design ideas

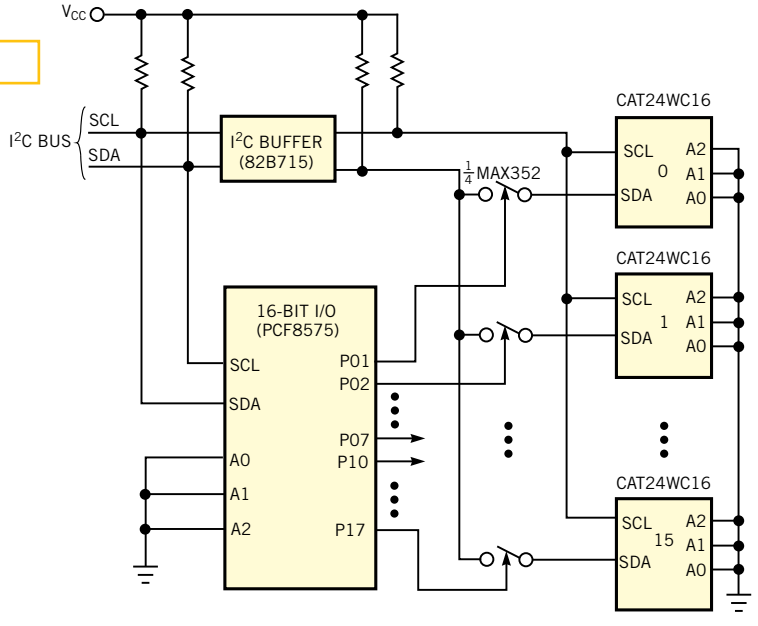
Edited by Bill Travis and Anne Watson Swager

Circuit gang-programs EEPROMs over I²C bus

Denisa Stefan, Catalyst Semiconductor, Sunnyvale, CA

You use the fully controlled circuit in **Figure 1** to parallel-program two-wire serial EEPROMs via the I²C bus. Gang programmers must address all memory devices during a write operation. To verify the memory contents, however, the system must address only one memory at a time during read operations. Therefore, the system in **Figure 1** addresses the memory devices either in parallel or one at a time. Information transfer between devices connected to the I²C bus system requires a SDA (serial-data) and SCL (serial-clock) signals. A device connected to the bus can operate as a transmitter or a receiver. A master device initiates a data transfer on the bus, generates clock signals, and terminates the transfer. The master addresses a slave device. To connect devices on an I²C multimaster bus, the SDA and SCL lines must be bidirectional and must connect to a positive supply voltage through pullup resistors.

Figure 1



An I²C expander and analog switches provide gang programming and serial read access for multiple EEPROMs.

In I²C-bus addressing, the first byte after a Start condition determines the slave that the master selects. A slave address is seven bits long and usually comprises a fixed part and a programmable part. The eighth bit, or LSB, determines the direction of the transfer, either read or write. The programmable part of the slave's address allows you to connect the maximum possible number of identical devices to the I²C bus. This number depends on the number of address-input pins the I²C device has. In **Figure 1**, the serial-data line, SDA, connects to each CAT24WC16 EEPROM via bidirectional Maxim (www.maxim-ic.com) MAX352 quad SPST analog switches. The switches derive their control from a 16-bit Philips (www.semiconductors.philips.com) PCF8575 I/O expander for the I²C bus. The clock line, SCL, connects to all memory devices. For driving the large capacitive loads required, a Philips 82B715 I²C-bus extender serves as a buffer. The software sequence for parallel writing to all memory devices is to set the port pins by writing to the PCF8575 to command closing all switches and then send an I²C-bus command to write to the CAT24WC16 EEPROMs.

The software flow for reading the contents of one memory device is to set the port pins by writing to the PCF8575 to close the switch associated with the memory to read, set all the other switches to open, and then send an I²C command to read the selected CAT24WC16 EEPROM.

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

LFSR provides encryption

Antonella Di Lillo and Giovanni Motta, Brandeis University, Waltham, MA

LFSRs (linear-feedback shift registers) find extensive use in cryptography. For example, the cryptographic algorithms in the GSM (Global System for Mobile communications) mobile-phone system rely on the use of LFSRs. An LFSR comprises a register containing a sequence of bits and a feedback function. In general, this function is an XOR (exclusive-OR) operation on certain bits in the register. The list of these bits is a "tap sequence." You use an LFSR to generate a pseudorandom sequence of bits that undergo an XOR operation. The XOR result then connects to the input of the LFSR. Repeating the process at the decoder side returns the original sequence of bits. **Listing 1** presents the encryption and decryption process. To generate a pseudorandom sequence, you load the register with a nonzero content, and the software then computes the XOR of the taps and shifts all bits in the register one bit to the left. Finally, the routine inserts the results of the XOR operation in the rightmost position (**Figure 1**). The program adds this 1-bit result to the sequence and repeats the procedure to generate other bits.

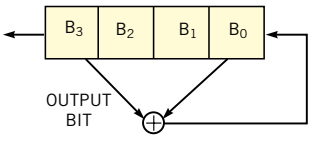


Figure 1

A linear-feedback shift register, combined with an XOR operation, is ideal for encryption.

LFSRs are well-suited to hardware implementations, but their use in software programs unfortunately often suffers from inefficient implementations. LFSRs with few taps, or sparse LFSRs, are easier to use because you need calculate the XOR of only a few bits. On the other hand, for cryptographic purposes, you must avoid sparse polynomials because the resulting algorithms are easy to break. The C program in **Listing 1** has the advantage of implementing the XOR of a 32-bit integer with an efficient algorithm, thus making an efficient way to implement the software of an LFSR. The program encrypts a standard input into a standard output one character at a time. You use an LFSR to generate eight random bits at a time, and the routine XORs the random bits to the current character. The encryption key is a nonzero integer that you use as the initial status of the register. The key is the same for both encoding and decoding. The variable taps represent the 32 binary coefficients of a primitive polynomial of degree 31. Several other choices are possible; **Listing 1** suggests five of them in the comments. You can safely remove the code that describes the number of characters encrypted and the running time to make the program even smaller. You can download the software from the Web version of this article at www.ednmag.com.

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

LISTING 1—LFSR ENCRYPTION AND DECRYPTION

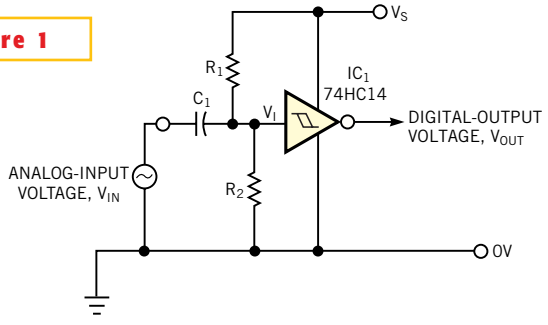
```
//  Usage: LSFR <Key>
//  Key is a non-zero 32 bit integer
//  Alternative taps : 0xBF75CC1F, 0xC679E105,
//    0x844EC703, 0xBE4F7253, 0xDF3B0B11,...
#include <stdio.h>
#include <time.h>

//  XOR of the bits in a 32 bit integer
int xor_32(int a) {
  a ^= a >> 1;
  a ^= a >> 2;
  a ^= a >> 4;
  a ^= a >> 8;
  a ^= a >> 16;
  return(a & 0x1);
}

int main(int argc, char *argv[]) {
  int LSFR, c_char, i, cnt = 0, kcps;
  int taps = 0xE04D11E7;          // Polynomial
  clock_t start;
  double interval;

  if(argc < 2) {                  // Usage
    fprintf(stderr, "Usage:\n");
    fprintf(stderr, "\t%s <Key>\n", argv[0]);
    return -1;
  }
  if(!(LSFR = atoi(argv[1]))) { // LSFR Status
    fprintf(stderr, "Key must be non zero\n");
    return -1;
  }

  start = clock();
  while((c_char = fgetc(stdin)) != EOF) {
    for(i=0; i<8; i++)
      LSFR = (LSFR<<1) | xor_32(LSFR & taps);
    fputc((LSFR & 0xFF) ^ c_char, stdout);
    cnt++;
  }

  interval = (double)(clock()-start) /
                  CLOCKS_PER_SEC;
  kcps = (int)(cnt / interval);
  fprintf(stderr, "Encoded %d chars ", cnt);
  fprintf(stderr, "in %f seconds ", interval);
  fprintf(stderr, "(%d char/sec)\n", kcps);
  return 0;
}
```

Resistor network extends Schmitt trigger's reach

Anthony Smith, Scitech, Biddenham, UK

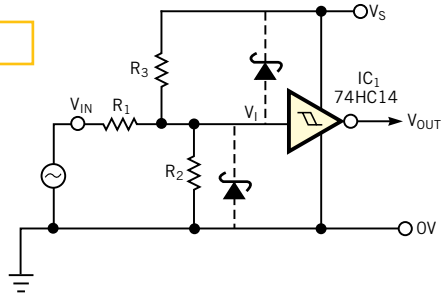
THE CIRCUIT IN **Figure 1** shows a familiar technique for converting a low-level analog signal to digital form. Resistors R_1 and R_2 set the quiescent dc level at the Schmitt inverter's input to a value roughly equal to the midpoint of the hysteresis band. Capacitor C_1 removes dc content from V_{IN}, such that the Schmitt trigger's input signal, V_I, centers itself on the midhysteresis level. Provided that V_{IN} is large enough to cross IC_1's threshold level, the output signal, V_{OUT}, provides a faithful digital representation of V_{IN}. Unfortunately, the circuit suffers from several drawbacks. The presence of C_1 makes it impossible for IC_1 to switch at specifically defined dc levels on V_{IN}. Furthermore, for low-frequency waveforms, C_1 must be extremely large to prevent unwanted signal attenuation. Also, if V_{IN} is of random period or is asymmetrical with time (for example, a pulse train with low duty cycle), the signal at V_I will not swing symmetrically about the quiescent dc level and may fail to cross one of IC_1's thresholds. You can solve all these problems by replacing C_1 with a resistor, as in **Figure 2**.

Figure 1



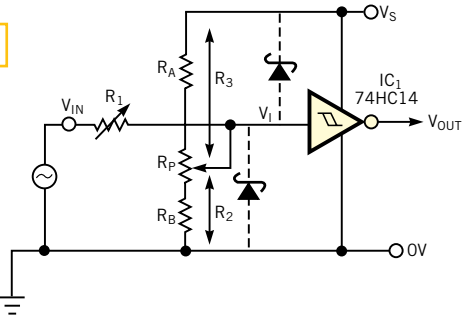
This Schmitt-trigger circuit is useful for converting an ac signal to digital form.

Figure 2



Eliminating the input capacitor avoids problems with asymmetrical input waveforms.

Figure 3



The potentiometer networks solve the problem of large spreads in component values.

In **Figure 2**, R_1 and the parallel combination of R_2 and R_3 act as an attenuator that allows IC_1 to switch at specific, user-defined dc levels that may be much greater than IC_1's switching thresholds. Furthermore, R_2 and R_3 introduce an offset that allows V_{IN}'s lower threshold to be negative if required. R_1 and R_2 relate to R_3 as follows:

$$R_1 = \frac{R_3(V_{TL}V_P - V_{TU}V_N)}{V_S(V_{TU} - V_{TL})},$$

$$R_2 = \frac{R_3(V_{TL}V_P - V_{TU}V_N)}{V_S(V_P - V_N + V_{TL} - V_{TU}) + V_{TU}V_N - V_{TL}V_P},$$

where V_S is the supply voltage; V_P and V_N are the required upper and lower V_{IN} thresholds, respectively; and V_{TU} and V_{TL} are the Schmitt trigger's upper and lower switching thresholds. By measuring V_{TU} and V_{TL} for a given Schmitt inverter and selecting a suitable value for R_3, you can calculate the corresponding values of R_1 and R_2. The circuit accommodates almost any values of V_P and V_N. The only restriction is that the hysteresis ($V_P - V_N$) is sufficiently larger than IC_1's hysteresis ($V_{TU} - V_{TL}$); otherwise, the equations can yield negative resistor values. If IC_1 is a CMOS device (for example, 74HC14, 74AC14, 4093B, or 40106B), you can use large resistances, thus ensuring high input impedance.

For cases in which it is inconvenient to measure the exact values of V_{TU} and V_{TL}, you can replace R_1 and R_2 with variable resistors to accommodate the worst-case spread in V_{TU} and V_{TL}. However, because R_2 and R_3 have a large influence on R_1, the spread of values you need for R_2 results in a broad variation in the R_2-R_3 parallel combination and results in an even broader spread of values for R_1. Replacing R_2 and R_3 with a potentiometer network, as in **Figure 3**, provides a solution to the "spread" problem. Because R_2 varies with R_3, the spread in the R_2-R_3 parallel combination, and hence in R_1, is narrower. This arrangement results in some fairly onerous equations relating the variables. However, you can simplify matters by observing that for a particular CMOS Schmitt inverter, each of its thresholds is a constant fraction of the supply voltage, V_S. Therefore, you can

define $V_{TU}=UV_S$ and $V_{TL}=LV_S$, where U and L are the respective fractions. This simplification results in the following equations:

$$R_1 = R_2 \frac{V_S(L-U)+V_P(1-L)+V_N(U-1)}{V_S(U-L)},$$

$$R_2 = R_X \frac{LV_P-UV_N}{V_S(L-U)+V_P-V_N}, \text{ and}$$

$$R_3 = R_X-R_2.$$

The design procedure is to select the desired values for V_S, V_P, and V_N and then to calculate R_1, R_2, and R_3 in terms of R_X for the worst-case spread in U and L. You can then scale the values of R_1, R_2, and R_3 accordingly. As an example, assume that you need to set V_P at 6V and V_N at −7.5V using a 74HC14 operating from a 5V supply. Although slight differences exist between manufacturers, the "typical" spread in thresholds for the 74HC14 on a 5V rail yields the following values: U=0.5 (minimum) to 0.7 (maximum), and L=0.2 (minimum) to 0.44 (maximum). These values are subject to the restrictions on hysteresis: (U−L)=0.09 (minimum) to 0.5 (maximum). You can intuitively see that R_1 is at a maximum when IC_1's hysteresis is small and the parallel combination of R_2 and R_3 is large. This scenario occurs when IC_1 has a narrow hysteresis band centered roughly on $V_S/2$. In this example, R_1 is a maximum of $7.25R_X$ when L=0.435 and U=0.525. Conversely, R_1 is at a minimum when IC_1's hysteresis is large and the parallel combination of R_2 and R_3 is small. This scenario occurs when L=0.2 and U=0.7, resulting in $R_1=1.067R_X$. The range of potentiometer R_P must allow you to set the quiescent value of V_I anywhere from the minimum midhysteresis band level (occurring when L and U are both minima), to the maximum midhysteresis level (occurring when L and U are both maxima). In this example, the values are $R_2=0.4125R_X$ and $R_3=0.5875R_X$ (when L=0.2 and U=0.5) and $R_2=0.6467R_X$ and $R_3=0.3533R_X$ (when L=0.44 and U=0.7). Assuming that you use resistors with ±1% tolerance and potentiometers with ±10% tolerance, you can accommodate the required spread in R_2 and R_3 with an adequate margin by making $R_A=1.1$ kΩ, $R_P=1$ kΩ, and $R_B=1.3$ kΩ. The corresponding spread in R_1 (including the tolerance in R_X itself) is 3.495 to 25.549 kΩ. You can obtain this range by using a parallel connection of a 50-kΩ potentiometer and a 51-kΩ resistor that is in series with a 3.3-kΩ resistor.

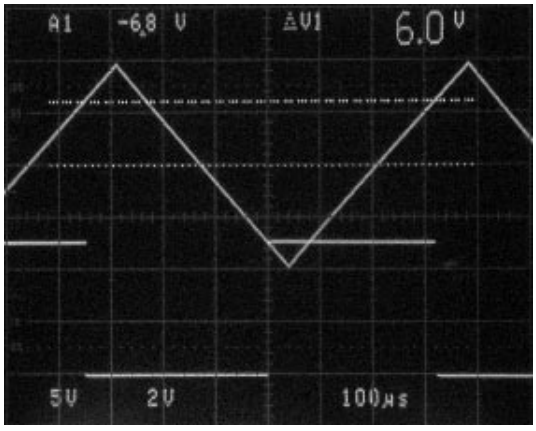


Figure 4 These waveforms indicate clean hysteretic switching with a triangle-wave input.

The oscilloscope screen in **Figure 4** illustrates the performance of the example circuit, in which V_{IN} is a ±10V triangle wave. By adjusting the two potentiometers in turn, we made the output waveform switch when $V_{IN}=6$V and −7.5V. Despite the interaction between the potentiometers, you can fairly easily (with a little patience) set the thresholds. Although the circuit is not intended for precision applications, it does extend the range of the garden-variety Schmitt inverter and allows you to implement positive and negative thresholds of several tens or even hundreds of volts. Moreover, the circuit allows V_N to be positive, provided that V_P is sufficiently greater than V_N to avoid negative resistance values. You can obtain operation of greater than 10-MHz frequency if you use suitable devices for IC_1. The 74AC14 or 74HC14 yield response times of just a few nanoseconds with a rail-to-rail output. For best high-frequency performance, use low resistor values, a shunt trimmer capacitor across R_1 to provide compensation, or both. Finally, use Schottky clamp diodes as in **Figure 3** to protect the inputs of IC_1 from overvoltage conditions.

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

Routine yields fast bit reversing for DSP algorithms

Mohammed Aziz, University of Leeds, UK

If you need efficient real-time performance in DSP applications, you need an efficient bit-reversing routine. For several FFT programs, data permutation can take 10 to 50% of the computation time, depending on the input-data dimensions and length. The idea behind bit reversing is to shuffle the data by flipping the address bits around the mid-

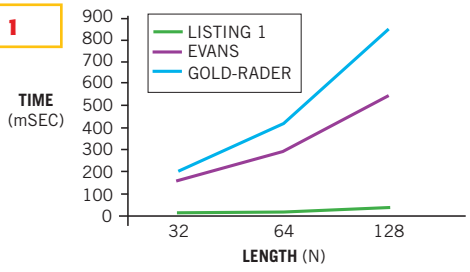
TABLE 1–BIT-REVERSING RUNTIME RESULTS

Length (N)	Listing 1 (msec)	Evans (msec)	Gold-Radar (msec)
32	10	159	200
64	18	294	418
128	34	549	847

dle of the address length so that if the data length is N=16, four bits from 0000 to 1111 represent the address. You achieve data shuffling by swapping the address bits around the middle so that $B_3B_2B_1B_0$ becomes $B_0B_1B_2B_3$, which represents the new data location. Note that this operation is not byte flipping unless the input-data length happens to be N=255 elements. Look-up-table techniques are inefficient, because the input data may be very long, and memory space is limited. For real-time DSP applications, you bit-reverse a data array by swapping each position in the data array with the position of its corresponding bit-reversed address by using DSP architectural features. You implement the method shown in **Listing 1** using the SHARC DSP chip.

The routine uses the data-memory segment and program-memory segments for input and output. This feature of the DSP-memory architecture allows reading from the input array "in-arry" and writing to the bit-reversed output array "out-arry" in parallel to double the speed. It then uses circular buffers and indirect addressing to go through the N elements, lending itself to simple and straightforward data moving. Existing techniques, such as Evans and Gold-Rader algorithms, do data checking and bit reversing before moving the data, thereby incurring overhead. **Table 1** and **Figure 1** give the runtime results for the different algorithms (**Listing 1**, Evans, and Gold-Rader) simulated on a 40-MHz SHARC DSP (ADSP-21060) from Analog Devices (www.analog.com). The runs represent three lengths of 32, 64, and 128 elements and involve 10,000 iterations each in simulating the 2-D case.

Figure 1



The routine in Listing 1 produces markedly faster results than other algorithms.

LISTING 1—BIT-REVERSE ALGORITHM

```
#define  BRmod      //bit-reverse of N/2, N-length of input data
#define  in-arry    //bit-reversed address of input data location
#define  out-arry   //address of output array
#define  N          //length of input data

BR-Algorithm:
bit set mode1 BR0;                    //this allows BR-mode
b0=in-arry;  l0=0;  m0=BRmod;         //this is circular buffers
b8=out-arry;  l8=N;  m8=1;
r0=dm(i0,m0);
lcntr = (N-1), do br until lce;       //this is a loop counter
br: r0=dm(i0,m0), pm(i8,m8)=r0;
pm(i8,m8)=r0;
bit clr mode1 BR0;                    //this clears the BR-mode
```

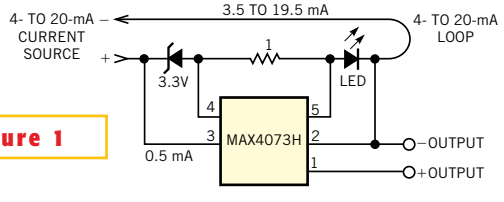
Is this the best Design Idea in this issue? Vote at www.ednmag.com.

A 4- to 20-mA loop needs no external power source

Shyam Tiwari, Sensors Private Ltd, Gwalior, India

THE SIMPLE CIRCUIT in **Figure 1** uses a low-current-drain MAX-4073H amplifier to sense the current flowing through a 4- to 20-mA loop. The circuit senses the current through a 1Ω resistor with a fixed gain of 100 and uses no battery or dc power supply. The low current drain of the amplifier (0.5 mA) enables the circuit to tap its power from the 4- to 20-mA loop to power the amplifier chip. Note that the current flowing in the amplifier's power-supply Pin 3 (nominally 0.5 mA but may vary slightly) is not part of the sensing loop. It forms a negative offset in the measurement and is not a serious problem. To make this current nearly constant, a 3.3V zener diode and an LED in series with the sensing resistor form a voltage drop of 4 to 4.5V across pins 2 and 3 of the amplifier chip. The amplifier works well over 3 to 28V, so this 4 to 4.5V power-supply range presents no problems.

Figure 1



A current-sensing circuit derives its power from the 4- to 20-mA current loop.

The output of the amplifier is linear from 350 to 1950 mV for 4 to 20 mA through the loop. The measurement meter at the output must not draw more than 5 µA from the output for 1% full-scale measurement accuracy. The LED shows visual intensity variation for changing current in the loop. Its main purpose is to raise the voltage by approximately 1V across the sense resistor with respect to the power-supply return Pin 2 of the amplifier. This increased voltage gives better common-mode performance to the amplifier against common-mode noise in the sensing resistor and prevents the amplifier from saturating near the power-supply rails.

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