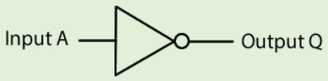
Boolean

Logic Gates

NOT gate - The output is the opposite of the input

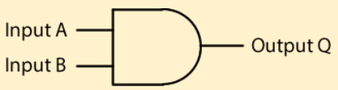
$$Q = \bar{A}$$
$$Q = NOT\ A$$



NOT truth table

Input	Output
0	1
1	0

AND gate - has two inputs and will have a true output if the two inputs are true otherwise the output will be false



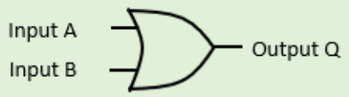
$$Q = A.B$$
$$Q = A\ AND\ B$$

AND truth table

Input A	Input B	Output
0	0	0
1	0	0
0	1	0
1	1	1

OR gate - has two inputs and will have a true output if either or both the inputs are true

c

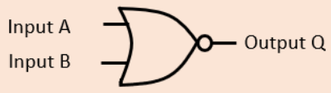


OR truth table

Input A	Input B	Output
0	0	0
1	0	1
0	1	1
1	1	1

NOR gate - has two inputs and will have a true output only if either or both the inputs are false

$$Q = \overline{A + B}$$
$$Q = NOT\ (A + B)$$

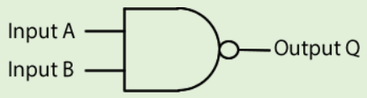


NOR truth table

Input A	Input B	Output
0	0	1
1	0	0
0	1	0
1	1	0

NAND gate - has two inputs and will have a true output if either or both the inputs are false

$$Q = \overline{A.B}$$
$$Q = NOT\ (A.B)$$

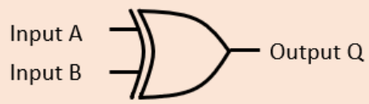


OR truth table

Input A	Input B	Output
0	0	1
1	0	1
0	1	1
1	1	0

XOR gate - has two inputs and will have a true output if either the inputs are true but not both

$$Q = A \oplus B$$
$$Q = A\ XOR\ B$$



OR truth table

Input A	Input B	Output
0	0	0
1	0	1
0	1	1
1	1	0

Boolean Identities

Commutative laws	$A + B = B + A$ $A.B = B.A$
Inverse law	$\bar{\bar{A}} = \mathrm{A}$
AND laws	$A.\bar{A} = 0$ $A.\ A = A$ $0.\ A = 0$ $1.\ A = A$
OR laws	$1 + A = 1$ $0 + A = A$ $A + A = A$ $A + \bar{A}\ = 1$
Associative laws	$(A.B).C = A.(B.C)$ $(A + B) + C = A + (B + C)$
Distributive law	$A.(B + C) = A.B + A.C$ $(A + B).(A + B) = A.A + B.B + A.B + A.B$
More identities	$A + A.B = A$ $A.(A + B) = A$
De Morgan's law	$\overline{\bar{A}.\ \bar{B}} = \mathrm{A} + \mathrm{B}$ $\overline{\bar{A} + \bar{B}} = \mathrm{A}.\mathrm{B}$

Applying De Morgan's Law:

1. Apply NOT operator the whole expression — $\overline{\overline{\bar{A}.\bar{B}}} = \bar{A}\ .\ \bar{B}$
2. Switch the operator (If the operator is AND switch to OR operator, If the operator is OR switch to AND operator) — $\bar{A} + \bar{B}$
3. NOT the individual terms — $\mathrm{A} + \mathrm{B}$

Order of operation

1. Brackets
2. NOT
3. AND
4. OR
5. XOR

Converting a truth table to a logic circuit

There is a general approach to converting a truth table into a logic circuit.

We consider only the lines with an output of 1.
We take in the input of each and then AND.

We then OR between each statement such that
(NOT A AND B) OR (A AND NOT B). We can then draw the logic circuit.

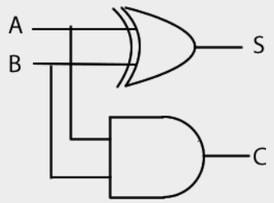
Worked example: What is the logic circuit for the following truth table

Input - A	Input - B	Output
0	0	0
1	0	1
0	1	0
1	1	1

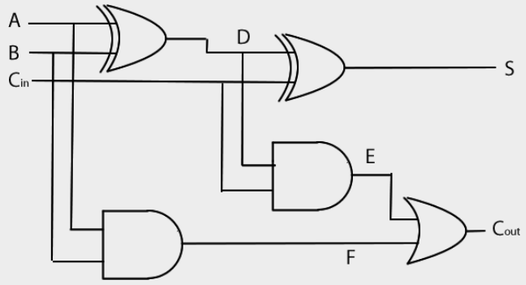
(A AND NOT B) OR (A AND A)

Half adder - A half adder has two bits as inputs (A and B) and adds the two bits and outputs two bits the sum (S) and the carry (C). It is made up of an AND and an XOR gate

Input A		***Input B***		***C (carry)***	***S (sum)***
0	*+*	*0*	*=*	*0*	*0*
1	*+*	*0*	*=*	*0*	*1*
0	*+*	*1*	*=*	*0*	*1*
1	*+*	*1*	*=*	*1*	*0*



Full Adder - A full adder has three bits as inputs one of which is the carry bit. It adds the three bits and outputs two bits the sum (S) and the carry (C). It is made up of an AND, XOR and OR gates

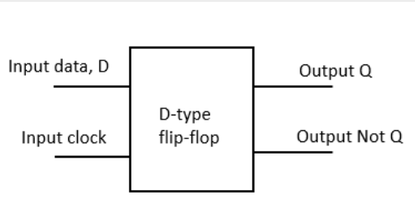


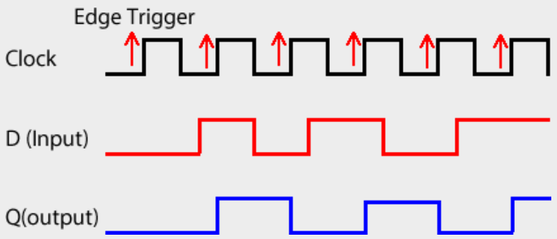
A		***B***		***Cin***		***Cout***	***S***	***D***	***E***	***F***
0	*+*	*0*	*+*	*0*	*=*	*0*	*0*	*0*	*0*	*0*
0	*+*	*0*	*+*	*1*	*=*	*0*	*1*	*0*	*0*	*0*
0	*+*	*1*	*+*	*0*	*=*	*0*	*1*	*1*	*0*	*0*
0	*+*	*1*	*+*	*1*	*=*	*1*	*0*	*1*	*1*	*0*
1	*+*	*0*	*+*	*0*	*=*	*0*	*1*	*1*	*0*	*0*
1	*+*	*0*	*+*	*1*	*=*	*1*	*0*	*1*	*1*	*0*
1	*+*	*1*	*+*	*0*	*=*	*1*	*0*	*0*	*0*	*1*
1	*+*	*1*	*+*	*1*	*=*	*1*	*1*	*0*	*0*	*1*

D-type flip flops

- A flip-flop can store the value of a bit.
- D (delay)-type flip flops are used to store one bit and flip between two states: 1 and 0.

- The inputs are a control value (0 or 1) and also a clock value (0 or 1) that changes the state at a regular rate.
- For a positive edge triggered flip-flop the output state can only change when the clock changes from 0 to 1. If D is in the same state as it was on the previous edge then the output is unchanged. However, if D has changed state, the output state will change to the value of D.





D	*Clock*	*Q*
0	↑	*0*
1	↑	*1*