Spreadsheets for Digital Circuits

Nuts & Volts Magazine recently had two articles about linear feedback shift registers — shift registers with some of their outputs fed back to the input — for generating CRC check bits or random numbers. Short of actually building these (and other digital) circuits, it's sometimes hard to check that they do what they claim to do. Here is an interesting and simple technique for simulating these circuits on a computer, using a spreadsheet program.

It doesn't have to be an expensive or powerful spreadsheet program — I will use Excel for the examples below, but sometimes old versions of Lotus 123, or shareware programs like As-Easy-As, work even better.

The Inverter

Fig. 1 shows the basic idea. Looking at an inverter, such as a 7404, in the TTL manual, you'll often see the inputs and outputs labelled as *In* and *Out*, or perhaps *A* and *B*. But simply label them *A1* and *B1*, and right away you get the idea: just use cell *A1* in the spreadsheet for the input, and cell *B1* for the output (nothing magic about these two cells; you could use any two cells in the spreadsheet.). Now all you have to do is put the right formula into cell *B1* so it's always the opposite of *A1*, and you're done.

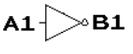

Fig. 1. A Simple Inverter

For the inverter in Fig. 1, if the input is a 0 then the output is a 1, and if the input is 1 then the output is 0 (we have to use 0 and 1; it won't work if you try to use H and L, or True and False, instead).

There are two ways to do this. The simplest is to write this as the simple formula

$$Output = 1 - Input$$

In other words, in the spreadsheet, cell B1 should be equal to 1 minus whatever is in cell A1: 1 minus 0 is 1, whereas 1 minus 1 is 0. The formula then becomes

$$B1 = 1 - A1$$

In Excel, you'd type =1–A1 into cell B1 (with the equal sign), whereas in most other spreadsheets it would be just 1–A1. Check the exact format for whichever spreadsheet you are using.

Another way is to use the spreadsheet's IF function, like this:

$$B1 = IF(A1=0,1,0)$$

The parentheses contain three arguments: a test, what to do if the test is correct, and what to do if it isn't. In this case, it says "if A1=0 then make this cell equal to 1, otherwise make it 0." (For obvious reasons, I prefer to use the "one minus" approach!)

Fig. 2. shows some other simple gates. Let's do them one at a time.

The AND Gate

For the AND gate at top left, the truth table (which describes how it works) is

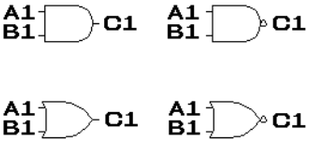

Fig. 2. Simple digital gates

A1	B1	C1
0	0	0
0	1	0
1	0	0
1	1	1

from which we can see that the output is just the product of the two inputs — a simple multiplication. So the formula to put into cell C1 is A1*B1 (the * means multiply).

Some spreadsheets actually have functions for doing simple logical operations. Excel insists on using words like TRUE and FALSE, so that is not very useful to us, but 123 and others use ones and zeroes. The corresponding syntax for this formula in 123 and some others would be +A1#AND#B1. Don't bother — A1*B1 is much simpler.

The NAND Gate

In the NAND gate at the top right, the output is similar to that of the AND, but inverted. The equation

C1 = 1 – (A1 * B1)

should just about do it, since the "1–" part does the inversion just as in the inverter.

The OR Gate

The truth table for the OR is

A1	B1	C1
0	0	0
0	1	1
1	0	1
1	1	1

The easiest way to implement this is with an IF, like this:

C1 = IF(A1+B1=0,0,1)

which says that if the two inputs add up to 0 (which means they are both 0) make the output a 0, otherwise make it a 1.

The NOR Gate

The NOR is just the opposite of an OR:

A1	B1	C1
0	0	1
0	1	0
1	0	0
1	1	0

Here the easiest is to again use an IF:

C1 = IF(A1+B1=0,1,0)

which is the same as the OR, except that it flips the 0 and 1 in the output.

XOR - The Exclusive OR

Fig. 3 shows the Exclusive OR gate, which has the truth table

A1	B1	C1
0	0	0
0	1	1
1	0	1
1	1	0

The word *exclusive* means that the output is a 1 if *either* A1 or B1 is 1, *but not both*.

If you use the spreadsheet's IF statement, the IF reads

IF(A1=B1,0,1)

which just means that if the two inputs are the same then the output is 0; if they are not the same, then the output is 1.

Alternatively, you could also write

C1=ABS(A1–B1)

which tells the spreadsheet that the output is the difference between A1 and B1; in case A1-B1 is –1, then the ABS function just changes it into +1.

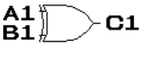

Fig. 3. The Exclusive OR gate

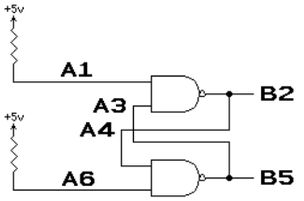

Fig. 4. Flip-flop made of gates

Handling Flip-Flops

Flip-flops are interesting because they create some special problems that require unique solutions.

Fig. 4 shows a simple flip-flop made from two NAND gates. As you know, flip-flops have two outputs, normally labelled Q and $\overline{Q}$ (which is pronounced Q NOT or NOT Q), which are always opposites of each other. We say that the flip-flop is set if Q is a 1 (while $\overline{Q}$ is 0), and the flip-flop is reset if Q is a 0 (while $\overline{Q}$ is a 1). If B2 is Q and B5 is $\overline{Q}$ in Fig. 4, then we can set the flip-flop by temporarily grounding input A1, or reset it by temporarily grounding input A6. (The two resistors make both inputs normally 1.)

Fig. 5 shows the spreadsheet for this circuit as it would look in Excel; on the left are the formulas (Excel shows them when you press CTRL and ‘, the reverse apostrophe at the top left of the keyboard), while the right shows what happens.

The formulas at the left follow the circuit almost exactly. For example, cells A1 and A6 are both equal to 1, cell B2 is the NAND of cells A1 and A3, and so on. The problem shows up at the right — Excel inserts a bunch of arrows linking the cells.

The catch is that the circuit uses circular reasoning: A3 depends on B2, which depends on A4, which depends on B5, which depends on A3, which... The actual outputs (i.e., whether the flip-flop is set or reset) depend on timing, and what happened most recently.

Excel doesn’t know what to do with this. It turns out that Excel is too *smart* — older spreadsheets, such as 123, will do what you want as long as you press the “recalculate” key a few times each time you change an input, whereas Excel won’t.

To handle this, we have to bring time into the spreadsheet — *before* and *after*. Common JK or Type D flip-flops have a clock input. Think of the inputs as they existed *before* the clock pulse, and the outputs as they exist *after* the clock pulse. The trick for handling flip-flops in a spreadsheet is to bring in time by using the sheet’s rows. For example, if row 1 describes things before a clock pulse, then row 2 describes things after it. And if there is another clock pulse afterward, then row 3 describes what happens after that, and so on. In other words, time goes down in the sheet.

Mod 3 Counter

The best way to show this is with an example. Fig. 6 shows a synchronous mod-3 counter that counts 0..1..2..0..1..2.. and so on. At any instant of time, the flip-flops get certain inputs and provide certain outputs, but nothing happens until a clock pulse arrives. Let’s see how this is done.

Look at Fig. 7 as we fill in the spreadsheet formulas in the following sequence:

To begin, assume that the two flip-flops both start reset. Thus the two Q outputs should be 0,

	A	B
1	1	
2		=1-A1*A3
3	=B5	
4	=B2	
5		=1-A4*A6
6	1	

	A	B
1	1	
2		1
3	0	
4	1	
5		0
6	1	

Fig. 5. Excel spreadsheet for the flip-flop of Fig. 4

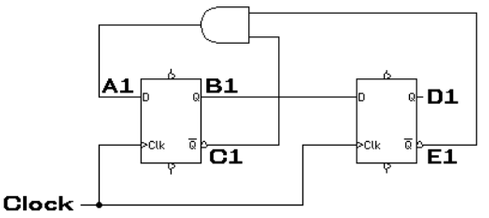

Fig. 6. Mod 3 synchronous counter

while the two $\overline{Q}$ should be their opposites. Thus we fill in

B1=0
C1=1–B1
D1=0
E1=1–D1

Next, the D input into the first flip-flop is just C1 and E1 ANDed together, so

A1=C1*E1

The top row of the sheet is now complete; it describes the starting or initial conditions.

Let's now assume that a clock pulse comes in. Row 1 describes the situation *before* the clock pulse, so let's use row 2 to describe the signals *after* the clock pulse.

At a clock pulse, the D input into a flip-flop determines whether the flip-flop will set or reset; that is, the Q output *after* the clock (in cell B2) will be the same as the D was *before* the clock (in cell A1). The same occurs with the second flip-flop, so we fill in

B2=A1
D2=B1

	A	B	C	D	E
1	=C1*E1	0	=1-B1	0	=1-D1
2	=C2*E2	=A1	=1-B2	=B1	=1-D2
3	=C3*E3	=A2	=1-B3	=B2	=1-D3
4	=C4*E4	=A3	=1-B4	=B3	=1-D4
5	=C5*E5	=A4	=1-B5	=B4	=1-D5
6	=C6*E6	=A5	=1-B6	=B5	=1-D6
7	=C7*E7	=A6	=1-B7	=B6	=1-D7
8	=C8*E8	=A7	=1-B8	=B7	=1-D8

	A	B	C	D	E
1	1	0	1	0	1
2	0	1	0	0	1
3	0	0	1	1	0
4	1	0	1	0	1
5	0	1	0	0	1
6	0	0	1	1	0
7	1	0	1	0	1
8	0	1	0	0	1

Fig. 7. Spreadsheet for the mod-3 counter of Fig. 6

At any time, the $\overline{Q}$ outputs will still be the opposites of Q, so we can either write

C2=1–B2
E2=1–D2

or else just copy the C and E columns all the way down the sheet (and let the spreadsheet update the row numbers).

Finally, the output of the gate (and D input to the first flip-flop) is still the ANDed signal from columns C and E, so we can either write

A2=C2*E2

or else again just copy the A column down the sheet.

At this point we're almost done — the second row is finished, and now all we have to do is copy the remaining entries from row 2 down as far as you want to go.

The result is then as shown in Fig. 7. To interpret it, look at the B and D columns; these are the two Q outputs from the flip-flops. D is most significant while B is least significant, so we see that the counter outputs are

00 = 0
01 = 1
10 = 2
00 = 0
01 = 1
and so on.

The CRC Generator

Now that you've gotten the idea, let's apply this stuff to two real problems. The first is a CRC generator which appeared in the July 2004 issue of Nuts & Volts in an article by James Antonakos on error detection in digital data. The circuit, shown in Fig. 8, uses a shift register with its output

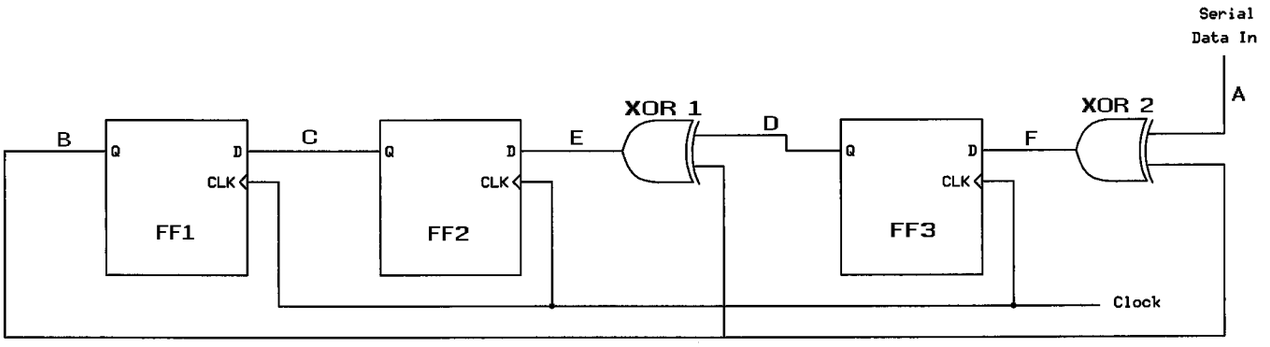

Fig. 8. The CRC circuit from July 2004 Nuts & Volts

	A	B	C	D	E	F
1	1	0	0	0	=ABS(D1-B1)	=ABS(A1-B1)
2	0	=C1	=E1	=F1	=ABS(D2-B2)	=ABS(A2-B2)
3	1	=C2	=E2	=F2	=ABS(D3-B3)	=ABS(A3-B3)
4	0	=C3	=E3	=F3	=ABS(D4-B4)	=ABS(A4-B4)
5	1	=C4	=E4	=F4	=ABS(D5-B5)	=ABS(A5-B5)
6	1	=C5	=E5	=F5	=ABS(D6-B6)	=ABS(A6-B6)
7	0	=C6	=E6	=F6	=ABS(D7-B7)	=ABS(A7-B7)
8	0	=C7	=E7	=F7	=ABS(D8-B8)	=ABS(A8-B8)
9	0	=C8	=E8	=F8	=ABS(D9-B9)	=ABS(A9-B9)
10	0	=C9	=E9	=F9	=ABS(D10-B10)	=ABS(A10-B10)
11	0	=C10	=E10	=F10	=ABS(D11-B11)	=ABS(A11-B11)
12		=C11	=E11	=F11		
13						
14	Data	FF1	FF2	FF3	XOR1	XOR2

	A	B	C	D	E	F
1	1	0	0	0	0	1
2	0	0	0	1	1	0
3	1	0	1	0	0	1
4	0	1	0	1	0	1
5	1	0	0	1	1	1
6	1	0	1	1	1	1
7	0	1	1	1	0	1
8	0	1	0	1	0	1
9	0	0	0	1	1	0
10	0	0	1	0	0	0
11	0	1	0	0	1	1
12		0	1	1		
13						
14	Data	FF1	FF2	FF3	XOR1	XOR2

Fig. 9. The spreadsheet for the CRC circuit of Fig. 8

fed back into the inputs through two XOR gates. Fig. 9 shows the resulting spreadsheet, with the formulas on the left and actual results on the right. The letters A through F in Fig. 8 tell us which column of the sheet has that signal.

Rather than explain every cell of the sheet, let's explain what the circuit does.

The general idea is this: When digital data is sent from one place to another, a CRC generator is used at the sending end to generate a check number (called a Cyclic Redundancy Check or CRC) which is then attached to the end of the outgoing data. At the receiving end, an identical CRC generator gets the data plus the CRC check digits from the sender, and tests whether there are any errors. If there are no errors, the receiver's CRC circuit outputs all zeroes; if the output is non-zero, then there was an error.

Let's assume that the outgoing data is the eight-bit number 10101100, and that the CRC check number will consist of three bits. The sender starts with the circuit of Fig. 8, and the initial condition that

all three flip-flops are reset to 000; this is the 000 in cells B1 through D1.

In order to generate the three-bit CRC number, the sender appends an extra three 000 bits to the end of the data stream, and sends it into the input labelled SERIAL DATA IN of Fig. 8 of its own CRC generator. This string of 10101100000 (a total of 11 bits) appears going down in column A of the spreadsheet. There is a clock pulse after each input bit, which brings us down one row of the spreadsheet.

At each row, the spreadsheet calculates the outputs of the three flip-flops (columns B through D) and the two XOR gates (columns E and F), and these signals determine what will happen in the next row, after the clock pulse.

At the very end, whatever is left in the three flip-flops (in row 12) is the CRC number which is appended to the outgoing data. In this particular case, the CRC check number is 011. So the actual data going from the sender to the receiver will be the original data 10101100, plus the extra 011 from its own CRC circuit. Thus the sender's CRC circuit gets the data plus three zeroes, whereas the receiver's CRC circuit gets the data plus 011 (assuming there were no errors).

Now let's assume that the receiver has the exact same CRC circuit, so we can use the same spreadsheet as before, with one change: whereas the bottom three bits in the A column of the sender were 000, in the receiver they will be 011. Try it in your spreadsheet — change the 000 in cells A9 through A11 to 011, and the three bits in row 12 become 000, signalling that no error occurred.

Now try changing one or more of the bits in column A to simulate an error in transmission; you will see that the last row is no longer 000. This tells the receiver that there was a transmission error in the data.

A Random-Number Generator

Fig. 10 shows a random-number generator taken from an article in the August 2004 Nuts & Volts.

Starting with an initial value called the *seed*, the circuit is supposed to cycle through some sequence of operations and output a string of random eight-bit numbers; these numbers are the contents of the eight flip-flops after every clock pulse.

A fairly simple, yet useful definition of "random" would be that a number is random if even an intelligent observer, knowing the past numbers in the sequence, can't guess what the next number will be.

The numbers output by most random-number generators are actually called *pseudo-random* because they are not truly random — they only look it. They are pre-

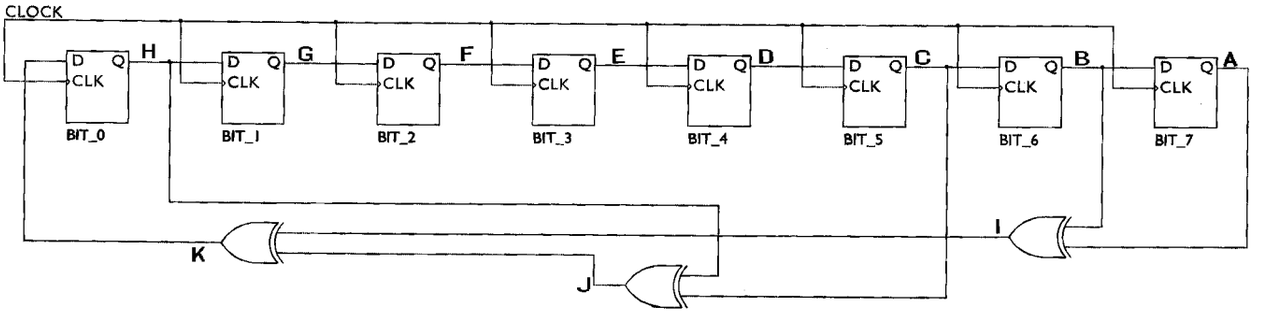

Fig. 10. Random number generator from Nuts & Volts Aug. 2004

	A	B	C	D	E	F	G	H	I	J	K
1	1	0	0	0	0	0	0	1	=ABS(B1-A1)	=ABS(C1-H1)	=ABS(I1-J1)
2	=B1	=C1	=D1	=E1	=F1	=G1	=H1	=K1	=ABS(B2-A2)	=ABS(C2-H2)	=ABS(I2-J2)
3	=B2	=C2	=D2	=E2	=F2	=G2	=H2	=K2	=ABS(B3-A3)	=ABS(C3-H3)	=ABS(I3-J3)
4	=B3	=C3	=D3	=E3	=F3	=G3	=H3	=K3	=ABS(B4-A4)	=ABS(C4-H4)	=ABS(I4-J4)
5	=B4	=C4	=D4	=E4	=F4	=G4	=H4	=K4	=ABS(B5-A5)	=ABS(C5-H5)	=ABS(I5-J5)
6	=B5	=C5	=D5	=E5	=F5	=G5	=H5	=K5	=ABS(B6-A6)	=ABS(C6-H6)	=ABS(I6-J6)
7	=B6	=C6	=D6	=E6	=F6	=G6	=H6	=K6	=ABS(B7-A7)	=ABS(C7-H7)	=ABS(I7-J7)
8	=B7	=C7	=D7	=E7	=F7	=G7	=H7	=K7	=ABS(B8-A8)	=ABS(C8-H8)	=ABS(I8-J8)
9	=B8	=C8	=D8	=E8	=F8	=G8	=H8	=K8	=ABS(B9-A9)	=ABS(C9-H9)	=ABS(I9-J9)

FF7 FF6 FF5 FF4 FF3 FF2 FF1 FF0 XOR3 XOR2 XOR1

Fig. 11. The “random” number generator - formulas

	A	B	C	D	E	F	G	H	I	J	K	L
1	1	0	0	0	0	0	0	1	1	1	0	81
2	0	0	0	0	0	0	1	0	0	0	0	2
3	0	0	0	0	0	1	0	0	0	0	0	4
4	0	0	0	0	1	0	0	0	0	0	0	8
5	0	0	0	1	0	0	0	0	0	0	0	10
6	0	0	1	0	0	0	0	0	0	1	1	20
7	0	1	0	0	0	0	0	1	1	1	0	41
8	1	0	0	0	0	0	1	0	1	0	1	82
9	0	0	0	0	0	1	0	1	0	1	1	5
10	0	0	0	0	1	0	1	1	0	1	1	B
11	0	0	0	1	0	1	1	1	0	1	1	17
12	0	0	1	0	1	1	1	1	0	0	0	2F
13	0	1	0	1	1	1	1	0	1	0	1	5E
14	1	0	1	1	1	1	0	1	1	0	1	BD
15	0	1	1	1	1	0	1	1	1	0	1	7B
16	1	1	1	1	0	1	1	1	0	0	0	F7
17	1	1	1	0	1	1	1	0	0	1	1	EE
18	1	1	0	1	1	1	0	1	0	1	1	DD

Fig. 12. The “random” number generator - results

dictable if you know the circuit or process that produced them. Still, for many applications, pseudo-random numbers are good enough.

The circuit is somewhat similar to the CRC circuit — it also has a shift register with some XOR gates to feed its outputs back into the input. But this time, there is only a clock input, no data input. As before, I’ve added letters to the diagram to show which signal will be in which column of the spreadsheet.

Figs. 11 and 12 show the formulas and results of this spreadsheet (I added an extra column at the right in Fig. 12 that converts the eight-bit binary output into hexadecimal).

The initial seed is what would be in the flip-flops at the start; it is at the top, in cells A1 through H1. Starting with any seed, the circuit should produce 255 num-

bers before it starts to repeat itself. In order to get the same sequence of numbers as the Nuts & Volts article, I used an initial seed of 10000001, which is hexadecimal 81. Using this as a starting value, the circuit outputs the following sequence (starting with row 2):

00000010 = hex 02
00000100 = hex 04
00001000 = hex 08
00010000 = hex 10
00100000 = hex 20
01000001 = hex 41
10000010 = hex 82
00000101 = hex 05

and so on.

The problem is that these numbers are not very random at all — the bits in consecutive numbers just shift to the left, with an occasional new 1 appearing at the right end. Hence if you know the previous numbers, you have a slightly better-than 50% chance of guessing the next number — you always know the first seven bits of it, and so all you need do is take a guess as to whether the last bit will be a 0 or a 1.

But that's another story. The point is that using a spreadsheet to analyze this circuit and look at the bits output at each clock pulse is very instructive — in this case it allows us to see that the circuit has a defect which would not be very noticeable if we just looked at the hexadecimal output.

One last comment

This is a simple and cheap technique. Even though I used Excel in my examples, you can use any of the free or shareware spreadsheet programs out there to do exactly the same thing. No exotic spreadsheet functions are needed.

About the Author

Peter Stark is Professor of Electrical and Computer Engineering Technology at Queensborough Community College, part of the City University of New York. He has been working in engineering and teaching for over 45 years, and has written several books, as well as close to a hundred magazine articles. He has a website at **www.cloud9.net/~pastark**