

Edited by Bill Travis and Anne Watson Swager

Make a simple PC-based frequency meter

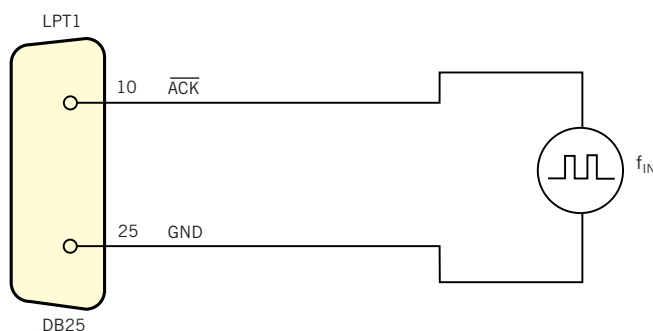
Radovan Stojanovic, University of Patras, Patras, Greece

A SIMPLE, LOW COST, and precision frequency meter uses only two pins of a pc parallel port (Figure 1). The TTL-level periodic input signal with frequency f_{IN} connects to the \overline{ACK} pin of LPT1. This input produces an IRQ7 hardware interrupt on every rising edge (Figure 2). The software counts the number of IRQ7 interrupts in the time unit of timebase T. If this timebase is 1 sec, the frequency of the input signal equals the number of IRQ7 interrupts in 1 sec.

You can use many ways to precisely generate the timebase, T, such as using the delay() or sleep() functions in C. You can also use software calibration loops. Unfortunately, these techniques are unreliable because they are based on polling. The best choice is to use a second interrupt, such as IRQ0 (software 0x1C). This interrupt is related to the internal PC timer and occurs 18.2 times/sec. For precise measurements, you can use proportional constants. For example, if your timebase equals the 18 IRQ0, you must correct the result by a factor of 18.2/18.

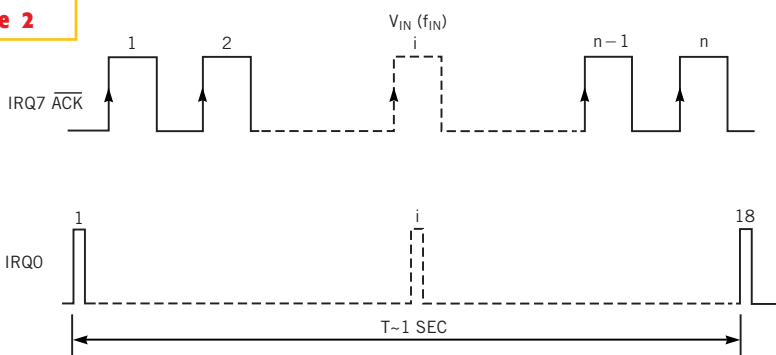
The software in Listing 1 is simple and

Figure 1



A simple, easy-to-make PC-based frequency meter uses just two parallel-port pins.

Figure 2



Input f_{IN} , which connects to the \overline{ACK} pin of LPT1, produces an IRQ7 hardware interrupt on every rising edge. The software counts the number of rising edges that occur during the timebase, T, which IRQ0 helps to generate.

interrupt-based, which allows for resident operation in an MS-DOS environment and for multitasking mode under Windows. For a timebase of 1 sec and using a 100-MHz Pentium PC, the frequency meter gives good results in the range of 10 Hz to 10 kHz with errors less

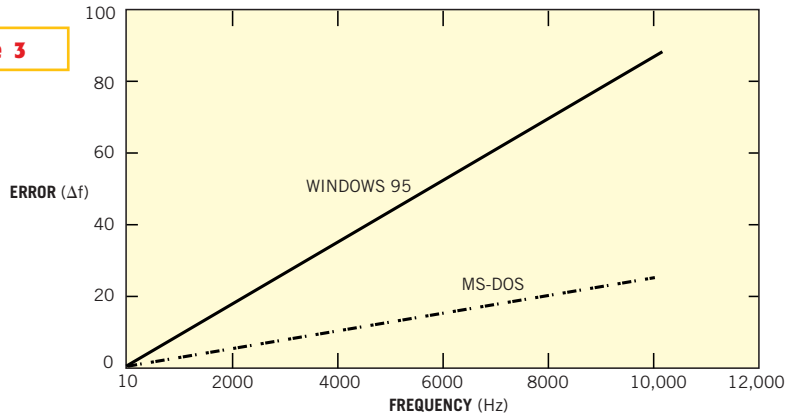
than 0.26% for DOS and 0.94% for Windows (Figure 3). This design uses a Tektronix (www.tek.com) CFG280 function generator for reference. A faster PC should produce even better results.

Listing 1 is written in Borland C++ compiler, and you can use the same pro-

| | |
|--|----|
| Make a simple PC-based frequency meter | 89 |
| Simple circuit safely deep-discharges NiCd battery | 92 |
| Algorithm extracts roots of decimal numbers | 94 |
| Reset circuit provides snappy action | 96 |
| Diodes improve inverter efficiency | 96 |

gram for DOS-based C compilers, such as Turbo C. Also, you can use the second parallel port LPT2, with interrupt IRQ5, to perform this same frequency-meter function. If you add one voltage-to-frequency, current-to-frequency, or temperature-to-frequency converter, the frequency meter can also perform simple, low-cost, and high-resolution measurements of analog values. (DI #2342)

Figure 3



TO VOTE FOR THIS DESIGN,
CIRCLE NO. 323

For a timebase of 1 sec and using a 100-MHz Pentium PC, the frequency meter gives good results that range from 10 Hz to 10 kHz with errors less than 0.26% for DOS and 0.94% for Windows.

LISTING 1—C++ FREQUENCY-METER PROGRAM

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#define INTR_1 0X08 /* The clock tick interrupt */
#define INTR_2 0x0f /* IRQ7 */
#define ISR 0x20 /*interrupt service register*/
#define IMR 0x21 /*interrupt mask register*/
#define EOI 0x20 /* End-of-interrupt*/
#ifdef __cplusplus
#define __CPPARGS ...
#else
#define __CPPARGS
#endif

int count=0; /*time base*/
int freq=0; /*frequency*/

/*interrupt functions*/
void interrupt ( *oldbase)(__CPPARGS);
void interrupt base(__CPPARGS); /*Interrupt routine for time base*/
void interrupt (*oldfreq) (__CPPARGS);
void interrupt frequ(__CPPARGS); /*Interrupt routine for frequency*/

void interrupt base(__CPPARGS)
{
    count++; /* increase the time counter */
    outp(ISR,EOI); /*end of interrupt*/
}

void interrupt frequ(__CPPARGS)
{
    freq++; /* increase the frequency counter */
    outp(ISR,EOI); /*end of interrupt*/
}

int main(void) /*main program*/
{
    int a;
    int port;
    int stop;
    float factor;

    factor=18.2/18; /*prop. constant for 1s*/
    port=peek(0x40,8); /*find the address of the LPT1*/
    a=inp(port+2); /*read the Control register of the LPT1*/
    a=a|0x10;
    outp(port+2,a); /*enable interrupts on the pin ACK*/
    outp(ISR,EOI);
    disable();
    outp(IMR,0x20); /*Enable IRQ7 in the interrupt mask register*/
    outp(ISR,EOI);
    oldbase = getvect(INTR_1); /* the settings of the interrupts*/
    oldfreq=getvect(INTR_2);
    setvect(INTR_1,base);
    setvect(INTR_2,frequ);
    enable();

    while(!kbhit())
    {
        if(count>=18) /* display frequency*/
        {
            stop=freq;
            printf("\n F=%2f [Hz]",stop*factor);
            freq=0; /*reset variables*/
            count=0;
        }
    }

    setvect(INTR_1, oldbase); /* reset interrupts vectors*/
    setvect(INTR_2,oldfreq);

    return 0;
}
```

Simple circuit safely deep-discharges NiCd battery

Jim Hagerman, Science & Technology International, Honolulu, HI

NICKEL-CADMIUM (NiCd) batteries can possess an undesirable memory effect due to partial discharges. The remedy is a complete discharge before charging again. **Figure 1a** shows a simple circuit that performs this feat.

Although relatively straightforward in concept, the circuit has three redeeming features: It receives its power from the battery undergoing the discharge; after the battery is fully discharged, the current drain is only approximately 4 μ A, which is usually well below the self-discharge level of a battery alone; and an LED flashes at end-of-charge.

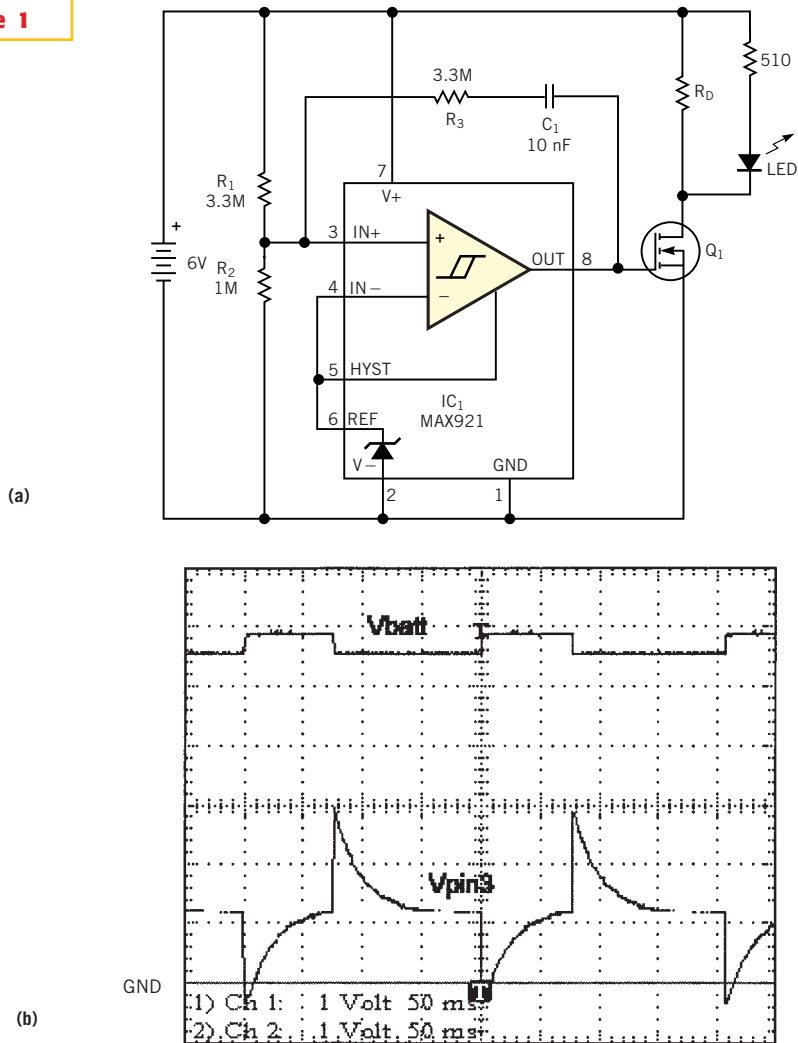
Ignoring LED current, R_D determines the rate at which the circuit discharges the battery, as follows:

$$I_{\text{DISCHARGE}} = \frac{V_{\text{BATTERY}}}{R_D}$$

A NiCd or nickel-metal-hydride (NiMH) battery has a nominal cell voltage of approximately 1.2V at midcharge and approximately 1V at end of charge. Do not discharge past the end-of-charge voltage, because you may damage the battery. The values in **Figure 1a** are for a four-cell battery. R_1 and R_2 determine the end-of-voltage limit referenced to the built-in 1.182V bandgap reference. When battery voltage is high, comparator IC_1 turns on Q_1 , a power n-channel MOSFET, which discharges the battery through R_D .

When the battery reaches the end-of-charge voltage, the circuit's behavior gets interesting. R_3 and C_1 provide positive ac feedback to ensure that the comparator fully switches and prevents the circuit from becoming a linear regulator. However, the intrinsic internal resistance of the battery also causes negative dc feedback. As the MOSFET turns off the battery terminal voltage, the comparator turns the MOSFET back on. The positive ac feedback overwhelms the negative feedback, thus ensuring switching, but only for a short time, and the circuit oscillates. The frequency is roughly

Figure 1



When the battery reaches the end of charge, this deep-discharge circuit (a) oscillates until the battery voltage stays below the hysteresis threshold (b).

$$f = \frac{1}{2 \cdot p \cdot (R_3 + R_1 || R_2) \cdot C_1}$$

Oscillation eventually stops when the battery voltage stays below the hysteresis threshold, which the intrinsic resistance of the battery determines. The higher the resistance, the longer the LED will flash. The circuit takes advantage of the fact that this resistance is greatest at the end

of charge. When the discharge cycle stops, the LED stays off, and the only current drain is from IC_1 and the R_1/R_2 divider. Both of these values are small enough to leave the battery connected indefinitely. (DI #2348)

TO VOTE FOR THIS DESIGN,
CIRCLE NO. 324

Algorithm extracts roots of decimal numbers

Frank Vitaljic, Bellingham, WA

ALGORITHM EXTRACTS CUBE ROOT[®] (EDN, Jan 15, 1998, pg 100) covers only the one-third power (cube root). In contrast, the C routine in **Listing 1** calculates the Kth root ($X^{1/K}$) of positive decimal numbers X. Both K and X can vary widely. You type in X, K, and an estimate of the root; the routine then calls the calcRoot function in the software program. Upon calculating the root, the routine prints on screen the number of iterations performed and the root result $X^{1/K}$. The routine raises this result to the Kth power and displays the result so you can make a comparison with the original X. The algorithm applies a Newton-Raphson approach to the equation $Y=X^{1/K}$. If you differentiate the equation and express it in recursive form, you obtain

$$\text{ERROR} = Y(n+1) - Y(n) = \frac{X - Y(n)^K}{KY(n)^{K-1}},$$

FOR $n = 0, 1, 2, 3K$ AND

$K = 2, 3, 4, 5K$

This recursion monotonically converges toward the Kth root of the num-

TABLE 1—CUBE-ROOT MAXIMUM ERRORS

| X | % Error |
|-------|---------|
| 0.001 | 0.001 |
| 0.01 | 0.0005 |
| 0.1 | 0.0002 |
| 10 | 0.00005 |
| 100 | 0.00002 |
| 1000 | 0.00001 |

TABLE 2—CUBE-ROOT ITERATIONS

| X | Number of iterations 1998 Design Idea | Number this Design Idea |
|-------|---------------------------------------|-------------------------|
| 0.001 | 10 | 10 |
| 0.01 | 12 | Eight |
| 0.1 | 14 | Six |
| 10 | 19 | Seven |
| 100 | 23 | 11 |
| 1000 | 22 | 14 |

LISTING 1—KTH-ROOT EXTRACTION FOR DECIMAL NUMBERS

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>

#define MIN(x,y) ((x) < (y)) ? (x) : (y) /* min, max */
#define MAX(x,y) ((x) > (y)) ? (x) : (y) /* values */

#define VIEWDATA(arrayname,numdata,datatype,lines,colms, \
    colmwidth,dataform,prec,justify) \
{ \
    int i, j=0, k=0; \
    double sum=0.0, sum2=0.0; \
    double mean, variance; \
    datatype datamin, datamax; /* min, max data values */ \
    datatype *ptr = arrayname; /* array pointer */ \
    cout.precision (prec); /* tabulated precision */ \
    cout.setf (ios::justify); /* left or right */ \
    cout.setf (ios::uppercase); \
    cout.setf (ios::showbase); /* base of integers */ \
    clrscr(); /* clear screen */ \
    for (i=0; i < numdata; i++) { /* tabulate numdata items */ \
        if (j==0) cout << dec << i << " | "; \
        cout.width (colmwidth); \
        cout.setf (ios::dataform); /* dec, hex, oct, scientific, fixed */ \
        cout << *ptr; /* display data columns */ \
        if (++j==colms) { \
            j = 0; k++; \
            cout << "\n"; \
        } \
        if (k==lines) { /* display data lines */ \
            cout << "\n"; \
            k = 0; \
            getch (); /* pause */ \
        } \
        if (i==1) { /* Initialize data min, max */ \
            datamin = MIN (*ptr, *(ptr-1)); \
            datamax = MAX (*ptr, *(ptr-1)); \
        } \
        if (i > 1) { /* find min, max values */ \
            datamin = MIN (*ptr, datamin); \
            datamax = MAX (*ptr, datamax); \
        } \
        sum += (double)(*ptr); \
        sum2 += ((double)(*ptr))*((double)(*ptr)); \
        ptr++; /* increment array pointer */ \
    } \
    /* end for() */ \
    mean = sum / numdata; /* calculate data stats */ \
    variance = (sum2 - sum*mean) / numdata; \
    cout << "\nSTATS: "; \
    cout << "\nmin = " << datamin; \
    cout << " max = " << datamax; \
    cout << "\nmean = " << mean; \
    cout << " variance = " << variance; \
    cout.unsetf (ios::dataform); /* reset flags */ \
    cout.unsetf (ios::justify); \
    getch (); /* pause */ \
} \
/* end viewdata() */

void main (void)
{
    double a[1000]; /* declare array of doubles */
    int i; /* counter variable */

    /* generate 200 elements of doubles */
    for (i=0; i < 200; i++) a[i] = 0.333333*i-16.0;

    /* Display first 100 elements in increments of 14 lines,
    3 columns each 20 spaces wide, in scientific form, 3-digit
    precision, right justified. */

    VIEWDATA (a,100,double,14,3,20,scientific,3,right)
}
```

ber. The routine terminates the iteration when

$$|Y(n+1)-Y(n)| \leq \text{ERROR} = 10^{-6}.$$

The error in the calculation is

$$\% \text{ERROR} \leq \frac{\text{ERROR} \cdot 100}{Y(\text{ACTUAL})}.$$

Table 1 shows cube-root errors for

numbers 0.001 to 1000. The number of iterations varies from six to 14. Table 2 compares the number of iterations inherent in the algorithm with the number of iterations of the earlier Design Idea. For increased accuracy, you can set the maximum ERROR define to $1.0e-14$ at the expense of increasing the number of iterations. You can download the C list-

ing from EDN's Web site, www.ednmag.com. Click on "Search Databases/Links Page" and then enter the Software Center to download the file for Design Idea #2339. (DI #2339).

TO VOTE FOR THIS DESIGN,
CIRCLE NO. 325

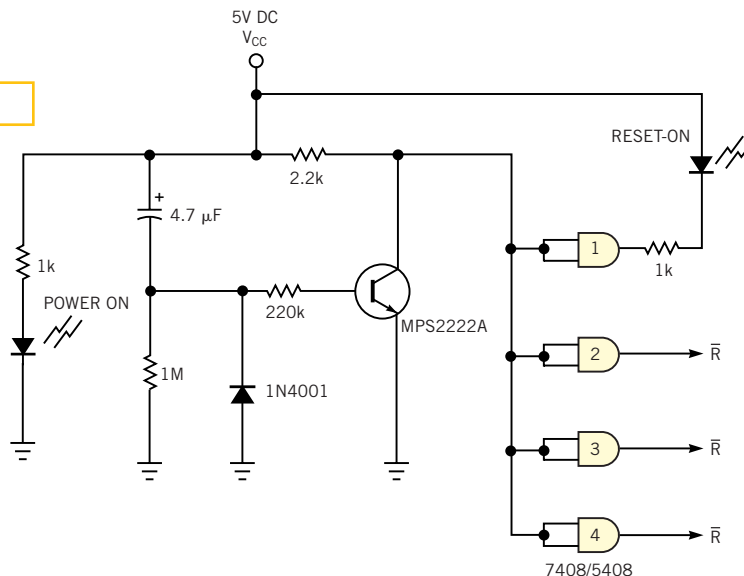
Reset circuit provides snappy action

Steve Kelley, Protobyte Inc, Liberty, MO

THE CIRCUIT IN Figure 1 provides a 1-sec reset, using multiple AND gates for increased fanout. Intended for breadboarding or prototyping, the circuit generates a "snappy" master reset that remains active long enough to eliminate questionable start-ups. When you apply power to the circuit, the timing capacitor begins charging through the 1-MΩ resistor, turning on the npn transistor. The AND-gate inputs go low, generating the reset signal and turning on the reset LED. When the timing capacitor brings the transistor's base nearer to ground, the transistor turns off, and the 2.2-kΩ resistor pulls the AND gates' inputs high. The transistor isolates the capacitor from the AND gates' internal pull-up resistors and provides a smooth transition through the logic threshold. Upon removal of power, the 1N4001 diode discharges the timing capacitor. (DI #2351).

TO VOTE FOR THIS DESIGN,
CIRCLE NO. 326

Figure 1



Eschew marginal, wimpy reset signals by using this high-fanout, snappy-reset circuit.

Diodes improve inverter efficiency

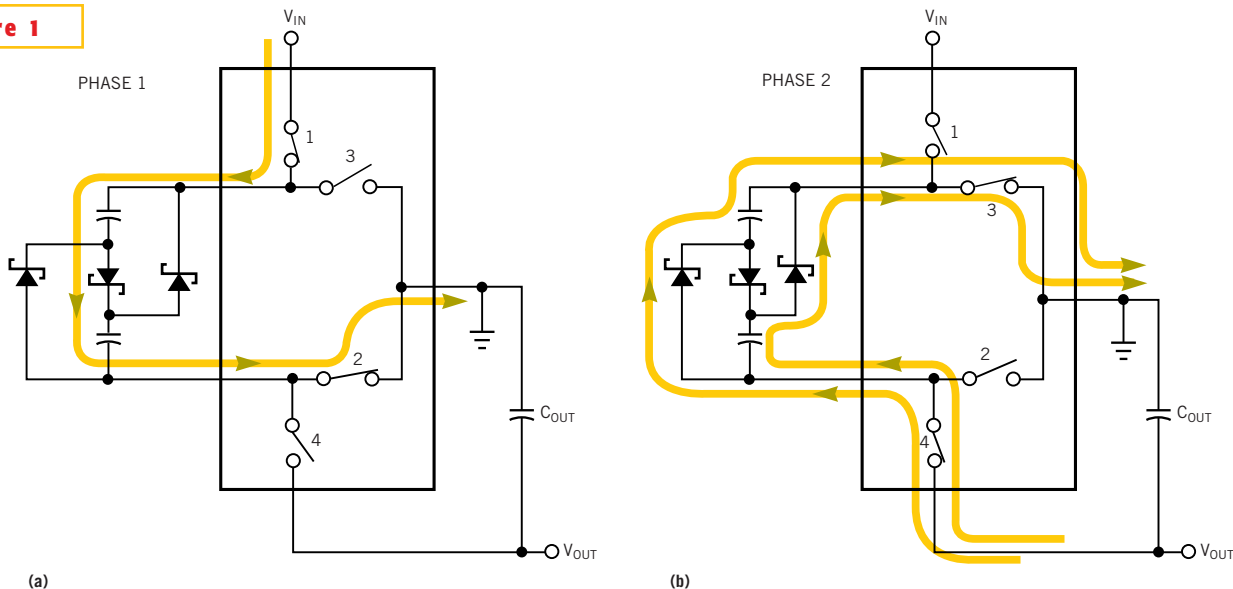
Jeff Witt, Linear Technology Corp, Milpitas, CA

IN "SWITCHED-CAPACITOR REGULATOR PROVIDES GAIN" (EDN, March 13, 1998, pg 80) a switched-capacitor voltage inverter wired as a supply splitter steps a positive voltage down to an

output voltage equal to half the supply voltage. The input current to such a circuit is one-half the output current; thus, you obtain higher efficiency and lower power loss than you would from a linear

regulator performing the same function. You can obtain the same benefits while inverting the input voltage. Figure 1 shows the principles of operation. Typical switched-capacitor inverters contain

Figure 1

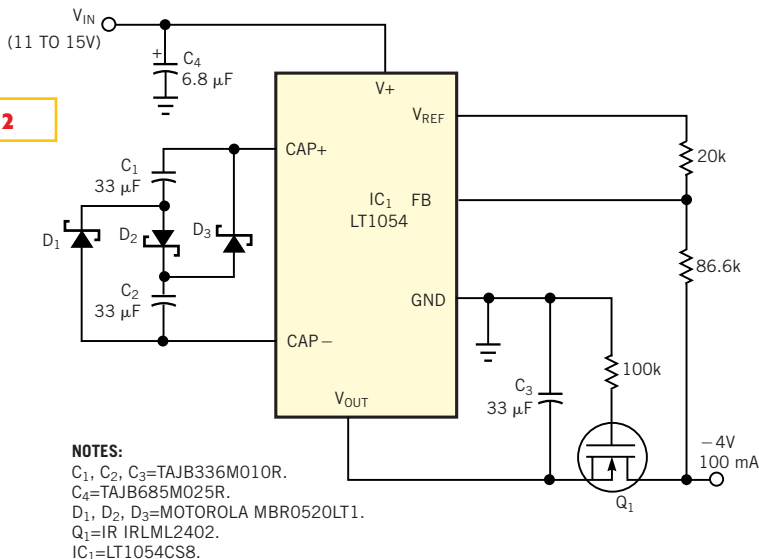


Adding a few diodes to a switched-capacitor inverter doubles the current from input to output.

four internal switches. Two switches charge a flying capacitor from the input voltage to ground; the other two switches discharge the capacitor, pulling the positive side to ground and generating a negative output voltage.

By adding three diodes to steer the current, you can use the switches to charge two capacitors in series and then discharge them in parallel to an output capacitor. **Figure 1** shows the current paths. The absolute value of the output voltage equals half the input voltage minus some loss from the switches and diodes. **Figure 2** shows a practical circuit, which converts 12V to $-4V$ at 100 mA. IC_1 modulates the voltage drop across Switch 1 to regulate the output, maintaining $-4V$ from an input of 11 to 15V. Many negative supplies power loads that can pull the output above ground (op-amp circuits in particular). Q_1 prevents such a load from pulling IC_1 's output pin above the voltage on the op amp's ground pin. Because most of IC_1 's operating current flows from its ground pin, the input current to this circuit is slightly more than half the output current. When the circuit delivers 100-mA load current, measurements

Figure 2



This 12V to $-4V$ converter delivers 100-mA output current with 63-mA input current.

show that the 12V input delivers 64 mA. The circuit dissipates only 350 mW, so an all-surface-mount configuration runs cool. (DI #2352).

TO VOTE FOR THIS DESIGN,
 CIRCLE NO. 327

Design Idea Entry Blank

Entry blank must accompany all entries. \$100 cash award for all published Design Ideas. An additional \$100 cash award for the winning design of each issue, determined by vote of readers. Additional \$1500 cash award for annual Grand Prize Design, selected among biweekly winners by vote of editors.

To: Design Ideas Editor, EDN Magazine
275 Washington St, Newton, MA 02458

I hereby submit my Design Ideas entry.

Name _____

Title _____

Phone _____

E-mail _____ Fax _____

Company _____

Address _____

Country _____ ZIP _____

Design Idea Title _____

Social Security Number _____
(US authors only)

Entry blank must accompany all entries. (A separate entry blank for each author must accompany every entry.) Design entered must be submitted exclusively to *EDN*, must not be patented, and must have no patent pending. Design must be original with author(s), must not have been previously published (limited-distribution house organs excepted), and must have been constructed and tested. Fully annotate all circuit diagrams. Please submit software listings and all other computer-readable documentation on a IBM PC disk in plain ASCII.

Exclusive publishing rights remain with Cahners Business Information unless entry is returned to author or editor gives written permission for publication elsewhere.

In submitting my entry, I agree to abide by the rules of the Design Ideas Program.

Signed _____

Date _____

Your vote determines this issue's winner. Vote now, by circling the appropriate number on the reader inquiry card.