__/__/____

☆ Digital Systems

- Interconnection of digital modules and a system that manipulates discrete elements (0 or 1) of information represented internally in binary form.

– Characteristics

- Manipulates discrete elements of information (digits such as 10 decimal digits or 26 letters of the alphabet)

☆ Logical Operations

① AND (·)

② OR (+)

③ NOT ($^-$)(')(~)

Note • No. of combinations that can be generated is 2^n ($n \rightarrow$ no. of inputs)

Eqs – $Y = A.B \rightarrow$ Y is equal to A AND B

$Z = x + y \rightarrow$ z is equal to x OR y

AND	OR	NOT
$0.0 = 0$	$0+0=0$	$\bar{1} = 0$
$0.1 = 0$	$0+1=1$	$\bar{0} = 1$
$1.0 = 0$	$1+0=1$	
$1.1 = 1$	$1+1=1$	

__/__/__

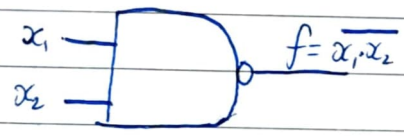
- Truth table is used to represent Boolean expression of a logic gate function.

a	b	AND	OR	NOT	NAND	NOR	XOR	XNOR
0	0	0	0	1	1	1	0	1
0	1	0	1	1	1	0	1	0
1	0	0	1	0	1	0	1	0
1	1	1	1	0	0	0	0	1

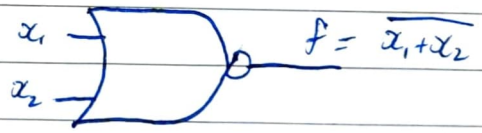
Note: AND gate gives HIGH(1) output if and only if all inputs are HIGH

OR gate gives LOW(0) output if and only if all inputs are LOW.

④ NAND (Complement of AND)



⑤ NOR (Complement of OR)



Note NAND gate gives LOW output if and only if all of it's inputs are HIGH

NOR gate gives HIGH output if and only if all of it's inputs are LOW.

⑥ Exclusive OR (XOR)

Note XOR gate output is HIGH if it has odd no. of HIGH inputs.

Note
$x + 1 = x$
$x \cdot x = x$
$x + x = x$

$x \cdot \bar{x} = 0$
$x + \bar{x} = 1$

$\bar{\bar{x}} = x$

$x + \bar{x}y = x + y$

— Duality

- Obtained by replacing $+ \rightleftarrows \cdot$, $1 \rightleftarrows 0$

— De Morgan's Theorem

$\overline{x \cdot y} = \bar{x} + \bar{y}$

$\overline{x + y} = \bar{x} \cdot \bar{y}$

Note Precedence : NOT > AND > OR

Ex Prove validity of given logical equation:

$$x_1 \cdot \bar{x}_3 + \bar{x}_2 \cdot \bar{x}_3 + x_1 \cdot x_3 + \bar{x}_2 x_3 = \bar{x}_1 \bar{x}_2 + x_1 x_2 + x_1 \cdot \bar{x}_2$$

LHS: $x_1 (\bar{x}_3 + x_3) + \bar{x}_2 (x_3 + \bar{x}_3)$
$= x_1 + \bar{x}_2$

RHS: $x_1 (x_2 + \bar{x}_2) + \bar{x}_1 \bar{x}_2$
$= x_1 + \bar{x}_1 \bar{x}_2$
$= x_1 + \bar{x}_2$

$\therefore$ LHS = RHS

__/__/____

– Sum-of-Product form (SOP)

- A product term in which each of n variables appears once, either complemented or uncomplemented form is called a minterm.

x_1	x_2	x_3	Minterm
0	0	0	$m_0 = \bar{x}_1 \bar{x}_2 \bar{x}_3$
0	0	1	$m_1 = \bar{x}_1 \bar{x}_2 x_3$
0	1	0	$m_2 = \bar{x}_1 x_2 \bar{x}_3$
0	1	1	$m_3 = \bar{x}_1 x_2 x_3$
1	0	0	$m_4 = x_1 \bar{x}_2 \bar{x}_3$
1	0	1	$m_5 = x_1 \bar{x}_2 x_3$
1	1	0	$m_6 = x_1 x_2 \bar{x}_3$
1	1	1	$m_7 = x_1 x_2 x_3$

- Any function f can be represented as sum of minterms (SOP) in truth table for which $f = 1$

Note: Here $A = 1$, $\bar{A} = 0$

Ex:- $f(x_1, x_2, x_3) = \sum m(1, 4, 5, 6)$

$= \bar{x}_1 \bar{x}_2 x_3 + x_1 \bar{x}_2 \bar{x}_3 + x_1 \bar{x}_2 x_3 + x_1 x_2 \bar{x}_3$

Note: Cost of logic circuit = Total no. of gates + Total no. of inputs to all gates.

__/__/____

Ex: $f(x_1, x_2, x_3) = \sum m(2, 3, 4, 6, 7)$

$= \bar{x}_1 x_2 \bar{x}_3 + \bar{x}_1 x_2 x_3 + x_1 \bar{x}_2 \bar{x}_3 + x_1 x_2 \bar{x}_3 + x_1 x_2 x_3$

$= \bar{x}_1 x_2 (\bar{x}_3 + x_3) + x_1 \bar{x}_3 (\bar{x}_2 + x_2) + x_1 x_2 (x_3 + \bar{x}_3)$

$= \bar{x}_1 x_2 + x_1 \bar{x}_3 + x_1 x_2$

$= x_2 + x_1 \cdot \bar{x}_3$

— Product-of-Sum form (POS)

- A sum (OR) term in which each of n variables appear once either complemented or uncomplemented form is called maxterm.
- Any function f can be represented as product of maxterms (POS) in truth table for which $f = 0$.

Note: Here, $A = 0$, $\bar{A} = 1$

x_1	x_2	x_3	Maxterm
0	0	0	$x_1 + x_2 + x_3$
0	0	1	$x_1 + x_2 + \bar{x}_3$
0	1	0	$x_1 + \bar{x}_2 + x_3$
0	1	1	$x_1 + \bar{x}_2 + \bar{x}_3$
1	0	0	$\bar{x}_1 + x_2 + x_3$
1	0	1	$\bar{x}_1 + x_2 + \bar{x}_3$
1	1	0	$\bar{x}_1 + \bar{x}_2 + x_3$
1	1	1	$\bar{x}_1 + \bar{x}_2 + \bar{x}_3$

$f(x_1, x_2, x_3) = \sum m(0, 1, 3, 5, 6, 7)$

$= \prod M(0, 1, 5)$ (leftovers)

$= (x_1 + x_2 + x_3)(x_1 + x_2 + \bar{x}_3)(\bar{x}_1 + x_2 + \bar{x}_3)$

Ex: $f = x_1 x_3 + x_1 \bar{x}_2 + \bar{x}_1 x_2 x_3 + \bar{x}_1 \bar{x}_2 \bar{x}_3$

$= \bar{x}_2 (x_1 + \bar{x}_1 \bar{x}_3) + x_3 (x_1 + \bar{x}_1 x_2)$

$= \bar{x}_2 (x_1 + x_3) + x_3 (x_1 + x_2)$

$= x_1 \bar{x}_2 + \bar{x}_2 x_3 + x_1 x_3 + x_2 x_3$

$= x_1 \bar{x}_2 + x_1 x_3 + x_3$

✱ Karnaugh (K) Map

- Allow easy recognition of minterms, or maxterms, that can be grouped

– 3-Variable

x_3 \ x_1x_2	00	01	11	10
0	m_0	m_2	m_6	m_4
1	m_1	m_3	m_7	m_5

Note: Minterms can be grouped in forms of $2, 4, 8, .., 2^n$ or even single groups. (Bigger the better)

– 4-Variable

x_1x_2 \ x_3x_4	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

__/__/____

– 5- Variable

x_2x_3 \ x_4x_5	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

$x_1 = 0$

x_2x_3 \ x_4x_5	00	01	11	10
00	m_{16}	m_{17}	m_{19}	m_{18}
01	m_{20}	m_{21}	m_{23}	m_{22}
11	m_{28}	m_{29}	m_{31}	m_{30}
10	m_{24}	m_{25}	m_{27}	m_{26}

$x_1 = 1$

Note: In SOP form, 1's are to be grouped
In POS form, 0's are to be grouped.

- Don't Care terms (x) are used as fillers to make huge combinations in K-map. These terms don't care if they are left alone, unlike minterms and maxterms.
- Literals: $x_1 \bar{x}_2 x_3$ has 3 literals.
- Implicants: No. of minterms in f + No. of possible pairs ($f=1$) of minterms (2) + No. of groups of 4 or 2^n.

Ex: $F(x_1, x_2, x_3) = \sum m(0, 1, 2, 3, 7)$

Here, No. of implicants =
5 minterms $(m_0, m_1, m_2, m_3, m_7)$
+ 5 pairs of two $(m_0\text{-}m_2, m_1\text{-}m_3, m_2\text{-}m_3, m_3\text{-}m_7, m_0\text{-}m_1)$
+ 1 group of 4 $(\bar{x}_1)$ = 11

x_3 \ x_1x_2	00	01	11	10
0	1	1	0	0
1	1	1	1	0

__/__/____

- Prime implicants cannot be combined into another implicant having fewer literals.
- Cover is a collection of implicants for which $f = 1$.

— Minimization

- This procedure is used to minimize cost.

Ex: $f = (\bar{x}_1 + x_2)(\bar{x}_1 + x_3)$

$f = x_1 \bar{x}_2 + x_1 \bar{x}_3$

Ex: Find POS implementation and compare cost with SOP form.

(a) $f = \bar{x}_1 \bar{x}_3 \bar{x}_4 + x_2 \bar{x}_3 x_4 + x_1 x_3 x_4 + x_1 \bar{x}_2 x_3 \bar{x}_4$

(SOP: 3 input AND gate = $(8 \times 4) + (1 \times 4) = 16$
4 input OR gate $\Rightarrow 4 + 1 = 5$
Cost = 21)

$f = (x_1 + x_3 + x_4)(\bar{x}_2 + x_3 + \bar{x}_4)(\bar{x}_1 + \bar{x}_3 + \bar{x}_4)(x_2 + \bar{x}_3 + x_4)$

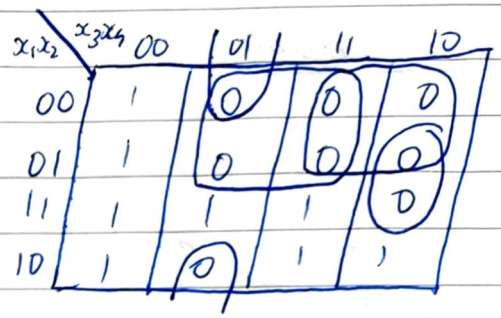
(POS: 3 input OR gate = 16
4 input AND gate = 5
Cost = 21)

(b)

x_1x_2 \ x_3x_4	00	01	11	10
00	1	0	0	0
01	1	0	0	0
11	1	1	1	0
10	1	0	1	1

$f = \bar{x}_3 \bar{x}_4 + x_1 x_2 x_4 + x_1 \bar{x}_2 x_3$

Cost: 2 input AND $\rightarrow 2+1=3$
3 input AND $\rightarrow 6+2=8$
3 input OR $\rightarrow 3+1=4$



$$f = (x_1 + \bar{x}_4)(x_1 + \bar{x}_3)(x_2 + x_3 + x_4)(\bar{x}_2 + \bar{x}_3 + x_4)$$

Cost: 2 input OR → 4+2=6
3 input OR → 6+2=8
4 input AND → 4+1=5 /19

Note: Majority function is a function that is equal to 1 if any two / all three of it's variables = 1.

Note: Fan-in is the no. of inputs that a gate can drive.

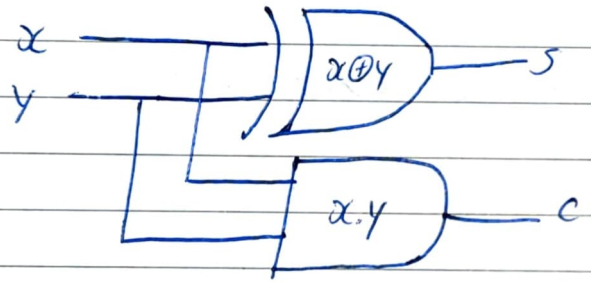
__/__/____

Arithmetic Circuits

✱ Adders

x	0	0	1	1
+ y	+ 0	+ 1	+ 0	+ 1
c s	0 0	0 1	0 1	1 0

carry ↑ sum ↑



Circuit

— Half Adder

- Implements addition of only two bits.

— Full Adder

- Implements addition of multiple bits
- For every bit position i, addition operation may include carry-in from position (i-1).

PTO↓

__/__/__

c_i	x_i	y_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

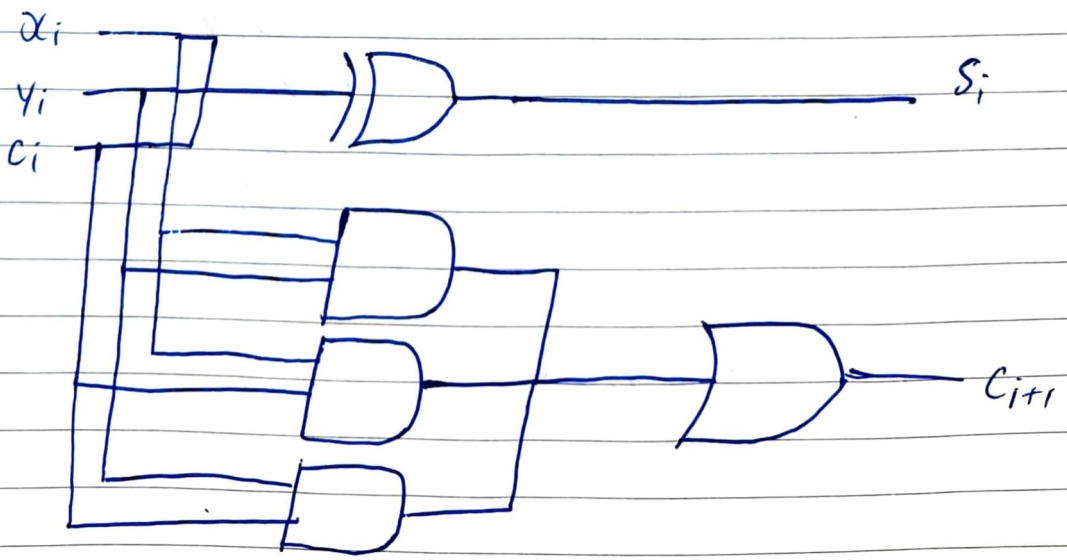
s_i \ x_iy_i	00	01	11	10
0		1		1
1	1		1	

c_i \ x_iy_i	00	01	11	10
0			1	
1		1	1	1

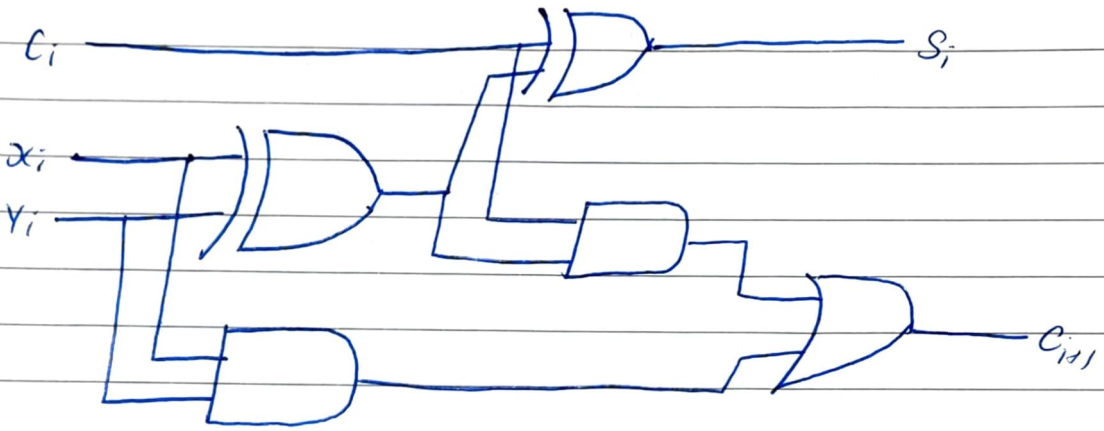
$$S_i = x_i \oplus y_i \oplus c_i \qquad C_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

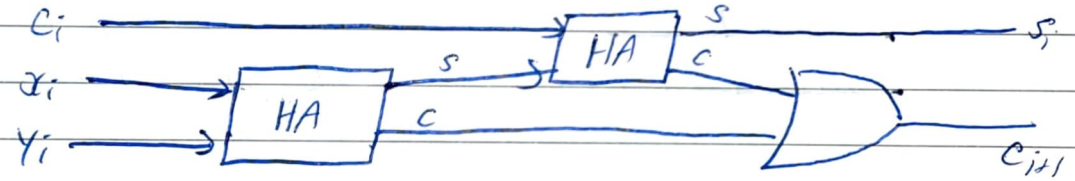
$$\downarrow$$

$$\begin{aligned} S_i &= \bar{x}_i y_i \bar{c}_i + x_i \bar{y}_i \bar{c}_i + \bar{x}_i \bar{y}_i c_i + x_i y_i c_i \\ &= (\bar{x}_i y_i + x_i \bar{y}_i)\bar{c}_i + (\bar{x}_i \bar{y}_i + x_i y_i) c_i \\ &= (x_i \oplus y_i)\bar{c}_i + (x_i \oplus y_i) c_i \\ &= (x_i \oplus y_i) \oplus c_i \end{aligned}$$



Decomposed Full Adder





$$C_{out} = (x_i \oplus y_i) C_{in} + x_i y_i$$

$$C_{i+1} = (x_i \bar{y_i} + \bar{x_i} y_i) c_i + x_i y_i$$

$$= x_i \bar{y_i} c_i + y_i (\bar{x_i} c_i + x_i)$$

$$= x_i \bar{y_i} c_i + y_i (c_i + x_i)$$

$$= x_i \bar{y_i} c_i + c_i y_i + x_i y_i$$

$$= x_i (y_i + \bar{y_i} c_i) + c_i y_i$$

$$= x_i (y_i + c_i) + c_i y_i$$

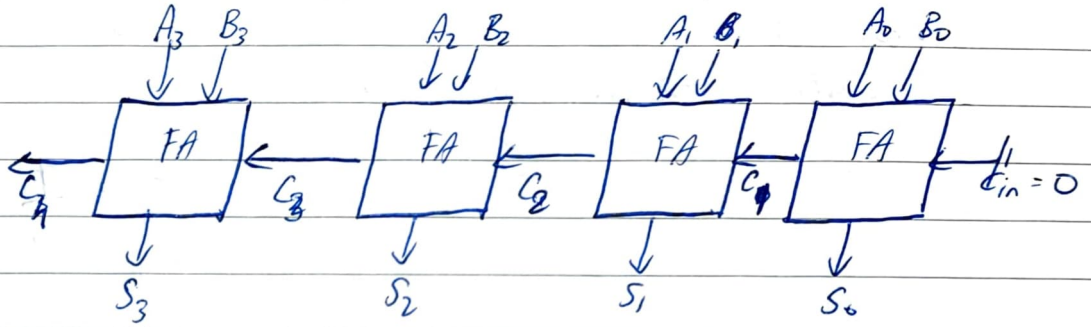
$$= x_i y_i + x_i c_i + c_i y_i$$

$$S_i = x_i \oplus y_i \oplus c_i$$

– Ripple-Carry Adder

- Combinational logic circuit
- Adding two n-bit binary numbers

Let us consider a 4-bit adder:



- Operands A_i and B_i are applied as inputs to adder, takes time before output sum, S. Full adder brings delay before S_i and C_{i+1}, delay is denoted by t.
- For addition of n-bits, time units required is n delay.

$$S_i = A_i \oplus B_i \oplus C_{in}$$

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

where $P_i = A_i \oplus B_i$, $G_i = A_i B_i$

P_i: carry propagates; G_i: carry generates

- Carry out from first stage c_1, arrives at second stage with delay t; carry out from second stage c_2 arrives at third stage with 2t delay.

- Signal C_{n-1} is valid after delay of $(n-1)t$, which means complete sum is available after delay of nt.

Arithmetic Overflow

- Result of addition / subtraction must fit in the range of -2^{n-1} to $2^{n-1}-1$ (where $n \rightarrow$ bits)
Otherwise overflow has occured

Ex:

7	0111	$C_4 = 0$
+ 2	+ 0010	$C_3 = 1$
9	1001	
(-7)	1001	$C_4 = 1$
+ (-2)	+ 1110	$C_3 = 0$
(-9)	10111	

- When numbers have opposite signs, there is no overflow.
- Overflow $\rightarrow$ $C_{n-1} \neq C_n$

$$\text{Overflow} = C_{n-1} \oplus C_n$$

- Carry-out is meaningful only for signed numbers

Ex: Let $X = x_3 x_2 x_1 x_0$
$Y = y_3 y_2 y_1 y_0$
$S = X + Y = s_3 s_2 s_1 s_0$

$$\text{Overflow} = x_3 y_3 \bar{s_3} + \bar{x_3} \bar{y_3} s_3$$

__/__/____

- Delay for carry-out signal for FA, Δt = 2 gate delays.
- Final result of addition will be valid after delay of $n\Delta t = 2n$ gate delays.
- For ripple carry adders, including XOR gates, delay = $(2n+1)$ gate delays.
- Longest delay → critical path delay
  Cause → critical path.
- To improve performance of complete adder, a carry-lookahead adder is used to evaluate C_{i+1} very quickly

$$C_{i+1} = g_i + p_i c_i$$
$$= g_i + p_i (g_{i-1} + p_{i-1} c_{i-1}) \quad \text{(in terms of } i-1\text{)}$$
$$= g_i + p_i g_{i-1} + p_i p_{i-1} c_{i-1}$$

$$= g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} \cdots p_2 p_1 g_0 + p_i p_{i-1} \cdots p_1 p_0 c_0$$

✱ Multiplication

- Two binary numbers can be multiplied similar to decimal numbers.

$$\begin{array}{r} 101 \\ \times\ 011 \\ \hline 101 \\ 101\times \\ 000\times\times \\ \hline 01111 \end{array}$$

0101 + 0001 = 0110

0111 + 0101 = 1100 + 0110 = 00010010

1001 + 0110 = 1111 + 0110 = 00010101

__/__/____

★ Binary-Code-Decimal (BCD)

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	0001 0000 (last 5 bits)

(1) (0)

Ex:

$$\begin{array}{r} 0111 \leftarrow 7 \\ +\,0101 \leftarrow 5 \\ \hline 1100 \leftarrow 12 \\ +\,0110 \leftarrow 6 \text{ (when } >9) \\ \hline 0001\,0010 \text{ (BCD)} \end{array}$$

12 → 0001 0010

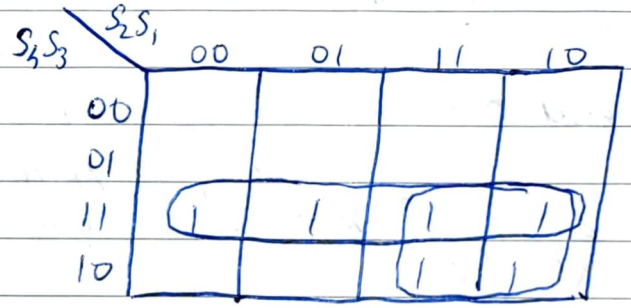
(1) (2)

N	S_4	S_3	S_2	S_1	C (BCD)
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0

$0101 \rightarrow \begin{array}{r} 1010 \\ +\quad 1 \\ \hline 1011 \end{array}$

__/__/____

8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

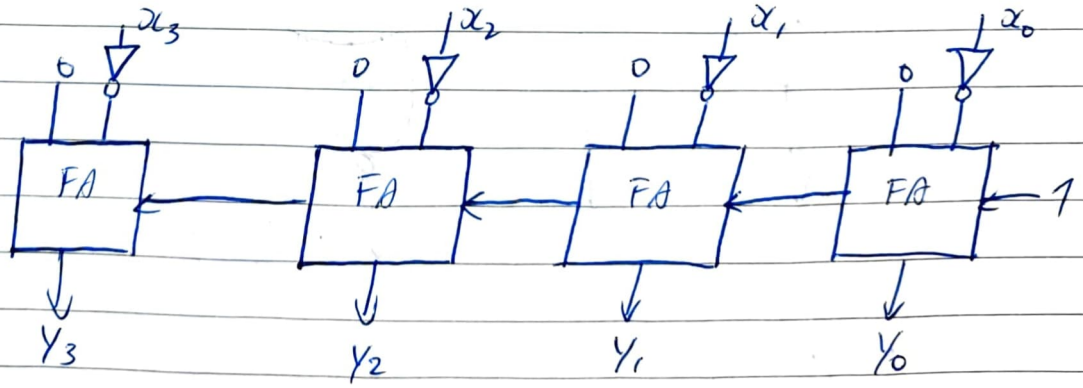


$F = S_4 S_3 + S_4 S_2$

- Correction should be done when carry = 1

$\therefore F = C + S_4 S_3 + S_4 S_2$

Ex:- Circuit using FA to generate 2's complement of 4-bit number X. $(x_3\, x_2\, x_1\, x_0)$



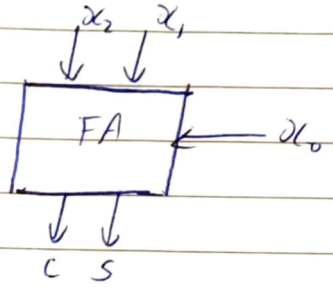
$X'' = Y = Y_3 Y_2 Y_1 Y_0$

$101 \rightsquigarrow 2$ $(c=1, s=0)$

0010 (x) 0111 (y)

$s_0=1, c_1=0$
$s_1=1, c_2=0$
$s_2=0, c_3=0$
$s_3=1, c_n=0$

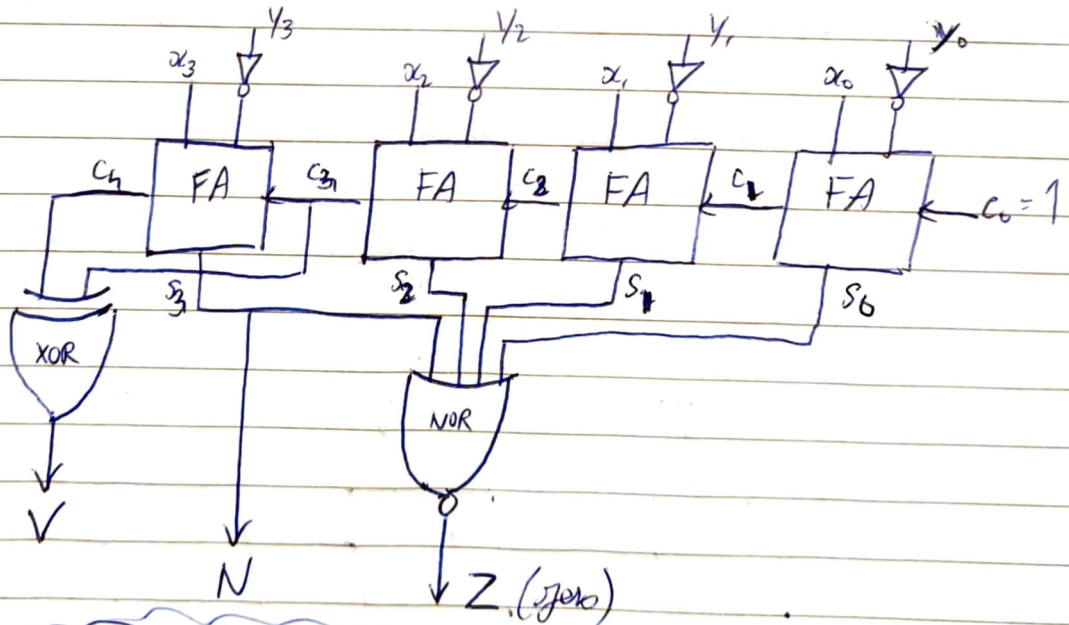
Ex: Design a circuit using FA to find no. of 1's in 3-bit unsigned numbers.



Result = CS (in binary)

— Arithmetic Comparison (using Subtractor)

- Two 4-bit signed numbers: $X = x_3 x_2 x_1 x_0$
  $Y = y_3 y_2 y_1 y_0$
- Outputs: (a) Z (=1 if result = 0, else = 0)
  (b) N (⊖) = 1, (+) = 0)
  (c) V (overflow = 1, else = 0)



- $X = Y \Rightarrow Z = 1$
  $X < Y \Rightarrow N \oplus V = 1$
  $X > Y \Rightarrow (Z + (N \oplus V))' = 1$

- $X \leq Y \Rightarrow Z + (N \oplus V) = 1$
  $X \geq Y \Rightarrow Z(N \oplus V)' = 1$

$V = C_4 \oplus C_3$
$N = S_3$
$Z = (S_3 + S_2 + S_1 + S_0)'$

Verilog HDL

- Structural specification of Logic Gates

$y = x1 \& x2 \Rightarrow$ and (y, x1, x2)

- Behavioral specification of Logic Gates

```
module example1(x1, x2, x3, f);
    input x1, x2, x3;
    output f;
    assign f = (x1&x2) | (~x2&x3);
endmodule
```

- Hierachial Verilog Code

```
module ex1 (carryin, x3, x2, x1, x0, y3, y2, y1, y0
                                  s3, s2, s1, s0, carryout);
    input carryin, x3, x2, x1, x0, y3, y2, y1, y0;
    output s3, s2, s1, s0, carryout;
    fulladd stage0 (carryin, x0, y0, s0, c1);
    fulladd stage1 (c1, x1, y1, s1, c2);
    fulladd stage2 (c2, x2, y2, s2, c3);
```

```
    fulladd stage3(c3, x3, y3, s3, carryout);
endmodule

module fulladd (Cin, x, y, s, Cout);
    input Cin, x, y;
    output s, Cout;
    assign s = x ^ y ^ Cin;
    assign Cout = (x&y) | (x&Cin) | (y&Cin);
endmodule
```

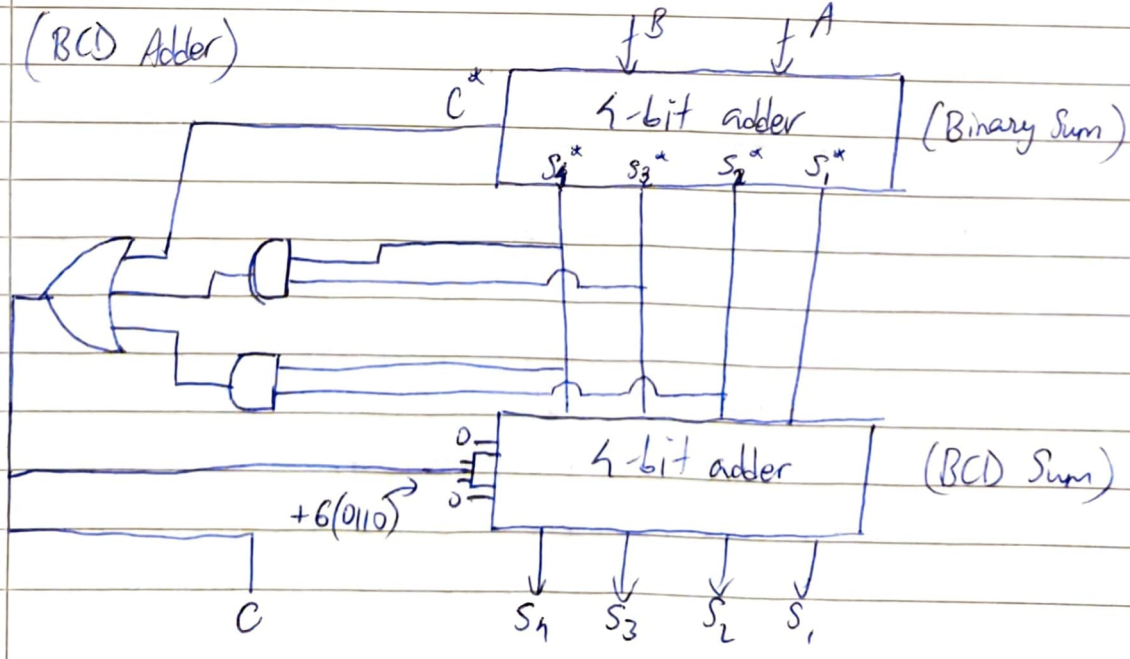
- Multibit signals are called vectors.

Ex: Four-bit vector W → input [3:0] W;

Here W[3] W[2] W[1] W[0] = W

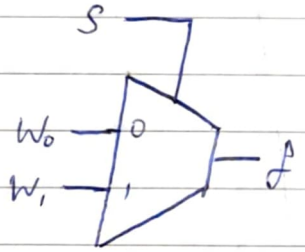
For input [0:3] W ⇒ W = W[0] W[1] W[2] W[3]

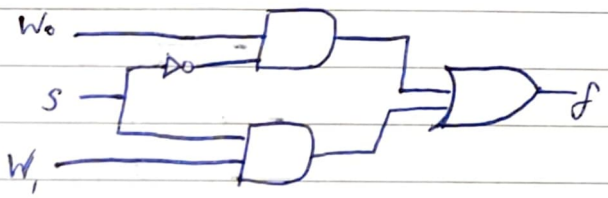
- Each bit can be referred by using index value.



Multiplexers

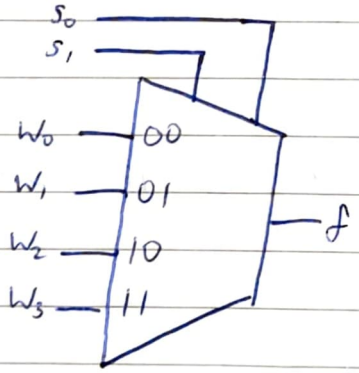
- A MUX has a no. of data inputs, one or more select inputs and one output.
- Data input is selected by values of select inputs.
- If there are n selection lines, there will be 2^n combs. of 0's and 1's.

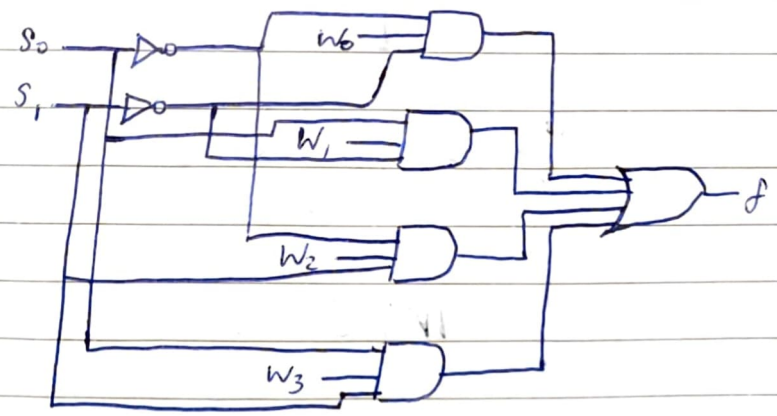




2:1 MUX

S	f
0	W_0
1	W_1



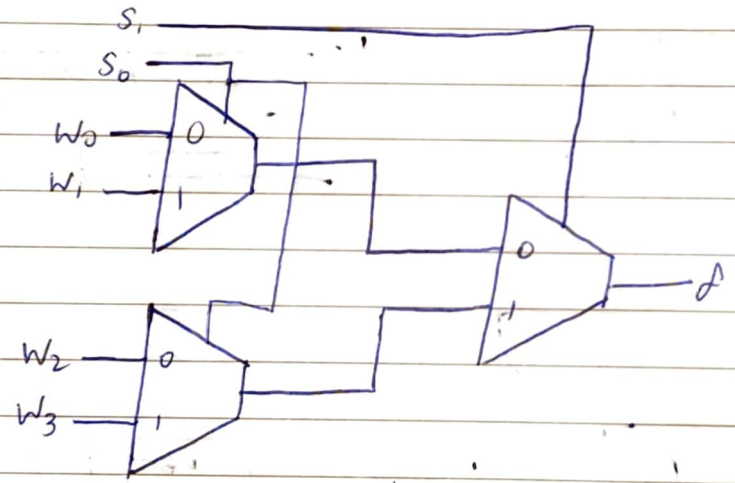


4:1 MUX

S_1	S_0	f
0	0	W_0
0	1	W_1
1	0	W_2
1	1	W_3

- A MUX having n data inputs requires $(\log 2n)$ select (s) inputs.
- For 2:1 MUX, $f = W_0(\sim S) + W_1 S$
- For 4:1 MUX, $f = \bar{S_1}\bar{S_0} W_0 + \bar{S_1} S_0 W_1 + S_1 \bar{S_0} W_2 + S_1 S_0 W_3$

→ • 4:1 MUX from 2:1 MUX:



- Similarly we can build 16:1 MUX from 4:1 MUX.

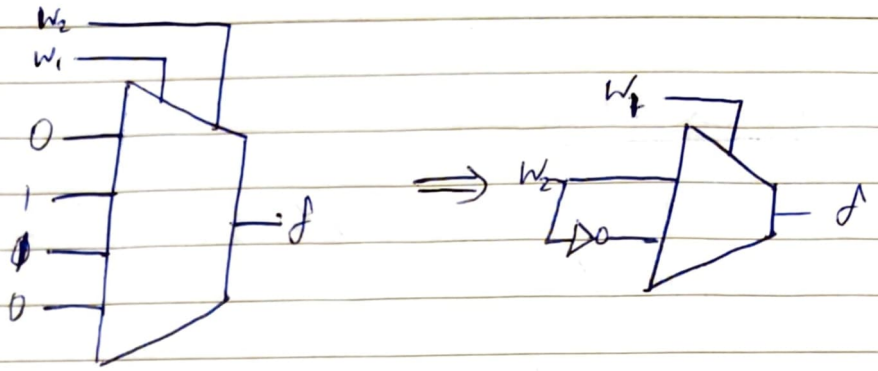
Ex: $f = W1 \oplus W2$

W_1	W_2	f
0	0	0
0	1	1
1	0	1
1	1	0

(4:1)

W_1 (S)	f
0	W_2
1	$\bar{W_2}$

(2:1)

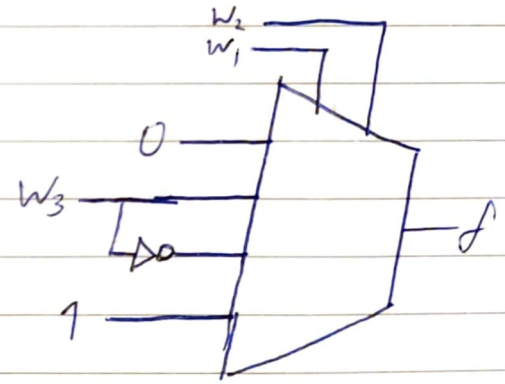


Ex: For three-input majority function,

W_1	W_2	W_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

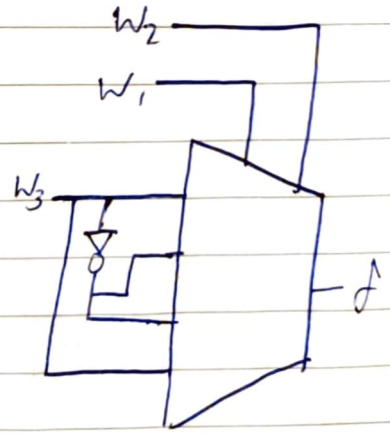
⇒

W_1	W_2	f
0	0	0
0	1	W_3
1	0	W_3
1	1	1



Ex: Implement 3-input XOR gate using 4:1 MUX.

W_1	W_2	W_3	f	
0	0	0	0	W_3
0	0	1	1	
0	1	0	1	$\bar{W}_3$
0	1	1	0	
1	0	0	1	$\bar{W}_3$
1	0	1	0	
1	1	0	0	W_3
1	1	1	1	



Shannon's Expansion

- Inputs to MUX are constants 0 and 1 or some variable or its complement.
- It is also possible to connect more complex circuits as inputs to MUX, allowing functions to be synthesized using combination of MUX and other logic gates.

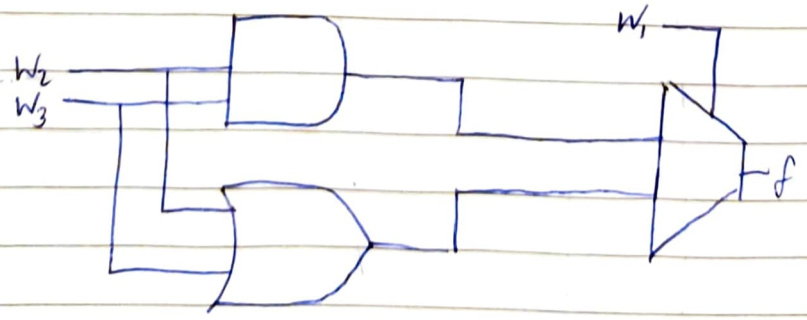
$$f = \bar{w}_1 w_2 w_3 + w_1 \bar{w}_2 w_3 + w_1 w_2 \bar{w}_3 + w_1 w_2 w_3$$

$$= \bar{w}_1 (w_2 w_3) + w_1 (w_2 + w_3)$$

$$= w_1 \oplus w_2 \oplus w_3 \qquad = w_1 w_2 + w_2 w_3 + w_1 w_3$$

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	f
0	$w_2 w_3$
1	$w_2 + w_3$



Ex: Implement the following function using (a) 2 to 1 (b) 4 to 1 MUX:

$f = \bar{w}_1 \bar{w}_3 + w_1 w_2 + w_1 w_3$

(a) Using w_1, $f = \bar{w}_1 f_{\bar{w}_1} + w_1 f_{w_1}$

$= \bar{w}_1 (\bar{w}_3) + w_1 (w_2 + w_3)$

(b) Using w_1 & w_2, $f = \bar{w}_1 \bar{w}_2 f_{\bar{w}_1 \bar{w}_2} + \bar{w}_1 w_2 f_{\bar{w}_1 w_2} + w_1 \bar{w}_2 f_{w_1 \bar{w}_2} + w_1 w_2 f_{w_1 w_2}$

$= \bar{w}_1 \bar{w}_2 (\bar{w}_3) + \bar{w}_1 w_2 (\bar{w}_3) + w_1 \bar{w}_2 (w_3) + w_1 w_2 (1)$

Ex: Implement using only 2 to 1 MUX:

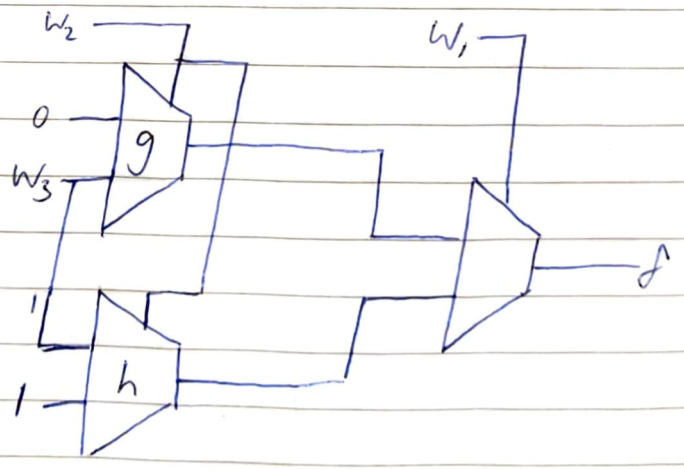
$f = w_1 w_2 + w_1 w_3 + w_2 w_3$

Using w_1

$f = \bar{w}_1 \underbrace{(w_2 w_3)}_{g} + w_1 \underbrace{(w_2 + w_3)}_{h}$

$g = \bar{w}_2 (0) + w_2 (w_3)$

$h = \bar{w}_2 (w_3) + w_2 (1)$

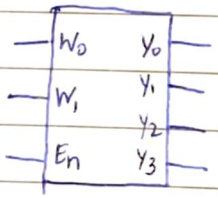


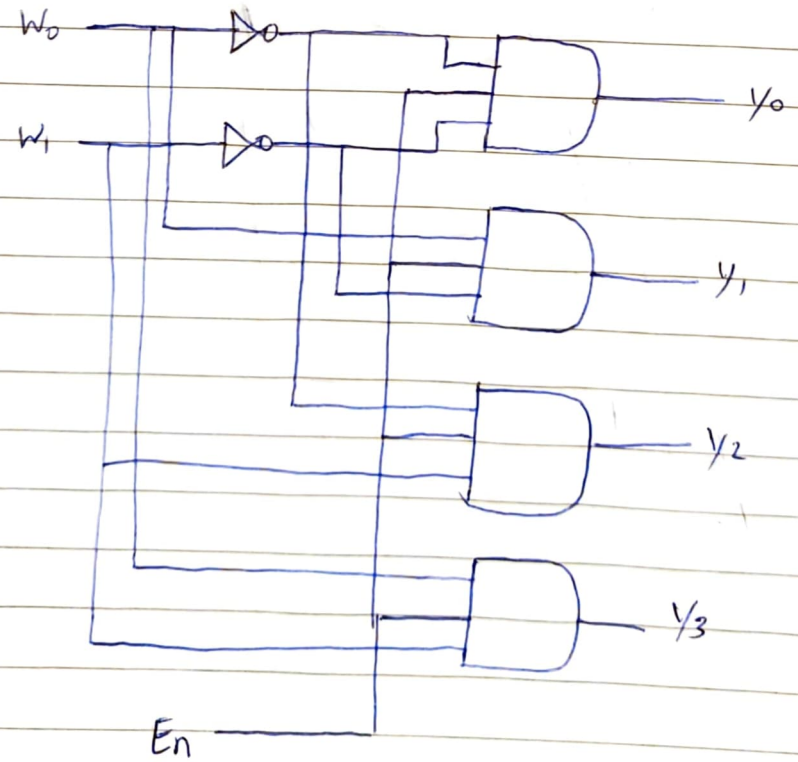
✱ Decoders

- It is a logic circuit with n inputs and 2^n outputs also having an enable input En.
- If $E_n = 1$, valuation of $W_{n-1}, \ldots, W_1, W_0$ determines which output to be asserted
  $E_n = 0$, none of decoder outputs asserted.

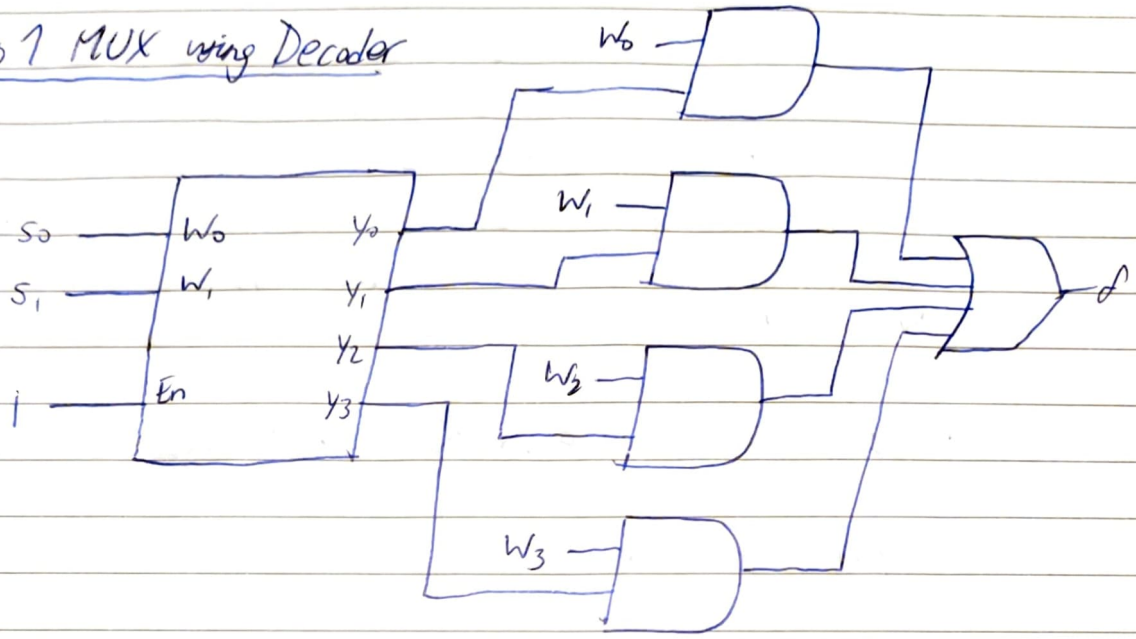
— 2 to 4 Decoder

E_n	W_1	W_0	Y_0	Y_1	Y_2	Y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0





- 4 to 1 MUX using Decoder



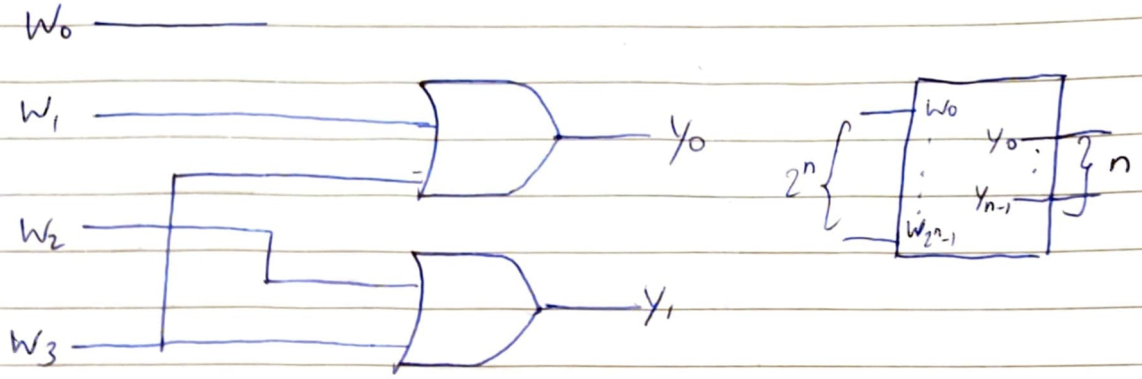
Encoder

- It performs opposite function of a decoder, it encodes information from 2^n inputs into n-bit code.

Note: An n-bit binary code in which exactly one bit is set to 1 → one-hot encoded. Outputs of binary decoder are one-hot encoded.

- Pattern having multiple inputs set to 1 (not shown in truth table) are treated as don't care conditions.

W_3	W_2	W_1	W_0	Y_1	Y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1



- Used for transmitting information in a digital system.

Priority Encoder

- Each input has priority level associated with it.
- When input of high priority is asserted, other inputs are ignored (low priority) (don't cares)

W_3	W_2	W_1	W_0	Y_1	Y_0	Z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	⊗	0	1	1
0	1	⊗	⊗	1	0	1
1	⊗	⊗	⊗	1	1	1

Here $z = 1$ (atleast one input = 1)
$z = 0$ (all inputs are 0)

- Here, W_3 has higher priority than W_0
- Each intermediate signal is = 1 only if input wk represents high-priority input that is set to 1.

$\bar{W}_3\bar{W}_2(W_1+W_0) + (W_3+W_2) = W_2 + \bar{W}_3 W_0 + W$

__/__/____

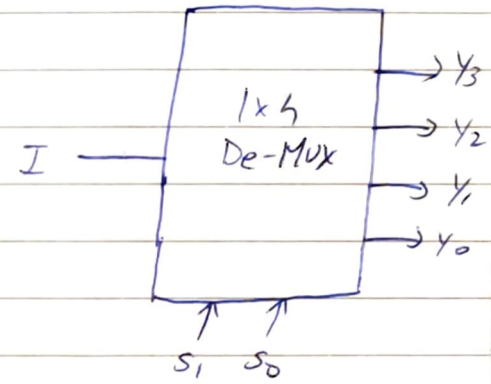
$i_0 = \bar{W}_3\bar{W}_2\bar{W}_1 W_0$
$i_1 = \bar{W}_3\bar{W}_2 W_1$
$i_2 = \bar{W}_3 W_2$
$i_3 = W_3$

From K-map
$\therefore Y_0 = i_1 + i_3$
$Y_1 = i_2 + i_3$
$Z = i_0 + i_1 + i_2 + i_3$

✡ De-MUX

- It has single input, 'n' selection lines and 2^n (max) outputs.

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



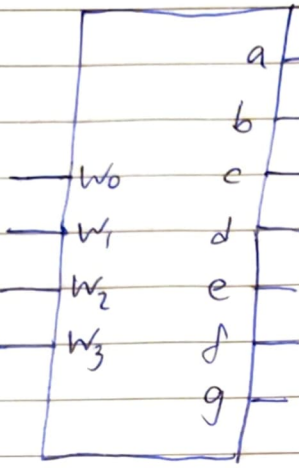
$Y_3 = S_1 S_0 I$
$Y_2 = S_1 \bar{S}_0 I$
$Y_1 = \bar{S}_1 S_0 I$
$Y_0 = \bar{S}_1 \bar{S}_0 I$

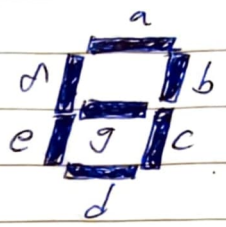
- Each combo can select only 1 output.

✡ Code Converters

- Purpose of decoder/encoder is to convert from one type of input encoding to diff output encoding.
- For example, BCD → 7-segment decoder which converts one binary-coded decimal digit to information suitable for driving a digital display.
- Each segment is a small LED whose glow is driven by electrical signal.

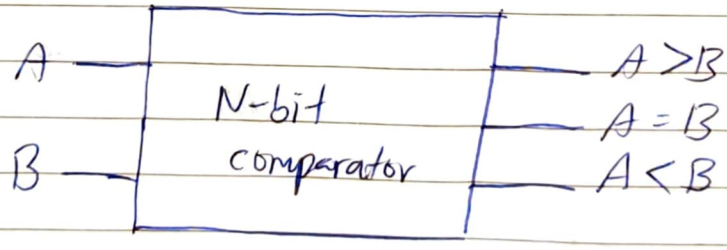
__/__/____





W_3	W_2	W_1	W_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	0	1	1
1	0	0	1	1	1	1	1	0	1	1

☆ Arithmetic Comparator



- For 1-bit number, $A > B : A\bar{B}$

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

$A < B : \bar{A}B$

2-bit Comparator

A1	A0	B1	B0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

- From the K-map of $A<B$, $A=B$, $A>B$,

$A>B : A1\bar{B}1 + A0\bar{B}1\bar{B}0 + A1A0\bar{B}0$

$A<B : \bar{A}1B1 + \bar{A}0B1B0 + \bar{A}1\bar{A}0B0$

$A=B : \bar{A}1\bar{A}0\bar{B}1\bar{B}0 + \bar{A}1\bar{A}0B1B0 + A1A0B1B0 + A1\bar{A}0B1\bar{B}0$

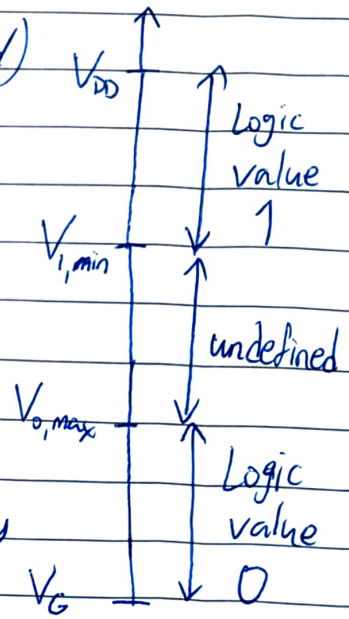
$= (A0 \odot B0)(A1 \odot B1)$

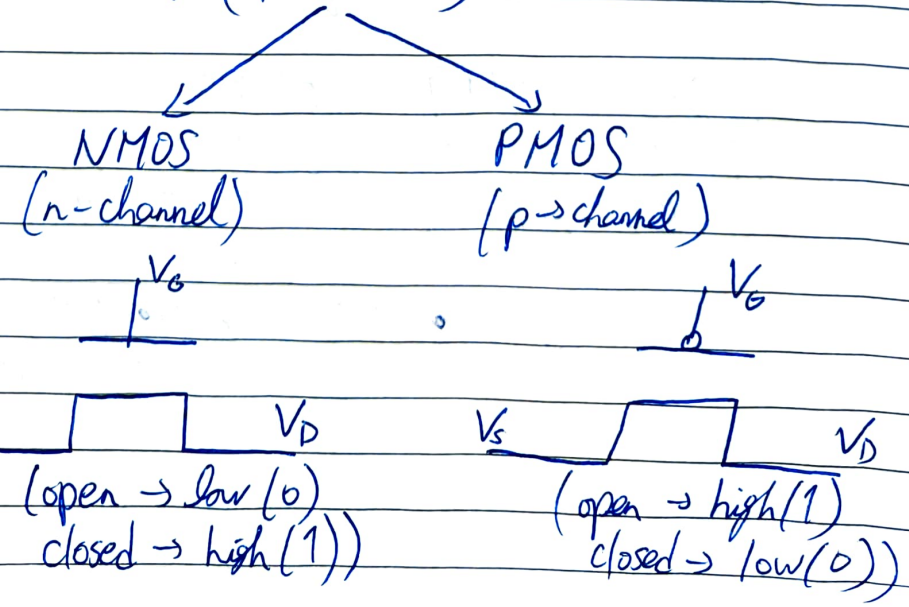
↳ X-NOR

- Used in CPU's, MCU's, process controllers, servo motor control
- Used in password verification and biometric applications.

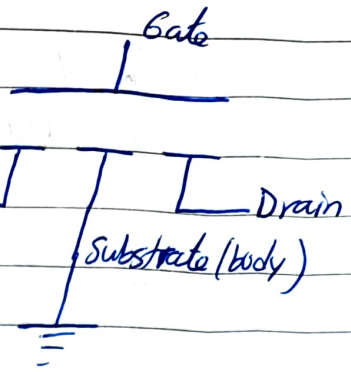
Switching Circuits (Module - 5)

- Minimum voltage is V_{SS} (or) V_G (ground)
  Maximum voltage is V_{DD} (supply)
- $V_{0,max}$ → max voltage level recognized as low
  $V_{1,min}$ → min voltage level recognized as high.
- Logic circuits are built with transistors implementing the switch, known as Metal - Oxide - Semiconductor - Field - Effect - Transistor. (MOSFET)





__/__/__

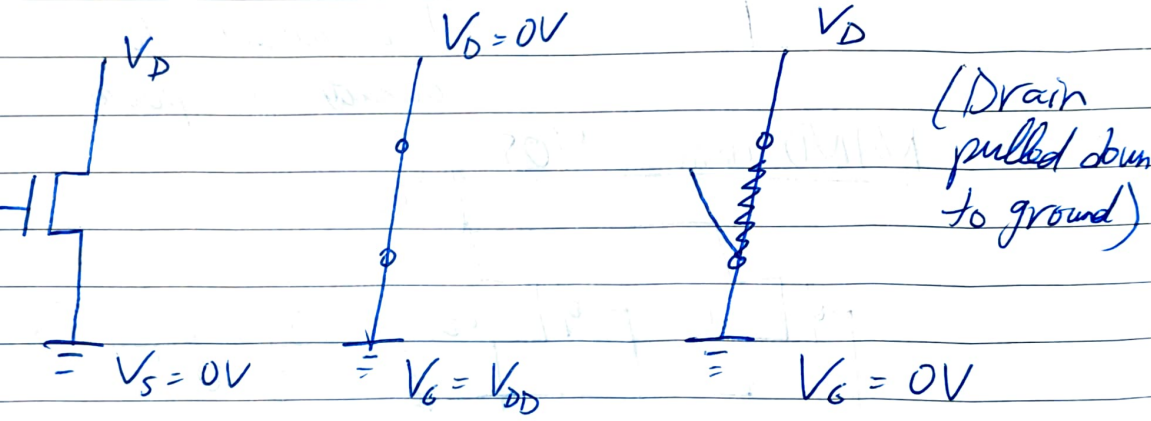


- Here, substrate (body) is connected to ground. Terminal ~~with~~ (Source with lower voltage level is source (in NMOS)

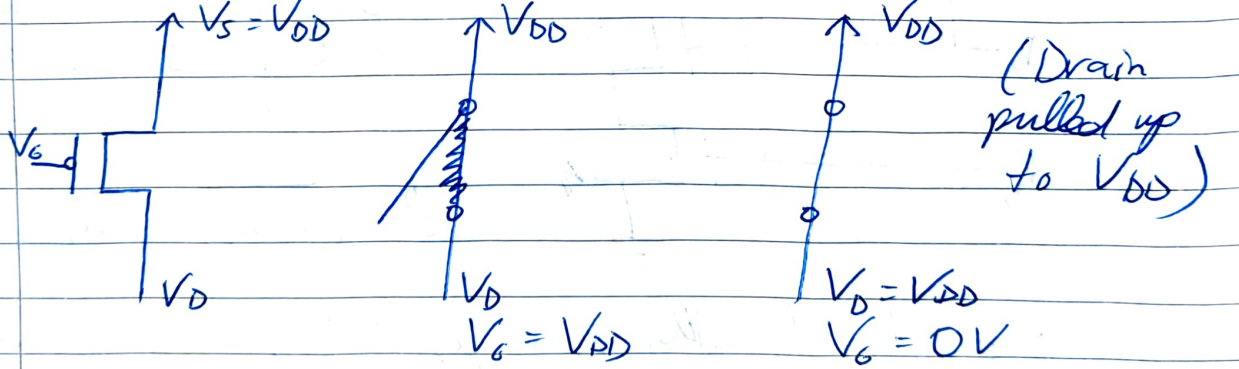
- V_G → low ⇒ transistor → OFF
  V_G → high ⇒ transistor → ON

- In PMOS, on the other hand, terminal with higher voltage acts as source.

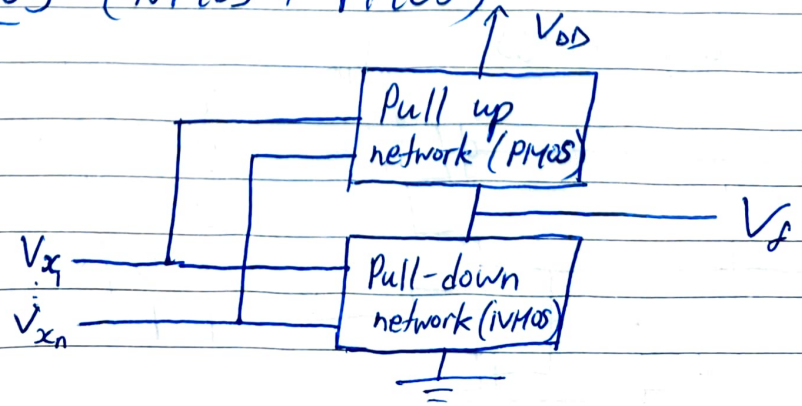
NMOS:



PMOS:



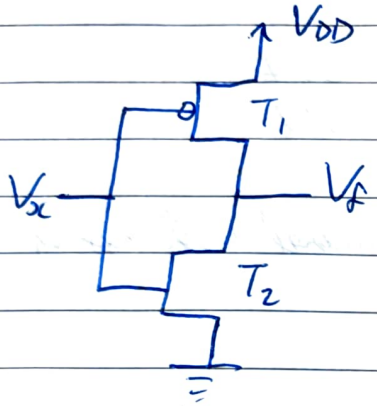
~~A~~ CMOS (NMOS + PMOS)



__/__/____

- In NMOS circuit, pull-up device is represented by a resistor

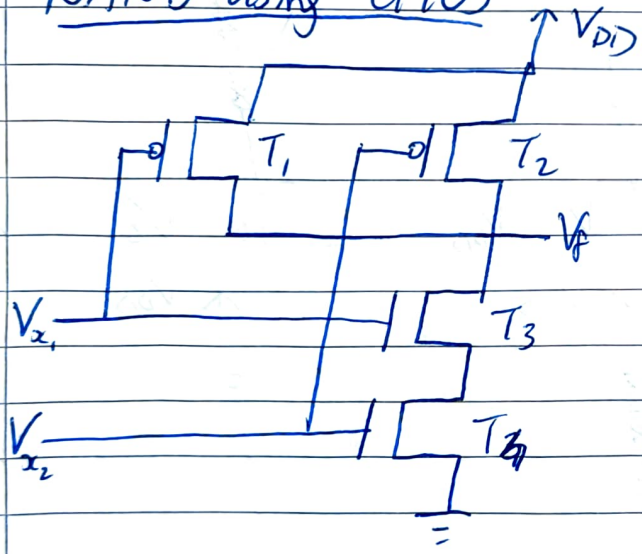
— NOT gate using CMOS



x	T_1	T_2	f
0	ON	OFF	1
1	OFF	ON	0

No current flow in all CMOS circuits. No power dissipated

— NAND using CMOS

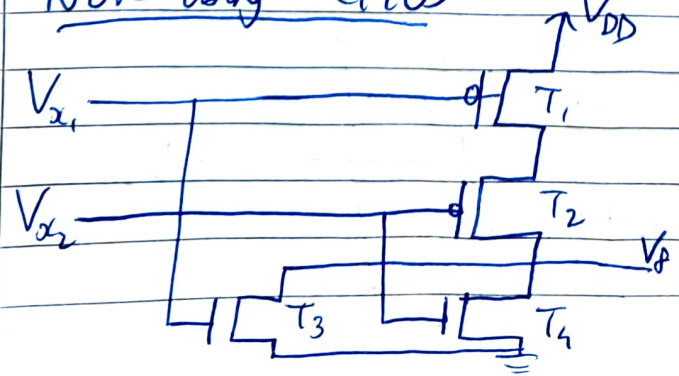


x_1	x_2	T_1	T_2	T_3	T_4	f
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

$$f = \overline{(x_1, x_2)} = \bar{x}_1 + \bar{x}_2$$

- $f = 1$, when either x_1 or $x_2 = 0$, $\Rightarrow$ 2 PMOS in parallel
  $f = 0$, when both x_1 and $x_2 = 1 \Rightarrow$ 2 NMOS in series.

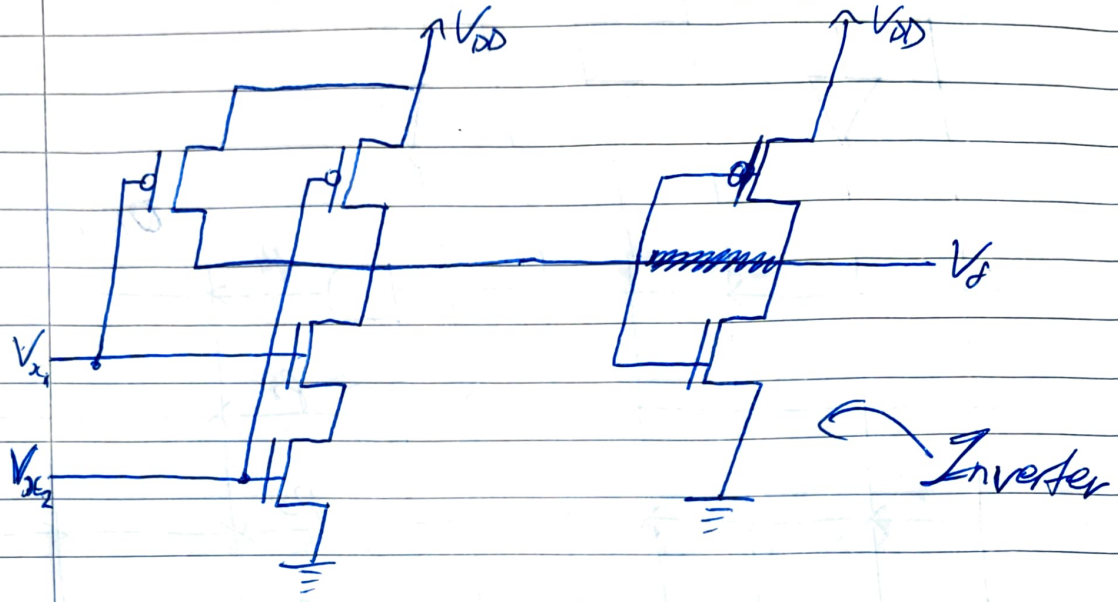
— NOR using CMOS



x_1	x_2	T_1	T_2	T_3	T_4	f
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	0
1	0	OFF	ON	ON	OFF	0
1	1	OFF	OFF	ON	ON	0

- $f = 1$, when both x_1 and $x_2 = 0 \Rightarrow$ 2 PMOS in series
  $f = 0$, when ~~both~~ either x_1 or $x_2 = 1 \Rightarrow$ 2 NMOS in parallel.

— AND using CMOS (NAND + Inverter (NOT))



Programmable Logic Array (PLA)

- It is possible to manufacture chips containing large amounts of logic circuitry with non-fixed structure.
- PLA → collection of AND gates feeding set of OR gates.

Ex: $F_1(A,B,C) = \Sigma m(0,1,6,7)$

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

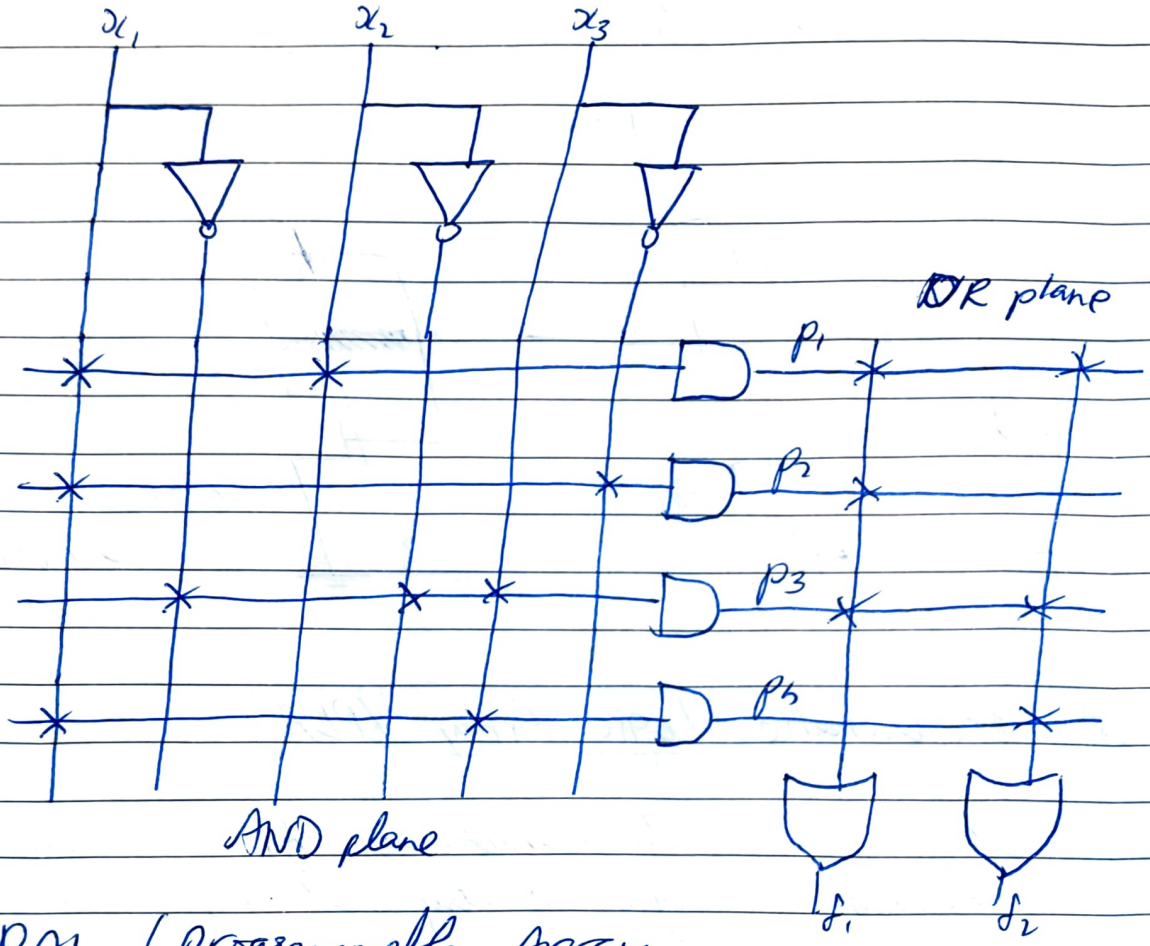
$F_2(A,B,C) = \Sigma m(2,3,4,5)$

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

__/__/____

Ex: $f_1 = x_1 x_2 + x_1 \bar{x}_3 + \bar{x}_1 \bar{x}_2 x_3$

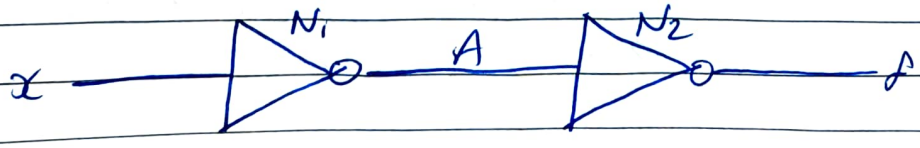
$f_2 = x_1 x_2 + \bar{x}_1 \bar{x}_2 x_3 + x_1 x_3$



Note: PAL (Programmable Array Logic) chips are simpler and better performing than PLA's, using fixed number of OR gates. (no OR plane)

✱ Noise Margin

- They are the random perturbations / disturbances that electronic circuits are constantly subjected to, which can alter output voltage levels.



- Consider N_1 produces low voltage level V_{OL} (output); presence of noise may alter voltage level, but as long as it remains less than V_{IL} (low input voltage level), it will be interpreted correctly.

- Noise margin is the ability to tolerate noise w/o affecting correct operation of circuit.

- Low noise margin: $NM_L = V_{IL} - V_{OL}$

- When N_1 produces V_{OH}, N_2 will interpret correct output as long as voltage is greater than V_{IH}.

- High noise margin: $NM_H = V_{OH} - V_{IH}$

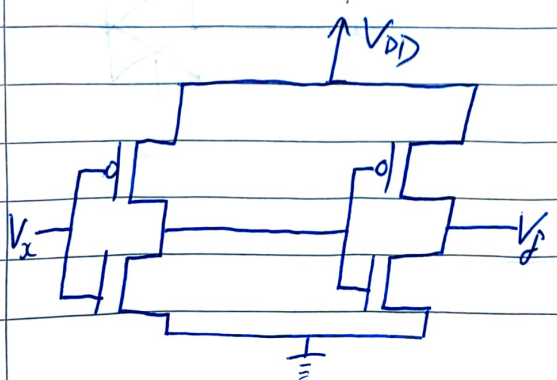
★ Fan-in, Fan-out

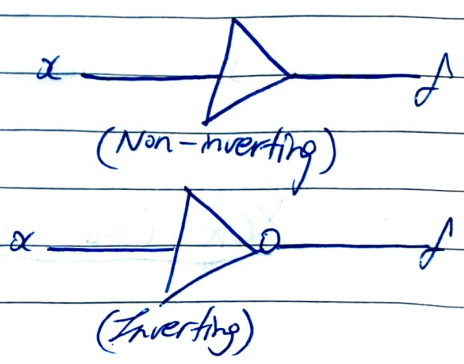
- Fan-in is defined as no. of inputs to gate.

- Fan-out is defined as no. of other gates that a specific gate drives.

★ Buffers

- Logic gate with one input x and one output f which produces $f = x$. Simplest implementation uses two inverters

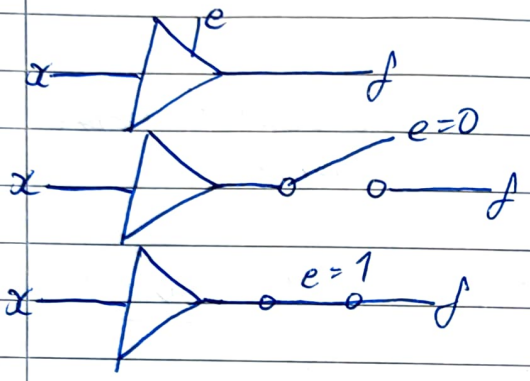




__/__/____

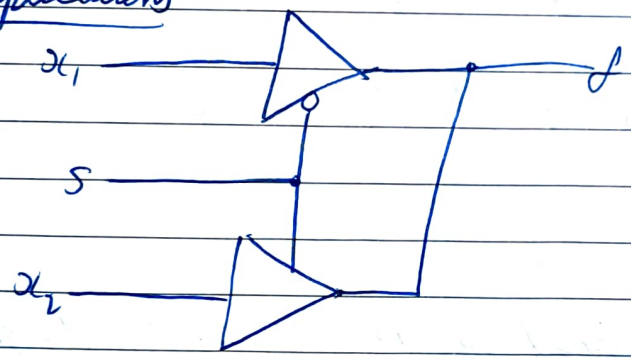
— Tri-state Buffers (0, 1, z)

- One input x, one output f, control input → enable e.



e	x	f
0	0	z
0	1	z
1	0	0
1	1	1

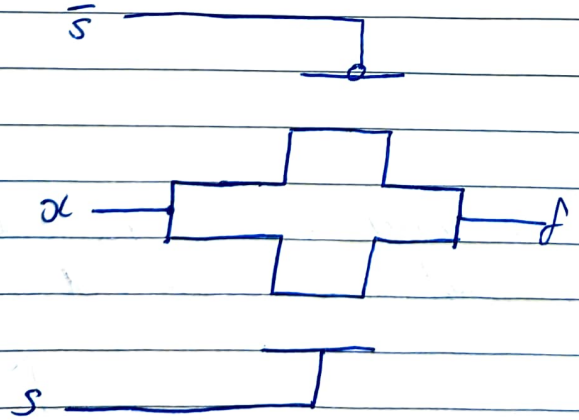
- Applications



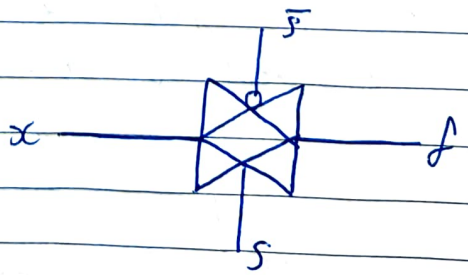
s	f
0	x_1
1	x_2

(MUX)

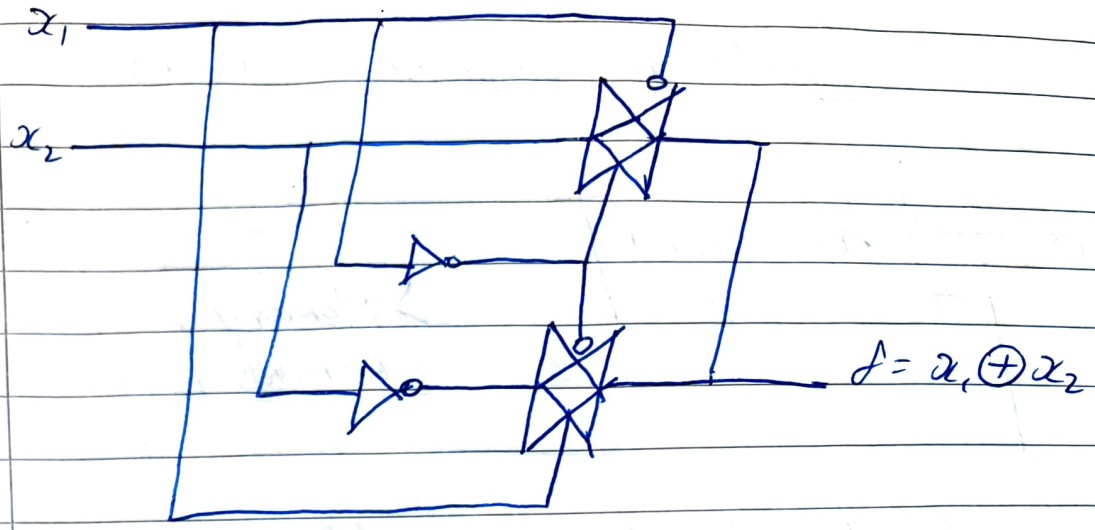
☆ Transmission Gates (NMOS + PMOS)



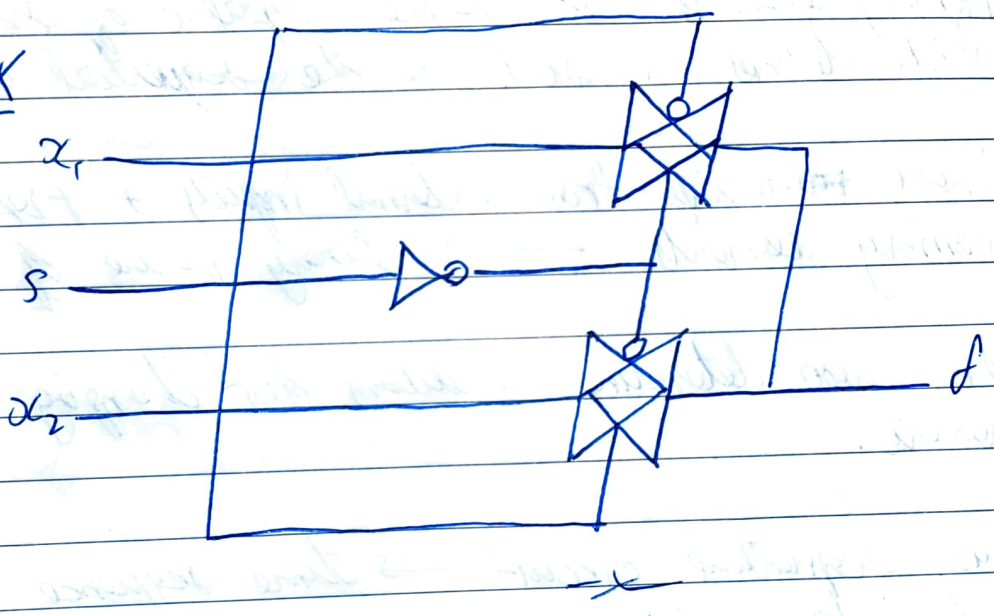
s	f
0	z
1	x



— XOR Gate using TG



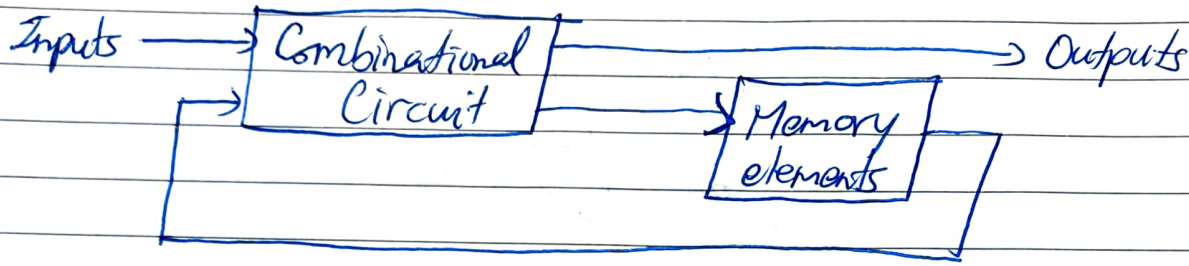
— MUX



Present O/P's → Present I/P + Past O/P

__/__/____

Synchronous Sequential Circuits



- Memory elements → devices capable of storing binary info. which defines the state of the sequential circuit.
- Binary ~~value~~ info from external inputs + present state of memory elements → binary value of output terminals
- They also determine conditions for changing state in memory elements.
- Thus, sequential circuit → time sequence of I/P, O/P and internal states.

Synchronous Seq. Circuit

- System whose behavior can be defined by signals at discrete instants of time, achieved by a timing device called clock generator, which produces a clock signal in the form of periodic clock pulses.
- Activity within circuit and updates in stored values is synchronized by occurrence of clock pulses.
- On the other hand, asynchronous sequential circuits

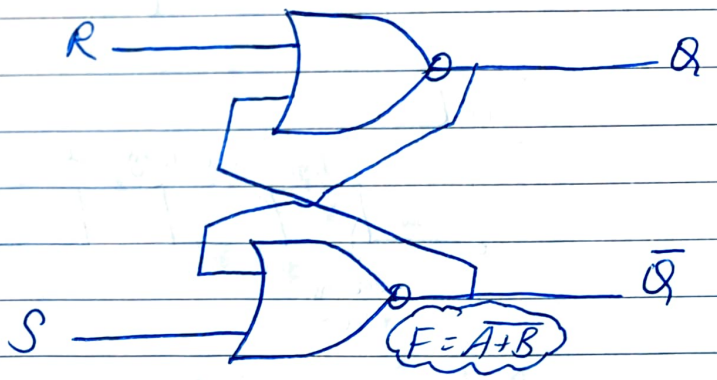
depend on order in which input signals change and can be affected at any instant of time.

✱ Flip - Flops

- A flip-flop is a binary storage device capable of storing one bit of information.
- In stable state, output is either 0 or 1.
- There are various types of FF, depending on no. of inputs and how inputs affect binary state.

— SR Latch (NOR)

- If (reset)(R), $Q = 0$
  If (set)(S), $Q = 1$
- For NOR gate,



A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

S	R	Q	$\bar{Q}$
0	0	Memory (prev)	
0	1	0	1
1	0	1	0
1	1	✗	

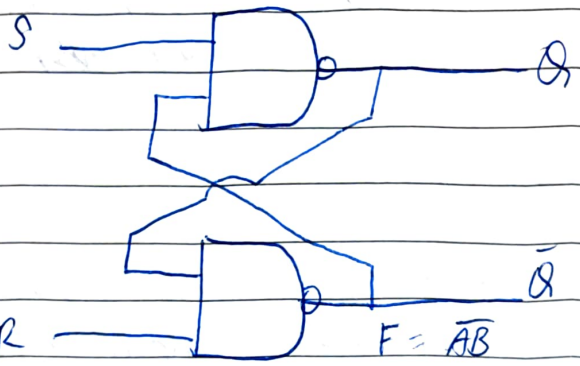
I) $S=0,\ R=1,\ Q=0,\ \bar{Q}=1$
$S=0,\ R=0,\ Q=0,\ \bar{Q}=1$ (as before)

II) $S=1,\ R=0,\ Q=1,\ \bar{Q}=0$
$S=0,\ R=0,\ Q=1,\ \bar{Q}=0$ (as before)

III) $S=1,\ R=1,\ Q=\bar{Q}=0$ (forbidden)

SR Latch (NAND)

S	R	Q	$\bar{Q}$
0	0	X	
0	1	1	0
1	0	0	1
1	1	Memory (prev)	



- For NAND gate,

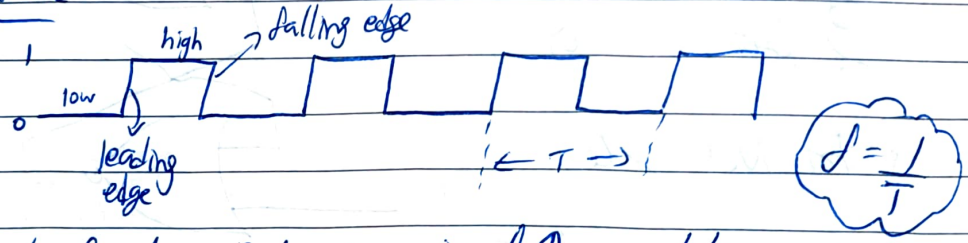
A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

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

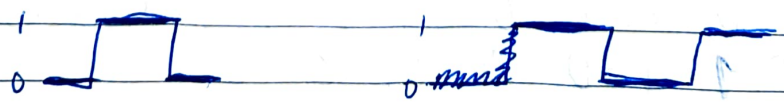
II) S = 1, R = 0, Q = 0, $\bar{Q}$ = 1
S = 1, R = 1, Q = 0, $\bar{Q}$ = 1

III) S = 0, R = 0, Q = $\bar{Q}$ = 1 (forbidden)

Clock

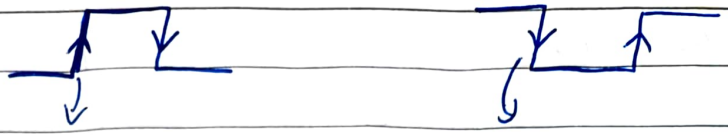


- Duty Cycle: Ratio of $\frac{\text{signal } \uparrow}{\text{Total time}} = \frac{t/2}{t} = \frac{1}{2} = 50\%$
- Clock is a signal that oscillates between high and low state
- State of FF is switched by momentary change / triggers in input signal.



(Positive pulse) (0→1) (Negative pulse) (1→0)

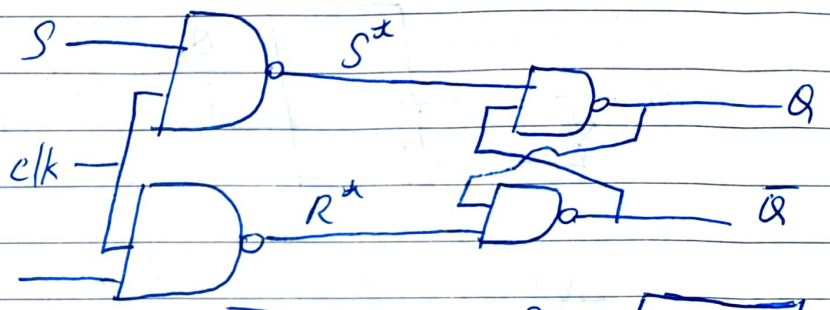
(Level Triggering)



(+) edge triggering (−) edge triggering

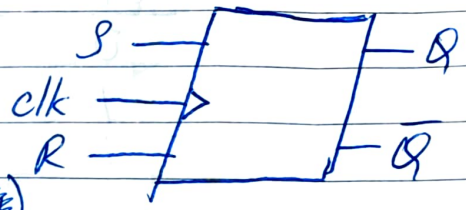
Note Seq. circuit will act as a LATCH if an enable input is used, which is level triggered during operation; and will act as a FLIP-FLOP when clock is used as input, which is edge-triggered, operational when clock is changing.

— SR Flip-flop



$S^* = \overline{S \cdot clk} = \bar{S} + \overline{clk}$

$R^* = \overline{R \cdot clk} = \bar{R} + \overline{clk}$



clk	S	R	Q_n Q̄
0	x	x	Memory (Q_n)
1	0	0	Memory (Q_n)
1	0	1	0 1
1	1	0	1 0
1	1	1	X

S^*	R^*	Q Q̄
0	0	X
0	1	1 0
1	0	0 1
1	1	Memory

	Q_n	S	R	Q_{n+1} (next state)
clk=1	0	0	0	0 (prev)
	0	0	1	0
	0	1	0	1
	0	1	1	X
	1	0	0	1 (prev)
	1	0	1	0
	1	1	0	1
	1	1	1	X

(Characteristic Table)

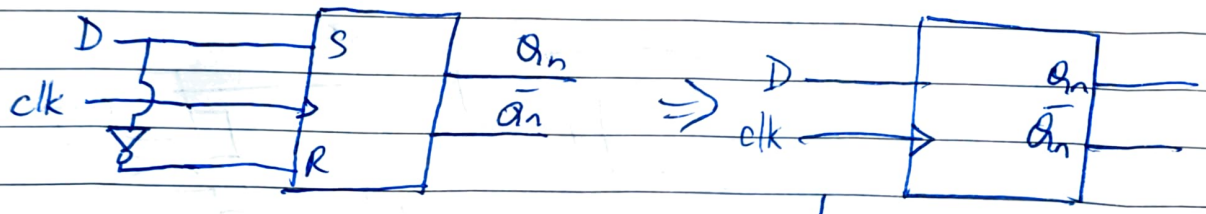
__/__/____

- **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

Note: $Q_{n+1} = S + Q_n \bar{R}$, $SR = 0$

— **D Flip-Flop**



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

TT

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

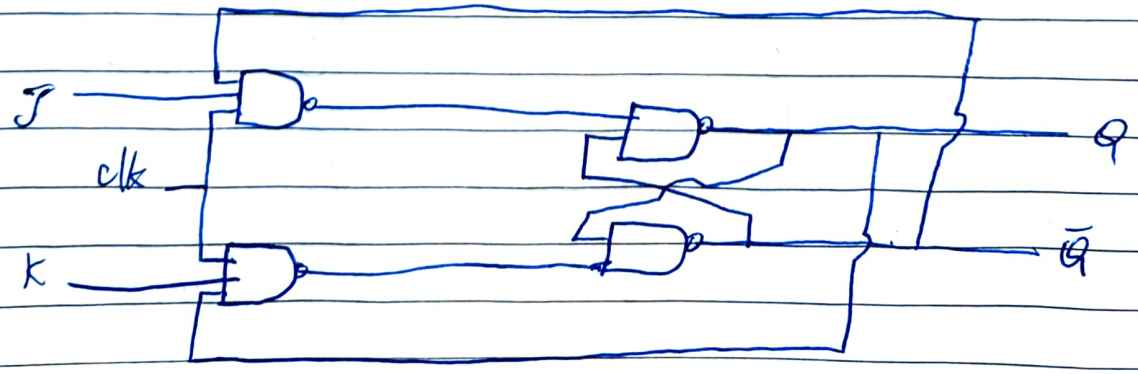
CT

$Q_{n+1} = D$

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

ET:

— **JK Flip-Flop**



I) $J=1,\ K=0,\ Q=1,\ \bar{Q}=0$
II) $J=0,\ K=1,\ Q=0,\ \bar{Q}=1$
III) $J=1,\ K=1,$

TT:

clk	J	K	Q_{n+1}
0	x	x	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	$\bar{Q}_n$

ET:

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

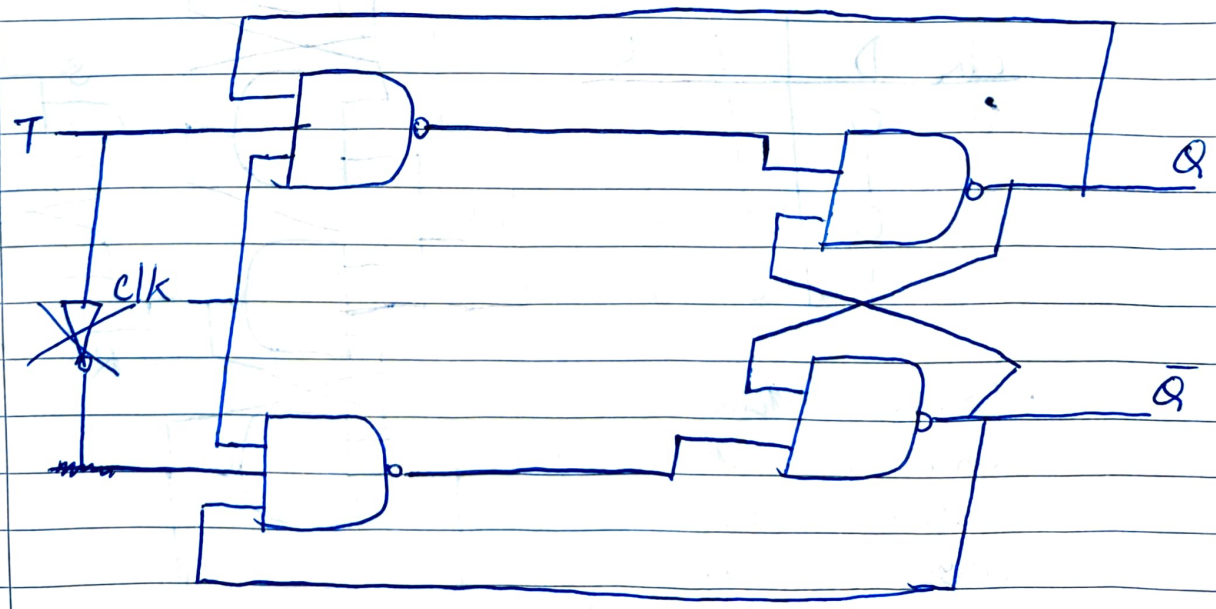
$Q_{n+1} = \bar{Q}_n J + Q_n \cdot \bar{K}$

CT

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

$Q_{n+1} = J$
$\overline{Q_{n+1}} = K$

— T Flip-Flop



__/__/____

TT:

clk	T	Q_{n+1}
0	x	Q_n
↓	0	Q_n
↓	1	$\overline{Q_n}$
↑		

CT:

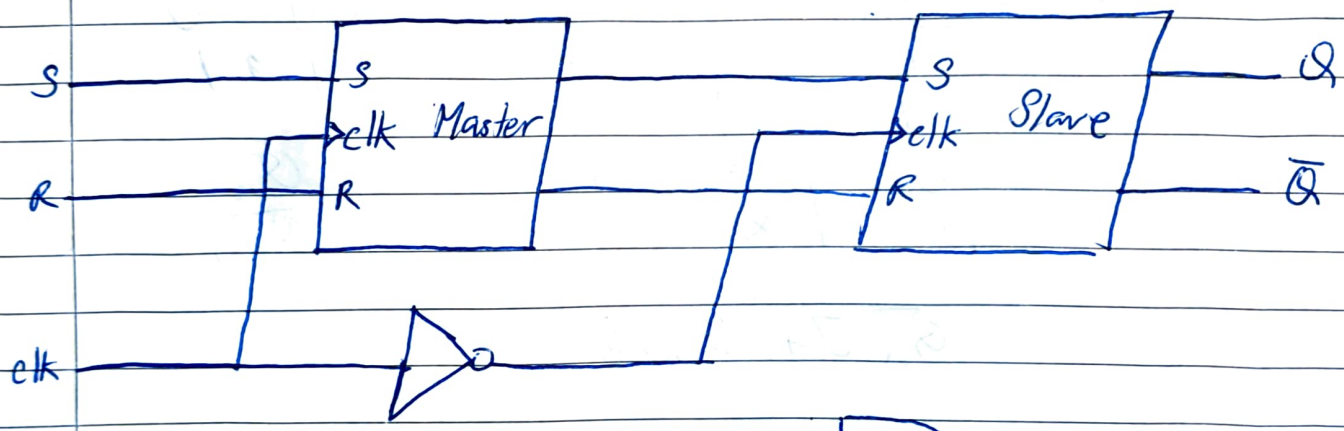
Q_n	T	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

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

ET:

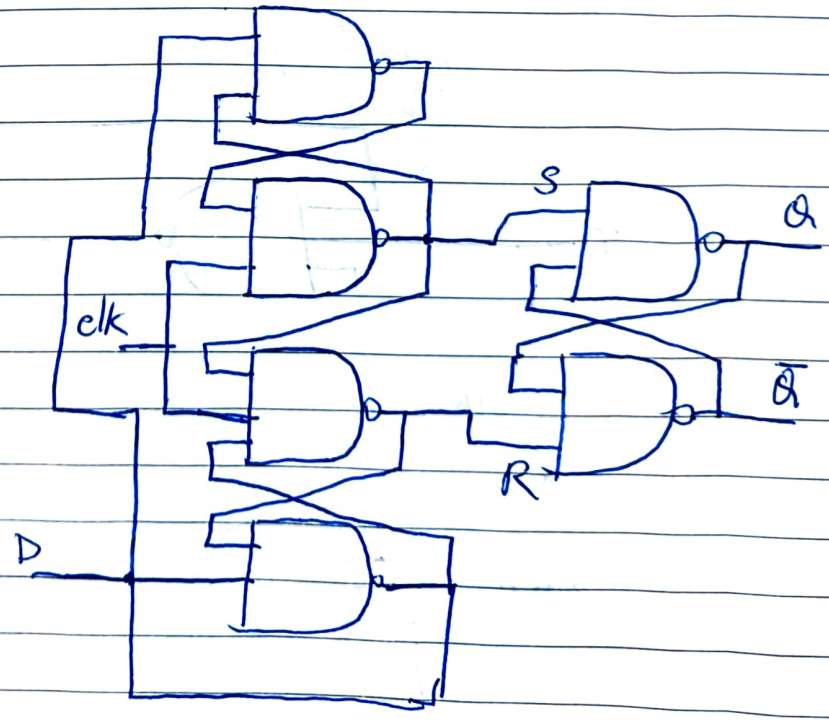
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

– Master Slave FF

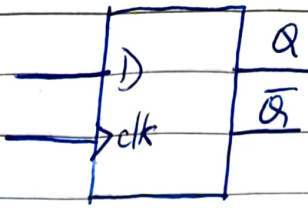


– Edge Trigged FF

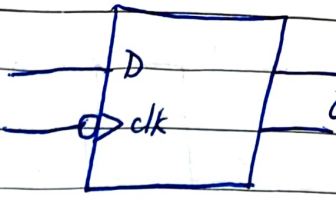
clk	D	S	R
0	0	1	1
0	1	1	1
↓	0	1	0
↓	1	0	1



- Propagation delay time of FF is interval between trigger edge and stabelization of output to a new state

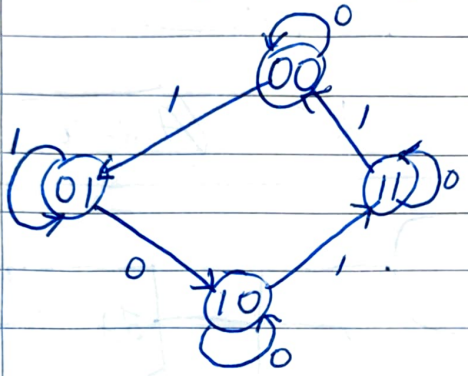


positive-edge triggered FF



negative-edge triggered FF

Exi- Design the synchronous sequential circuit for the following state diagram using JK FF.

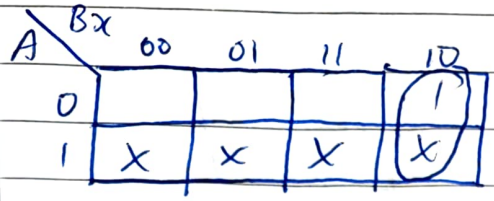


Present State		Next State			
		x=0		x=1	
A	B	A	B	A	B
0	0	0	0	0	1
0	1	1	0	0	1
1	0	1	0	1	1
1	1	1	1	0	0

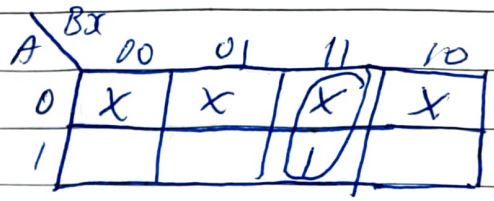
Input			Output			
Present State			Next State			
A	B	x	A	B	JA	KA	JB	KB
0	0	0	0	0	0	x	0	x
0	0	1	0	1	0	x	1	x
0	1	0	1	0	1	x	x	1
0	1	1	0	1	0	x	x	0
1	0	0	1	0	x	0	0	x
1	0	1	1	1	x	0	1	x
1	1	0	1	1	x	0	x	0
1	1	1	0	0	x	1	x	1

- Here JA and KA are based on A(present) and A(next) according to excitation table of JK flipflop

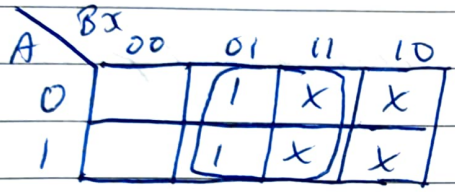
__/__/____



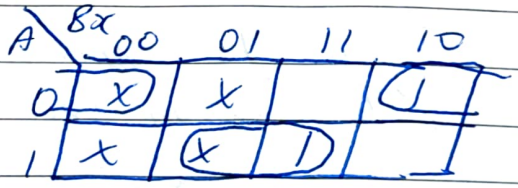
$JA = B\bar{x}$



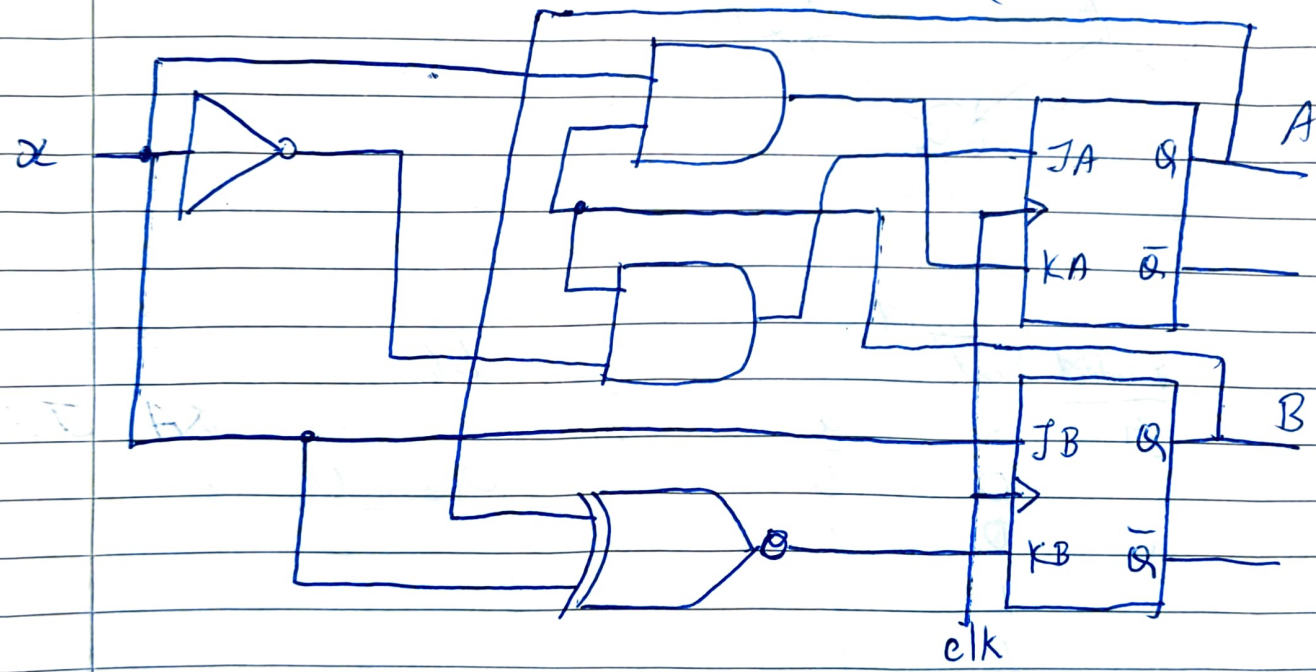
$KA = Bx$



$JB = x$



$KB = \overline{(A \oplus x)}$

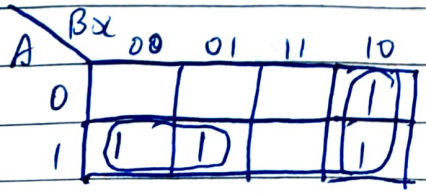


Ejer. Design with D FF.

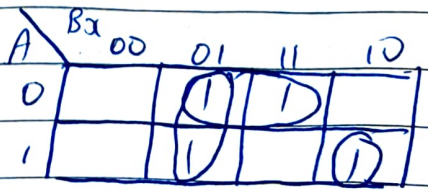
$A(t+1) = DA(A,B,x) = \sum(2,4,5,6)$

$B(t+1) = DB(A,B,x) = \sum(1,3,5,6)$

$Y(A,B,x) = \sum(1,5)$

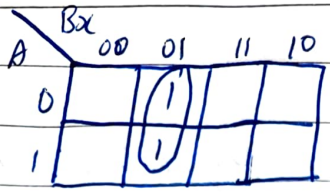


DA



DB

__/__/____

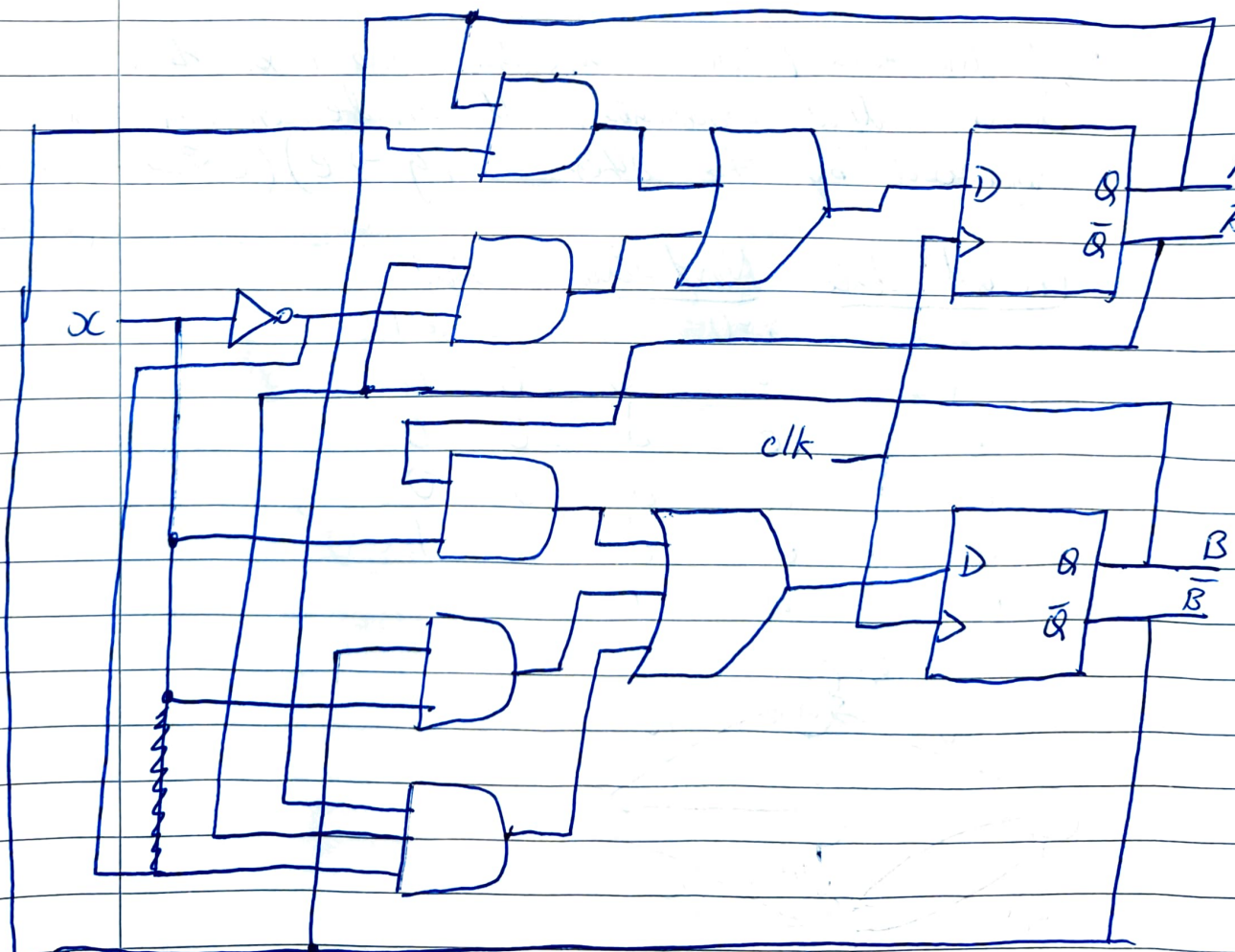


y

$$DA = A\bar{B} + B\bar{x}$$
$$DB = \bar{A}x + \bar{B}x + AB\bar{x}$$
$$y = \bar{B}x$$

(from equations)

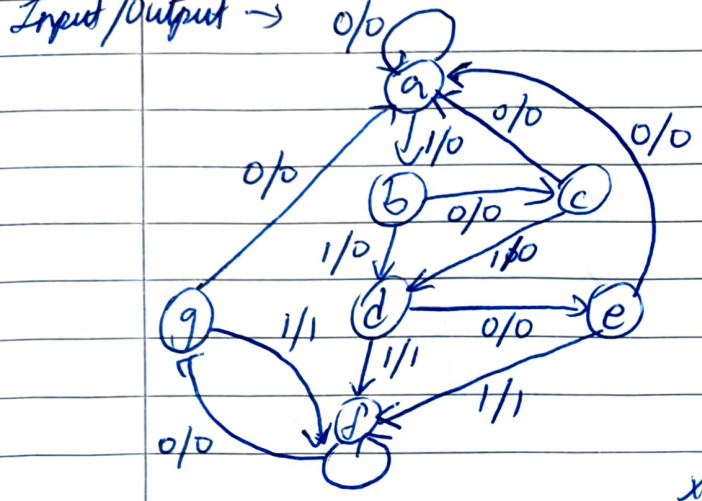
A(t)	B(t)	x	A(t+1)	B(t+1)	y
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	0	0
1	1	1	0	0	0



__/__/____

☆ State Reduction

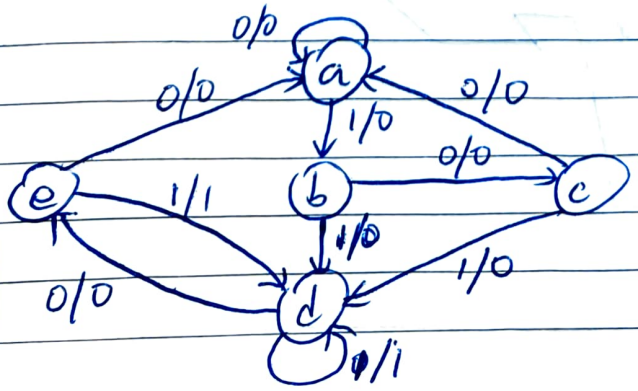
- Reduction in no. of flipflops in sequential circuit.
  (n flipflops produce 2^n states)



Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
[e	a	f	0	1]
f	g	f	0	1
x[g	a	~~f~~	0	1]

- If two states (present) have the same next state and same output combination, it can be removed and replaced by the other. $(g \to e)(d \Leftrightarrow f)(f\,X)(f \to d)$

Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
a	a	b	0	0
b	e	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

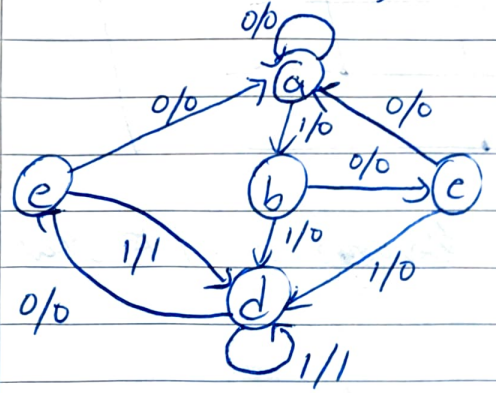


__/__/____

State Assignment

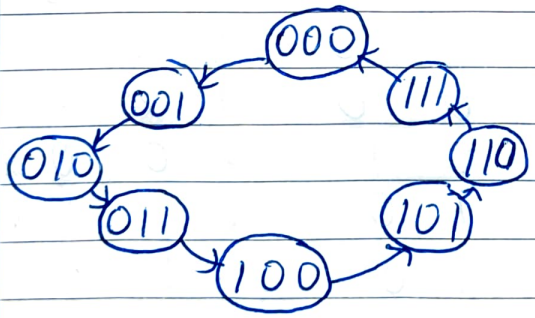
- To design a sequential circuit with physical components, it is necessary to assign unique binary-coded values to states. For m states, code must contain n bits such that $2^n \geq m$. Leftover states are treated as don't cares.

- Simplest way to code states is in binary counter order or in gray code (where only one bit changes between consecutive codes).



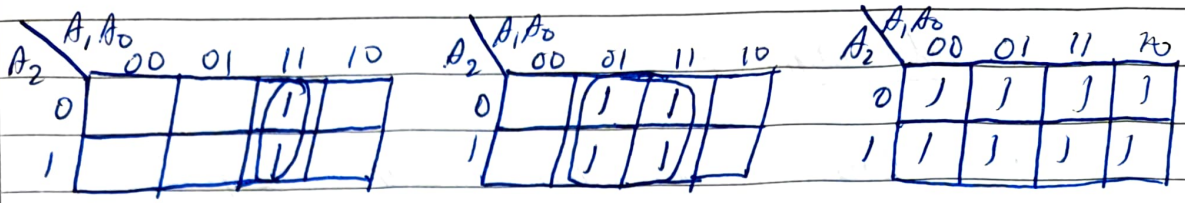
Present State	Next State		Output	
	x=0	x=1	x=0	x=1
000	000	001	0	0
001	010	011	0	0
010	000	011	0	0
011	100	011	0	1
100	000	011	0	1

3-Bit Counter

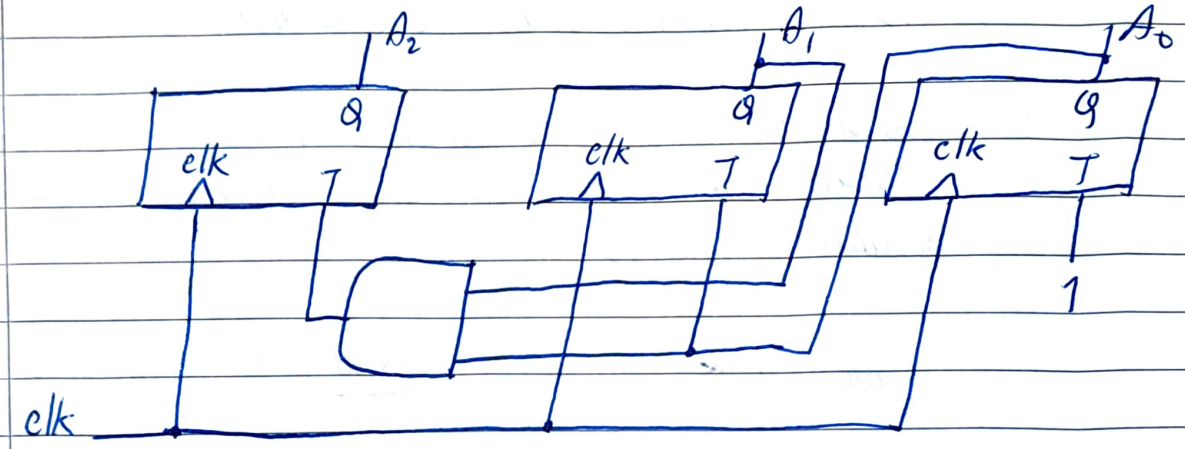


Present State $A_2 A_1 A_0$	Next State $A_2 A_1 A_0$	FF Inputs $TA_2\ TA_1\ TA_0$
000	001	001
001	010	011
010	011	001
011	100	111
100	101	001
101	110	011
110	111	001
111	000	111

→ current state



$TA_2 = A_1 A_0$ $\quad TA_1 = A_0$ $\quad TA_0 = 1$



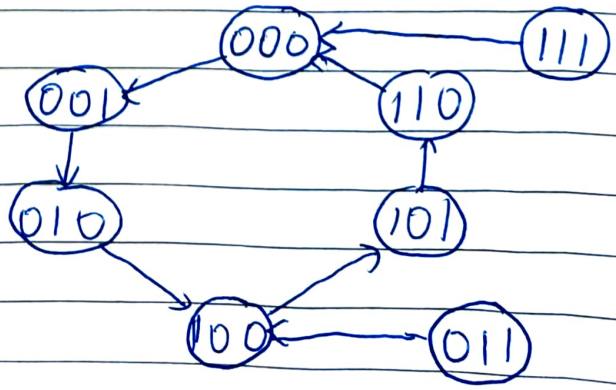
Ex: Design a 3-bit counter using JK FF that counts in the sequence 0, 1, 2, 4, 5, 6, 0, ...

Present State			Next State			FF Inputs					
A	B	C	A	B	C	JA	KA	JB	KB	JC	KC
0	0	0	0	0	1	0	x	0	x	1	x
0	0	1	0	1	0	0	x	1	x	x	1
0	1	0	1	0	0	1	x	x	1	0	x
1	0	0	1	0	1	x	0	0	x	1	x
1	0	1	1	1	0	x	0	1	x	x	1
1	1	0	0	0	0	x	1	x	1	0	x

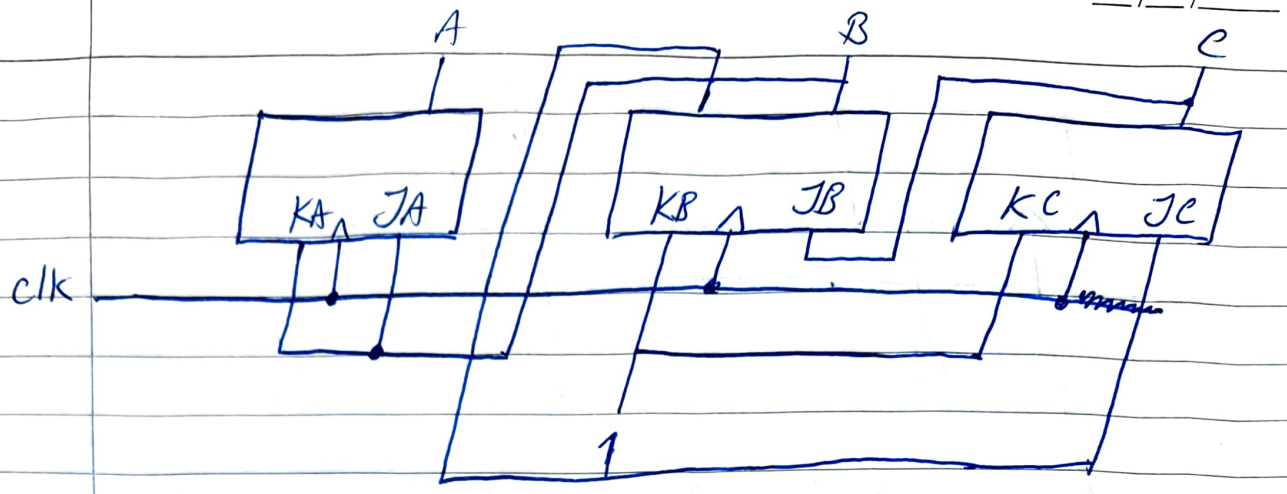
$JA = B$ $\quad KA = B$

$JB = C$ $\quad KB = 1$

$JC = \bar{B}$ $\quad KC = 1$



111 011
A B C A B C



- To check if it is self correcting, we got to find next state of leftover, if they transition to valid state.

3 → 011 (A B C), 7 → 111 (A B C)

~~Present State~~ ~~Next State~~ ~~FF Inputs~~

~~A B C~~ ~~A B C~~ ~~JA JB~~

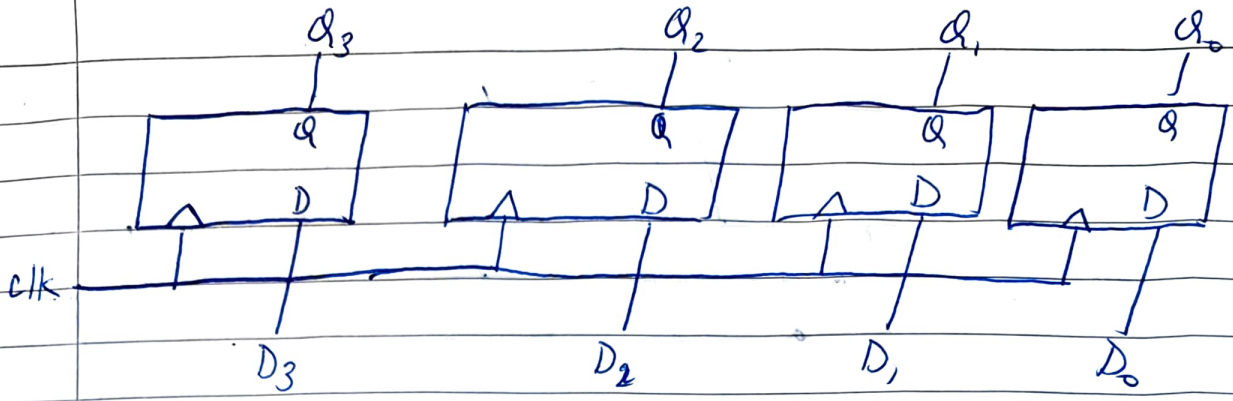
(from equation)

Present State			FF Inputs						Next State		
A	B	C	JA	KA	JB	KB	JC	KC	A	B	C
0	1	1	1	1	1	1	0	1	1	0	0
1	1	1	1	1	1	1	0	1	0	0	0

Registers

- A register includes a set of flip-flops which hold data. Gates determine new or transformed data to be transferred into FFs.

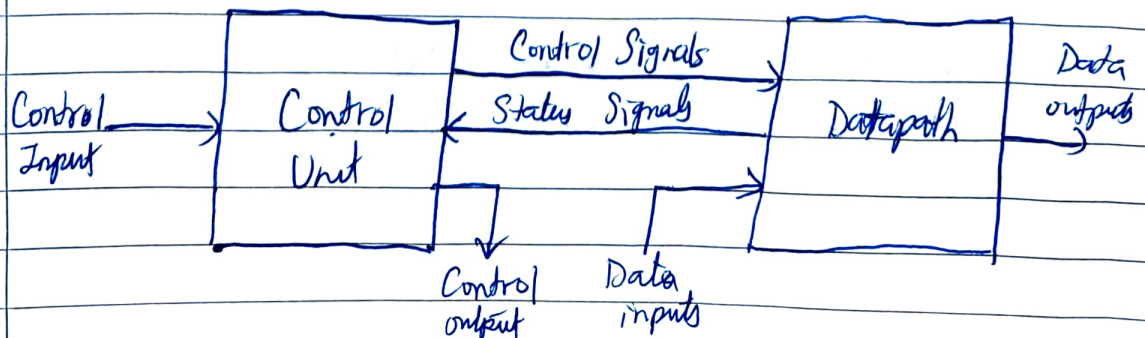
- A counter is a register that goes through a predetermined sequence of states upon application of clock pulses



- **Register with Parallel Load:** $C_{in} = \overline{Load} + Clock$
- When Load signal = 1, C_{in} = clock (register is clocked normally, new information transferred on positive edge trigger)
  When load signal = 0, C_{in} = 1 (no positive transitions on C inputs, contents of register unchanged).

☆ Register Transfers

- Most digital systems are divided into (a) datapath (which performs data processing operations) and (b) control unit (which determine sequence of operations).
- Control signals are binary signals that activate various data-processing operations. Control unit sends proper sequence of control signals to datapath.
- Datapath sends status bits to control unit, which describe state of datapath to define operation to be performed.



__/__/__

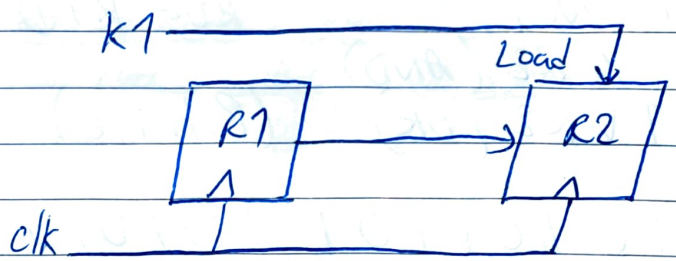
- Registers transfer operations are specified by (a) set of registers in system (b) operations performed on data and (c) control that supervises sequence of operations in system.

Part — Counter is a register that increments a number by 1. Single flipflop is a 1-bit register that can be set/cleared. Right shift register shifts data to the right.

- An elementary operation performed on data stored in registers → microoperation.

— Transfer microoperations (transfer data from one register to another)

$R2 \leftarrow R1$ (copy contents of R1 to R2)
$K1: R2 \leftarrow R1$ (if (K1 = 1), then $R2 \leftarrow R1$)



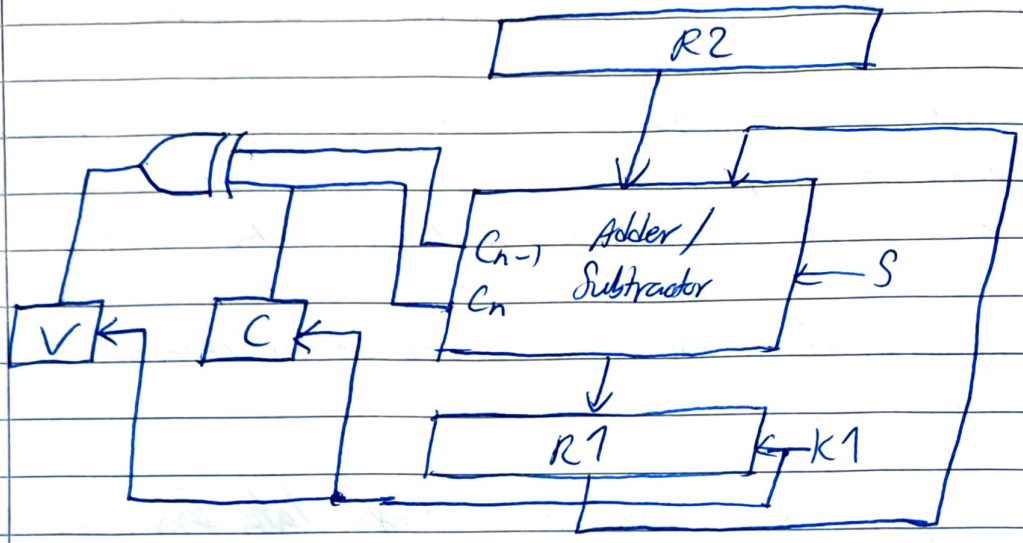
Note: Letters denote register, arrow denotes transfer of data, comma separates simultaneous transfers, [ ] specify address for memory

— Arithmetic Microoperations

- Select Input S selects operation in adder - subtractor circuit
- When $S=0$, $R1 \leftarrow R1 + R2$
  When $S=1$, $R1 \leftarrow R1 - R2$
- Control variable S selects operation, K1 loads result into R1.

__/__/____

- Overflow is transferred to FF V and output carry of MSB is transferred to FF C.



— Logical Microoperations (bit manipulation)

$R0 \leftarrow \overline{R1}$ (NOT) $\quad R0 \leftarrow R1 \oplus R2$ (XOR)
$R0 \leftarrow R1 \wedge R2$ (AND)
$R0 \leftarrow R1 \vee R2$ (OR) (Bitwise)

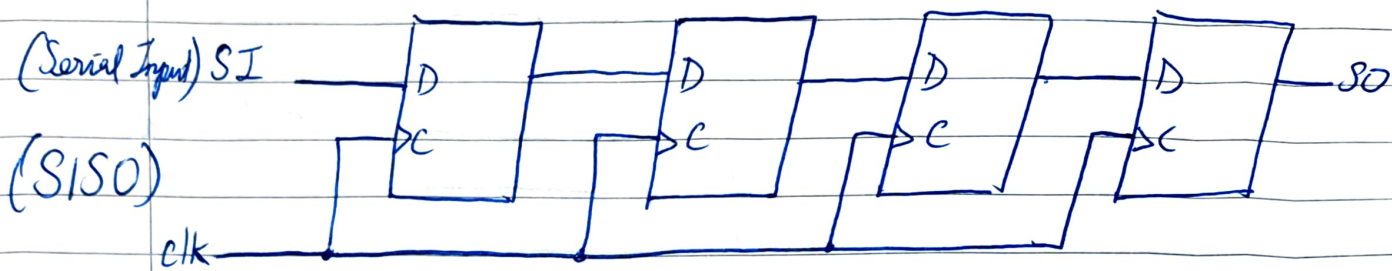
R1 → 10101101 10101011
R2 → 11111010 10010100
R1∧R2 → 10101000 10000000
R1∨R2 → 11111111 10111111

— Shift Microoperation

R2 → 10011110
R1 ← sl R2 ⇒ 00111100
R1 ← sr R2 ⇒ 01001111

☆ Shift Registers

- Register capable of shifting it's stored bits laterally, either in direction is called shift register.



- With parallel load,

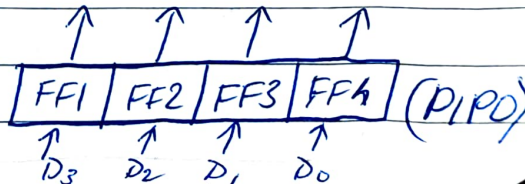
Shift	Load	Operation
0	0	No change
0	1	Load parallel data
1	x	Shift left

Bidirectional Shift Register

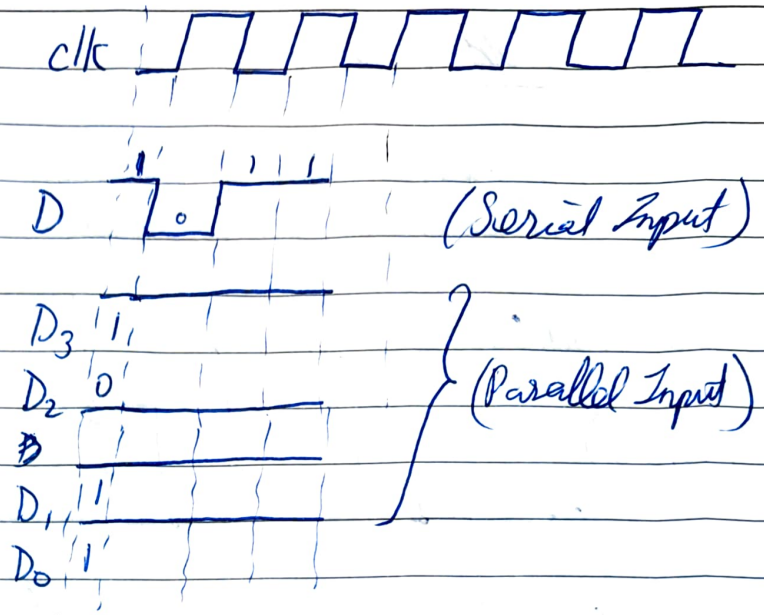
	Mode Control S_1	S_0	Operation
MUX select input	0	0	No change
	0	1	Shift left
	1	0	Shift right
	1	1	Parallel load

Note: Type of registers : Based on I/P and O/P : (a) SISO
(b) SIPO
(c) PISO
(d) PIPO

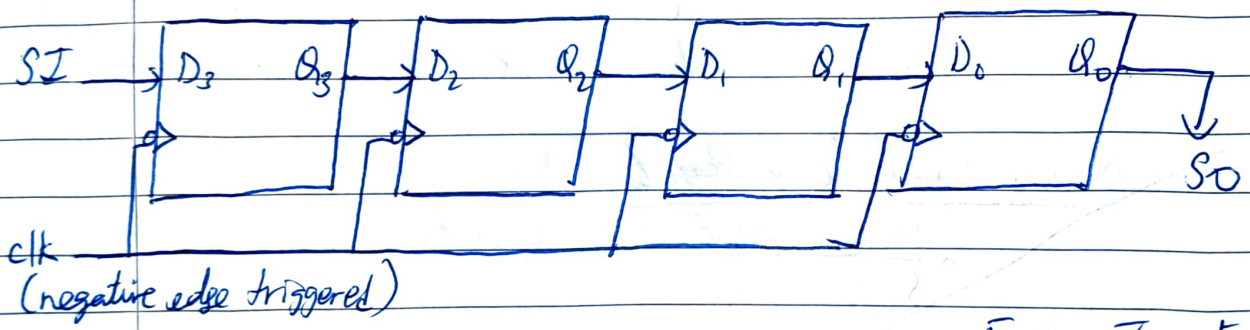
Based on application : (a) Shift registers
(b) Storage registers



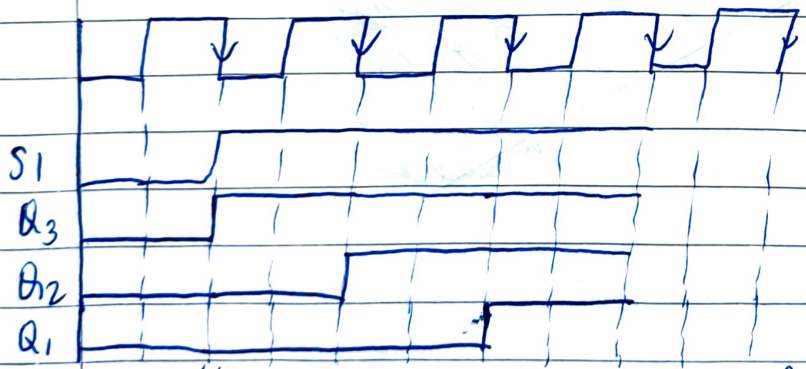
Note: Timing Diagram:



- SISO Shift Register



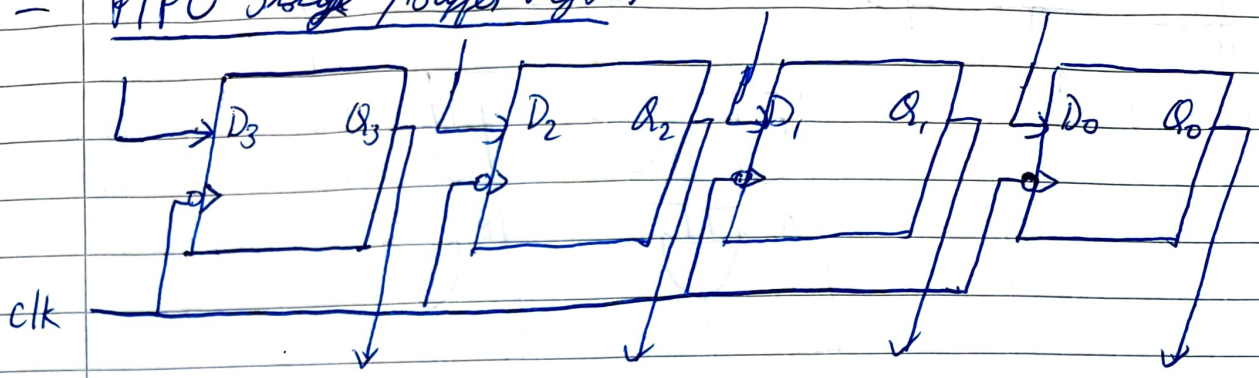
Ex:- To store 1111 in register,



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

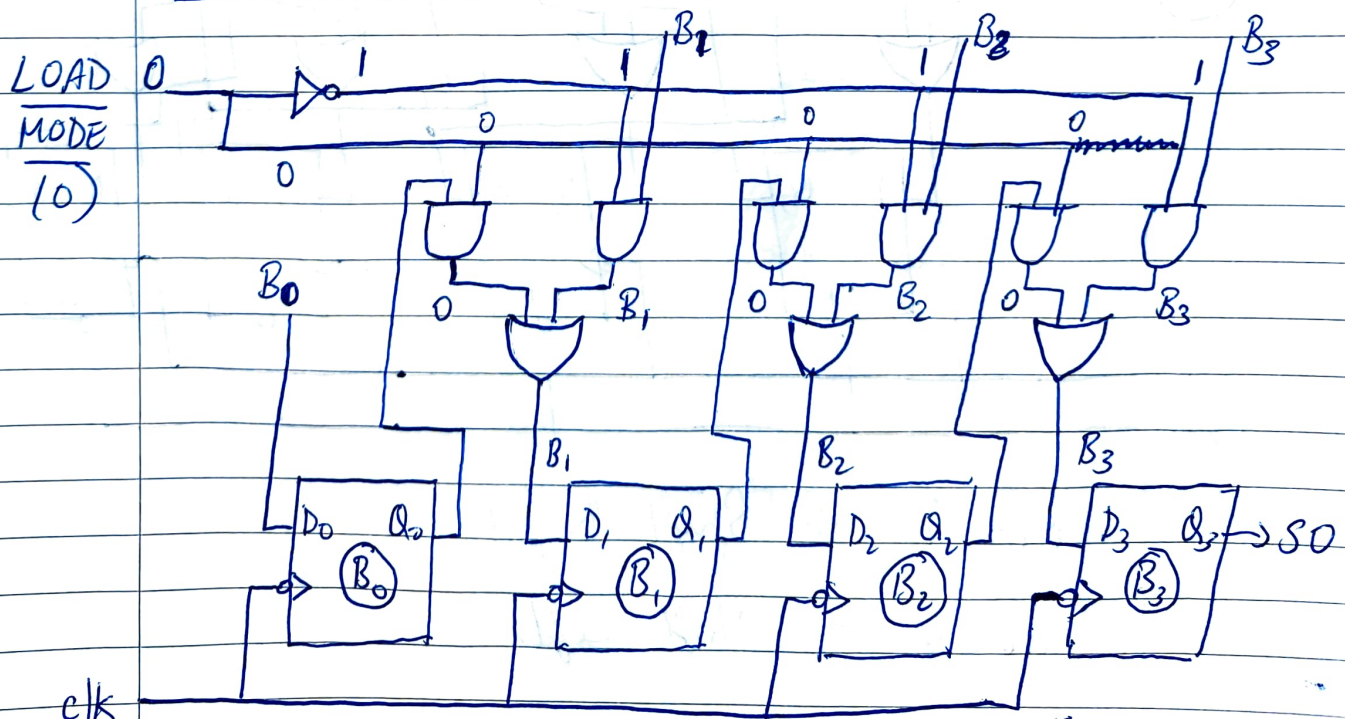
__/__/___

- PIPO Storage / Buffer Register

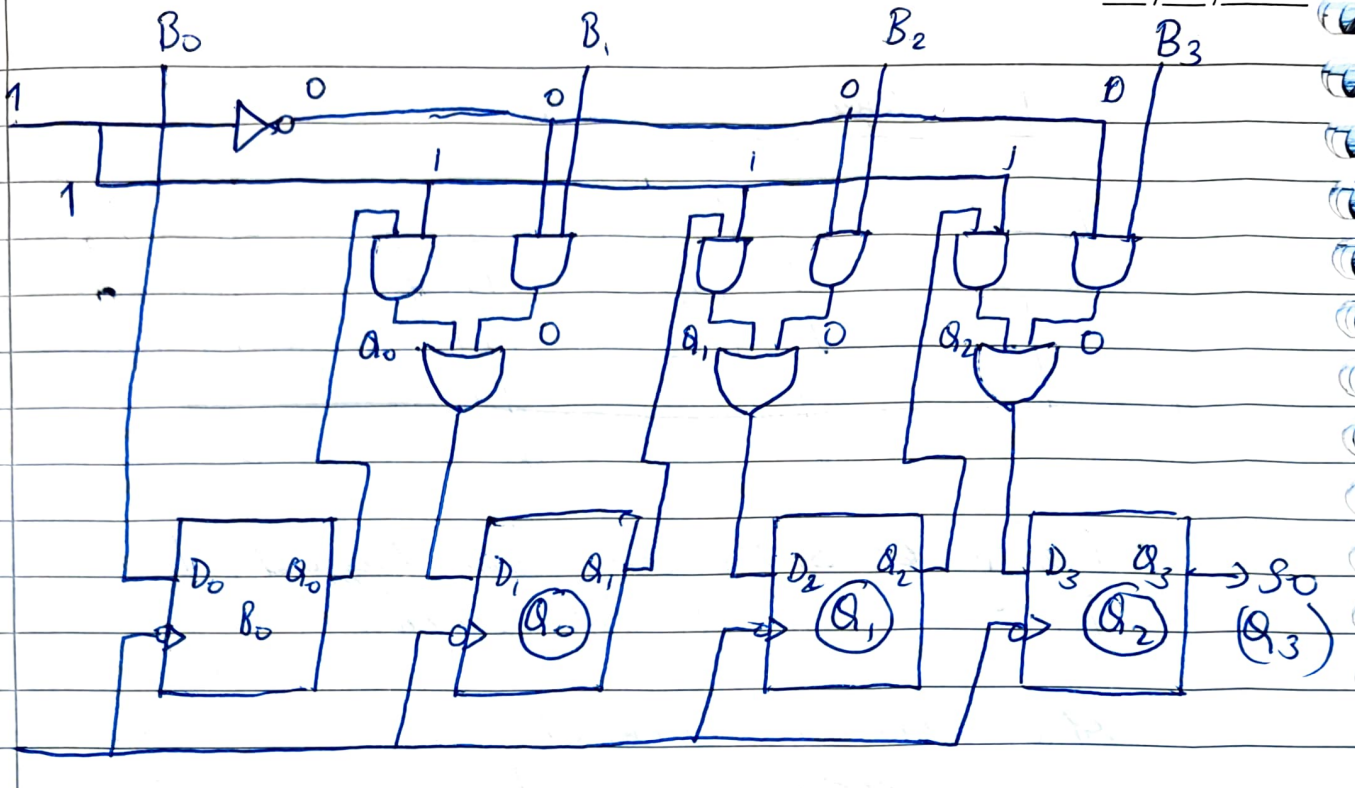


- Before negative clock pulse triggered, $D_3 = 1$, $D_2 = 0$, $D_1 = 1$, $D_2 = 1$ (as example)
- After clock pulse triggered, all D FFs store their respective values according to : $D = 0$, $Q_{n+1} = 0$ (clock = 1)
  $D = 1$, $Q_{n+1} = 1$
- After storing data, when clock = 0, $Q_{n+1} = Q_n$ irrespective of value of D, each FF give out respective output parallely.

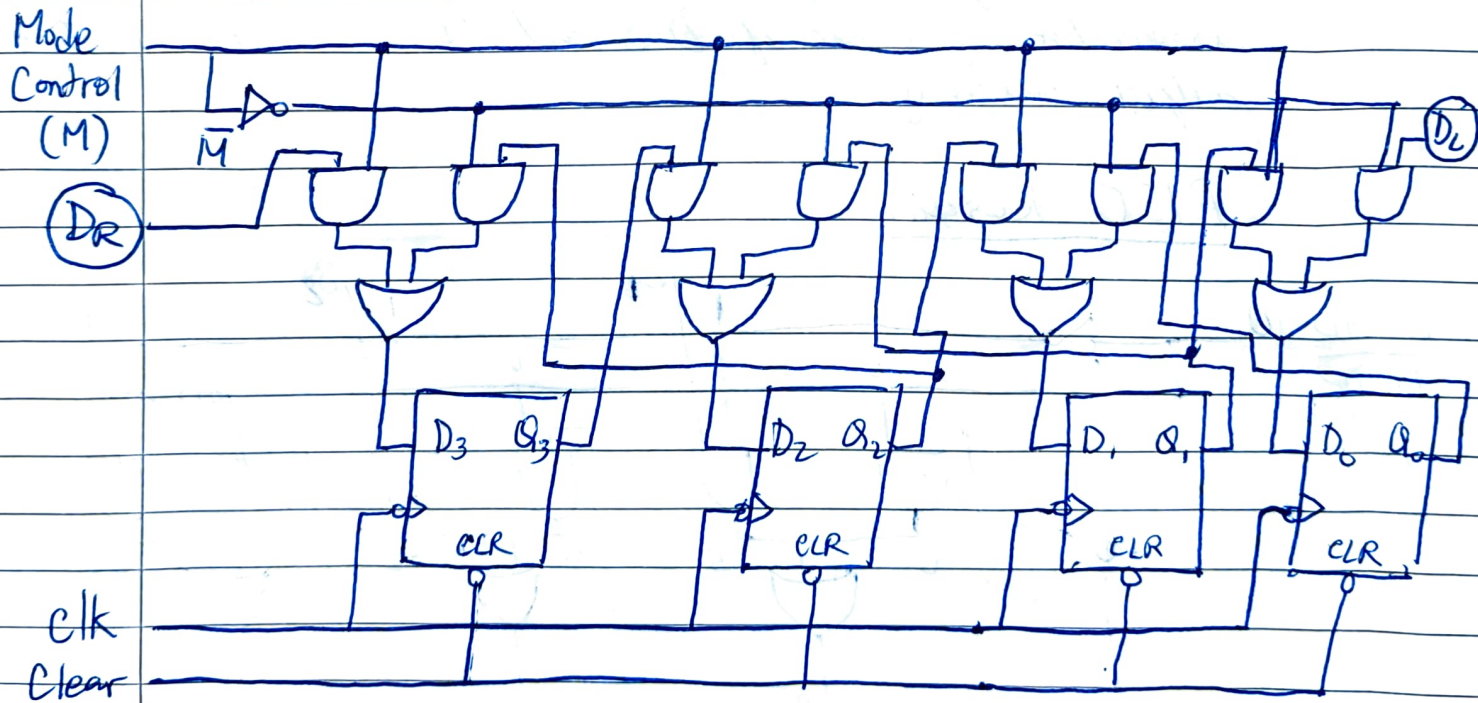
- PISO Register



0 1011 0



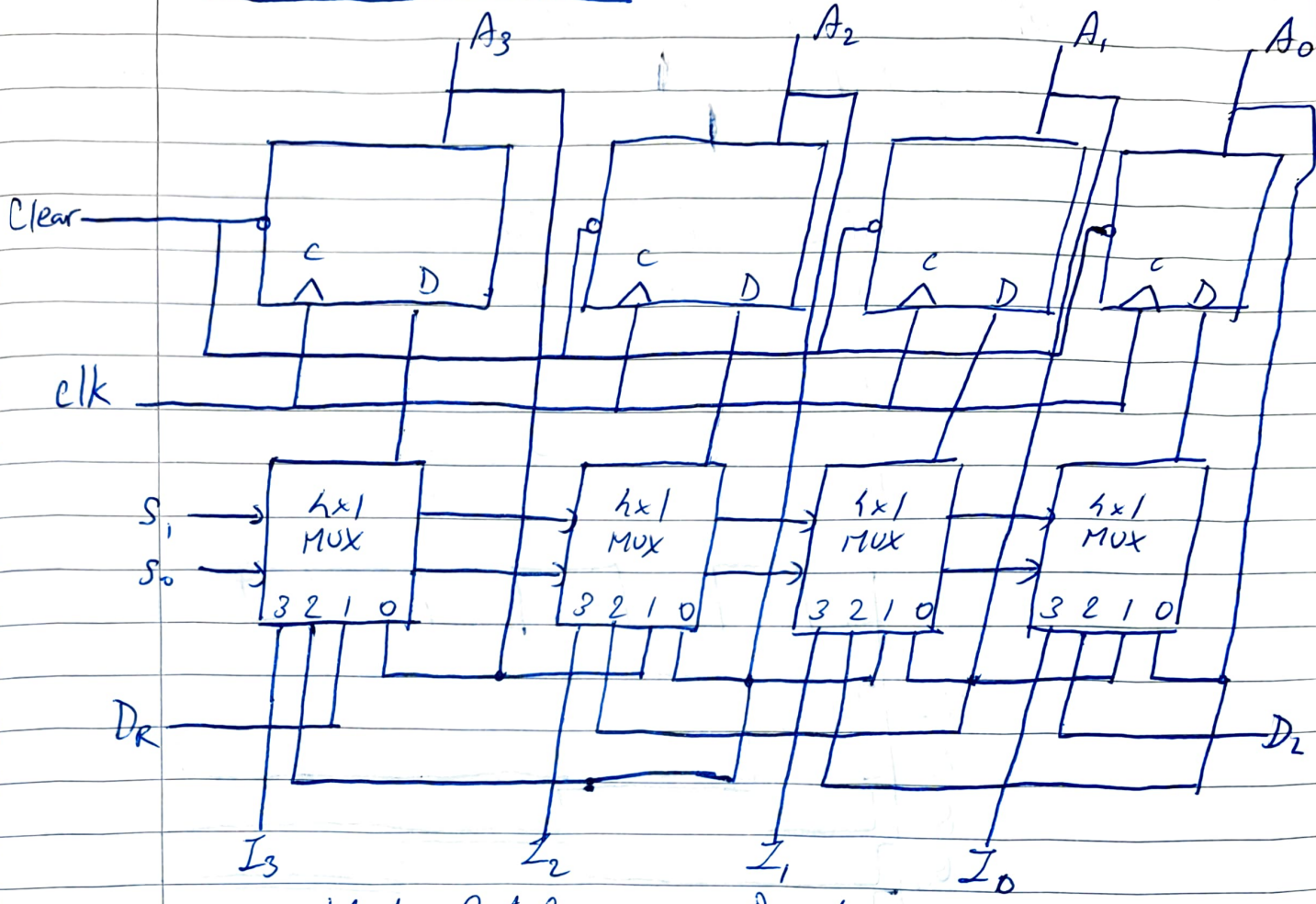
— Bidirectional Shift Register (Diagram)



M = 1 ⇒ Shift Right

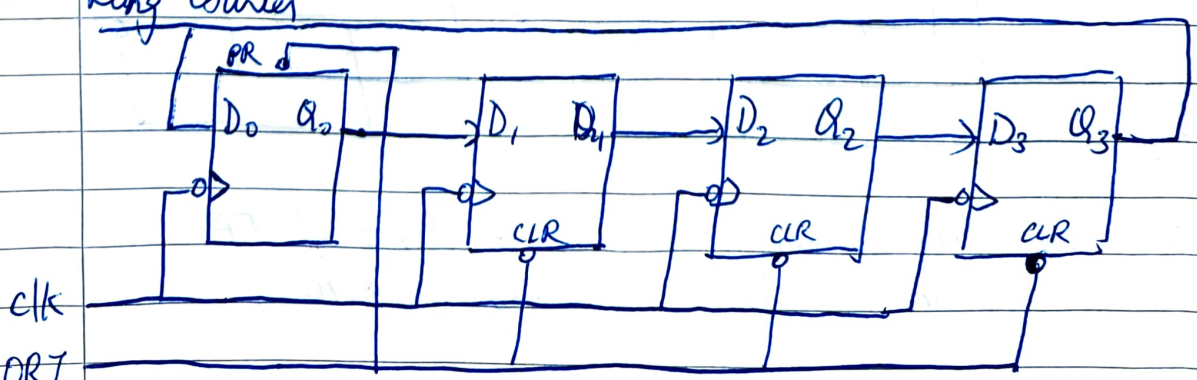
M = 0 ⇒ Shift Left.

– Universal Shift Register



Mode Control		Operation
S_1	S_0	
0	0	No change
0	1	Shift Right
1	0	Shift Left
1	1	Parallel load

– Ring Counter

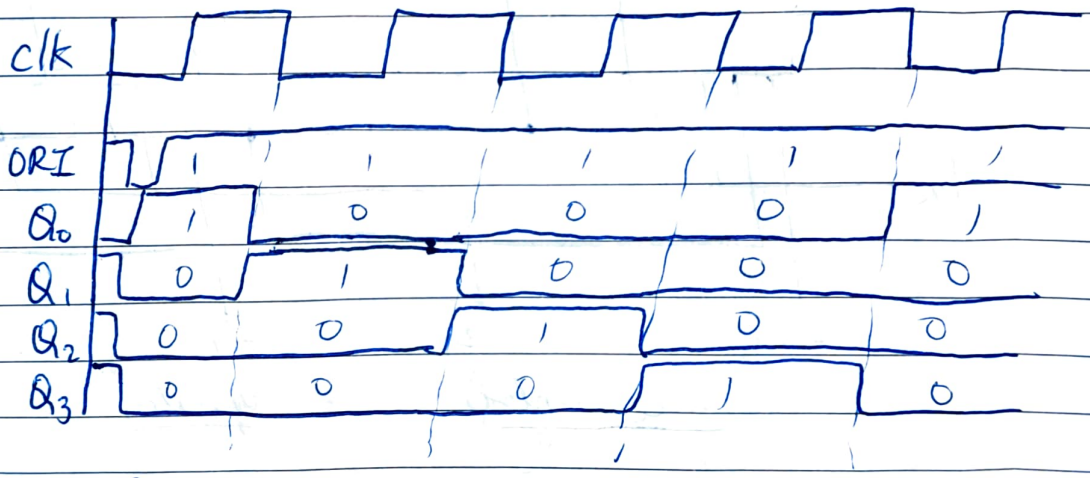


__/__/____

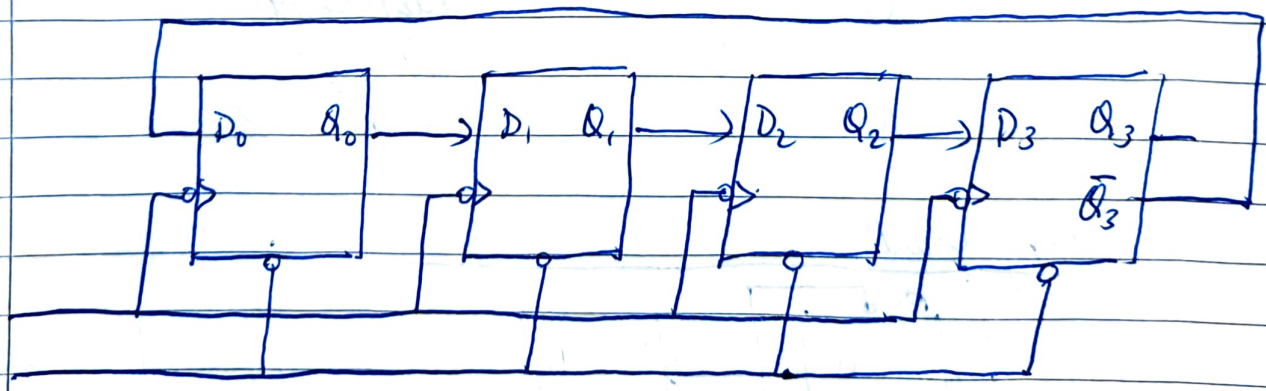
- Ring counter is a typical shift register with output of last FF connected to input of first FF.

  (PR (preset) = 0, Q = 1)
  (CLR = 0, Q = 0) (ORI → overriding input)

ORI	CLK	Q_0	Q_1	Q_2	Q_3
0	X	1	0	0	0
1	↓	0	1	0	0
1	↓	0	0	1	0
1	↓	0	0	0	1
1	↓	1	0	0	0



— Johnson's Counter



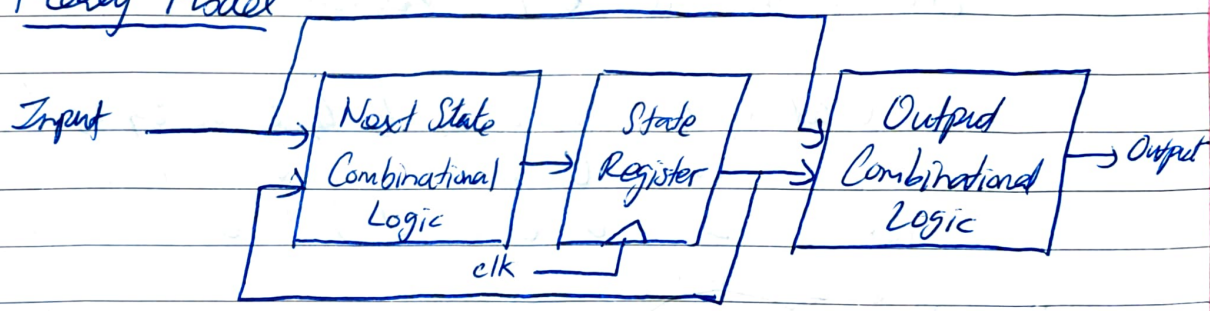
Note: In ring counter, no. of states = no. of FF (4)
In Johnson counter, no. of states = 2× no. of FF (8)

__/__/____

CLR	CLK	Q_0	Q_1	Q_2	Q_3
0	x	0	0	0	0
1	↓	1	0	0	0
1	↓	1	1	0	0
1	↓	1	1	1	0
1	↓	1	1	1	1
1	↓	0	1	1	1
1	↓	0	0	1	1
1	↓	0	0	0	1
1	↓	0	0	0	0

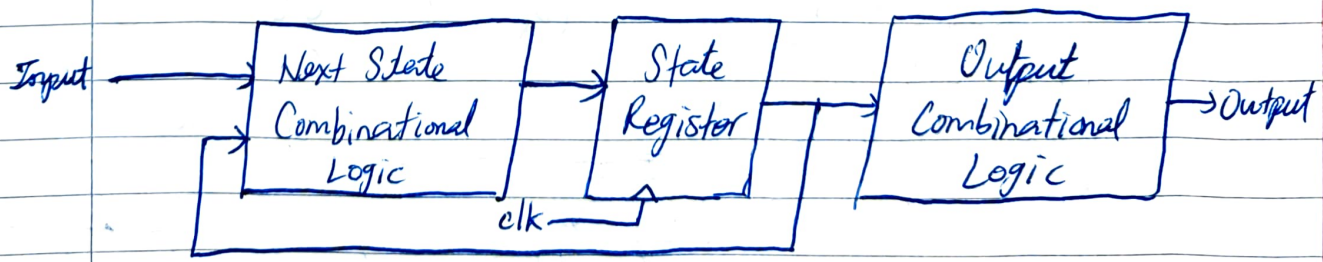
☆ Finite State Machines (FSM)

– Mealy Model



- Here, output is function of both present state and input

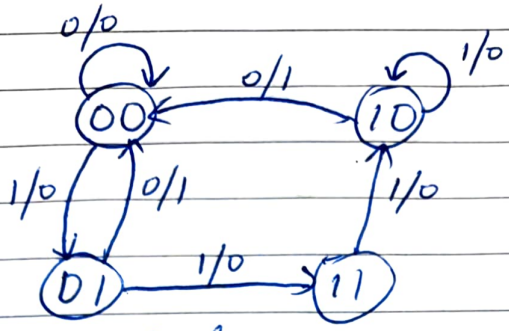
– Moore Model



- Here however, output is function of only present state.

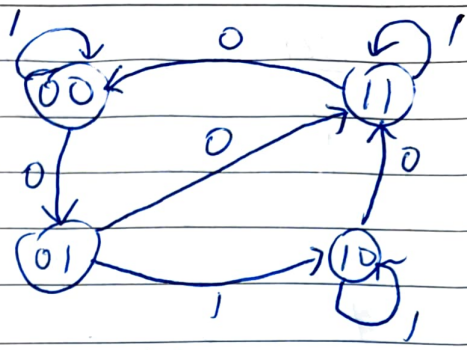
__/__/____

Example:



Mealy

(Both input and output are shown along directed lines)



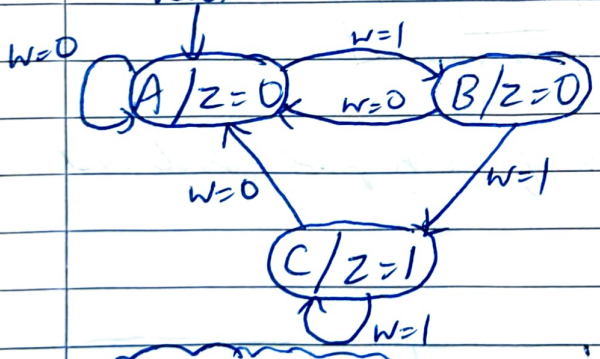
Moore

(Input → along directed lines)
(Output → FF states)

Ex:- Design a circuit having one input w and one output z, where all changes in circuit occur on positive edge of clock signal. Output $z=1$ if during last two clock cycles, $w=1$, otherwise $z=0$.

Clock Cycle: $t_0\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_8\ t_9\ t_{10}$

w:	0	1	0	1	1	0	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	0

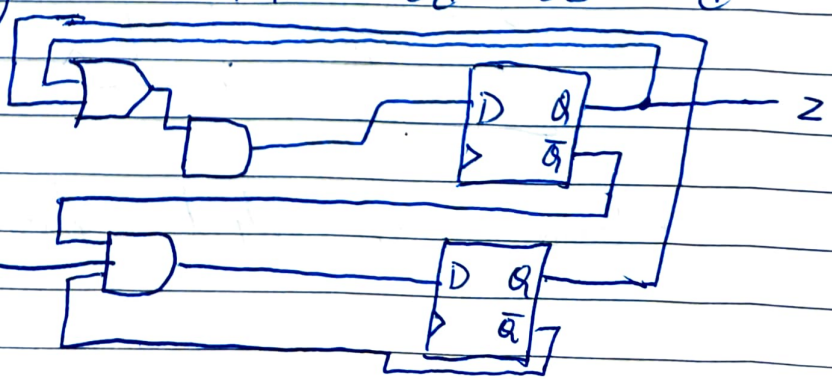


Present State		Next State		Output
	Y_2Y_1	$w=0$ Y_2Y_1	$w=1$ Y_2Y_1	z
(A)	00	00	01	0
(B)	01	00	10	0
(C)	10	00	10	1
	11	dd	dd	d

$Y_1 = w\bar{Y}_1\bar{Y}_2$

$Y_2 = w(Y_1 + Y_2)$

$z = Y_2$



(To be continued)

__/__/____

DSD (MODULE 4)

- Verilog for D FF

```
module flipflop (D, Clock, Q);
    input D, Clock;
    output reg Q;
    always @(posedge Clock)
        Q = D;
endmodule
```

☆ Blocked & Non-Blocking Assignment

- Blocked (=)

- Executes sequentially
- a = b
  c = a (c gets new value of a)
- Commonly used in always block

- Non-Blocking (<=)

- Executes simultaneously
- a <= b
  c <= a (c gets old value of a)
- Commonly used in sequential logic in always block with clock.

☆ Synchronous & Asynchronous Clear

- Synchronous Clear

- Clear signal is synched with clock signal.

```
always @ (posedge clk) begin
    if (reset)
    ....
```

- Asynchronous Clear

- Clear signal not synched, can occur any time.

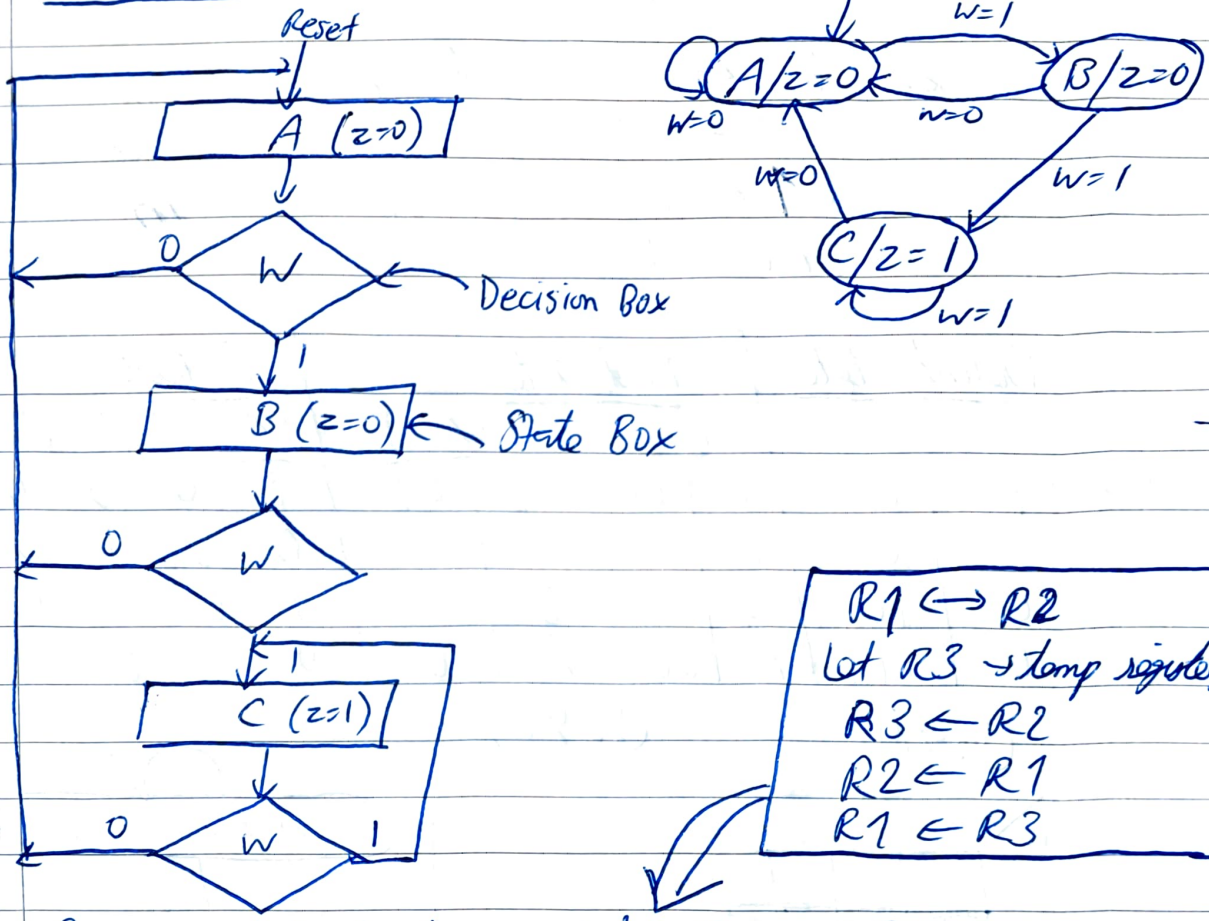
```
always @ (posedge clk or posedge reset)
begin
    if (reset)
    ....
```

__/__/____

- More predictable
- Preferred in designs where precise timing is crucial.

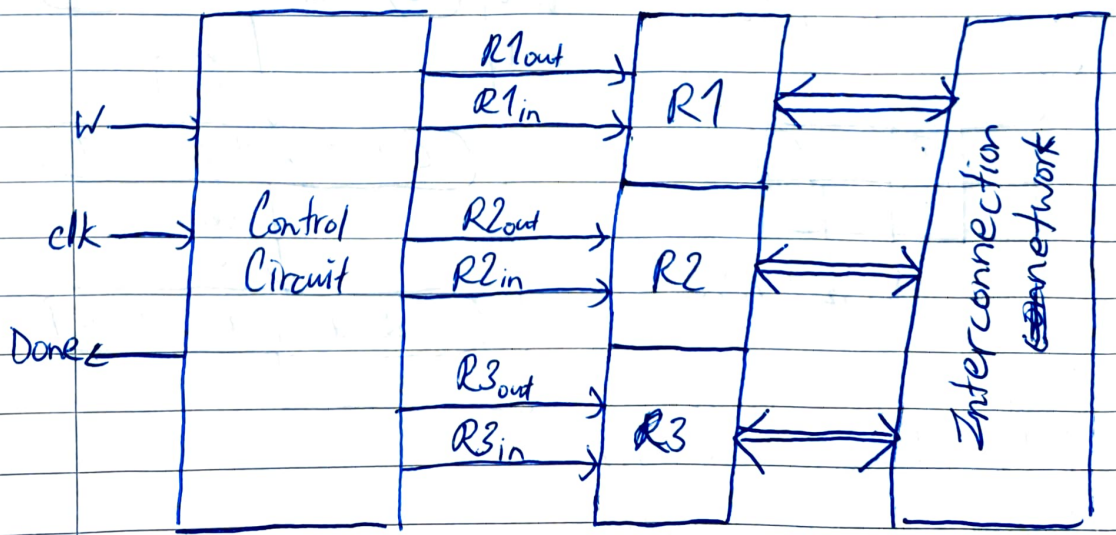
- less predictable
- Used when immediate reset is necessary.

☆ ASM (Arithmetic State Machine) Chart



R1 ⟷ R2
Let R3 → temp register
R3 ← R2
R2 ← R1
R1 ← R3

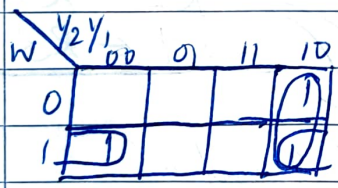
Ex: Swap contents of two registers R1 and R2.



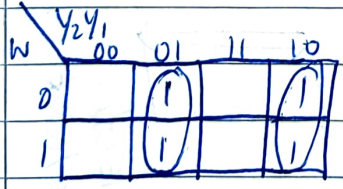
__/__/____

- Inputs : w, Clock
  Outputs : $R1_{out}$, $R1_{in}$, $R2_{out}$, $R2_{in}$, $R3_{out}$, $R3_{in}$, Done.
- Rk_{out} signals causes contents of Rk → interconnection network
  Rk_{in} signals cause contents of interconnection network → Rk.
- Hence $R3 \leftarrow R2$ $(R2_{out} = 1, R3_{in} = 1)$
  $R2 \leftarrow R1$, $(R2_{in} = 1, R1_{out} = 1)$
  $R1 \leftarrow R3$ $(R3_{out} = 1, R1_{in} = 1)$
  Done = 1

Present State		Next State w=0		Next State w=1		Output					
						$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$
(No transfer) A	00	A	00	B	01	0	0	0	0	0	0
B	01	C	10	C	10	0	0	1	0	0	1
C	10	D	11	D	11	1	0	0	1	0	0
D	11	A	00	A	00	0	1	0	0	1	0
	$(Y_2 Y_1)$		$(Y_2 Y_1)$								

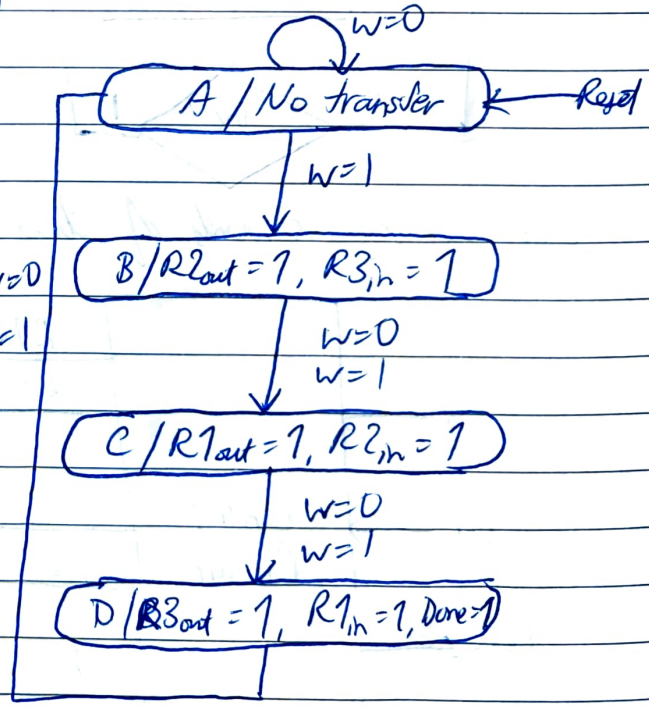


$Y_1 = w\bar{y}_1 + \bar{y}_1 Y_2$



$Y_2 = Y_1 \bar{Y}_2 + \bar{Y}_1 Y_2$

$R1_{out} = R2_{in} = \bar{Y}_1 Y_2$
$R1_{in} = R3_{out} = Done = Y_1 Y_2$
$R2_{out} = R3_{in} = Y_1 \bar{Y}_2$



DSD (MODULE 6)

– Bit-Counting Circuit

- To count no. of bits in register A having value of 1.

Pseudocode:

```
B = 0
while A ≠ 0 do
    if a0 = 1 then
        B = B + 1 ;
    end if ;
    Right shift A ; (Logical >>)
end while ;
```