

design ideas

Edited by Bill Travis and Anne Watson Swager

Current source has high compliance

Frank Vitaljic, Bellingham, WA

CURRENT SOURCES ARE useful in many areas of electronics, such as voltage and current division, current transmitters, excitation of thermistors, RTDs, bridges, potentiometers, and circuit biasing. Current sources are either fixed- or adjustable-current types. For the dual-op-amp current source in **Figure 1**, the load current, I_L , is adjustable by varying V_{IN} (0 to $\pm 1.2V$), according to the following equation:

$$I_L = \left[\frac{V_{IN}}{R_{SENSE}} \right] + \left[\frac{V_{OS1} + V_{OS2}}{R_{SENSE}} \right].$$

IDEAL OFFSET ERROR

The maximum variation in load current is ± 6 mA for a 200Ω current-sensing resistor, R_{SENSE} . This variation yields a maximum load voltage of $\pm 12V$ (3V less than the supply voltages). The maximum current-offset error is 0.01 mA for V_{OS1} and V_{OS2} of 1 mV. This amount of offset is normal for lower grade op amps. Note that you connect

one end of the load to virtual ground (VG). Amplifier IC_1 acts as a voltage-error driver, and the transconductance amplifier, IC_2 , senses the load current and converts it to V_A . The control equations are:

$$(V_A - V_{IN})G_A = Z_L I_L \quad (1)$$

$$\left(1 + \frac{1}{G_B} \right) V_A = -R_{SENSE} I_L, \quad (2)$$

where G equals open-loop gain; and

$$\frac{G_0}{1 + (G_0/W)s}, \quad (3)$$

where G_0 = dc gain, W = unity-gain bandwidth, and $s = j\omega$ complex frequency.

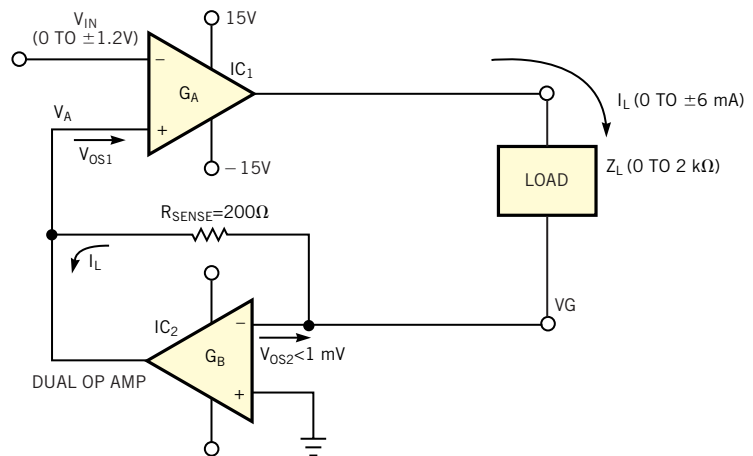
Solving **equations 1** and **2** for I_L and substituting G in **Equation 3** for $G_A G_B$, the load current becomes:

$$I_L = \frac{-WV_{IN}}{R_{SENSE}W + Z_L s} \approx \frac{-V_{IN}}{R_{SENSE}} \text{ for } R_{SENSE}W \gg Z_L W,$$

which demonstrates the load current's insensitivity to the load, Z_L . (DI #2579)

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/infoaccess.asp, enter No. 435 in the "Circle Number" field, and hit the Search bar. Put a tick in the "Select" box and hit "Submit."

Figure 1



A ± 6 -mA current source has a $\pm 12V$ compliance range.

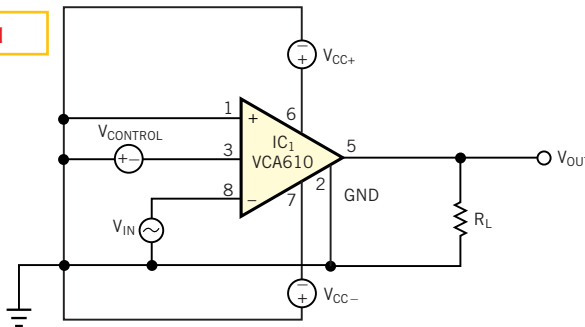
Current source has high compliance	147
Modification improves VCA's Spice simulation	148
BIOS INT1Ch measures frequency	152
Circuit variations produce negative voltages	158
μC takes control of comparator	158

Modification improves VCA's Spice simulation

Alex Rysin, RSVI Acuity CiMatrix, Canton, MA

THE BURR-BROWN VCA610 is a good wideband voltage-controlled amplifier (VCA) that you can use successfully for your application. The Spice model from Burr-Brown simulates normally when the VCA uses dual power supplies (Figure 1). However, when you attempt to use the Spice model in a single-supply application, which is equivalent to moving the GND symbol to the negative V_{CC} pin of IC₁ (Figure 2), you get an error message indicating that the circuit fails to converge. To find the source of the problem, restore the original schematic used for the Spice-model creation of the VCA610 and carefully analyze it. The model has several problems that prevent you from using it in single-supply simulations. The main problem arises from the faulty usage of

Figure 1



With dual supplies for the VCA610, the Spice simulation converges.

“0” nodes in the model description. Spice considers subcircuit nodes as “local” and assigns them new names for simulations. This trait of Spice holds true for all subcircuit nodes except for ground nodes named “0,” which are global for the entire simulation file. You can use the ground node “0” in subcircuit descriptions, but

you must be careful that the node does not create unwanted dc references in a circuit that is intended to be floating. To avoid problems, do not use “0” nodes in the model description but rather use different names, such as GND or AGND. Otherwise, you must carefully separate them the “0” nodes from dc connections that may affect the internal voltages and currents of the subcircuit. You

can use control sources to separate the nodes. Listing 1 is Burr-Brown’s model of the VCA610.

In Listing 1, all entries with problematic usage of the global node “0” appear in boldface. The line “E1 11 7 POLY (1)(3,0)0.45 -0.11911” is a polynomial voltage-controlled voltage source con-

LISTING 1—VCA610 SIMULATION FOR DUAL SUPPLIES

```
* CONNECTIONS:
* GROUND
* NON-INVERTING INPUT
* OUTPUT
* POSITIVE SUPPLY VOLTAGE
* NEGATIVE SUPPLY VOLTAGE
* INVERTING INPUT
* GAIN STAGE 2
* GAIN STAGE 1
* INPUT STAGE
* OUTPUT STAGE
* ENDS
```


BIOS INT1Ch measures frequency

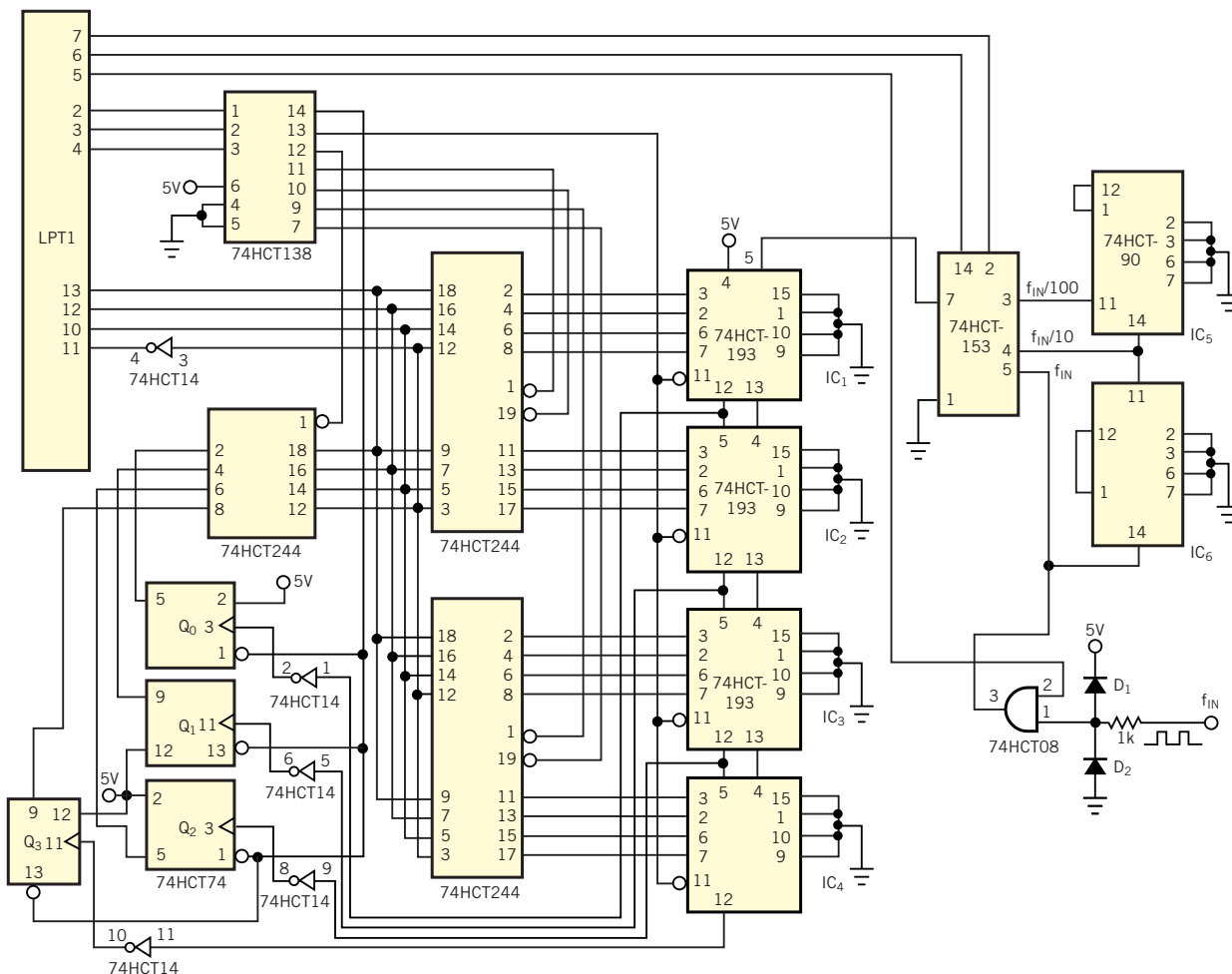
K Suresh, Indira Ghandi Centre for Atomic Research, Tamil Nadu, India

THE DESIGN IN **Figure 1** shows how you can exploit BIOS INT1Ch of a PC, which occurs 18.2 times/sec, along with a few inexpensive components and LPT1, to measure frequencies from 1 Hz to into the megahertz range. The design requires no expensive general-purpose or tailor-made data-acquisition card/logger. It also overcomes the disad-

vantage of continued interrupt of the PC in the interrupt-based frequency-measurement method, in which the speed of the PC decides the maximum frequency. The design exploits a few resources of the PC—for example, BIOS INT1Ch (without affecting its normal interrupt-service routine) and the Data and Status ports of LPT1—to measure frequency independ-

ently of the speed of the PC. The input pulses, with frequency f_{IN} , drive a 16-bit up/down counter, comprising four 4-bit counters (IC_1 to IC_4 , 74HCT193) in cascade. The drive path includes a 74HCT08 AND gate and a 74HCT153 4-to-1 digital multiplexer. The multiplexer feeds any one of the frequency inputs, f_{IN} , $f_{IN}/10$, or $f_{IN}/100$, from the decade counters (IC_5 ,

Figure 1



By using BIOS INT1Ch, you can measure frequency, using few resources of the host PC.

and IC₆, 74HCT90) to the counter.

The output of the counters and the flip-flops' Q outputs connect to the Status port (at 0x379h) of the LPT1 port through the 74HCT244 buffers. A 74HCT138 3-to-8 encoder and the digital multiplexer, controlled from the Data port (at 0x378h), provide various control signals, as dictated by the software in Listing 1. This simple Turbo C program controls the frequency measurement. The BIOS INT1Ch generates two timing windows—SWINDOW of approximately 100 msec, and MWINDOW of 1 sec—by incrementing the SINTR and MINTR variables (set to zero initially), respectively.

At the end of every SWINDOW, the

software determines the range of the input frequency f_{IN} —hertz, kilohertz, or megahertz. The routine then selects a range, so that the number of input pulses per second nearly equals, but stays within, the counter's capacity. Once the routine selects the range, the program executes the MWINDOW routine to measure the frequency. At the start, disabling the AND gate inhibits the input pulses, and the routine clears the counters and flip-flops. The program sets the digital multiplexer to apply the input frequency, f_{IN} , to the counters. The next INT1Ch enables the AND gate and increments SINTR and MINTR by executing the SWINDOW routine. When each counter's output crosses from 0x0Fh to 0x00h dur-

ing count-up, its Carry output sets the D flip-flop (74HCT74) to logic 1. When SINTR reaches 2 (SWINDOW=109 msec), the routine inhibits the input pulses and reads the SNIBBLE at the flip-flop's outputs (Q₃Q₂Q₁Q₀). If SNIBBLE equals 0x07h or greater, the program sets the digital multiplexer to apply $f_{IN}/10$ to the counters. The routine then generates SWINDOW. The routine then again reads SNIBBLE to determine whether to apply $f_{IN}/100$ (if SNIBBLE \geq 0x07h) or $f_{IN}/10$ (if SNIBBLE<0x07h) to the counters. The loop continues until SNIBBLE, read at the end of SWINDOW, becomes less than 0x07h.

At any stage, if SNIBBLE is less than 0x07h, MWINDOW generates a timing

LISTING 1—TURBO C ROUTINE FOR FREQUENCY MEASUREMENT

```
#include<stdio.h>
#include<dos.h>
#include<conio.h>
#include<process.h>
#include<math.h>
#define INT1C 0x1C /*INT 1Ch*/

int MINTR=0,SINTR=0,RANGE=0,MF=1; /* Sample and Measure windows Variable*/
float FREQ=0.0,CF=18.2/18; /* Measured frequency& correction factor*/
int CUWORD=0,MCR,DP,SP,a,b=0x10;
float y1 = 0.0,y2=0.0;

void interrupt (*oldvect)(); /* INT1Ch pointer*/
void interrupt TIMER(); /*Routine for INT1Ch*/
unsigned int FREQREAD(); /*Routine declaration for reading the counter output*/
int SWINDOW(); /*Routine declaration for generation of 100msec*/
void MWINDOW(void); /*Routine declaration for generation of 1 sec*/

void interrupt TIMER() /*Our ISR for INT1Ch*/
{
    disable();
    outp(DP,(b+0x08)); /*Enable AND gate and feed input pulses*/
    SINTR++; /*Increment Continues till SINTR becomes 2*/
    MINTR++; /*Increment continues till MINTR becomes 18*/
    oldvect();
    enable();
}

int SWINDOW() /*Routine for generation of 100msec*/
{
    oldvect=getvect(0x1C); /*store vector address of INT1Ch*/
    outp(DP,(b+0x00)); /*Disable AND and inhibit pulses to counter*/
    outp(DP,b); /*set the digital multiplexer*/
    outp(DP,(b+0x02)); /* Load the counters with 0x00h*/
    outp(DP,(b+0x01)); /*Clear all Flip Flops*/

    while(SINTR<=2) /*Wait for generation of app 100msec*/
    {
        setvect(0x1C,TIMER);
    }

    outp(DP,(b+0x00)); /*Disable AND*/
}

RANGE++;
setvect(0x1C,oldvect);
SINTR=0;
MINTR=0;
outp(DP,b+0x03);
a=((inp(SP)>>4) & 0x0F); /*Read the SNIBBLE*/
return a;
}

void MWINDOW() /*Routine for generation of 1 sec*/
{
    oldvect=getvect(0x1C); /*store vector address of INT1Ch*/
    outp(DP,(b+0x00)); /*Disable AND and inhibit pulses to counter*/
    outp(DP,(b+0x02)); /* Clear the counters */
    outp(DP,(b+0x01)); /*Clear all Flip Flops*/
    outp(DP,b); /*set the digital multiplexer*/

    while(MINTR<=18) /*Wait for generation of 0.989sec*/
    {
        setvect(0x1C,TIMER);
    }

    outp(DP,(b+0x00)); /*Disable AND*/
    setvect(0x1C,oldvect);
    FREQREAD();
    SINTR=0;
    MINTR=0;
    return 0;
}

unsigned int FREQREAD()
{
    unsigned char nib0,nib1,nib2,nib3,byte1,byte2;
    int temp,i;
    y1=0.0;
    y2=0.0;
    byte1=0;
    byte2=0;
    outp(DP,b+0x03);
    a=((inp(SP)>>4) & 0x07); /*Read the SNIBBLE*/

    switch(a)
    {
        case 0: /*Counter 1 alone counted, read nibble 0*/
            outp(DP,(b+0x04));
            nib0=inp(SP); /*Read nibble 0*/
            nib0=nib0>>4;
            byte1=nib0|0x00;
            byte2=0;
    }
}

```

(continued on pg 156)

window of 0.989 sec by incrementing MINTR with each INT1Ch from 0 to 18. During this incrementation, the routine counts the frequency pulses from the digital multiplexer. The program then corrects the timing window to 1 sec by applying a correction factor (CF=18.2/18). At the end of MWINDOW, the program inhibits the pulses and again reads SNIBBLE. SNIBBLE indicates the progress of the counters during the MWINDOW interval. This operation relieves the PC from reading all the counters; instead, it reads only the counters that actually did the counting. The routine thereby reduces the number of read operations and

subsequent manipulations. The FREQREAD routine reads the counters' output, a nibble at a time, starting with Counter 1 (IC₁, nib0), depending on the last-read SNIBBLE, to Counter 4 (IC₄, nib3). The program manipulates the nibbles and displays the frequency in hertz. At the end of a measurement cycle, the routine sets SINTR and MINTR to zero and clears the counters and flip-flops to make them ready for another measurement cycle. You can use the design for frequencies of 1 Hz to 7 MHz. However, you can extend the range to 25 MHz or greater by increasing the counter capacity to 20 bits or more (for example, by us-

ing two 74HCT4040s in cascade) and by using appropriate FAST logic devices. 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 #2575.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/infoaccess.asp, enter No. 437 in the "Circle Number" field, and hit the Search bar. Put a tick in the "Select" box and hit "Submit."

LISTING 1—TURBO C ROUTINE FOR FREQUENCY MEASUREMENT (Continued)

```

break;

case 1: /*Counter 1 & 2 involved in counting*/
  outp(DP, (b+0x04));
  nib0=inp(SP)>>4; /*Read nibble 0*/
  outp(DP, (b+0x05));
  nib1=inp(SP)&0xF0; /*Read nibble 1*/
  byte1=nib1|nib0;
  byte2=0;
  break;

case 3: /*counters 1,2 & 3 involved in counting*/
  outp(DP, (b+0x04));
  nib0=inp(SP)>>4; /*Read nibble 0*/
  outp(DP, (b+0x05));
  nib1=inp(SP)&0xF0; /*Read nibble 1*/
  outp(DP, (b+0x06));
  nib2=inp(SP)>>4; /*Read nibble 2*/
  byte1=nib1|nib0;
  byte2=nib2&0x0F;
  break;

case 7: /*all counters involved in counting*/
  outp(DP, (b+0x04));
  nib0=inp(SP)>>4; /*Read nibble 0*/
  outp(DP, (b+0x05));
  nib1=inp(SP)&0xF0; /*Read nibble 1*/
  outp(DP, (b+0x06));
  nib2=inp(SP)>>4; /*Read nibble 2*/
  outp(DP, (b+0x07));
  nib3=inp(SP)&0xF0; /*Read nibble 3*/
  byte1 = nib1|nib0;
  byte2 = nib3|nib2;
  break;

default:
  printf("\n Error ! ...Check the flip flops,....exiting");
  exit(1);
}
/* Convert binary words byte1,byte2 to decimal*/
for(i=8; i<16;i++)
{
  temp=byte2;
  byte2=byte2 &(0x01);
  y1=y1+byte2*pow(2,i);
  byte2=temp;
  byte2=byte2>>1;
}
for(i=0; i<8;i++)
{
  temp=byte1;
  byte1=byte1 &(0x01);
  y2=y2+byte1*pow(2,i);
  byte1=temp;
  byte1=byte1>>1;
}

}
CUWORD=(y1+y2);
FREQ= CUWORD*CF*MF;
gotoxy(10,10);
printf("\nFREQ in Hz=%.1f", FREQ); /*Display freq in Hz*/
return 0;
}

int main() /*Main Program starts here*/
{
  clrscr();
  printf("\n\tBeyond its time of day, INT1Ch measures frequency");
  printf("\n\t\t\t by");
  printf("\n\tK.Suresh,MSD,IGCAR,Kalpakkam,Tamil Nadu,India");
  DP = peek(0x40,8); /*check up for availability of printer port */
  if (DP==0)
  {
    printf("\n\n LPT NOT AVAILABLE! EXITING");
    exit(1);
  }
  printf("\n\n LPT1 address =0x%X",DP); /* Address of DATAPORT*/
  SP = DP+1; /*Address of STATUS PORT*/

  while(!kbhit())
  {
    SWINDOW();
    MF=1;
    if (a >=0x07 && RANGE==1)
    {
      b = 0x20; /*Apply fm/10 to the counters*/
      MF=10;
      SWINDOW();
    }
    if(a>= 7 && RANGE==2)
    {
      b=0x30; /*Apply fm/100 to the counters*/
      MF=100;
      SWINDOW();
    }
    if (a>=0x07 && RANGE>2)
    {
      printf("\n over range!... exiting");
      setvect(0x1C,oldvect);
      exit(1);
    }
  }
}
MWINDOW();
RANGE=0;
} /*Freq measurement continues till a key is pressed*/
setvect(0x1C, oldvect);
return 0;
} /*End of Main Program*/

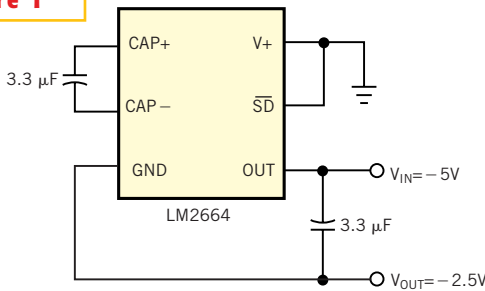
```

Circuit variations produce negative voltages

Clinton Jensen, National Semiconductor Corp, Santa Clara, CA

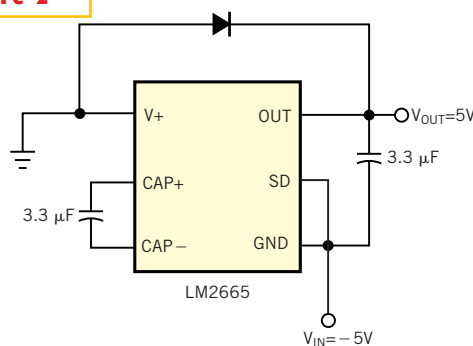
BASIC SWITCHED-capacitor converters generally provide one simple conversion. They commonly double, usually invert, and sometimes halve a positive input voltage. Because they are not regulated converters and do not have stability problems, you can easily configure them to also do some negative conversions. However, given that they are unregulated, it is a good idea to use a regulated input voltage to obtain a predictable output. All these converters suffer from some voltage change on the output as a result of loading, but this voltage drop is often acceptable. The potential space and cost savings are often worth the trade-off if the circuit fits the application needs. **Figures 1** and **2** present two application circuits. The first is useful in systems that require multiple negative voltages. You can set up a basic switched-capacitor converter capable of inversion to halve a negative voltage. **Figure 1** shows the schematic of the circuit, using an LM2664 as an example. The circuit is capable of the full rated load (40 mA, in this case). The operating-voltage range is -3.6 to -11 V. The circuit can be useful for creating multiple bias voltages for op amps, power amplifiers, or displays.

Figure 1



A switched-capacitor converter handily splits a negative voltage in half.

Figure 2



You can use a switched-capacitor converter to derive a positive voltage from a negative one.

The second application is useful when a negative voltage is present and you need an equal but inverted (positive) voltage. In **Figure 2**, an LM2665 produces a 5V output from a -5 V input. The topology shown works with any switched-capacitor circuit designed for the doubling function. This circuit can also supply the full rated load current and works with an input-voltage range of -2.5 to -5.5 V. The output resistance of the circuit equals that of the basic positive doubler. The circuit can be useful for op-amp biasing when only a negative voltage is present. Another benefit is that, if you have a loosely regulated negative supply for an op amp, the positive supply tracks the negative supply and keeps the output dc-biased near to or at ground. Another possible application is in systems using two-cell lithium-ion batteries or four alkaline batteries and needing ± 5 V supplies. (DI #2580)

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/infoaccess.asp, enter No. 438 in the "Circle Number" field, and hit the Search bar. Put a tick in the "Select" box and hit "Submit."

μ C takes control of comparator

Abel Raynus, Armatron International, Melrose, MA

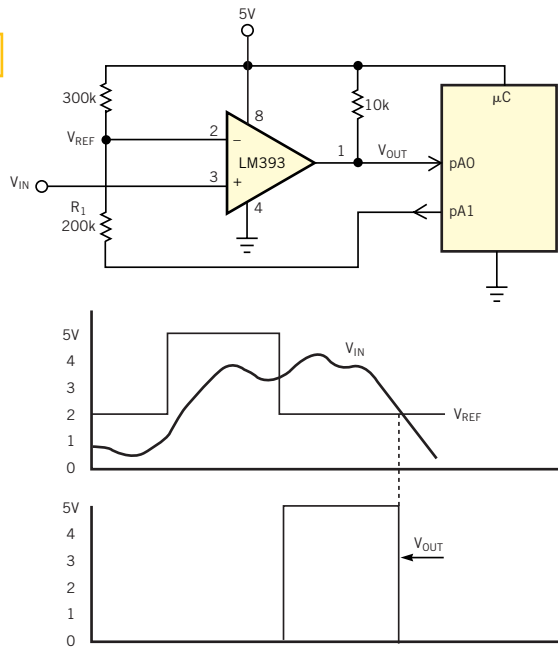
IN MANY μ C applications, a signal conditioner modifies an analog input signal from a sensor and passes it to the μ C for processing. The last stage of the signal conditioner is always a device that

transforms the signal level to a value acceptable for the μ C's input. Usually, it is a comparator. Often, this signal channel is open or closed for some programmable time period. For this purpose, you can

use a variety of analog switches. **Figure 1** shows an alternative: a cost-effective technique that needs no additional switches. The circuit exploits the fact that the μ C's output pin can work as a pro-

programmable SPDT switch, connecting the lower end of R_1 either to 5V or to ground. When you program the μC 's output pin (pA1 in **Figure 1**) to a low level, R_1 connects to ground, and the predetermined reference voltage connects to Pin 2 of the comparator. In **Figure 2**, $V_{\text{REF}} = 2\text{V}$. The channel is open, and for input signals greater than V_{REF} , the comparator provides 5V to the μC 's Pin pA0. When the output pin pA1 is high, the reference voltage becomes 5V, and the comparator's output switches to 0V. The channel is then closed. (DI #2582)

Figure 1



Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/infoaccess.asp, enter No. 439 in the "Circle Number" field, and hit the Search bar. Put a tick in the "Select" box and hit "Submit."

You can use a μC to control a comparator, thus avoiding the need for analog switches.