Combinational Circuits

What is combinational circuit?

A combinal circuit is an interconnected set of gates whose output at any time * is a function only of the input at that time. As with a single gate, the appearance of the input is followed almost immediately by the appearance of the output, with only gate delays.

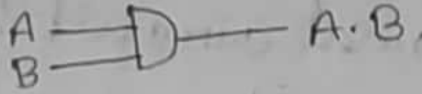
In general terms, a combinational circuit consists of n binary inputs and m binary outputs.

Binary variables: Variables used in Boolean Algebra. Identifiers can be letters of Alphabet (eg. A, B, x, y etc). A binary variable can have one of two binary values e.g. True/false, On/off, Yes/No, 0/1.

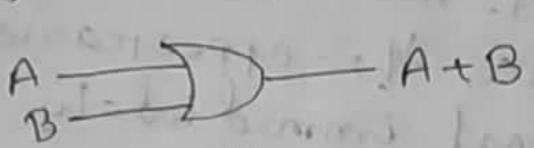
Logical Operations: Operation of binary variables. There are three basic logical operations.

1. ~~And~~ AND: Symbollically represented by a Dot. The operation yeilds true if and

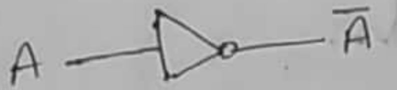
only if both of its operands is true.



2. OR: symbolically represented by plus sign. The operation yeilds true if either or both of its operands are true.



3. NOT: symbolically represented by over-bare or apostrophe. The operation inverts the value of its operand.



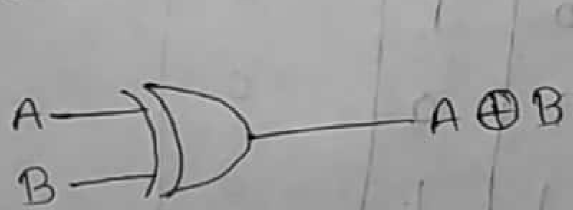
Truth table: It is a tabular listing of the values of a function for all possible combinations of values on its arguments.

Timing Diagram: A digital timing diagram is a representation of a set of signals in the time domain. A timing diagram may contain many rows, usually one of them being the clock.

Logic Diagram and Expressions: Logic diagram is graphical representation of a logic circuit that shows the wiring and connections of each individual logic gate, represented by a specific graphical symbol, that implements logic circuit.

Logical Expressions (also known as Boolean expressions) are the result of applying logical operators to relational or arithmatic operations.

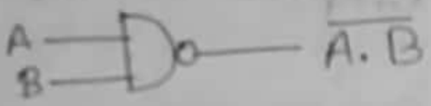
XOR Gate: It implements an exclusive or. The operation yeild true if one and only one of the inputs to the gate is true.



A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

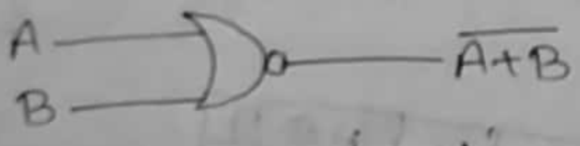
Universal Gate: A universal gate can implement any booleam function without need to use any othere gate type

NAND gate: ~~The~~ It yields true if either ore both of its inputs are false.



A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

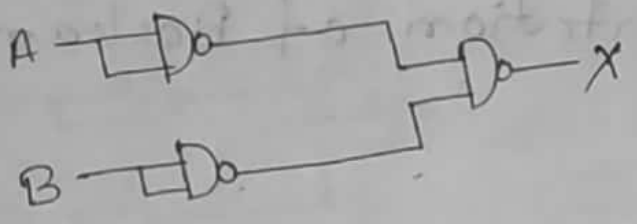
NOR Gate: It yields true if both of its ~~op~~ inputs are false.



A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Basic gates using NAND:

1. OR Gate:

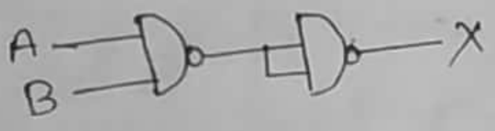


$X = \overline{\overline{A.A}\,.\,\overline{B.B}}$

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

$= A+B$

2. AND Gate:

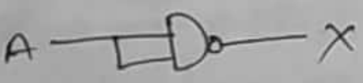


$X = \overline{\overline{A.B}\,.\,\overline{A.B}}$

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

$= A.B$

3. NOT Gate:

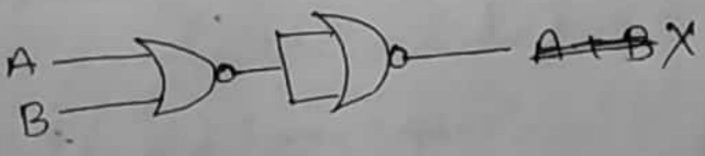


$X = \overline{A.A}$

$= \overline{A}$

Basic Gates using NOR:

1. OR Gate:

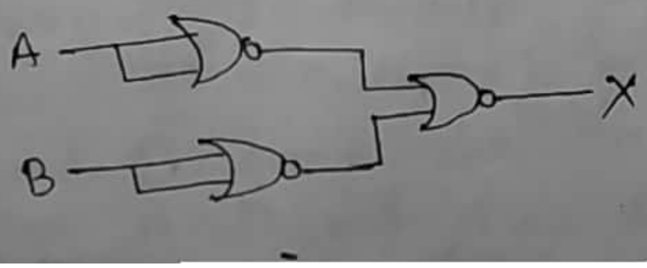


$X = \overline{\overline{A+B} + \overline{A+B}}$

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

$= A+B$

2. AND Gate:

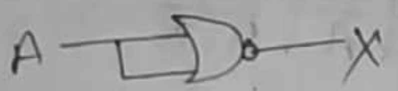


$X = \overline{\overline{A+A} + \overline{B+B}}$

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

$= A.B$

3. NOT Gate:



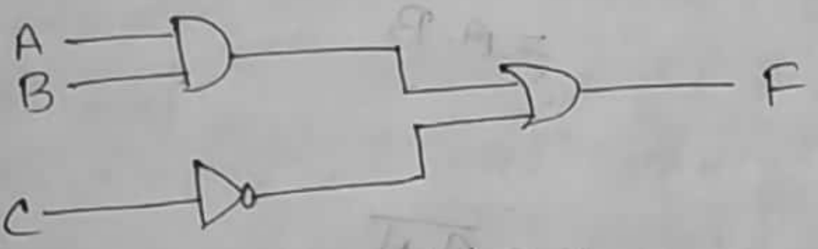
$X = \overline{A+A}$

$= \overline{A}$

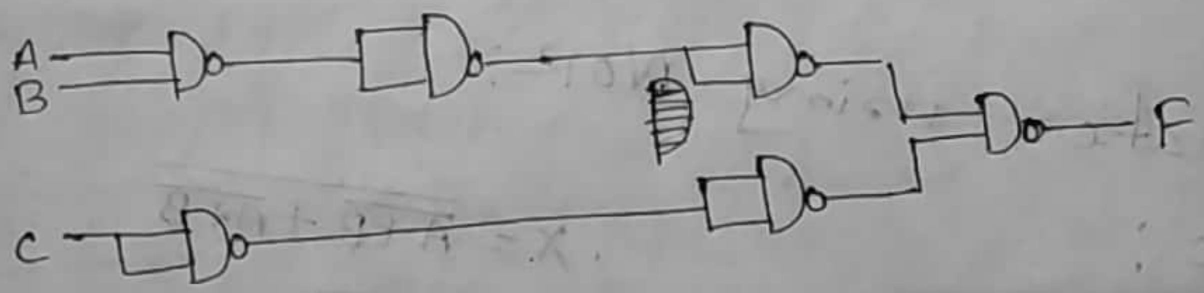
NAND Gate implementation of Boolean function:

$F = AB + \overline{C}$

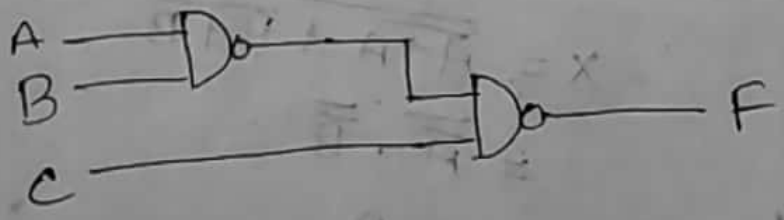
Using Basic Gates:



Using NAND Gate:

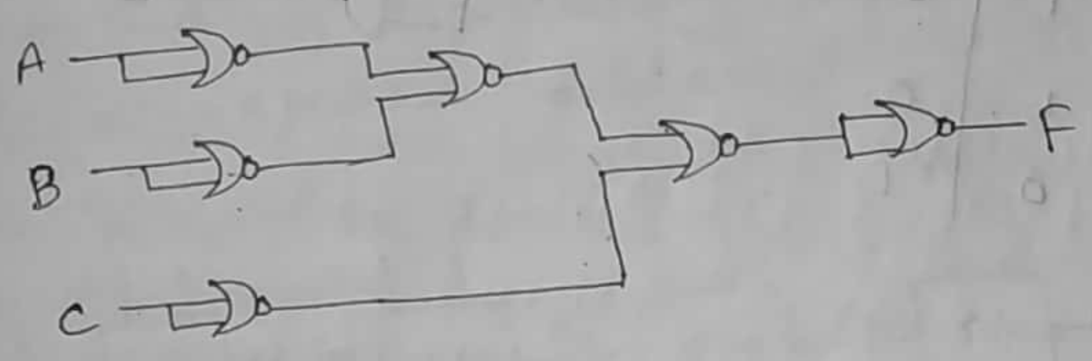


Simplify:



NOR Gate Implementation of Boolean Function:

$F = AB + \bar{C}$



Application of Combinational Circuit:

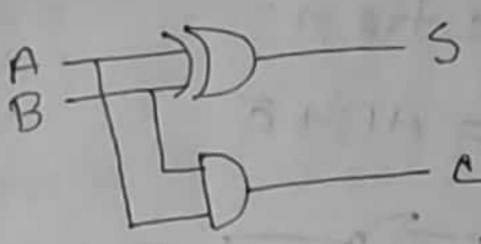
Adder: Binary addition differs from Boolean algebra as the result includes a carry term. An adder is a device that will add together two bits and give the result as the output. There are two kinds of adders, half adders and full adders.

Half Adder: A half adder just adds two bits together and gives a two bit output. If A and B are the input bits, then the sum bit S is the X-OR of A and B and the carry bit C will be the AND of A and B.

Truth table

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

Circuit Diagram



$S = A \oplus B$

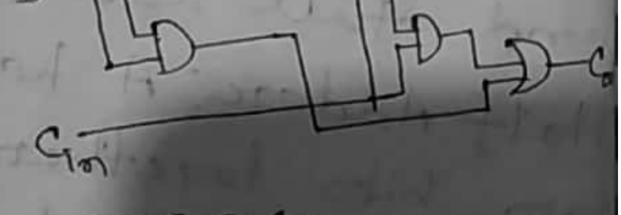
$C = AB$

Full Addere: A full addere, adds two inputs and a carrried input from anothere addere and outputs two bit (a sum bit S and, a carry bit C).

Truth table

C_{in}	A	B	S	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

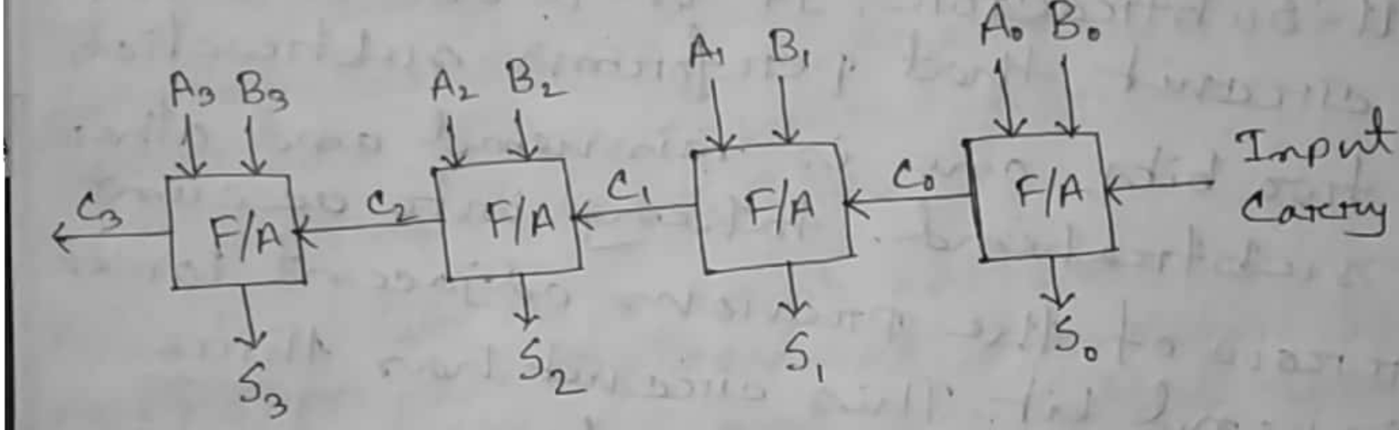
Circuit Diagram



$S = A \oplus B \oplus C_{in}$

$C_{out} = AB + (A \oplus B) C_{in}$

4-bit Parallel Adder: For a multiple-bit adder to work, each of the single-bit adders must have three inputs, including the carry from the next-lower-order adder.

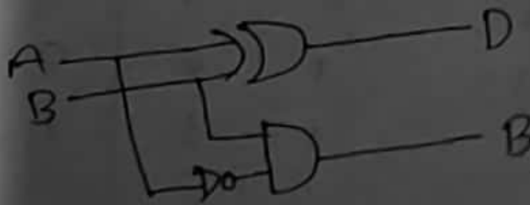


Half Subtractor: It is a combinational circuit which is used to perform subtraction of two bits. It has two inputs, the minuend and subtrahend. And ~~two~~ outputs the difference and borrow out

A	B	D	B_o
0	0	0	0
0	1	1	1
1	0	~~0~~ 1	0
1	1	0	0

$D = A \oplus B$

$B_o = \bar{A}B$



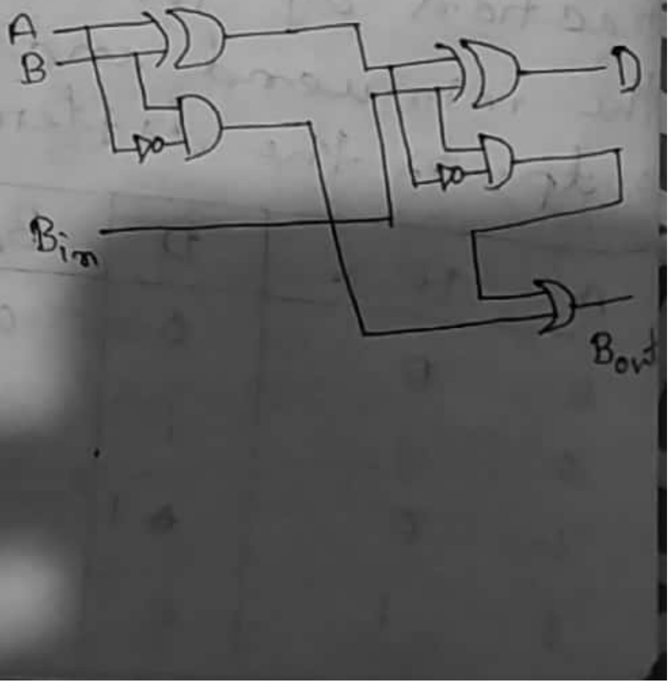
The borrow out signal is set ~~&~~ when the subtractor needs to borrow from the next digit in a multi-digit subtraction.

Full-Subtractor: It 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.

A	B	B_{in}	D	B_{out}
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

$D = A \oplus B \oplus B_{in}$

$B_{out} = B_{in}(\overline{A \oplus B}) + \bar{A}B$



Decoding: It is the process of conversion of an n-bit input code with an m-bit output code with $n \le m \le 2^n$ such that each valid code word produces a unique output code. The functional blocks for decoding are called n-to-m line ~~decoreds~~ decoders, where $m \le 2^n$ and generate 2^n or fewer minterms for n-input variables.

the n-input

Decoder: It is a combinal circuit that performs the process of decoding. It has n-input lines and 2^n-output lines.

2-to-4 line Decoder

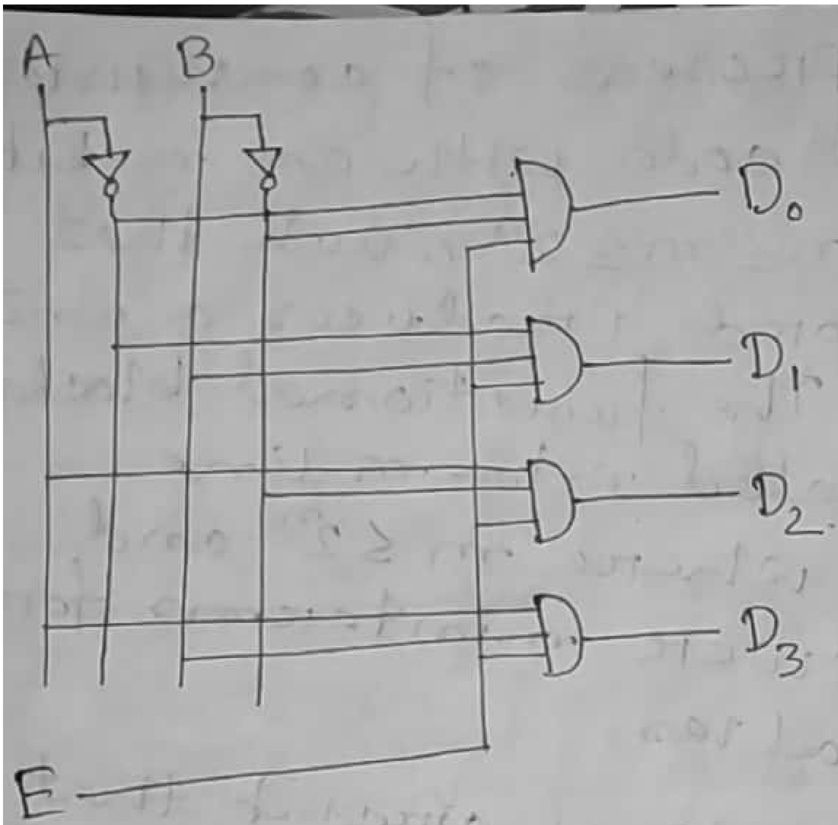
A	B	E	D_0	D_1	D_2	D_3
X	X	0	0	0	0	0
0	0	1	1	0	0	0
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	1

$D_0 = \bar{A}\bar{B}E$

$D_1 = \bar{A}BE$

$D_2 = A\bar{B}E$

$D_3 = ABE$



3-to-8 Decoder (without Enable)

A_2	A_1	A_0	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1							
0	0	1		1						
0	1	0			1					
0	1	1				1				
1	0	0					1			
1	0	1						1		
1	1	0							1	
1	1	1								1

$D_0 =$ ~~$\bar{A}\bar{B}\bar{C}$~~ $\bar{A}_2 \bar{A}_1 \bar{A}_0$

$D_1 =$ ~~$\bar{A}\bar{B}C$~~ $\bar{A}_2 \bar{A}_1 A_0$

$D_2 = \bar{A}_2 A_1 \bar{A}_0$

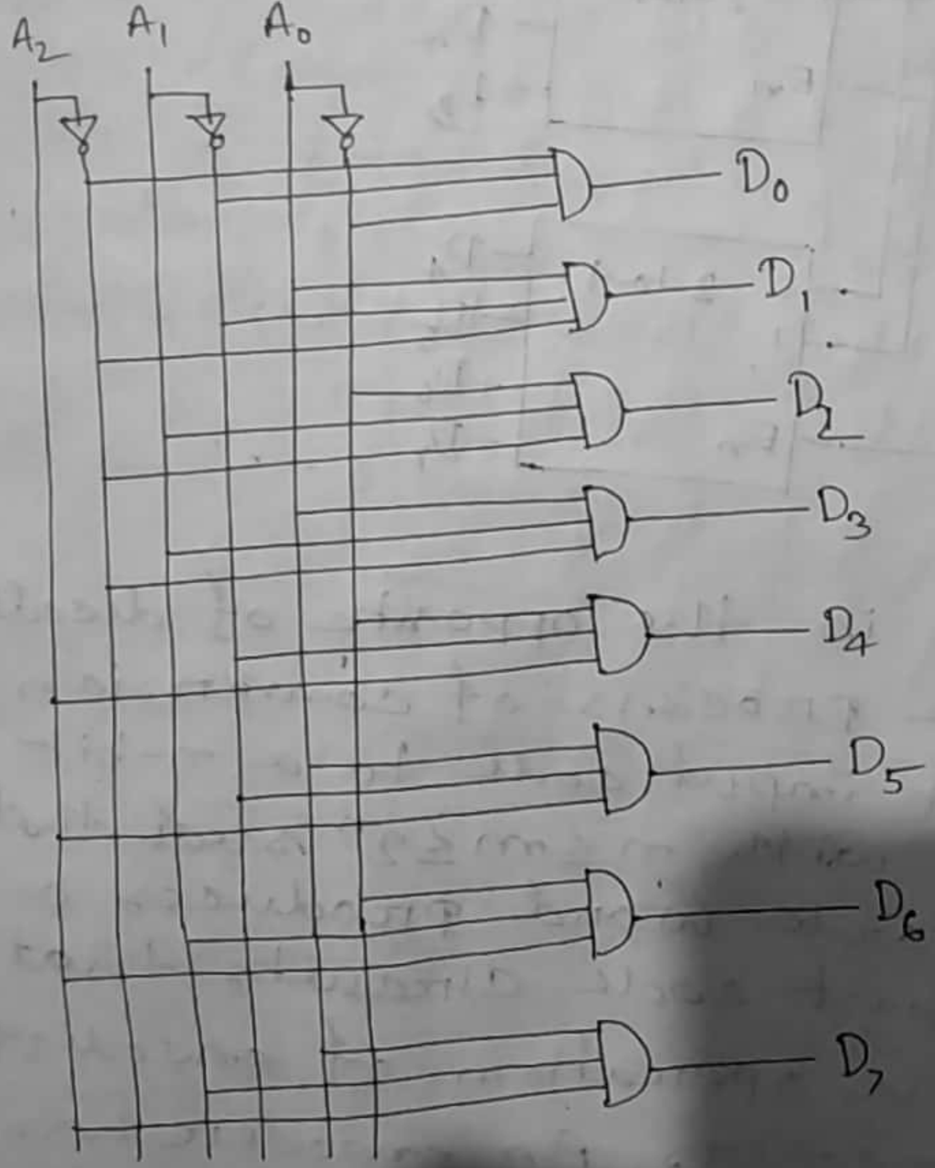
$D_3 = \bar{A}_2 \bar{A}_1 \bar{A}_0$

$D_4 = A_2 \bar{A}_1 \bar{A}_0$

$D_5 = A_2 \bar{A}_1 A_0$

$D_6 = A_2 A_1 \bar{A}_0$

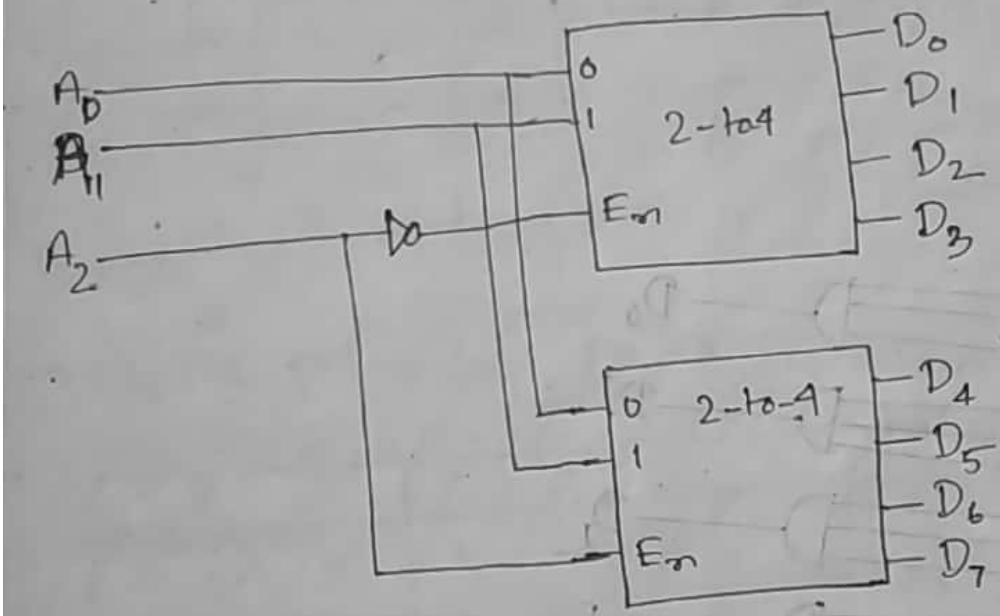
$D_7 = A_2 A_1 A_0$



Decoder Expansion:

3-to-8 using two 2-to-4

If we use A_2 from the previous truth table as Enable, then



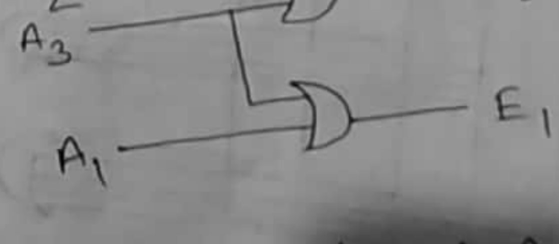
Encoding: It is the opposite of decoding. It is the process of conversion of an m-bit input code to a n-bit output code, with $n \le m \le 2^n$ such that each valid code word produces a unique output code. Circuits that perform the operation of encoding are called encoders. An encoder has 2^n or fewer input lines and n output lines which generate the binary code corresponding to the input values. Typically, an encoder converts a code containing exactly one bit that is 1 to a binary code corresponding to the position in which the 1 appears.

4-to-2 Encoder:

A_3	A_2	A_1	A_0	E_0	E_1
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	X	X

$E_0 = A_2 + A_3$

$E_1 = A_1 + A_3$

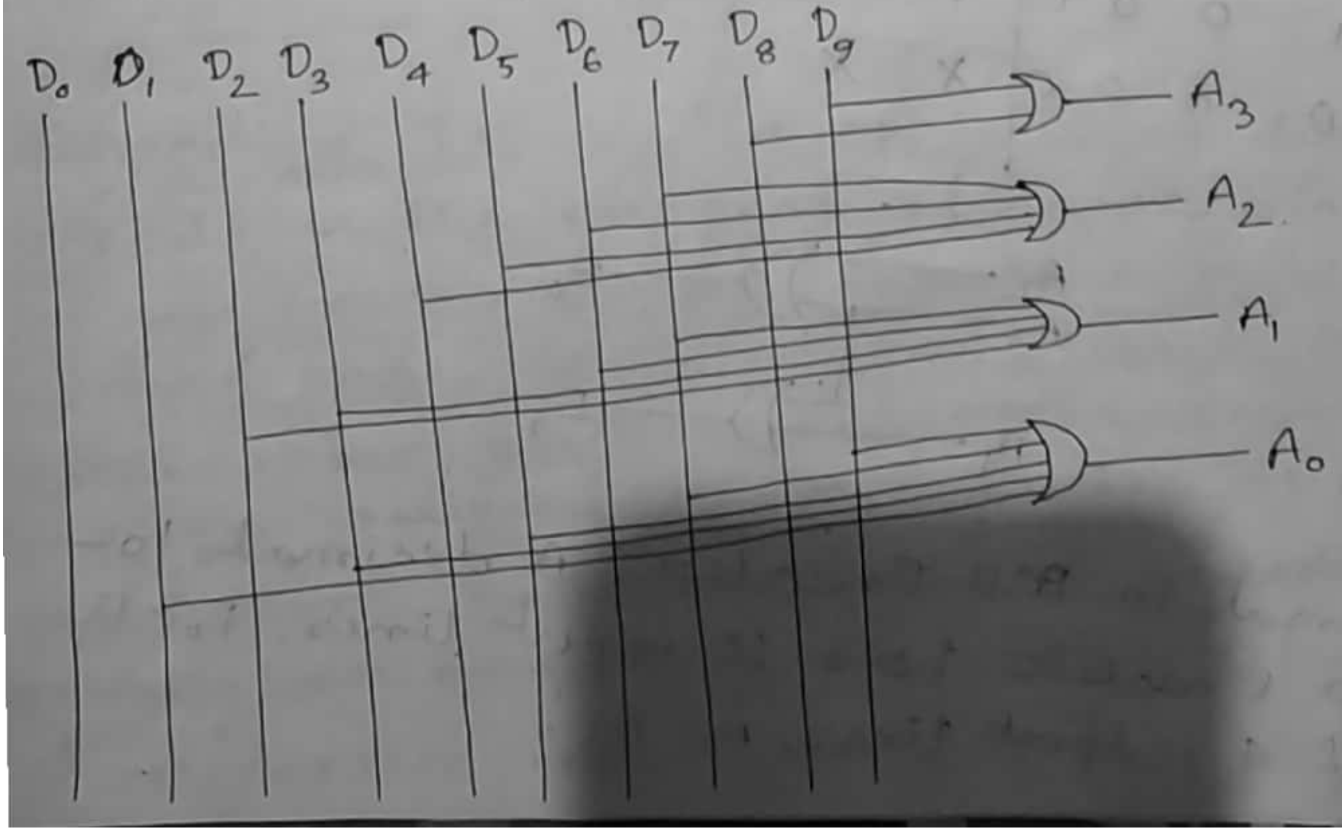


Decimal-to-BCD Encoder: A decimal-to-BCD Encoder has 10 input lines, $D_0 - D_7$ and 4 output lines, $A_0 - A_3$.

Decimal	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_3	A_2	A_1	A_0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	1	0	0	0	0	1
2	0	0	0	0	0	0	0	1	0	0	0	0	1	0
3	0	0	0	0	0	0	1	0	0	0	0	0	1	1
4	0	0	0	0	0	1	0	0	0	0	0	1	0	0
5	0	0	0	0	1	0	0	0	0	0	0	1	0	1
6	0	0	0	1	0	0	0	0	0	0	0	1	1	0
7	0	0	1	0	0	0	0	0	0	0	0	1	1	1
8	0	1	0	0	0	0	0	0	0	0	1	0	0	0
9	1	0	0	0	0	0	0	0	0	0	1	0	0	1

$A_0 = D_1 + D_3 + D_5 + D_7 + D_9$ $\quad$ $A_1 = D_2 + D_3 + D_6 + D_7 \cancel{+ D_9}$

$A_2 = D_4 + D_5 + D_6 + D_7$ $\quad$ $A_3 = D_8 + D_9$

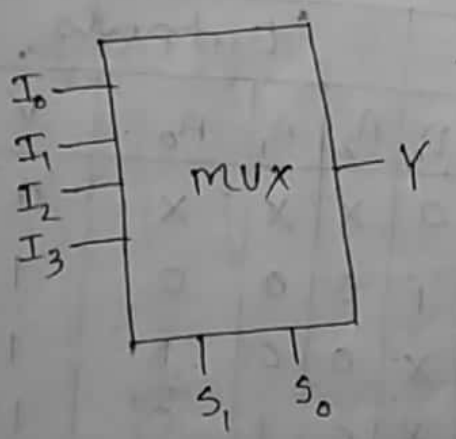


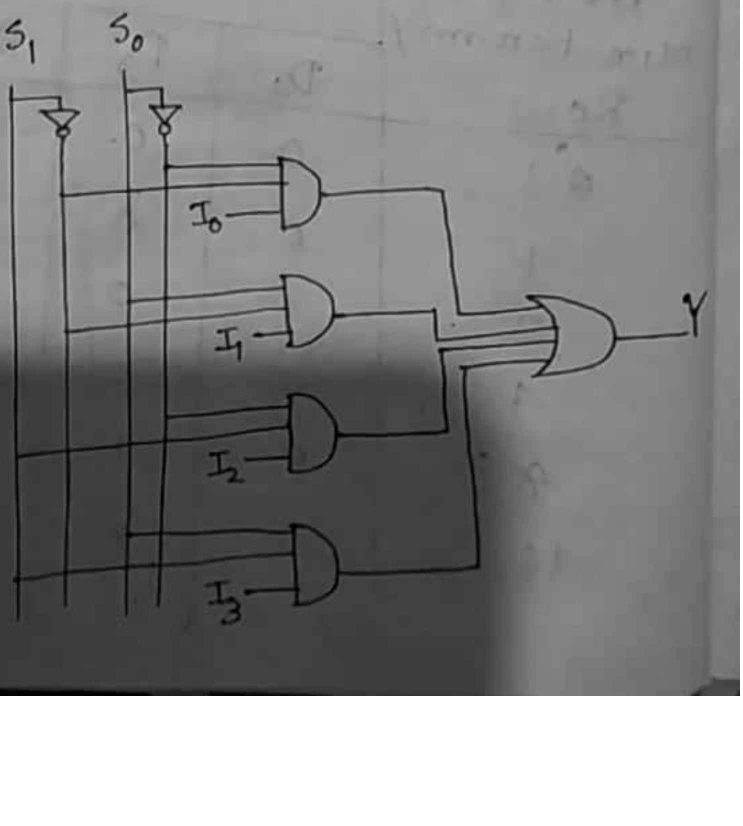
Priority Encoder: If more than one input has value of 1, then previously designed encoders do not work. The encoder that can work when multiple input has value of 1, it is called priority encoder. It can take all possible input values and still work properly. Among the 1's that appear, it selects the most significant input position (or the least significant input position) containing a 1 and responds with the binary code for that position.

No of Min-terms/ Row	Inputs					Outputs			
	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0	V
~~0~~ 1	0	0	0	0	0	X	X	X	0
1	0	0	0	0	1	0	0	0	1
2	0	0	0	1	X	0	0	1	1
4	0	0	1	X	X	0	1	0	1
8	0	1	X	X	X	0	1	1	1
16	1	X	X	X	X	1	0	0	1

Multiplexer: Also known as a data selector, is a device that selects between several input signals and forewards it to a single output line. A typical multiplexer has n control inputs $(S_0, S_1, \ldots, S_{n-1})$ called Selection inputs, 2^n information inputs $(I_0, I_1, \ldots, I_{2^n-1})$ and one output Y. A multiplexer can be designed to have m information inputs with $m < 2^n$ as well as n selection inputs.

$$Y = I_0 \bar{S}_0 \bar{S}_1 + I_1 S_0 \bar{S}_1 + I_2 \bar{S}_0 S_1 + I_3 S_0 S_1$$





S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Demultiplexer; It is a device that takes a single input line and routes it to one of several output lines based on the selection input. A demultiplexer of 2^n output line has n selection line.

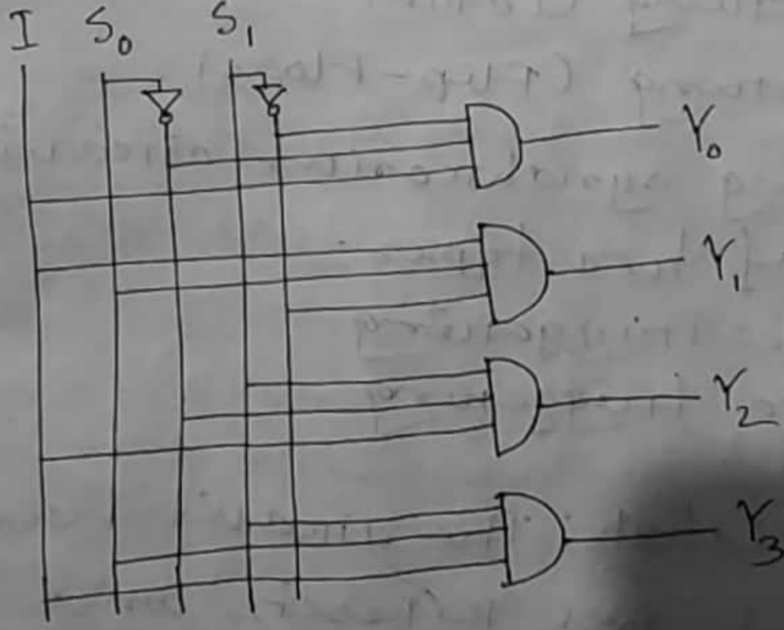
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

$Y_0 = I \bar{S_0} \bar{S_1}$

$Y_1 = I S_0 \bar{S_1}$

$Y_2 = I \bar{S_0} S_1$

$Y_3 = I S_0 S_1$



Sequential Circuit

The output of sequential circuit depends of on the current input and the current state of the circuit. It stores current state in a memory. eg Latch/Flip-Flop.

There are two types of sequential circuit:

1. Asynchronous (no-clock)
2. Synchronous (clock)

Synchronous circuits are of two types:

a. Label triggering (Latch)

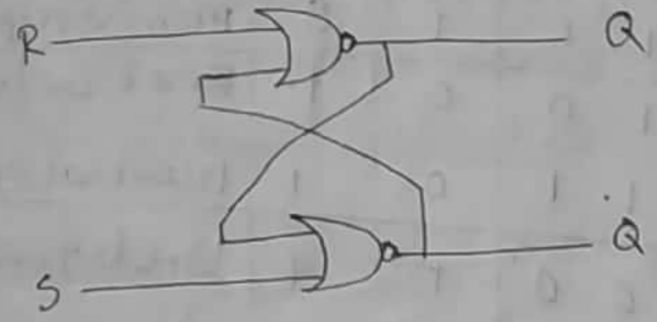
b. Edge triggering (Flip-Flops)

Edge triggering synchronous circuits ~~again~~ can be of two types:

i. Positive edge triggering

ii. Negative edge triggering

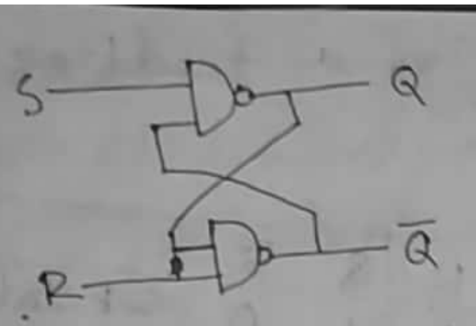
Basic (NOR) S-R Latch: The circuit has two inputs, S (Set) and R (Reset), and two outputs, Q and $\bar{Q}$, and consists of two NOR gates connected in a feedback arrangement.



S	R	Q	$\bar{Q}$	
1	0	1	0	Set state
0	0	1	0	Memory
0	1	0	1	Reset State
0	0	0	1	memory
1	1	0	0	Undefined

1. When S and R, both are 0, S-R Latch is in memory state.
2. When S is 0 and R is 1, S-R Latch is in Reset state.
3. When S is 1 and R is 0, S-R Latch is in Set state.
4. When both S and R are 1, it generates illogical result.

Basic (NAND) S-R Latch/ $\bar{S}$-$\bar{R}$ Latch: This Circuit has similare characteristics as the previous, but it consists of two NAND gates.

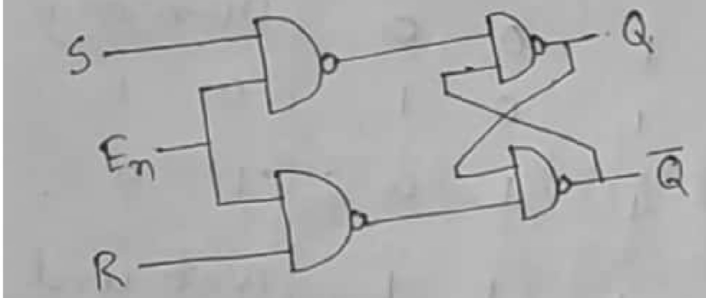


S	R	Q	$\overline{Q}$	
0	1	1	0	Set State
1	1	1	0	Memory
1	0.	0	1	Reset sate
1	1	0	1	Memory
0	0	1	1	Undefined

1. When S and R both are 1, the latch works as memory.
2. When S is 0 and R is 1, the latch is in Set State.
3. When S is 1 and R is 0, the latch is in Reset state.
4. When S and R are both 0, it generates illogical result.

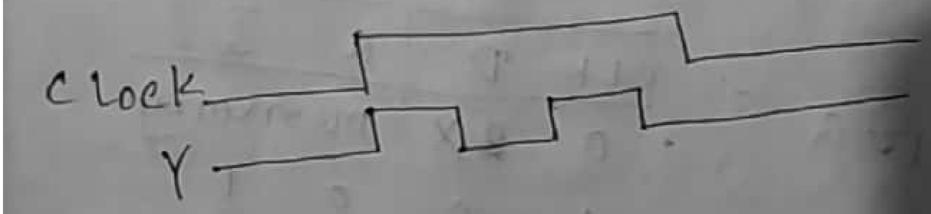
S-R Latch with control (Enable): When the enable input is high, the circuit acts just like NOR Latch. When the enable input is low the Set Reset inputs are disabled ~~har~~ and have no effect on the outputs leaving the circuit in the latched state.

This kind of latch circuit can be constructed eithere from two AND gates and two NOR Gates, ore from foure NAND gates.

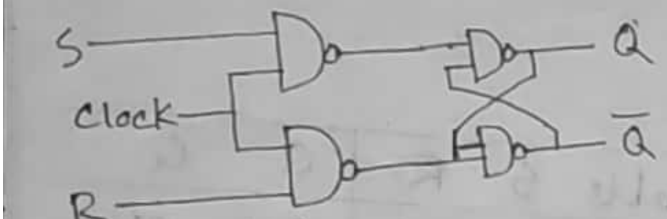


Enable	S	R	Q	$\overline{Q}$
0	X	X	Memory	
1	0	0	Memory	
1	0	1	0	1
1	1	0	1	0
1	1	1	Invalid/ Not used	

Latch timing problem: When two different inputs are sent to latch on the same clock pulse, it does not know which one to choose. This problem is solved using Flip-Flops.



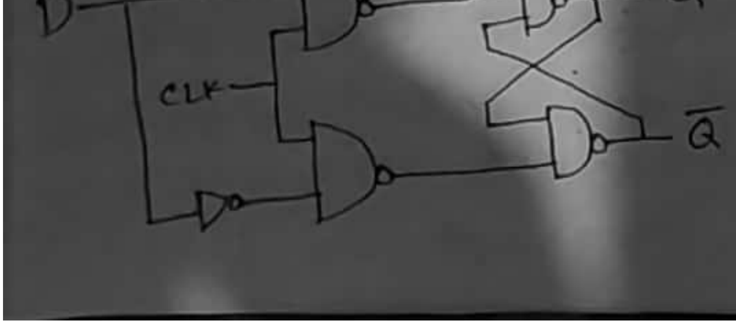
S-R Flip-Flop with Clock; The output of this circuit changes only when ~~the~~ a clock pulse occurs.



CLK	S	R	Q	$\overline{Q}$
0	X	X	Memory	
1	0	0	Memory	
1	0	1	0	1
1	1	0	1	0
1	1	1	Not used	

The problem with this flip-flop is when both S and R becomes 1. The output in this case is illogical. So it is not used.

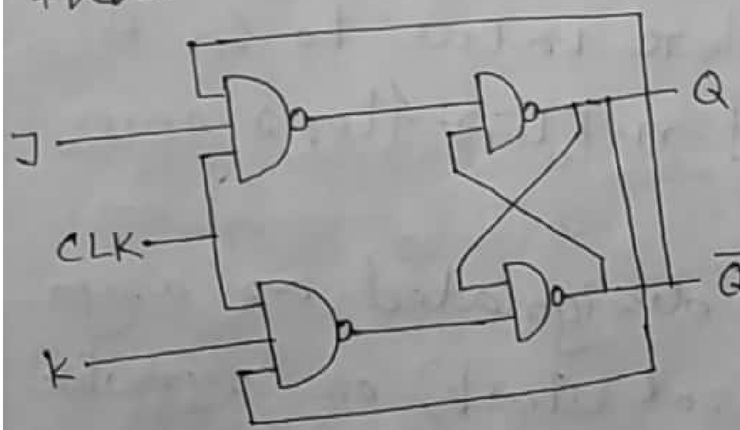
D Flip-Flop: It overccomes the illogical output problem of S-R FF using only one input.



CLK	D	Q	$\overline{Q}$
0	X	memory	
1	0	0	1
1	1	1	0

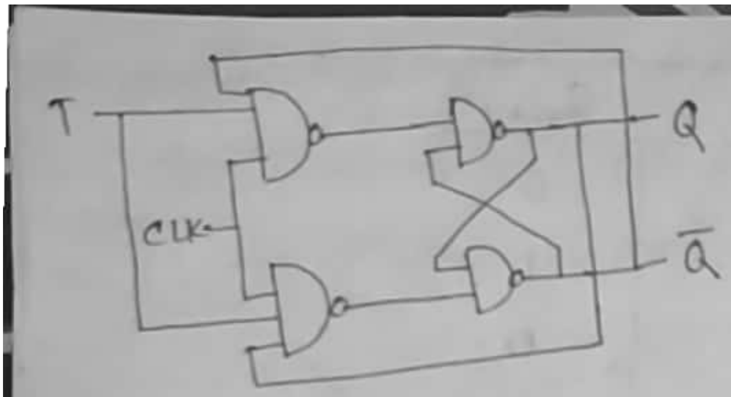
Q_n	D	Q_{n+1}
0	0	0
0	1	1
1	0	0
1	1	1

J-K Flip-Flop: Although it has two inputs like S-R FF, it has such circuit that its inputs are valid.



J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

T Flip-Flop: The "T" stands fore toggle. when the input is high, the output toggles or flips.

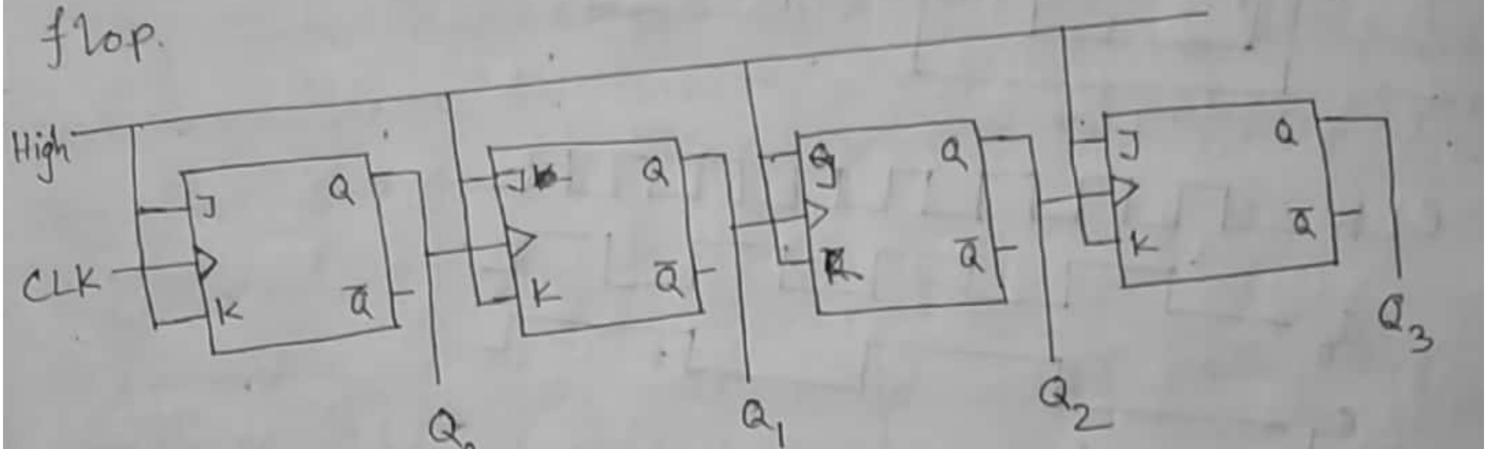


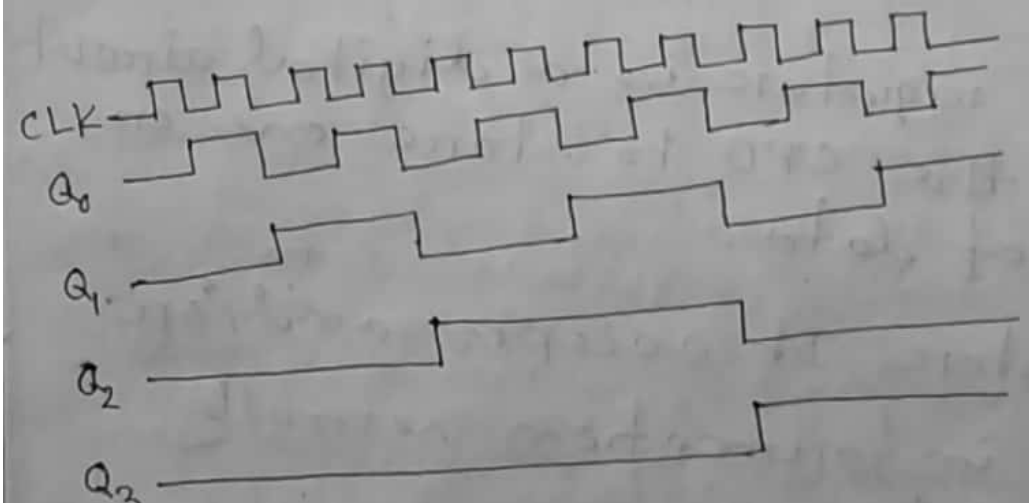
T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Counterr: It is a registerr whose value is easily incremented by 1 modulo the capacity of the registeter. When the maximum value is achieved, the next increment sets the counterr value to 0. A counterr made up of n-Flip-flops can count up to 2^n-1.

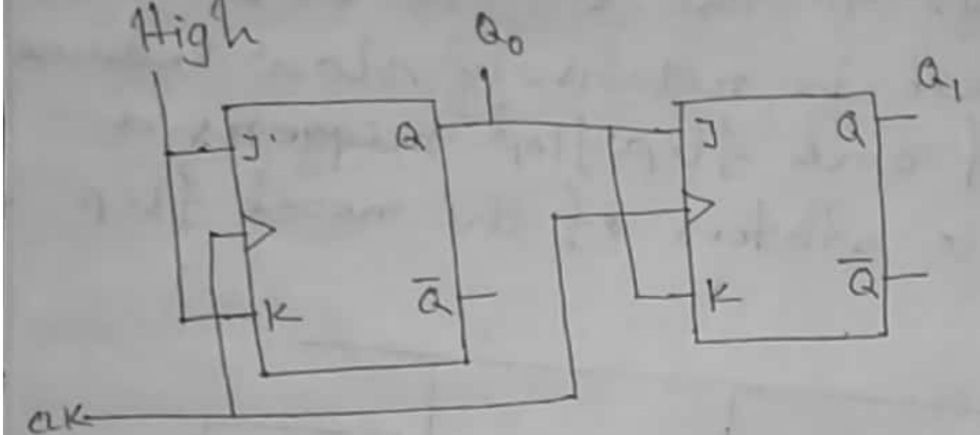
Counterrs can be designated as asynchronus (no universal clock) or synchronus (universal clock), depending on the way in which they operate.

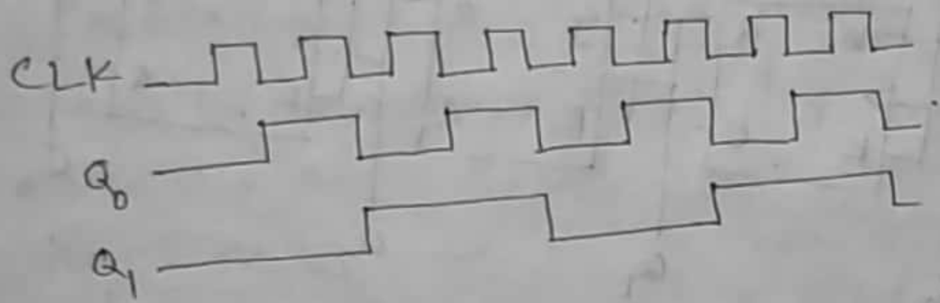
Assynchronus Counterr: It is also referred to as ripple counterr, because the change that occurrs to increment the countere starts at one end and ripples through to the othere end. This type of countere is relatively slow because the output of one flip-flop triggers a change in the status of the next flip-flop.





Synchronus Countere: In this type of countere, all of the flip-flops change states at the same time, unlike ripple countere. Since this type is much fastere it is the kind used in CPUs.





Register: A register is a digital circuit used within the CPU to store one or more bits of data.

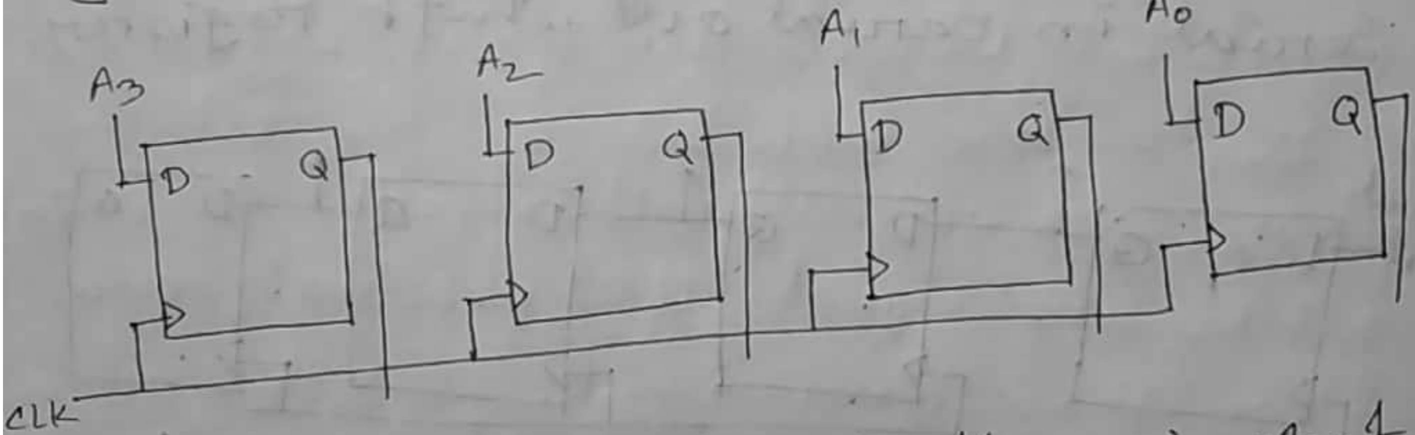
Shift Register: It accepts and/or transfers information serially.

They are of 4 types.

1. Serial in serial out shift register
2. Serial in parallel out shift register
3. Parallel in serial out shift register

4. Parallel in parallel out shift registere.

Parallel in Parallel out shift register
The parallel data is loaded simultaneo-
usly into the registere, and transferr
ed togethere to theire respective outputs
by the same & the same clock pulse.

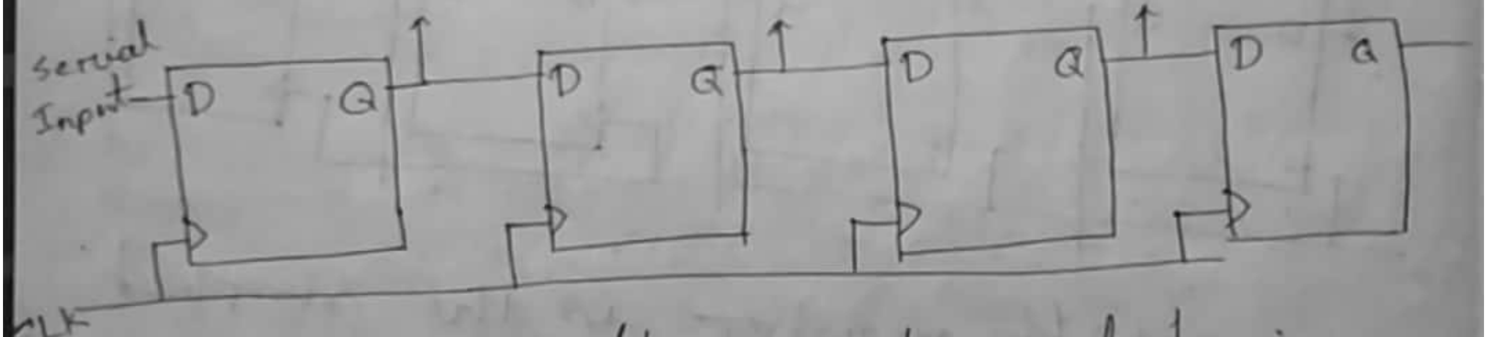


This shift registere is the simplest
of the foure configurations, as it has
only three connections, the parallel
input which determines what enters
the flip-flop, the parallel output and
the sequencing clock signal.

Before any clock pulse, data is

connected to the flip-flops. At the first clock cycle, the flip-flops stores the corresponding bit. Fore another clock, the flip-flops outputs theire memory (i.e. bits they stored in the previous cycle).

Serial in Serial out shift registere:



In this shift registere, data is shifted "IN" and "OUT" serially. one bit at a time, eithere in left ore right direction undere clock control.

In every clock cycle one bit is fed to a registere, The registere stores the data and sends a copy of that bit to the next neighbourc for next clock cycle.