

Introduction to Digital Electronics: How does that adder work?

This is additional material for the Introduction to Electronics class. In the class, a floor exercise was held in which all the students acted as if each was a simple gate: AND, OR, or XOR. With a circuit laid out on the floor (and shown again in a figure further below on this page), the "Tester" person was able to specify binary numbers for the two inputs, and the output was the sum of those numbers... The circuit did additions by just manipulating 0s and 1s! The magic of the circuit was not explained in the class... Just that it worked by each student (just like each gate, or each transistor) doing his or her own job.

So the magic of addition is explained here, on this web page. Hold on tight, it will be getting pretty technical...

Before explaining how the 3-bit adder from the class works, we will have to take a better look at how numbers are represented in a computer, in terms of 1s and 0s. That is called the "binary number" system. But to understand binary numbers, we first need to take a closer look at the familiar decimal numbers that we humans work with every day.

Humans: the Decimal Number System

A decimal number is composed of a series of digits. Each digit is between 0 and 9. For example the number 4357 is defined by the place values of the digits 4, 3, 5 and 7:

$$4357 = 4*1000 + 3*100 + 5*10 + 7 \text{ or } 4357 = 4*10^3 + 3*10^2 + 5*10^1 + 7*10^0$$

Thus, decimal numbers are based on powers of 10.

Computers: the Binary Number System

Similarly, binary numbers are based on powers of 2, and use only the digits 0 and 1.

Binary	Represents		Decimal
000	$= 0*2^2 + 0*2^1 + 0*2^0$	$= 0*4 + 0*2 + 0*1 =$	0
001	$= 0*2^2 + 0*2^1 + 1*2^0$	$= 0*4 + 0*2 + 1*1 =$	1
010	$= 0*2^2 + 1*2^1 + 0*2^0$	$= 0*4 + 1*2 + 0*1 =$	2
011	$= 0*2^2 + 1*2^1 + 1*2^0$	$= 0*4 + 1*2 + 1*1 =$	3
100	$= 1*2^2 + 0*2^1 + 0*2^0$	$= 1*4 + 0*2 + 0*1 =$	4
101	$= 1*2^2 + 0*2^1 + 1*2^0$	$= 1*4 + 0*2 + 1*1 =$	5
110	$= 1*2^2 + 1*2^1 + 0*2^0$	$= 1*4 + 1*2 + 0*1 =$	6
111	$= 1*2^2 + 1*2^1 + 1*2^0$	$= 1*4 + 1*2 + 1*1 =$	7

Thus, binary numbers are written in terms of powers of 2:

$$\begin{aligned}
 2^0 &= 1 \\
 2^1 &= 2 \\
 2^2 &= 4 \\
 2^3 &= 8 \\
 2^4 &= 16 \\
 2^5 &= 32 \\
 2^6 &= 64 \\
 2^7 &= 128 \\
 &\text{etc.}
 \end{aligned}$$

To give another example, the decimal number 147 can be represented as:

$$\begin{aligned}
 147 &= 128 + 16 + 2 + 1 \\
 &= 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\
 &= 1001\ 0011 \text{ (binary)}
 \end{aligned}$$

The individual "digits" of a binary number, that is, the 0s and 1s, are often called "bits". Thus, the above binary number was written as an 8 bit number.

Binary addition

To add binary numbers, we need only to consider 4 cases:

A	0	0	1	1				
B	0	1	0	1				
	---	+	---	+	---	+	---	+
Result	0	1	1	10				

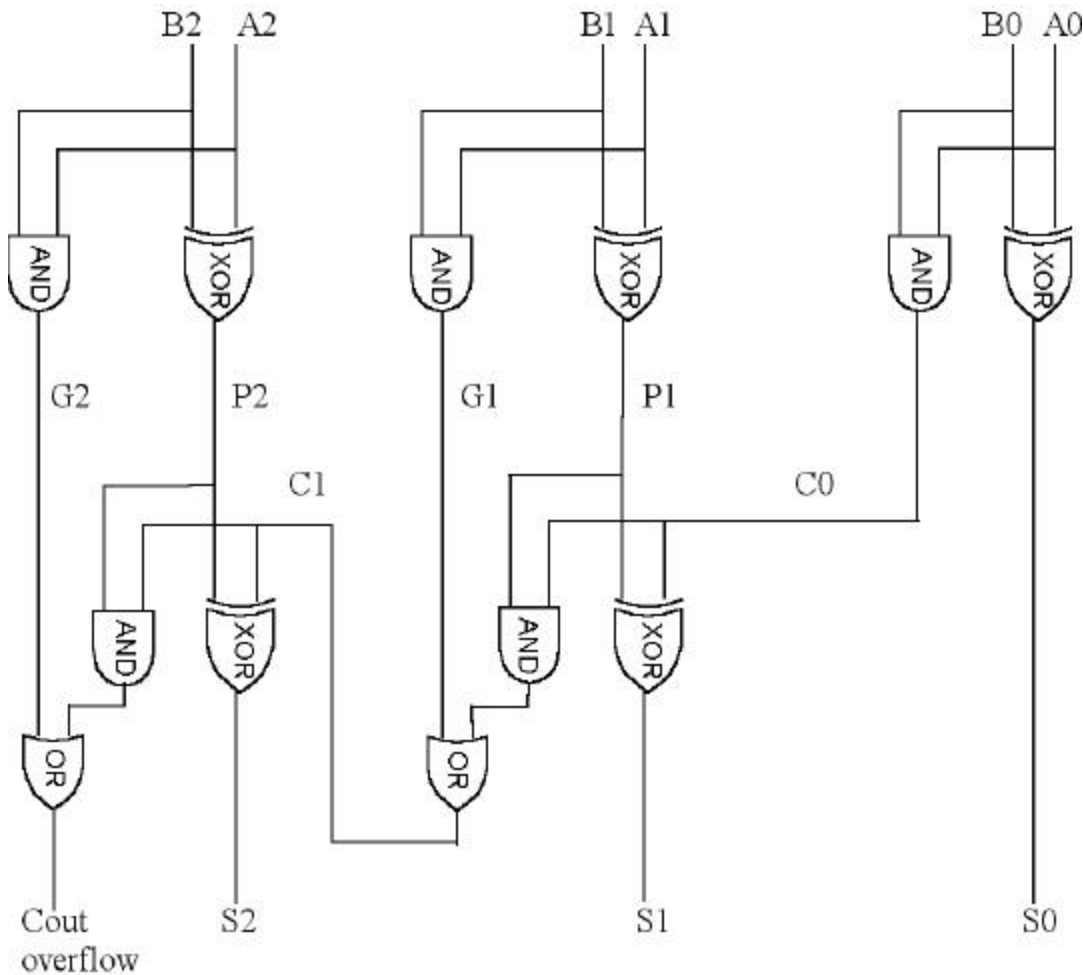
Note that $1+1 = 2$ decimal = 10 (binary) .

The black numbers in the Result row are the Sum bits, and the red 1 in the last case is the "Carry" bit. Looking only at the black numbers (sum bits) above, and comparing them with the truth tables we worked with in the class, you can see that the sum bits can be obtained with the XOR (not-the-same) function:

$$S = A \text{ xor } B.$$

Similarly, the red Carry bit can be obtained with an AND function:

$$C = A \text{ and } B.$$



Thus, the input (topmost) row of our circuit here takes the XOR and AND of the inputs:

$$\begin{aligned} S_0 &= A_0 \text{ xor } B_0, & C_0 &= A_0 \text{ and } B_0 \\ P_1 &= A_1 \text{ xor } B_1, & G_1 &= A_1 \text{ and } B_1 \\ P_2 &= A_2 \text{ xor } B_2, & G_2 &= A_2 \text{ and } B_2 \end{aligned}$$

For the 0-bit (least significant, rightmost bit) that is all. However, for the other bits we have to do more. Look at the next bit, with A_1 and B_1 as inputs. If the next less significant bit (to the right) generated a carry (that is, if $C_0 = 1$), then that carry bit needs to be added to the sum. Thus, $S_1 = (A_1 \text{ xor } B_1) \text{ xor } C_0$. Thus the first part, $A_1 \text{ xor } B_1 (= P_1)$ is only a partial sum! The 2nd xor, to compute S_1 , is done in the rightmost XOR gate in the bottom row of the circuit, which gives $S_1 = P_1 \text{ xor } C_0$.

Unfortunately, that is not all. For example, if $A_1=1$ and $B_1=0$, then adding the carry $C_0=1$ gives a total sum of 2 (decimal), which is 10 (binary). Thus, $S_1 = 0$, but a new carry, C_1 needs to be generated, to add to the next bit on the left, the one with the A_2, B_2 inputs.

We see that if $P_1 = (A_1 \text{ xor } B_1) = 1$, that is, if $A_1=1, B_1=0$ or if $A_1=0, B_1=1$, then the carry circuit is "primed": if $C_0=1$ then also C_1 should be set to 1. Thus, the carry bit "propagates". That's why P_1 is called P_1 .

If $A_1=0$ and $B_1=0$, then C_0 cannot propagate to C_1 .

If $A_1=1$ and $B_1=1$, then we know that the sum is going to be 2 or 3 (decimal), and thus a carry, C_1 should be generated. We don't even have to wait for C_0 to know this. Hence $G_1=(A_1 \text{ and } B_1)$ is called a "Generate" signal.

Putting all this together, C1 is set if G1 is set (generate), OR if P1 is set (propagate) AND C0 is set.
In a formula: $C1 = G1 \text{ or } (P1 \text{ and } C0)$.

This is exactly what the rightmost AND and OR gate in the bottom row of the circuit accomplish.

On the leftmost side of the circuit, exactly the same thing happens.

```

Propagate:           P2 = A2 xor B2
Generate:           G2 = A2 and B2
Sum:                S2 = A2 xor B2 xor C1 = P2 xor C1
Carry-out (or overflow): Cout = G2 or (P2 and C1).

```

Note that the Carry-out is an overflow (error!) if the rest of the circuit can only handle 3 bit numbers. That's a bit dull, because then you can only make additions with sums up to 7 decimal (= 111 binary). If the Carry-out is regarded as valid by the rest of the circuit (as it was by us in the class), then the circuit calculates sums up to 15 decimal (= 1111 binary).

Note also that Cout and S2 have to wait for C1, which has to wait for C0. Thus, if you think of a bigger adder (16 or 32 or 64 bits wide are common), then the effect of C0 would "ripple through", as long as the Propagates along the way are set. Therefore, this adder circuit is called a "ripple carry" adder.

Many alternative adder designs have been made to speed up the ripple effect.... but that is another class / webpage... or will be your invention!

[Ruud Haring <ruud@us.ibm.com>](mailto:ruud@us.ibm.com) --created 01/19/01

Last updated Saturday, January 20, 2001 14:36:40