

Edited by Bill Travis and Anne Watson Swager

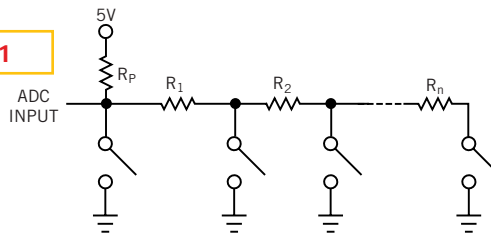
ADC circuit optimizes key encoding

Vitor Amorim, Siemens, and J Simões, University of Coimbra, Coimbra, Portugal

You can implement a simple and low-cost keyboard encoder by using a key-controlled resistive divider connected to an ADC (**Figure 1**). This type of circuit is especially suitable for embedded systems that use a μC with an integrated ADC section. To obtain the best noise margin between keys, select resistors to yield an equal division of the voltage levels. To meet this criterion using the circuit in **Figure 1**, you must use a set of resistors with all-different, specific values. For example, a 10-key keyboard uses a 10-k Ω pullup resistor, R_p , and values of 1.1, 1.3, 1.8, 2.4, 3.3, 5.1, 8.2, 16, and 51 k Ω for R_1 through R_9 . Using commonly available resistor values and tolerances and taking account of the key resistivity and ADC linearity errors, you're limited in the number of keys you can encode with a safe noise margin. Using an 8-bit ADC, the cited 10-key example is close to that limit.

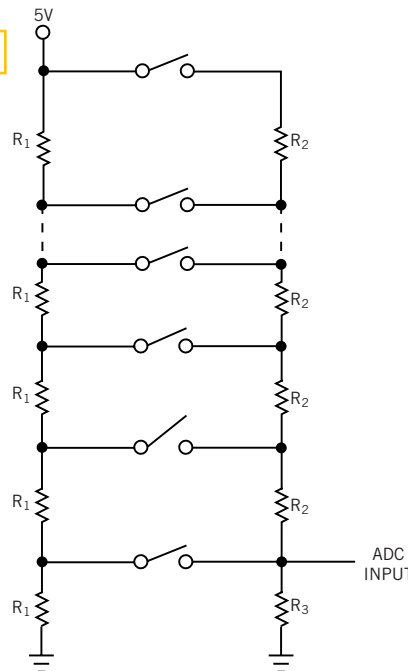
The circuit in **Figure 2**, despite its many resistors, overcomes the problem of the many resistor values. The circuit is symmetric and uses the same two resis-

Figure 1



This is a simple way to encode keys, but it requires a range of resistor values for a large number of keys.

Figure 2



This key-encoding circuit simplifies resistor inventories, and provides a comfortable noise margin to boot.

tor values for all keys. It is also easy to expand the circuit for more keys. As in **Figure 1**'s circuit, if you simultaneously press more than one key, the circuit detects only the key whose connection is closest to the ADC. The chain of R_1 resistors de-

fines the voltage level associated with each key. Their nominal value is a trade-off between the power dissipation and the values of R_2 and R_3 , which depend on R_1 ; the total number of keys; and the desired noise margin. R_2 minimizes the voltage deviations at the nodes of the R_1 chain whenever you simultaneously press two or more keys. Therefore, you should calculate its value, much higher than that of R_1 , by taking into account R_1 and the desired width of the voltage window associated with each key. Similarly, R_3 's value should be much higher than that of R_2 to ensure that the voltage level associated with each key almost completely transfers to the ADC's input.

The circuit was tested by encoding 15 keys with one 8-bit analog input of the Microchip PIC16C71 μC . The resistor values are 47 Ω , 3.9 k Ω , and 4.7 M Ω for R_1 , R_2 , and R_3 , respectively. Only R_1 needs a tight tolerance; the others are less demanding. The window of key acceptance is set to a 7-LSB interval centered on the theoretical key level. This interval is large enough to accommodate the worst case (simultaneously pressing the two more-distant keys from the analog input) with a safety noise margin between valid keys of 10 LSBs. (DI #2319).

To Vote For This Design,
Circle No. 332

ADC circuit optimizes key encoding	101
RS-232C circuit has galvanic isolation	102
Digital pot adjusts LCD's contrast	104
Add speech encoding/decoding to your design	106
Cheap PWM IC makes synchronous gate driver	110
Circuit detects total on-time	112

RS-232C circuit has galvanic isolation

Ioan Ciascai, REI Data, Napoca, Romania

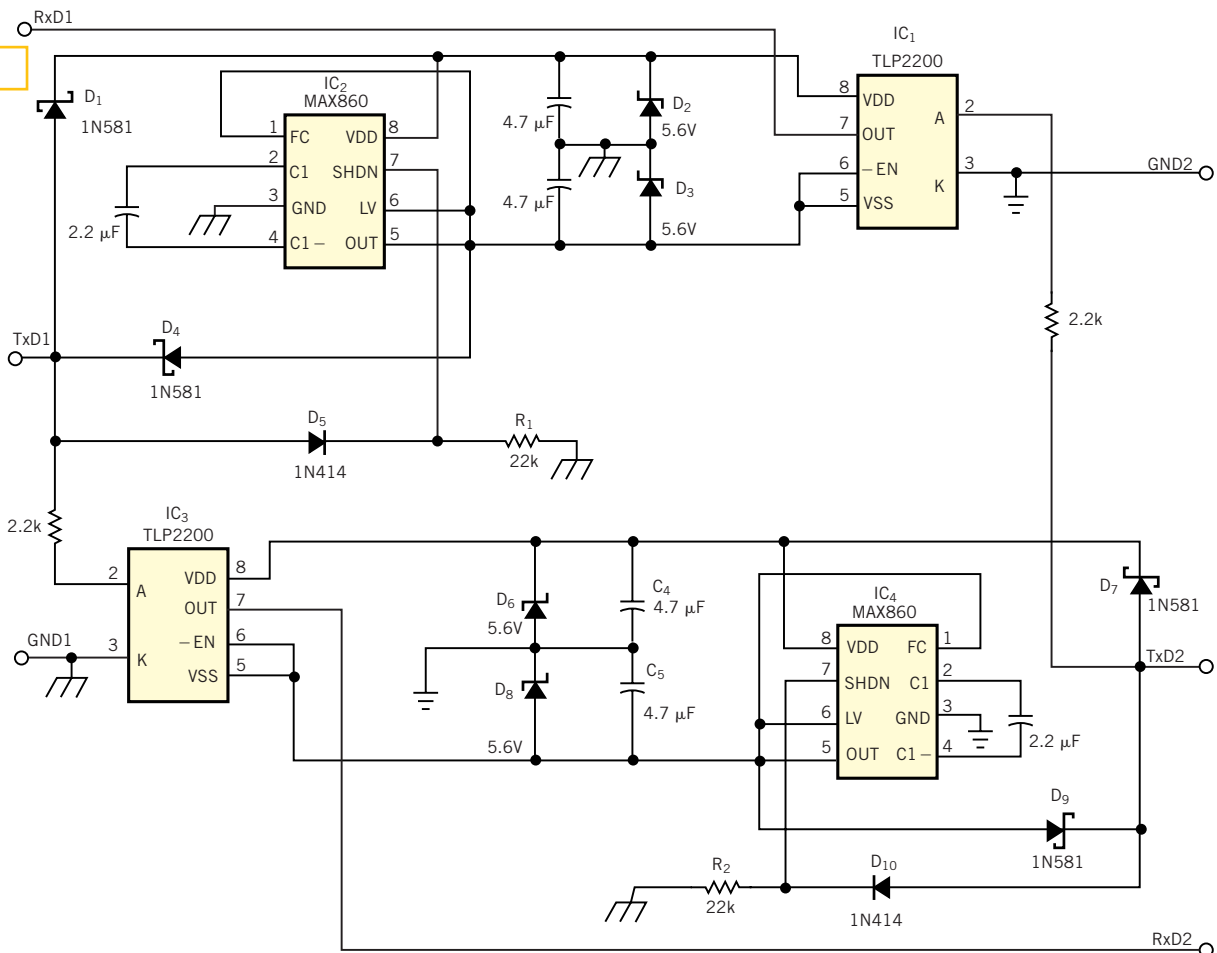
You can obtain longer transmission distances with the RS-232C interface if you use galvanic isolation between the two linked terminals. Galvanic isolation also eliminates problems arising from disparate potentials between terminals. Using two MAX860/861 ICs and two TPL2200 optoisolators, you can obtain galvanic isolation for three-wire transmission without external supplies (**Figure 1**). The MAX860/861 circuits, which generate two voltages of different polarity, regardless of the polarity of TxD, which provides the supply voltage, are the basis of the design.

For the positive TxD polarity, the MAX860/861 ICs function as voltage inverters, whereas for negative-polarity TxD, the ICs function as voltage doublers. Diodes D_1 and D_7 provide the positive supply voltages; D_4 and D_9 provide the negative supply voltages, depending on the polarity of TxD. The diode-resistor pairs D_5, R_1 and D_{10}, R_2 determine the operating mode (doubler or inverter) of the MAX860/861 ICs, depending on the polarity of TxD. Zener diodes D_2, D_3, D_6 , and D_8 protect the MAX860/861 ICs from supply overvoltages.

The digital-output TLP2200 optoisolators provide galvanic isolation and generate the RxD received signals with RS-232C logic levels. The circuit provides galvanically isolated communication in full-duplex mode at any standard transmission speed. The use of galvanic isolation allows transmission over nearly twice the distance for nonisolated systems. If you use a four-wire cable and split the separation circuits at the cable ends, you can further increase the transmission distance. (DI #2315).

To Vote For This Design,
Circle No. 333

Figure 1



RS-232C-interface ICs and optoisolators provide a supplyless RS-232C transmission link with galvanic isolation for increased distance.

Digital pot adjusts LCD's contrast

Jef Collin, CSE Systems, Turnhout, Belgium

You can use a digitally controlled potentiometer for many purposes. In this example, you can use the device to regulate the contrast of a standard (such as two lines by 40 characters) LCD. You can use the circuit in **Figure 1** in a portable test system, in which you need to change the contrast of the LCD as a function of the viewing angle. You choose the contrast setup from a menu and then use up or down buttons with μC IC₁ to adjust the contrast. The μC stores the contrast value in the digital potentiometer, IC₂. This design uses a Xicor 10-k Ω unit (dubbed "EEPOT"), but you could use other devices in the design.

The EEPROM connects to the LCD's VO line. You could connect the other side of the potentiometer to ground, but, for better contrast, you can apply a negative voltage to this terminal. This circuit has a

```

LISTING 1—ROUTINE FOR LCD-CONTRAST ADJUSTMENT

; *** BIT DEFINITIONS ***
EEPOT_INC BIT P1.4      ; EEPROM CONNECTIONS
EEPOT_UD BIT P1.5
EEPOT_CS BIT P1.6

; *** INITIAL SETTINGS ***
SETS EEPROM_CS
SETS EEPROM_UD
SETS EEPROM_UD

; *** UP CONTRAST ***
CONT_UP:  NOP
          SETB EEPROM_UD
          SETB EEPROM_UD
          NOP
          CLR EEPROM_CS
          NOP
          CLR EEPROM_INC
          NOP
          SETB EEPROM_INC
          NOP
          SETB EEPROM_INC
          NOP
          CLR EEPROM_INC
          ...

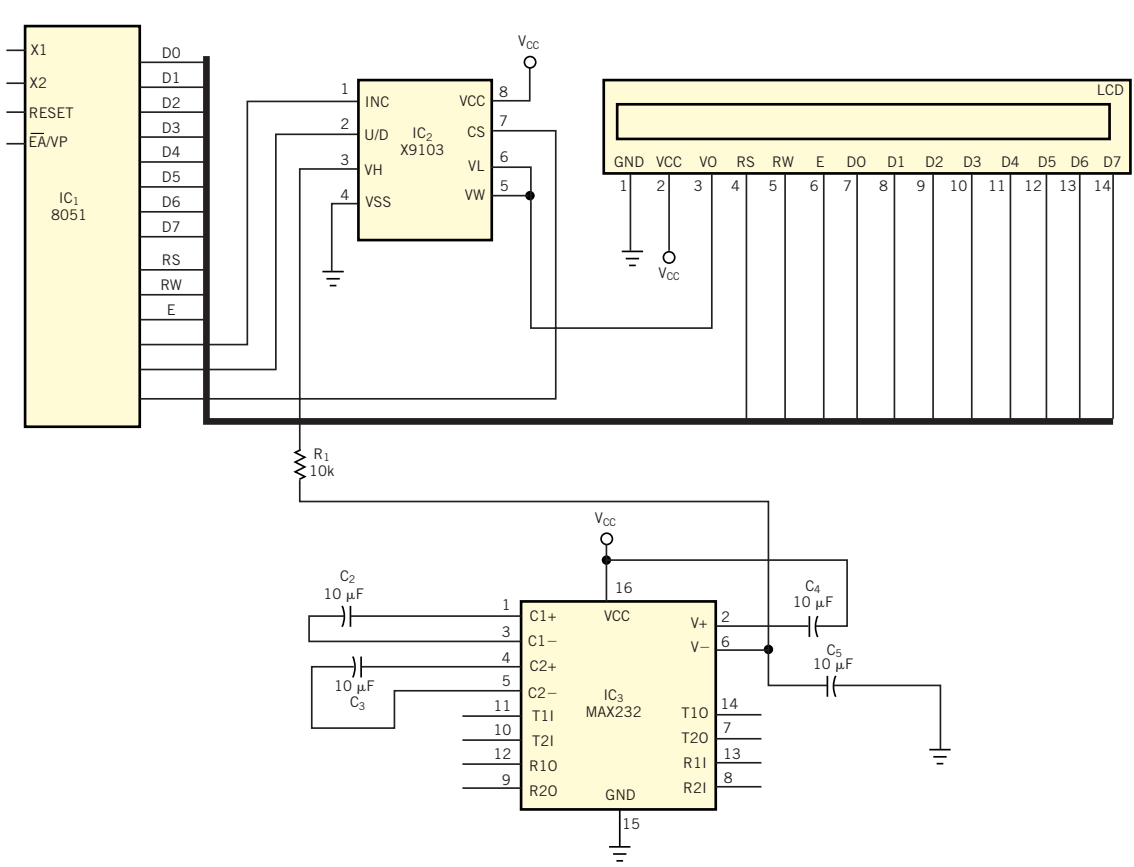
; *** DOWN CONTRAST ***
CONT_DWN: NOP
          CLR EEPROM_UD
          SETB EEPROM_UD
          NOP
          CLR EEPROM_CS
          NOP
          CLR EEPROM_INC
          NOP
          SETB EEPROM_INC
          NOP
          CLR EEPROM_INC
          NOP
          SETB EEPROM_INC
          NOP
          SETB EEPROM_CS
          RET
    
```

serial interface, so it uses the negative-voltage generator in the MAX232 RS-232C/TTL converter. **Listing 1** is the sub-routine for the 8051 μC . The program

calls the routines for contrast-up or -down. (DI #2314).

To Vote For This Design,
Circle No. 334

Figure 1



A standard 8051 μC and a digitally controlled potentiometer provide a convenient way to vary the contrast of an LCD, using contrast-up and -down buttons.

Add speech encoding/decoding to your design

Rodger Richey, Microchip Technology Inc, Chandler, AZ

Adding speech capabilities to a design can sometimes lead to complex algorithms and expensive DSPs or specialized audio chips. However, with the completion of a simplified adaptive differential pulse-code modulation (ADPCM) algorithm, you can now implement these audio capabilities in low-cost 8-bit μ Cs, which typically have lower power consumption and cost than their DSP or audio-chip counterparts. A two-chip design is feasible by offloading the encoding and decoding tasks onto the μ C as if it were a peripheral.

Since 1991, the Interactive Multimedia Association (IMA) Digital Audio Technical Working Group (DATWG) has been working to define a cross-platform digital-audio exchange format. An inherent problem exists with the exchange of audio data between PC, Mac, and workstation computers. Each computer has its own data types and sampling rates. In May 1992, IMA DATWG published the first revision of the Cross-Platform Digital Audio Interchange recommendation that specifies three uncompressed and one compressed data type at various sample rates. The compressed data type is the Intel (www.intel.com) 4-bit ADPCM algorithm. This algorithm compresses a 16-bit signed audio sample into 4 bits and takes advantage of the high correlation between consecutive samples, which enables the prediction of future samples. Instead of encoding the sample itself, ADPCM encodes the difference between a predicted sample and the actual sample. This method provides more efficient

compression with fewer bits per sample and yet preserves the overall quality of the audio signal. Both the encoder routine (**Listing 1**) and the decoder routine (**Listing 2**) are written in C to ease readability.

LISTING 1—ADPCM ENCODER

```

/* Table of index changes */
const char IndexTable[16] = {-1,-1,-1,-1,2,4,6,8,-1,-1,-1,-1,2,4,6,8};

/* Quantizer step size lookup table */
const long StepSizeTable[89] = {7,8,9,10,11,12,13,14,16,17,19,21,23,25,28,31,34,37,41,
45,50,55,60,66,73,80,88,97,107,118,130,143,157,173,190,
209,230,253,279,307,337,371,408,449,494,544,598,658,724,
796,876,963,1060,1166,1282,1411,1552,1707,1878,2066,2272,
2499,2749,3024,3327,3660,4026,4428,4871,5358,5894,6484,
7132,7845,8630,9493,10442,11487,12635,13899,15289,16818,
18500,20350,22385,24623,27086,29794,32767};

signed int diff;
int step;
signed int predsample,diffq;
char index;

char ADPCMEncoder( signed int sample )
{
    char code;

    predsample = state.prevsample;
    index = state.previndex;
    step = StepSizeTable[index];

    diff = sample - predsample;
    if(diff >= 0)
        code = 0;
    else
        code = 8;
        diff = -diff;
}

diffq = step >> 3;
if( diff >= step )
{
    code |= 4;
    diff -= step;
    diffq += step;
}
step >>= 1;
if( diff >= step )
{
    code |= 2;
    diff -= step;
    diffq += step;
}
step >>= 1;
if( diff >= step )
{
    code |= 1;
    diffq += step;
}

if( code & 8 )
    predsample -= diffq;
else
    predsample += diffq;
if( predsample > 32767 )
    predsample = 32767;
else if( predsample < -32768 )
    predsample = -32768;

index += IndexTable[code];
if( index < 0 )
    index = 0;
if( index > 88 )
    index = 88;

state.prevsample = predsample;
state.previndex = index;
return ( code & 0x0f );
}

```

The hardware implementation depends on the type of interface. With a parallel interface, you can use the PIC16C556A (Microchip Technology, www.microchip.com) (**Figure 1a**). A standard parallel interface uses the chip-

select (CS), output-enable (OE), and write-enable (WR) pins on Port A. The 8-bit data interface connects to the 8-bit Port B on the μ C. You can use two additional I/O lines for status information, such as encode/decode select, to the μ C or an interrupt line to the main controller to indicate when data is ready. CS, which connects to the RA₄ pin, can interrupt the PIC16C556A on the start of a transmission.

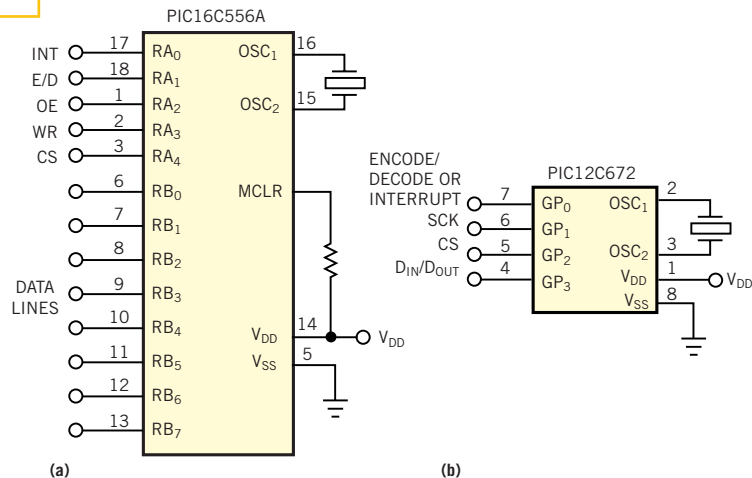
The second hardware implementation uses a serial interface and an eight-pin μ C (Figure 1b). The PIC12C672 uses four of the pins for power, ground, and oscillator input and output. Three of the remaining I/O pins are for clock (SCK), data in/out (D_{IN}/D_{OUT}), and CS. You can use the other I/O pin to indicate the desired encode/decode operation or as an interrupt to the main controller. CS connects to the external interrupt pin, GP₂, to detect the start of a transmission.

Both μ Cs have a flexible oscillator structure for use with a crystal, a resonator, or an external clock signal. Both parts of the figure show the μ C using an external crystal as the clock source. Because these devices are fully static, the main controller can provide the clock source to the μ C. By turning the clock source on and off as necessary, the main controller can further decrease overall power consumption. This method also allows control of the speed at which the algorithm runs, which is proportional to the sample rate of the system.

To fully implement the μ C as an ADPCM-encoder/decoder peripheral requires firmware to implement the serial or parallel interface, such as listings 1 and 2, and a main routine to tie everything together. The main controller is responsible for sampling the incoming audio waveform, storing and retrieving the ADPCM codes from nonvolatile memory, and then playing the resulting samples. The main controller feeds samples or ADPCM codes to the μ C and then reads the resulting ADPCM codes or samples from the μ C.

The listings are available for down-

Figure 1



Using the simplified ADPCM algorithm, you can now implement audio capabilities in μ Cs by offloading the encoding and decoding tasks onto the parallel (a) or serial (b) μ C as if it were a peripheral.

LISTING 2—ADPCM DECODER

```
// This routine also uses the IndexTable and StepSizeTable from Listing 1
signed int ADPCMDecoder( char code )
{
    predsamp = state.prevsamp;           // Restore previous values
    index = state.previndex;

    step = StepSizeTable[index];         // Find quantizer step size

    diffq = step >> 3;                   // Inverse quantize ADPCM code into a
    if( code & 4 )                         // diff. using the quantizer step
        diffq += step;
    if( code & 2 )
        diffq += step >> 1;
    if( code & 1 )
        diffq += step >> 2;

    if( code & 8 )                         // Add the difference to predicted sample
        predsamp -= diffq;
    else
        predsamp += diffq;

    if( predsamp > 32767 )                 // Check if overflow of new predicted sample
        predsamp = 32767;
    else if( predsamp < -32768 )
        predsamp = -32768;

    index += IndexTable[code];           // Find new quantizer step size

    if( index < 0 )                       // Check if overflow of new quantizer step
        index = 0;
    if( index > 88 )
        index = 88;

    state.prevsamp = predsamp;           // Save values for next iteration
    state.previndex = index;

    return( predsamp );                  // Return new sample
}
```

loading from at EDN's Web site, www.ednmag.com. At the registered-user area, go into the Software Center to download the file from DI-SIG, #2292. (DI #2292)

To Vote For This Design,
Circle No. 335

Cheap PWM IC makes synchronous gate driver

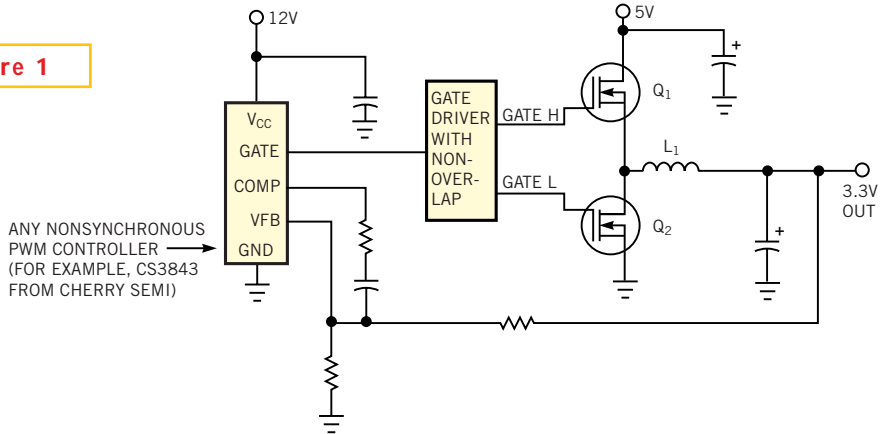
Dimitry Goder, Switch Power Inc, Campbell, CA

A system with a μP , memory, and peripherals usually requires several power-supply voltages. Designers typically use local switching regulators to produce the desired voltage rails. One of the most common topologies, the synchronous buck regulator, converts a 5 or 12V bus to some other, lower voltage. This approach has gained vast acceptance, thanks to its relative simplicity and high conversion efficiency. Specialized synchronous buck controllers are available, but you generally pay a premium for these ICs. Meanwhile, many inexpensive, general-purpose PWM controllers are available, but they require you to implement synchronous rectification with discrete circuitry. The implementation is a difficult task, because it involves building two out-of-phase, mutually exclusive gate drivers. The block diagram in **Figure 1** shows how to use any general-purpose PWM controller (for example, the CS3843 from Cherry Semiconductor in East Greenwich, RI) to configure a synchronous buck regulator.

The most critical aspect of the driver design is nonoverlap timing, which prevents simultaneous high states at gates H and L, thus eliminating the simultaneous turn-on of Q_1 and Q_2 , with resulting shoot-through currents. **Figure 2** shows details of the driver block. The input signal passes through two inverters, IC_{1A} and IC_{1B} , to generate the Gate H signal in phase with the input. The complementary follower, Q_1 and Q_2 , forms a current amplifier to provide sufficient drive for the top MOSFET. Meanwhile, IC_{1D} inverts the input, and Q_3 and Q_4 amplify the signal to drive the bottom MOSFET.

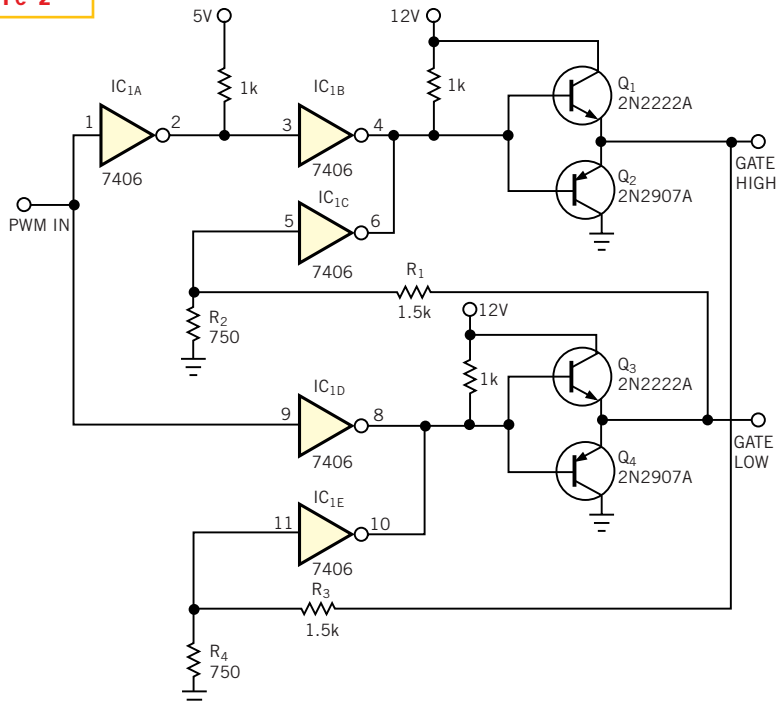
IC_{1C} and IC_{1E} provide the nonoverlap function. Each of the inverters ensures a low state at the corresponding gate-drive output until the other driver's output falls below a certain threshold. This

Figure 1



All it takes to drive high- and low-side MOSFETs from a PWM controller is a simple driver with 180° out-of-phase outputs, but you must beware of simultaneous MOSFET conduction.

Figure 2

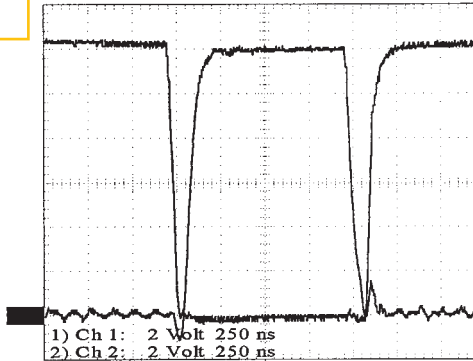


A sprinkling of TTL gates and four bipolar transistors provide efficient, nonoverlap drive to the two MOSFETs in a synchronous rectifier.

function prevents simultaneous high states, regardless of the circuit delays and the type of MOSFETs you use. The TTL switching threshold and the values of R_1 , R_2 and R_3 , R_4 determine the nonoverlap threshold. The threshold is approximately 3V for the circuit in **Figure 2**, but you can easily adjust it for other values.

Creating a high-side driver requires a bias voltage higher than the input voltage. A 12V line is commonly available, so you can use it to power the driver. The outputs of IC_{1B} and IC_{1D} must swing between ground and 12V; you thus need a 7406 open-collector inverter with a

Figure 3



At least 60 nsec separates the turn-on drive from the upper and lower drivers, thus preventing simultaneous MOSFET conduction.

high-voltage capability. If 12V is unavailable, you can use charge-pump circuitry to double the input voltage. The driver shows excellent performance, with 60-nsec nonoverlap time, superior to that of many available ICs. **Figure 3** shows the two gate drivers' outputs, with each driver switching an IRL3103 n-channel MOSFET, a good choice for a 10A converter. The total cost of the driver does not exceed 30 cents. (DI #2306).

To Vote For This Design,
Circle No. 336

Circuit detects total on-time

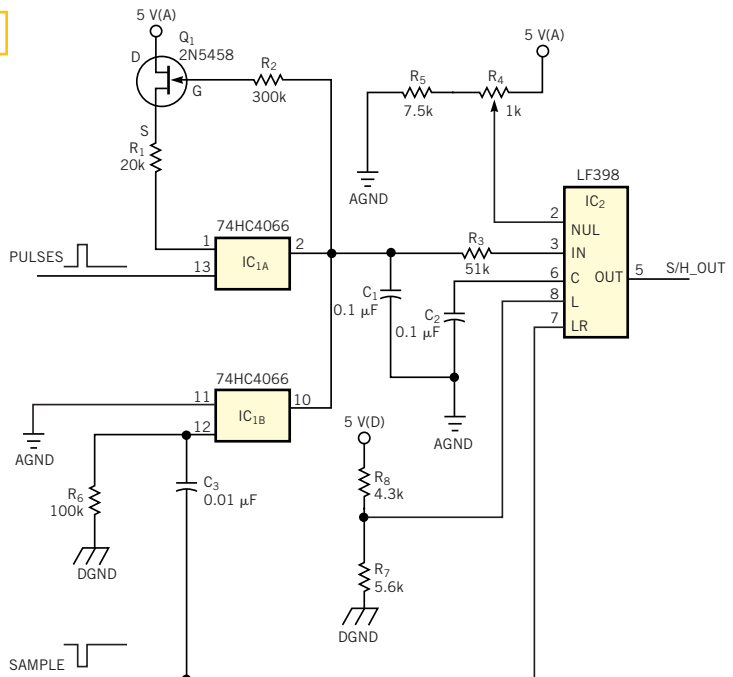
Richard Nachazel, Vista Electronics Co, Ramona, CA

The circuit in **Figure 1** provides a voltage that is directly proportional to the sum of the total on-times for a number of positive pulses that occur during the interval between sample signals. When a positive pulse arrives at the control input to switch IC_{1A} , a constant-current source charges capacitor C_1 through IC_{1A} for the duration of that pulse and for each subsequent pulse. The constant-current source comprises Q_1 ; R_1 ; C_1 ; and R_2 , which drives Q_1 's gate at C_1 's potential. C_1 and C_2 should be noninductive and have low dielectric absorption.

When you apply a low signal to the Sample line, the sample/hold circuit (IC_2) latches the analog voltage on C_1 . You adjust R_4 for 0V at the output with no pulses at the input. The trailing edge of the Sample signal switches IC_{1B} on, discharging C_1 to an initial 0V potential. You can connect unused analog switches in parallel with IC_{1B} to speed the discharge. Higher supply voltages also improve the circuit's speed and performance. (DI #2300).

To Vote For This Design,
Circle No. 337

Figure 1



Two ICs, a FET, and a handful of components form a pulse-width-to-voltage converter.

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 02158

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.