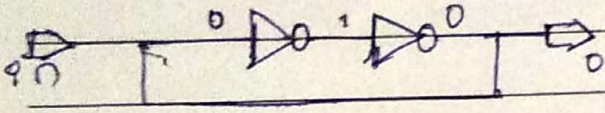
MODULE - 3

DATE | | SEQUENTIAL LOGIC CIRCUIT. 11th Oct 19

* At a particular instant of time for combination of inputs we get different combinations of outputs - Combinational Circuits

* A Sequential ckt which depends on present inputs & past states not on past inputs. previous states.

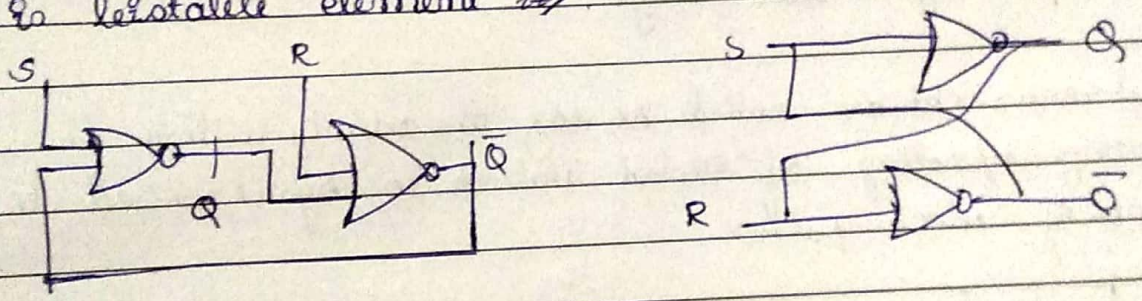
How do we store a value electronically?



It is locked, latched, stored
CASCADED 2 INVERTOR

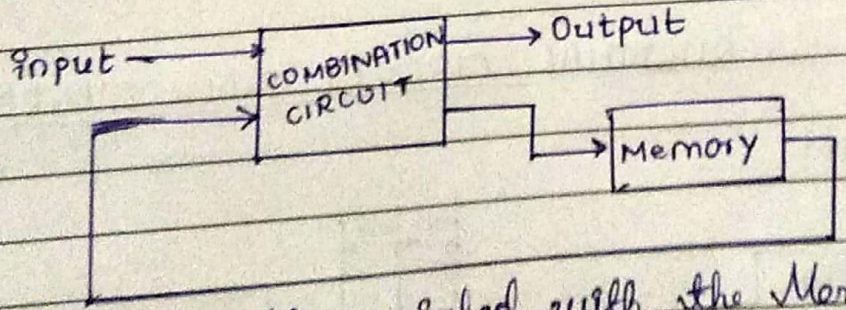
For work it as flip flop / latch add ip port & op port

* It is bistable element ⇒



SEQUENTIAL CKTS: The op of Sequential ckt not only depends on the present inputs but also on the past history (previous state)

BASIC BLOCK DIAGRAM OF SEQUENTIAL CKT:



Combination ckt associated with the Memory is known a Synchronus.

Sequential ckt is classified into

1) SYNCHRONUS SEQUENTIAL CIRCUIT: It is a system whose behaviour can be defined from the knowledge of its signal at discrete instance of time

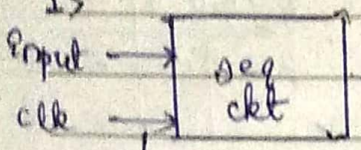
2) ASYNCHRONUS: It is a system whose behaviour depends upon at any instant of time & the order in which the input changes.

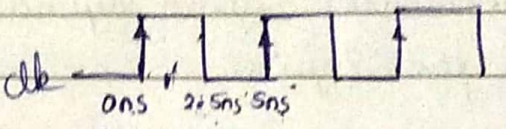
DATE

Discrete time → regular time

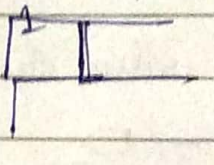
Outputs depends on input in synchronised with the clock (it may be pos edge or neg edge)

1>





→ discrete instant of time

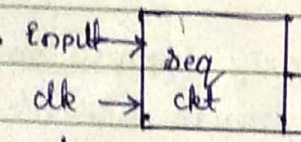


for 0ns → 1 → ip 1 → op

0.25ns → 0 → ip 1 → op

only at pos edge it will change

* The synchronous ckt are called so coz, the activity within the ckt & the resulting updating of stored values is synchronised to the occurance of clock pulses

2> 

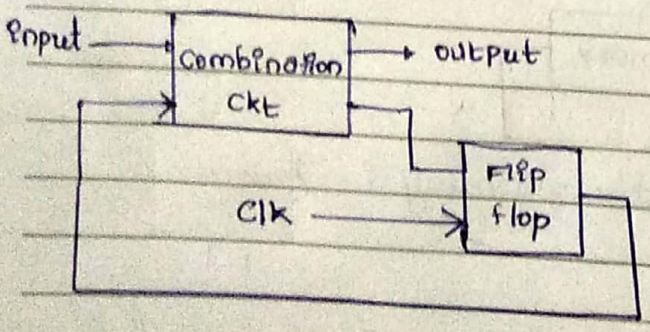
→ at any instant ~~input~~ time

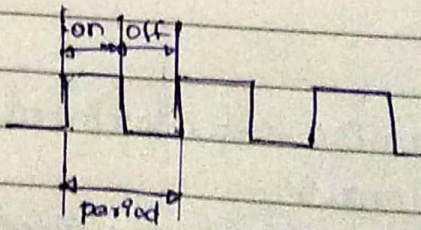
→ edge triggered

for 0ns → 1 - ip 1 → op

0.25ns → 0 - ip → 0 - op

SYNCHRONUS CLOCKED SEQUENTIAL CIRCUIT BLOCK DIAGRAM





STORAGE ELEMENTS:

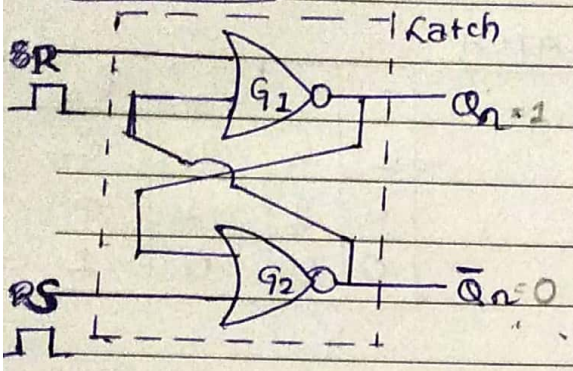
1) Latches → it always obey the set to Zero → level triggered
2) Flipflops. → edge

DATE

The storage elements that operate with signal levels are reffered to as Latches, and those that are controlled by a clock transition are reffered to as flipflops

LATCHES: [Set Reset Latches] : It can latch/store a value.

NOR Gates:



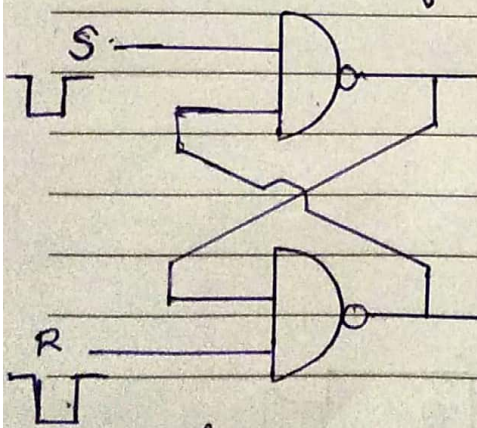
TRUTH TABLE / STATE TABLE

S	R	Q_{n+1}	$\bar{Q}_{n+1}$
0	0	Q_n	$\bar{Q}_n$
0	1	0	1
1	0	1	0
1	1	?	?

→ forbidden

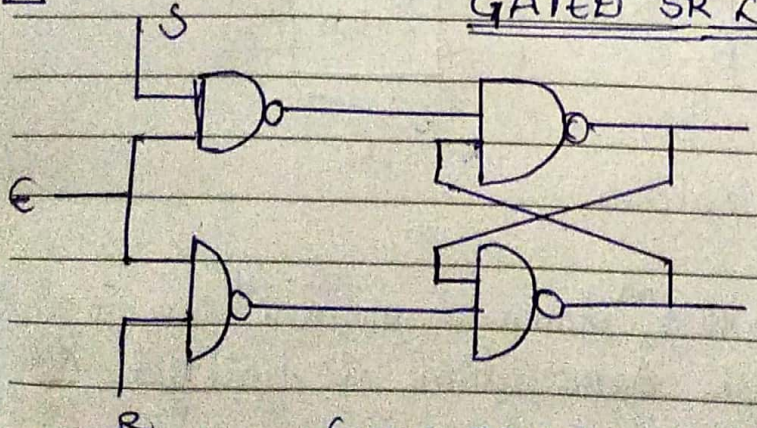
for S = 1 & R = 1 we perfectly get value as 0 & 0 if we gone to next again S = 0 & R = 0 we get Q_{n+1} & $\bar{Q}_{n+1}$ as 1 & 1 but we should get 0 & 0 only hence the state where S = 1 & R = 1 as not valid state

NAND Gates: feed Inputs as $\bar{S}$ & $\bar{R}$ ($\bar{S}\bar{R}$ Latch)



S	R	Q_{n+1}	$\bar{Q}_{n+1}$
0	0	Q_n	$\bar{Q}_n$
1	0	1	0
0	1	0	1
1	1	?	?

GATED SR LATCH



SR Latch with Control Input

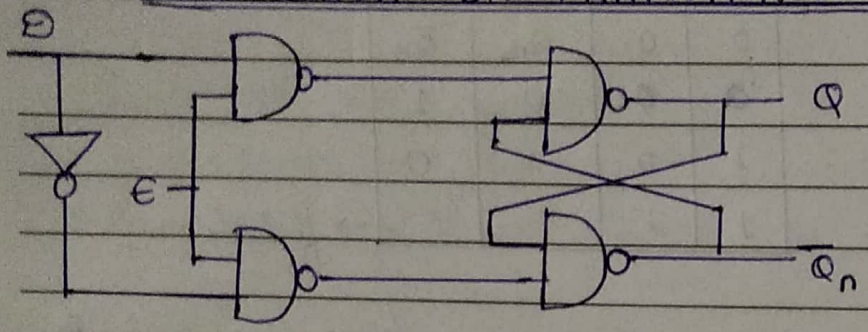
LEVEL TRIGGERED SR Latch

PES UNIVERSITY

DATE

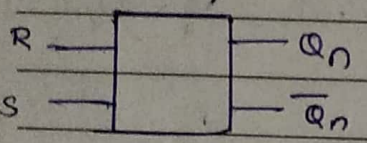
E	S	R	Q_{n+1}	$\bar{Q}_{n+1}$
1	x	x	Q_n	$\bar{Q}_{n+1}$
0	0	0	Q_n	$\bar{Q}_{n+1}$
0	0	1	0	1
0	1	0	1	0
0	1	1	invalid	

D- LATCH OR TRANSPARENT LATCH



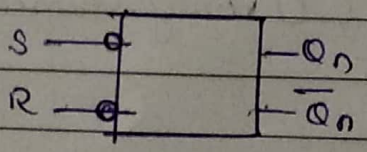
E	D	Q_{n+1}	$\bar{Q}_{n+1}$
1	x	Q_n	$\bar{Q}_n$
0	1	1	0
0	0	0	1

1> SR Latch (NOR Gates)



(a)

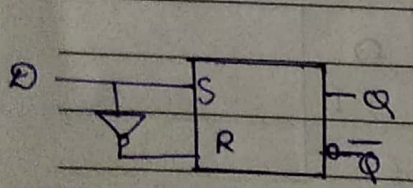
2> SR Latch (NAND)

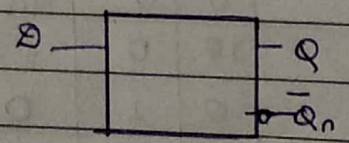


SR

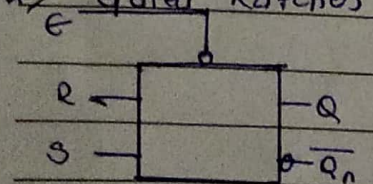
Data Latch
Delay ||

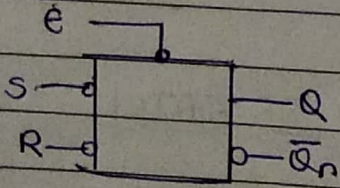
3> D - Latch

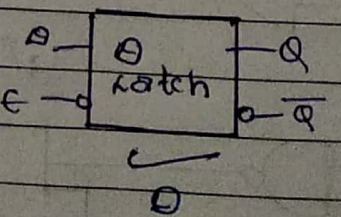




4> Gated Latches







Write a verilog code for D- Latches

= blocking statement → it blocks 2nd statement until 1st one is get done
<= non blocking " → at the same time it will get executed, concurrently

1. exp 8

DATE

latch.v

```
module (D, E, Q, Qb);
input D, E;
output Q, Qb;
reg Q, Qb;
always @ (D, E)
  begin
    if (!E)
      begin
        Q <= D;
        Q <= ~D;
      end
  end
endmodule
```

q = 1'b0;
q = 1'b1;

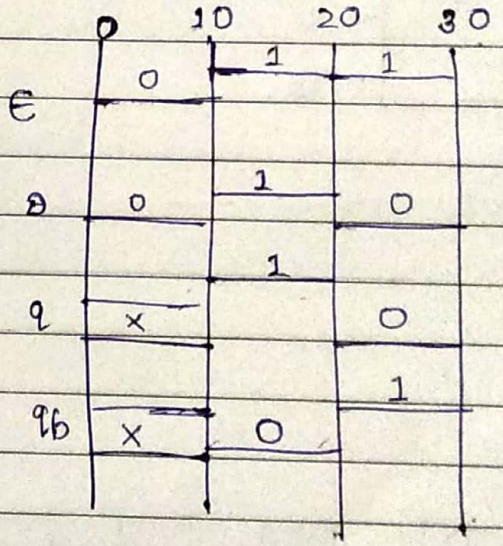
use non blocking statements whenever we are designing latches & flip flops.

latch.tb

```
module dlatch;
reg D, E;
wire Q, Qb;
dlatch T1 (D, E, Q, Qb);
initial
begin
```

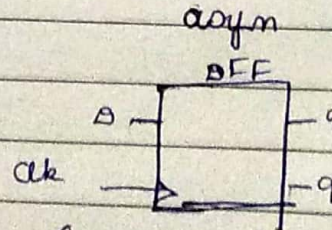
Test bench

```
module delatch_tb;
reg D, E;
wire Q, Qb;
delatch T1 (D, E, Q, Qb);
initial
begin
    E = 0 ; D = 0;
#10 E = 1 ; D = 1;
#10 D = D;
# 10 D
```

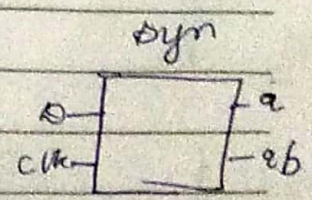


D-FF

clk	D	q	qb
0/1	x	q	qb
↑	1	1	0
↑	0	0	1

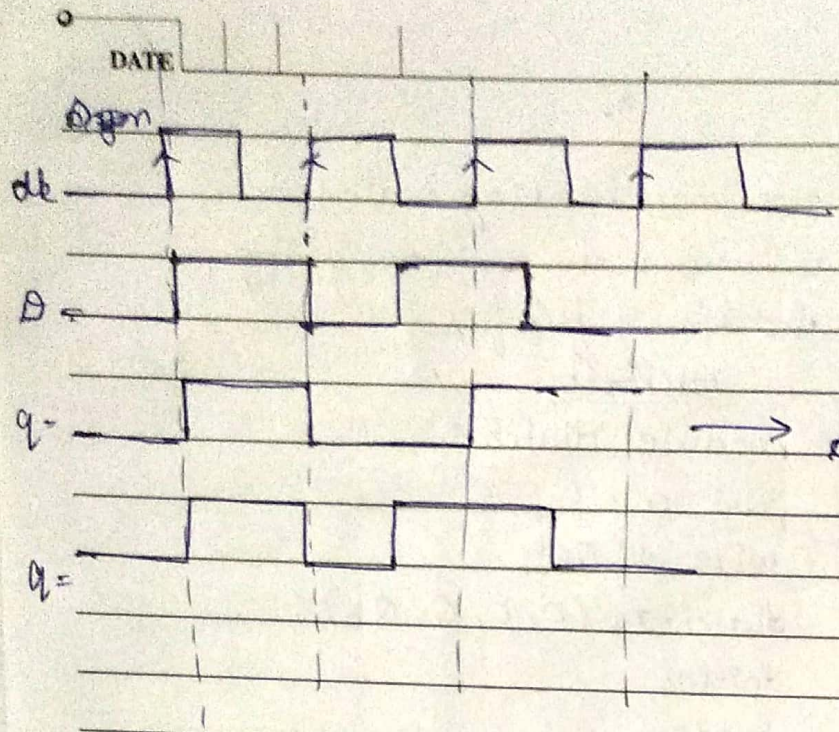


inputs - highest priority



clk - high priority

DATE

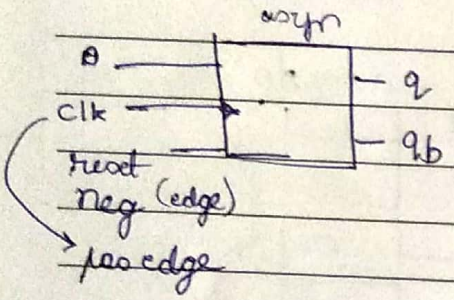


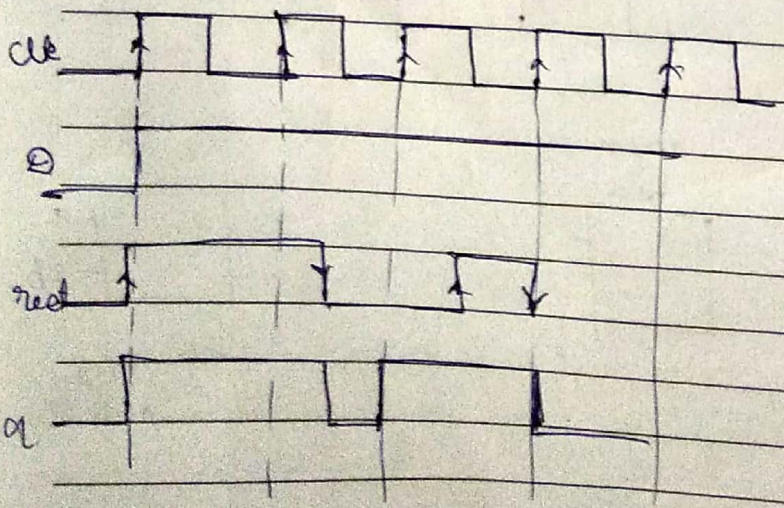
→ Synchronous

```
module dff_asyn (D, clk, rst, q, qb);
input D, clk, rst;
output q, qb;
reg q, qb;
always @ (posedge dk or negedge rst)
begin
  if (!rest)
    q <= 0;
    qb <= 1;
else
rest
  q <= D
  qb <= ~D;
end
endmodule
```

for Synchronous (only posedge)

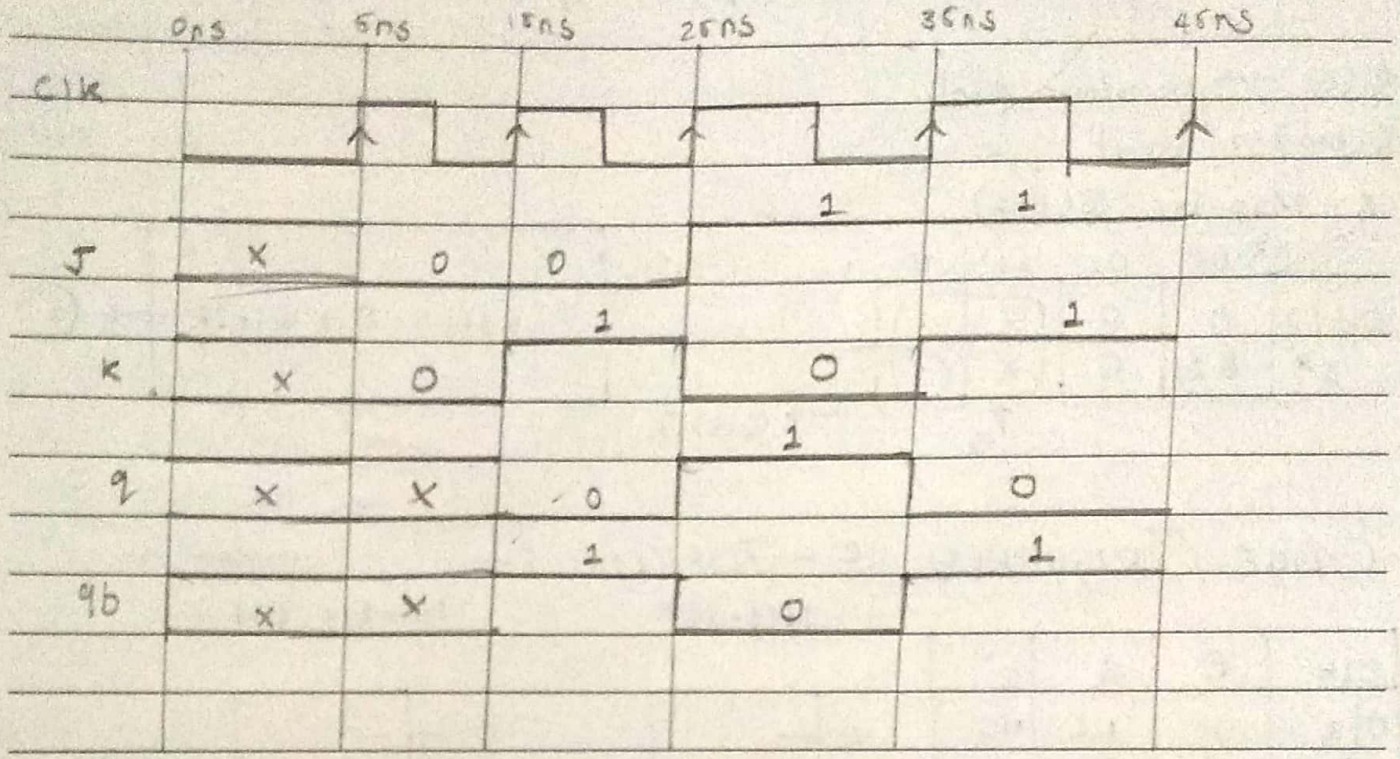
D FF





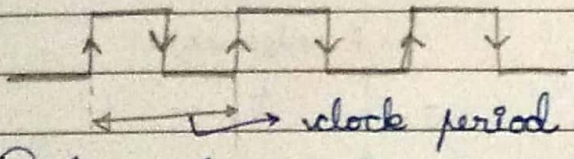
DATE

JK - ff waveform



FLIP-FLOPS

* All flip-flops are edge triggered
* clk - dynamic input → it keeps on changing at regular interval
* Static ip → it changes statically



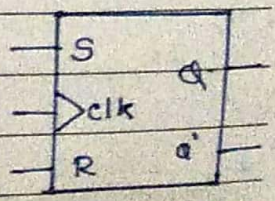
* Duty cycle is ratio of on time & off time
* clk period will be have both posedge & negedge

1) EDGE TRIGGERED ISR FLIP-FLOP :

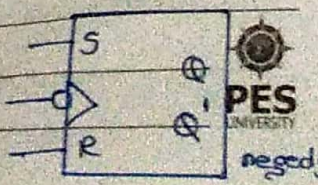
STORAGE ELEMENTS : FLIP FLOP.

ET / ST

Clk	S	R	Q	Q'
0/1	x	x	NC	NC
↑	0	0	"	"
↑	0	1	0	1
↑	1	0	1	0
↑	1	1	?	?



posedge triggered



negedge

DATE

Characteristic eqn :- how do we represent o/p interms of i/p

$Q(t)$ → previous state

$Q(t+1)$ → next "

K-Map for $Q(t+1)$

SR / Q(t)	00	01	11	10
0	0	0	x	1
1	1	0	x	1

↑ S

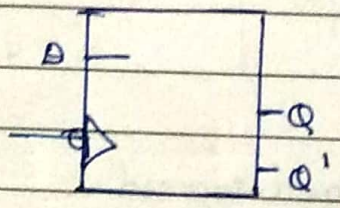
→ $Q(t).\bar{R}$

$Q(t+1) = S + Q(t)R'$ → (1)

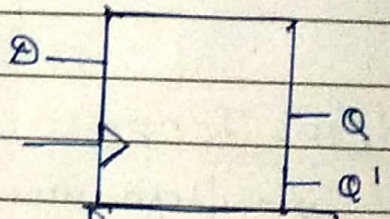
EDGE TRIGGERED D-FLIPFLOP :-

Clk	D	Q	Q'
0/1	x	NC	NC
↑	1	1	0
↑	0	0	1

Negedge



Posedge Dff

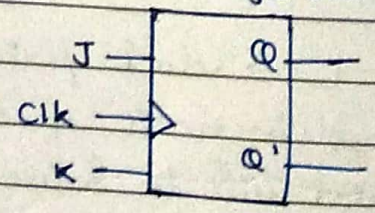


$D = Q(t+1)$

EDGE TRIGGERED JK FLIPFLOP :-

Clk	J	K	Q(t+1)	Q(t+1)'
0/1	x	x	Q(t)	Q(t)'
↑	0	0	Q(t)	Q(t)'
↑	0	1	0	1
↑	1	0	1	0
↑	1	1	Q'(t)	Q(t)

Posedge JKff



Jk / Q(t)	00	01	11	10
0	0	0	1	1
1	1	0	0	1

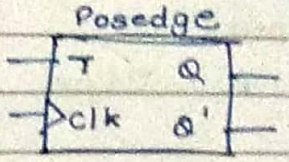
→ $Q'(t)J$

→ $Q(t).K'$

characteristic $= JQ'(t) + K'Q(t)$

DATE

EDGE TRIGGERED TOGGLE FLIP FLOP



Posedge triggered State table

Clk	T	Q(t+1)	Q'(t+1)
0|1	x	Q(t)	Q'(t)
↑	0	Q(t)	Q(t)'
↑	1	Q'(t)	Q(t)

Q(t) \ T	0	1
0	0	(1)
1	(1)	0

C E = Q(t)+1 = T ⊕ Q(t)

CHARACTERSTIC EQUATIONS :~

SR Flipflop = S + Q(t)R'

JK Flipflop = JQ'(t) + K'Q(t)

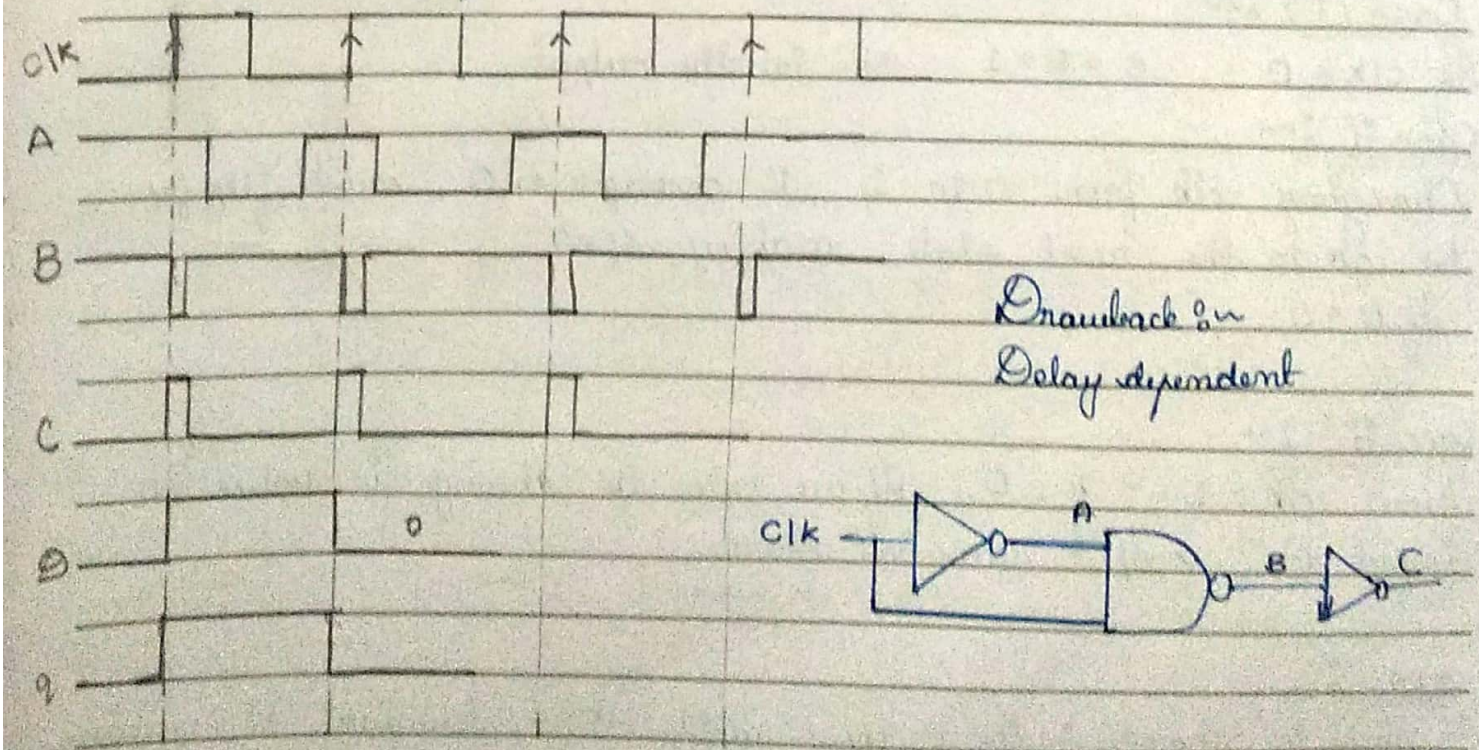
D Flipflop = D

T Flipflop = T ⊕ Q(t)

Imp. 4M

* Unwanted oscillation is avoided by using edge triggered D flipflop
* To get controlled output.

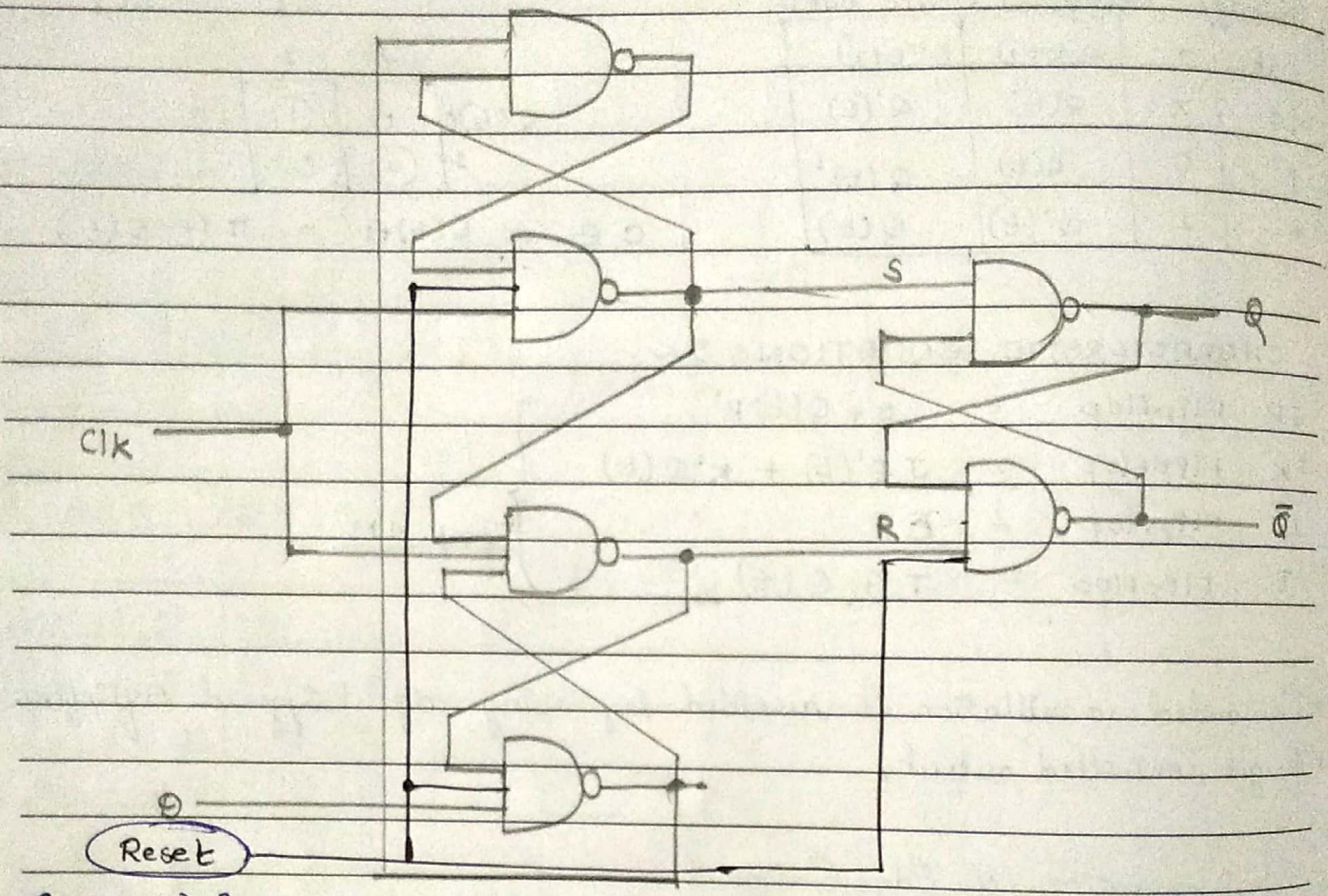
Implementation of Edge Triggered



Drawback :~
Delay dependent

DATE

D- Type positive edge triggered Flip Flop



Case (i) :-

If clk = 0 , S = R = 1 NC in the output.

Case ii :-

Changing clk from 0 to 1 , R changes to 0 , cause flip flop to go to the reset state making Q = 0

If D = 0

Case iii :-

Now clk = 1 , R = 0 , If we tries to change the value in input D , output remains same.

NOTE :

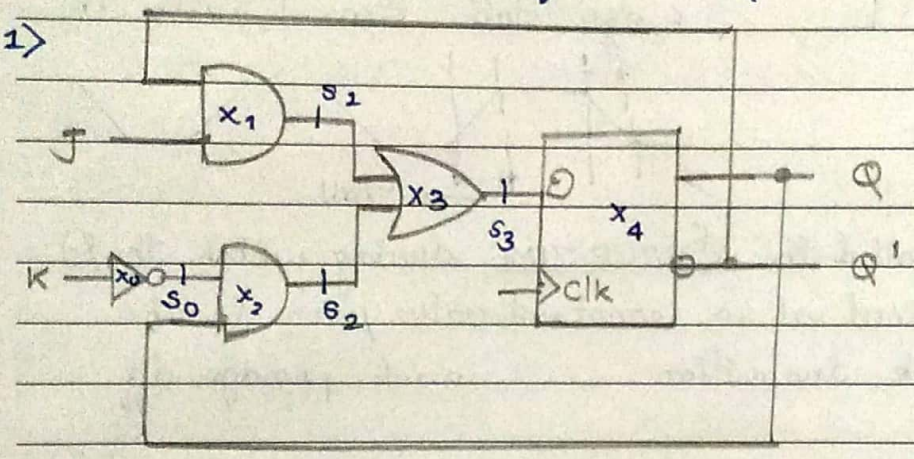
If there is change in the D input while clk = 1, terminal R remains at 0. Thus, the flip-flop is locked out & is unresponsive to further changes in the input.

DATE

* When the clock returns to 0, R goes to 1, placing the output latch in the quiescent condition without changing the output.
* Similarly, if D=1 when clk goes from 0 to 1, S changes to 0. This causes the circuit to go to the set state, making Q = 1. Any change in D while clk = 1 does not affect the output.

Implement all 4 types of FF using the D-flipflop (using CE).

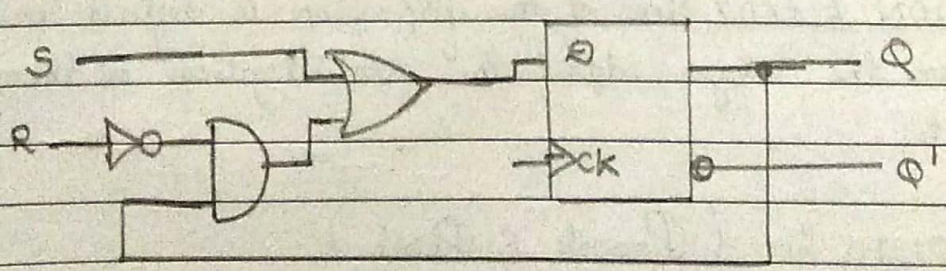
1)



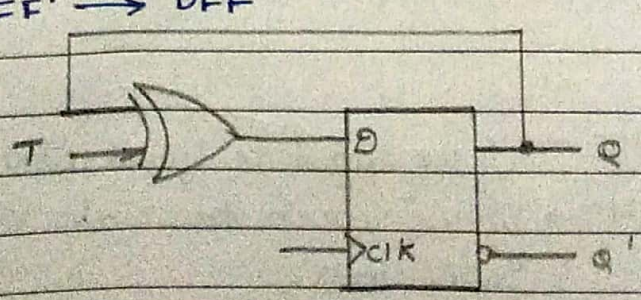
JKFF using DFF

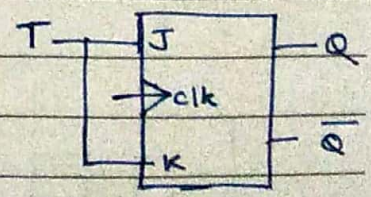
2)

SRFF using DFF



3) TFF → DFF





T → JKFF

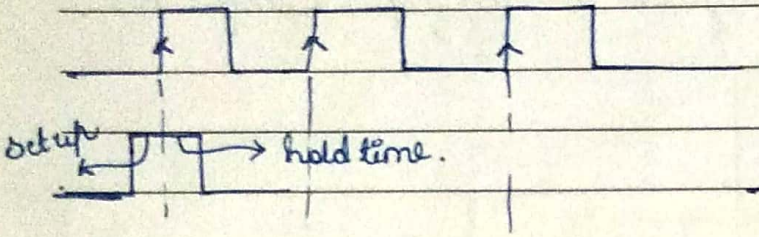
PES UNIVERSITY

DATE

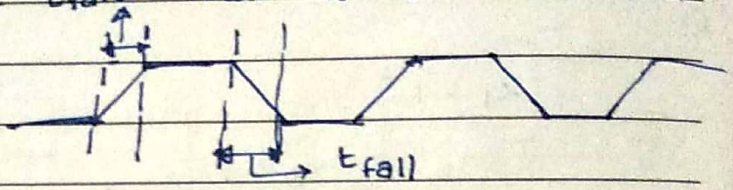
18th Oct'19.

1) Setup Time } associated with Input } 1 MCBTQ
2) Hold Time }

1) * before & after arrival of clk
before arrival of clk the input should be stable shld not oscillate.



Raise time } associated
Fall time } with clk.



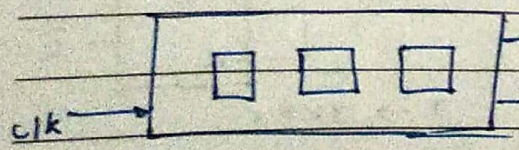
* There is a min time called the SETUP TIME during which the D input must be maintained at a constant value prior to the occurance of the clock transition w.r.t posedge dff.

* There is a minimum time called the HOLD TIME during which the D input must not change after the application of the positive transition of the clock.

* The PROPAGATION DELAY time of the flip-flop is defined as the interval between the trigger edge & the stabilization of the output to a new state.

DIRECT INPUTS :- (Preset & Reset)

* Each flipflop can store 1 bit.



preset → if we want to final state
reset → if we want to make it to initial state

D FLIP FLOP WITH ASYNCHRONOUS RESET & PRESET

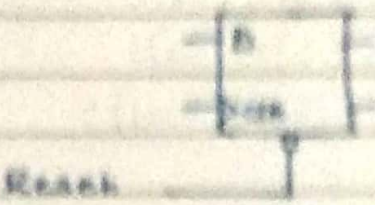
→ It is necessary to initialize flipflop to a known state before the start operation starts

→ preset (sets Q = 1) & clear (sets Q = 0)

→ These are usually implemented by using additional inputs

D FLIPFLOP WITH ASYN RESET Is applied input to the 2nd gate of all the latches

(a) Graphic symbol

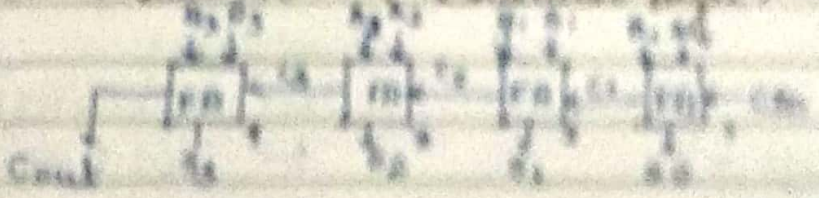


(b) Function Table

R	CLK	D	Q	Q'
1	x	x	0	1
0	↑	0	0	1
0	↑	1	1	0

* Active low Reset

4 - BIT RIPPLE CARRY ADDER (Structural style)



Full Adder :-

```
module fulladder (A, B, Cin, Sum, Cout);
input A, B, Cin;
output Sum, Cout;
assign sum = A ^ B ^ Cin;
assign Cout = (A & B) | (B & Cin) | (Cin & A);
endmodule
```

DATE

FLIPFLOP WITH ASYNCHRONOUS PRESET & CLEAR

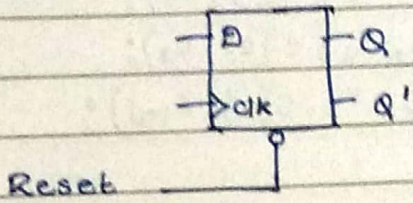
It is necessary to initialize flipflop to a known state before the ckt operation starts

→ preset (sets Q = 1) & Clear (sets Q = 0).

→ these are asyn ops, & can be easily implemented by using add^n lgs.

① FLIPFLOP WITH ASYN RESET :- applied reset to the 2nd gate of all the latches

(a) Graphic Symbol

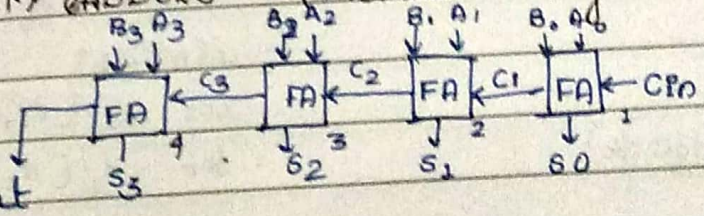


(b) Function Table

R	clk	D	Q	$\bar{Q}$
1	x	x	0	1
0	↑	0	0	1
0	↑	1	1	0

* Active low Reset

4-BIT RIPPLE CARRY ADDERS (Structural style)



Full Adder :-

```
module fulladder (A, B, Cin, Sum, Cout);
input A, B, Cin;
output sum, Cout;
assign sum = A ^ B ^ Cin;
assign Cout = (A & B) | (B & Cin) | (A & Cin);
endmodule
```

DATE

RIPPLE CARRY ADDER

Simulate verilog code of fulladder. (Bottom Up approach)

```
module RCA (A, B, Cin, S, Cout);
input [3:0]A, B;
input Cin;
output [3:0]S;
output Cout;
wire C1, C2, C3;
fulladder FA1 ( A[0], B[0], Cin, S[0], C1);
fulladder FA2( A[1], B[1], C1, S[1], C2);
fulladder FA3(A[2], B[2], C2, S[2], C3);
fulladder FA4 (A[3], B[3], C3, S[3], Cout);
endmodule
```

Q: Write a verilog code to construct a T-flipflop using D ff & necessary gates

D Flip flop. v

```
module dff (D, clk, Q);
input D, clk;
output Q;
reg Q;
always @ (posedge clk)
begin
    Q <= D;
end
endmodule
```

Also by MIXED STYLE

```
module Tff (T, clk, Q);
input T, clk;
output Q;
wire S1;
xor x1 (S1, T, q);
dff x2 (S1, clk, q);
endmodule
```

Tff.v

```
module Tff (T, clk, Q);
input T, clk;
Output Q;
wire S1;
assign S1 = T ^ Q;
dff x1 (S1, ck, Q);
endmodule.
```

DATE

Q: Write a verilog code for a JK flipflop using D flipflop & appropriate gate

JKff.v

```
module Jkff (J, K, clk, Q, Qb);
input  J, K, clk;
output q, Qb; wire s0, s1, s2, s3;
 and  x1 (s1, J, Qb);
 not  x0 (s0, K);
 and  x2 (s2, q, s0);
 or   x3 (s3, s1, s2);
 dff  x4 (s3, clk, q);
endmodule
```

J	K	Q
0	0	Q
0	1	0
1	0	1
1	1	$\bar{q}$

S	R	Q
0	0	q
0	1	0
1	0	1
1	1	?

DFF

D	Q
0	0
1	1

TFF

T	Q
0	1
1	Q

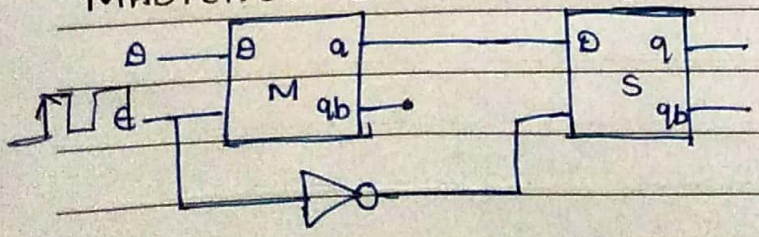
MASTER-SLAVE CONFIGURATION:

Race around: race between stable ips & unstable ops which gives unpredictable ops.

In order to overcome race around condition in the flip flops we use Master Slave Configuration.

* unstable outputs are fed back as ips since q, qb are stable & other unstable which after feedback gives unpredictable outputs

MASTER SLAVE D-LATCH:~



REGISTERS:

Registers are group of flipflops each of which are connected by a common clock. and is capable of storing 1 bit of information.

A n bit register consist of n - flipflop.

↳ shift reg's
storage reg's } application level