FEATURE

Designing with Logic, Part 2

Understanding hardware design with logic involves understanding binary arithmetic. This month we'll look at the basis of binary numbers.

Steve Rimmer

You can't really design logic circuitry from an electronic point of view... or, at least, attempting to do so is exceedingly frustrating. Traditional concepts of signals and amplitudes don't really apply to logic. In its place, one has to start thinking about things in terms of data.

A single logical state is usually pretty meaningless. Combined with other logical states, however, it may represent data and hence the true "signals" of logic. This takes some getting used to when one is thinking about how logic works, and considerable head scratching when one is trying to debug the stuff.

This month we're going to have a look at the basis of digital data, binary numbers. While a bit awkward unless you were born with sixteen fingers... and like to count on them... binary arithmetic is the fundamental key to understanding logic design. Without it, you'll probably drive yourself insane trying to design half adders and counters as if they were radios.

Count by Twos

Numbers of the sorts we're used to work in base ten. Computer numbers work in base two. We are predisposed to think of things which happen in clumps of ten, that is, ten things, groups of ten things, groups of groups of ten things and so on. This corresponds to the positions in a decimal number. The rightmost digit represents the number of things up to ten. The next to rightmost digit is the number of groups of ten, followed by the number of hundreds, or groups of groups of ten.

Because logic only has two states to concern itself with, rather than ten, it must deal with numbers either as base two... which is rather useless if you have more than two of something... or as some base which is an even power of two. The commonly used one is base sixteen, hexadecimal, although a lot of early logic design used octal, or base eight. In a real sense the logic doesn't care, and much of the numerical head bashing which goes on about logic design is for the convenience of logic designers.

Allowing that we have a series of logical states, like this:

```
0000
```

The rightmost state will represent the number of ones in the number, the next to rightmost state will be the number of twos, followed by the fours and the eights. These values are two raised to the power of zero, one, two and three respectively.

We can use this arrangement to represent actual numerical data... in this case, numbers from zero through fifteen. The numbers work out like this.

```
0000  0
0001  1
0010  2
0011  3
0100  4
0101  5
0110  6
0111  7
1000  8
1001  9
1010  10
1011  11
1100  12
1101  13
1110  14
1111  15
```

There's all sorts of significance to this table, some of which won't be apparent for a while. However, note that each of the state diagrams... binary numbers in computer terms... can be derived from the previous one by doing a binary addition of one to the previous value. Let's see how this works in state terms.

The binary value for eleven is

```
1011
```

If we were to add one to this, we would do the following in state terms.

```
 1011
+0001
```

Now, let's work through this, starting with the rightmost value. One plus one is two... in most parts of the universe... but two is an illegal value for a system which can only represent zeros and ones. As such, the result of this calculation is zero with carry. The next state would be oneplus zero plus the one represented by the carry. This would be zero, and again there would be a carry. The next state would be zero plus zero plus the one of the carry, for a grand total of one. The final, or ''high order'' state would be one plus zero with no carry. The resulting binary number would be

1100

which is, in fact, twelve.

Obviously, this is something which could be handled using gates. A binary adder is a simple logic array which accepts two binary numbers and produces a binary result. We'll discuss the design of such an array shortly.

There's another way of looking at the process of binary counting, one which is important in discussing counting circuits, a primary tenet of logic design. Consider that each state can be derived from the last by flipping individual logic lines, or bits. Beginning with zero, flipping the low order, or rightmost, state gives you one. Flip it again and it returns to zero and its carry flips the next state, giving you two. Flip it again and it becomes one, giving you three. Flip it again and it becomes zero, Its carry flips the next state, which also becomes zero. Its carry, it turn, flips the next state, giving you four.

Postulating a black box which behaves in this way, four such boxes would allow you to count in increments of one from zero through fifteen. Such boxes are, of course, the basic logic elements called flip-flops, of which much more will be said later.

Hard Design

Figure one illustrates a complete schematic for a four bit binary adder, this one pinched from a logic manual. If you study this thing for a moment you'll understand what it's up to... and that it's not a fierce as it appears.

If you look at any one of the four groups of gates at the left of the diagram, you'll pretty well see how a single bit of binary addition works. The NOR gates handle the addition and NAND gates handle the carry. If both inputs are one, the result of the addition must be to set the carry for this binary digit, passing the carry on to the next one. Otherwise, the result of the two input bits will be found by ORing them. If either is one, the result will be one.

The binary adder represents a typical example of logic design. We can analyse it here with the leisure of hindsight, but the approach to creating it will apply to any logic array which accepts a finite number of binary states as input and produces a finite number of states as output.

In order to create a binary adder from scratch, you would start with the problem of adding two single binary states together. We can represent them and the result of the as yet undesigned adder with truth tables.

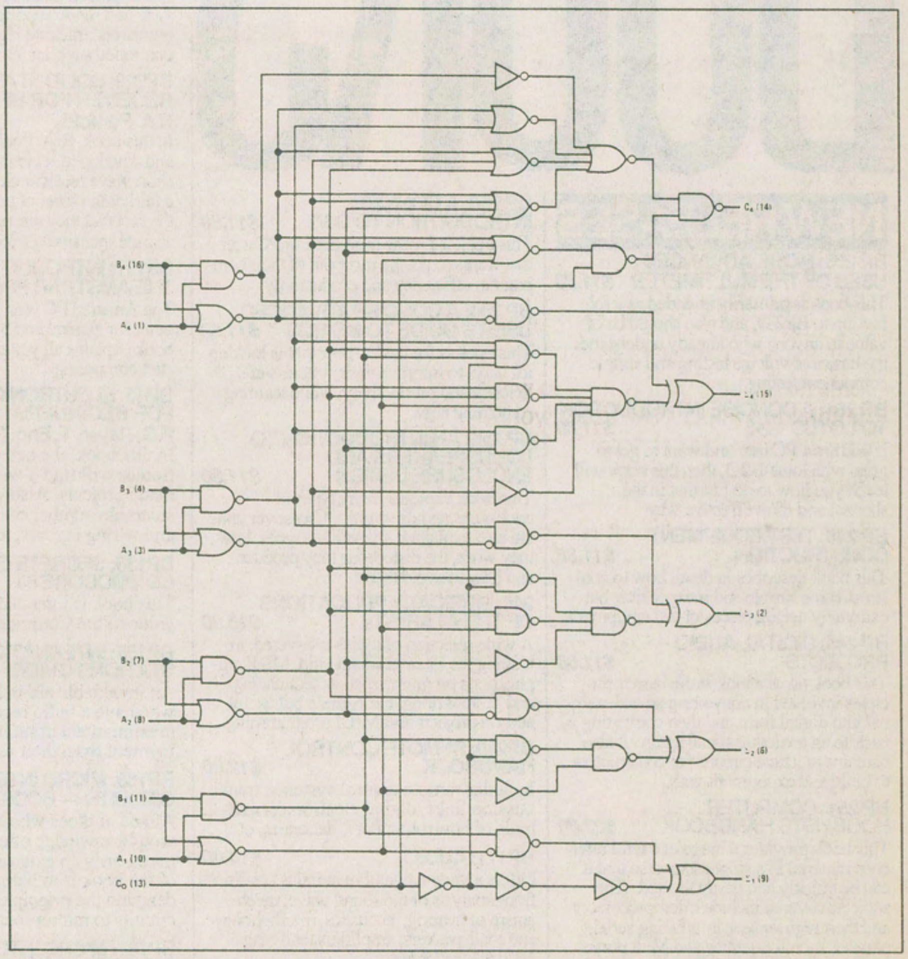
STATE 1	STATE 2	OUTPUT	CARRY
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	0

It would be easy enough to design a logic array which would accept these two sets of input states and produce these two output states. In a sense, you don't have to know how to do binary addition... all you have to do is to create a gate array which produces the desired truth table.

The approach to designing the four bit binary adder in Figure one is essentially the same, and the design problem would typically involve a fairly massive truth table. Once again, the project is not really to design a circuit which adds *per se*, but rather one which produces the desired truth table.

Many complex logic circuits can be dealt with this way. This approach tends to fall apart when you get into dynamic logic circuits, those for which you cannot develop a state diagram or truth table. A counter is a good example of this. Its output states are based both on its current input states and on the its previous input states.

We'll get into the rational for designing dynamic logic arrays later in this series.



The logical functions of the 74C83 4-bit binary full adder.