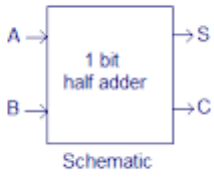
HALF ADDER, FULL ADDER/SUBTRACTOR : A STEP TOWARDS AN ARITHMETIC UNIT

As a modest beginning towards implementing the arithmetic unit the core doing the basic operation we start with the problem of adding two bits. A circuitry that can add 2 bits is known as Half Adder (HA)

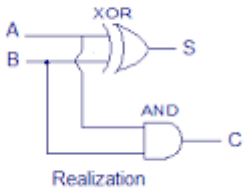
Half-adder Truth table

Inputs		Outputs	
A	B	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Truth table



Schematic



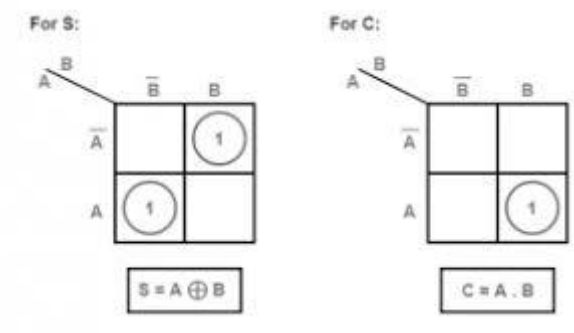
Realization

The schematic shows it as a black-box with 2 inputs and 2 outputs.

S(SUM) = AB' + A'B = A xor B;

C(CARRY) = A.B

k-map FOR IMPLEMENTATION OF SUM AND carry FOR HA



Note that from the expression we see that no simplification is possible—a fact corroborated in K-Map as well. Simplification in K-Map is possible if we can form a rectangle – e.g A rectangle encompassing two adjacent cells horizontally (or vertically) is equivalent to, say, A'B' + A'B which is equivalent to A'(B'+B) = A'.

Now we may venture to implement a full-adder (FA). For that we will be starting with the truth table. For a FA we have 3-inputs (A, B and Cin) and 2-outputs, namely, S and Cout.

Also, note that in reality we will be adding n-bit operands together so we need to have a FA for each column.

Full-adder Truth table, schematic and implementation

Inputs			Outputs	
A	B	C_{in}	C_{out} (Carry)	S (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

SUM = A'B'Cin' + A'BCin' + AB'Cin' + ABCin = A(BC + B'Cin') + A'(BCin' + B'Cin)

= A (B XOR C)' + A'(BCin' + B'Cin) = A xor B xor Cin

We need a 3-input XOR gate for the SUM part [No rectangle is formed in the K-Map – so simplification is not possible]

Cout = ABCin' + AB'Cin + A'BCin + ABCin = AB(Cin'+Cin) + AB'Cin + A'BCin

= AB + Cin(A xor B) ----- (1) (not fully simplified but useful -- as we shall see later)

= ABCin' + AB'Cin + A'BCin + ABCin + ABCin + ABCin + ABCin [Since, A = A + A + A]

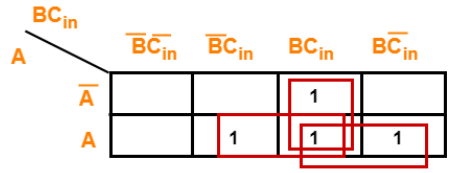
= AB(Cin + Cin') + BCin(A'+A) + ACin(B+B') = AB + BCin + ACin (Simplified – shown in K map as well)

And we need three 2-input AND gate and one 3-input OR gate to implement the carry part.

For S:

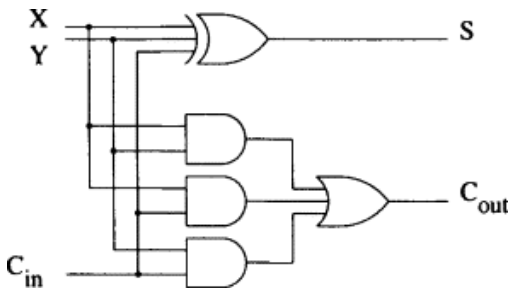
$$S = A \oplus B \oplus C_{in}$$

For C_{in}:

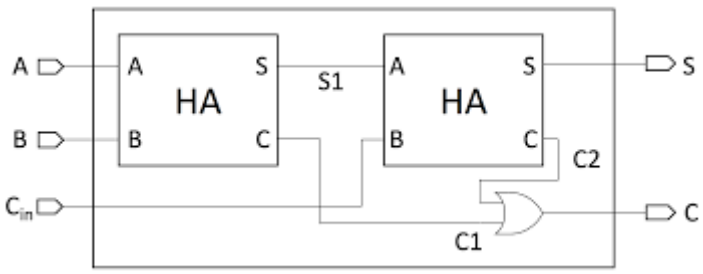


$$C_{out} = AB + BC_{in} + C_{in}A$$

So, the implementation directly following the expressions of S and Cout is shown below.



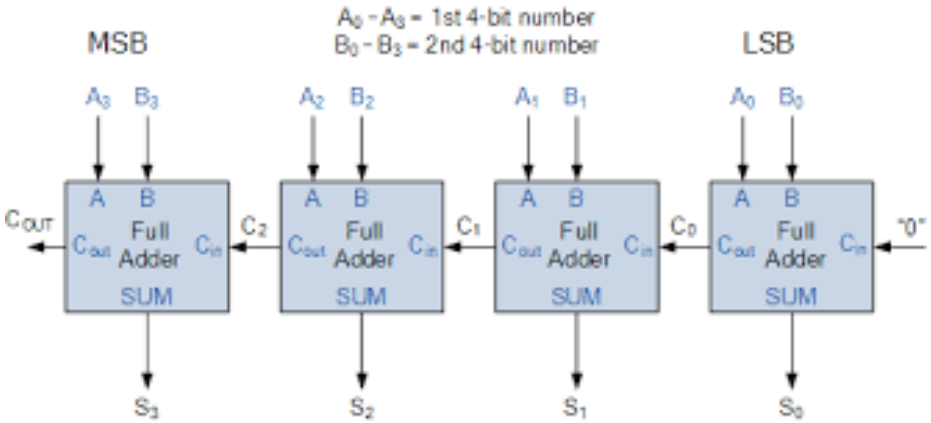
As I have mentioned in the passing that in digital logic circuit we try to use modular design and try to utilise if we already have something similar. So, using two HAs' and one 2-input OR gate we could achieve the same as shown below:



Note that: S1 = A xor B; so, S = (A xor B) xor C = A xor B xor C

Similarly, C = C1 + C2 = AB + (A xor B) Cin = AB + AB'Cin + A'BCin [Note the use of equation (1)]

As indicated earlier that in reality we add n-bit operands (say, n = 4 together) and the circuitry would look like below.

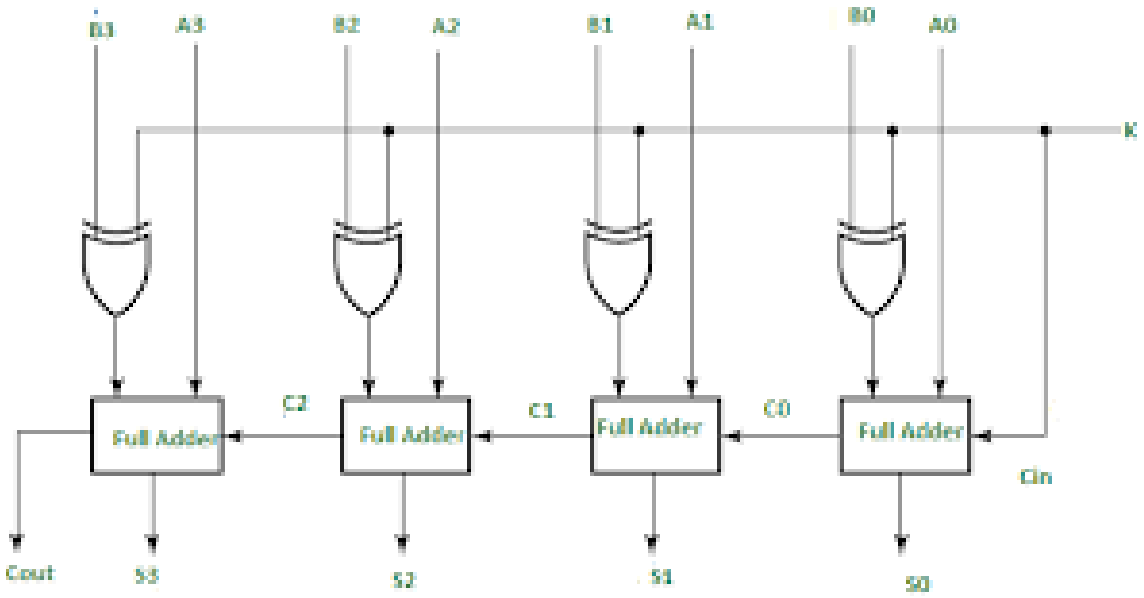


Note i) here A is A3A2A1A0 and B is B3B2B1B0; ii) For the first stage we could have a HA only as the carry is 0 always; iii) Cout of each stage is driving the Cin of the next stage. So, for a 64-bit full adder there would be 64 such full adders cascaded together.

Finally, let us see a full adder/full subtractor implementation.

You may recall that A – B can be thought as A + (-B) – and this leads to the representation of negative numbers. As we already know 2-s complement is best suited for this job here we see a simple way to convert a 4-bit full adder into a 4-bit Full adder cum full-subtractor. If the control input K = 0 then B is not converted to its 2-s complement form and the circuits adds. When K = 1 note that B is not only complemented (i.e., 1's complement) but an extra 1 is added as well; which makes B as the 2-s complement representation (i.,e., negative B) and the circuit is essentially doing subtraction [i.e., A + (-B]).

[Note: if one input of a 2-input XOR gate is 1 the output is the complement of the input. If one input to XOR is 0 then the output is equal to the input]



Now we see that our primitive computer can at least ADD (also SUBTRACT) and if we stretch our imagination that we have repetition control circuitry then multiplication (repeated addition) and division (repeated subtraction) are also possible.

You may appreciate the fact that all other higher-level mathematical computations are based on these four operations. Or to put in other ways the higher-level mathematical operations can be broken into lower-level operations; namely addition. SO, our computer is ready to solve mathematical problems provided it can carry out millions (or more) of lower-level computations in a second.

[Note: Don't think computer can solve all problems. There are many which cannot be solved at all or cannot be solved within a meaningful time-frame]