Logic

د. محمد عبد المجيد

Esraa Ahmad AL-Ajou
Computer Engineering

موضوع الدرس _____ اليوم _____ التاريخ / /

5 Binary Varibles * رح نتعامل مع

0 1

* رح اجري عمليات على Binary varib

logical operatos تسمى

AND
OR
NOT
→ logical operbos 4

AND
OR
Not
→ عبارة عن رمز operator ل Gates * logic

تشبه لإشارة

6

T
F
→ Binary varible *

on
off

Yes No 1 0

7] logical operation

And (·) مستعملة اكثر

OR (+)

Not (~) (') (-)

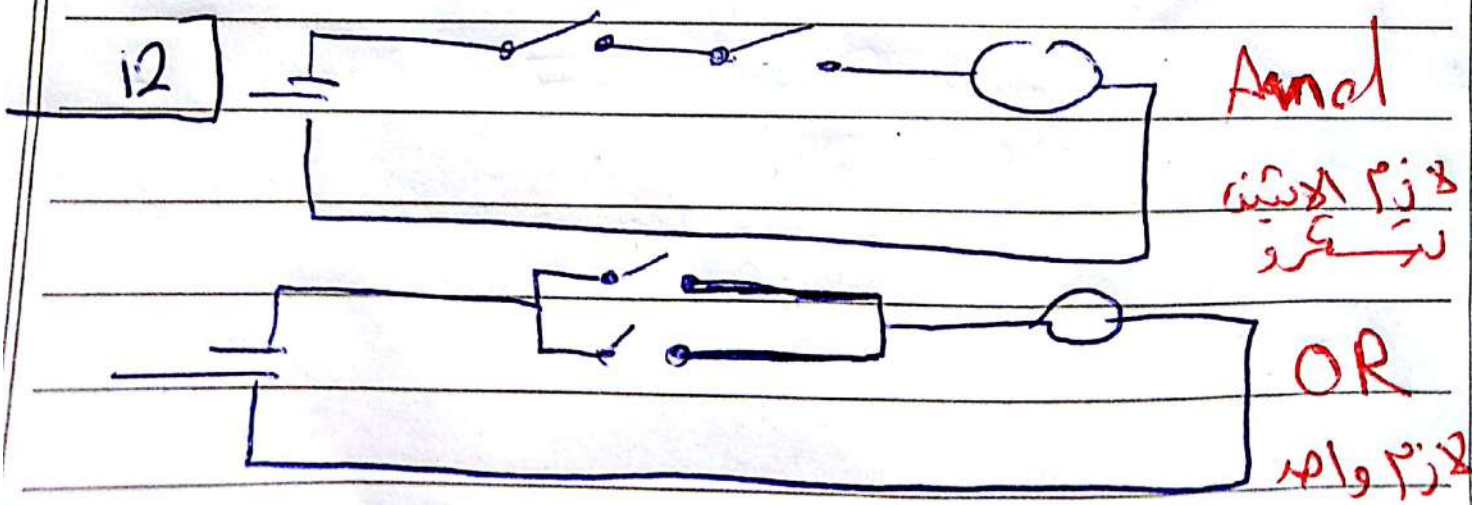
8] $z = x + y = x \vee y$ (OR)

$z=1$ if $x=1$ / $y=1$ / $x=1$ $y=1$

$z = x \cdot y = x \wedge y$ (and)

$z=1$ only if $\longrightarrow$ $x=1$ and $y=1$

12]



And

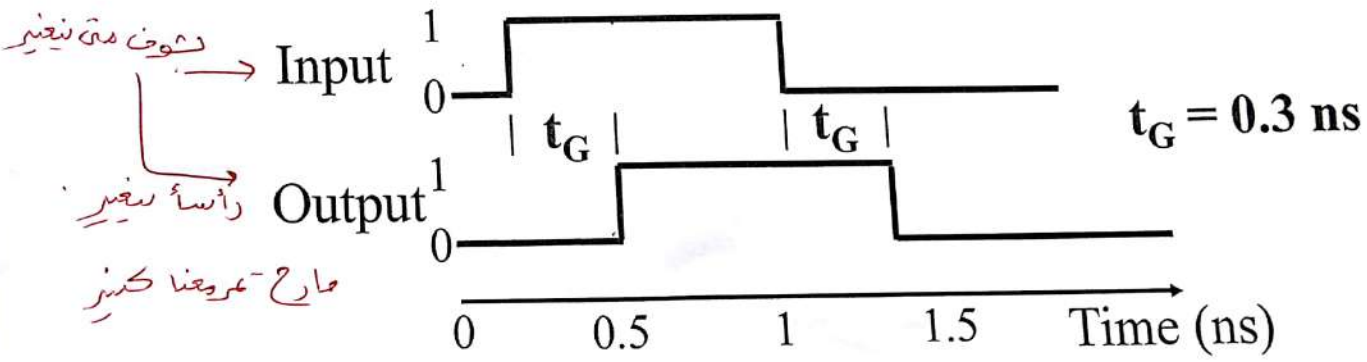
لازم الاشيئ يسكرو

OR

لازم واحد على الاقل يسكر

Gate Delay

- In actual physical gates, if one or more input changes causes the output to change, the output change does not occur instantaneously
- The delay between an input change(s) and the resulting output change is the ***gate delay*** denoted by t_G:



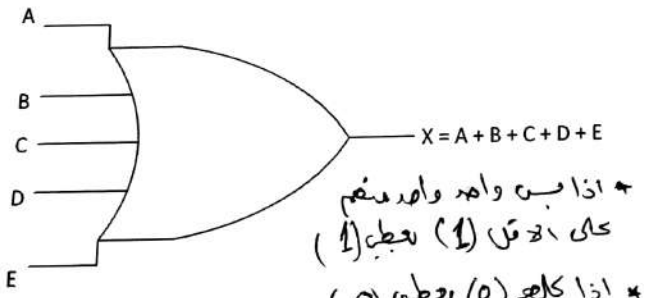
Logic Gates: Inputs and Outputs

- NOT (inverter)
  - Always one input and one output

لازم واحد فقط ← مدخل / مخرج

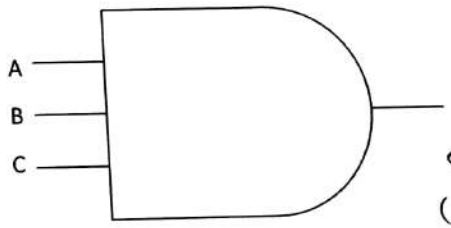
- AND and OR gates
  - Always one output
  - Two or more inputs

واحد او اكثر



* اذا بس واحد منهم على الاقل (1) بعطي (1)
* اذا كلهم (0) بعطي (0)

$X = A + B + C + D + E$



* اذا بس واحد على الاقل (0) بعطي (0)

اذا كان كلهم (1) بعطي (1)

Boolean Algebra

- An algebra dealing with binary variables and logic operations
  - Variables are designated by letters of the alphabet
  - Basic logic operations: AND, OR, and NOT

- ***A Boolean expression*** is an algebraic expression formed by using **binary variables, constants 0 and 1,** the **logic operation symbols,** and **parentheses**
  - E.g.: X . 1, A + B + C, (A + B)(C + D) → Boolean expression

- ***A Boolean function*** consists of a binary variable identifying the function followed by equals sign and a Boolean expression
  - E.g.: $F = A + B + C$, $L(D, X, A) = DX + \bar{A}$ → Boolean function

Fun

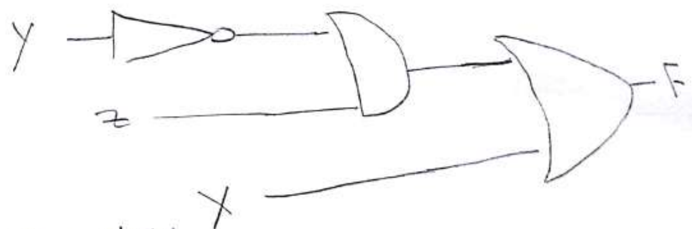
Logic Diagrams and Expressions

الأولويات
Not
And
OR

1. Equation: $F = X + \bar{Y}Z$

2. Logic Diagram:

$2^3 = 8$



3. Truth Table:

input

output $X + \bar{Y}Z$

X	Y	Z	output
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	1

or → واحد يعطي 1

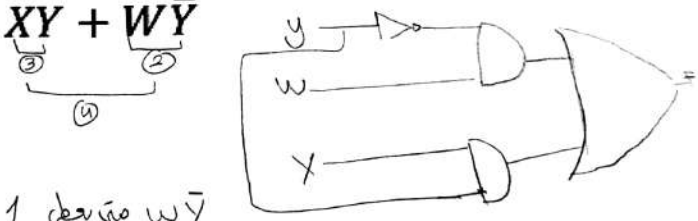
and → الاثنين 1

اذا كان
$\bar{Y}$ X
0 1 → 1

X + 0
اذا كانت
1 + 0
1

Example

- Draw the logic diagram and the truth table of the following Boolean function: $F(W, X, Y) = XY + W\bar{Y}$



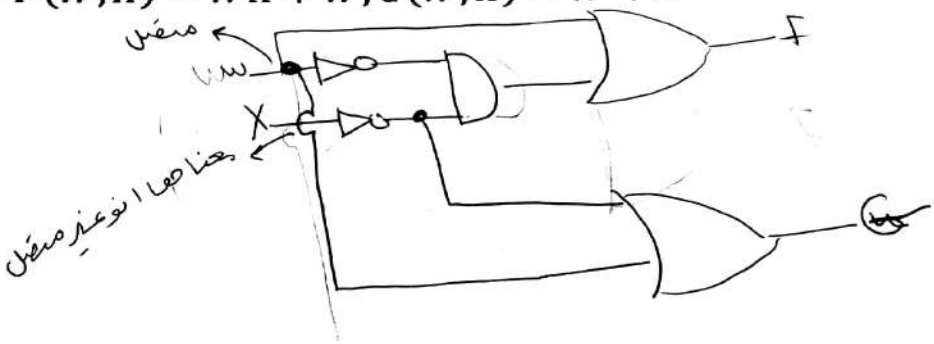
- Logic Diagram:
- Truth Table:

W	X	Y	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	1

- This example represents a ***Single Output Function***

- Draw the logic diagram and the truth table of the following Boolean functions: $F(W,X) = \bar{W}\bar{X} + W, G(W,X) = W + \bar{X}$

- Logic Diagram:



- Truth Table:

W	X	F	G
0	0	1	1
0	1	0	0
1	0	1	1
1	1	1	1

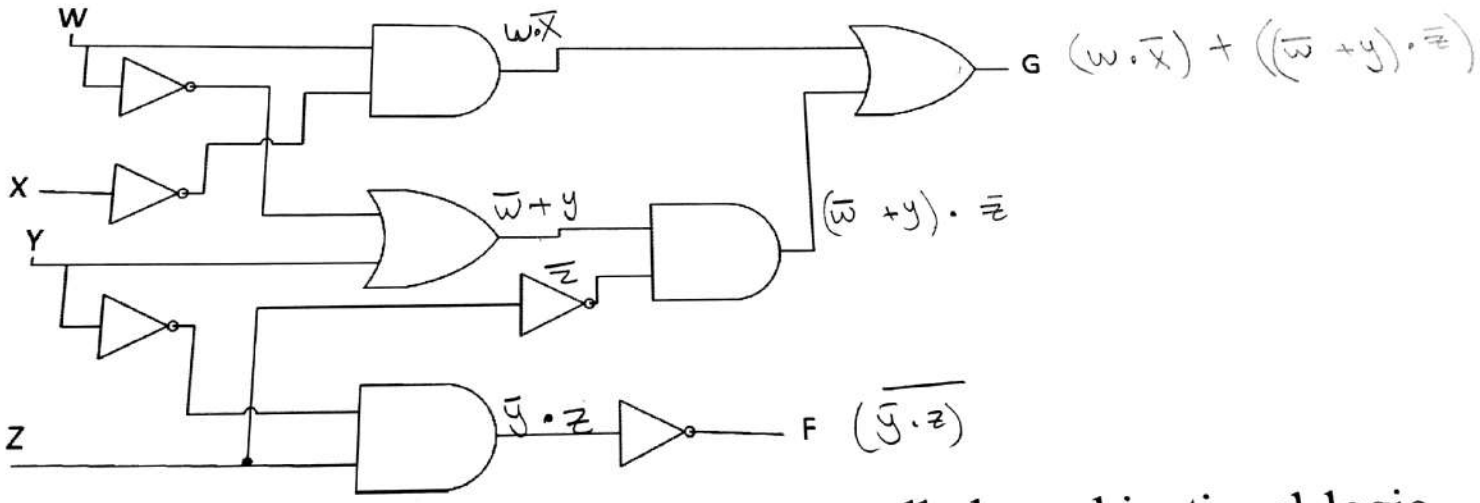
$\bar{W}\bar{X}$ يعطي 1 اذا الاثنين zero

$+ W$ يعطي 1 اذا كان 1

W يعطي 1 اذا كان 1

$+ \bar{X}$ يعطي 1 اذا كان zero

- Given the following logic diagram, write the corresponding Boolean equation:



- *Logic circuits of this type are called combinational logic circuits since the variables are combined by logical operations*

ic Identities of Boolean Algebra

1. $X + 0 = X$	2. $X \cdot 1 = X$	Existence of 0 and 1
3. $X + 1 = 1$	4. $X \cdot 0 = 0$	
5. $X + X = X$	6. $X \cdot X = X$	Idempotence
7. $X + \bar{X} = 1$	8. $X \cdot \bar{X} = 0$	Existence of complement
9. $\bar{\bar{X}} = X$		Involution
10. $X + Y = Y + X$	11. $XY = YX$	Commutative Laws
12. $(X + Y) + Z = X + (Y + Z)$	13. $(XY)Z = X(YZ)$	Associative Laws
14. $X(Y + Z) = XY + XZ$	15. $X + YZ = (X + Y)(X + Z)$	Distributive Laws
16. $\overline{X + Y} = \bar{X}.\bar{Y}$	17. $\overline{X.Y} = \bar{X} + \bar{Y}$	DeMorgan's Laws

Some Properties of Identities & the Algebra

- If the meaning is unambiguous, we leave out the symbol "·"
- The identities above are organized into pairs
  - The ***dual*** of an algebraic expression is obtained by interchanging (+) and (·) and interchanging **0's** and **1's**
  - The identities appear in ***dual*** pairs. When there is only one identity on a line the identity is ***self-dual***, i. e., the dual expression = the original expression.

demorg و dual معرفة
لازم احفظ على الاولايات والاقواس

X
X
كما هي

* demorg يعني Var يبقى كما هو
* يتم تغيير
  · ← +
  + ← ·
  1 ← 0
  0 ← 1

Some Properties of Identities & the Algebra (Continued)

- Unless it happens to be self-dual, the dual of an expression does not equal the expression itself
- Examples:
  - $F = (A + \bar{C}) \cdot B + 0$
    - *Dual* $F = ((A \cdot \bar{C}) + B) \cdot 1$
  - $G = XY + (\overline{W + Z})$
    - *Dual* $G = (X + Y) \cdot (\overline{W}\,\overline{Z}) = (X + Y) \cdot (\overline{W} + \overline{Z})$
  - $H = AB + AC + BC$
    - *Dual* $H = (A + B) \cdot (A + C) \cdot (B + C)$
- Are any of these functions self-dual?
  - Yes, H is self-dual

Some Properties of Identities & the Algebra (Continued)

- Unless it happens to be self-dual, the dual of an expression does not equal the expression itself
- Examples:
  - $F = (A + \bar{C})\,.\,B + 0$
    - $Dual\ F = (A\,.\,\bar{C}) + \mathrm{B}\,.1 = \mathrm{A}\,.\,\bar{C} + B$ (Not Accurate)
    - $Dual\ F = \big((A\,.\,\bar{C}) + \mathrm{B}\big).1 = \mathrm{A}\,.\,\bar{C} + B$ (Accurate)
  - $G = \mathrm{XY} + \overline{(W + Z)}$
    - $Dual\ G = (X + Y).\overline{WZ} = (X + Y)\,.\,(\bar{W} + \bar{Z})$
  - $H = AB + AC + BC$
    - $Dual\ H = (A + B)(A + C)(B + C) = (A + BC)(\mathrm{B} + \mathrm{C})$

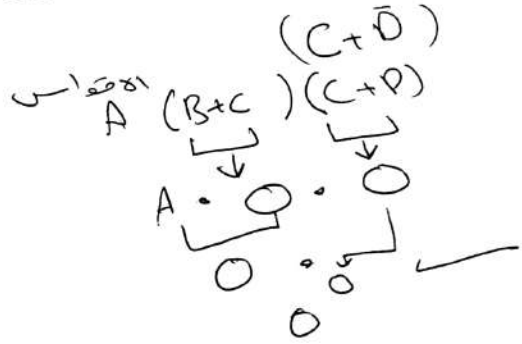
    $= AB + AC + BC$
- Are anv of these functions self-dual?

Boolean Operator Precedence

- The order of evaluation in a Boolean expression is:
  1. Parentheses
  2. NOT
  3. AND
  4. OR

- Consequence: Parentheses appear around OR expressions

- Examples:
  - $F = A(B + C)(C + \bar{D})$
  - $F = \sim\!AB = \bar{A}B$
  - $F = \bar{A}B + C$
  - $F = A(B + C)$



Useful Boolean Theorems

Theorem	Dual	Name
(1) $x.y + \bar{x}.y = y$	(2) $(x+y)(\bar{x}+y) = y$	Minimization
(3) $x + x.y = x$	(4) $x.(x+y) = x$	Absorption
(5) $x + \bar{x}.y = x + y$	(6) $x.(\bar{x}+y) = x.y$	Simplification
$x.y + \bar{x}.z + y.z = x.y + \bar{x}.z$		Consensus
$(x+y)(\bar{x}+z)(y+z) = (x+y)(\bar{x}+z)$		

(6) $x.(\bar{x}+y)$ توزيع
$(x.\bar{x}) + (x.y)$
$0 + x.y = x.y$

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

(4) $x.(x+y)$ توزيع
$(x.x) + (x.y)$
$x + (xy)$
$x(1+y)$
x

(5) $x + \bar{x}.y = x + y$
$(x+\bar{x})(x+y)$
$1\,(x+y)$
$(x+y) = (x+y)$

$x + yz = (x+y)(x+z)$
$a + bc = (a+b)(a+c)$

Variable + Variable (نفس المتغير) = Variable

Example 1: Boolean Algebraic Proof

3. $A + A \cdot B = A$ (Absorption Theorem)

$A(1+B)$
$= A \cdot 1$
$= A$

$= A \cdot 1 + A \cdot B$	$X = X \cdot 1$
$= A \cdot 1$	$1 + X = 1$

- Our primary reason for doing proofs is to learn:
  - Careful and efficient use of the identities and theorems of Boolean algebra
  - How to choose the appropriate identity or theorem to apply to make forward progress, irrespective of the application

Example 2: Boolean Algebraic Proofs

- $AB + \bar{A}C + BC = AB + \bar{A}C$ (Consensus Theorem)

Proof Steps	Justification (identity or theorem)
$AB + \bar{A}C + BC$	
$= AB + \bar{A}C + 1.BC$	$1.X = X$
$= AB + \bar{A}C + (A + \bar{A}).BC$	$X + \bar{X} = 1$
$= AB + \bar{A}C + ABC + \bar{A}BC$	*Distributive Law*
$= AB + ABC + \bar{A}C + \bar{A}BC$	*Commutative Law*
$= AB.1 + AB.C + \bar{A}C.1 + \bar{A}C.B$	$X.1 = X$ *and Commutative Law*
$= AB(1 + C) + \bar{A}C(1 + B)$	*Distributive Law*
$= AB.1 + \bar{A}C.1$	$1 + X = 1$
$= AB + \bar{A}C$	$X.1 = X$

Proof of Simplification

5. $A + \bar{A}.B = A + B$ (Simplification Theorem)

$A + \bar{A}.B$	
$=(A+\bar{A})(A+B)$	*Distributive law*
	Factor B and Distributive Law
$= (A + B)$	$X + \bar{X} = 1$

4. $A.(\bar{A} + B) = AB$ (Simplification Theorem)

$A.(\bar{A} + B)$	
$= (A.\bar{A}) + (A.B)$	*Distributive Law*
$= 0 + AB$	$X.\bar{X} = 0$
$= AB$	$X + 0 = X$

Proof of Minimization

1. $A.B + \bar{A}.B = B$ (Minimization Theorem)

$A.B + \bar{A}.B$	
$= B(A + \bar{A})$	*Distributive Law*
$= B.1$	$X + \bar{X} = 1$
$= B$	$X.1 = X$

2. $(A + B)(\bar{A} + B) = B$ (Minimization Theorem)

$(A + B)(\bar{A} + B)$	
$= B + (A.\bar{A})$	*Distributive Law*
$= B + 0$	$X.\bar{X} = 0$
$= B$	$X + 0 = X$

Proof of DeMorgan's Laws (1)

- $\overline{X+Y} = \bar{X}.\bar{Y}$ (DeMorgan's Law)
  - We will show that, $\bar{X}.\bar{Y}$, satisfies the definition of the complement of $(X+Y)$, defined as $\overline{X+Y}$ by DeMorgan's Law.
  - To show this, we need to show that $A + A' = 1$ and $A.A' = 0$ with $A = X + Y$ and $A' = X'.Y'$. This proves that $X'.Y' = \overline{X+Y}$.
- Part 1: Show $(X + Y) + X'.Y' = 1$

طريقة تصنيع

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

$(1+y)$. $(x+1)$

1 . 1

1

Proof of DeMorgan's Laws (2)

- Part 2: Show $(X + Y).X'.Y' = 0$

$(x\bar{x}\bar{y}) + (y\bar{y}\bar{x}) = 0$

$(0 \cdot \bar{y}) + (0\bar{x}) =$

$0 + 0$

$0 = 0$

- Based on the above two parts, $X'.Y' = \overline{X + Y}$
- The second DeMorgans' law is proved by duality
- Note that DeMorgan's law, given as an identity is not an axiom in the sense that it can be proved using the other identities.

Example 3: Boolean Algebraic Proofs

- $\overline{(X+Y)}Z + X\bar{Y} = \bar{Y}(X+Z)$

Example 3: Boolean Algebraic Proofs

- $\overline{(X+Y)}Z + X\bar{Y} = \bar{Y}(X+Z)$

$\overline{(X+Y)}Z + X\bar{Y}$	
$= X'Y'Z + X(Y')$	*DeMorgan's law*
$= Y'(X'Z + X)$	*Distributive law*
$= Y'(X + X'Z)$	*Commutative law*
$= Y'(X + Z)$	*Simplification theorem*

Boolean Function Evaluation

- $F_1 = xy\bar{z}$
- $F_2 = x + \bar{y}z$
- $F_3 = \bar{x}\bar{y}\bar{z} + \bar{x}yz + x\bar{y}$
- $F_4 = x\bar{y} + \bar{x}z$

x	y	z	F_1	F_2	F_3	F_4
0	0	0	0	0	1	0
0	0	1	0	1	0	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

Expression Simplification

- An application of Boolean algebra
- Simplify to contain the smallest number of literals (complemented and uncomplemented variables)
- Example: Simplify the following Boolean expression
  - $AB + A'CD + A'BD + A'CD' + ABCD$

14 literals → 5 literals

$AB + A'CD + A'BD + A'CD' + ABCD$	
$= AB + ABCD + A'CD + A'CD' + A'BD$	*Commutative law*
$= AB(1 + CD) + A'C(D + D') + A'BD$	*Distributive law*
$= AB.1 + A'C.1 + A'BD$	$1 + X = 1$ *and* $X + X' = 1$
$= AB + A'C + A'BD$	$X.1 = X$
$= AB + A'BD + A'C$	*Commutative law*
$= B(A + A'D) + A'C$	*Distributive law*
$= B(A + D) + A'C$ → 5 Literals	*Simplification Theorem*

Complementing Functions

- Use DeMorgan's Theorem to complement a function:
  1. Interchange AND and OR operators
  2. Complement each constant value and literal

- Example: Complement $F = (x'yz') + (xy'z')$

$$F' = (x + y' + z)(x' + y + z)$$

- Example: Complement $G = (a' + (bc))d' + e$

$$G' = (a(b' + c') + d).e'$$

Example

- Simplify the following:
  - $F = X'YZ + X'YZ' + XZ$

$x'y(z+\bar{z}) + xz$

$xy'1 + xz$

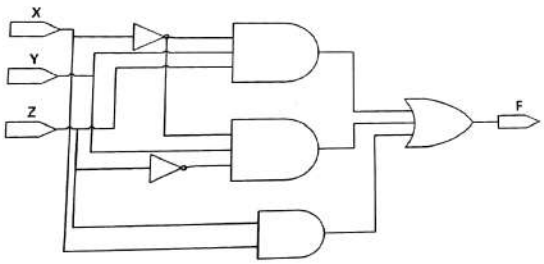
$x(y'+z)$

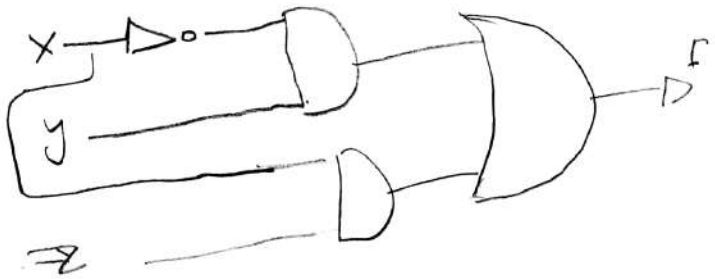
Example

- Simplify the following:
  - $F = X'YZ + X'YZ' + XZ$

$xy(z + z') + xz$

$xy + xz$





Example

Show that Truth table / Boolean Algebra

- Show that $F = x'y' + xy' + x'y + xy = 1$
  - Solution1: Truth Table

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

$y'(x'+x) + y(x'+x)$

$y' \cdot 1 + y$

1

  - Solution2: Boolean Algebra

$x'y' + xy' + x'y + xy$	
$= y'(x' + x) + y(x' + x)$	*Distributive law*
$= y'.1 + y.1$	$X + X' = 1$
$= y' + y$	$X.1 = X$
$= 1$	$X + X' = 1$

Examples

- Show that $ABC + A'C' + AC' = AB + C'$ using Boolean algebra.

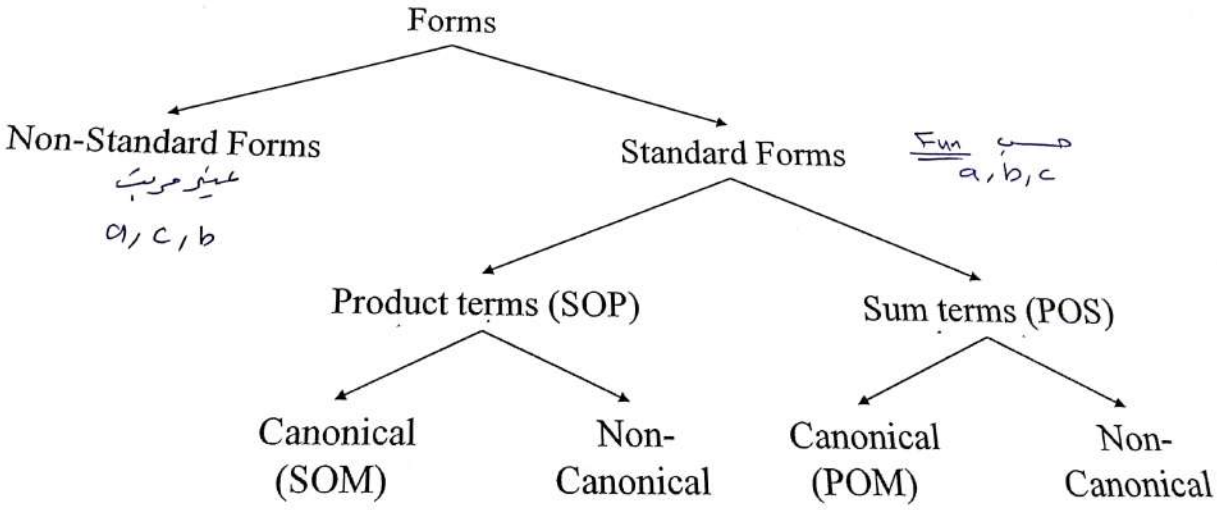
$ABC + A'C' + AC'$	
$= ABC + C'(A' + A)$	*Distributive law*
$= ABC + C'.1$	$X + X' = 1$
$= ABC + C'$	$X.1 = X$
$= (AB + C')(C + C')$	
$= (AB + C').1$	$X + X' = 1$
$= AB + C'$	$X.1 = X$

- Find the dual and the complement of $f = wx + y'z.0 + w'z$
  - $Dual(f) = (w + x)(y' + z + 1)(w' + z)$
  - $f' = (w' + x')(y + z' + 1)(w + z')$

Overview – Canonical Form

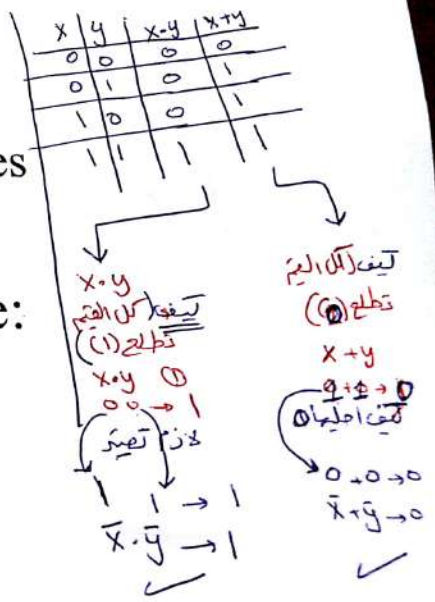
- What are Canonical Forms?
- Minterms and Maxterms
- Index Representation of Minterms and Maxterms
- Sum-of-Minterm (SOM) Representations
- Product-of-Maxterm (POM) Representations
- Representation of Complements of Functions
- Conversions between Representations

Boolean Representation Forms



Canonical Forms

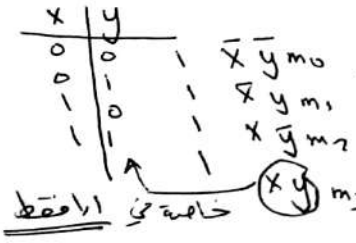
- It is useful to specify Boolean functions in a form that:
  - Allows comparison for equality
  - Has a correspondence to the truth tables
  - Facilitates simplification
- Canonical Forms in common usage:
  - Sum of Minterms (SOM)
  - Product of Maxterms (POM)



Minterms

- ***Minterms*** are AND terms with ***every variable*** present in either true or complemented form
- Given that each binary variable may appear normal (e.g., x) or complemented (e.g., $\bar{x}$), there are 2^n minterms for n variables
- Example: Two variables (X and Y) produce $2^2 = 4$ combinations:

XY	(both normal)
$X\bar{Y}$	(X normal, Y complemented)
$\bar{X}Y$	(X complemented, Y normal)
$\bar{X}\bar{Y}$	(both complemented)



- Thus there are ***four minterms*** of two variables

Maxterms

- ***Maxterms*** are OR terms with ***every variable*** in true or complemented form
- Given that each binary variable may appear normal (e.g., x) or complemented (e.g., $\bar{x}$), there are 2^n maxterms for n variables
- Example: Two variables (X and Y) produce $2^2 = 4$ combinations:

$X + Y$ (both normal)
$X + \bar{Y}$ (X normal, Y complemented)
$\bar{X} + Y$ (X complemented, Y normal)
$\bar{X} + \bar{Y}$ (both complemented)

* عبارة عن terms في كل الحيز الموجودين Valas

① ممكن x,y او x,y,z

② الاشارة اي بينهم (+) or

③ لازم يطلع الجواب Zero

x	y	
0	0	$M_0 \; x \; y$
0	1	$M_1 \; x \; \bar{y}$
1	0	$M_2 \; \bar{y} \; x$
1	1	$M_3 \; \bar{x} \; \bar{y}$

مكان كل (1) بحطه (-)

* الجزء المتمثل ك

Maxterms and Minterms

- Examples: Three variable (X, Y, Z) minterms and maxterms

Index	X,Y,Z	Minterm (m)	Maxterm (M)
0	000	m_0 $\bar{X}\bar{Y}\bar{Z}$	M_0 $X + Y + Z$
1	001	m_1 $\bar{X}\bar{Y}Z$	M_1 $X + Y + \bar{Z}$
2	010	m_2 $\bar{X}Y\bar{Z}$	M_2 $X + \bar{Y} + Z$
3	011	m_3 $\bar{X}YZ$	M_3 $X + \bar{Y} + \bar{Z}$
4	100	m_4 $X\bar{Y}\bar{Z}$	M_4 $\bar{X} + Y + Z$
5	101	m_5 $X\bar{Y}Z$	M_5 $\bar{X} + Y + \bar{Z}$
6	110	m_6 $XY\bar{Z}$	M_6 $\bar{X} + \bar{Y} + Z$
7	111	m_7 XYZ	M_7 $\bar{X} + \bar{Y} + \bar{Z}$

- **The *index* above is important for describing which variables in the terms are true and which are complemented**

Standard Order

- Minterms and maxterms are designated with a subscript
- The subscript is a number, corresponding to a binary pattern
- The bits in the pattern represent the complemented or normal state of each variable listed in a standard order
- All variables will be present in a minterm or maxterm and will be listed in the ***same order (usually alphabetically)***
- **Example: For variables a, b, c:**
  - **Maxterms:** $(a + b + \bar{c})$, $(a + b + c)$
  - **Terms:** $(b + a + c)$, $a\bar{c}b$, **and** $(c + b + a)$ **are NOT in standard order.**
  - **Minterms:** $a\bar{b}c$, abc, $\bar{a}\bar{b}c$
  - **Terms:** $(a + c)$, $\bar{b}c$, **and** $(\bar{a} + b)$ **do not contain all variables**

Purpose of the Index

- The ***index*** for the minterm or maxterm, expressed as a binary number, is used to determine whether the variable is shown in the true form or complemented form

Min
0 → com
1 → not com

- **For Minterms:** → لازم يطلع بالجواب (1)
  - **"0" means the variable is "Complemented"**
  - **"1" means the variable is "Not Complemented"**

Max
0 → not com
1 → com

- **For Maxterms:** → لازم يطلع بالجواب (0)
  - **"0" means the variable is "Not Complemented"**
  - **"1" means the variable is "Complemented"**

Index Example: Three Variables

Index (Decimal)	Index (Binary) n = 3 Variables	Minterm (m)	Maxterm (M)
0	000	$m_0 = \bar{X}\bar{Y}\bar{Z}$	$M_0 = X + Y + Z$
1	001	$m_1 = \bar{X}\bar{Y}Z$	$M_1 = X + Y + \bar{Z}$
2	010	$m_2 = \bar{X}Y\bar{Z}$	$M_2 = X + \bar{Y} + Z$
3	011	$m_3 = \bar{X}YZ$	$M_3 = X + \bar{Y} + \bar{Z}$
4	100	$m_4 = X\bar{Y}\bar{Z}$	$M_4 = \bar{X} + Y + Z$
5	101	$m_5 = X\bar{Y}Z$	$M_5 = \bar{X} + Y + \bar{Z}$
6	110	$m_6 = XY\bar{Z}$	$M_6 = \bar{X} + \bar{Y} + Z$
7	111	$m_7 = XYZ$	$M_7 = \bar{X} + \bar{Y} + \bar{Z}$

Index Example: Four Variables

i (Decimal)	i (Binary) n = 4 Variables	m_i	M_i
0	0000	$\bar{a}\bar{b}\bar{c}\bar{d}$	$a+b+c+d$
1	0001	$\bar{a}\bar{b}\bar{c}d$	$a+b+c+\bar{d}$
3	0011	$\bar{a}\bar{b}cd$	$a+b+\bar{c}+\bar{d}$
5	0101	$\bar{a}b\bar{c}d$	$a+\bar{b}+c+\bar{d}$
7	0111	$\bar{a}bcd$	$a+\bar{b}+\bar{c}+\bar{d}$
10	1010	$a\bar{b}c\bar{d}$	$\bar{a}+b+\bar{c}+d$
13	1101	$ab\bar{c}d$	$\bar{a}+\bar{b}+c+\bar{d}$
15	1111	$abcd$	$\bar{a}+\bar{b}+\bar{c}+\bar{d}$

یکا فی ک بعض ← $a+b+\bar{c}+\bar{b}d \stackrel{?}{=} M_0$?

0 0 1 1 M_3

Minterm and Maxterm Relationship

- Review: DeMorgan's Theorem
  - $\overline{x.y} = \bar{x} + \bar{y}$ and $\overline{x + y} = \bar{x}.\bar{y}$
- Two-variable example:
  - $M_2 = \bar{x} + y$ and $m_2 = x.\bar{y}$
  - Using DeMorgan's Theorem → $\overline{\bar{x} + y} = \bar{\bar{x}}.\bar{y} = x.\bar{y}$
  - Using DeMorgan's Theorem → $\overline{x.\bar{y}} = \bar{x} + \bar{\bar{y}} = \bar{x}.y$
  - Thus, M_2 is the complement of m_2 and vice-versa
- Since DeMorgan's Theorem holds for n variables, the above holds for terms of n variables:

$$M_i = \overline{m_i} \text{ and } m_i = \overline{M_i}$$

- Thus, M_i is the complement of m_i and vice-versa

Observations

- In the function tables:
  - Each ***minterm*** has one and only one 1 present in the 2^n terms (a minimum of 1s). All other entries are 0.
  - Each ***maxterm*** has one and only one 0 present in the 2^n terms All other entries are 1 (a maximum of 1s).
- We can implement any function by
  - "ORing" the minterms corresponding to "1" entries in the function table. These are called the minterms of the function.
  - "ANDing" the maxterms corresponding to "0" entries in the function table. These are called the maxterms of the function.
- This gives us two canonical forms for stating any Boolean function:
  - ***Sum of Minterms (SOM)***
  - ***Product of Maxterms (POM)***

Minterm Function Example

- **Example: Find $F_1 = m_1 + m_4 + m_7$**
- **$F_1 = x'y'z + xy'z' + xyz$**

16 8 4 2 1

xyz	Index	$m_1 + m_4 + m_7 = F_1$
000	0	0 + 0 + 0 = 0
001	1	1 + 0 + 0 = 1
010	2	0 + 0 + 0 = 0
011	3	0 + 0 + 0 = 0
100	4	0 + 1 + 0 = 1
101	5	0 + 0 + 0 = 0
110	6	0 + 0 + 0 = 0
111	7	0 + 0 + 1 = 1

m_1 001 $\bar{x}\bar{y}z$

m_4 100 $x\bar{y}\bar{z}$

m_7 111 xyz

or

$\bar{x}\bar{y}z + x\bar{y}\bar{z} + xyz$

Minterm Function Example

- $F(A, B, C, D, E) = m_2 + m_9 + m_{17} + m_{23}$

- $F(A, B, C, D, E) = A'B'C'DE' + A'BC'D'E + AB'C'D'E + AB'CDE$

كل حد ÷ ⊥

① m_2 0 0 0 1 0 — $\bar{A}\bar{B}\bar{C}D\bar{E}$

② m_9 0 1 0 0 1 — $\bar{A}B\bar{C}\bar{D}E$

③ m_{17} 1 0 0 0 1 — $A\bar{B}\bar{C}\bar{D}E$

④ m_{23} 1 0 1 1 1 — $A\bar{B}CDE$

OR بعد نعم

Maxterm Function Example

- **Example: Implement F1 in maxterms:**
- $F_1 = M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6$ and
- $F_1 = (x + y + z) \,.\, (x + y' + z) \,.\, (x + y' + z') \,.\, (x' + y + z') \,.\, (x' + y' + z)$

xyz	Index	$M_0 \,.\, M_2 \,.\, M_3 \,.\, M_5 \,.\, M_6 = F_1$
000	0	0 . 1 . 1 . 1 . 1 = 0
001	1	1 . 1 . 1 . 1 . 1 = 1
010	2	1 . 0 . 1 . 1 . 1 = 0
011	3	1 . 1 . 0 . 1 . 1 = 0
100	4	1 . 1 . 1 . 1 . 1 = 1
101	5	1 . 1 . 1 . 0 . 1 = 0
110	6	1 . 1 . 1 . 1 . 0 = 0
111	7	1 . 1 . 1 . 1 . 1 = 1

16 8 4 2 1

Maxterm Function Example

- $F(A, B, C, D) = M_3 \cdot M_8 \cdot M_{11} \cdot M_{14}$

- $F(A, B, C, D)$
  $= (A + B + C' + D') \cdot (A' + B + C + D) \cdot$
  $(A' + B + C' + D') \cdot (A' + B' + C' + D)$

M_3 $A+B+\bar{C}+\bar{D}$

0 0 1 1

M_8 $\bar{A}+B+C+D$

0 0 0

M_{11} $\bar{A}+B+\bar{C}+\bar{D}$

1 0 1 1

M_{14} $\bar{A}+\bar{B}+\bar{C}+D$

and

anonical Sum of Minterms

Any Boolean function can be expressed as a Sum of Minterms (SOM):

- For the function table, the minterms used are the terms corresponding to the 1's
- For expressions, expand all terms first to explicitly list all minterms. Do this by "ANDing" any term missing a variable v with a term $(v + \bar{v})$

- Example: Implement $f = x + \bar{x}\bar{y}$ as a SOM?
  1. Expand terms → $f = x(y + \bar{y}) + \bar{x}\bar{y}$
  2. Distributive law → $f = xy + x\bar{y} + \bar{x}\bar{y}$
  3. Express as SOM → $f = m_3 + m_2 + m_0 = m_0 + m_2 + m_3$

Another SOM Example

- Example: $F = A + \bar{B}C$ ⟶ $A(B+\bar{B})(C+\bar{C}) + \bar{B}C(A+\bar{A})$
  AB ناقص ↓ A ناقص ↓
- There are three variables: A, B, and C which we take to be the standard order
- Expanding the terms with missing variables:
  - $F = A(B+\bar{B})(C+\bar{C}) + (A+\bar{A})\bar{B}C$
    $(A\bar{B} + A\bar{B})(C+\bar{C}) +$
- Distributive law:
  - $F = ABC + A\bar{B}C + AB\bar{C} + A\bar{B}\bar{C} + A\bar{B}C + \bar{A}\bar{B}C$

* مهم مهم انتبه اذا ما كانو مرتبين حسب F(a,b,c) Function كيف ترتيبهم ثم حساب m

- Collect terms (removing all but one of duplicate terms):
  - $F = ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + \bar{A}\bar{B}C$
- Express as SOM:
  - $F = m_7 + m_6 + m_5 + m_4 + m_1$
  - $F = m_7 + m_6 + m_5 + m_4 + m_1$

Shorthand SOM Form

- From the previous example, we started with:
  - $F = A + \bar{B}C$
- We ended up with:
  - $F = m_1 + m_4 + m_5 + m_6 + m_7$
- This can be denoted in the *formal shorthand*:
  - $F(A, B, C) = \sum_m(1,4,5,6,7)$
- Note that we explicitly show the standard variables in order and drop the "m" designators.

Canonical Product of Maxterms

- Any Boolean Function can be expressed as a Product of Maxterms (POM):
  - For the function table, the maxterms used are the terms corresponding to the 0's
  - For an expression, expand all terms first to explicitly list all maxterms. Do this by first applying the second distributive law , "ORing" terms missing variable v with $(v \,.\, \bar{v})$ and then applying the distributive law again
- Example: Convert $f(x, y, z) = x + \bar{x}\bar{y}$ to POM?
  - Distributive law → $f = (x + \bar{x}) \,.\, (x + \bar{y}) = x + \bar{y}$
  - ORing with missing variable (z) → $f = x + \bar{y} + z \,.\, \bar{z}$
  - Distributive law → $f = (x + \bar{y} + z) \,.\, (x + \bar{y} + \bar{z})$
  - Express as POS → $f = M_2 \,.\, M_3$

Another POM Example

- Convert $f(A,B,C) = AC' + BC + A'B'$ to POM?
- Use $x + yz = (x+y).(x+z)$, assuming $x = AC' + BC$ and $y = A'$ and $z = B'$
  - $f(A,B,C) = (AC' + BC + A').(AC' + BC + B')$
- Use Simplification theorem to get:
  - $f(A,B,C) = (BC + A' + C').(AC' + B' + C)$
- Use Simplification theorem again to get:
  - $f(A,B,C) = (A' + B + C').(A + B' + C) = M_5 . M_2$
  - $f(A,B,C) = M_2 . M_5 = \prod_M(2,5)$ → ***Shorthand POM form***

Function Complements

- The complement of a function expressed as a sum of minterms is constructed by selecting the minterms missing in the sum-of-minterms canonical forms.
- Alternatively, the complement of a function expressed by a sum of minterms form is simply the Product of Maxterms with the same indices.
- Example: Given $F(x, y, z) = \sum_m(1,3,5,7)$, find complement F as SOM and POM?
  - $\overline{F}(x, y, z) = \sum_m(0,2,4,6)$
  - $\overline{F}(x, y, z) = \prod_M(1,3,5,7)$

Conversion Between Forms

- To convert between sum-of-minterms and product-of-maxterms form (or vice-versa) we follow these steps:
  - Find the function complement by swapping terms in the list with terms not in the list.
  - Change from products to sums, or vice versa.

- **Example:** Given F as before: $F(x, y, z) = \sum_m(1,3,5,7)$
  - Form the Complement: $\bar{F}$

    $\bar{F}(x, y, z) = \sum_m(0,2,4,6)$
  - Then use the other form with the same indices – this forms complement again, giving the other form of the original function:

    $F(x, y, z) = \prod_M(0,2,4,6)$

$\bar{F}(x,y,z) = \prod (1,3,5,7)$

Standard Forms

SOP	POS
or +	and ·
and ·	or +
Varb	حافیظ کل Varb

- Standard Sum-of-Products (SOP) form: equations are written as an OR of AND terms
- Standard Product-of-Sums (POS) form: equations are written as an AND of OR terms
- **Examples:**
  - SOP: $\cancel{ABC} + \bar{A}\bar{B}C + B$
  - POS: $(A + B) \cdot (A + \bar{B} + \bar{C}) \cdot C$
- **These "mixed" forms are neither SOP nor POS**
  - $(AB + C)(A + C)$ (AC + AB)
  - $AB\bar{C} + AC(A + B)$

في حالة صفرة ⇒ sop / pos

* $(A\overline{B+C})\ (A+C)$

$(A+C)(\overline{B}+C)\ (A+C)$ pos

* $AB\bar{C} + AC\ (A+B)$

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

Standard Sum-of-Products (SOP)

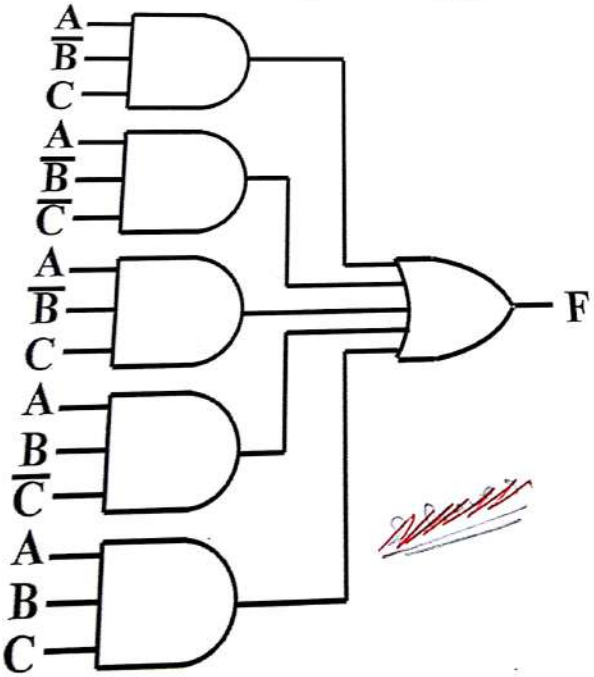
- A Simplification Example: $F(A,B,C) = \sum_m(1,4,5,6,7)$
- Writing the minterm expression:
  - $F(A,B,C) = A'B'C + AB'C' + AB'C + ABC' + ABC$
- Simplifying using boolean Algebra:

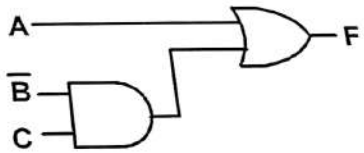
$A'B'C + AB'C' + AB'C + ABC' + ABC$	
$= A'B'C + AB'(C' + C) + AB(C' + C)$	*Distributive law*
$= A'B'C + AB' + AB$	$X + X' = 1$
$= A'B'C + A(B' + B)$	*Distributive law*
$= A'B'C + A$	*Simplification Theorem*
$= A + B'C$	

- Simplified F contains 3 literals compared to 15 in minterm F

AND/OR Two-level Implementation of SOP Expression

- **The two implementations for F are shown below – it is quite apparent which is simpler!**





2 level

$A_{n-1} \cdots A_2 A_1 A_0 , A_{-1} A_{-2} A_{-3} \cdots A_{-m}$

the most significant digit [MSD]

(يمين) العدد يبدأ منا (يسار)

العدد يبدأ منا (يمين)

the Least significant digit (LSD)

(coefficient موقع) i ← هم اشتبه

مثلاً 0، 1، 2، ...

-1، -2، ...

weight by which coefficient is Multiplied (يتضرب فيها)

r^i ← (radix Base)

	General (شكل عام)	Decimal	Binary ← Number system
Base radix	r	10	2
Digits	$r \to r-1$	$0 \Rightarrow 9$	$0 \Rightarrow 1$
$i=0$	r^0	$10^0 = 1$	$2^0 = 1$
1	r^1	$10^1 = 10$	$2^1 = 2$
2	r^2	$10^2 = 100$	$2^2 = 4$
-1	r^{-1}	$10^{-1} = 0.1$	$2^{-1} = 0.5$
-2	r^{-2}	$10^{-2} = 0.01$	$2^{-2} = 0.25$

موضوع الدرس _____ اليوم _____ التاريخ / /

22

$2^0 = 1$ $2^3 = 8$

$2^1 = 2$ $2^4 = 16$ ← لتذكير

$2^2 = 4$ $2^5 = 32$

Useful for Base conversion

23

	2^{10}	2^{20}	2^{30}	2^{40}
الاسم	Kilo	Mega	Giga	Tera
الرمز	K	M	G	T
	1024	1,048,576	1,073,7...	1,099,511...

Special powers of 2

24

Name	Radix(r)	Digits
Binary	2	(0, 1)
Octal	8	(0, 1, 2, 3, 4, 5, 6, 7)
Decimal	10	(0 -9)
Hexadecimal	16	0-9, A, B, C, D, E, F

A=10 B=11 C=12

D=13 E=14 F=15

يمكن استخدام a, b, c, d, e, f

Commonly Occuring Bases

موضوع الدرس _____ اليوم _____ التاريخ / /

25 * Binary System

r	Digits	Notes
2	0, 1	bit ← binary digit

Ex :- (10011, 101)

4 3 2 1 0 -1 -2 -3

$1\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 , 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$

$16 + 0 + 0 + 2 + 1 , 5 + 0 + 0.125$

$19 , 625$

27 # Octal system

r	Digits	Notes
8	0-7	3bits ← كل digits لكل More compact than Binary

Ex $(124.4)_8$

2 1 0 -1

$8^2 1 + 8^1 2 + 8^0 4 . 4 \; 8^{-1}$

$64 + 16 + 4 . \frac{1}{2}$

128.5

موضوع الدرس _____ اليوم _____ التاريخ / /

29 Hexadecimal System

r	Digits	Notes
16	0-9 A B C D E F 10 11 12 13 14 15	4 bits ← كل digits

* Ex $(B65F)_{16}$ F = 15, B = 11

3 2 1 0

$16^3 11 + 16^2 6 + 16^1 5 + 16^0 15$

$45056 + 1536 + 80 + 15$

46687

35

تحويل الرقم العشري

Decimal

Bin r=2 ✓ Oct r=8 Hex r=16

* يكون بالقسمة على r

Ex $(46)_{10} \longrightarrow (10110)_2$

46 | 2	الناتج	الباقي	
46 | 2	23	0	LSD
23 | 2	11	1	
11 2	5	1	
5 2	2	1	
2 2	1	0	
1 2	0	1	MSD

يكتب هنا تحت الخط

الناتج نضل نمشي بوقف

$(10110)_2$

موضوع الدرس ____ اليوم ____ التاريخ / /

35

تحويل الجزء العشري

Decimal

↙ Bin = 2 ↓ Oct = 8 ↘ Hex = 16

* يكون بضرب العدد المراد تحويله × ٢

ex

$(0.6875)_{10} = (0.1011)_2$

		فاصلة
0.6875 * 2		
0.6875 * 2	MSD 1	375
0.375 * 2	0	75
0.75 * 2	1	5
	LSD 1	0

يكتب من مقدار الخانة

1011

ما يوصل صفر يوقف

موضوع الدرس _____ اليوم _____ التاريخ / /

36 $(153.513)_{10} \longrightarrow (231.407)_8$

		الناتج	الباقي
153	8	19	125
153	8	2	3
19	8	0	2
2	8		

$(153)_{10} \longrightarrow (231)_8$

.513 * 8	___ .	___
.513 * 8	4 .	104
.104 * 8	0 .	832
.832 * 8	6 .	656
.656 * 8	5 .	248

$(0.513)_{10} = (.406)_8$

⇓ Round

$(0.513)_{10} = (.407)_8$

up to 3 digits

* عملية تحويل اللون الى الرمز (رمز)
Coding

* مثلاً أيام الاسبوع ← ست ١
الاحد 2

* عشان أعمل coding لازم اعرف عدد digits عدد الخانات

Decimal ___ $10^2 = 100$ (٢)
٢ خانة

octal ___ $8^2 = 64$
٢ خانة

Binary ___ $2^2 = 4$
٢ خانة

عدد الرموز الى انت بدك تمثلها ← عدد الخيارات الى موجود عندك

49 Oct

مثلاً عدد ايام السنة بدي اعرفها بـ

ex $8^n \geq 365$

$n = 1$

$n = 2 \quad 64$

$n = \quad 512$ ✓ بزيادة ✓ $512 > 365$

طريقة كمان ما أحلى

اعرف $\log_r r^n = \log_r m$

$$n = \log_r m$$

مثلاً عدد ايام السنة $n = \log_8 365$

$$n = \lceil \log_r m \rceil$$

دائماً بقرب

معهم ↓

$\lceil 3.00001 \rceil$ $\lceil 3.9999 \rceil$

4 4

بدي احمل Coding لـ أرقام
في decimal [لأنه بيكون بـ decimal]
موضوع الدرس
(BCD)

	8 4 2 1	Excess 3	8, 4, -2, -1
0	0 0 0 0	0 0 1 1	0 0 0 0
1	0 0 0 1	0 1 1 1	0 1 1 1
2	0 0 1 0	0 1 1 0	0 1 1 0
3	0 0 1 1	0 1 0 1	0 1 0 1

↑
اكبر رقم ٩
(0-9)

Gray

0 0 0 0 في فرق واحد
0 0 0 1
0 0 1 1 في فرق واحد

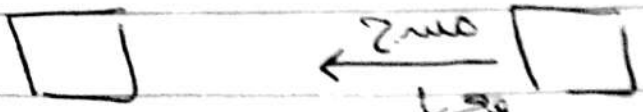
* مو داخل لكيف امثل
* المهم انه في فرق بين اللي قبله واللي بعده

* انتبه ✓
$(8)_{10}$ ⟶ $(0111)_{(2)}$
مثلاً
هون بحول عادي

$(8)_{10}$ ⟶ ( )
BCD
↑
هون ما في 2
بعمل كودing من عنده F=2

Excess (3)
① باخذ الرقم بـ Decimal
② بزيد عليه 3
③ بحولـه لـ Binary

60



متفقتين على
وصف ← code معين

* PARITY

(عدد) البت 1 الموجود في المسج؟	parity عبارة عن bit واحد
odd: 0111	even: 1111

Overview

- Part 1 – Gate Circuits and Boolean Equations
  - Binary Logic and Gates
  - Boolean Algebra
  - Standard Forms
- Part 2 – Circuit Optimization
  - Two-Level Optimization
  - Map Manipulation
- Part 3 – Additional Gates and Circuits
  - Other Gate Types
  - Exclusive-OR Operator and Gates
  - High-Impedance Outputs

Sosa

Circuit Optimization

- *Goal:* To obtain the simplest *implementation* for a given function
- *Optimization* is a more formal approach to *simplification* that is performed using a *specific procedure or algorithm*
- *Optimization* requires a cost criterion to measure the simplicity of a circuit
- *Distinct* cost criteria we will use: cost انواع

1. **Literal cost (L)**
2. **Gate input cost (G)**
3. **Gate input cost with NOTs (GN)**

Literal Cost (L)

- ***Literal:*** a variable or its complement
- ***Literal cost (L):*** the number of literal appearances in a Boolean expression corresponding to the logic circuit diagram
- Examples:
  - $F = BD + AB'C + AC'D'$ → 8
    - $L = 8$ (Minimum cost → Best solution)
  - $F = BD + AB'C + AB'D' + ABC'$
    - $L = 11$
  - $F = (A + B)(A + D)(B + C + D')(B' + C' + D)$
    - $L = 10$

Gate Input Cost (G)

- ***Gate input cost (G):*** the number of inputs to the gates in the implementation corresponding exactly to the given equation or equations. (***G: inverters not counted, GN: inverters counted***)
- For SOP and POS equations, it can be found from the equation(s) by finding the sum of:
  - **All literal appearances**
  - **The number of terms excluding single literal terms,(G) and**
  - **optionally, the number of distinct complemented single literals (GN).**
- Examples:
  - $F = BD + AB'C + AC'D'$
    - $G = 11, GN = 14$ (Minimum cost → Best solution)
  - $F = BD + AB'C + AB'D' + ABC'$
    - $G = 15, GN = 18$
  - $F = (A + B)(A + D)(B + C + D')(B' + C' + D)$
    - $G = 14, GN = 17$

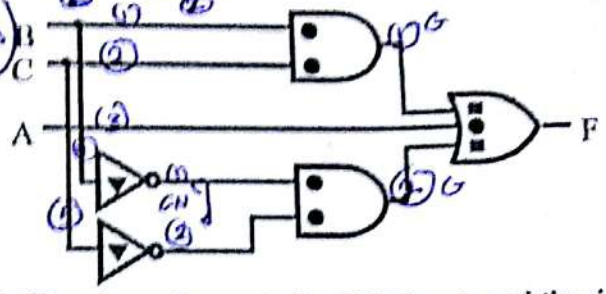
Cost Criteria (continued)

- **Example 1:**
- $F = A + BC + \overline{B}\,\overline{C}$

GN = G + 2 = 9

L = 5

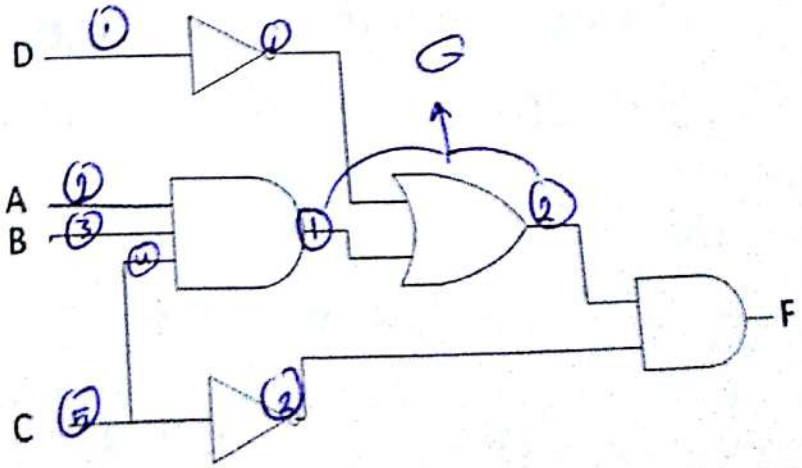
G = L + 2 = 7



- **L (literal count) counts the AND inputs and the single literal OR input.**
- **G (gate input count) adds the remaining OR gate inputs**
- **GN(gate input count with NOTs) adds the inverter inputs**

Cost Criteria (continued)

- **Example 2:**
- $F = (A, B, C, D) = (ABC + D').C'$
  - $L = 5$
  - $G = 5 + 2 = 7$
  - $GN = 7 + 2 = 9$



K-Map Function Representation

- Example: $F(x, y) = x$

$F(x,y) = x$	$y = 0$	$y = 1$
$x = 0$	0	0
$x = 1$	1	1

- For function $F(x, y)$, the two adjacent cells containing 1's can be combined using the Minimization Theorem:

$$F(x, y) = x\bar{y} + xy = x$$

K-Map Function Representation

- Example: $G(x, y) = x + y$

$G(x,y) = x + y$	$y = 0$	$y = 1$
$x = 0$	0	1
$x = 1$	1	1

- For $G(x, y)$, two pairs of adjacent cells containing 1's can be combined using the Minimization Theorem:

$$G(x, y) = (x\bar{y} + xy) + (\bar{x}y + xy)$$

$$G(x, y) = x + y$$

K-Map and Truth Tables

- The K-Map is just a different form of the truth table
- Example: Two variable function
  - We choose a,b,c and d from the set {0,1} to implement a particular function, $F(x, y)$

طريقة توزيع في K truth table

Input Values (x, y)	F(x, y)
0 0	a
0 1	b
1 0	c
1 1	d

Truth Table

	$\bar{y}$ y = 0	y = 1
$\bar{x}$ x = 0	a m_0	b m_1
x = 1	c m_2	d m_3

K-Map

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

x	y	F	
0	0	0	m_0 a
0	1	1	m_1 b
1	0	1	m_2 c
1	1	0	m_3 d

f	$\bar{y}$	y
$\bar{x}$	0 (a, m_0)	1 (b, m_1)
x	1 (c, m_2)	0 (d, m_3)

① برسم k

② نفرغ troth ب k

③ عدد 1 .

Three Variable Maps

- A three-variable K-map:

	yz = 00	yz = 01	yz = 11	yz = 10
x = 0	m_0	m_1	m_3	m_2
x = 1	m_4	m_5	m_7	m_6

- Where each minterm corresponds to the product terms:

	yz = 00	yz = 01	yz = 11	yz = 10
x = 0	$\bar{x}\bar{y}\bar{z}$	$\bar{x}\bar{y}z$	$\bar{x}yz$	$\bar{x}y\bar{z}$
x = 1	$x\bar{y}\bar{z}$	$x\bar{y}z$	xyz	$xy\bar{z}$

- *Note that if the binary value for an index differs in one bit position, the minterms are adjacent on the K-Map*

Alternative Map Labeling

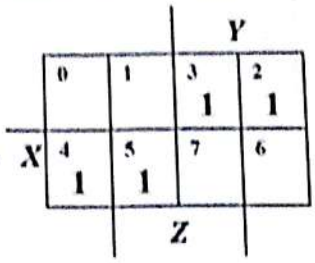
- Map use largely involves:
  - Entering values into the map, and
  - Reading off product terms from the map
- Alternate labelings are useful:

	$\bar{Y}$		Y	
$\bar{X}$	0	1	3	2
X	4	5	7	6
	$\bar{Z}$	Z		$\bar{Z}$

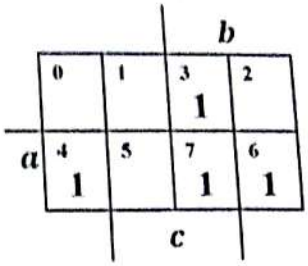
YZ / X	00	01	11	10
0	0	1	3	2
1	4	5	7	6

Y (11, 10); Z (01, 11)

- By convention, we represent the minterms of F by a "1" in the map and leave the minterms of $\bar{F}$ blank
- Example:
  - $F(x, y, z) = \sum_m(2,3,4,5)$



- Example:
  - $G(a, b, c) = \sum_m(3,4,6,7)$

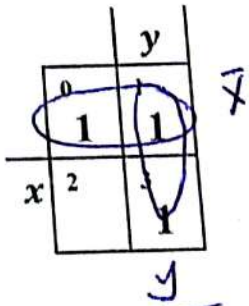


- Learn the locations of the 8 indices based on the variable order shown (X, most significant and Z, least significant) on the map boundaries

Steps for using K-Maps to Simplify Boolean Functions

- Enter the function on the K-Map
  - Function can be given in truth table, shorthand notation, SOP,…etc
  - Example:
    - $F(x, y) = \bar{x} + xy$
    - $F(x, y) = \sum_m(0,1,3)$

x	y	F(x,y)
0	0	1
0	1	1
1	0	0
1	1	1

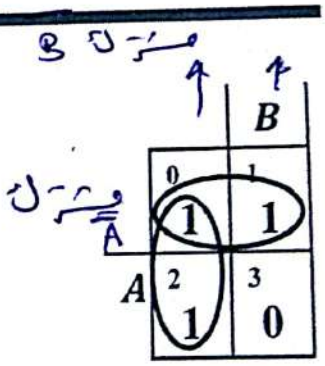


- Combining squares for simplification
  - Rectangles that include power of 2 squares {1, 2, 4, 8, …}
  - Goal: Fewest rectangles that cover all 1's → as large as possible
- Determine if any rectangle is not needed
- Read-off the SOP terms

Example: Combining Squares

- Example: $F(A,B) = \sum_m(0,1,2)$

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

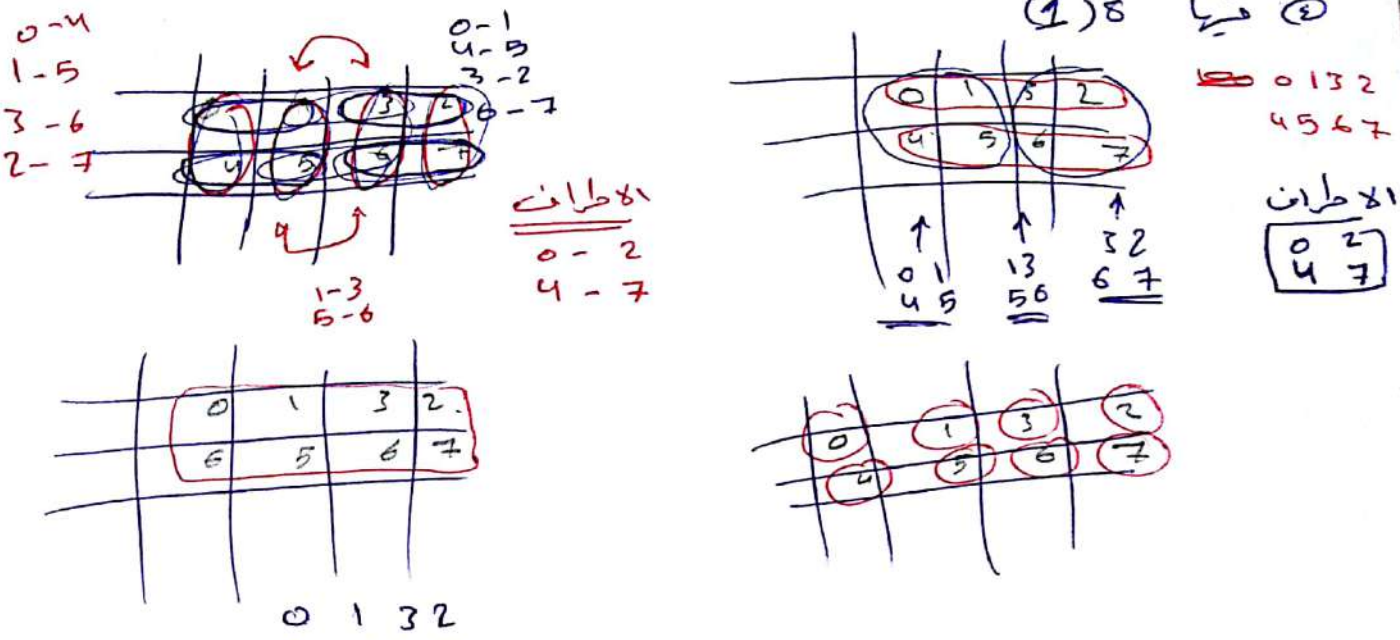


- Using Distributive law
  - $F(A,B) = \bar{A} + A\bar{B}$
- Using simplification theorem
  - $F(A,B) = \bar{A} + \bar{B}$

- **Thus, every two adjacent terms that form a 2×1 rectangle correspond to a product term with one variable**

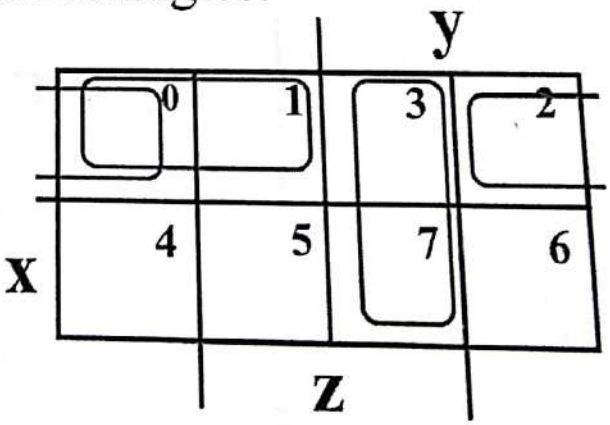
Three-Variable Maps

- Example shapes of 2-cell rectangles:



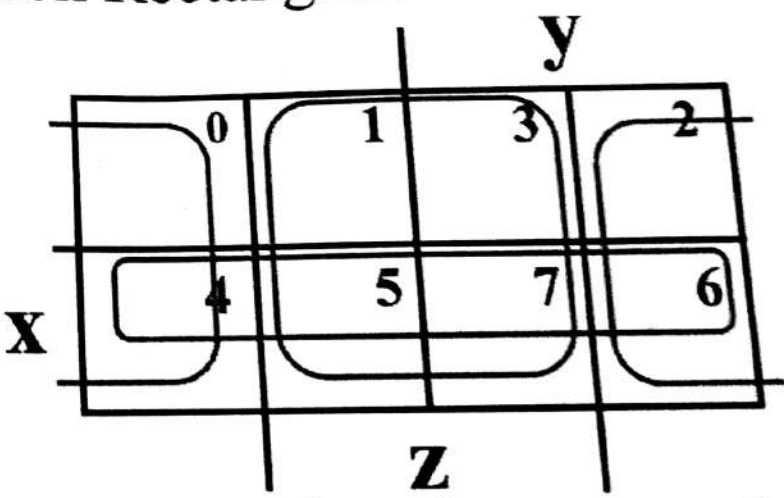
Three-Variable Maps

- Example shapes of 2-cell rectangles:



- Read-off the product terms for the rectangles shown:
  - $Rect(0,1) = \bar{X}\bar{Y}$
  - $Rect(0,2) = \bar{X}\bar{Z}$
  - $Rect(3,7) = YZ$

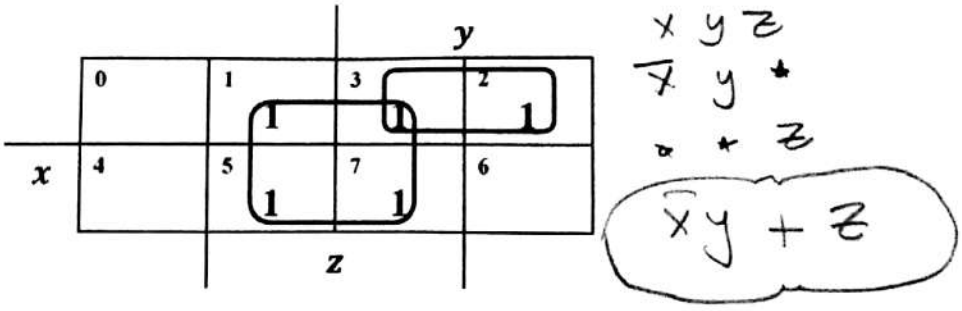
- Example shapes of 4-cell Rectangles:



- Read off the product terms for the rectangle shown:
  - $Rect(1,3,5,7) = Z$
  - $Rect(0,2,4,6) = \bar{Z}$
  - $Rect(4,5,6,7) = X$

x y z
* * z
* * $\bar{z}$
x * *

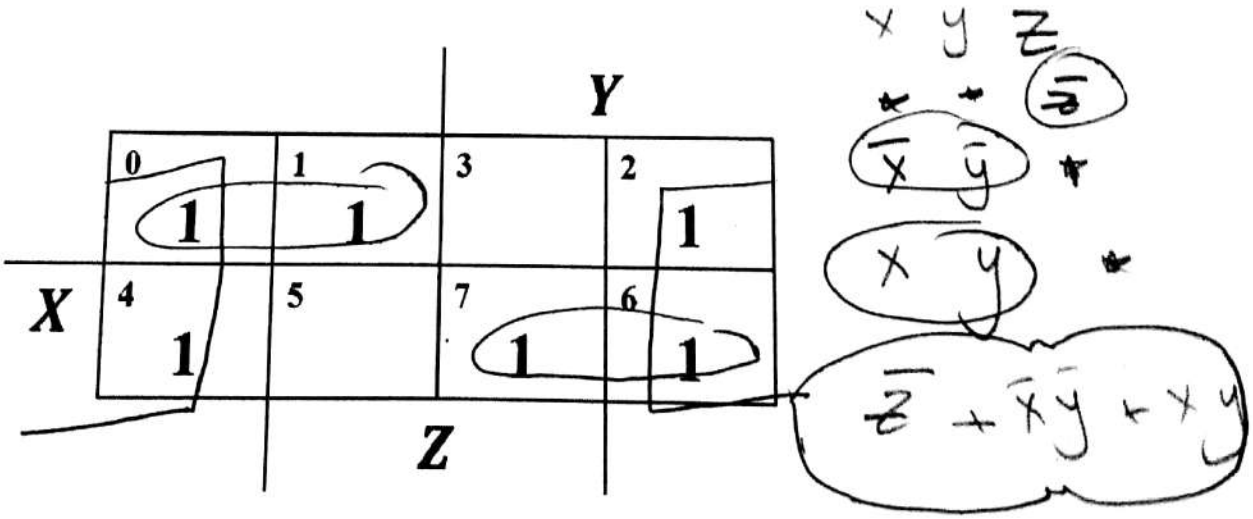
- K-maps can be used to simplify Boolean functions by systematic methods. Terms are selected to cover the "1s"in the map.
- Example: Simplify $F(x, y, z) = \sum_m(1,2,3,5,7)$



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

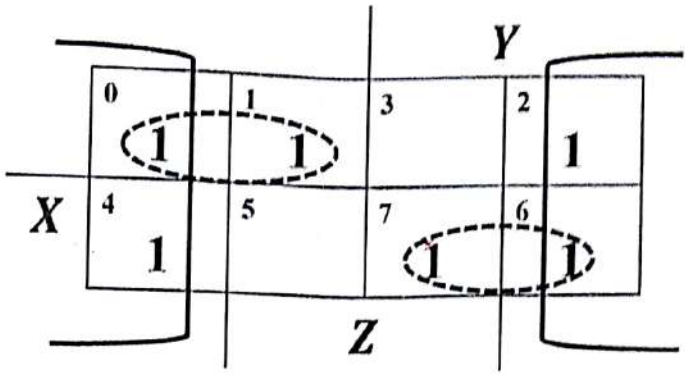
Three-Variable Map Simplification

- Use a K-map to find an optimum SOP equation for $F(X, Y, Z) = \sum_m(0,1,2,4,6,7)$



Three-Variable Map Simplification

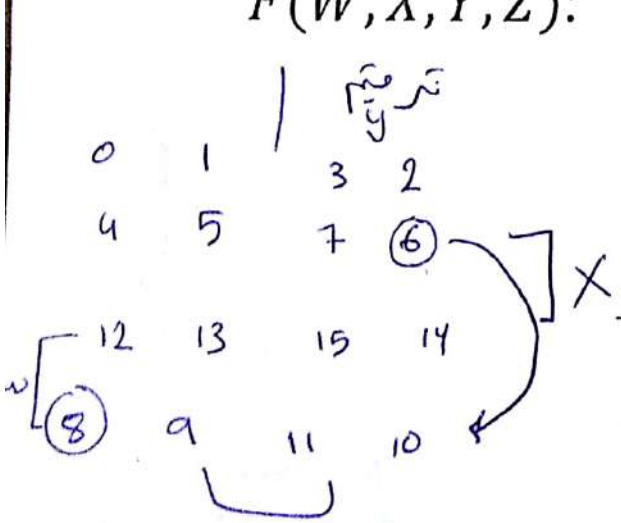
- Use a K-map to find an optimum SOP equation for $F(X,Y,Z) = \sum_m(0,1,2,4,6,7)$

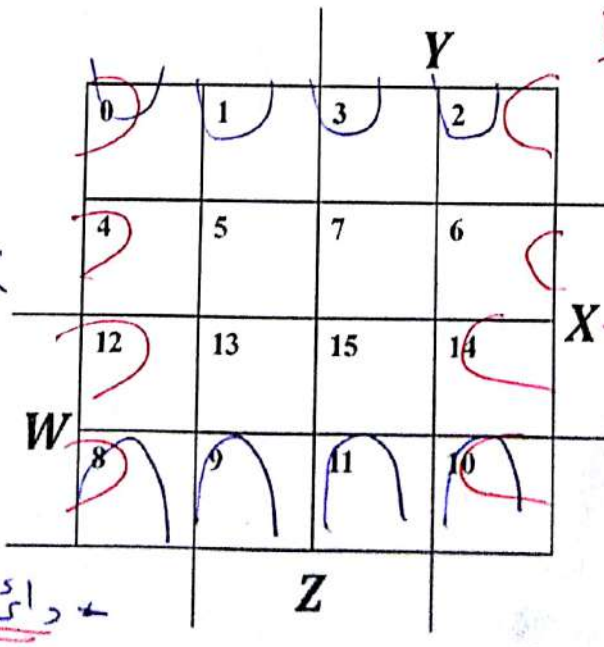


$$F(X,Y,Z) = \bar{Z} + \bar{X}\bar{Y} + XY$$

Four Variable Maps

- Map and location of minterms $F(W,X,Y,Z)$:



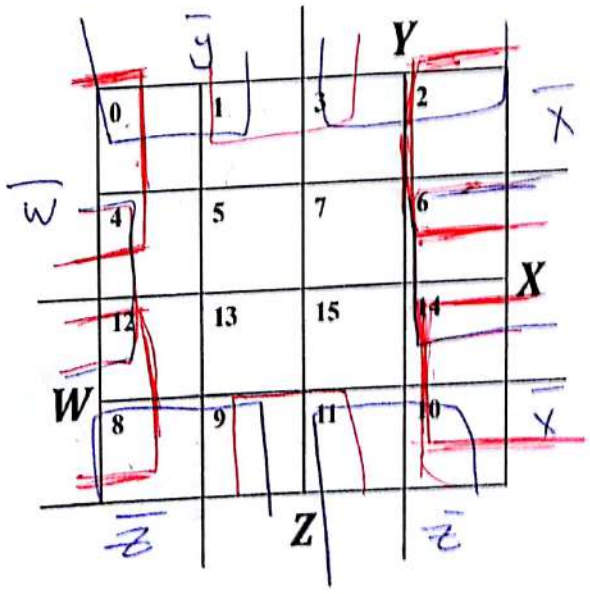


Four Variable Terms

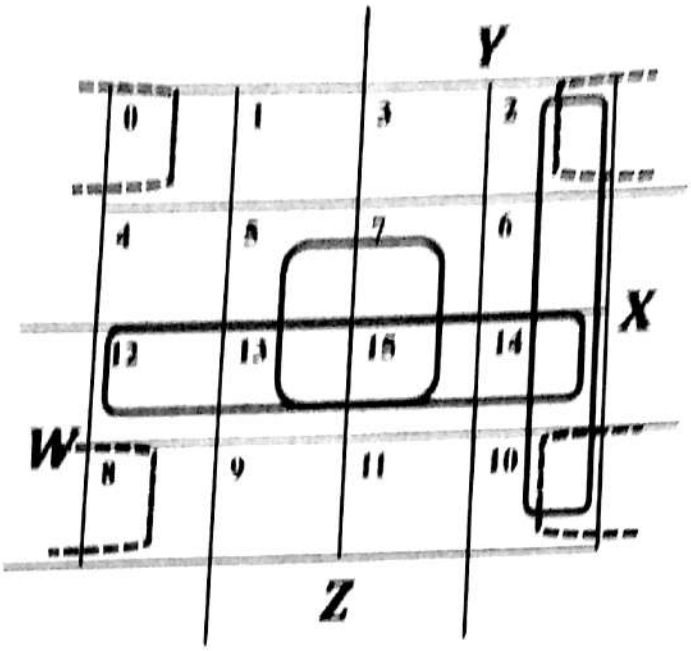
- Four variable maps can have rectangles corresponding to:
  - A single 1: 4 variables (i.e. Minterm)
  - Two 1's: 3 variables
  - Four 1's: 2 variables
  - Eight 1's: 1 variable
  - Sixteen 1's: zero variables (function of all ones)

Four-Variable Maps

- Example shapes of 4-cell rectangles:

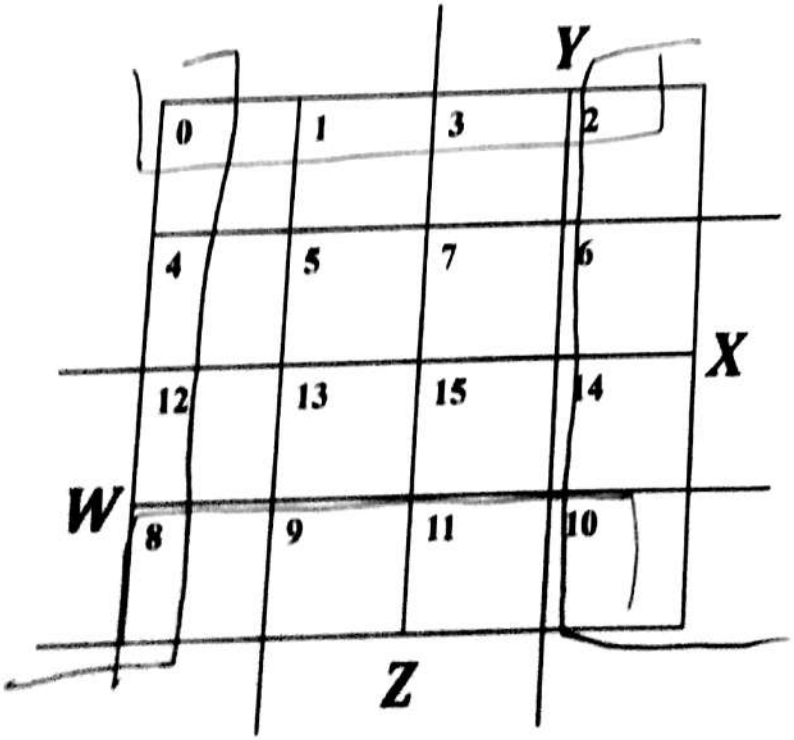


- Example shapes of 4-cell rectangles:



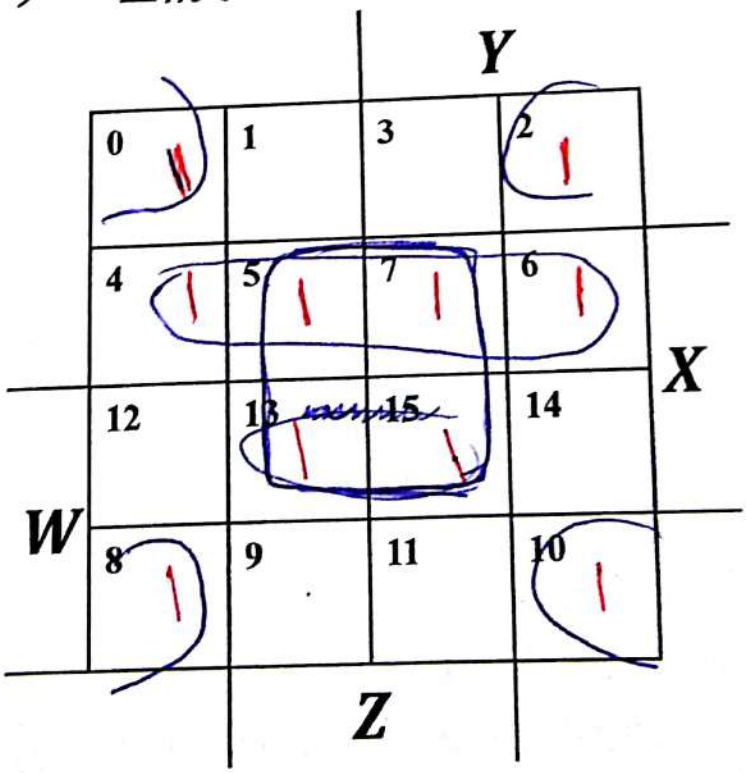
Four-Variable Maps

- Example shapes of 8-cell rectangles:



Four-Variable Map Simplification

- $F(W,X,Y,Z) = \sum_m(0,2,4,5,6,7,8,10,13,15)$



بدوّر على كل 1 وشوف
أكبر دائرة حولها

2 → 0 - 8

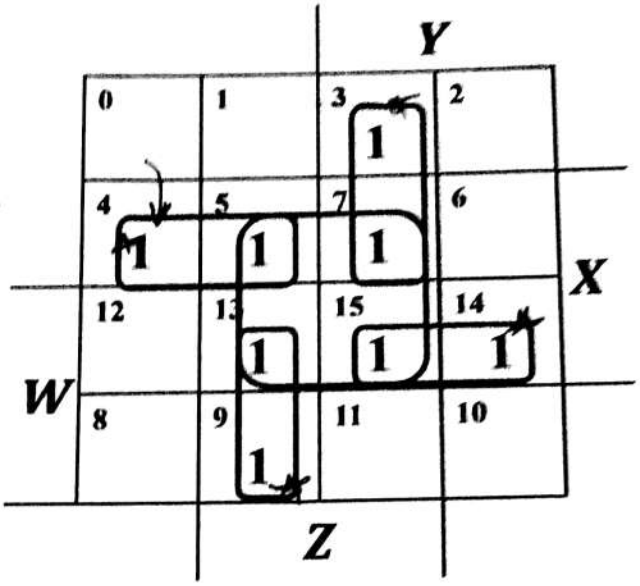
4 → 4 5 7 6

2 → 2 - 10

4 → 13 - 15 5 7

Four-Variable Map Simplification

- $F(W,X,Y,Z) = \sum_m(3,4,5,7,9,13,14,15)$



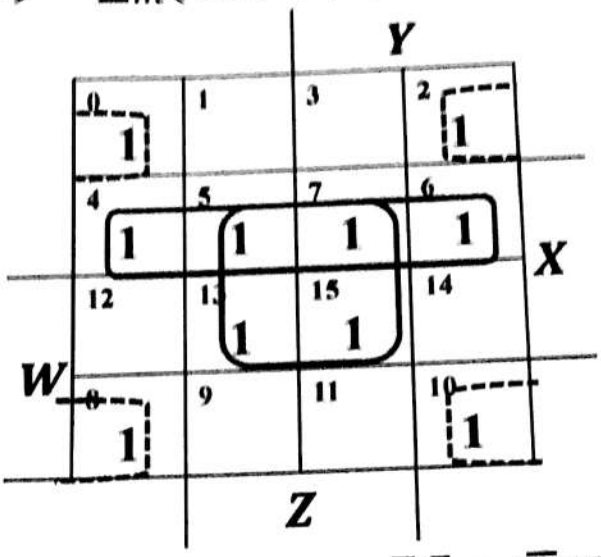
$$F(W,X,Y,Z) = \bar{W}YZ + \bar{W}X\bar{Y} + WXY + W\bar{Y}Z$$

Systematic Simplification

- ***Prime Implicant:*** is a product term obtained by combining the **maximum** possible number of adjacent squares in the map into a rectangle with the number of squares a power of 2
- A prime implicant is called an ***Essential Prime Implicant*** if it is the **only** prime implicant that covers (includes) one or more minterms
- Prime Implicants and Essential Prime Implicants can be determined by inspection of a K-Map
- A set of prime implicants ***"covers all minterms"*** if, for each minterm of the function, at least one prime implicant in the set of prime implicants includes the minterm

Four-Variable Map Simplification

- $F(W,X,Y,Z) = \sum_m(0,2,4,5,6,7,8,10,13,15)$

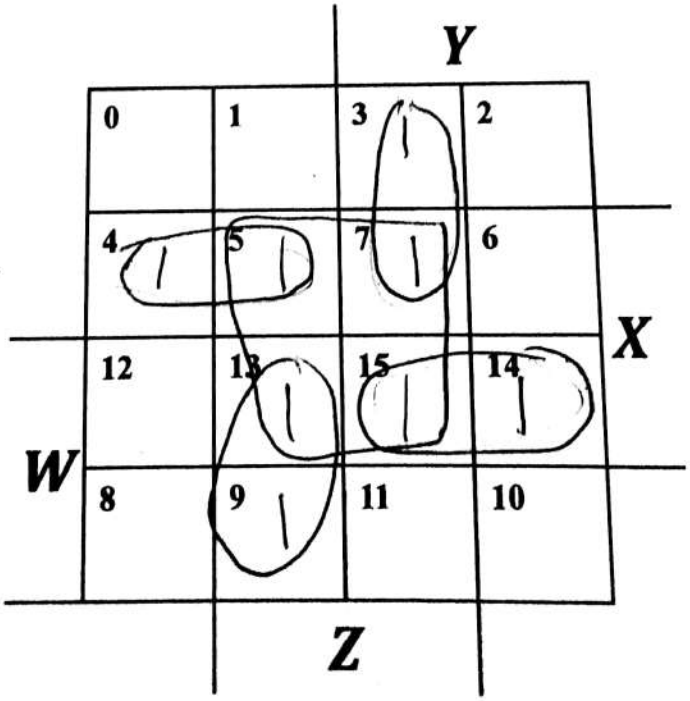


$$F(W,X,Y,Z) = XZ + \bar{X}\bar{Z} + \bar{W}X$$

Four-Variable Map Simplification

- $F(W,X,Y,Z) = \sum_m(3,4,5,7,9,13,14,15)$

لازم أمر على كل
واحد واشوف أكبر
دائرة حوله



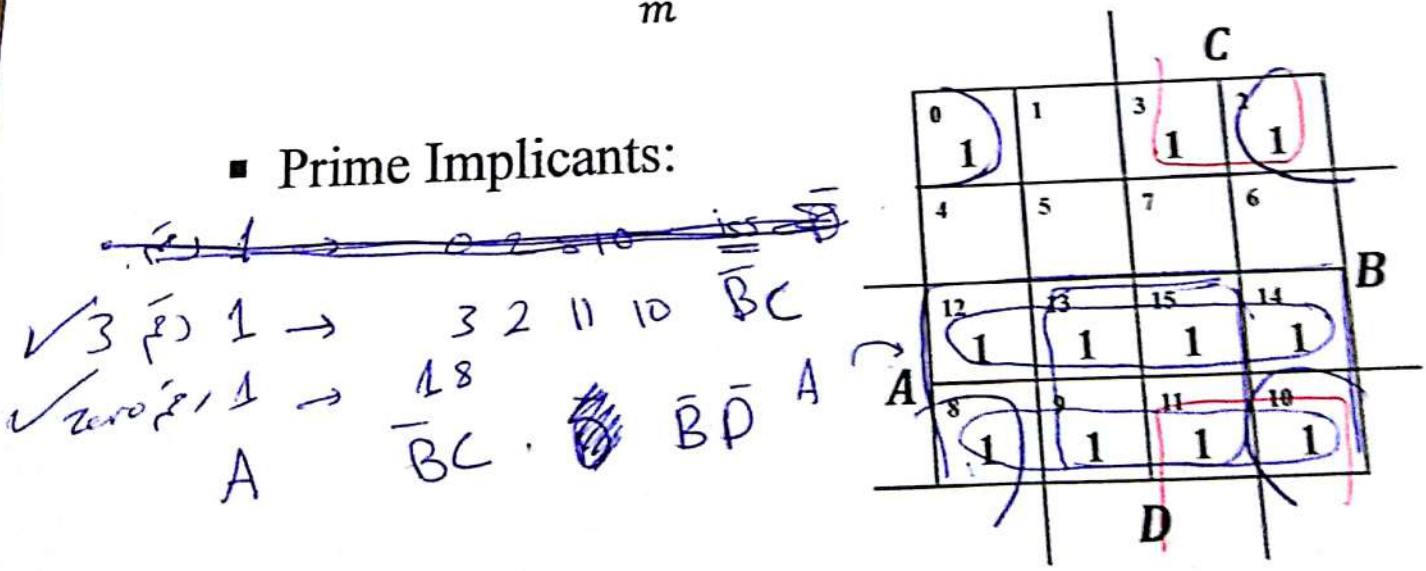
$$F(W,X,Y,Z) = \bar{W}YZ + \bar{W}X\bar{Y} + WXY + W\bar{Y}Z$$

Prime Implicant Practice

- Find all prime implicants for:

$$F(A,B,C,D) = \sum_{m}(0,2,3,8,9,10,11,12,13,14,15)$$

- Prime Implicants:



Another Example

- Find all prime implicants for:

$$G(A,B,C,D) = \sum_{m}(0,2,3,4,7,12,13,14,15)$$

- Hint: There are seven prime implicants!
- Prime Implicants:

0-2 $\bar{A}\bar{B}\bar{C}$

0-4 $\bar{A}\bar{B}\bar{D}$

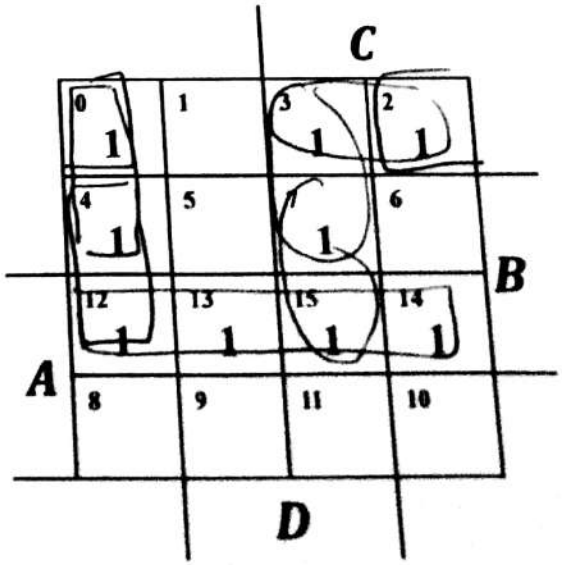
4-12 $B\bar{C}\bar{D}$

12 13 15 14 AB

2-3 $\bar{A}C\bar{B}$

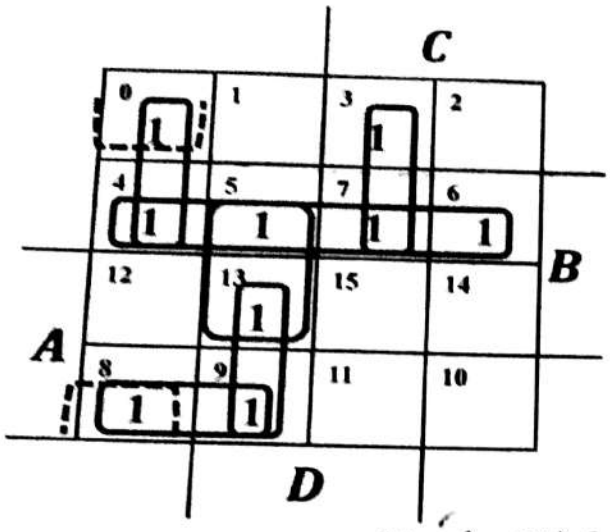
3-7 $\bar{A}CD$

15 CBD



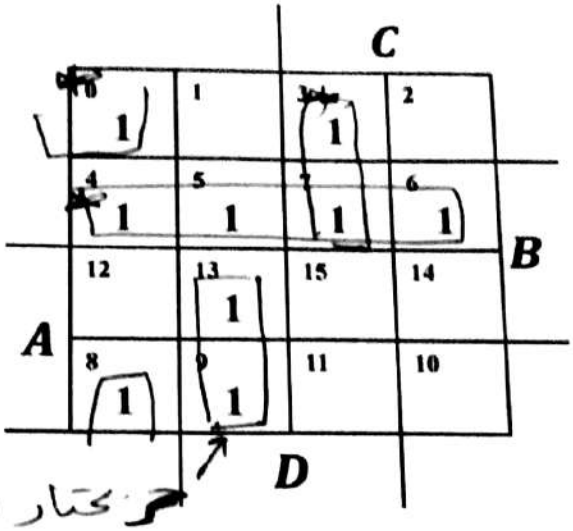
Selection Rule Example

- Simplify F(A, B, C, D) given on the K-map



Prime Implicants

Selected Non-essential Prime Implicants



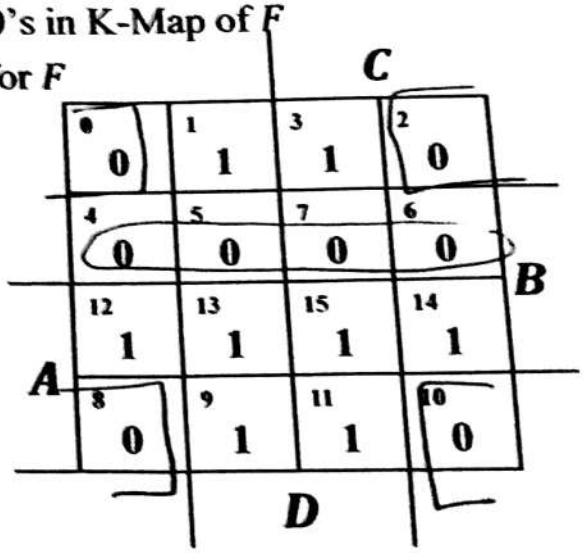
Essential and Selected Non-essential Prime Implicants

Selection Rule Example

- Find the optimum POS solution for:

$$F(A,B,C,D) = \sum_{m}(1,3,9,11,12,13,14,15)$$

- Solution:
  - Find optimized SOP for $\bar{F}$ by combining 0's in K-Map of F
  - Complement $\bar{F}$ to obtain optimized POS for F



SOP $\bar{F}$ يعني group zero يعني أكبر دائرة ممكن حول كل zero

0, 2, 8, 10 → $\bar{B}\bar{D}$

4, 5, 7, 6 → $\bar{A}B$

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

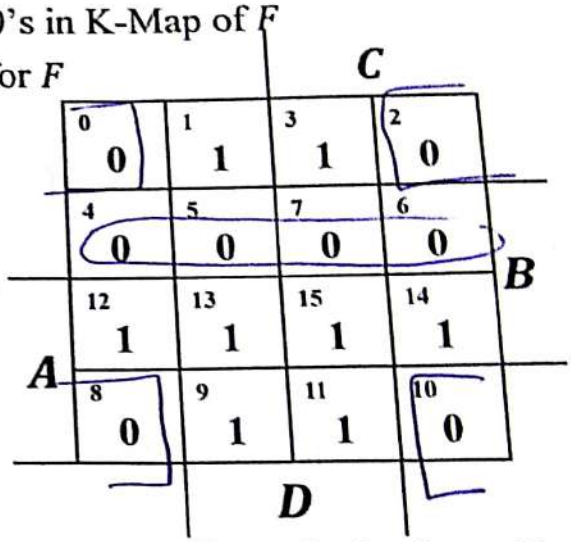
POS F → SOP $\bar{F}$ يعني Demorgan $(A+\bar{B}) \cdot (B+D)$

Product of Sums Example

- Find the optimum POS solution for:

$$F(A,B,C,D) = \sum_{m}(1,3,9,11,12,13,14,15)$$

- Solution:
  - Find optimized SOP for $\bar{F}$ by combining 0's in K-Map of F
  - Complement $\bar{F}$ to obtain optimized POS for F



sop $\bar{F}$ → بعمل group zero بعمل أكبر دائرة صول zero لـ

0, 2, 8, 10 → $\bar{B}\bar{D}$

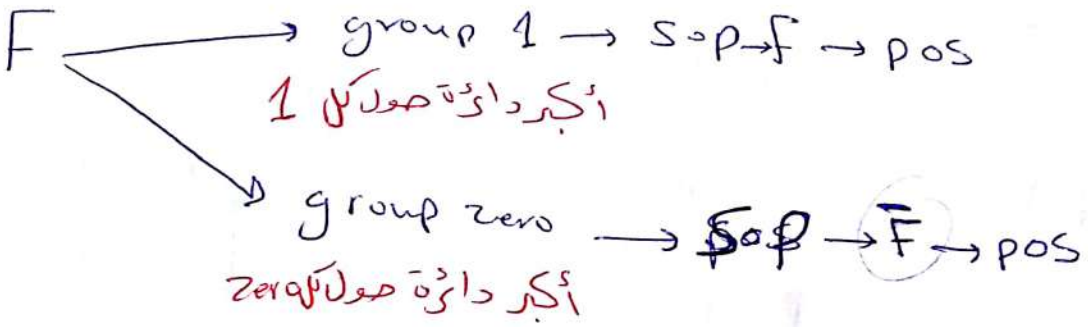
4, 5, 7, 6 → $\bar{A}B$

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

POS F → sop $\bar{F}$ → بعمل Demorgan $(A+\bar{B}) \cdot (B+D)$

Product of Sums Example

pos



اذا طلب pos F → sop $\bar{F}$ → group يقل zero

اذا طلب pos $\bar{F}$ → sop F → group يقل 1

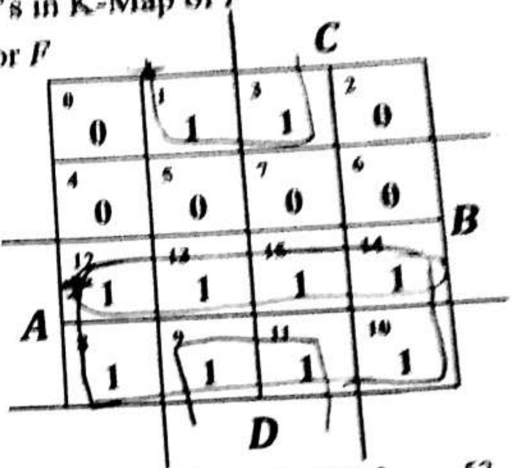
Example

- Find the optimum POS and SOP solution for:

$$F(A,B,C,D) = \prod_{M}(0,2,4,5,6,7)$$

M → Maxterms

- POS solution (Red):
  - Find optimized SOP for $\bar{F}$ by combining 0's in K-Map of F
  - Complement $\bar{F}$ to obtain optimized POS for F



- SOP solution (Blue):

SOP F → 1

ess + 1 3 9 11 → $\bar{B}D$

ess + 12 13 15 14 8 9 11 10 → 1 A

SOP F → SOP $\bar{F}$

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

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

Example

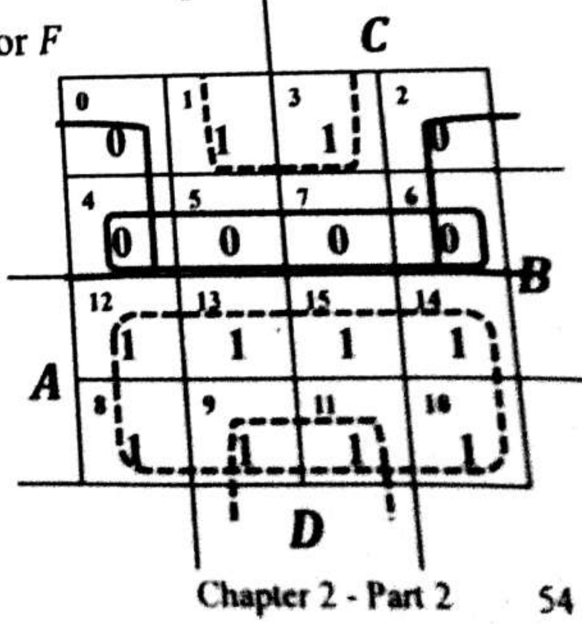
- Find the optimum POS and SOP solution for:

$$F(A,B,C,D) = \prod_{M}(0,2,4,5,6,7)$$

- POS solution (Red):
  - Find optimized SOP for $\bar{F}$ by combining 0's in K-Map of F
  - Complement $\bar{F}$ to obtain optimized POS for F

SOP $\bar{F}(A,B,C,D) = \bar{A}B + \bar{A}\bar{D}$ ← zero

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



- SOP solution (Blue):
  - Combining 1's in K-Map of F

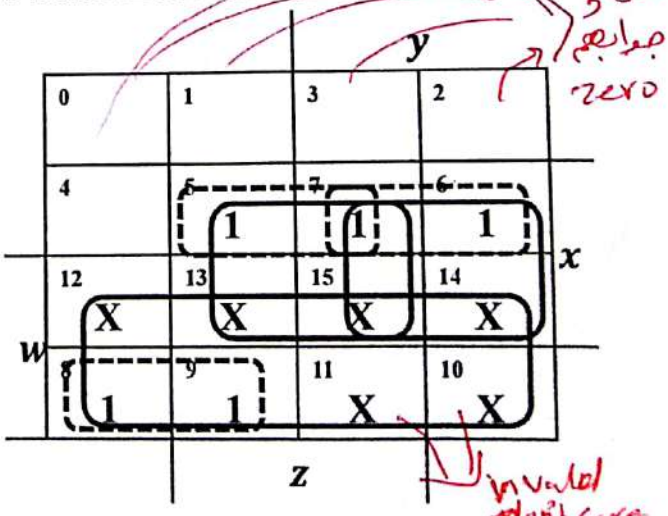
$$F(A,B,C,D) = A + \bar{B}D$$

Don't Cares in K-Maps

- Incompletely specified functions: Sometimes a function table or map contains entries for which it is known:
  - the input values for the minterm will never occur, or
  - The output value for the minterm is not used
- In these cases, the output value is defined as a "don't care"
- By placing "don't cares" (an "x" entry) in the function table or map, the cost of the logic circuit may be lowered
- **Example:** A logic function having the binary codes for the BCD digits as its inputs. Only the codes for 0 through 9 are used. The six codes, 1010 through 1111 never occur, so the output values for these codes are "x" to represent "don't cares"
- ***"Don't care" minterms cannot be replaced with 1's or 0's because that would require the function to be always 1 or 0 for the associated input combination***

Example: BCD "5 or More"

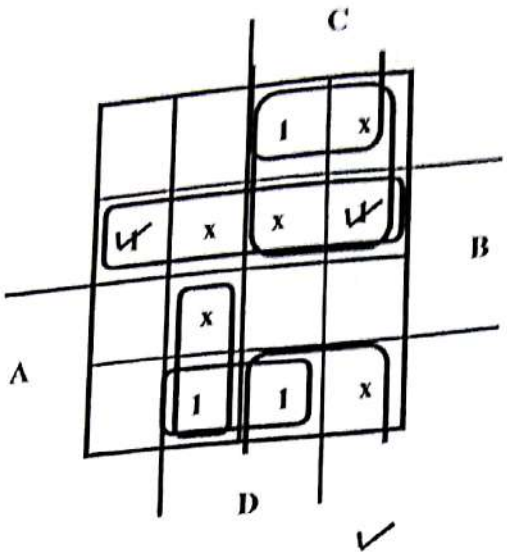
- The map below gives a function $F(w, x, y, z)$ which is defined as "5 or more" over BCD inputs. With the don't cares used for the 6 non-BCD combinations:
- If don't cares are treated as 1's (Red):
- $F_1(w, x, y, z) = w + xy + xz$
- $G = 7$
- If don't cares are treated as 0's (Blue):
- $F_2(w, x, y, z) = \bar{w}xz + \bar{w}xy + w\bar{x}\bar{y}$
- $G = 12$

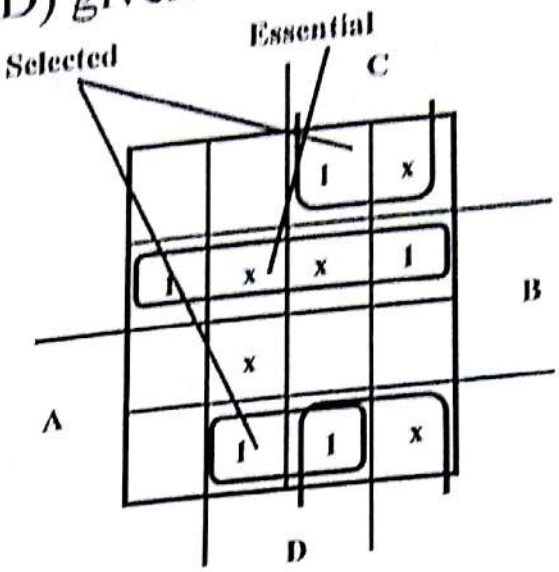


- ***For this particular function, cost G for the POS solution for $F(w, x, y, z)$ is not changed by using the don't cares***
  - ***Choose the one less inverters (i.e. less GN)***

Selection Rule Example with Don't Cares

- Simplify F(A, B, C, D) given on the K-map.



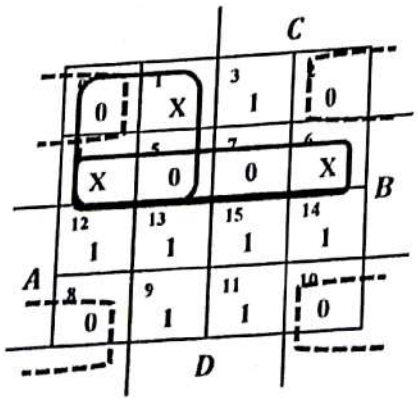


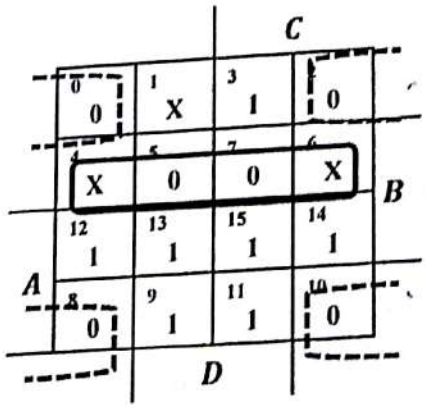
Minterms covered by essential prime implicants

Product of Sums with Don't Care Example

- Find the optimum **POS** solution for:

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





POSF → SOPF → SOP ?

0 2 8 10 → $\bar{B}\bar{D}$

4 5 7 6 → $\bar{A}B$

SOP $\bar{F}(A,B,C,D) = \bar{A}B + \bar{B}\bar{D}$ ⟵ Demorgan

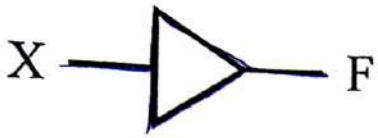
SOP $F(A,B,C,D) = (A+\bar{B})(B+D)$

Buffer

output ⟷ input نفس

تستخدم لتقوية الاشارة

- A ***buffer*** is a gate with the function $F = X$:



X	F
0	0
1	1

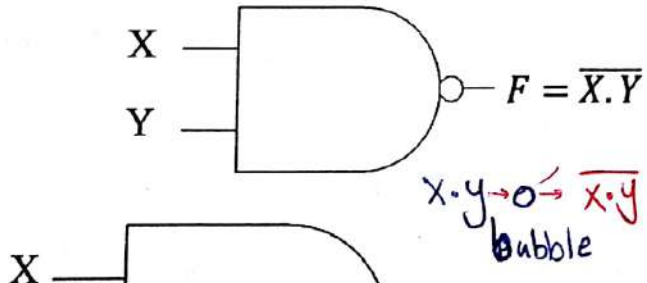
input output

- In terms of Boolean function, a buffer is the same as a connection!
- **So why use it?**
  - A buffer is an electronic amplifier used to improve circuit voltage levels and increase the speed of circuit operation
  - Protection and isolation between circuits

NAND Gate ⇒ Not + AND

ملاحظة :-
وين ما الشوف (○) كرة
مني لوجيك معناها Not

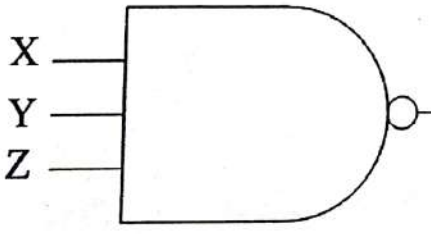
- The NAND gate has the following symbol and truth table:



X	Y	F
0	0	1
0	1	1
1	0	1
1	1	0

x·y → 0 → $\overline{x \cdot y}$
bubble

x·y x̄·y x·ȳ
0,0 → 0 → 1
0·1 → 0̄ → 1
1·0 → 0̄ → 1
1·1 → 0̄ → 0



اذا كان اي وحدة ← منهم zero نجعل 1

X yz
0 1 0
1 1 1

x·y·z $\overline{x \cdot y \cdot z}$
0 1
1 0

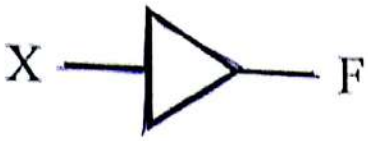
- **NAND** represents **NOT-AND**, i.e., the AND function with a NOT applied. The symbol shown is an **AND-Invert**. The small circle ("bubble") represents the invert function

$\overline{F} = \overline{x \cdot y \cdot z} \longrightarrow \bar{x} + \bar{y} + \bar{z}$

Buffer

output ⟷ input
نفس

- A ***buffer*** is a gate with the function $F = X$:



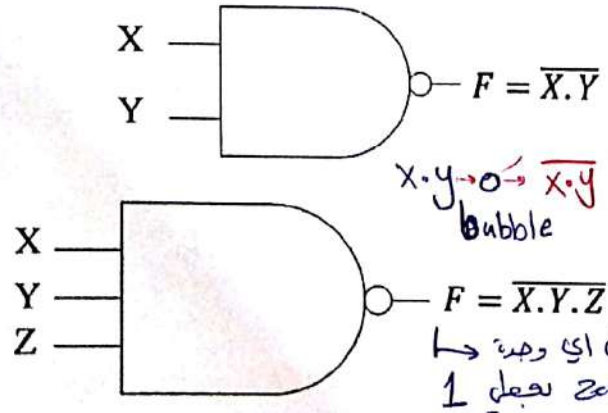
X	F
0	0
1	1

input output

- In terms of Boolean function, a buffer is the same as a connection!
- **So why use it?**
  - A buffer is an electronic amplifier used to improve circuit voltage levels and increase the speed of circuit operation
  - Protection and isolation between circuits

NAND Gate ⇒ Not + AND

- The NAND gate has the following symbol and truth table:



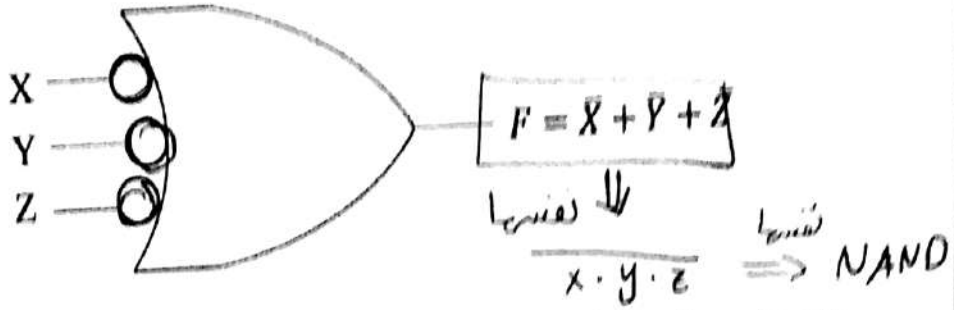
X	Y	F
0	0	1
0	1	1
1	0	1
1	1	0

- **NAND** represents **NOT-AND**, i.e., the AND function with a NOT applied. The symbol shown is an **AND-Invert**. The small circle ("bubble") represents the invert function

$\overline{F} = \overline{X \cdot Y \cdot Z} \longrightarrow \overline{X} + \overline{Y} + \overline{Z}$ (De Morgan)

NAND Gates (continued)

- Applying DeMorgan's Law gives **Invert-OR** (NAND)

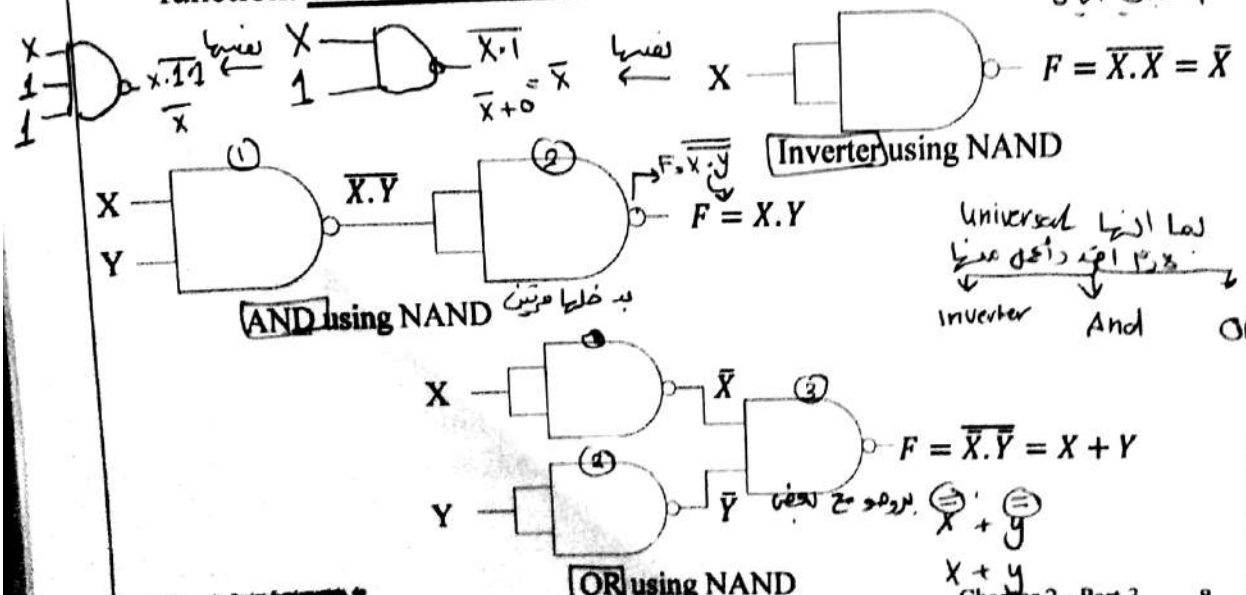


- This NAND symbol is called **Invert-OR**, since inputs are inverted and then ORed together

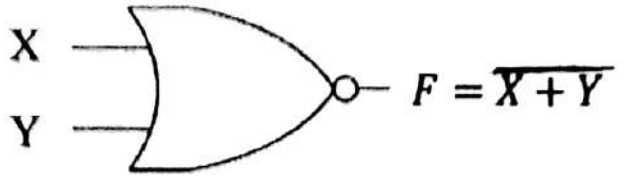
- **AND-Invert** and **Invert-OR** both represent the NAND gate. Having both makes visualization of circuit function easier

NAND Gates (continued)

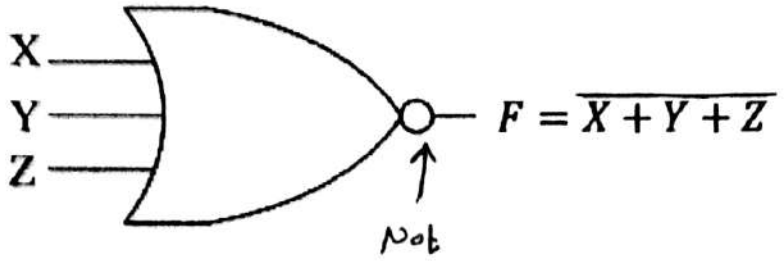
- ***Universal gate***: a gate type that can implement any Boolean function. **The NAND gate is a universal gate**



- The NOR gate has the following symbol and truth table:



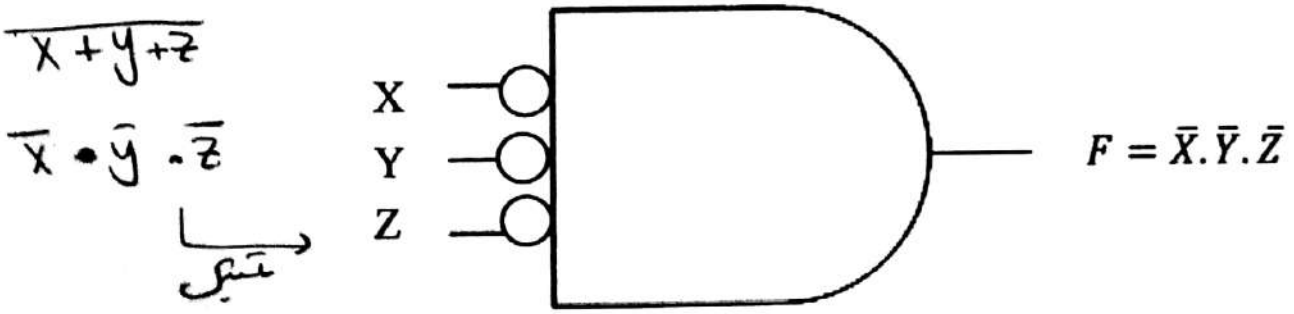
X	Y	F
0	0	1
0	1	0
1	0	0
1	1	0



- **NOR** represents **NOT-OR**, i.e., the OR function with a NOT applied. The symbol shown is an **OR-Invert**. The small circle ("bubble") represents the invert function

NOR Gates (continued)

- Applying DeMorgan's Law gives **Invert-AND** (NOR)

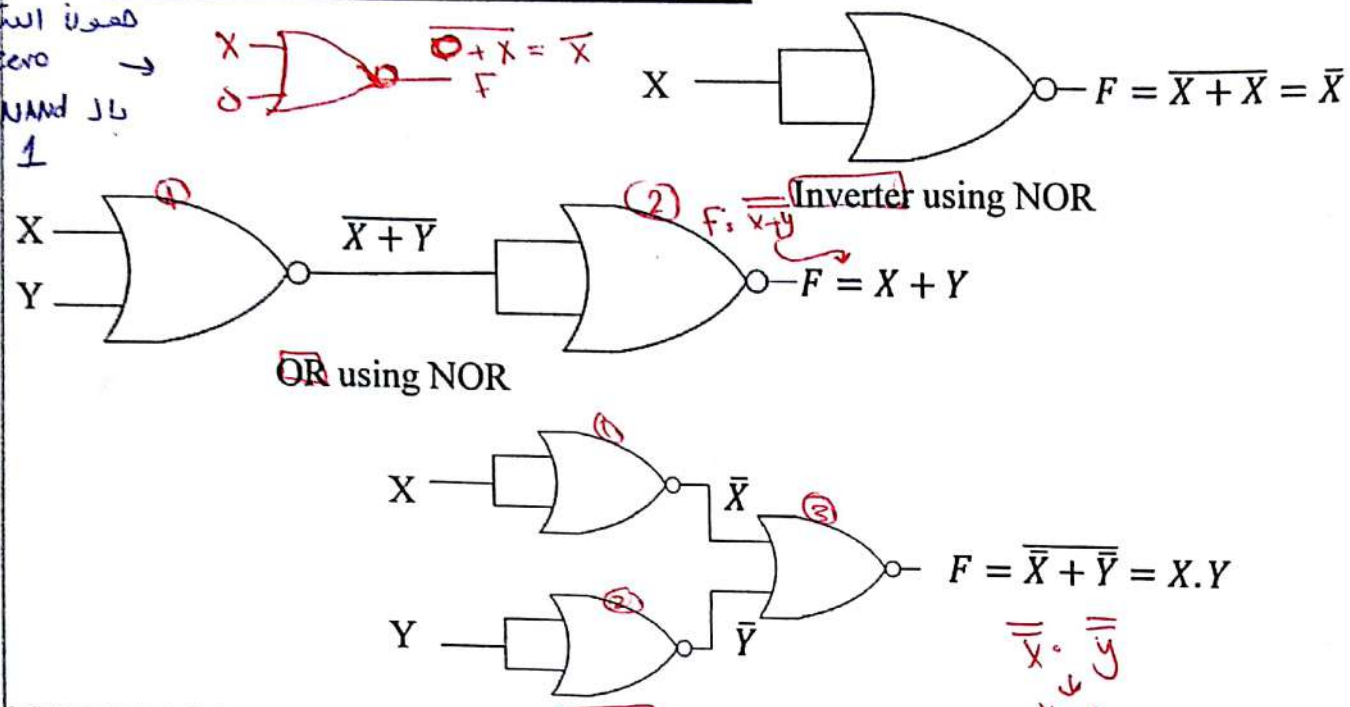


- This NOR symbol is called **Invert-AND**, since inputs are inverted and then ANDed together

- **OR-Invert** and **Invert-AND** both represent the NOR gate. Having both makes visualization of circuit function easier

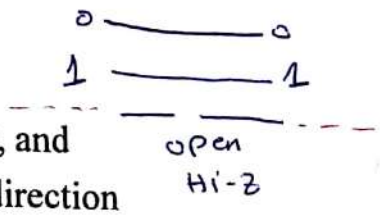
NOR Gates (continued)

- **The NOR gate is a universal gate:**



Hi-Impedance Outputs (Hi-Z)

- Logic gates introduced thus far
  - have 1 and 0 output values,
  - cannot have their outputs connected together, and
  - transmit signals on connections in only one direction



- Three-state logic adds a third logic value, **Hi-Impedance (Hi-Z)**, giving three states: 0, 1, and Hi-Z on the outputs.
- ***Hi-Z can be also denoted as Z or z***
- The presence of a Hi-Z state makes a gate output as described above behave quite differently:
  - "1 and 0" become "1, 0, and Hi-Z"
  - "cannot" becomes "can," and
  - "only one" becomes "two"

- This means that, looking back into the circuit, the output appears to be disconnected
- It is as if a switch between the internal circuitry and the output has been opened

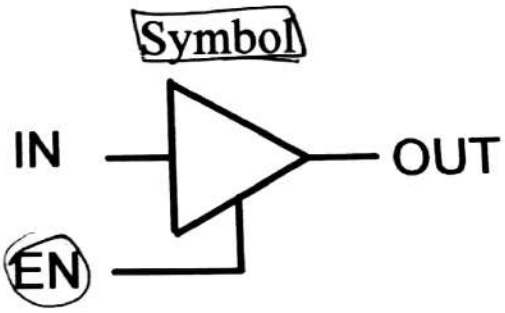
- Hi-Z may appear on the output of any gate, but we restrict gates to ***3-state buffer***

Tri-State Buffer (3-State Buffer)

- For the symbol and truth table, **IN** is the **data input,** and **EN** is the **control input**

- For **EN = 0**, regardless of the value on IN (denoted by X), the output value is Hi-Z

- For **EN = 1**, the output value follows the input value



Truth Table

EN	IN	OUT	
0	X	Hi-Z	①
1	0	0	②
1	1	1	③

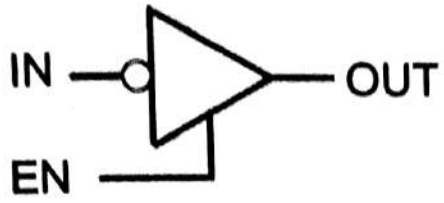
buffer يشتغل ما ←

active
buffer داخل
buffer يشتغل كـ

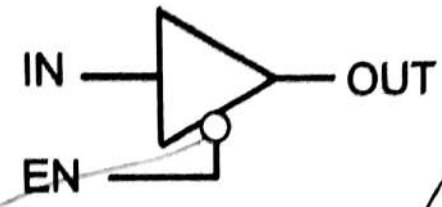
IN ⟶ output
نفس الشيء

Tri-State Buffer Variations

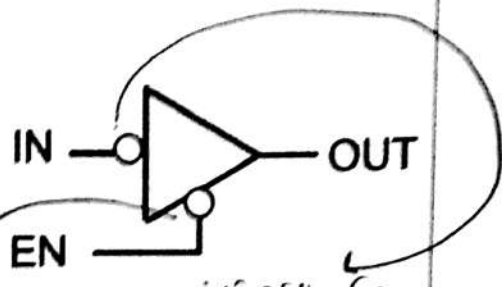
- By adding "bubbles" to signals:
  - Data input, IN, can be inverted
  - Control input, EN, can be inverted



EN	IN	OUT
0	X	Hi-Z
1	0	1
1	1	0



EN	IN	OUT
0	0	0
0	1	1
1	X	Hi-Z



EN	IN	OUT
0	0	1
0	1	0
1	X	Hi-Z

Tri-State Buffer Variations

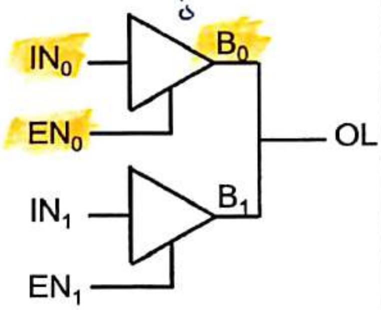
0 → ما يشتغل → Hi-Z

1 → يشتغل

Resolving 3-State Values on a Connection

- Connection of two tri-state buffer outputs, B_1 and B_0, to a wire, OL (Output Line) → Multiplexed Output

EN_1	EN_0	IN_1	IN_0	B_1	B_0	OL
0	0	X	X	Hi-Z	Hi-Z	Hi-Z
0	1	X	0	Hi-Z	0	0
0	1	X	1	Hi-Z	1	1
1	0	0	X	0	Hi-Z	0
1	0	1	X	1	Hi-Z	1
1	1	0	0	0	0	0
1	1	1	1	1	1	1
1	1	0	1	0	1	
1	1	1	0	1	0	



4 input $2^4 = 16$

Resolving 3-State Values on a Connection

- **<u>Resulting Rule:</u> At least one buffer output value must be Hi-Z. Why?**
  - Because any data combinations including (0,1) and (1,0) can occur. If one of these combinations occurs, and no buffers are Hi-Z, then high currents can occur, destroying or damaging the circuit
- **How many valid buffer output combinations exist?**
  - 5 valid output combination
- **What is the rule for "*n*" tri-state buffers connected to wire, OL?**
  - At least "n-1" buffer outputs must be Hi-Z
  - **How many valid buffer output combinations exist ?**
    - Each of the n-buffers can have a 0 or 1 output with all others at Hi-Z. Also all buffers can be Hi-Z. So there are 2n + 1 valid combinations.

1 او zero input یعنی

Exclusive OR/ Exclusive NOR

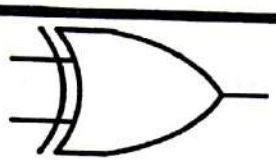
- Uses for the XOR and XNORs gate include:
  - Adders/subtractors/multipliers
  - Counters/incrementers/decrementers
  - Parity generators/checkers
- Definitions
  - The XOR function is: $X \oplus Y = \bar{X}Y + X\bar{Y}$
  - The XNOR function is: $X \odot Y = \overline{X \oplus Y} = XY + \bar{X}\bar{Y}$
- Strictly speaking, XOR and XNOR gates ***do no exist for more than two inputs***. Instead, they are replaced by odd and even functions

Proof: XNOR is the complement of XOR

- $\overline{X \oplus Y} = \overline{\bar{X}Y + X\bar{Y}}$
- $\overline{X \oplus Y} = \overline{\bar{X}Y} . \overline{X\bar{Y}}$
- $\overline{X \oplus Y} = (X + \bar{Y})(\bar{X} + Y)$
- $\overline{X \oplus Y} = X\bar{X} + XY + \bar{X}\bar{Y} + Y\bar{Y}$
- $X \odot Y = \overline{X \oplus Y} = XY + \bar{X}\bar{Y}$

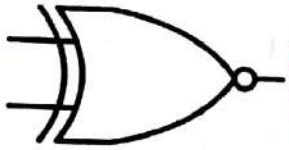
Symbols For XOR and XNOR

- XOR symbol:



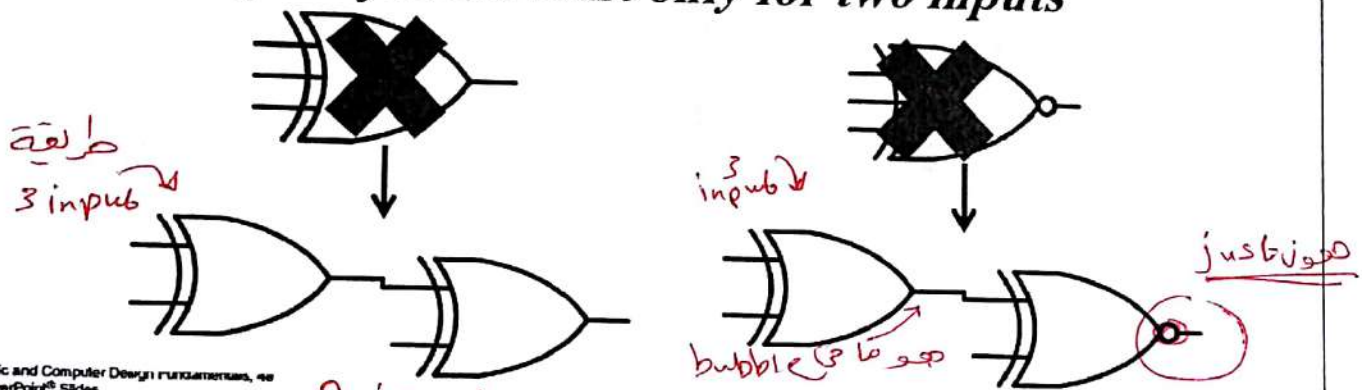
رمز ⊕ | $F = \bar{x}y + x\bar{y}$ | 2 inputs لازم

- XNOR symbol:



⊙ | $F = xy + \bar{x}\bar{y}$

- ***Shaped symbols exist only for two inputs***



2 input just

Truth Tables for XOR/XNOR

X	Y	$X \oplus Y$
0	0	0
0	1	1 m_1
1	0	1 m_2
1	1	0

m_1 m_2 — 1 — الباقي صفر

الاشارة zero — 1 — مختلف 1

X	Y	$X \odot Y (X \equiv Y)$
0	0	m_0 1
0	1	0
1	0	0
1	1	m_3 1

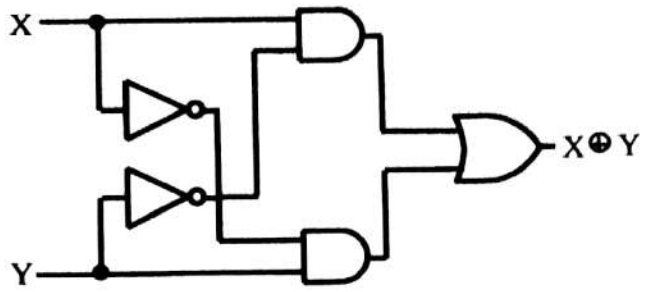
m_0 m_3 — 1 — الباقي صفر

يجب zero لما يكونوا متشابهين

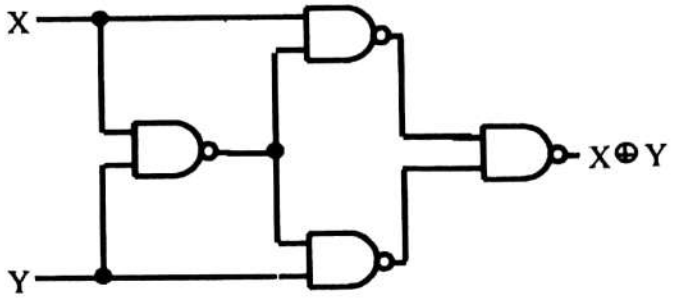
يجب 1 لما يكونوا مختلفين

- The XOR function means: ***X OR Y, but NOT BOTH***
- Why is the XNOR function also known as the ***equivalence*** function, denoted by the operator ≡?
  - Because the function equals 1 if and only if $X = Y$

- **The simple SOP implementation uses the following structure:**



- **A NAND only implementation is:**



XOR

- The XOR identities:

لتبسيط

$X \oplus 0 = X$	$X \oplus 1 = \overline{X}$ → inverter
$X \oplus X = 0$	$X \oplus \overline{X} = 1$
$X \oplus \overline{Y} = \overline{X \oplus Y}$	$\overline{X} \oplus Y = \overline{X \oplus Y}$
$X \oplus Y = Y \oplus X$	] ما فرق ترتيب
$(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z) = X \oplus Y \oplus Z$	

- **The XOR function can be extended to 3 or more variables. For more than 2 variables, it is called an *odd function* or *modulo 2 sum* (*Mod 2 sum*), not an XOR:**

$$X \oplus Y \oplus Z = \overline{X}\overline{Y}Z + \overline{X}Y\overline{Z} + X\overline{Y}\overline{Z} + XYZ \text{ (Odd \# of 1's)}$$

odd 1 عدد → 1

1 عدد → zero

1 عدد odd → 1

XNOR

- The XNOR identities:

$X \odot 0 = \bar{X}$	$X \odot 1 = X$
$X \odot X = 1$	$X \odot \bar{X} = 0$
$X \odot Y = Y \odot X$	
$(X \odot Y) \odot Z \neq X \odot (Y \odot Z) = X \odot Y \odot Z$	

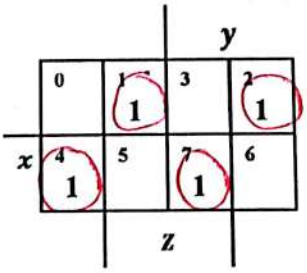
  - The XNOR function can be extended to 3 or more variables. For more than 2 variables, it is called an ***even function***, not an XNOR:

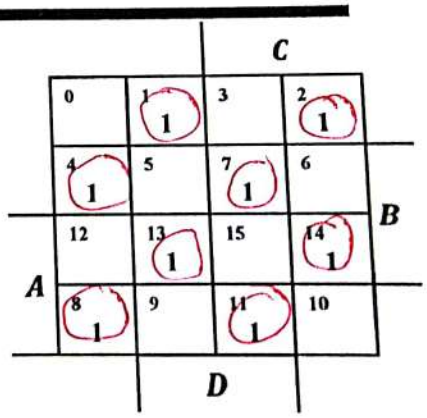
$$X \odot Y \odot Z = \bar{X}YZ + X\bar{Y}Z + XY\bar{Z} + \bar{X}\bar{Y}\bar{Z}$$ (Even # of 1's)

  - ***The even function is the complement of the odd function***

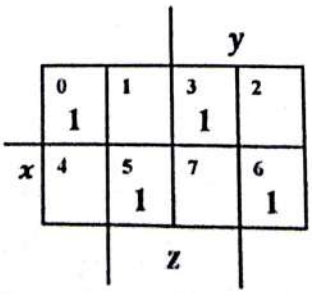
Odd and Even Functions

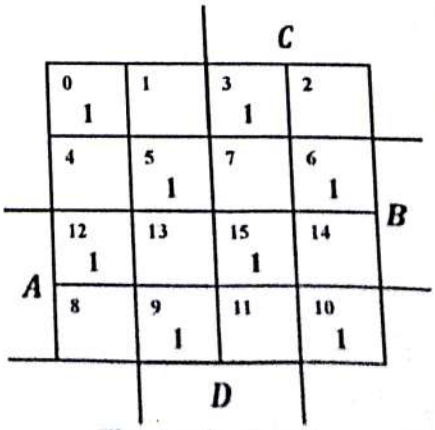
- The 1s of an ***odd function*** correspond to minterms having an index with an odd number of 1s.





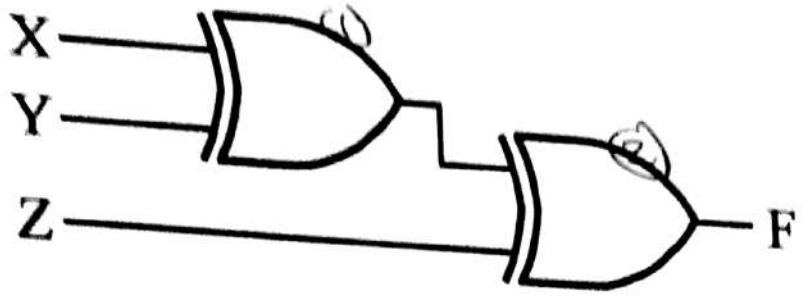
- The 1s of an ***even function*** correspond to minterms having an index with an even number of 1s.





Example: Odd Function Implementation

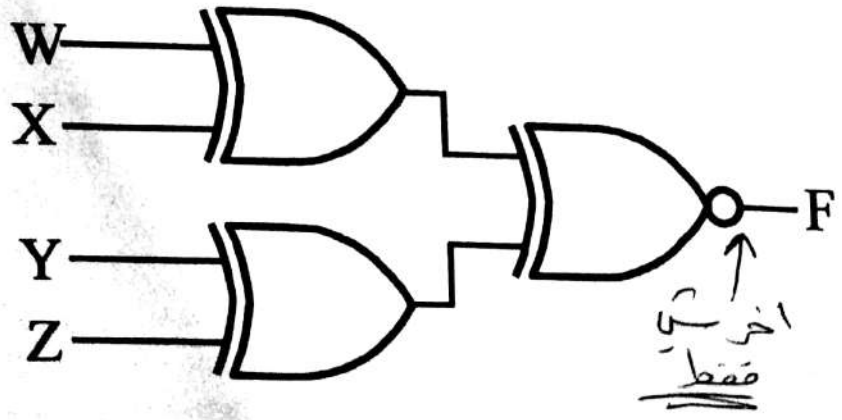
- Design a 3-input odd function $F = X \oplus Y \oplus Z$ with 2-input XOR gates
- Factoring, $F = (X \oplus Y) \oplus Z$
- The circuit:



2 input لازم اباخذ
لازم احطهم مرتين

Example: Even Function Implementation

- Design 4-input even function $F = \overline{W \oplus X \oplus Y \oplus Z}$ with 2-input XOR and XNOR gates
- Factoring, $F = \overline{(W \oplus X) \oplus (Y \oplus Z)}$
- The circuit:



اذا عدد 1 odd يضيف 1 ← even parity

اذا عدد 1 even يضيف 0

يستخدم XOR (odd fun)

110110 →

اصلا even parity

* بعد عدد 1

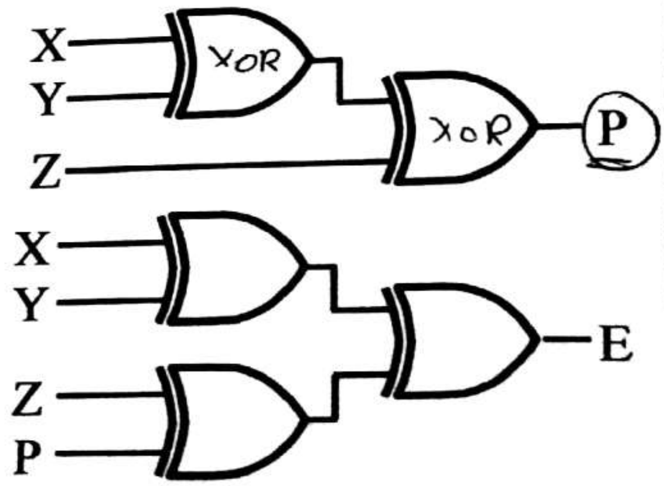
* نستخدم XOR لحسب قيمة parity

* تستخدم اذا صار في خطأ او لا

Parity Generators and Checkers

- In Chapter 1, a parity bit added to n-bit code to produce an n+1 bit code:
  - Add odd parity bit to generate code words with even parity
  - Add even parity bit to generate code words with odd parity
  - Use odd parity circuit to check code words with even parity
  - Use even parity circuit to check code words with odd parity

Even parity



- Example: n = 3. Generate **even parity** code words of length four **with odd parity generator (XOR)**:
- Check **even parity** code words of length four with **odd parity checker (XOR)**:
- Operation: (X,Y,Z) = (0,0,1) gives (X,Y,Z,P) = (0,0,1,1) and E = 0. If Y changes from 0 to 1 between generator and checker, then E = 1 indicates an error

Overview

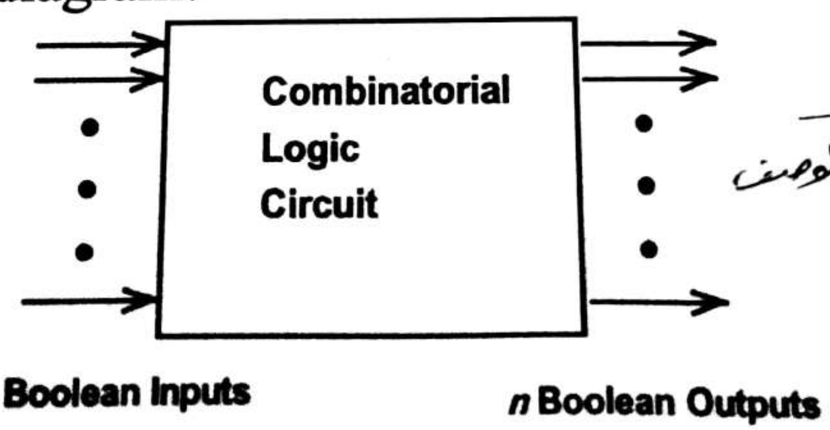
- Part 1 – Design Procedure
  - Steps
    - Specification
    - Formulation
    - Optimization
    - Technology Mapping
    - Verification
  - Technology Mapping - AND, OR, and NOT to NAND or NOR

Combinational Circuits

- A combinational logic circuit has:
  - A set of *m* Boolean inputs,
  - A set of *n* Boolean outputs, and
  - *n* switching functions, each mapping the 2^m input combinations to an output such that the ***current output depends only on the current input values***

من الوصف لازم اعرف

- A block diagram:



يصير تحي من F يعطي 1

بناءً على هذا الوصف بجيب truth table ← يستخدم k map ← يصير Som او Pom

Design Exam

- **Specification:** Design a combinational circuit that has *3 inputs (X, Y, Z)* and *one output F*, such that $F = 1$ when the number of 1's in the input is greater than the number of 0's (i.e. number of 1's ≥ 2)
  - This is called ***majority function*** (i.e. majority of inputs must be 1 for the function to be 1).

- **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

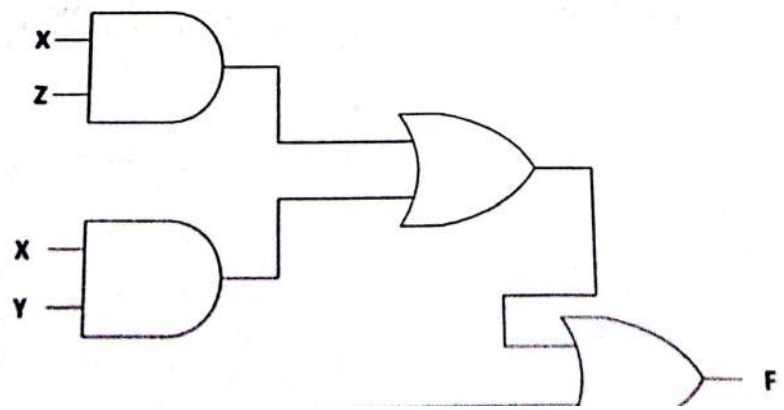
Design Example1 Cont.

- **Optimization:**

$$F(X,Y,Z) = \underline{XY} + \underline{XZ} + \underline{YZ}$$
$$(6,7) \quad (5,7) \quad (3,7)$$

X \ YZ	00	01	11	10
0	0	1	3	2
1	4	5	7	6

- **Technology Mapping:**
  - Mapping with a library containing inverters, 2-input AND, 2-input OR



- **Specification:** Design a combinational circuit that compares *2-bit* Binary number *(A, B)* and produce two outputs (O_1, O_0), such that:

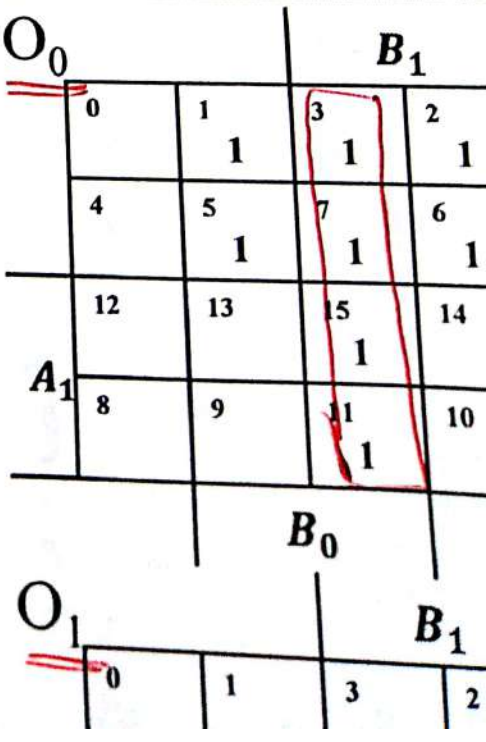
$O_1O_0 = 00$	When $A = B$ and Both are even
$O_1O_0 = 01$	When $A < B$
$O_1O_0 = 10$	When $A > B$
$O_1O_0 = 11$	When $A = B$ and Both are odd

$A(A_1A_0)$	$B(B_1B_0)$	$O(O_1O_0)$
00	00	00
00	01	01
00	10	01
00	11	01
01	00	10
01	01	11
01	10	01
01	11	01
10	00	10
10	01	10
10	10	00
10	11	01
11	00	10
11	01	10
11	10	10
11	11	11

- **Formulation:**

Design Example2 Cont.

Optimization and Technology Mapping:

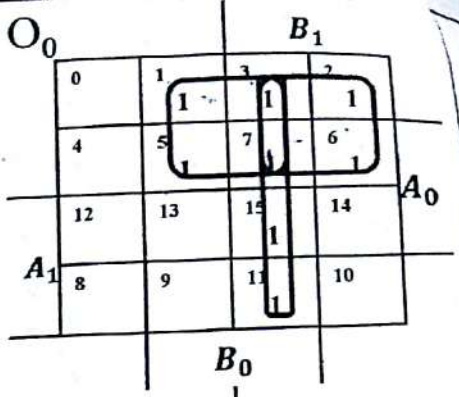


Optimization and Technology Mapping:

3 7 15 11 → $B_1 B_0$

2 3 1 7 → $\overline{A_1} B_0$

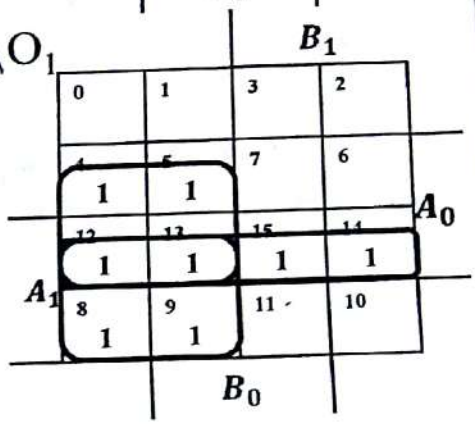
2 3 7 6 → $\overline{A_1} B_1$



12 13 15 14 → $A_1 A_0$

4 5 12 13 → $\overline{B_1} A_0$

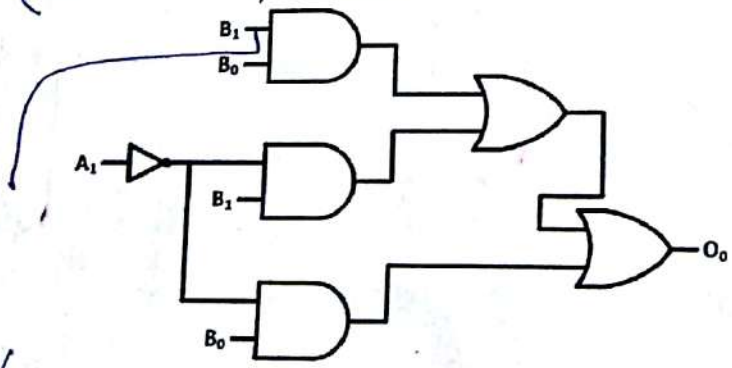
12 13 8 9 → ~~A_1~~ $\overline{B_1}$

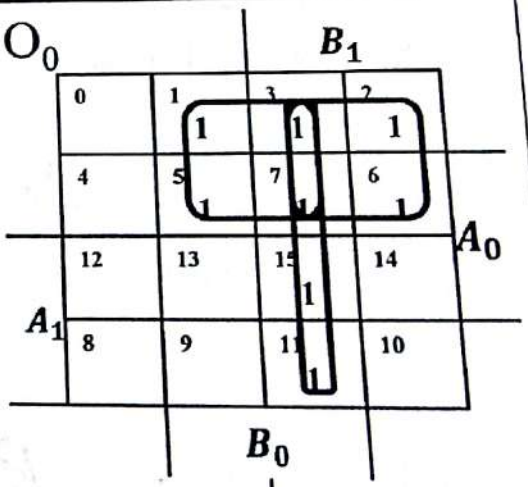


Design Example2 Cont.

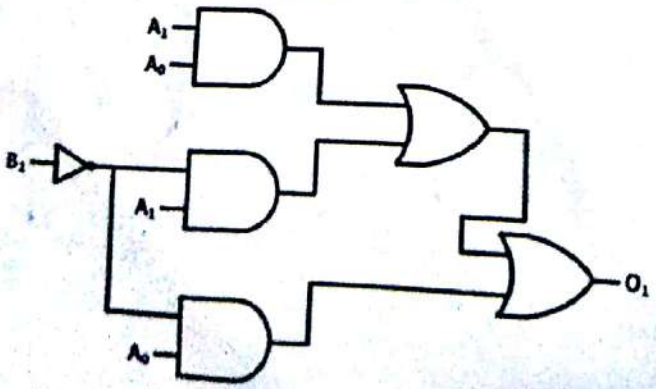
- **Optimization and Technology Mapping:**

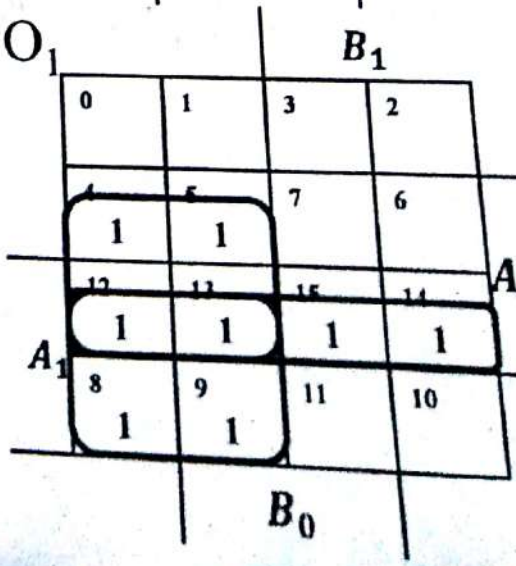
$$O_0 = B_1 B_0 + \overline{A_1} B_1 + \overline{A_1} B_0$$





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





Design Example3

1. Specification

BCD to Excess-3 code converter

Transforms BCD code for the decimal digits to Excess-3 code for the decimal digits

- BCD code words for digits 0 through 9: 4-bit patterns 0000 to 1001, respectively
- Excess-3 code words for digits 0 through 9: 4-bit patterns consisting of 3 (binary 0011) added to each BCD code word
- ***BCD input is labeled A, B, C, D***
- ***Excess-3 output is labeled W, X, Y, Z***

Design Example3 Cont.

2. Formulation

BCD ABCD	Excess-3 WXYZ
0000	0011
0001	0100
0010	0101
0011	0110
0100	0111
0101	1000
0110	1001
0111	1010
1000	1011
1001	1100
1010	XXXX
1011	XXXX
1100	XXXX
1101	XXXX
1110	XXXX
1111	XXXX

CLo, ε

Keymab ε

1	2	3	4
5 7 13 15	4 12	0 4 12 8	0 4 12 8 2 6 14 10
B D	$\bar{C}\bar{D}B$	$\bar{C}\bar{D}$	$\bar{D}$
7 6 15 14	13 9 11	3 7 15 11	
C B	$\bar{B}D$	C D	
8 − 14	3 2 11 10		
	$C\bar{B}$		

13 15 9 11 — A D — Essinational

Design Example3 Cont.

3. Optimization

$2^4 = 16$

W

AB \ CD	00	01	11	10
00	0	1	3	2
01	4	5: 1	7: 1	6: 1
11	12: X	13: X	15: X	14: X
10	8: 1	9: 1	11: X	10: X

X

AB \ CD	00	01	11	10
00	0	1: 1	3: 1	2: 1
01	4: 1	5	7	6
11	12: X	13: X	15: X	14: X
10	8	9: 1	11: X	10: X

Y

AB \ CD	00	01	11	10
00	0: 1	1	3: 1	2
01	4: 1	5	7: 1	6
11	12: X	13: X	15: X	14: X
10	8: 1	9	11: X	10: X

Z

AB \ CD	00	01	11	10
00	0: 1	1	3	2: 1
01	4: 1	5	7	6: 1
11	12: X	13: X	15: X	14: X
10	8: 1	9	11: X	10: X

A B C D

Design Example3 Cont.

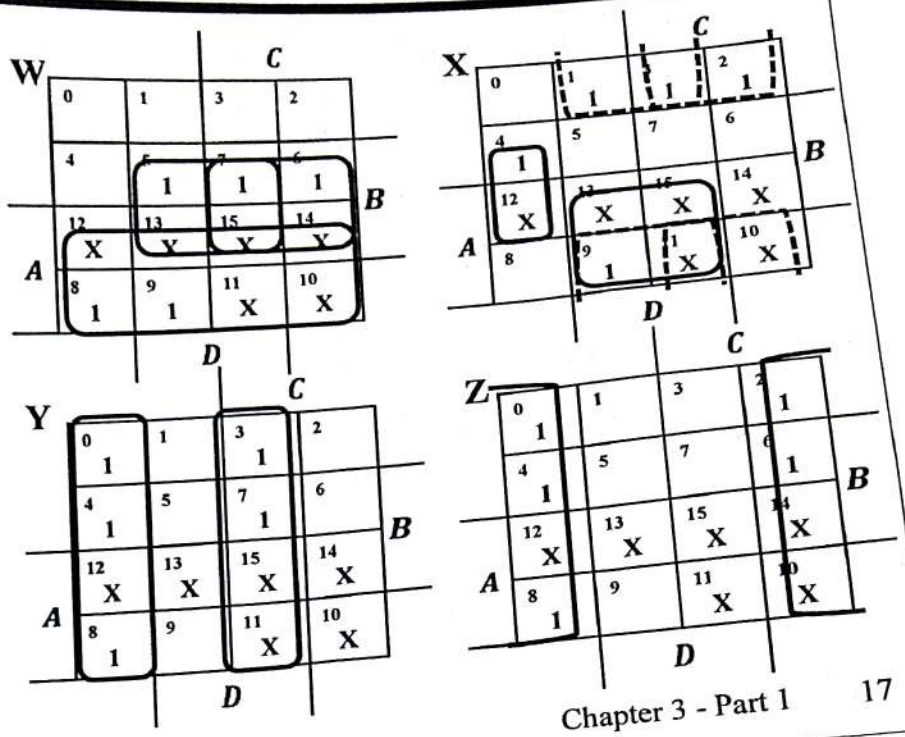
3. Optimization

$$W = A + (BC + BD)$$

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

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

$$Z = \bar{D}$$

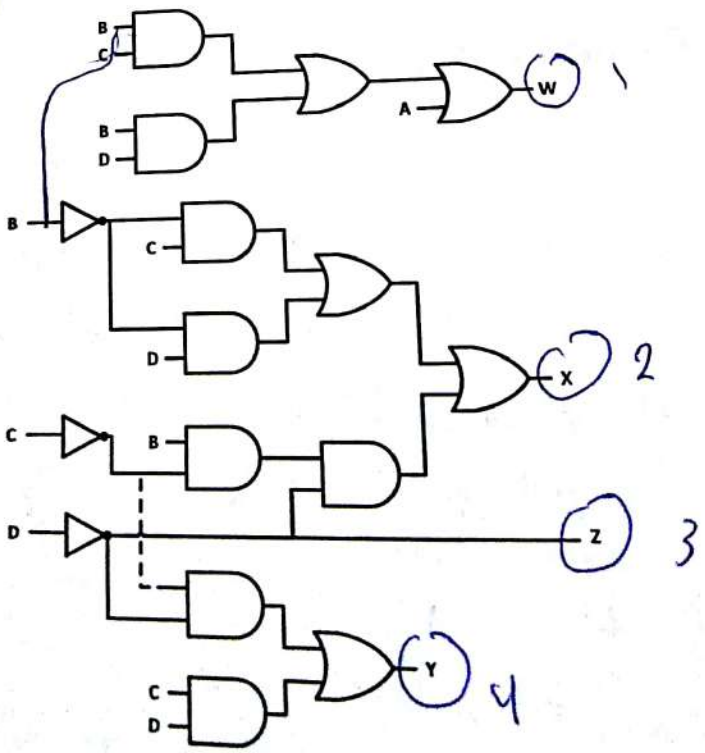


Design Example3 Cont.

4. Technology Mapping

- Mapping with a library containing inverters, 2-input AND, 2-input OR

2 input AND
2 input OR

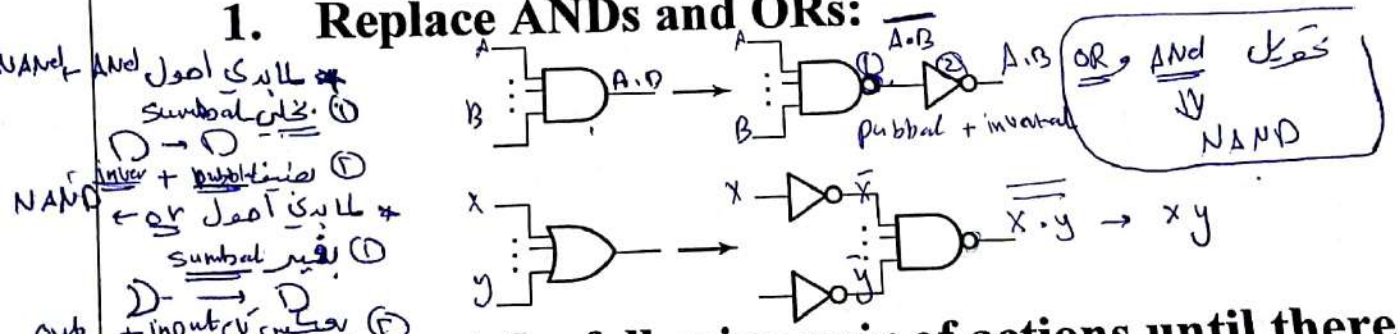


Mapping to NAND gates

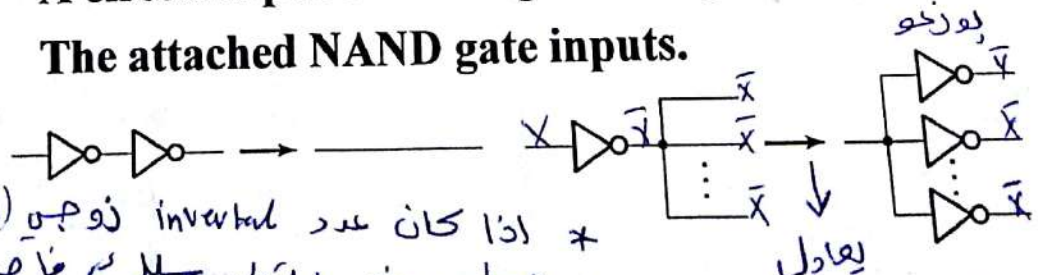
- **Assumptions:**
  - Gate loading and delay are ignored
  - Cell library contains an inverter and *n*-input NAND gates, $n = 2, 3, \ldots$
  - An AND, OR, inverter schematic for the circuit is available
- **The mapping is accomplished by:**
  - Replacing AND and OR symbols,
  - Pushing inverters through circuit fan-out points, and
  - Canceling inverter pairs

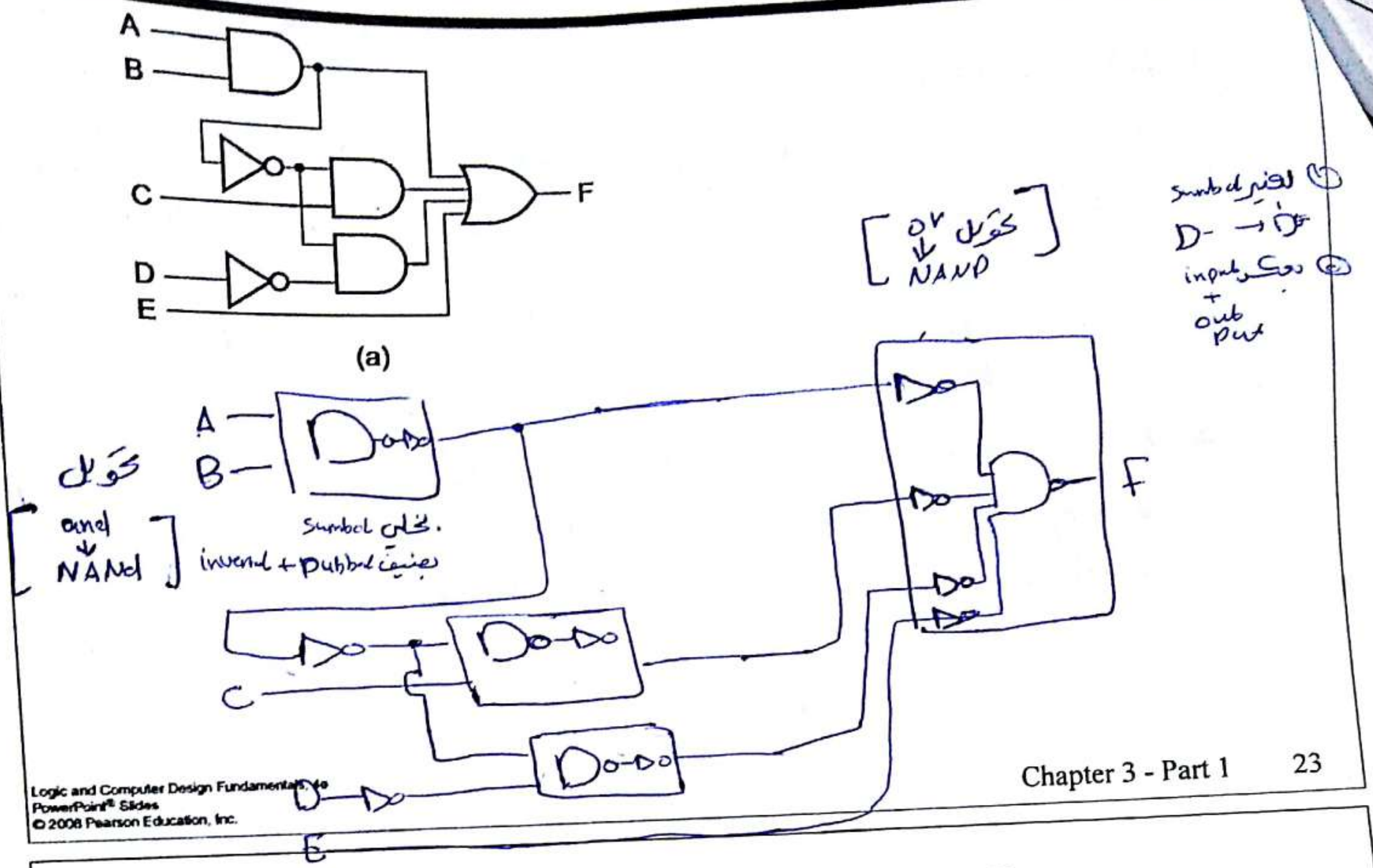
NAND Mapping Algorithm

1. **Replace ANDs and ORs:**

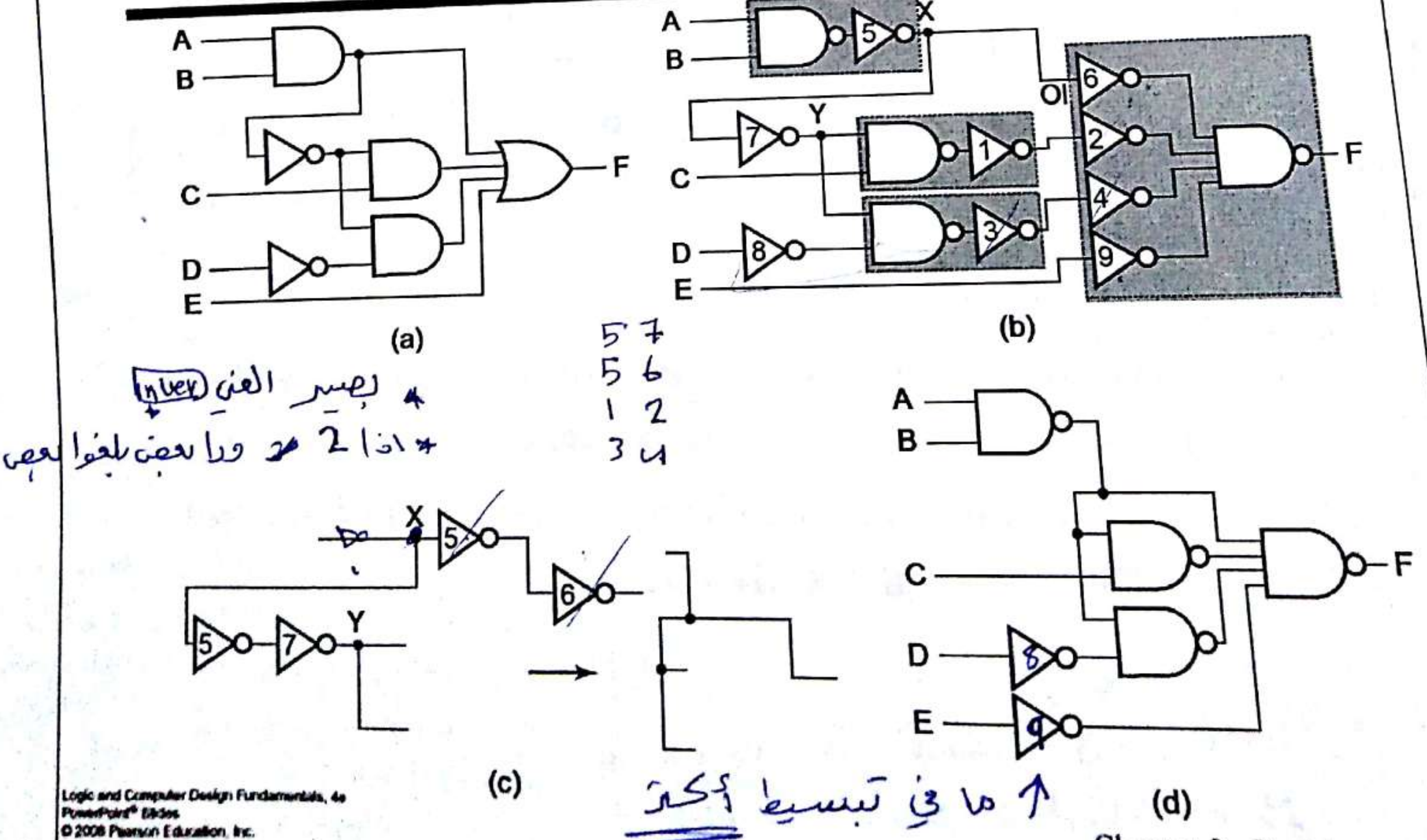


2. **Repeat the following pair of actions until there is at most one inverter between :**
   a. **A circuit input or driving NAND gate output, and**
   b. **The attached NAND gate inputs.**





NAND Mapping Example

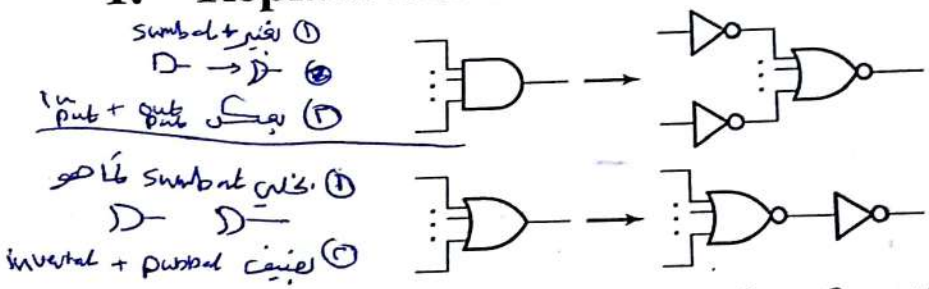


Mapping to NOR gates

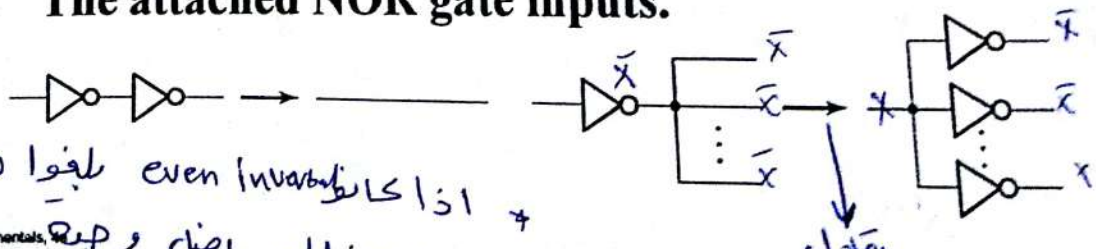
- **Assumptions:**
  - Gate loading and delay are ignored
  - Cell library contains an inverter and *n*-input NOR gates, $n = 2, 3, \ldots$
  - An AND, OR, inverter schematic for the circuit is available
- **The mapping is accomplished by:**
  - Replacing AND and OR symbols,
  - Pushing inverters through circuit fan-out points, and
  - Canceling inverter pairs

NOR Mapping Algorithm

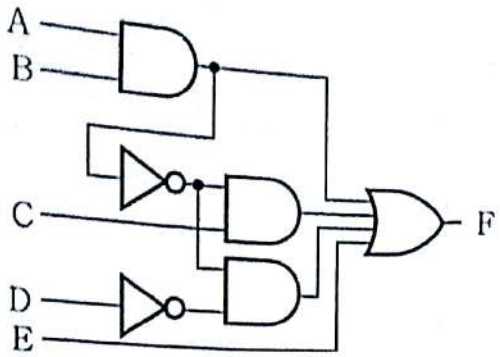
1. **Replace ANDs and ORs:**



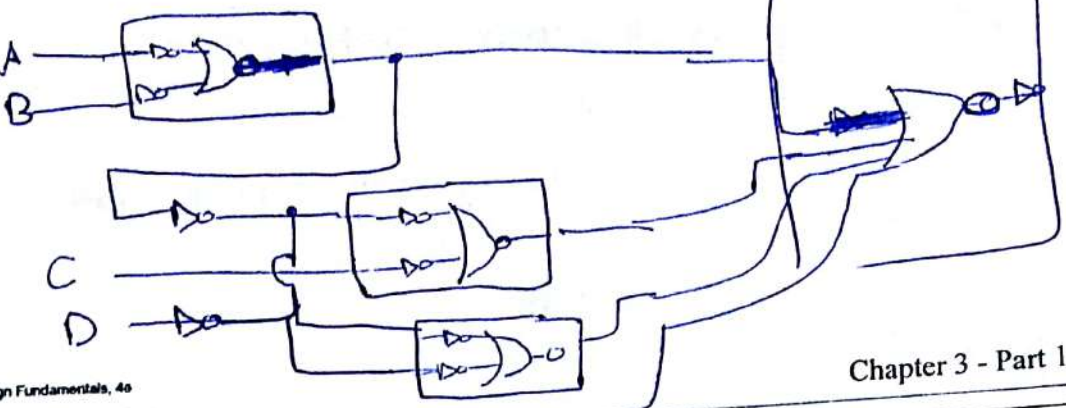
2. **Repeat the following pair of actions until there is at most one inverter between :**
   a. **A circuit input or driving NOR gate output, and**
   b. **The attached NOR gate inputs.**



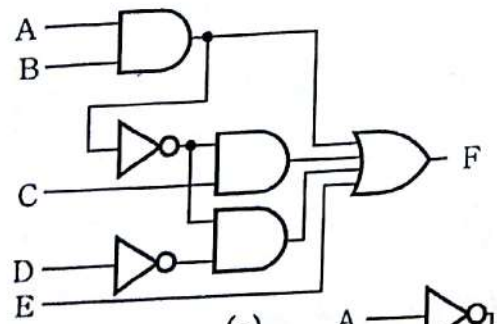
NOR Mapping Example



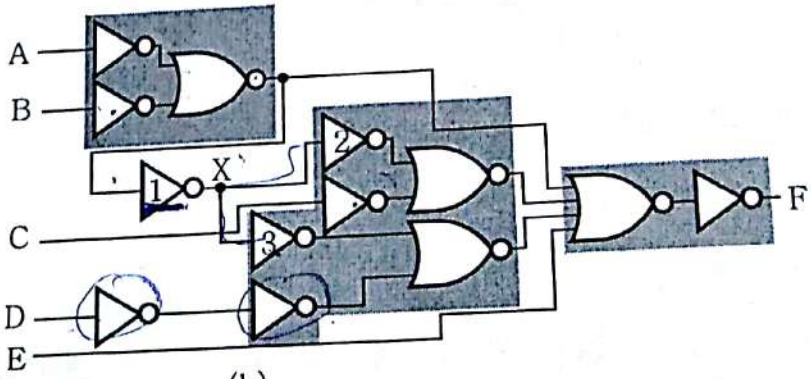
(a)



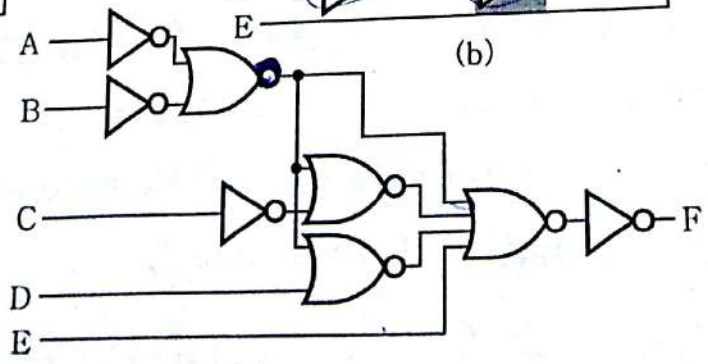
NOR Mapping Example



(a)



(b)



(c)

- **Part 2 – Combinational Logic**
  - Functions and functional blocks
  - Rudimentary logic functions
  - Decoding using Decoders
    - Implementing Combinational Functions with Decoders
  - Encoding using Encoders
  - Selecting using Multiplexers
    - Implementing Combinational Functions with Multiplexers

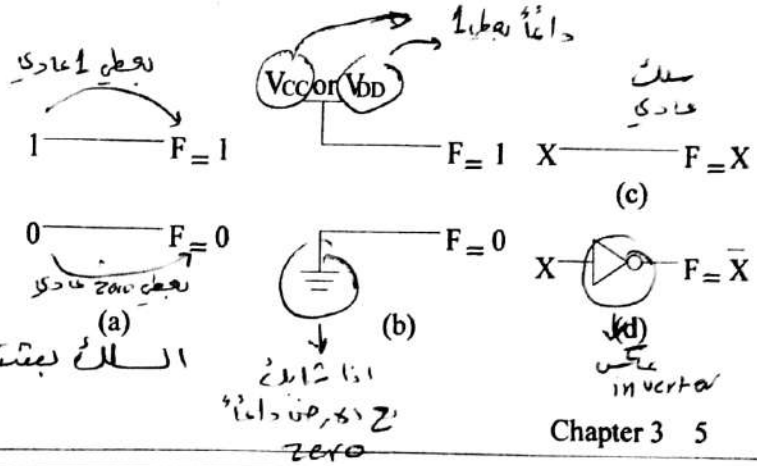
Functions and Functional Blocks

- The functions considered are those found to be very useful in design
- Corresponding to each of the functions is a combinational circuit implementation called a ***functional block***
- In the past, functional blocks were packaged as small-scale-integrated (SSI), medium-scale integrated (MSI), and large-scale-integrated (LSI) circuits
- Today, they are often simply implemented within a very-large-scale-integrated (VLSI) circuit

Functional Logic Functions

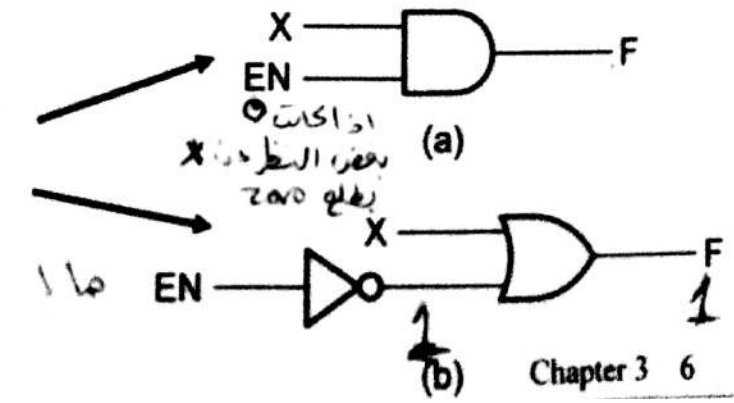
- Functions of a single variable X
- Can be used on the inputs to functional blocks to implement other than the block's intended function
  - ***Value fixing*** **: a, b**
  - ***Transferring*** **: c**
  - ***Inverting*** **: d**
  - ***Enabling*** **: next slide**

Functions of One Variable				
X	$F = 0$	$F = 1$	$F = X$	$F = \bar{X}$
0	0	1	0	1
1	0	1	1	0



Enabling Function

- ***Enabling*** permits an input signal to pass through to an output
- ***Disabling*** blocks an input signal from passing through to an output, replacing it with a fixed value
- The value on the output when it is disable can be ***Hi-Z*** (as for three-state buffers and transmission gates), 0 , or 1
- When disabled, ***0 output***
- When disabled, ***1 output***

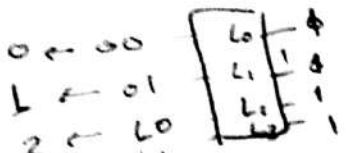


Decoding

- ***Decoding:*** the conversion of an ***n-bit*** input code to an ***m-bit*** output code with $n \leq m \leq 2^n$ such that each valid code word produces a unique output code
  - Circuits that perform decoding are called ***decoders***

- Functional blocks for decoding are
  - called ***n-to-m line decoders***, where $m \leq 2^n$, and
  - generate 2^n (or fewer) minterms for the *n* input variables

1-to-2 Line Decoder

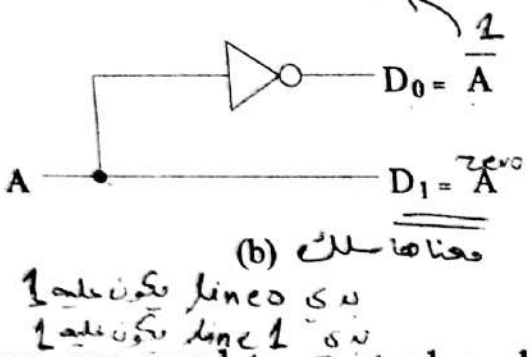


- When the decimal value of A equals the subscript of D_i, that D_i will be 1 and all others will be 0's

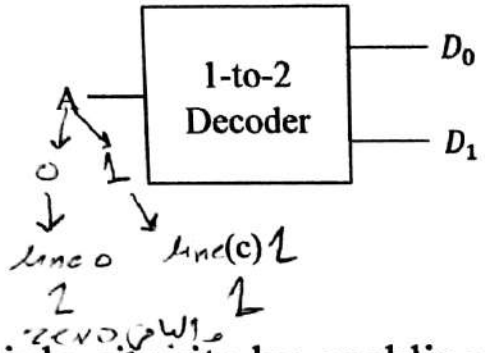
- Only one output is active at a time

A	D_0	D_1
0	1	0
1	0	1

(a)



(b)



(c)

- Decoders are used to control multiple circuits by enabling only one of them at a time

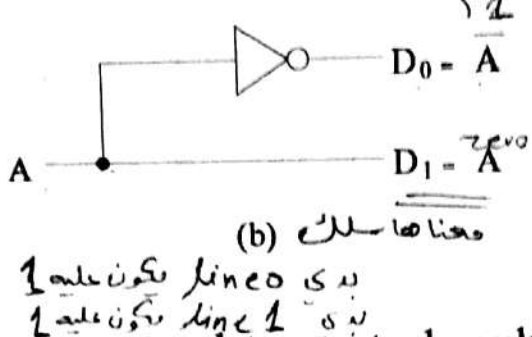
2-to-4 Line Decoder

- When the decimal value of A equals the subscript of D_i, that D_i will be 1 and all others will be 0's

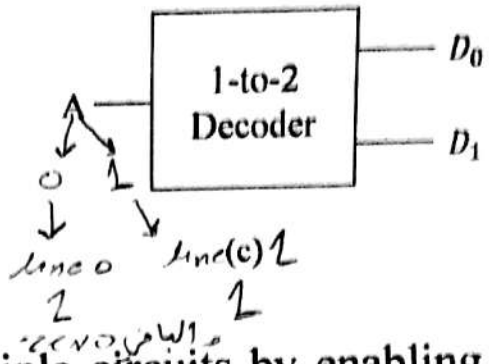
- Only one output is active at a time

A	D_0	D_1
0	1	0
1	0	1

(a)



(b)

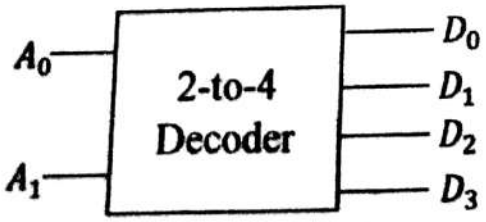


- Decoders are used to control multiple circuits by enabling only one of them at a time

2-to-4 Line Decoder

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

(a)



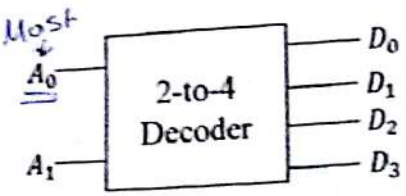
(c)

- No more optimization is possible
- ***Note that the 2-to-4 line decoder is made up of two 1-to-***

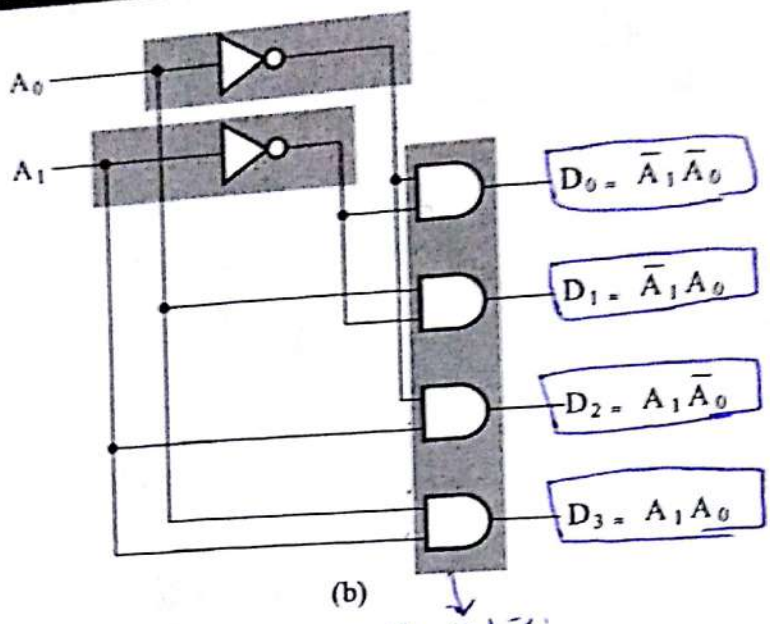
2-to-4 Line Decoder

A_1	A_0	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

(a)



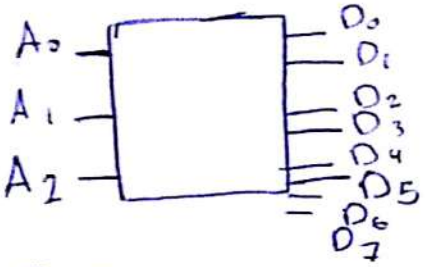
(c)



(b)

- No more optimization is possible
- ***Note that the 2-to-4 line decoder is made up of two 1-to-2- line decoders and 4 AND gates***

3 to 8 decoder



A_2	A_1	A_0
0	0	0
1	0	1
2	1	0
3	1	1
4	0	0
5	0	1
6	1	0
7	1	1

D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
		1					
			1				
				1			
					1		
						1	
							1

$D_0\ \bar{A}_2 \bar{A}_1 \bar{A}_0$
$D_1\ \bar{A}_2 \bar{A}_1 A_0$
$D_2\ \bar{A}_2 A_1 \bar{A}_0$
$D_3\ \bar{A}_2 A_1 A_0$
$D_4\ A_2 \bar{A}_1 \bar{A}_0$
$D_5\ A_2 \bar{A}_1 A_0$
$D_6\ A_2 A_1 \bar{A}_0$
$D_7\ A_2 A_1 A_0$

3 2-lis inverted and 8 2-lis. and gates

Cost and
$L = 24\ (8 \times 3)$
$G = 24 + 3 = 27$
inverted

cost

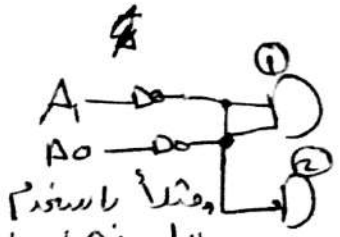
L = 16×4 = 64

G = 64+4 = 68

مثلاً

01 D_9 A_3 $\overline{A}_2$ $\overline{A}_1$ A_0

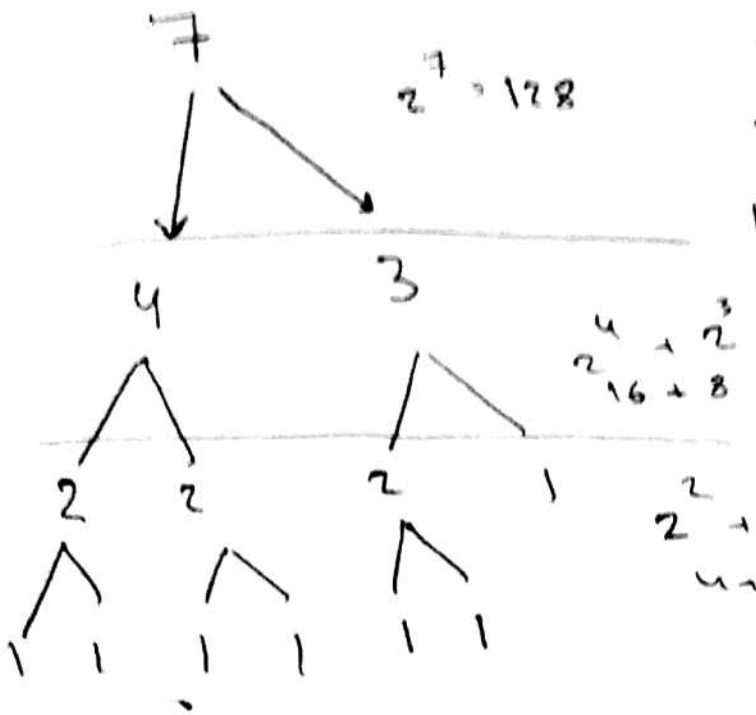
00 D_{12} A_3 A_2 $\overline{A}_1$ $\overline{A}_0$



مثلاً باستخدام سلك واحد
بدل سلكين في (2) and
بقلل cost

Decoder Expansion

ببني decoder بـ cost اقل باستخدام



$2^7 = 128$

128 and gat

24 and gat

128 and gat

164

$2^4 + 2^3 = 24$

$16 + 8$

$2^2 + 2^2 + 2^2 =$

$4 + 4 + 4 = 12$

cost

$(164 * 2) + 7 = 335$

street ford method

$(128 * 7) + 7$

nabling circuits to the outputs

able below for function

nation containing two X's represent four binary combinations

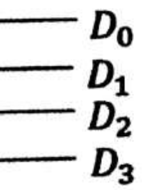
ely, can be viewed as distributing value of signal EN to 1 of 4

case, it is called a *Demultiplexer*

disabled decoder ابتدي

EN → 0 كلو zeros بغض النظر عن input

EN → 1 بشتغل decoder عادي

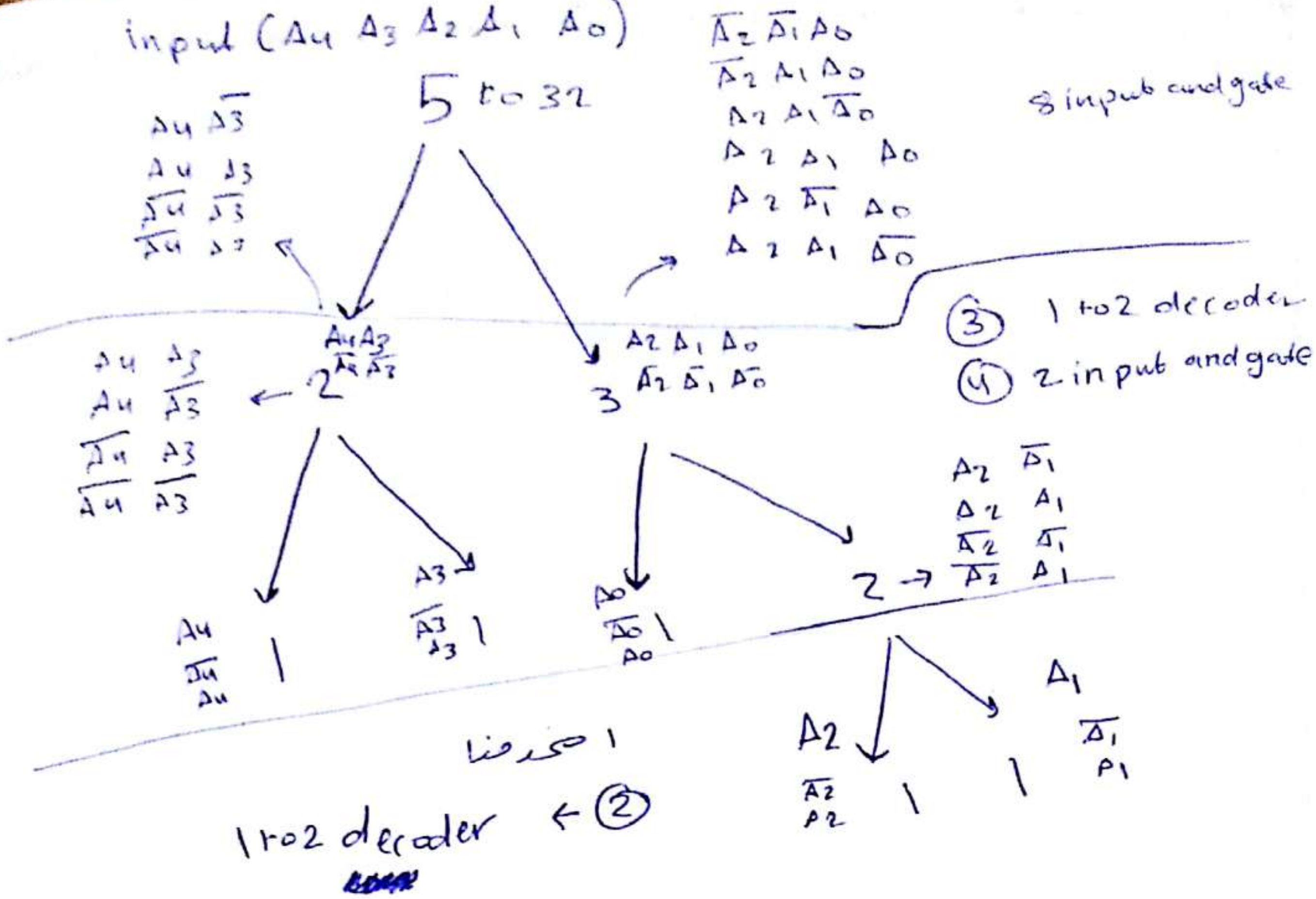


EN	A_1	A_0	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

(a)

← بعطي 0 كلو

→ بشتغل عادي



Cost

$(5 * 32) + 5 = 165$

* اول شي بقسم 5 على 2 = 2.5
* ثاني شي بجبر واحد وبخلي (3) وبعتبل الضل من ثاني وبخلي 2
* بقسم 2÷2 = 1
* بقسم 3÷2 وبجبر واحد (1.5) بخلي 2 وثاني لعتبل الضل وبخلي 1

---

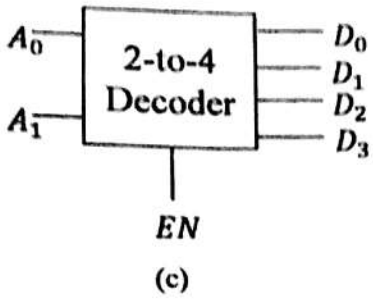
* وين ما اشوف 1 معناها أنا بكون مستعمل 1to2 decoder

الهدف منو عشان يعطيني varabel ونعكسو

* لما اشوف 2 بجمع كل var مع بعض الي جنبو (and)

gab and 4 2 يعني

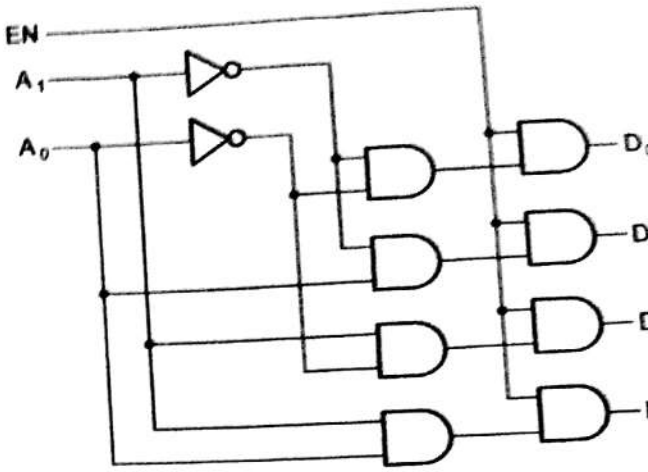
- Attach ***4-enabling*** circuits to the outputs
- See truth table below for function
  - Combination containing two X's represent four binary combinations
- Alternatively, can be viewed as distributing value of signal EN to 1 of 4 outputs
  - In this case, it is called a *Demultiplexer*



(c)

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

(a)



(b)

2-to-4 Decoder using 1-to-2 Decoders and Inverters

A_1
Zero ← 0
1 ← 0
2 ← 1
3 ← 1

A_0
0
1
0
1

D_0	D_1
1	0
0	1
0	0
0	0

1st 1-to-2 Decoder
مسؤول عن D_0D_1

D_2	D_3
0 zero	0
0 كلهم	0
1	0
0	1

2nd 1-to-2 Decoder
مسؤول عن D_2D_3

كل line يكون عليه 1 حسب رقم صفار

1 to 2 لازم 2 خرج

اذن بجيب تنين 1 to 2

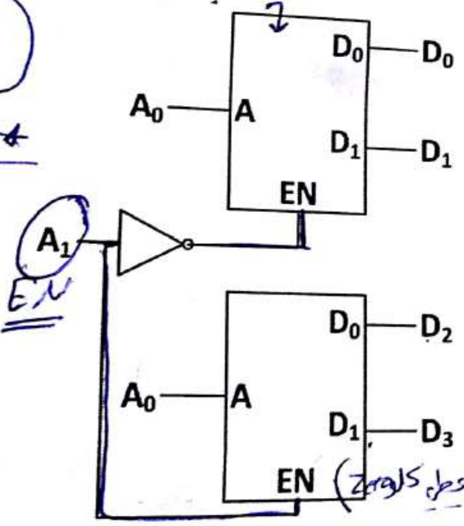
... 2 Decoders and Inverters

Most → ادن هي EN

A_1	A_0	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

1^{st} 1-to-2 Decoder (D_0, D_1) — 2^{nd} 1-to-2 Decoder (D_2, D_3)

(zero) (zero)



* ما بنعطيها حرف signal بدنا نخدم EN عشان واحد يكون EN والباقي disable

عشان واحد فيهم يكون (1) والباقي zero

(1) Most لازم استخدم signal S والباقي لا تكون A1 zero

1 ← راح يشتغل
2 ← ما راح يشتغل (2)

لا يكون A1 (1) ← ما راح يشتغل (راح يعطي كلها zero)
2 ← راح يشتغل

لازم الحرف متى تعطي عشان أصفر

... 2 to 4 Decoders and Inverters

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
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
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 2-to4 Decoder   2nd 2-to4 Decoder

Most A_2 → MSV Most significant varb

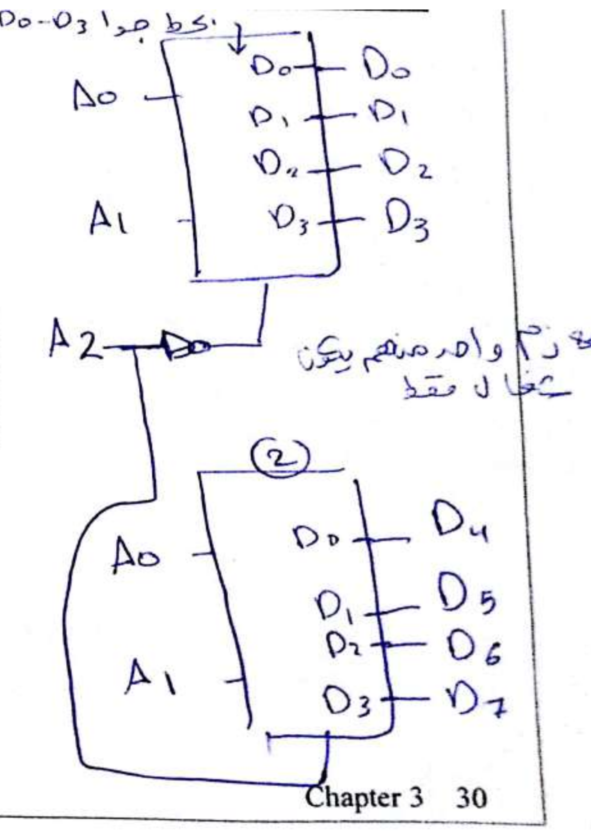
A_2 Most اذن هي EN

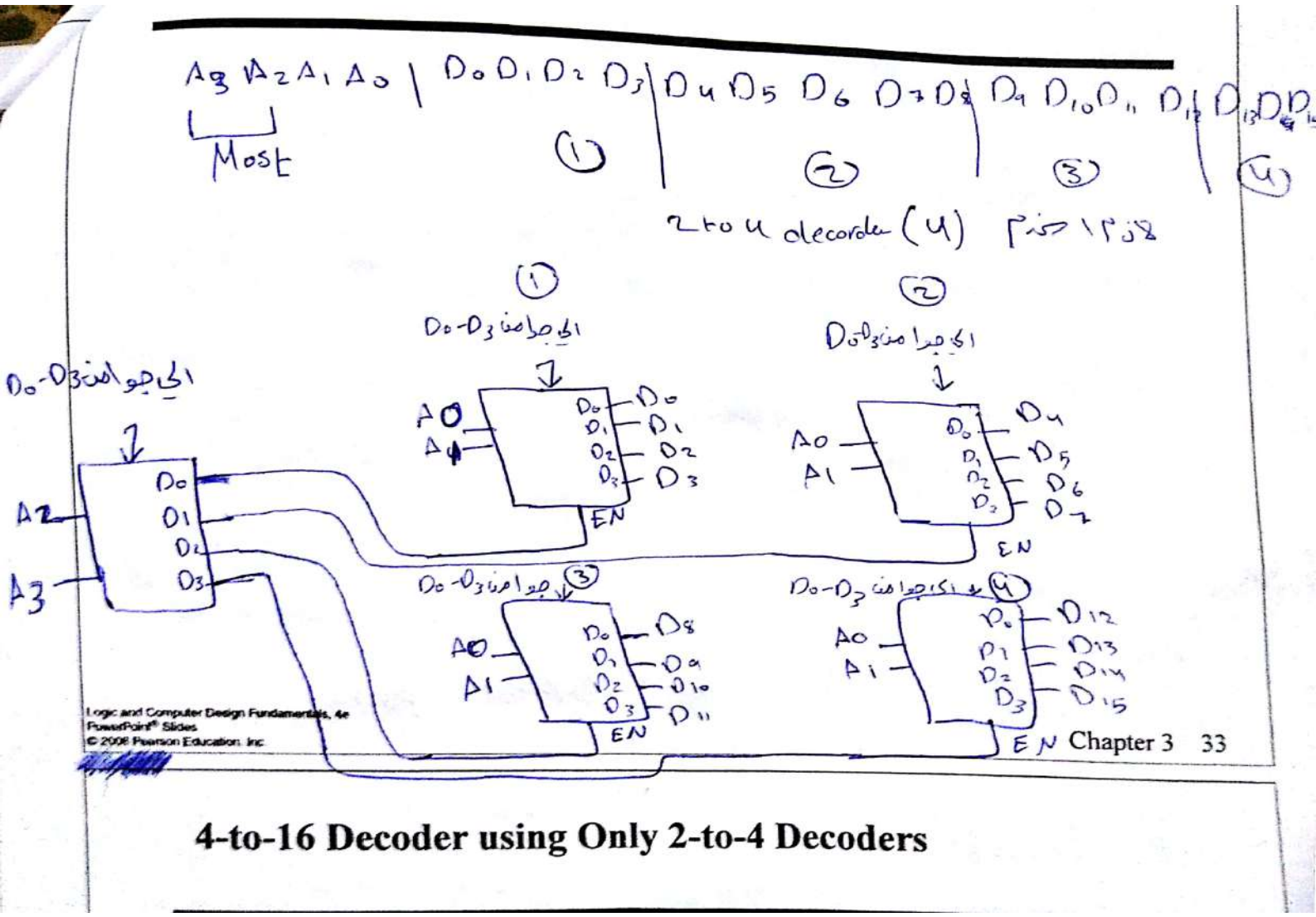
(1) → $D_0 D_1 D_2 D_3$ مسؤولة عن

(2) → $D_4 D_5 D_6 D_7$ مسؤولة عن

بجدين بشبك A_2 لل EN من تحت بعين

و بربط بواحدة منهم inverter





Logic and Computer Design Fundamentals, 4e
PowerPoint® Slides
© 2008 Pearson Education, Inc.

4-to-16 Decoder using Only 2-to-4 Decoders

- Sum-of-minterms expr...
- One n-to-2^n-line decoder
- m OR gates, one for each function
- For each function, the OR gate has k inputs, where k is the number of minterms in the function

- **Approach 1:**
  - Find the truth table for the functions
  - Make a connection to the corresponding OR from the corresponding decoder output wherever a 1 appears in the truth table

- **Approach 2**
  - Find the minterms for each output function
  - OR the minterms together

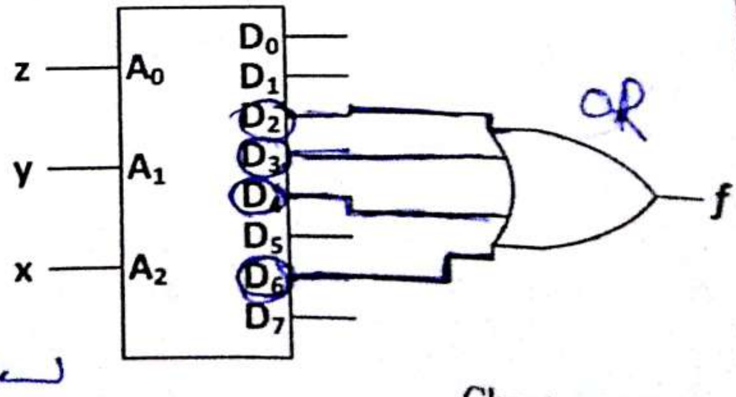
Example1

- Implement function f using decoder and OR gate:

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

- $n = 3$ variables → ***3-to-8 decoder***
- One function → ***One OR gate***
- Solution: Convert f to SOM format
  - $f = x\bar{z}(y+\bar{y}) + \bar{x}y(z+\bar{z}) = xy\bar{z} + x\bar{y}\bar{z} + \bar{x}yz + \bar{x}y\bar{z}$
  - $f(x,y,z) = \sum_m(2,3,4,6)$ → ***4-input OR gate***

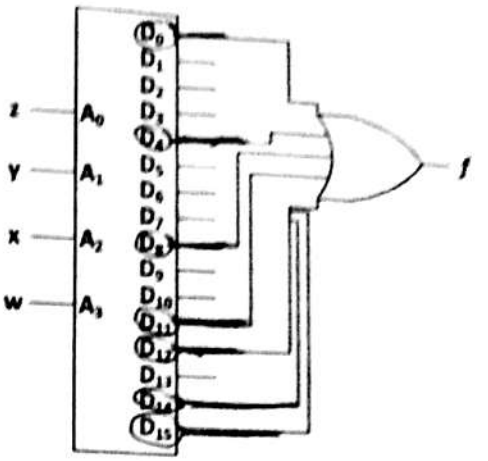
- ***Decoder is a Minterm Generator***



- Implement function f using decoder and OR gate:

$$f(w,x,y,z) = \sum_m (0,4,8,11,12,14,15)$$

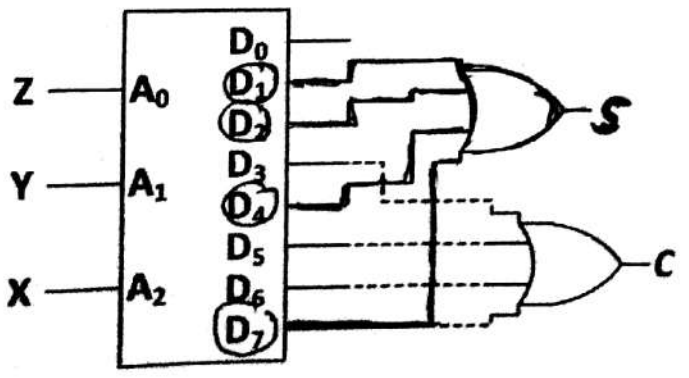
- $n = 4$ variables → *(4-to-16 decoder)*
- One function with 7 minterms → ***One 7-input OR gate***
- ***If number of minterms is greater than $\frac{2^n}{2}$, then design for complement F ($\bar{F}$) and use NOR gate instead of OR to generate F***



Example3

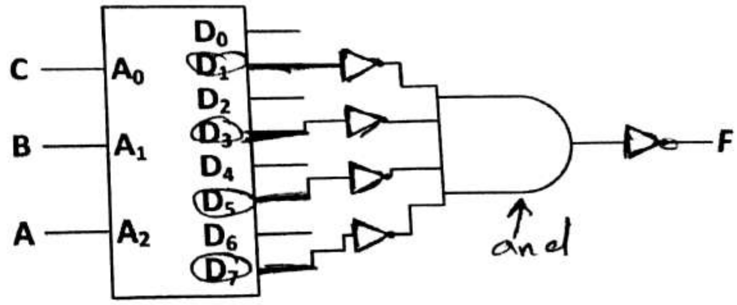
- Implement functions *C and S* using decoder and OR gates:
- $n = 3$ variables → ***3-to-8 decoder***
- Two function → ***Two OR gates***
- Solution:
  - $C = \sum_m(3,5,6,7)$ → ***4-input OR gate***
  - $S = \sum_m(1,2,4,7)$ → ***4-input OR gate***

X	Y	Z	C	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

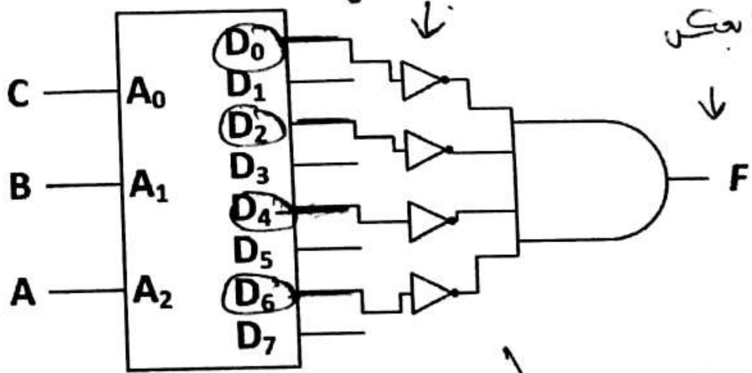


Example5

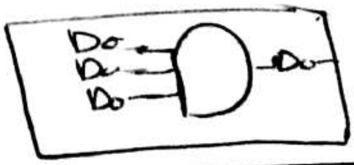
- Implement function F using 3-to-8 decoder, AND gate and inverters: $F(A,B,C) = \sum_m(1,3,5,7)$
- Solution with 5 inverters:



- Solution with 4 inverters:
  - $F(A,B,C) = \prod_M(0,2,4,6)$



(input + output) and



Scanned by CamScanner

- **Implement the following set of odd parity functions**

  (A_7, A_6, A_5, A_4)

  $P_1 = A_7 \oplus A_5 \oplus A_4$

  $P_2 = A_7 \oplus A_6 \oplus A_4$

  $P_3 = A_7 \oplus A_6 \oplus A_5$

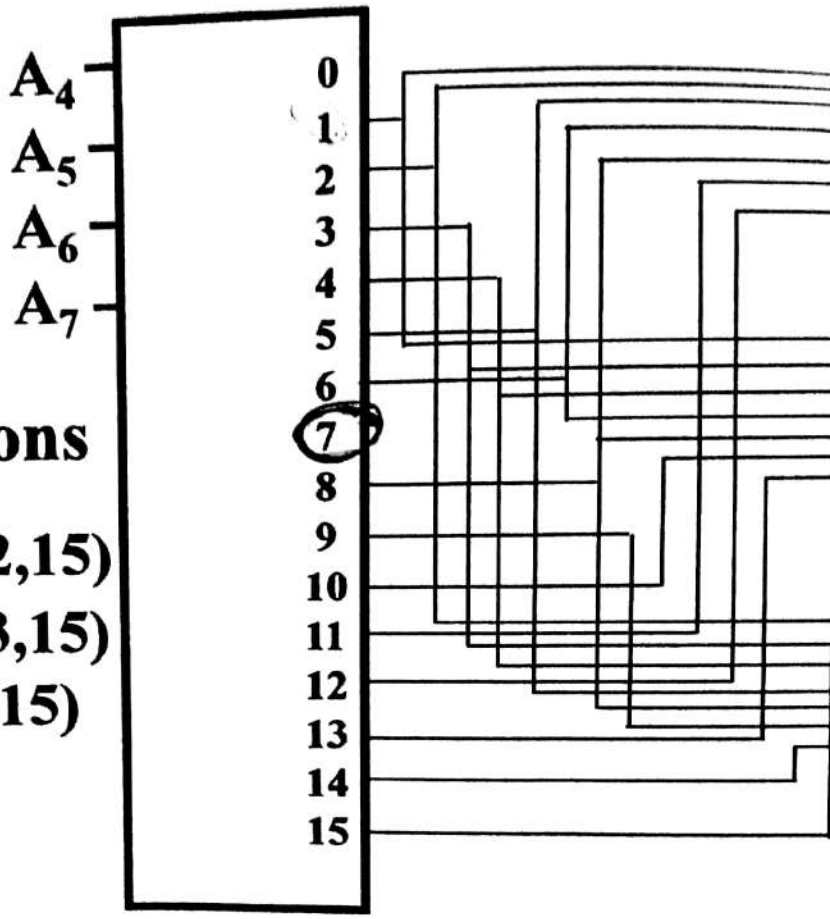
- **Finding sum of minterms expressions**

  $P_1 = \Sigma_m(1,2,5,6,8,11,12,15)$

  $P_2 = \Sigma_m(1,3,4,6,8,10,13,15)$

  $P_3 = \Sigma_m(2,3,4,5,8,9,14,15)$

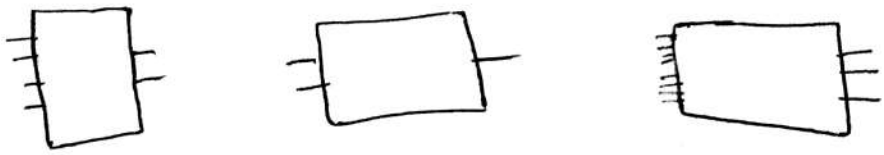
- **Find circuit**
- **Is this a good idea?**



4to16

Encoding

- ***Encoding:*** the opposite of decoding - the conversion of an m-bit input code to a n-bit output code with $n \leq m \leq 2^n$ such that each valid code word produces a unique output code
- Circuits that perform encoding are called ***encoders***
- An encoder has 2^n (or fewer) input lines and n output lines which ***generate the binary code corresponding to the input values***
- Typically, an encoder converts a code containing exactly one bit that is 1 to a binary code corresponding to the position in which the 1 appears



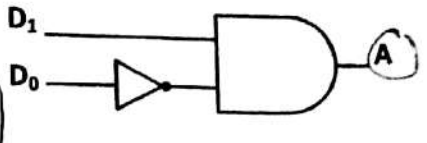
في شروط
① لحتوى يكون عندى أكثر من 1
(لو كان داخ ندرسها بعدين)

كيف يشتغل
① بطلع على input سوف line
out اى عليه 1 ، بطلعلي رقم على

عشان يشتغل هيكا

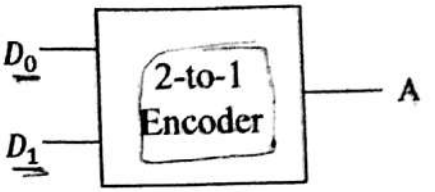
D_1	D_0	A
0	0	Invalid Input
0	1	0
1	0	1
1	1	Invalid Input

(a)



$A = D_1 . \overline{D_0}$

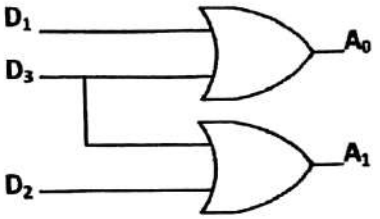
(b)



(c)

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

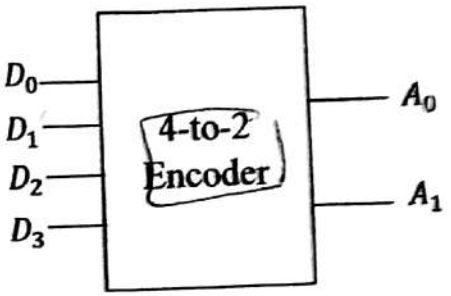
(a)



$A_0 = D_1 + D_3$

$A_1 = D_2 + D_3$

(b)

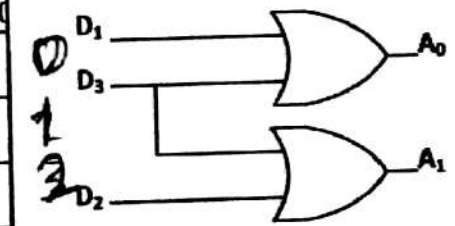


(c)

(c)

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

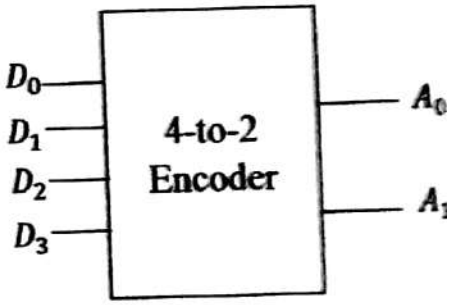
(a)



$A_0 = D_1 + D_3$

$A_1 = D_2 + D_3$

(b)



(c)

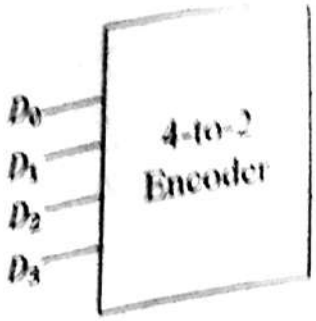
D_3	D_2	D_1	D_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	0	1	0
0	1	0	0	1	0
1	0	0	0	1	1

(a)

$A_0 = D_1 + D_3$

$A_1 = D_2 + D_3$

(b)



(c)

A_0 (واحد منها) $\rightarrow D_1$ $\rightarrow D_3$ $D_1 + D_3$

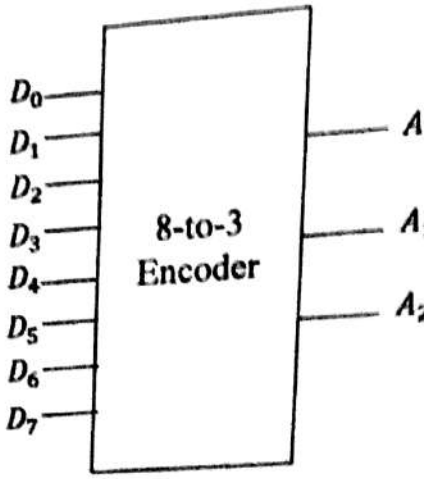
A_1 D_2 D_3 $D_2 + D_3$

Chapter 3 47

8-to-3 Encoder (Octal-to-Binary Encoder)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

(a)



$A_0 = D_1 + D_3 + D_5 + D_7$

$A_1 = D_2 + D_3 + D_6 + D_7$

$A_2 = D_4 + D_5 + D_6 + D_7$

(b)

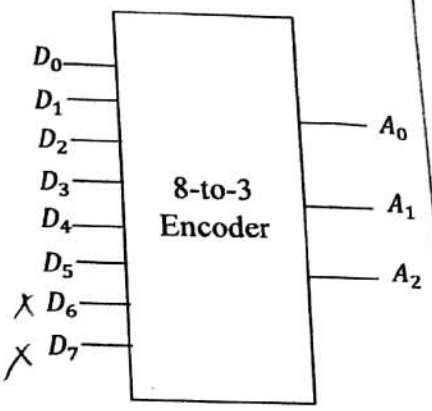
(c) مثال واحد كامل

A_0 (واحد منها) $\rightarrow D_1$ D_3 D_5 D_7

Chapter 3 48

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

(a)



(c)

$$A_0 = D_1 + D_3 + D_5 + D_7$$
$$A_1 = D_2 + D_3 + D_6 + D_7$$
$$A_2 = D_4 + D_5 + D_6 + D_7$$

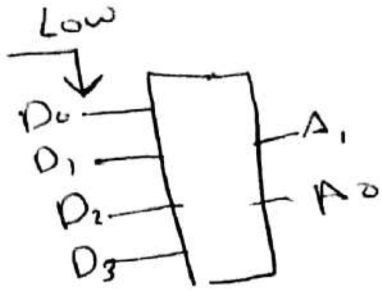
(b)

Decimal-to-BCD Encoder

- ***Inputs:*** 10 bits corresponding to decimal digits 0 through 9, (D_0, …, D_9)
- ***Outputs:*** 4 bits with BCD codes (A_3, A_2, A_1, A_0)
- ***Function:*** If input bit D_i is a 1, then the output is the BCD code for i
- The truth table could be formed, but alternatively, the equations for each of the four outputs can be obtained directly

4-to-2 Low Priority Encoder

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



فكرة Low
بطلع من فوق
لازم الاقي واحد
بطريقي من فوق
و 1

تا معون لازم أكون الي بعطي 1 و بعطي صفر

$A_1 = D_2\overline{D_1}\,\overline{D_0} + D_3\overline{D_2}\,\overline{D_1}\,\overline{D_0}$

$A_0 = D_1\overline{D_0} + D_3\overline{D_2}\,\overline{D_1}\,\overline{D_0}$

مثلا لازم

D_0 0, D_1 0, D_2 1, D_3 X

* اذا كان عند 2 1 لازم D_1 D_0 صفر

هاد don't care

4-to-2 Low Priority Encoder

#_of_Minterms/ Rows	D_3	D_2	D_1	D_0	A_1	A_0	V
1	0	0	0	0	X	X	0
8	X	X	X	1			
4	X	X	1	0			
2	X	1	0	0			
1	1	0	0	0			

(a)

$A_0 = D_1\overline{D_0} + D_3\overline{D_2}\,\overline{D_1}\,\overline{D_0}$

$A_0 = \overline{D_0}(D_1 + D_3\overline{D_2}\,\overline{D_1})$

معون $A_0 = \overline{D_0}(D_1 + D_3\overline{D_2})$

$\rightarrow A_0 = D_1\overline{D_0} + D_3\overline{D_2}\,\overline{D_0}$

مختصر

مو ضروري $A_1 = D_2\overline{D_1}\,\overline{D_0} + D_3\overline{D_2}\,\overline{D_1}\,\overline{D_0}$

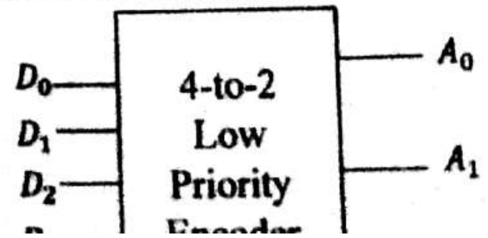
$A_1 = \overline{D_1}\,\overline{D_0}(D_2 + D_3\overline{D_2})$

$A_1 = \overline{D_1}\,\overline{D_0}(D_2 + D_3)$

$A_1 = D_2\overline{D_1}\,\overline{D_0} + D_3\overline{D_1}\,\overline{D_0}$

$V = D_3 + D_2 + D_1 + D_0$

(b)



4-to-2 High Priority Encoder

#_of_Minterms/ Rows	D_3	D_2	D_1	D_0	A_1	A_0	V
1	0	0	0	0	X	X	0
1	0	0	0	1	0	0	1
2	0	0	1	X	0	1	1
4	0	1	X	X	1	0	1
8	1	X	X	X	1	1	1

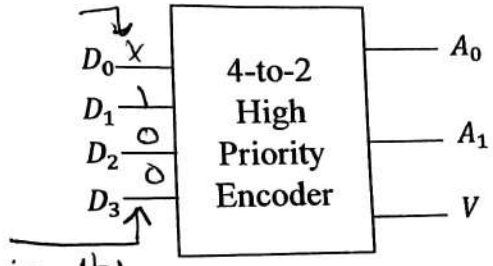
(a)

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

$$A_1 = D_3 + \overline{D_3}D_2$$
$$A_1 = D_3 + D_2$$

$$V = D_3 + D_2 + D_1 + D_0$$

(b)



(c)

5-input Priority Encoder

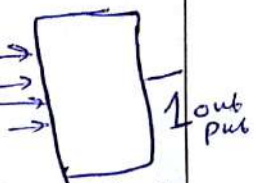
- Priority encoder with 5 inputs (D_4, D_3, D_2, D_1, D_0) - highest priority to most significant 1 present - Code outputs A_2, A_1, A_0 and V where V indicates at least one 1 present

No. of Min-terms/Row	Inputs					Outputs			
	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0	V
1	0	0	0	0	0				
1	0	0	0	0	1				
2	0	0	0	1	X				
4	0	0	1	X	X				
8	0	1	X	X	X				
16	1	X	X	X	X				

- X's in input part of table represent 0 or 1; thus table entries correspond to product terms instead of minterms. The column on the left shows that all 32 product terms in the table

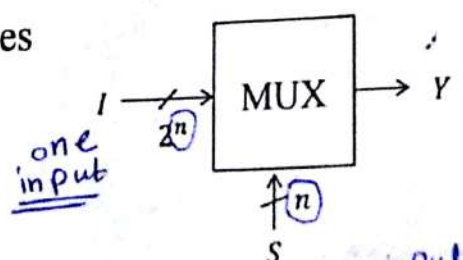
Selecting

- Selecting of data or information is a critical function in digital systems and computers
- Circuits that perform selecting have:
  - A set of information inputs from which the selection is made
  - A single output
  - A set of control lines for making the selection
- Logic circuits that perform selecting are called ***multiplexers***
- Selecting can also be done by three-state logic



Multiplexers (MUX) (Data Selectors)

- A multiplexer selects information from an input line and directs the information to an output line
- A typical multiplexer has **n control inputs** (S_{n-1}, … S_0) called ***selection inputs***, **2^n information inputs** (I_{2^n-1}, … I_0), and **one output Y**
- A multiplexer can be designed to have m information inputs with **$m < 2^n$** as well as n selection inputs
- Multiplexers allow sharing of resources and reduce the cost by reducing the number of wires



2-to-1-Line MUX

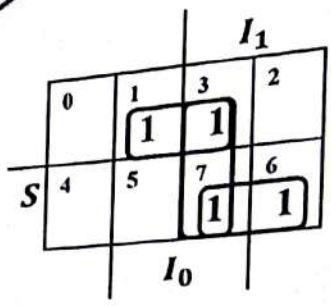
- Since $2 = 2^1$, n = 1
- The single selection variable S has two values:
  - S = 0 selects input I_0
  - S = 1 selects input I_1
- The equation:

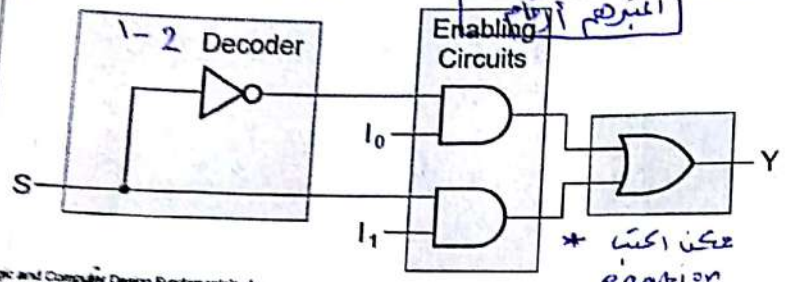
$$Y = \bar{S} I_0 + S I_1$$

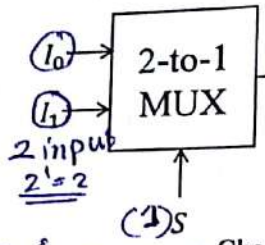
- The circuit:

	S	I_1	I_0	Y	
0	0	0	0	0	$Y = I_0$
1	0	0	1	1	
2	0	1	0	0	
3	0	1	1	1	
4	1	0	0	0	$Y = I_1$
5	1	0	1	0	
6	1	1	0	1	
7	1	1	1	1	

S	Y
0	I_0
1	I_1





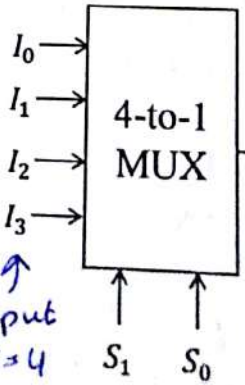


4-to-1-Line MUX

- Since $4 = 2^2$, n = 2
- There are two selection variables ($S_1 S_0$) and they have four values:
  - $S_1 S_0 = 00$ selects input I_0
  - $S_1 S_0 = 01$ selects input I_1
  - $S_1 S_0 = 10$ selects input I_2
  - $S_1 S_0 = 11$ selects input I_3
- The equation:

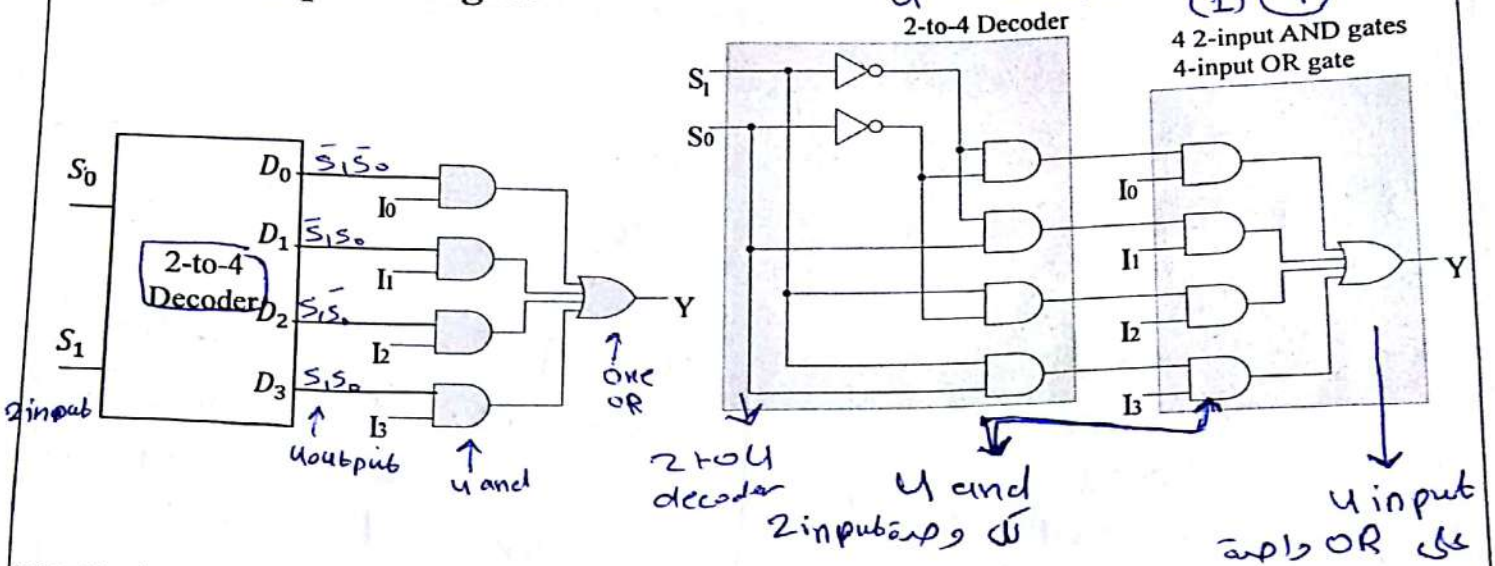
$$Y = \overline{S_1}\,\overline{S_0} I_0 + \overline{S_1} S_0 I_1 + S_1 \overline{S_0} I_2 + S_1 S_0 I_3$$

	S_1	S_0	Y
0	0	0	I_0
1	0	1	I_1
2	1	0	I_2
3	1	1	I_3



- 2-to-4-line decoder
- 4 2-input AND gates
- 4-input OR gate

decoder n to 2^n
2 to 4
and 2^n 2input
4 2-input
OR
(1) 2^n input
(1) (4)



2-to-1-Line MUX Cont.

- Note the regions of the multiplexer circuit shown: انا بحاجة لـ:
  - 1-to-2-line Decoder
  - 2 Enabling circuits
  - 2-input OR gate

- *In general, for an 2^n-to-1-line multiplexer:*
  - *n-to-2^n-line decoder*
  - *2^n 2-input AND gate*
  - *One 2^n-input OR gate*

line decoder $n - 2^n$

OR
(1) 2^n input
AND
(2^n) 2 input

Homework

- **Implement 8-to-1-Line MUX and 64-to-1 MUX:**
  - **How many select lines are needed?** 3 / 6
  - **Decoder size?** 6-to 64
  - **How many 2-input AND gates are needed?** 64
  - **What is the size of the OR gate?**

(32) → 1
select line 5
5 to 32 decoder
32 2-and
1 or 2^5 input

8 → 1
select line → 3
3-8 decoder
8 2-input and
1 OR 8 input

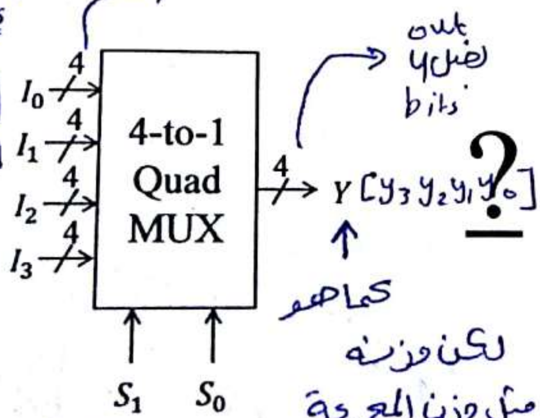
64 to 1
select line 6
6-64 decoder
6 2-input and
1 OR 64 input

Multiplexer Width Expansion

- Select "vectors of bits" instead of "bits"
- Example: *4-to-1-line quad multiplexer*

*كأنك طالب input عبارة عن اشخاص مثلا I_0 I_1

* هلأ صار جروب I_0 جروب 1، I_1 جروب 2



*بدل ما كان bit واحد (صار) مجموعة bits

* Selection S_0 و S_1 ما بتتغير

يعني بنادي على جروب ما يهمني

كم وزنه 2bits او اكثر يهمني

اسمه [عدد يضل ثابت ما بتغير]

اذا 2bits هو 2bits وهكذا

4-1 multiplexe

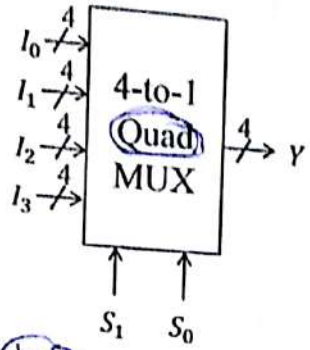
بس لما خليت input عبارة عن

صار اسمه dual كل جروب عبارة عن 2bit

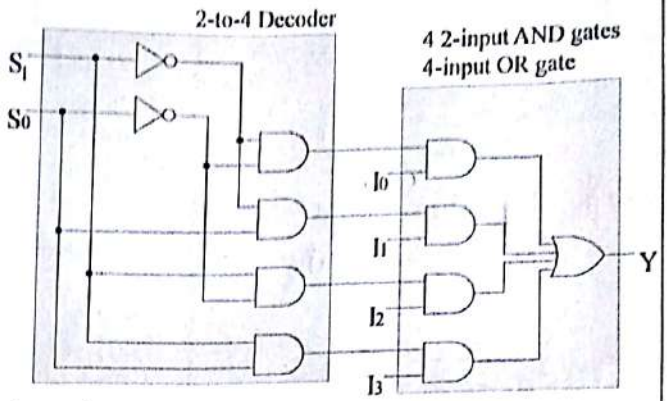
Multiplexer Width Expansion

- Select "vectors of bits" instead of "bits"
- Example: ***4-to-1-line quad multiplexer***

Quad 4 رباعي
Dual 2 ثنائي



?



(m) bit wide 2^n to 1 Mux

(1) n to 2^n line decoder ⟹ فك الترميز

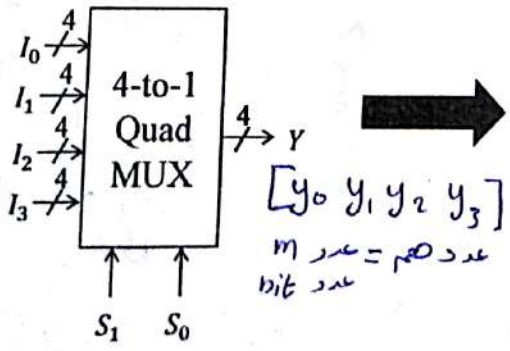
(2) m 2^n 2 input and gate ⟹ $m+2^n$ عدد البوابات

(3) m 2^n-input OR gate ⟹ تجميع البوابات

1 ⟹ m bits

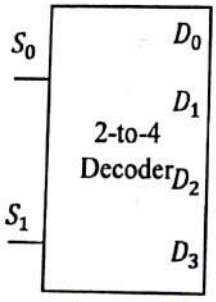
Multiplexer Width Expansion

- Select "vectors of bits" instead of "bits"
- Example: ***4-to-1-line quad multiplexer***



$[y_0\ y_1\ y_2\ y_3]$

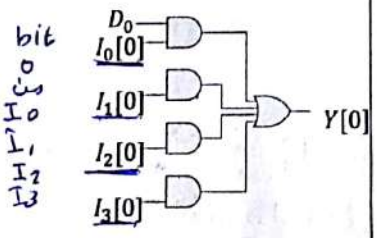
m عدد = عدد الـ bit



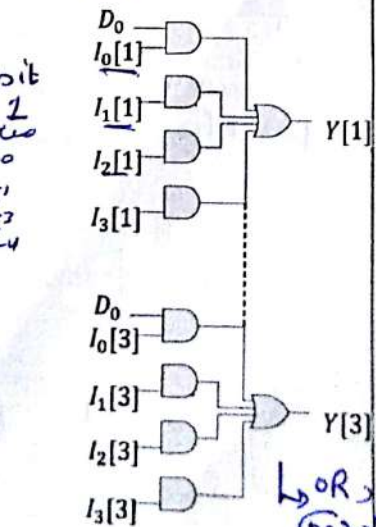
n to 2^n

2 to 4

bit 0 من I_0 I_1 I_2 I_3



bit 1 من I_0 I_1 I_2 I_4



analgat عدد

$m + 2^n$

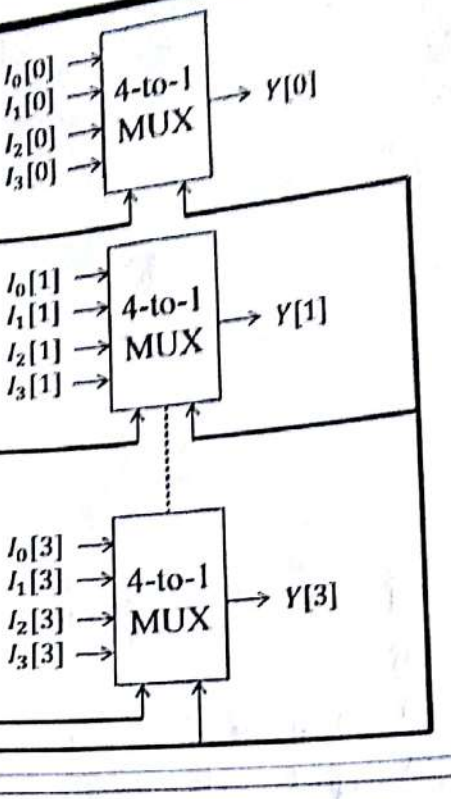
$4 + 4 = 16$

L عدد OR

(m) (4)

Multiplexer Width Expansion Cont.

- Can be thought of as four 4-to-1 MUXes:



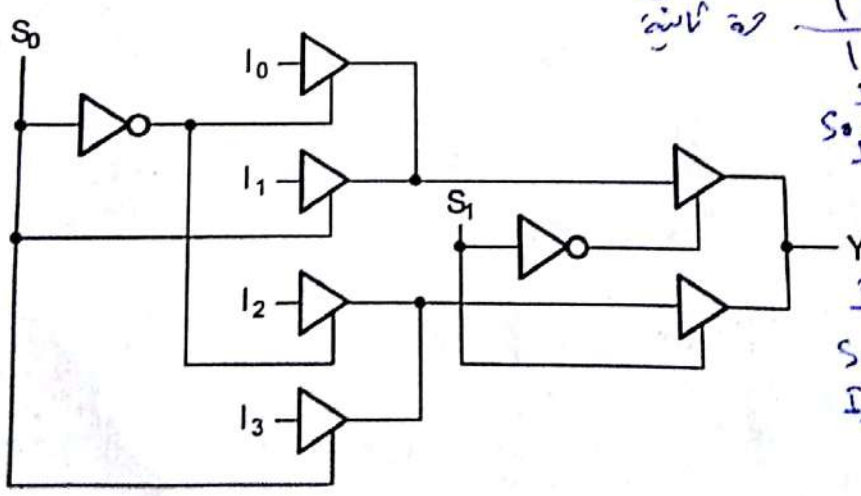
يمكن اعتبارها بطريقة اخرى 1bit width و عندي Mux بـ 4 لكن S1 S0 ثابتين

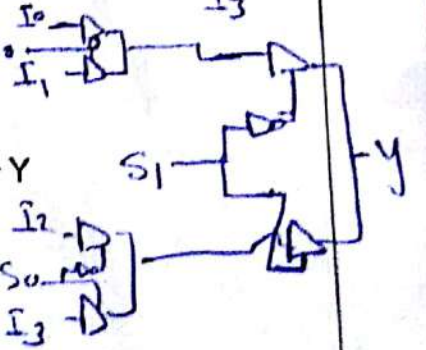
لازم نفس

Other Selection Implementations

Three-state logic

S_1	S_0	Y	
0	0	I_0	كود ثنائي
0	1	I_1	اول مرة
1	0	I_2	مرة ثانية
1	1	I_3	



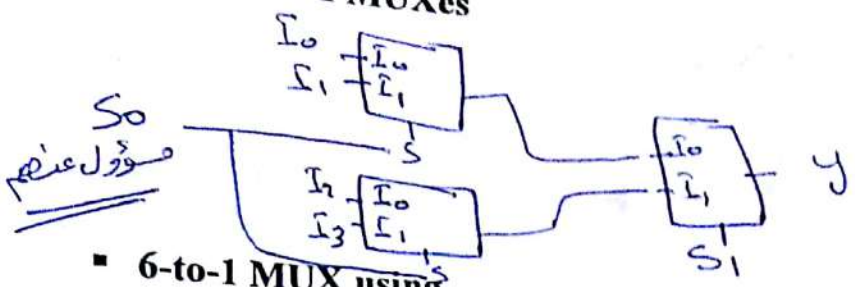


* قسمت أول مرة بناءً على S_1
* قسمت ثاني مرة بناءً على S_0

$I_0\ I_1\ I_2\ I_3 \Rightarrow$ 4 inputs 4 to 1 Mux

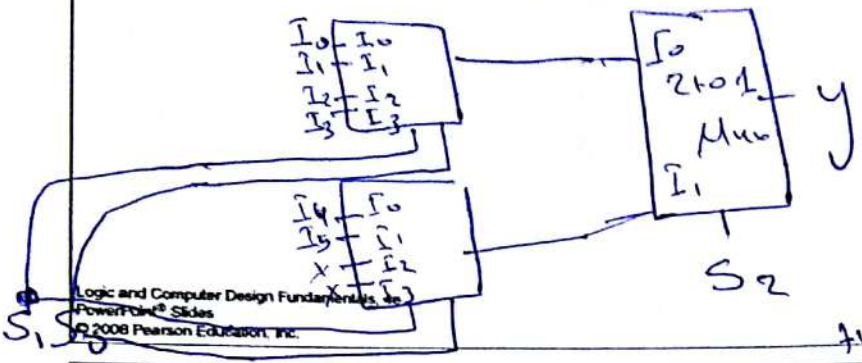
$S_0\ S_1 \Rightarrow$ 2 select line

- **4-to-1 MUX using three 2-to-1 MUXes**



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

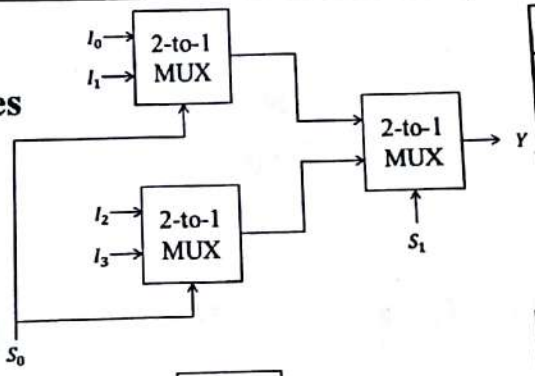
- **6-to-1 MUX using two 4-to-1 MUXes and one 2-to-1 MUX**



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

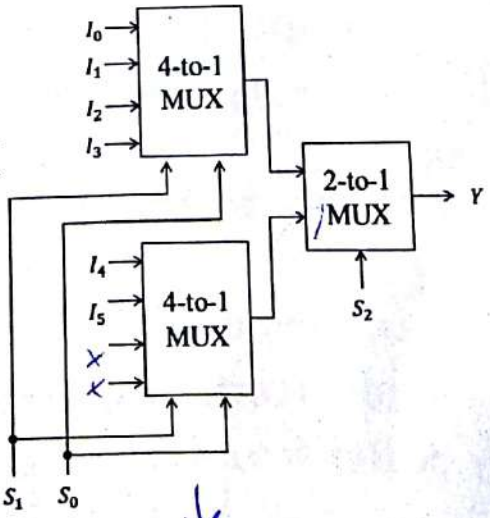
Building Large MUXes from Smaller Ones

- **4-to-1 MUX using three 2-to-1 MUXes**



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

- **6-to-1 MUX using two 4-to-1 MUXes and one 2-to-1 MUX**



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

Homework

- **Build an 8-to-1 MUX using:**
  - **Two 4-to-1 MUX and one 2-to-1 MUX**
  - **One 4-to-1 MUX and multiple 2-to-1 MUXes**
  - **Only 2-to-1 MUXes (How many MUXes are need?)**

Combinational Logic Implementation - Multiplexer Approach 1

- **Implement *m* functions of *n* variables with:**
  - **Sum-of-minterms expressions**
  - **An *m*-wide 2^n-to-1-line multiplexer**
- **Design:**
  - **Find the truth table for the functions**
  - **In the order they appear in the truth table:**
    - **Apply the *function input variables* to the multiplexer *select inputs* $S_{n-1}, \ldots, S_0$**
    - **Label the outputs of the multiplexer with the output variables**
  - **Value-fix the information inputs to the multiplexer using the values from the truth table (for don't cares, apply either 0 or 1)**

m function n variables

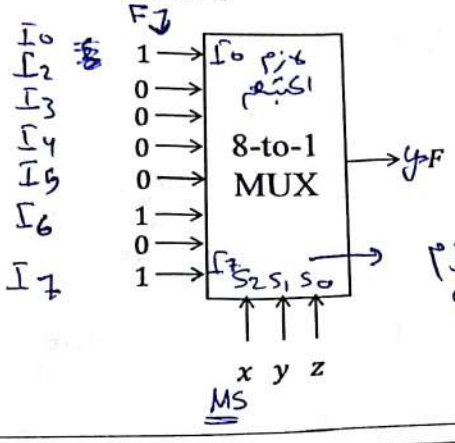
① m-bit 2^n to 1 max

② m-bit 2^{n-1} to 1 max + Inverter

Example1

- **Implement the following function using a single MUX based on Approach1 : $F(x, y, z) = \sum_m(0, 5, 7)$**
- Solution:
  - Single function → m = 1
  - 3 variables → n = 3 → 8-to-1 MUX
  - Fill the truth table of F

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



1 bit 3 vars
2^n to 1 8 to 1 MUX

Example2: Gray to Binary Code

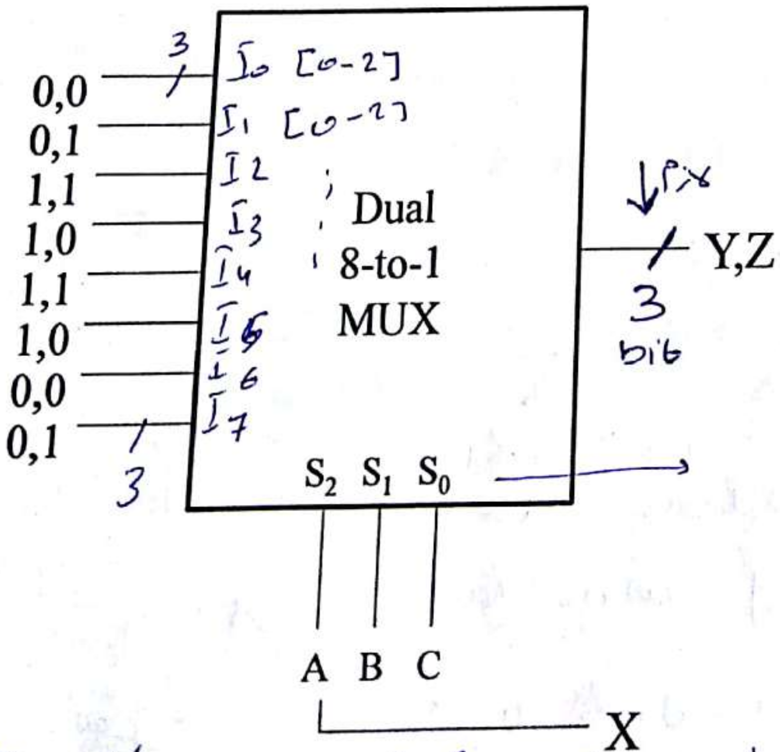
- Design a circuit to convert a 3-bit Gray code to a binary code
- The formulation gives the truth table on the right

Gray Code ABC	Binary Code XYZ
000	000
001	001
011	010
010	011
110	100
111	101
101	110
100	111

- Rearrange the table so that the input combinations are in counting order
- It is obvious from this table that X = A. However, Y and Z are more complex
- Two functions (Y and Z) → m = 2
- 3 variables (A, B, and C) → n = 3
- Functions Y and Z can be implemented using a dual 8-to-1-line multiplexer by:
  - connecting A, B, and C to the multiplexer select inputs
  - placing Y and Z on the two multiplexer outputs
  - connecting their respective truth table values to the inputs

Gray Code ABC	Binary Code XYZ
000	000
001	001
010	011
011	010
100	111
101	110
110	100
111	101

Gray to Binary Code Cont.



Combinational Logic Implementation - Multiplexer Approach 2

- **Implement any *m* functions of *n* variables by using:**
  - **An m-wide $2^{(n-1)}$-to-1-line multiplexer**
  - **A single inverter if needed**
- **Design:**
  - **Find the truth table for the functions**
  - **Based on the values of the most significant *(n-1)* variables, separate the truth table rows into pairs**
  - **For each pair and output, define a rudimentary function of the least significant variable $(0, 1, X, \overline{X})$**
  - **Connect the most significant *(n-1)* variables to the select lines of the MUX, value-fix the information inputs to the multiplexer with the corresponding rudimentary functions**
  - **Use the inverter to generate the rudimentary function $\overline{X}$**

m function n variables
m bit 2^{n-1}-to-1 MUX
(+) m inverter

Example1

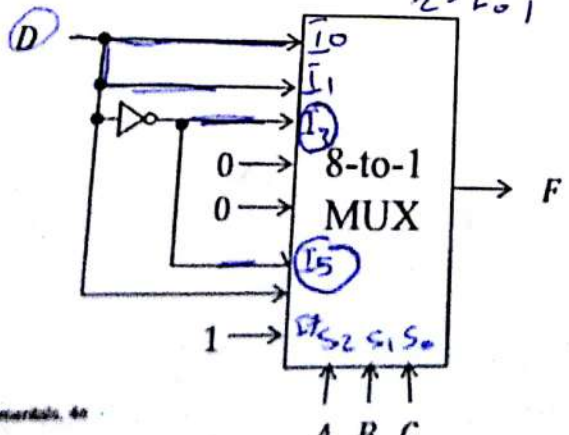
- **Implement the following function using a single MUX and an inverter (if needed) based on Approach2 :**

$$F(A, B, C, D) = \sum_m (1, 3, 4, 10, 13, 14, 15)$$

Most

1 function
4 varible

- Solution:
  - Single function → m = 1
  - 4 variables → n = 4 → ~~8-to-1 MUX~~
  - Fill the truth table of F 2^{4-1} to 1, 2^3 to 1



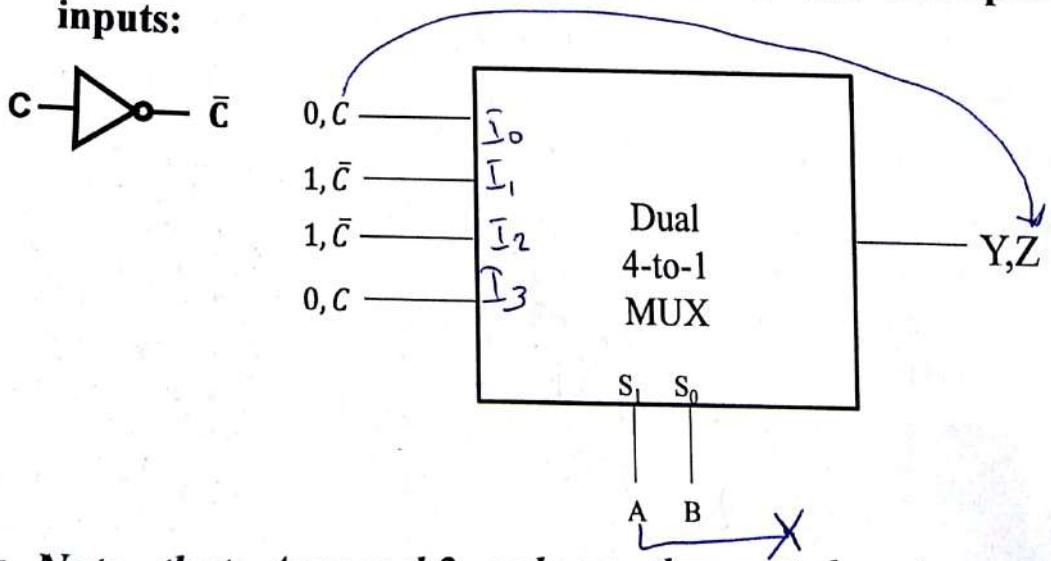
A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = \overline{D}$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	1	$F = \overline{D}$
1	0	1	1	0	
1	1	0	0	0	$F = D$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

Example2: Gray to Binary Code

Gray Code ABC	Binary Code XYZ	Rudimentary Functions of C for Y	Rudimentary Functions of C for Z
000	000	$Y = 0$	$Z = C$
001	001		
010	011	$Y = 1$	$Z = \bar{C}$
011	010		
100	111	$Y = 1$	$Z = \bar{C}$
101	110		
110	100	$Y = 0$	$Z = C$
111	101		

Gray to Binary Code Cont.

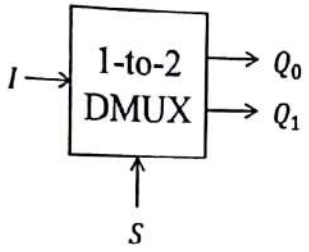
- **Assign the variables and functions to the multiplexer inputs:**



- ***Note that Approach2 reduces the cost by almost half compared to Approach1***

Demultiplexer (DMUX)

- Opposite of multiplexer
- Receives **one** input and directs it to one from 2^n outputs based on **n-select** lines
- Example: 1-to-2 DMUX

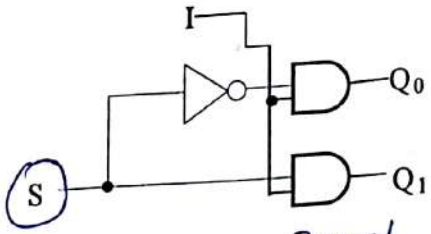


equation

$Q_0 = \bar{S}I$

$Q_1 = SI$

S	I	Q_1	Q_0
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

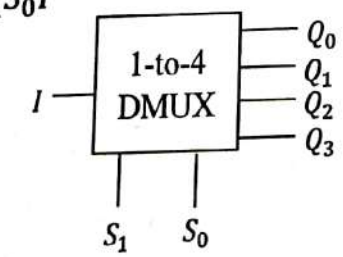


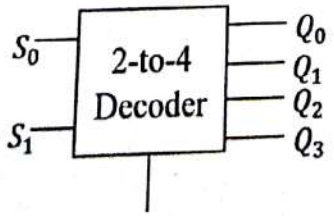
- ***DMUX ≡ Decoder with Enable***

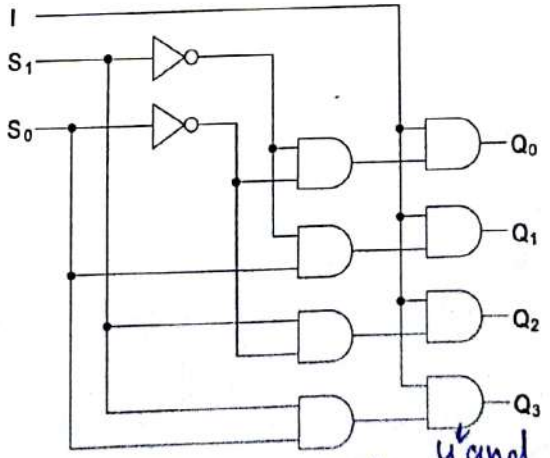
1-to-4 DMUX

- $Q_0 = \bar{S}_1 \bar{S}_0 I$
- $Q_1 = \bar{S}_1 S_0 I$
- $Q_2 = S_1 \bar{S}_0 I$
- $Q_3 = S_1 S_0 I$

	S_1	S_0	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	0	I
1	0	1	0	0	I	0
2	1	0	0	I	0	0
3	1	1	I	0	0	0







Functional Block: Half-Adder

- A 2-input, 1-bit width binary adder that performs the following computations:

X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

- A half adder adds two bits to produce a two-bit sum
- The sum is expressed as a **sum bit (S)** and a **carry bit (C)**
- The half adder can be specified as a truth table for S and C ⇒

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Logic Simplification and Implementation: Half-Adder

- The K-Map for S, C is:

S:

	$\bar{Y}$	Y
$\bar{X}$	$_0$	1 $_1$
X	1 $_2$	$_3$

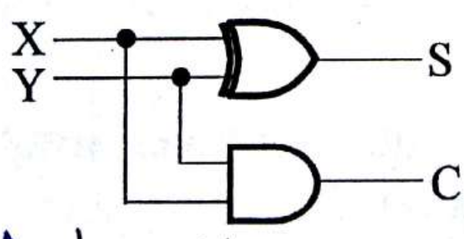
C:

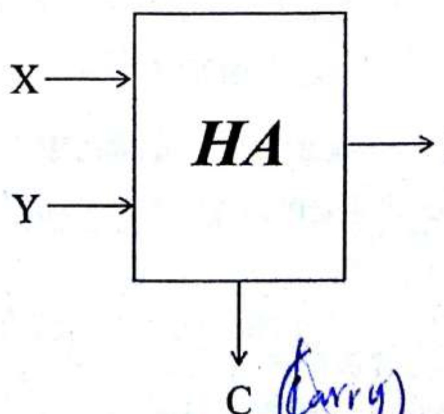
	$\bar{Y}$	Y
$\bar{X}$	$_0$	$_1$
X	$_2$	1 $_3$

$$S = X \cdot \overline{Y} + \overline{X} \cdot Y = X \oplus Y$$

$$C = X \cdot Y$$

- The most common half adder implementation is:



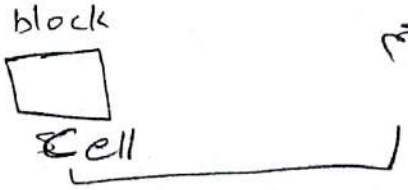


- Iterative combinational circuits
- Binary adders
  - Half and full adders
  - Ripple carry adders
- Binary subtraction
- Binary adder-subtractors
  - Signed binary numbers
  - Signed binary addition and subtraction
  - Overflow
- Binary multiplication → 2
- Other arithmetic functions
  - Design by contraction

بدي اتكلم عن العمليات الحسابية كيف بتصير على Binary number

Iterative Combinational Circuits

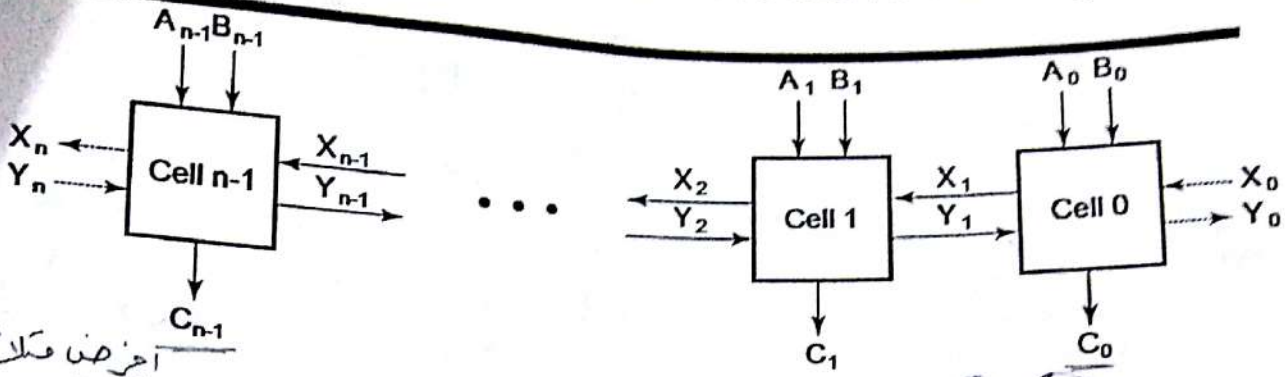
- Arithmetic functions
  - Operate on binary vectors
  - Use the same sub-function in each bit position
- Can design functional block for the sub-function and repeat to obtain functional block for overall function
- ***Cell:*** sub-function block



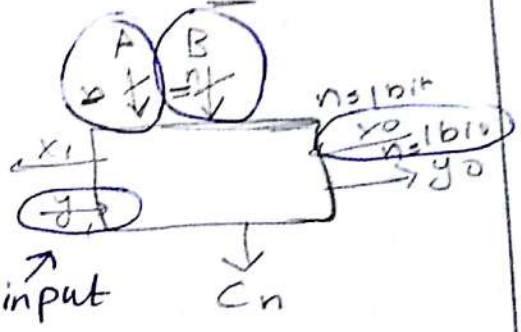
الصورة ↓ Iterative array

- ***Iterative array:*** array of interconnected cells

Block Diagram of an Iterative Array



- Example: n = 32
  - Number of inputs = 32*2 + 1 + 1 = 66
  - Truth table rows = 2^{66}
  - Equations with up to 66 input variables
  - Equations with huge number of terms
  - Design impractical!



- Iterative array takes advantage of the regularity to make design feasible

Functional Blocks: Addition

- Binary addition used frequently

- Addition Development:
  - ***Half-Adder (HA):*** a 2-input bit-wise addition functional block
  - ***Full-Adder (FA):*** a 3-input bit-wise addition functional block
  - ***Ripple Carry Adder:*** an iterative array to perform vector binary addition

Functional Block: Half-Adder (HA)

- A 2-input, 1-bit width binary adder that performs the following computations:

X	0	0	1	1
+Y	+0	+1	+0	+1
C S	0 0	0 1	0 1	1 0
	C S	C S	C S	C S

Sum — carry (الى بطلع على اذا بدة الى بعدها)

مثل تمثيل الـ Binary

- A half adder adds two bits to produce a two-bit sum
- The sum is expressed as a ***sum bit (S)*** and a ***carry bit (C)***
- The half adder can be specified as a truth table for S and C ⇒

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Logic Simplification and Implementation: Half-Adder ⇒

(جمع 2bit مع بعض و بطلعلي الجواب) [2bit جواز لـ HA]

- The K-Map for S, C is:

$$S = X \cdot \overline{Y} + \overline{X} \cdot Y = X \oplus Y$$

(بتوقف بتكون 1 بمكان)

$$C = X \cdot Y$$

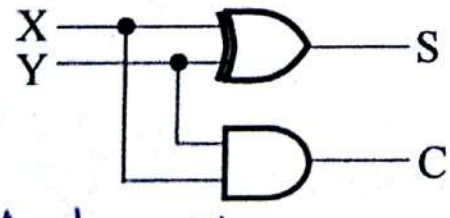
S:

X \ Y	0	1
0	0	1 (1)
1	1 (2)	3

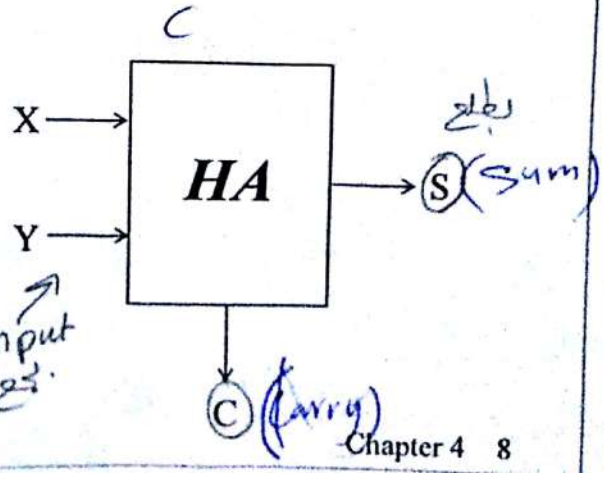
C:

X \ Y	0	1
0	0	1
1	2	1 (3)

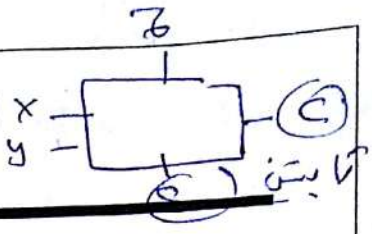
- The most common half adder implementation is:



هاد ديزاين جوا HA



Functional Block: Full-Adder



- A full adder is similar to a half adder, but includes a carry-in bit from lower stages. Like the half-adder, it computes a ***sum bit (S)*** and a ***carry bit (C)***

- For a carry-in (Z) of 0, it is the same as the half-adder:

Z	0	0	0	0
X	0	0	1	1
+Y	+0	+1	+0	+1
C S	0 0	0 1	0 1	1 0

- For a carry- in (Z) of 1:

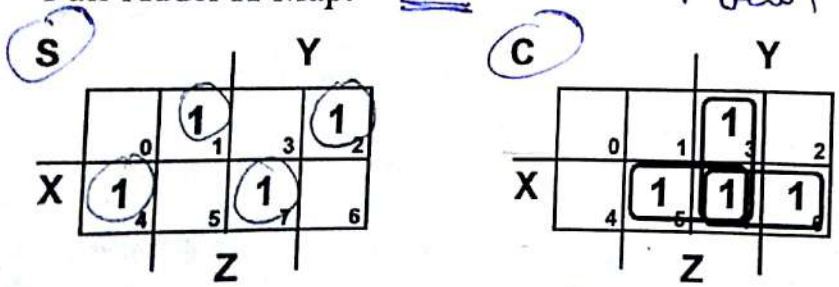
Z	1	1	1	1
X	0	0	1	1
+Y	+0	+1	+0	+1
C S	0 1	1 0	1 0	1 1

Logic Optimization: Full-Adder

- Full-Adder Truth Table:

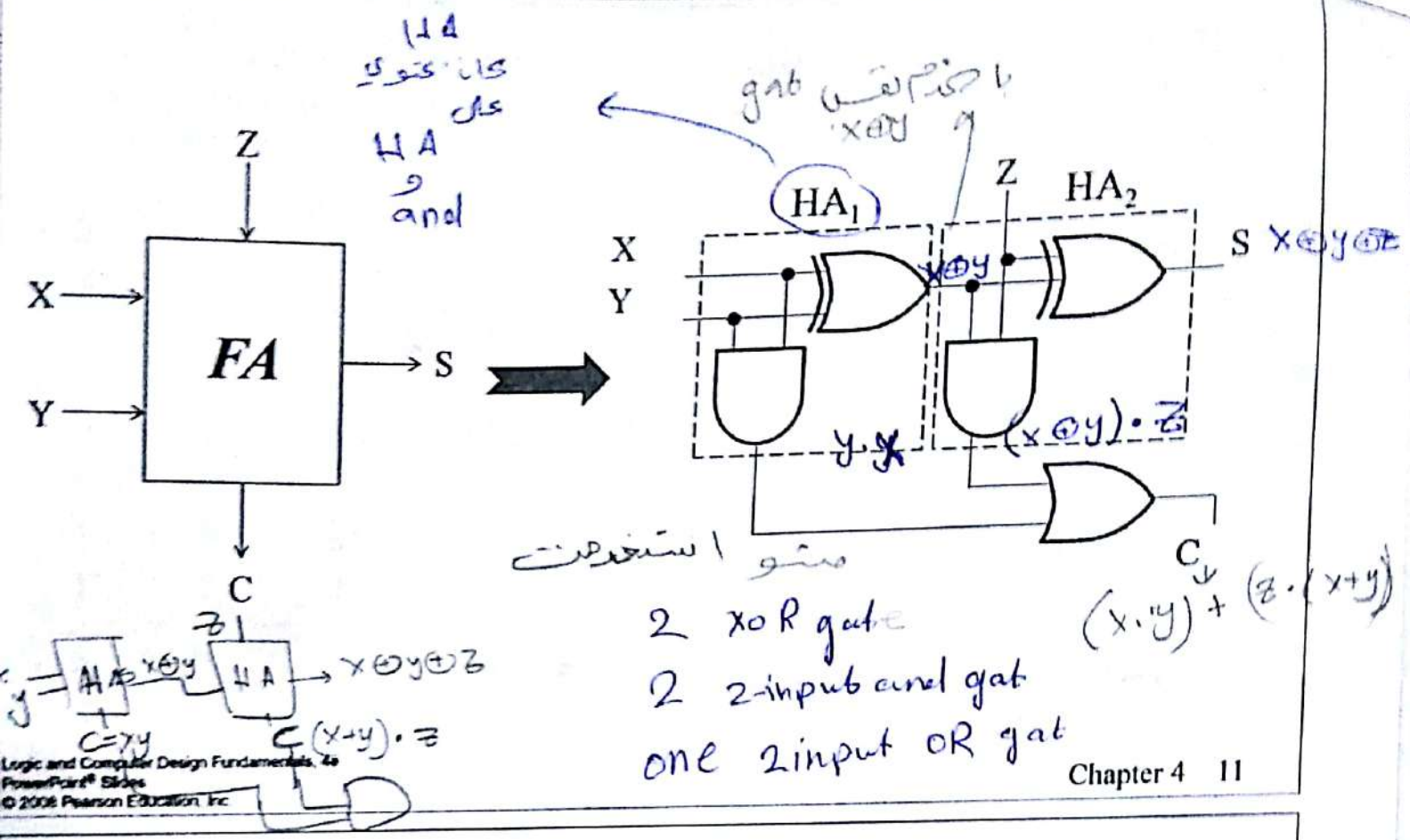
X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- Full-Adder K-Map:



$S = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ$ $C = XZ + XY + YZ$

- The S function is the three-bit XOR function (Odd Function):
  - $S = X \oplus Y \oplus Z$
- The Carry bit C is 1 if both X and Y are 1 (the sum is 2), or if the sum is 1 and a carry-in (Z) occurs. Thus C can be re-written as:
  - $C = XY + (X \oplus Y)Z$



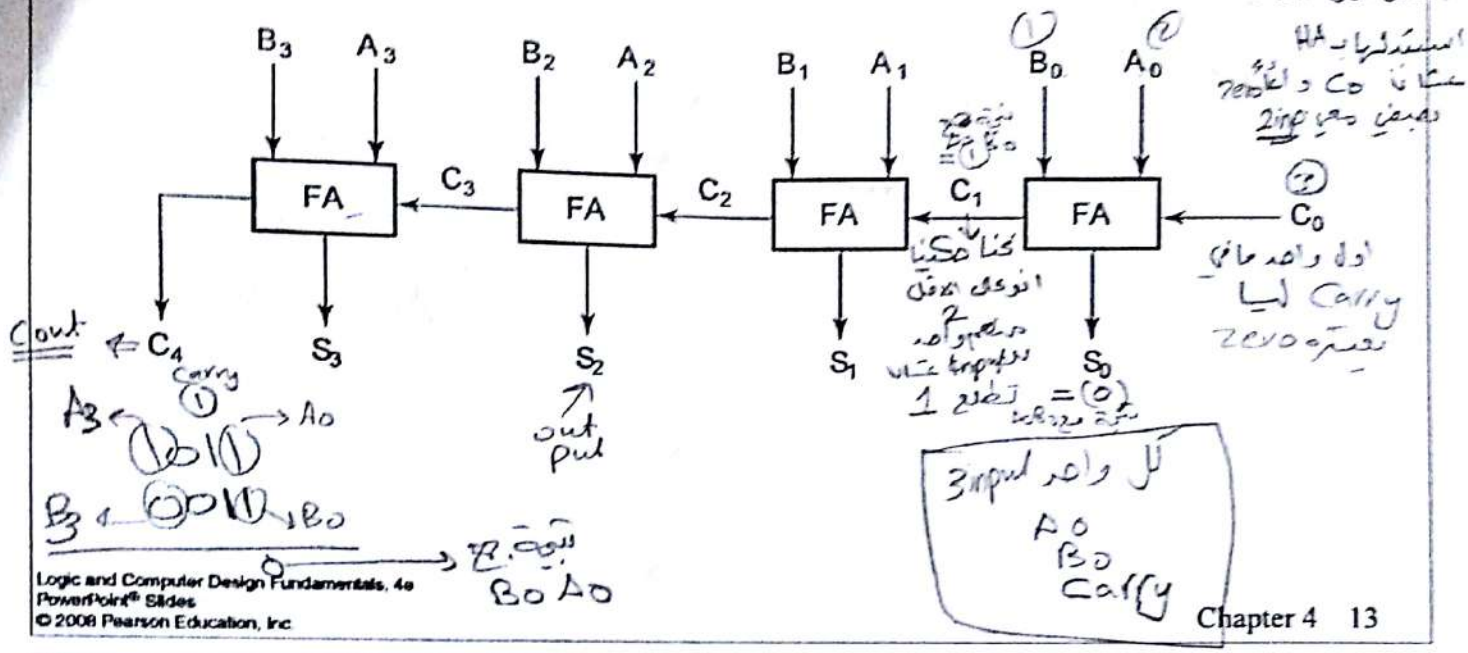
Binary Adders

- To add multiple operands, we "bundle" logical signals together into vectors and use functional blocks that operate on the vectors
- Example: ***4-bit ripple carry adder*** adds input vectors A(3:0) and B(3:0) to get a sum vector S(3:0)
- Note: carry-out of ***cell i*** becomes carry-in of ***cell i + 1***

Description	Subscript 3 2 1 0	Name
Carry In →	0 1 1 0	C_i
Augend	1 0 1 1	A_i
Addend	0 0 1 1	B_i
Sum	1 1 1 0	S_i
Carry out →	0 0 1 1	C_{i+1}

4-bit Ripple-Carry Binary Adder

- A four-bit Ripple Carry Adder made from four 1-bit Full Adders:



Homework

- When we subtract one bit from another, two bits are produced: ***difference bit (D)*** and ***borrow bit (B)***

X	0	1	0	0
−Y	−0	−1	−0	−1
B D	0 0	1 1	0 1	0 0

Borrow — نتيجة الطرح

- **Algorithm:**
  - Subtract the ***subtrahend (N)*** from the ***minuend (M)***
  - If **no end borrow occurs**, then $M \geq N$ and the result is a non-negative number and correct
  - If **an end borrow occurs**, then $N > M$ and the difference $(M - N + 2^n)$ is subtracted from 2^n, and a minus sign is appended to the result

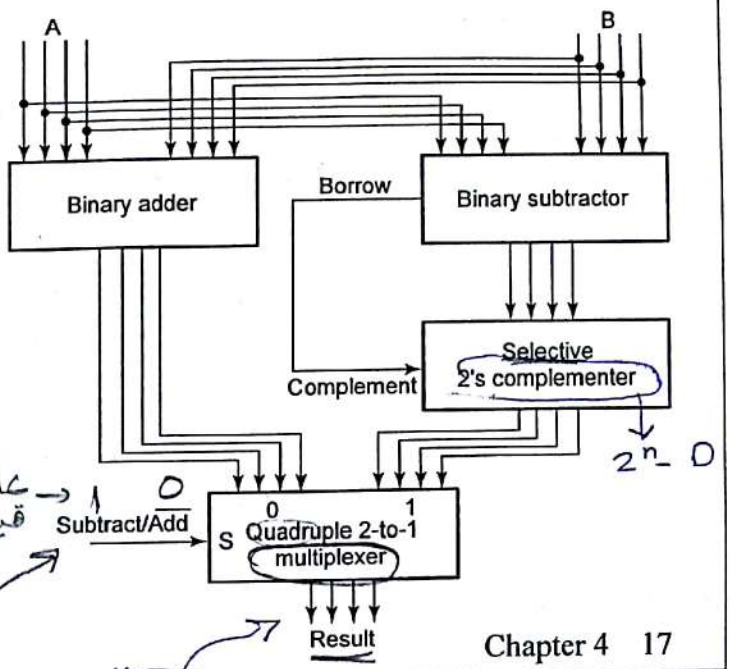
Unsigned Subtraction

- **Examples:**

10019	0100	10011	10010110	01100100
−0111	−0111	−11110	−01100100	−10010110
0010	B 1101	B 10101	00110010	B 11001110
No Borrow				
	2^n 10000	2^n 100000		2^n 100000000
	−1101	−10101		−11001110
	(−) 0011	(−) 01011		(−) 00110010
	2's com	2's com		2's com

Borrow 1

- The subtraction, $2^n - D$, is taking the ***2's complement of D***
- To do both unsigned addition and unsigned subtraction requires:
  - Addition and Subtraction are performed in parallel and Subtract/$\overline{Add}$ chooses between them
- Quite complex!
- ***Goal:*** Shared simpler logic for both addition and subtraction
- Introduce complements as an approach



Complements

- For a number system with radix (r), there are two complements:
  - ***Diminished Radix Complement***
    - Famously known as ***(r – 1)'s complement***
    - Examples:
      - **1's complement for radix 2**
      - **9's complement for radix 10**
    - For a number (N) with n-digits, the diminished radix complement is defined as:
      - $(r^n - 1) - N$
  - ***Radix Complement***
    - Famously known as ***r's complement*** for radix r
    - Examples:
      - **2's complement in binary**
      - **10's complement in decimal**
    - For a number (N) with n-digits, r's complement is defined as:
      - $r^n - N$, *when* $N \neq 0$

Diminished Radix Complement

- If N is a number of n-digits with radix (r), then
  - $N + (r-1)$'s complement of $N = \underbrace{(r-1)(r-1)(r-1)\ldots(r-1)}_{\text{n-digits}}$
  - *The $(r-1)$'s complement can be computed by subtracting each digit from $(r-1)$*
- Example: Find 1's complement of $(1011)_2$
  - $r = 2, n = 4$
  - Answer is $(2^4 - 1) - (1011)_2 = (0100)_2$
  - Notice that $(1011)_2 + (0100)_2 = (1111)_2$ which is $\underbrace{(2-1)(2-1)(2-1)(2-1)}_{\text{4-digits}}$
- Example: Find 9's complement of $(45)_{10}$
  - $r = 10, n = 2$
  - Answer is $(10^2 - 1) - (45)_{10} = (54)_{10}$
  - Notice that $(45)_{10} + (54)_{10} = (99)_{10}$ which is $\underbrace{(10-1)(10-1)}_{\text{2-digits}}$
- Example: Find 7's complement of $(671)_8$
  - $r = 8, n = 3$
  - Answer is $(8^3 - 1) - (671)_8 = (106)_8$
  - Notice that $(671)_8 + (106)_8 = (777)_8$ which is $\underbrace{(8-1)(8-1)(8-1)}_{\text{3-digits}}$

① Binary 1's Complement

- For $r = 2$, $N = 01110011_2$, $n = 8$ (8 digits):

  $(r^n - 1) = 256 - 1 = 255_{10}$ or 11111111_2
- The 1's complement of 01110011_2 is then:

$$\begin{array}{r} 11111111 \\ -\ 01110011 \\ \hline 10001100 \end{array}$$

- Since the $2^n - 1$ factor consists of all 1's and since $1 - 0 = 1$ and $1 - 1 = 0$, the one's complement is obtained ***by complementing each individual bit (bitwise NOT)***

Radix Complement

10 → 10's com
2 → 2's com

r's complement

- For number N with n-digit and radix (r):
  - If N ≠ 0, r's complement of $N = r^n - N$ ⟹ $r^n - n$
    - *r's complement = (r-1)'s complement + 1*
  - If N = 0, r's complement of N = 0 ⟹ 0
- Example: Find 10's complement of $(92)_{10}$
  - r = 10, n = 2 — $r^n - n$
  - Answer is $10^2 - (92)_{10} = (8)_{10}$
  - Notice that 9's complement of $(92)_{10}$ is $(7)_{10}$
    10's complement = 9's complement + 1
- Example: Find 16's complement of $(3AE7)_{16}$
  - r = 16, n = 4
  - Answer is $16^4 - (3AE7)_{16} = (10000)_{16} - (3AE7)_{16} = (C519)_{16}$
  - 15's complement = $(C518)_{16}$ → 16's complement = $(C518)_{16} + 1 = (C519)_{16}$

$(16^4)_{10} \rightarrow (2^4)_2$

طريقة سهلة → 15's بطريقة ⇒ $15^5 + 1 = 16^5$

$0 + FFFF - 3AE7 = (C519)$ (1) طريقة

$(r-1)_{com} + 1 = r_{com}$

Binary 2's Complement

- For r = 2, $N = 01110011_2$, n = 8 (8 digits), we have:
  - $(r^n) = 256_{10}$ or 100000000_2
- The 2's complement of 01110011 is then:

r^n 100000000
n − 01110011
10001101

طريقة (1)

باخذ عكس الرقم 15com + 1 = 2com

$10001100 + 1 = 10001101$ (2) طريقة

$(r-1) + 1 = r_{com}$

- ***Note the result is the 1's complement plus 1, a fact that can be used in designing hardware***
- ***Remember the 2's complement of $(000..00)_2$ is $(000..00)_2$***
- ***Complement of a complement restores the number to its original value:***
  - *The Complement of complement* $N = 2^n - (2^n - N) = N$

$2^n - 2n = n$

اذا كملت complement مرتين برجع n زي ما هو

Alternate 2's Complement Method

- Given: an *n*-bit binary number, beginning at the least significant bit and proceeding upward:
  - **Copy all least significant 0's**
  - **Copy the first 1**
  - **Complement all bits thereafter**

هذا طريقة سهلة وسريعة
(3) طريقة

- 2's Complement Example:

  10010100

  - **Copy underlined bits:**

    100

  - **and complement bits to the left:**

    01101100

(1) يبدو على اول واحد
(2) جزلو ويعكس الباقي
10010100 ← تكميلي للاثنين
بعكسهم
01101100

Subtraction with 2's Complement

- **For n-digit, unsigned numbers M and N, find M – N in base 2:**
  - Add the 2's complement of the subtrahend N to the minuend M: اصول طرح ← مع.

  - **M – N ⟶ M + (2ⁿ – N) = M – N + 2ⁿ**

- If M ≥ N, the *sum* produces end carry 2^n which is discarded; and from above, M – N remains (Carry(1) → borrow(0)) M – N وخلص
- If M < N, the *sum* does not produce end carry, and from above, is equal to 2^n – (N – M) which is the 2's complement of (N – M) Carry(0) → borrow(1)
  وقتها لازم اطلع قيمة حقيقية في سلوم 16
- To obtain the result – (N – M) , take the 2's complement of the sum and place a "–" to its left

Unsigned 2's Complement Subtraction Example: (M > N)

- Find $01010100_2 - 01000011_2$

$$\begin{array}{rl} & 01010100 \\ - & \underline{01000011} \xrightarrow{\text{2's comp}} \\ & \end{array} \quad \begin{array}{l} (1)01010100 \\ + \underline{10111101} \\ 00010001 \end{array}$$

- The carry of 1 indicates that no correction of the result is required

Unsigned 2's Complement Subtraction Example: (M < N)

- Find $01000011_2 - 01010100_2$

$$\begin{array}{rl} & 01000011 \\ - & \underline{01010100} \xrightarrow{\text{2's comp}} \\ & \end{array} \quad \begin{array}{l} (0)01000011 \\ + \underline{10101100} \\ 11101111 \xrightarrow{\text{2's comp}} \\ (-)\ 00010001 \end{array}$$

- The carry of 0 indicates that a correction of the result is required

- Result = – (00010001)

Signed Integer Representations

- ***Signed-Magnitude:*** here the (n – 1) digits are interpreted as a positive magnitude
  - Max = $+(2^{n-1} - 1)$
  - Min = $-(2^{n-1} - 1)$
  - Two representation for zero (i.e. ± 0)
- ***Signed-Complement:*** here the digits are interpreted as the rest of the complement of the number. There are two possibilities here:
  - ***Signed 1's Complement:*** Uses 1's Complement Arithmetic
    - Max = $+(2^{n-1} - 1)$
    - Min = $-(2^{n-1} - 1)$
    - Two representation for zero (i.e. ± 0)
  - ***Signed 2's Complement:*** Uses 2's Complement Arithmetic
    - Max = $+(2^{n-1} - 1)$
    - Min = -2^{n-1}
    - Single representation for zero

Signed Integer Representation Example

- **r = 2, n = 3**

Number	Signed-Magnitude	1's Complement	2's Complement
+3	011	011	011
+2	010	010	010
+1	001	001	001
+0	000	000	000
-0	100	111	----
-1	101	110	111
-2	110	101	110
-3	111	100	101
-4	----	----	100

- **Represent the number -9 using 8-bits**
  - Sign-Magnitude = 10001001
  - 1's complement = 11110110
  - 2's complement = 11110111

2's Complement Signed Numbers

- Signed 2's complement is the most common representation for signed numbers
  - Focus of the course

- For any n-bit 2's complement signed number ($b_{n-1}b_{n-2}b_{n-3} \ldots b_2b_1b_0$), the decimal value is given by

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

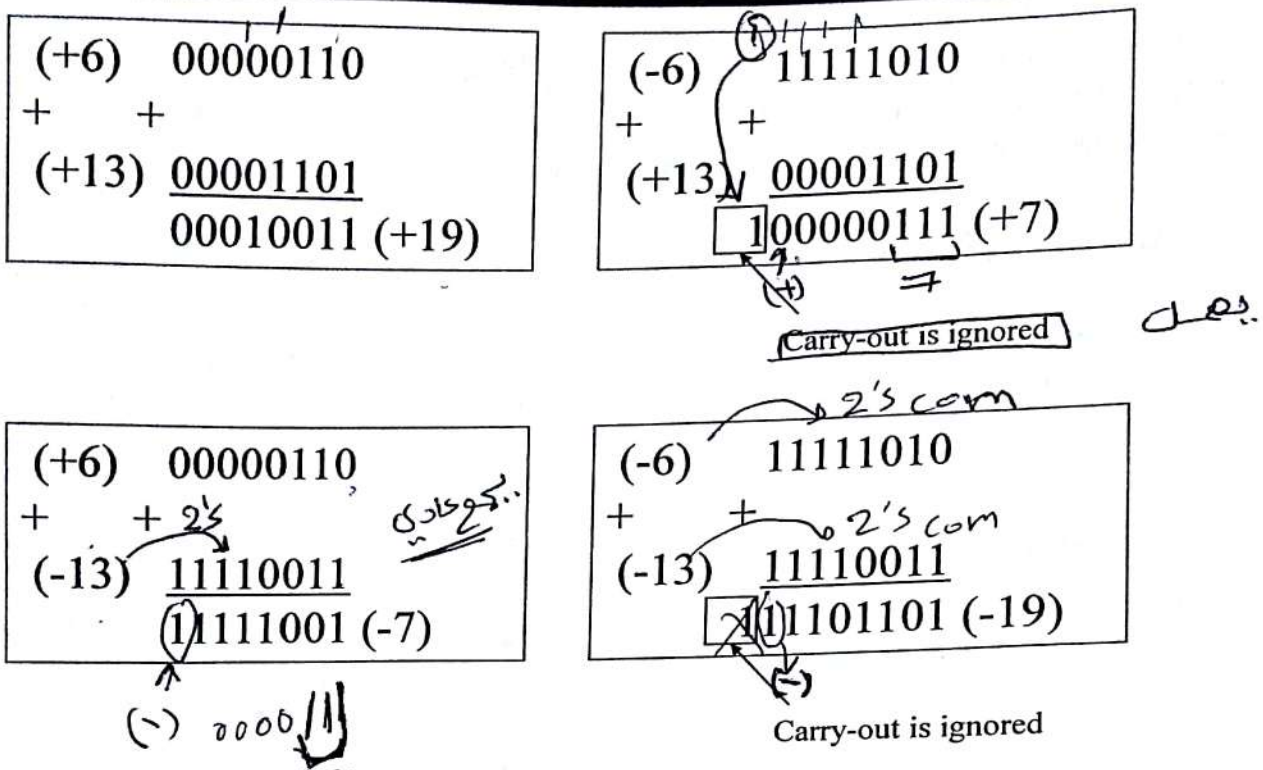
- Example: What is value of the 2's complement number $(100111)_2$?

$$Value = -2^5 \times 1 + 7 = -25$$

Signed-2's Complement Arithmetic

- **Addition:**
  - **Add the numbers including the sign bits**
  - **Discard the carry out of the sign bits**

- **Subtraction:**
  - **Form the complement of the number you are subtracting**
  - **Follow the same rules for addition**
  - $A - B = A + (-B) = A + (\overline{B} + 1)$



Signed 2's Complement Subtraction

Overflow Detection

- In computers, the number of bits is fixed
- *Overflow* occurs if $n + 1$ bits are required to contain the result from an n-bit addition or subtraction
- ***Unsigned number overflow*** is detected from the **end carry-out** when ***adding*** two unsigned numbers
  - **Overflow is *impossible* for unsigned subtraction**

(8)	1000
+	+
(12)	1100
	10100 (4)

Carry-out = 1 → Overflow

- ***Signed number overflow*** can occur for:
  - **Addition of two operands with the same sign**
  - **Subtraction of operands with different signs**

Signed-number Overflow Detection

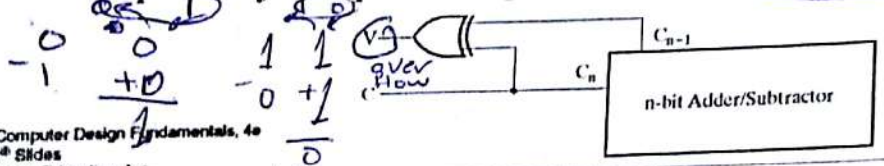
- **Signed number cases with carries C_n and C_{n-1} shown for correct result signs:**

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

- **Signed number cases with carries shown for erroneous result signs (indicating overflow):**

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

- ***Simplest way to implement signed overflow is:*** $V = C_n \oplus C_{n-1}$



Signed-number Overflow Examples

- **8-bit signed number range between: -128 to +127**

(+70) 01000110
\+ +
(+80) 01010000
10010110 (-106)

$V = C_7 \oplus C_8 = 1 \oplus 0 = 1$

(-70) 10111010
\+ +
(-80) 10110000
101101010 (+106)

$V = C_7 \oplus C_8 = 0 \oplus 1 = 1$

(+70) 01000110
\- +
(-80) 01010000
10010110 (-106)

$V = C_7 \oplus C_8 = 1 \oplus 0 = 1$

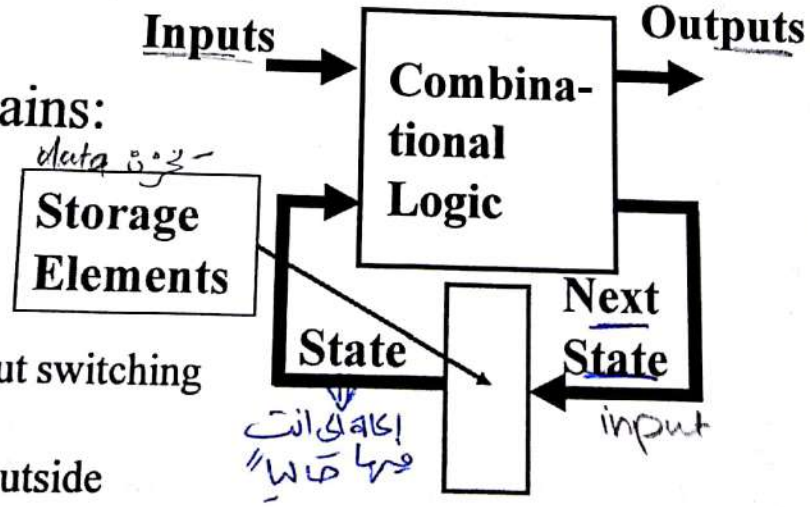
(-70) 10111010
\- +
(+80) 10110000
101101010 (+106)

$V = C_7 \oplus C_8 = 0 \oplus 1 = 1$

Overview

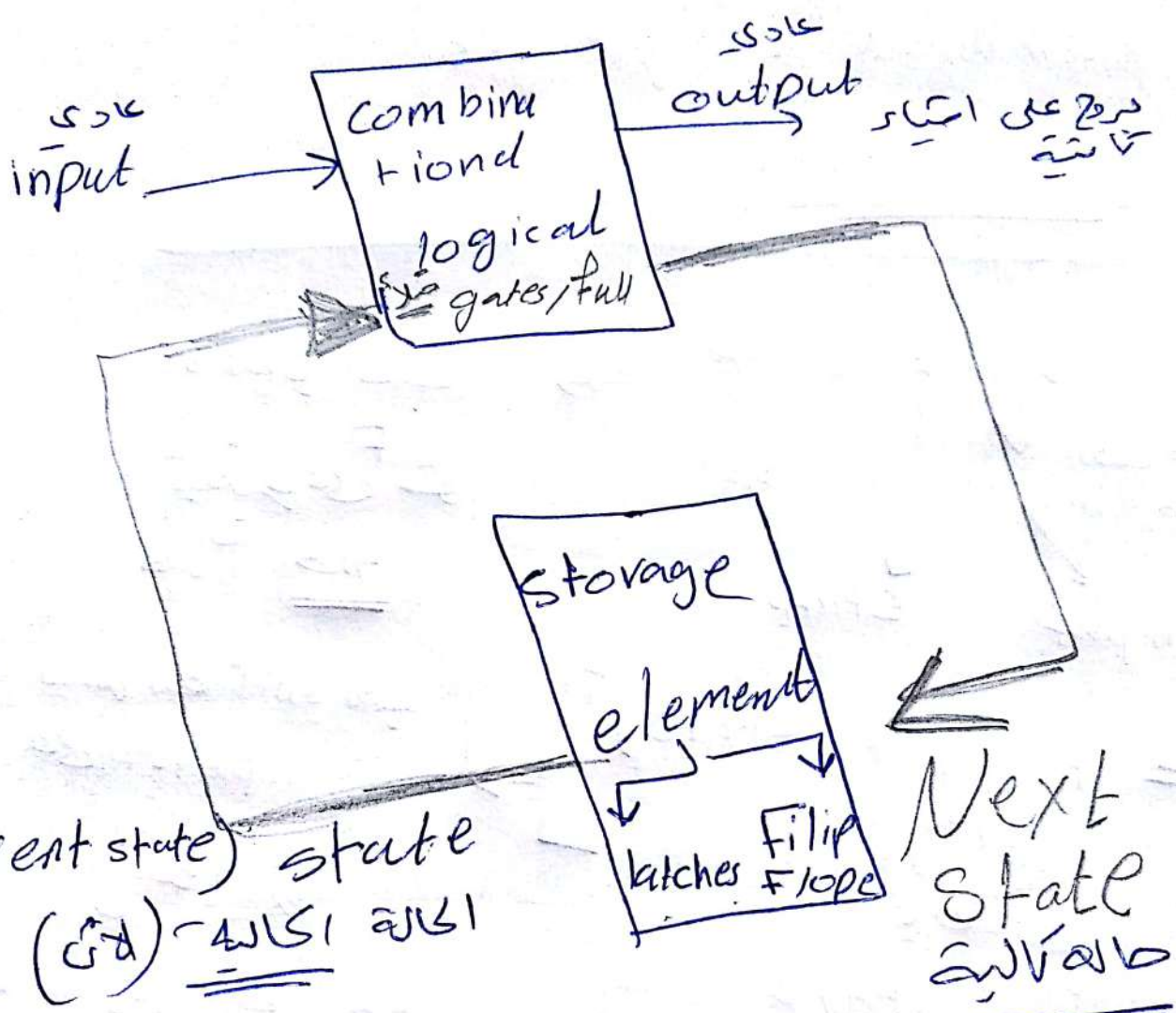
- **Part 1 - Storage Elements and Analysis**
  - **Introduction to sequential circuits**
  - **Types of sequential circuits**
  - **Storage elements**
    - **Latches**
    - **Flip-flops**
  - **Sequential circuit analysis**
    - **State tables**
    - **State diagrams**
    - **Equivalent states**
    - **Moore and Mealy Models**
- **Part 2 - Sequential Circuit Design**

Introduction to Sequential Circuits



- *A Sequential circuit contains:*
  - ***Storage elements:***
    - Latches or Flip-Flops
  - ***Combinational Logic:***
    - Implements a multiple-output switching function
    - Inputs are signals from the outside
    - Outputs are signals to the outside
    - Other inputs, State or Present State, are signals from storage elements
    - The remaining outputs, Next State are inputs to storage elements

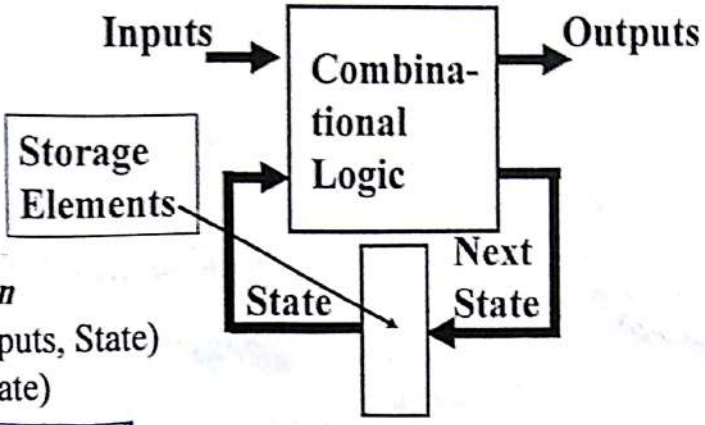
Sequential circuits تتضمن



(present state) الحالة الحالية

Next State حالة تالية

Introduction to Sequential Circuits

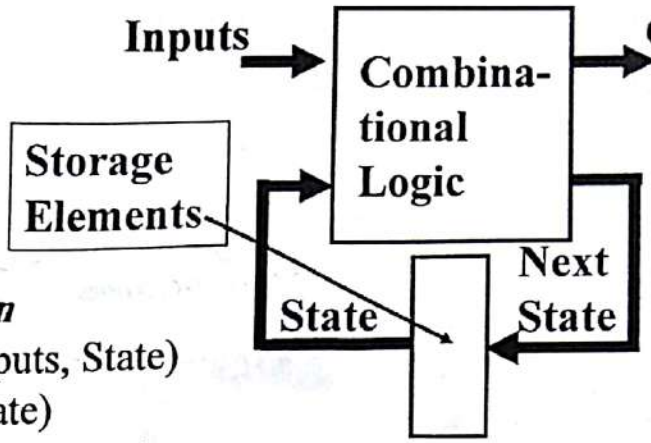


- **Combinatorial Logic**
  - ***Next state function***
    Next State = f(Inputs, State)
    OR Next State = f(State)
  - ***Output function (Mealy)***
    Outputs = g(Inputs, State)
  - ***Output function (Moore)***
    Outputs = g(State)
- **Output function type depends on specification and affects the design significantly**

Types of Sequential Circuits

- **Depends on the times at which:**
  - storage elements observe their inputs, and
  - storage elements change their state

1. **Synchronous**
   - Behavior defined from knowledge of its signals at ***discrete*** instances of time
   - **Storage elements observe inputs and can change state only in relation to a timing signal (*clock pulses* from a *clock*)**
   - ***Simple to design but slow***
2. **Asynchronous**
   - Behavior defined from knowledge of inputs at any instant of time and the order in ***continuous*** time in which inputs change
   - ***Complex to design but fast***



- **Combinatorial Logic**
  - ***Next state function***
    Next State = f(Inputs, State)
    OR Next State = f(State)
  - ***Output function (Mealy)***
    Outputs = g(Inputs, State)
  - ***Output function (Moore)***
    Outputs = g(State)
- **Output function type depends on specification and affects the design significantly**

Types of Sequential Circuits

- **Depends on the times at which:**
  - storage elements observe their inputs, and
  - storage elements change their state

1. **Synchronous**
   - Behavior defined from knowledge of its signals at *discrete* instances of time
   - **Storage elements observe inputs and can change state only in relation to a timing signal (*clock pulses* from a *clock*)**
   - ***Simple to design but slow***
2. **Asynchronous**
   - Behavior defined from knowledge of inputs at any instant of time and the order in *continuous* time in which inputs change
   - ***Complex to design but fast***

اذا بدي اكتب

Combinational logic As a function

Next state ⇒ اذا بدي اكتب

تعتمد على (لازم الاثنين)

input state

= f (input, state)

او

في حالات

= f (state)

تعتمد على state فقط

F تعتبر على شو

لازم اطلع

+ يسمى النظام الذي تعتمد على input+state

Mealy

+ يسمى النظام الذي تعتمد على state فقط

Moore

output

= g (input, state)

function

ممكن تعتمد (1)

حالات ©

(أو) g (state)

ممكن تعتمد فقط

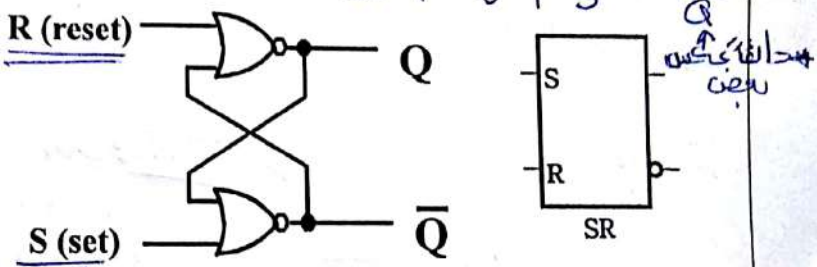
Storage Elements

- Any storage element can maintain a binary state indefinitely (as long as the power is on) until directed by the input signals to switch
- Storage elements: ***Latches*** and ***Flip-flops (FFs)***
- Latches and FFs differ in:
  - Number of inputs
  - Manner in which the inputs affect the binary state
- Latch:
  - Asynchronous
  - Although difficult to design, we discuss latches first because they are the building blocks for flip-flops

Basic (NOR) SR Latch

- **Cross-coupling two NOR gates**

R	S	Q	$\bar{Q}$	Comment
0	0	Q	$\bar{Q}$	Hold, no change
0	1	1	0	Set
1	0	0	1	Reset
1	1	0	0	Not allowed

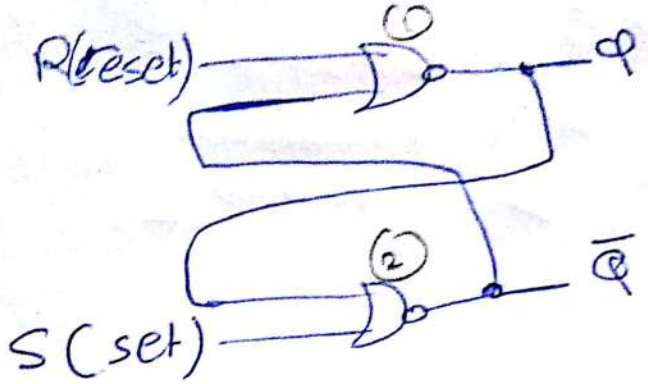


- **Time sequence behavior:**
- **S = 1, R = 1 is forbidden as input pattern**

R	S	Q	$\bar{Q}$	Comment
0	0	?	?	Stored state unknown
0	1	1	0	"Set" Q to 1
0	0	1	0	Now Q "remembers" 1
1	0	0	1	"Reset" Q to 0
0	0	0	1	Now Q "remembers" 0
1	1	0	0	Both go low
0	0	?	?	Unstable!

Time

(8)



R	S	Q	$\overline{Q}$	
0	0	Q	$\overline{Q}$	⇒ Hold No change
0	1	1	0	⇒ set
1	0	0	1	⇒ Reset
1	1	0	0	⇒ not allowed ⇒ لازم input ما يكون 11

جواب نهائي

RS
00

(1)

0 NOR $\overline{Q}$ ⇒ $\overline{Q}$ inverter Q

(2)

0 NOR Q ⇒ Q inverter $\overline{Q}$

Hold ← نسمى هذه الحالة
No change

---

(1)

R S
0 1

0 NOR $\overline{Q}$ ⇒ $\overline{Q}$ inverter Q
zero zero (1)

Set ← نسمى 1 NOR Q ⇒ 1 inver Zero
$\overline{Q}$

③ RS
1 0

1 NOR $\bar{Q}$ ⇒ 1 inverter 0 (Q)

0 NOR 0 ⇒ 0 inverter 1 ($\bar{Q}$)

يسمى الحالة
Reset

④ RS
1 1

1 NOR $\bar{Q}$ ⇒ 1 inverter 0

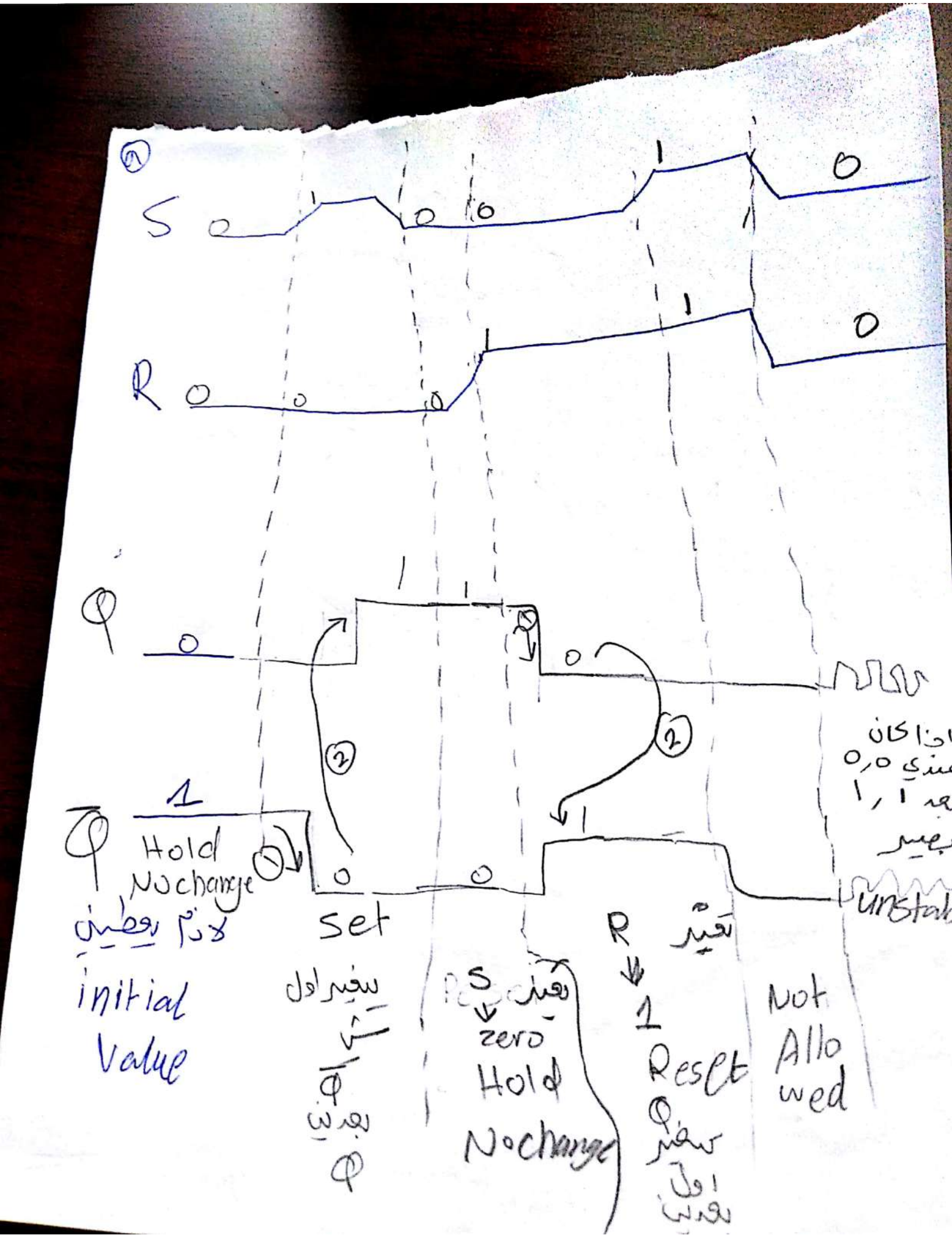
1 NOR 0 ⇒ 1 inverter 0

يسمى هذه الحالة
not allowed

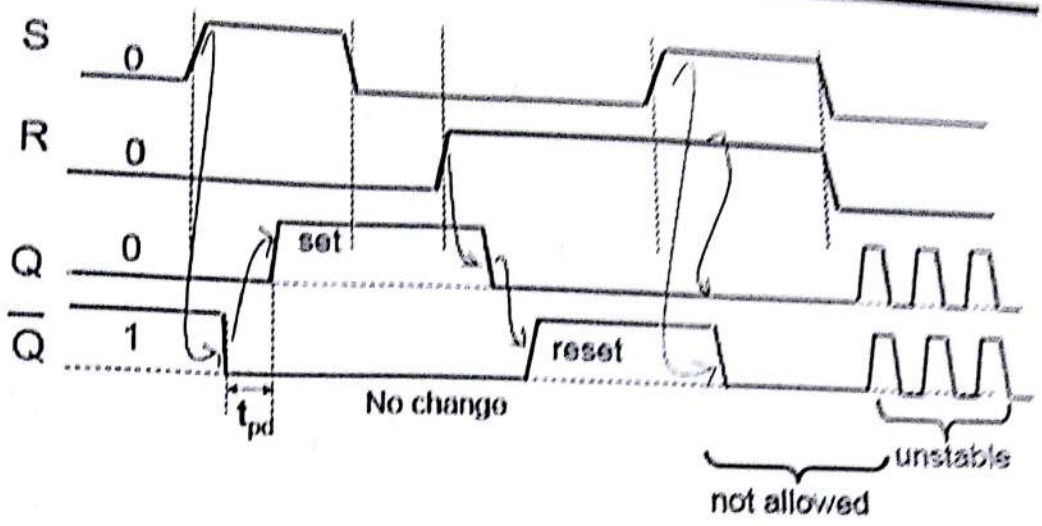
دائماً اذا حطينا

R S	Q $\bar{Q}$	
0 1	1 0	⇒ Set
1 0	0 1	⇒ Reset
0 0	Q $\bar{Q}$	⇒ بضل قبلو بضلو ثابتين

Scanned by CamScanner



Timing Waveform of NOR SR Latch

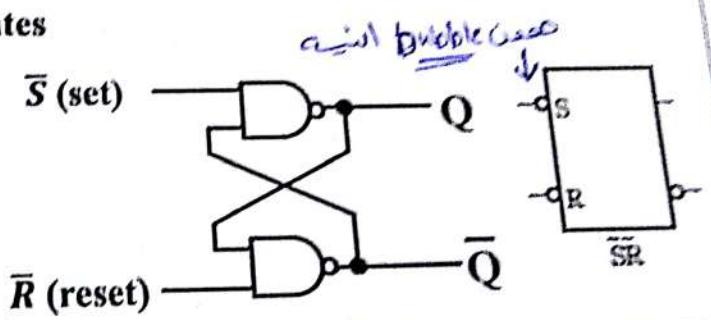


Basic (NAND) $\overline{S}\overline{R}$ Latch

active low latch
$\overline{S}\overline{R}$ Latch

- Cross-coupling two NAND gates
- *Active low inputs*

$\overline{R}$	$\overline{S}$	Q	$\overline{Q}$	Comment
0	0	1	1	Not allowed
0	1	0	1	Reset
1	0	1	0	Set
1	1	Q	$\overline{Q}$	Hold, no change

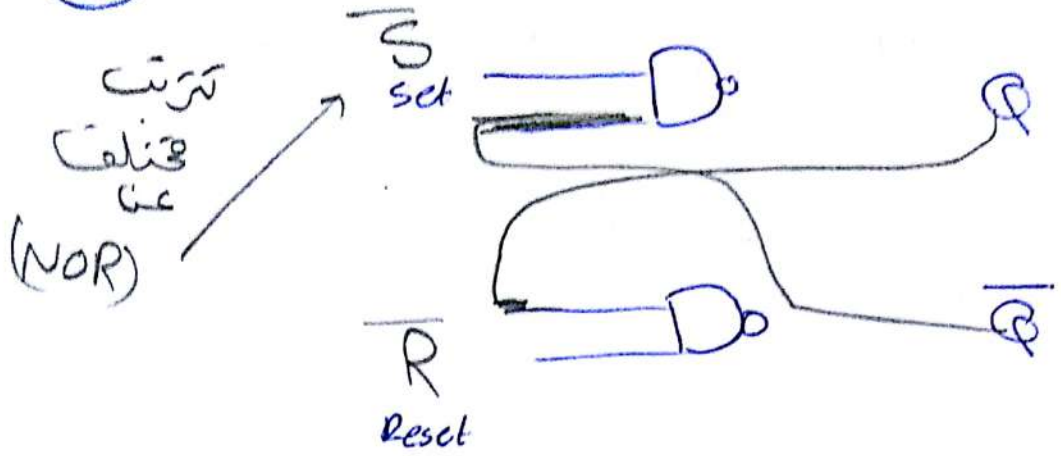


- Time sequence behavior:
- $\overline{S} = 0, \overline{R} = 0$ is forbidden as

Time

$\overline{R}$	$\overline{S}$	Q	$\overline{Q}$	Comment
1	1	?	?	Stored state unknown
1	0	1	0	"Set" Q to 1
1	1	1	0	Now Q "remembers" 1
0	1	0	1	"Reset" Q to 0
1	1	0	1	Now Q "remembers" 0
0	0	1	1	Both go high
1	1	?	?	Unstable!

(10)



ترتيب مختلف عن (NOR)

set → دخلنا ترتيب 1

$\overline{S}$	$\overline{R}$
1	0

1 NAND $\overline{Q}$ = $\overline{Q}$ inverter Q
1 ← 1 — 0

0 NAND Q = 0 inverter $\overline{Q}$ 1

$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	
1	0	0	1	Reset

$\overline{S}$	$\overline{R}$
0	1

0 Nand $\overline{Q}$ = 0 inverter 1

1 Nand Q = 1 inverter 0
1

$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	
0	1	1	0	set

$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	
1	0	0	1	Reset
0	1	1	0	set

$\bar{S}$ $\bar{R}$ 1 Nand $\bar{Q}$ = $\bar{Q}$ inverse Q

1 1

1 Nand Q = Q inverse $\bar{Q}$

Hold ↑

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

Hold No change

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

Not allowed

الفرق بين

NOR

NAND Latch

NOR

0 0

Hold

1 1

Not allowed

NAND

0 0

Not allowed

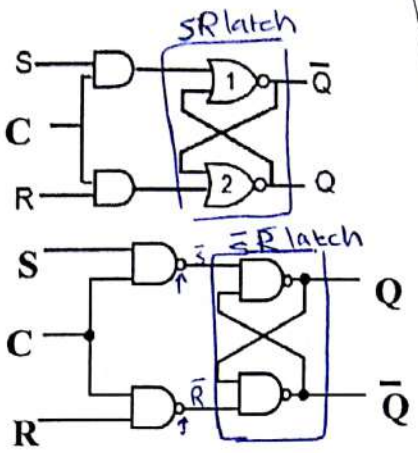
1 1

No change

* اذا كان معنى 0 0 بعدين 1 1 unstable

Clocked SR Latch (Pulse-triggered Latch)

- The operation of the basic NOR and basic NAND latches can be modified by a providing a control input (C) that determines when the state of the latch can be changed
  - **Adding two AND gates to basic SR latch *OR***
  - **Adding two NAND gates to $\overline{SR}$ basic latch**



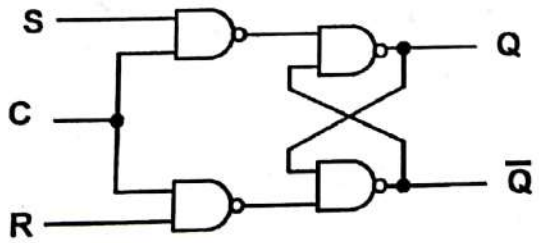
C	R	S	Q	$\overline{Q}$	Comment
0	x	x	Q	$\overline{Q}$	Hold, no change
1	0	0	Q	$\overline{Q}$	Hold, no change
1	0	1	1	0	Set
1	1	0	0	1	Reset
1	1	1	Not allowed		

- Has a time sequence behavior similar to the basic S-R latch except that the S and R inputs are only observed when the line C is high
- C means "control" or "clock"

NOR: S → Q, R → $\overline{Q}$ | NAND: S → $\overline{Q}$, R → Q

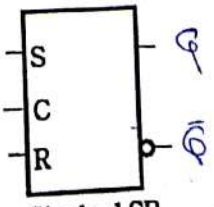
Clocked SR Latch (continued)

- The Clocked SR Latch can be described by a table:



Q(t)	S	R	Q(t+1)	Comment
0	0	0	0	No change
0	0	1	0	Clear Q (Reset)
0	1	0	1	Set Q
0	1	1	???	Indeterminate
1	0	0	1	No change
1	0	1	0	Clear Q
1	1	0	1	Set Q
1	1	1	???	Indeterminate

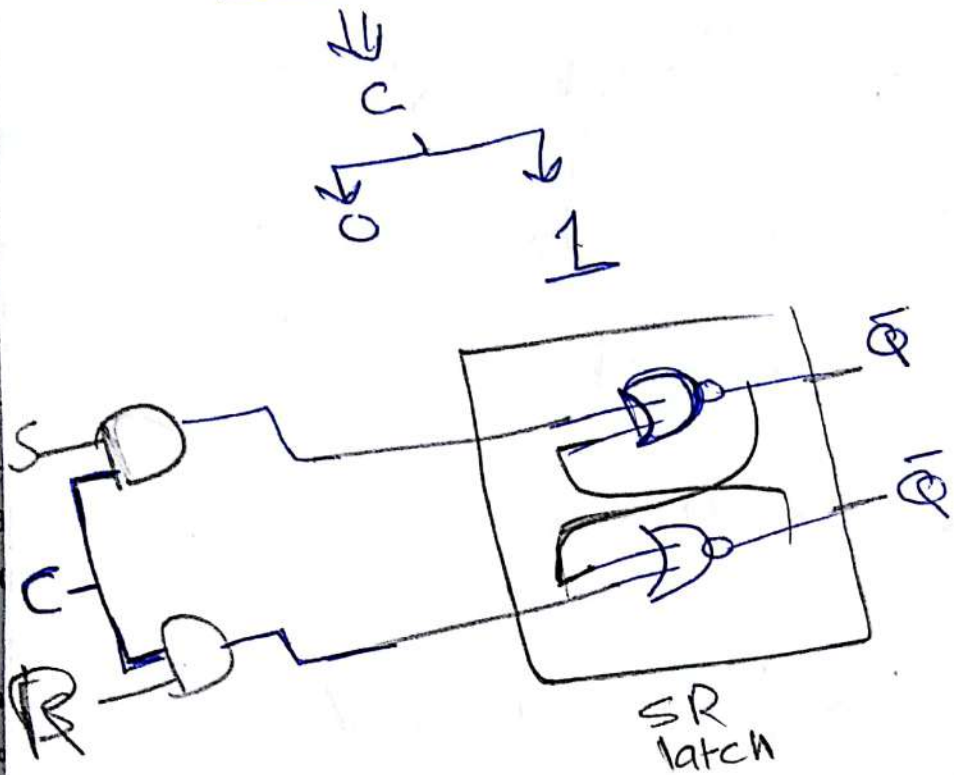
- The table describes what happens after the clock [at time (t+1)] based on:
  - current inputs (S,R) and
  - current state Q(t)

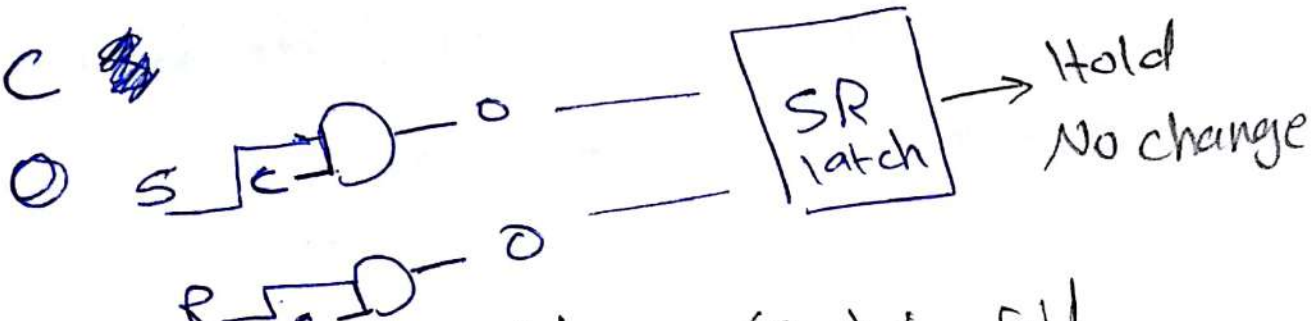


Clocked SR

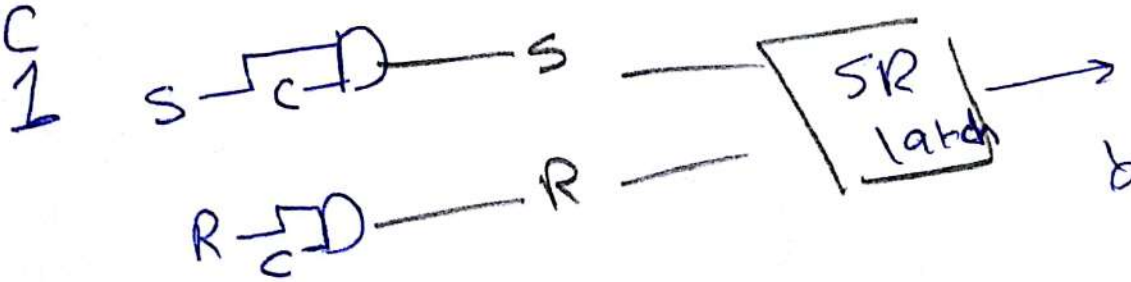
(11)

Clocked SR latch





Hold ← (C=0) لحالة

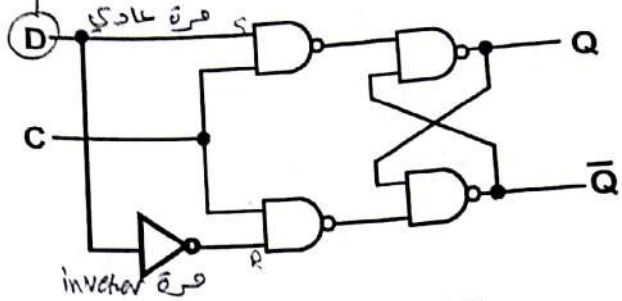


لطيف
حالة
من شروط
SR
latch

C	R S	Q $\bar{Q}$	
0	x x	Q $\bar{Q}$	⇒ Hold ⇐
1	0 0	Q $\bar{Q}$	⇒ Hold
1	0 1	1 0	⇒ set
1	1 0	0 1	⇒ Reset
1	1 1	—	⇒ Not allowed

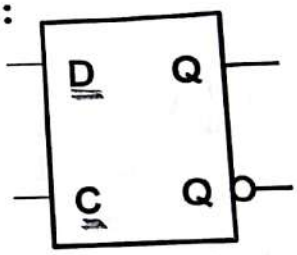
D Latch

- **Adding an inverter to the S-R Latch, gives the D Latch:**
- **Note that there are no "indeterminate" states!**

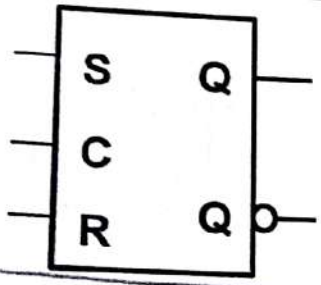


C	D	Q	$\overline{Q}$	Comment
0	x	Q	$\overline{Q}$	Hold, no change
1	0	0	1	Reset
1	1	1	0	Set

The graphic symbol for a D Latch is:



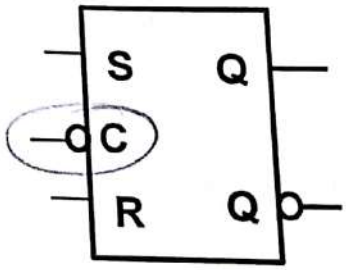
Variations of Clocked SR and D Latches



+ve pulse-triggered SR latch

C = 0 → Hold

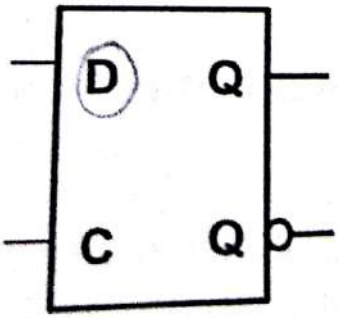
C = 1 → Change



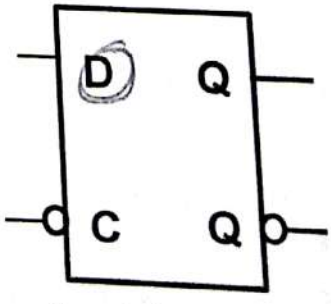
-ve pulse-triggered SR latch

C = 0 → Change

C = 1 → Hold



+ve pulse-triggered D latch



-ve pulse-triggered D latch

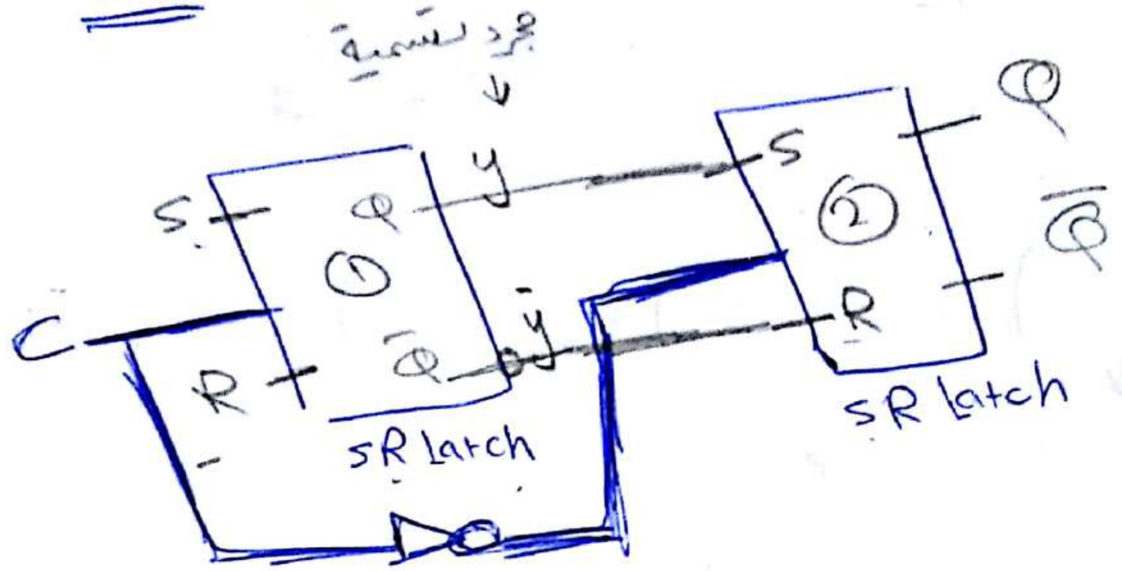
Flip-Flops

storge element
عنصر تخزين
1 bit

- Master-slave flip-flop
- Edge-triggered flip-flop
- Standard symbols for storage elements
- Direct inputs to flip-flops

* storge element
* 1 يسمح تخزين
bit
0 او 1
* ما يخزن value تعتمد
او بـ وقت منفصل

SR Master-slave



① C=1 بشتغل لما → Master
C=0 ⇒ بعمل Hold

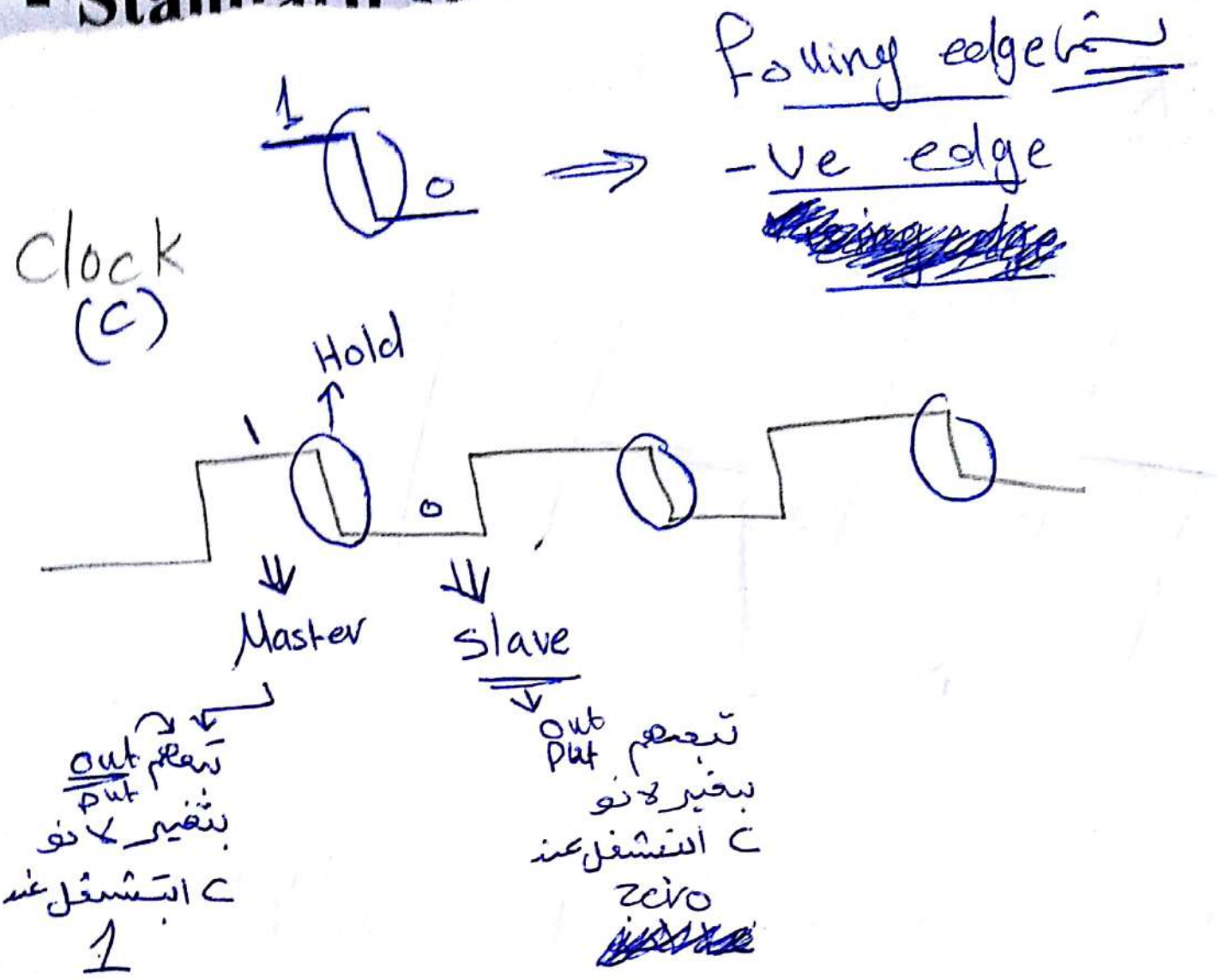
② C=0 بشتغل لما → slave
C=1 بعمل Hold

عكس بعض

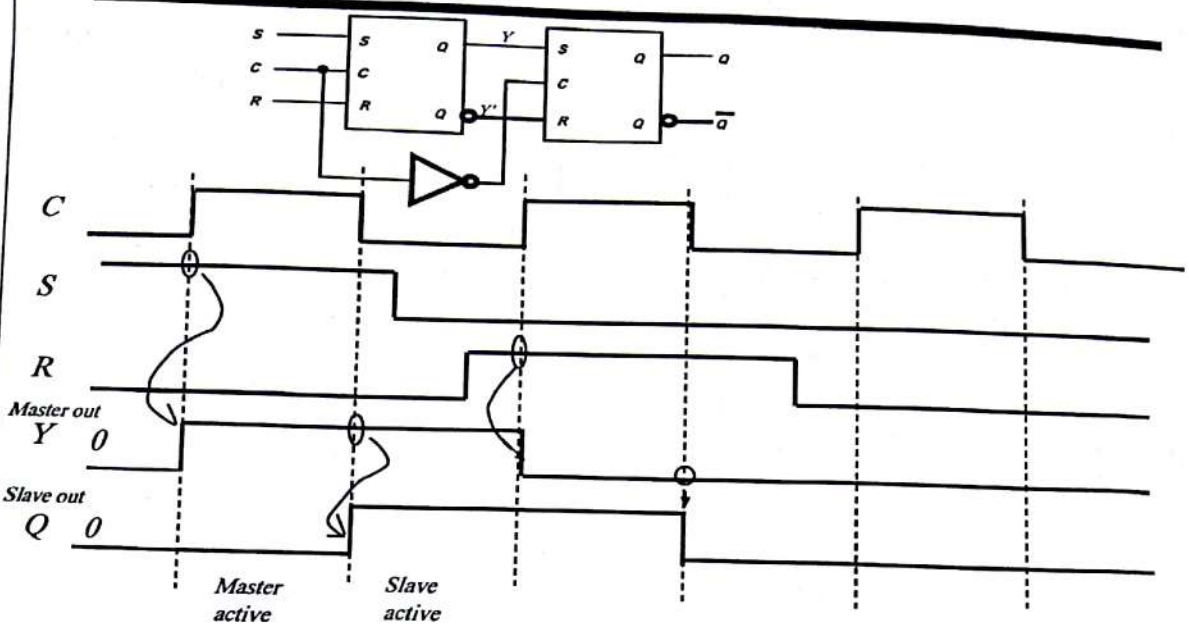
⑯

Scanned by CamScanner

- **Standard symbols f**



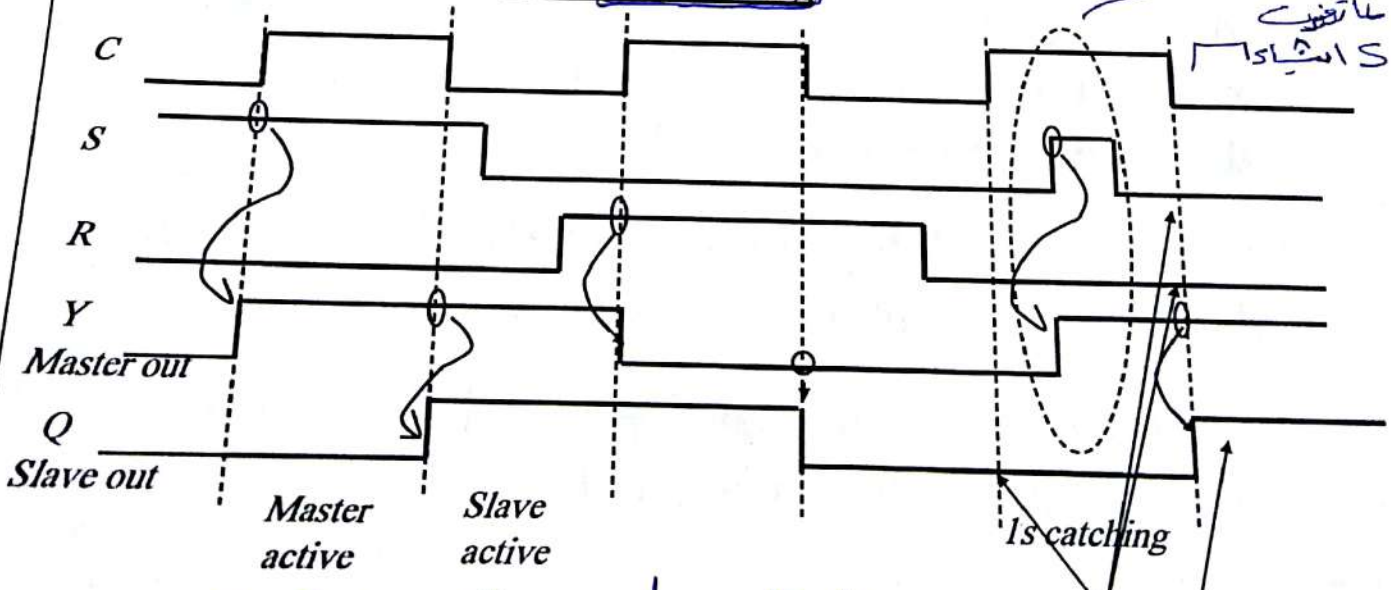
Timing diagram for SR Master-Slave Flip-Flop



Master-Slave Flip-Flop Problem

S, R Change during positive pulse عند master

- **S and/or R are permitted to change while C = 1**
  - **Chances of 0s or 1s catching**



اذا تغيرت S ⇒ 1s catching

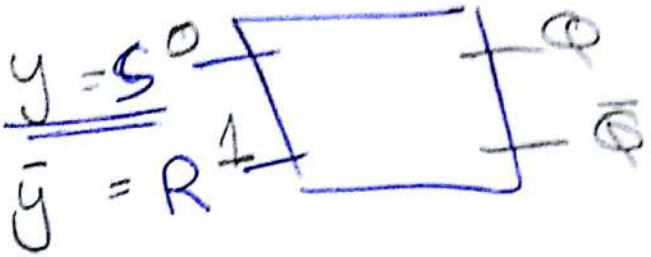
اذا تغيرت R ⇒ 0s catching

* C=0 لما الـ 2 بيشتغل

Slave

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

Reset



y = 0

Q = 0

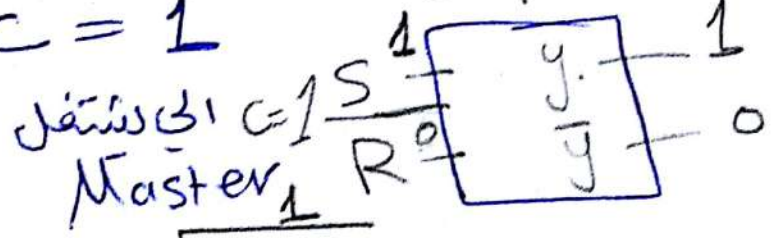
y 0

Q 0

---

* C = 1 عكس الوحدة

الي بيشتغل C=1 Master 1



Rs	Q	$\bar{a}$
0	1	1 0

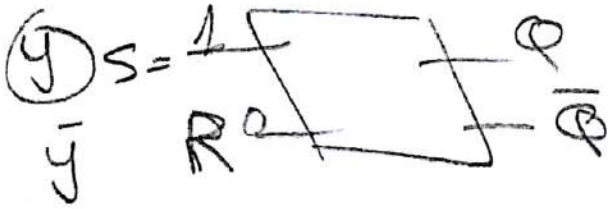
set

Q ⇒ Hold

Q ; J

C=0
Slave
تعمل

Master
Hold



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

set

C = 1

Master
active

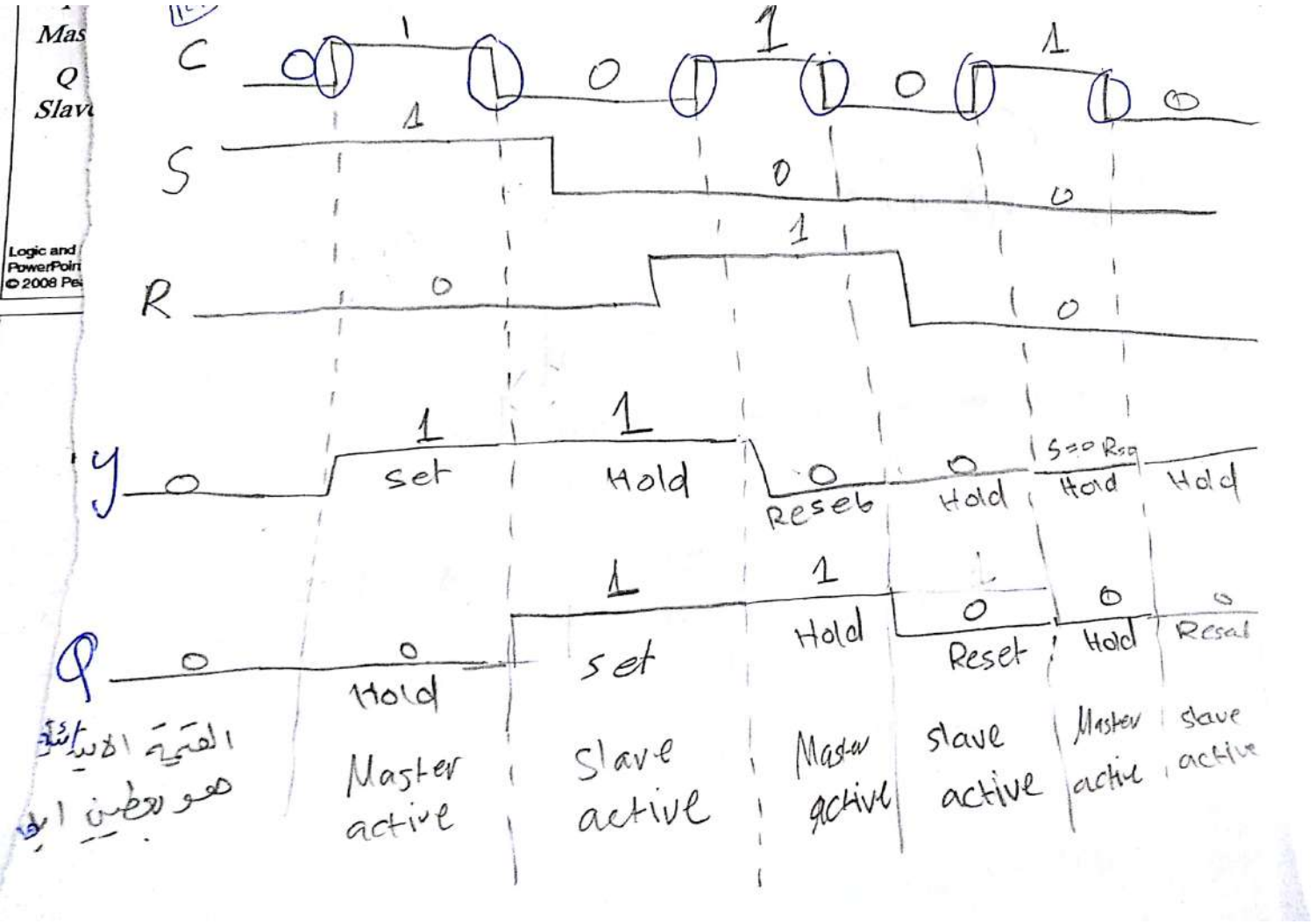
slave
Hold

بالعكس

R	S	(y) Q	($\bar{y}$) $\bar{Q}$
1	0	0	1

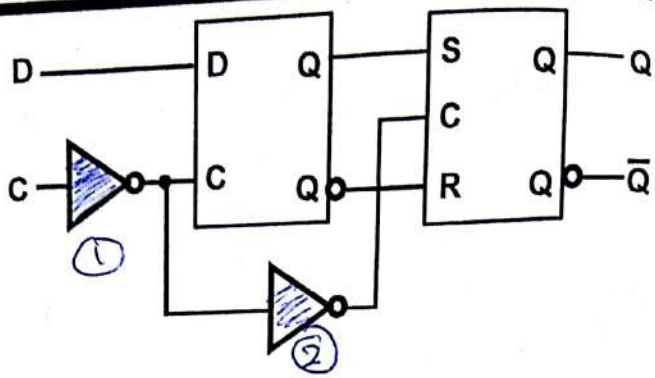
Reset

17



Positive-Edge Triggered D Flip-Flop

- Formed by adding inverter to clock input

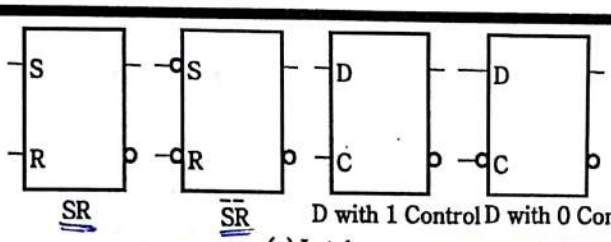


- Q changes to the value on D applied at the positive clock edge

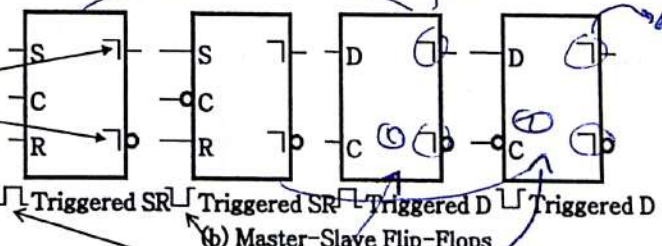
- Our choice as the standard flip-flop for most sequential circuits

Standard Symbols for Storage Elements

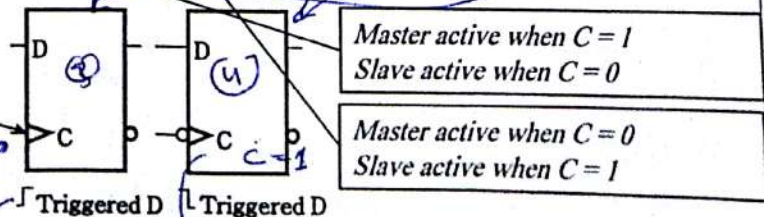
- Latches:

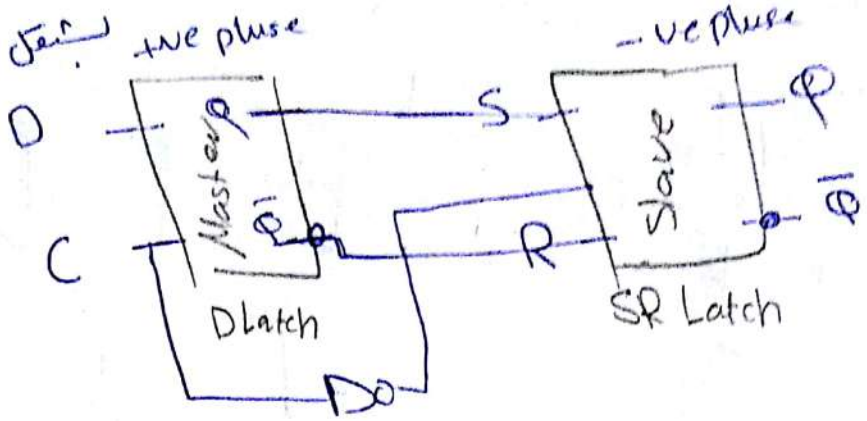


- Master-Slave: Postponed output indicators

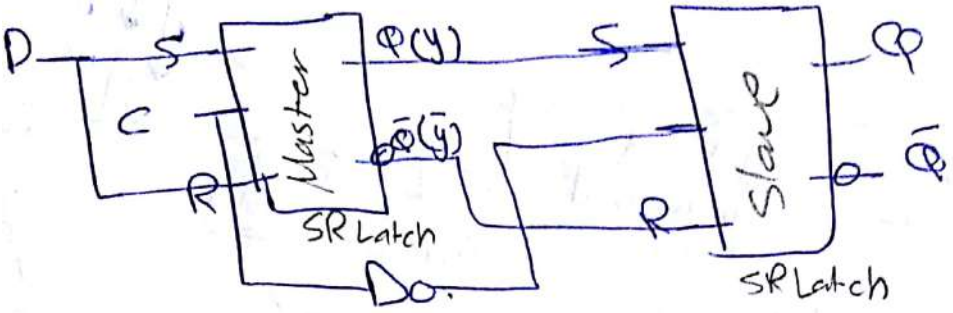


- Edge-Triggered: Dynamic indicator



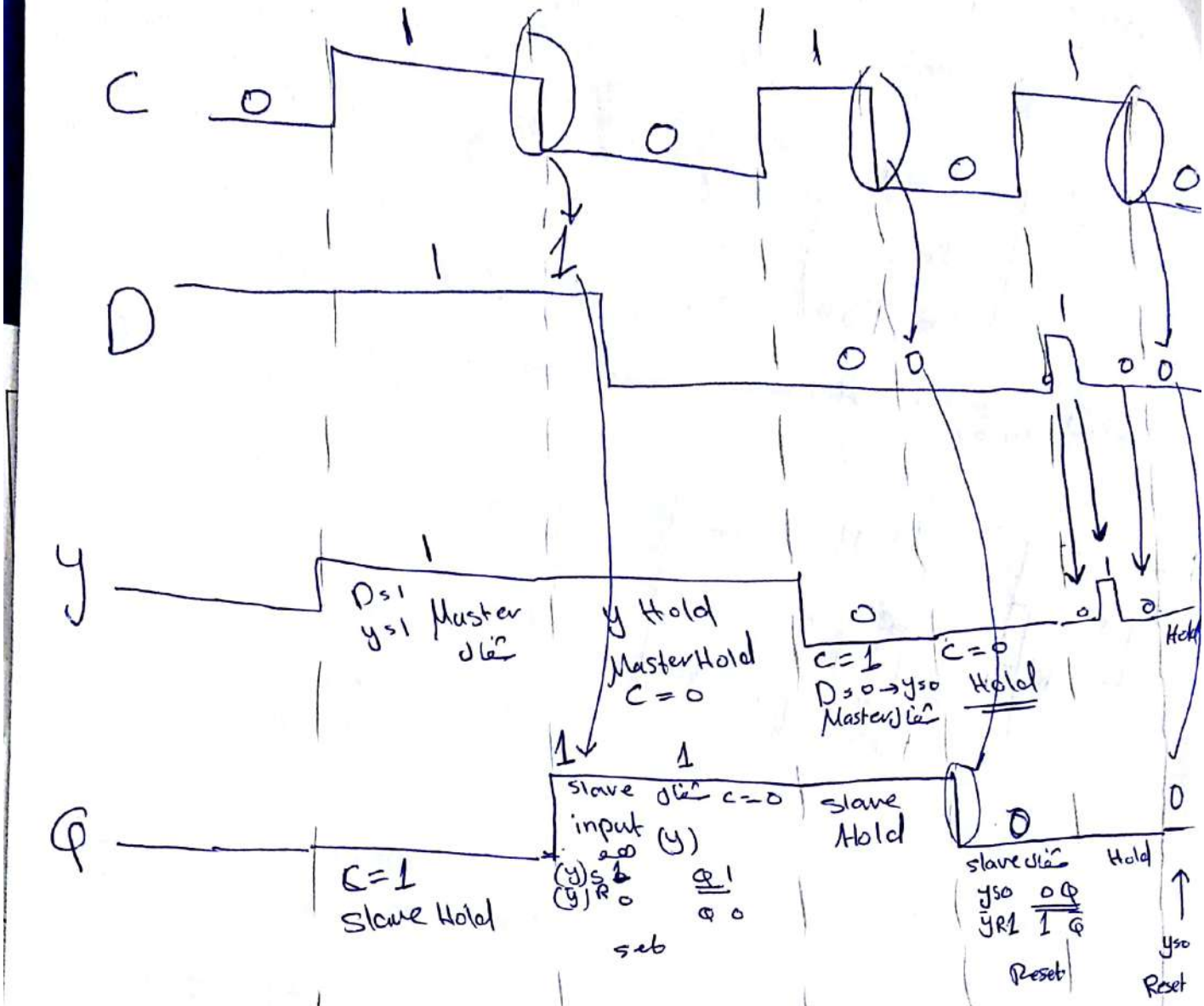


DLatch
SR Latch



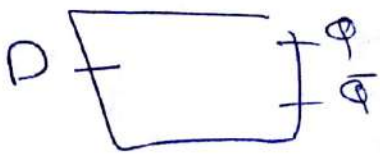
SR Latch
SR Latch

Master slave D.ff



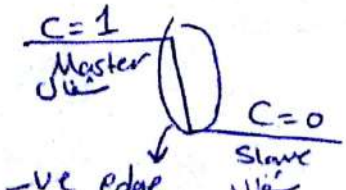
C=1
Master شغال
D=1
y=1

C=0
slave شغال
Master
(Hold)
(y)



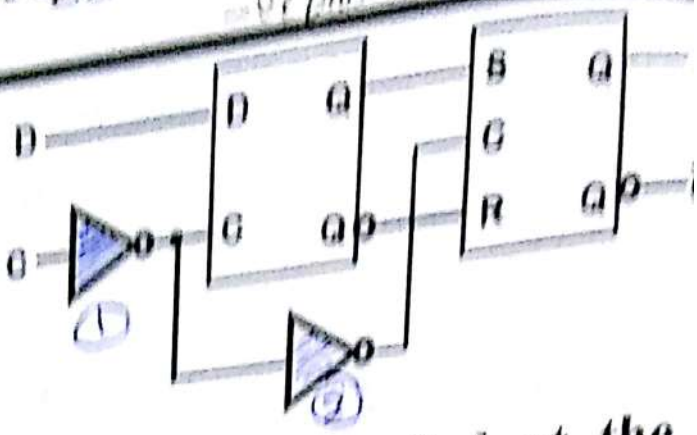
لو الطلع كل -ve edge
وارتبط D بـ $\overline{\Phi}$ مباشرة

طلع معايا
-ve edge



Positive-Edge Triggered D Flip-Flop

- Formed by adding inverter to clock input

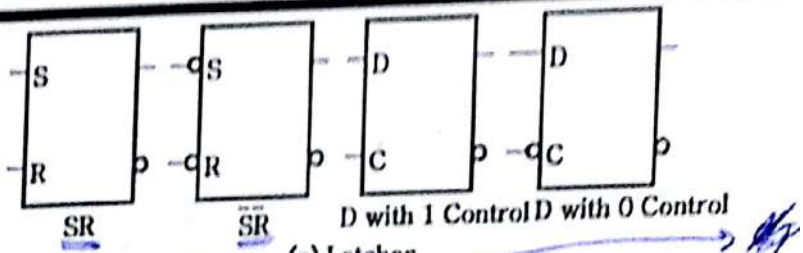


- Q changes to the value on D applied at the positive clock edge

- Our choice as the standard flip-flop for most sequential circuits

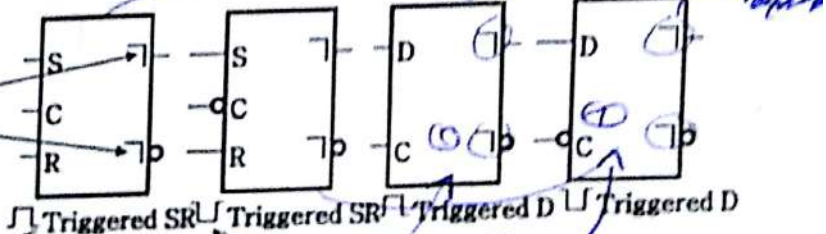
Standard Symbols for Storage Elements

- Latches:



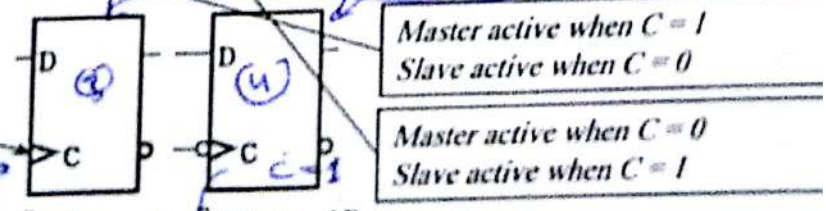
(a) Latches

- Master-Slave: Postponed output indicators



(b) Master-Slave Flip-Flops

- Edge-Triggered: Dynamic indicator

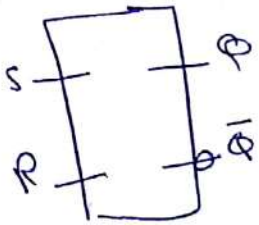


(c) Edge-Triggered Flip-Flops

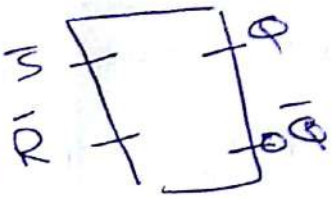
slave
Master

Positive-Edge Triggered D Flip-Fl

Overvie

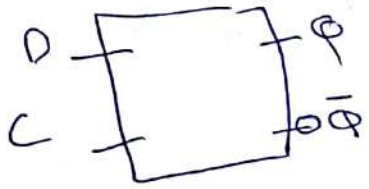


(1) NOR
SR Latch



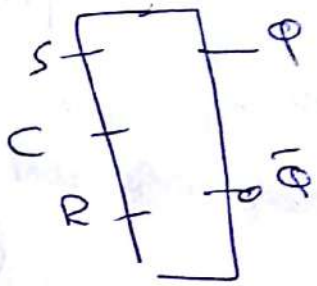
NAND
(2) $S\bar{R}$ latch

(3)

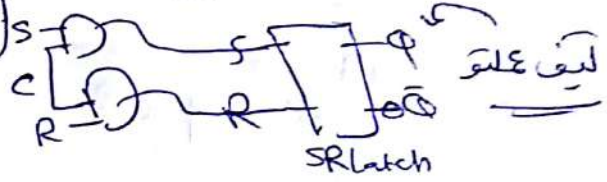


D Latch
1 input
+ve pluse
نبضة
C = 1

(4)

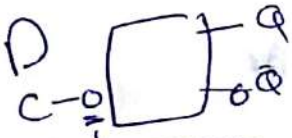


Clocked SR Latch



لبنة على

(5)



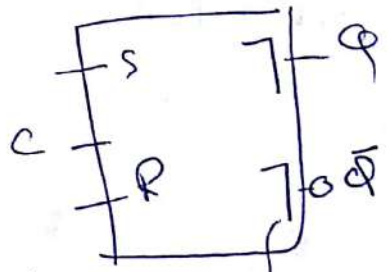
D Latch
-ve pluse
نبضة
C = 0

Latch انواع

Master

Master slave SR FF

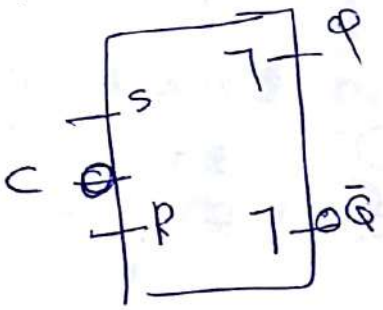
(catching) 1's و 0's کا سبب



Master
C=1 +ve pulse

Slave
C=0 -ve pulse

Latch (Q) کو

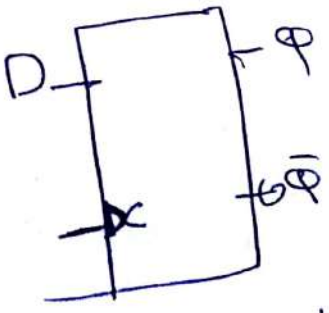


Master
C=0 -ve pulse

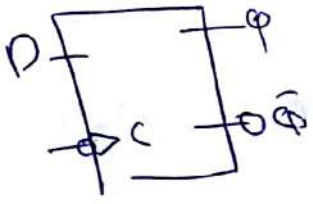
Slave
C=1 +ve pulse

Catching 0's اور 1's

DFF اور Edge → +Positive
↓ - Negative



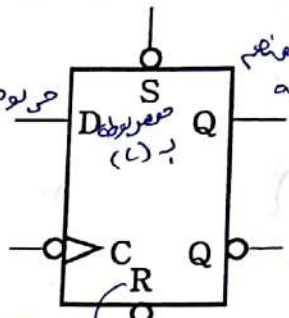
+ve edge



-ve edge

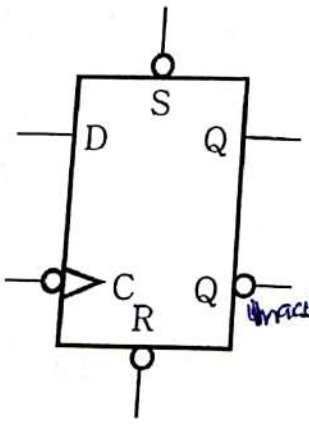
Direct Inputs

- At power up or at reset, all or part of a sequential circuit usually is initialized to a known state before it begins operation
- This initialization is often done outside of the clocked behavior of the circuit, i.e., asynchronously
- Direct R and/or S inputs that control the state of the latches within the flip-flops are used for this initialization
- For the example flip-flop shown
  - 0 applied to R resets the flip-flop to the 0 state
  - 0 applied to S sets the flip-flop to the 1 state



Direct inputs

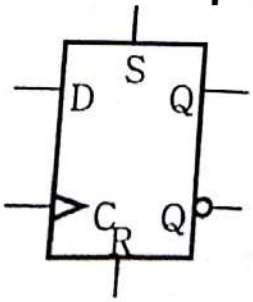
- D flip-flop with active-low direct inputs :



Direct inputs

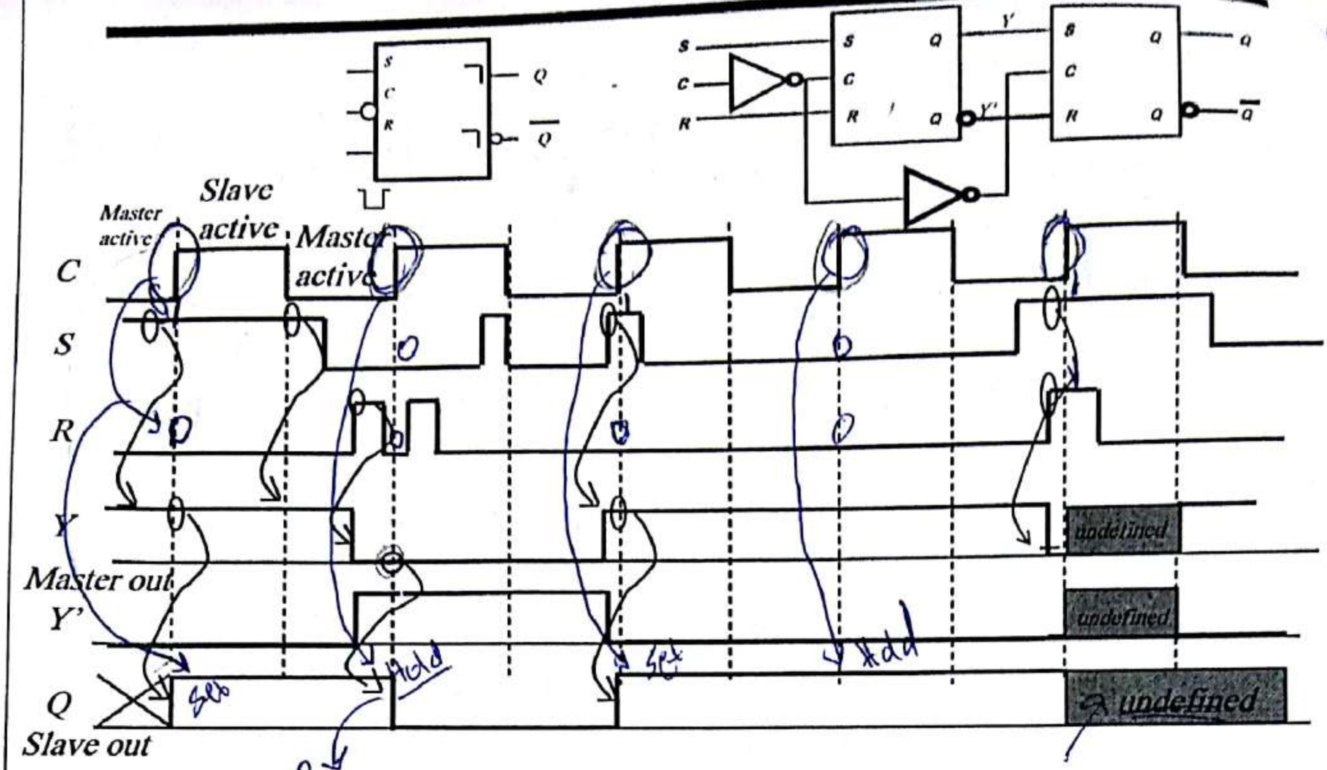
S	R	C	D	Q	Q'
0	1	X	X	1	0
1	0	X	X	0	1
1	1	↓	0	0	1
1	1	↓	1	1	0

- Active high direct inputs:



S	R	C	D	Q	Q'
0	1	X	X	0	1
1	0	X	X	1	0
0	0	↑	0	0	1
0	0	↑	1	1	0

Timing diagram of A SR Master-Slave Flip-Flop



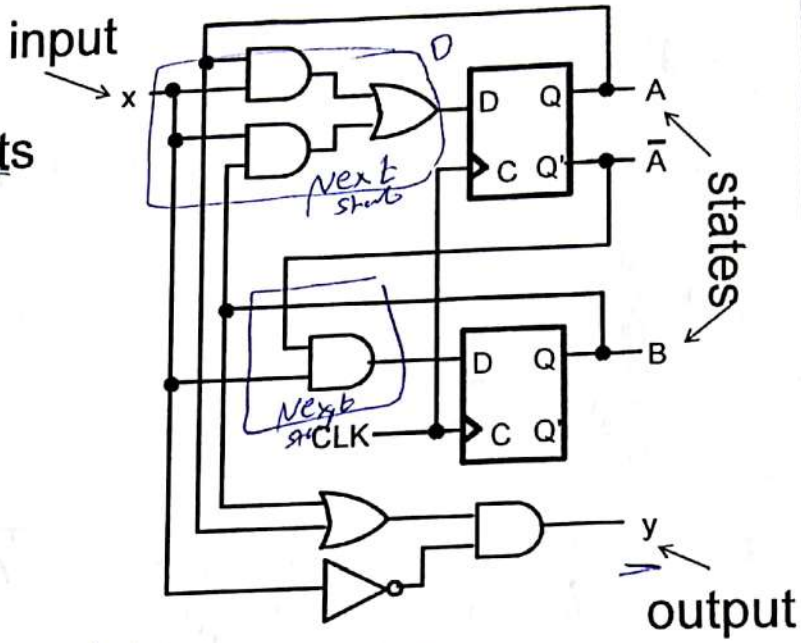
لو اعطاني الرسمة ومكني طلعو 0's + او 1's Catching

* طلع كل SR → Q $\bar{Q}$

* اعرف +ve edge

5-4 Sequential Circuit Analysis

- Consider the following circuit:

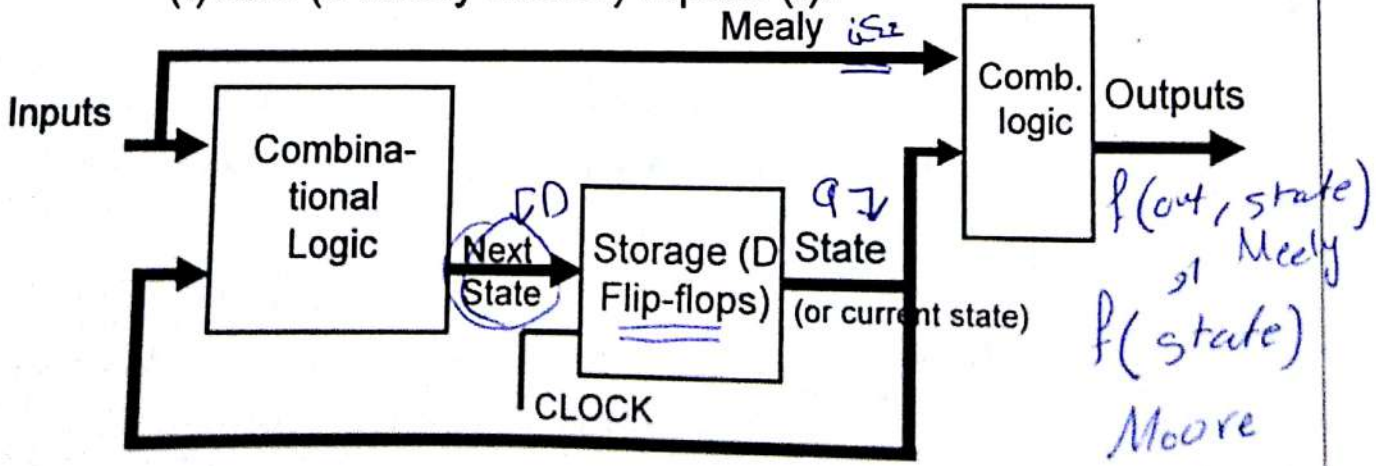


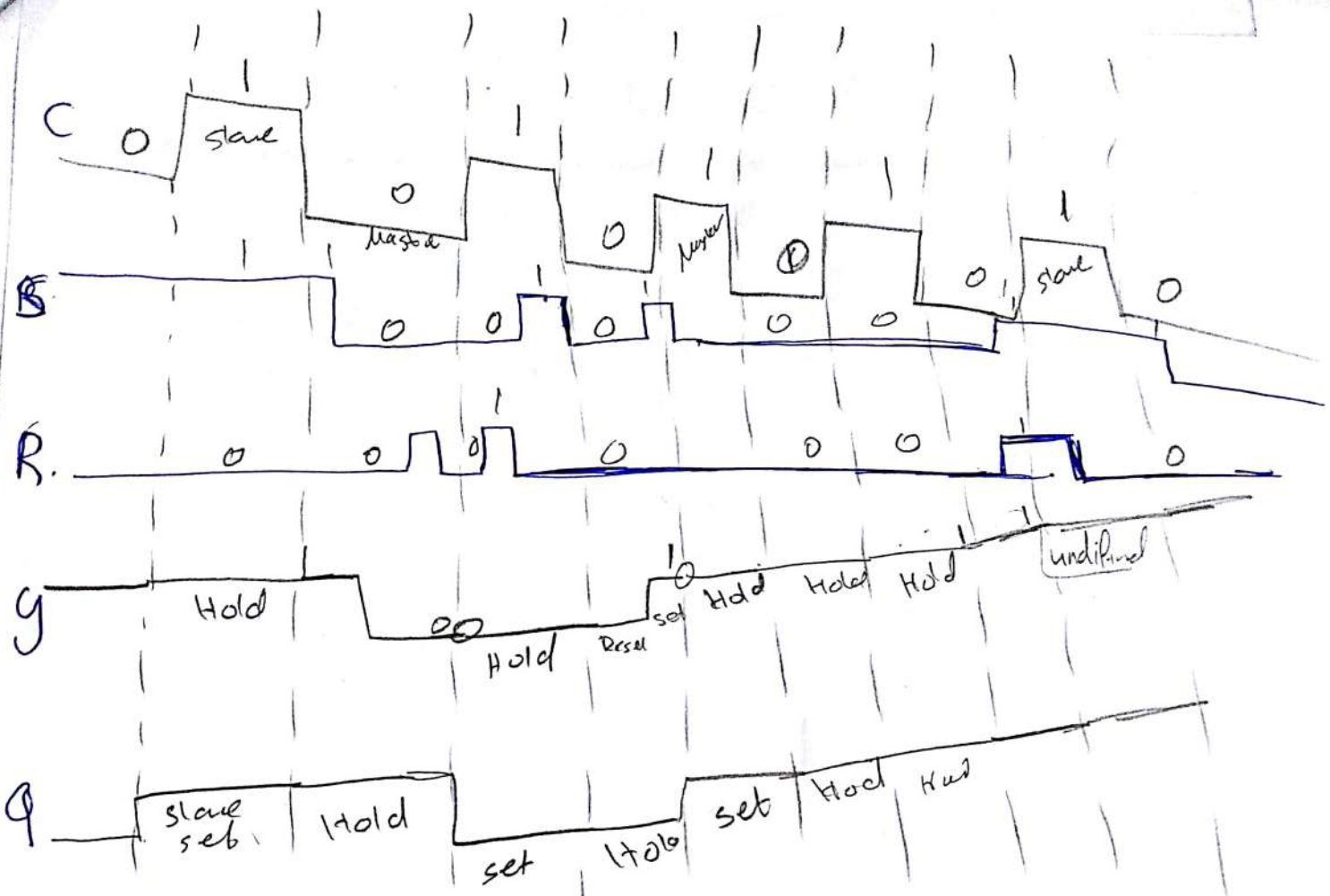
- What does it do?
- How do the outputs change when an input arrives?

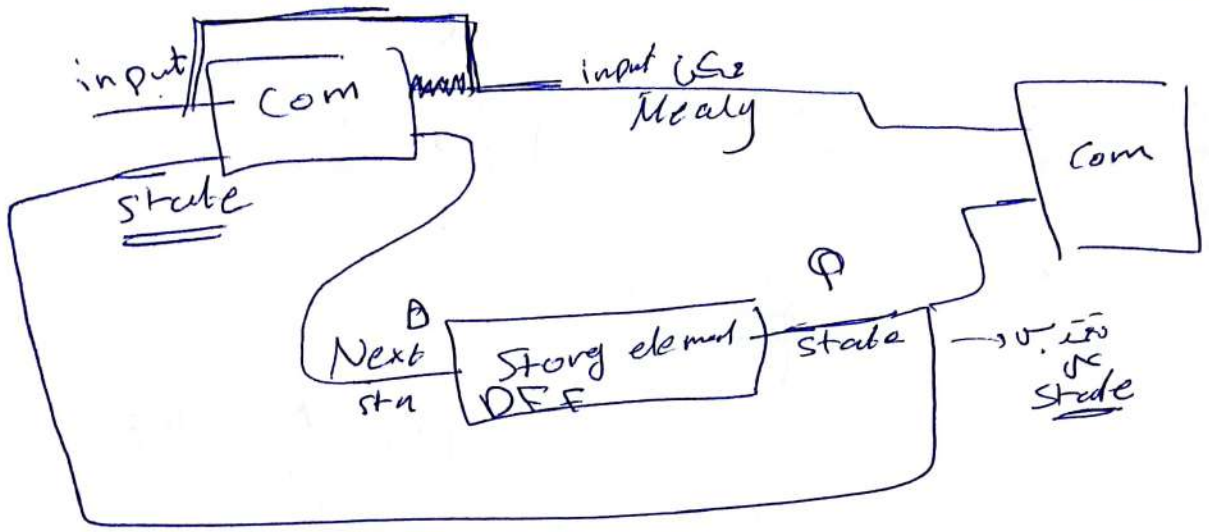
عشان اعرف الف أنا في (sequential circuit) بتتوفر في F F و gat

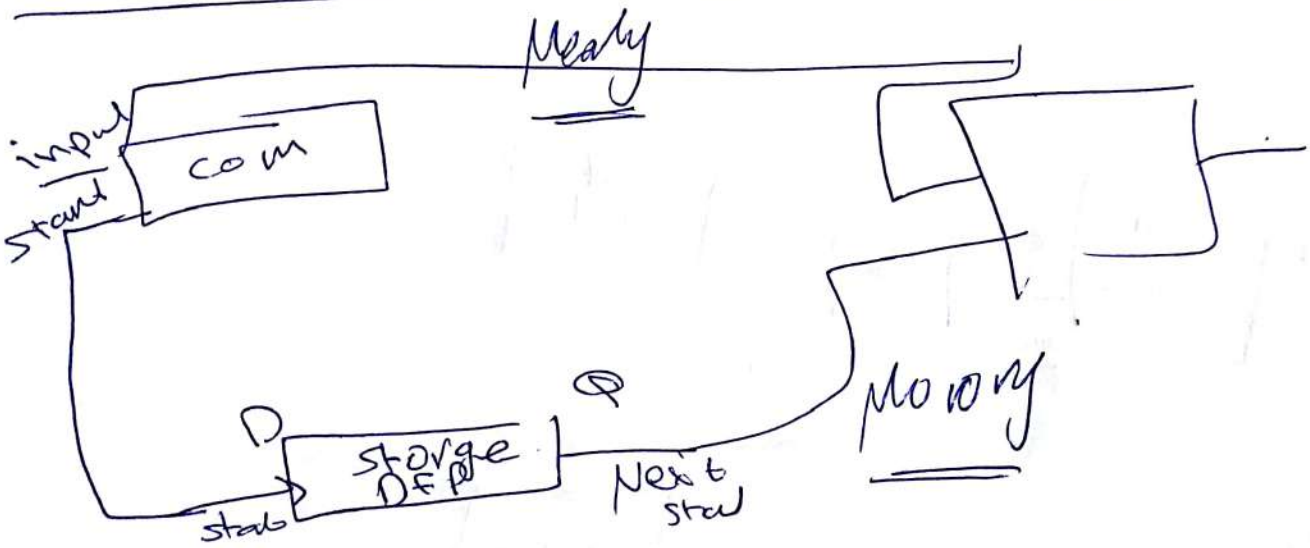
Sequential Circuit Model

- General Model
  - Current or Present State at time (t) is stored in an array of flip-flops.
  - Next State is a Boolean function of State and Inputs.
  - Outputs at time (t) are a Boolean function of State (t) and (if Mealy model) Inputs (t).









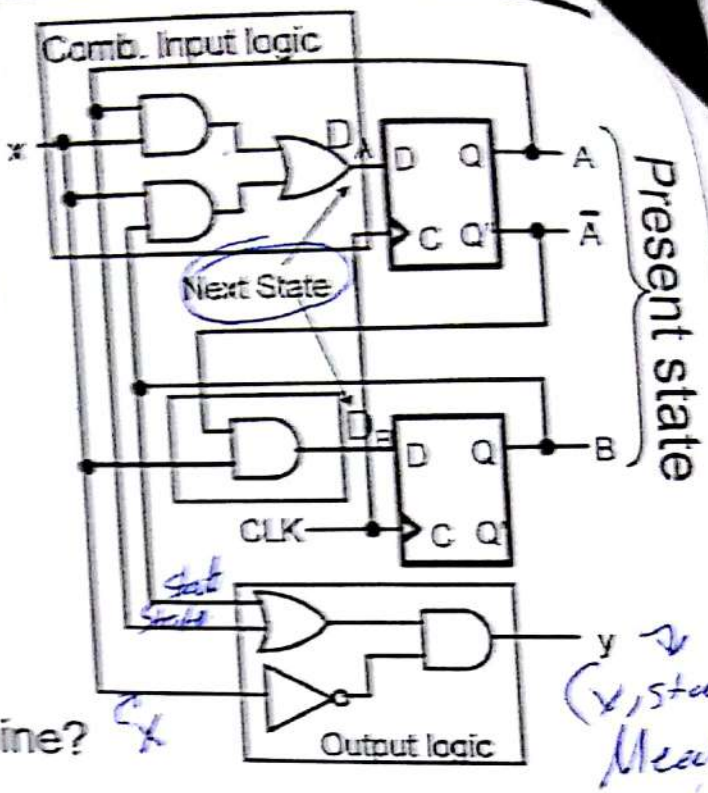
Previous Example (from Fig. 5-15)

- Input: X
- Output: Y
- State: (A(t), B(t))

  Example: (AB)= (01), (10)
- Next State:

  $(D_A(t), D_B(t))$

  $= (A(t+1), B(t+1))$



Is this a Moore or Mealy machine?

31

Steps for Analyzing a Sequential Circuit

1. Find the input equations (D_A, D_B) to the flip-flops (next state equations) and the output equation.
2. Derive the State Table (describes the behavior of a sequential circuit).
3. Draw the State Diagram (graphical description of the behavior of the sequential circuit).
4. Simulation

32

Previous Example (from Fig. 5-15)

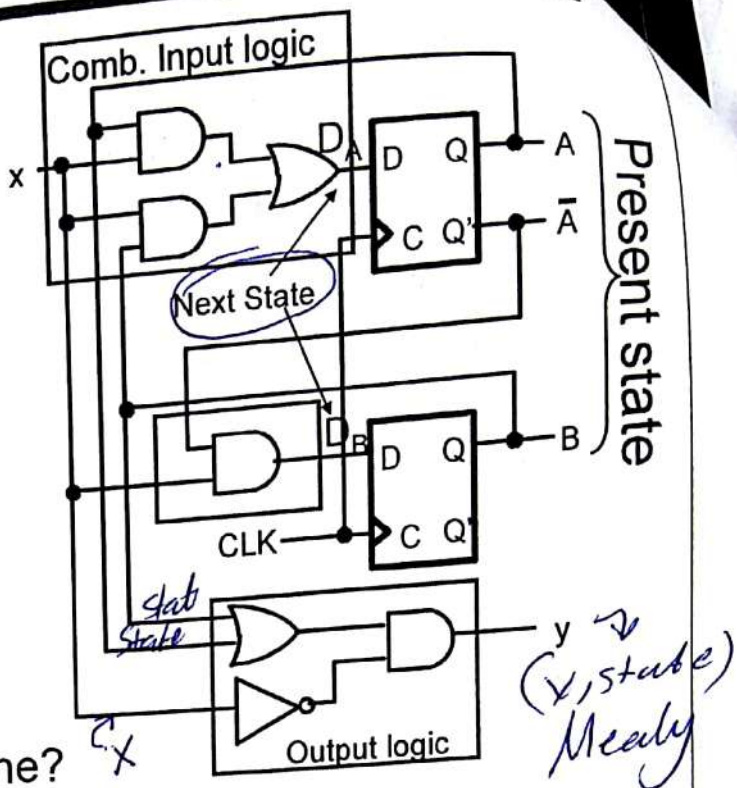
- Input: X
- Output: Y
- State: (A(t), B(t))

Example: (AB)= (01), (10)

- Next State:

$(D_A(t), D_B(t))$

$= (A(t+1), B(t+1))$



Is this a Moore or Mealy machine?

31

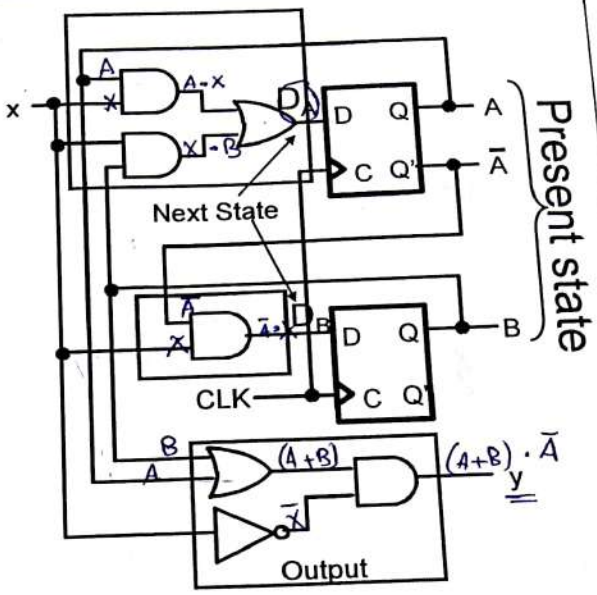
Steps for Analyzing a Sequential Circuit

1. Find the input equations (D_A, D_B) to the flip-flops (next state equations) and the output equation.
2. Derive the State Table (describes the behavior of a sequential circuit).
3. Draw the State Diagram (graphical description of the behavior of the sequential circuit).
4. Simulation

32

Step 1: Input and output equations

- Boolean equations for the inputs to the flip flops:
  - $A(t+1) = D_A = AX + BX$
  - $D_B = \overline{A}X$
- Output Y
  - $Y = \overline{X}(A + B)$
- Also can be written as
  - $A(t+1) = D_A = A(t)X + B(t)X$
  - $B(t+1) = D_B = \overline{A}(t)X$
  - $Y = \overline{X}(A(t) + B(t))$



Step 2: State Table

- The state table: shows what the *next state* and the *output* will be as a function of the present state and the input:

Inputs of the combinational circuit | Outputs of the table

Present State		Input	Next State		Output
A	B	X	D_A	D_B	y
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			

- The State Table can be considered a truth table defining the combinational circuits:
  - the inputs are *Present State* and *Input*,
  - and the outputs are *Next State* and *Output*

State Table For The Example

- For the example: $A(t+1) = A(t)\,x + B(t)\,x$
  $B(t+1) = A'(t)\,x$
  $Y(t) = x'\,(B(t) + A(t))$

Inputs of the table | Outputs of the table

Present State A(t) B(t)	Input x	Next State A(t+1) B(t+1)	Output Y
0 0	0	0 0	0
0 0	1	0 1	0
0 1	0	0 0	1
0 1	1	1 1	0
1 0	0	0 0	1
1 0	1	1 0	0
1 1	0	0 0	1
1 1	1	1 0	0

2^3 rows ($2^{(m+n)}$ rows)

m: no. of flip-flops
n: no. of inputs

35

Alternate State Table

- The previous (1-dimensional table) can become quite lengthy with 2^{m+n} rows (m=no. of flip-flops; n=no. of inputs)
- Alternatively, a 2-dimensional table has the present state in the left column and inputs across the top row
  - $A(t+1) = A(t)\,X + B(t)\,X$
  - $B(t+1) = A'(t)\,X$
  - $Y = X'\,(B(t) + A(t))$

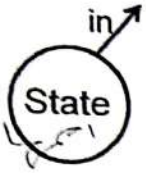
2D

Present State A(t) B(t)	Next State X = 0 A(t+1) B(t+1)	Next State X = 1 A(t+1) B(t+1)	Output X=0 Y	Output X=1 Y
0 0	0 0	0 1	0	0
0 1	0 0	1 1	1	0
1 0	0 0	1 0	1	0
1 1	0 0	1 0	1	0

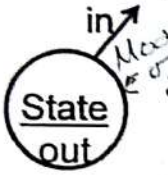
2^m

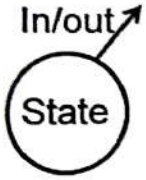
36

- The sequential circuit function can be represented in graphical form as a state diagram with the following components:



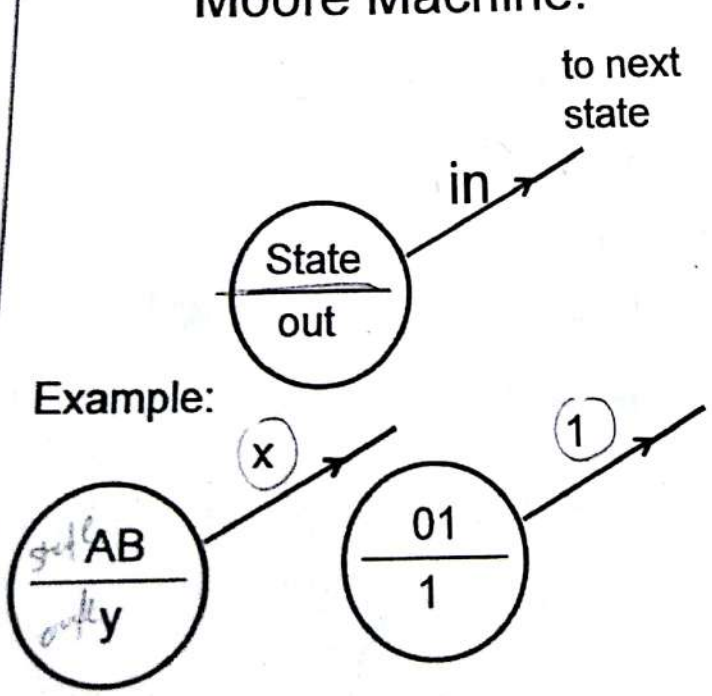
  - A circle with the state name in it for each state
  - A directed arc from the Present State to the Next State for each state transition
  - A label on each directed arc with the Input values which causes the state transition, and
  - A label:
    - In each circle with the output value produced, or
    - On each directed arc with the output value produced.





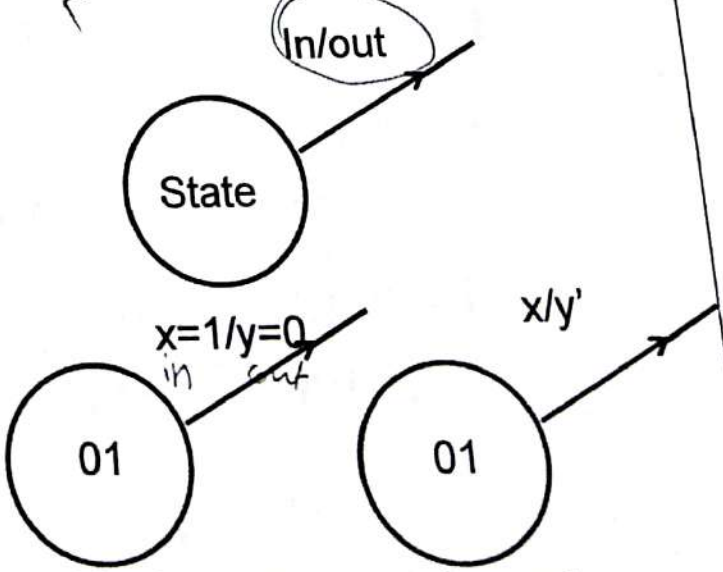
State Diagram Convention

Moore Machine:



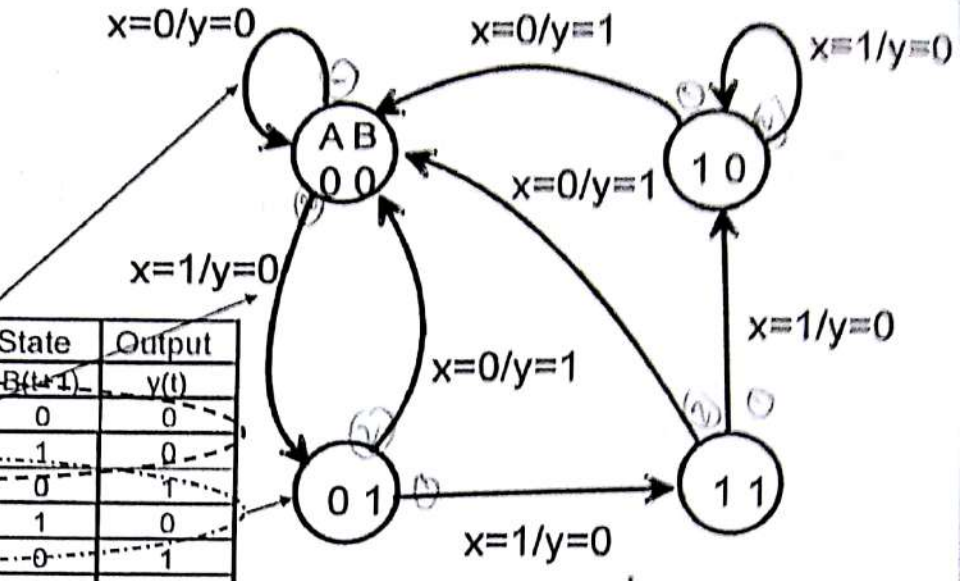
Moore type output depends only on state

Mealy Machine:



Mealy type output depends on state and input

- Graphical representation of the state table:



Present State		Input	Next State		Output
A(t)	B(t)	x(t)	A(t+1)	B(t+1)	y(t)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

Step 4: Simulation

① input
x y
output
Z
state A
Next state D_A

② input + output eq

input eqation

$D_A = (x \oplus y) \oplus A$

output eqation

$Z = A$

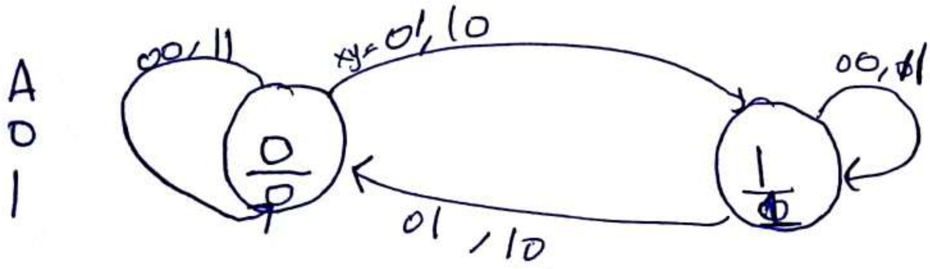
Output funtion for state just ∴ Moore

$0 \oplus 0 = 0$ متشابهة zero
$0 \oplus 1 = 1$ 1 مختلف

present state A	input x	input y	Next state D_A	output Z
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

نفس

Moory output جوا الدائرة



input x

output y

state

A

B

Next state

DA

DB

input + output equation

input equation

$D_A = (X \cdot \bar{A}) \oplus B$

$D_B = A \oplus B$

$y = \bar{B} \cdot \bar{A} \cdot \bar{X}$

$y = \bar{\bar{B}} + \bar{\bar{A}} + \bar{\bar{X}}$

$y = B + A + X$

متشابه

1 غلط

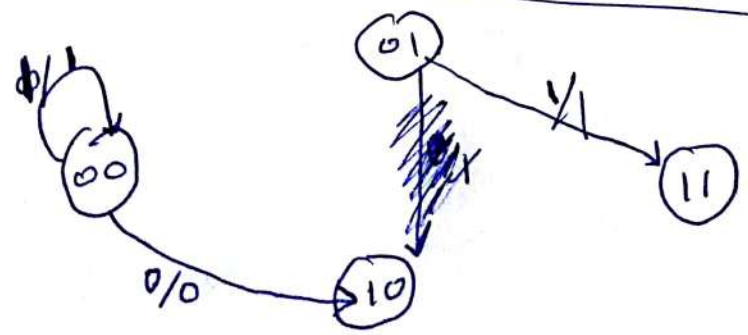
Mealy

present state		input	Next stat		output
A	B	X	DA	DB	y
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	0	1	0	0	1
1	1	0	0	0	1
1	1	1	0	0	1

input/output

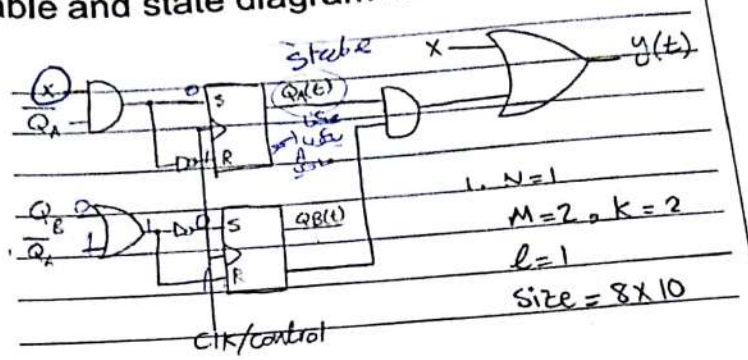
State

A	B
0	0
0	1
1	0
1	1



Example4

- Derive the state table and state diagram for the sequential circuit:



Input
x
output
y

state
Q_A Q_B

Input equation
① $S_A = x\bar{Q}_A$
② $R_A = \bar{x} + Q_A$
③ $S_B = \bar{Q}_B \cdot Q_A$
④ $S_B = Q_B + \bar{Q}_A$

equation output Master slave
$x + (Q_A \cdot Q_B)$

45

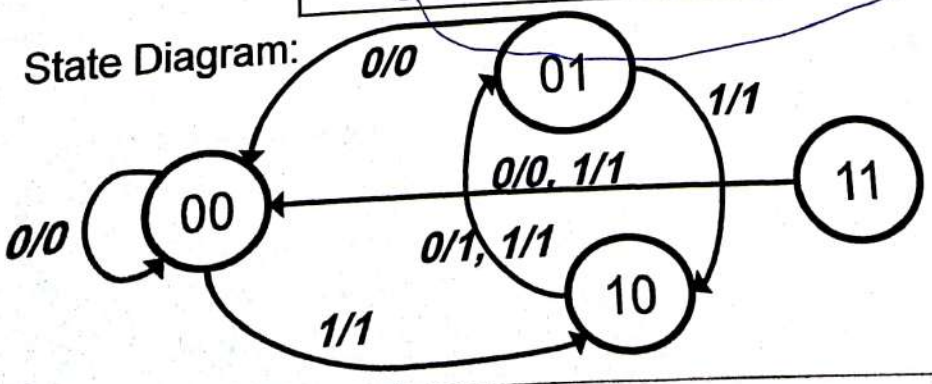
Example4 Cont.

SR FF

- State Table

Present State Q_A Q_B	Input X	S_A R_A	S_B R_B	Next State $Q_A(t+1)$ $Q_B(t+1)$	Output Y
0 0	0	0 1	0 1	Reset 0 0	0
0 0	1	1 0	0 1	set 1 0	1
0 1	0	0 1	0 1	Reset 0 0	0
0 1	1	1 0	0 1	set 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	Reset 0 0	0
1 1	1	0 1	0 1	0 0	1

- State Diagram:

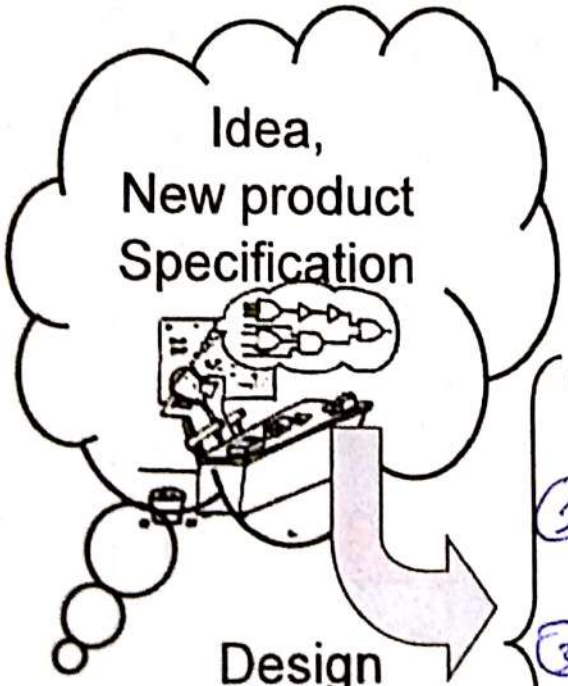


Q_A Q_B
0 0
0 1
1 0
1 1

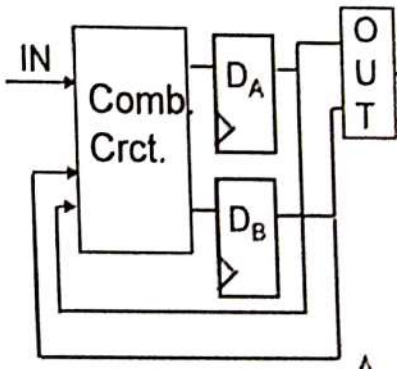
0/0, 1/1

46

5-5 Sequential Circuit Design



?



- Word description
1. State Diagram
2. State Table
   - *State encoding*
3. Select type of Flip-flop
4. Input equations to FF, output eq.
- Verification

Design procedure

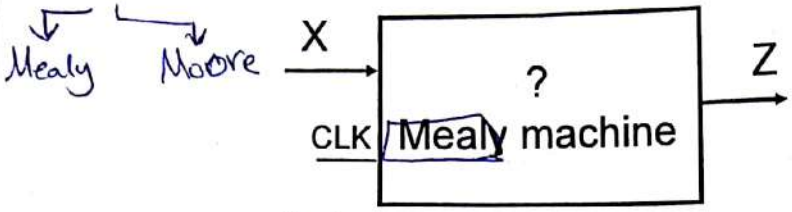
هون بدي اكمل وبزاين اكعلوان بالعكس

- Component Forms of Specification
  - Written description
  - Mathematical description
    - Hardware description language
    - Tabular description
    - Equation description
    - Diagram describing operation (not just structure)

Formulation: Finding a State Diagram

- In specifying a circuit, we use states to remember meaningful properties of past input sequences that are essential to predicting future output values.
- As an example, a sequence recognizer is a sequential circuit that produces a distinct output value whenever a prescribed pattern of input symbols occur in sequence, i.e, recognizes an input sequence occurrence.
- Next, the state diagram, will be converted to a state table from which the circuit will be designed.

Sequence Detector Example: 1101



Input X: 0011100110101101101001111010111

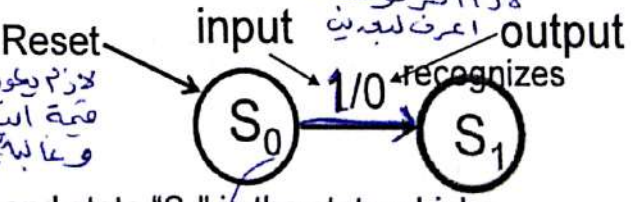
Output Z: 0000000000010000100000000100

Overlapping sequences are allowed

59

Step2: Finding A State Diagram

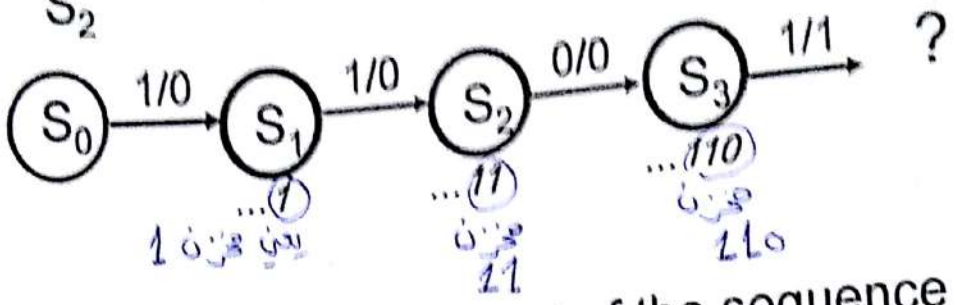
- Define states for the sequence to be recognized:
  - assuming it starts with first symbol X=1,
  - continues through the right sequence to be recognized, and
  - uses output 1 to mean the full sequence has occurred,
  - with output 0 otherwise.
- Starting in the initial state (named "S_0"):
  - Add a state that the first "1."
  - State "S_0" is the initial state, and state "S_1" is the state which represents the fact that the "first" one in the input subsequence has occurred. The first "1" occurred while being in state S_0 during the clock edge.



60

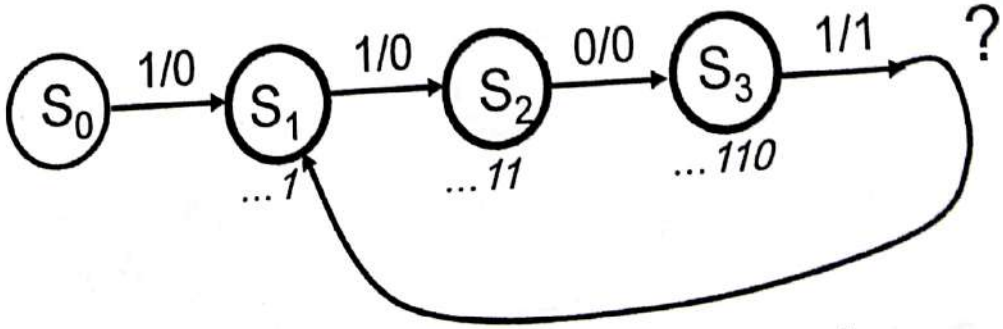
Finding a State Diagram(cont.)

- Assume that the 2nd 1 arrives of the sequence 1101: needs to be remembered: add a state S_2

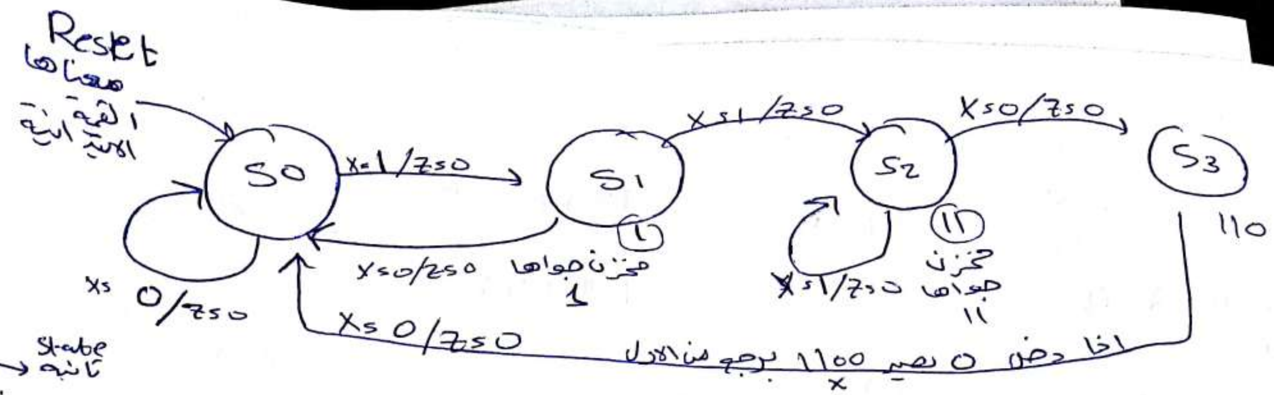


- Next, a "0" arrives: part of the sequence 1101 that needs to be remembered; add state S_3
- The next input is "1" which is part of the right sequence 1101; now output Z=1

Completing The State Diagram



- Where does the final arrow go to:
  - The final 1 of the sequence 1101 can be the beginning of another sequence; thus the arrow should go to state S_1



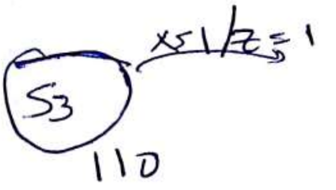
stabe ثانيه

اذا دخل 1 لازم اخزنو في
واذا دخل 0 مو لازم اخزنو
1101 ← لانه الحدث صفر صفر
اول 1

S1
اذا دخل عليها zero
لازم يرجع S0
لانه صار صفر صفر
1101
ثاني (1)
اذا دخل (1) لازم
روح على state جديدة

S2
x=0 لازم
اخزنها لانه الحدث
110
ثالث 0

S2
x=1
يرجع على
نفس state
لانه اصلا
راح يخزن
11
نفس
اعني



يروح
على
S1

لانه يصير
110(1)
ممكن يكون بداية
عشان مسموح overlab

أكمل

مثلاً دا عاوز انا اختارهم

Perest State (A B)	input (X)	Naxtstat (D_A D_B)	Z
S_0 (00)	0	S_0 (00)	0
S_0 (00)	1	S_1 (01)	0
S_1 (01)	0	S_0 (00)	0
S_1 (01)	1	S_2 (10)	0
S_2 (10)	0	S_3 (11)	0
S_2 (10)	1	S_2 (10)	0
S_3 (11)	0	S_0 (00)	0
S_3 (11)	1	S_1 (01)	1

4 — لازم لكل state
بابنارى كود لميزها
عن غيرها

عدد bit ← 2

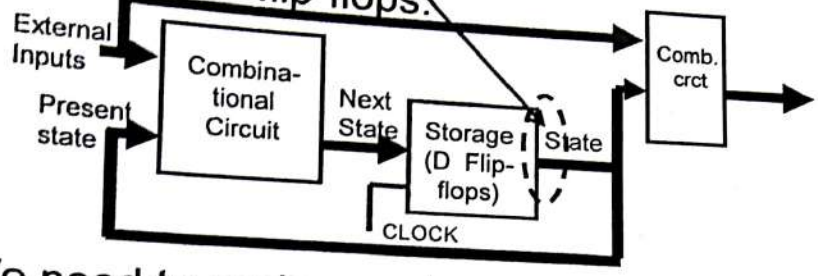
state encoding (counting order) → مراجعه سلايد 66

$S_0 = 00$
$S_1 = 01$
$S_2 = 10$
$S_3 = 11$

نعدين نروح على state ونحط مكان S_0 00
تابل
مكان present state و Next state

State Assignment

- Right now States have names such as S_0, S_1, S_2 and S_3
- In actuality these state need to be represented by the outputs of the flip-flops.



- We need to assign each state to a certain output combination AB of the flip-flops:
  - e.g. State S_0=00, S_1=01, S_2=10, S_3=11
  - Other combinations are possible: S_0=00, S_1=10, S_2=11, S_3=01

Popular State Assignments

State. ترتيب انواع

- 1. Counting order assignment:
  - 00, 01, 10, 11
- 2. Gray code assignment:
  - 00, 01, 11, 10
- 3. One-hot state assignment
  - 0001, 0010, 0100, 1000
- Does state assignment make a *difference* in cost?

order

D_A

A \ B	0	1	3	2
0	0	0	0	0
4	1	5	7	6
1	0	0	0	0

X

$D_A = \bar{A}BX + A\bar{B}$

G عدد الخلايا المجاورة

G = 7

D_B

A \ B	0	1	3	2
0	0	1	0	0
4	1	0	1	0
1	0	0	0	0

X

$D_B = \bar{A}\bar{B}X + A\bar{B}\bar{X} + ABX$

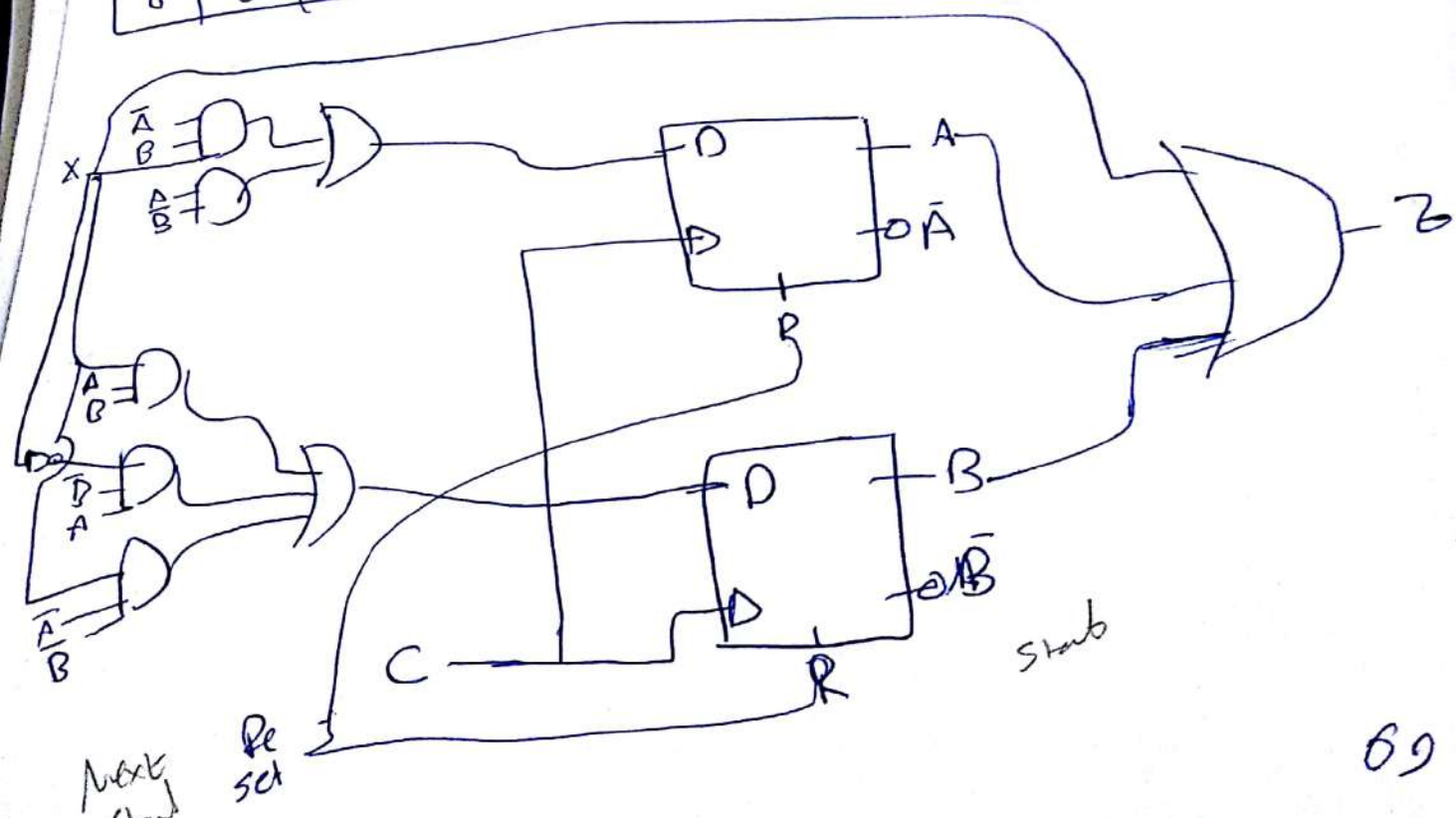
G = 12

Z

0	0	0	0
0	0	1	0

$Z = AB\bar{X}$

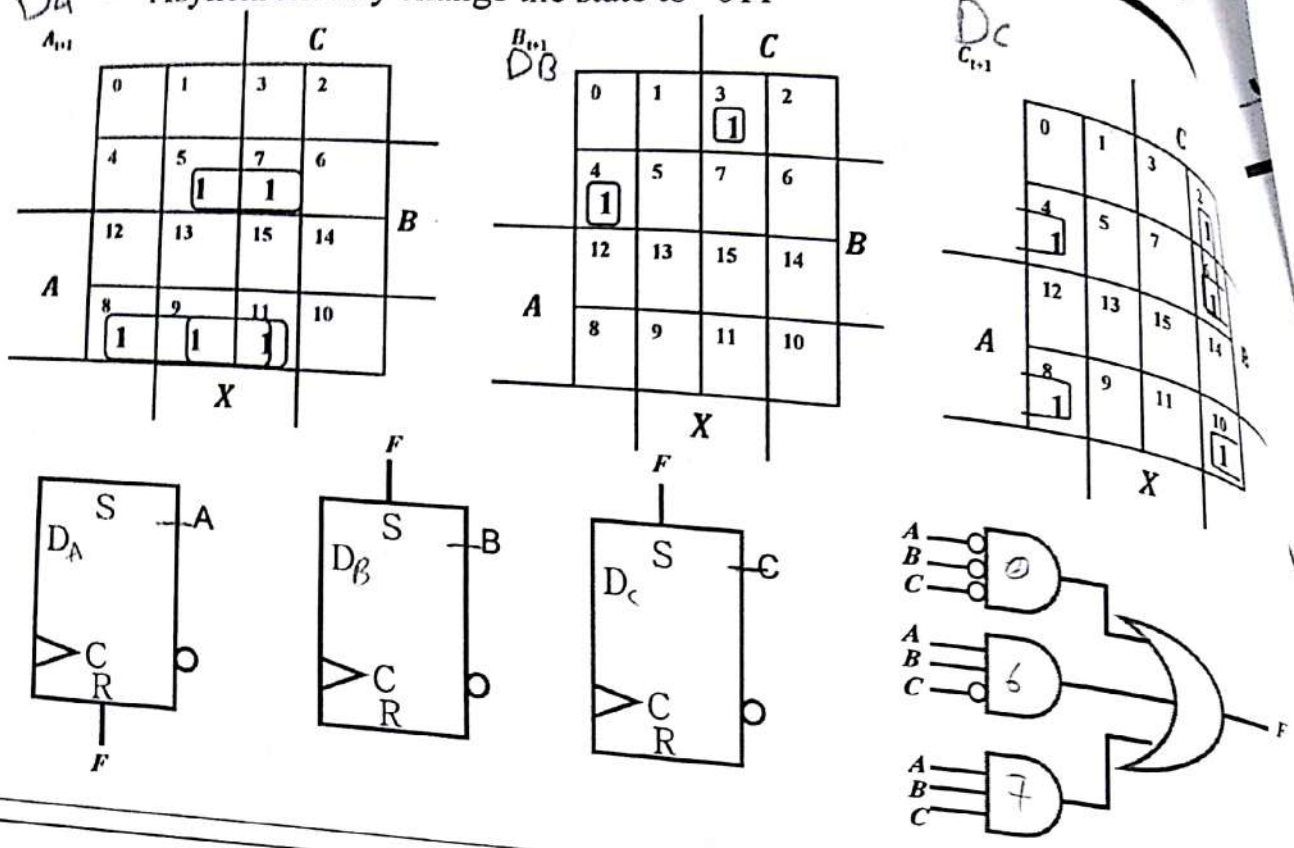
G = 3



69

Solution

- Asynchronously change the state to "011"

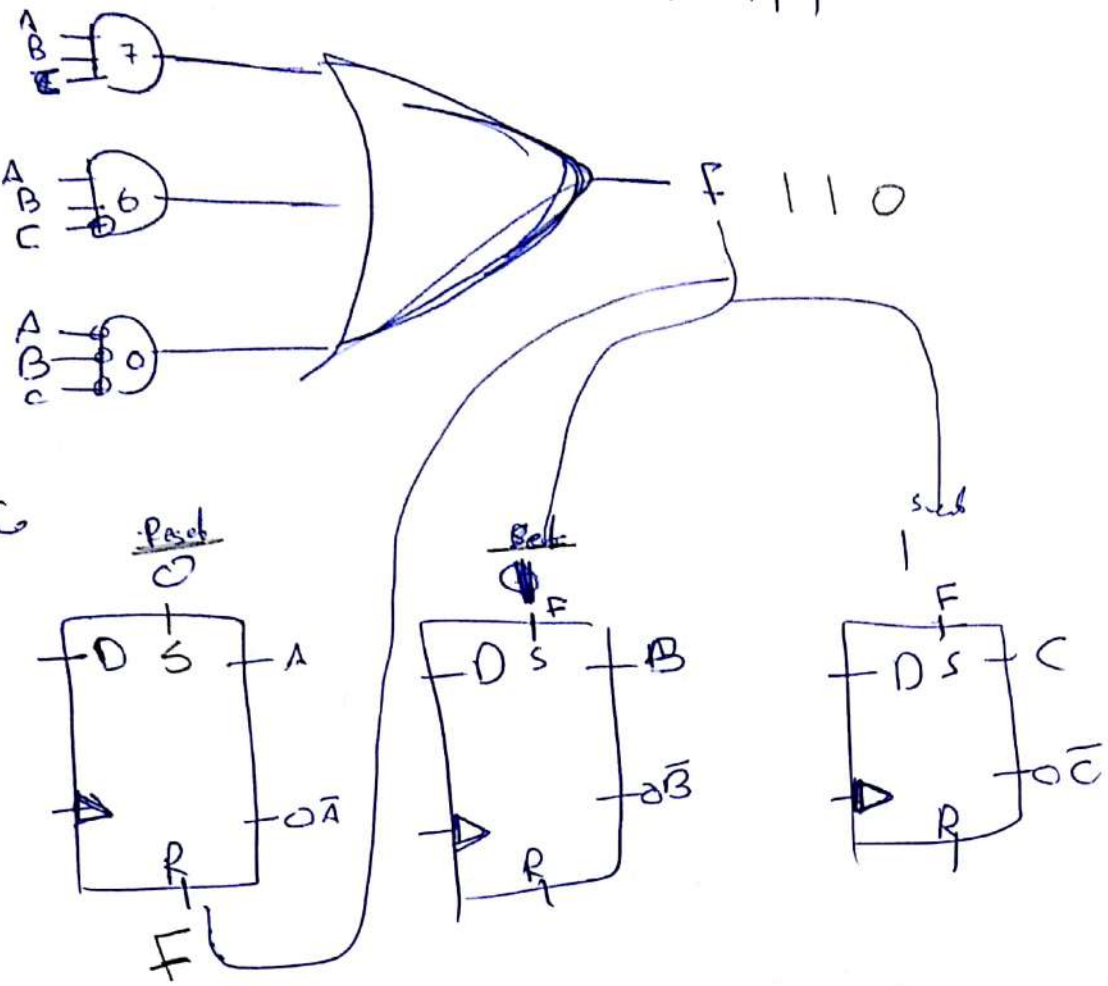


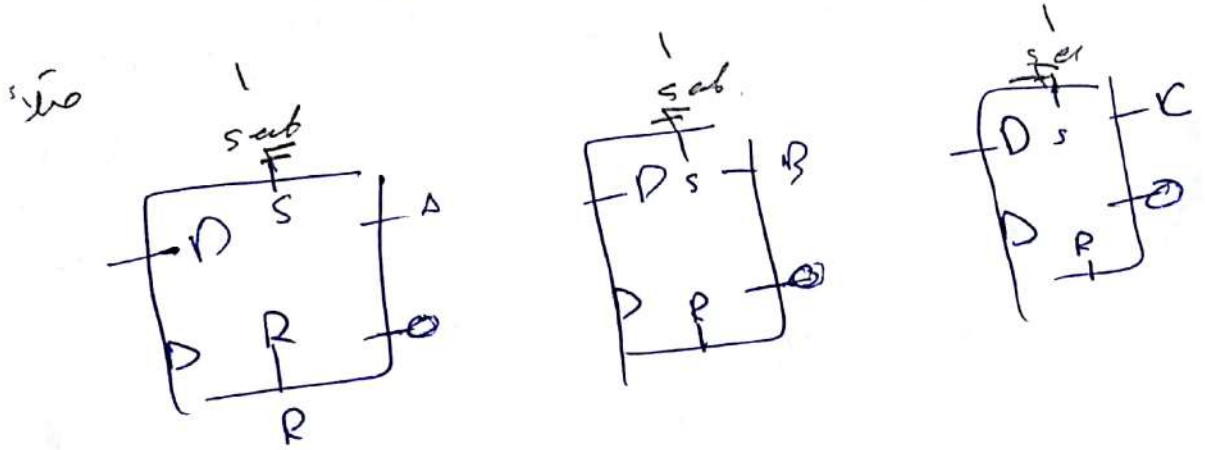
5-6 Other Flip-Flop Types

- J-K and T flip-flops
  - Behavior
  - Implementation
- Basic descriptors for understanding and using different flip-flop types
  - Characteristic tables
    - Defines the next state as a function of the present state and input
  - Characteristic equations
  - Excitation tables

+ve pluse

unused state 000, 110, 111

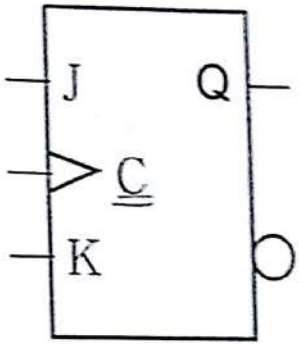




F الى 1 اسبك
Set على

J-K Flip-flop

- Behavior of JK flip-flop:
  - Same as S-R flip-flop with J analogous to S and K analogous to R
  - Except that J = K = 1 is allowed, and
  - For J = K = 1, the flip-flop changes to the *opposite state* (toggle)
- Behavior described by the characteristic table (function table):



J	K	Q(t+1)	
0	0	Q(t)	no change
0	1	0	reset
1	0	1	set
1	1	$\overline{Q(t)}$	toggle

Design of an edge-triggered J-K Flip-Flop

State table of a JK FF:

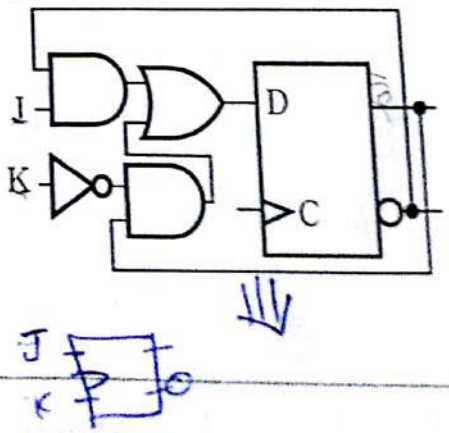
Present state Q	Inputs J	K	Next state Q(t+1)	
0	0	0	0	Hold
0	0	1	0	Reset
0	1	0	1	set
0	1	1	1	toggle $\overline{Q}$
1	0	0	1	Hold
1	0	1	0	Reset
1	1	0	1	set
1	1	1	0	toggle

$Q(t+1) = D_A$

Q \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

$$Q(t+1) = D_A = J\overline{Q} + \overline{K}Q$$

Called the characteristic equation



Q(t)	Q(t+1)	J	K	Operation
0	0	0	X	No change
0	1	1	X	Set
1	0	X	1	Reset
1	1	X	0	No Change

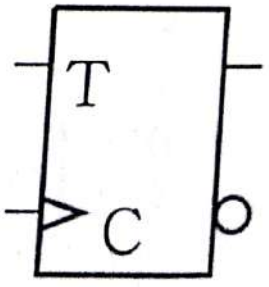
T Flip-Flop

- Behavior described by its characteristic table:
  - Has a single input T
    - For T = 0, no change to state
    - For T = 1, changes to opposite state
- Same as a J-K flip-flop with J = K = T

T	Q(t+1)
0	Q(t) no change
1	$\overline{Q}$(t) complement

Characteristic equation:

$$Q(t+1)=T'Q(t) + TQ'(t)$$
$$= T \oplus Q(t)$$



T Flip-Flop Excitation Table

Q(t +1)	T	Operation
$Q(t)$	0	No change Hold
$\overline{Q}(t)$	1	Complement Toggle

For analysis

- *Characteristic equation* - defines the next state of the flip-flop as a Boolean function of the flip-flop inputs and the current state.
- *Excitation table* - defines the flip-flop input variable values as function of the current state and next state. In other words, the table tells us what input is needed to cause a transition from the current state to a specific next state.

For design

D Flip-Flop Descriptors

- Characteristic Table

D	Q(t+1)	Operation
0	0	Reset
1	1	Set

- Characteristic Equation

$$Q(t+1) = D$$

- Excitation Table

Q(t+1)	D	Operation
0	0	Reset
1	1	Set

- Characteristic Table

S	R	Q(t+1)	Operation
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	?	Undefined

- Characteristic Equation

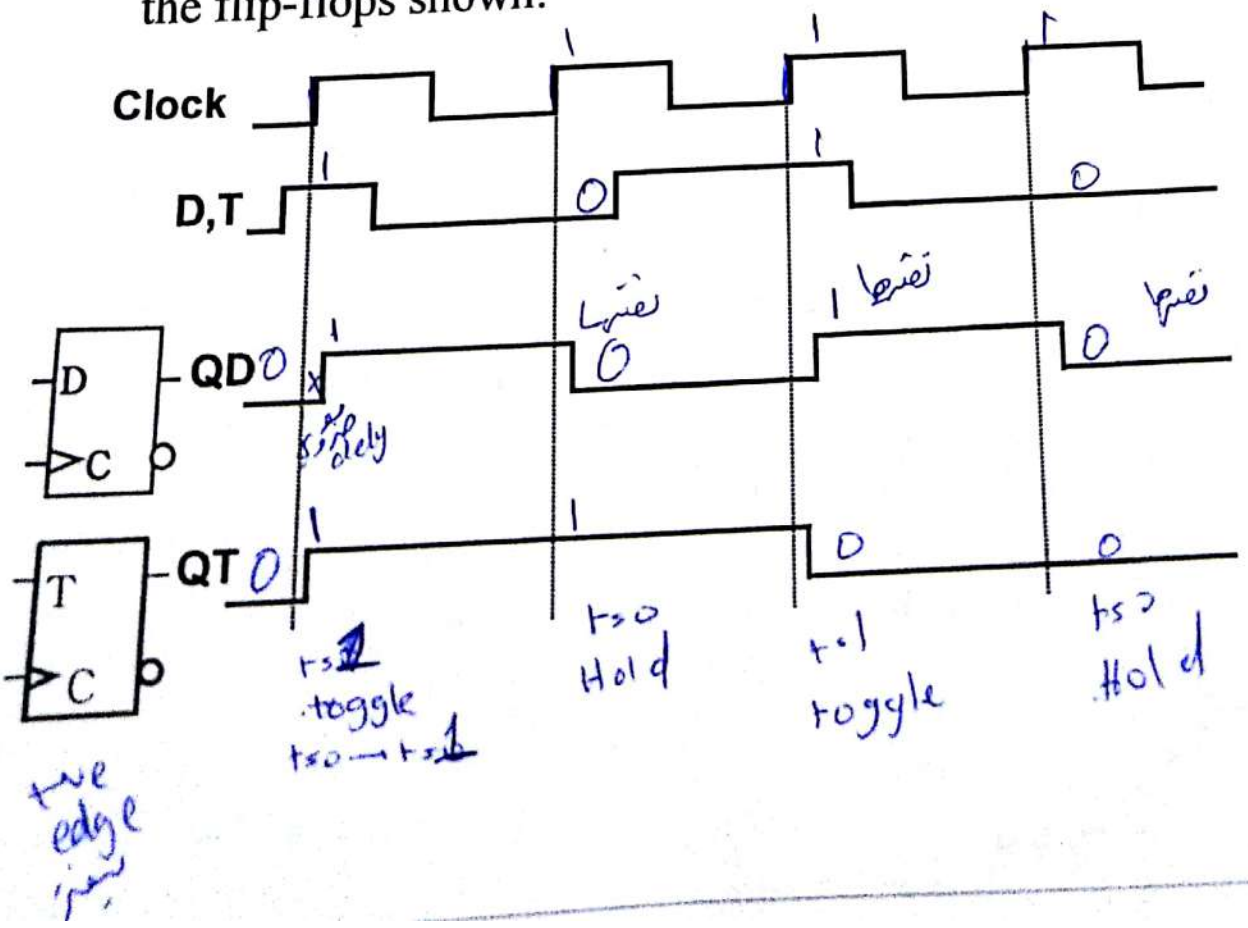
$$Q(t+1) = S + \overline{R}\,Q, \quad S \cdot R = 0$$

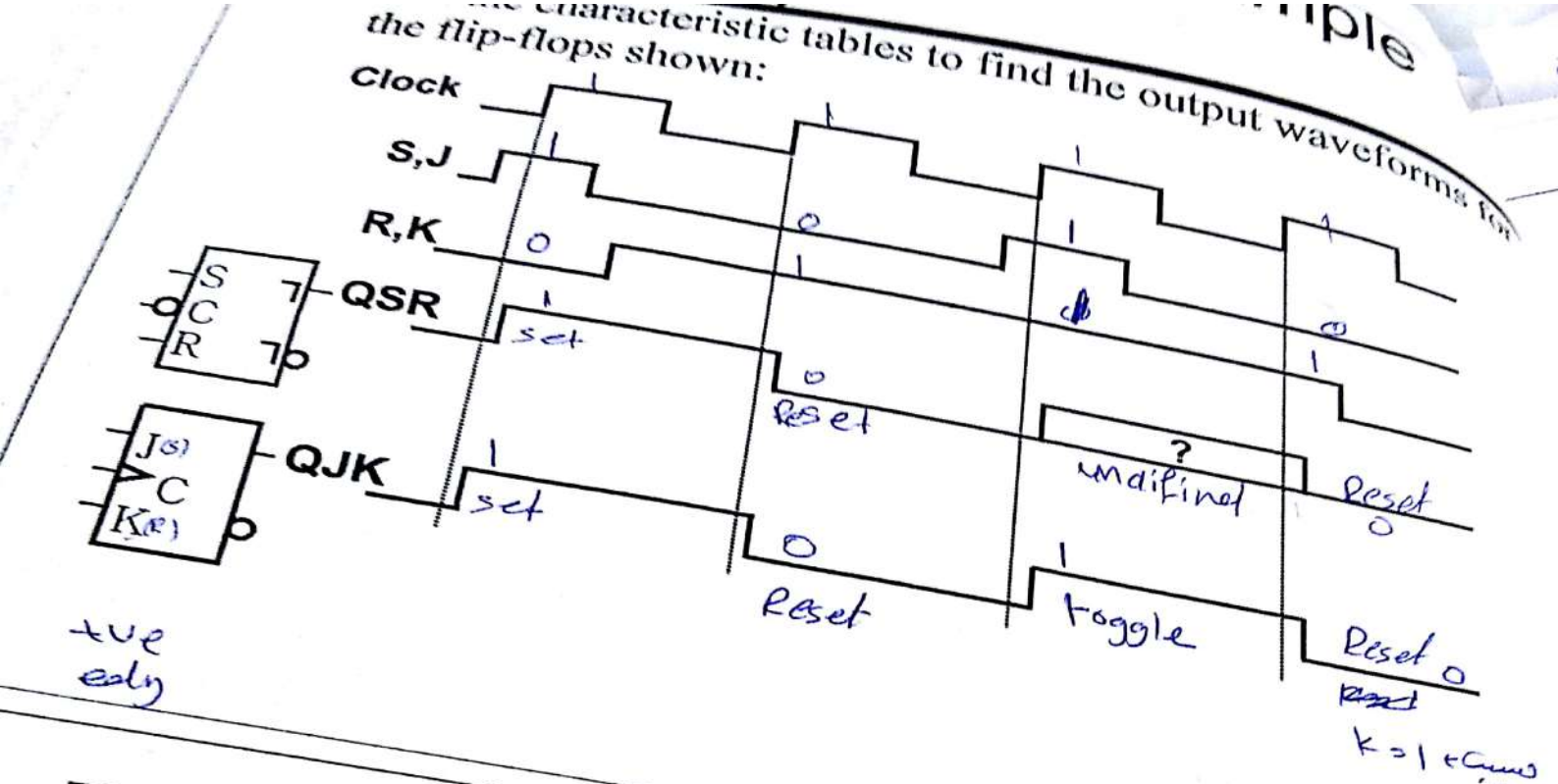
- Excitation Table

Q(t)	Q(t+1)	S	R	Operation
0	0	0	X	No change
0	1	1	0	Set
1	0	0	1	Reset
1	1	X	0	No change

Flip-flop Behavior Example

- Use the characteristic tables to find the output waveforms for the flip-flops shown:





Exercise: Find State Diagram