design ideas

Edited by Bill Travis and Anne Watson Swager

Low-cost bias circuits serve HF and VHF bands

Richard M Kurzrok, RMK Consultants, Queens Village, NY

Bias circuits are passive networks that you use to apply dc excitation to various active circuits. The monitor tee, which is also known as the bias tee, has been commercially available for more than 40 years at microwave frequencies. The original products provide useful frequency ranges of two to one through five to one. More recent cost-effective bias tees cover 0.1 MHz to more than 4 GHz. Other designs are available that extend well into higher microwave-frequency bands (Reference 1). Another bias circuit is the bias-passing attenuator, which is also commercially available at microwave frequencies.

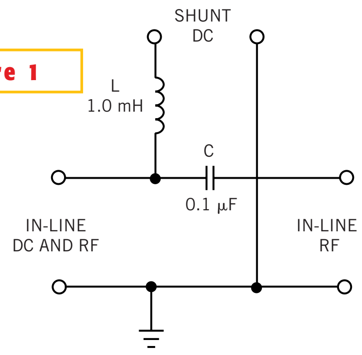
You can realize simple bias circuits at HF and VHF frequencies with minimal engineering and at much lower cost than those that must operate at microwave frequencies. You can obtain usable performance over a frequency range exceeding two decades. You can optimize the cost of these bias circuits by integrating them into subsystems and systems using surface-mount fabrication.

The monitor tee is a three-port network (**Figure 1**). One inline port handles both dc and RF. The second inline port handles only RF. The shunt port passes only dc. With values of L=1 mH and C=0.1 μF, the measured insertion loss from 0.5 to 100 MHz for inline transmission is less than 0.2 dB in a 50Ω test setup. The shunt port terminates in 50Ω.

The bias passing attenuator is a two-port network containing a fixed pad attenuator, input and output dc blocking capacitors, and a bridging RF choke (**Figure 2**). The values for L, C_1, C_2, and R_1 to R_3 in the **figure** create a nominally 6-dB attenuator. From 0.5 to 100 MHz, measured insertion loss is 6±0.5 dB in a 50Ω test setup.

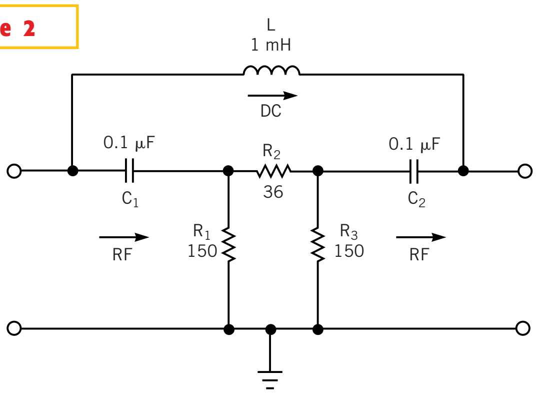
Other bias-passing circuits include lowpass, highpass, and bandpass filters. The LC lowpass filter has inherent bias-passing capability via the cascaded series inductors. The LC highpass filter needs additional circuit elements for bias passing. Sometimes, this bias-passing circuit substantially degrades the stopband selectivity of the highpass filter. You can design the LC bandpass filter for bias passing using coupled shunt resonators. The filter input and output couplings must all be inductive. Also, the shunt inductors of the individual resonators must be floating with series RF bypass capacitors for ground returns.

Figure 1



With L and C values of 1 mH and 0.1 μF, respectively, this monitor-tee network exhibits a measured insertion loss of 0.5 to 100 MHz for inline transmission of less than 0.2 dB in a 50Ω test setup.

Figure 2



This bias-passing attenuator has a measured insertion loss of 0.5 to 100 MHz of 6±0.5 dB in a 50Ω test setup.

Reference

1. Andrews, JR, "Broadband Coaxial Bias Tees," Application Note AN-1d, Picosecond Pulse Labs, Copyright February 1998.

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

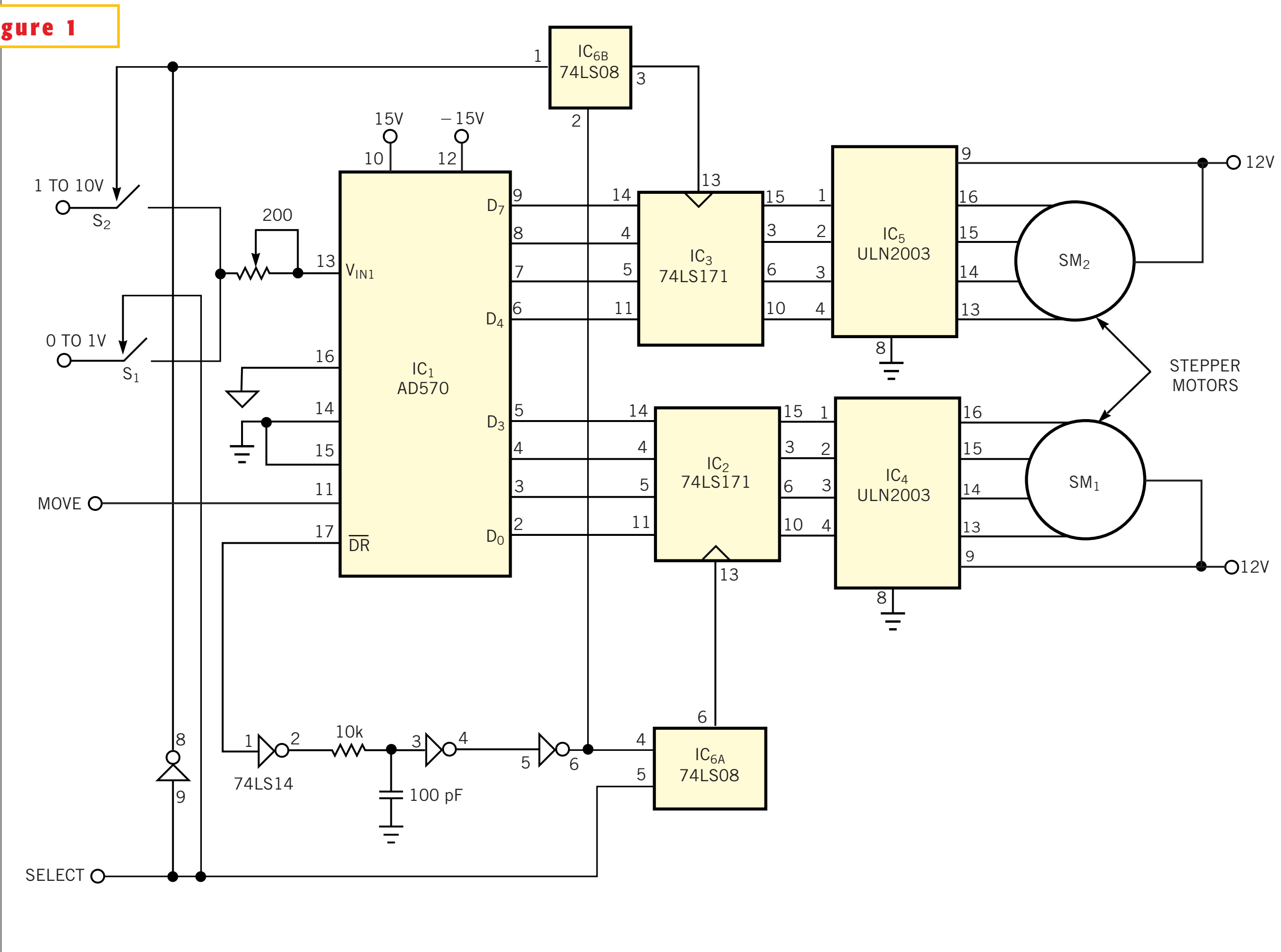
ADC controls multiple stepper motors

K Suresh, Indira Gandhi Centre for Atomic Research, Kalpakkam, India

Typical stepper-motor control circuits use either logic gates and flip-flops or shift registers to generate the proper sequences of binary codes that produce bidirectional stepper-motor movement. A conventional stepper-motor-control circuit uses a square-wave generator, a sequence generator, or a shift register and current translators to control one stepper motor apart from the logic circuit necessary for producing a known and valid binary code at start-up. When you need to control more than one stepper motor, as is the case of 2- and 3-D position control, the conventional type of control circuit becomes voluminous, complicated, and expensive due to the increased number of identical stages, the increased power dissipation, and the larger pc board.

Figure 1 shows a multiple-stepper-motor-control circuit that uses an ADC to control multiple stepper motors. The heart of this stepper-control circuit is an 8-bit, successive-approximation type ADC, (IC_1), whose 8-bit output forms two nibbles: LNIBBLE, D_0 to D_3, and UNIBBLE, D_4 to D_7. Each of these two nibbles carries valid 4-bit binary codes and drives the stepper-motor coils through a corresponding quad latch, IC_2 and IC_3, and buffer, IC_4 and IC_5. A set of four discrete analog voltages of 0 to 1V at V_{IN1} in the proper sequence control the stepper motor, SM_1, and another set of discrete analog voltages of 1 to 10V control stepper motor SM_2. A timing waveform at the MOVE input controls the start and end of each A/D conversion. The $\overline{DR}$ output of the ADC and AND gates in IC_6 generate enable signals to latch the ADC output nibbles to the

Figure 1



A simple and inexpensive ADC controls multiple stepper motors.

buffers. The SELECT input determines the selected stepper motor. A logic 1 at the SELECT input enables IC_{6A} and closes S_1, and the ADC's LNIBBLE latches in IC_2 to drive SM_1. A logic 0 at the SELECT input enables IC_{6B} and closes switch S_2, and the UNIBBLE latches in IC_3 to drive SM_2.

Tables 1 and **2** list the discrete analog voltages you must apply to the circuit in sequence and the corresponding valid ADC-generated codes. **Figure 2** shows the necessary timing waveforms for the stepper-control process. Initially, the MOVE input is at logic 1, which keeps the ADC in low-power mode and the ADC output in an open state so that both the motors are on hold. Next, you select SM_1 or SM_2 by applying a logic 1 or 0 at SELECT, and you apply any one of the four discrete analog voltages to the ADC. Then, pulling MOVE to logic 0 initiates a conversion. After approximately 42 µsec, the ADC generates a valid binary code, which the $\overline{DR}$ output of the ADC latches to the latch. The corresponding motor coils receive power according to the generated binary code through the buffer that the SELECT input enables. The MOVE input then returns back to logic 1. In this way, the circuit generates the successive valid codes by reading the corresponding analog voltages in a sequence, each time performing the A/D conversion. To rotate the selected motor in the opposite direction, you simply reverse the sequence of applied voltages.

The circuit in **Figure 1** is one way of using an ADC to control multiple stepper motors, and you can modify the circuit to suit individual requirements. Any low-cost, low-speed ADC is suitable for this circuit because the minimum time between application of successive codes, which the LR characteristics of the stepper-motor coils determine, is usually on the order of tens of microseconds. Because each motor uses only 4 bits of the ADC output, an 8-bit ADC can control two stepper motors. You can extend this concept to control three stepper motors using a 12-bit, low-speed ADC. You can apply the analog voltage to the ADC either through an analog multiplexer and a 2-bit up/down counter or through a DAC.

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

TABLE 1—SM_1 CONTROL VOLTAGES

Voltage to V_{IN1}	ADC-generated LNIBBLE			
(V_{REF}=10V)	D_4	D_3	D_1	D_0
0.118V	0	0	1	1
0.352V	1	0	0	1
0.469V	1	1	0	0
0.235V	0	1	1	0

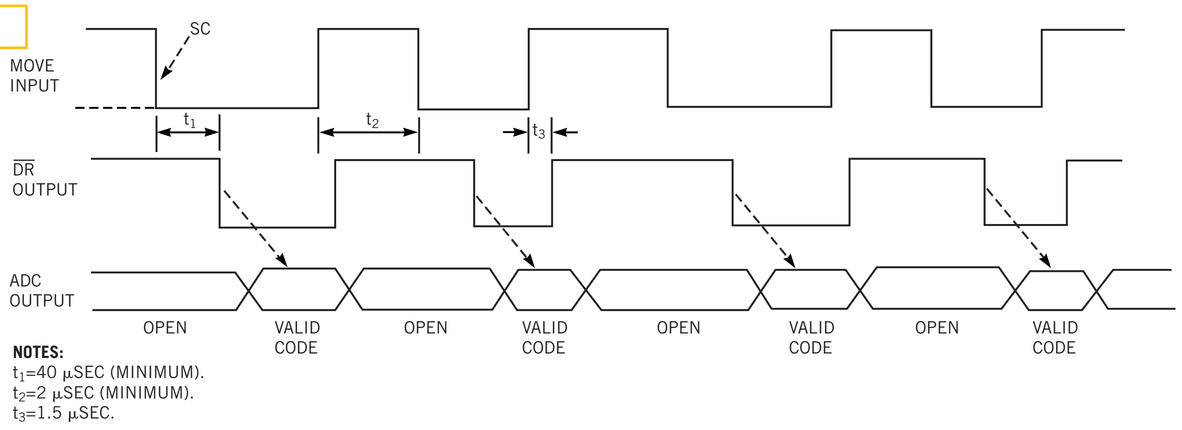
Notes:
- SELECT=logic 1
- IC_{6A} enabled
- S_1 closed
- IC_2 enabled

TABLE 2—SM_2 CONTROL VOLTAGES

Voltage to V_{IN1}	ADC-generated UNIBBLE			
(V_{REF}=10V)	D_7	D_6	D_5	D_4
1.875V	0	0	1	1
5.625V	1	0	0	1
7.500V	1	1	0	0
3.750V	0	1	1	0

Notes:
- SELECT=logic 0
- IC_{6B} enabled
- S_2 closed
- IC_3 enabled

Figure 2



Timing waveforms show the MOVE input's control of the ADC output.

PC's BIOS interrupt drives twin stepper motors

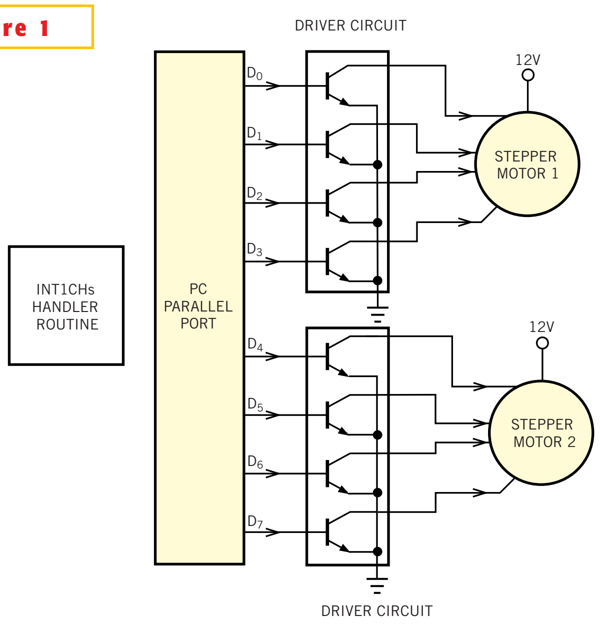
J Jayapandian, Indira Gandhi Centre for Atomic Research, Kalpakkam, India

A SIMPLE AND LOW-COST design achieves a twin stepper drive using a PC's special BIOS interrupt, INT1Ch, through the PC's parallel port (**Figure 1**). Turbo C control software programs the parallel port for the task of independently running two stepper motors. Users can write the appropriate control software for the required movement of two independent steppers. **Listing 1** is simple program to run the twin steppers in full step-clockwise mode. 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 #2590. This program can run the steppers independently through the PC's LPT2 port using a handler routine for the interrupt INT1Ch.

Figure 1 shows a block diagram of the required hardware, which the INT1Ch-handler routine can activate according to user requirements. The variable "TICKER" in the handler routine recognizes the occurrence of INT1Ch. For every occurrence of INT1Ch and thus every occurrence of TICKER, the handler routine writes the DATA necessary for the sequential step to move the stepper clockwise or counter-clockwise to the PC's LPT2 port. Every occurrence of TICKER causes the routine to go through the stepper cycle. The routine sends all the sequential DATA that the routine's SWITCH operator selects to the LPT2 port.

Full-step clockwise movement of the stepper requires four steps for the stepper coil using the following data: 1100, 0110, 0011, and 1001. The corresponding data in hex code are 0x0C, 0x06, 0x03, and 0x09, respectively, for a single stepper. For a twin stepper, the sequential-data pattern is 0xCC, 0x66, 0x33, and 0x99, respectively. The first 4 bits (least significant bits) in the 8-bit data control

Figure 1



A block diagram shows the control of two stepper motors through the PC's LPT2 port.

LISTING 1—STEPPER-MOTOR-CONTROL PROGRAM

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define OUT_PORT  0X378   /* Out port address of LPT2 */
#define CTRL_PORT 0X37A   /* Control port address of LPT2 */
#define INTRTIMER 0x1C    /* BIOS Timer (INT 1CH)Interrupt */
/*----------------------------------------------------------------------*/

/*--------------------------GLOBAL VARS-----------------------------------*/
static int TICKER;
static int  DATA;
void interrupt (*timerhandler)();
void interrupt STEPPERHANDLER();
/*-----------------------------Handler routine for INT 1C---------------*/
void interrupt STEPPERHANDLER()
{
   disable();
   switch(TICKER % 16)  /* Reminder in  No. of TICKER divide by 16  sets four cases
                          for  setting  four DATA type.  This  method of TICKER
                          Processing sets the ON time of the  stepper  of 220 milli
                          seconds. For various ON time tuning  this division factor is
                          to be changed  and the corresponding reminder is to be
                          included in the case field.  */
   {
     case  0: DATA = 0xCC; break; /* First step for clockwise full step i.e., 1 1 0 0 */
     case  4: DATA = 0x66; break; /* Second step for clockwise full step i.e., 0 1 1 0 */
     case  8: DATA = 0x33; break; /* Third  step for clockwise full step i.e., 0 0 1 1 */
     case 12: DATA = 0x99; break; /* Fourth  step for clockwise full step i.e., 1 0 0 1 */
    }
   outportb(OUT_PORT,DATA);
   ++TICKER;
   enable();

 } /* END OF STEPPERHANDLER */

 void INSTALLSTEPPERHANDLER()
 {
          disable();
          timerhandler = getvect(INTRTIMER);
          setvect(INTRTIMER,STEPPERHANDLER);
          enable();
 }

void CLEARSTEPPERHANDLER()
{
         disable();
         setvect(INTRTIMER,timerhandler);
         enable();
}

void main(void)
{
         clrscr();
         outportb(CTRL_PORT,0x01);
         INSTALLSTEPPERHANDLER();
         while (TICKER <=100)  /* This loop is only to test whether the   Sequential
                                  DATA  is properly sent to LPT2 port. This can be
                                  removed in the original program*/
         printf("TICKER No: %d  Value %X sent to port number  %x\n", TICKER,
                                                                   DATA,OUT_PORT);
          CLEARSTEPPERHANDLER();
         return;
}
```

the first stepper, and the second nibble (most significant bits) control the second stepper. You can properly assign the least significant bits and most significant bits in the data field independently for the different mode of rotation of the steppers.

The handler routine shows identical movement for both steppers, and thus the data pattern for the least significant bits and most significant bits are the same. This sample program writes the variable "DATA" necessary for the step movement of the stepper once for four occurrences of INT1Ch. Because INT1-Ch occurs once in 55 msec, the delay is 220 msec. Hence, the on-time of the stepper is fixed at 220 msec. To vary the on-time of the stepper, you can write the handler routine accordingly to change the necessary interrupt occurrences between writing new data for the stepper.

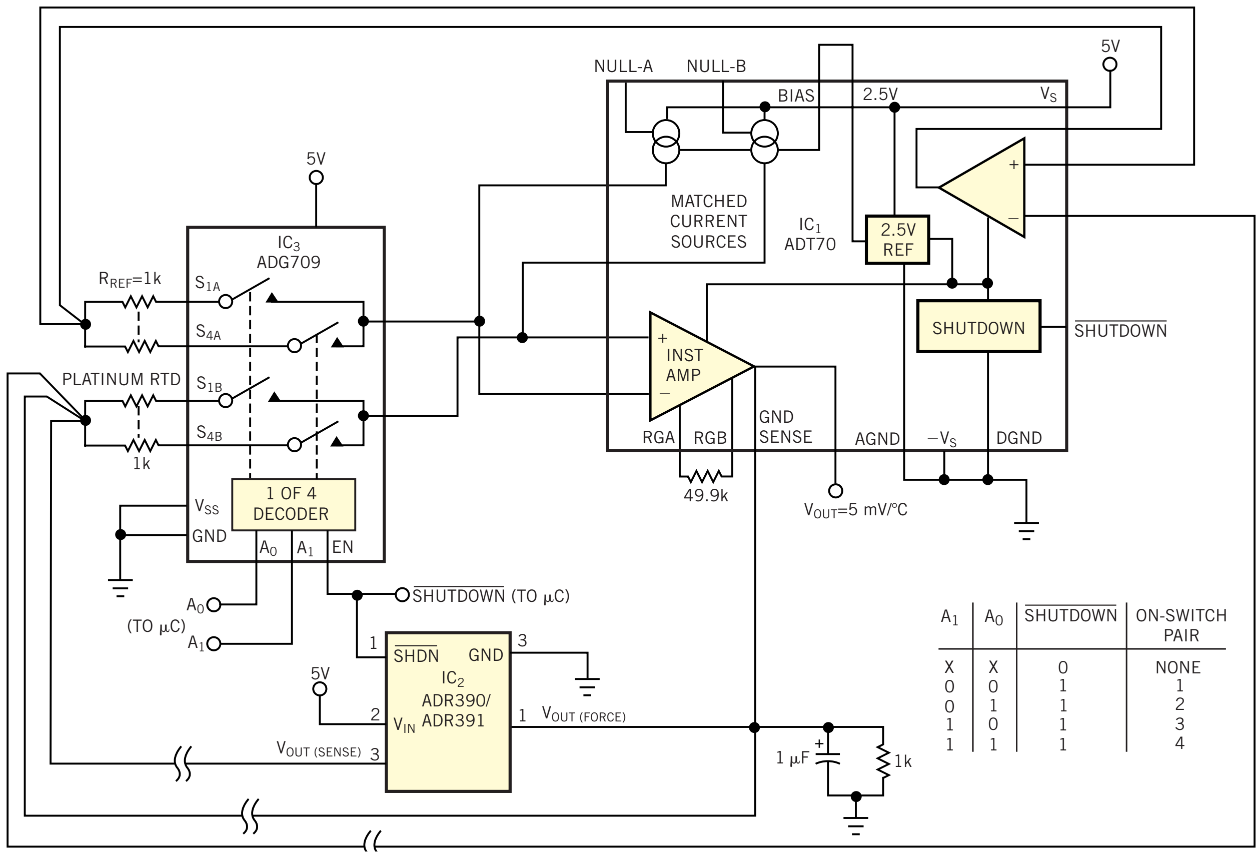
Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

Low-error platinum RTD circuit has shutdown capability

Reza Moghimi, Analog Devices, Santa Clara, CA

Resistor-temperature detectors (RTDs), are the most stable and popular temperature sensors. Platinum RTDs allow for a much wider range of temperatures than silicon-based sensors. In many cases, platinum RTDs sit far from the measurement circuitry, which adds a great deal of error into the measurement system. The circuit in **Figure 1** eliminates error by using a general-purpose amplifier and Kelvin-connected voltage references. The circuit also allows

Figure 1



A_1	A_0	$\overline{SHUTDOWN}$	ON-SWITCH PAIR
X	X	0	NONE
0	0	1	1
0	1	1	2
1	0	1	3
1	1	1	4

A general-purpose amplifier in IC_1, Kelvin-connected voltage references, and platinum RTDs enable this circuit to accurately detect temperatures of −200 to +400°C.

for single-supply operation and can detect temperatures of −200 to +400°C with an output-voltage-scaling factor (sensitivity) of 5 mV/°C. To reduce errors due to self-heating, the circuit uses the largest RTD—value, in this case 1 kΩ—that results in an acceptable response time. The larger the RTD, the longer the response time.

IC_1 provides excitation and signal conditioning for the RTD, and internal current sources provide a matching excitation of 1 mA to the platinum RTD and reference resistor, R_{REF}. The instrumentation amplifier compares the voltage drop across the platinum RTD to the drop across R_{REF} and provides an amplified output signal that is proportional to temperature.

The lead resistance of wires connecting the RTD and R_{REF} can add inaccuracy to the temperature measurement. Voltage reference IC_2 creates a pseudo ground for IC_1 to overcome this inaccuracy. IC_2 has good temperature stability and low noise and can provide 5 mA of drive current. IC_2 also has a sense pin for sensing the drop on the line and compensating for the drop. Thus, the circuit provides stable and identical voltages at the bottom of the platinum RTD and R_{REF}. The circuit also buffers this voltage using the internal amplifier of IC_1. A 1-kΩ resistor in parallel with a 1-μF capacitor at the output of IC_2 provides a path for the current to flow to ground.

IC_3 makes it possible for the μC to address the platinum RTD. The circuit in **Figure 1** can accommodate four platinum RTDs, but you can increase this number by using other differential multiplexers. IC_3's low on-resistance match between channels of 0.4Ω does not introduce large errors into the system.

Another feature of this circuit is that it allows a programmer to put the circuit into shutdown mode. Initiating shutdown disables the enable pin of IC_3 so that none of the switch pairs are on. Also, IC_1 and IC_2 shut-down to conserve power.

The tracking of the current sources of IC_1 is 2 μA, and, as stated, the matching on-resistance of IC_3 is 0.4Ω . Thus, the worst-case mismatch resulting from the current sources and switches is 0.4Ω×2 μA=0.8 μV. If higher precision matching among current sources is necessary, you can connect a 50-kΩ potentiometer between the NULL-A and NULL-B pins and connect the center tap of the potentiometer to 5V.

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

Software provides three-priority-level interrupt for 8051

Deng Yong, Department of Instrumentation Engineering, Shanghai Jiaotong University, Shanghai, China

By using a pseudo-RETI instruction, the program in **Listing 1** provides an 8051 μC with a three-level-priority interrupt system. Among the three interrupt sources in the **listing**, External Request 0 (INT0) has the highest priority, and Internal Time/Counter 0 (IT0) has the lowest priority. The IT0 interrupt-service routine, before the pseu-

LISTING 1—THREE-PRIORITY-LEVEL INTERRUPT

```
        ORG    0000H
        LJMP   START
        ORG    0003H
        LJMP   INT0
        ORG    000BH
        LJMP   IT0
        ORG    0013H
        LJMP   INT1
START:  MOV  SP, #60H
        MOV  IP,#01H              ;INT0 has high priority
        MOV  TMOD,#01H
        MOV  TH0, #00H
        MOV  TL0, #00H
        SETB  EA                  ;enable INT0,INT1,IT0
        SETB  EX0
        SETB  EX1
        SETB  ET0
        SETB  TR0
              ...
              ...
INT0:         ...    ............................;INT0 interrupt service program
              ...
INT0:         ...    ............................;INT0 interrupt service program
              ...
        RETI
INT1:         ...                                ;INT1 interrupt service program
              ...
        RETI
IT0:    CLR   TR0                                ;IT0 interrupt service program
        PUSH  DPL
        PUSH  DPH
        MOV   DPTR, #GO_ON
        PUSH  DPL
        PUSH  DPH
        RETI                                     ;"pseudo-RETI"
GO_ON:  NOP
           ...
           ...
        MOV   TH0, #00H
        MOV   TL0, #00H
        POP   DPH
        POP   DPL
        SETB  TR0
        RETI
```

do-RETI instruction, pushes the address of the first instruction behind the pseudo-RETI instruction onto the stack. The code clears the internal nonaddressable flip-flop of IT0 to acknowledge a higher interrupt after the pseudo-RETI instruction executes, and the IT0 interrupt-service routine continuously executes until the RETI instruction. You can download the listing 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 #2589.

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

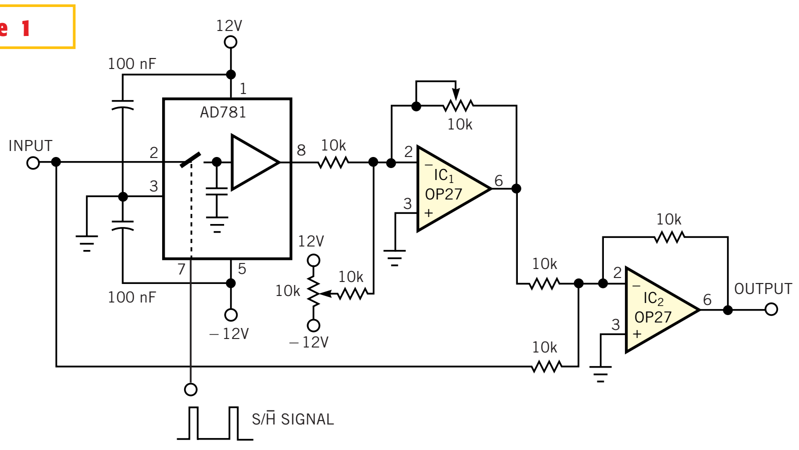
Circuit samples derivative of a waveform

Joaquin Garcia and Jose Carrasco, Universidad Miguel Hernandez, Elche, Spain

Basic derivative (differentiating) circuits use a lowpass filter or an active operational-amplifier implementation. These circuits need a large-value capacitor when differentiating a slowly varying signal. Moreover, the circuits are not useful for nonperiodic waveforms. The circuit in **Figure 1** offers a simple way of differentiating a waveform, even if it changes slowly or is nonperiodic. The circuit uses an AD781 sample/hold circuit and a subtracting circuit. By sampling the value of the input waveform at a given instant and then subtracting it from itself, op amp IC_2 produces a voltage proportional to the rate of change of the original waveform and to the duration of the hold time (the time the sample/hold input is held low). You can use a μC to transform the circuit's output to the derivative or use the output as is. IC_1 simplifies the subtraction operation and compensates the voltage shift of the sample/hold circuit during its sampling period. **Figure 2** shows a sinusoidal voltage and its derivative at the hold instants. Note that the sign of the derivative is inverted. To optimize the circuit, you should adjust the gain of the IC_2 subtraction circuit as a function of the input frequency.

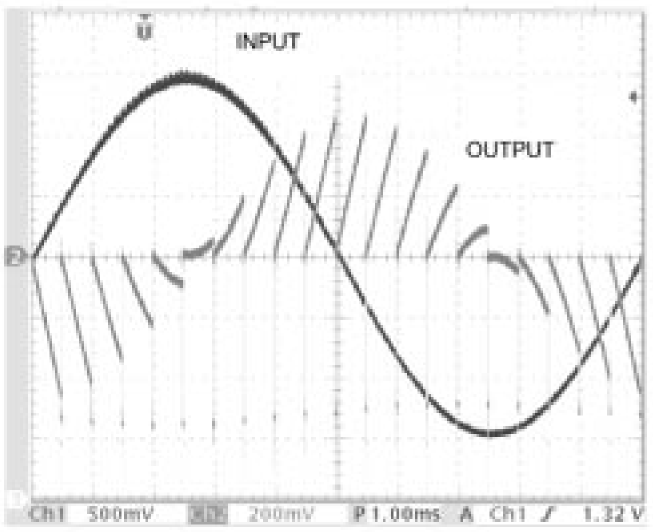
Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

Figure 1



This circuit differentiates slowly varying signals without the need for large capacitors.

Figure 2



At the hold instants of the circuit in Figure 1, the circuit produces the inverted derivative of the input waveform.