Contents

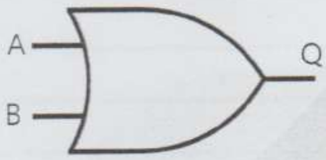
Digital Logic Gates and Truth Tables

Basic Logic Gates

There are four basic logic gates OR, AND, NOT, XOR

OR Gate

Represents logical addition (+). The OR gate is an electronic circuit that gives a high output (1) if one or more of its inputs are high. A plus (+) is used to show the OR operation.

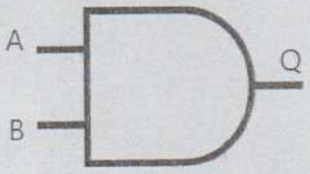


$Q = A + B$

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

AND Gate

Represents logical multiplication. The AND gate is an electronic circuit that gives a high output (1) only if all its inputs are high. A dot (.) is used to show the AND operation i.e. A.B. Bear in mind that this dot is sometimes omitted i.e. AB

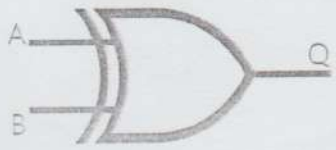


$Q = A.B$

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

XOR Gate

Represents logical exclusive addition. The 'Exclusive-OR' gate is a circuit which will give a high output if either, but not both, of its two inputs are high. An encircled plus sign (⊕) is used to show the EOR operation.

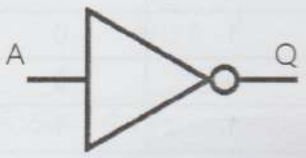


$Q = A \oplus B$

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

NOT Gate

The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an inverter. If the input variable is A, the inverted output is known as NOT A. This is also shown as A', or A with a bar over the top, as shown at the outputs. The NAND logic gate can be configured to produce a NOT gate. It can also be done using NOR logic gates in the same way.



A	Q
0	1
1	0

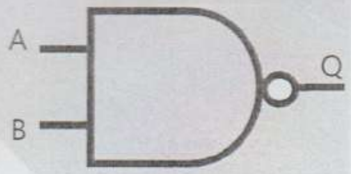
$Q = \overline{A}$

COMBINATIONAL GATES

These are formed by combining the basic logic gates except not gate, with not gate.

NAND Gate

This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are high if any of the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

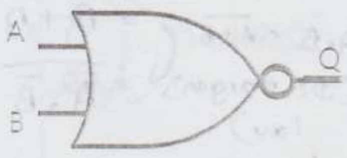


A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

$Q = \overline{A.B}$

NOR Gate

This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are low if any of the inputs are high. The symbol is an OR gate with a small circle on the output. The small circle represents inversion.

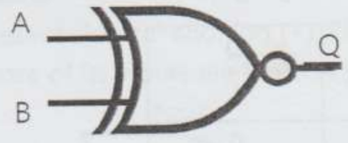


A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

$Q = \overline{A + B}$

XNOR Gate

The 'Exclusive-NOR' gate circuit does the opposite to the EOR gate. It will give a low output if either, but not both, of its two inputs are high. The symbol is an EXOR gate with a small circle on the output. The small circle represents inversion.



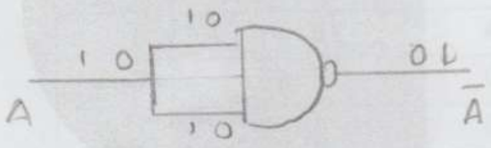
$Q = \overline{A \oplus B}$

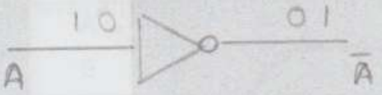
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	1

Universal Gates

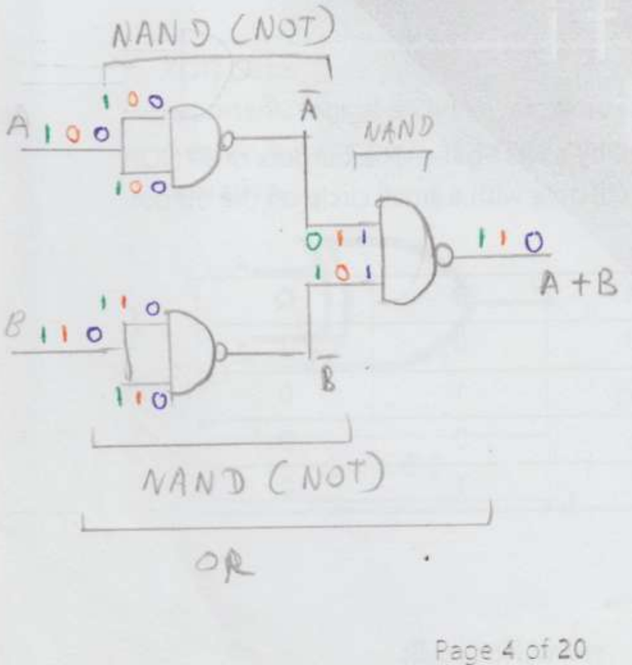
The **NAND gate and the NOR gate** can be said to be universal gates since combinations of them can be used to accomplish any of the basic operations and can thus produce an inverter, an OR gate or an AND gate. In practice, this is advantageous since NAND and NOR gates are **economical** and easier to fabricate and are the basic gates used in all IC digital logic families.

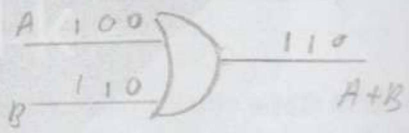
NOT using NAND





OR using NAND



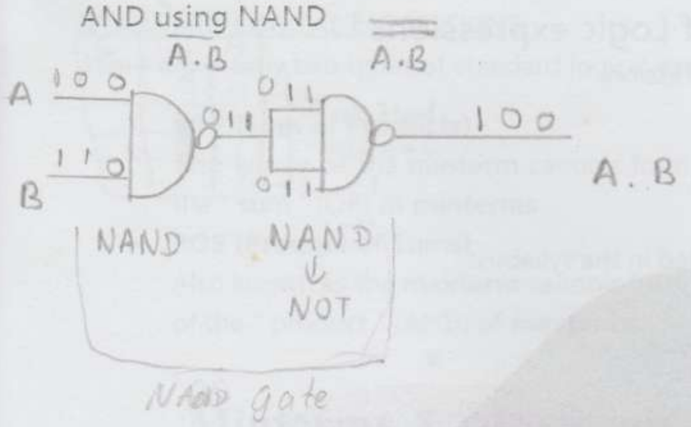


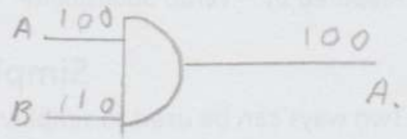
$= A + B$

$\overline{A}\cdot\overline{B} = \overline{A+B}$ $= \overline{\overline{A+B}}$

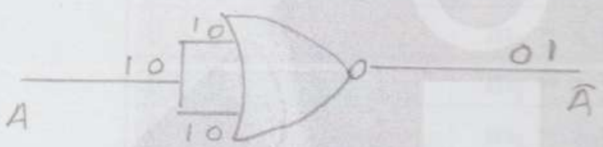
(De margan's law) $= \overline{\overline{A}\cdot\overline{B}}$

AND using NAND

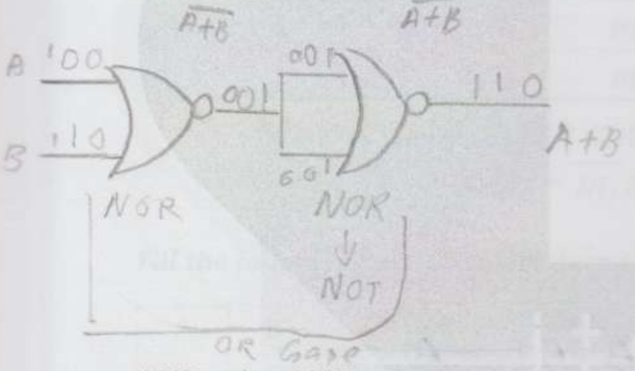


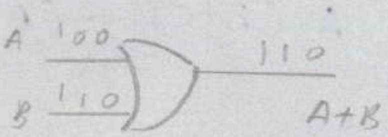


NOT using NOR

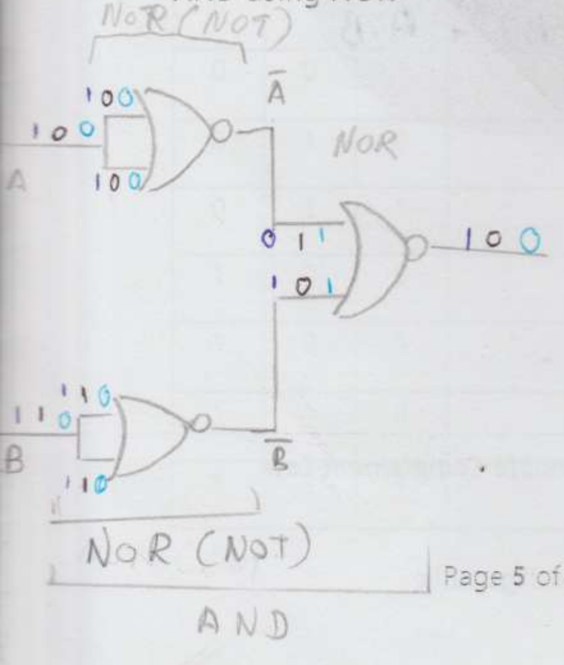


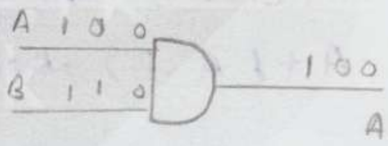
OR using NOR





AND using NOR





$$\overline{A}+\overline{B}=\overline{A.B}$$

(De Morgans Law)

$= A.B$ ← AND

$= \overline{\overline{A.B}}$ ← AND using NAND

$= \overline{\overline{A}+\overline{B}}$ ← AND using NOR

No of inputs $= 2^2 = 4$

Prepared By – Teran Subasinghe

Input 1	Input 2	
A	B	Q
0	0	
0	1	
1	0	
1	1	

half 0 and half 1

Simplification of Logic expressions

Two ways can be used to simplify a Boolean expression:

- Laws of Boolean Algebra
- Karnaugh Maps

Laws of Boolean Algebra

Following are the laws of Boolean algebra included in the syllabus:

Commutative

$A+B = B+A$

$A.B = B.A$

Associative

$(A+B)+C = A+(B+C)$

$(A.B).C = A.(B.C)$

Distributive

$A+(B.C) = (A+B).(A+C)$

$A.(B+C) = (A.B)+(A.C)$

Identity

$A+0 = A$ $\quad A.0 = 0$

$A+1 = 1$ $\quad A.1 = A$

Redundancy

$A+(A.B) = A$ $\quad A.(A+B) = A$

$A+(\bar{A}.B) = A+B)$ $\quad A.(\bar{A}+B) = A.B$

De Morgan's

*In addition to the above laws following axioms are used

**Laws from TIM: Idempotent Law, Inverse/Complement Law, Double Complement Law*

STANDARD LOGICAL EXPRESSIONS

There are mainly two types of standard logical expressions:

- **SOP (Sum of Products)**
  Also known as the **minterm** canonic form or canonic sum function. A function in the form of the " sum " (OR) of **minterms**.
- **POS (Product of Sums)**
  Also known as the **maxterm** canonic form or canonic product function. A function in the form of the " product " (AND) of **maxterms**.

Minterms & Maxterms for 3 variables

x	y	z	Index	Minterm	Maxterm
0	0	0	0	$m_0 = \overline{x}\,\overline{y}\,\overline{z}$	$M_0 = x + y + z$
0	0	1	1	$m_1 = \overline{x}\,\overline{y}\,z$	$M_1 = x + y + \overline{z}$
0	1	0	2	$m_2 = \overline{x}\,y\,\overline{z}$	$M_2 = x + \overline{y} + z$
0	1	1	3	$m_3 = \overline{x}\,y\,z$	$M_3 = x + \overline{y} + \overline{z}$
1	0	0	4	$m_4 = x\,\overline{y}\,\overline{z}$	$M_4 = \overline{x} + y + z$
1	0	1	5	$m_5 = x\,\overline{y}\,z$	$M_5 = \overline{x} + y + \overline{z}$
1	1	0	6	$m_6 = x\,y\,\overline{z}$	$M_6 = \overline{x} + \overline{y} + z$
1	1	1	7	$m_7 = x\,y\,z$	$M_7 = \overline{x} + \overline{y} + \overline{z}$

Maxterm M_i is the complement of minterm m_i

$$M_i = \overline{m_i} \text{ and } m_i = \overline{M_i}$$

Fill the following table with Minterms and Maxterms based on A, B and C.

A	B	C	Minterm	Maxterm
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Consider the following Truth table.

A	B	C	F (Output)
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

SOP (Sum of Products)

The function has **value 1** for the combinations shown, therefore:

$F(A, B, C) =$

POS (Product of Sums)

The function has **value 0** for the combinations shown, therefore:

$F(A, B, C) =$

Simplify Logic Expressions (Boolean Expression Simplification)

Two ways can be used to simplify a Boolean expression:

- Laws of Boolean Algebra
- Karnaugh Maps

Using Laws to Simplify the following expressions

Using Karnaugh Maps to Simplify
K-map has everything in a truth table.

Two inputs K-Map

Three inputs K-Map

Four inputs K-Map

Grouping Rules
1. *No diagonals.*
2. *Only power of 2 number of cells in each group.*
3. *Groups should be as large as possible.*
4. *Every "1" must be in at least one group.*
5. *Overlapping allowed.*
6. *Wrap around allowed.*
7. *Fewest number of groups possible.*

Simplify the following expressions using Karnaugh Maps:

CPU and Memory with Logic Gates

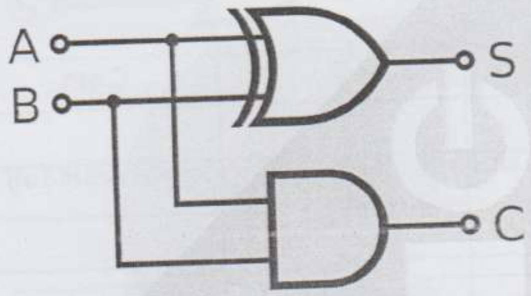
CPU uses combinational logic gates while memory uses sequential logic gates.

BUILDING BLOCKS OF CPU (COMBINATIONAL CIRCUITS)

CPU consist of Half Adder and Full Adder Circuits to perform calculations.

Half Adder

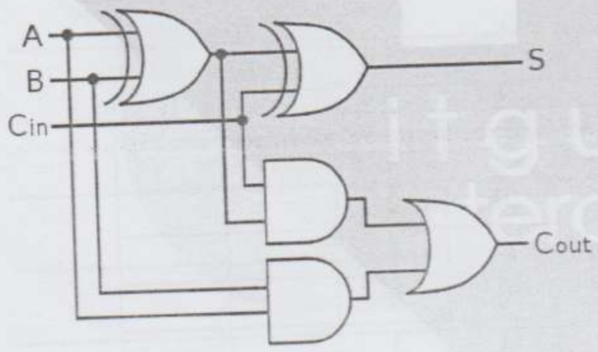
With the help of half adder, we can design circuits that are capable of performing simple addition with the help of logic gates.



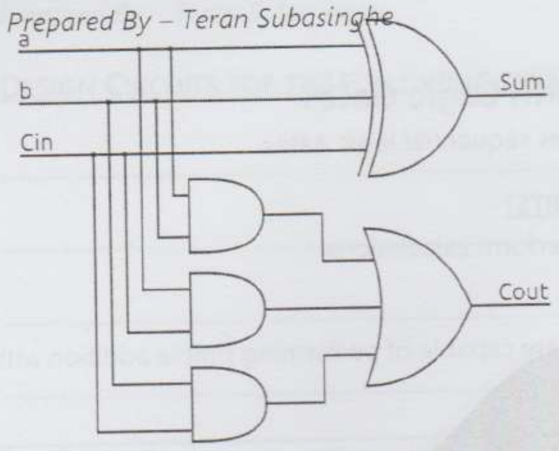
A	B	S	C
0	0		
0	1		
1	0		
1	1		

Full Adder

The main difference between a half-adder and a full-adder is that the full-adder **has three inputs** and two outputs. The first two inputs are A and B and the third input is an input carry designated as Carry In. When a full adder logic is designed we will be able to string eight of them together to create an 8-bit adder and cascade the carry bit from one adder to the next. However there are two versions which performs the same function.



A	B	C in	S	C out
0	0			
0	1			
1	0			
1	1			



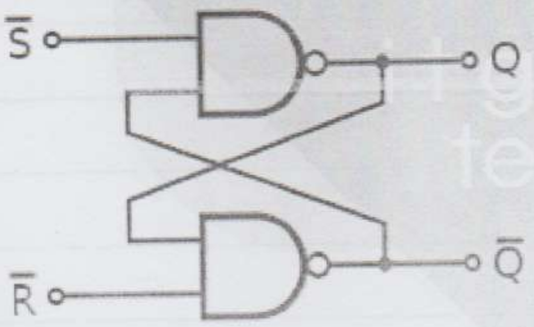
A	B	C in	S	C out
0	0			
0	1			
1	0			
1	1			

Storing Bits in Digital Circuits (Memory – Sequential Circuits)

Combinational circuits: Output depends only on the input of that time.

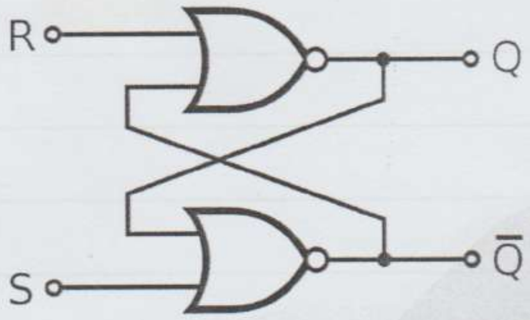
Sequential Circuit: Output depends not only on the present inputs but also on the previous inputs and outputs. This type of circuit is required to perform sequence of actions without getting any further inputs. Use for memory storage (SRAM)

SR Flipflop using NAND gates



$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
1	0		
0	0		
0	1		
0	0		
1	1		

RS Flipflop using NOR gates



S	R	Q	$\bar{Q}$
1	0	1	0
0	0		
0	1		
0	0		
1	1		

But How Memory is Formed with This?