Digital Logic Notebook

Dr. Iyad Ja'afar

* Analog to Digital Conversion Process (A/D):

Analog (continuous) —1. Sampling→ Digital (discrete).
(taking finite number of points).

* 2. Trancation:- (اختصار، عدد الـDigits حسب الذاكرة)

0.001 → 0.00
0.018 → 0.01
0.014 → 0.01

quantization:- I can't move continuously but I move in quantum levels.

Encoding

	multi-valued	Binary-valued
-0.9	0	000
-0.6	1	001
-0.3	2	010
0.0	4	011
0.3	4	100
0.6	5	101
0.9	6	110
	* needs 1 wire. * we can't correct the error because we have too many choices	* needs 3 wires * we can correct the error because we have 2 choices only (0, 1).

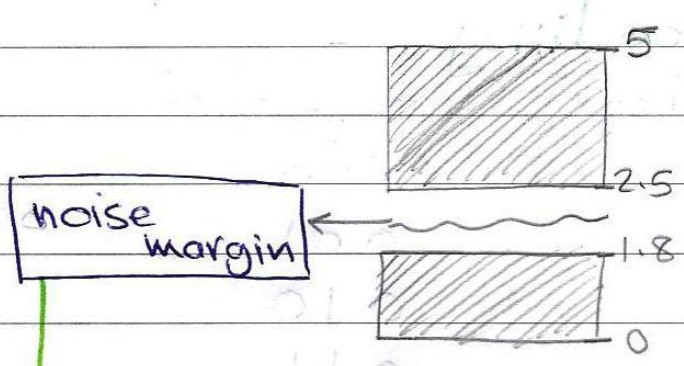
* Binary valued systems has a higher immunity to noise.

* Digital doesn't mean Binary, Digital may be:-
  - Binary valued
  - Multi-valued

1) CPU:

→ In CPU (0,1) are represented by Voltage.

ex: 0 → 0-1.8 volts
1 → 2.5-5 volts



↳ we can't guarantee what the value will be (0,1)

2) RAM (DRAM):

→ 0,1 are represented by the charge stored in capacitors.

3) Hard Disk :-

→ by magnatic surface. (counter clockwise)

ex:- when it's magnetised CCW → 0

= = CW → 1

4) CD-ROM :-

→ by reflection of light (burned surface).

* Numbering Systems :-

→ Decimal :

- radix ≡ base = 10
- Allowed numbers = {0, 1, ---, 9}

ex: $710 = 0 \times 10^{0} + 1 \times 10^{1} + 7 \times 10^{2}$ → Possition

* possitional +ve radix.

in general for any possitional +ve radix numbering system with base (r)

1. $A_{n-1}\ A_{n-2}\ \cdots\ A_0 \cdot A_{-1}\ A_{-2}\ A_{-3}\ \cdots\ A_{-m}$

most significant digit (MSD) — n — radix Point — m — least significant digit (LSD)

2. Allowed numbers to be used [0, r-1]
3. The decimal values of any number is given by:

$$\text{Value} = \sum_{i=0}^{n-1} A_i r^i + \sum_{j=-m}^{-1} A_j r^j$$

ex: $(213)_{4} = (39)_{10}$

base

$2*4^2 + 1*4^1 + 3*4^0 = 39$

→ Binary:

- r = 2
- Allowed numbers = {0, 1}

ex:- $(1011.10)_2 = ?! \ (11.5)_{10}$

3 2 1 0 −1 −2

$= 1*2^3 + 1*2^1 + 1*2^0 + 1*2^{-1} = 11.5$

- Binary digit ≡ bit.

→ Octal:

- $r = 8 \equiv 2^3$
- {0, 1, ... , 7} = [0, 7]

ex:- $(31)_8 = (25)_{10}$

$3*8^1 + 1*8^0 = 25$

→ Hexadecimal:-

- r = 16
- [0, 15]

A → 10
B → 11
C → 12
D → 13
E → 14
F → 15

ex:* $(1453)_{16} = (\ ?\ )_{10}$

$= 3*1 + 5*16 + 4*16^2 + 1*16^3$

* $(F3)_{16} = (243)_{10}$

$15*16 + 3 = 243$

* $(D.3)_{16} = (13\frac{3}{16})_{10}$

$13*1 + 3*16^{-1} = 13\frac{3}{16}$

Decimal	Binary	Octal	Hex
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

* Conversion between numbering systems:-

1) convesion from base (r) into decimal.

* Value = $\sum_{i=0}^{n-1} A_i r^i + \sum_{j=-m}^{-1} A_j r^j$

2) conversion from base (10) to base (r).

→ for integer part

* use repeated integer division on Succesive quotients with base (r) & save the remainders until the qaotient is zero.

* read the remainders from bottom to top.

ex: $(153)_{10} = (231)_8$

	Quotient الناتج	Remainder الباقي	
153 ÷ 8 =	19	1	LSD
19 ÷ 8 =	2	3	↑
2 ÷ 8 =	0	2	MSD

$(73)_{10} = (1001001)_2$

73 ÷ 2 =	36	1
36 ÷ 2 =	18	0
18 ÷ 2 =	9	0
9 ÷ 2 =	4	1
4 ÷ 2 =	2	0
2 ÷ 2 =	1	0
1 ÷ 2 =	0	1

→ for the fractional part:-

* use repeated multiplication with base (r) on the fractional part, save the integer part, stop when the fraction is zero.
* Read the answer (integer part) from top to bottom

ex:- $(0.375)_{10} = (0.011)_2$

$0.375 * 2 = 0.75$

$0.75 * 2 = 1.5$

$0.5 * 2 = 1.00$ stop

$(0.011)_2$

$(41.6875)_{10} = (101001.1011)_2$

$41 \div 2 =$	20	1
$20 \div 2 =$	10	0
$10 \div 2 =$	5	0
$5 \div 2 =$	2	1
$2 \div 2 =$	1	0
$1 \div 2 =$	0	1

$0.6875 * 2 = 1.375$

$0.375 * 2 = 0.75$

$0.75 * 2 = 1.5$

$0.5 * 2 = 1.00$

ex:- $(0.515)_{10} = (0.407)_8$ (3 digits)

$0.515 * 8 = 4.104$

$0.104 * 8 = 0.832$

$0.832 * 8 = 6.656$

$0.656 * 8 = 5.248$

→ 0.4065

ex:- $(245)_{10} = (F5)_{16}$

$245 \div 16 =$	15	5
$15 \div 16 =$	0	15

* Conversion between Octal & Binary:-

$(3 \times 4)_{10} = (12)_{10}$

$(3 \times 4)_{8} = (14)_{8}$

$(5+6)_{10} = (11)_{10}$

$(5+6)_{8} = (13)_{8}$

#1 * Octal $\rightleftharpoons$ Decimal $\rightleftharpoons$ Binary

$(71)_8 \rightarrow (?)_2$

$\equiv (71)_8 \rightarrow (57)_{10} \rightarrow (111001)_2$

#2 * from binary to octal $\rightarrow 8 = 2^3$

3 : 1

bit grouping in 3

ex: $(011\ 000\ 011 . 001\ 100)_2$

$(303.14)_8$

* from Octal to binary

ex:- $(7563.1102)_8 \rightarrow (?)_2$

$(\underset{7}{111}\ \underset{5}{101}\ \underset{6}{110}\ \underset{3}{011} . \underset{1}{001}\ \underset{1}{001}\ \underset{0}{000}\ \underset{2}{010})_2$

* from Hex to binary $16 = 2^4$ (4:1) bit hex

ex: $(F13.67)_{16} \rightarrow (?)_2$

$$(\underset{15}{1111}\ \underset{1}{0001}\ \underset{3}{0011} . \underset{6}{0110}\ \underset{7}{0111})_2$$

ex: $(011,011,0001.1101,1110)_2 = (\ ?\ )_{16}$

$(371.DC)_{16}$

$(001,101,110001.110,111,0)_2 = (\ ?\ )_8$

$(1561.67)_8$

$(D041.32)_{16} \longrightarrow (?)_2$

$= (\overline{1101}\ \overline{0000}\ \overline{0100}\ \overline{0001} . \overline{0011}\ \overline{0010})_2$

D 0 4 1 3 2

$= (150101.144)_8$

* Binary Codes:-
  - Numeric Codes
  - non-numeric codes.

- if we have (M) items, number of bits that are required to encode the items uniquely

$$n = \lceil \log_2 M \rceil$$ → ceil = أكبر عدد صحيح أكبر من أو يساوي القيمة

ex: $n = \lceil \log_2 67 \rceil = 7$ bits.

- if we have n bits, then the maximum number of items that can be encoded is:-

$$M \leq 2^n$$

#1: Binary coded Decimal (BCD)

* numeric.

→ $(23)_{10} = (10111)_2$

→ $(23)_{10} = (\underbrace{0010}_{2}\ \underbrace{0011}_{3})_{BCD}$

→ $(1000\ 0111)_{BCD} = (87)_{10}$

→ $(1000\ 0111)_2 = (135)_{10}$

#2: ASCII

* non-numeric
* 7 bits for encoding.
* $2^7 = 128$ characters (items).
  - ↳ letters (A-Z), (a-z)
  - ↳ numbers (0-9)
  - ↳ special characters → Printable (و، : /)...
    - ↳ non-Printable (space, enter, ---)

* 2 (ASCII) + 3 (ASCII) ≠ 5 (ASCII).

#3: UNI CODE :-

* 16 bits
* non-numeric.
* $2^{16} = 64 * 1024$

ex:- '0' → $(30)_{16} = 0110000$

#4: Gray code :-

* non-numeric.
* there's only (1-bit) change difference between adjacent codes.

ex:-

	binary	gray Scale.
orange	00	00
apple	01	01
melon	10	11
Strawberry	11	10

كل خانة متجاورة مختلفة في 1-bit فقط.

#5: Parity Code:-

* non-numeric
* extra bit(s) that's added to the original data to aid error detection.
* even / odd Parity.

ex:- $(1100)_2$

	even (1bit)	odd (1bit).
1100 →	01100	11100
1000 →	11000	01000

* يتم إضافة (1,0) في الـ bit الإضافية بحيث يبقى عدد الـ (1s) = even أو odd حسب النوع.

* عندما تصل البيانات للمستخدم ، إذا كان عدد الـ (1s) even ونوع البيانات odd سيكتشف وجود خطأ لكنه دون تحديد مكانه.

ex:- $(365)_r = (194)_{10}$

$3r^2 + 6r + 5 = 194$

$3r^2 + 6r - 189 = 0$

$(r-7)(r+9) = 0$

$\therefore \boxed{r = 7}$, ~~-9~~ نهمل

CHapter #2: combined logic circuit.

- logic circuits are the circuits that manipulates binary info.
- circuit → transistors, interconnects, resistors.
- Basic circuit → logic gates.
- each logic gate implement a basic logic opperation.

* Binary logic:

→ deals with binary variables that may take 2 values only (0,1)/(low, hight)/(on,off)/(true,false).

x, y, z .. binary variables

→ Binary operators:

1. AND

* symbol : '.' (dot

x.y ≡ x AND y

x.y ≡ xy (x, y are differant variables)

* Possible combinations

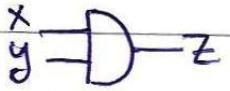
0.0 = 0

0.1 = 0

1.0 = 0

1.1 = 1

* AND Gate



* truth table

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

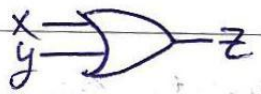
2. OR

* symbol = '+'

$z = x + y$

$0 + 0 = 0$
$1 + 0 = 1$
$0 + 1 = 1$
$1 + 1 = 1$

* OR Gate



3. NOT

* symbol : (~ , ' , ¯)
or or

$z = \bar{x}$ or x' or $\sim x$

$\bar{0} = 1$
$\bar{1} = 0$

* NOT Gate

(1 input, 1 output).

* (Boolean/logic) (expressions/functions)

* expressions that's formed by binary variables, constants (0, 1), binary operators & panthesis.

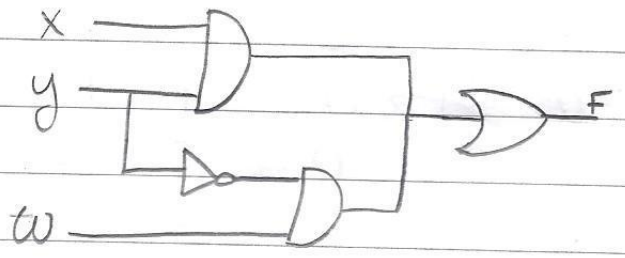
ex:- $f(w, x, y) = x \cdot y + w \cdot \bar{y}$

→ truth table

	w	x	y	F
Combinations	0	0	0	0
	0	0	1	0
	0	1	0	0
	0	1	1	1
& Possible	1	0	0	1
	1	0	1	0
	1	1	0	1
	1	1	1	1

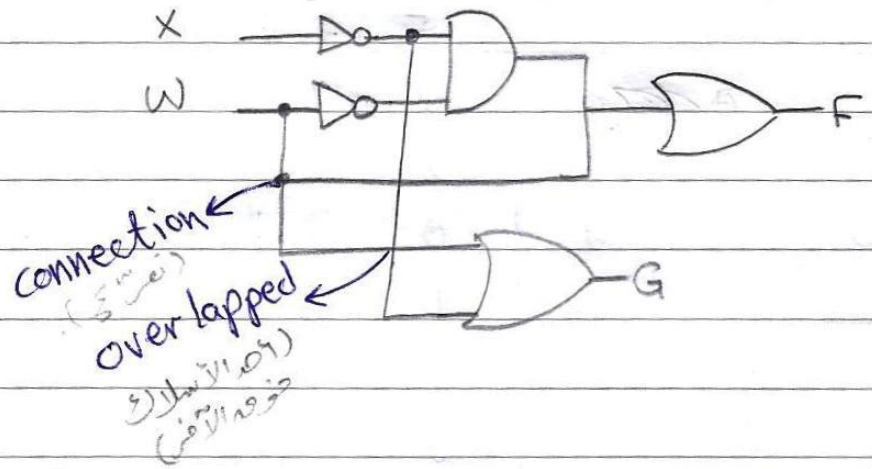
→ logic digram :- (ways to express logic functions).

ex:- $f(x, y, w) = x \cdot y + w \cdot \bar{y}$



ex:- consider $f(w,x,y) = \bar{w}\bar{x} + w$ & $G(w,x,y) = w + \bar{x}$

a) draw the logic diagram for the 2 functions

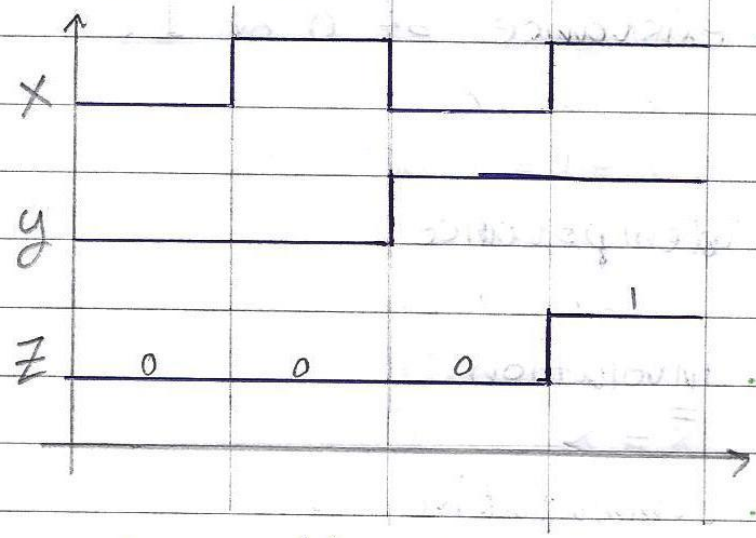


b) derive the truth table (t.t) for F & G

x	w	y	G	F
0	0	0	1	1
0	0	1	1	0
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	1	0

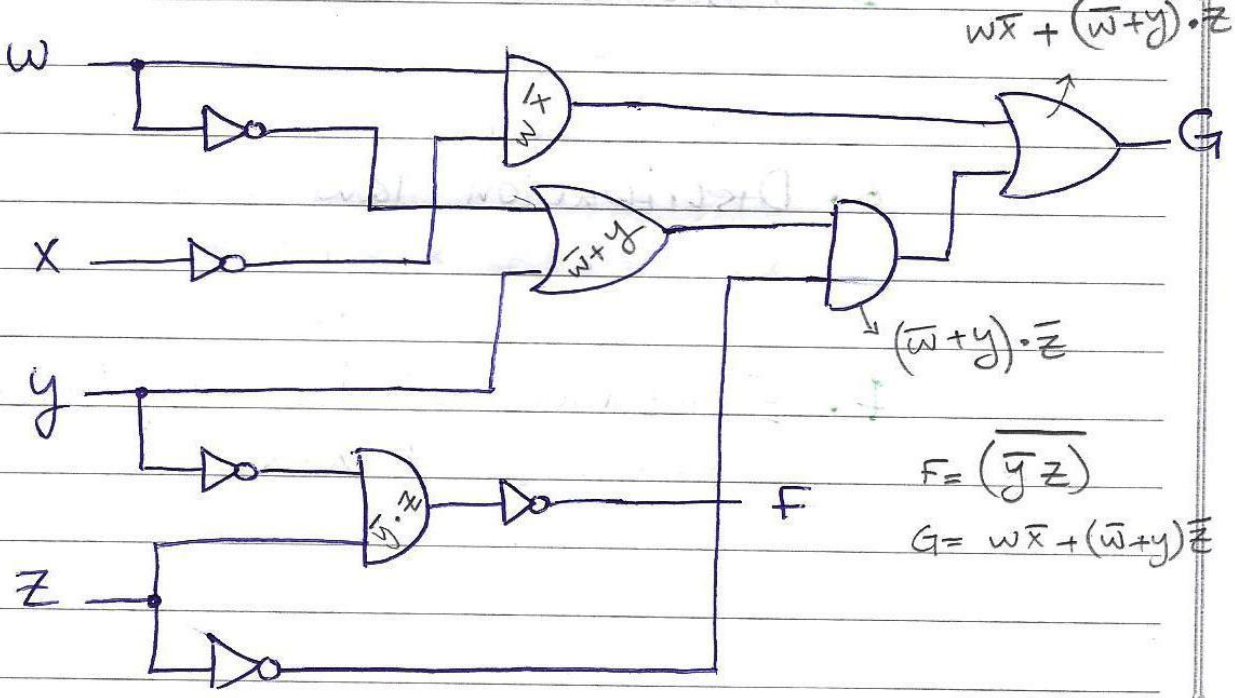
* Note:- for the same function there's only 1 truth table, but it might has more than one logic diagram

→ timing diagram:-



* it's unique for the same function.

exercise:- derive the boolean exepression for F & G from the following logic diagram



$F = \overline{(\bar{y}z)}$

$G = w\bar{x} + (\bar{w}+y)\bar{z}$

* Boolean Algebra :-

* to simplify boolean functions
* smaller / simpler expressions directly affect logic circuits.

→ Boolean identities:-

1. existance of 0 or 1.

$$X + 0 = X \quad \| \quad X \cdot 0 = 0$$
$$X + 1 = 1 \quad \| \quad X \cdot 1 = X$$

2. idempotance.

$$X + X = X \quad | \quad X \cdot X = X$$

3. involution

$$\overline{\overline{X}} = X$$

4. commutative law

$$X + Y = Y + X$$
$$X \cdot Y = Y \cdot X$$

5. Associative law

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

6. Distribution law

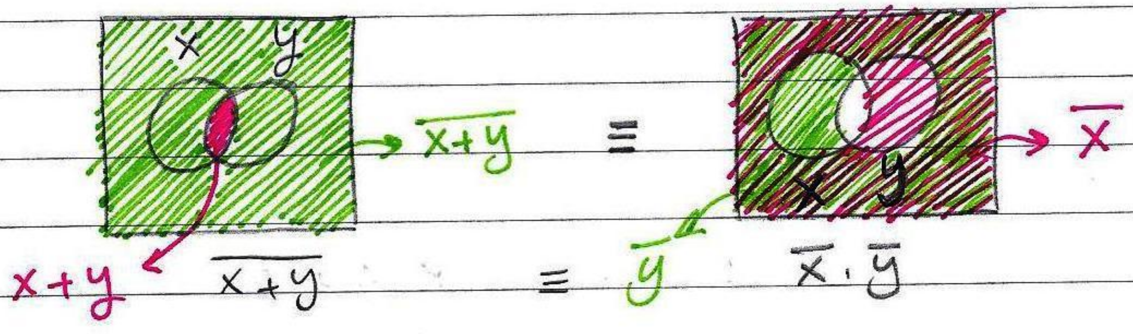
$$X \cdot (Y + Z) = X \cdot Y + X \cdot Z$$
$$X + YZ = (X + Y) \cdot (X + Z)$$

7. Existance of its complement.

$$X + \overline{X} = 1 \quad // \quad X \cdot \overline{X} = 0$$

8. Demorgan Thm

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



* Boolean Theoroms :-

[1] Simplification Thm.

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

Proof → $(x+\bar{x}) \cdot (x+y)$

$= 1 \cdot (x+y) = x+y$ ※

[2] Absorption Thm.

$x + x \cdot y = x$

Proof → $= x(1+y) = x(1) = x$ ※

[3] Minimization Thm:

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

Proof → $= y(x+\bar{x}) = y(1) = y$ ※

[4] Consensus thm:

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

Proof → (مثل فكرة اكمال المربع (زدنا عنصر لا يؤثر على قيمة المقدار)

$x \cdot y + \bar{x} z + y \cdot z \cdot 1$

$xy + \bar{x} z + yz(x+\bar{x})$

$xy + \bar{x} z + yzx + yz\bar{x}$

$xy(1+z) + \bar{x}z(1+y) = xy + \bar{x}z$ ※

(1) (1)

ex: show that

$F = \bar{x}\bar{y} + x\bar{y} + \bar{x}y + xy = 1$

there's 2 ways

*1.

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

*

*2: $\bar{x}(\bar{y}+y) + x(\bar{y}+y)$

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

$= \bar{x} + x = 1$ **

ex: show that $ABC + \bar{A}\bar{C} + A\bar{C} = AB + \bar{C}$

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

$= \bar{C} + AB \cdot C$

$= (AB + \bar{C})(C + \bar{C})$

$= AB + \bar{C}$ **

* Precedance & operaters.

parantlesis ( )

NOT ~

AND .

OR +

→ complement of boolean function:

ex: $F(A,B,C) = AB + \bar{C}$, find $\bar{F}$?

$\bar{F} = \overline{(AB+\bar{C})} = \overline{AB} \cdot C$

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

ex:- if $f(w,x) = \bar{w}x + wx$, $\bar{F}$?

$f = x(\bar{w}+w) = x$

$\therefore \bar{f} = \bar{x}$ ※

ex:- if $f(A,B,C,D) = AB(A+\bar{C}) + \bar{B}(\bar{A}D+C)$, $\bar{f}$?

$$\bar{f} = \overline{AB(A+\bar{C}) + \bar{B}(\bar{A}D+C)}$$
$$= \overline{AB(A+\bar{C})} \cdot \overline{\bar{B}(\bar{A}D+C)}$$
$$= (\overline{AB} + \overline{(A+\bar{C})}) \cdot (B + \overline{(\bar{A}D+C)})$$
$$= (\bar{A}+\bar{B}+\bar{A}C) \cdot (B + \overline{\bar{A}D} \cdot \bar{C})$$
$$= ((\bar{A}+\bar{B})+\bar{A}C) \cdot (B + (A+\bar{D}) \cdot \bar{C})$$

* Dual of a boolean function :-

* replace $+ \rightleftarrows \cdot$

* replace $0 \rightleftarrows 1$

ex:-

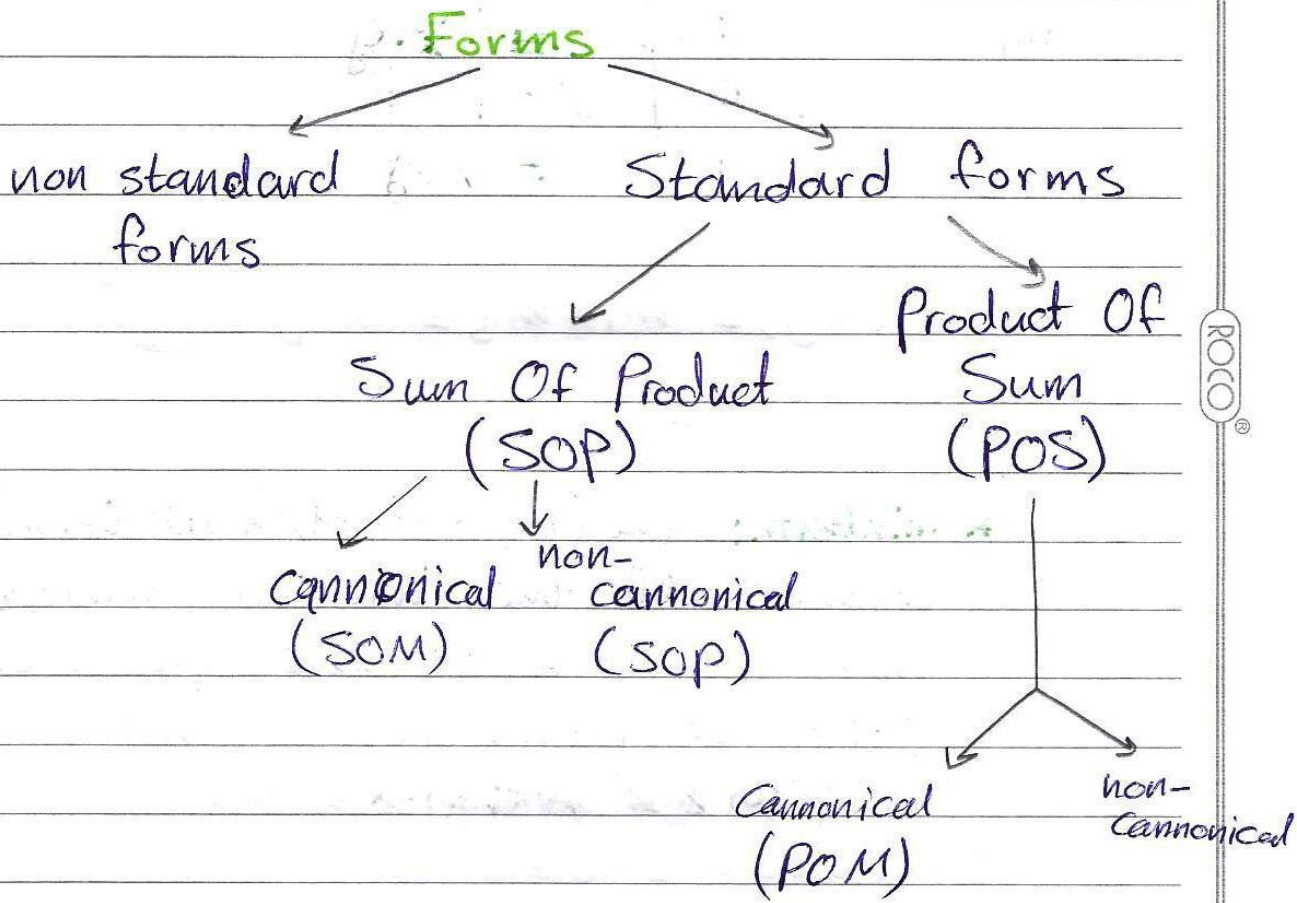
$f = wx + \bar{y}z \cdot 0 + \bar{w}z$, dual of f is:-

$(w+x) \cdot (\bar{y}+z+1) \cdot (\bar{w}+z)$

Standard forms:-

→ we want to express booleam functions in a way such that:-

1. it corresponds to the truth table.
2. it simplifies comparison.
3. it facilitates simplification.



SOM : Sum of minterms :

* given a truth table, we use the (1)s of the function to write an OR expression for the ANDed input values that evaluate the function to 1.

ex:-

		x	y	F	
	0	0	0	0	
m_1	1	0	1	1	$\equiv \bar{x} \cdot y$
	2	1	0	0	
m_3	3	1	1	1	$\equiv x \cdot y$

$$\therefore f(x, y) = m_1 + m_3 = \overbrace{\bar{x}\, y + xy}^{SOM}$$

$$\equiv \sum_m (1, 3)$$

(xy → minterm)

* **minterm:** the (Product of/AND) term that contains all the function variables such that each variable apears once either complemented or not.

سمي min term لأن الـ AND يخرج الناتج مساوياً
لأصغر قيمة تعوض فيه 1.1=1 // 0.0=0.1=1.0=0

ex:-

	x	y	z	F	
0	0	0	0	0	
1	0	0	1	1	→ $m_1 = \bar{x}\bar{y}z$
2	0	1	0	1	→ $m_2 = \bar{x}y\bar{z}$
3	0	1	1	0	
4	1	0	0	1	→ $m_4 = x\bar{y}\bar{z}$
5	1	0	1	0	
6	1	1	0	0	
7	1	1	1	1	→ $m_7 = xyz$

$$F = m_1 + m_2 + m_4 + m_7 = \sum_m (1,2,4,7)$$
$$= \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz \ *$$

ex:- $F(A,B,C) = \sum_m (3,4,6)$

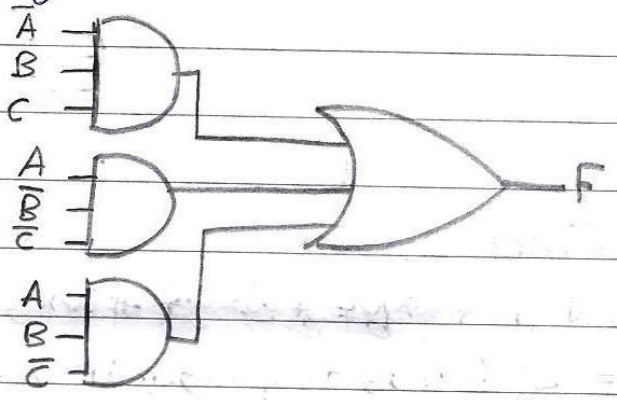
a) $F = \underset{(011)}{m_3} + \underset{(100)}{m_4} + \underset{(110)}{m_6}$

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

b) T.T?

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

c) draw logic diagram for f



ex	w	x	y	z	F	
0	0	0	0	0	0	
1	0	0	0	1	1	m_1
2	0	0	1	0	0	
3	0	0	1	1	1	m_3
4	0	1	0	0	1	m_4
5	0	1	0	1	1	m_5
6	0	1	1	0	0	
7	0	1	1	1	0	
8	1	0	0	0	1	m_8
9	1	0	0	1	0	
10	1	0	1	0	0	
11	1	0	1	1	1	m_{11}
12	1	1	0	0	0	
13	1	1	0	1	0	
14	1	1	1	0	0	
15	1	1	1	1	1	m_{15}

a) find SOM

$$F = \bar{w}\bar{x}\bar{y}z + \bar{w}\bar{x}yz + \bar{w}x\bar{y}\bar{z} + \bar{w}xy\bar{z} + w\bar{x}\bar{y}\bar{z} + w\bar{x}yz + wxyz$$

$$= \sum_m (1,3,4,5,8,11,15)$$

b) SOM for $\bar{f}(w,x,y,z)$

$$\bar{f} = \sum_m (0,2,6,7,9,10,12,13,14)$$

$$= \bar{w}\bar{x}\bar{y}\bar{z} + wx\bar{y}z + \bar{w}xy\bar{z} + \bar{w}xyz + w\bar{x}\bar{y}z + w\bar{x}y\bar{z} + wx\bar{y}\bar{z} + wx\bar{y}z + wxy\bar{z}$$

* Converting to SOM from Boolean expressions:-

* $f(x,y,z) = xy + \bar{y}z \Rightarrow$ (SOP) $\neq$ SOM (minterm)

Sum (OR) of Product (AND)

$f(x,y,z) = xyz + \bar{x}\bar{y}z \Rightarrow$ SOM

↳ ① derive the truth table, then find SOM.

↳ ② convert the expression to SOP, then check the product terms for any missing variable (v), for each missing variable AND the term with $(\bar{v}+v)$.

ex:- $F(A,B,C) = (A+C)B + \bar{B}C$

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

$= \underset{111}{ABC} + \underset{110}{AB\bar{C}} + \underset{011}{\bar{A}BC} + \cancel{ABC} + \bar{A}\bar{B}C + A\bar{B}C$

$= \underset{111}{ABC} + \underset{110}{AB\bar{C}} + \underset{011}{\bar{A}BC} + \underset{001}{\bar{A}\bar{B}C} + \underset{101}{A\bar{B}C}$

$= \sum_m (1,3,5,6,7)$

$\bar{F}(A,B,C) = \sum_m (0,2,4)$

n (variable)

↳ max unsigned value $= 2^n - 1$

↳ # of combinations $= 2^n$

⇝ POM: product of Maxterm:-

*given the T.T we use (0)s to write an AND expression for the ORed input values such that each OR term evaluates to zero.

ex:-

	x	y	F	
0	0	0	0	$\rightarrow x+y = m_0$
1	0	1	1	
2	1	0	0	$\rightarrow \bar{x}+y = m_2$
3	1	1	1	

$$f = \Pi_M(0,2) = (x+y).(\bar{x}+y) = M_0 \cdot M_2$$

(x+y) and (x̄+y): max term

(x+y).(x̄+y): POM

*max term:-

an (OR/Sum) expression that contains all function variables such that each variable apears once eithe complemented or not.

ex:

	x	y	z	F	
0	0	0	0	0	$M_0 \rightarrow x+y+z$
1	0	0	1	1	
2	0	1	0	1	
3	0	1	1	0	$M_3 \rightarrow x+\bar{y}+\bar{z}$
4	1	0	0	1	
5	1	0	1	0	$M_5 \rightarrow \bar{x}+y+\bar{z}$
6	1	1	0	0	$M_6 \rightarrow \bar{x}+\bar{y}+z$
7	1	1	1	1	

$$F = M_0 \cdot M_3 \cdot M_5 \cdot M_6 = \Pi_M(0,3,5,6)$$

$$= (x+y+z)(x+\bar{y}+\bar{z}).(\bar{x}+y+\bar{z}).(\bar{x}+\bar{y}+z)$$

ex:- given that:

$F(w,x,y,z) = \sum_m (1,2,4,9,12,13,15)$

① $f = \Pi_M (0,3,5,6,7,8,10,11,14)$

② $\bar{f} = \sum_m (0,3,5,6,7,8,10,11,14)$

③ $\bar{f} = \Pi_M (1,2,4,9,12,13,15)$

* Converting to POM from logic expression.

↳ ① t.t

↳ ② using boolean Algebra compute the POS, the check the sum terms for missing variables, for each missing variable (v) → OR the term with $(v.\bar{v})$

ex:- $f(w,x,y) = wx + \bar{x}y$, find POM?!

$= (wx + \bar{x}) \cdot (wx + y)$

$= (w + \bar{x}) \cdot (x + \bar{x}) \cdot (w + y)(x + y)$

$= (w + \bar{x} + y\bar{y}) \cdot (w + x\bar{x} + y)(w\bar{w} + x + y)$

$= (w+\bar{x}+y)(w+\bar{x}+\bar{y})(w+x+y)(w+\bar{x}+y)(w+x+y)(\bar{w}+x+y)$

$= (w+\bar{x}+y) \cdot (w+\bar{x}+\bar{y}) \cdot (w+x+y) \cdot (\bar{w}+x+y)$

0 1 0 , 0 1 1 , 0 0 0 , 1 0 0

2 , 3 , 0 , 4

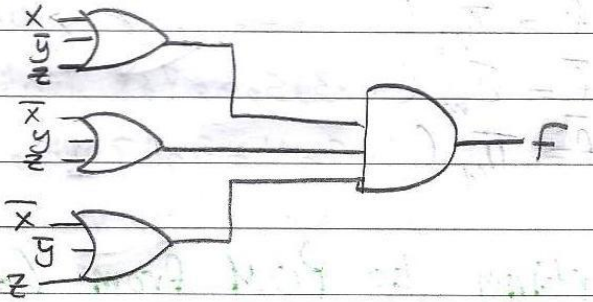
$= \Pi_M (0,2,3,4) = M_0 \cdot M_2 \cdot M_3 \cdot M_4$

2 4 7

0 1 0 1 0 0 1 1 1

ex:- $f(x,y,z) = (x+\bar{y}+z)(\bar{x}+y+z)(\bar{x}+\bar{y}+\bar{z})$

a) draw logic digram for f.



$F = \Pi_M (2,4,7)$

* Relationship between minterms & max terms:-

$m_0 = \bar{x}\,\bar{y}\,\bar{z}$

$M_0 = x+y+z$

$\therefore m_0 \rightleftharpoons \overline{M_0} \rightarrow \begin{cases} m_j = \overline{M_j} \\ \overline{m_j} = M_j \end{cases}$

* Ch 2- Part 2

two level circuit optimization.

(systematic approach).

* for any boolean function, truth table is unique. However, algebric isn't, this directly affect the cost!
* we need to write the function in its simplest form, using:-
  1. Boolean Algebra (not-systematic)
  2. K-maps
  3. tabulation] Systematic

** Cost Criteria.

1. literal cost (ℓ): number of letters in the expression, it reflects no. of inputs to the first level of gates.

2. Gate input cost (G): it's the literal cost plus number of terms in the expression excluding single variable terms.
   → Optionaly we can add the number of Inverters (NOT Gates). In this case it's called (GM)

ex:- $F(A,B,C,D) = ABC + \bar{A}C + B\bar{D} + C$

$\ell = 8$

$G = \ell$ + number of terms $= 8 + 3 = 11$

$GM = G$ + No. of inverters $= 11 + 2 = 13$

ex:- $F(A,B,C) = ABC + \bar{A}\bar{B}\bar{C}$

$\ell = 6$

$G = 6 + 2 = 8$

$GM = 8 + 3 = 11$

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

$\ell = 6$

$G = 9$

$GM = 12$

* Karnaugh Maps (K-maps). ≡ (Graphical representation of T.T)

* it's a diagram that's made up of squares, each square is a minterm, maxterm or a raw from T.T
* for an n-variable function, there are 2^n squares.
* We can draw K-map for any n-variable function.

* 2-variable K-map

(x → MSD, y → LSD)

x \ y	$\bar{y}$ (0)	y (1)
$\bar{x}$ (0)	0: m_0	1: m_1
x (1)	2: m_2	3: m_3

ex:- $f(x,y) = \sum_m (0,1)$, Draw K-map

x \ y	$\bar{y}$	y
$\bar{x}$	0: 1	1: 1
x	2: 0	3: 0

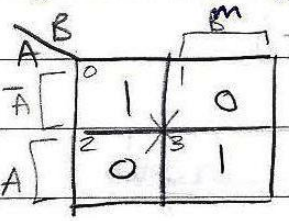
* they share an edge
* $m_0 + m_1 = \bar{x}$
* كده $\bar{x}$ لو وضعناها مع y أو $\bar{y}$ ستدي نفس القيمة = 1.

$F = \sum_m (0,1) = m_0 + m_1 = \bar{x}\bar{y} + \bar{x}y$

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

$F(A,B) = \sum (0,3) = m_0 + m_3 = \bar{A}\bar{B} + AB$ *1

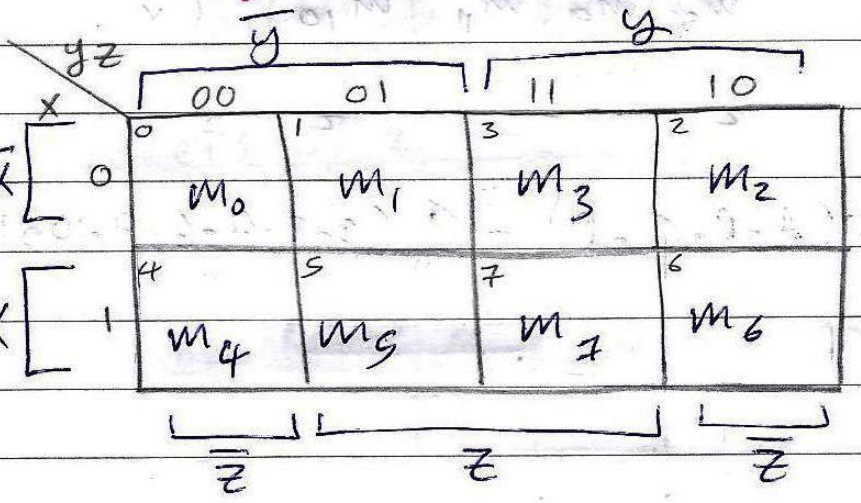
*2



⇒ نعمل اختصار

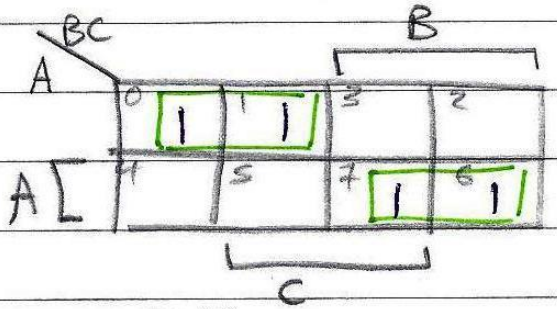
* 3-variable k-map:-

* $2^3 = 8$ (raws/squares)



* بدلنا بين الخود 3 & 4 حتى يكون الفرق بين كل مربعين متجاورين متغير واحد فقط.

ex:- $F(A,B,C) = \sum_m (0,1,3,7)$



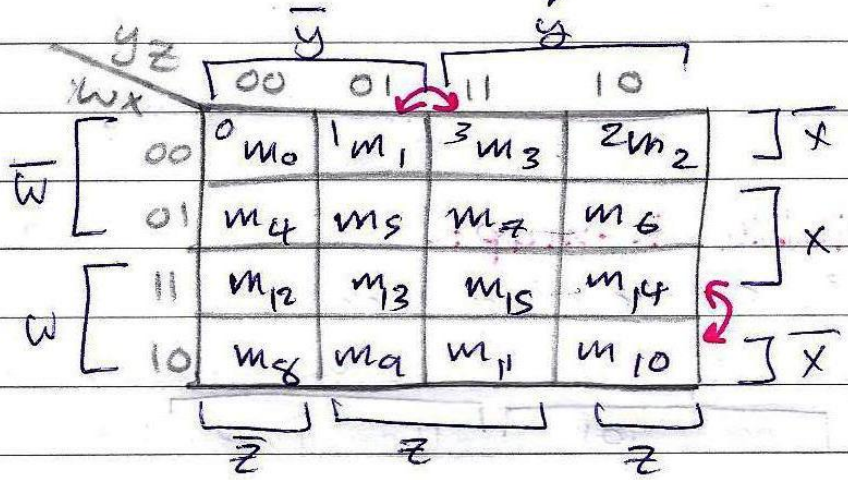
$m_0 + m_1 = \bar{A}\bar{B}$

$m_6 + m_7 = AB$

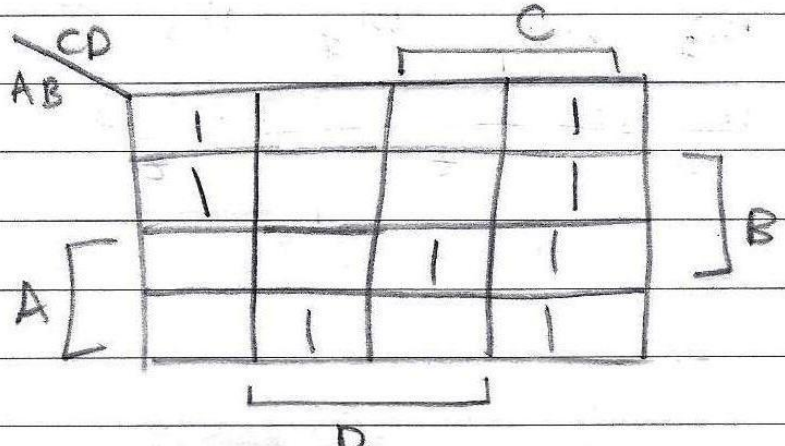
$\therefore F = \bar{A}\bar{B} + AB$

* 4-variable K-map:-

* $2^4 = 16$ (raws / squares).



ex :- $f(A,B,C,D) = \sum_m (0,2,4,6,9,10,14,15)$



Map Manipulation :-

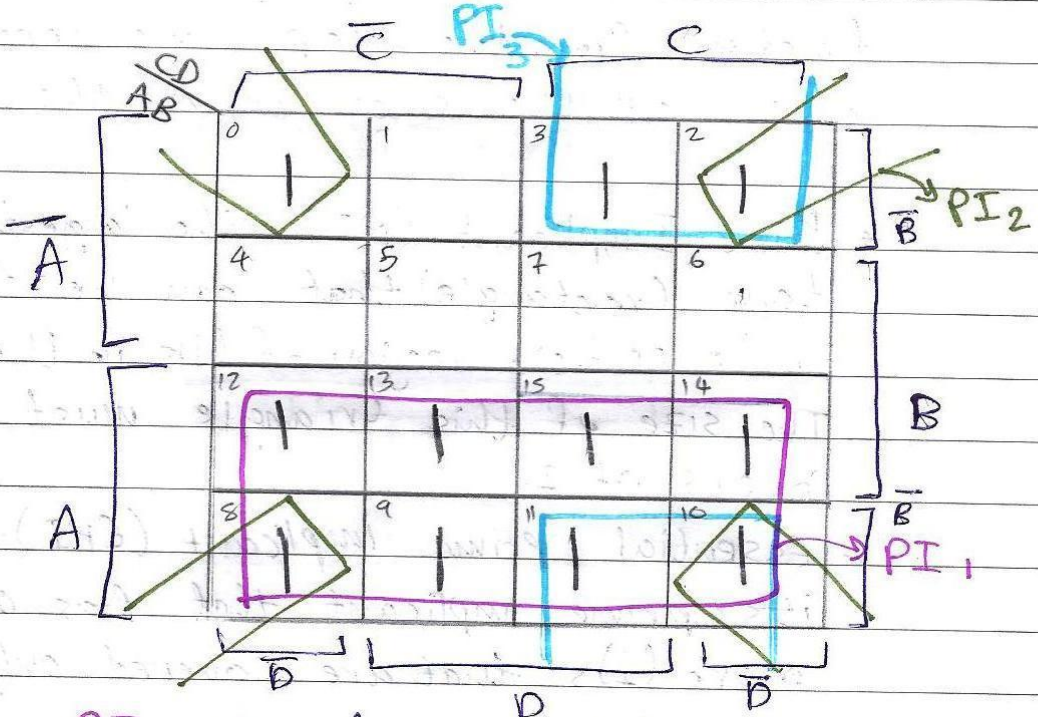
* In order to find the expression for the function from the map we need to cover all the ones or all the zeros
to get minterms · to get max terms

* Prime Implicant (PI): the largest product term (rectangle) that can be defined in a certain region of (1)s in the map. The size of this triangle must be powers of 2.
* Essential prime implicant (EPI): it's a prime implicant that has one or more (1)s that are covered only by this PI.

⇒ Steps:-

1. Draw & fill the map.
2. identify all (EPI)s & include them in the expression.
3. for the remaining (1)s use the minimum number of (PI)s with minimum overlap.

ex:- $F(A,B,C,D) = \sum_m (0,2,3,8,9,10,11,12,13,14,15)$



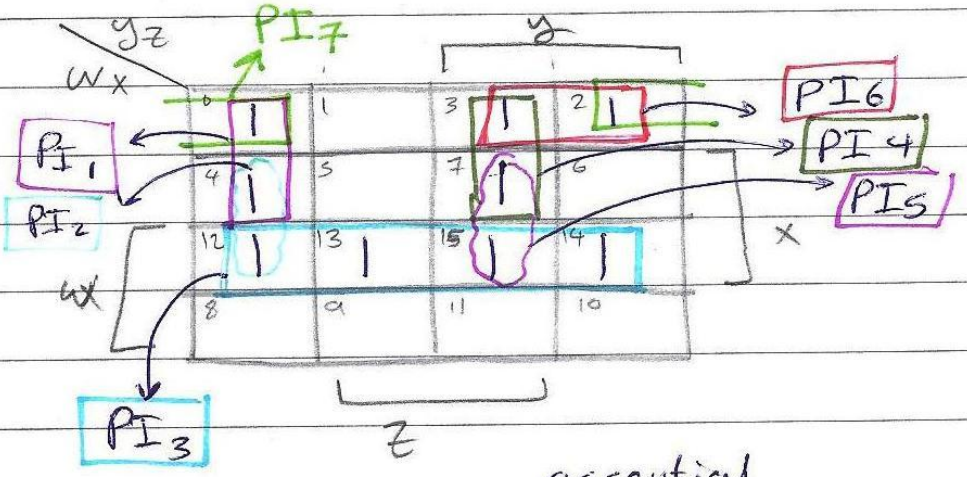
$PI_1 \rightarrow A$ essential

$PI_2 \rightarrow \bar{B}.\bar{D}$ "

$PI_3 \rightarrow \bar{B}.C$ "

$\therefore F = A + C\bar{B} + \bar{B}\bar{D}$

ex :- $F(w,x,y,z) = \sum_{im}(0,2,3,4,7,12,13,14,15)$



		essential
PI_1	$\rightarrow \bar{w}\bar{y}\bar{z}$	X
PI_2	$\rightarrow x\bar{y}\bar{z}$	X
PI_3	$\rightarrow w.x$	✓
PI_4	$\rightarrow \bar{w}yz$	X
PI_5	$\rightarrow xyz$	X
PI_6	$\rightarrow \bar{w}\bar{x}y$	X
PI_7	$\rightarrow \bar{w}\bar{x}\bar{z}$	X

$F(w,x,y,z) = wx + \bar{w}\bar{y}\bar{z} + \bar{w}yz +$ { $\bar{w}\bar{x}\bar{z}$ X
or $\bar{w}\bar{x}y$

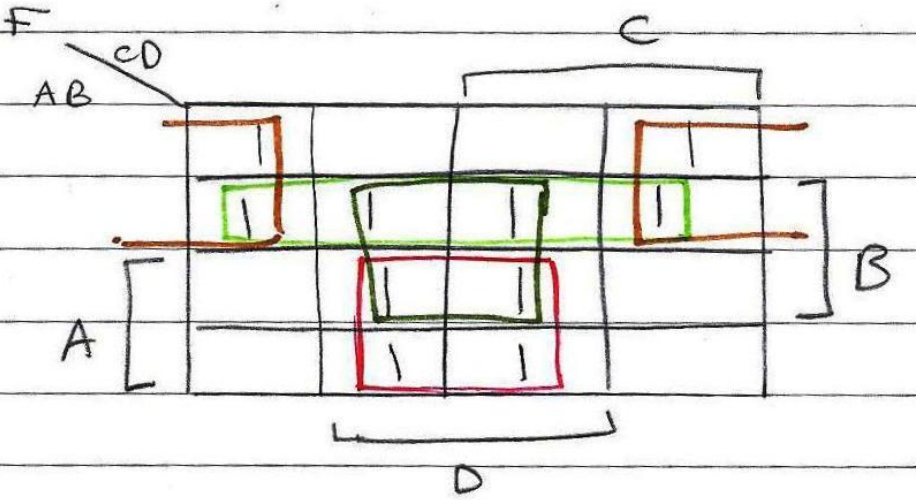
↳ less number of invertors.

$F = wx + \bar{w}\bar{y}\bar{z} + \bar{w}yz + \bar{w}\bar{x}y$

ex:- $F(A,B,C,D) = \Pi_M(1,3,8,10,12,14)$

simplify using K-maps

$$F = \sum_m(0,2,4,5,6,7,9,11,13,15)$$

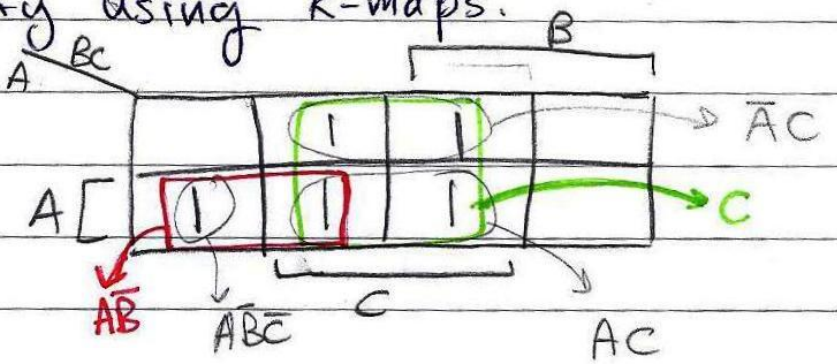


$F = AD + \bar{A}\bar{D} + \begin{cases}\bar{A}B \\ BD\end{cases}$ ⇒ we choose the less invertors (NOT Gates).

$\therefore F = AD + \bar{A}\bar{D} + BD$

ex:- $F(A,B,C) = A\bar{B}\bar{C} + AC + \bar{A}C$

simplify using K-maps.



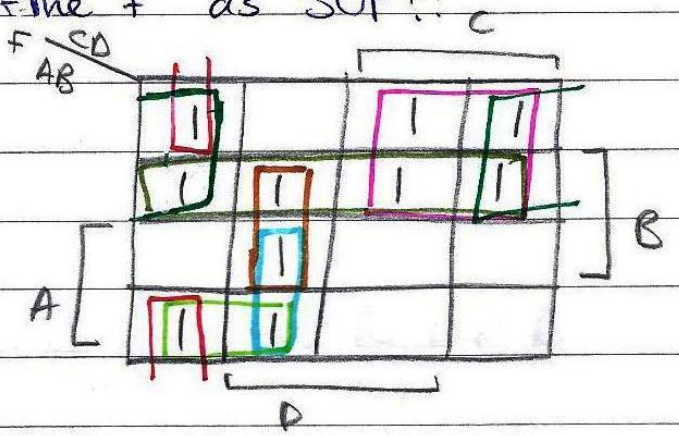
$\therefore F = A\bar{B} + C$

* Product of sum (POS) optimization

1. find $\bar{f}$ & fill the map.
2. Optimize to $\bar{f}$ as SOP by grouping zeros of $\bar{f}$.
3. take the complement of $\bar{f}$ to find f as POS.

ex:- $F(A,B,C,D) = \sum_m (0,2,3,4,5,6,7,8,9,13)$

a) define f as SOP?!



essential

$\bar{A}C$ ✓

$\bar{A}B$ ✗

$B\bar{C}D$ ✗

$\bar{A}\bar{D}$ ✗

$A\bar{B}\bar{C}$ ✗

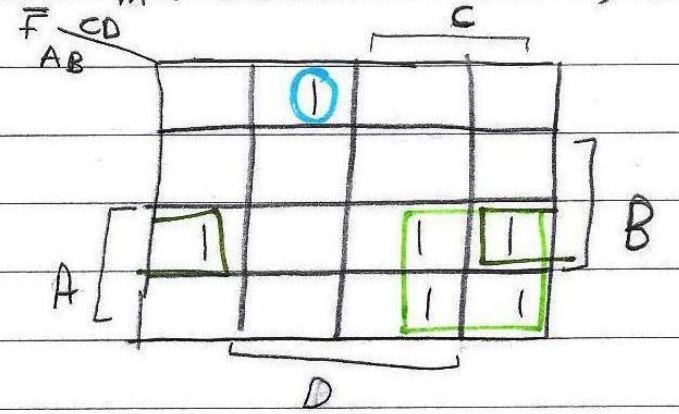
$A\bar{C}D$ ✗

$\bar{B}\bar{C}\bar{D}$ ✗

$$F = \bar{A}C + \bar{A}B + A\bar{C}D + \bar{B}\bar{C}\bar{D}$$

b) define f as POS?!

$$\bar{F} = \sum_m (1,10,11,12,14,15)$$



ess.

AC ✓

$AB\bar{D}$ ✓

$\bar{A}\bar{B}\bar{C}D$ ✓

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

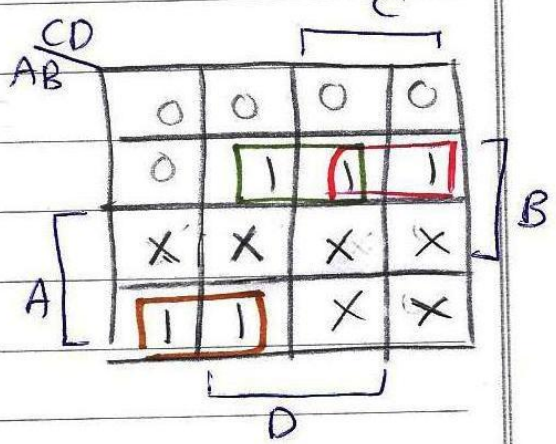
$$F = \bar{\bar{F}} = (\bar{A}+\bar{C})(\bar{A}+\bar{B}+D)\cdot(A+B+C+\bar{D}).$$

* Don't care conditions:-

* for some functions it's not unusual to have entries in the T.T such that :-
  1. The input values for some minterms will never occur.
  2. The output of the function for some inputs isn't used
     → in such case the output need not to be defined. Instead, we use "Don't cares" (X).
     → using (X) may reduce the cost.

ex:- A circuit with inputs A, B, C, & D which represents a BCD numbers outputs 1 if the input > 4. ↳ BCD ∈ [0-9]

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	1	0	0	0
0	1	0	1	1
1	0	0	1	1
1	0	1	0	X
				X



↝ without x : (x = 0)

$F = A\bar{B}\bar{C} + \bar{A}BD + \bar{A}BC$ ← $A\bar{B}\bar{C}$, $\bar{A}BD$, $\bar{A}BC$

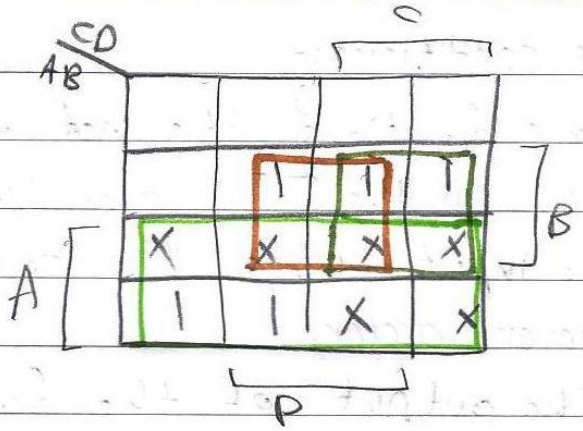
G = 12

⇝ with x = 1 whenever needed

A

BC

BD



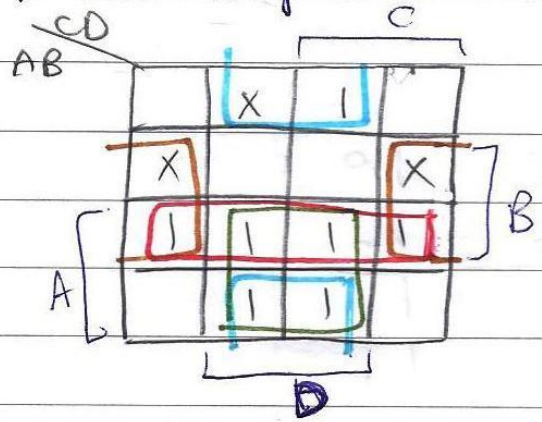
* we need to cover all the (1)s not all the (X)s.

$$F = BC + BD + A$$

G = 7

ex:- $F(A,B,C,D) = \sum_m (3,9,11,12,13,14,15) + \sum_d (1,4,6)$

① F as SOP

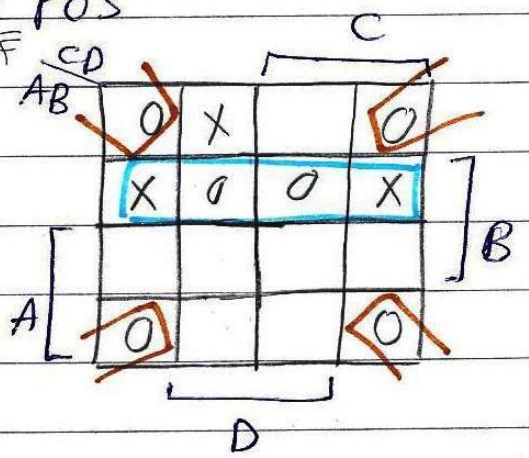


→ short hand notation for (don't cares)

	css.
AD	×
$\bar{B}D$	✓
$B\bar{D}$	×
AB	×

$$F = \bar{B}D + AB$$

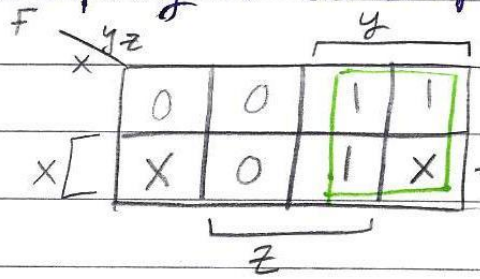
② find f as POS



$$\overline{F} = \bar{A}B + \bar{B}\bar{D}$$

$$F = \overline{\overline{F}} = (A + \bar{B})(B + D)$$

ex: if $f(x,y,z) = \prod_M (0,1,5) + \sum_d (4,6)$ (zeros)
then simplify f as SOP.



⇒ assume x=1 when needed.

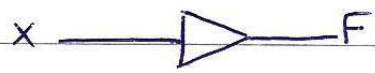
$f(x,y,z) = y$

* Other types of gates:-

* we already saw AND, OR & NOT gates.
* logic gates can be: Permitive or Complex
  ↓ includes more than one permitive operation

A. Permitive Gates:-

1. Buffer:

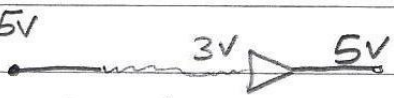


F = X

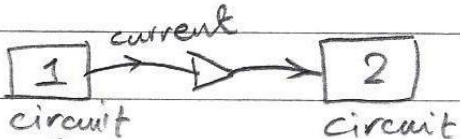
* T.T

X	F
0	0
1	1

- It can be used to amplify voltage levels.
- It can be used for protection & isolation.

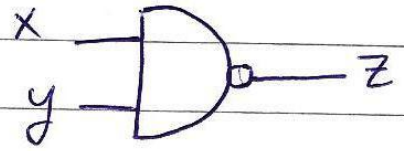


*Amplification.



(*isolation between circuits/systems)

2. NAND (NOT-AND)



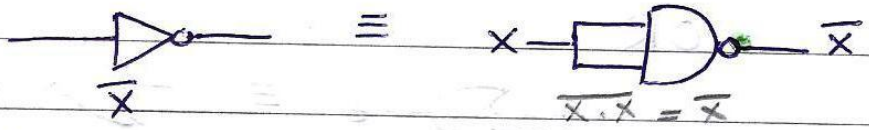
$Z = \overline{x \cdot y}$

*T.T

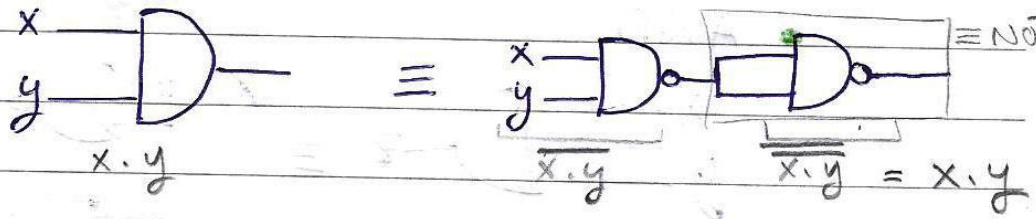
x	y	z
0	0	1
0	1	1
1	0	1
1	1	0

*NAND is a universal gate. OR, AND يعني نستطيع عمل كل البوابات المنطقية NOT

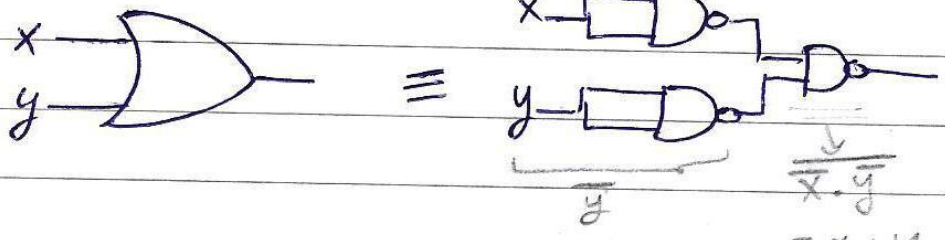
- NOT



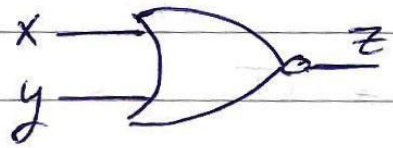
- AND



- OR



3. NOR (NOT-OR)



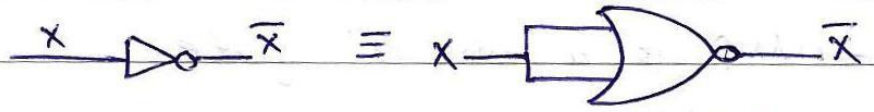
*T.T

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

$z = \overline{x+y}$

*NOR gate is also a universal gate.

- NOT

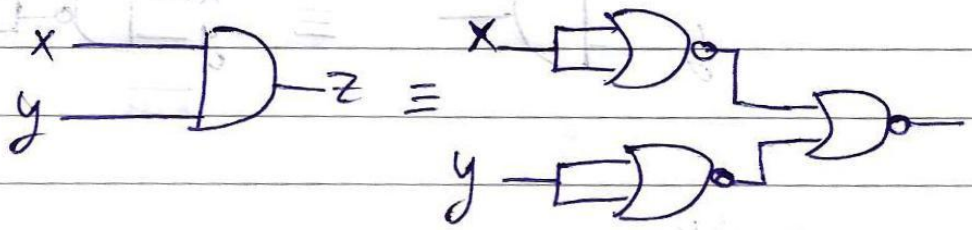


$\overline{x+x} = \bar{x}$

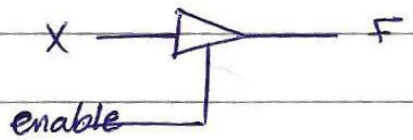
- OR

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

- AND



4. Tri-state Buffer.



* T.T

enable	X	F
0	0	Z
0	1	Z
1	0	0
1	1	1

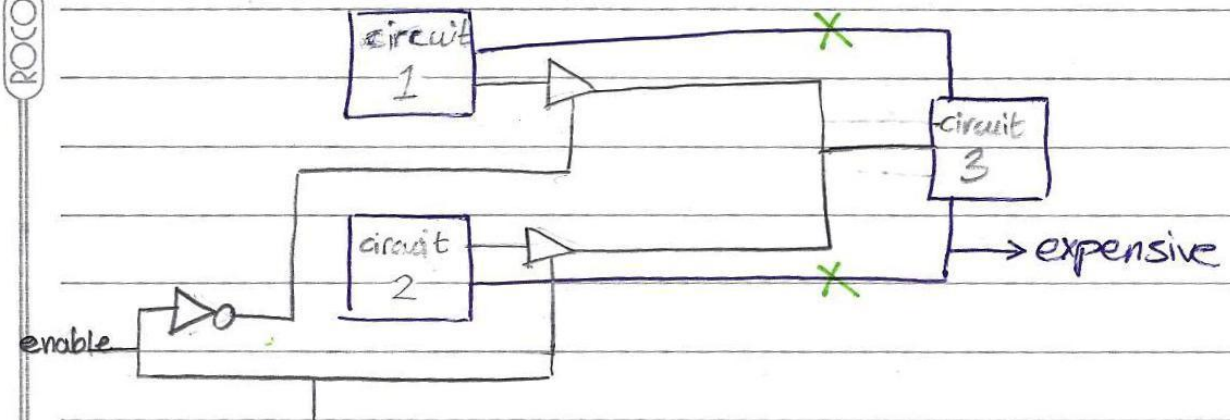
Buffer is disabled

Buffer is enabled

* Tri-state (0, 1, Z)

* when (enable = 0), buffer is disabled and acts as an open circuit (high impedance).

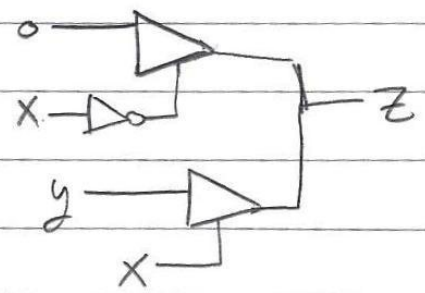
* when (enable = 1), buffer is enabled and acts as short circuit.
* Tri-state buffer allows sharing of wires (links).



↳ in this case one of them only will be connected at any moment of time.

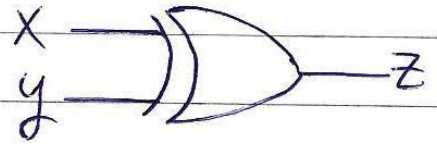
ex:- implement the AND gate using tri-state & invertors.

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



B. Complex gates:-

1. XOR (Exclusive -OR)



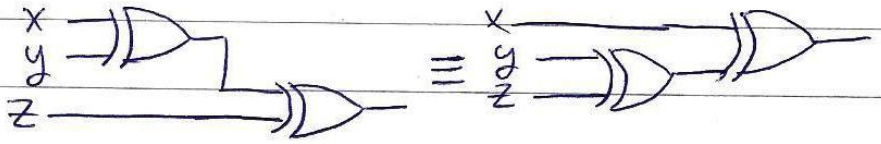
*T.T

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

$Z = X \oplus Y \equiv \bar{X}Y + X\bar{Y}$ $\qquad F = \bar{X}y + X\bar{y}$

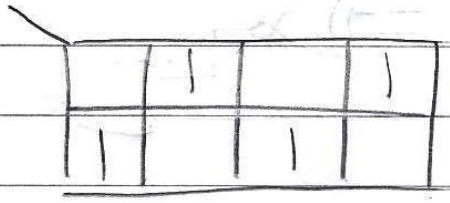
*Properties:-

- $X \oplus 0 = X$
  $X \oplus 1 = \bar{X}$
  $X \oplus X = 0$
  $X \oplus \bar{X} = 1$
- $X \oplus Y = Y \oplus X$ (comutative)
- $X \oplus (Y \oplus Z) = (X \oplus Y) \oplus Z$ (Associative)



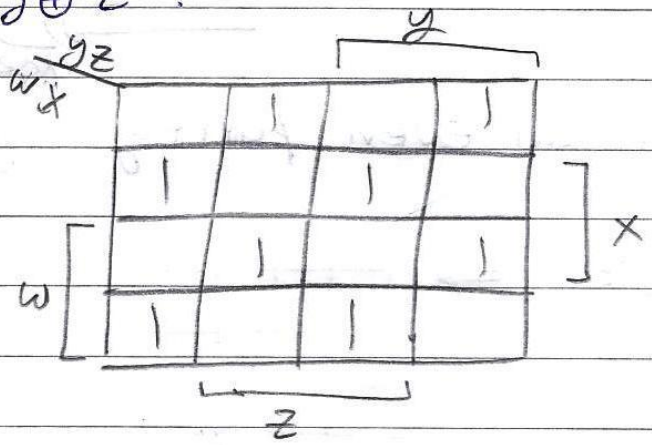
$$X \oplus Y \oplus Z = \overline{(X \oplus Y)}\,Z + (X \oplus Y)\bar{Z}$$
$$= \overline{(\bar{X}Y + X\bar{Y})}\,Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z}$$
$$= (X + \bar{Y})(\bar{X} + Y)Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z}$$
$$= (X\bar{X} + XY + \bar{X}\bar{Y} + Y\bar{Y})Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z}$$
$$= XYZ + \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z}$$
111 001 010 100

*number of (1)s is odd



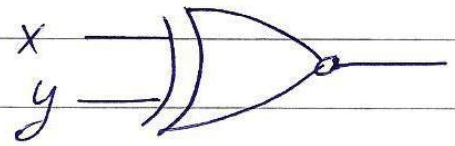
* XOR is an odd function.

$w \oplus x \oplus y \oplus z$:



*XOR is an odd parity generator

2. XNOR (Exclusive-NOT-OR)



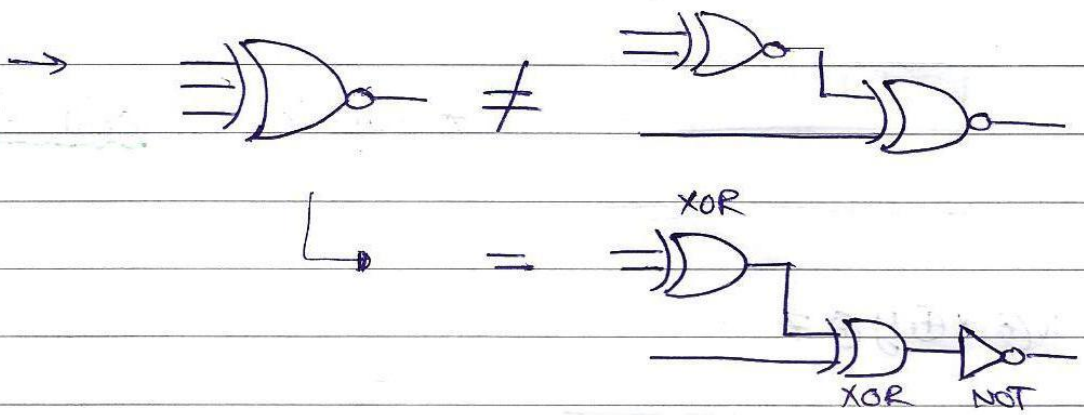
*T.T

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

$$z = x \odot y \equiv \bar{x}\bar{y} + xy$$

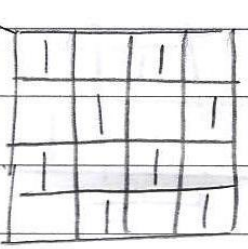
*Properties:-

- $x \odot 0 = \bar{x}$
  $x \odot 1 = x$
  $x \odot x = 1$
  $x \odot \bar{x} = 0$
- $x \odot y = y \odot x$
- $x \odot (y \odot z) \neq (x \odot y) \odot z$ (not associative).



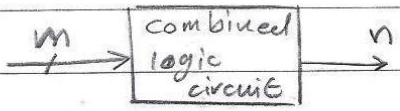
* XNOR is an even parity generator.

$x \odot y \odot z \Rightarrow$



*Combinational logic design:-

* The circuits whose output at any instant of time depends on the current input only
* They may have (m) inputs, (n) outputs, and N logic functions:-



*Design Steps:-

1. specification : it specifies what the circuit does
2. Formulation: derive the T.T for an initial boolean expression.
3. Optimization : apply 2-level optimization to obtain the function expression.
4. technology mapping : draw the logic diagram using AOI (AND, OR, Inverter), NAND, NOR.
5. Verification : verify the design manualy or by simulators.

ex:- design a circuit that has 3 inputs (x,y,z) & it outputs 1 if the number of (1)s in the input is greater than the number of (0)s (majority function).

① Specification

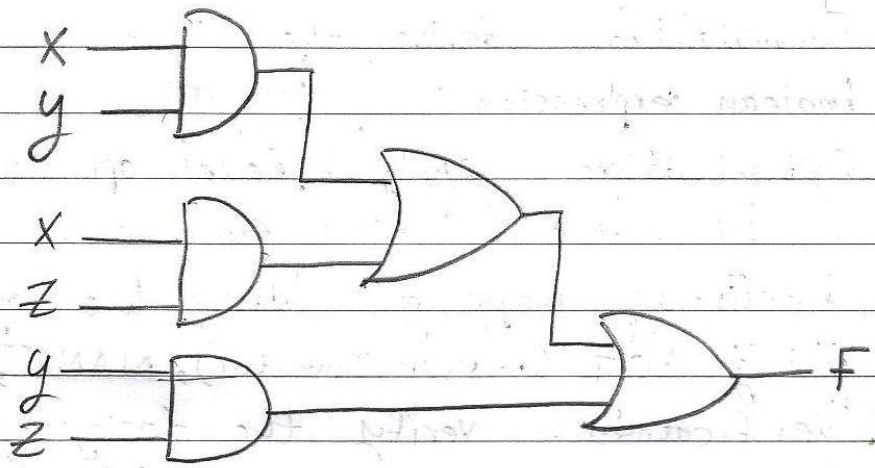
② Formulation

x	y	z	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

③ Optimization

$F = yz + xy + xz$

④ technology mapping



⑤ Verification.

ex :- Design a circuit that converts from BCD to excess-3-code

① * inputs = 4 (BCD digits (0-9))
* outputs = 4 (BCD+3 = 3-12)

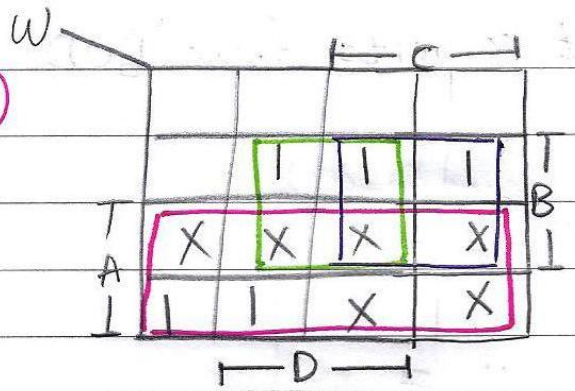
②

A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	x	x	x	x
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

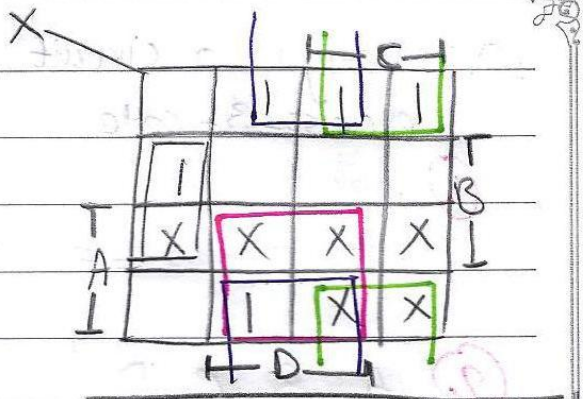
(don't cares)

⇒ 10-15 aren't BCD digits but they could be BCD numbers

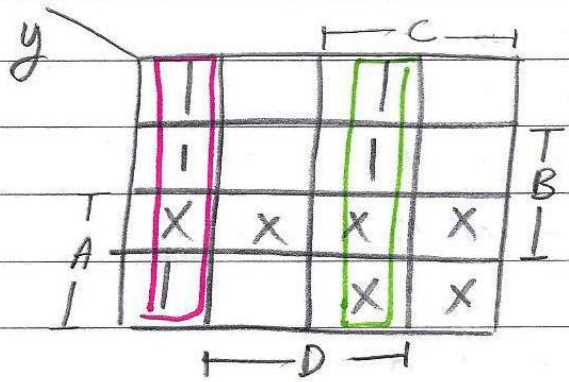
③



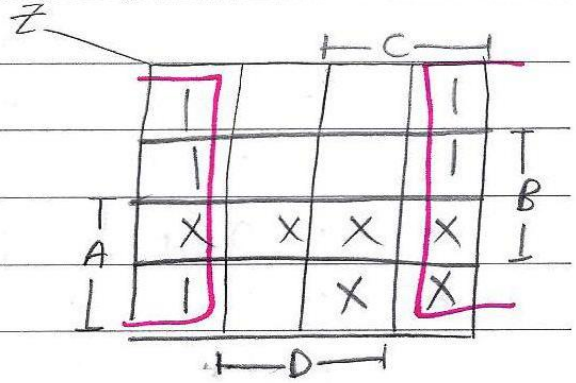
$$W = A + BC + B\bar{D}$$



$$X = \bar{B}D + \bar{B}C + B\bar{C}\bar{D}$$

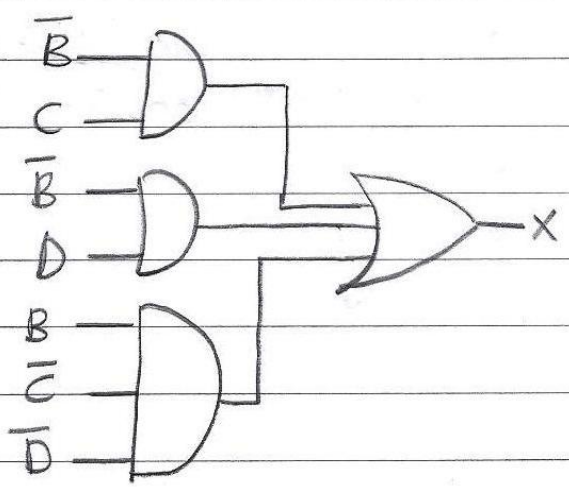


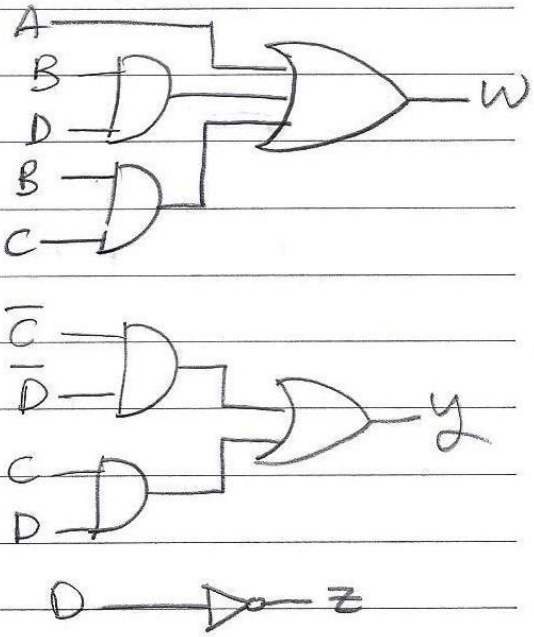
$$y = \bar{C}\bar{D} + CD$$



$$Z = \bar{D}$$

④





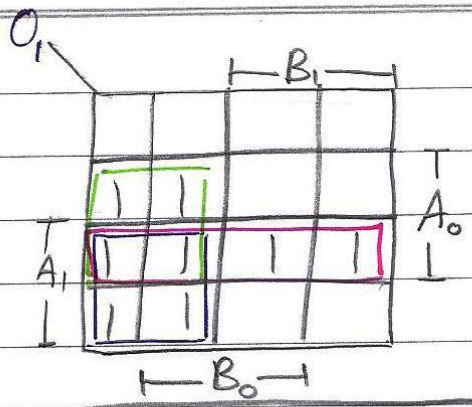
ex:- design a circuit that compares two 2-bit binary numbers ; a & b and outputs 2 signals

① O_1, O_0 such that $O_1O_0 =$ { 00 , if a=b & both are even
01 , " a<b
10 , " a>b
00 , " a=b & both are odd

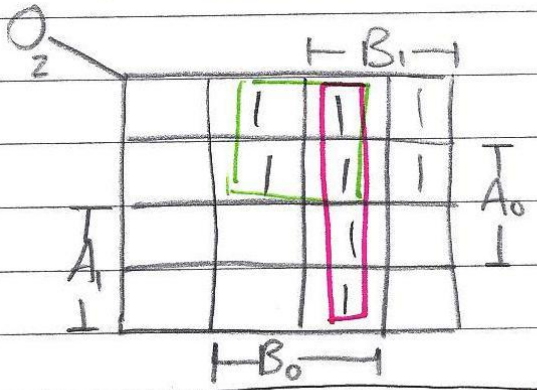
②

A		B		O_1	O_0
A_1	A_0	B_1	B_0		
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	1

(3)

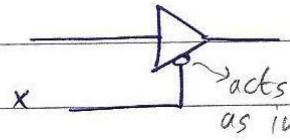


$$O_1 = A_1\overline{B_1} + A_0\overline{B_1} + A_1A_0$$

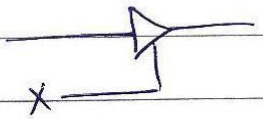


$$O_2 = B_1\overline{A_1} + B_0\overline{A_1} + B_1B_0$$

* Note:-



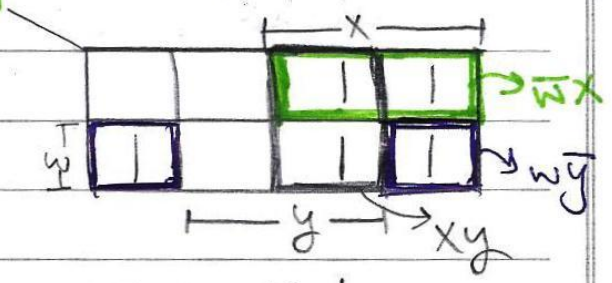
Active low signal enabled when x = 0 (low)

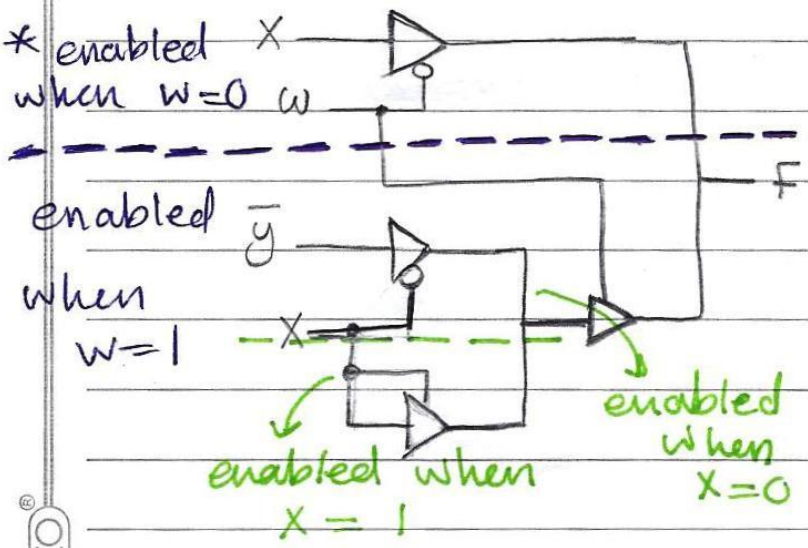


Active high signal enabled when x = 1 (high)

* Tri-state buffer representation:-

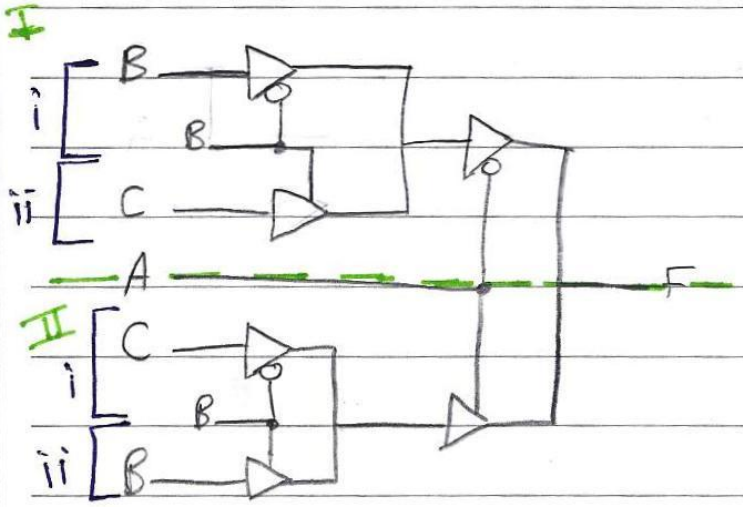
ex:- $F = \bar{w}x + w\bar{y} + xy$





w	x	y	F	
0	0	0	0	
0	0	1	0	
0	1	0	1	x
0	1	1	1	
1	0	0	1	
1	0	1	0	$\bar{y}$
1	1	0	1	
1	1	1	1	x

*ex: $F = \bar{A}BC + A\bar{B}C$



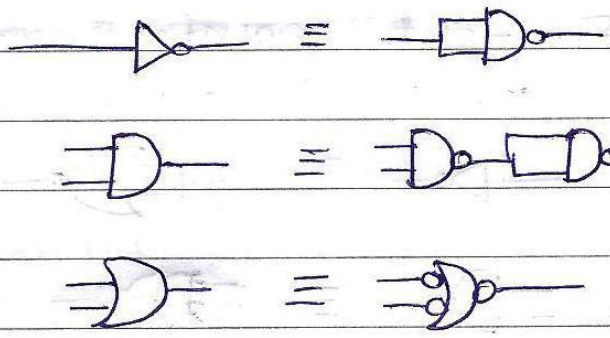
A	B	C	F	
0	0	0	0	
0	0	1	0	B
0	1	0	0	
0	1	1	1	C
1	0	0	0	
1	0	1	1	C
1	1	0	0	
1	1	1	0	$\bar{B}$

* Technology mapping :-

* we want to map the AOI diagram to NAND or NOR representation only.

#1: mapping to NAND:

1. Replace each AOI with its equivelant NAND representation.

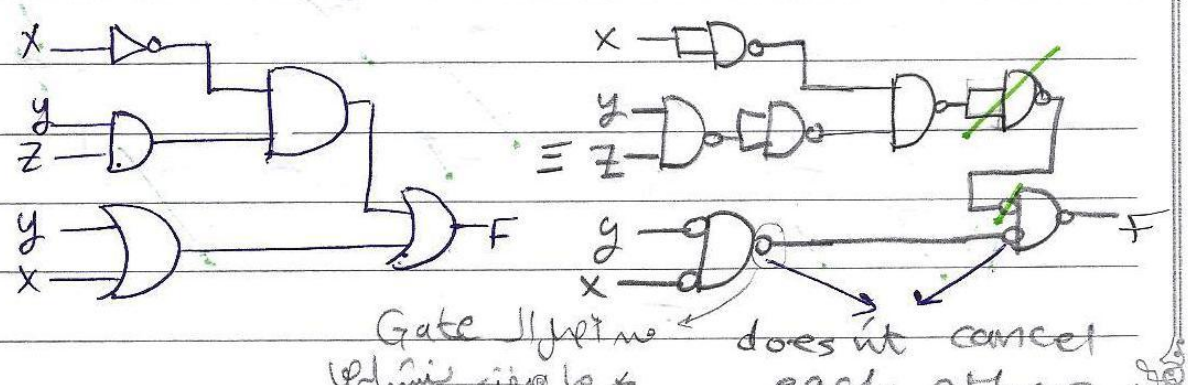


2. Repeat the following until there is one invertor per wire.

i) Push invertors through circuit Fan-out points.

ii) Cancel invertor pairs.

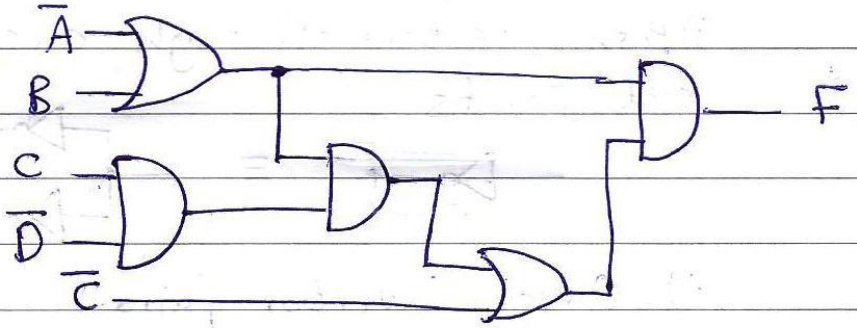
ex:

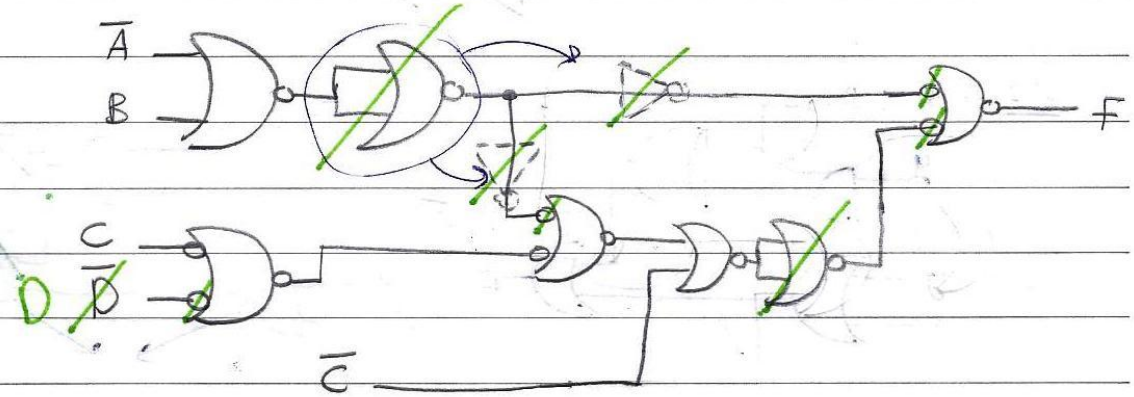


※2: mapping to NOR

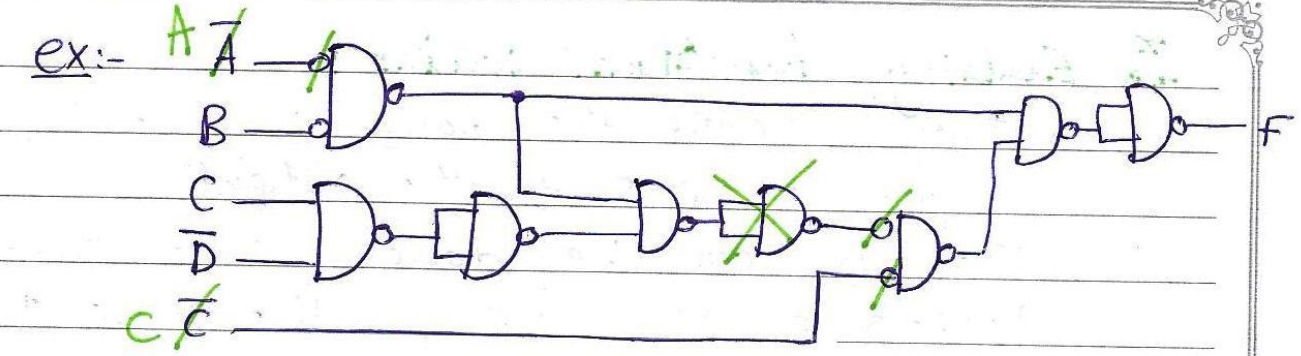
1. Replace AOI gates with their NOR equivelant.
2. Repeat until there's one inverter per wire.
   i) push inverters through circuit Fan-out points.
   ii) cancel inverter pairs.

ex:-





ex:-



* CH 3 - Part II

* combinational functional Blocks:-

* They are circuits that implement useful & common operations.
* come in SSI, MSI, LSI, VLSI forms.

Small Scale Integrated circuit

← يزيد عدد الترانزستورز

→ Examples:

- Decoders.
- Encoders.
- Multiplexers.
- Demultiplexers.

1) Decoders:-

- it's a combinational logic circuit that converts n-bit input to m-bit output, such that $n \leq m \leq 2^n$
- decoder functions are usually called n-to-m line decoder.

↳ Redumentary functions:-

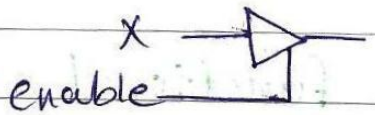
i) basic functions of 1 variable.

- $F = x$ transfer
- $F = \overline{x}$ inverting
- $v = 0$
- $v = 1$

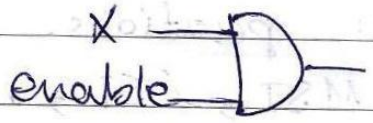
] value fixing

** Enabling functions / circuits :

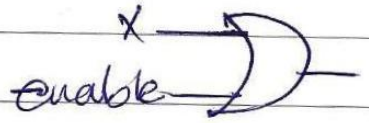
* They permit the value of the input signal to pass to the output.
* Enabling can be done using tri-state buffers, or value-fixing the value to (0) or (1)



if enable == 1
x passes through.

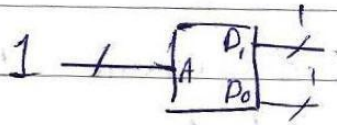


fixed-value; x passes through if enable == 1



x passes through if enable == 0

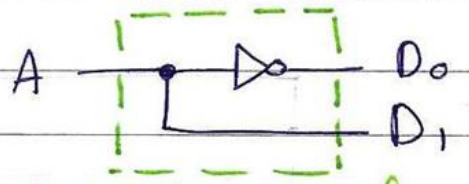
* 1-to-2 line decoder:-



A	D_1	D_0
0	0	(1)
1	(1)	0

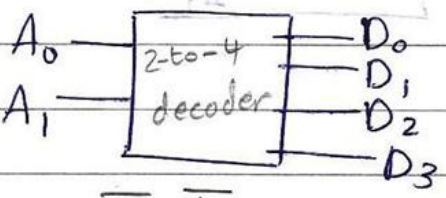
* when D has a subscript value equal to the decimal value of A, then its output value will be == 1
* at any moment of time only one output will be 1.

$D_0 = \overline{A}$
$D_1 = A$
from T.T



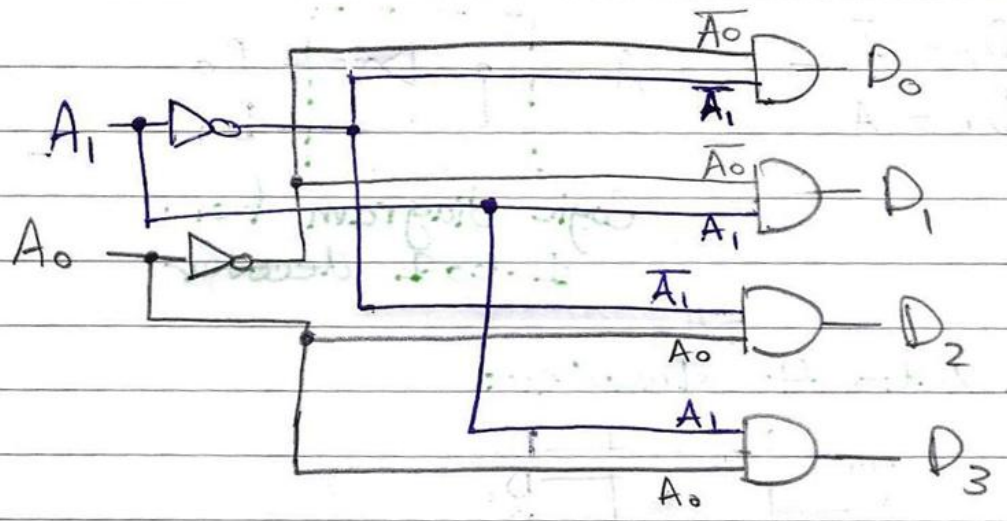
Logic diagram for 1-to-2 decoder

* 2-to-4 decoder:



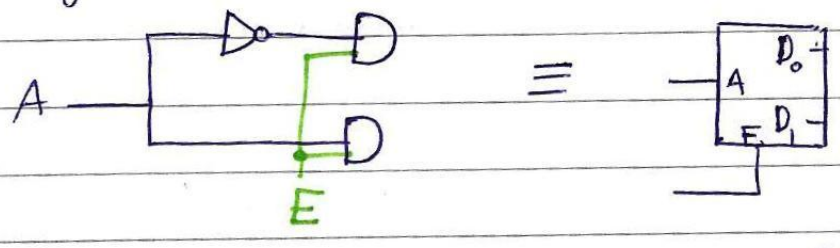
A_0	A_1	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

$D_0 = \overline{A_0}\,\overline{A_1}$ → لا يوجد اختصار للدائرة
$D_1 = \overline{A_0}\,A_1$ لأنه لدينا 1 واحدة فقط
$D_2 = A_0\,\overline{A_1}$ في كل حالة ⇐ لا يوجد
$D_3 = A_0\,A_1$ أي اختصار ممكن.



* Building larger decoders:

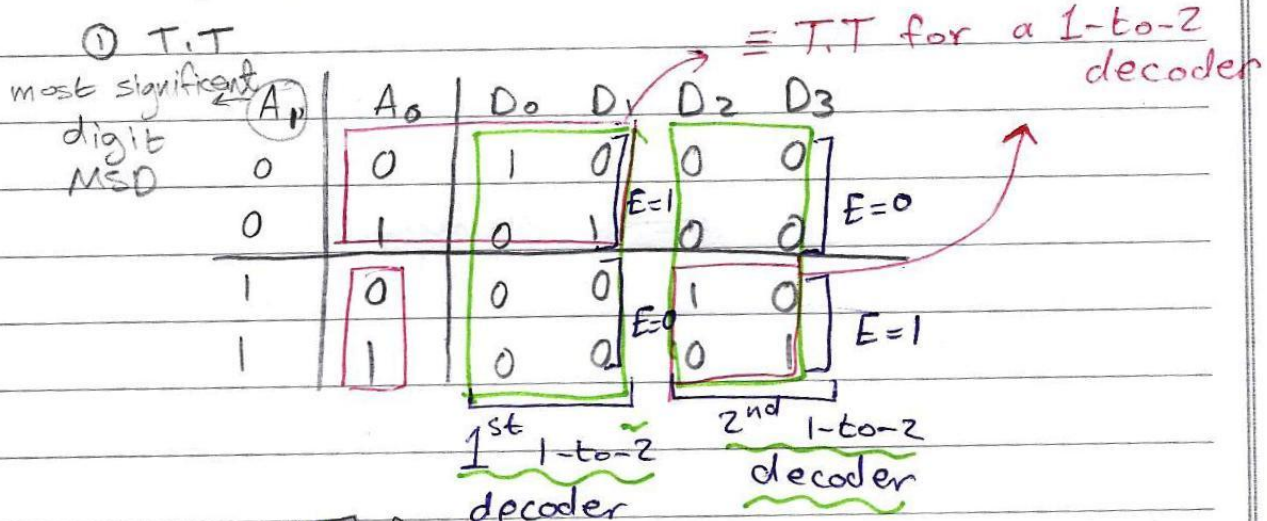
1) Decoder expansion. (on slides).
2) Using small decoders with enable.
   * building 1-to-2 decoders with enable:

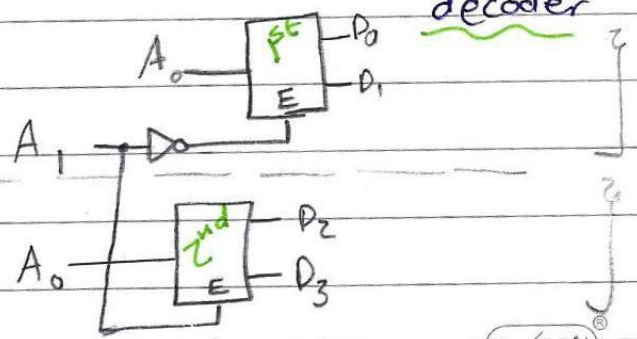


* when $E=0$, decoder is disabled ($D_1=D_0=0$)
* when $E=1$, decoder is enabled; $D_0=\bar{A}$, $D_1=A$

ex:- Build a 2-to-4 decoder using 1-to-2 decoders, invertors
by default: with enable

① T.T





enabled only when $A_1=0$

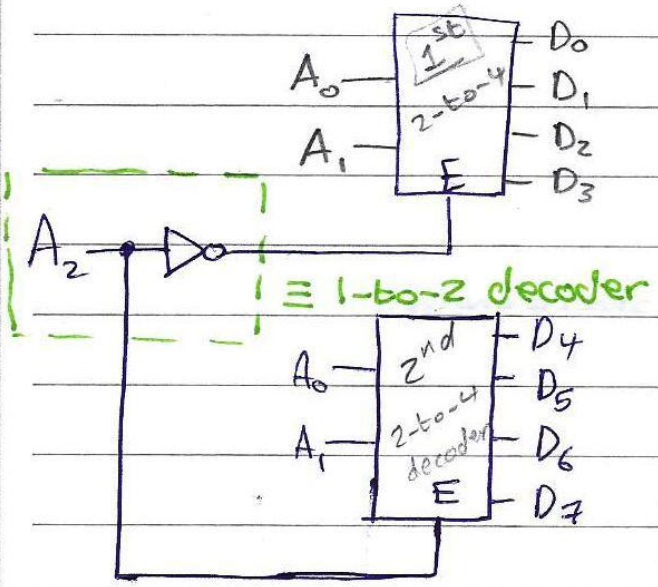
enabled only when $A_1=1$

ex: Build 3-to-8 decoder using two 2-to-4 decoders & invertors.

T.T for a 2-to-4 decoder ≡ أقل عدد يمكن استخدامه من الـ 2 = decoders

A_2	A_1	A_0	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
0	0	0	1	0	0	0	0	0	0	0	enabled when $A_2=0$
0	0	1	0	1	0	0	0	0	0	0	
0	1	0	0	0	1	0	0	0	0	0	
0	1	1	0	0	0	1	0	0	0	0	
1	0	0	0	0	0	0	1	0	0	0	enabled when $A_2=1$
1	0	1	0	0	0	0	0	1	0	0	
1	1	0	0	0	0	0	0	0	1	0	
1	1	1	0	0	0	0	0	0	0	1	

1st decoder | 2nd decoder



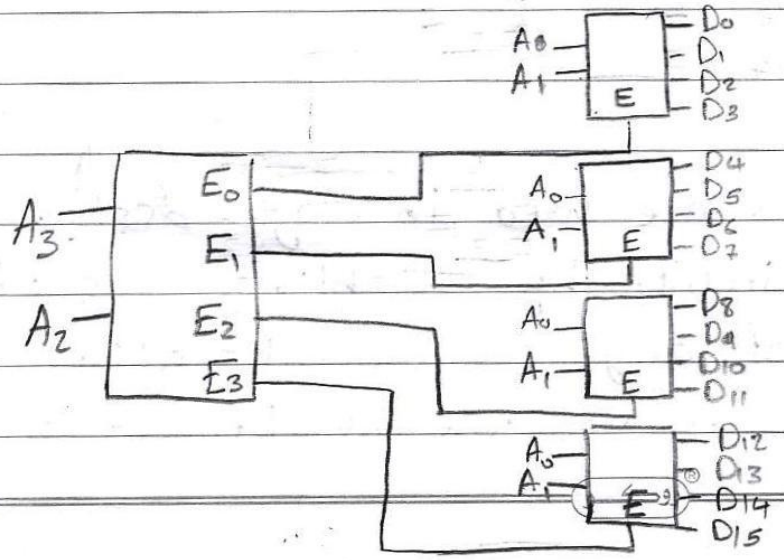
* Note:- decoders are used in logic design to control circuits by enabling one of them only at any moment.
* using decoders reduces the number of wires and the cost.

ex:- Build 4-to-16 decoder using 2-to-4 decoders & inverters.

	A_3	A_2	A_1	A_0	D_{15} D_{14} D_{13} D_{12} D_{11} D_{10} D_9 D_8 D_7 D_6 D_5 D_4	D_3 D_2 D_1 D_0
	0	0	0	0		0 0 0 1
	0	0	0	1		0 0 1 0
$E_0=1$ ←	0	0	1	0		0 1 0 0
	0	0	1	1		1 0 0 0
$E_1=1$ ←	0	1	0	0	0 0 0 1	
	0	1	0	1	0 0 1 0	
	0	1	1	0	0 1 0 0	
	0	1	1	1	1 0 0 0	
$E_2=1$ ←	1	0	0	0	0 0 0 1	
	1	0	0	1	0 0 1 0	
	1	0	1	0	0 1 0 0	
	1	0	1	1	1 0 0 0	
$E_3=1$ ←	1	1	0	0	0 0 0 1	
	1	1	0	1	0 0 1 0	
	1	1	1	0	0 1 0 0	
	1	1	1	1	1 0 0 0	

Disabled = 0

Disabled = 0

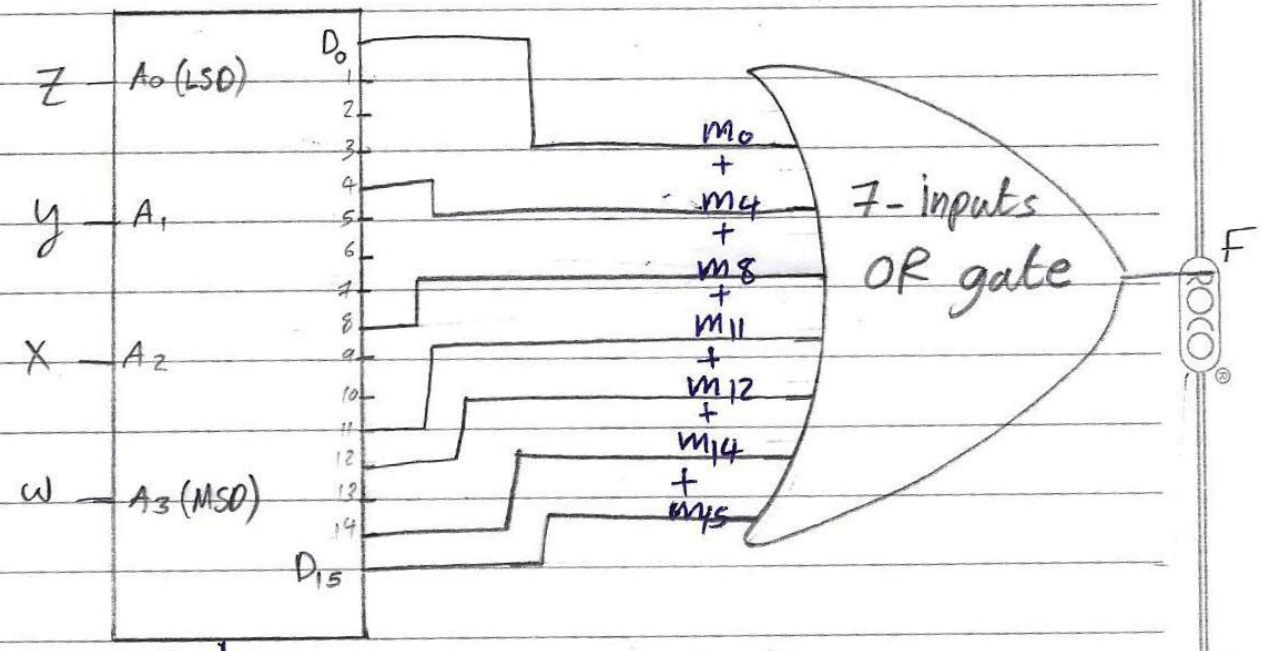


*Implementing combinational logic using decoders:-

*we can implement any function of (n)-variables using a n-to-m line decoder and k-input OR gate, where (K) is the number of (1)s in the function.

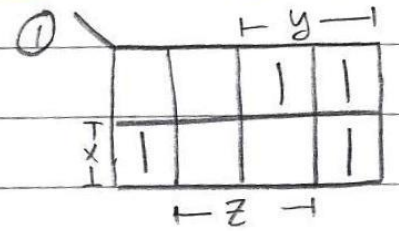
ex:- $F(w,x,y,z) = \sum_m (0,4,8,11,12,14,15)$

no. of $(1)_s = 7$



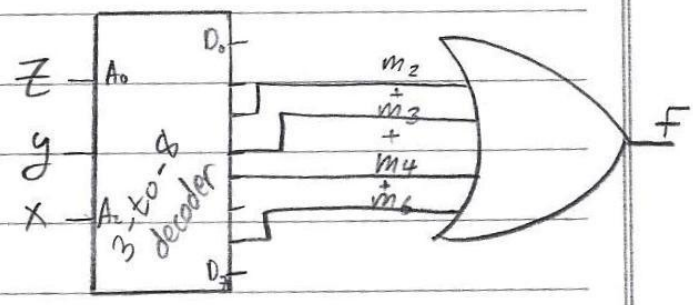
it's called a (minterms generator)

ex:- $F(x,y,z) = x\bar{z} + y\bar{x}$



$\therefore F = \sum_m (2,3,4,6)$

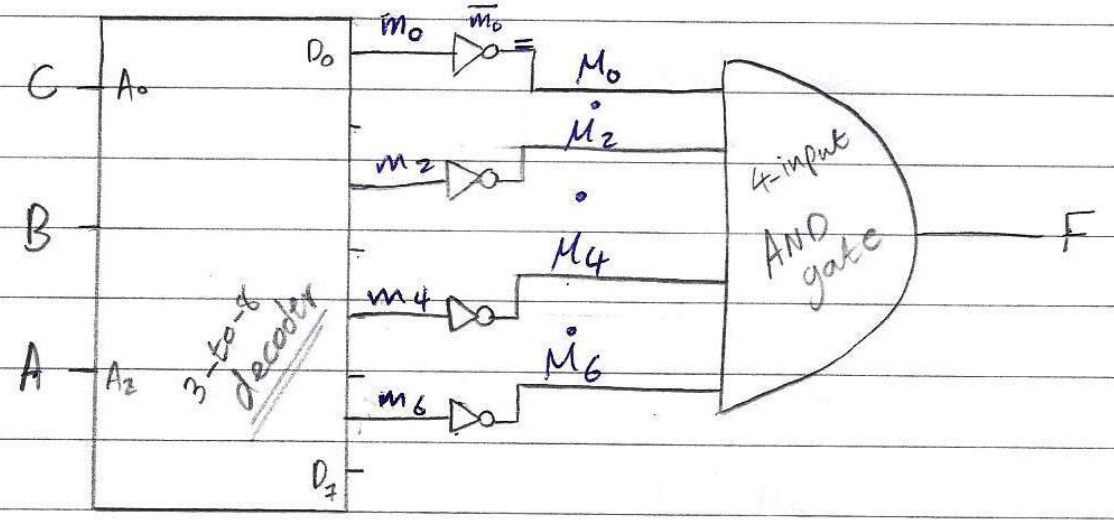
No. of $(1)_s = 4$



ex: Build $F(A,B,C) = \sum_m(1,3,5,7)$ using 3-to-8 decoder, AND gates, & inverters.

POS $F(A,B,C) = \prod_M(0,2,4,6) = M_0 \cdot M_2 \cdot M_4 \cdot M_6$

* $\overline{m} = M$



* since the decoder is a (minterm generator), we complement minterms in order to get Max-terms.

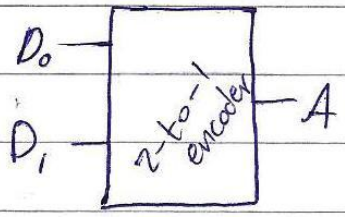
2) Encoders :-

* it's just the oppisite of the decoders.
* it converts m-bit binary input code into n-bit binary output code; such that:

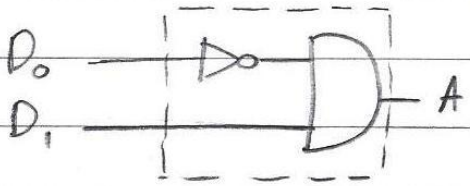
$$n \leq m \leq 2^n$$

* it converts the input value that has exactly single 1 into a binary code that corresponds to the possition of the 1.

ex:- 2-to-1 encoder (assuming that input value has only single 1)

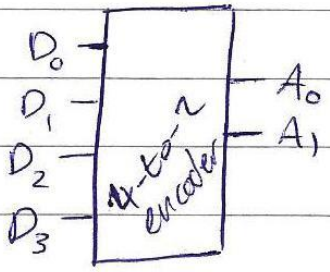


D_1	D_0	A
0	0	X ← invalid combinations
0	1	0
1	0	1
1	1	X



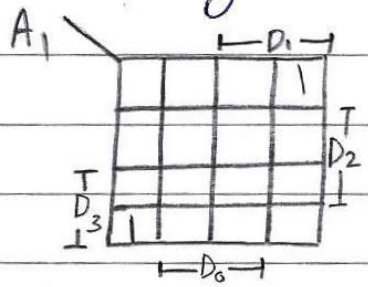
2-to-1 encoder

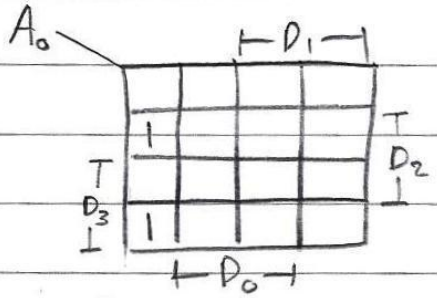
4-to-2 encoder



D_3	D_2	D_1	D_0	A_1	A_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

#1: Using K-maps





$A_1 = \bar{D}_3 D_2 \bar{D}_1 \bar{D}_0 + D_3 \bar{D}_2 \bar{D}_1 \bar{D}_0$

$A_0 = \bar{D}_3 \bar{D}_2 D_1 \bar{D}_0 + D_3 \bar{D}_2 D_1 \bar{D}_0$ → near optimal

#2: from T.T

نلاحظ من الجدول أن الفرق بين الـ minterms التي لها فيها 1 هو أكثر من bit
∴ لا فائدة من استخدام (K-maps).

من الجدول مباشرة :-

$$\left.\begin{aligned} A_1 &= D_2 + D_3 \\ A_0 &= D_1 + D_3 \end{aligned}\right] \text{Optimal}$$

* Priority Encoder:-

- what if the input has more than one (1)?!

  1010 → 01 : low priority encoder (Priority for LSD)

  ↘ 11 : high priority encoder (" " MSD)

  → Assign the output to a value that corresponds to the 1 in the lowest/highest possition.

- what if the input has no (1)s ?!

  → define an output to endecate wether the input is valid or not.

ex:- 4-to-2 high priority encoder.

1.

D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	1
0	0	0	1	0	0	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	0	0
0	1	0	1	1	0	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	1	1	0
1	0	0	1	1	1	0
1	0	1	0	1	1	0
1	0	1	1	1	1	0
1	1	0	0	1	1	0
1	1	0	1	1	1	0
1	1	1	0	1	1	0
1	1	1	1	1	1	0

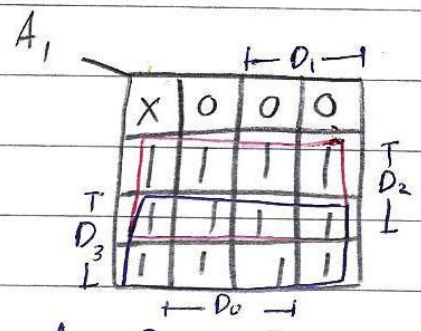
001X

01XX

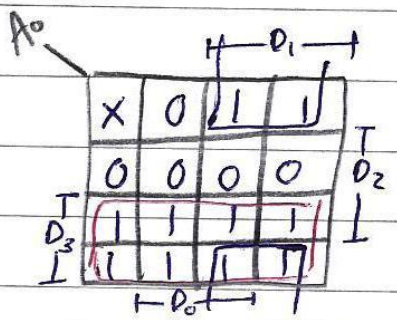
1XXX

D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	1
0	0	0	1	0	0	0
0	0	1	X	0	1	0
0	1	X	X	1	0	0
1	X	X	X	1	1	0

=



$A_1 = D_3 + D_2$



$A_0 = D_3 + \overline{D_2} D_1$

$V = \overline{D_3}\,\overline{D_2}\,\overline{D_1}\,\overline{D_0}$

H.W

*ex:- Write a T.T for a 5-to-3 high Priority encoder in compact form.

A_4	A_3	A_2	A_1	A_0	D_2	D_1	D_0	Error
0	0	0	0	0	X	X	X	1
0	0	0	0	1	0	0	0	0
0	0	0	1	X	0	0	1	0
0	0	1	X	X	0	1	0	0
0	1	X	X	X	0	1	1	0
1	X	X	X	X	1	0	0	0

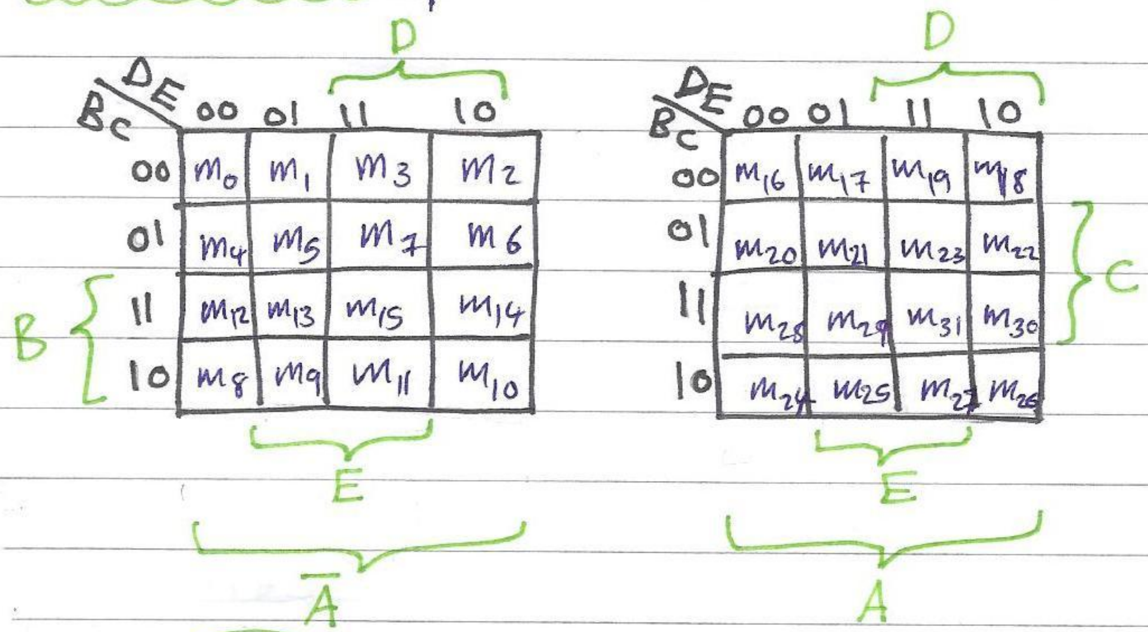
$D_2 = A_4$

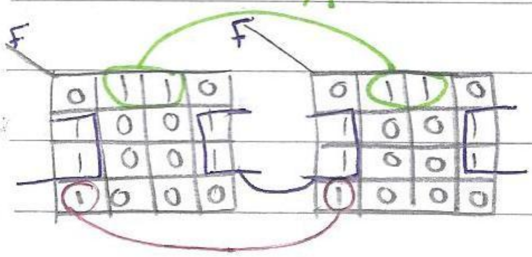
$D_1 = \bar{A}_4 A_3 + \bar{A}_4 A_2$

مافي داعي نحط $\bar{A}_3$ لأنو حتى لو كانت $A_3 = 1$ و $A_4 = 0$ فالناتج = 1.

$D_0 = \bar{A}_4 \bar{A}_2 \bar{A}_1 + \bar{A}_4 A_3$

* 5-variable k-map

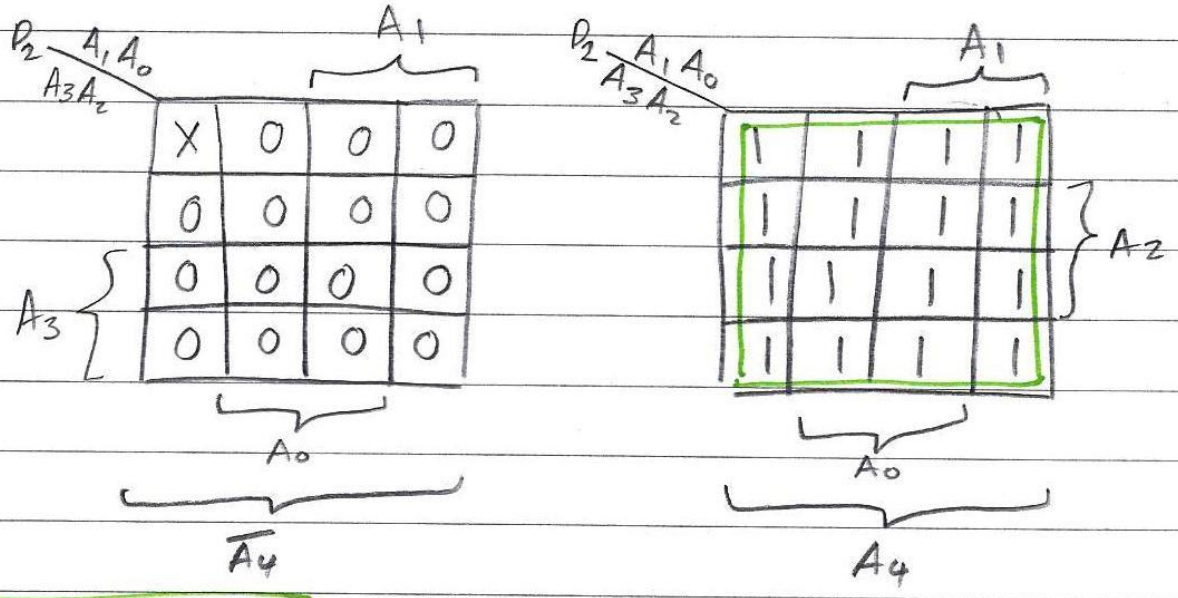




*examples of Prime Implicant (PI)'s in a 5-variables k-map.

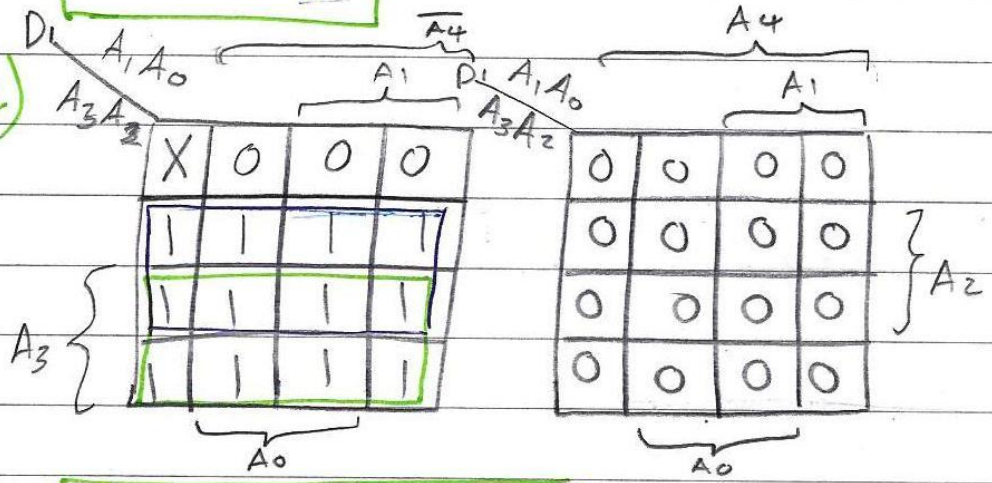
for the Previous ex :-

1)



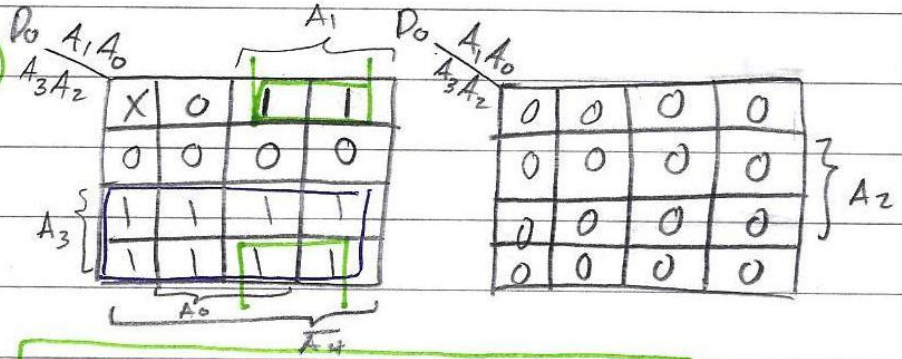
$$D_2 = A_4$$

2)



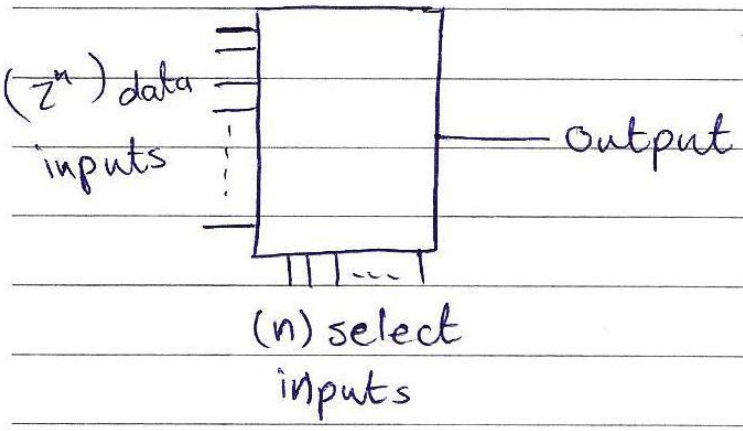
$$D_1 = \overline{A_4}\, A_3 + \overline{A_4}\, A_2$$

3)



$$D_0 = \overline{A_4}\, A_3 + \overline{A_4}\, \overline{A_2}\, A_1$$

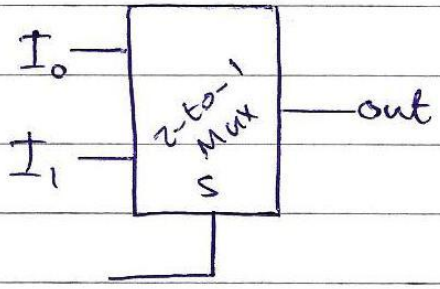
3) Multiplexers (selectors):-



* Muxes Allow sharing one resourse & reduce the cost by reducing number of wires.

* it's a circuit that can be select 1 out of (2^n) inputs. to pass to the output using (n) select inputs.

⇝ 1-to-1 Mux:



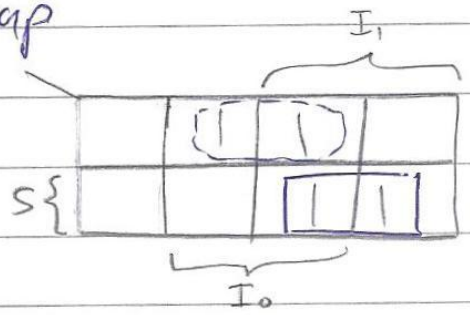
* T.T

S	I_1 I_0	out	
0	0 0	0	I_0
0	0 1	1	
0	1 0	0	
0	1 1	1	
1	0 0	0	I_1
1	0 1	0	
1	1 0	1	
1	1 1	1	

|||

S	out
0	I_0
1	I_1

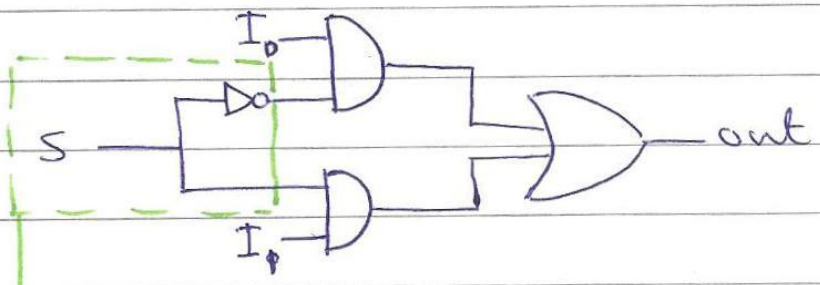
* k-map



$Out = S I_1 + \bar{S} I_0$

*** 2-to-1 Mux:**

- 1-to-2 decoder
- 2 (2-inputs AND)
- 1 (2-inputs OR)

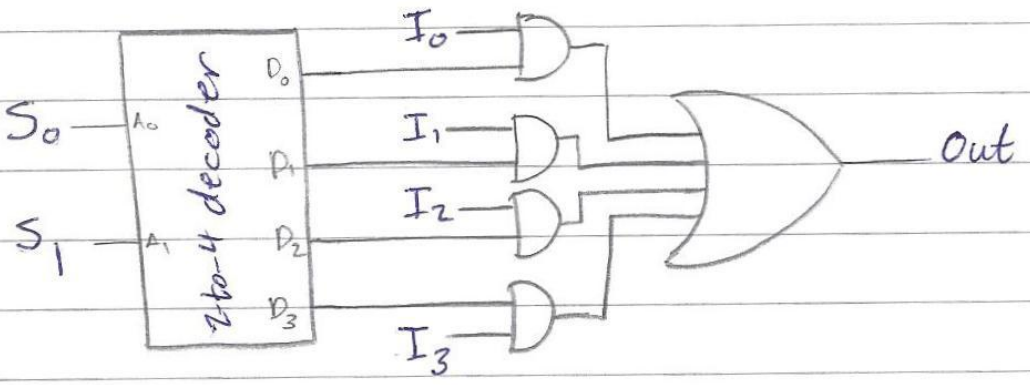


↳ 1-to-2 decoder

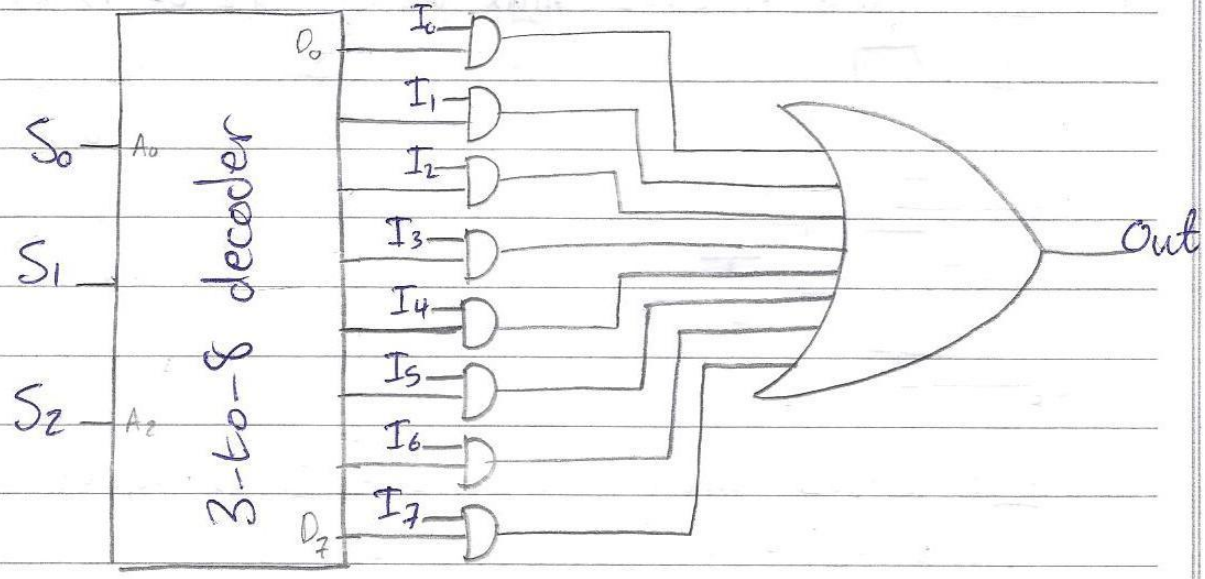
* To build 2^n-to-1 Mux, we need:-

- n-to-2^n decoder.
- 2^n (2-inputs AND).
- 1 (2^n-inputs OR)

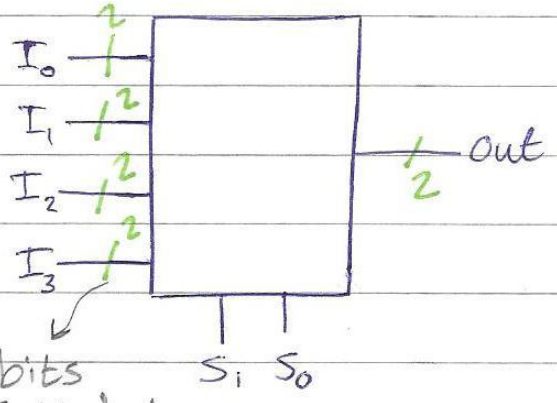
ex: 4-to-1 Mux:- (n = 2) → 2-to-4 decoder; → 4 (2-input AND); → 1 (4-input OR)



H.W 8-to-1 Mux ∴ n=3 → 3-to-8 decoder
→ 8 (2-input AND)
→ 1 (8-input OR)

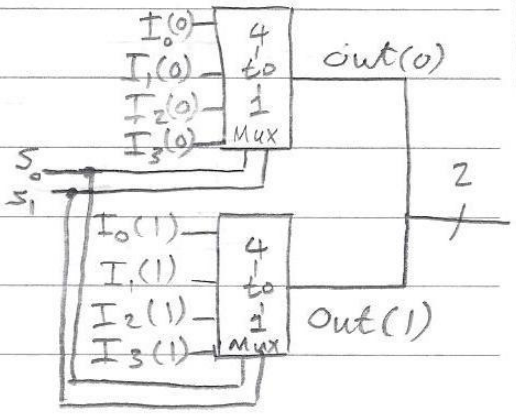


* Mux with expansion:-



Quad (4-to-1) Mux

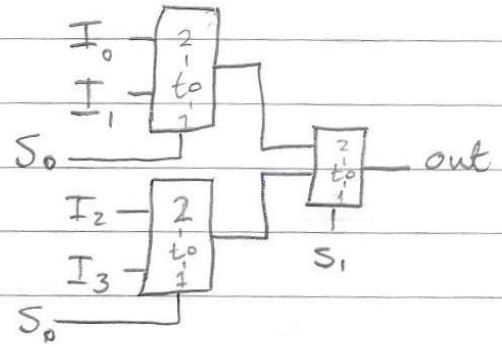
≡



2 (4-to-1) Mux

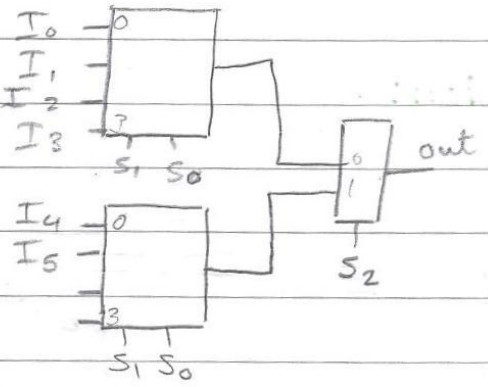
* Building large muxes from smaller ones:

ex:- build 4-to-1 mux using 3(2-to-1) muxes.



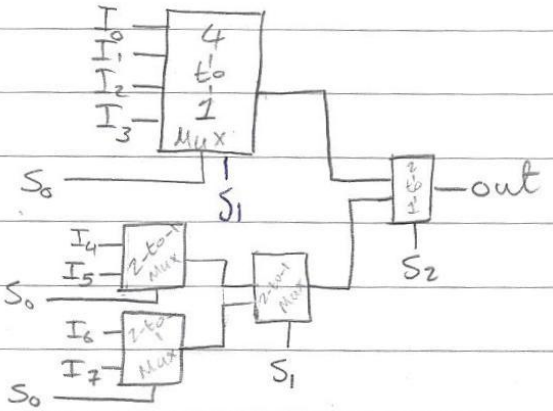
S_1(MSD)	S_0(LSD)	out
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

ex:- 6-to-1 mux using 2(4-to-1) Muxes & 1 (2-to-1) Mux.



S_2	S_1	S_0	out
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	X
1	1	1	X

ex:- 8-to-1 Mux using 1(4-to-1) Mux & the neceesary number of 2-to-1 Muxes



S_2	S_1	S_0	out
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

*Implementing logic functions using Muxes:

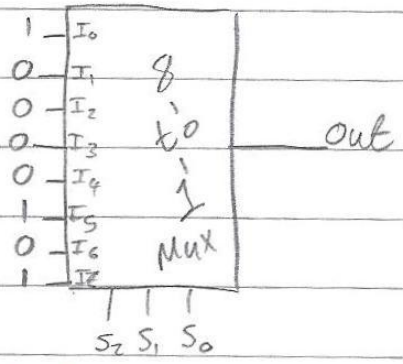
**Mux ≡ selective OR gate.

↳ Method #1:-

1. for a function of n-variables, we need 2^n-to-1 Mux.
2. Connect the input variables to the select lines.
3. Value-fix the Mux inputs to (0) or (1) depending on the input of select lines and function minterms.

ex: $F(x,y,z) = \sum_m(0,5,7)$

n=3 / 8-to-1 Mux



↳ Method #2:-

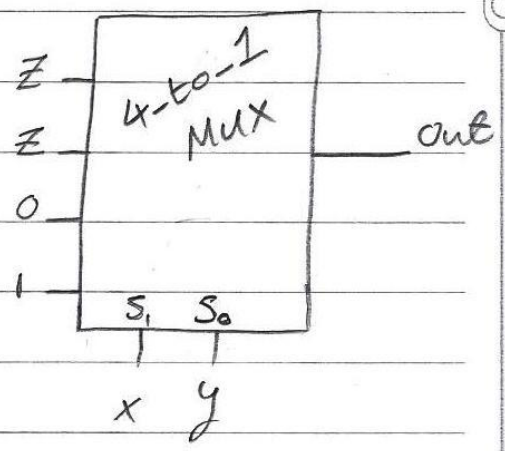
- for any function of n-variables, we can implement it using $(2^{(n-1)}$-to-1) Mux & an invertor (if needed).

* Steps:

1. generate T.T.
2. based on the value of the (n-1) most significent bits (MSB), seperate the raws of the table into pairs.
3. for each pair & function output write the redumentary function in terms of X, where X is the LSB.
4. Connect the (n-1) MSB to the select lines of the Mux.

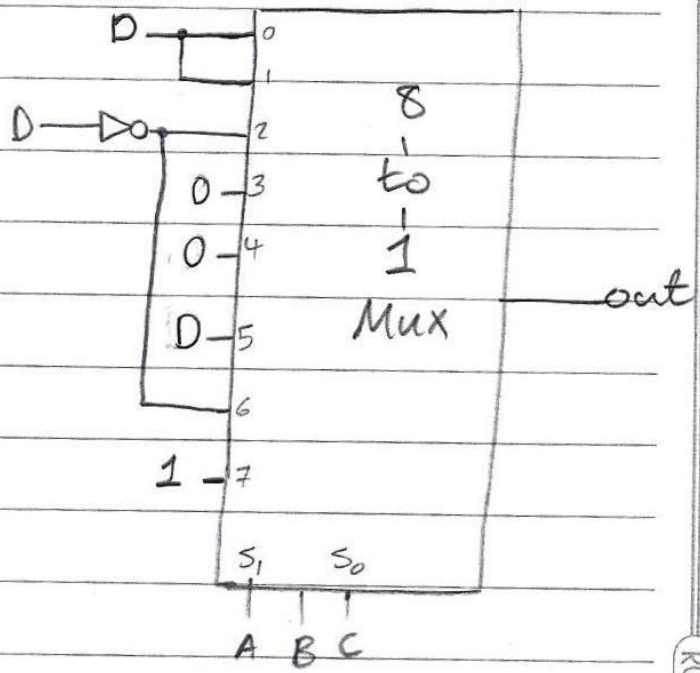
ex:- $F(x,y,z) = \sum_m (1,3,6,7)$, implement f using 4-to-1 Mux.

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



ex:- $F(A,B,C,D) = \sum_m (1,3,4,11,12,14,15)$

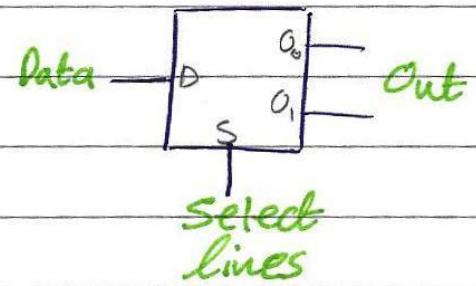
A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1



4) Demultiplexers (DMux):

* A circuit that performs the oppisite opperation of Muxes.
* It passes the signal input to one of the 2^n-outputs using n-select lines.

ex:- 1-to-2 DMux:-

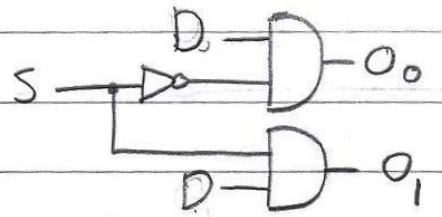


S	D	O_1	O_0
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

$O_1 = m_3 = SD$

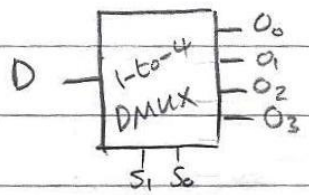
$O_0 = m_1 = \bar{S}D$

* DMux $\overleftarrow{\equiv}$ decoder with Enable



$\equiv$ decoder with E

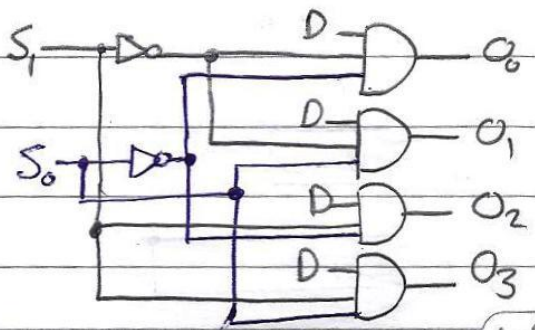
H.W:- build 1-to-4 DMux:-



①

S_1	S_0	D	O_0	O_1	O_2	O_3
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	0
1	1	1	0	0	0	1

③



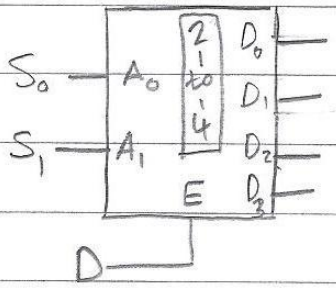
② $O_0 = \bar{S_1}\bar{S_0}D$, $O_1 = \bar{S_1}S_0D$

$O_2 = S_1\bar{S_0}D$, $O_3 = S_1S_0D$

- we can use n-to-m decoder with enable to implement 1-to-m DMux

  data ≡ E / S ≡ input

ex :-



≡ 2-to-4 decoder with E

≡ 1-to-4 DMux

عنوان الدرس ..... اليوم الخميس ..... التاريخ 5/7/

* CH# 4: Arithmatic logic :-

• Binary addition

$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0^{+} & 0^{+} & 1^{+} & 1^{+} \\ \hline 0 & 1 & 1 & 10 \end{array}$$

ex:-
$$\begin{array}{r} {\scriptstyle 1\ 1\ 1\ 1\ \ \ } \\ 10111 \\ +00101 \\ \hline 11100 \end{array}$$

• Binary Subtraction

$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ 0^{-} & 1^{-} & 0^{-} & 1^{-} \\ \hline 0 & -1\,! & 1 & 0 \end{array}$$

ex:
$$\begin{array}{r} 011 \\ -010 \\ \hline 001 \end{array} \qquad \begin{array}{r} 101 \\ -110 \end{array} \equiv \begin{array}{r} {\scriptstyle 0\ 2\ \ } \\ 1\not{0} \\ -101 \\ \hline -001 \end{array}$$

← In decemal when we borrow we add $\underline{\underline{10}}$ to the number = r (base)

$5-6 \equiv -(6-5)$

* In binary r = 2 ∴ we add $\underline{\underline{2}}$

** Signed numbers representation:-

* we need to define a notation to represent signed numbers.
* we can add an extra bit for the sign.
* In general, there are two approaches :-
  1. Signed - magnitude.
  2. Signed complement.

↳ 1. Signed – magnitude:

1	n - 1
1-bit Sign {0, +ve; 1, -ve}	magnitude.

ex: list all 3-bit signed numbers that are represented using signed-mag.

Sign	mag	mag	
A_2	A_1	A_0	Num.
0	0	0	+0
0	0	1	+1
0	1	0	+2
0	1	1	+3 ⇒ Max +ve value = $+3 \equiv 2^{3-1}-1$
1	0	0	−0
1	0	1	−1
1	1	0	−2
1	1	1	−3 ⇒ Min −ve value = $-3 \equiv -(2^{n-1}-1)$

⇝ Note:: for a n-bit signed number in signed-magnitude notation:-

- Max +ve value is $2^{n-1}-1$
- Min −ve value is $-(2^{n-1}-1)$

* Challanges:

- +ve & −ve zeros.
- Hardware implementation is expensive & complex.

in decimal: $+3 + (-3) = 0$ in sign-mag: $011 + 111 = 1010$

↳ 2. Signed complement

↳ Diminished radix complement. ($\tilde{N}$)

↳ Radix complement (N^*) (Most afficient approach)

a. Diminished radix complement:

* for a number (N) that is represented using (n) digits in base 'r', the diminished radix complement is given by:-

$$\tilde{N} = r^n - 1 - N$$

n: number of bits used to represent the number N.
r: base.

* $\tilde{N} + N = \underbrace{(r-1)(r-1)\dots}_{n\text{-bits}}$ ⇒ it's called (r-1)'s complement

ex:- $(18)_{10}$, find $\tilde{N}$?

$N = 18$, $r = 10$, $n = 2$ bits (in decimal)

$\tilde{N} = 10^2 - 1 - 18 = (81)_{10}$

$N + \tilde{N} = 18 + 81 = (99)_{10}$

ex:- $(23)_{10}$, find $\tilde{N}$?

$N = 23$, $r = 10$, $n = 2$

$\tilde{N} = 10^2 - 1 - 23 = (76)_{10}$

$N + \tilde{N} = 23 + 76 = (99)_{10}$

⇒ 9's complement

ex:- $N = (01)_2$, find $\tilde{N}$?

$\tilde{N} = 2^2 - 1 - 1 = (2)_{10} \equiv (10)_2$

* adding 1-bit for the sign:-

$(001)_2 \Rightarrow$

$\tilde{N} = 2^3 - 1 - 1 = (6)_{10} \equiv (110)_2$ (-ve)

⇒ 1's complement.

⇒ for this example, if we assume $(10)_2$ is the -ve of $(01)_2$, then how can we distinguish $(-1)_{10} \equiv (10)_2$ & $(2)_{10} \equiv (10)_2$?!

* we can add/append a sign bit to the number before computing the complement.

** +ve numbers are always the same.

ex:- $(01)_2 \Rightarrow (001)_2 \xrightarrow{1's} (110)_2$

$(10)_2 \Rightarrow (010)_2 \xrightarrow{1's} (101)_2$

* list all 3-bit signed numbers using 1's complement.

Sign A_2	A_1	A_0	Value
0	0	0	+0
0	0	1	+1
0	1	0	+2
0	1	1	+3
1	0	0	-3
1	0	1	-2
1	1	0	-1
1	1	1	-0

⇝ Note: for a n-bit signed 1's complement number:-

- Max +ve value is $2^{n-1}-1$
- Min -ve value is $-(2^{n-1}-1)$

* Challanges:-

- +ve, -ve zeros.
- Hardware is complex.

in Decimal: $3 + (-2) = 1$

in 1's com.: $011 + 101 = 1000$ (carry) → 1^+ → 001 ✓

** adjusting results in 1's complement is easier than signed-magnitude.

* Shortcut:

* to compute the 1's complement for a number, just complement indivisual bits!

b. Radix Complement (most afficient approach).

* for a number N represented using (n) digits in base r, the radix complement is given by:-

$$N^* = r^n - N$$

$$N + N^* = \underbrace{\text{carry}}_{1}\ \underbrace{0\,0\,\cdots}_{(n-1)\text{bits}} \Rightarrow (r)\text{'s complement}$$

base

ex:- $N = (18)_{10}$, N^*?

$N^* = 10^2 - 18 = 82$

$82 + 18 = \overset{\text{carry}}{1}00$

} 10's complement

ex: $N = (01)_2$, N^*?

$N^* = 4 - 1 = (3)_{10} \equiv (11)_2$ } 2's complement

but... how to distinguish between $(-1)_{10} \equiv (11)_2$ & $(3)_{10} \equiv (11)_2$?!

→ append a sign bit before computing the complement.

⇒ $N = (01)_2 \longrightarrow (001)_2$

$N^* = 2^3 - 1 = (7)_{10} \equiv (111)_2$

* now we'll have 3 bits, one of them is assigned to represent the sign of the number and the other 2 are to represent the magnitude. and since $(7)_{10}$ can't be represented in 2 binary digits, we can distinguish between

$(+7)_{10} = (0111)_2$ & $(-1)_{10} \equiv (111)_2$

* list all 3-bit signed 2's complement numbers.

Sign A_2	mag A_1 A_0	Value
0	0 0	+ 0
0	0 1	+ 1
0	1 0	+ 2
0	1 1	+ 3
1	0 0	- 4
1	0 1	- 3
1	1 0	- 2
1	1 1	- 1

* there's no -0 in 2's complement.

* $(100)_2 = -4$

$\equiv 0*2^0 + 0*2^1 - 1*2^2$

$= -4$

→ Arithmetic:

ex ① in decimal: 3 + (-3) = 0

in 2's comp: 011 + 101 = 000 (carry)

ex ② in decimal 3 + (-2) = 1

in 2's comp: 011 + 110 = 001 (carry)

⇝ Note:- for a n-bit signed 2's complement number :-

max +ve value is $2^{n-1} - 1$

min -ve value is $-(2^{n-1})$

ex:- find the 2's complement of the following numbers using 6 bits to represent your solution.

1) -13

method # 1: $(13)_{10} = (001101)_2 \xrightarrow{2's} (110011)_2 \equiv (-13)_{10}$

~ # 2: $(13)_{10} = (001101)_2 \xrightarrow{1's} (110010)_2 \xrightarrow{+1} (110011)_2$ #

*Shortcut:

※1: compute the 1's complement ~~and~~ then just add 1.

$N^* = \tilde{N} + 1$

※2: start from right to left, keep the (0)s up to the first 1, then complement the remaining bits.

2) −26

$(26)_{10} = (011010)_2 \xrightarrow{2's} (100110)_2$

للتأكد:-

```
 1111
 100110
+011010
 ------
X000000   ✻
carry
```

ex:- show the binary representation of the following numbers in the signed-mag, 1's complement & 2's complement, assume 5-bits.

Number	Sign-mag	1's	2's
+12	01100	01100	01100
−7	10111	+7 = 00111 → 11000	+1 → 11001
−11	11011	10100	10101
19	X (over-flow)	X	X
−16	X	X	10000

** for any n-bit signed 2's complement number, the decimal value is given by:-

$$\text{Value} = \sum_{i=0}^{n-2} 2^i * b_i + (-1)^S * 2^{n-1} * b_{n-1}$$

(b_{n-1}) b_{n-2}

2's comp ⇒ [1 bit] [(n-1) bits]

sign magnitude

ex: $(100111)_2 = (\ !?\ )_{10}$

considering it as 2's complement signed number.

#1: $1*2^0 + 1*2^1 + 1*2^2 - 1*2^5 = (-25)_{10}$

#2: $N = -N^*$ (2's comp.)

$(100111)_2 \xrightarrow[\text{comp.}]{2's} (011001)_2 = (25)_{10}$

$\therefore N = (-25)_{10}$

Arithmetic using 2's complement:

↳ Addition:

* add individual bits and ignore the carry out of the MSB

ex: 15 + -20

$(15)_{10} \xrightarrow{2's} (001111)_2$

$(-20)_{10} \xrightarrow{2's} (20)_{10} = (010100)_2 \xrightarrow{2's} (101100)_2$

⇒ $15 + (-20) \equiv 001111 + 101100$

$= 1\,110\,11$ = (-5)$_{10}$

carry -ve

↳ Subtraction :-

* to perform M-N using 2's complement (-N) & add it to M!

ex:- 10 − 8 (5-bits)

$(10)_{10} \longrightarrow (01010)_2$

$(-8)_{10} \longrightarrow 8 = (01000)_2 \xrightarrow{2's} (11000)_2$

$$\begin{array}{rcl} 10 & & 01010 \\ +(-8) & \equiv & +11000 \\ \hline & & \overset{*}{\underset{\text{carry}}{1}}00010 \equiv (+2)_{10} \end{array}$$

Overflow (OF):

→ OF occurs when the result can't be represented using the available number of bits.

→ we need to detect overflow.

ex: 1. unsigned numbers (3-bits)

$$\begin{array}{r} 011 \\ 010 + \\ \hline 101 \end{array} \qquad \begin{array}{r} 100 \\ 101 + \\ \hline 1001 \end{array}$$

$$= OF$$

** In unsigned numbers overflow is detected if there's a carry = 1 out of the MSB.

ex:- 2. signed numbers (3-bits)...(2's comp).

$$\begin{array}{r} +ve\ 011 \\ +ve\ 001 + \\ \hline \not{=}\ 100 \\ carry \quad -ve!! \end{array} \qquad \begin{array}{r} -ve\ 101 \\ -ve\ 110 + \\ \hline carry\ 1011 \\ +ve \end{array}$$

Overflow!

** In signed numbers, overflow may occur when adding numbers of similar signs or subtracting numbers of differant signs.

* $OF = C_n$

① 0 mag + 0 mag = 1 mag → OF

② C_n | 1 (n-1) mag (n-2) + 1 mag = 0 mag, carry → OF

③ 1 mag + 1 mag = 1 mag → No OF

④ 0 mag + 0 mag = 0 mag → No OF

** to avoid overflow, carry into & out of the sign bit must be the same (both (0)s or both (1)s)

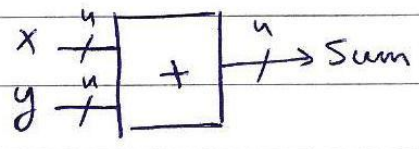
(t.t):

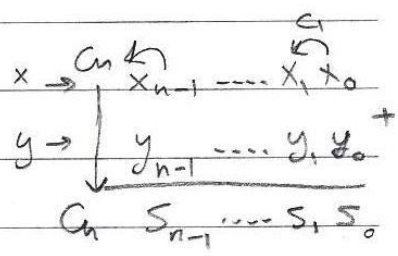
C_n	C_{n-1}	OF
0	0	0
0	1	1
1	0	1
1	1	0

$\Rightarrow OF(C_n, C_{n-1}) = C_n \oplus C_{n-1}$

* hardware (Hw) design:-

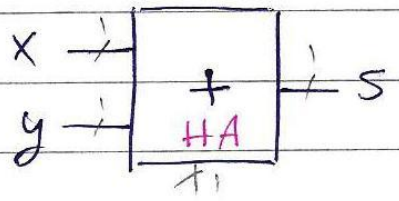
Adder: (n-bit)





*Iterative circuits: circuits designed using small similar circuits that perform simple opperations. these circuits are called cells.

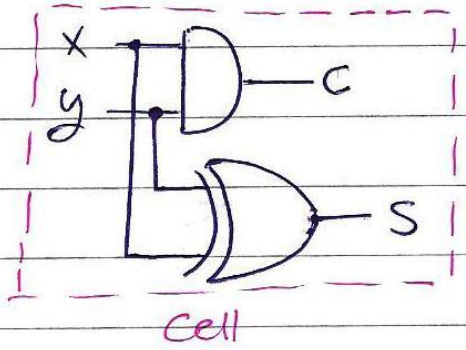
1-bit Adder



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

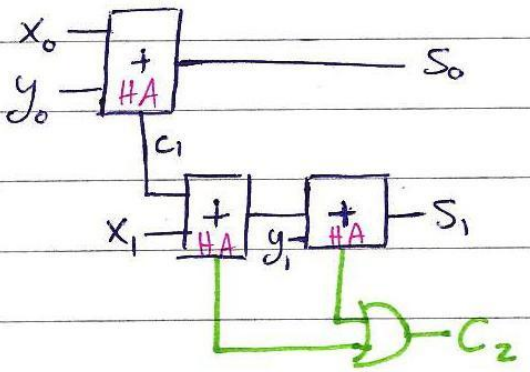
$C = x.y$

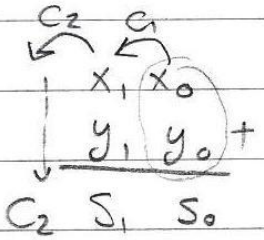
$S = x \oplus y$



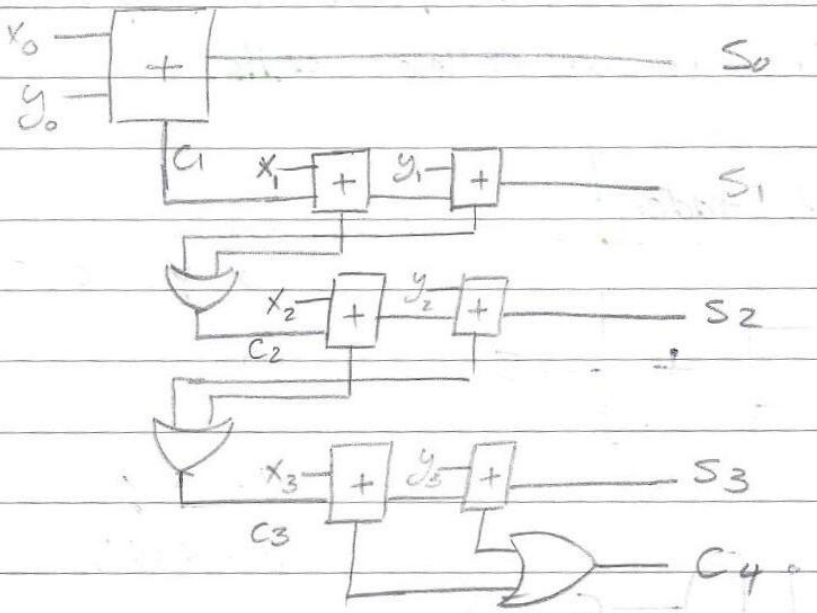
≡ Half Adder (HA)

2-bit Adder

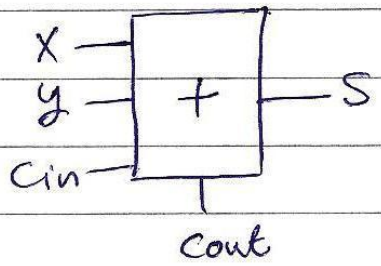




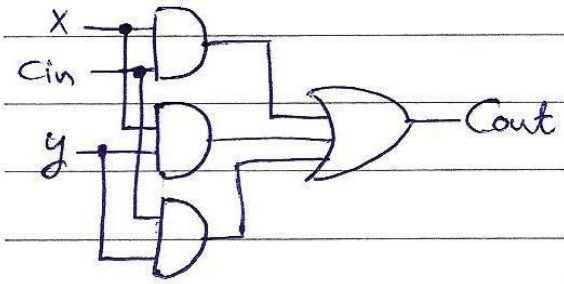
H.W design a 4-bit adder

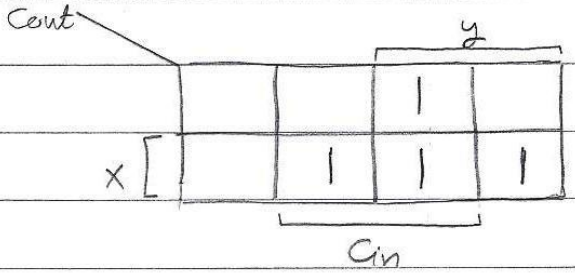


1-bit full adder

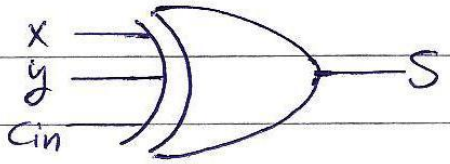


x	y	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



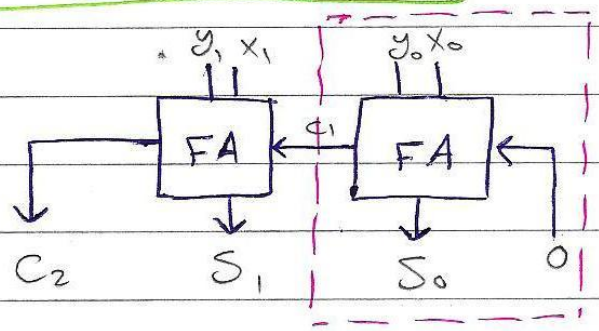


$Cout = xCin + xy + yCin$

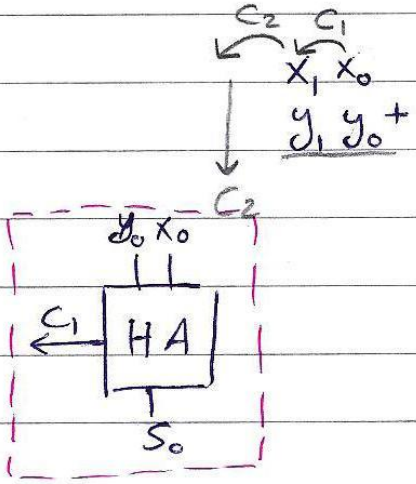


$Sum = x \oplus y \oplus Cin$

2-bit full adder

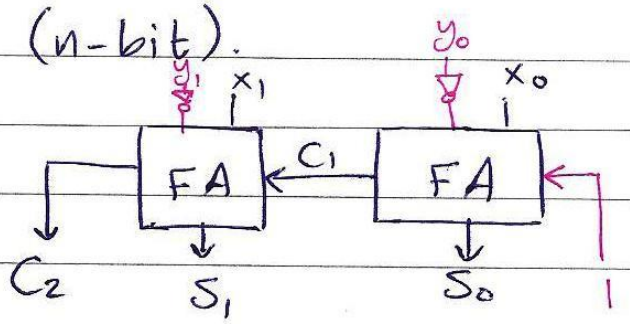


≡



((Ripple carry adder))

Subtractor (n-bit).



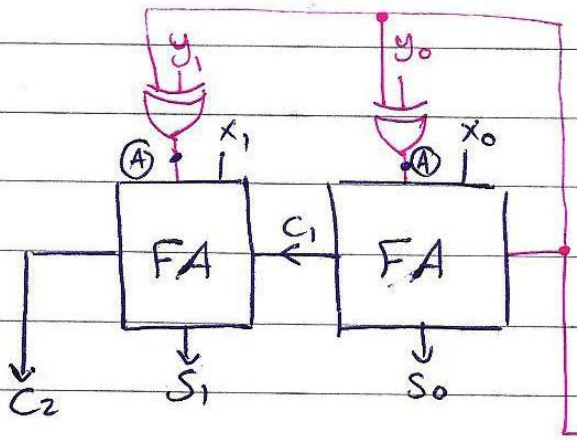
$$x - y \equiv x + (-y)$$

2's comp.

$$= x + \left(\overline{y_1}\,\overline{y_0} + 1 \right)$$

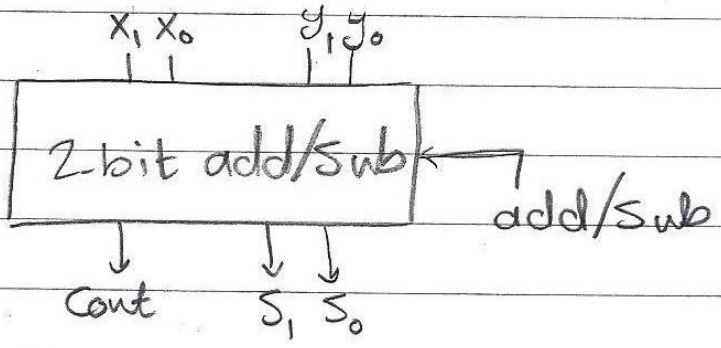
1's comp.

* Adder / Subtractor (2-bits)



** when $C_0 = 0$
at Ⓐ = y
when $C_0 = 1$
at Ⓐ = $\overline{y}$

111



* Multiplication & division:-

↳ 1. multiplication by powers of 2.

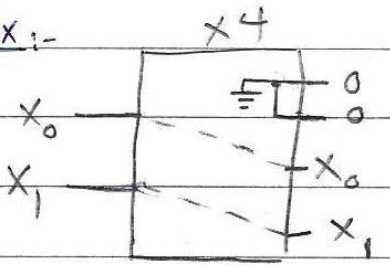
$1001 \xrightarrow{\times 2} 10010$

$1001 \xrightarrow{\times 4} 100100$

$1001 \xrightarrow{\times 8} 1001000$

- multiplication by 2^n is the same as shifting the number to the left by n-bits. مثل الضرب بـ 10 في الـ Decimal

ex:-



↳ 2. division by powers of 2.

$(10000)_{\text{unsigned}} = (16)_{10} \xrightarrow{\div 2} (01000)_2$

$(16)_{10} \xrightarrow{\div 4} (00100)_2$

$(16)_{10} \xrightarrow{\div 8} (00010)_2$

** it's an integar division:-

ex:- $0111 \xrightarrow{\div 2} 0011$

$(7)_{10} \xrightarrow{\div 2} (3)_{10}$

- for unsigned numbers, division by 2^n is the same as shifting the number to the right by n-bits.

$(-4)_{10} \equiv (1100)_2 \xrightarrow{/2} (\cancel{1}1110)_2$

$(1110)_2 = -2$

$(-4)_{10} \equiv (1100)_2 \xrightarrow{/4} (\cancel{00}11)_2$

$(1111)_2 = -1$

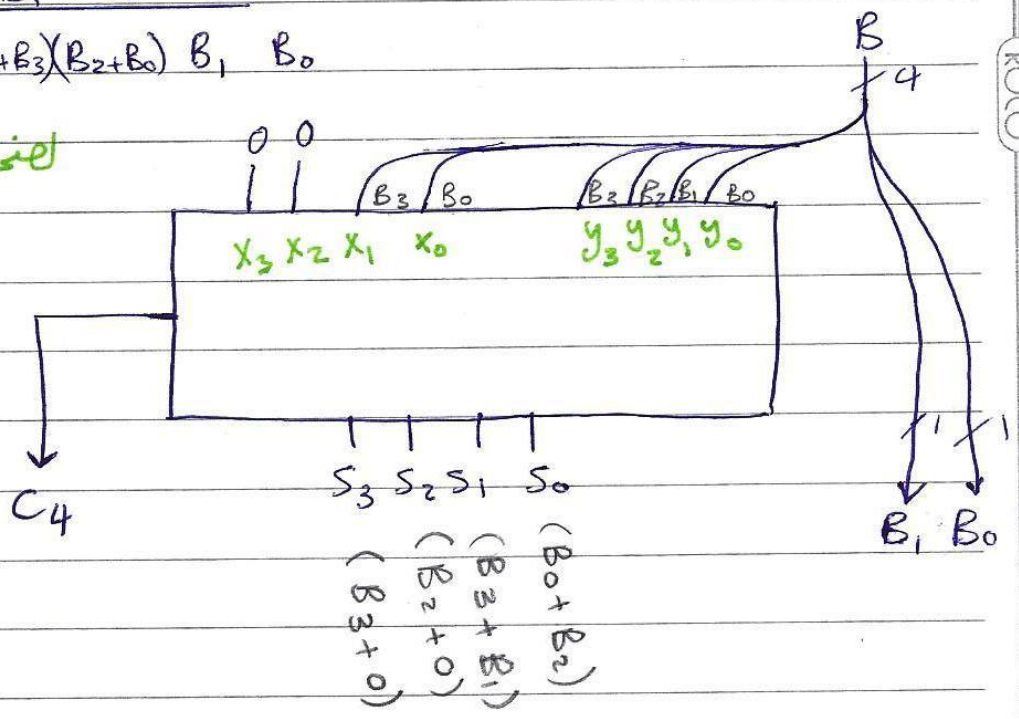
- for signed numbers, division by 2^n is the same as shifting to right by n-bits while preserving the sign of the number.

$$(0110)_2 \xrightarrow{\div 2} (0011)_2 \equiv (3)_{10}$$
$$\equiv (6)_{10}$$

↳ multiplication by a constant.

$$\begin{array}{cccc} B_3 & B_2 & B_1 & B_0 \\ & & 1 & 0 \quad 1 \quad \times \\ \hline B_3 & B_2 & B_1 & B_0 \\ 0 & 0 & 0 & 0 \quad 0 \\ B_3 & B_2 & B_1 & B_0 \\ \hline (B_3+0) & (0+B_2) & (B_1+B_3) & (B_2+B_0) \quad B_1 \quad B_0 \end{array}$$

لضمان عدم وجود Carry



H.W $B_3\ B_2\ B_1\ B_0$

1 1 0 1

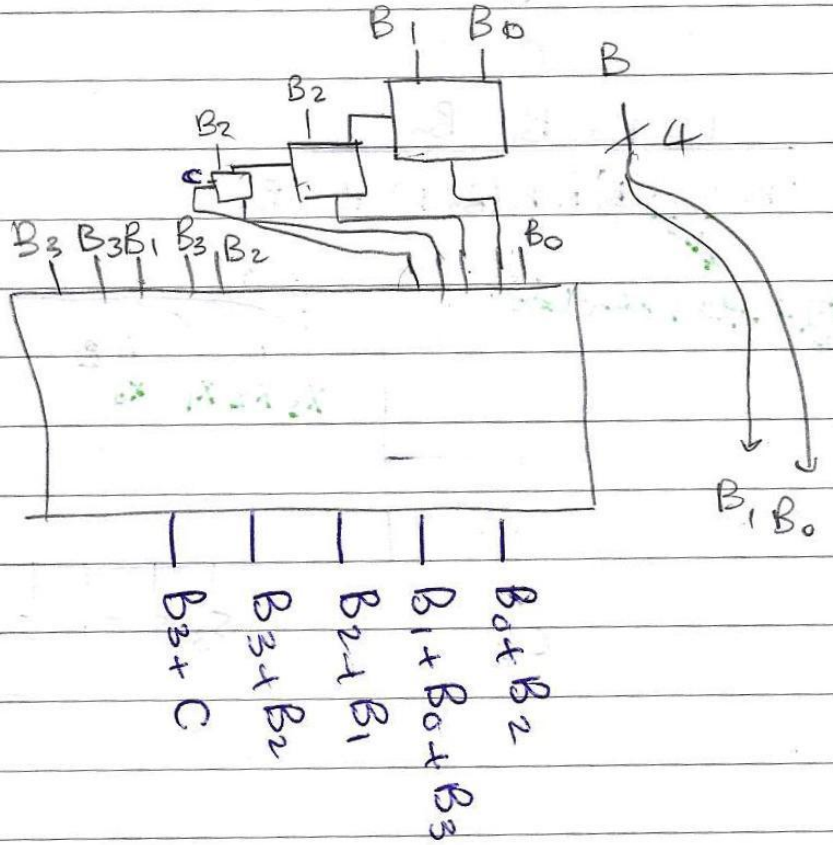
$B_3\ B_2'\ B_1\ B_0$

0 0 0 0

$B_3\ B_2\ B_1\ B_0$

$B_3\ B_2\ B_1\ B_0$

$B_3\left(B_3+B_2\right)\left(B_2+B_1\right)\left(B_1+B_0+B_3+C\right)\left(B_0+B_2\right)B_1\ B_0$



* Data width extension:

- HW units have fixed sizes for their inputs.
- the oprands size should be equal and match the size of the HW inputs.

⇝ there's two approaches to extend the data width.

1. Zero extension.

$$0011 \xrightarrow[\text{unsigned}]{\text{8-bits}} 0000\ 0011$$

2. Sign extension

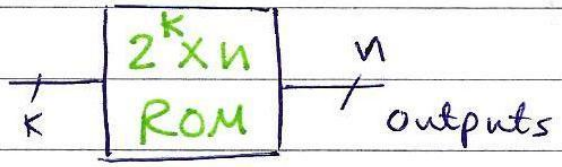
$$(1100) \xrightarrow[\text{signed number}]{\text{8-bits}} 1111\ 1100$$
$$\equiv -4$$

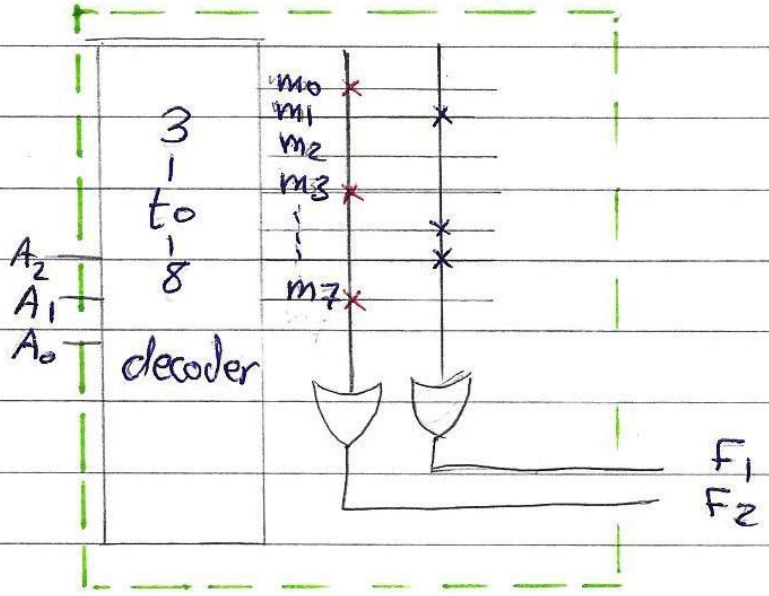
$$(0100) \xrightarrow[\text{signed}]{\text{8-bits}} 0000\ 0100$$

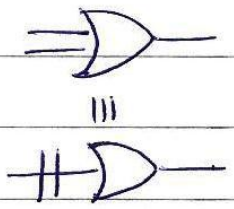
Chapter #6: Section 6-8: Programmable logic devices (PLD)s

① ROM:

it has a fixed array of AND gates & programmable array of OR gates.







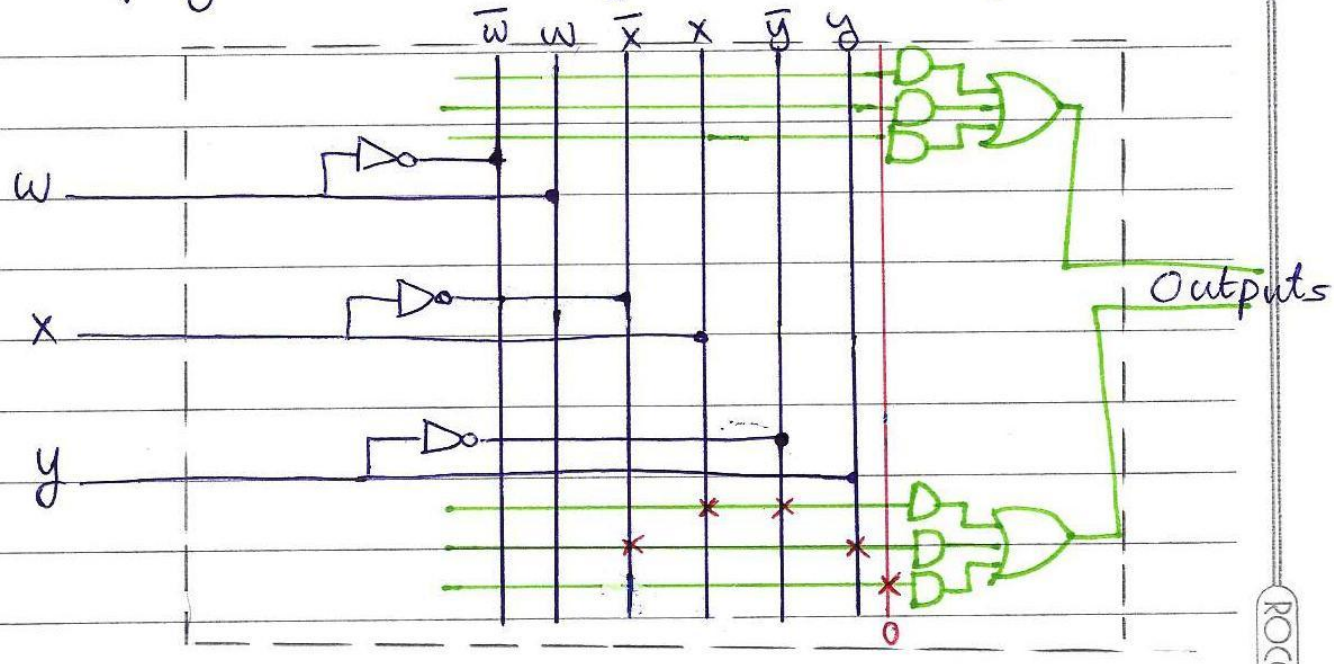
$$F_2(w, y, z) = \sum_m (0, 3, 7)$$

$$F_1(w, y, z) = \sum_m (1, 4, 5)$$

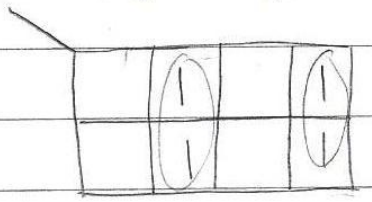
*In ROM, the function needs not to be simplified.

② Programmable Array logic (PAL)

* it has a fixed array of OR gates & programmable array of AND gates.



$F(w, x, y) = \sum_m (1, 2, 5, 6)$: (SOM)

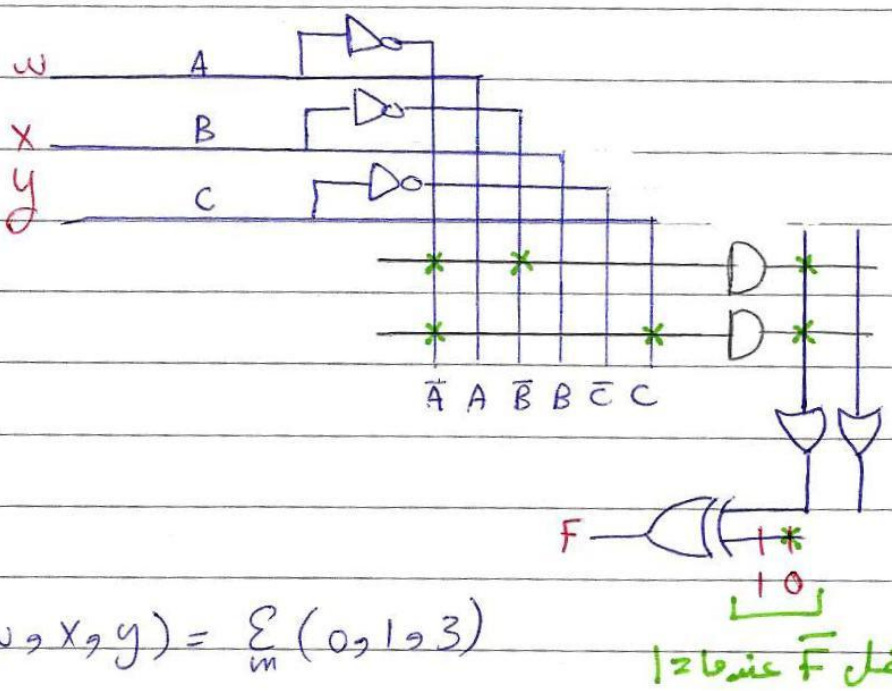


$\rightarrow F = x\bar{y} + \bar{x}y$

* In PAL, we might need to express f as sop if number of AND gates <= number of minterms.

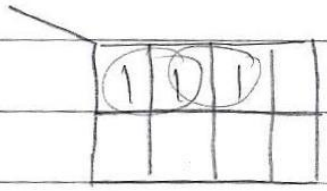
③ Programmable logic Array (PLA).

* it's the most flexible because it has programmable OR & AND



لعمل $\bar{F}$ عندما = 1

$F(w,x,y) = \sum_{im}(0,1,3)$

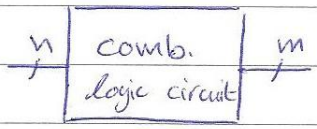


$\bar{A}\bar{B} + \bar{A}C$

Chapter #5: Sequential logic circuits (finite state machine).

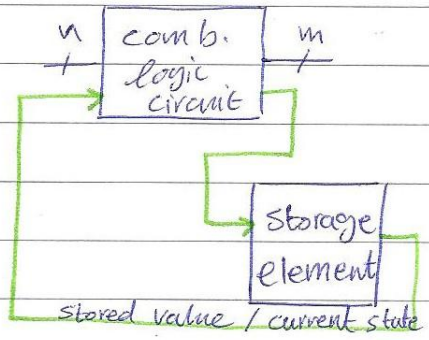
* Logic Circuits

↳ Combinational logic circuits:



** the outputs depend on the current input only.

↳ Sequential logic circuits:



Sequential logic circuit

** the output is dependent on the current input & the stored value (state).

* there are 2 kinds of sequential machines:-

1. Synchronous:-
   - the state is allowed to change at specific instants of time.
   - there's a clock input

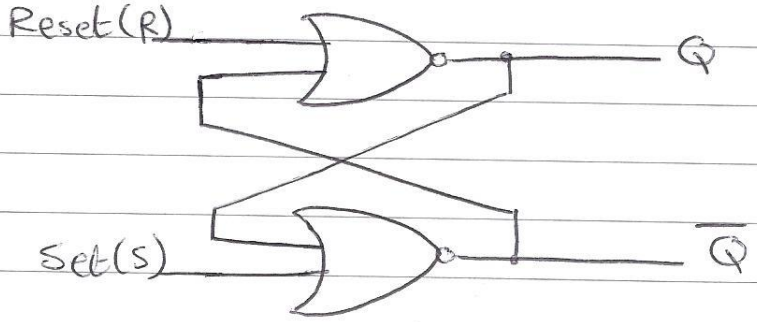
   - Simpler to design.
   - Slower.
2. Asynchronous:-
   - the stored value is allowed to change at any instant of time.
   - there is no clock.
   - complex to design.
   - faster.

**Storage elements:

1) latches:

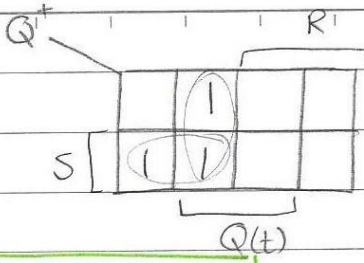
* A latch is a circuit that can store a binary state indefinitly & as long as power is On.
* it has inputs that are used to operate on the stored value.
* we have 4 basic opperations:-
  1. Set (1)
  2. Reset (0)
  3. No change.
  4. Complement (not common).

A. Cross-coupled-NOR gates latch (SR latch)



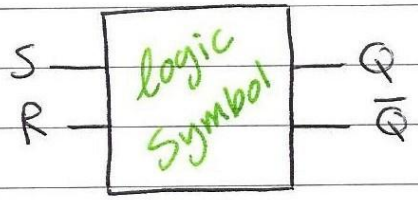
time	S	R	Q	$\overline{Q}$	
t_0	X	X	X	X	
t_1	1	0	1	0	=> Set.
t_2	0	0	1	0	=> No change.
t_3	0	1	0	1	=> Reset.
t_4	0	0	0	1	=> No change.
t_5	1	1	Invalid (this state isn't allowed) because $Q \neq \overline{Q}$		
			(0 0)		

S	R	Q(t)	$Q^+ \equiv Q(t+1)$	
0	0	0	0	} No change
0	0	1	1	
0	1	0	0	} Reset
0	1	1	0	
1	0	0	1	} set
1	0	1	1	
1	1	0	X	} invalid
1	1	1		

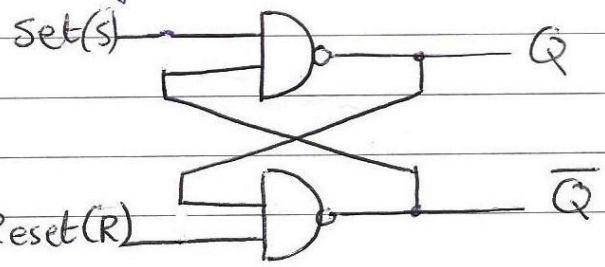


$Q^+ = S\bar{R} + \bar{R}Q(t)$

↳ charachteristic equation...

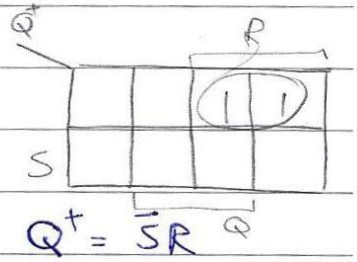


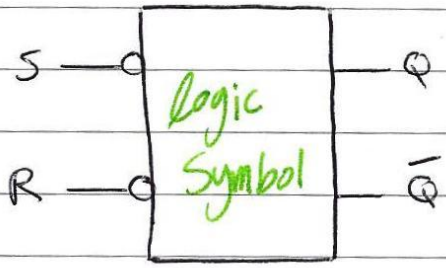
B. Cross-Coupled - NAND gates latch ($\bar{S}\bar{R}$ latch).



ex	time	S	R	Q	$\bar{Q}$	
	t_0	X	X	X	X	
	t_1	0	1	1	0	⇒ set.
	t_2	1	1	1	0	⇒ No change.
	t_3	1	0	0	1	⇒ Reset
	t_4	1	1	0	1	⇒ No change.
	t_5	0	0	Invalid		

$\bar{S}$	$\bar{R}$	$Q(t)$	Q^+	
0	0	0	X	Invalid (undefined)
0	0	1	X	
0	1	0	1	Set
0	1	1	1	
1	0	0	0	Reset
1	0	1	0	
1	1	0	0	No change
1	1	1	1	

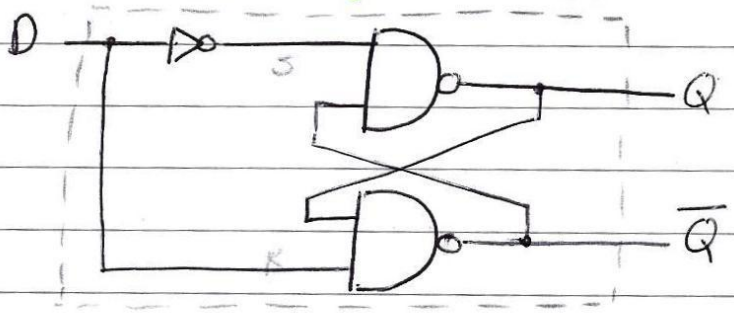




⇒ active-low input

C. Data-latch (D-latch).

(to solve the (undifined state) problem).



D	Q^+
0	0
1	1

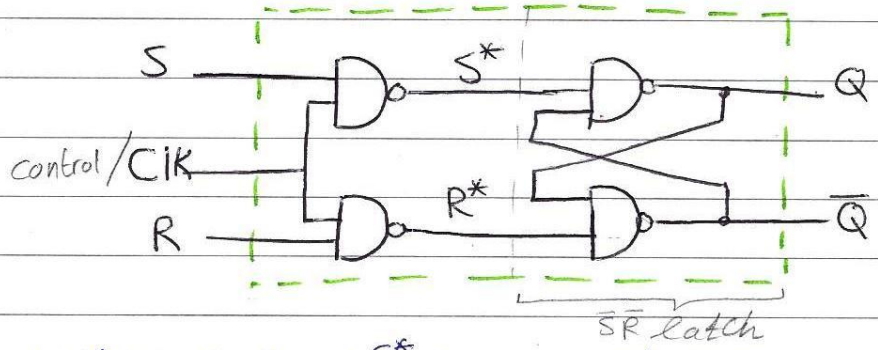
* when D = 0 → S = 1, R = 0 ⇒ $Q^+ = 0$
* when D = 1 → S = 0, R = 1 ⇒ $Q^+ = 1$

14/7/2012 السبت

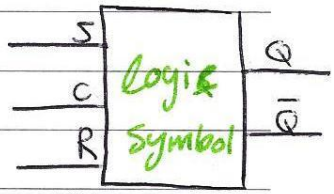
2) Clocked latches:

* All latches discussed perviously are asynchronous
* The state of the storage element may change as soon as the input changes.
* However, sometimes we need to force the state to change (during/at) certain time.

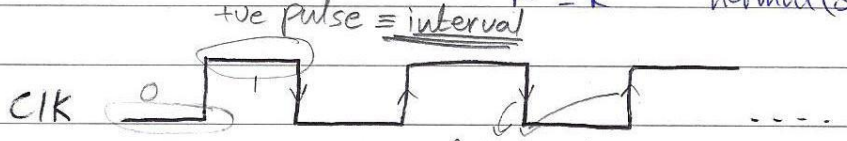
A. Clocked SR latch.



* when C=0 → $S^*=1$ → no change
  $R^*=1$ $Q^+ = Q(t)$
* when C=1 → $S^*=\bar{S}$ → works as a
  $R^*=\bar{R}$ normal ($\bar{S}\bar{R}$ latch).

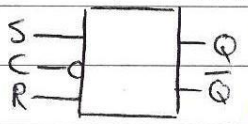


+ve pulse triggered SR latch



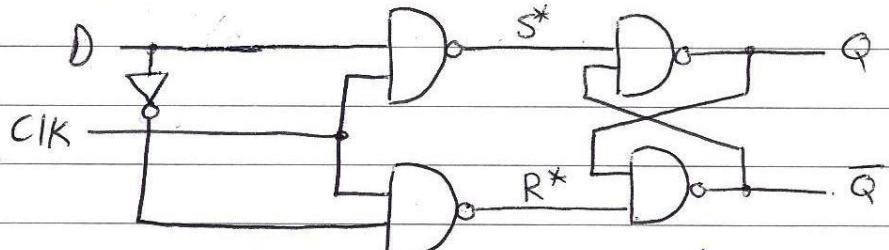
logic low
-ve pulse

falling edge = instant
& rising "



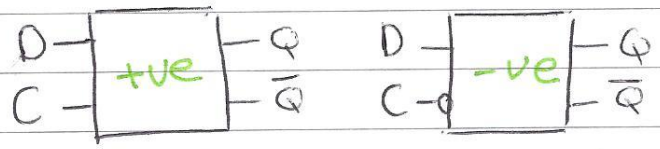
-ve pulse triggered SR latch.

B. Clocked D-latch.



* when C=0 → $S^*=1$ → No change
  $R^*=1$ $Q^+ = Q(t)$
* when C=1 → $S^*=S\equiv\bar{D}$ → work as a normal $\bar{S}\bar{R}$
  $R^*=R\equiv D$ D-latch.

cont.



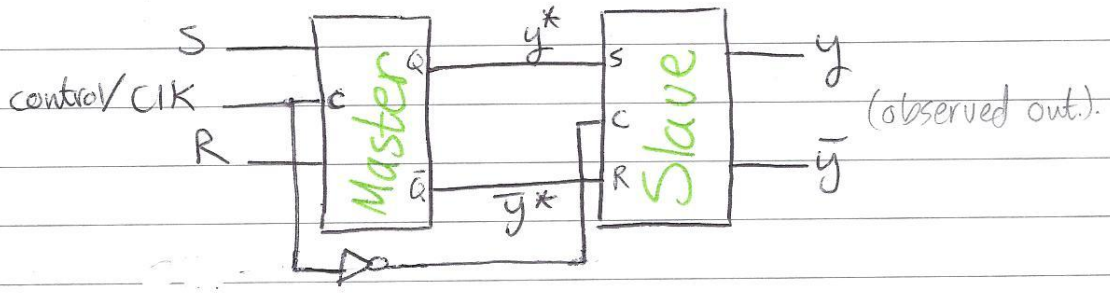
+ve pulse triggered D-latch

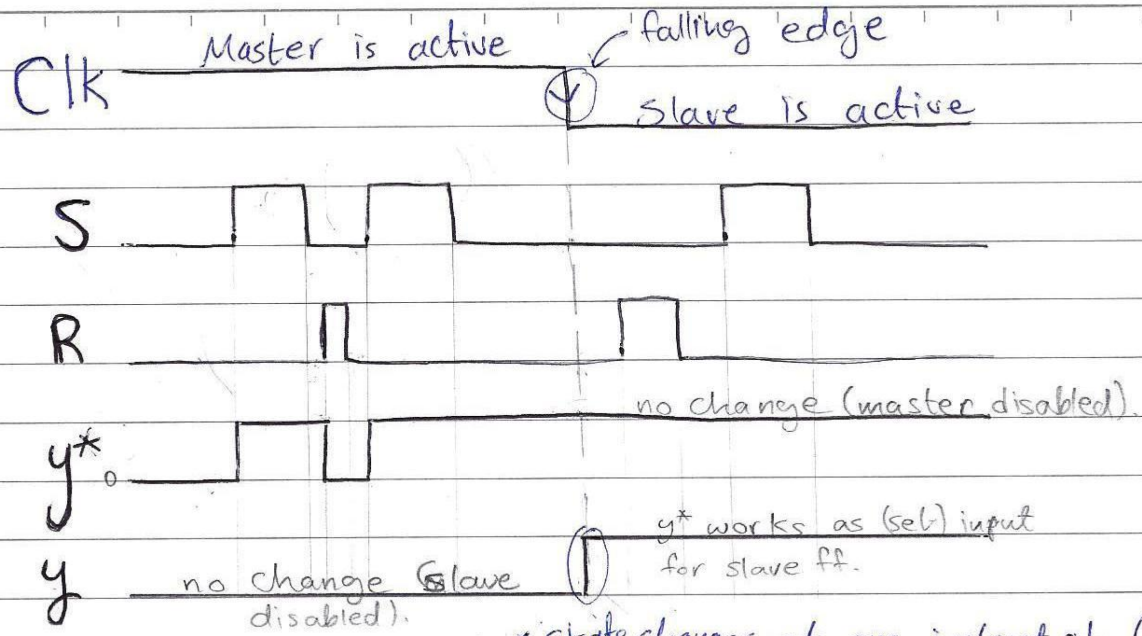
-ve pulse triggered D-latch.

3) Flip-Flops

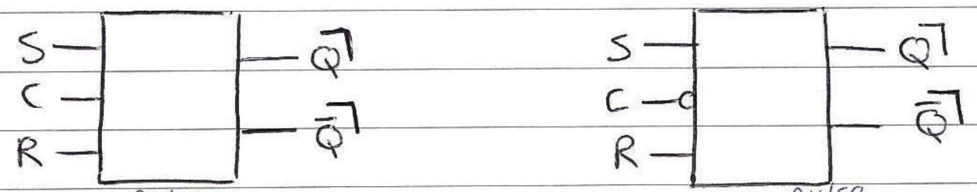
* the state of the clocked latches is allowed to change during specific time (±ve pulse).
* the state in the clocked latches may change as soon as the input changes & the clock is active.
* However, we want a storage element that allows the state to change at specific time if the input changes!
* we have 2 types of Flip-Flops:-
  - Master-slave flip-flop.
  - Edge triggered flip-flop.

1. Master-slave ff:





* state changes at one instant only (falling edge). based on the last observed value of y* (Master)!

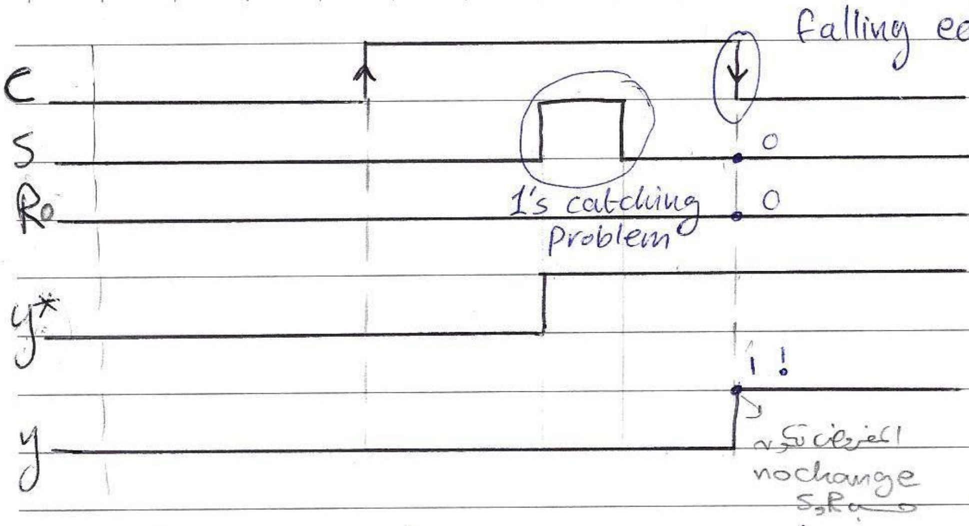


* +ve pulse (edge) triggered FF (صح التخطيط!)

-ve pulse (edge) triggered ff.

* we can call it ±ve edge triggered because state changes only at the (falling/rising) edge instant. while, in the previous SR ff's it was called ±ve pulse triggered because state changes during the whole ±ve pulse duration.

falling edge ≡ state changes



⇒ master-slave FF.

* المفروض انه تكون y = 0 *

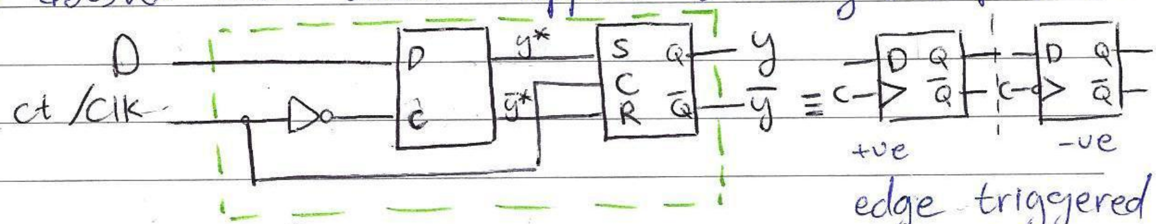
عند ال falling edge ⇐ no change

* y gives you the last observed value of y*.

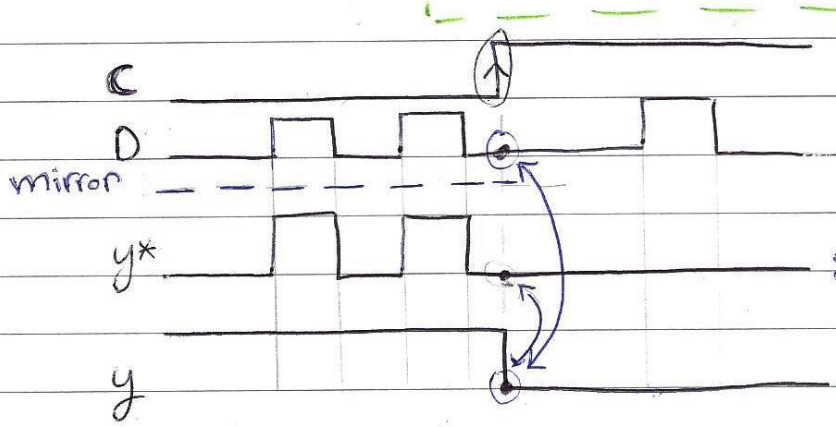
* **problem:** stored value is observed during the pulse not <u>at</u> the falling edge time.

2. edge triggered ff:

* output changes based on the last observed inputs.
* doesn't care what happens during the pulse.



edge triggered D-ff.



y* works as mirror of D before the rising edge.

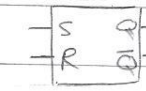
* Summary :

storage elements

↳ 1. Latches

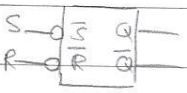
Asynchronous

→ SR



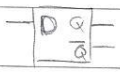
S R
0 0 → No change
0 1 → Reset
1 0 → set
1 1 → x invalid

→ $\bar{S}\bar{R}$



S R
0 0 x invalid
0 1 set
1 0 reset
1 1 No change

→ D-latch

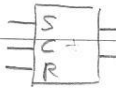


D	Q
0	0
1	1

↳ changes value whenever input changes.

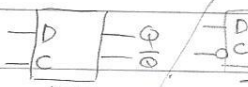
↳ 2. Clocked latches : state is allowed to change during specific time (±ve pulse)

→ clocked SR latch



$\bar{C}$ ∓ve pulse triggered SR latch.

→ clocked D latch



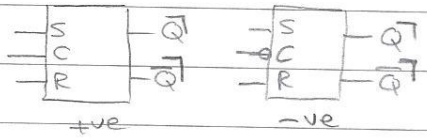
∓ve pulse triggered d-latch.

↳ 3. Flip-Flops

→ master-slave ff

→ state is allowed to change at specific time.

→ Problem: 1's & 0's catching.

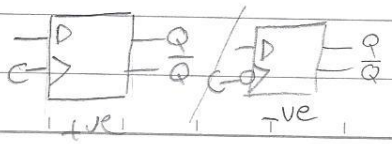


∓ve (edge) triggered ff (لا يوجد).

→ Edge triggered ff

→ outputs are determined based on the inputs at the (falling/rising) edge instant.

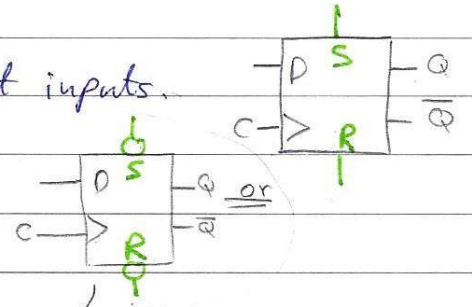
∓ve edge triggered d-ff

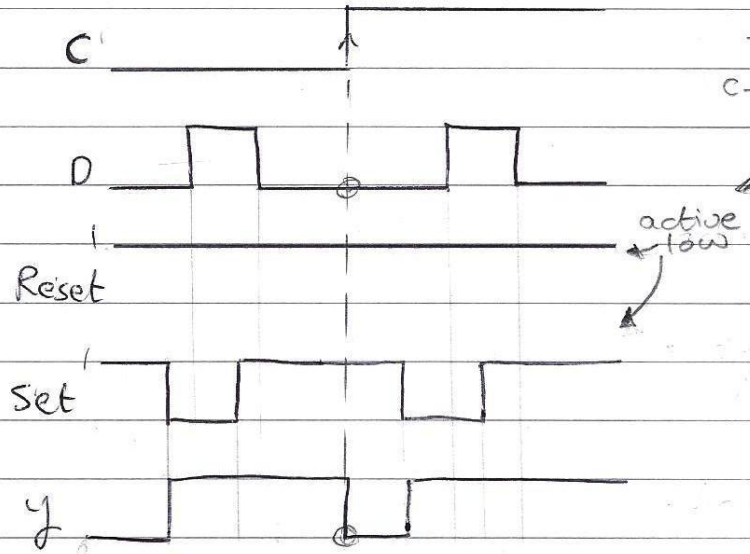


** Direct inputs:

- Sometimes, it's required to change the stored value/state independently from the clock (Asynchronously).
- We can achieve this using the direct inputs (set & Reset)
- they are Asynchronous inputs.

ex:- edge triggred ff with direct inputs.



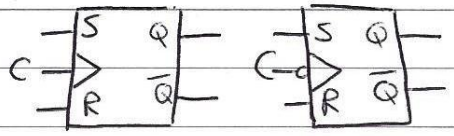


* using Direct input is an optional choice to be able to change state Asynchronously (without waiting for next falling/rising edge).

* it's commonly used for: • initialization • dealing with unused states.

* More On FFs:

1. Edge triggered SR ff.



* logic Symbol

* Charachteristic table

S	R'	Q^+
0	0	Q(t)
0	1	0
1	0	1
1	1	?? invalid

* Characteristic equ.

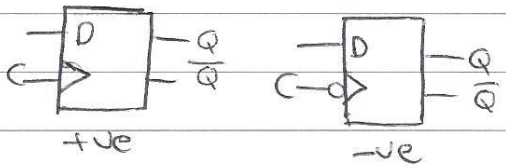
$$Q(t+1) = S\bar{R} + \bar{R}Q(t)$$

* Excitation table

Q(t)	Q^+	S R
0	0	0 x
0	1	1 0
1	0	0 1
1	1	x 0

2. Edge triggered Dff:

*



* Characteristic table

D	Q^+
0	0
1	1

* Characteristic equ.

$Q^+ = D$

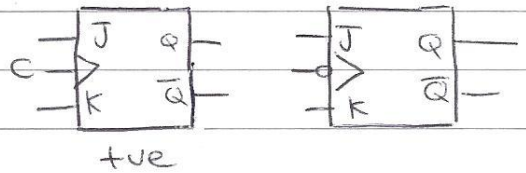
transfering ≡ (No change).

* Excitation table

Q	Q^+	D
0	0	0
0	1	1
1	0	0
1	1	1

3. Edge triggered JK ff.

*



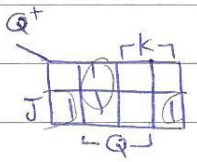
* char. table

J	K	Q^+
0	0	Q
0	1	0
1	0	1
1	1	$\bar{Q}$

toggle: to complement the state.

* char. equ.

J	k	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



$$Q^+ = J\bar{Q} + \bar{k}Q$$

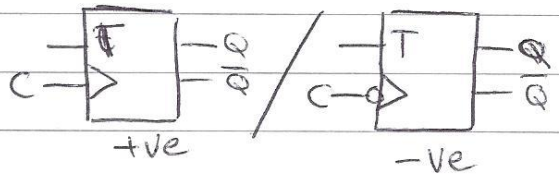
* excitation table

Q	Q^+	J K
0	0	0 x
0	1	1 x
1	0	x 1
1	1	x 0

Toggle = complement

4. Edge triggred T-ff

*

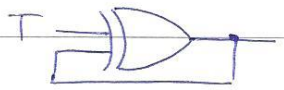


* char. table

T	Q^+
0	Q
1	$\bar{Q}$

* char equ.

$$Q^+ = T \oplus Q$$



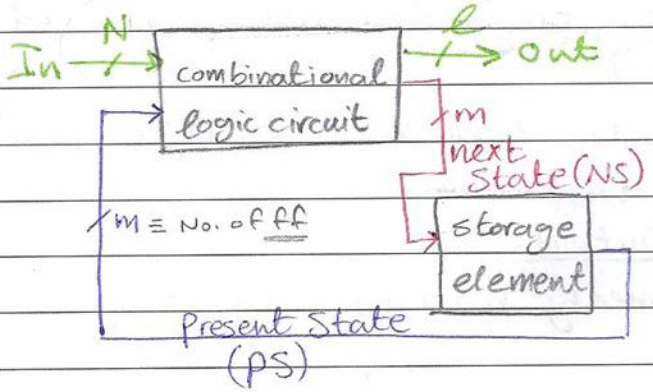
* excitation table

Q	Q^+	T
0	0	0
0	1	1
1	0	1
1	1	0

15/7 الأحد

*Sequential circuit analysis:

- we want to study the behavior of circuit over time.



- Analysis involves obtaining suitable description of inputs, outputs & state:

⇒Steps:

1. determine the ff input equations & output equations
   *these can be read directly from the logic diagram.
2. Write the State table ; it's similar to the truth table.
   - the input section on this table includes:
     - inputs (N) - Present states (m).
   - the output section includes:
     - Outputs (l) - Next States (m).

** for a sequential machine circuit with N inputs, m K-inputs, l outputs, the size of the state table is :-

$$\text{Size} = \underbrace{2^{(N+m)}}_{\text{raws}} \times \underbrace{(2m + N + l + K*M)}_{\text{Columns}}$$

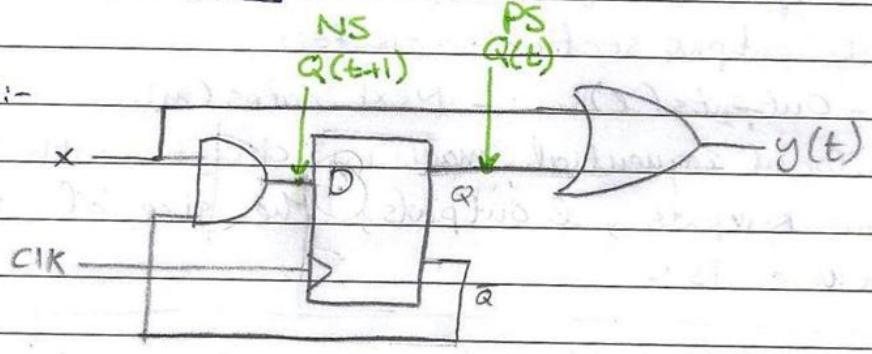
3. Optionally; we can draw the state diagram. It's a graphical representation of the state table, where:-

- each state is represented by a circle.
- each state transition is represented by an arc.
- each arc is labled with input value that causes the transition.
- for the Outputs:-
  - * If a mealy machine: output values are placed on the arc → (input/output) ≡ arc label
  - * If a moore machine: outputs are placed inside the circle → (state/out) because since out depends on Ps only, so it's fixed for the same state (circle).
  - * mealy machine depends on input & PS, while moore machine depends on PS only.

Note: for a sequential machine/circuit with (M) FF & (N) inputs:-

- * Number of states (≡ circles) is $\boxed{2^M}$
- * Number of arcs going out of any state is $\boxed{2^N}$

ex:-



1. N = 1

M = 1, K = 1

ℓ = 1

⇒ Size = $2^2 \times 2 * 1 + 1 + 1 + 1 = 4 \times 5$

2. Input & output equ.

In → D = X $\bar{Q}(t)$

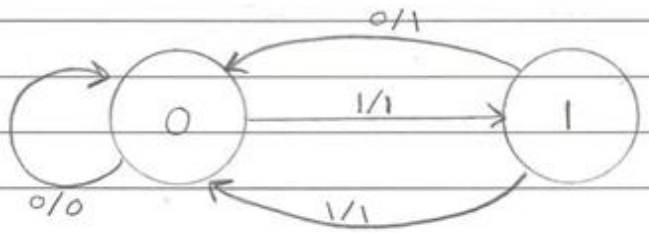
$Q^+ = D = X \cdot \bar{Q}(t)$

Out → $y(t) = X + Q(t)$ ... mealy machine.

3 State table

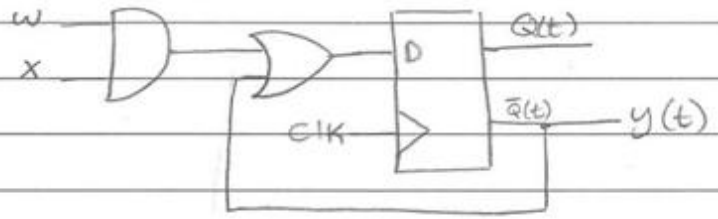
PS	In		NS	out	this table tells
Q(t)	X		Q^+	y(t)	you what NS
state #1 0	0		0	0	& output would
state #1 0	1		1	1	by when clock=1 (falling/rising edge)
state #2 1	0		0	1	& the state = Q(t)
state #2 1	1		0	1	& Input is X

4. State diagram



* tables:
I/O
↓ ↘
input output

ex 2:-



1. N=2
M=1, K=1 } size = 8×6
L=1

2. input & output eqn.

$Q^+ = D = wx + \bar{Q}(t)$

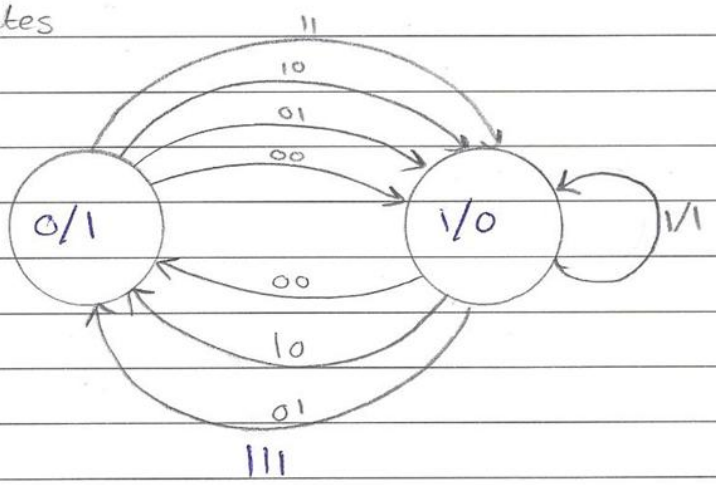
$y(t) = \bar{Q}(t)$

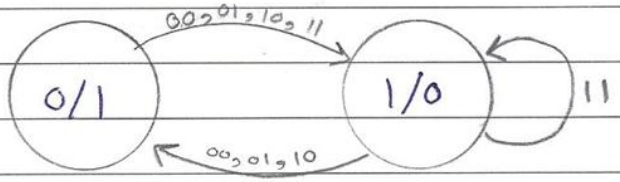
3. State table

	Ps	In	NS	out
	Q(t)	w x	D≡Q⁺	y(t)
state #1	0	0 0	1	1
	0	0 1	1	1
	0	1 0	1	1
	0	1 1	1	1
state #2	1	0 0	0	0
	1	0 1	0	0
	1	1 0	0	0
III	1	1 1	1	0

2 states

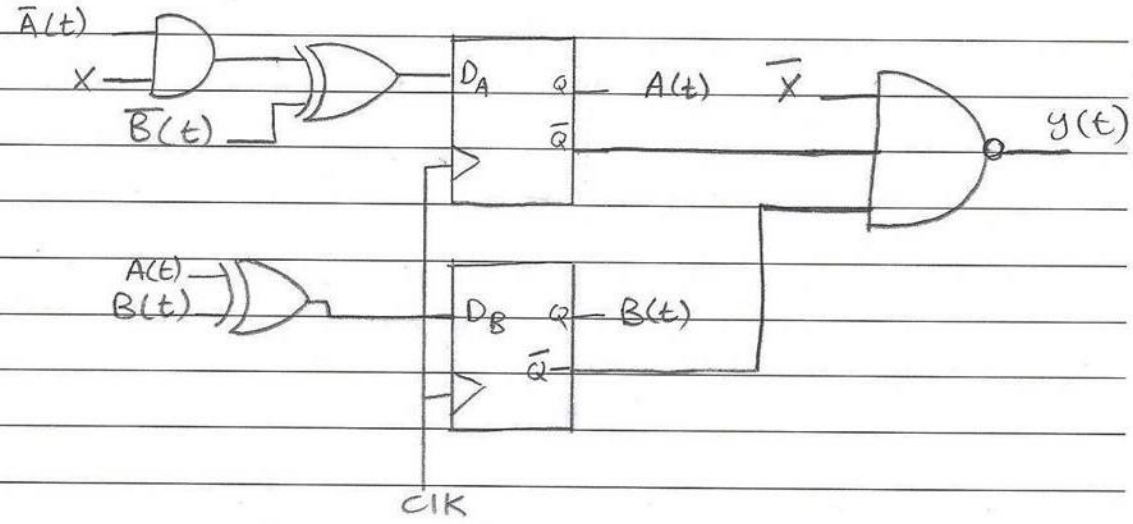
4. State diagram





16/7 الاثنين

H.W ex: derive the state table & diagram for this:-



① $N=1$
$M=2, k=1$
$\ell=1$
$\Rightarrow$ size $= 2^3 \times (4+1+1+2) = 8 \times 8$

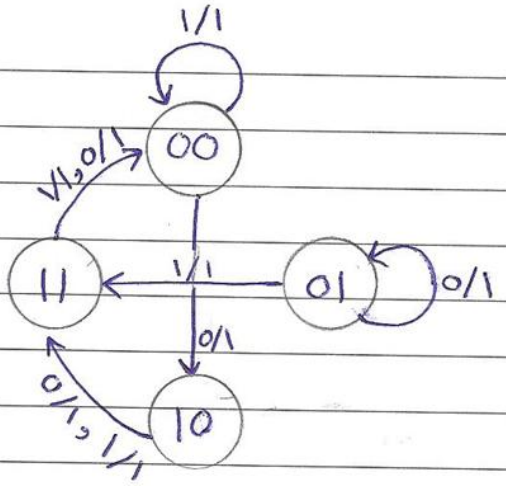
② $A^+ = D_A = X.\bar{A}(t) \oplus \bar{B}(t)$
$B^+ = D_B = A(t) \oplus B(t)$
$y(t) = \overline{\bar{x}.\bar{A}(t).\bar{B}(t)} = x + A(t) + B(t)$

③ State table

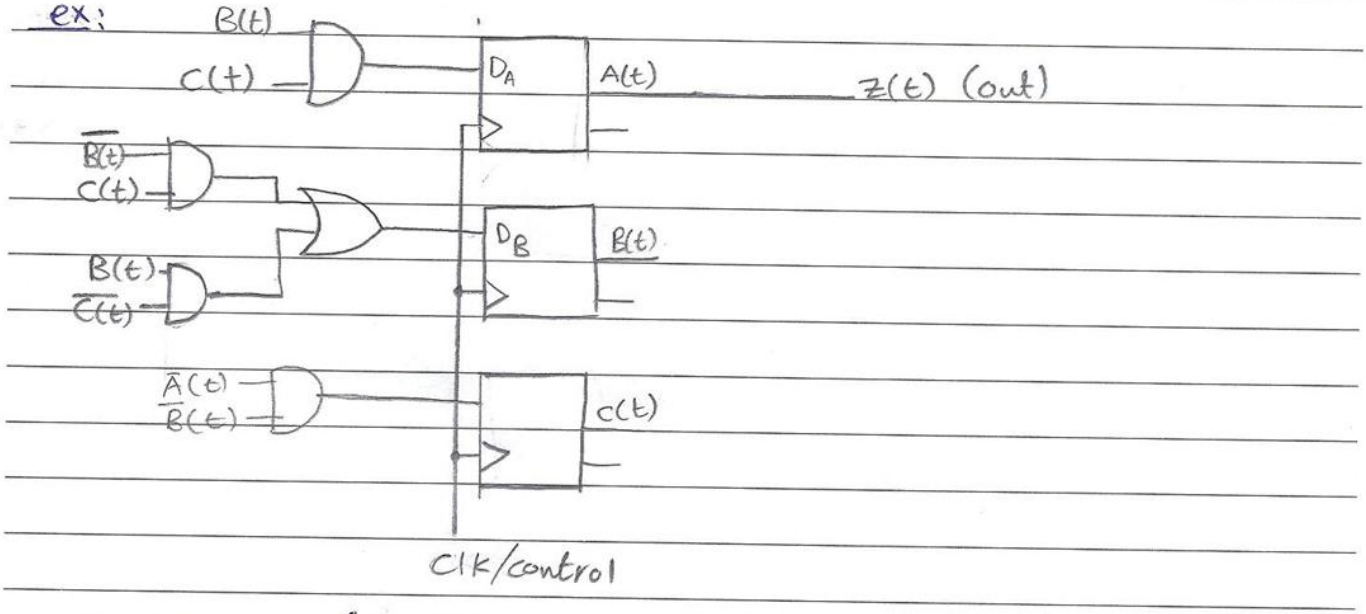
PS A(t)	PS B(t)	In X	D_A	D_B	NS A^+	NS B^+	Out y(t)
0	0	0			1	0	0
0	0	1			0	0	1
0	1	0			0	1	1
0	1	1			1	1	1
1	0	0			1	1	1
1	0	1			1	1	1
1	1	0			0	0	1
1	1	1			0	0	1

4 states

④ State diagram



ex:



1. $N=0$ (No external inputs)

$M=3$, $K=1$

$\ell=1$

$\Rightarrow$ Size $= 2^{3+0} \times (0+3+1+6) = 8 \times 10$

2. $A^+ = D_A = C(t) \cdot B(t)$

$B^+ = D_B = B(t) \oplus C(t)$

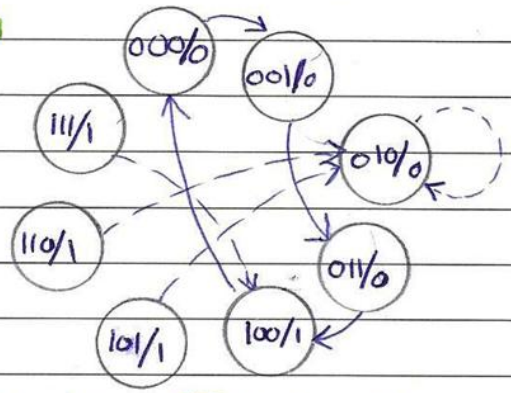
$C^+ = D_C = \bar{A}(t)\bar{B}(t) = \overline{A(t)+B(t)}$ (NOR)

$z(t) = A(t)$ ... Moore machine

3.

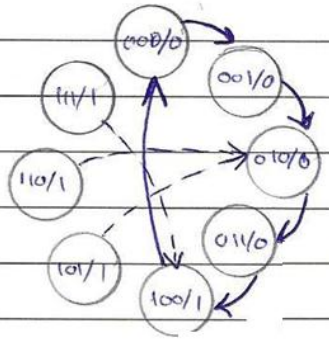
	PS			NS			Out
	A(t)	B(t)	C(t)	A^+	B^+	C^+	Z(t)
states	0	0	0	0	0	1	0
	0	0	1	0	1	1	0
	0	1	0	0	1	0	0
	0	1	1	1	0	0	0
	1	0	0	0	0	0	1
&	1	0	1	0	1	0	1
*	1	1	0	0	1	0	1
unused states ←	1	1	1	1	0	0	1

4. State diagram

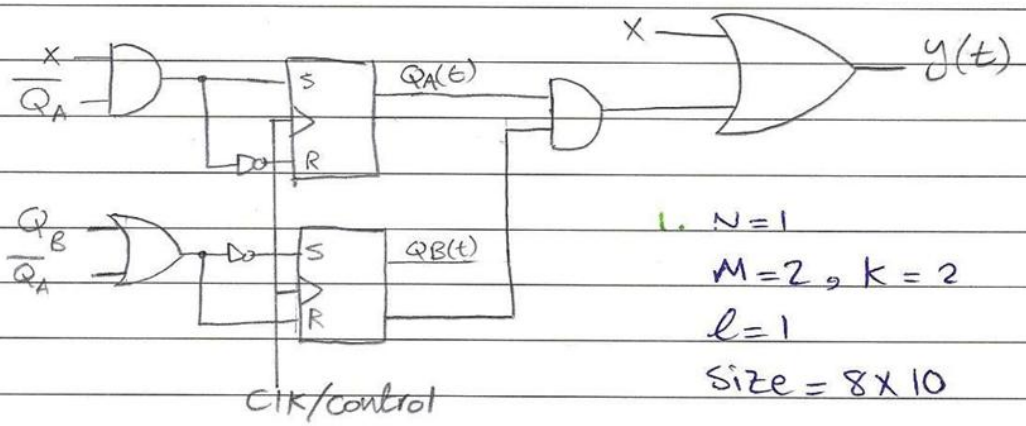


* when replacing $\overline{B(t)}$ in the question with $\overline{C(t)}$

PS			NS			Out
A	B	C	A^+	B^+	C^+	Z(t)
0	0	0	0	0	1	0
0	0	1	0	1	0	0
0	1	0	0	1	1	0
0	1	1	1	0	0	0
1	0	0	0	0	0	1
1	0	1	0	1	0	1
1	1	0	0	1	0	1
1	1	1	1	0	0	1



ex:-



1. N = 1
   M = 2 , K = 2
   $\ell = 1$
   Size = 8 X 10

2. input equations:

$S_A = X\overline{Q}_A$

$R_A = \overline{X} + Q_A \equiv \overline{S}_A$

$S_B = \overline{Q}_B Q_A$

$R_B = \overline{S}_B$

(out) → $y(t) = Q_A \cdot \overline{Q}_B + X$

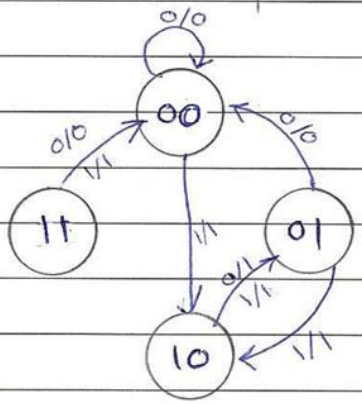
3. State table

PS		In					NS		Out
Q_A	Q_B	X	S_A	R_A	S_B	R_B	Q_A^+	Q_B^+	y(t)
0	0	0	0	1	0	1	0	0	0
0	0	1	1	0	0	1	1	0	1
0	1	0	0	1	0	1	0	0	0
0	1	1	1	0	0	1	1	0	1
1	0	0	0	1	1	0	0	1	1
1	0	1	0	1	1	0	0	1	1
1	1	0	0	1	0	1	0	0	0
1	1	1	0	1	0	1	0	0	1

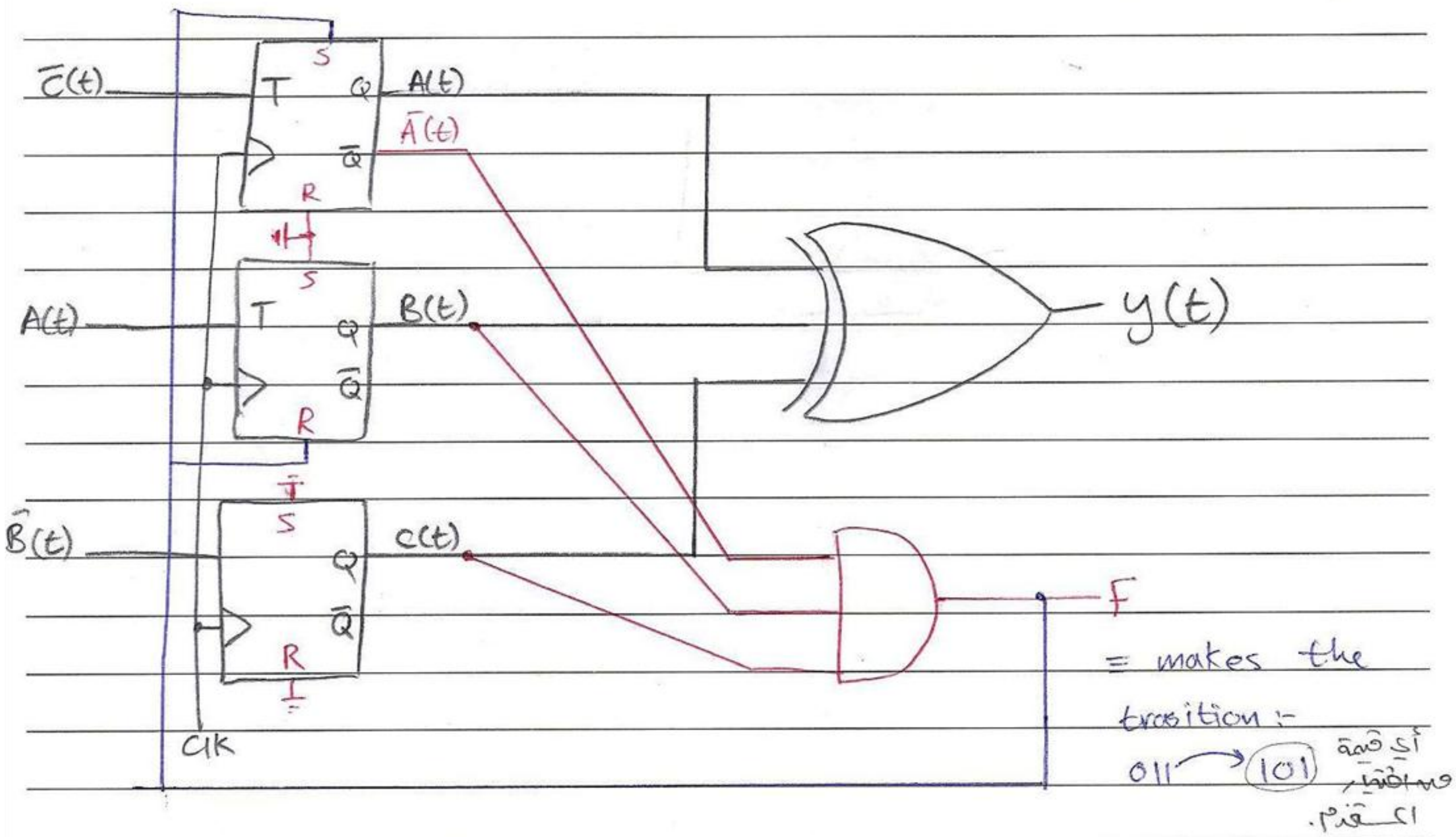
4 states

4. State Diagram

Mealy machine



الثلاثاء ١٧/٧

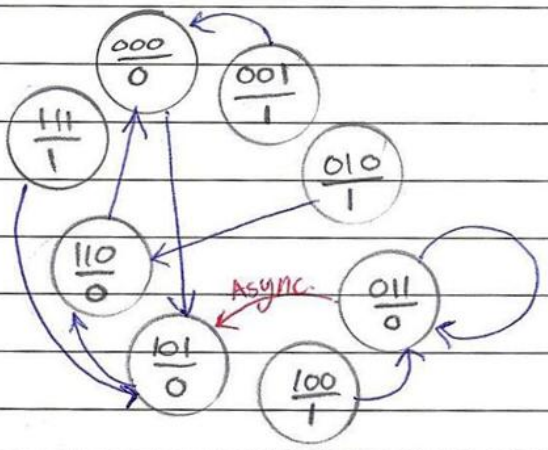


1. N=0
M=3, K=1
l=1

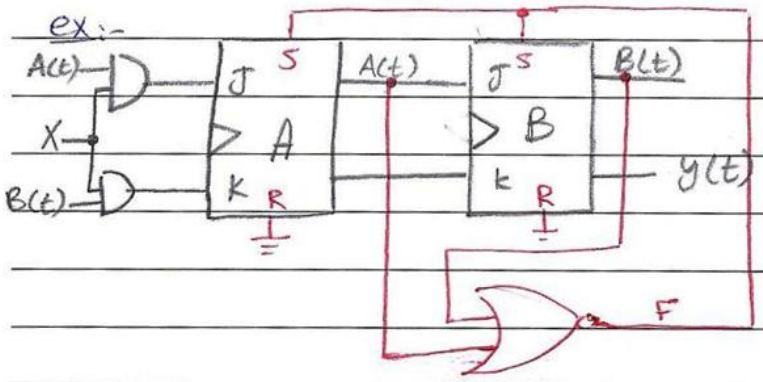
2. $T_A = \bar{C}(t)$
$T_B = A(t)$
$T_C = \bar{B}(t)$
$y = A(t) \oplus B(t) \oplus C(t)$ (odd function).
(Moore machine)

3.

PS						NS			Out
A	B	C	T_A	T_B	T_C	A^+	B^+	C^+	y(t)
0	0	0	1	0	1	1	0	1	0
0	0	1	0	0	1	0	0	0	1
0	1	0	1	0	0	1	1	0	1
0	1	1	0	0	0	0	1	1	0
1	0	0	1	1	1	0	1	1	1
1	0	1	0	1	1	1	1	0	0
1	1	0	1	1	0	0	0	0	0
1	1	1	0	1	0	1	0	1	1



ex:-



1. $J_A = X.A(t)$

$K_A = X.B(t)$

$J_B = A(t)$

$K_B = \overline{A(t)}$

(out) $\rightarrow y(t) = \overline{B(t)}$ (Moore machine)

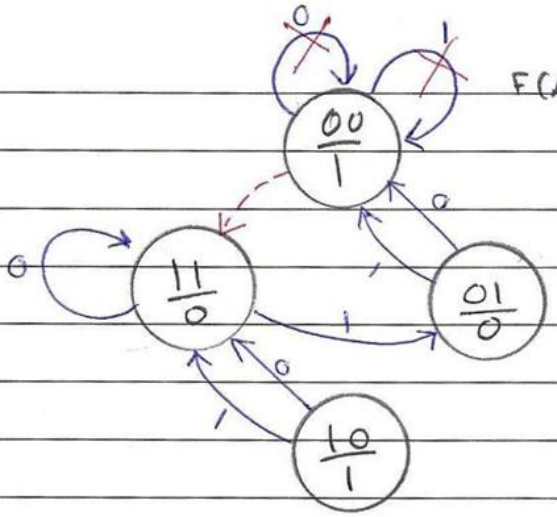
JK flip-flop charachteristic table:-

J	K	Q^+
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

2.

PS		In	FF inputs				NS		Out
A	B	X	J_A	K_A	J_B	K_B	A^+	B^+	y(t)
0	0	0	0	0	0	1	0	0	1
0	0	1	0	0	0	1	0	0	1
0	1	0	0	0	0	1	0	0	0
0	1	1	0	1	0	1	0	0	0
1	0	0	0	0	1	0	1	1	1
1	0	1	1	0	1	0	1	1	1
1	1	0	0	0	1	0	1	1	0
1	1	1	1	1	1	0	0	1	0

3.



$F(A,B) = m_0 = \bar{A}\bar{B} = \overline{A+B}$ (NOR)

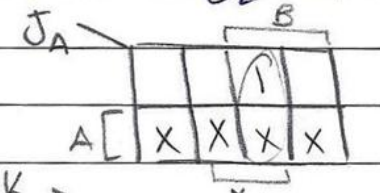
Design a sequential circuit that implements the following state table, use JK ff. *Assume mealy machine.

PS	in	NS	Out	$J_A K_A$	$J_B K_B$
A B	x	A^+ B^+	y(t)		
0 0	0	0 0	0	0 X	0 X
0 0	1	0 1	1	0 X	1 X
0 1	0	0 1	0	0 X	X 0
0 1	1	1 0	1	1 X	X 1
1 0	0	1 0	1	X 0	0 X
1 0	1	1 1	1	X 0	1 X
1 1	0	1 1	1	X 0	X 0
1 1	1	0 0	1	X 1	X 1

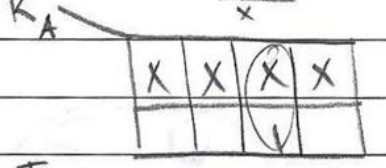
JK ff excitation table:

Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

* $m = \lceil \log_2 \#\text{ of states} \rceil = 2$



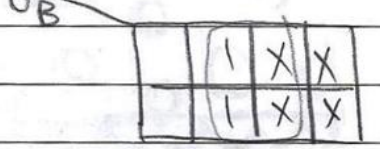
$J_A = XB$

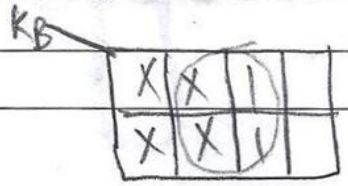


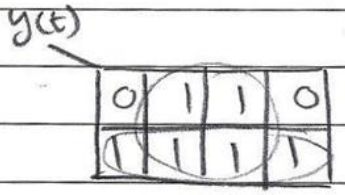
$K_A = XB$

$J_B = X$

$K_B = X$



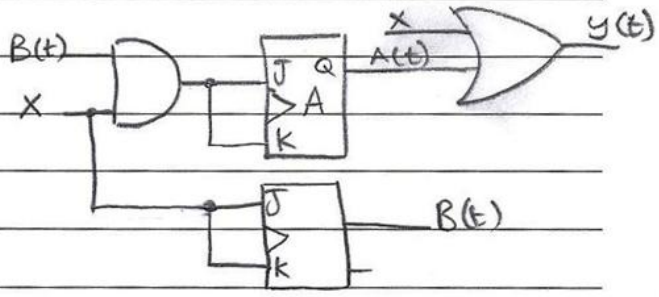




$y = A + X$

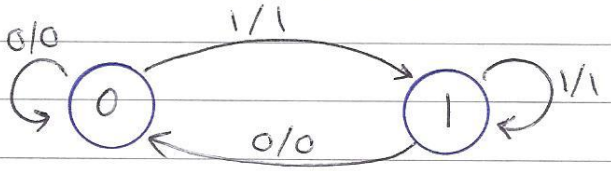
= mealy machine

3-var. (PS & In)



18/7 الاربعاء

ex:- Given the following state diagram, design the sequential circuit that implements it. compare the design when Tff & Dff are used.



PS	In	NS	out		
A(t)	X	A(t+1)	y(t)	T	D ≡ A+
0	0	0	0	0	0
0	1	1	0	1	1
1	0	0	0	1	0
1	1	1	1	0	1

T flip-flop excitatio table

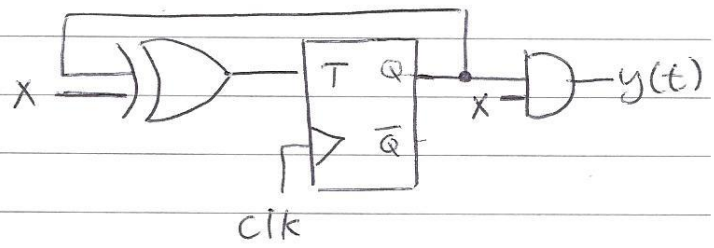
Q	Q+	T
0	0	0
0	1	1
1	0	1
1	1	0

# of ffs = $m = \lceil \log_2 2 \rceil = 1$ ← # of states.

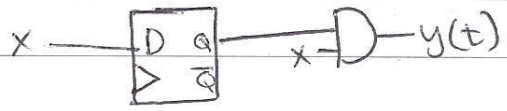
* T-ff

T	X=0	X=1
A=0	0	1
A=1	1	0

$T = A(t) \oplus X$



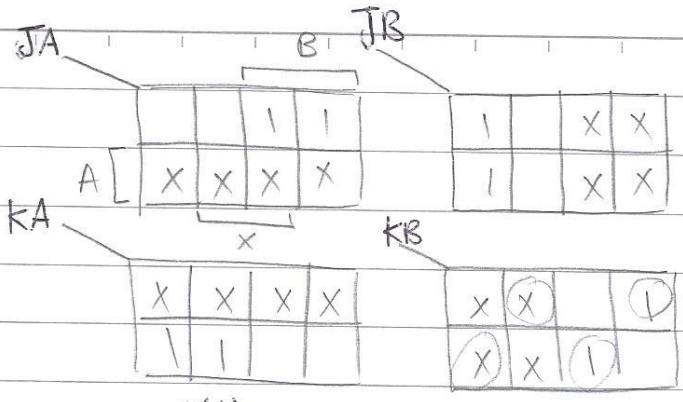
* D-ff



*ex: use JK ff to design the sequential circuit that implements the following state table.

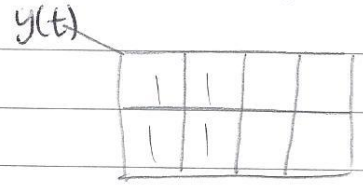
PS		In	NS		out				
A	B	X	A+	B+	y(t)	J_A	K_A	J_B	K_B
0	0	0	0	1	1	0	X	1	X
0	0	1	0	0	1	0	X	0	X
0	1	0	1	0	0	1	X	X	1
0	1	1	1	1	0	1	X	X	0
1	0	0	0	1	1	X	1	1	X
1	0	1	0	0	1	X	1	0	X
1	1	0	1	1	0	X	0	X	0
1	1	1	1	0	0	X	0	X	1

① # of ffs = $m = \lceil \log_2 4 \rceil = 2$

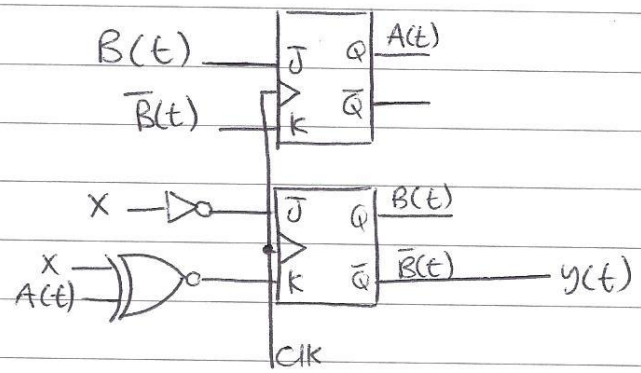


$J_A = B(t)$
$k_A = \overline{B}(t)$

$J_B = \overline{x}$
$k_B = A(t) \odot x$



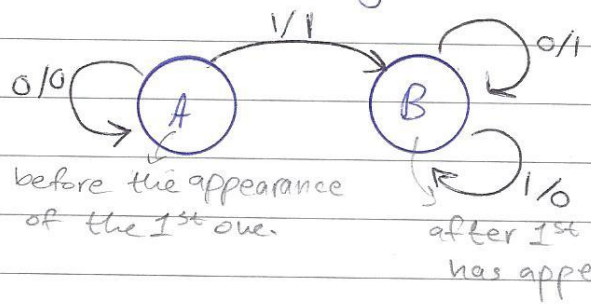
$y(t) = \overline{B}(t)$ ... Moore machine



ex:- Design a sequential machine that implements a serial (1-bit at a time) 2's complement

* in order to calculate 2's complement for any binary number, we need to keep zeros up to first 1 (machine needs to remember the appearance of first 1) then complement the remaining bits.

Step#1: State Diagram



- we have to start with at least 1 state.
- we add new state to store new states the machine had to remember.

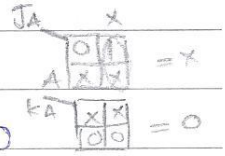
step#2: state assignment (Renaming states).

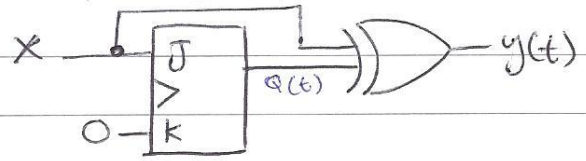
let A = 0

B = 1

step#3: State table:

PS	In	PS	out		
A	X	A^+	y(t)	J_A K_A	
0	0	0	0	0 x	$J_A = X$
0	1	1	1	1 x	$K_A = 0$
1	0	1	1	x 0	
1	1	1	0	x 0	





19/7 الخميس

ex:- Design a counter that counts 00, 01, 10, 11, 00, 01, ---- and it has 1 input (x); such that: (use Dff)

* if $x = 0$, counter halts (pauses)
* if $x = 1$, counter proceeds to next count.

* counter goes through pre-defined sequence of states.
* usually it's a moore machine (output = PS).

⇒ * # of states = 4
* # of ffs = 2

PS $Q_1\ Q_0$	In x	NS $Q_1^+ \equiv D_1,\ Q_0^+ \equiv D_0$
0 0	0	0 0
	1	0 1
0 1	0	0 1
	1	1 0
1 0	0	1 0
	1	1 1
1 1	0	1 1
	1	0 0

4 possible states.

$D_1 \equiv Q_1^+$

Q_1 \ Q_0	0	1
0	0	0
1	1	0

(x)

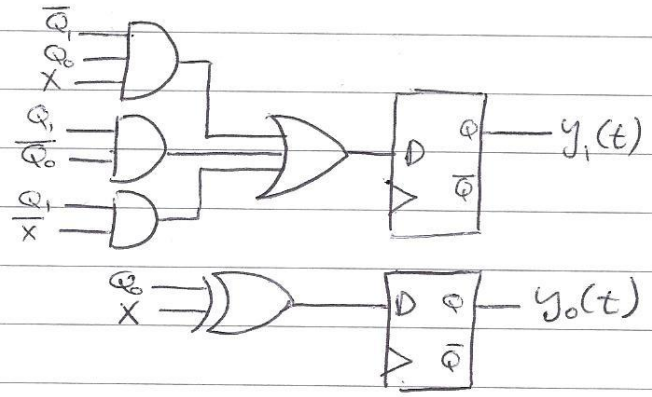
$$D_1 \equiv Q_1^+ = \overline{Q_1} Q_0 x + Q_1 \overline{Q_0} + Q_1 \overline{x}$$

$D_0 \equiv Q_0^+$

Q_1 \ Q_0	0	1
0	0	1
1	0	1

(x)

$$D_0 \equiv Q_0^+ = \overline{Q_0}\, x + Q_0 \overline{x} = Q_0 \oplus x$$



* Design with unused states:-

* How to accomodate for unused states ?! 3approaches

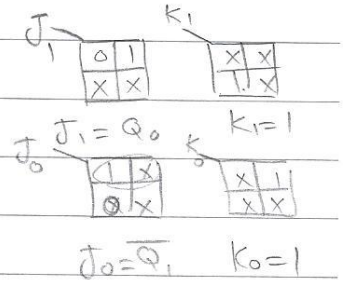
1. Assume the next state & the unused states to be X (don't cares).
2. Force NS of the unused state to be one of the used states.
3. Include a special output to indecate that the PS is unused. This output can be used to change the state asynchronously.

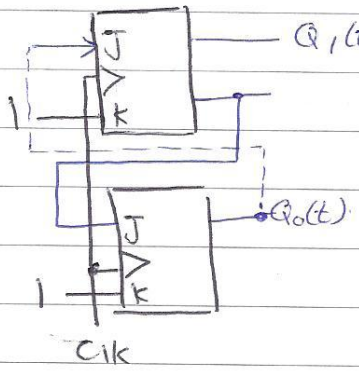
ex: Design a counter that counts 0, 1, 2, 0, 1, 2, ... using JK ff

3 used states → 2 ff's → 4 available states (1 unused)

approach #1: Assume (X).

PS $Q_1\ Q_0$	NS $Q_1^+\ Q_0^+$	$J_1\ K_1$	$J_0\ K_0$
0 0	0 1	0 X	1 X
0 1	1 0	1 X	X 1
1 0	0 0	X 1	0 X
1 1	X X	X X	X X

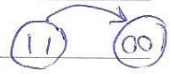




at 11

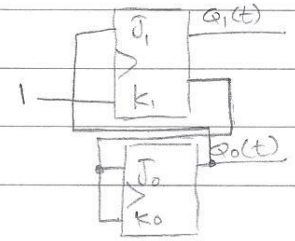
$J_1 = 1$, $K_1 = 1$ → $Q_1 = 0$

$J_0 = 0$, $K_0 = 1$ → $Q_0 = 0$



app#2: Force the circuit to go to state (01) if in (11).

PS		NS			
Q	Q_0	Q_1^+	Q_0^+	J_1K_1	J_0K_0
0	0	0	1	0 X	1 X
0	1	1	0	1 X	X 1
1	0	0	0	X 1	0 X
1	1	0	1	X 1	X 0

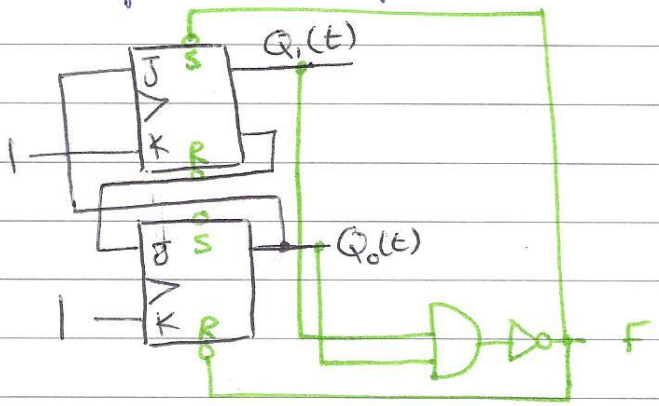


* transition happens synchronously

$\Rightarrow J_1 = Q_0(t)$ $\quad J_0 = \overline{Q}_1(t)$

$K_1 = 1$ $\quad K_0 = \overline{Q}_1(t)$

app#3: Special output



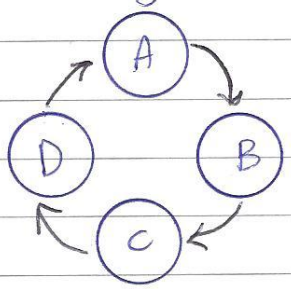
* transition happens Asynchronously.

** State Assignment:-

* we have many approaches:-

1. counting order.
2. gray - code.
3. One hot key.
4. Random.

ex: Design using counting order assignment.



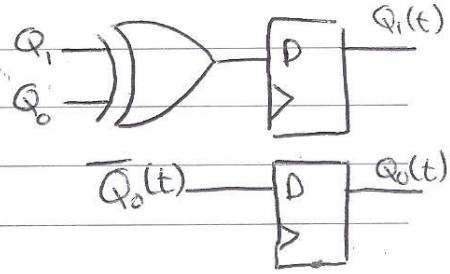
A=00 B=01 C=10 D=11

# of ffs = 2 # of states = 4

PS	NS
Q_1Q_0	$Q_1^+ Q_0^+$
0 0	0 1
0 1	1 0
1 0	1 1
1 1	0 0

$Q_1^+ = Q_1 \oplus Q_0$

$Q_0^+ = \overline{Q_0}(t)$



#2 Design using gray-code:

A=00, B=01, C=11, D=10

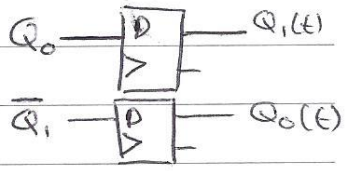
# of ffs = 2 / # of states = 4

PS	NS
Q_1Q_0	$Q_1^+ Q_0^+$
0 0	0 1
0 1	1 1
1 0	0 0
1 1	1 0

$D_1 = Q_1^+ = Q_0(t)$

$D_0 = Q_0^+ = \overline{Q_1}(t)$

* decreases the cost !



#3 design One-hot-Key:

A = 0001

B = 0010

C = 0100

D = 1000

PS	NS
$Q_3Q_2Q_1Q_0$	$Q_3^+ Q_2^+ Q_1^+ Q_0^+$
0 0 0 1	0 0 1 0
0 0 1 0	0 1 0 0
0 1 0 0	1 0 0 0
1 0 0 0	0 0 0 1

$Q_3^+ = Q_2$

$Q_2^+ = Q_1$

$Q_1^+ = Q_0$

$Q_0^+ = Q_3$

** State equivalence & Reduction.

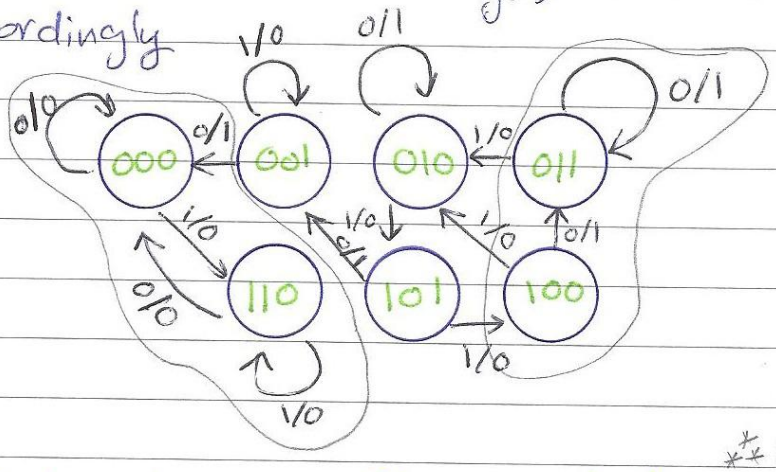
→ if we have 2^n states then # of ff is n.

→ we can reduce the cost of the machine if we reduce # of ffs. However, this requires reducing # of states.

→ we can take advantage of state equivalence!

** two states are equivalent if for any input sequence they have the same next state & output.

ex:- study the following state diagram & identify any equivelant states (if any), reduce the diagram accordingly



test for equivelance:-

$000 \neq 001$ (diff. outputs).

$000 \neq 010$

$000 \neq 011$

$000 \neq 100$

$000 \neq 101$

$000 \equiv 110$

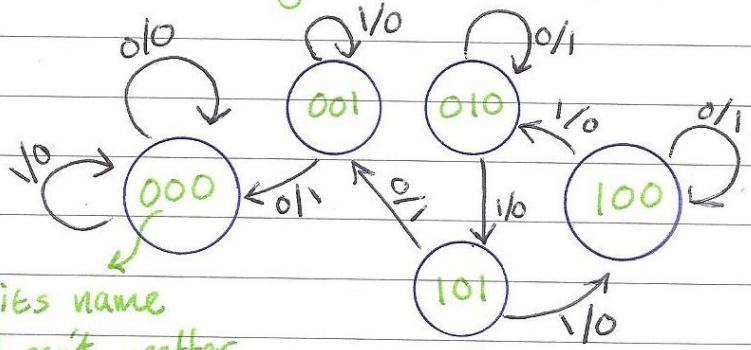
** have the same out & NS for any input (0/1)

$001 \neq 010$

⋮

$011 \equiv 100$

⇝ state diagram after reduction:-



its name doesn't matter

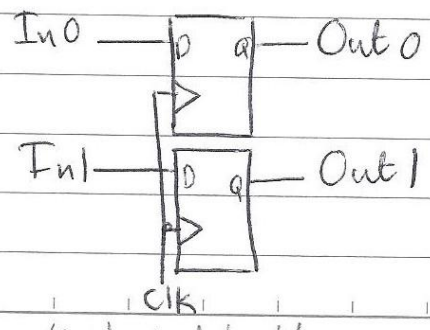
*Chapter #7: Registers & counters

≡ special (frequently used) types of sequential circuits.

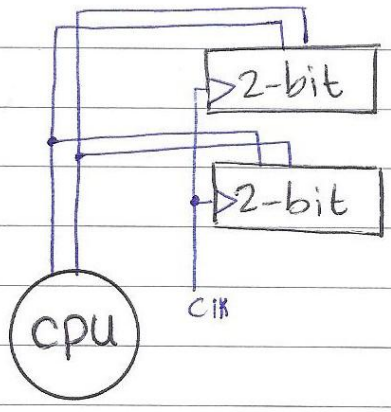
A) Registers:

- A collection of (n) storage elements that can be used to store n bits.

ex: 2-bit Register.



(synchronised to the same clock).



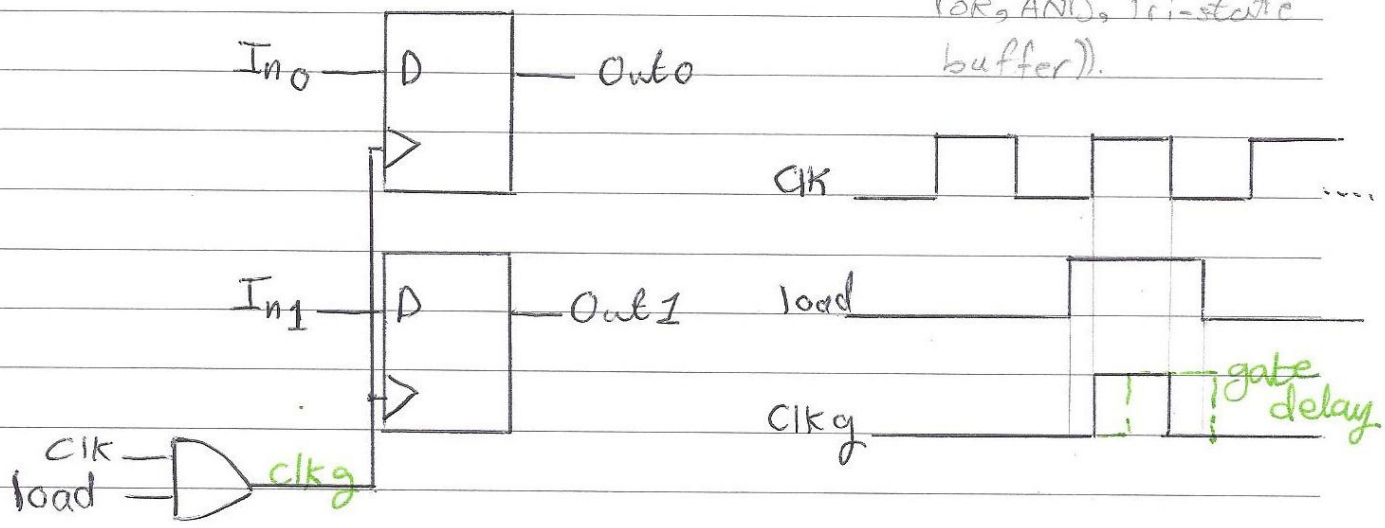
* Problem: whenever we write to one of the registers it will be written on the other!

So... we need to control when to hold & when to load new value on the register.

! What do we expect from a register?

* to hold data for multiple cycles & to an explicit input to determine weither to load the new data or not.

→ approach #1: Gated Clock (using enabling circuit (OR, AND, Tri-state buffer)).

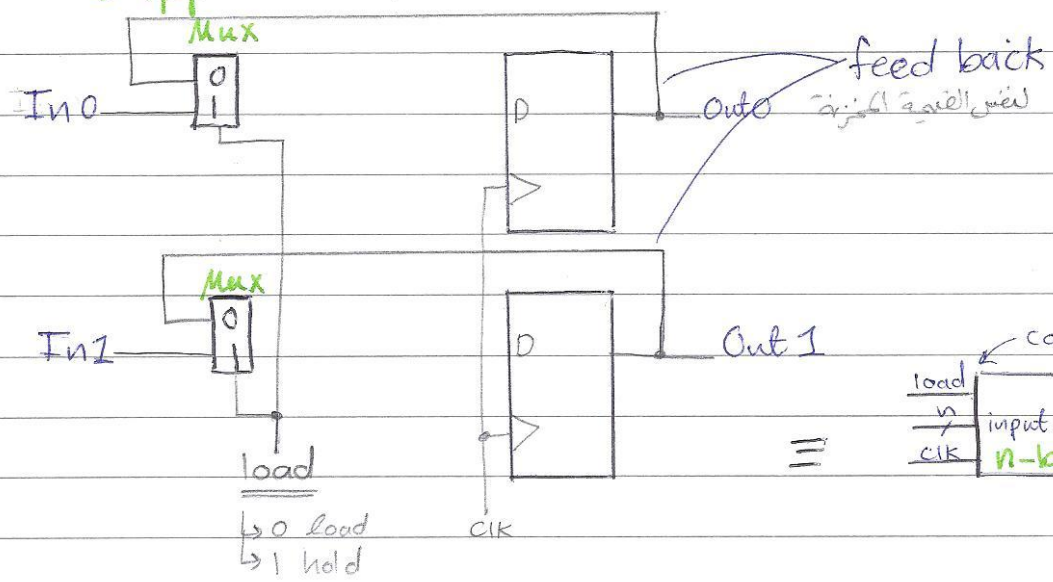


Any problem?!

→ Clock skew (تأخير)

when we add gates to clock it'll be delayed unlike the remaining system elements that'll be synchronized to the original clock (without delay).

→ approach ✱2:



✱✱ Shift Register :-

- It's a register that can shift its contents toward the (most Significant bit) MSB (Shift left) or toward the LSB (Shift right)

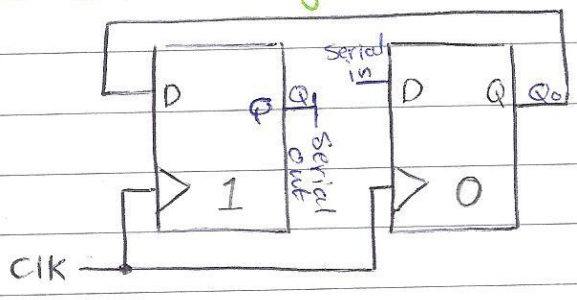
* Why do we need shift registers ?!

1. Cheap multiplication & division by powers of 2.
2. Most (CPUs support (have) shift instructions.
3. Serial communications.

Note:- In parallel we deal with all bits as one intity so it's more expensive (need lot of wires) & may have problems.
while in serial bits are processed bit by bit so it's slower but cheaper.
In order to process data in serial we need to use shift registers.

~~Ex~~

ex:- 2-bit Shift Register:



clock pulse			serial In
CP	Q_1	Q_0	
t_0	?	?	0
t_1	?	0	1
t_2	0	1	1
t_3	1	1	0

shift left { PS, NS } Q_1 out ← Q_0 ← Q_0 ? = external input (serial in)

* Note: : Shift register في delay الـ فائدة

* Assume all gates in FFs are ideal, then time needed to store a value in Q_0, Q_1 is 0, and since Q_0 is connected to the 1st FF, value of Q_1 will be equal to Q_0 before the (falling/rising) edge ends, so it's not a shift register any more!!

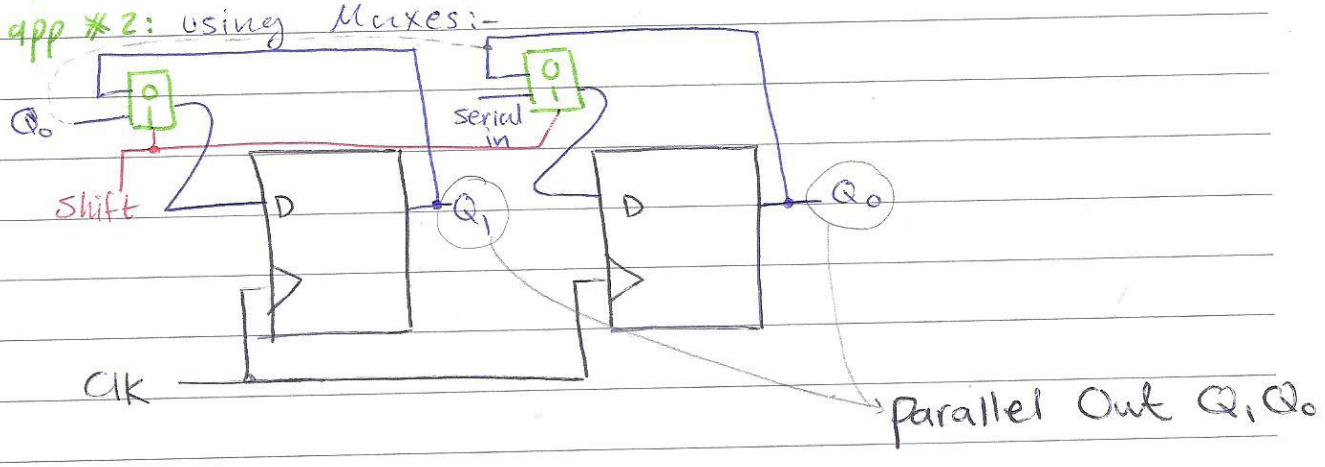
* while delay will allow the (falling/rising) edge to end before the new state of Q_0 enters the 1st FF.

** Modifying shift register to control when to perform the shift operation.

* the previous register shifts 1-bit per cycle.

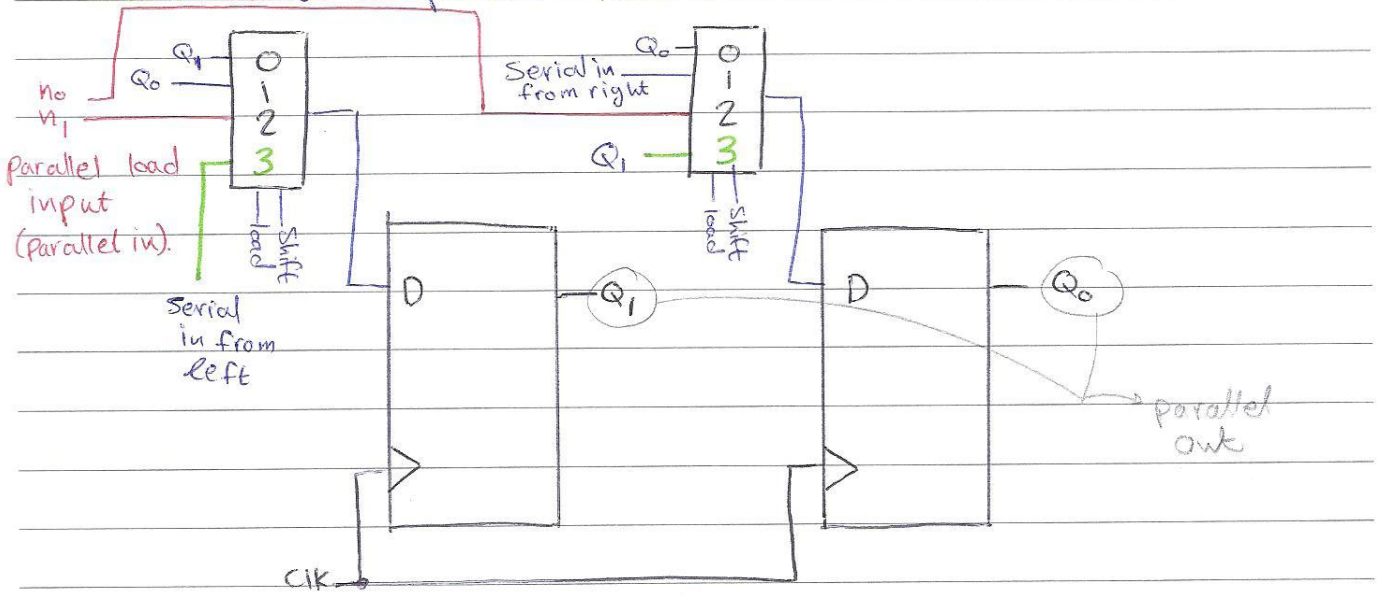
app *1: Gated clock (delay)

app *2: using Muxes:-



* In the previous example, the ~~of~~ only way we can initialize this register is serially.

So... How to do parallel load ?

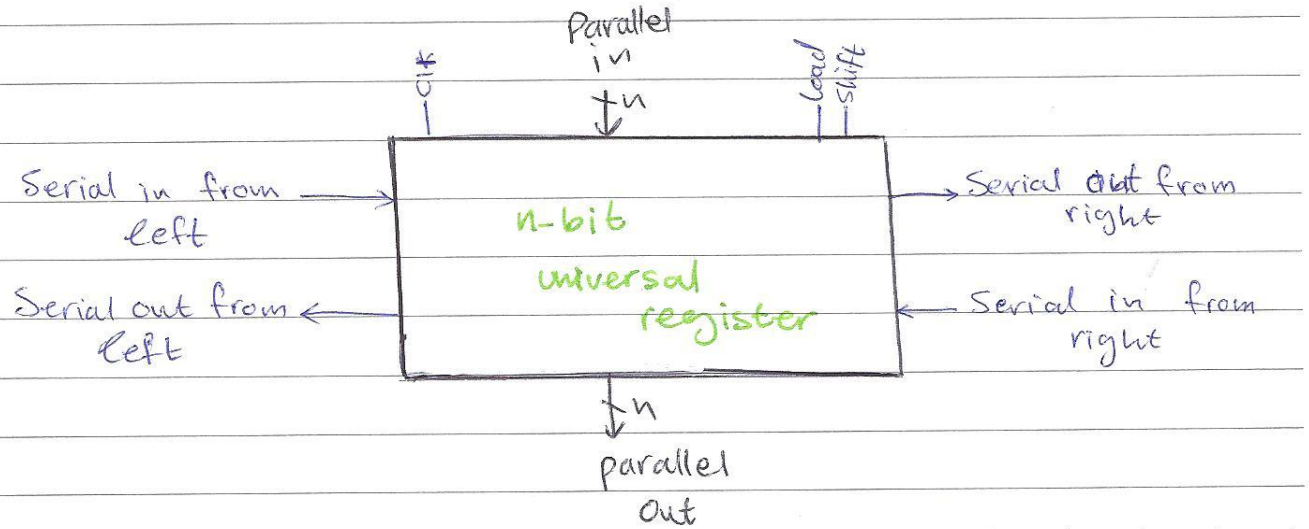


Load	Shift	Operation
0	0	HOLD
0	1	Shift left
1	0	Parallel load
1	1	Shif Right

choice of the user.

Q_0 ⇒ serial out from right
Q_1 ⇒ Serial " " left
$Q_1 Q_0$ ⇒ Parallel out
$n_1 n_0$ ⇒ parallel in

⇒ Universal shift Register : it does 4 basic operations
1. HOLD. 2. Shift Right 3. Shift left 4. Parallel load.



Counters:-

* It's a sequential circuit that goes through a predefined sequence of states.

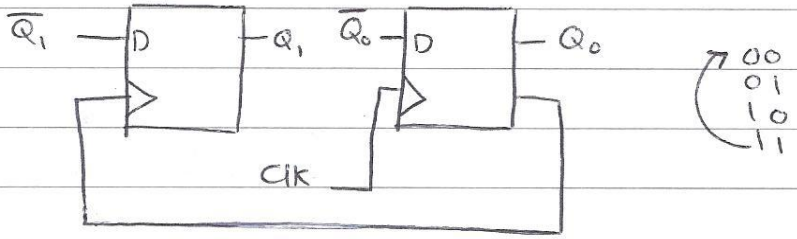
→ two types:

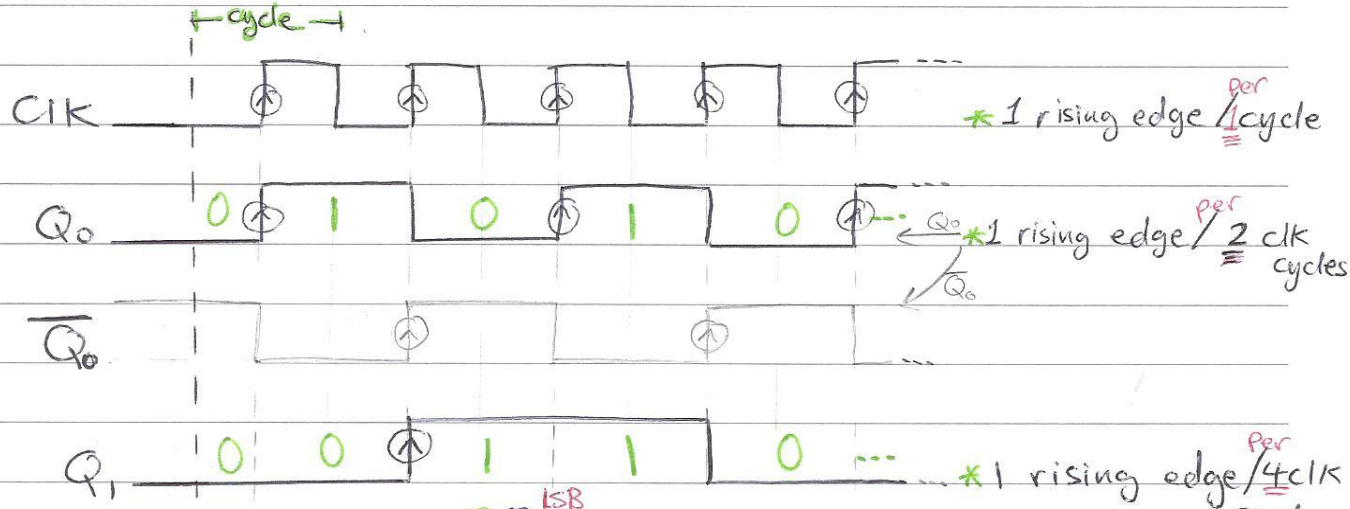
1. Asynchronous (there's no time synchronization) → usually cheaper but harder to design.
2. Synchronous (discussed earlier).

* Ripple counter (Asynch).

* Not All storage elements are synchronised to the same time base (clk).
  * * there's a clock that drives LSB.
  * * the clock inputs for the remaining FFs are connected to the output of previous FFs.

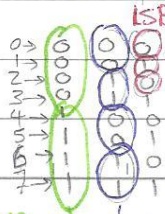
2-bit ripple counter:-





* 1 rising edge per 1 cycle
* 1 rising edge per 2 clk cycles.
* 1 rising edge per 4 clk cycles

* when we count (3 bits as an ex).

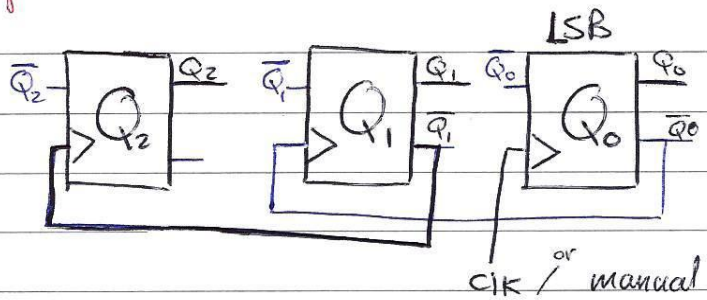


→ changes every 1 state (that's why it's connected to clk).

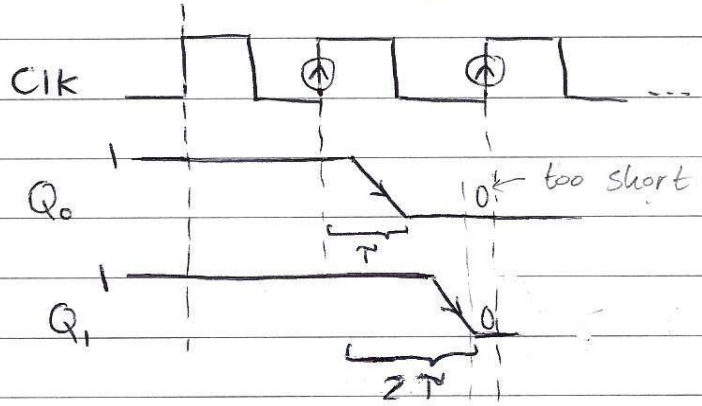
changes every 4 states

→ changes every 2 states

→ 3-bit Ripple Counter:-



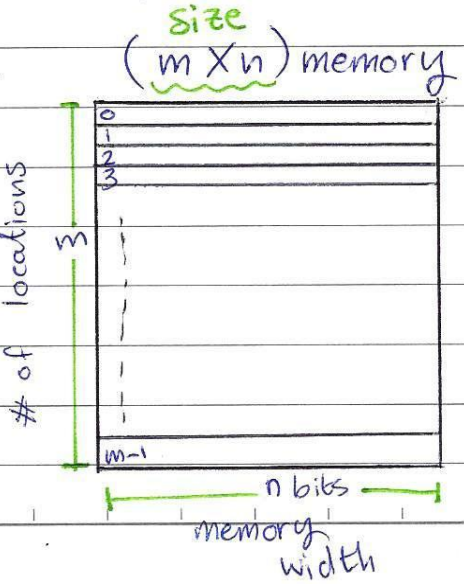
** the problem of (accumelated delay) because every ↑ one of FFs is driven by an other.



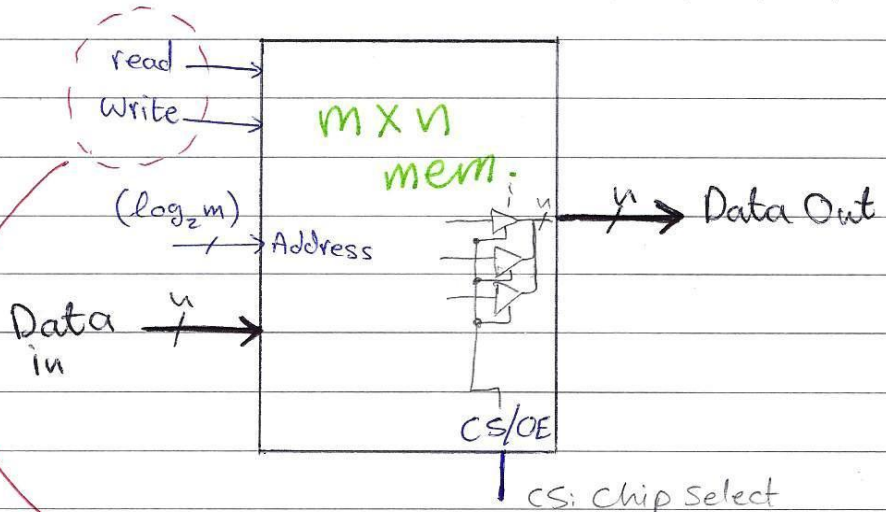
* Solution:
1. buy better FFs (less delay).
2. Reduce the frequency of clock.

Chapter # 8: Memory Basics:-

* Memory is a collection of storage cells that are arranged as an array.



* 8bits ≡ 1 byte
* memory size in bits = m X n
* " " " bytes = $\frac{m \times n}{8}$



we can replace these 2 signals with (dual purpose) $input ⇒ R/W → 0 : read
↳ 1 : write

** Building 800x8 mem. from 2 (400x8) mems.