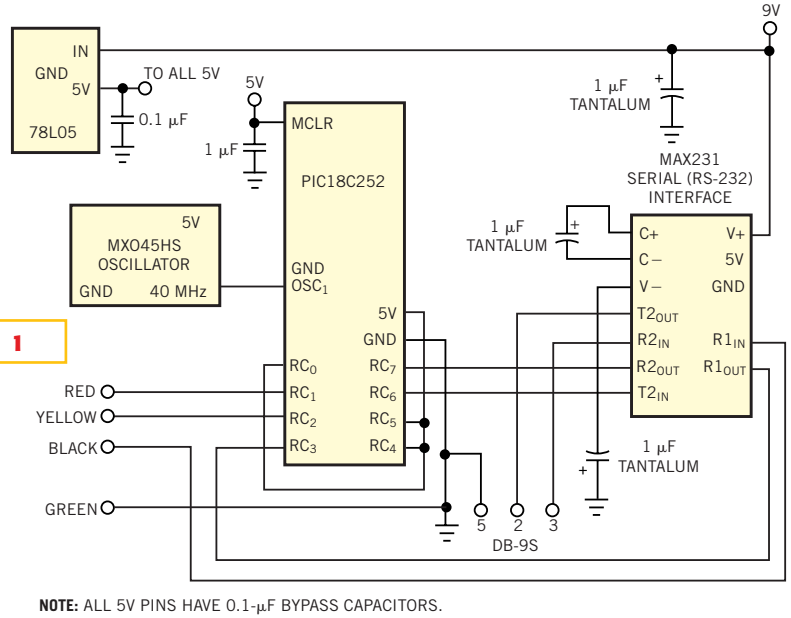


Figure 1

NOTE: ALL 5V PINS HAVE 0.1-μF BYPASS CAPACITORS.

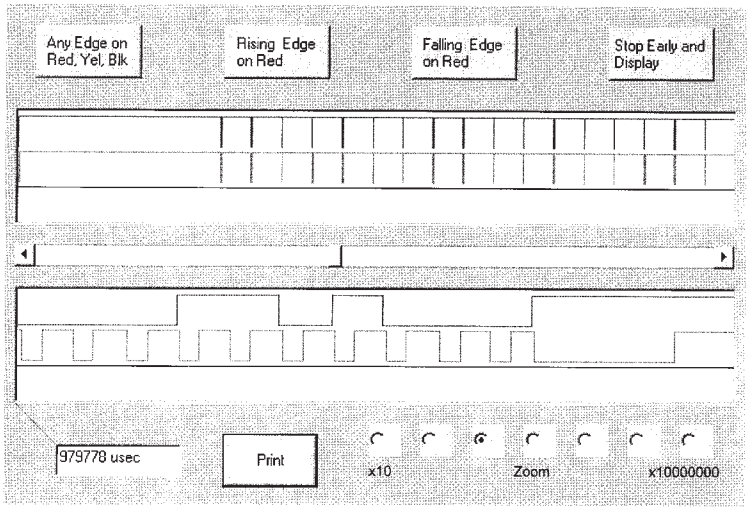
The external wiring is simple because the Schmitt triggers, memory, and UART are all within the μC.

may "roll over" if the intervals are long. After completing the run, the program makes a pass through the data to convert the recorded times to true intervals before sending the results to the PC. The resulting format is RYB-MMMMM IIIIIII LLLLLLL, where R, Y, and B are the logic states of the red, yellow, and black channels, and M, I, and L are the most-, intermediate-, and least-significant bits of the time interval between transitions in microseconds.

After the data from a run transmits through a PC communications port at 19,200 baud, a Visual Basic program displays the data. The screen displays the three traces in colors corresponding to the channel test leads. It also permits the user to expand the traces for detailed examination by using the mouse to manipulate a horizontal scroll bar and zoom buttons, which select the X-position and magnification of the expanded traces (**Figure 2**).

The Visual Basic program and both the PIC source and the object code are available 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 #2617.

Figure 2



The display shows the entire run in the upper plot and a magnified portion in the lower plot. The small window at the lower left reports the exact time of the leftmost displayed transition.

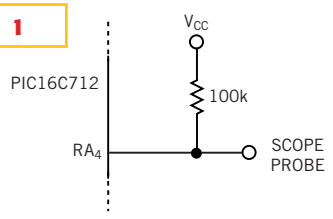
Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

PIC debugging routine reads out binary numbers

Brad Peeters, Theta Engineering Inc, Costa Mesa, CA

The "burn and learn" method of firmware development excludes an in-circuit emulator and a serial port. With this method, it is common practice to use spare I/O pins on a µC as a debugging aid. By strategically placing instructions to set and clear these I/O pins in the code and then observing the pins with a scope, you can obtain limited real-time information about the execution of the code. An I/O pin serves as a 1-bit debugging port.

Figure 1



Attaching a large-valued pullup resistor on an open-drain I/O pin results in fast falling edges but slow rising edges.

You can overcome this limitation somewhat by writing a function or subroutine that shifts data out serially on the port pin. Then you can use the scope to capture and observe several bits of information. However, setting the port pin high for a one bit and low for a zero bit results in a display that requires careful reading. Unless you know and accurately measure the timing of the bits, judging which bit corresponds to which position on the scope display presents a challenge. If the data includes several zeros or several ones in a row, no transitions exist with which to align the timing. This problem becomes particularly acute when attempting to capture more than 8 bits.

A software routine for midrange PIC µCs (**Listing 1**) overcomes this difficulty by producing a scope display that you can read at a glance. The routine encodes zero as a short pulse and one as a long pulse. Using an open-drain I/O pin with a large-value pullup resistor results in fast falling and slow rising edges, which is due to the RC time constant of the pullup resistor and the capacitance of the pin and scope probe (**Figure 1**). Consequently, zeros show up as short spikes in the display, and ones appear as medium spikes. The separation between consecutive bytes appears as a tall spike or pulse (**Figure 2**). Each byte starts with a clean falling edge, which serves as a convenient trigger signal for the scope. A midrange PIC running at 4 MHz using a 100-kΩ pullup resistor produces the plot in **Figure 2**. The resistor value is not critical. To use another clock rate, you can scale the resistor approximately as the inverse of

the clock rate. For example, at 8 MHz, a pullup resistor of 47 kΩ produces equivalent results.

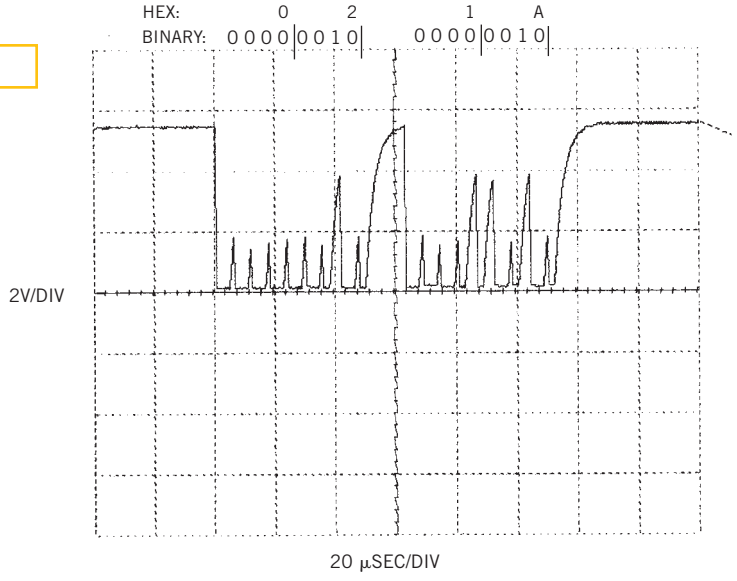
The scope plot depicts a 16-bit value of 00000010 for the first byte and 00011010 for the second byte. Each invocation of the subroutine in **Listing 1** displays the most-significant bit of 1 byte first. By taking care to invoke the subroutine on the most-significant byte of a multibyte value first, the scope display naturally reads from left to right. Hence, the depicted value is 021A hex. By slowing the timebase of the scope, you can display a 32-bit value. The resolution of the scope is the only limitation on the amount of data the scope can display.

Because the subroutine preserves all registers and flags, except for the general-purpose register for the subroutine itself, you can safely insert a call to the subroutine at any point in your code to obtain visibility into the value of the W register. The limitation on the W register is not a severe restriction because in the PIC architecture, most operations pass through the W register. One additional instruction suffices to load any general-purpose register into W before calling the subroutine.

The code snippet in **Listing 2** shows the addition of a 16-bit value called *Result* to a 24-bit *Base* value. Inserting *call Debug* instructions at the points that the arrows indicate makes the 16-bit *Result* value visible (**Figure 2**). You can download the subroutine 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 #2594.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

Figure 2



Inserting "call Debug" instructions into the code makes the 16-bit result visible. Zeros appear as short spikes, ones appear as medium spikes, and a tall spike indicates separation between consecutive bytes.

LISTING 2–CODE SNIPPET

```
movfw   ResultHi
addwf   BaseHi
btfsc   C
incf    BaseEx
movfw   ResultLo
addwf   BaseLo
btfsc   C
incfsz  BaseHi
goto    $+2
incf    BaseEx
```

LISTING 1–DEBUGGING SUBROUTINE

```
;Define the pin to be used as the test point:
#define  TP1        5,4         ;bit 4 of Port A.

;Allocate a register for use by the debug subroutine:
    cblock
    DebugReg
    endc

;-----------------------------------------------------------------
; This subroutine takes the contents of the W register and shifts
; it out, bit by bit, on the TP1 port pin for viewing on a scope.
; Long pulse is a one and short pulse is a zero.  W register and
; flags are preserved!
Debug
    movwf   DebugReg            ;Store value in reg.
    rlf     DebugReg            ;Shift MSB into CY.
    bcf     TP1                 ;"Start of byte" transition.
    btfsc   C                   ;If bit is set,
    bsf     TP1                 ;start pulse sooner,
    rlf     DebugReg
    bsf     TP1                 ;rather than later.
    bcf     TP1                 ;End of pulse.
    btfsc   C                   ;Test next bit and repeat…
    bsf     TP1
    rlf     DebugReg
    bsf     TP1
    bcf     TP1
    btfsc   C
    bsf     TP1
    rlf     DebugReg
    bsf     TP1
    bcf     TP1
    btfsc   C
    bsf     TP1
    rlf     DebugReg
    bsf     TP1
    bcf     TP1
    btfsc   C
    bsf     TP1
    rlf     DebugReg
    bsf     TP1
    bcf     TP1
    btfsc   C
    bsf     TP1
    rlf     DebugReg
    bsf     TP1
    bcf     TP1
    btfsc   C
    bsf     TP1
    rlf     DebugReg
    bsf     TP1
    bcf     TP1
    btfsc   C
    bsf     TP1
    rlf     DebugReg            ;Restore CY flag.
    bsf     TP1
    bcf     TP1
    nop                         ;Optional to provide consistent timin
    bsf     TP1                 ;"End of byte" transition.
    return
```