MUX & De-MUX

Multiplexers

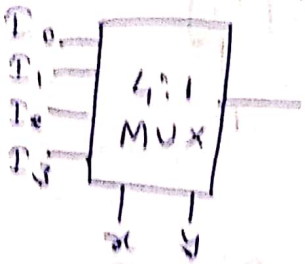
↓

2:1, 4:1, 8:1, 16:1

select lines

n = select lines

2^n = inputs



x = 0	y = 0	$f = I_0$
x = 0	y = 1	$f = I_1$
x = 1	y = 0	$f = I_2$
x = 1	y = 1	$f = I_3$

$$f = \bar{x}\bar{y}I_0 + \bar{x}yI_1 + x\bar{y}I_2 + xyI_3$$

So $x = 0, f = I_0$

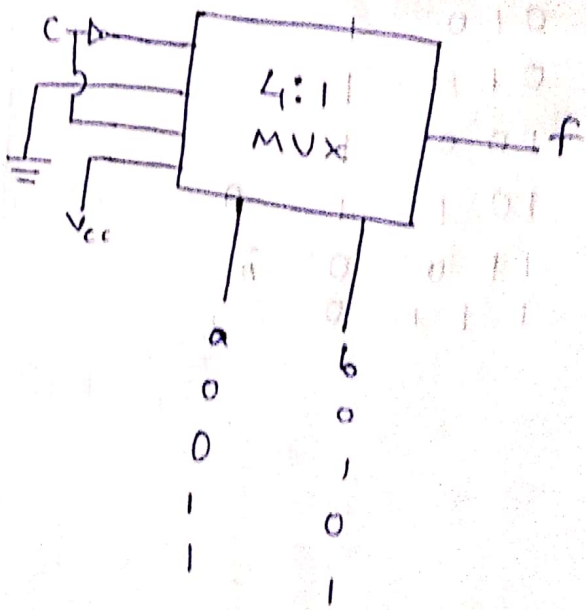
$x = 1, f = I_1$

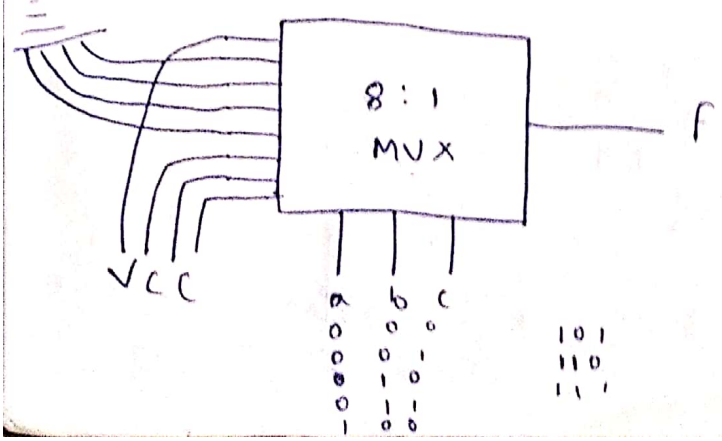
$$f = \bar{x}I_0 + xI_1$$

→ Implement $f(a,b,c) = \Sigma(0,5,6,7)$ using 4:1 MUX

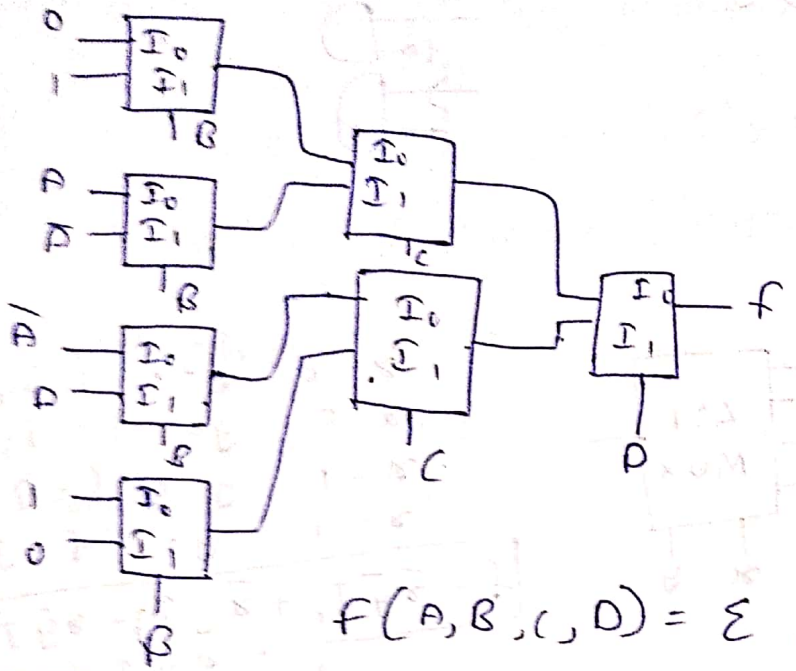
A)

a	b	c	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	1	1
1	1	0	1





Scanned with CamScanner



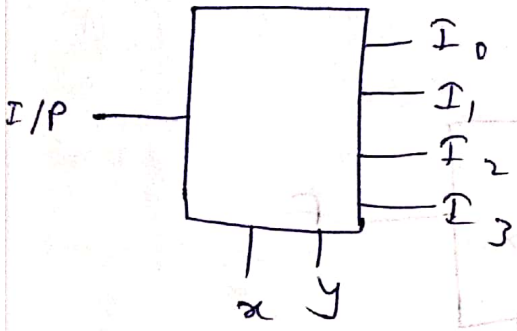
$f(A,B,C,D) = \Sigma(\text{- - - -})$

$= \Sigma(1, 3, 4, 6, 8, 10, 11, 13, 2)$

A	B	C	D	f	
0	0	0	0	0	
0	0	0	1	1	
0	0	1	0	0	
0	0	1	1	1	
0	1	0	0	1	
0	1	0	1	0	$-B$
0	1	1	0	1	$-\bar{D}$
0	1	1	1	0	
1	0	0	0	1	
1	0	0	1	0	
1	0	1	0	1	
1	0	1	1	1	
1	1	0	0	1	
1	1	0	1	1	$-A$
1	1	1	0	0	$-\bar{A}$
1	1	1	1	0	

Scanned with CamScanner

De-MUX -



I/P	x	y	I_0	I_1	I_2	I_3
0/1	0	0	0/1	x	x	x
	0	1	x	0/1	x	x
	1	0	x	x	0/1	x
	1	1	x	x	x	0/1

Priority Encoder -

D_0	D_1	D_2	D_3	x	y
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

(high priority) (low priority)

D_0	D_1	D_2	D_3	x	y
1	0	0	0	0	0
x	1	0	0	0	1
x	x	1	0	1	0
x	x	x	1	1	1

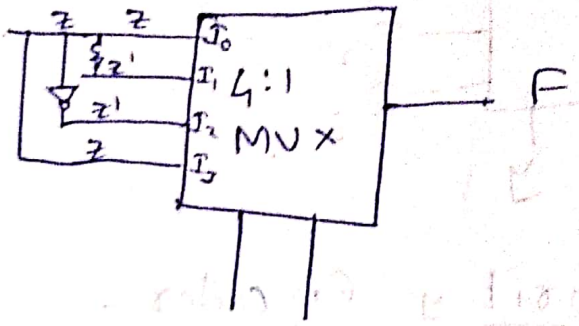
D_0	D_1	D_2	D_3	x	y
0	0	0	0	x	x
0	0	0	1	1	1
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	1	0	1
1	1	1	0	1	0
1	1	1	1	1	1

→ One Enable line must be present. to Activate block in Multiplexer.

Scanned with CamScanner

$F(x,y,z) = \Sigma(m(1,2,4,7))$

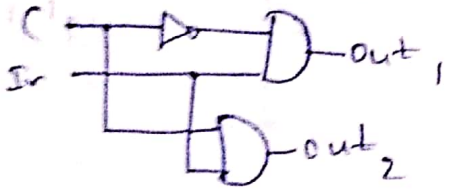
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



x	y
0	0
0	1
1	0
1	1

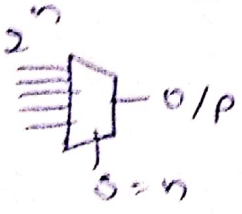
1-to-2

In	C	O_1	O_2
0	0	0	0
1	0	1	0
0	1	0	0
1	1	0	1



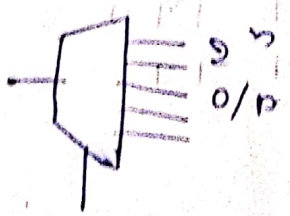
MUX

→ Schematics (2×1, 4×1, 8×1)
→ Truth Tables
→ Canonical forms
→ Internal circuit diagram (2 level AND-OR)
→ Symbol

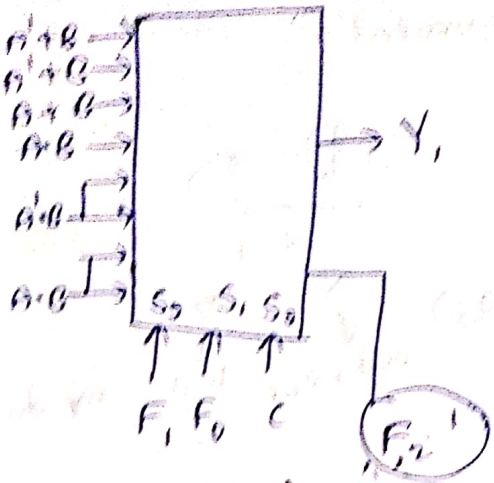


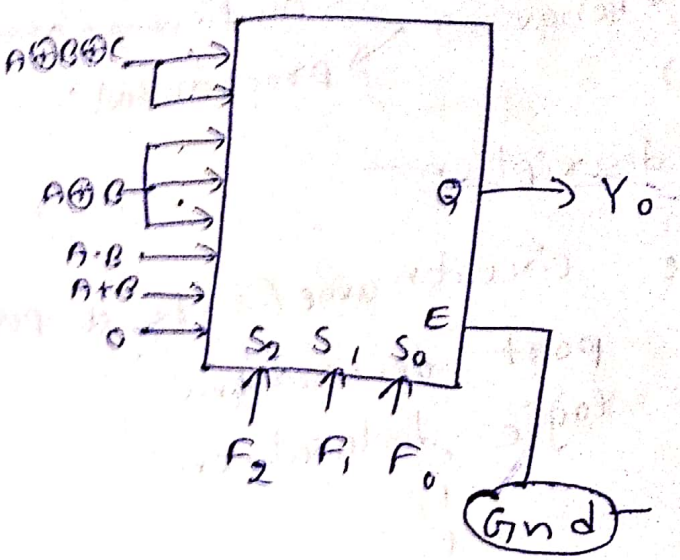
DeMUX

→ Schematics (1×2, 1×4, 1×8, 1×16)
→ Truth tables
→ Canonical forms
→ Internal circuit diagram (one level only AND)
→ Symbol



Scanned with CamScanner





F_1	F_0	C	Y_1
0	0	0	A⊕B
0	0	1	A·B
0	1	0	A'·B
0	1	1	A'·B
1	0	0	A·B
1	0	1	A+B
1	1	0	A'+B
1	1	1	A'+B

C=0 , C_{out} = A·B

C=1 , C_{out} = A+B

A	B	C	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

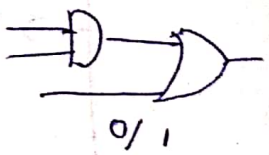
F_2	F_1	F_0	Y_0	Y
0	0	0	0	–
0	0	1	A+B	–
0	1	0	A·B	–
0	1	1	A⊕B	–
1	0	0	A⊕B	C
1	0	1	A⊕B	B
1	1	0	A⊕B⊕C	C
1	1	1	A⊕B⊕C	B

Scanned with CamScanner

Verilog -

- structural (smaller)
- Behaviour → Continuous - assignment
- Behaviour → procedural

Code description -



X ↓ unknown. Z ↓ high imped

```
module  circuitname (list of ports)
        port declarations
        logic declaration
        .
        .
        .
End module.
```

Data types

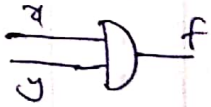
- nets
  - wire
  - tri
  - Supply 0
  - Supply 1
  - wor
  - wand
- variables
  - reg
  - time
  - integer

And gate -

order is not imp.

```
module  ckt:name(x, y, f);
        input  x, y;
        output f;
        add    (f, x, y);
```

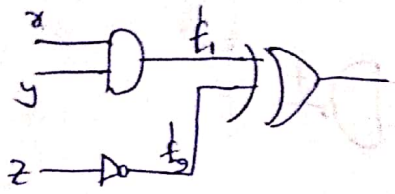
↳ out should be 1st.




```
end module
```

↳ No semicolon.

It is case sensitive inputs - Error



optional


```
Module Rockstars(x, y, z, f);
    input  x, y, z;
    output ~~f1, f2~~, f;
    wire   t1, t2;
    and    (t1, x, y);
    not    (t2, z);
    xor    (f, t1, t2);
end module
```

and G1(t1, x, y);

In default every variable have one bit we can increase no of bits by;

```
    int  [0:3] x, y, z;
         (or)
         [3:0]
```

x = 4'b1010, 8'b10101111

32'b0

32'b1

↳ bits.

0 —— 0.

1 —— 1.

Verilog operators

&& - logical and.

|| - logical or.

! - logical Not.

(A = 1) && (B = 0).

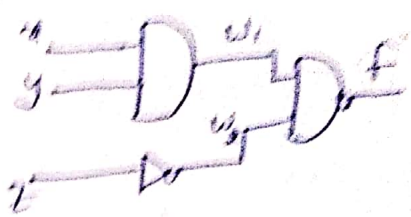
To declare x

assign x = 4'b1011;

Scanned with CamScanner

Behavioural

```
input (x, y, z)
output f;
wire w1, w2;
assign w1 = x && y;
assign w2 = !z;
assign f = !(w1 && w2);
end module.
```



A = 4'b0000 B = 4'b1010. (comments)

!A // logic 1.

!B // logic 0.

A && B // logic 0.

A > B // logic 0.

A < B // logic 1.

A >= B // logic 0

A <= B // logic 1.

Bitwise xor ^.

~A // 1111

A & B // 0000

Reduction operators

& A // 0 0 && 0 && 0 && 0.

& B // 0 1 && 0 && 1 && 0.

~& B // 1

concatination

{A, B} // 00001010.

{3 {A}} 00000000.

A = 3 B = 0

!A // logic 0

!B // logic 1

A && B // logic 0

x = 0 z = 1

case equality & case inequality

A = 4'b 11xz B = 4'b 00

C = 4'b 11xz D = 4'b 10

A === D // 0

A == D // unknown

A != D // x

A !== D // 1.

A === C // logic 1

A == C // unknown x.

Arithemetic right shift.

B >>> 1 // 1101

B <<< 1 // 0100

Scanned with CamScanner

{A, 4'b101, B}

0000101 1010.

{B[1], A}.

10000.

conditional operators - ?:

f = condition ? A : B;

if cond. true
A;

if cond. false
B;

Binary Multiplication

```
  101
  011
 -----
  101
 1010
00000
-------
 01111
```

A = 4'b 1111

A >>> 1 // 1111

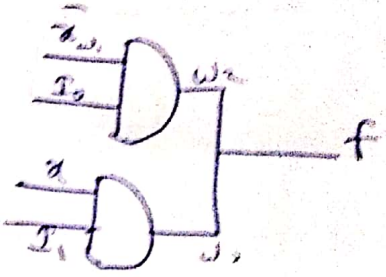
A = 4'b 0114, 1010

A >>> 1 // 0011

Verilog for 2:1 MUX -

$f = \bar{s} I_0 + s I_1$.

```
module MUX-2-1 (I0, I1, s, f);
Input s, I0, I1;
Output f;
wire w1, w2, w3;
not (w3, s);
And (w1, w3, I0);
and (w2, w3, I1);
or (f, w1, w2);
end module
```

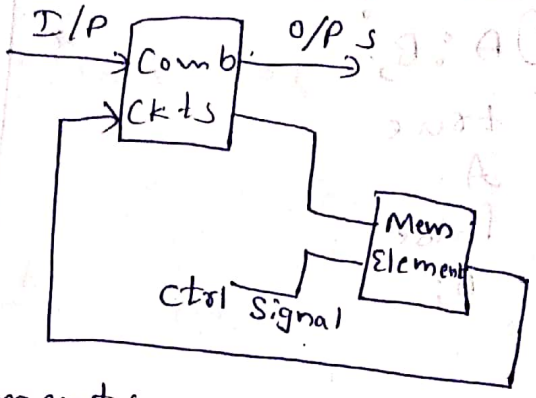


Scanned with CamScanner

Sequential Circuits –

Until Now we learnt about combinational Circuits.

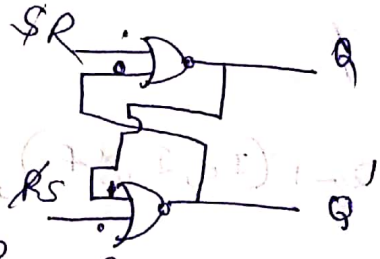
Sequential circuits



Memory Elements

Latch — S-R, D, J-K

flip-flop. — 1 bit inf.

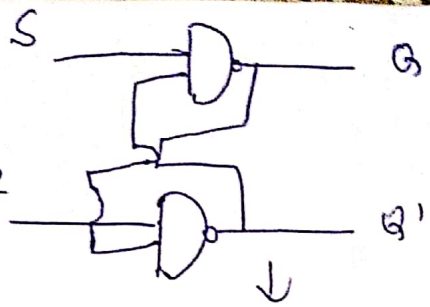


R	S	Q	$\bar{Q}$
0	0	0	1
0	1	0	1
0	0	1	0
1	0	1	0

S	R	Q	$\bar{Q}$
1	0	1	0
0	0	1	0
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
0	1	0	1
0	0	0	1

Memory State

Scanned with CamScanner



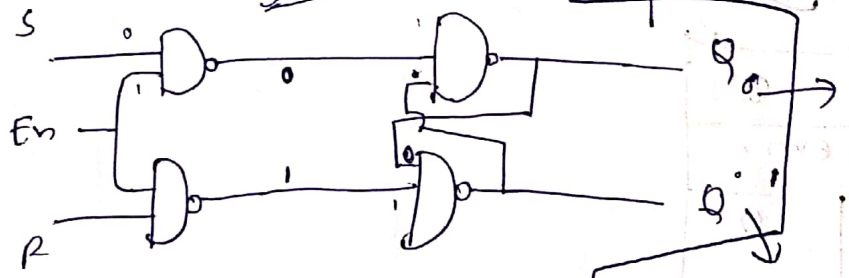
S	R	Q	Q'
0	1	1	0
1	1	1	0
0	1	1	0
1	1	1	0
1	0	0	1
1	1	0	1
0	0		

Mem.

N.A

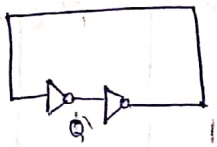
En	S	R	Q	Q'
En=0	X	X	Mem	
En=1	0	0	Mem	
	1	0	1	0
	0	0	1	0
	0	1	0	1
	1	1	Not App	

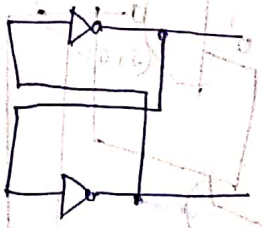
S-R- latch-



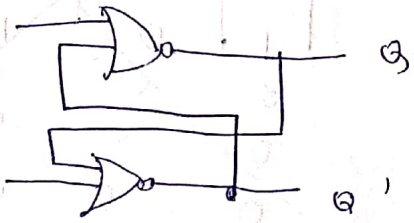
Seq. circuit — Input + state or Memory.

The basic idea of storage.



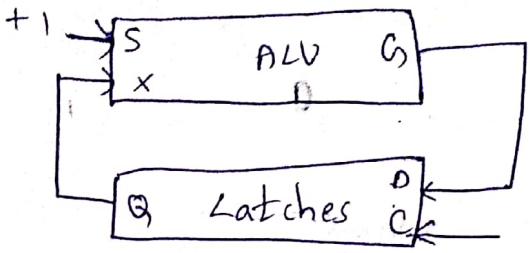


SR latch



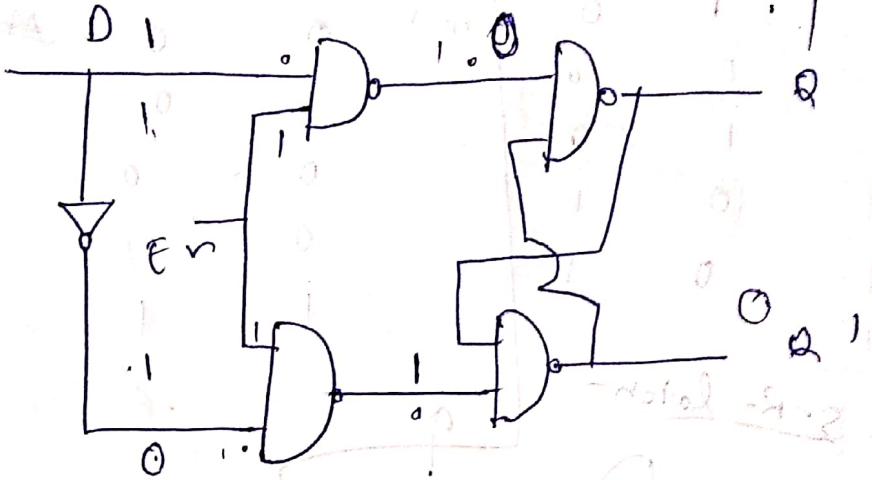
$$Q = (R + Q')_{current}$$

Scanned with CamScanner



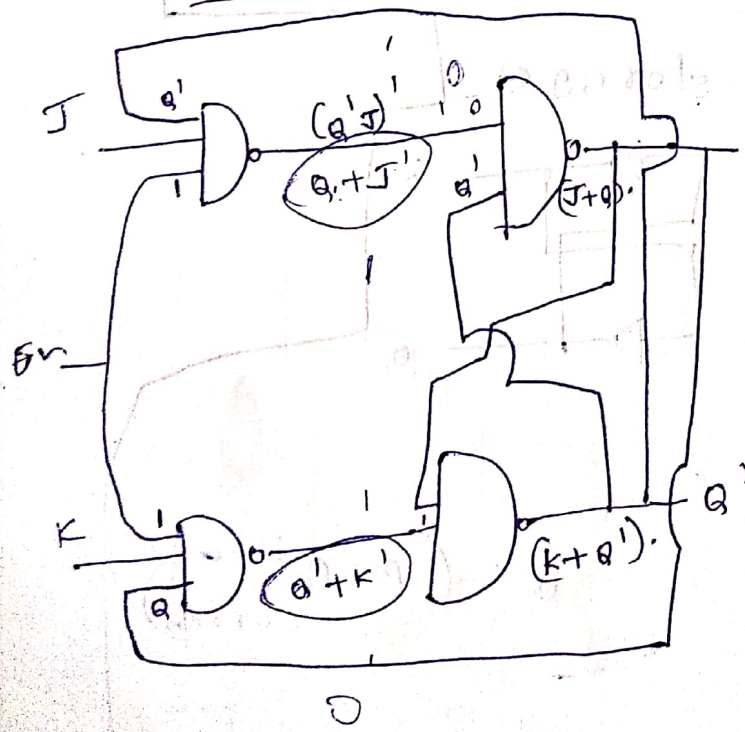
always @ (*) begin

D-latch -



D	Q	Q'
En=0 X	Mem.	
En=1 1	1	0
0	0	1

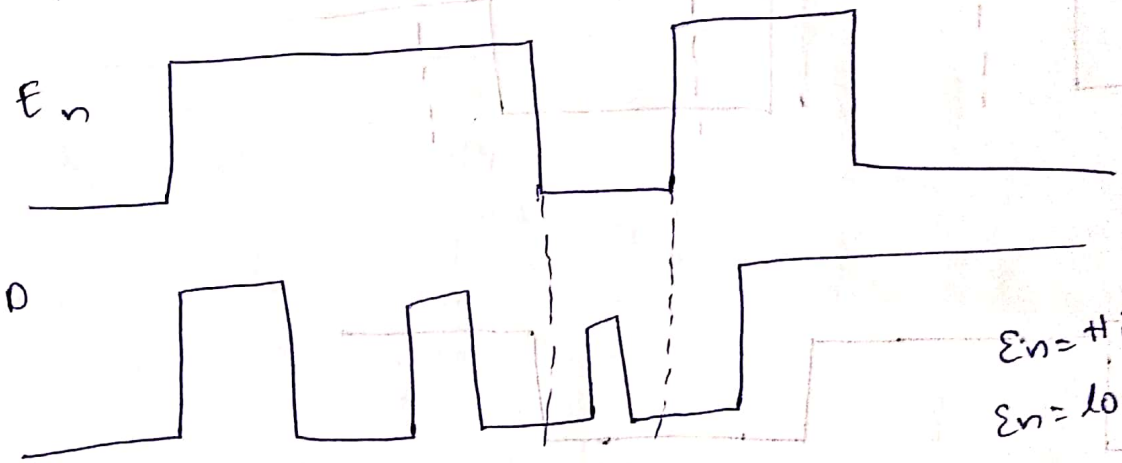
J-K latch -



J	K	Q	Q'
En=0 X	X	Mem	
En=1 1	1	1	0
1	0	1	0
0	0	Mem	
1	0	1	0
0	0	1	1
0	1	0	1
0	0	0	1

Scanned with CamScanner

	J	K	Q_n	$\bar{Q}$
En = 0	x	x	Mem	
En = 1				
$Q_p = 0$	0	1	0	1
$Q_p = 0$	0	0	0	1
$Q_p = 0$	0	1	0	1
$Q_p = 0$	0	0	0	1
$Q_p = 0$	1	0	1	0
$Q_p = 1$	0	0	1	0
$Q_p = 1$	1	1	0	1
$Q_p = 0$	1	1	1	0
$Q_p = 1$	1	1	0	1



En = High Q = D

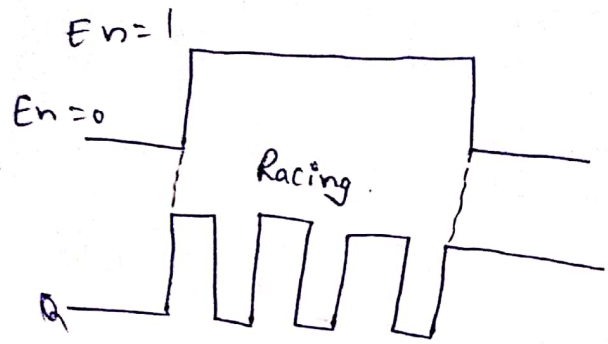
En = low Q = prev. state

$\bar{Q}$ For J = K = 1.

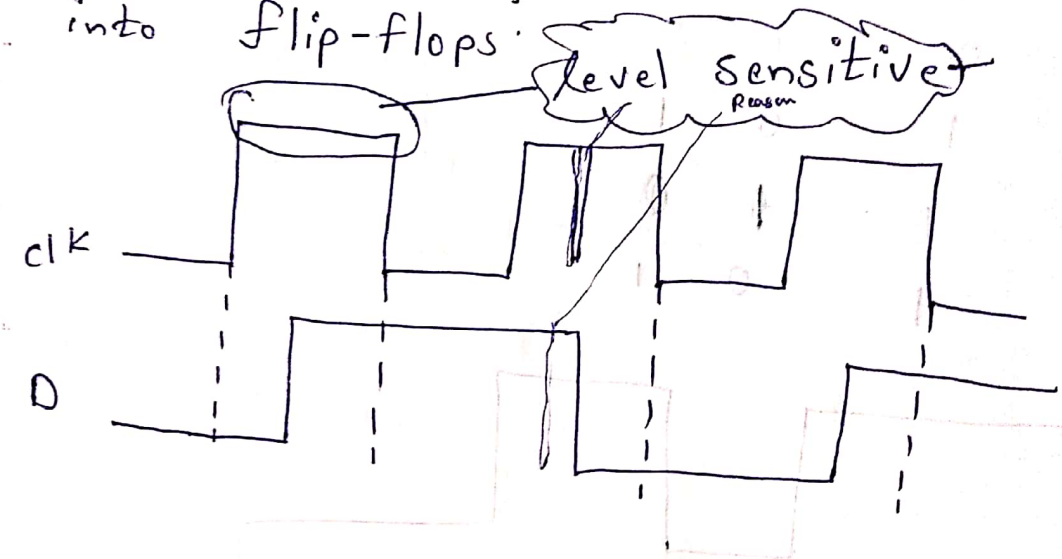
$\bar{Q}$

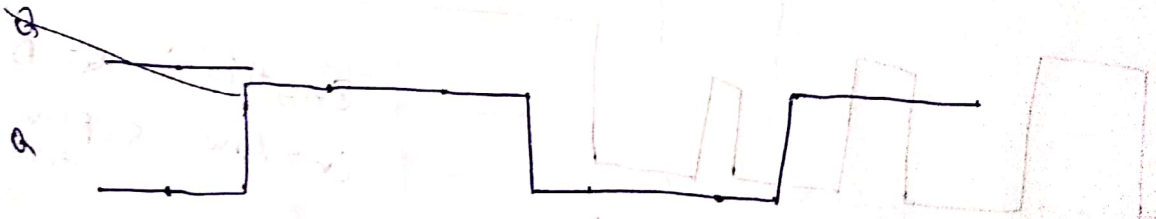
Scanned with CamScanner

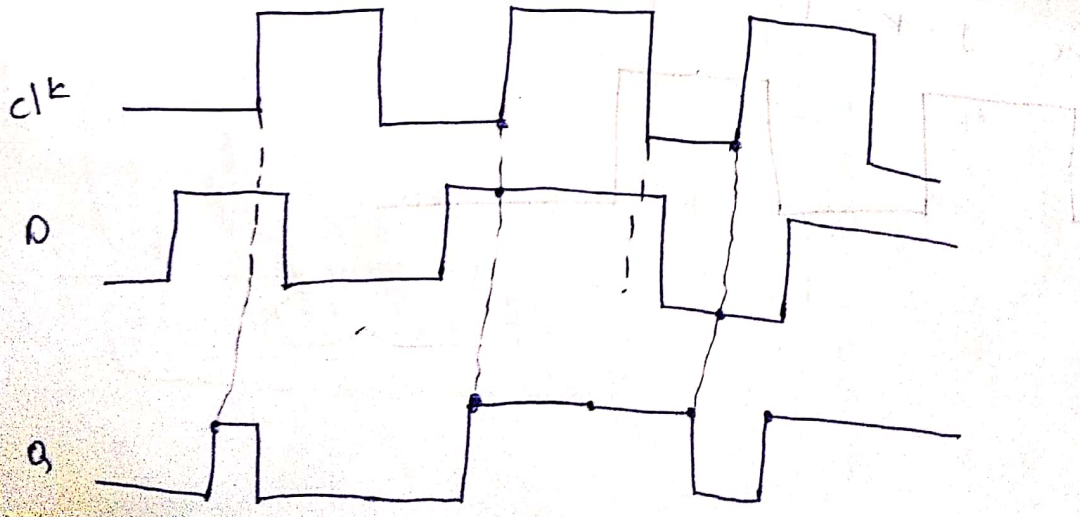
For J = k = 1



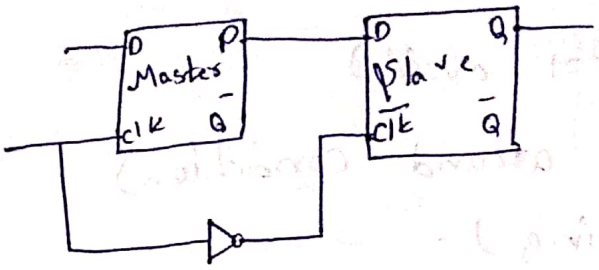
→ If we have clock instead of Enable in latches (D-latch, J-k latch) then it is mod into flip-flops.







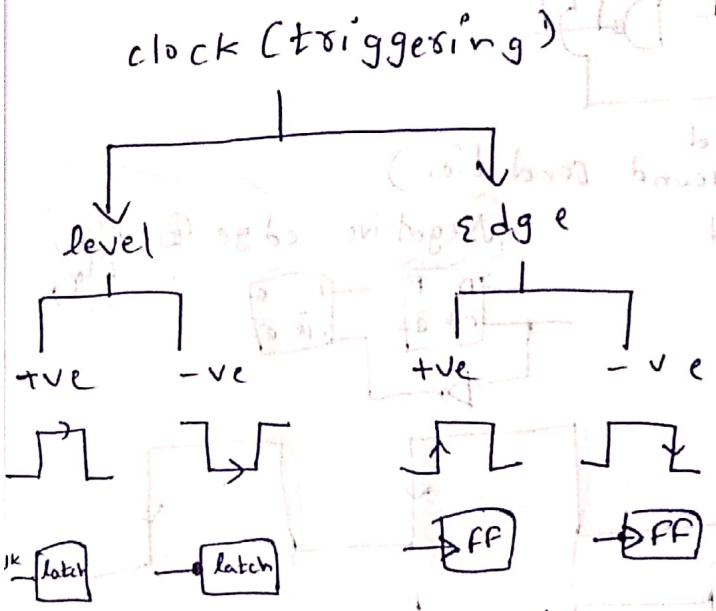
Scanned with CamScanner



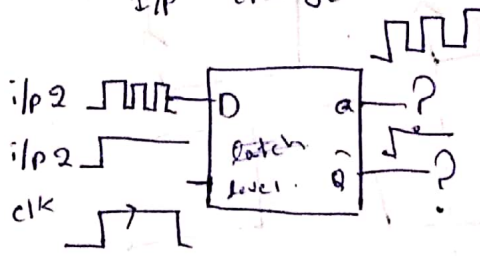
Latch
En
level sensitive

flip-flop
clock.
level & edge sensitive → (+ve). (-ve).
↓
Master Slave.

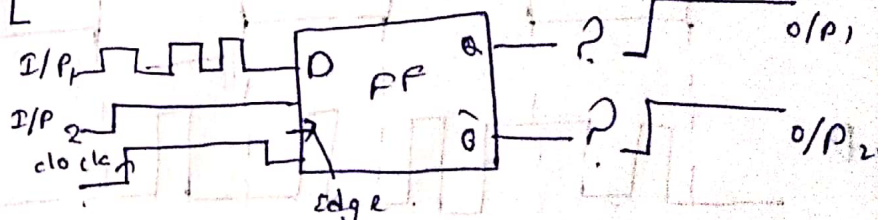
Basics of Sequential Circuits -



I/p change → output changes.



I/p changes, o/p changes max. once in a clock cycle



Race around condition

Diff. b/w toggling & Race around condition.
(1 → 0 → 1)

J	K	Q
0	0	hold
0	1	0
1	0	1
1	1	Toggle

Scanned with CamScanner

Latch

↓

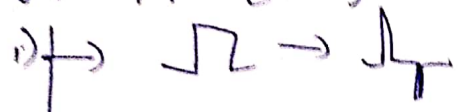
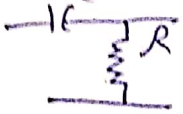
SR latch (drawback S=1, R=1 avoid)

↓

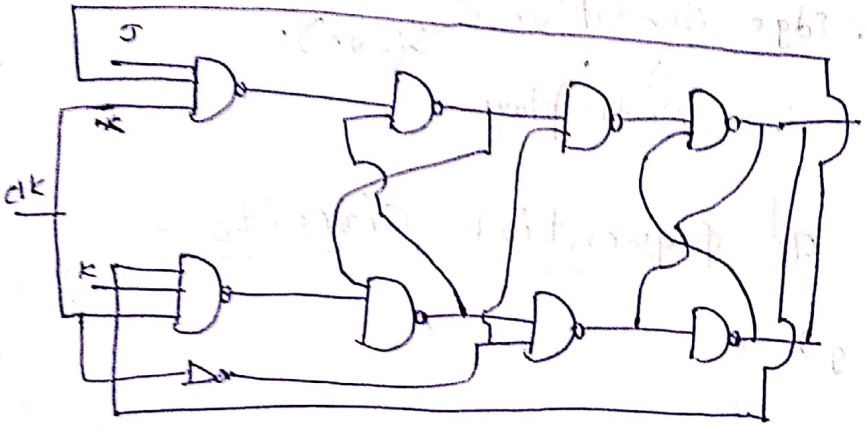
JK latch (J=1, K=1, Race around condition)

↓

JK FF (J=1, K=1, toggling).

1. →  → 

2. → Master-slave F.F

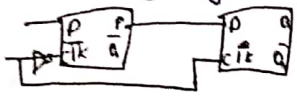


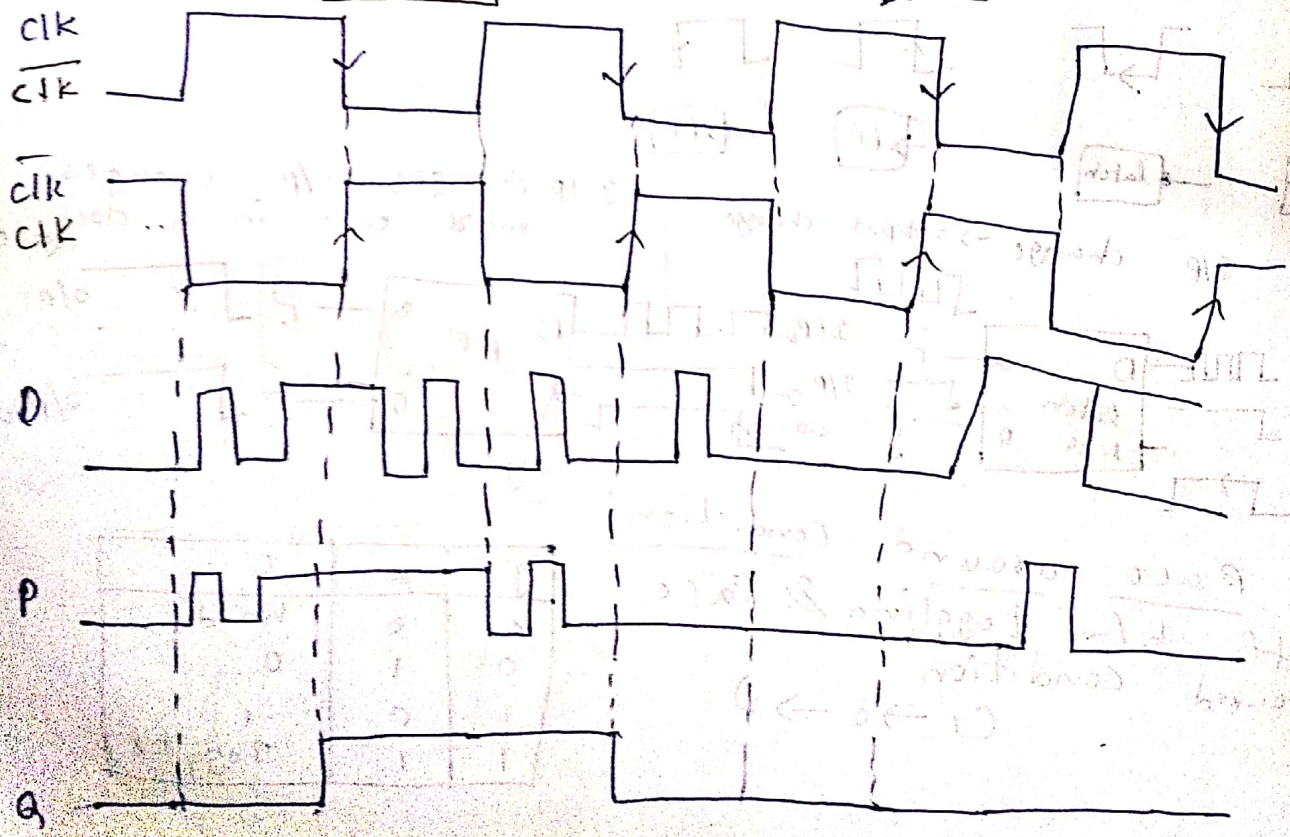
Main O/P

- clk 1 : Master activated (Race around condition).

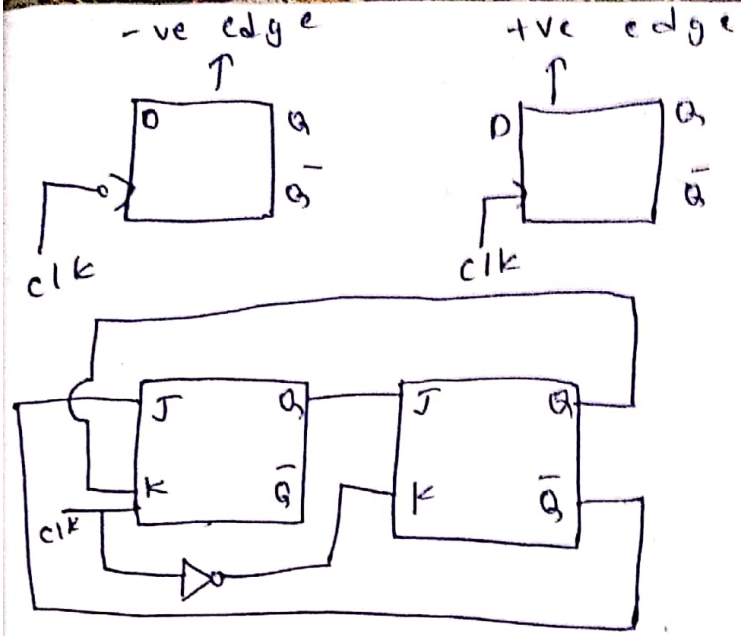
clk 0 : Slave activated (+ve edge)(Toggle).

Negative edge ⏚ - flip flop





Scanned with CamScanner



Scanned with CamScanner