design ideas

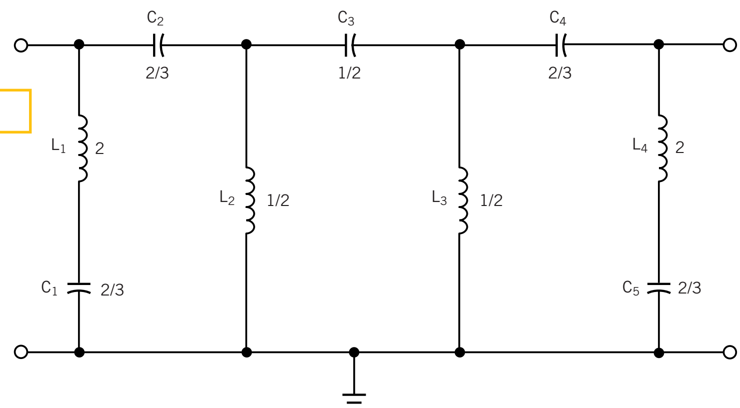
Edited by Bill Travis and Anne Watson Swager

Filter design uses image parameters

Richard Kurzrok, RMK Consultants, Queens Village, NY

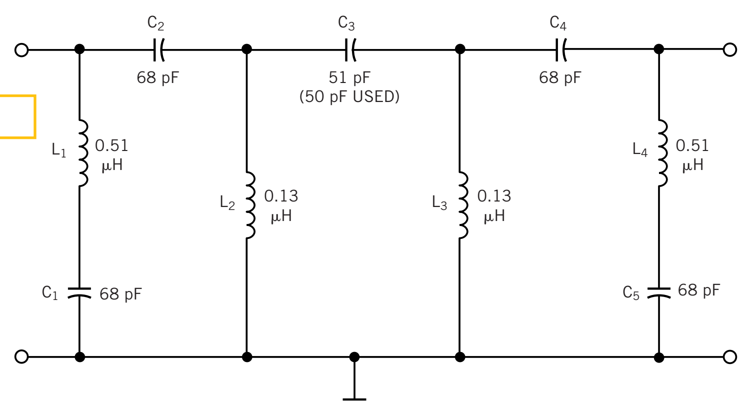
Reference 1 gives low-cost image-parameter design techniques for LC lowpass filters. Filter design using a low number of circuit elements results in reduced costs for both parts procurement and manufacturing. The technique applies to highpass filters. You derive a composite highpass filter by using m-derived terminating half-sections with one or more constant-k interior full sections. Classic image-parameter design used m-derived half-sections with m=0.6 for best passband impedance matching (in other words, high input and output return losses). The design uses a value of m=0.5 for the terminating half-sections. This value provides sharper close-in selectivity while maintaining passband return losses that are satisfactory for most applications. **Figure 1** shows the normalized schematic for the composite highpass filter. It uses midseries, m-derived, terminating half-sections with m=0.5, plus two interior constant-k full sections. The 3-dB cutoff frequency, f_0, is 31.2 MHz, and source and load impedances, Z_0, are 50Ω. Reference levels of filter inductance

Figure 1



Using image parameters results in a low number of circuit elements in a filter.

Figure 2



Multiplying the normalized filter in Figure 1 by the reference inductance and capacitance values yields this 31.2-MHz, 50V filter.

TABLE 1—FILTER PARTS LIST

Function	Value	Type	Quantity
L_1, L_4	0.51 µH	Micro-Metals T 25-10 14T- #26	Two
L_2, L_3	0.13 µH	Micro-Metals T 25-10 14T- #26	Two
C_1, C_2, C_4, C_5	68 pF	CD-15 Series dipped mica	Four
C_3	50 pF	DC-15 Series dipped mica	One
Connectors	BNC female	Pomona 2447 panel receptacle	Two
Enclosure	Aluminum box	Hammond 1590A/Bud CU-123	One
Board	Cut by hand	Vector board 169P44C1	One
Standoffs	Male/female	Amatom 9794-SS-0440	Four

Note: all fixed capacitors have ±15% tolerance.

and capacitance are as follows:

$$L_0 = \frac{Z_0}{2\pi f_0} = 0.255\ \mu H;$$

$$C_0 = \frac{10^6}{2\pi f_0 Z_0} = 102\ pF.$$

You obtain the actual inductance and capacitance values for the highpass filter by denormalization; in other words, by multiplying the normalized inductances and capacitances in **Figure 1** by L_0 and C_0, respectively. **Figure 2** shows the actual component values for a dissipationless highpass filter. **Table 1** gives the parts list for the filter. **Table 2** gives the measured amplitude response for the composite highpass filter. The results indicate

TABLE 2–MEASURED AMPLITUDE RESPONSE

Frequency (MHz)	Insertion loss (dB)
29	23.7
30	12.8
31	3.7
31.5	1.8
32	1
33	0.6
35	0.5
40	0.5
45	0.4
50	0.2
55	0.2
60	0.2
70	0.4
100	0.5
130	0.6

inductor unloaded Qs of approximately 100. As the passband frequency approaches 100 MHz, some modest shape degradation occurs. You can reduce the degradation by using microstrip construction with surface-mount components. You can trim the filter's cutoff frequency by spreading or squeezing the turns of the toroidal inductors. (DI #2533)

Reference

1. "Low Cost Lowpass Filter Design Using Image Parameters," *Applied Microwave & Wireless*, February 1999, pg 72, plus correction May 1999, pg 12.

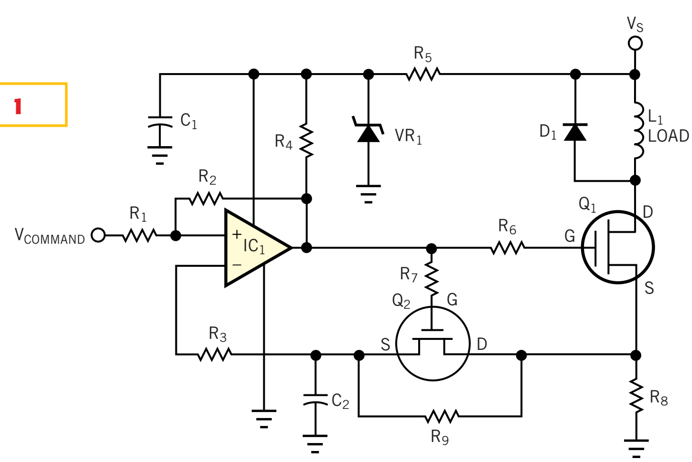
To Vote For This Design, Enter No. 362 at www.ednmag.com/infoaccess.asp

Circuit efficiently drives inductive loads

Carlisle Dolland, Honeywell Engines and Systems, Torrance, CA

In the driver circuit in **Figure 1**, the system controller provides the $V_{COMMAND}$ signal. $V_{COMMAND}$ equals the desired load current multiplied by R_8. When the controller applies this voltage to R_1, the output of IC_1 goes high, applying voltage to the gates of Q_1 and Q_2. These transistors turn on, allowing load current to flow to ground through Q_1 and R_8. The current in the load ramps up, and a voltage proportional to the load current, sensed by R_8, feeds back to the inverting input of the comparator IC_1. When this voltage exceeds the voltage at the noninverting input, the output of IC_1 goes to ground. Q_1 and Q_2 then switch off. The load current now circulates around the loop comprising D_1 and L_1. During this time, the slope of the load current becomes negative because of the dissipation in D_1 and the load resistance. The duration of this phase of the circuit's operation is a function of the hysteresis (set by R_1, R_2, and R_4) and the decay of the voltage across C_2 (essentially a function of R_9). C_2 and R_9 determine the ripple current in the load. The circuit cannot use a power MOSFET for Q_2, because of the intrinsic drain-to-source diode. You must use a device without the intrinsic diode, such as a 3N71. (DI #2535)

Figure 1



Inductive loads are tricky to drive. This circuit provides efficient drive to relays and solenoids.

To Vote For This Design, Enter No. 363 at www.ednmag.com/infoaccess.asp

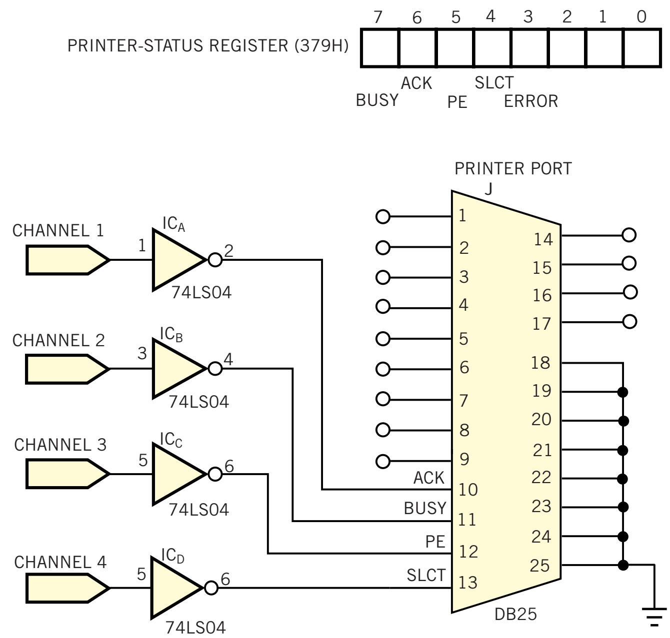
Use a PC to record four-channel waveforms

Dean Chen, Dycam Inc, Chatsworth, CA

THIS DESIGN IDEA is a sequel to a previous one, "Use a printer port to record digital waveforms," *EDN*, June 18, 1998, pg 136. Both ideas are similar: Use the PC's printer port to sample waveforms, and use the PC's memory to store data. The technique presented here expands the capability to four channels. The advantage is that you can see the relationships of the waveforms in the four channels. **Figure 1** depicts the sampling circuit. It uses printer-port pins ACK, BUSY, PE, and SLCT to record signals. The 74LS04 is a buffer between the sampled signals and the printer port. **Listing 1** is the sampling program, written in assembly language. Because there are four channels, every sample needs 4 bits (one nibble) to record. One byte can store two samples: odd and even samples. To accurately record signals, the sampling program needs exclusive access to the CPU.

Execution of the program must take

Figure 1



Use your PC's printer port to record four-channel waveforms.

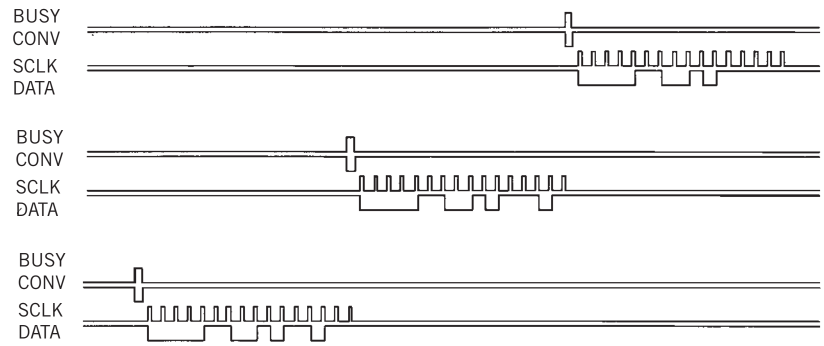
LISTING 1—FOUR-CHANNEL PC-PORT WAVEFORM-SAMPLING ROUTINE

```
;                       Printer Status Register
;
;                       7  6  5  4  3  2  1  0
;                        -------------------------
;                        |  |  |  |  |  |  |  |  |
;                        -------------------------
;                        |  |  |  |  |------------- ERROR
;                        |  |  |  |---------------- SLCT
;                        |  |  |------------------- PE
;                          |  |--------------------- ACK
;                          |------------------------ BUSY
;
;
;sta_reg               equ    0379h
;mask_reg              equ    021h
;
code                 segment para    public 'code'
                     assume  cs:code
                     org     100h
begin:                 jmp main
file_name    db        'samsig.dat',0
msg          db        'Sample Signal is saved in samsig.dat ! $'
;
main                 proc near
             lea di,buffer
             mov dx,sta_reg
pe_l:        in al,dx                        ; PE is as Trigger Signal
             and al,20h
             jz  pe_l                        ; When PE from 0 to 1,
pe_h:        in al,dx
             and al,20h                      ; Start Sampling.
             jz  pe_h
             mov dx,mask_reg
             in al,dx
             or al,01h                 ; Mask Time Interrupt
             out dx,al
             mov dx,sta_reg
             mov cx,0f000h                   ; Sample Size
sam_lp:
             in al,dx                        ; Even Sample
             shr al,1
             shr al,1
             shr al,1
shr al,1
xchg al,ah              ; high nibble save in ah
nop
nop
nop
in al,dx                ; Odd Sample
and al,0f0h             ; get low nibble
or al,ah                ; 2 nibble form 1 byte
xor al,077h                   ; correct bits polarity
stosb


            loop sam_lp
            mov dx,mask_reg
            in al,dx
            and al,0feh                     ; Allow Time Interrupt
            out dx,al
create_file:
            lea dx,file_name                ; memory is full,
            mov cx,0                        ; save data to 'samsig.dat' file
            mov ah,3ch
            int 21h
            mov bx,ax
write_data:
            mov cx,0f000h
            lea dx,buffer
            mov ah,40h
            int 21h
close:          mov ah,3eh
            int 21h
            lea dx,msg
            mov ah,9
            int 21h
            int 20h
main            endp
;
buffer:
code            ends
                end begin
```

place in pure MS-DOS mode, and not in a Windows multitasking environment. Second, it does not allow interrupts to occur during sampling. You must thus mask interrupts during the sampling procedure. Moreover, you need to equalize the odd and even sampling periods. Because the even sampling period is shorter then the odd one, the routine adds three nonoperation (NOP) instructions in the even sampling period. When the sampled data attains approximately 60 kbytes, the program restores the interrupt-mask register and generates a file named samsig.dat. **Listing 2** is a QBasic program for displaying the recorded waveforms. The program reads and then displays the samsig.dat file. **Figure 2** provides an example, a recording of the command and data signals from an Analog Devices AD7896 A/D converter. You can increase the sampling period by inserting some NOP instructions in the sampling routine. 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 #2536. (DI #2536)

Figure 2



Four channels of data from an AD7896 and the timing relationships thereof are visible.

TO VOTE FOR THIS DESIGN, ENTER NO. 364 AT WWW.EDNMAG.COM/INFOACCESS.ASP

LISTING 2—DISPLAY PROGRAM FOR SAMPLED WAVEFORMS

```
KEY 20, CHR$(0) + CHR$(72):    ON KEY(20) GOSUB UpLine
KEY 21, CHR$(0) + CHR$(80):    ON KEY(21) GOSUB DownLine
KEY 15, CHR$(0) + CHR$(73):    ON KEY(15) GOSUB UpPage
KEY 16, CHR$(0) + CHR$(81):    ON KEY(16) GOSUB DownPage
KEY 22, CHR$(0) + CHR$(75):    ON KEY(22) GOSUB Left
KEY 23, CHR$(0) + CHR$(77):    ON KEY(23) GOSUB Right
KEY 17, CHR$(0) + CHR$(1):     ON KEY(17) GOSUB Finish

SCREEN 12
DIM chr AS STRING * 1
DIM prv(3) AS INTEGER
DIM ptr AS LONG
OPEN "samsig.dat" FOR BINARY AS #1
 GET #1, 1, chr: lo% = ASC(chr) MOD 16
 FOR k% = 0 TO 3
  prv(k%) = lo% MOD 2: lo% = lo% \ 2
 NEXT k%
 d% = 12: dd% = 16: fl% = 0: ptr = 0
 KEY(17) ON

WHILE fl% = 0
 KEY(15) ON: KEY(16) ON: KEY(20) ON
 KEY(21) ON: KEY(22) ON: KEY(23) ON
 LOCATE 1, 36: PRINT ptr
 FOR i% = 0 TO 255: FOR j% = 0 TO 128: NEXT j%
 NEXT i%
 KEY(15) STOP: KEY(16) STOP: KEY(20) STOP
 KEY(21) STOP: KEY(22) STOP: KEY(23) STOP
 FOR i% = 0 TO 4
  y% = i% * 96 + 32
  FOR j% = 1 TO 320
   GET #1, ptr + j% + i% * 320, chr
   lo% = ASC(chr) MOD 16: hi% = ASC(chr) \ 16
   x% = 2 * j%
   FOR k% = 0 TO 3
    IF prv(k%) <> lo% MOD 2 THEN
     LINE (x%, y% + k% * dd%)-(x%, y% + k% * dd% + d%)
    ELSE
     IF (lo% MOD 2) THEN
      PSET (x%, y% + k% * dd%)
     ELSE PSET (x%, y% + k% * dd% + d%)
     END IF
    END IF
    prv(k%) = lo% MOD 2: lo% = lo% \ 2
   NEXT k%
   x% = x% + 1
   FOR k% = 0 TO 3
    IF prv(k%) <> hi% MOD 2 THEN
     LINE (x%, y% + k% * dd%)-(x%, y% + k% * dd% + d%)
    ELSE
     IF (hi% MOD 2) THEN
      PSET (x%, y% + k% * dd%)
     ELSE PSET (x%, y% + k% * dd% + d%)
     END IF
    END IF

    prv(k%) = hi% MOD 2: hi% = hi% \ 2
   NEXT k%
  NEXT j%
 NEXT i%
WEND
 KEY(15) OFF: KEY(16) OFF: KEY(20) OFF
 KEY(21) OFF: KEY(22) OFF: KEY(23) OFF
CLOSE #1
END

UpLine:
 IF ptr < 61120 THEN
  CLS 1: ptr = ptr + 320
 END IF
RETURN
Left:
 IF ptr < 61440 THEN
  CLS 1: ptr = ptr + 1
 END IF
RETURN
UpPage:
 IF ptr < 59840 THEN
  CLS 1: ptr = ptr + 1600
 END IF
RETURN
DownLine:
 IF ptr >= 320 THEN
  CLS 1: ptr = ptr - 320
 END IF
RETURN
Right:
 IF ptr >= 1 THEN
  CLS 1: ptr = ptr - 1
 END IF
RETURN
DownPage:
 IF ptr >= 1600 THEN
  CLS 1: ptr = ptr - 1600
 END IF
RETURN
Finish:
 fl% = 1
RETURN
```

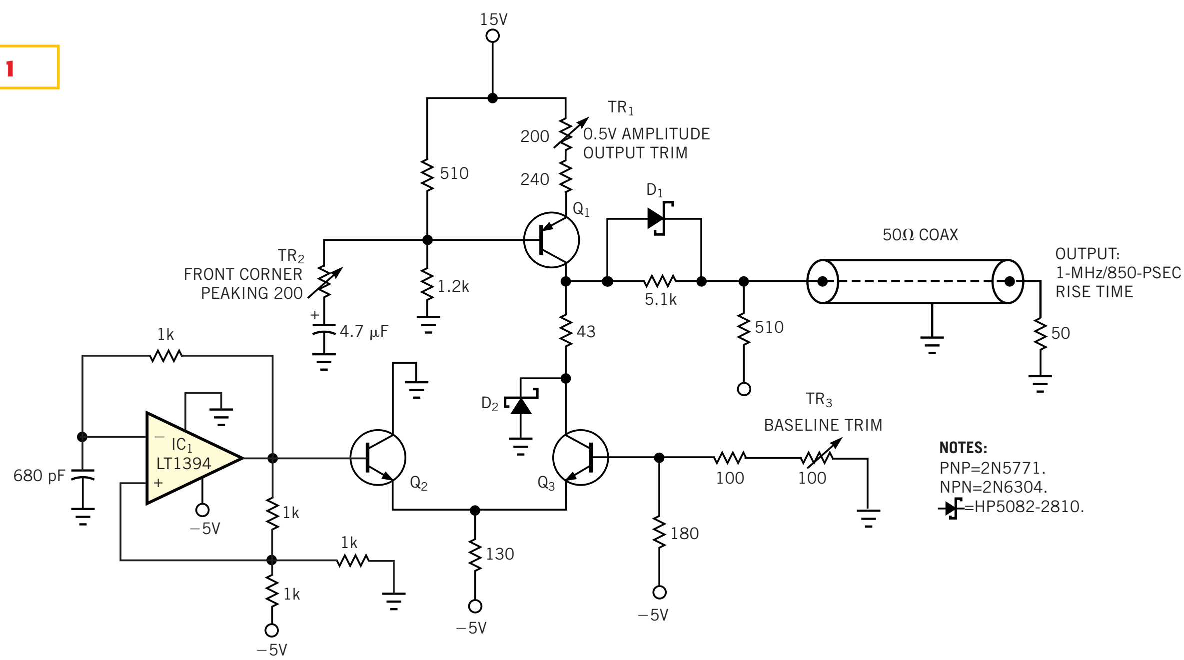
Pulse generator has low top-side aberrations

Jim Williams, Linear Technology Corp, Milpitas, CA

Impulse-response and rise-time testing often require a fast-rise-time source with a high degree of pulse purity. These parameters are difficult to achieve simultaneously, particularly at subnanosecond speeds. The circuit in **Figure 1**, derived from oscilloscope calibrators (**Reference 1**), meets the speed and purity criteria. It delivers an 850-psec output with less than 1% pulse-top aberrations. Comparator IC_1 delivers a 1-MHz square wave to current-mode switch Q_2-Q_3. Note that IC_1 obtains power between ground and −5V to meet the transistors' biasing requirements. Q_1 provides drive to Q_2 and Q_3. When IC_1 biases Q_2, Q_3 turns off. Q_3's collector rises rapidly to a potential determined by Q_1's collector current, D_1, and the output resistors combined with the 50Ω termination resistor. When IC_1 goes low, Q_2 turns off, Q_3 turns on, and the output settles to 0V. D_2 prevents Q_3 from saturating.

The circuit's output transition is extremely fast and singularly clean. **Figure 2**, viewed on a 1-GHz real-time-bandwidth oscilloscope, shows 850-psec rise

Figure 1



Need a fast, clean pulse? This simple circuit provides 500-mV, 850-nsec pulses with a high degree of purity.

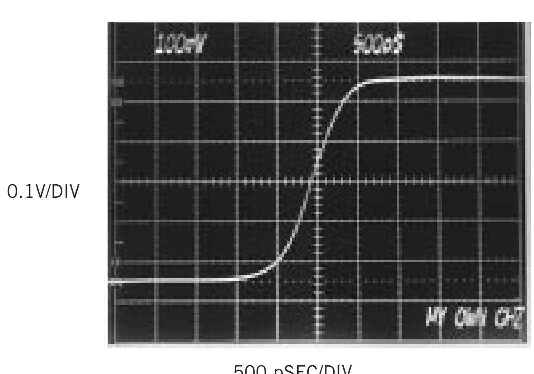


Figure 2

Viewed with 1-GHz bandwidth, pulses are free of discontinuities and anomalies.

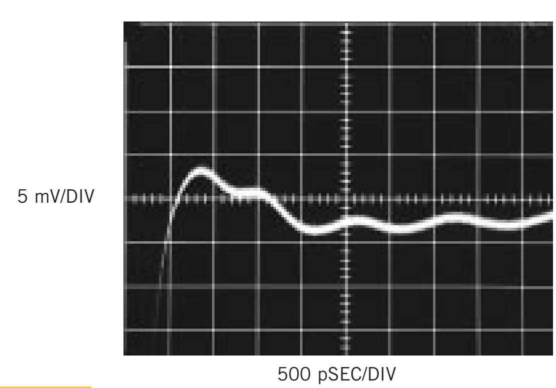


Figure 3

Pulse tops exhibit less than 64 mV, or less than ±1%, aberrations.

time with exceptionally pure pretransition and post-transition characteristics. **Figure 3** details the pulse-top settling. The photo shows the pulse-top region immediately following the positive 500-mV transition. Settling occurs within 400 psec of the edge's completion with all activity within ±4 mV. The 1-mV, 1-GHz ringing undoubtedly stems from breadboard-construction limitations; you can probably eliminate it by using stripline-layout techniques. The level of performance of this circuit requires some trimming. The oscilloscope you use should have at least 1-GHz bandwidth. You adjust trimmers TR_2 and TR_3 for the best pulse presentation. TR_1 sets the output amplitude at 500 mV across the 50Ω termination. The trims are somewhat interactive, although not unduly so, and converge quickly to give the results described. (DI #2530)

REFERENCE

1. 485 Oscilloscope Service and Instruction Manual, "Calibrator," pg 3 to 15, Tektronix Inc, 1973.

TO VOTE FOR THIS DESIGN,
ENTER NO. 365 AT
WWW.EDNMAG.COM/INFOACCESS.ASP

Circuit provides ADSL frequency reference

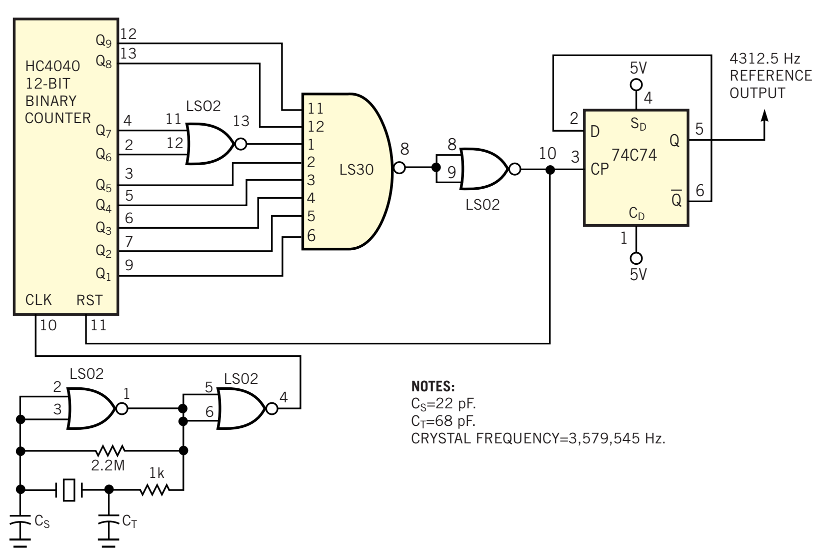
Bert Erickson, Fayetteville, NY

THE DISCRETE-MULTITONE (DMT) frequencies that asymmetrical-digital-subscriber lines (ADSL) use are integral multiples of a common frequency, and the symbol period is the inverse of this frequency. Integration over the symbol period allows the sine and cosine orthogonal waveform products to vanish for all multiples of the common frequency except for those having the same frequency. As the ADSL standards (TI.413) specify, the 256 channels are separated by 69/16 kHz. You can generate the midchannel frequencies with a PLL, but the reference frequency differs from that of crystals for computers and clocks. However, by using the circuit in **Figure 1**, you can generate the frequency by using a 3.58-MHz crystal to control the horizontal scanning rate in television sets. A typical 3.58-MHz crystal has a tolerance of ±50 ppm and a load capacitance of 18 pF. This tolerance provides a frequency of 3.579366 to 3.579724 MHz. If you multiply this common DMT frequency by 830, the result is 830×69/16 kHz, or 3.579375 MHz, which is 9 Hz above the crystal's lower tolerance limit. Assuming that you can select the C_S and C_T capacitors at either side of the crystal to tune the frequency near the lower tolerance limit, you can also select them for the desired frequency.

In other words, reduce the oscillator frequency with bistable flip-flops and combine the outputs in a NAND gate to divide by 830. For the 3.58-MHz crystal, design values for C_S and C_T were 23.6 and 75.7 pF, respectively. We chose 22 pF for C_S and 68 pF for C_T. A trimmer capacitor in parallel with C_T reduces the frequency. When C_T increased from 22 to 90 pF, the frequency decreased by 448 Hz and handily bridged the 3.579545- and 3.579375-MHz frequencies. Tests showed that the lower frequency was more than 100 Hz below 3.579357 MHz, but the exact number depends on the calibration of the counter. Because 830 is a 10-bit binary number, the circuit divides by 415 first to permit combining with an eight-input NAND gate. A strobe applied to a flip-flop then creates a square wave for the reference-frequency output. (DI #2531)

TO VOTE FOR THIS DESIGN,
ENTER NO. 366 AT
WWW.EDNMAG.COM/INFOACCESS.ASP

Figure 1



Using a common TV crystal, you can generate the reference frequency for ADSL systems.

ActiveX control brings bit manipulation to Windows

Steve Hageman, Agilent Technologies, Santa Rosa, Ca

Nothing compares with the C language for working with bits. C provides a rich set of signed and unsigned number formats, along with many intrinsic bit-manipulation operators. However, most of the popular rapid-application-development Windows languages lack C's ability to easily work with bits. Visual Basic is such a language. Although it's hard to find a faster language to develop a small to midsized application in Windows, Visual Basic starts to show its weakness when it comes time to talk to hardware. Hardware programming is usually bit-oriented. That is, it's necessary to turn bits on and off or shift out serial streams to get the hardware to operate correctly. The ActiveX control serves just these types of bit-manipulation needs (**Figure 1**). The control includes functions for changing binary strings to numbers, a hex-output function, the ability to set and clear bits in a word, and the ever-needed shift-left and -right functions. As an example, many of the three-wire serial devices need to have a setup word shifted to them. Suppose you need to shift the setup word 0111 1101 first to an A/D converter to initiate a conversion on some channel. You can use the functions in the ActiveX control to easily effect the shift operation, as follows:

```
Setup_word = Bits ("01111101")
`Returns 125
For i = 0 to 7
    Val = ShiftRight_8(setup_word,0)
    `write val to the A/D here
next i
```

In the above example, val has the values 1, 0, 1, 1, 1, 1, 1, 0 during each iteration of the loop. The routine can then clock these bits to the A/D converter as required by the hardware. If the operation requires MSB first, you can use the ShiftLeft function. The SetBit and ClearBit functions are useful when using a port as clock and data lines, because you can set individual bits as needed instead of doing entire port writes. Any modern programming language that can use ActiveX controls, such as Agilent VEE, Visual Basic, Delphi, and others, can use the functions given here. You can download the ActiveX control 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 #2534. The routine includes all the functions listed in **Figure 1**, plus a few more, with application examples. (DI #2534)

To Vote For This Design, Enter No. 367 at www.ednmag.com/infoaccess.asp

Figure 1

```
Function GetBit(ByVal x As Long, ByVal n As Integer) As Integer
Returns the value of bit n in input value x. Returns 1 or 0 if bit
is set or not. x = 1 to 16 bit, n = 0 = LSB.
Example: GetBit(16,5) returns 1.

Function Bits(ByVal inval As String) As Long
Given a  representation of a binary string, returns the
value. inval may be any length from 1 to 16 bits
Example: Bits("101") returns 5.

Function BitsStr(ByVal inval As Long, ByVal sizeof As Integer) As String
Given a number, returns with a representation of a binary string.
sizeof is the width of the return field (1 to 16 bits).
Example: BitsStr(82,8) returns "01010010"

Function HexStr(ByVal inval As Long, ByVal sizeof As Integer)
Given a number, returns with a representation of a hex string.
sizeof is the width of the return field (1 to 16 bits).
Example: HexStr(179,8) returns "B3"

Function ClearBit(ByVal x As Long, ByVal n As Integer) As Long
Clears bit position n in input x. Returns new x value.
x may be 1 to 16 bits, n = 0 = LSB
Example: ClearBit(16,4) returns 0.

Function SetBit(ByVal x As Long, ByVal n As Integer) As Long
Sets bit n in input value x. Returns new  x. x may be any
width 1 to 16 bits, n = 0 = LSB.
Example: SetBit(0,4) returns 16

Function ShiftRight_8(ByRef x As Integer, ByVal y As Integer) As Integer
Shifts the 8 bit value x right by 1 place. Bit shifted in is y.
Returns bit shifted out.
Example: ShiftRight_8(129,1) Returns 1 and the new value for x (was 129)
is 192.

Function ShiftRight_16(ByRef x As Long, ByVal y As Integer) As Integer
Shifts the 16 bit value x right by 1 place. Bit shifted in is y.
Returns bit shifted out.
Example: ShiftRight_16(1,1) Returns 1 and the new value for x (was 1)
is 32768.

Function ShiftLeft_8(ByRef x As Integer, ByVal y As Integer) As Integer
Shifts the 8 bit value x left by 1 place. Bit shifted in is y.
Returns bit shifted out.
Example: ShiftLeft_8(1,0) returns 0 and the new value for x (was 1)
is 2.

Function ShiftLeft_16(ByRef x As Long, ByVal y As Integer) As Integer
Shifts the 16 bit value x left by 1 place. Bit shifted in is y.
Returns bit shifted out.
Example: ShiftLeft_16(32768,1) returns 1 and the new value for x
(was 32768) is 1

Function RotateRight_8(ByVal x As Integer) As Integer
Rotates the 8 bit value x right by 1 place. Returns new value.
Example: RotateRight_8(1) returns 128.

Function RotateRight_16(ByVal x As Long) As Long
Rotates the 16 bit value x right by 1 place. Returns new value.
Example: RotateRight_16(1) returns 32768.

Function RotateLeft_8(ByVal x As Integer) As Integer
Rotates the 8 bit value x left by 1 place. Returns new value.
Example: RotateLeft_8(64) returns 128

Function RotateLeft_16(ByVal x As Long) As Long
Rotates the 16 bit value x left by 1 place. Returns new value.
Example: RotateLeft_16(32769) returns 3.

End.....
```

An ActiveX control offers many handy functions for bit manipulation.

Circuit breaker handles voltages to 32V

Greg Sutterlin and Craig Gestler, Maxim Integrated Products, Sunnyvale, CA

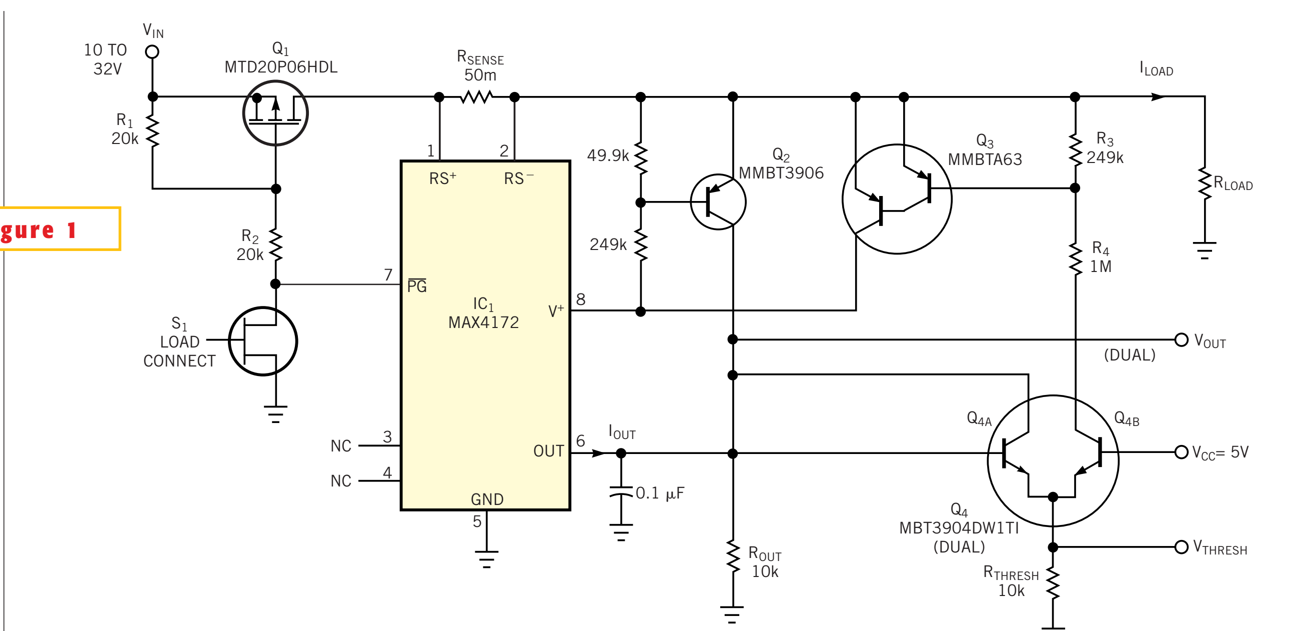
THE SIMPLICITY of low-side current monitoring can mask the advantages of a high-side approach. You can monitor load currents in a power supply, a motor driver, or another power circuit on either the high or the low side (ground). However, don't let the ease of low-side monitoring cause you to overlook its dangers or the advantages of a high-side approach. Various fault conditions can bypass the low-side monitor, thereby subjecting the load to dangerous and undetected stresses. On the other hand, a high-side monitor connected directly to the power source can detect any downstream failure and trigger the appropriate corrective action. Traditionally, such monitors required a precision op amp, a boost power supply to accommodate the op amp's limited common-mode range, and a handful of precision resistors. Now, the MAX4172 IC can sense high-side currents in the presence of common-mode voltages as high as 32V (**Figure 1**). IC_1 provides a ground-referenced current-source output proportional to the high-side current of interest. This output current, equal to the voltage across an external sense resistor divided by 100, produces a voltage output across a load resistor.

IC_1 and a few external parts form a low-cost circuit breaker. R_{SENSE} senses load currents, and Q_1 controls the currents. The design accepts inputs of 10 to 32V; you can easily modify it to operate from voltages as low as 6.5V. The initial application of V_{IN} and V_{CC} places the breaker in its trip state. Pressing S_1 resets the breaker and connects power to the load, thereby activating Q_1, Q_3, and Q_{4B}. Q_3 powers IC_1, and Q_{4B} establishes the overcurrent threshold, $V_{THRESH}=V_{CC}-V_{BE(4B)}$. Because V_{CC} (2.7 to 5.5V typical) equals 5V and the base-emitter voltage of Q_{4B} is approximately 0.7V, V_{THRESH} is typically 4.4V. The circuit trips at a nominal load current of 1A. The values for R_{SENSE}, R_{THRESH}, and R_{OUT} are functions of the system's accuracy and power-dissipation requirements. First, select $R_{SENSE}=50\ \text{m}\Omega$ and $R_{THRESH}=10\ \text{k}\Omega$. Then, calculate $R_{OUT}=V_{CC}/I_{LOAD}R_{SENSE}G_m$, where I_{LOAD} is the trip point (1A) and G_m (IC_1's typical transconductance) equals 0.01A/v. Thus, $R_{OUT}=10\ \text{k}\Omega$.

Applying power to Q_3 and Q_{4B} causes Q_{4B} to conduct, which establishes V_{THRESH} and activates Q_3 to power IC_1. A fraction of the load current through R_{SENSE} mirrors to the IC_1 output and appears as a voltage, V_{OUT}, across R_{OUT}. Q_{4B} turns off when V_{OUT} increases above $(V_{THRESH}+V_{BE(4BA)})$, turning off Q_3 and causing a drop in V^+ (IC_1, pin 8). When V^+ reaches 2.67V (typical), $\overline{PG}$ goes high, thereby tripping the breaker by turning off Q_1. Q_2 adds feedback to ensure a clean turn-off at the trip level. Current draw in the tripped state is minuscule and equals the V_{CC} load current, 0.5 mA typical. Press S_1 to reset the breaker. The design is intended for low-cost applications in which the absolute accuracy of the trip current is not critical. The accuracy, which depends on variations in V_{CC} and the base-emitter voltages of Q_{4A} and Q_{4B} and on the error current through R_4, is approximately ±15% at a trip current of 1A. (DI #2532)

TO VOTE FOR THIS DESIGN, ENTER NO. 368 AT WWW.EDNMAG.COM/INFOACCESS.ASP

Figure 1



A current-sense amplifier and a few transistors form a low-cost circuit breaker.