Digital Electronics

Hitesh
Pranav

~~Boolean algebra~~ BOOLEAN ALGEBRA

- Data containing binary values 1's and 0's
  1 ⇒ ON state / True / High voltage
  0 ⇒ OFF state / False / Low voltage
  ] → bit

- Used to analyse and simplify logic or digital circuits
- No of possible input states = 2^n where n = no. of inputs

- Logic gates :
  - Basic gates : AND, OR, NOT
  - Derived gates : NAND, NOR, EXOR, EXNOR

Q. Explain principle of duality of Boolean with an eg.
Q. What is need for Boolean laws

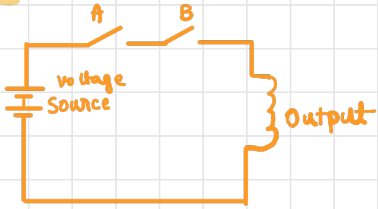
Q. Explain Commutitave Law $A + B = B + A$
Q. Explain Associative Law $A+(B+C)=(A+B)+C$, $A.(B.C)=(A.B).C$
Q. Explain Distributive Law
Q. Explain Demorgan's Law
Q. Explain Consensus Theorem

- Actual Boolean Algebra :
  - $A+1 = A$
  - $A+0 = A$
  - $A.1 = A$
  - $A.0 = 0$
  - $A+A = A$
  - $A.A = A$
  - $A+\bar{A} = 1$
  - $A.\bar{A} = 0$
  - $A+\bar{A}B = A+B$

- Demorgan's Laws
  - $\overline{A+B} = \bar{A}.\bar{B}$
  - $\overline{A.B} = \bar{A}+\bar{B}$
- $\bar{\bar{A}} = A$ → Involuntary Law
- $A + AB = A$ → Absorption Law
- $(A+B)(A+C) = A+BC$ → Distributive Law
- $A.(B+C) = AB + AC$ → Dual of Distributive Law
- $AB + \bar{A}C + BC = AB + \bar{A}C$ → Consensus Theorem
- $(A+B)(\bar{A}+C)(B+C) = (A+B)(\bar{A}+C)$ → Dual of Consensus Theorem

AND GATE

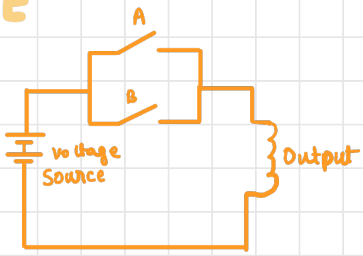
- $Y = A.B$



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

OR GATE

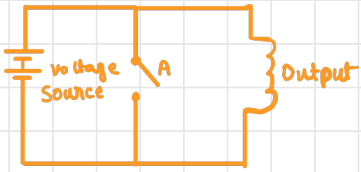
- $Y = A + B$



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

NOT GATE

- $Y = \bar{A}$

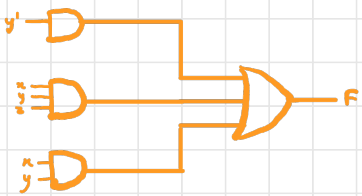


Input	Output
A	Y
0	1
1	0

SUM OF PRODUCTS (SOP)

- The boolean expression containing AND terms called product terms with one or more literals each. The sum denotes the ORing of these terms.

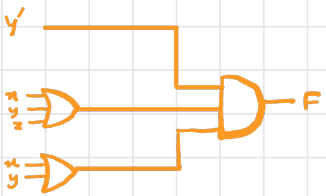
ex: $F_1 = xyz + y' + xy$



PRODUCT OF SUM

- The boolean expression containing OR terms called sum terms with one or more literals each. The sum denotes the ANDing of these terms.

ex: $F = y'.(x+y+z).(x+y)$



Q. $A.B + A.C \Rightarrow$ Express on Canonical SOP Form

$f = A.B.1 + A.1.C$

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

$= ABC + AB\bar{C} + ABC + A\bar{B}C$

$= ABC + AB\bar{C} + A\bar{B}C$

	A	B	C	AB	AC	AB+AC	
0	0	0	0	0	0	0	
1	0	0	1	0	0	0	
2	0	1	0	0	0	0	
3	0	1	1	0	0	0	
4	1	0	0	0	0	0	
5	1	0	1	0	1	1	→ $A\bar{B}C$
6	1	1	0	1	0	1	→ $AB\bar{C}$
7	1	1	1	1	1	1	→ ABC

$f = \Sigma(5,6,7)$

Canonical POS

x	y	z	Term	Design
0	0	0	$x+y+z$	M_0
0	0	1	$x+y+z'$	M_1
0	1	0	$x+y'+z$	M_2
0	1	1	$x+y'+z'$	M_3
1	0	0	$x'+y+z$	M_4
1	0	1	$x'+y+z'$	M_5
1	1	0	$x'+y'+z$	M_6
1	1	1	$x'+y'+z'$	M_7

→ Each individual terms is called Maxterm

For $(A+B+C).(A'+B'+C).(A'+B+C')$

$= M_0 . M_6 . M_5$

Q. $F = A + BC$

$= A \cdot 1 \cdot 1 + 1 \cdot B \cdot C$

$= A \cdot (B + \bar{B})(C + \bar{C}) + (A + \bar{A}) \cdot B \cdot C$

$= (AB + A\bar{B})(C + \bar{C}) + (ABC) + \bar{A}BC$

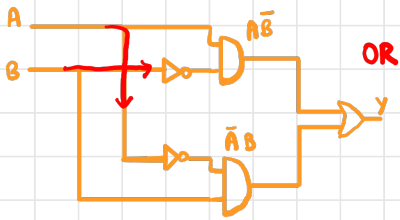
$= ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + ABC + \bar{A}BC$

$= ABC + \bar{A}BC + A\bar{B}C + AB\bar{C} + A\bar{B}\bar{C}$

$= \Sigma(3, 4, 5, 6, 7)$

XOR GATE

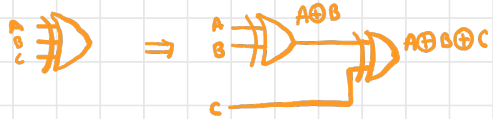
- $Y = A \cdot B' + A' \cdot B$
  $Y = A \text{ XOR } B$
  $Y = A \oplus B$



OR

A	B	C	A⊕B	A⊕B⊕C
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1

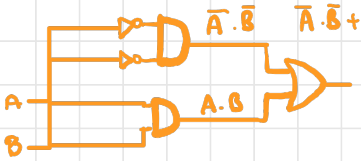
- 3 Inputs $\Rightarrow$ $Y = A \oplus B \oplus C$



XNOR GATE

- $Y = (A \oplus B)'$

$\Rightarrow$



$\bar{A} \cdot \bar{B} + AB = \overline{A \oplus B} = A \odot B$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

- XOR is used in processor's Arithmetic Logic Unit for binary addition
- XOR is used to generate pseudo random numbers in hardware
- To generate parity bits and error detection
- Equality detector

UNIVERSAL GATES

- NAND
- NOR

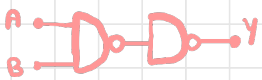
→ Any logic circuit can be made using these 2 logic gates

- They are used in all integrated circuits

NOT

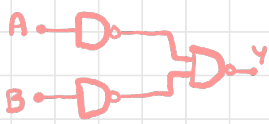
$Y = (A \cdot A)' = A' + A' = A'$

AND



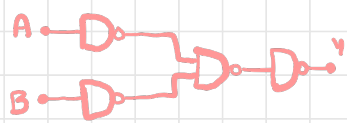
$Y = ((A \cdot B)')' = A \cdot B$

OR



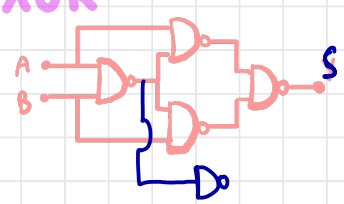
$Y = \left((A+B)'\right)' = (A' \cdot B')' = A + B$

NOR



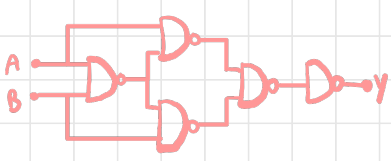
$Y = \left(((A+B)')'\right)' = \left((A' \cdot B')'\right)' = (A+B)'$

XOR



$Y = A(A \cdot B)' + B(A \cdot B)'$
$= A(A'+B') + B \cdot (A'+B')$
$= A \cdot A' + AB' + A'B + BB'$
$= AB' + A'B$

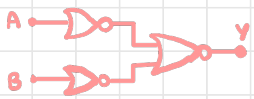
XNOR



$Y = (A \oplus B)' = A \odot B$

USING NOR GATES

AND

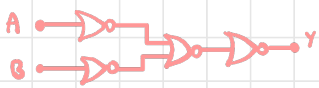


$Y = ((A \cdot B)')' = (A' + B')' = A \cdot B$

OR

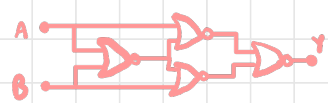
$Y = ((A+B)')' = A + B$

NAND



$Y = ((A' + B')')' = (A \cdot B)$

XNOR



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

$(\overline{A} \cdot \overline{B} + A)' + (A \cdot \overline{B} + B)')'$

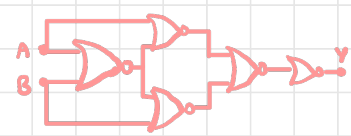
$(\overline{(A+B)} + \overline{(\overline{A}+B)})'$

$(\overline{A} \cdot B + A \cdot \overline{B})' = (\overline{A}\overline{B} + AB)$

$= A' \cdot B' + A \cdot B$

$= A \odot B$

XOR



$Y = (A'B' + A \cdot B)' = A'B + AB'$

$= A \oplus B$

Q. $F = X \cdot Y + Z$

$= ((X \cdot Y + Z)')'$

$= (\overline{X \cdot Y} \cdot \overline{Z})'$

Q. $A \cdot B \cdot C + A' + A \cdot B' \cdot C$

$= A(B + B') \cdot C + A'$

$= AC + A'$

$= (A' + A) \cdot (A' + C)$

$= A' + C$

Q. $Y = A'B'C' + AB'C' + A'B' + AC'$

$= A'B'(1 + C') + AC'(B' + 1)$

$= A'B' + AC'$

Q. $Y = A(B + C) + DE$

$= AB + AC + DE$

$= \overline{(\overline{AB + AC + DE})}$

$= \overline{(\overline{AB} \cdot \overline{AC} \cdot \overline{DE})}$

Combinational circuits

- Constructed by interconnection of logic gates whose outputs determined from present combination of inputs.

- It performs an operation that can be specified logically by Boolean function

ex: Binary adders, Multiplexers etc.,

- **Binary Adders**
  - Half Adder : Comb^n Circuit that performs addition of 2 bits
    Full Adder : Comb^n Circuit that performs addition of 3 bits

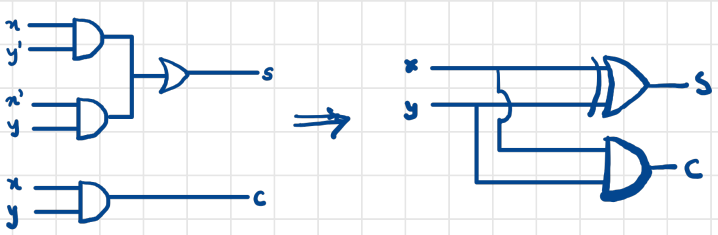
  - They are the basic components of Adders like:
    - Ripple carry adder
    - Carry look ahead adder
    - and, other fast adders

- **Half Adder**
  - x & y are binary inputs
    C & S are binary outputs

$S = x'.y + xy' = x \oplus y$

$C = x.y$

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



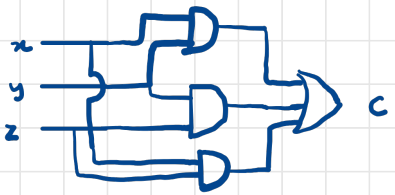
- **Full Adder**

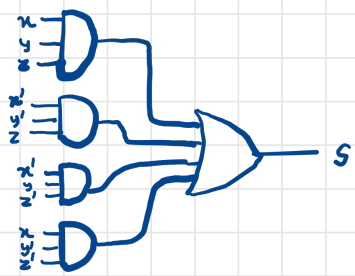
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

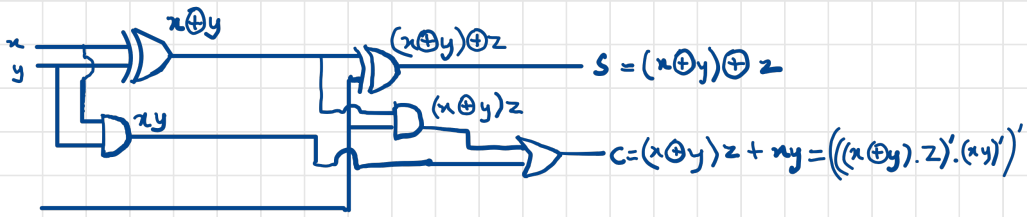
$C = xy + yz + xz = (x \oplus y)z + xy$

$= x'yz + xy'z + xyz' + xyz$

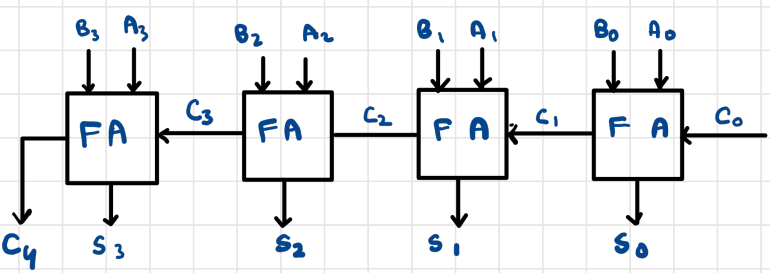
$S = xyz + x'y'z + xy'z' + x'yz' = x \oplus y \oplus z$







- **Four Bit Adder**
  - Using four Full Adder, Ripple adder circuit is constructed



ex:

C_3	C_2	C_1	C_0	
1	1	1	0	→ 14
A_3	A_2	A_1	A_0	
1	1	0	1	→ 13
0	1	1	1	→ 7
B_3	B_2	B_1	B_0	
1	0	1	0	0 → 20
C_4	S_3	S_2	S_1	S_0

$$C = x'yz + xy'z + xyz' + xyz$$
$$= x'yz + xy'z + xy(z'+z)$$
$$= x'yz + xy'z + xy$$
$$= xyz + x(y'z + y)$$
$$= x'yz + x(y + z)$$
$$= x'yz + xy + xz$$
$$= y(x'z + x) + xz$$
$$= y(x + z) + xz$$
$$= xy + yz + xz$$

$$\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \rightarrow 10 \\ 1 & 1 & 0 & 0 \rightarrow 12 \\ \hline 1 & 0 & 1 & 1 & 0 \rightarrow 22 \end{array}$$

$x'y'z + x'yz' + xy'z' + xyz$

$x'(y'z + yz') + xy'z' + xyz$

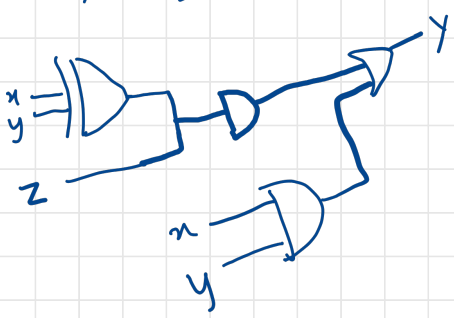
$x'(y \oplus z) + x(y \odot z)$

$= x \oplus (y \oplus z)$

$x'y'z + x'yz' + xy'z' + xyz$

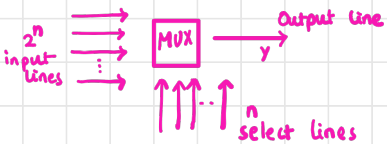
$z(x \oplus y) + xy(z' + z)$

$(x \oplus y) \cdot z + xy$



MULTIPLEXER AND DEMULTIPLEXER

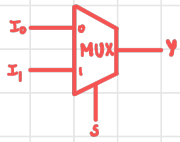
- Multiplexer is the combinational circuit of multiple inputs single output.
  The select line determines which input is connected to output
- MUX has 2^n inputs
  n select lines
  1 output

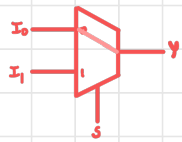


- Used for data compression & shares single transmission channel.
- Demultiplexer inverses the function of multiplexer.

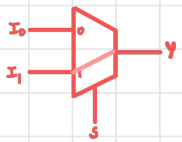
2:1 MUX

- Block Diagram





If S=0, Y=I0



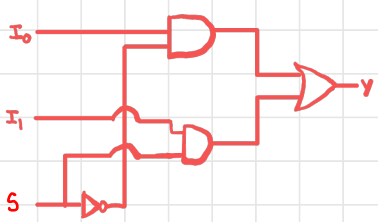
If S=1, Y=I1

S	Y
0	I_0
1	I_1

$Y = S'.I_0 + S.I_1$ → Using NAND,

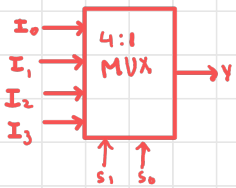
$((S'I_0 + SI_1)')'$

$= ((SI_0').(S'I_1'))'$

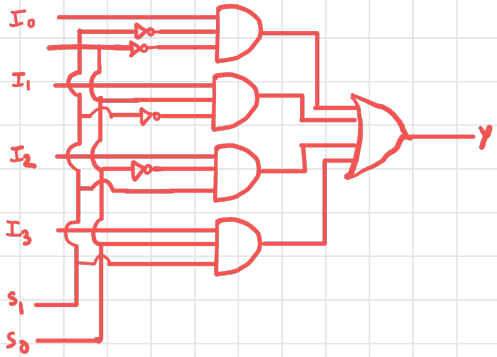


4:1 MUX

- Block Diagram



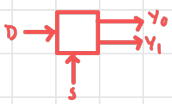
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



$$Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3$$

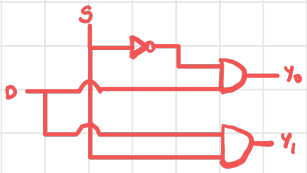
1:2 Demux

- Block Diagram



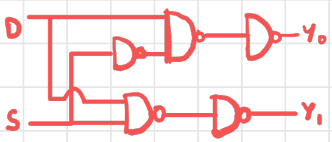
Select input	Outputs	
S	Y_0	Y_1
0	D	–
1	–	D

$$Y_0 = S'.D$$
$$Y_1 = S.D$$

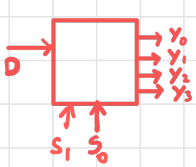


Using NAND Gates,

$$Y_0 = ((S'.D)')' = (S + D')'$$
$$Y_1 = ((S.D)')' = (S' + D')'$$



1:4 DEMUX

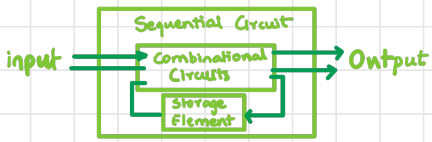


Select Input		Outputs			
S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

$$(S_0'.S_1'D) + (S_0'S_1D) + (S_0S_1'D) + (S_0S_1D)$$

SEQUENTIAL CIRCUITS

- Combinational circuit output depend only on present input
- We want circuits that produce output depending on current & past input values
-



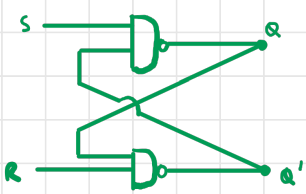
- Sequential Circuits of 2 Types :
  - **Synchronous Circuits**
    - The state of device changes only at discrete times in response to clock pulse
  - **Asynchronous Circuits**
    - Circuit is not synchronized by clock signal. The outputs change directly in response to change in input

-

Flip-Flop	Latch
• Flip-flop utilizes edge triggering	• Latch follows level triggering
• Clock signal is present	• Clock signal is absent
• Slow operating speed	• Comparitively Fast
• Flip-flop is synchronous Works based on clock signal	• Latch is Asynchronous Doesn't work based on time signal
• Sensitive to applied input & Clock signal	• Sensitive only to applied input signal when enabled

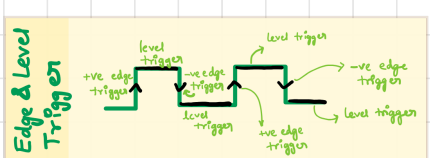
R-S LATCH (Reset - Set)

-



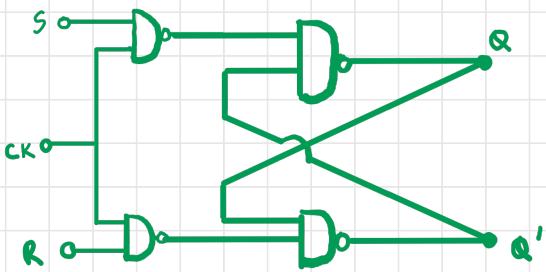
R	S	Q
1	1	Q_{prev}
1	0	1
0	1	0
0	0	Invalid

Q. Clearly explain about S-R latch



- At Idle State, S & R are at 1 (quiescent / idle state)
  - If S drives to 0, Q changes to 1 (Set)
  - If R drives to 0, Q changes to 0 (Reset)
- S & R can't be 0 at same time (Can't set & reset data at same time)

- **GATED R-S LATCH / R-S FLIP-FLOP**



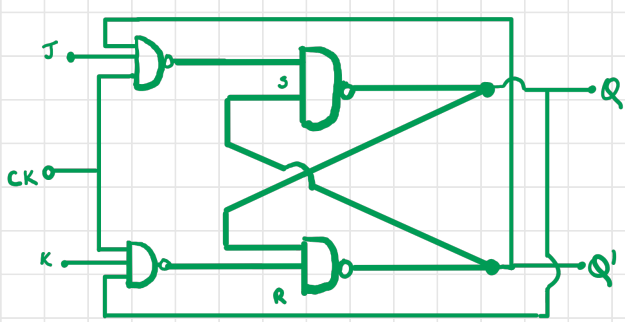
CK	S(t)	R(t)	Q(t)	Q(t+1)	Operation
1	0	0	0	0	No Change
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	set
1	1	0	1	1	
1	1	1	0	X	Undefined
1	1	1	1	X	
0	X	X	X	Q(t)	No Change

∵ CK - Clock

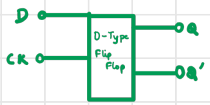
- Flip-flops are building blocks of sequential circuits but are built from latches
- Flip-flops are sensitive to signal change.
  transfer data only at single time instant & won't change till next signal change
- They are used as registers
  Output & next state input changes when there's a change in clock pulse (+ve/-ve)
  ↳ Edge Triggered Circuit

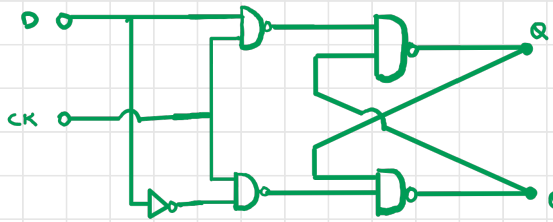
- **J-K FLIP FLOP**
  - S & R inputs of SR Flip flops replaced by J-K inputs respectively

	Clock	Input		Output			Description
	CK	J	K	$Q_{(n)}$	Q'	$Q_{(n+1)}$	
Same as S-R Latch	X	0	0	1	0	1	Memory
	X	0	0	0	1	0	No Change
	↓	0	1	1	0	0	Reset
	X	0	1	0	1	0	Q >> 0
	↓	1	0	0	1	1	Set
	X	1	0	1	0	1	Q >> 1
Toggle Action	↓	1	1	0	1	1	Toggle
	↓	1	1	1	0	0	



- **DATA (D) FLIP FLOP**





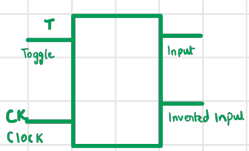
Q. Explain with a neat diagram D - Flipflop

- It ensures S & R are never equal to 1 at same time

CK	D	Q_n	$\bar{Q}$	Q_{n+1}	Description
↓ >> 0	X	Q	$\bar{Q}$	Q	Memory No change
↑ >> 1	0	0	1	0	Reset Q >> 0
↑ >> 1	0	1	0	0	Set Q >> 1
↑ >> 1	1	0	1	1	Toggle
↑ >> 1	1	1	0	1	

- D Flip Flop will store & Output whatever is applied to DATA Input
- Once the clock input goes Low, set & reset input of flip-flop held at 1, so state won't change)
- ↑ & ↓ indicate direction clock pulse (cuz D-Type flip-flops are assumed as edge triggered)

- **T - FLIP FLOP**

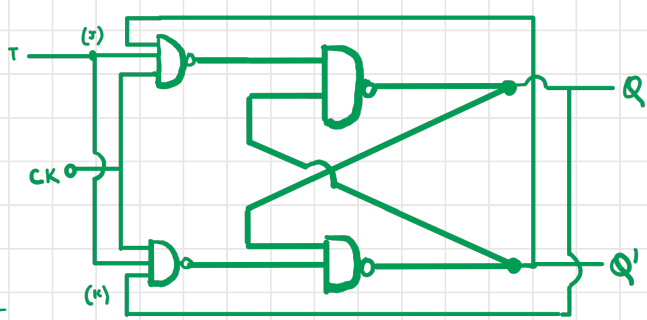


- It is a synchronous device, where high to low (or) low to high transitions is passed through clock signal which changes output state of Flip Flop

	Previous		Next	
T	Q_{prev}	Q'_{prev}	Q_{next}	Q'_{next}
0	0	1	0	1
0	1	0	1	0
1	0	1	1	0
1	1	0	0	1

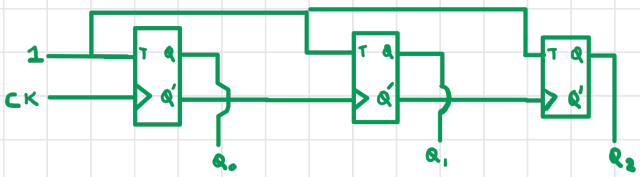
Toggle -
$\Rightarrow Q_{next} = Q_{prev}$

Toggle on
$\Rightarrow Q_{next} = \overline{Q_{prev}}$



- **Advantages :**
  - The Flip-flop has toggle input & a clock which inverts value of Flip-flop
  - Used for designing counters

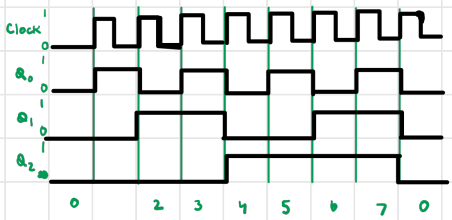
- **3-bit Asynchronous up-counter**



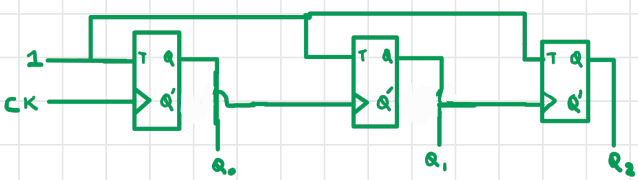
MSB – Most Significant Bit
LSB – Least Significant Bit
ex → 101 (MSB ← 1 0 1 → LSB)

No. of -ve edge of clock	Q_0 LSB	Q_1	Q_2 MSB
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	0
4	0	0	1
5	1	0	1
6	0	1	1
7	1	1	1

Timing Diagram

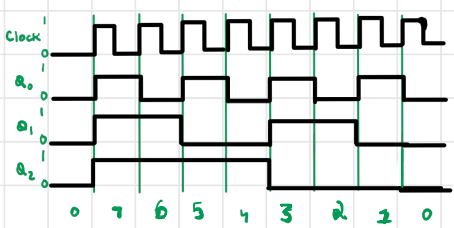


- **3-bit Asynchronous down-counter**

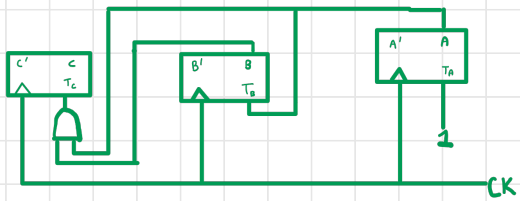


No. of -ve edge of clock	Q_0 LSB	Q_1	Q_2 MSB
0	0	0	0
1	1	1	1
2	0	1	1
3	1	0	1
4	0	0	1
5	1	1	0
6	0	1	0
7	1	0	0

Timing Diagram



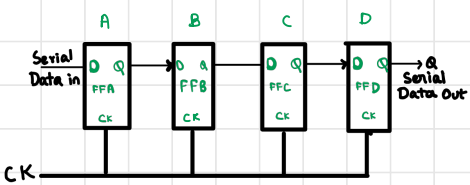
- **Design of 3 bit Synchronous up-Counter**

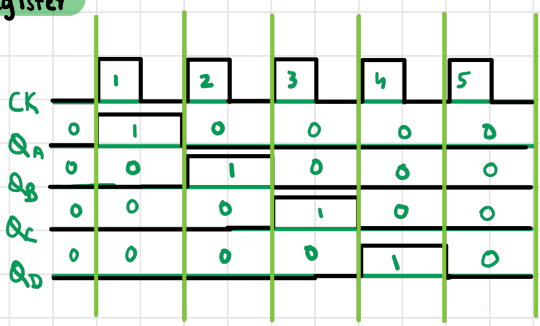


- It consists of 3 flip-flops & 1 2-input AND Gate
- In Flip-flop state,
  - 0 → No change from present to next state
  - 1 → Yes change from present to next state

	Present State			Next State			Flip-Flop State		
	C	B	A	C^+	B^+	A^+	T_C	T_B	T_A
0 ←	0	0	0	0	0	1 (1)	0	0	1
1 ←	0	0	1	0	1	0 (2)	0	1	1
2 ←	0	1	0	0	1	1 (3)	0	0	1
3 ←	0	1	1	1	0	0 (4)	1	1	1
4 ←	1	0	0	1	0	1 (5)	0	0	1
5 ←	1	0	1	1	1	0 (6)	0	1	1
6 ←	1	1	0	1	1	1 (7)	0	0	1
7 ←	1	1	1	0	0	0 (0)	1	1	1

- **4-bit Serial Input Serial Output shift register**





Clock Pulse No	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0

Q. Realise JK, T & D Flipflop
Q. Realise SR Flipflop using NAND Gates
Q. Write 1:2 DEMUX Expression