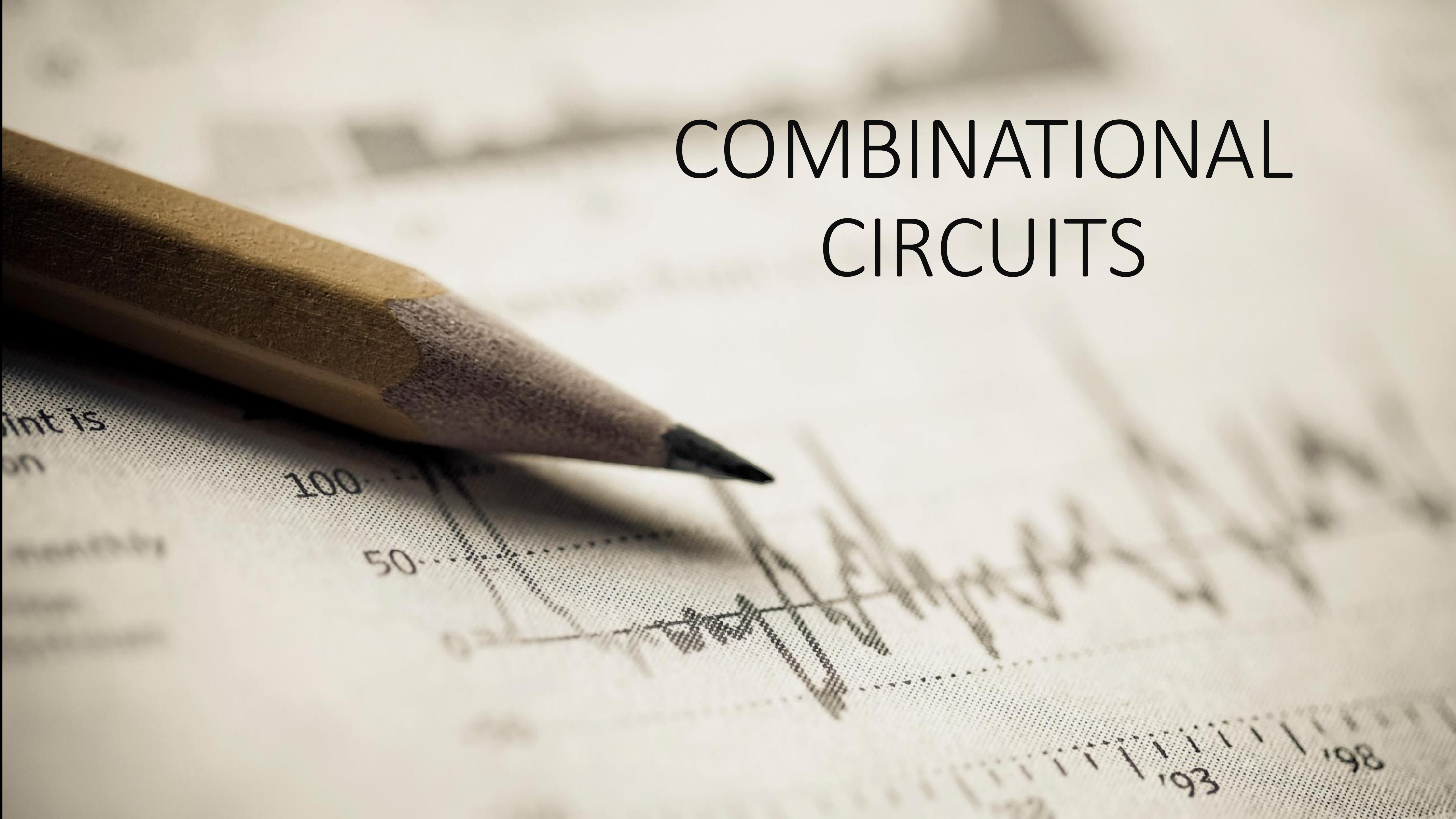
COMBINATIONAL CIRCUITS



Combinational logic circuits

- **Combinational circuits** are defined as the time independent circuits which do not depends upon previous inputs to generate any output are termed as combinational circuits. **Sequential circuits** are those which are dependent on clock cycles and depends on present as well as past inputs to generate any output.
- Examples – Encoder, Decoder, Multiplexer, Demultiplexer

<u>Combinational Circuit –</u>

1. In this output depends only upon present input.
2. Speed is fast.
3. It is designed easy.
4. There is no feedback between input and output.
5. This is time independent.
6. Elementary building blocks: Logic gates
7. Used for arithmetic as well as boolean operations.
8. Combinational circuits don’t have capability to store any state.
9. As combinational circuits don’t have clock, they don’t require triggering.
10. These circuits do not have any memory element.
11. It is easy to use and handle.

Block Diagram –

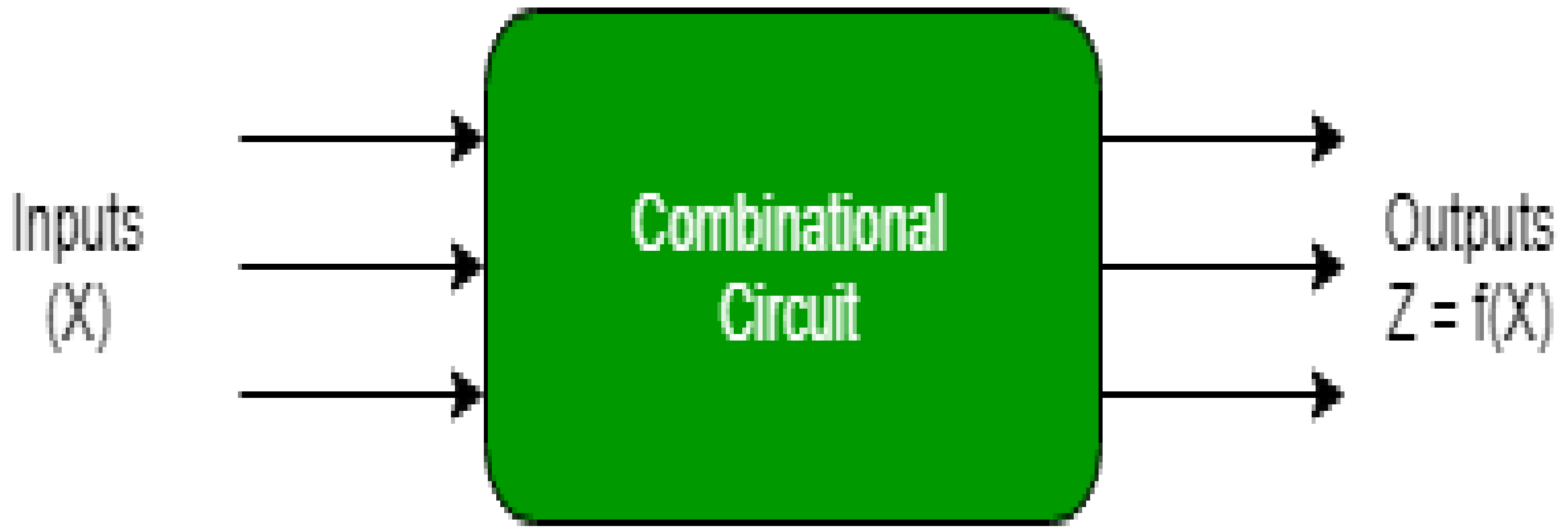


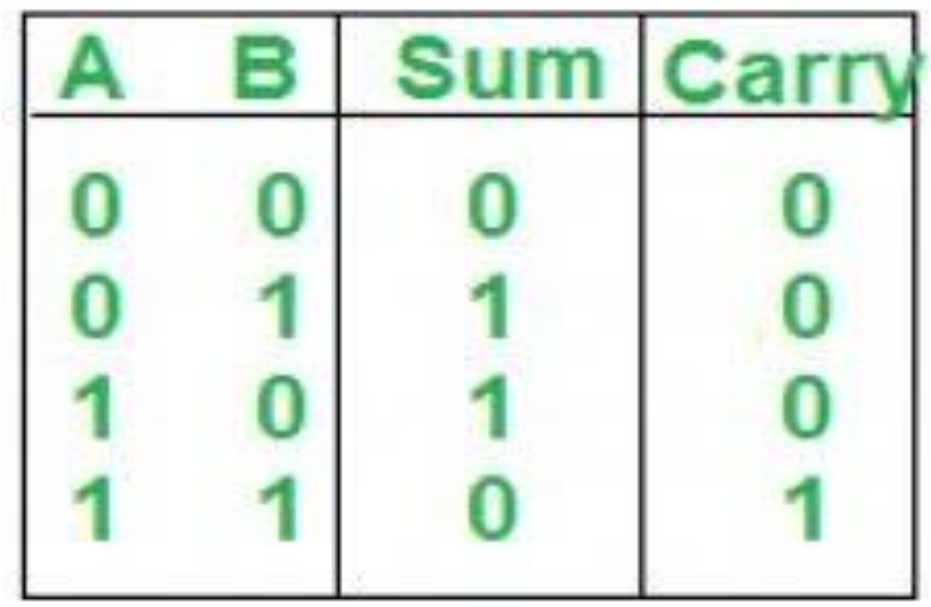
Figure: Combinational Circuits

HALF ADDER

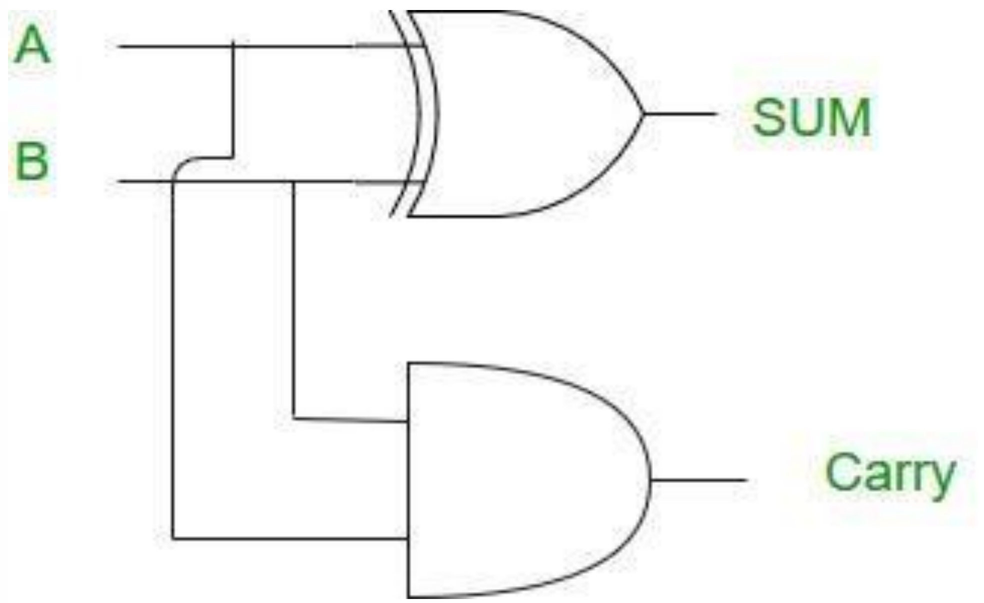
- Half adder is the simplest of all adder circuits. Half adder is a combinational arithmetic circuit that adds two numbers and produces a sum bit (s) and carry bit (c) both as output. The addition of 2 bits is done using a combination circuit called a Half adder. The input variables are augend and addend bits and output variables are sum & carry bits. A and B are the two input bits.
- let us consider two input bits A and B, then sum bit (s) is the X-OR of A and B. it is evident from the function of a half adder that it requires one X-OR gate and one AND gate for its construction

Truth Table:

Here we perform two operations Sum and Carry, thus we need two K-maps one for each to derive the expression.



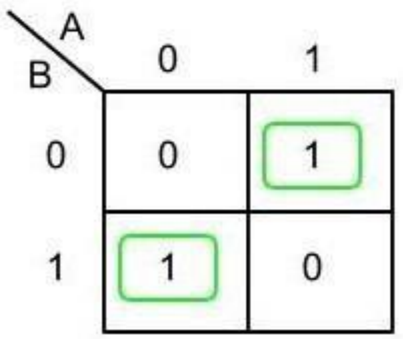
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



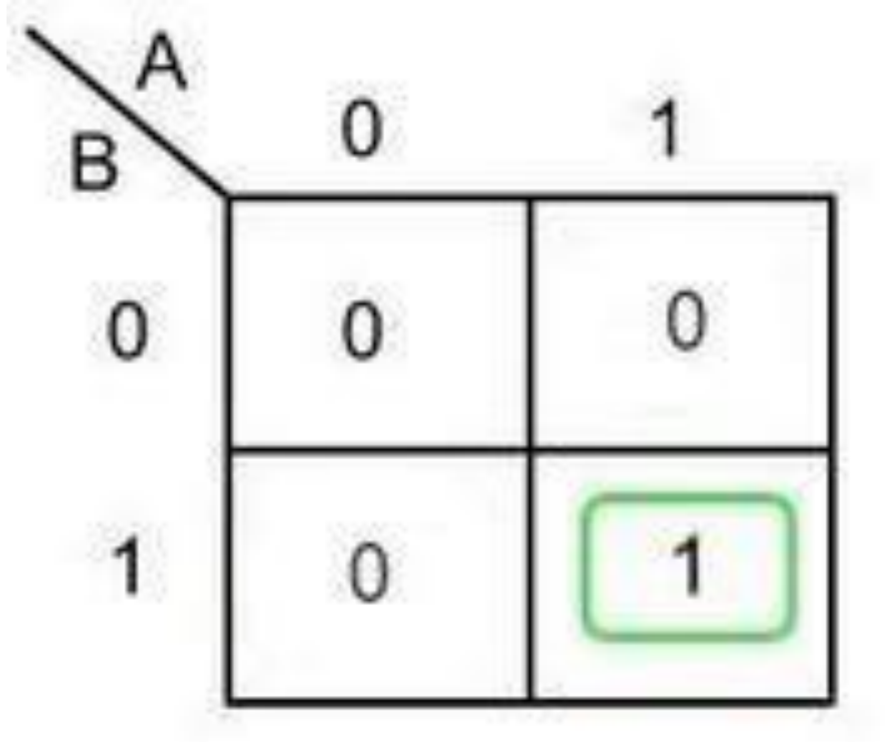
Sum = A XOR B

Carry = A AND B

Sum = A XOR B



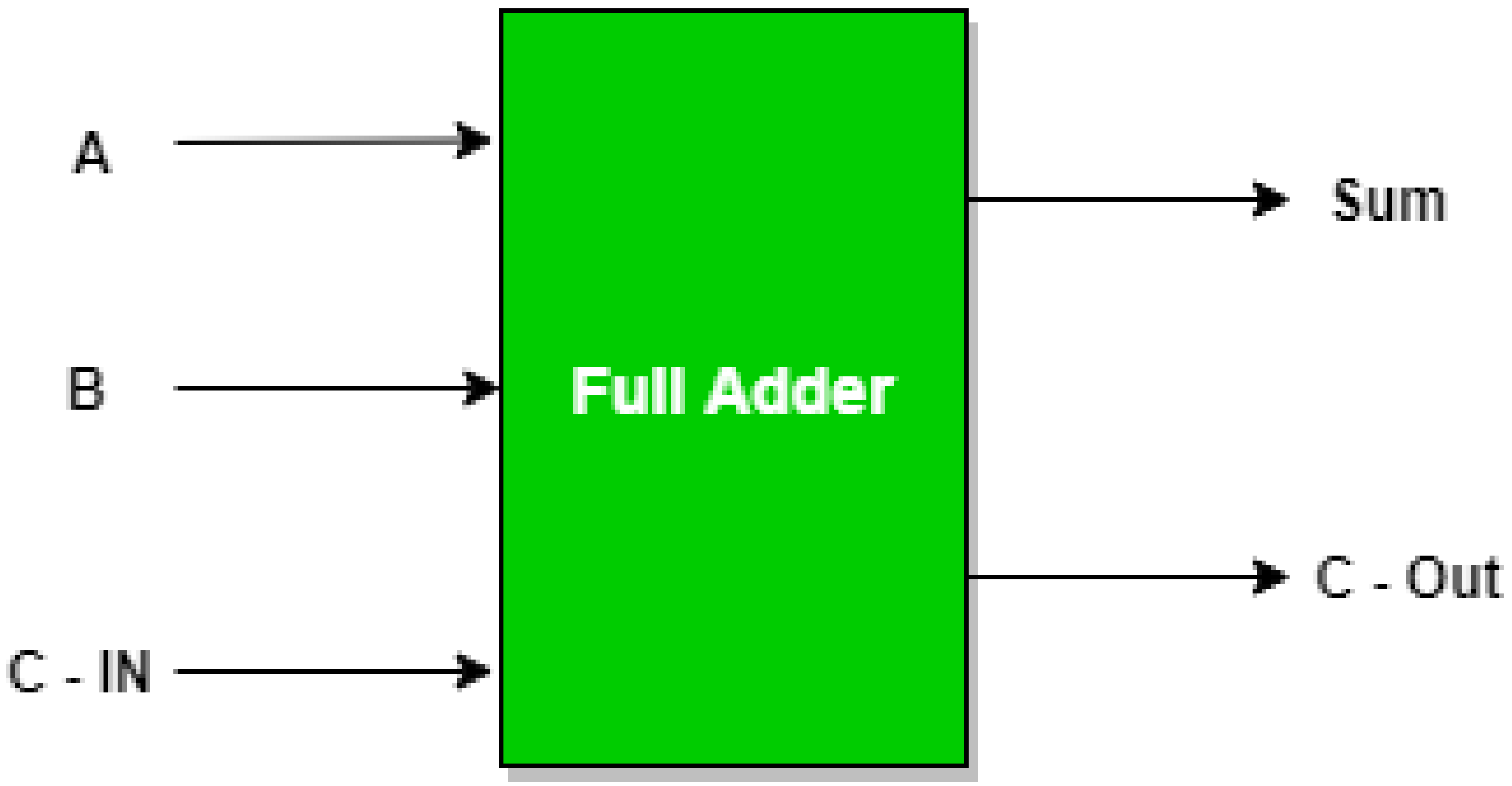
B \ A	0	1
0	0	1
1	1	0



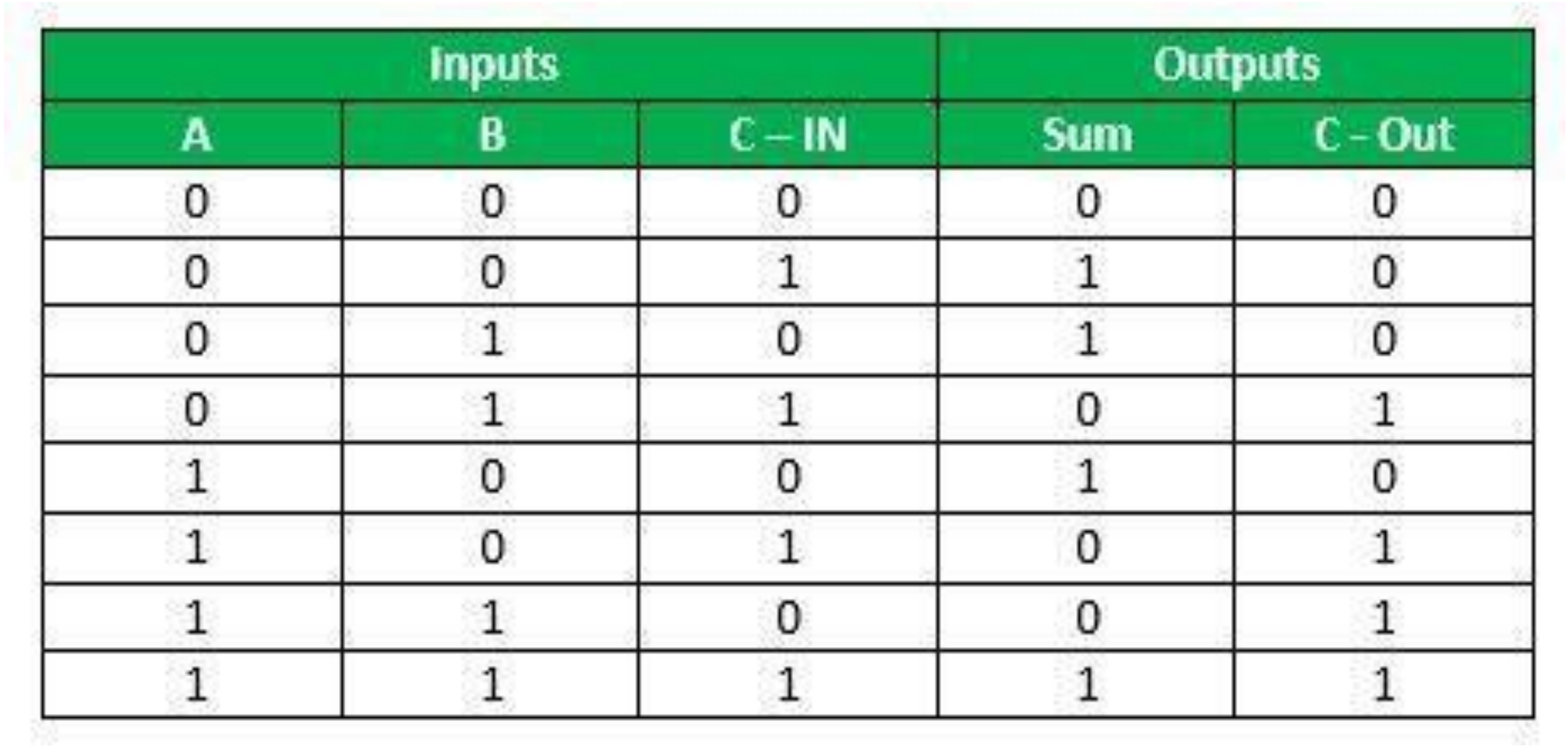
B \ A	0	1
0	0	0
1	0	1

FULL ADDER

- Full Adder is the adder that adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM. A full adder logic is designed in such a manner that can take eight inputs together to create a byte-wide adder and cascade the carry bit from one adder to another. we use a full adder because when a carry-in bit is available, another 1-bit adder must be used since a 1-bit half-adder does not take a carry-in bit. A 1-bit full adder adds three operands and generates 2-bit results.


A
B
C - IN
Full Adder
Sum
C - Out

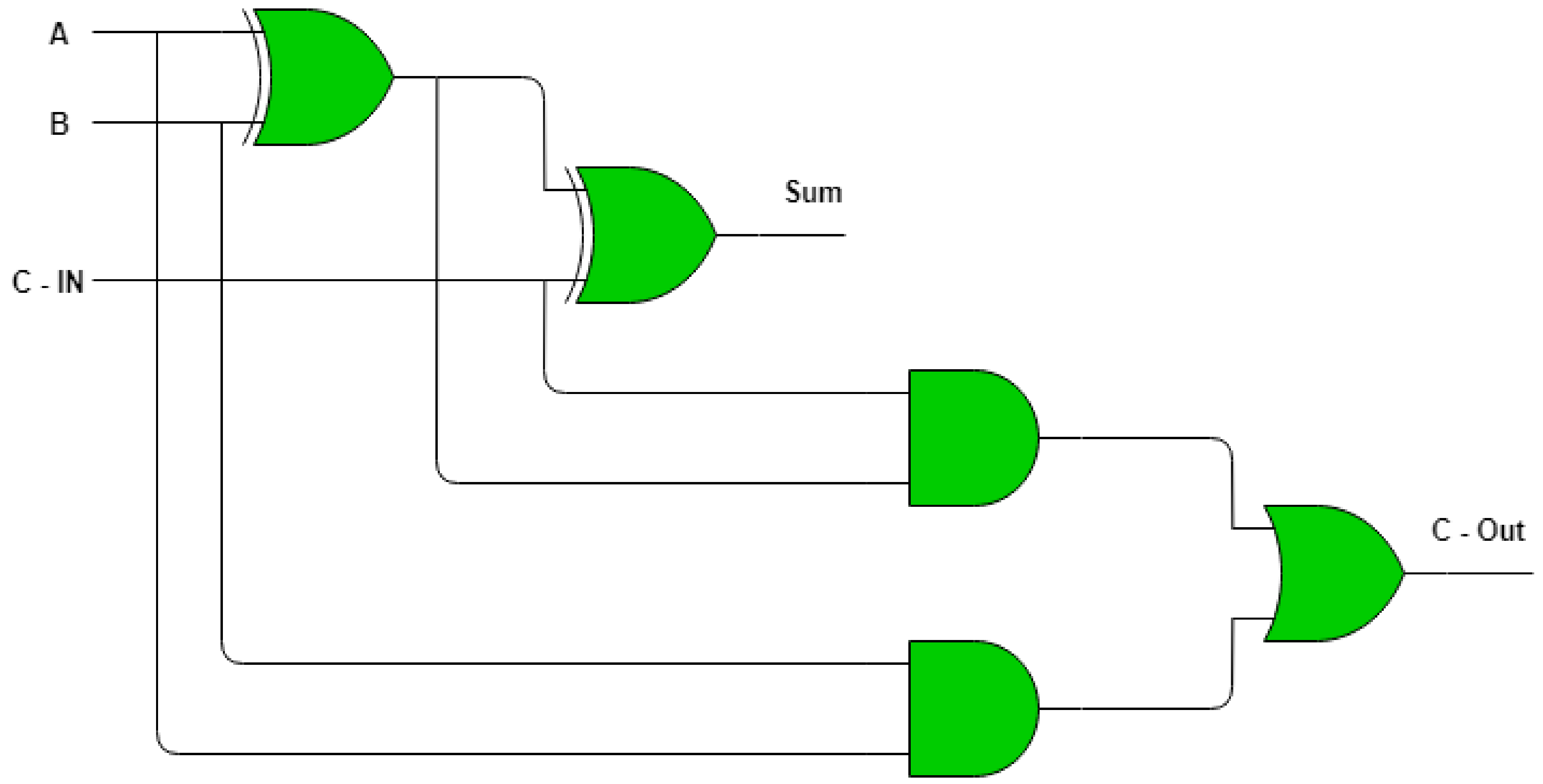
Full Adder Truth Table:



Inputs			Outputs	
A	B	C – IN	Sum	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

Logical Expression for SUM: = A’ B’ C-IN + A’ B C-IN’ + A B’ C-IN’ + A B C-IN = C-IN (A’ B’ + A B) + C-IN’ (A’ B + A B’) = C-IN XOR (A XOR B) = (1,2,4,7)
Logical Expression for C-OUT: = A’ B C-IN + A B’ C-IN + A B C-IN’ + A B C-IN = A B + B C-IN + A C-IN = (3,5,6,7)

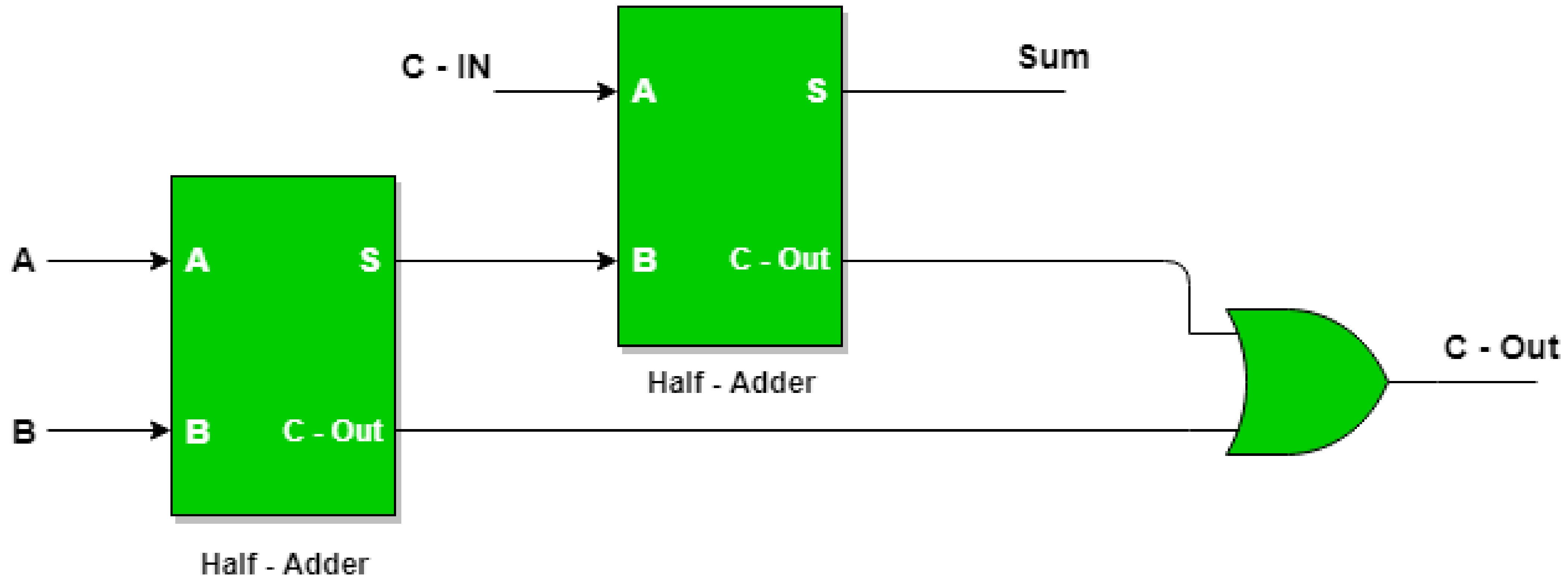
Another form in which C-OUT can be implemented: = A B + A C-IN + B C-IN (A + A’) = A B C-IN + A B + A C-IN + A’ B C-IN = A B (1 +C-IN) + A C-IN + A’ B C-IN = A B + A C-IN + A’ B C-IN = A B + A C-IN (B + B’) + A’ B C-IN = A B C-IN + A B + A B’ C-IN + A’ B C-IN = A B (C-IN + 1) + A B’ C-IN + A’ B C-IN = A B + A B’ C-IN + A’ B C-IN = AB + C-IN (A’ B + A B’)
Therefore COUT = AB + C-IN (A EX – OR B)



Full Adder logic circuit

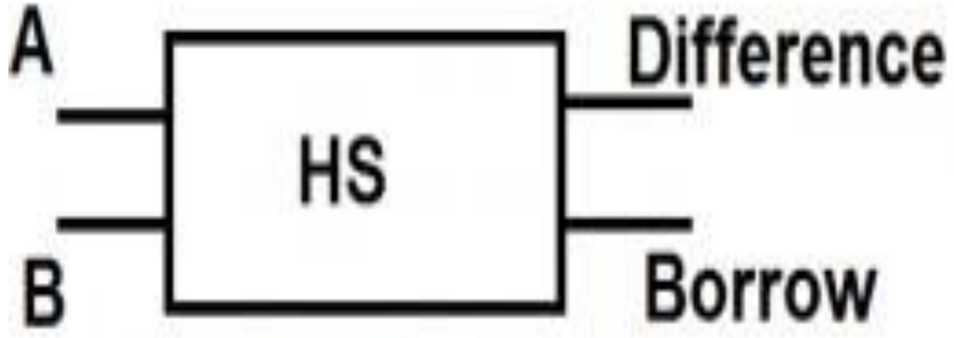
Implementation of Full Adder using Half Adders:

2 Half Adders and an OR gate is required to implement a Full Adder.



HALF SUBTRACTOR

- **Half Subtractor(HS):** Half subtractor is a combination circuit with two inputs and two outputs which is **difference** and **borrow**.
- It produces the difference between the two binary bits at the input and also produces an output (Borrow) to indicate if a 1 has been borrowed. In the subtraction (A-B), A is called a **Minuend bit** and B is called as **Subtrahend bit.**

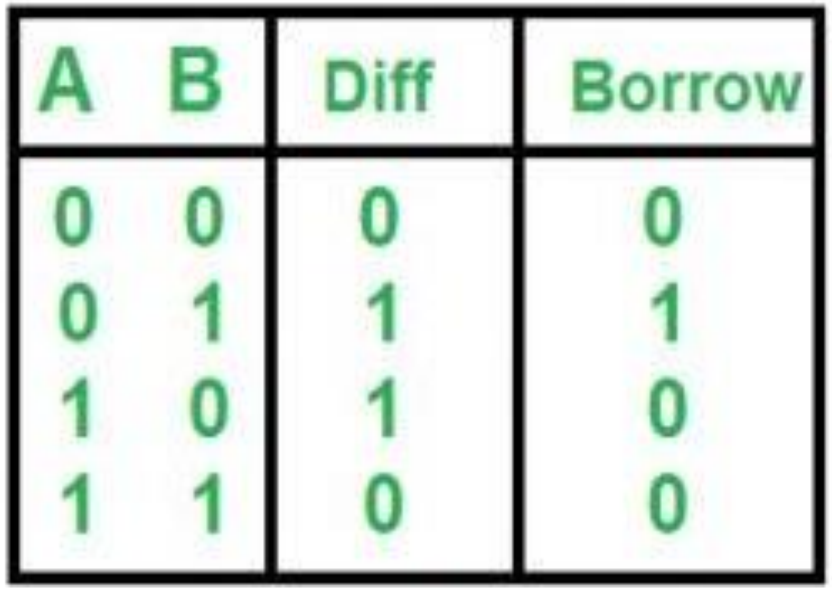


Half Subtractor

Truth table

Diff = A’B + AB’

Borrow = A’B

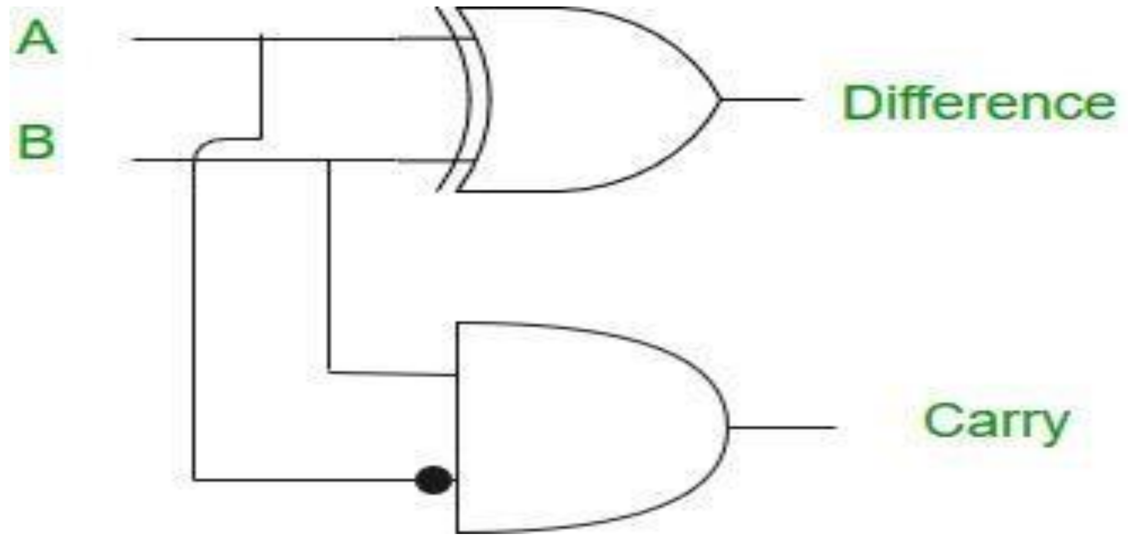


A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Logical Expression

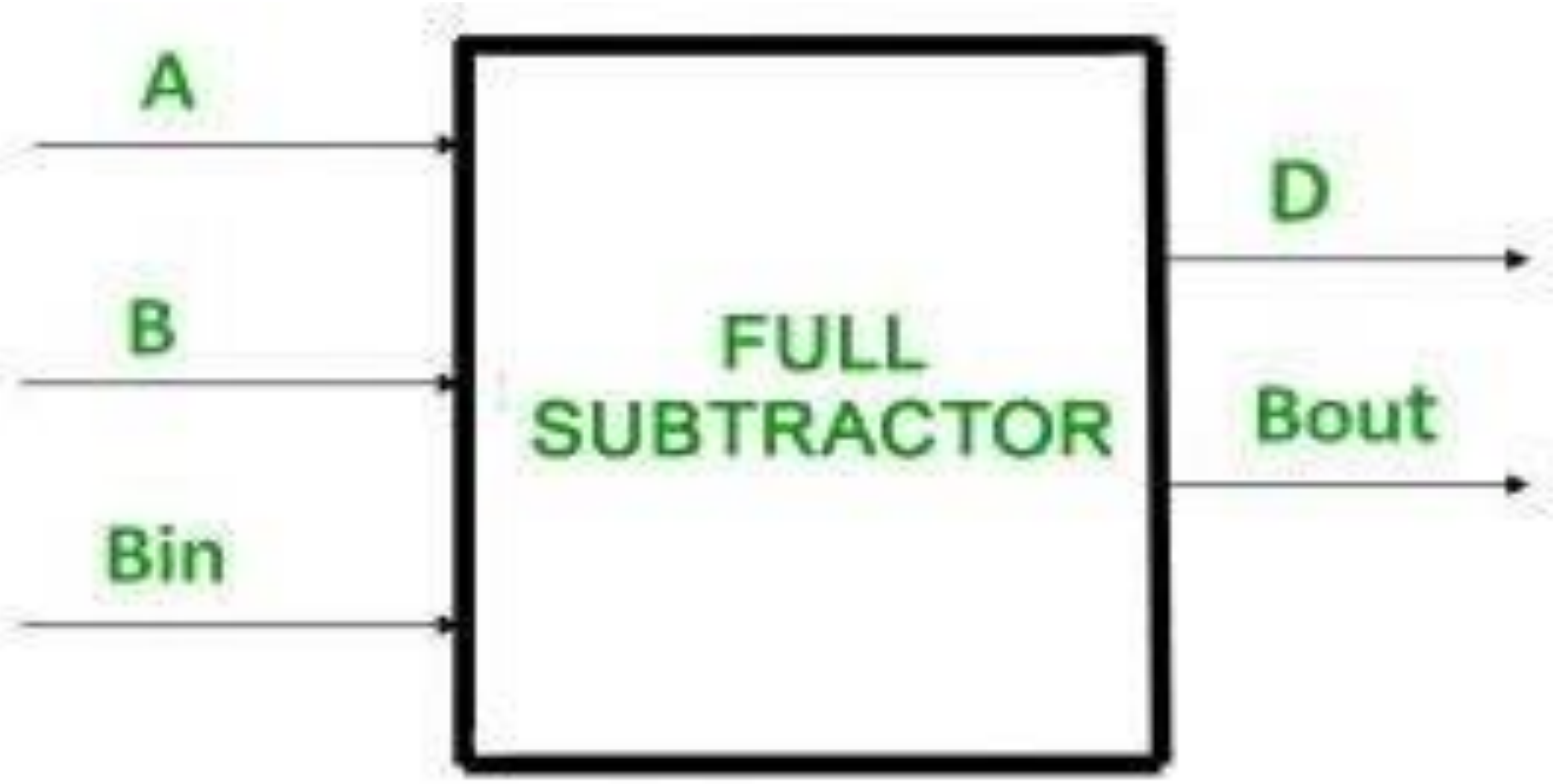
Difference = A XOR B

Borrow = A’B

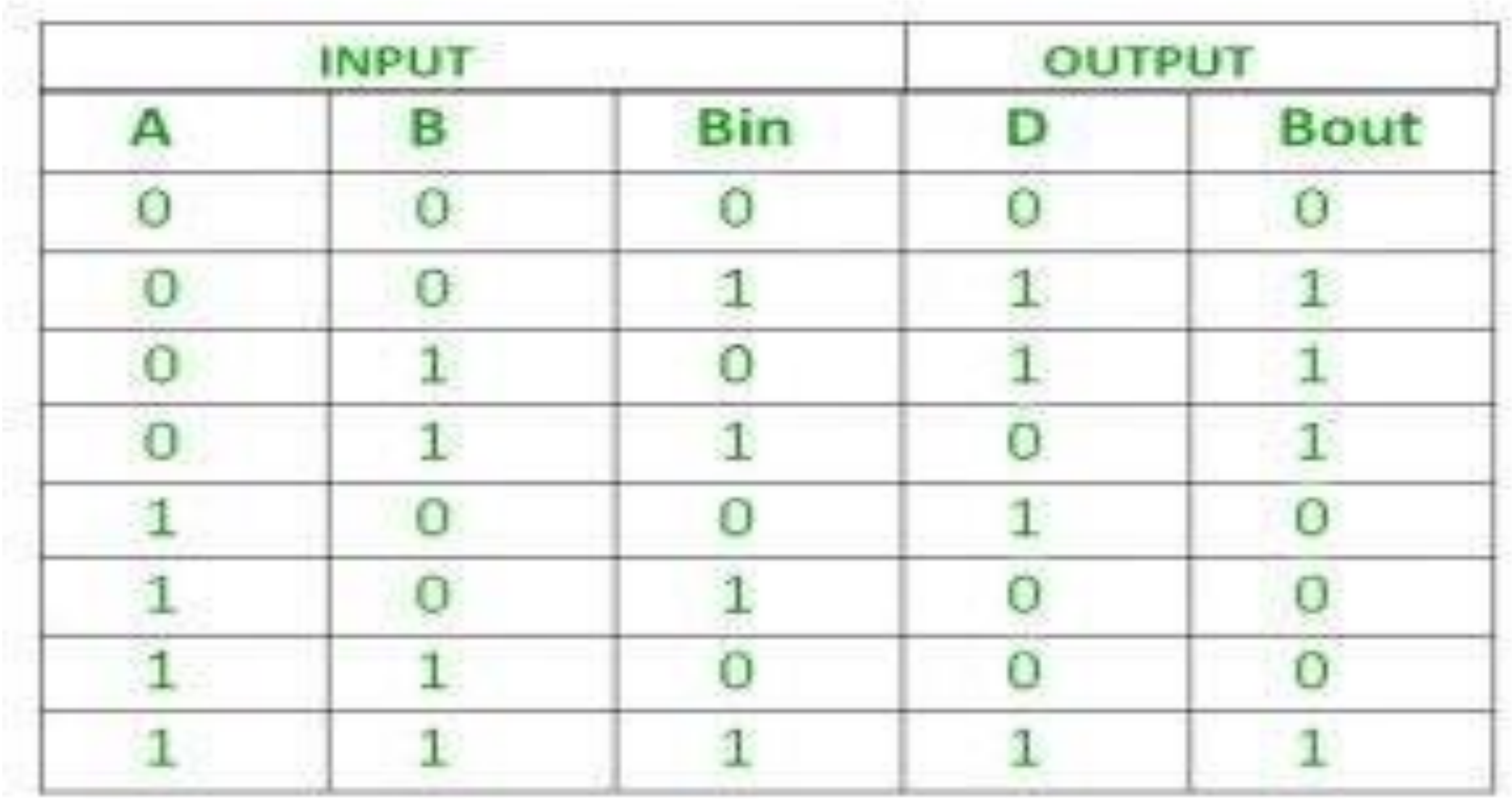


FULL SUBTRACTOR

- A full subtractor is a **combinational circuit** that performs subtraction of two bits, one is minuend and other is subtrahend, taking into account borrow of the previous adjacent lower minuend bit. This circuit **has three inputs and two outputs**. The three inputs A, B and Bin, denote the minuend, subtrahend, and previous borrow, respectively. The two outputs, D and Bout represent the difference and output borrow, respectively. Although subtraction is usually achieved by adding the complement of subtrahend to the minuend, it is of academic interest to work out the Truth Table and logic realisation of a full subtractor; x is the minuend; y is the subtrahend; z is the input borrow; D is the difference; and B denotes the output borrow. The corresponding maps for logic functions for outputs of the full subtractor namely difference and borrow.

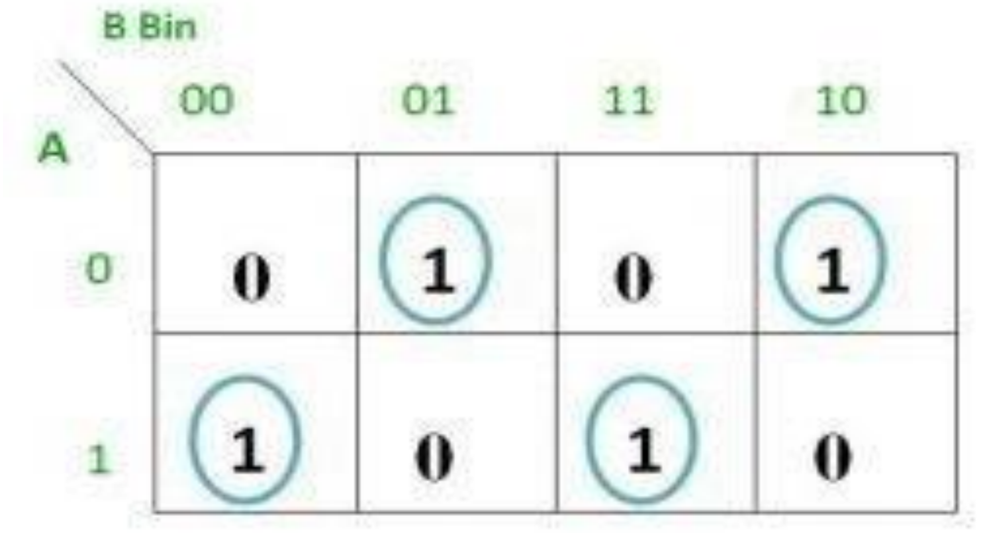

A
B
Bin
FULL SUBTRACTOR
D
Bout

Truth Table –



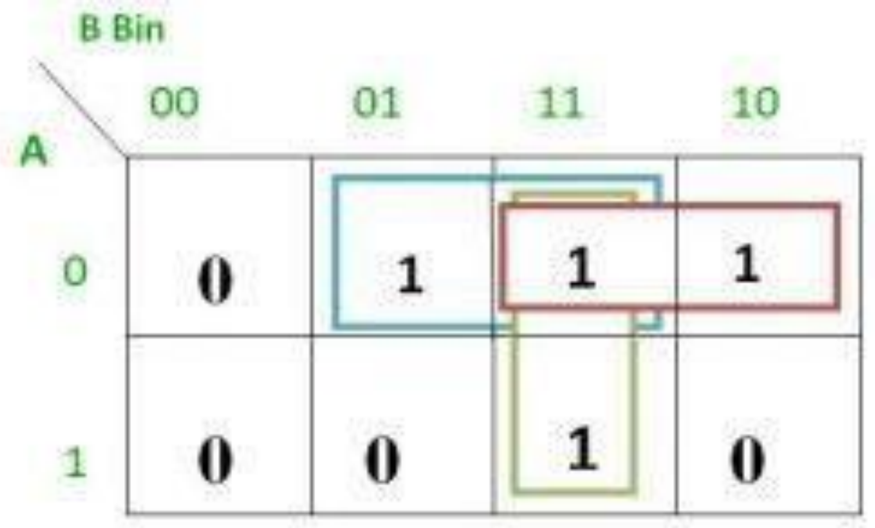
INPUT			OUTPUT	
A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

From above table we can draw the K-Map as shown for "difference" and "borrow".



A \ B Bin	00	01	11	10
0	0	1	0	1
1	1	0	1	0

D=A'B'Bin + AB'Bin' + A'BBin' + ABBin



A \ B Bin	00	01	11	10
0	0	1	1	1
1	0	0	1	0

Bout=A'Bin + A'B + BBin

Logical expression for difference –

D = A’B’Bin + A’BBin’ + AB’Bin’ + ABBin

= Bin(A’B’ + AB) + Bin’(AB’ + A’B)

= Bin(A XNOR B) + Bin’(A XOR B)

= Bin (A XOR B)’ + Bin’(A XOR B)

= Bin XOR (A XOR B)

= (A XOR B) XOR Bin

Logical expression for borrow –

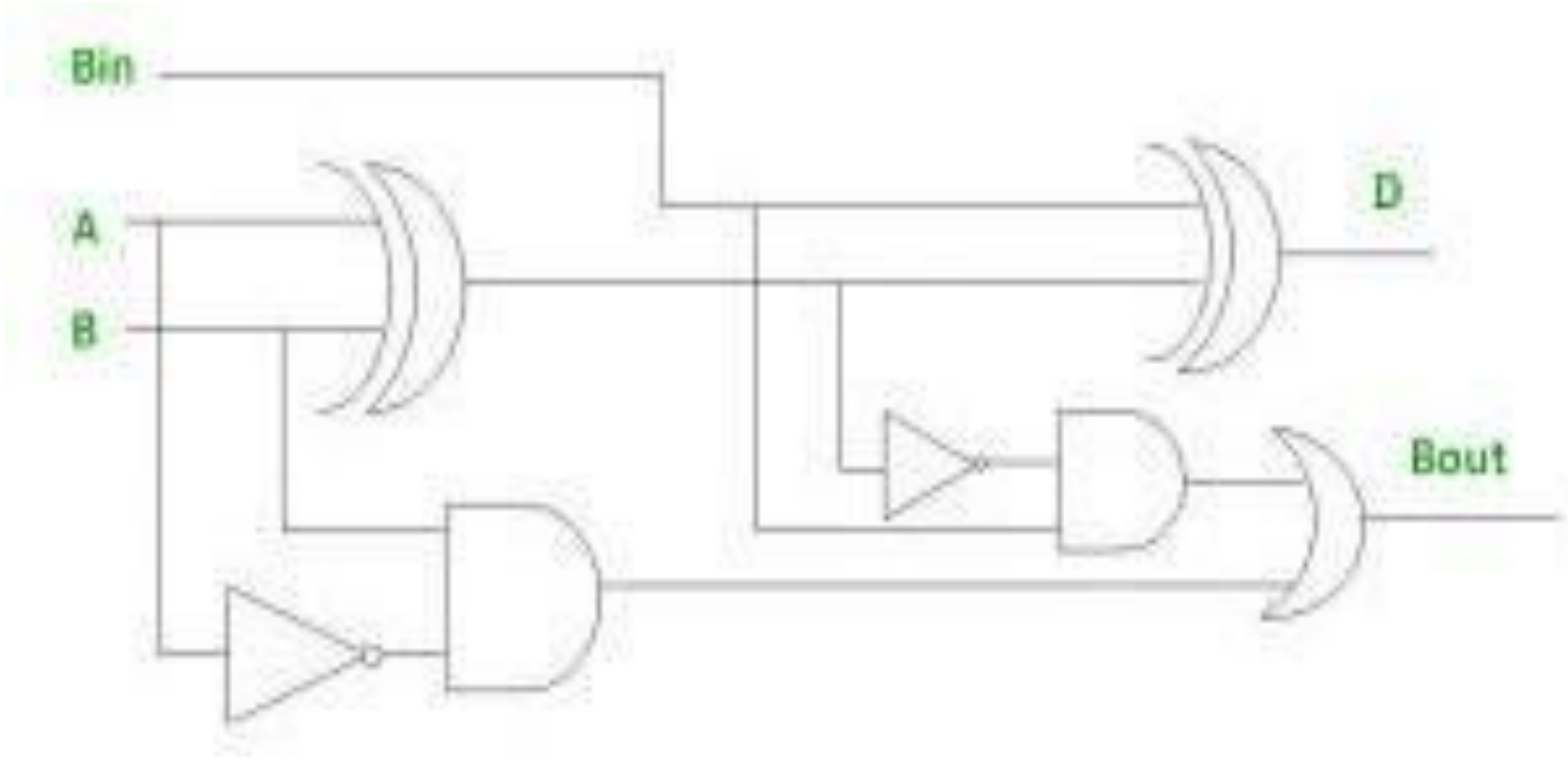
Bout = A’B’Bin + A’BBin’ + A’BBin + ABBin

= A’B’Bin +A’BBin’ + A’BBin + A’BBin + A’BBin + ABBin

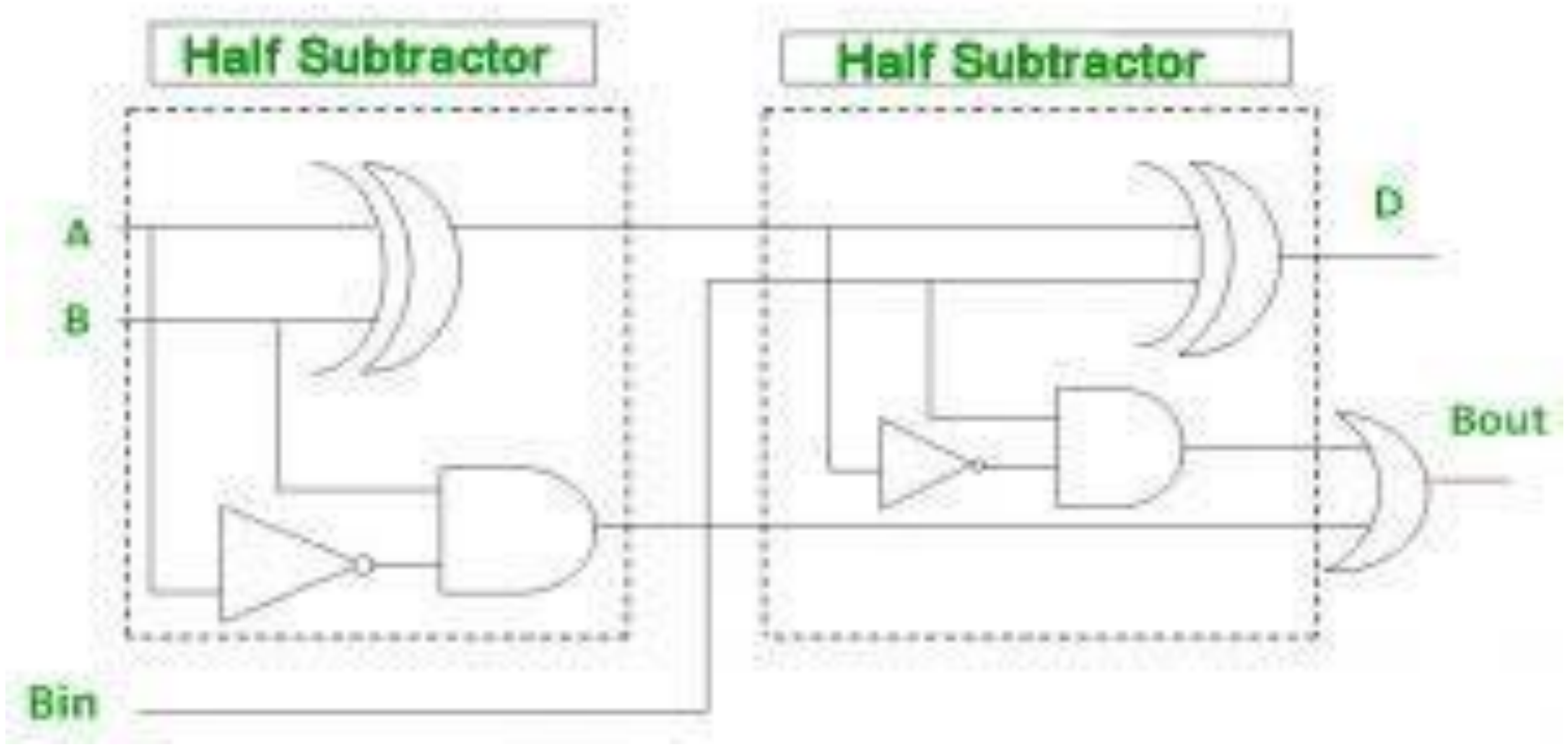
= A’Bin(B + B’) + A’B(Bin + Bin’) + BBin(A + A’)

= A’Bin + A’B + BBin

Logic Circuit for Full Subtractor –



Implementation of Full Subtractor using Half Subtractors – 2 Half Subtractors and an OR gate is required to implement a Full Subtractor.


Half Subtractor
Half Subtractor
A
B
D
Bout
Bin