DIGITAL ELECTRONICS — FUNDAMENTAL NOTES

---

1. LOGIC FUNCTIONS & LOGIC GATES

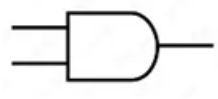
- Logic gates are the basic building blocks of digital circuits.
- They operate on *binary values* — **0 (LOW) (FALSE)** and **1 (HIGH) (TRUE)**.

1.1 Basic Logic Gates

A) AND Gate

- Output is **1 only when all inputs are 1**.

Symbol



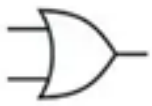
Truth Table

A	B	Y = A.B
0	0	0
0	1	0
1	0	0
1	1	1

B) OR Gate

- Output is **1 if any input is 1**.

Symbol



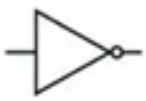
Truth Table

A	B	Y = A+B
0	0	0
0	1	1
1	0	1
1	1	1

C) NOT Gate

- Produces **inverse** of input.

Symbol



Truth Table

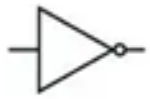
A	Y = A’
0	1
1	0

1.2 Universal Gates

A) NAND Gate

- **NOT + AND**
- Output is **0 only when all inputs are 1**.

Symbol



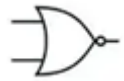
Truth Table

A	B	Y = (A.B)’
0	0	1
0	1	1
1	0	1
1	1	0

B) NOR Gate

- **NOT + OR**
- Output is **1 only when all inputs are 0**

Symbol



Truth Table

A	B	Y = (A+B)’
0	0	1
0	1	0
1	0	0
1	1	0

1.3 XOR & XNOR Gates

A) XOR Gate

- Output is **1 if inputs are different**.

Symbol

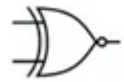
Truth Table

A	B	Y = A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

B) XNOR Gate

- Output is **1 if inputs are same**.

Symbol



Truth Table

A	B	Y = (A ⊕ B)’
0	0	1
0	1	0

1	0	0
1	1	1

2. BOOLEAN ALGEBRA

Boolean algebra is used to analyze and simplify logic circuits.

2.1 Basic Boolean Operations

Operation	Symbol	Meaning
AND	A.B	Both must be 1
OR	A + B	Any one must be 1
NOT	A’	Invert

2.2 Important Boolean Laws

A) Identity Laws

- A + 0 = A
- A · 1 = A

B) Null Laws

- A + 1 = 1
- A · 0 = 0

C) Idempotent Laws

- A + A = A
- A · A = A

D) Inverse Laws

- A + A’ = 1
- A · A’ = 0

E) Commutative Laws

- A + B = B + A
- A · B = B · A

F) Distributive Laws

- A (B + C) = AB + AC
- A + BC = (A + B)(A + C)

G) De Morgan’s Theorems

1. (A·B)’ = A’ + B’
2. (A + B)’ = A’·B’

2.3 Boolean Expression Simplification (Example)

Simplify:
Y = A·B + A·B'

Y = A(B + B')
But B + B' = 1

So:
Y = A · 1 = A

3. DE MORGAN'S LAWS

De Morgan's laws help convert AND ↔ OR operations when a NOT (inversion) is applied to the entire expression.
These are essential for simplifying circuits and implementing designs using only **NAND** or **NOR** gates.

3.1 First De Morgan's Law

(A · B)' = A' + B'

Meaning:

- The **complement of an AND** is equal to the **OR of the complements**.

Intuition:

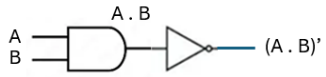
If NOT is applied to an AND operation,
you can distribute the NOT to each input,
but AND becomes OR.

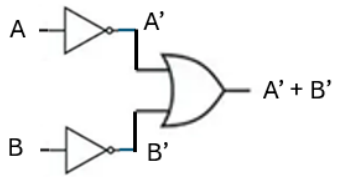
Truth Table Demonstration

A	B	A . B	(A . B)'	A'	B'	A' + B'
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Both (A·B)' and A'+B' produce the same output.

Logic Diagram





Both diagrams give identical results.

3.2 Second De Morgan's Law

(A + B)' = A' · B'

Meaning:

- The **complement of an OR** is equal to the **AND of the complements**.

Intuition:

If NOT is applied to an OR operation,
distribute the NOT,
but OR becomes AND.

Truth Table Demonstration

A	B	A + B	(A + B)'	A'	B'	A' + B'
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Both expressions match exactly.

3.3 Generalized De Morgan's Laws

These apply to expressions with more than two variables:

For AND:

(A·B·C·D)' = A' + B' + C' + D'

For OR:

(A + B + C + D)' = A'·B'·C'·D'

This is widely used in simplification.

3.4 Using De Morgan's Laws in Circuit Design

Converting a NAND implementation:

Since NAND = (AND)'

NAND(A, B) = (A·B)' = A' + B'

So a NAND gate behaves like an OR gate with inverted inputs.

Converting a NOR implementation:

Since NOR = (OR)'

NOR(A, B) = (A + B)' = A'·B'

So a NOR gate behaves like an AND gate with inverted inputs.

These transformations allow designers to build **entire circuits using only NAND or only NOR**, which is cost-effective in IC design.

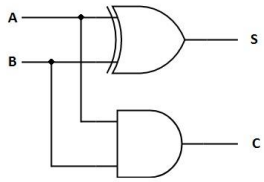
4. COMBINATIONAL LOGIC CIRCUITS

Combinational circuits depend only on **current inputs**, not memory.

4.1 Half Adder

Performs **addition of two bits**.

Logic Diagram



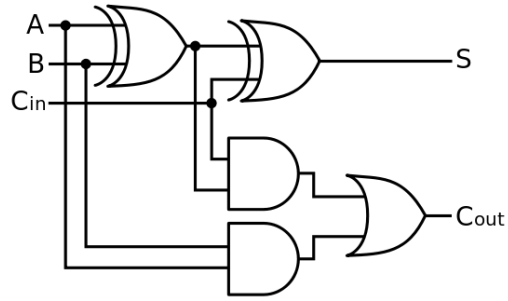
Truth Table

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

4.2 Full Adder

Adds **three inputs**: A, B, Cin.

Logic Diagram



Truth Table

A	B	Cin	Sum	Cout
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

SUM = $A \oplus B \oplus Cin$

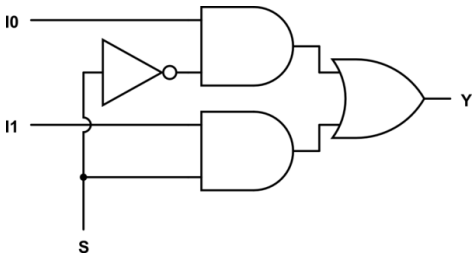
CARRY = Carry1 + (SUM1 · Cin)

4.3 Multiplexer (MUX)

Selects **one input out of many**.

Example: 2-to-1 MUX

Logic Diagram



Symbol

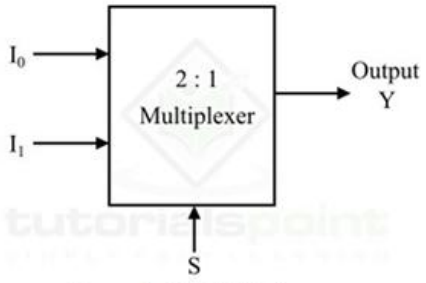


Figure 1 - 2:1 Multiplexer

Truth Table

S	Y
0	I_0
1	I_1

Boolean Expression

Y = S'·A0 + S·A1

4.4 Decoder

Opposite of MUX — converts **n-bit input into 2^n outputs**.

Example: 2-to-4 Decoder

A_1	A_0	Output
0	0	$Y_0 = 1$
0	1	$Y_1 = 1$
1	0	$Y_2 = 1$
1	1	$Y_3 = 1$

4.5 Encoder, Comparator, Parity Generator

(Brief definitions)

- **Encoder:** Converts many inputs → fewer outputs.
- **Comparator:** Compares two binary numbers.
- **Parity Generator:** Produces parity bit for error checking.