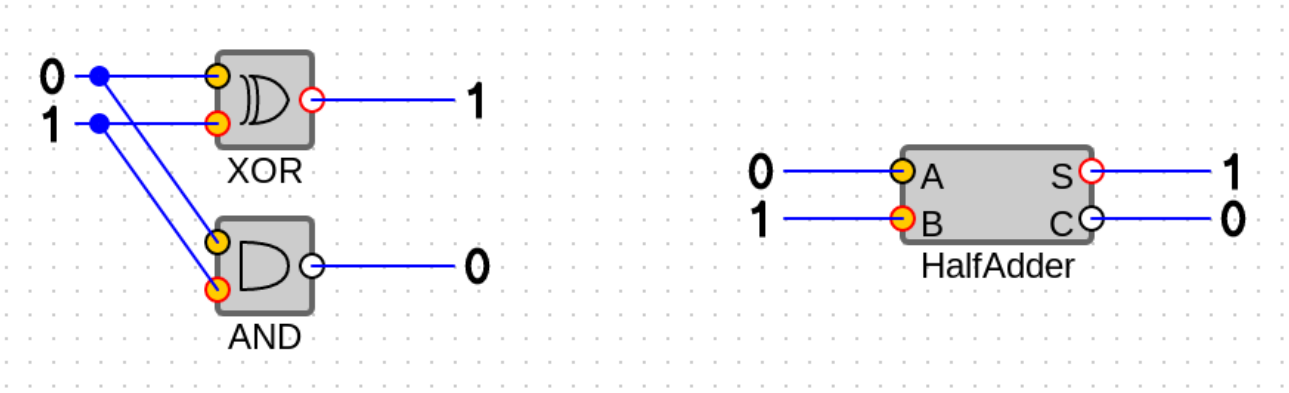


This adder circuit is a common circuit, so we give it a name: the “Half Adder”. It can add two binary numbers (0 or 1) together. This is exactly how a computer, which is just an electrical circuit, adds two numbers together.

Like the XOR gate, it has its own component in the simulator. Place a HalfAdder component in the simulator and connect up its inputs and outputs as on the right, above. Try all possible input combinations: 00, 01, 10 and 11 to verify that it behaves the same as the Half Adder that you made with XOR and AND gates. If you double click on the simulator's HalfAdder component you can look inside it, and you will see that it contains the same adder circuit that we constructed. The “S” and “C” labels on the half adder's outputs are short for “Sum” and “Carry”.

Does the word “carry” seem familiar? Think of when you add two numbers together in class. You carry a 10 when the answer gets bigger than 9. This half-adder circuit is doing exactly the same! It produces a carry output when the sum (S) gets bigger than 1 and can't fit into a single binary digit.

Preview of next week: Can you see where this is going? Next week we cover the full-adder circuit. It can add three bits together, instead of just two bits. This lets us add in a carry from a previous column. We can then use a chain of full adders to add large numbers together, not just 0 and 1.

7/8/2018 Lesson 7: (Term 3, Week 3)

(This lesson needs work to make it more understandable, or it should be split in two?)

In the last lesson we learned about the half adder, a circuit that can add two 1s or 0s together. It could do the following sums:

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 2 = 10_2$ (in binary)

The Full Adder

In this lesson we will learn how to combine two half adders to make a **full adder**: a circuit which can add three bits together. A full adder has three inputs and can do the following sums:

$0 + 0 + 0 = 0 = 00_2$
$0 + 0 + 1 = 1 = 01_2$
$0 + 1 + 0 = 1 = 01_2$
$0 + 1 + 1 = 2 = 10_2$
$1 + 0 + 0 = 1 = 01_2$
$1 + 0 + 1 = 2 = 10_2$
$1 + 1 + 0 = 2 = 10_2$
$1 + 1 + 1 = 3 = 11_2$

In the above, we have given the answers as both decimal numbers and binary numbers. Notice that the biggest answer is 3 (= 1+1+1), which can be represented as a binary number (11_2) which has a length of two bits. Hence our full adder will have two output bits.

If we call the three input bits C_{IN}, A and B, and the two output bits C_{OUT} and S, we can write the above eight sums in the form of a truth table:

C_{IN}	A	B	C_{OUT}	S
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

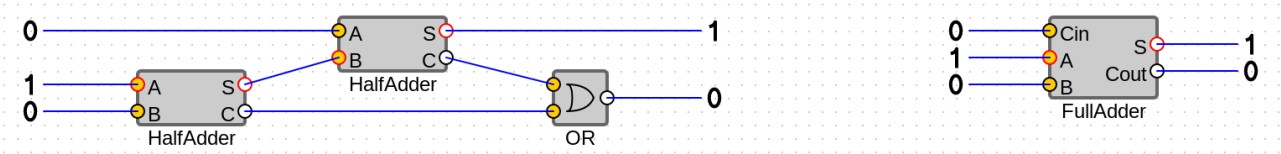
Each row of the truth table is a sum. In each row, when you add the three inputs together (0 or 1) you get an answer in the range 0-3, represented by the two output bits.

We chose the labels C_{IN} and C_{OUT} because these bits are a carry input and carry output. This will make more sense in the next section.

How might we build a circuit to implement this truth table add three bits together? Think of how you might add three numbers together. One strategy would be to first add two of the numbers together, then add the third number to the answer, to get the sum of all three numbers. This is the strategy we will use when building a full adder.

We already have a circuit to add two bits together: the half adder. to make a full adder, we will use a half adder to add the first two bits together. We will then use a second half adder to add the third bit to the sum of the other two. See the picture below.

Note how we generate the second bit of the output of the full adder: the bit which represents the 2s column in our answer. The 2s column will be equal to "1" if the final answer is 2 ($=10_2$) or 3 ($=11_2$), which will only happen if the answer from at least one of the half adders is equal to 2 ($=10_2$), meaning its carry (C) output is "1". Thus we use an OR gate to combine the carry outputs from the two half adders, to form the carry output from the full adder. The final logic network is shown below.



Enter the above network for a full adder into the logic simulator. Verify that the behaviour of the circuit matches the truth table for the adder circuit, and that it really can add three 0s and 1s together. The output of the full adder is a 2-bit binary number: with the output of the OR gate generating the 2s column (the C_{OUT} output) and the other output being the 1s column (the S output).

Like the half adder, the full adder has its own component in the simulator. Place a FullAdder component in the simulator and connect up its inputs and outputs as on the right, above. Try all possible input combinations to verify that it behaves the same as the Full Adder that you made with two half adders and an OR gate. Also verify that the output is the sum of the inputs. Given that we are only adding 1s together, the output is basically counting the number of inputs which are set to one. If you double click on the simulator's FullAdder component you can look inside it, and you will see that it contains the same circuit that we constructed. The "S" and "C_{OUT}" labels on the half adder's outputs are short for "Sum" and "Carry Output".

How Computers Add Numbers

We are now in a position to understand how a computer adds numbers of any size together.

Let us start by remembering how we normally add large numbers together. Consider the following sum:

$$\begin{array}{r} 238 \\ +326 \\ \hline 564 \\ \hline \end{array}$$

We start by adding the 1s column: 8+6=14. We can't fit the number 14 in the 1s column though, so we carry the 10 to the 10s column, leaving a 4 in the 1s column. Now we add the 10s column: 1+3+2=6. Note that we are adding three numbers together, as we have to include the carry from the 1s column. In this case the answer fits in the 10s column, we we write the 6 in and finish off by adding the 100s column: 2+3=5.

Let's rewrite the above sum to show how we worked it out, and carried numbers between columns:

```
 238
+326
 01
 \\
 564
```

We have used an arrow to indicate the carry from each column to the next, and the amount carried is shown. In the above sum, we carried 1 group of 10 from the 1s column to the 10s column and 0 groups of 100 from the 10s column to the 100s column.

Adding large binary numbers works in exactly the same way as adding large decimal numbers, but we carry groups of 2 rather then groups of 10. An example is below. In this example, we have also shown the equivalent decimal numbers, so you can see that binary addition gives the same answer as decimal addition.

```
 1001          9
+0011         +3
0011
\\\\
 1100         12
```

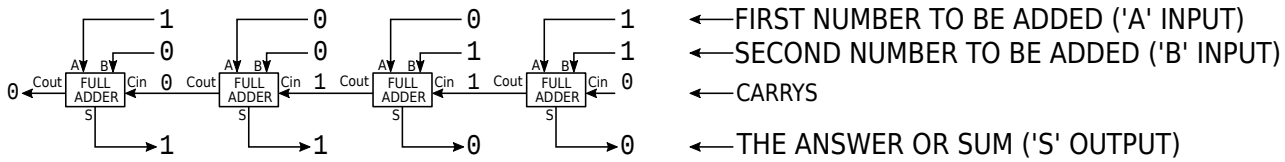
We first add the 1s column: $1+1=2=10_2$. We don't have a binary digit for 2 (only 0 and 1), so we carry one group of 2 to the 2s column, leaving a 0 in the 1s column. Now we add the numbers, in the 2s column: $0+1+1=2=10_2$, noting that we are adding three digits together because we have to include the carry from the 1s column. Again the answer is 2 (10_2), so we put the 0 in the twos column and carry one group of 4 to the 4s column. Now we add the digits in the 4s column: 1+0+0=1. The answer 1 can fit in the 4s column, so we write it there and there is no carry. No carry is another way of saying that we carried no groups of 8, so we can write a 0 in the row where are are recording the carries. Now we finish off by adding the numbers in the 8s column: 1+0+0=1, and there is no carry.

As before, we have used an arrow to indicate the carry from each column to the next, and the amount carried is shown. In the above sum, we carried 1 group of 2 from the 1s column to the 2s column, and 1 group of 4 from the 2s column to the 4s column, 0 groups of 8 from the 4s column to the 8s column and there were 0 groups of 16 carried from the 8s column.

From this example of adding two large binary numbers, we can work out how to build a circuit to add two large binary numbers together. For each column, we need to add three bits together: one bit from each of the numbers being added and the carry bit from the previous column. The result of adding these bits together is two new bits: one bit which goes into the answer and a second bit which is the carry to the next column. We already have a circuit which can do exactly this task of adding three bits together to give a two bit answer: **the full adder**. We can use a full adder to add the three bits together in each column. Thus the number of full adders we need is equal to the number of columns, which is also the length, in bits, of the two numbers to be added together.

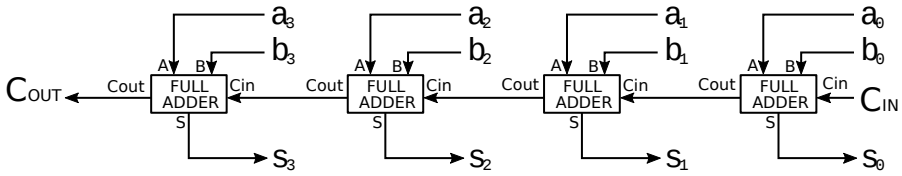
We make an adder circuit by joining full adders into a chain, one full adder for each bit in the numbers to be added. The carry output from each full adder goes to the carry input of the next full adder in the chain. There is no carry into the 1s column, so we can either use a half adder for the 1s column, or a full adder with its carry input always set to 0.

An adder circuit is shown below:



The adder circuit is drawn with the binary addition example that we used previously ($1001_2+0011_2=1100_2$), so you can see how it works: the numbers to be added are in the first and second rows, the carry bits are in the third row and the answer is in the bottom row. The digits are arranged in columns, the 1s column on the right and the place value of each column increasing by 2 as we move to the left: 1, 2, 4, 8, ... We have chosen to use a full adder with a zero on the carry input for the 1s column. This means exactly the same circuit (a full adder) is used for each column of the addition. Verify the calculation manually, working from the 1s column, to the 2s column then to the 4s column and finishing with the 8s column.

Finally we draw the same adder circuit as above, but without the annotated example. This circuit will add two four bit numbers: $A=a_3,a_2,a_1,a_0$ and $B=b_3,b_2,b_1,b_0$ to give a four bit sum $S=s_3,s_2,s_1,s_0$.



Enter the network for the above 4-bit adder circuit into the logic simulator, as on the left hand side below. Use the AltFullAdder component, as it has its inputs and output in convenient locations. Add a 4bit7seg display to each input and the output. These 7-segment displays will show the numbers being added and the sum as decimal numbers, saving you having to convert between binary and decimal in your head. Now try a few different numbers on the inputs. You will see that the output of the adder is always the sum of the two input numbers!

Note: If the answer is greater than 9, you will see a letter 'a', 'b', 'c', 'd', 'e' or 'f' on 7-segment display. This happens because the 7-segment display can only display one digit, and we don't have a single digit for the numbers 10 to 15. Hence the displays use the letters 'a' to 'f' to represent the numbers 10 to 15. This type of representation is called “base 16” or hexadecimal. It's like binary, but we use the number 16 in place of the number 2.