

UNIT-3

Digital Electronics

Unit Outcome

At the end of this unit, student will be able to

- Understand the working of logic gates
- Solve the Boolean expressions based on laws of Boolean algebra
- Apply the logic gates in arithmetic building block, data processing unit and combinational & sequential circuits.

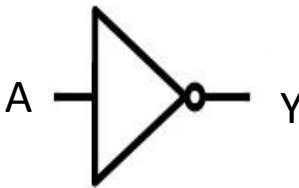
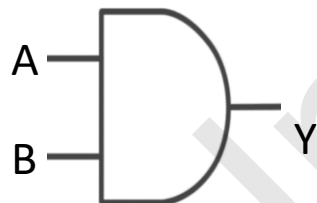
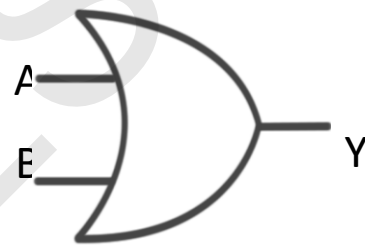
Boolean Algebra

- ❖ A branch of algebra in which the values of the variables are the truth values **true** and **false**, usually denoted **1** and **0** respectively.
- ❖ In digital logic circuits, logic symbol **0** represents **Open** circuit or **OFF** state of switch and logic symbol **1** represents **Closed** circuit or **ON** state of switch

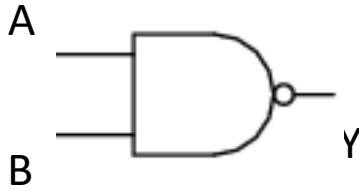
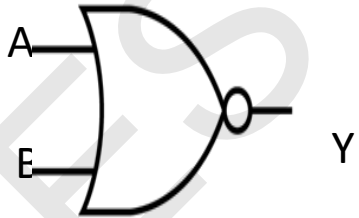
LOGIC GATES

- ❖ A physical device which implements a **Boolean function** i.e a **Boolean logic** : a logical operation is performed using one or more **binary** inputs and produces a single binary output
- ❖ They are implemented using diodes but mostly transistors acting electronic switches
- ❖ Logic circuits : MUX, DeMUX, Registers, ALU, Memory, Microprocessor
- ❖ Types :
 - **Basic** Gates : NOT, AND, OR
 - **Universal** Gates: NAND, NOR

Logic Gates: Basic Gates

Name	Symbol	Boolean expression	Truth table		
NOT gate		$Y = \overline{A}$ i.e $Y = \text{NOT } A$	A	Y	
			1	0	
			0	1	
AND gate		$Y = A \cdot B$ i.e $Y = A \text{ AND } B$	A	B	Y = A AND B
			0	0	0
			0	1	0
			1	0	0
			1	1	1
OR gate		$Y = A + B$ i.e $Y = A \text{ OR } B$	A	B	Y
			0	0	0
			0	1	1
			1	0	1
			1	1	1

Universal Gates

Name	Symbol	Boolean expression	Truth table																	
NAND gate		$Y = \overline{A \cdot B}$ $Y = \text{NOT (A AND B)}$	<table><tr><th>A</th><th>B</th><th>$Y = \overline{A \cdot B}$</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	$Y = \overline{A \cdot B}$	0	0	1	0	1	1	1	0	1	1	1	0		
A	B	$Y = \overline{A \cdot B}$																		
0	0	1																		
0	1	1																		
1	0	1																		
1	1	0																		
NOR gate		$Y = \overline{A + B}$ $Y = \text{NOT (A OR B)}$	<table><tr><th>A</th><th>B</th><th>$Y = \overline{A + B}$</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	$Y = \overline{A + B}$	0	0	1	0	1	0	1	0	0	1	1	0		
A	B	$Y = \overline{A + B}$																		
0	0	1																		
0	1	0																		
1	0	0																		
1	1	0																		

Basic Theorem and Properties of Boolean Algebra

Boolean Law	Expressions
Idempotent Law	$A * A = A$ $A + A = A$
Associative Law	$(A * B) * C = A * (B * C)$ $(A + B) + C = A + (B + C)$
Commutative Law	$A * B = B * A$ $A + B = B + A$
Distributive Law	$A * (B + C) = A * B + A * C$ $A + (B * C) = (A + B) * (A + C)$
Identity Law	$A * 0 = 0 \quad A * 1 = A$ $A + 1 = 1 \quad \overline{A} + 0 = A$
Complement Law	$A * \overline{A} = 0$ $\overline{\overline{A}} + A = 1$
Involution Law	$A = \overline{\overline{A}}$

Boolean Laws and Theorems

Boolean Law	Expression
DeMorgan's Law	$(A * B) = \overline{A + B}$ $\overline{(A + B)} = \overline{A} * \overline{B}$
Redundancy Laws	<ul style="list-style-type: none"> • Absorption $A + (A * B) = A$ $A * (A + B) = A$ • $A + (A * \overline{B}) = A + B$ $A * (A + B) = A * B$
The Principle of duality is	<ul style="list-style-type: none"> • Interchanging the + (OR) and * (AND) operations of the expression. • Interchanging the 0 and 1 elements of the expression. • Not changing the form of the variables.

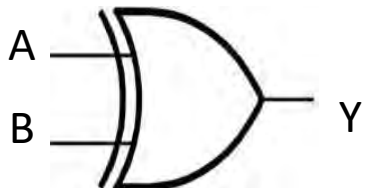
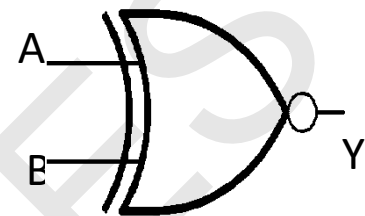
Simplification of Boolean expressions

- Using the Boolean laws and theorems, any Boolean expressions may be simplified to the least represented form.
- Basic gates realization
- NAND gate realization
- NOR gate realization

Canonical and Standard Form

- ❖ A boolean expression consisting entirely either of **minterm** or **maxterm** is called **canonical** expression.
- ❖ Example : If we have two variables X and Y then,
 - ❖ Following is a canonical expression consisting of minterm $XY + X'Y'$
 - ❖ Following is a canonical expression consisting of maxterm $(X+Y) \cdot (X' + Y')$
- ❖ There are two forms of canonical expression.
 - ❖ Sum of Products (**SOP**)
 - ❖ Product of Sums (**POS**)

Other logic operation: Exclusive Gates

Name	Symbol	Boolean expression	Truth table															
XOR gate		$Y = A \oplus B$ $Y = AB' + A'B$	<table><tr><th>A</th><th>B</th><th>$Y = A \oplus B$</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	$Y = A \oplus B$	0	0	0	0	1	1	1	0	1	1	1	0
A	B	$Y = A \oplus B$																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
XNOR gate		$Y = AB + A'B'$	<table><tr><th>A</th><th>B</th><th>$Y = AB + A'B'$</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	$Y = AB + A'B'$	0	0	1	0	1	0	1	0	0	1	1	1
A	B	$Y = AB + A'B'$																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

Combinational Circuits: Arithmetic Building Blocks

Arithmetic Building blocks may incorporate blocks like adders, subtractors, multipliers, logic unit, shifters.

An adder is a digital logic circuit in electronics that implements addition of numbers. **Adders** are classified into two types :

1. Half Adder
2. Full Adder

Logic Circuits

- **Combinational Logic:**

- Output depends only on current input
- Able to perform useful operations (add/subtract/multiply/...)
- Has no memory

- **Sequential Logic:**

- Output depends not only on current input but also on past input values, e.g., design a counter
- Need some type of memory to remember the past input values

Half Adder

The Half adder circuit has two inputs: A and B. It adds **two input** and generate a **carry** and **sum**.

Addition of two single bits :

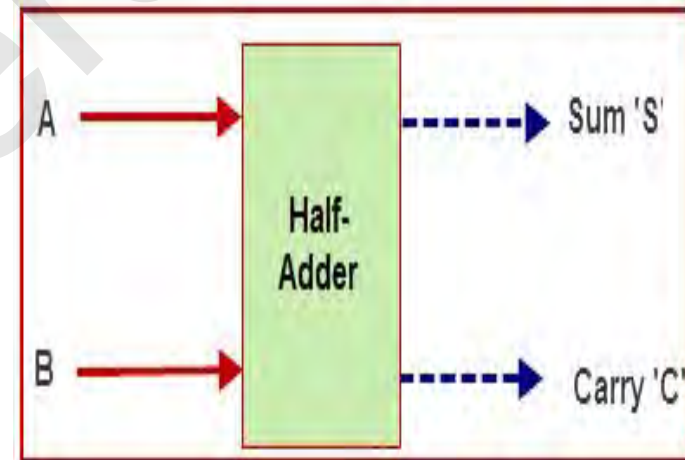
$$0+0 = 00$$

$$0+1 = 01$$

$$1+0 = 01$$

$$1+1 = 10$$

Note : All digits are in binary form i.e base 2

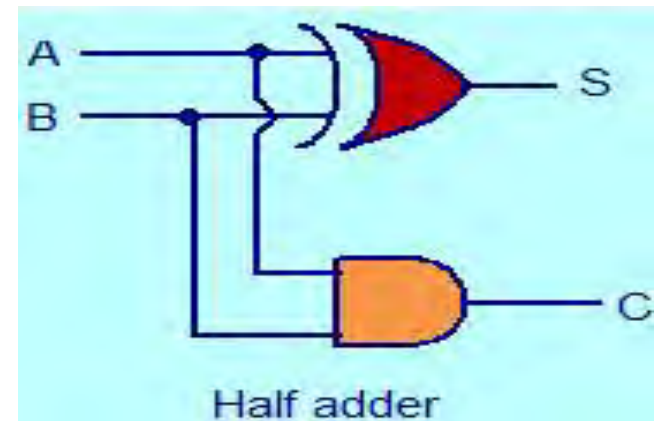


Half Adder contd.

1-bit adder can be easily implemented with the help of the XOR Gate for the output 'SUM' and an AND Gate for the 'Carry'

- For the **SUM** bit:
 $SUM = A \text{ XOR } B = A \oplus B$
- For the **CARRY** bit:
 $CARRY = A \text{ AND } B = A.B$

INPUTS		OUTPUTS	
A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Half adder using NAND gates

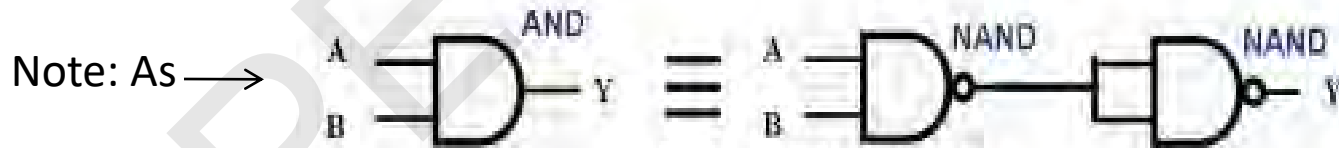
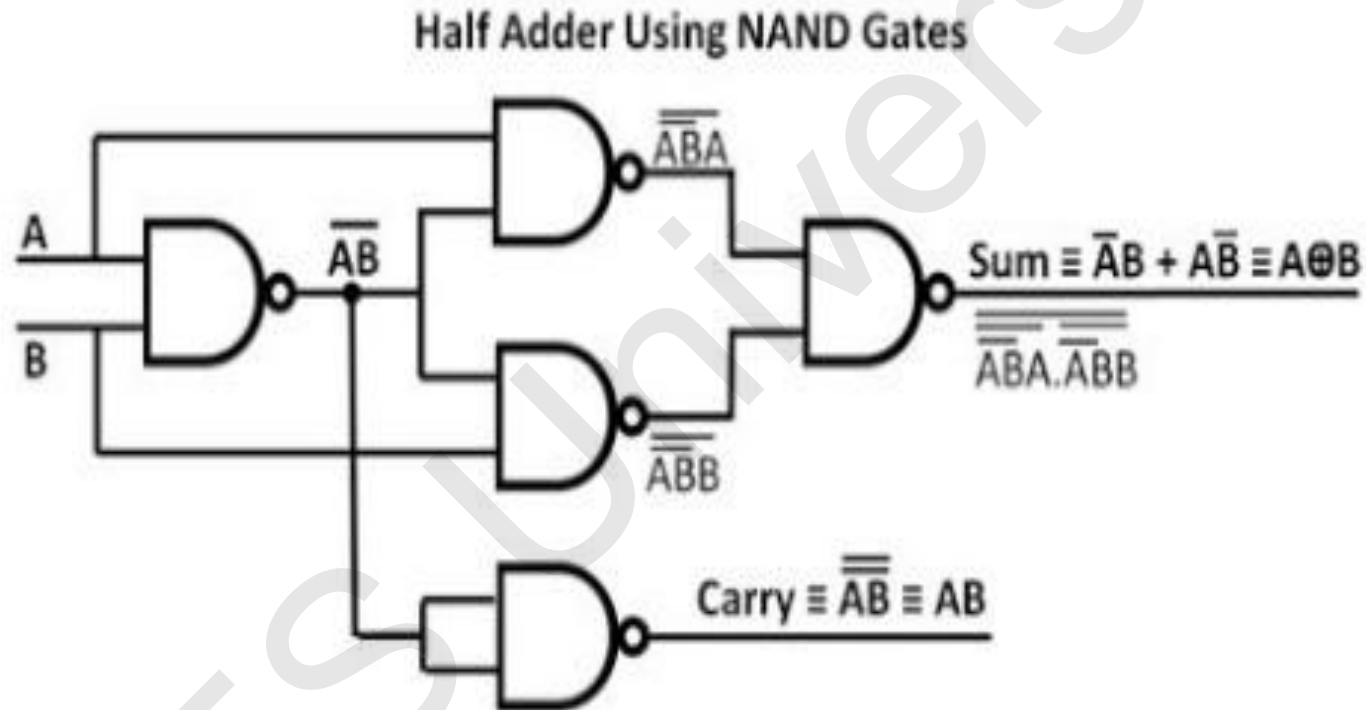
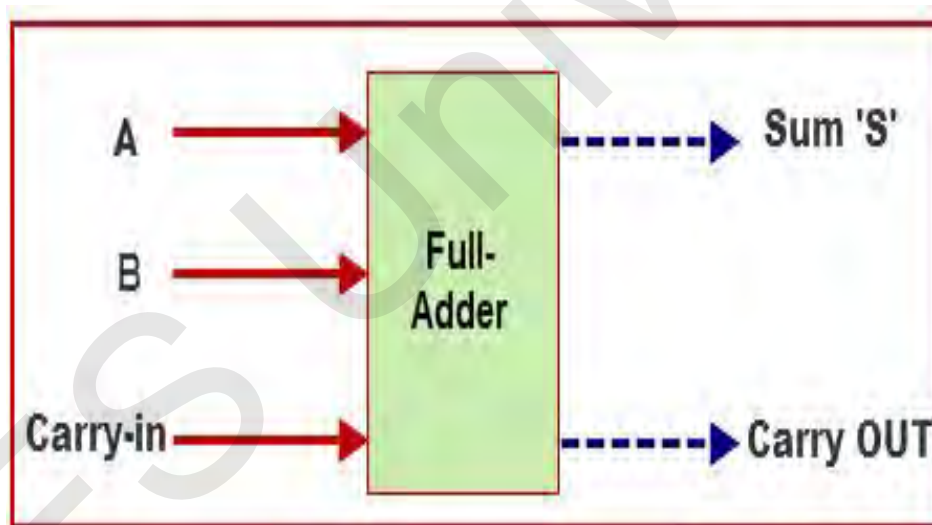


Fig 1: AND Using NANDs

Full Adder

The full adder circuit has three inputs: A, B and Cin. It adds the **three input** numbers and generate a **carry** and **sum**.

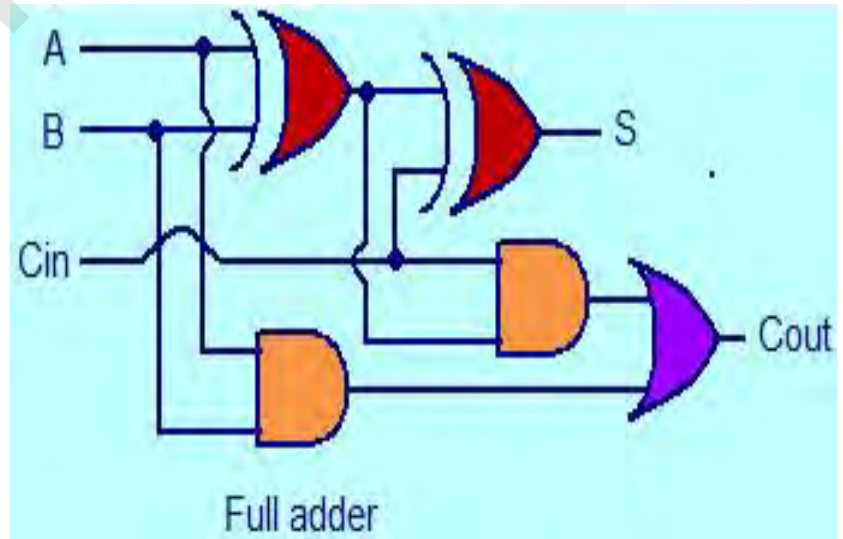


Full Adder contd.

The output S is an XOR between A, B and Cin.

Take C-OUT will only be true if any of the two inputs out of the three are HIGH.

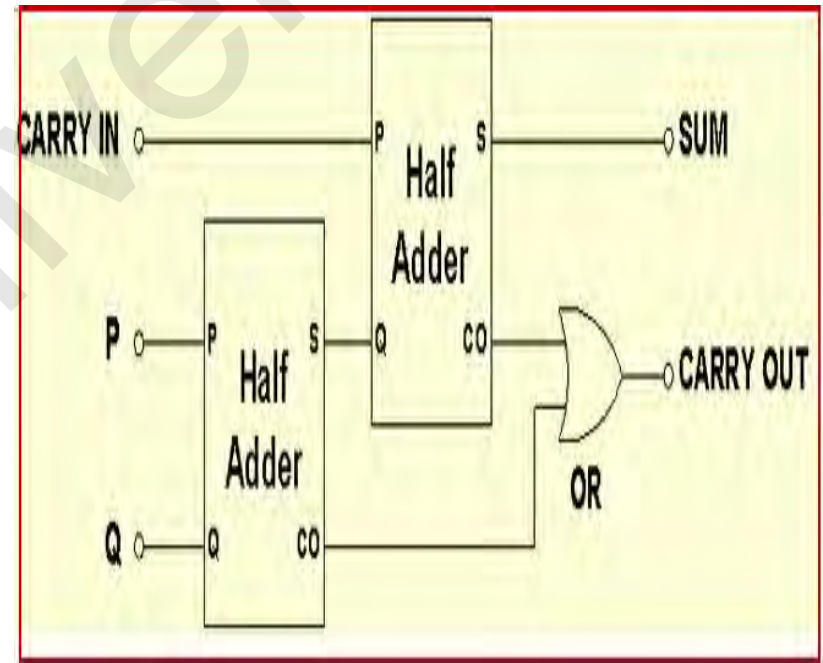
INPUTS			OUTPUT	
A	B	C-IN	C-OUT	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Full Adder Using Half Adder

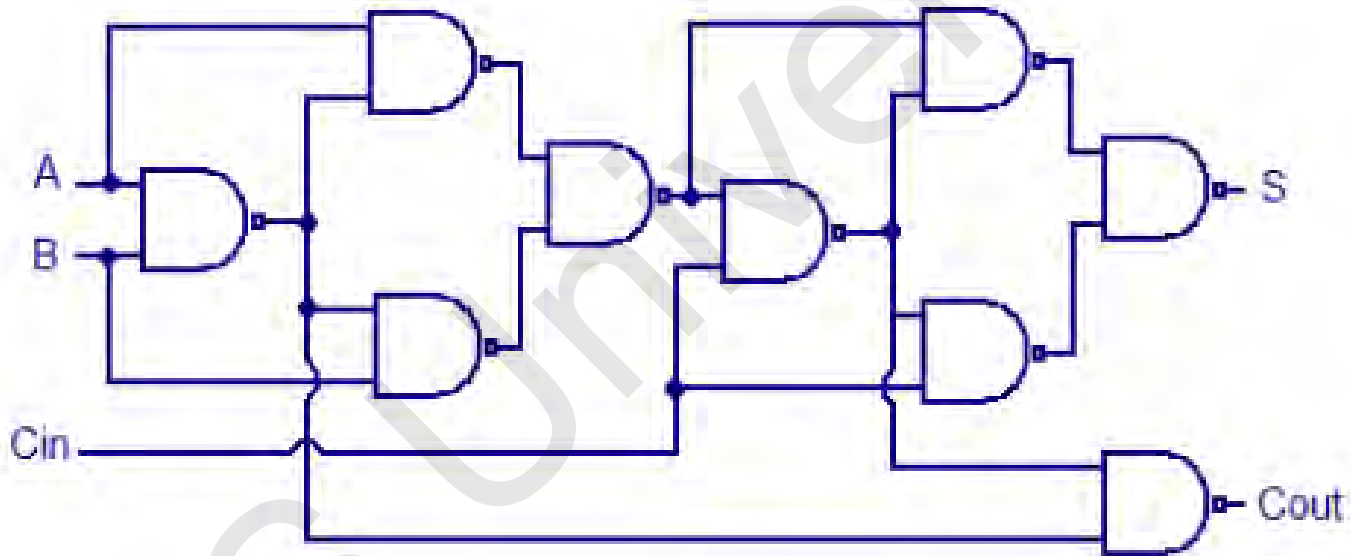
We can implement a full adder circuit with the help of two half adder circuits.

- First, half adder will be used to add P and Q to produce a partial Sum.
- A second half adder logic can be used to add C-IN to the Sum produced by the first half adder to get the final S output
- If any of the half adder logic produces a carry, there will be an output carry.
- A COUT will be an OR function of the half-adder Carry outputs.



As the full adder circuit above is basically two half adders connected together, the truth table for the full adder includes an additional column to take into account the *Carry-in*, C_{IN} input as well as the summed output, S and the Carry-out, C_{OUT} bit.

Full Adder Using NAND gates



Data Processing Circuits

To process a large amount of data or information, different forms of circuits are used in digital electronics like :

- **Multiplexer**
- **Demultiplexer**

What is a Multiplexer (MUX)?

- A MUX is a digital switch that has multiple inputs (sources) and a single output (destination).

- **MANY to ONE CIRCUIT**

- The select lines determine which input is connected to the output.

- Inputs :

Select Lines : N

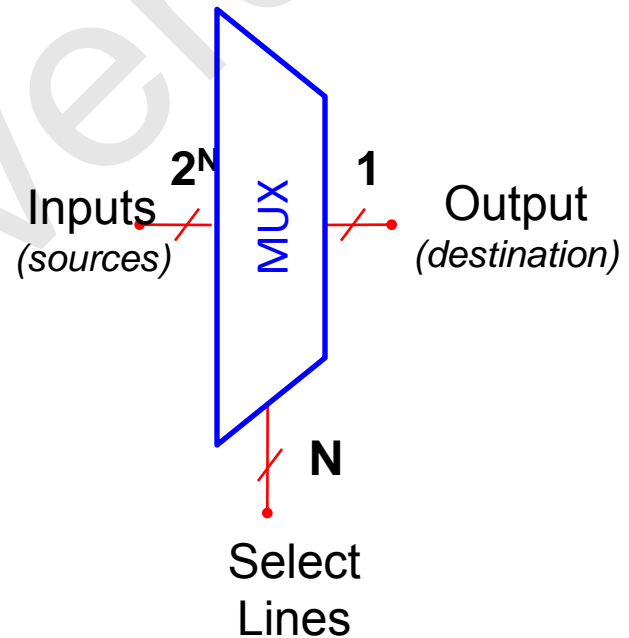
Output : 1

- MUX Types

- 2-to-1 (1 select line)

- 4-to-1 (2 select lines)

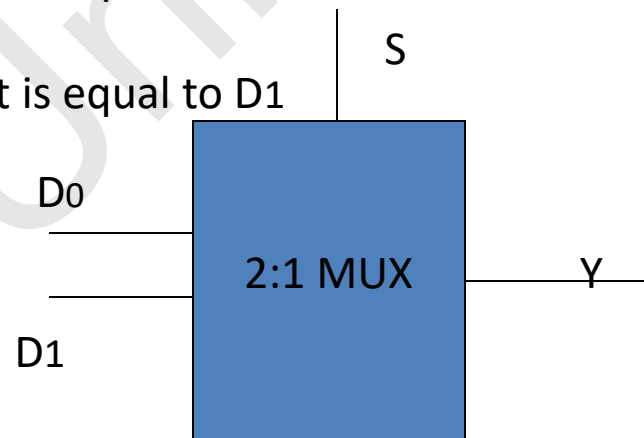
- 8-to-1 (3 select lines)



Multiplexer
Block Diagram

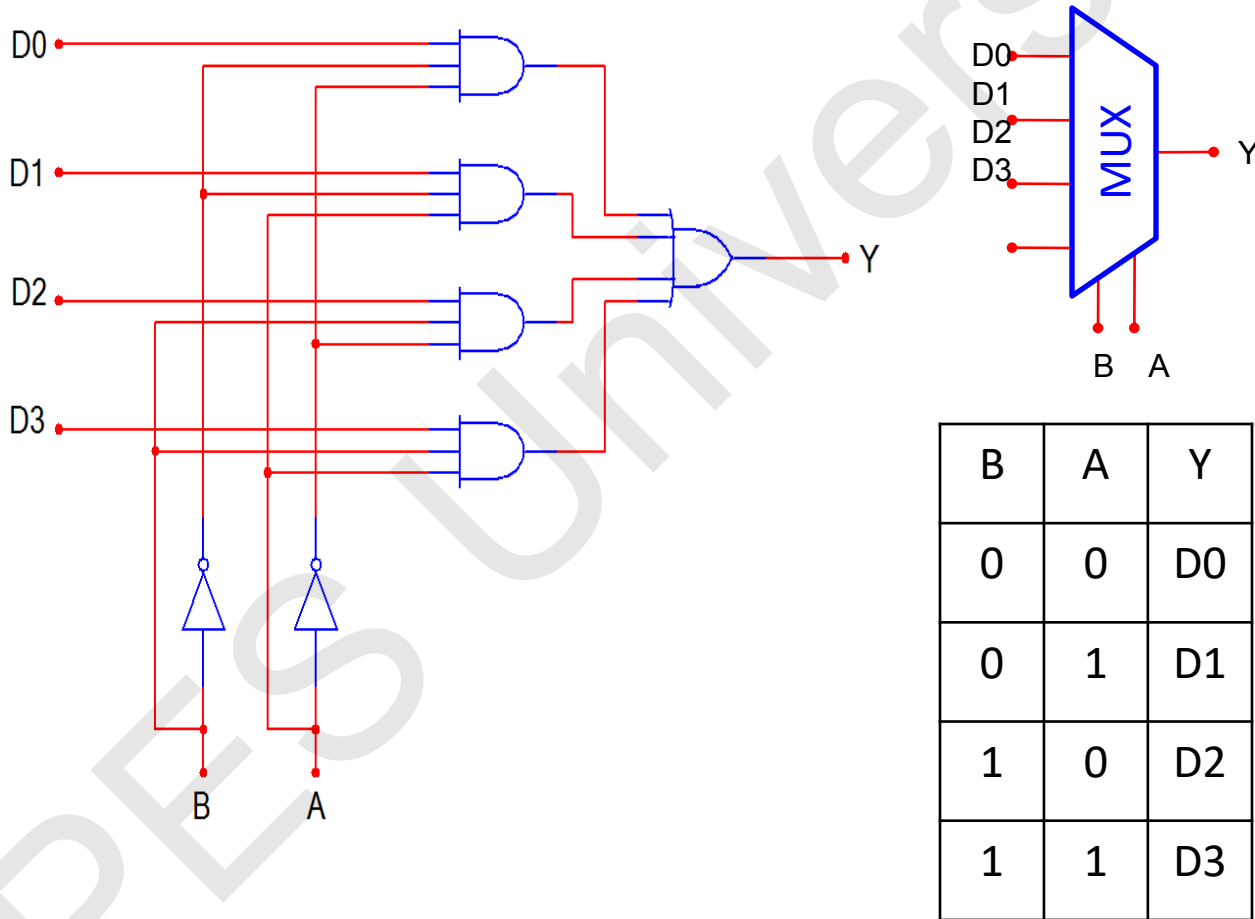
2-1 Multiplexer

- Has two inputs: D0 and D1
- out put Y
- Also has another input line S
- If $s=0$, then the output is equal to D0
- If $s=1$, then the output is equal to D1

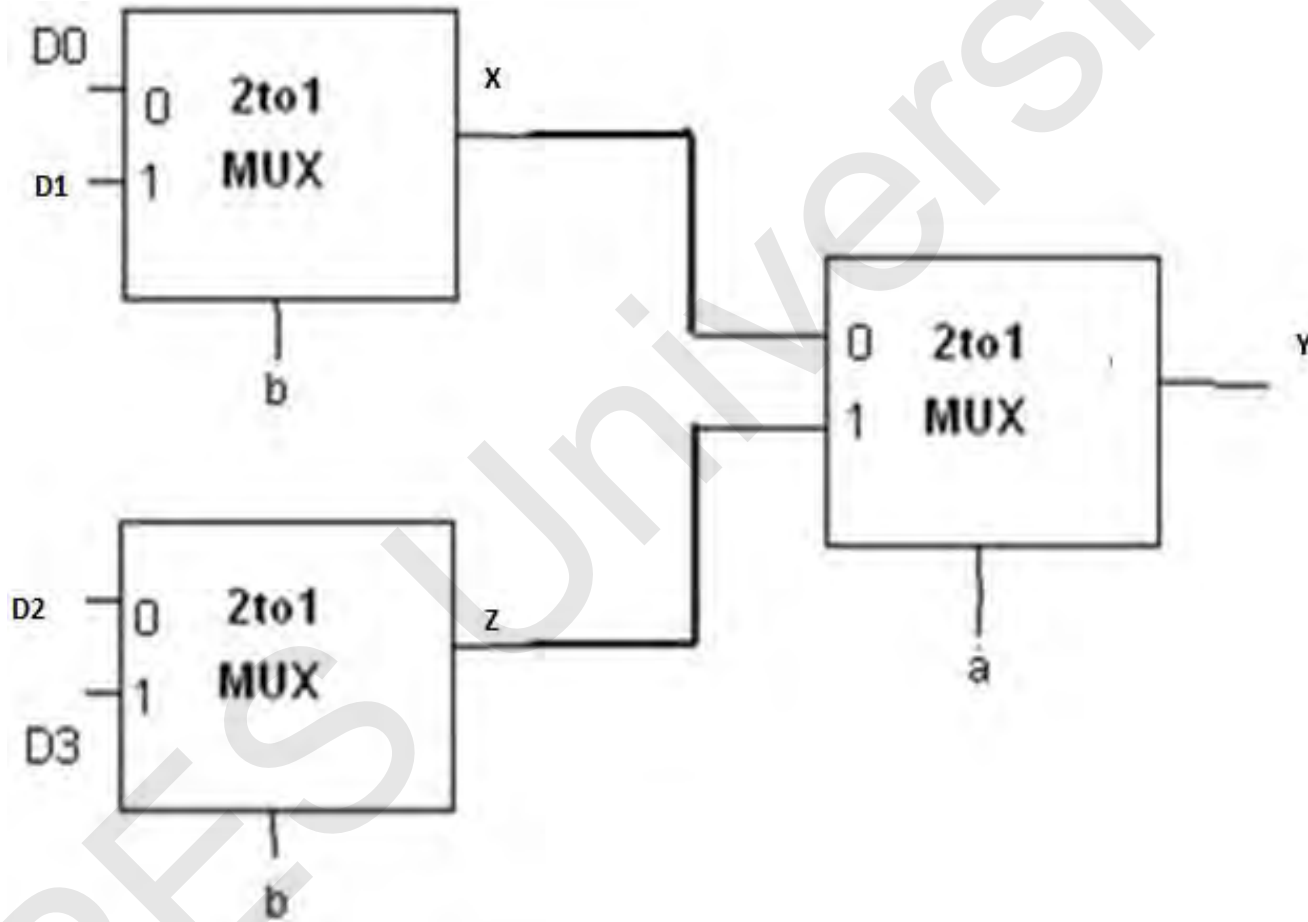


S	(O/P)Y
0	D0
1	D1

4-to-1 Multiplexer (MUX)



4:1 MUX from minimum 2:1



What is a Demultiplexer (DEMUX)?

- A DEMUX is a digital switch with a single input (source) and a multiple outputs (destinations).

ONE TO MANY Circuit

- The select lines determine which output the input is connected to.

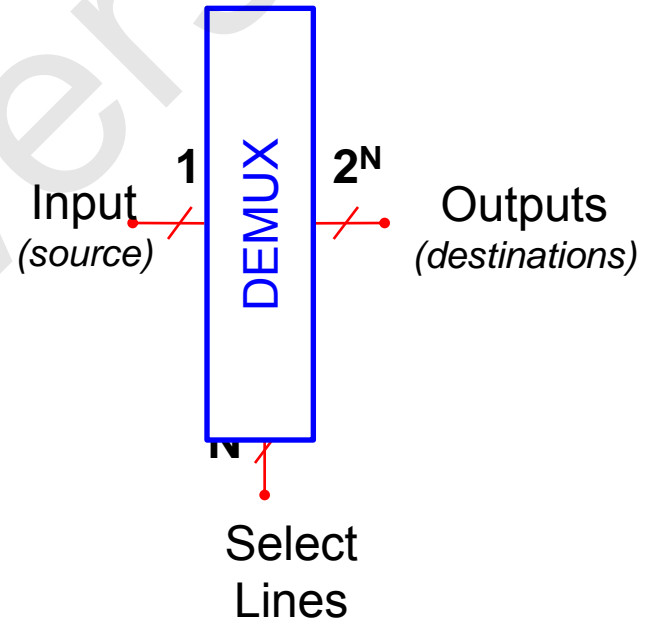
- Input : 1

Select line : N

Output : 2^N

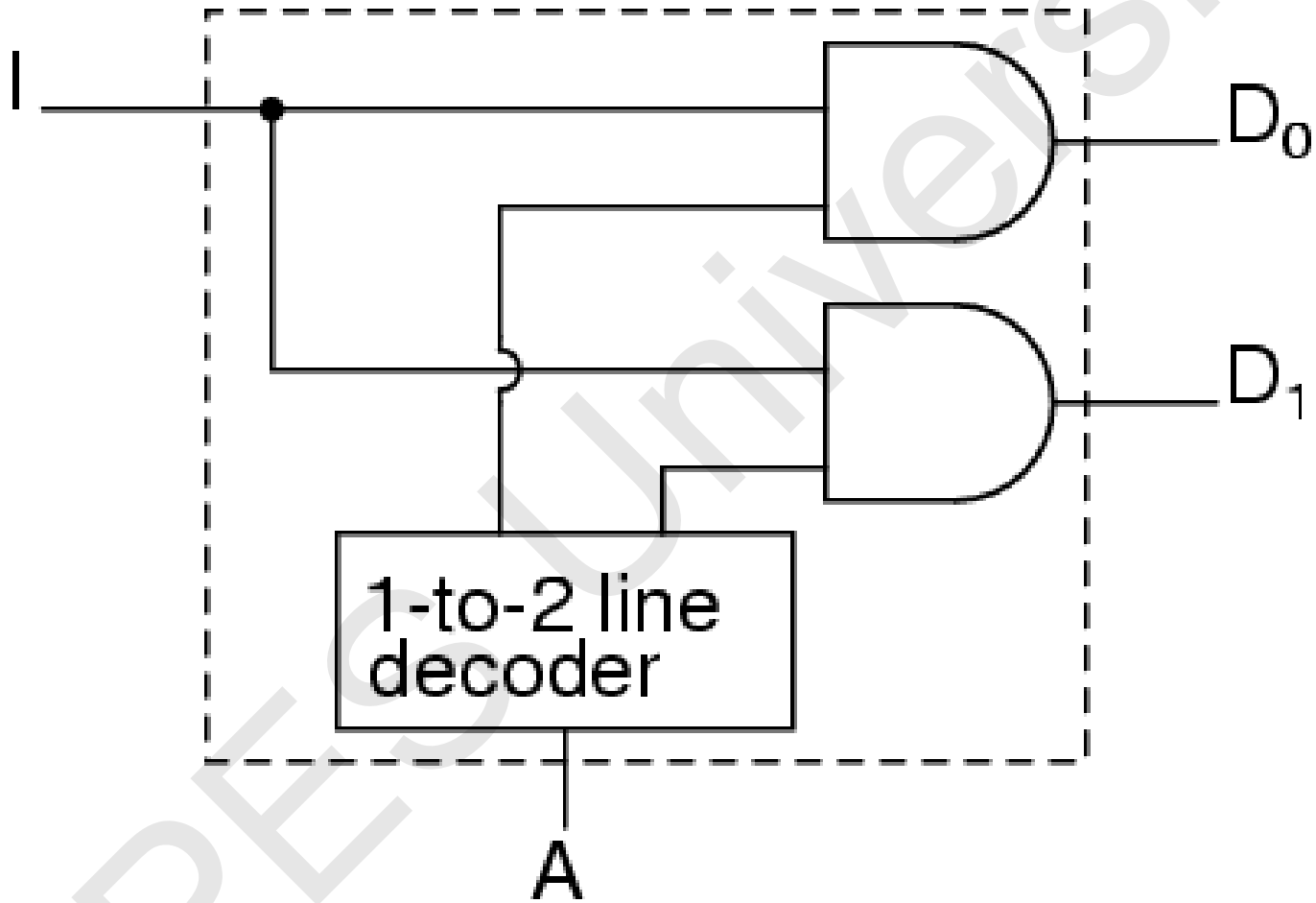
- DEMUX Types

- 1-to-2 (1 select line)
- 1-to-4 (2 select lines)
- 1-to-8 (3 select lines)

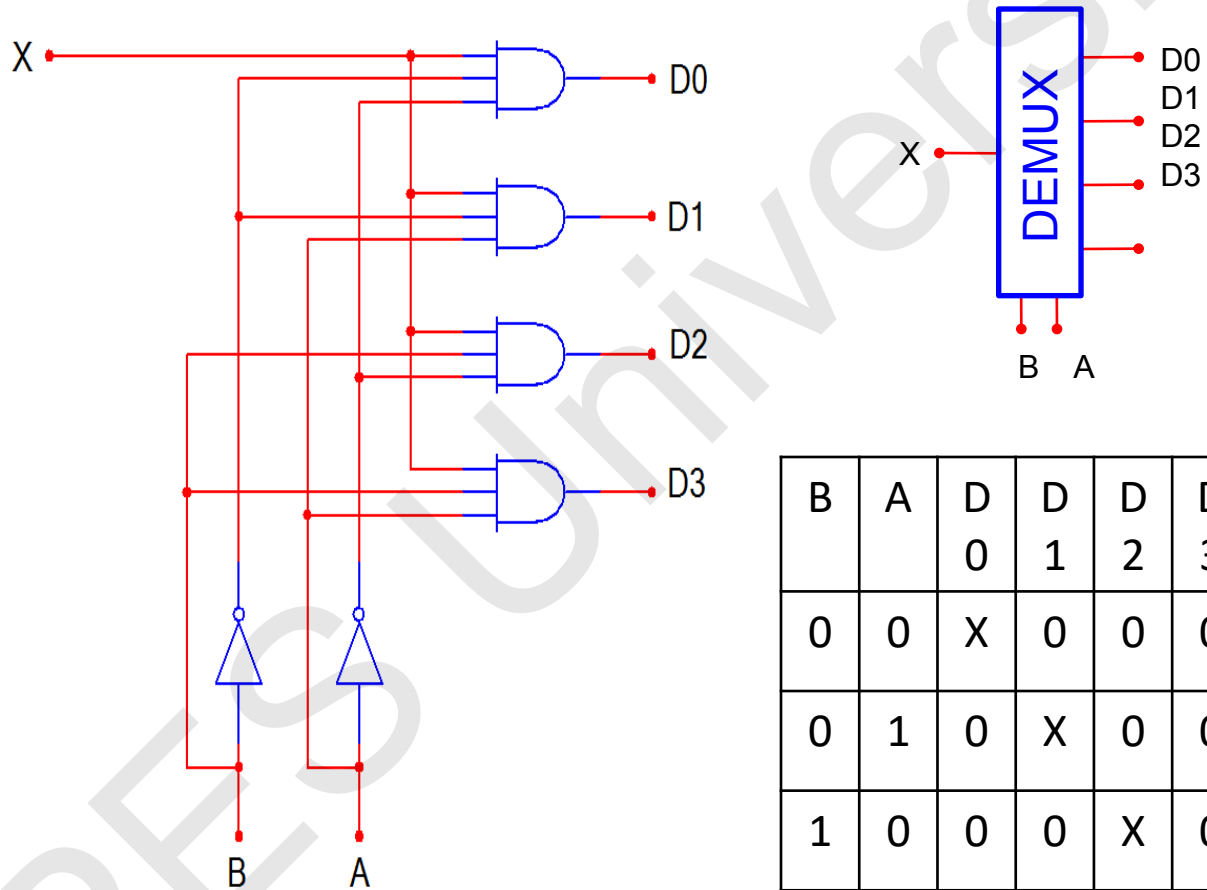


Demultiplexer
Block Diagram

1-to-2 De-Multiplexer (DEMUX)



1-to-4 De-Multiplexer (DEMUX)



B	A	D 0	D 1	D 2	D 3
0	0	X	0	0	0
0	1	0	X	0	0
1	0	0	0	X	0
1	1	0	0	0	X

Sequential Circuits

Sequential Logic circuits remember past inputs and past circuit state. Outputs from the system are “**fed back**” as new inputs with delay.

There are two types of sequential circuits:

➤ **Asynchronous** sequential circuit

Circuit output can change at **any** time (clockless). Ex: Latch

➤ **Synchronous** sequential circuit

Circuit output changes only at some discrete instants of time.

This type of circuits achieves synchronization by using a timing signal called the *clock*. Ex : Flip Flop

Latches and **Flip flops** are sequential circuit elements which, with the help of Boolean logic can create memory i,e a storage element(s) capable of holding binary information (1 bit of information per latch or flipflop only)

Sequential circuits : Latches and Flip

Flops

❖ **Latches** are “transparent” (= **any change on the inputs is seen at the outputs immediately**). This causes synchronization problems.

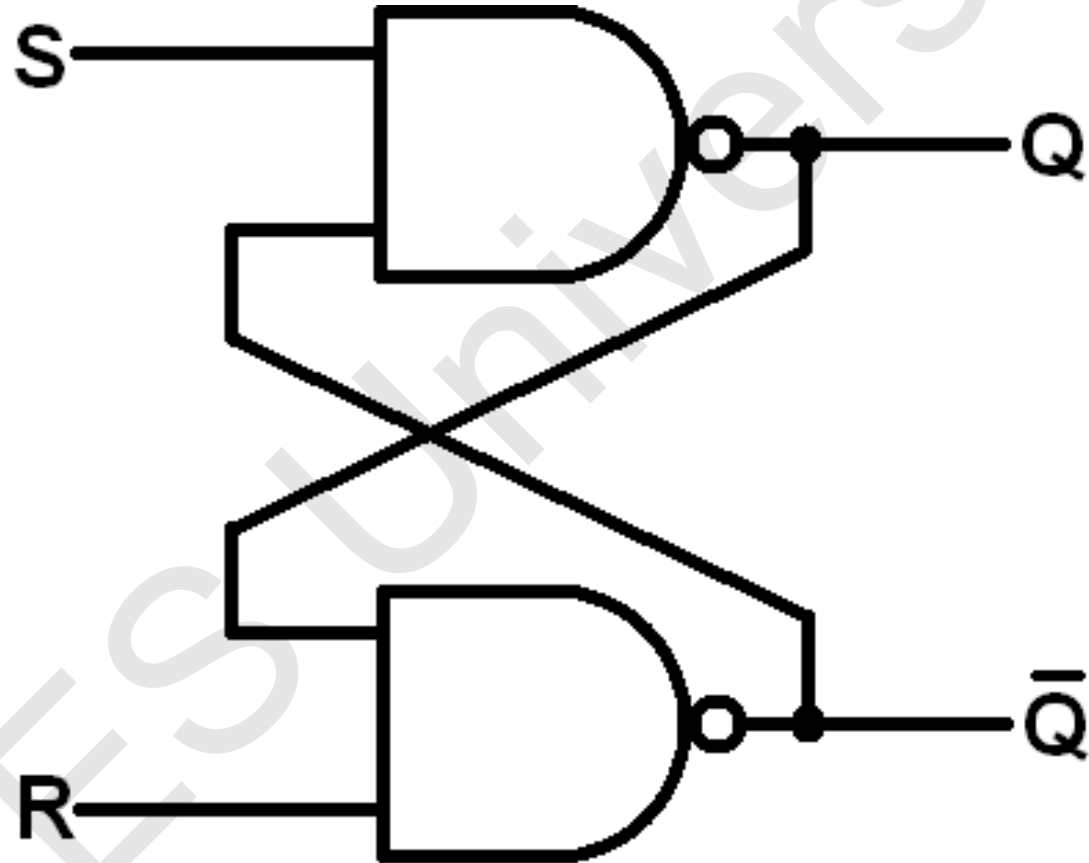
Solution: use latches to create **flip-flops** that can **respond (update) only on specific times** (instead of any time).

❖ There are four types of flip flops that are used in electronic circuits which are :

- 1.The basic Flip Flop or S-R Flip Flop
- 2.Delay Flip Flop [D Flip Flop]
- 3.J-K Flip Flop
- 4.T Flip Flop

The most commonly used application of flip flops is in the implementation of a **feedback** circuit. As a memory relies on the feedback concept, flip flops can be used to design it.

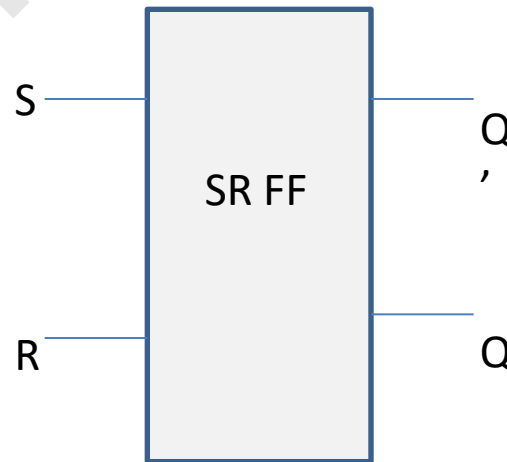
S R latch (NAND latch)



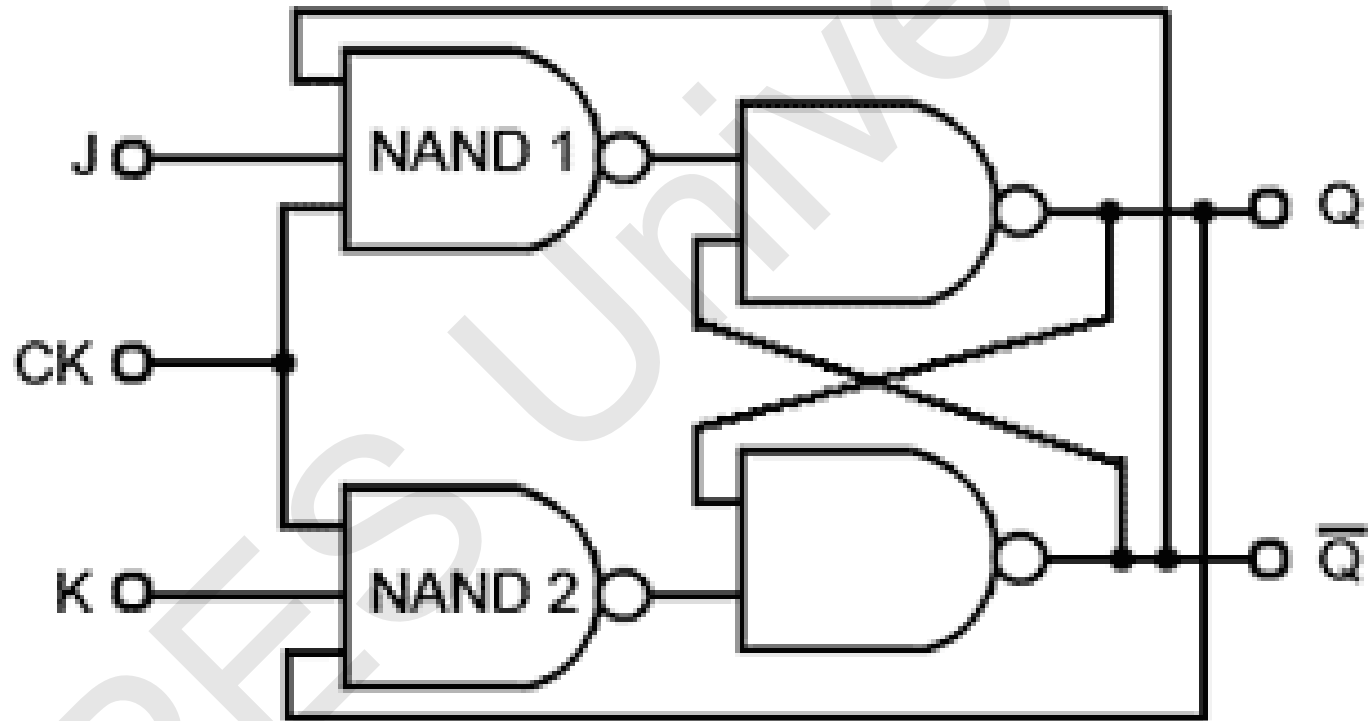
S-R Flip flop

S-R Flip Flop The **SET-RESET** flip flop is designed with the help of two NOR gates and also two NAND gate latch. (SR latch or S'R' latch)

The design of such a flip flop includes two inputs, called the SET [S] and RESET [R]. There are also two outputs, Q and Q'.

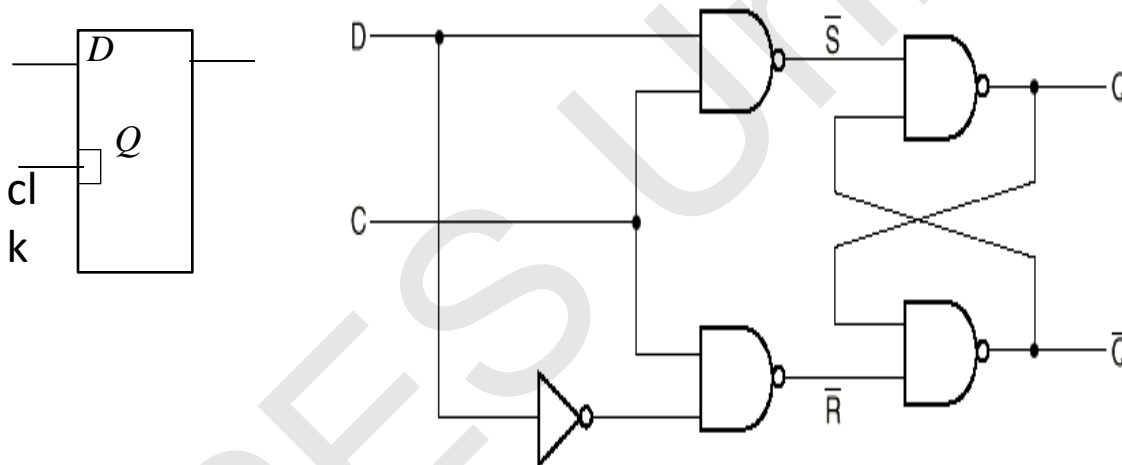


JK FF



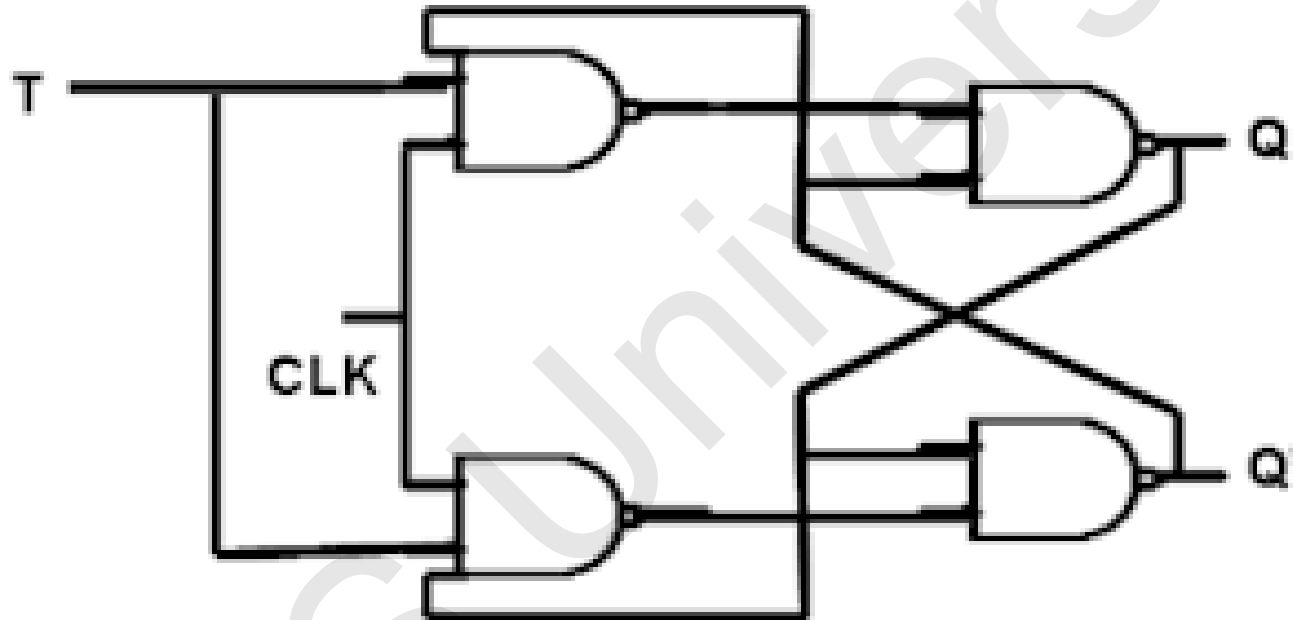
D FF

One way to eliminate the undesirable indeterminate state in the RS flip flop is to ensure that inputs S and R are **never 1 simultaneously** i.e never have **forbidden state**. This is done in the *D latch*



C	D	Next state of Q
0	X	No change
1	0	Q = 0; Reset state
1	1	Q = 1; Set state

T FF



Registers

- A storage element holding 1-bit of data
- Applications :
 - Shifters
 - Counters
- Types of registers :
 - Serial In Serial Out (SISO)
 - Serial In Parallel Out (SIPO)
 - Parallel In Serial Out (PISO)
 - Parallel In Parallel Out (PIPO)

Basic shift register

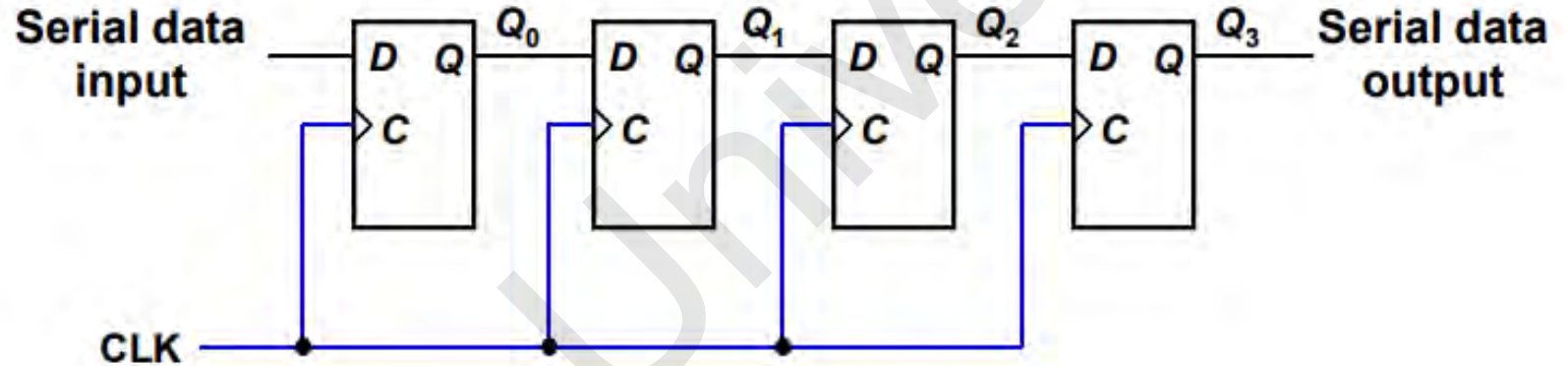
A basic shift register is simply a chain of ***D* flip-flops** with a common **clock**.

Each flip-flop transfers its *D* input to its *Q* output at a clock transition.

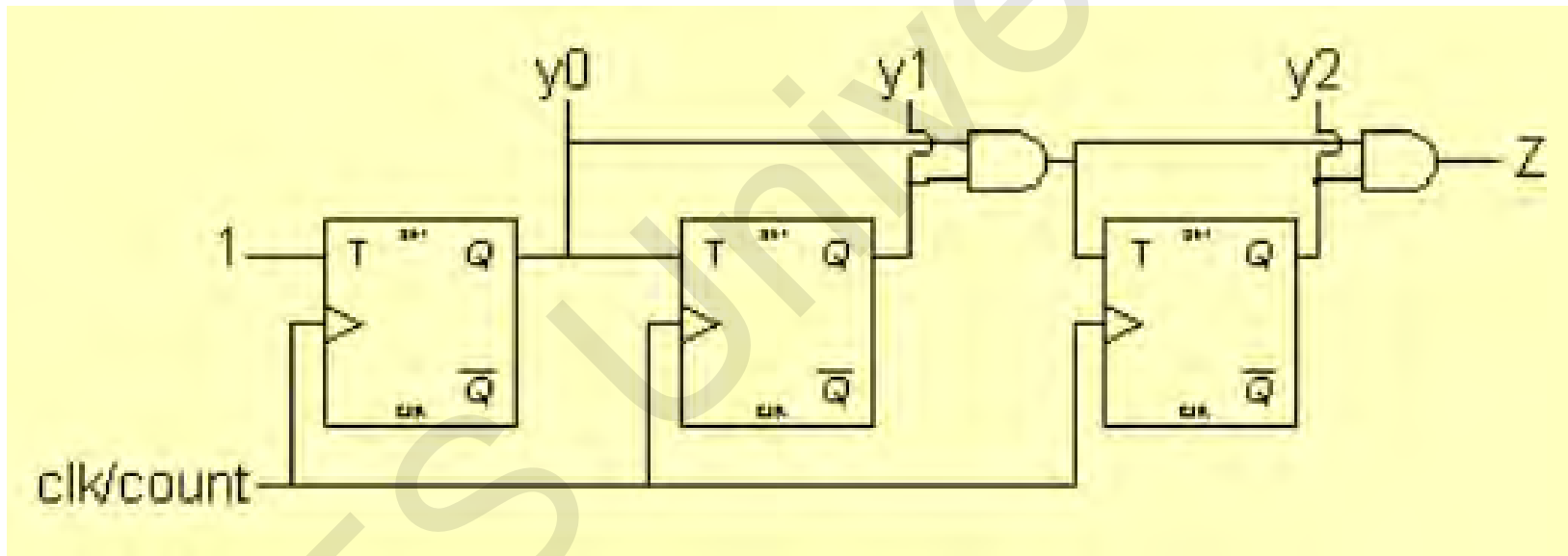
The effect is to transfer data along the register, one flip-flop per clock cycle.

This type of register is called a serial input-serial output (SISO).

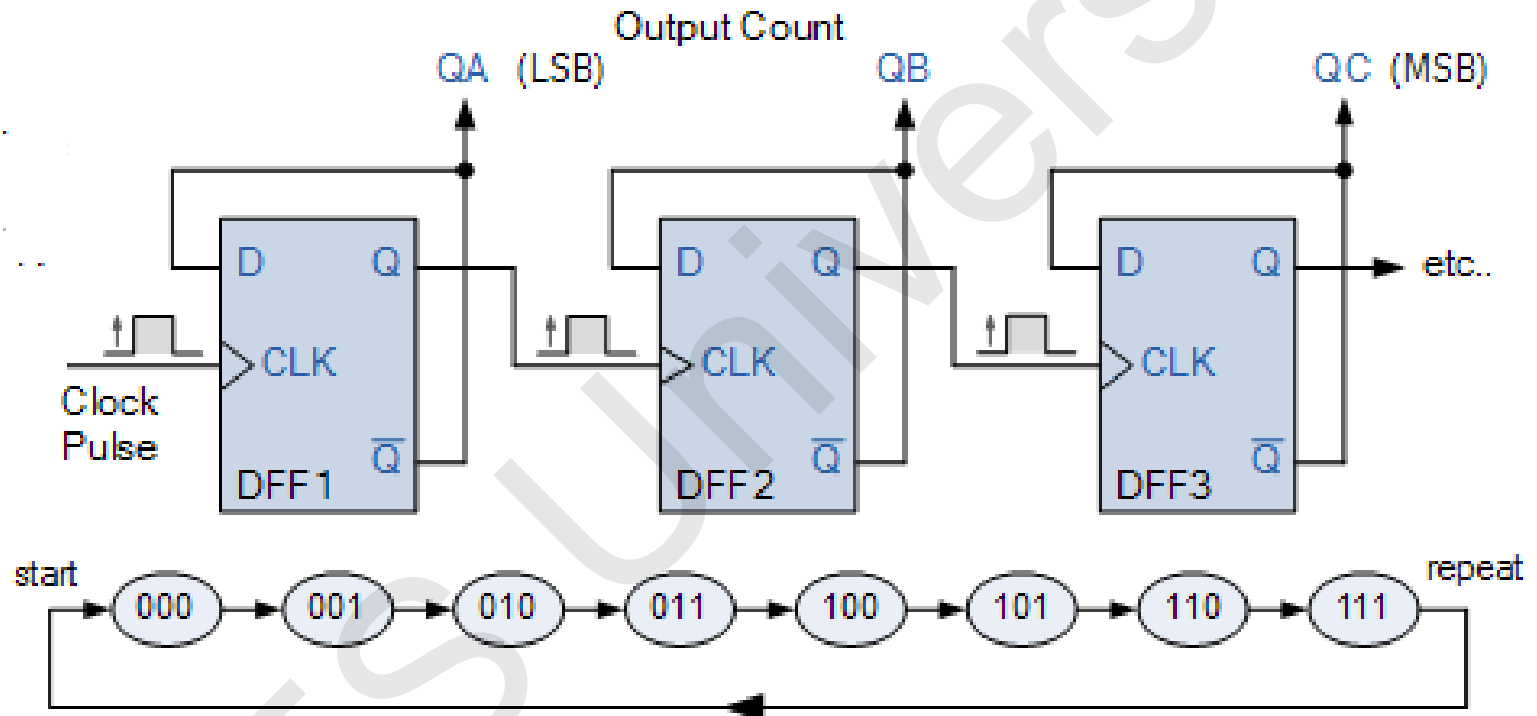
serial input-serial output (SISO).



3-bit Synchronous Counter



3-bit Asynchronous Counter



State transition diagram for a MOD-8 binary counter

Practice problem

1. Simplify the following Boolean expressions and realize using basic gates

a) $Y = AB'C' + A'B'C' + A'B' + AC'$

b) $Y = (A + B' + C)(A' + B + C')(A + B')$

2. Realize 3 input XOR gate using

a) 2 input NAND gates

b) 2 input NOR gates