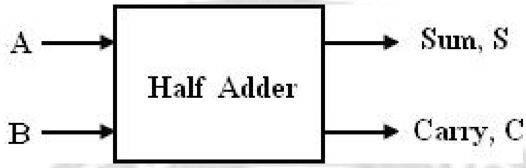
1.5.Two bits and three bits binary addition and Subtraction.

Half-Adder:

A half-adder is a combinational circuit that can be used to add two binary bits. It has two inputs that represent the two bits to be added and two outputs, with one producing the SUM output and the other producing the CARRY.



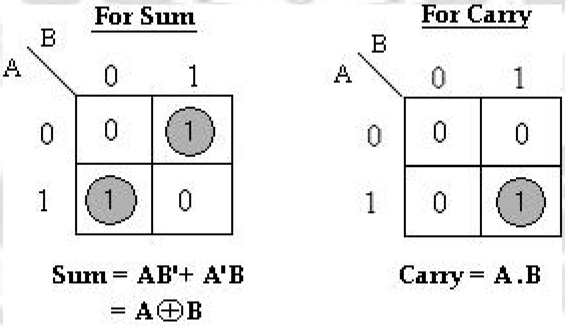
Block schematic of half-adder

The truth table of a half-adder, showing all possible input combinations and the corresponding outputs are shown below.

Truth Table:

Inputs		Outputs	
A	**B**	**Sum (S)**	**Carry (C)**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

K-map simplification:



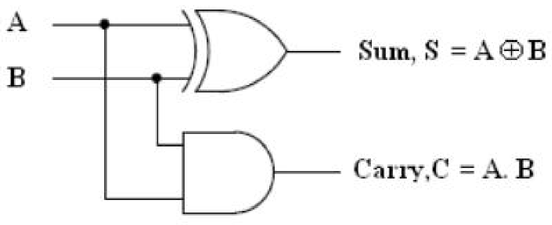
The Boolean expressions for the SUM and CARRY outputs are given by the equations,

Sum, S = A'B+ AB'= A ⊕ B

Carry, C = A . B

The first one representing the SUM output is that of an EX-OR gate, the second one representing the CARRY output is that of an AND gate.

The logic diagram of the half adder is,

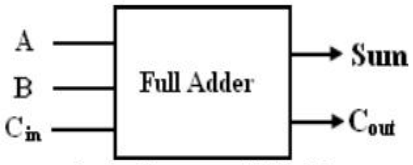


Logic Implementation of Half-adder

Full-Adder:

A full adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of three inputs and two outputs.

Two of the input variables, represent the significant bits to be added. The third input represents the carry from previous lower significant position. The block diagram of full adder is given by,



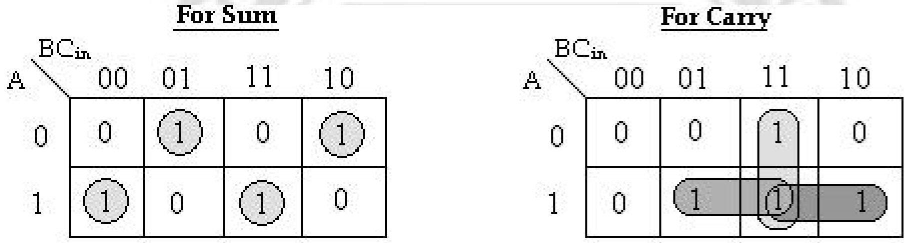
Block schematic of full-adder

The full adder circuit overcomes the limitation of the half-adder, which can be used to add two bits only. As there are three input variables, eight different input combinations are possible.

Truth Table:

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

K-map simplification:



For Sum

A \ BC_{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

For Carry

A \ BC_{in}	00	01	11	10
0	0	0	1	0
1	0	1	1	1

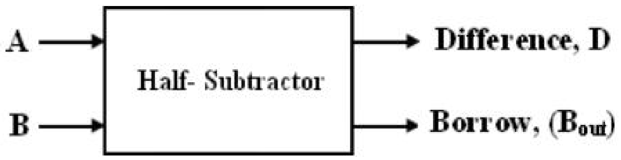
Sum, $S = A'B'C_{in} + A'BC'_{in} + AB'C'_{in} + ABC_{in}$

Carry, $C_{out} = AB + AC_{in} + BC_{in}$

Half -Subtractor

A *half-subtractor* is a combinational circuit that can be used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output.

The BORROW output here specifies whether a '1' has been borrowed to perform the subtraction.

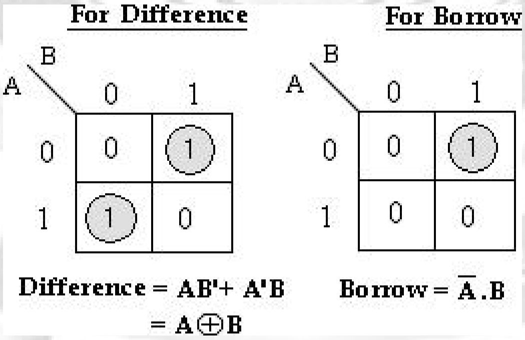


Block schematic of half-subtractor

The truth table of half-subtractor, showing all possible input combinations and the corresponding outputs are shown below.

Inputs		Outputs	
A	B	Difference (D)	Borrow (B_{out})
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K-map simplification:



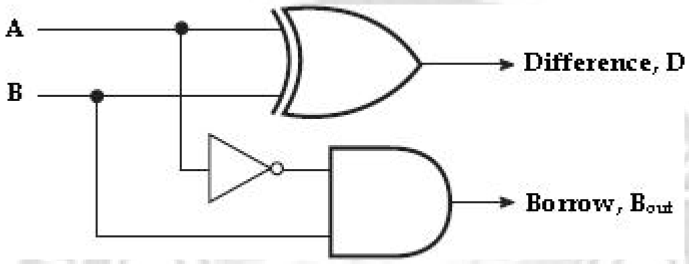
The Boolean expressions for the DIFFERENCE and BORROW outputs are given by the equations,

$$\text{Difference, } D = A'B + AB' = A \oplus B$$

$$\text{Borrow, } B_{out} = A' . B$$

The first one representing the DIFFERENCE (**D**)output is that of an exclusive- OR gate, the expression for the BORROW output (**B_{out}**) is that of an AND gate with input A complemented before it is fed to the gate.

The logic diagram of the half subractor is,



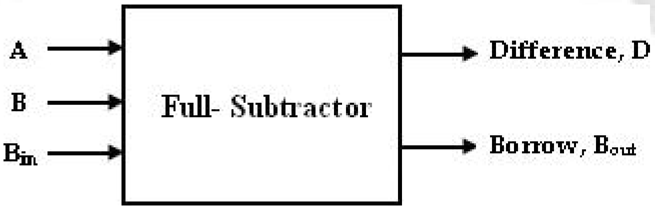
Logic Implementation of Half-Subtractor

Comparing a half-subtractor with a half-adder, we find that the expressions for the SUM and DIFFERENCE outputs are just the same. The expression for BORROW in the case of the half-subtractor is also similar to what we have for CARRY in the case of the half-adder. If the input A, ie., the minuend is complemented, an AND gate can be used to implement the BORROW output.

Full Subtractor:

A *full subtractor* performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into consideration whether a '1' has already been borrowed by the previous adjacent lower minuend bit or not.

As a result, there are three bits to be handled at the input of a full subtractor, namely the two bits to be subtracted and a borrow bit designated as B_{in}. There are two outputs, namely the DIFFERENCE output D and the BORROW output B_o. The BORROW output bit tells whether the minuend bit needs to borrow a '1' from the next possible higher minuend bit.

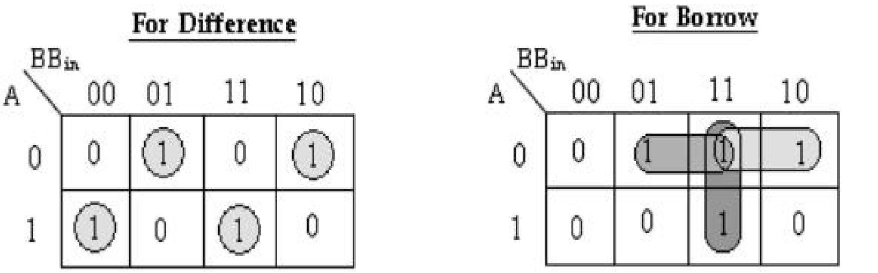


Block schematic of full-subtractor

The truth table for a full-subtractor is,

Inputs			Outputs	
A	B	B_{in}	Difference(D)	Borrow(B_{out})
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-map simplification:



For Difference

A \ BB_{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

For Borrow

A \ BB_{in}	00	01	11	10
0	0	1	1	1
1	0	0	1	0

Difference, $D = A'B'B_{in} + A'BB'_{in} + AB'B'_{in} + ABB_{in}$

Borrow, $B_{out} = A'B + A'B_{in} + BB_{in}$