

A circuit for binary addition

These notes describe one design of a binary addition circuit. We consider the problem of building a circuit to add two 4-bit binary numbers. Here is an example addition:

$$\begin{array}{r} 0011 \\ + 0111 \\ \hline \end{array}$$

Starting from the rightmost bits, we add 1 and 1 to get 2, which is 10 in binary, so we put down the digit 0 and show the carry of a 1 into the next column:

$$\begin{array}{r} 1 \\ 0011 \\ + 0111 \\ \hline 0 \end{array}$$

Now we add up the three 1's, getting 3, which is 11 in binary, so we put down the digit 1 and show the carry of a 1 into the next column:

$$\begin{array}{r} 11 \\ 0011 \\ + 0111 \\ \hline 10 \end{array}$$

Continuing in this way, we finally get:

$$\begin{array}{r} 111 \\ 0011 \\ + 0111 \\ \hline 1010 \end{array}$$

Converting back into decimal to check, we have $3 + 7 = 10$.

To construct a circuit, we focus on one column at a time. The rightmost column has two inputs, say x and y , and the rightmost result bit, say z is determined by the truth table:

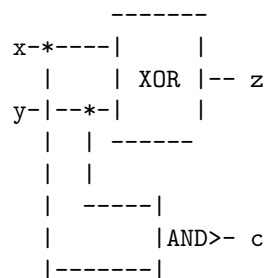
x	y	z
0	0	0
0	1	1
1	0	1
1	1	0

We observe that this is just the exclusive-or function. We'll assume we have gates available to compute exclusive-or, abbreviated XOR. If XOR gates are not available, we can construct one by using the sum-of-products method to construct a boolean expression:

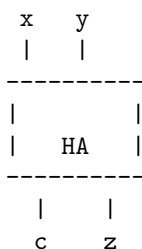
$$xy' + x'y,$$

which can be converted into a circuit with 2 NOT gates, 2 AND gates, and an OR gate.

The other thing we need to compute for the rightmost column is whether the carry into the next column is 1 or 0. The carry is 1 exactly in the case that x and y are both 1. Letting c denote the carry output, we have $c = xy$, implementable with just one AND gate. The final circuit for the rightmost column is:



Abstracting this to a box with inputs x and y and outputs z and c , we get the half-adder:



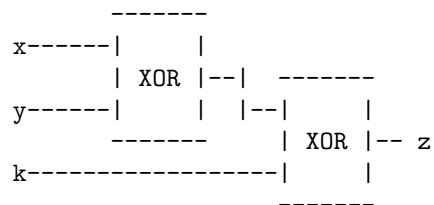
What about the next column? In the next column, we have the possibility of a carry-in to the addition. The function we consider has 3 inputs: x , y , and k . The value of the sum bit is determined by the following table:

x	y	k	z
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

This function can be realized as a sum-of-products:

$$x'y'k + x'yk' + xyk' + xyk$$

and the corresponding circuit. Or, we can note that two XOR gates will give the correct output:



This works because combining three inputs with XOR returns the parity of the sum of the three inputs: 1 if the sum is odd (1 or 3) and 0 if the sum is even (0 or 2).

The other output that we have to generate for this column is whether there is a carry into the next column. The truth-table for this function is:

x	y	k	c
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

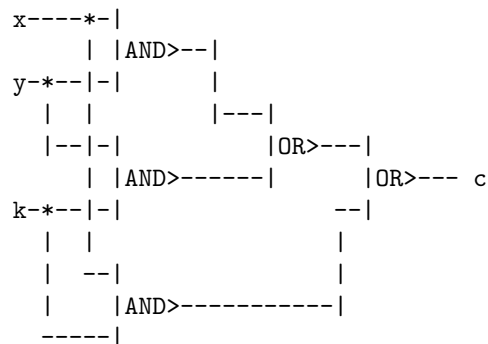
Note that there is a carry out when two or more of the inputs are 1. The sum-of-products algorithm gives the expression:

$$x'yk + xy'k + xyk' + xyk.$$

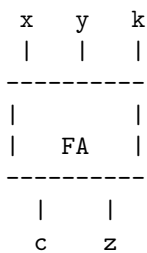
A simpler expression for the same function is:

$$yk + xk + xy.$$

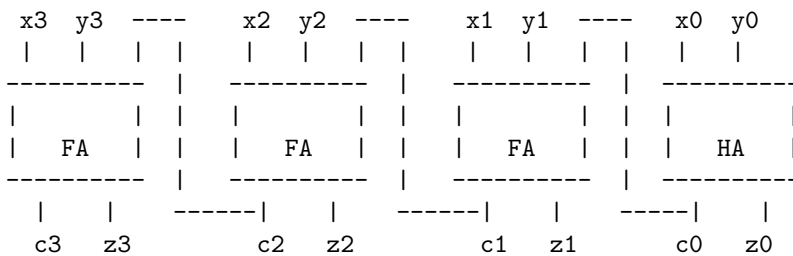
Implementing this as a circuit, we get:



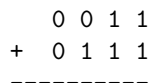
(Yes, we'll all be happier when I don't draw ASCII circuits!) Putting these last two circuits together into a box with inputs x , y , and k , and outputs z and c , we get a full-adder, which we'll symbolize thus:



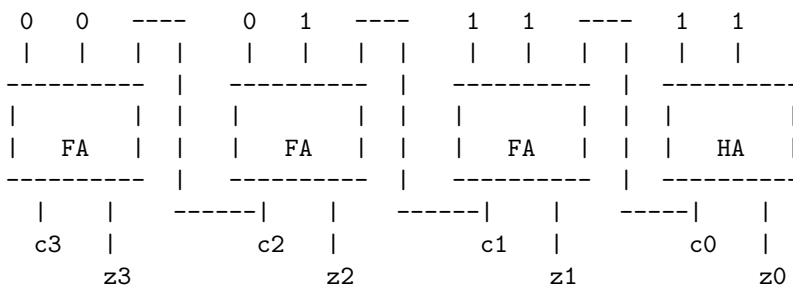
The full adder is what we need to compute the rest of the columns of the addition. Our four-bit addition can be computed by the following circuit connecting the carry-out from each addition to the carry-in of the next addition as shown below.



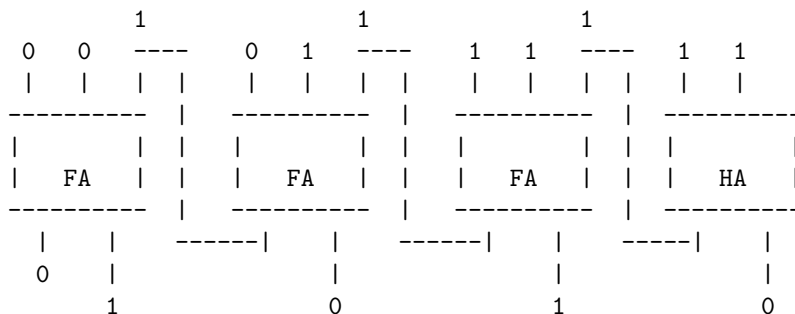
Referring back to the addition problem that we started with:



x_0 and y_0 are the two bits in the rightmost column, x_1 and y_1 are the two bits in the column to its left, and so on. Thus, we have inputs as shown below.



The outputs $z_0 = 0$ and $c_0 = 1$ are computed, and the value of c_0 is an input to the next circuit to the left, and so on, giving:



The resulting bits z_3, z_2, z_1, z_0 are 1010, as desired. Note that the last carry out bit (0 in this example) could be used to tell whether the sum is correctly expressed by the four result bits.

This circuit could be generalized to handle any number of bits. This is a ripple-carry adder, named for the way the carry “ripples” from the low-order to the high order bit. Note that for n bits, there will be about $3n$ gate delays before all the output bits can be assumed to have properly settled down. There are alternative designs for circuits to add two n -bit numbers that involve a gate delay proportional to $\log n$ rather than n , which are used in actual machines.