Addition

4 min · Justin Rajewski | Suggest Changes

▶ Table of Contents

Addition is a very important function that allows for the basis of a lot of designs. In order to add two numbers, we first need to decide what number encoding to use. Since the most common is binary, we will be using that.

The Half Adder

To get a feel for what addition entails with binary numbers, lets look first at the most simple case. What happens when we add two 1 bit numbers?

When designing a circuit it is often helpful to write out a truth table, so we will do that now.

A	B	Out
0	0	00
0	1	01
1	0	01
1	1	10

It is easy to see now that if we only output one bit then when you add 1 and 1 together the result will be incorrect. That means that our circuit will have to have two outputs. If we write out the truth table for each one we get.

A	B	Out[0]
0	0	0
0	1	1
1	0	1
1	1	0

A	B	Out[1]
0	0	0
0	1	0
1	0	0
1	1	1

Notice that these two truth tables are exactly the same as two of the logic gates we've covered!

For **Out[0]** you can see that it is only 1 when **the inputs are different**. In other words **Out[0]** is **A** xor **B**.

For **Out[1]** you can see that it is only 1 when **the inputs are both 1**. In other words **Out[1]** is **A** and **B**.

Let's now use that to draw up a circuit.

This circuit is known as the **half adder**.

You can see the outputs are renamed from **Out[0]** and **Out[1]** to **Sum** and **Cout**. That is because it is better to think of them in these terms later. **Cout** stands for **carry out**. Just like when you learned to do addition of big numbers you had to carry any overflow to the next addition, the same is done with binary addition.

The Full Adder

So we can now add two one bit numbers, but let's be honest, that's really not very useful. We need to be able to add any arbitrary sized number. This is where the full adder comes in handy. When we created the circuit for the half adder we assumed we only had two inputs, the two numbers we were adding. However, our circuit output a **carry bit**!

Going back to those very early days of school, when you add numbers it looks something like this

```
 1
19
+22
41
```

But how did we add that second column? We simply added all **three** numbers together, the two inputs 1 from the 19 and 2 from the 22, as well as the carry from

the previous addition. That brings us to the truth table for the full adder.

Cin	A	B	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

We are now adding the three inputs **Cin**, **A**, and **B**.

It would be a good exercise if you took this truth table and created a K-map for each output and used that to derive a circuit.

Here is a circuit that implements this.

The Carry Chain

Now that we have a full adder we can add any arbitrarily sized numbers by chaining the carry bits.

Here is a circuit that will add two 4 bit numbers.

You may notice that the output from adding two 4 bit numbers is not a 4 bit number, but actually a 5 bit one! That is because when you add two numbers they can overflow their respective ranges, but when adding only two numbers the sum can't be more than twice the range of the inputs.

Something else worth noting is that a full adder behaves the same as a half adder if you connect the carry in to 0. In the circuit above the half adder could be replaced with a full adder.

This type of circuit is called a **carry chain**. It is important to notice that the last bit must wait for all of the previous bits to be calculated. This makes adding very large numbers take more time than summing small numbers. This is an inherent problem with carry chain circuits.

« PREV
Multiplexers

NEXT »
Subtraction