SR. NO.	DATE	TITLE	PAGE NO.	TEACHER'S SIGN

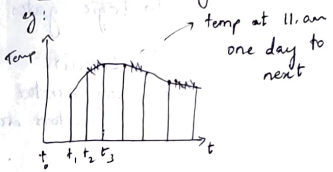
DIGITAL ELECTRONICS

* **Signal:** A function of some quantity w.r.t time

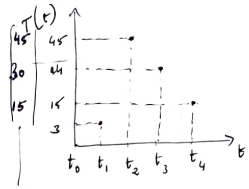
* **Analog:**
  - continuous x & y
  - Can know values in between

* **Discrete time signal:** time is discretised

eg:



* **Digital Signal:**
  - Both time & magnitude discretised
  - We divide magnitude axis into fixed number of levels & signal can take value equal to these levels only



Suppose

$T(t_1) = 9^\circ C$
$T(t_2) = 38^\circ C$
$T(t_3) = 29^\circ C$
$T(t_4) = 15^\circ C$
$T(t_5) = 45^\circ C$

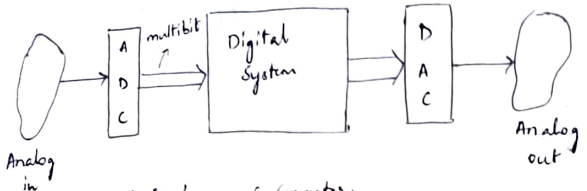
D.T.S

$T(t_1) = 0^\circ C$
$T(t_2) = 30^\circ C$
$T(t_3) = 15^\circ C$
$T(t_4) = 15^\circ C$
$T(t_5) = 45^\circ C$

D.S

* Digital signal is used in communicat^n process to minimize the effect of noise.

Transducer : Converts non-electrical signal to electrical.



(Digital system) ↳ (Sub systems) ↳ (Modules) ↳ Basic units (logic gates)
↳ Circuits (transister, resistor, capacitors etc

Advantages of Digital System:

i) Noise immunity
ii) Uses less bandwidth
iii) Encryption
iv) Efficiency is higher for long distance transmission.

Priority : NOT → AND → OR

Redundancy Theorem (Consensus theorem)

- 3 variables
- Each variable is repeated twice
- One variable is complemented
- Take the complemented variable

$Y = AB + A'C + BC$

$Y = A.B + A'.C$

Proof: $Y = A.B + A'.C + BC.1$

$= AB + A'C + BC[A + A']$

$= A.B + A'C + ABC + A'BC$

$= AB[1 + C] + A'C[1 + B]$

$= A.B + A'.C$

SOP Pos

$0 - A$
$1 - \bar{A}$
Pos

$0 - \bar{A}$
$1 - A$
SOP

For table,

A	B	C	Y
0	0	0	0 ✓
0	0	1	0 ✓
0	1	0	1
0	1	1	0 ✓
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Pos form ⇒ $Y = (A+B+C).(A+B+\bar{C}).(A+\bar{B}+\bar{C})$

If in SOP,

$\bar{Y} = \bar{A}.\bar{B}.\bar{C} + \bar{A}.\bar{B}.C + ABC$

complement on both sides

$(\bar{\bar{Y}}) = [\overline{\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + A\bar{B}\bar{C}}]$

$Y = (\bar{A}.\bar{B}\bar{C})' . (\bar{A}\bar{B}C)' . (\bar{A}BC)'$

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

on simplification, $Y = (A+B).(A+\bar{B}+\bar{C})$

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

$= A + B\bar{C}$ ✱

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

Minimal to Canonical : i) SOP

$Y = A + B'C$

$= A.1.1 + B'C.1$

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

$= (A.B + A.B')(C+C') + AB'C + A'B'C$

$= ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C$

$Y = ABC + ABC' + AB'C + AB'C' + A'B'C$

M to C : 2) POS

$F = (A+B+C')(A'+C)$

$= (A+B+C')(A'+C+0)$

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

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

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

Total no of logical expressions : 2^{2^n}

For $n=1$, A

1	0	A	A'

$= 4$

for $n=2$, (A, B)

0	1	A	A'	AB	$A\bar{B}$	$\bar{A}B$	$\bar{A}\bar{B}$
		B	B'	A+B	A+B'	A'+B	A'+B'
				$\bar{A}B + A\bar{B}$		$AB + \bar{A}\bar{B}$	

$= 16$

+ & -ve logic

+ logic AND Gate = -ve logic OR gate & vice-versa

Dual form : converting from ~~one form to~~ +ve to -ve or -ve to +ve

$A\cdot B \xrightarrow{\text{dual}} A+B$

(+ve)

Self Dual

→ For any logical expression, two times dual gives same expression.

eg: $F = AB\bar{C} + \bar{A}BC + ABC$

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

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

→ For self dual expression, one time dual gives the same expression

eg: $G = A\cdot B + B\cdot C + A\cdot C$

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

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

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

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

$= AB + BC + AC$

** For n variables

No. of self duals possible : $2^{2^{n-1}}$

For $n=2$, self duals $= 2^{2^1} = 4$

i.e A & B

$A \to A$, $\bar{A} \to \bar{A}$

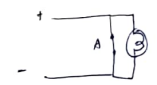
$\bar{B} \to \bar{B}$, $B \to B$

Switching Circuit

Series:

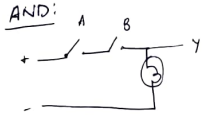
A	Y
0	0
1	1

Parallel: (NOT)

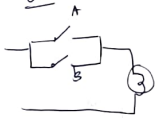


A	Y
0	1
1	0

AND:

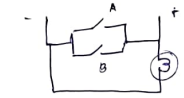


OR:



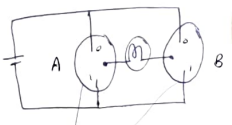
NAND : $\overline{A\cdot B}$

NOR : $\overline{A+B}$



EXOR - odd i/p detector

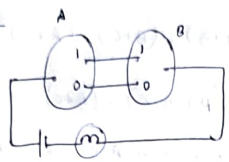
A	B	$Y = A'B + B'A$ $= A \oplus B$
0	0	
0	1	
1	0	
1	1	



Bidirectional switches

EX-NOR :

A	B	Y = A'B + AB = A ⊙ B
0	0	1
0	1	0
1	0	0
1	1	1



r's Complement

complement

- r's complement [Radix complement]
- (r-1)'s complement [Diminished complement]

eg: For decimal Number system

Comp of $(7)_{10}$ = 3

10 - 7 = 3
10 - 9 = 1
10 - 6 = 4

N = 7 (No.)
n = 1 (no. of digits in N)
r = 10 (base of no. system)

$r^n - N$ r's Complement

④ 5690

- $10^4 - 5690$
- 10000 - 5690
- 4310

⑤ N = 1101 find 2's complement

r = 2 n = 4

- $2^4 - 1101$
- $(16)_{10} - 1101$
- 10000 - 1101
- 00011

(r-1)'s Complement

	r's comp	(r-1)'s comp
r = 10	10's	9's
r = 2	2's	1's
r = 16	16's	F's

(r-1)'s comp = $r^n - N - 1$

" = r's comp - 1

(r-1)'s comp + 1 = r's comp.

So we can find (r-1)'s comp then find r's complement by adding 1.

in r's → borrowing involved

(r-1)'s → no borrowing

eg: 7's comp of octal no. 5674

- $8^4 - 5674 - 1$
- $(4096)_{10} - 5674 - 1$ = 10000 (octal) − 5674 − 1
- = 7777 − 5674

  7777
  − 5674
  2103

eg: 1's comp. of 1101

  1111
− 1101
  0010 — 1's
+ 1
  0011 — 2's

(r-1)' comp = $r^4 - N - 1$

= $(r^4 - 1) - N$

if r = 10 ⇒ 9999
if r = 8 ⇒ 7777
if r = 16 ⇒ FFFF
if r = 2 ⇒ 1111

Shortcut : For 1's complement → NOT gate ⇒ 0 → 1, 1 → 0

For 2's : ① Write down given no.
② Starting from LSB, copy all zeroes till first 1.
③ Copy first 1
④ Complement all remaining bits

eg: 10111000
1000
0100
01001000

Data Representation

- Magnitude
  - signed (both +ve & -ve binary no.s)
  - unsigned (only +ve no.s)
- Complement (both +ve & -ve)
  - 1's complement
  - 2's complement

* In all 4 rep, +ve no. are rep. in same way

Signed - magnitude

- Range : $-(2^{n-1}-1)$ to $(2^{n-1}-1)$, n → no. of variables

  For n = 4, $-(2^3-1)$ to 7 ⇒ -7 to 7

1's compliment range

2's complement range -2^{n-1} to $+(2^{n-1}-1)$

condition for overflow :

x & y are the sign bits of two no.s

z is sign bit of result.

$\bar{x}.\bar{y}.z + x.y.\bar{z} = 0$ (no overflow)

$= 1$ (overflow)

BCD Addition

1) Sum ≤ 9, Final Carry = 0 — Answer is correct
2) Sum ≤ 9, Final carry = 1 — Answers is incorrect ⇒ corrected by adding 6 (0110)
3) Sum > 9, Final Carry = 0 — Answer is incorrect ⇒ Add 6 (0110) to correct

eg $(3)_{10} + (7)_{10}$

```
  0 0 1 1
  0 1 1 1
---------
Sum = 1 0 1 0   (10) → +6
```

Sum > 9, F.C = 0

⇒ 1000 0000 (correct BCD of 10)

valid BCD till 9 (1001)

After 9 → invalid or don't care

$2^4 = 16$ can → 0-15 but rep 0-9

15 - 9 = 6

eg ② $(8)_{10} + (9)_{10}$

```
  1 0 0 0
+ 1 0 0 1
---------
 (1) 0 0 0 1
```

Final carry = 1, sum < 9

```
  0 1 1 0   +6
---------
  0 0 0 | 0 1 1 1
```

1 7 ✓

eg $(57)_{10} + (26)_{10}$

```
  0101  0111   (57)
  0010  0110   (26)
---------------
  0111  1101
```

7 > 9 ; 13 > 9

```
  0110
---------
  1000  0011
```

8 3 ✓

SHIFT - ADD-3 Method

to convert Binary to BCD

what we did before

15 → Binary 1111

BCD 0001 0101

shift, add-3

operation	Tens	Ones	Decimal
original no			1111
shift		1	111
shift		11	11
shift		111	1
Add-3		011	
shift		1010	1
	(1)	0101	→ stop
	1	5	

BCD + 3 (0011) gives xs-3 code

eg 24

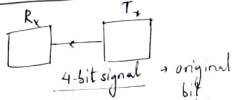
```
  0010  0100
+ 0011  0011
---------
  0101  0111
```

5 7 → xs-3 code for 24

PARITY

→ is a concept to detect errors

→ only single bit error is detected by it.



Two types : i) Even ii) Odd

i) Suppose bit string : 0100 (original) 1 (Parity bit) ⟶ overall even 1's.

ii) Odd parity : 11100 0 ⟶ overall odd no. of 1's

11100 1 ⟶ overall odd no. of 1's

LOGIC GATES

~~OR~~

① AND :

a) Enable & Disable :

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

(disable o/p = 0; enable buffer)

For AND gate, "0" → disable with o/p = 0

"1" → enable / buffer

b) Unused Input :

*** * In Transistor-Transistor logic (TTL), if any i/p is open or floating it will act as "1".
* ECL → "0"

For AND gate : ways to deal with unused i/p :

i) A, B → Y ; "1" (open or floating) we need to enable

ii) Vcc, A, B → Y

iii) A → Y = A.A = A

iv) TTL logic : A, 1 → Y

ECL ✗ because it will disable

② OR :

a) Enable & Disable :

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

(enable buffer; disable)

"0" → Enable / Buffer with o/p

"1" → Disable with o/p = 1

b) Unused Input :

i) ~~FFL~~ connect unused i/p to "0"

A, "0"

ii) A, B → Y = A+B+B = A+B

iii) ECL logic : Simply keep it floating

A → Y = A+0 = A

③ NAND :

A, B → Y = $\overline{A \cdot B}$ = $\bar{A} + \bar{B}$

(Bubbled OR) A, B → Y ≡ A, B → Y

A, B → Y

T.T :

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

(disable o/p = 1; enable inverter)

Associative : ✗

Commutative : ✓

Unused i/p : is treated same as AND Gate

④ NOR :

A, B → Y = $\overline{A + B}$ = $\bar{A} \cdot \bar{B}$

(Bubbled AND) A, B → Y = $\bar{A} \cdot \bar{B}$ ≡ A, B → Y

A, B → Y

T.T :

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

(enable inverter; disable o/p = 0)

Associative : ✗

Commutative : ✓

Unused i/p : same as OR Gate

3) Exclusive - OR Gate : Ex-OR / XOR

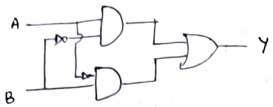
$Y = A \oplus B$

$= A \cdot \bar{B} + \bar{A} \cdot B$

or

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

IEEE Symbol



T.T

	A	B	Y
Enable buffer	0	0	0
	0	1	1
Enable inverter	1	0	1
	1	1	0

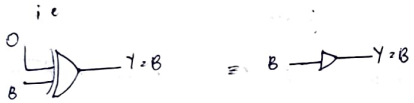
→ ODD I/s detector.

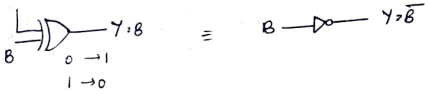
Enable & Disable:

Both 0 & 1 → enable

∴ XOR Gate ⇒ Controlled Inverter

i.e





Properties:

i) $A \oplus A = 0$

$A \oplus \bar{A} = A \cdot \bar{A} + \bar{A} \cdot \bar{A} = A + \bar{A} = 1$

$A \oplus 0 = A \cdot 1 + \bar{A} \cdot 0 = A + 0 = A$

$A \oplus 1 = \bar{A}$

ii) if $A \oplus B = C$ then $B \oplus C = A$, $A \oplus C = B$

$A \oplus B \oplus C = 0$

iii) $A \oplus A = 0$

$A \oplus A \oplus A = 0 \oplus A = A$

For n I/s ⇒ (odd) ⇒ A

even ⇒ 0

4) Exclusive NOR Gate: Ex-NOR / XNOR

$Y = A \odot B$

$= A \cdot B + \bar{A} \cdot \bar{B}$

Symbol:

IEEE Symbol

T.T

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

when A.B o/p = 1

A ≠ B o/p = 0

→ XNOR follows commutative & associative laws.

Enable & disable:

	A	B	Y
Enable inverter	0	0	1
	0	1	0
Enable Buffer	1	0	0
	1	1	1

Properties:

i) $A \odot A = 1$

$A \odot \bar{A} = 0$

$A \odot 0 = \bar{A}$

$A \odot 1 = A$

ii) $A \odot A \odot A = 1 \odot A = A$

$A \odot A \odot A \cdots \odot A$ (n)

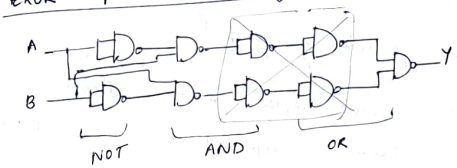
n → odd ⇒ A

even ⇒ 1

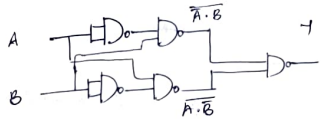
iii) XNOR & XOR are complement only when there are even i/p eg: $A \oplus B \oplus C \oplus D = \overline{A \odot B \odot C \odot D}$

XOR & XNOR → same → odd i/p eg: $A \oplus B \oplus C = A \odot B \odot C$

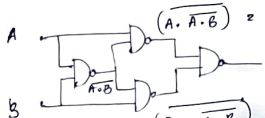
EXOR implementation using NAND



III



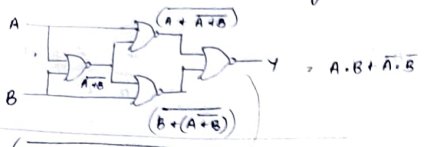
III



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

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

Implementation of XNOR using NOR



$Y = A \cdot B + \bar{A} \cdot \bar{B}$

XOR = XNOR + NOT
↳ 5 NOR gates

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

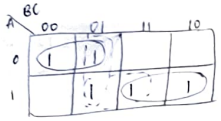
$\overline{(\overline{A \cdot (A+B)})} \cdot \overline{(\overline{B \cdot A+B})}$

$(A + (A+\bar{B})) \cdot (B + (\bar{A}+B))$ • $A \cdot B + \overline{A+B}$ $A \cdot B + \bar{A} \cdot \bar{B}$

K-Map

Note: Result of k-Map ~~may~~ is minimum but may not be same or unique

eg:



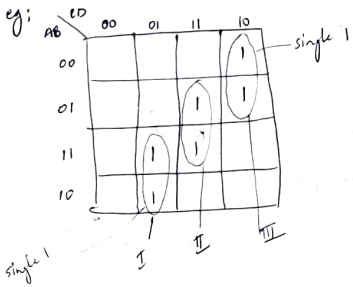
$F = AB + \bar{A}\bar{B} + AC$ | $F = AB + \bar{A}\bar{B} + \bar{B}C$

Implicants: Group of 1's
eg: Group of 1, 2, 4, 8, 16 ... etc

Prime implicant: Largest possible group of 1's

Essential prime implicants: prime implicants + but ~~,~~ atleast ~~Less~~ there is single 1 which cannot be combined in any other way.

eg:

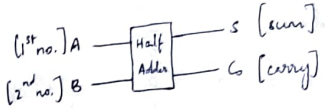


I & II → EPI's.

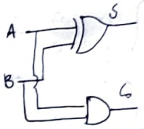
Half Adder [comb ckts]

* Single bit no.'s
* It doesn't take carry from previous scan

A	B	S	C_o
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



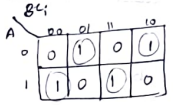
$S = A \oplus B$
$C_o = A \cdot B$



Full Adder

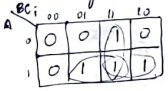
A	B	C_i	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

For Sum



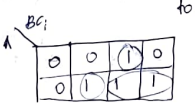
⇒ check board config.
⇒ $S = A \oplus B \oplus C_i$

For Carry out

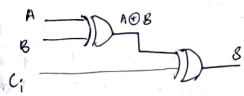


$C_o = BC_i + AB + AC_i$
$C_o = AB + C_i(A \oplus B)$
in most text books.

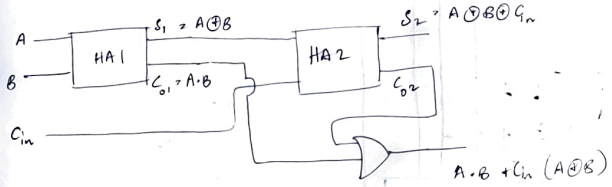
to get this



$C_o = AB + A\bar{B}C_i + \bar{A}BC_i$
$= AB + C_i(A\bar{B} + \bar{A}B) = AB + C_i(A \oplus B)$



Full Adder using Half Adder



CLA

→ predict carry before it calculated — no dependency on carry C_{in}

A	B	C_{in}	C_o	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	A⊕B = 1
1	0	0	0	$\bar{A}+\bar{B}=1$
1	0	1	1	$C_{in}=1$
1	1	0	1	
1	1	1	1	A·B

$C_o = A \cdot B + (A \oplus B) \cdot C_{in}$

carry generator · carry propagator (when C_{in} carry is C_{in}, P is propagated as 1, carry is generated)

$$C_o = G + PC_{in}$$

$$C_n = G_n + P_n C_{i-1}$$

or

$$C_i = G_i + P_i C_{i-1}$$

BCD Adder

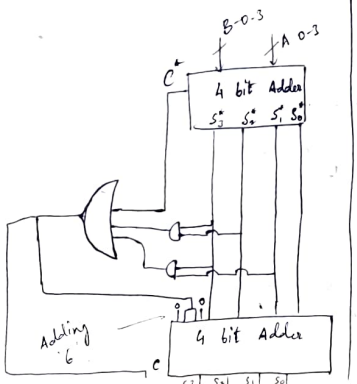
	BINARY	BCD
c		
9	same	same
	$C^* S_3^* S_2^* S_1^* S_0^*$	$C S_3 S_2 S_1 S_0$
10	0 1 0 1 0	1 0 0 0 0
11	0 1 0 1 1	1 0 0 0 1
12	0 1 1 0 0	1 0 0 1 0
13	0 1 1 0 1	1 0 0 1 1
14	0 1 1 1 0	1 0 1 0 0
15	0 1 1 1 1	1 0 1 0 1
16	1 0 0 0 0	1 0 1 1 0
17	1 0 0 0 1	1 0 1 1 1
18	1 0 0 1 0	1 1 0 0 0
19	1 0 0 1 1	1 1 0 0 1

Add 6 "0110" when i) $C^* = 1$

ii) $S_3^* \cdot (S_2^* + S_1^*)$

$= C^* + S_3^* \cdot (S_2^* + S_1^*)$

$= C^* + S_3^* S_2^* + S_3^* S_1^*$



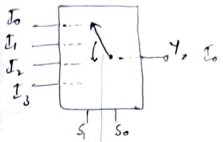
7-segment

b_3	b_2	b_1	b_0		a	b	c	d	e	f	g
0	0	0	0	(0)	1	1	1	1	1	1	0
0	0	0	1	(1)	0	1	1	0	0	0	0
0	0	1	0	(2)							
				(10)	Don't cares						
				(15)							

MULTIPLEXERS

→ Combinational circuit that selects binary information from one of the many i/p lines & directs it to the o/p line

→ It is simply a DATA SELECTOR



Select line or Select variable

Advantages:

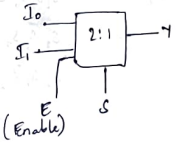
1. Reduces no. of wires
2. Reduces circuit complexity & cost.
3. Implementation of various circuit using MUX.

$n = 2^m$ → select lines

n → no. of input

MUX → Medium scaled Integrated circuits available in the form of ICs.

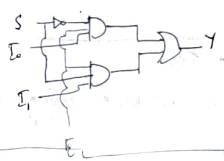
2:1 MUX



E	S	Y
0	x	0
1	0	I_0
1	1	I_1

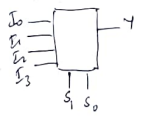
$Y = E \cdot \bar{S} I_0 + E \cdot S \cdot I_1$

$Y = E \cdot (\bar{S} \cdot I_0 + S \cdot I_1)$



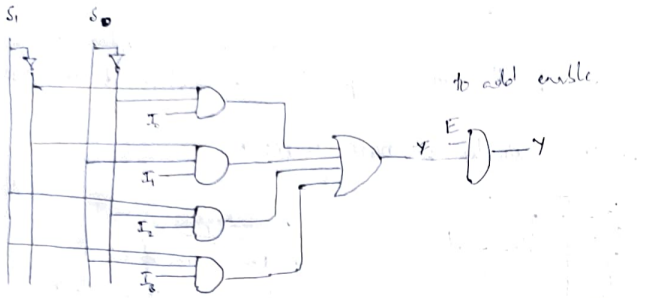
4:1 MUX

(we didn't add enable just to not make it complex. You can add.)

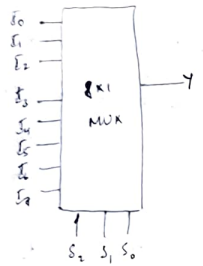


S_0	S_1	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

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



8×1 MUX



S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

$Y = \bar{S}_2\bar{S}_1\bar{S}_0 I_0 + \bar{S}_2\bar{S}_1 S_0 I_1 + \bar{S}_2 S_1 \bar{S}_0 I_2 + \bar{S}_2 S_1 S_0 I_3 + S_2\bar{S}_1\bar{S}_0 I_4 + S_2\bar{S}_1 S_0 I_5 + S_2 S_1 \bar{S}_0 I_6 + S_2 S_1 S_0 I_7$

MUX Tree

Implement 4×1 MUX using 2×1 MUX:

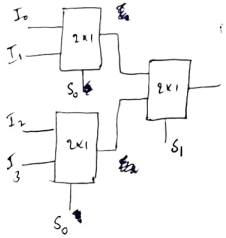
4×1

$\frac{4}{2} = 2$ +

$\frac{2}{2} = 1$

$= (3)$ 2×1 req. for

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



8×1 using 2×1 MUX

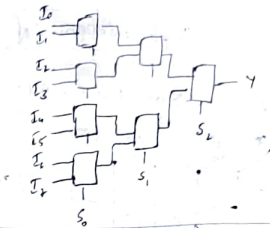
S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

$\frac{8}{2} = 4$ +

$\frac{4}{2} = 2$ +

$\frac{2}{2} = 1$

$\}$ (7) 2×1 MUX req.



8×1 using 4×1 MUX (Special case in MUX tree)

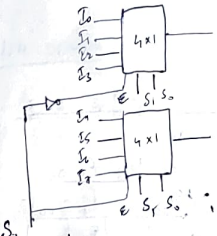
$\frac{8}{4} = 2$ +

$\frac{2}{4} = 0.5$

$\}$ not possible ∴ different approach.

we use ~ enable (varying)

→ All inputs of 3rd 4×1 MUX cannot be used.



3rd select line is implemented using enable

∴ whenever you get a non-whole no. as above, use enable

32×1 MUX using 8×1 MUX

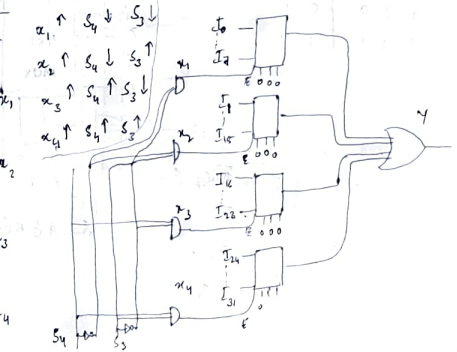
S_4	S_3	S_2	S_1	S_0	Y	
0	0	0	0	0	I_0	x_1
0	0	1	1	1	I_7	
0	1	0	0	0	I_8	x_2
0	1	1	1	1	I_{15}	
1	0	0	0	0	I_{16}	x_3
1	0	1	1	1	I_{23}	
1	1	0	0	0	I_{24}	x_4
1	1	1	1	1	I_{31}	

x_1 ↑ S_4 ↓ S_3 ↓

x_2 ↑ S_4 ↓ S_3 ↑

x_3 ↑ S_4 ↑ S_3 ↓

x_4 ↑ S_4 ↑ S_3 ↑



$x_1 = \bar{S_4} S_3$

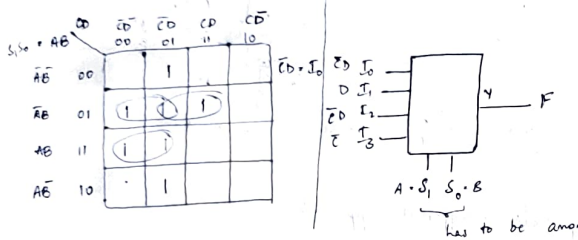
$x_2 = \bar{S_4} S_3$

$x_3 = S_4 \bar{S_3}$

$x_4 = S_4 S_3$

Implementation of Boolean Function using Multiplexer

① $F(A,B,C,D) = \Sigma m(1,4,5,7,9,12,13)$ using 4x1 MUX.



$S_0 = B$

$S_1 = A$

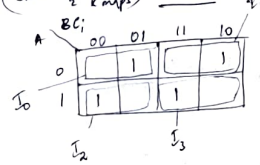
S_1	S_0	Y
0	0	$I_0 = \bar{C}D$
0	1	$I_1 = \bar{C} + D$
1	0	$I_2 = \bar{C}D$
1	1	$I_3 = \bar{C}\bar{D}$

1-Bit Full Adder Using Multiplexer

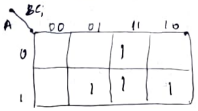
(Since 2 i/p maps)

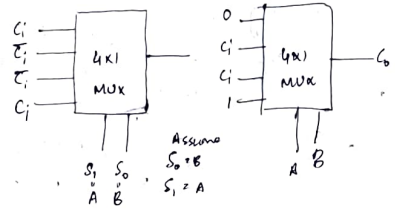
A	B	C_{in}	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

For S



For C_o



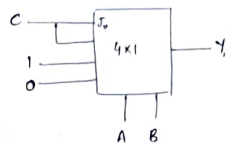


A	B	S
0	0	$I_0 = C_i$
0	1	$I_1 = \bar{C_i}$
1	0	$I_2 = \bar{C_i}$
1	1	$I_3 = C_i$

$S = A \oplus B \oplus C_i$

LOGICAL Expression from MUX

i)

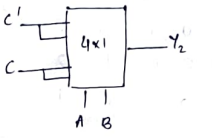


$Y_1 = A'B' \cdot C + A'B \cdot C + A \cdot B' \cdot 1 + A \cdot B \cdot 0$

$Y_1 = A'C[B'+B] + AB'$

$= A'C + AB'$

ii)



A	B	Y_2
0	0	C'
0	1	C'
1	0	C
1	1	C

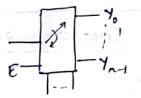
$Y_2 = A'C' + AC$

$= A \odot C$

DEMUX

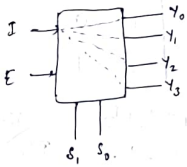
→ One i/p, many o/p

→ Reverse operation of MUX.



1:4 DEMUX

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

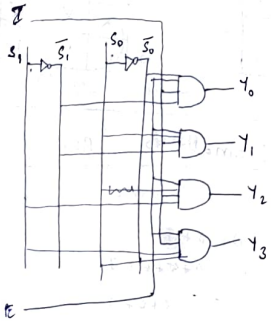


$Y_0 = E\bar{S_1}\bar{S_0} I$

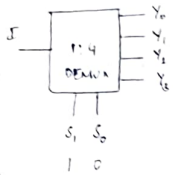
$Y_1 = E\bar{S_1} S_0 I$

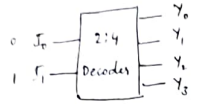
$Y_2 = E S_1 \bar{S_0} I$

$Y_3 = E S_1 S_0 I$



DEMUX as Decoder





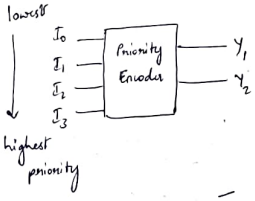
I_1	I_0	Y_0	Y_1	Y_2	Y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

we make i/p I logic 1 permanently i.e we connect +5V

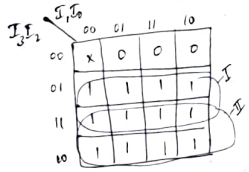
Encoders and Decoders

Priority Encoder

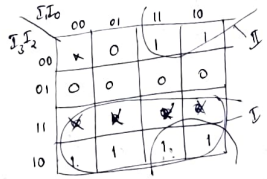
I_3	I_2	I_1	I_0	Y_1	Y_0
0	0	0	0	x	x
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1



$Y_1 = I_2 + I_3$



$Y_0 = I_3 + I_1\bar{I}_2$



(not completed)

Decimal to BCD Encoder

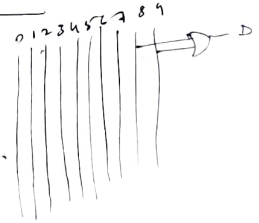
I/p	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4				
5				
6				
7				

$D = 8+9$

$C = 4+5+6+7$

$B = 2+3+6+7$

$A = 1+3+5+7+9$



Hexadecimal to Binary Encoder

H	B_3	B_2	B_1	B_0
0	0	0	0	0
1	0	0	0	1

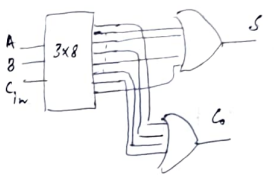
$B_0 = 1+3+5+\ldots+F$

$B_1 = 2+3+\ldots$

Full adder implementation using decoders

A	B	C_{in}	S	C_0	
0	0	0	0	0	m_0
					m_1
					m_2
					⋮
					m_7

(F.A F.T.S)



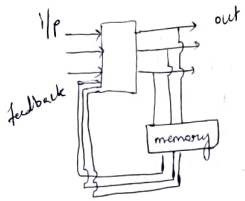
Full Adder using decoder

$F_S = \Sigma(m_1, m_2, m_4, m_7)$

$F_{C_0} = \Sigma(m_3, m_5, m_6, m_7)$

SEQUENTIAL CIRCUITS

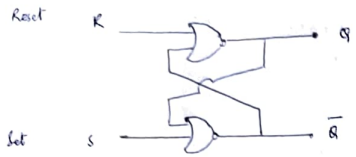
→ The present output depends on the present input as well as past o/p's.



SR Latch

- NOR
- &
- NAND

→ Basic storage element is latch. As the name suggests it latches "0" or "1".



Reset ⇒ Q = 0
set ⇒ Q = 1

TT for NOR

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

case : 1 : S=0, R=1, Q=0, $\bar{Q}$=1

S=0 R=0 Q=0 , $\bar{Q}$=1 memory

case 2: S=1, R=0, Q=1 $\bar{Q}$=0

S=0, R=0, Q=1 $\bar{Q}$=0 memory

∴

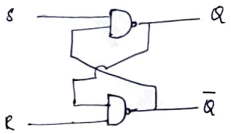
S	R	Q	$\bar{Q}$
0	0	Memory (as before)	
0	1	0	1
1	0	1	0
1	1	Not used	

case : 3 : S=1 R=1 Q=0 $\bar{Q}$=0 ✗
should not be same

∴ not used.

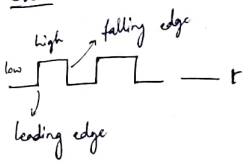
S=0, R=0, Q=0, $\bar{Q}$=1
Q=1, $\bar{Q}$=0 ✗

(Not RS, SR for Nand)



S	R	Q	$\bar{Q}$
0	0	Not used	
0	1	1	0
1	0	0	1
1	1	Memory	

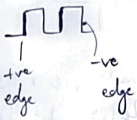
Clock



Duty Cycle : Ratio of time (for which signal ↑) / total time

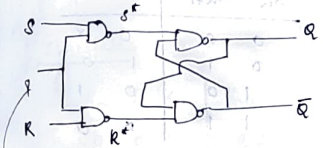
Triggering Methods

① level triggering : whenever clock - high, change in circuit happens.

② 

edge triggering :

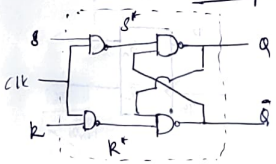
Difference blw Latch & Flip Flop :



Latch has Enable

Flip Flop has clock

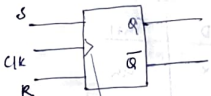
SR Flip Flop



S^*	R^*	Q	$\bar{Q}$
0	0	Not used	
0	1	1	0
1	0	0	1
1	1	Memory	

$S^* = (\overline{S \cdot Clk}) = \bar{S} + \overline{Clk}$

$R^* = (\overline{R \cdot Clk}) = \bar{R} + \overline{Clk}$



⇒ edge triggered

Clk	S	R	Q	$\bar{Q}$
0	x	x	Memory	
1	0	0	Memory	
1	0	1	0	1
1	1	0	1	0
1	1	Ø1	Not used.	

$S^* = 1$

$R^* = 1$

$S^* = \bar{S} = 1$ | $R^* = \bar{R} = 1$

$S^* = 1$

T.T

Clk	S	R	Q_{n+1}
0	x	x	Q_n
1	0	0	Q_n (Memory)
1	0	1	0
1	1	0	1
1	1	1	Invalid

Characteristic Table:

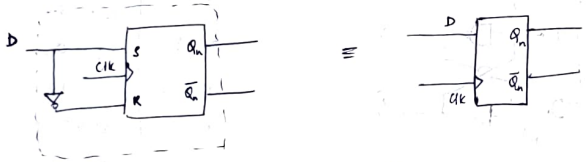
(present state) Q_n	S	R	(next state) Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	x
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	x

Excitation Table:

Q_n	Q_{n+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

D flip-flop (data)

T.T for SR



T.T for D ff

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

Char Table:

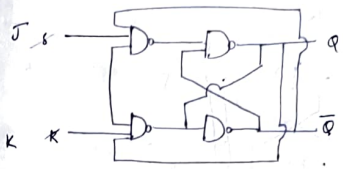
Q_n	D	Q_{n+1}
0	0	0
0	1	1
1	0	0
1	1	1

$Q_{n+1} = D$

Excitation Table:

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

JK flip-flop:



Clk = 0, Memory

Clk = 1, J=1, K=0, Q=1, $\bar{Q}$=0

Clk = 1, J=0, K=1, Q=0, $\bar{Q}$=1

T.T

Clk	J	K	Q_{n+1}
0	x	x	Q_n } memory
1	0	0	Q_n }
1	0	1	0
1	1	0	1
1	1	1	$\bar{Q}_n$ (toggle)

Clk = 1, J = 1, K = 1

assume Q=0 & $\bar{Q}$=1

Q = 0,1,0,1,...

$\bar{Q}$ = 1,0,1,0,...

$Q_{n+1} = \bar{Q}_n$ (Toggling)

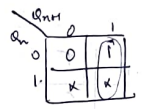
Ch. Table

Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Excitation table

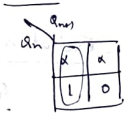
Q_n	Q_{n+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

For J:

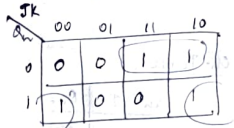


$J = Q_{n+1}$

For k:



$K = \bar{Q}_{n+1}$



$Q_{n+1} = \bar{Q}_n J + Q_n \bar{K}$

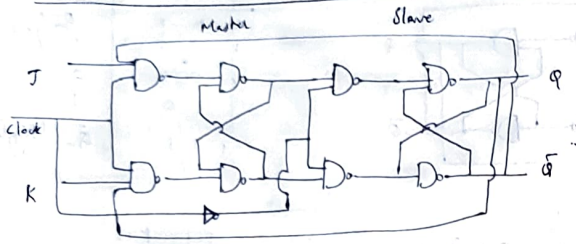
But JK → Racing at 11 (instead of toggling)

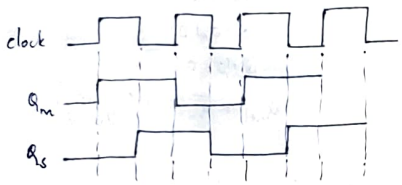
∴ 1) $T_{/2}$ < prop. delay (impractical)

2) Edge triggering

3) Master-Slave operation ⇒ is same as -ve edge triggered flip-flop

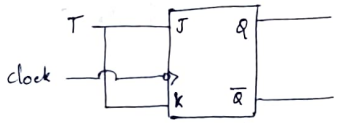
Master - Slave Operation of JK - flip-flop





T - flip-flop

(Toggle)



T-T

Clk	T	Q_{n+1}
0	x	Q_n (memory)
1	0	Q_n (memory)
1	1	$\bar{Q}_n$ (toggle)

Ch. Table

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

Excitation Table

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

$Q_{n+1} = Q_n \oplus T$

(odd 1's detector)

Flip Flop Conversion

eg: JK to D

Steps:
1. Identify available & req. flip flop
2. Make ch. table for req. flip flop.
3. Make excitation table for available flip flop.
4. Write boolean expression for available ff.
5. Draw the circuit.

1) av: JK req: D

2)

Q_n	D	Q_{n+1}	J	K
0	0	0	0	x
0	1	1	1	x
1	0	0	x	1
1	1	1	x	0

Q_n	Q_{n+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

3) For J:

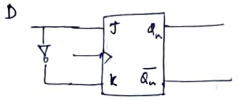
Q_n \ D	0	1
0	0	1
1	x	x

$J = D$

For K:

Q_n \ D	0	1
0	x	x
1	1	0

$K = \bar{D}$



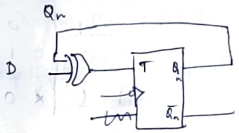
eg2: T to D ff

av. ff = T

req. ff = D

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

$T = D \oplus Q_n$



SR to JK

1) Av : SR
Req : JK

CL table

Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

EX table

Q_n	Q_{n+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

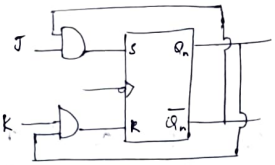
→ For S

Q_n \ JK	00	01	11	10
0	0	0	1	1
1	x	0	0	x

→ For R

Q_n \ JK	00	01	11	10
0	x	x	0	0
1	0	1	1	0

$S = \bar{Q}_n J$, $R = Q_n K$



SR to T

Q_n	T	Q_{n+1}	S	R
0	0	0	0	x
0	1	1	1	0
1	0	1	x	0
1	1	0	0	1

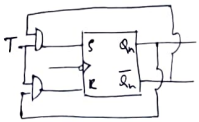
Q_n	Q_{n+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

Q_n \ T	0	1
0	0	1
1	x	0

$S = \bar{Q}_n T$

Q_n \ T	0	1
0	x	0
1	0	1

$R = Q_n T$



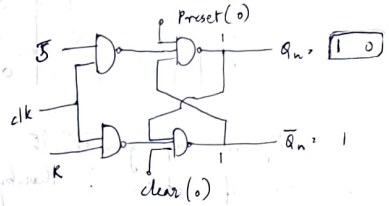
PRESET and clear inputs

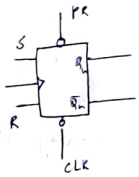
→ They are the direct inputs or overriding i/ps or asynchronous i/p's

→ The synchronous i/p's are S,R, J, K, D & T

preset = 0 ⇒ $Q_n = 1$
clear = 0 ⇒ $Q_n = 0$ (because $\bar{Q}_n = 1$)

whatever be the value of clock & syn. inputs.

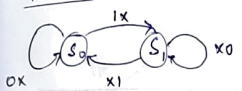




Preset	Clear	Q_n
0	0	Not used
0	1	1
1	0	0
1	1	FF will perform normally

State Diagram :

For JK:



State eqⁿ : LHS = R.HS

Q_{n+1} ↓ P.S & i/p

Present state

Design Procedure for clocked sequential circuits :

Step 1: A state diagram or timing diagram is given which describes the behaviour of the circuit.

Step 2: Obtain the state table

Step 3: The no. of states can be reduced by state reduction method.

Step 4: Do state assignment (if required)

Step 5: Determine no. of flip-flops required & assign letter symbols.

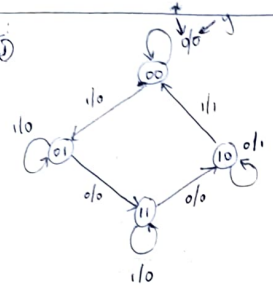
Step 6: Decide the type of flip-flop to be used

Step 7: Derive the circuit excitation table from state table

Step 8: Obtain the expression for circuit output & flip-flop i/p.

Step 9: Implement the circuit.

eg: (1)



State table:

P.S		Next state				output (y)	
		x=0		x=1			
Q_A	Q_B	Q_A^+	Q_B^+	Q_A^+	Q_B^+	x=0	x=1
0	0	0	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	1	0	0	0	1	1
1	1	1	0	1	1	0	0

Mealy: advantages: ① since it uses i/p in o/p generation, it requires less no. of states & thereby less hardware

② o/p generated is 1 clock cycle earlier

disadvantage: i/p transients, glitches etc. are directly conveyed to o/p.

if we want o/p transitions to be synchronised while i/p can change anytime Mealy model is not preferred.

State table: tells relation b/w present state, next state & o/p

2^n states where n → no. of flip flops

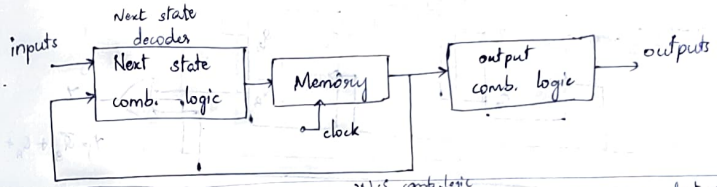
Mealy and Moore State Machines

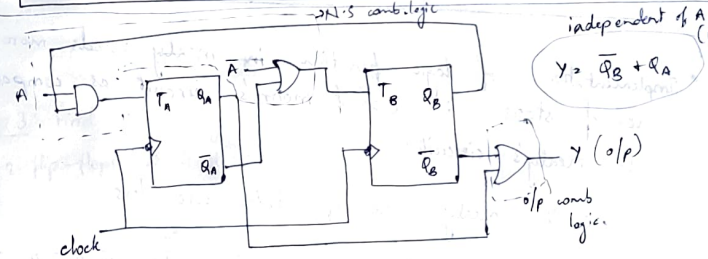
→ There are 2 models developed for representing synch. sequential circuits.

① Moore Ckt / Moore state machine (o/p is funⁿ of present state only)

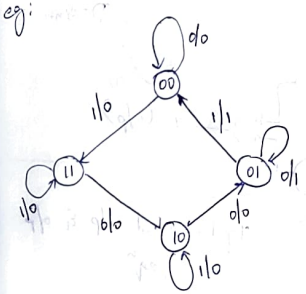
② Mealy Ckt / Mealy state machine (o/p is funⁿ of p.s as well as i/p)

Differ → the way o/p is generated

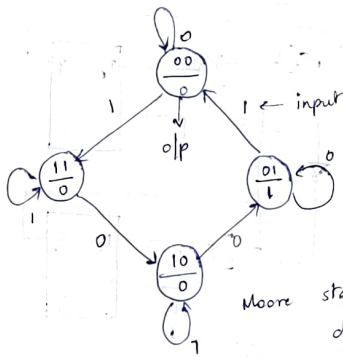




eg:

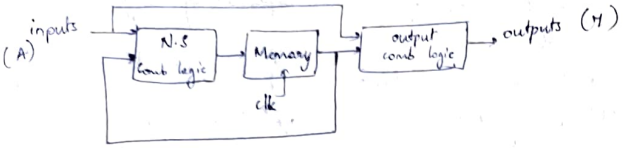


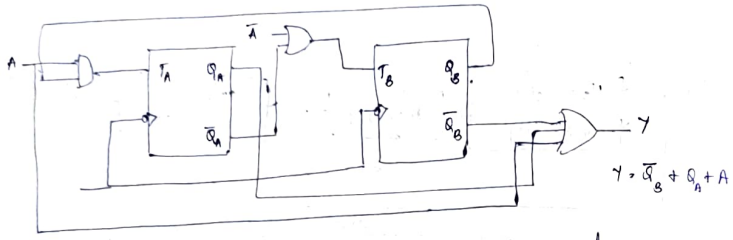
original state diagram



Moore state diagram

Mealy state :



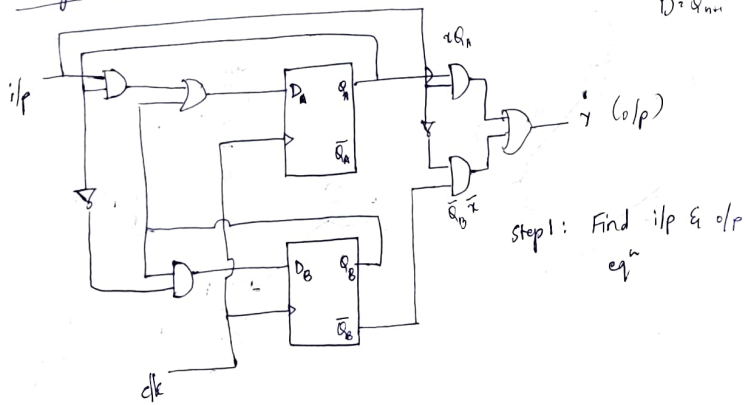


* Implementation of logic function ~~in~~ mealy needs more no. of states in case of moore's circuit as compared to mealy's circuit.

  i.e In mealy ; no. of states are less

Analysis of Clocked Sequential Circuits (with D flip flop)

$D = Q_{n+1}$



Step 1 : Find i/p & o/p eqⁿ

$D_A = Q_A x + Q_B$ $\quad Y = \bar{x}\bar{Q}_B + xQ_A$

$D_B = \bar{Q}_A Q_B$

$D_E = Q_{n+1}$

$Q_A^+ = D_A$

$Q_B^+ = D_B$

Step 2 : Make state table

P.S		i/p	N.S		o/p
Q_A	Q_B	x	Q_A^+	Q_B^+	y
0	0	0	0	0	1
0	0	1	0	0	0
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	1
1	1	0	1	0	0
1	1	1	1	0	1

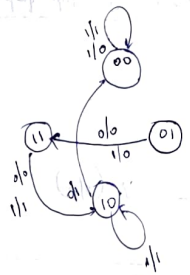
$Q_A^+ = D_A = xQ_A + Q_B$
$= 0.0 + 0 = 0$

$Q_B^+ = D_B = \bar{Q}_A Q_B = 1.0$
$= 0$

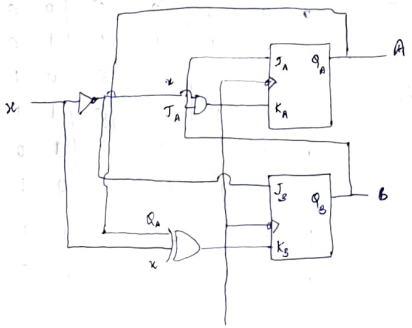
$y = 1.1 + 0.0 = 1$

Step 3 : Find state diagram :

2 flip-flops ⇒ 4 states



Analysis of clocked sequential circuits (with JK flip flop)



Step 1 : i/p eqⁿ

$J_A = Q_B$ $\quad K_A = \bar{x} J_A$

$J_B = \bar{x}$ $\quad K_B = x \oplus Q_A$

$= \bar{x}Q_A + \bar{Q}_A x$

Q_A	Q_B	x	T_A	K_A	J_B	K_B	Q_A^+	Q_B^+
0	0	0	0	0	1	0	0	1
0	0	1	0	0	0	1	0	0
0	1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	1	0
1	0	0	0	0	1	0	1	1
1	0	1	0	0	0	0	1	0
1	1	0	1	1	1	1	0	0
1	1	1	1	0	0	0	0	1

$K_A = \bar{x} J_B$

$K_B = \bar{x} \bar{Q}_A$ (or $\bar{Q}_A x$)

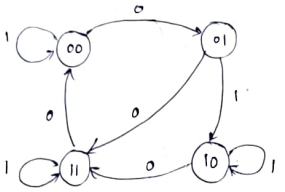
Step 3:

$S_0 = 0\ 0$

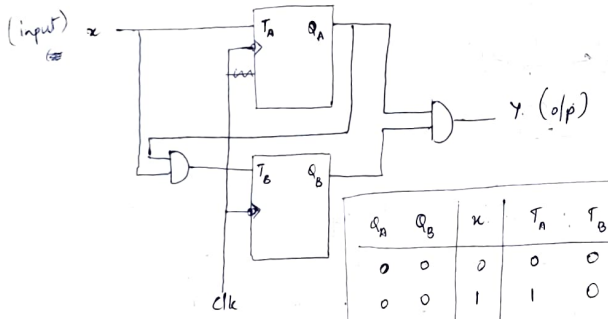
$S_1 = 0\ 1$

$S_2 = 1\ 0$

$S_3 = 1\ 1$



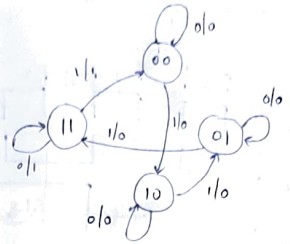
Analysis of Clocked Sequential Circuits (with T flip flop)



Step 1: $T_A = x$ $\quad Y = Q_A Q_B$

$T_B = Q_A x$

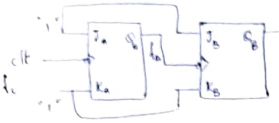
Q_A	Q_B	x	T_A	T_B	Q_A^+	Q_B^+	y
0	0	0	0	0	0	0	0
0	0	1	1	0	1	0	0
0	1	0	0	0	0	1	0
0	1	1	1	0	1	1	0
1	0	0	0	0	1	0	0
1	0	1	1	1	0	1	0
1	1	0	0	0	1	1	1
1	1	1	1	1	0	0	1

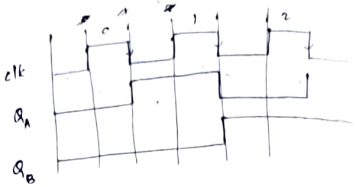


Pattern or Sequence Detector

Counters

IT as mod divide by 2 ckt:-





if n flip-flops, $f_0 = \frac{f_c}{2^n}$

Here, $2^n = 2^2 = 4$ $\Rightarrow 4$

$T_A = 2T_c \Rightarrow \frac{1}{f_A} = \frac{2}{f_c}$

$T_B = 2T_A$ $\quad f_A = \frac{f_c}{2}$

$f_B = \frac{f_A}{2} = \frac{f_c}{4}$

FFs are used to store memy
→ It can also be used for frequency division

Toggling condition is used in counters

Clk	Q_A	Q_B	
0	0	0	(0)
1	0	1	(1)
2	1	0	(2)
3	1	1	(3)

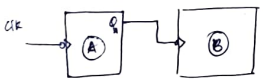
no. of clock pulses passed

Types of Counters :

↳ Asynchronous Counters (Ripple Counters)

↳ Synchronous Counters → (Synchronous) All ff's receive ext. clock signal simultaneously. (pulse)

(Asynchronous)



- ff's are connected in such a way that the o/p of 1st ff drives the clock of next ff
- Circuit is simple for more no. of states
- Speed is slow as clock is propagated through no. of stages.

- There is no connection b/w o/p of first flip flop & clock of next ff.
- Circuit becomes sophisticated as no. of states increases.
- Speed is high as clock is given at a same time.

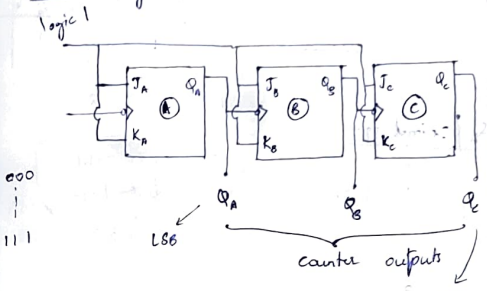
counters (As /s) based on the way counting progresses

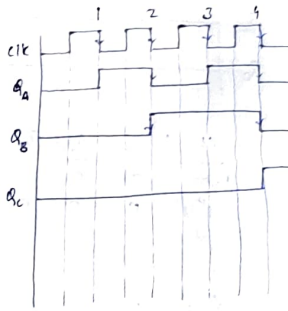
↳ Up Counters 0-1-2-3---

↳ Down counters 7-6-5---

↳ Up/Down counters

3-bit Asynchronous Up Counter





clk	Q_C	Q_B	Q_A	Decimal eq:
Initially	0	0	0	0
1st (↓)	0	0	1	1
2nd (↓)	0	1	0	2
3rd (↓)	0	1	1	3
4th (↓)	1	0	0	4
5th (↓)	1	0	1	5
6th (↓)	1	1	0	6
7th (↓)	1	1	1	7
8th (↓)	0	0	0	0

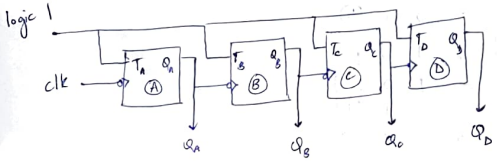
8 states

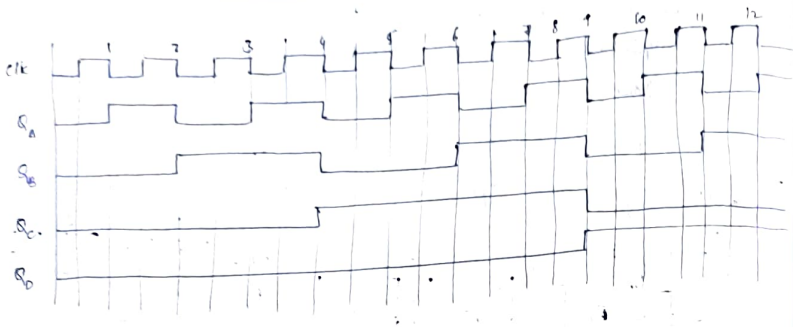
$2^n = 2^3 = 8$

n → no. of flip flops

Maximum count : $2^n - 1 = 7$

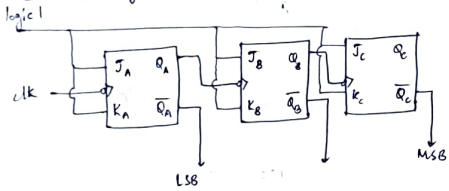
4-bit Asynchronous Up Counter

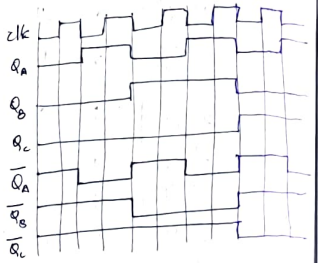




clk	Q_D	Q_C	Q_B	Q_A	Decimal eq.
Initial	0	0	0	0	0
1st (↓)	0	0	0	1	1
2nd	0	0	1	0	2
⋮					
9th (↓)	1	0	0	1	9
⋮					
14th (↓)	1	1	1	0	14
15th (↓)	1	1	1	1	15

3 bit Asynchronous Down Counter

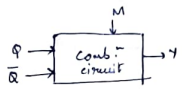




clk	Q_C	Q_B	Q_A	$\overline{Q}_C$	$\overline{Q}_B$	$\overline{Q}_A$	
Initially	0	0	0	1	1	1	(7)
1st (↓)	0	0	1	1	1	0	(6)
2nd (↓)	0	1	0	1	0	1	(5)
3rd (↓)	0	1	1	1	0	0	(4)
4	1	0	0	0	1	1	(3)
5	1	0	1	0	1	0	(2)
6	1	1	0	0	0	1	(1)
7	1	1	1	0	0	0	(0)

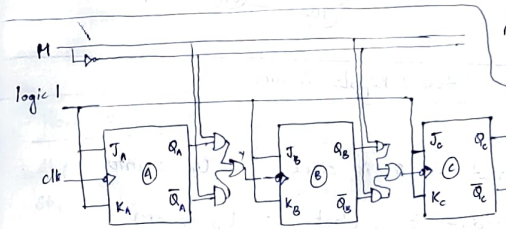
3 bit & 4 bit UP/DOWN Ripple Counters

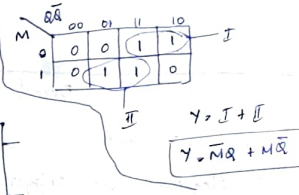
* We have designed up & down counters seperately.
* But in practice both these modes are combined.
* A mode control input (M) is used to select either up or down mode
* A combinational circuit is required between each pair of flip-flops.



M=0 → upcounting ⇒ Q is connected to clk of next ff
M=1 → downcounting
↳ $\overline{Q}$ is connected to clk of next ff

M	Q	$\overline{Q}$	Y	
0	0	0	0	Q
0	0	1	0	
0	1	0	1	
0	1	1	1	
1	0	0	0	$\overline{Q}$
1	0	1	1	
1	1	0	0	
1	1	1	1	





Modulus of the counter and counting up to Particular Value

→ 2-bit ripple counter is called MOD-4 or modulus 4 counter
→ 3-bit ripple counter is called MOD-8 counter.

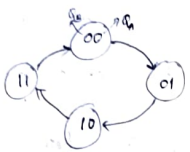
n → no. of bits State = 2^n MOD no. = state

MOD number = 2^n

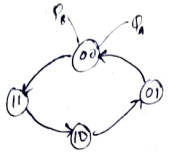
State Diagram of a Counter

2-bit up counter

Q_B	Q_A
0	0
0	1
1	0
1	1



(up counter)

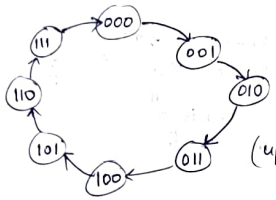


(down counter)

3-bit counter:

No. of states : $2^3 = 8$

Max. count = $2^n - 1 = 7$



(up counter)

Decade (BCD) Ripple Counter

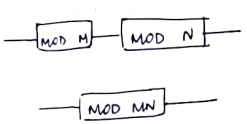
Important points:

① Negative edge triggered → Q is clock → Up counter

Positive edge triggered → $\bar{Q}$ is clock → Up counter

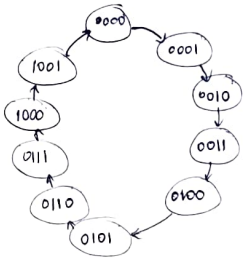
Negative edge triggered → $\bar{Q}$ is clock → Down counter

Positive edge triggered → Q is clock → Down counter

② 

2 flip flops cascaded

Resulting ff has MOD MN



No. of states = 10

Maximum count = 10 - 1 = 9

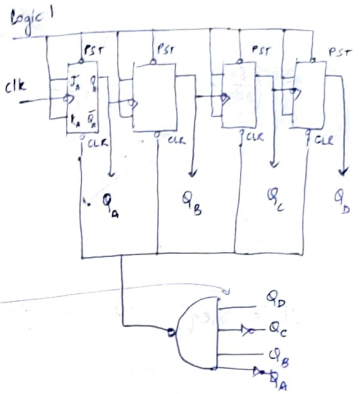
clock	Q_D	Q_C	Q_B	Q_A
initial	0	0	0	0
1st	0	0	0	1
2nd	0	0	1	0
	:			
	1	0	0	0
	1	0	0	1
	1	0	1	0
	:			
	1	1	1	1

Decade counter

- 10

CLR = 0 Q = 0 (Reset) ✓

PSR = 0 Q = 1 (SET)



This can also be done by only taking Q_D & Q_B since (from 0-19 1010 occurs only in 10)

How to Design Synchronous Counters

Step 1: Decide no. of flip flops

Step 2: Excitation table of FF

Step 3: State diagram & circuit excitation table

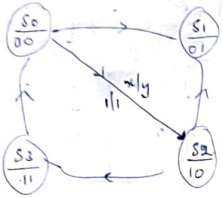
Step 4: Obtain simplified eqn using K'map

Step 4: Draw logic diagram.

Difference b/w Synchronous & Asynchronous ~~Ckts~~ Sequential Ckts

Synchronous	Asynchronous
→ Easy to design	→ Difficult to design
→ A clocked ff acts as memory element	→ An unclocked ff or time-delay element is used as memory element
→ They are slower	→ Faster as clock is not present
→ Edge triggered	→ level triggered

Introduction to State Table, State diagram & State Eq^n



eg^l

P.S		x	N.S		Y
Q_A	Q_B		Q_A^+	Q_B^+	
0	0	1	1	0	1

State eq^n: LHS = RHS
(Q_{n+1}) (P.S & i/p)

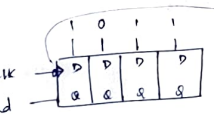
Combinat^n of P.S & i/p must make $Q_{n+1} = 1$

Registers:

→ FF is 1-bit memory cell
To increase the storage capacity, we have to use group of FF, known as register

→ To store n-bit data n FF req

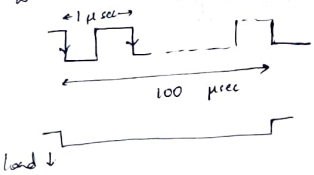
→ same clock is used in all ff's. ∴ we are bound to follow the clock.



Assume $f = 1\,MHz \Rightarrow T = \frac{1}{f} = 1\,\mu sec$
→ For every 1 μsec at -ve edge, value changes
Suppose we wish to hold the value 1011 for 100 μs,

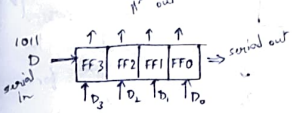
we use independent control called load.
Synchronous load : ∴ FF is operat^{n}al when Clk ↑ & load↑
Asynchronous load : only load ↑

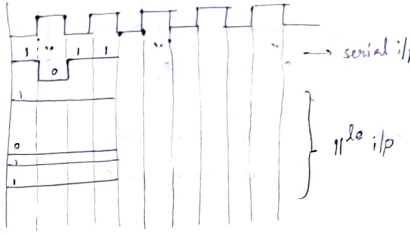
So for 100 μ sec we give load ↓



Data formats and Classification of Registers

→ Data can be entered in serial or in parallel form → called Special code
↓ called Temporal code
↓ one bit at a time
↓ all bits at a time





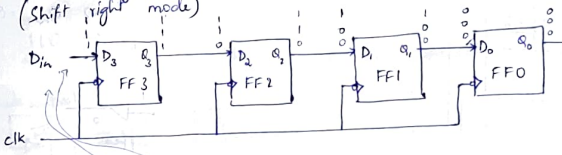
Classification of registers

i) Depending on i/p & o/p:
a) SISO b) SIPO c) PISO d) PIPO

ii) Depending on application
a) Shift reg.
b) Storage reg. (PIPO) (just stored)

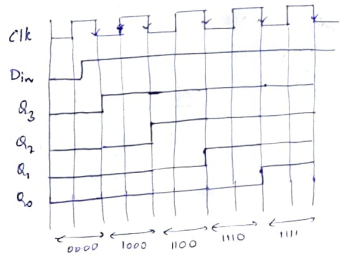
i) Shift Register (SISO)

(Shift right mode)



→ To store "1111"

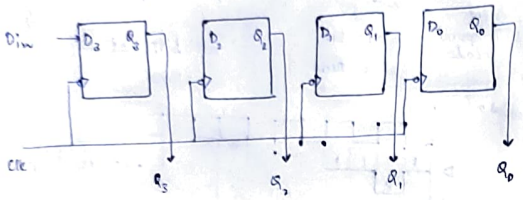
Clk	Q_3	Q_2	Q_1	Q_0
Initially	0	0	0	0
↓	1	0	0	0
↓	1	1	0	0
↓	1	1	1	0
↓	1	1	1	1



But since only Q_0 ↓ ↓ ↓ ↓
0 0 0 0 1

needs more clock cycles to get data

Shift Registers (SIPO)

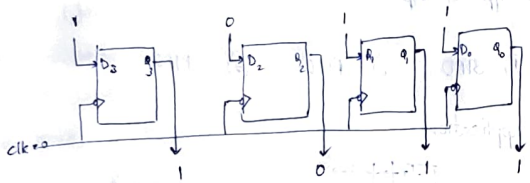


Applicat^n: when we req ouput serial data in ||^le form

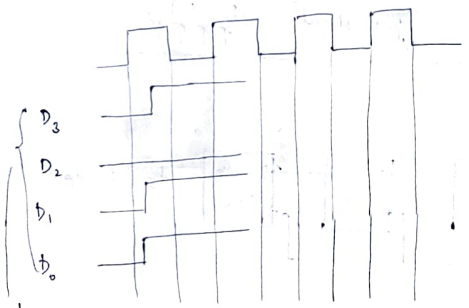
SISO → 4 clock pulses to store data → But more pulses to get out data

SIPO → 4 clock pulses " "

(Shift register) (PIPO) / Storage reg / Buffer reg:



Suppose to store "1011"

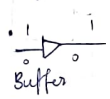


note how we give i/p's before edge falls

D = 0 ⇒ $Q_{n+1} = 0$

D = 1 ⇒ $Q_{n+1} = 1$

clk = 0 ⇒ $Q_{n+1} = Q_n$

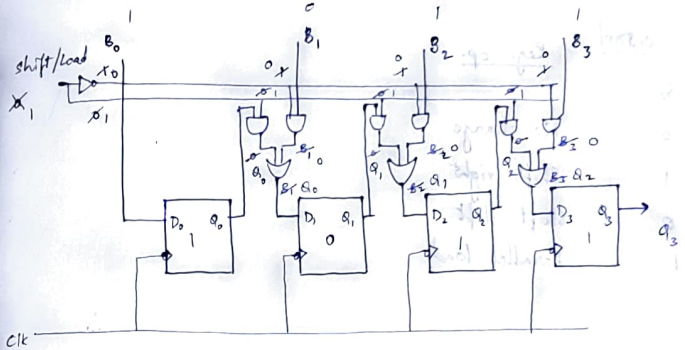


→ 1 clock pulse to store data

Shift register (PISO)

Two modes:

1) load mode → to achieve parallel i/p
2) shift mode



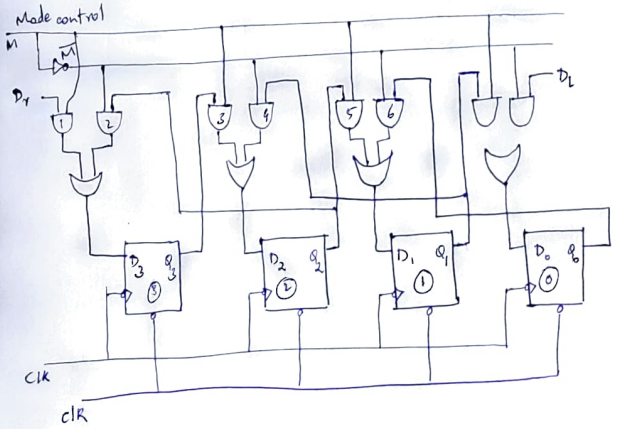
Bidirectional Shift Register

011 → (3)
shift left

$(3)_{10} \xrightarrow{\times 2} (6)_{10}$ (shift left)

110 → (6)

(6) $\xrightarrow{\div 2}$ (3) (shift right)



M>1 ⇒ shift right

Universal Shift Register

Bidirectional S.R + parallel loading = U.S.R
(data i/p is parallel)

Mode control		Reg. op.
S_1	S_0	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load