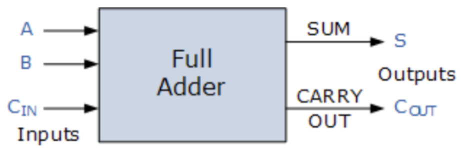
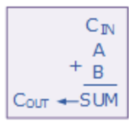




Binary Adder

Binary Adders are arithmetic circuits in the form of half-adders and full-addersb used to add together two binary digits

Another common and very useful combinational logic circuit which can be constructed using just a few basic logic gates allowing it to add together two or more binary numbers is the **Binary Adder**.

A basic Binary Adder circuit can be made from standard AND and Ex-OR gates allowing us to "add" together two single bit binary numbers, A and B.

The addition of these two digits produces an output called the SUM of the addition and a second output called the CARRY or Carry-out, (C_{OUT}) bit according to the rules for binary addition. One of the main uses for the *Binary Adder* is in arithmetic and counting circuits. Consider the simple addition of the two denary (base 10) numbers below.

A basic Binary Adder circuit can be made from standard AND and Ex-OR gates allowing us to "add" together two single bit binary numbers, A and B.

The addition of these two digits produces an output called the SUM of the addition and a second output called the CARRY or Carry-out, (C_{OUT}) bit according to the rules for binary addition. One of the main uses for the *Binary Adder* is in arithmetic and counting circuits. Consider the simple addition of the two denary (base 10) numbers below.

123	A	
+ 789	B	(Addend)
912	SUM	

From our maths lessons at school, we learnt that each number column is added together starting from the right hand side and that each digit has a weighted value depending upon its position within the columns.

From our maths lessons at school, we learnt that each number column is added together starting from the right hand side and that each digit has a weighted value depending upon its position within the columns.

When each column is added together a carry is generated if the result is greater or equal to 10, the base number. This carry is then added to the result of the addition of the next column to the left and so on, simple school math's addition, add the numbers and carry.

The adding of binary numbers is exactly the same idea as that for adding together decimal numbers but this time a carry is only generated when the result in any column is greater or equal to "2", the base number of binary. In other words 1 + 1 creates a carry.

Binary Addition

Binary Addition follows these same basic rules as for the denary addition above except in binary there are only two digits with the largest digit being "1". So when adding binary numbers, a carry out is generated when the "SUM" equals or is greater than two (1+1) and this becomes a "CARRY" bit for any subsequent addition being passed over to the next column for addition and so on. Consider the single bit addition below.

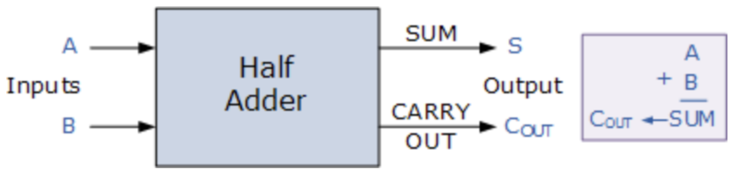
Binary Addition of Two Bits

0	0	1	1
$\underline{+0}$	$\underline{+1}$	$\underline{+0}$	$\underline{+1}$
0	1	1	(carry) 1←0

When the two single bits, A and B are added together, the addition of "0 + 0", "0 + 1" and "1 + 0" results in either a "0" or a "1" until you get to the final column of "1 + 1" then the sum is equal to "2". But the number two does not exists in binary however, 2 in binary is equal to 10, in other words a zero for the sum plus an extra carry bit.

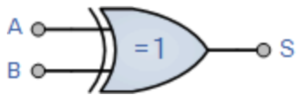
Then the operation of a simple adder requires two data inputs producing two outputs, the Sum (S) of the equation and a Carry (C) bit as shown.

Binary Adder Block Diagram



For the simple 1-bit addition problem above, the resulting carry bit could be ignored but you may have noticed something else with regards to the addition of these two bits, the sum of their binary addition resembles that of an **Exclusive-OR Gate**. If we label the two bits as A and B then the resulting truth table is the sum of the two bits but without the final carry.

2-input Exclusive-OR Gate

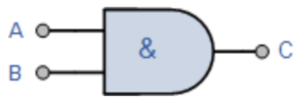
Symbol	Truth Table		
	B	A	S
 2-input Ex-OR Gate	0	0	0
	0	1	1
	1	0	1
	1	1	0

We can see from the truth table above, that an Exclusive-OR gate only produces an output "1" when either input is at logic "1", but not both the same as for the binary addition of the previous two bits. However in order to perform the addition of two numbers, microprocessors and electronic calculators require the extra carry bit to correctly calculate the equations so we need to rewrite the previous summation to include two-bits of output data as shown below.

00	00	01	01
+ 00	+ 01	+ 00	+ 01
00	01	01	10

From the above equations we now know that an **Exclusive-OR** gate will only produce an output "1" when "EITHER" input is at logic "1", so we need an additional output to produce the carry bit when "BOTH" inputs A and B are at logic "1". One digital gate that fits the bill perfectly producing an output "1" when both of its inputs A and B are "1" (HIGH) is the standard **AND Gate**.

2-input AND Gate

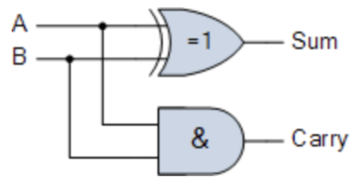
Symbol	Truth Table		
	B	A	C
 2-input AND Gate	0	0	0
	0	1	0
	1	0	0
	1	1	1

By combining the Exclusive-OR gate with the AND gate results in a simple digital binary adder circuit known commonly as the "**Half Adder**" circuit.

A Half Adder Circuit

A half adder is a logical circuit that performs an addition operation on two binary digits. The half adder produces a sum and a carry value which are both binary digits.

Half Adder Truth Table with Carry-Out

Symbol	Truth Table			
	B	A	SUM	CARRY
	0	0	0	0
	0	1	1	0
	1	0	1	0
	1	1	0	1

From the truth table of the half adder we can see that the SUM (S) output is the result of the Exclusive-OR gate and the Carry-out (Cout) is the result of the AND gate. Then the Boolean expression for a half adder is as follows.

For the **SUM** bit:

$$\text{SUM} = \text{A XOR B} = A \oplus B$$

For the **CARRY** bit:

$$\text{CARRY} = \text{A AND B} = A.B$$

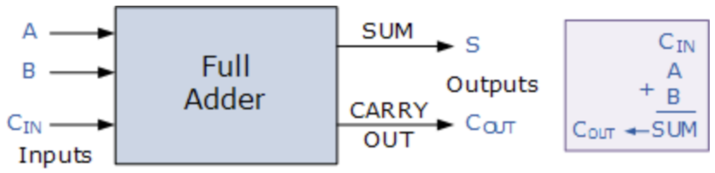
One major disadvantage of the *Half Adder* circuit when used as a binary adder, is that there is no provision for a "Carry-in" from the previous circuit when adding together multiple data bits.

For example, suppose we want to add together two 8-bit bytes of data, any resulting carry bit would need to be able to "ripple" or move across the bit patterns starting from the least significant bit (LSB). The most complicated operation the half adder can do is "1 + 1" but as the half adder has no carry input the resultant added value would be incorrect. One simple way to overcome this problem is to use a **Full Adder** type binary adder circuit.

A Full Adder Circuit

The main difference between the **Full Adder** and the previous **Half Adder** is that a full adder has three inputs. The same two single bit data inputs A and B as before plus an additional *Carry-in* (C-in) input to receive the carry from a previous stage as shown below.

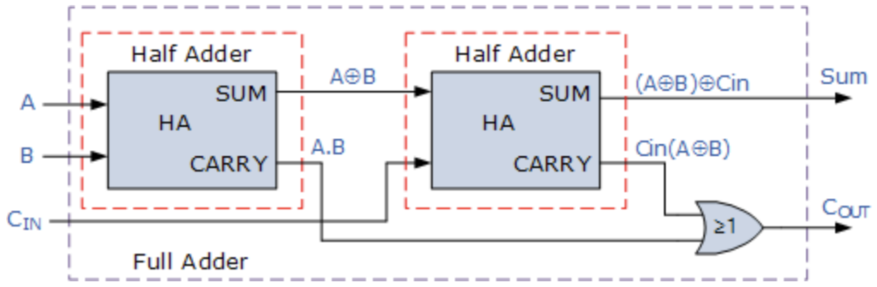
Full Adder Block Diagram



Then the **full adder** is a logical circuit that performs an addition operation on three binary digits and just like the half adder, it also generates a carry out to the next addition column. Then a *Carry-in* is a possible carry from a less significant digit, while a *Carry-out* represents a carry to a more significant digit.

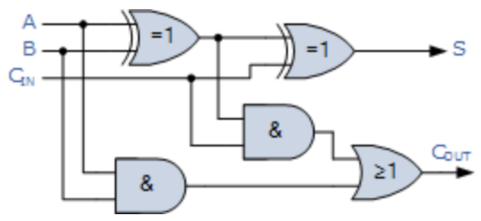
In many ways, the full adder can be thought of as two half adders connected together, with the first half adder passing its carry to the second half adder as shown.

Full Binary Adder Logic Diagram



As the full adder circuit above is basically two half adders connected together, the truth table for the full adder includes an additional column to take into account the *Carry-in*, C_{IN} input as well as the summed output, S and the Carry-out, C_{OUT} bit.

Full Adder Truth Table with Carry

Symbol	Truth Table				
	C-in	B	A	Sum	C-out
 A B C_{IN} =1 =1 S & & ≥1 C_{OUT}	0	0	0	0	0
	0	0	1	1	0
	0	1	0	1	0
	0	1	1	0	1
	1	0	0	1	0
	1	0	1	0	1
	1	1	0	0	1
	1	1	1	1	1

Then the Boolean expression for a full adder is as follows.

For the **SUM** (S) bit:

$$\text{SUM} = (\text{A XOR B}) \text{ XOR Cin} = (A \oplus B) \oplus \text{Cin}$$

For the **CARRY-OUT** (Cout) bit:

$$\text{CARRY-OUT} = \text{A AND B OR Cin(A XOR B)} = A.B + \text{Cin}(A \oplus B)$$

An n-bit Binary Adder

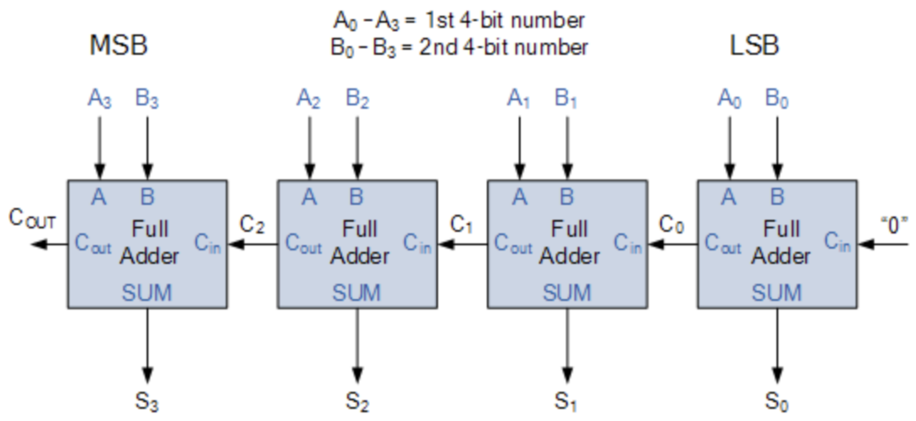
We have seen above that single 1-bit binary adders can be constructed from basic logic gates. But what if we wanted to add together two n-bit numbers, then n number of 1-bit full adders need to be connected or “cascaded” together to produce what is known as a **Ripple Carry Adder**.

A “ripple carry adder” is simply “n“, 1-bit full adders cascaded together with each full adder representing a single weighted column in a long binary addition. It is called a ripple carry adder because the carry signals produce a “ripple” effect through the binary adder from right to left, (LSB to MSB).

For example, suppose we want to “add” together two 4-bit numbers, the two outputs of the first full adder will provide the first place digit sum (S) of the addition plus a carry-out bit that acts as the carry-in digit of the next binary adder.

The second binary adder in the chain also produces a summed output (the 2nd bit) plus another carry-out bit and we can keep adding more full adders to the combination to add larger numbers, linking the carry bit output from the first full binary adder to the next full adder, and so forth. An example of a 4-bit adder is given below.

A 4-bit Ripple Carry Binary Adder



One main disadvantage of “cascading” together 1-bit **binary adders** to add large binary numbers is that if inputs A and B change, the sum at its output will not be valid until any carry-input has “rippled” through every full adder in the chain because the MSB (most significant bit) of the sum has to wait for any changes from the carry input of the LSB (less significant bit). Consequently, there will be a finite delay before the output of the adder responds to any change in its inputs resulting in a accumulated delay.

When the size of the bits being added is not too large for example, 4 or 8 bits, or the summing speed of the adder is not important, this delay may not be important. However, when the size of the bits is larger for example 32 or 64 bits used in multi-bit adders, or summation is required at a very high clock speed, this delay may become prohibitively large with the addition processes not being completed correctly within one clock cycle.

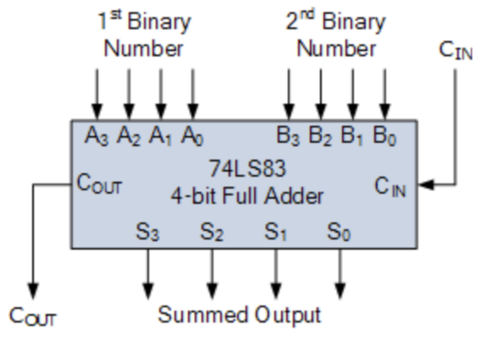
This unwanted delay time is called **Propagation delay**. Also another problem called "overflow" occurs when an n-bit adder adds two parallel numbers together whose sum is greater than or equal to 2^n

One solution is to generate the carry-input signals directly from the A and B inputs rather than using the ripple arrangement above. This then produces another type of binary adder circuit called a **Carry Look Ahead Binary Adder** where the speed of the parallel adder can be greatly improved using carry-look ahead logic.

The advantage of carry look ahead adders is that the length of time a carry look ahead adder needs in order to produce the correct SUM is independent of the number of data bits used in the operation, unlike the cycle time a parallel ripple adder needs to complete the SUM which is a function of the total number of bits in the addend.

4-bit full adder circuits with carry look ahead features are available as standard IC packages in the form of the TTL 4-bit binary adder 74LS83 or the 74LS283 and the CMOS 4008 which can add together two 4-bit binary numbers and generate a SUM and a CARRY output as shown.

74LS83 Logic Symbol



Summary About The Binary Adder

We have seen in this tutorial about **Binary Adders** that adder circuits can be used to “add” together two binary numbers producing a “carry-out”. In its most basic form, adders can be made from connecting together an Exclusive-OR gate with an AND gate to produce a **Half Adder** circuit. Two half adders can the be combined to produce a **Full Adder**.

There are a number of 4-bit full-adder ICs available such as the 74LS283 and CD4008. which will add two 4-bit binary number and provide an additional input carry bit, as well as an output carry bit, so you can cascade them together to produce 8-bit, 12-bit, 16-bit, adders but the carry propagation delay can be a major issue in large n-bit ripple adders.