POWER UNIT

DIGITAL LOGIC

WALEED DWEIK

Power Unit - ju

http://powerunit-ju.com

DIGITAL & COMPUTER SYSTEMS - Digital System

- Takes a set of *discrete* information *inputs* and discrete internal information (*system state*) and generates a set of *discrete* information *outputs*.

  منفصلة. مدخلات. مخرجات. حالة للنظام تطبق على المدخلات.

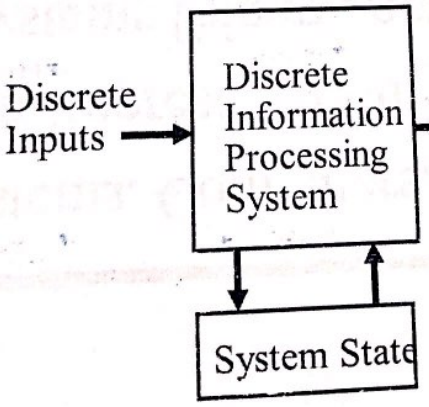
- Digits (Latin word for fingers) : Discrete numeric elements
- Logic : Circuits that operate on a set of two elements with values 0 (False), 1 (True)
- ***Computers are digital logic circuits***

systems :

Digital: (رقمي) — analog: (تناظري)

(توافقية) combinational. — (تسلسلي) sequential.

chapter (3 & 4). — chapter (5, 7).



Types of Digital Systems

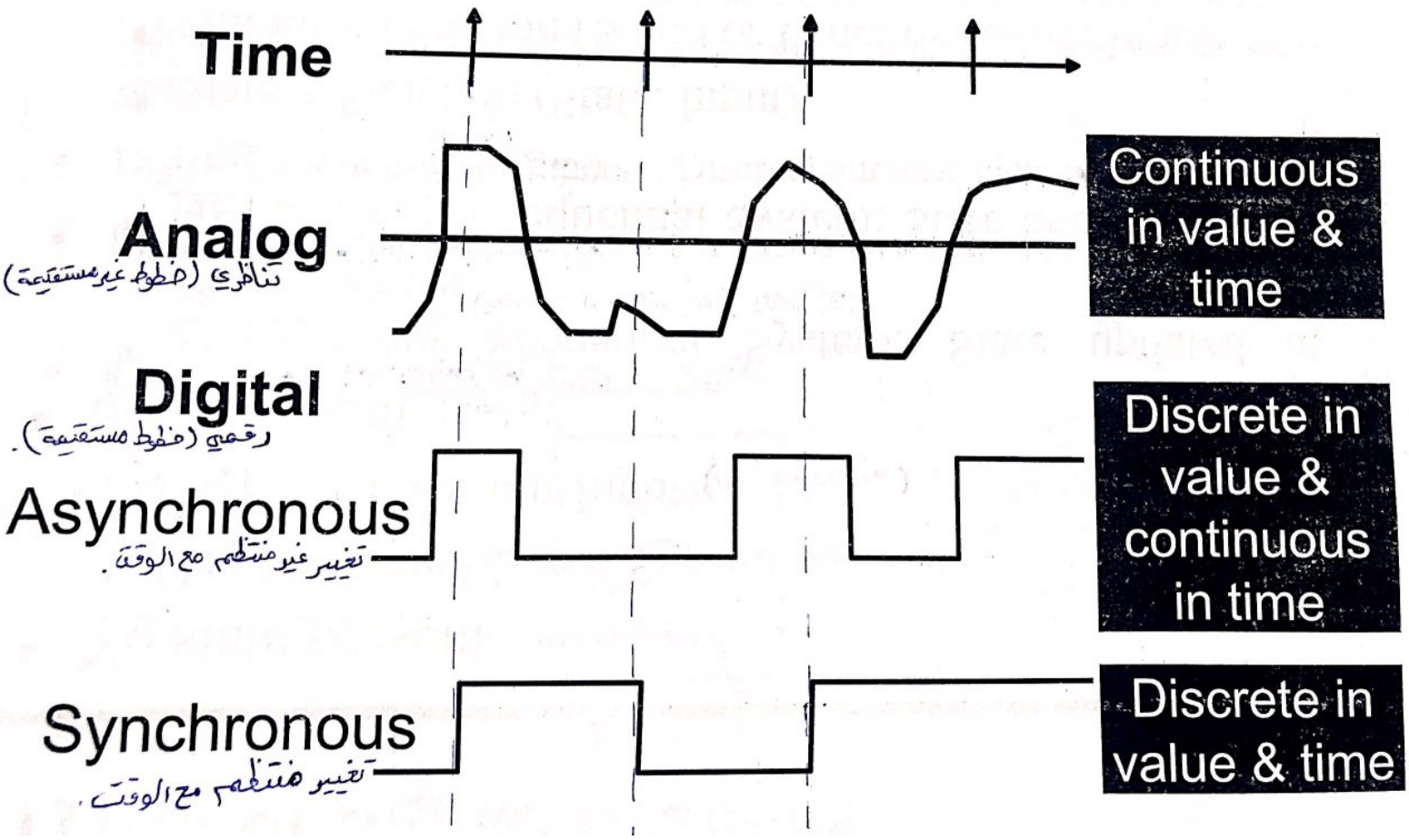
- No state present لا يخضع لأي حالة.
  - Combinational Logic System
  - Output = Function(Input) (no Difference).

That's why :

- State present يخضع لحالة.
  - *Synchronous* Sequential System: State updated at discrete times. (تسلسلي متزامن) (تغيير الحالة في وقت معين ، كل فترة معينة)
  - *Asynchronous* Sequential System: State updated at any time (تسلسلي غير متزامن) (تغيير الحالة في أي وقت) (عشوائي). تغيير/تطور.
  - State = Function (State, Input)
  - Output = Function (State) or Function (State, Input)

Moore **Mealy**

Signal Examples Over Time



Digital System Example

A Digital Counter (e. g., odometer): العدّاد الرقمي.

Count Up →
Reset →

0	0	1	3	5	6	4

Inputs: Count Up, Reset
Outputs: Visual Display
State: "Value" of stored digits

Synchronous or Asynchronous?

* حيث تتغير المسافة المقطوعة حسب سرعة السيارة .

And Beyond – Embedded Systems

- Computers as integral parts of other products
- Examples of embedded computers
  1. Microcomputers
  2. Microcontrollers
  3. Digital signal processors

تطبيقات على الأنظمة الحاسوبية المتضمّنة في الأجهزة.

- Examples of embedded systems applications

Cell phones	Dishwashers
Automobiles	Flat Panel TVs
Video games	Global Positioning Systems
Copiers	

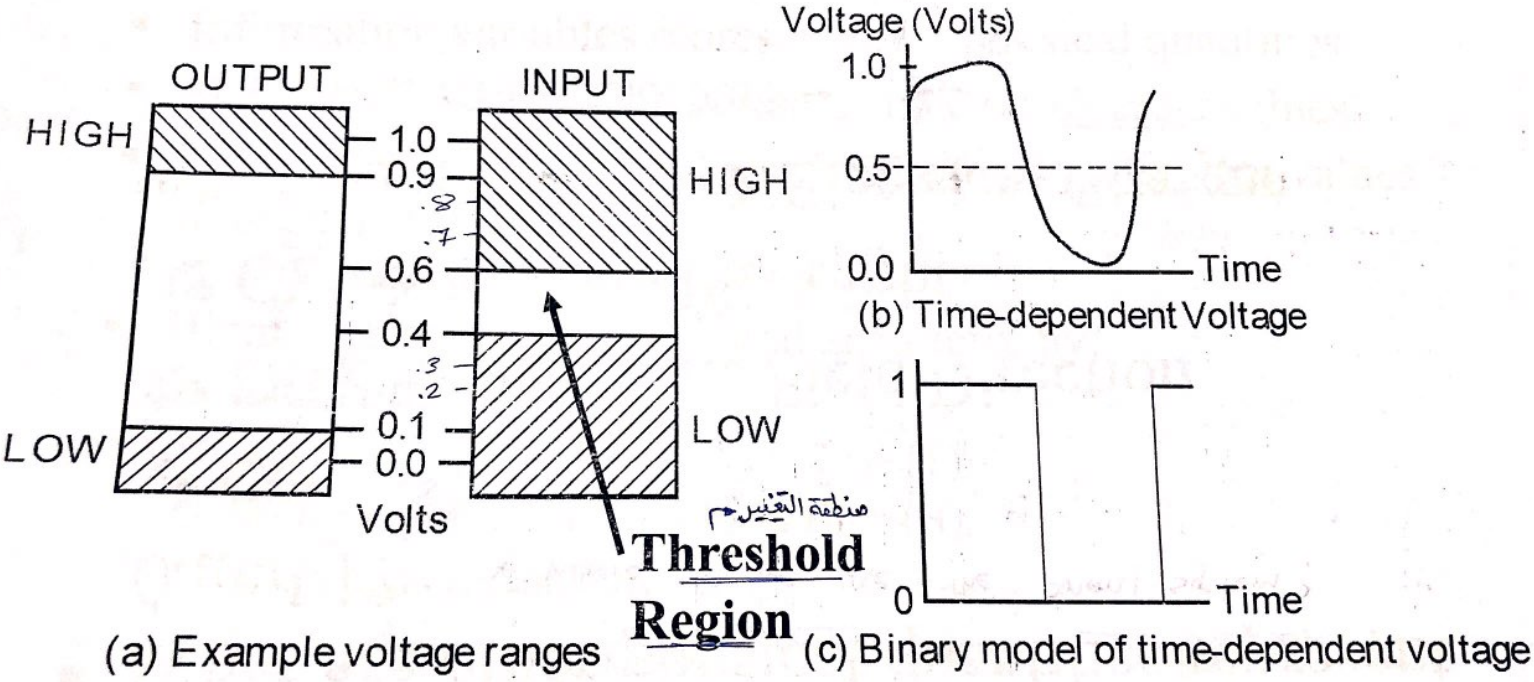
INFORMATION REPRESENTATION - Signals

- Information variables represented by physical quantities.
- For digital systems, the variables take on *discrete* values.
- Two level, or *binary* values are the *most prevalent* values in digital systems.
  - Binary systems have higher immunity to noise.
- Binary values are represented abstractly by:
  - digits 0 and 1
  - words (symbols) False (F) and True (T)
  - words (symbols) Low (L) and High (H)
  - and words On and Off.
- Binary values are represented by values or ranges of values of physical quantities.

Binary Values: Other Physical Quantities

- What are other physical quantities represent 0 and 1? (*) where do we use the binary system?
  - 1 CPU → Voltage
  - 2 Disk → Magnetic Field Direction
  - 3 CD → Surface Pits/Light
  - 4 Dynamic RAM → Electrical Charge stored in capacitors → (المواسع)

Signal Example – Physical Quantity: Voltage



(a) Example voltage ranges

(b) Time-dependent Voltage

(c) Binary model of time-dependent voltage

NUMBER SYSTEMS – Representation : (تمثيل - الأنظمة العددية)

- Positive radix (الأساس), positional number systems
- A number with *radix r* is represented by a string of digits:

$$A_{n-1}, A_{n-2}, \ldots, A_1, A_0 \,.\, A_{-1}, A_{-2}, \ldots, A_{-m+1}, A_{-m}$$

(MSD) أعلى قيمة خانة. (المنزلة ذات أعلى وزن)

* بدون مساواة: مستحيل يوصل لنفس عدد الأساس. دائماً أقل بواحد

radix-point : فاصلة عشرية

in which $0 \le A_i < r$ and . is the (*radix point*) or (Decimy-point). (LSD) أقل قيمة خانة. (المنزلة ذات أقل وزن).

- i represents the position of the coefficient (المنزلة)
- r^i represents the weight by which the coefficient is multiplied (وزن المنزلة)(الخانة)
- A_{n-1} is the most significant digit (MSD) and A_{-m} is the least significant digit (LSD)
- The string of digits represents the power series:

$$(Number)_r = \left(\sum_{i=0}^{n-1} A_i r^i\right) + \left(\sum_{j=-m}^{-1} A_j r^j\right)$$

(متسلسلة) $\sum_{i=-m}^{n-1} A_i r^i$

تجمعهم

Integer Portion للأعداد الصحيحة. **Fraction Portion** للأعداد العشرية.

في حالة وجود كسور عشرية.

* radix : الأساس
* Decimal system: النظام العشري. أساسه العدد (10)
* radix-point: فاصلة عشرية.
* Binary-point: فاصلة النظام الثنائي.

Number Systems – Examples

		General	Decimal (النظام العشري) (r=10)	Binary النظام الثنائي (r=2)
Radix (Base)		r	10	2
Digits		0 → r - 1 $(0 \leq A_i < r)$ or	0 → 9 $(0 \leq A_i \leq r-1)$	0 → 1
Powers of Radix	0	r^0	$10^0 = 1$	$2^0 = 1$
	1	r^1	$10^1 = 10$	$2^1 = 2$
	2	r^2	$10^2 = 100$	$2^2 = 4$
	3	r^3	$10^3 = 1000$	$2^3 = 8$
	4	r^4	$10^4 = 10{,}000$	$2^4 = 16$
	5	r^5	$10^5 = 100{,}000$	$2^5 = 32$
	-1	r^{-1}	$10^{-1} = 0.1 = \frac{1}{10}$	$2^{-1} = 0.5 = \frac{1}{2}$
	-2	r^{-2}	$10^{-2} = 0.01 = \frac{1}{100}$	$2^{-2} = 0.25 = \frac{1}{4}$
	-3	r^{-3}	$10^{-3} = 0.001 = \frac{1}{1000}$	$2^{-3} = 0.125 = \frac{1}{8}$
	-4	r^{-4}	$10^{-4} = 0.0001 = \frac{1}{10000}$	$2^{-4} = 0.0625 = \frac{1}{16}$
	-5	r^{-5}	$10^{-5} = 0.00001 = \frac{1}{100000}$	$2^{-5} = 0.03125 = \frac{1}{32}$

Example

1. $(403)_5 = 4 \times 5^2 + 0 \times 5^1 + 3 \times 5^0 = (103)_{10}$

2, 1, 0 : i

نوع النظام (خماسي) وهو نفسه الـ radix (الأساس).

* نلاحظ : digits must be :

$0 \leq A_i \leq 4.$

or

$0 \leq A_i < 5.$

2. $(103)_{10} = 1 \times 10^2 + 0 \times 10^1 + 3 \times 10^0 = 103$

* Example (1): 92871.

4 3 2 1 0 ← : position

r^i : جزء المنزلة : 10^0 : (1) : آحاد : (واحد)

10^1 : (10) : عشرات : (عشرون)

10^2 : (100) : مئات : (مائة)

10^3 : (1000) : ألوف : (ألفين)

وهكذا... :

* Example (2): change from binary system → to decimal system:

binary - point ... (فاصلة النظام الثنائي)

$\boxed{101.10}$ → in binary.

2 1 0 -1 -2

$2^2 * 1 + \cancel{2^1 * 0} + 2^0 * 1 + \cdot 2^{-1} * 1 + \cancel{2^{-2} * 0}.$

$4 + 1 + .5 =$

$\boxed{5.5}$ → in decimal.

BASE CONVERSION - Positive Powers of 2

- Useful for Base Conversion : * مفيد للتحويلات بين الأنظمة

* ما عليه نجمة (*) - يجب حفظه ضروري لتسهيل الحل ...

Exponent	Value
* 0 = 2^0:	1
* 1 = 2^1:	2
* 2 = 2^2:	4
* 3 = 2^3:	8
* 4 = 2^4:	16
* 5 = 2^5:	32
* 6 = 2^6:	64
* 7 = 2^7:	128
* 8 = 2^8:	256
* 9 = 2^9:	512
* 10 = 2^{10}:	1024

Exponent	Value
* 11 = 2^{11}:	2,048
12	4,096
13	8,192
14	16,384
15	32,768
16	65,536
17	131,072
18	262,144
19	524,288
* 20 = 2^{20}:	1,048,576
21	2,097,152

* 2^{30} : 1073741824.
* 2^{40} : $1.09 * 10^{12}$.
* 2^{50} : $1.1 * 10^{15}$.

Special Powers of 2

* memorize.

* ▪ 2^{10} (1024) is Kilo, denoted "K"

لأن الكيلو = 1000 وأقرب عدد لل (1000) هو (1024) وهو (2^{10})

* Example: 20. KB ⇒ $\boxed{20 * \overset{\text{كيلو}}{1024}}$ byte.

* ▪ 2^{20} (1,048,576) is Mega, denoted "M"

* ▪ 2^{30} (1,073, 741,824)is Giga, denoted "G"

* ▪ 2^{40} (1,099,511,627,776) is Tera, denoted "T"

* Note: byte = 8 bite . البايت = ٨ بت .

Commonly Occurring Bases

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

- The six letters A, B, C, D, E, and F represent the digits for values 10, 11, 12, 13, 14, 15 (given in decimal), respectively, in hexadecimal. Alternatively, a, b, c, d, e, f can be used.

Binary System النظام الثنائي :-

- $r = 2$ الأساس
- Digits = $\{0, 1\}$ or {false, True} or {low, high} or {off, on}.
- * Every binary digit is called a bit * مصطلح للنظام الثنائي فقط! فكل قيمة رقمية تسمى (بت). (1,0)
- When a bit is equal to zero, it does not contribute to the value of the number * يمكن استثناء الـ (0) من البداية قبل حل السؤال والتحويل للنظام العشري. لأن الصفر عند ضربه بأي قيمة يكون الجواب النهائي = صفر.
- Example:
  - $(10011.101)_2 = (1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) + (1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3})$ دون الاستثناء.
  - $(10011.101)_2 = (16 + 2 + 1) + \left(\frac{1}{2} + \frac{1}{8}\right) = (19.625)_{10}$ بالاستثناء.

Octal System النظام الثماني:

- r = 8
- Digits = {0, 1, 2, 3, 4, 5, 6, 7} : $(0 \leq A_i \leq r-1)$: الصيغة
- Every digit is represented by 3-bits → More compact than binary * كل قيمة رقمية في النظام الثماني تعادل (٣ قيم رقمية) في النظام الثنائي.
- Example:
  - $(\underset{2}{1}\underset{1}{2}\underset{0}{7}.\underset{-1}{4})_8 = (1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0) + (4 \times 8^{-1})$
  - $(127.4)_8 = (64 + 16 + 7) + \left(\frac{1}{2}\right) = (87.5)_{10}$

Hexadecimal System: النظام السادس عشري

- r = 16
- Digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, $\overset{10}{A}$, $\overset{11}{B}$, $\overset{12}{C}$, $\overset{13}{D}$, $\overset{14}{E}$, $\overset{15}{F}$} $(0 \leq A_i \leq 15)$
- Every digit is represented by 4-bits

  * كل قيمة رقمية في النظام السادس عشري تعادل (٤ قيم رقمية) في النظام الثنائي.
- Example:
  - $(\underset{=32}{B}\underset{1}{6}\underset{0}{5}F)_{16} = (11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0)$

    * انتبه! : نحوّل الرموز من النظام السادس عشري لقيمتهم العددية في النظام العشري ونضربهم بوزن المنزلة
  - $(B65F)_{16} = (46687)_{10}$

* التحويل من الأنظمة العددية المختلفة إلى النظام العشري:

Converting from any Base (r) to Decimal

$$(Number)_r = \underbrace{\left(\sum_{i=0}^{n-1} A_i r^i\right)}_{\text{Integer Portion}} + \underbrace{\left(\sum_{j=-m}^{-1} A_j r^j\right)}_{\text{Fraction Portion}}$$

- **Example: Convert $(11010)_2$ to $(N)_{10}$:**

Conversion from Decimal to Base (r)

- Convert the Integer Part
- Convert the Fraction Part
- Join the two results with a radix point

* binary: (r=2)

MSD → LSD

$(000)_2$ = $(0)_{10}$ تتغير كل اثنتين.

$(001)_2$ = $(1)_{10}$ كل وحدة

$(010)_2$ = $(2)_{10}$

$(011)_2$ = $(3)_{10}$ تتغير كل أربعة.

$(100)_2$ = $(4)_{10}$ وهكذا...

$(101)_2$ = $(5)_{10}$

$(110)_2$ = $(6)_{10}$

(111) = $(7)_{10}$

* octal: (r=8).

0
1
2
3
4
5
6
7

$(10)_8$ = $(8)_{10}$

$(11)_8$ = $(9)_{10}$

$(12)_8$ = $(10)_{10}$

$(17)_8$ = $(15)_{10}$

$(20)_8$ = $(16)_{10}$

$(100)_8 = (64)_{10}$

$(77)_8 = (63)_{10}$

* Hexadecimal: (r=16)

0
1
2
3
⋮
f

$(10)_{16}$ = $(16)_{10}$

$(11)_{16}$ = $(17)_{10}$

⋮

$(1f)_{16}$

$(20)_{16}$ = $(32)_{10}$

$(ff)_{16} = 255$

$(100)_{16} = 25$

Conversion Details

To Convert the Integral Part:

1. Repeatedly *divide* (نقسم) the number by the new radix and save (نحتفظ بالباقي) the *remainders* until *the quotient is zero* إلى أن يصبح الناتج = صفر
2. The digits for the new radix are the remainders in *reverse order* of their computation
3. If the new radix is > 10, then convert all remainders > 10 to digits A, B, …

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

To Convert the Fractional Part:

1. Repeatedly *multiply* (نضرب) the fraction by the new radix and save the *integer digits* of the results until the *fraction is zero or your reached the required number of fractional digits* لحين وصول الجزء العشري = صفر. نوقف الضرب !
2. The digits for the new radix are the integer digits *in order* of their computation
3. If the new radix is > 10, then convert all integers > 10 to digits A, B, …

Example: Convert 46.6875_{10} To Base 2

- Convert 46 to Base 2:

Division	Quotient (الناتج)	Remainder (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

* في حالة الـ integer part.

$$(46)_{10} = (\overset{32}{1}\overset{16}{0}\overset{8}{1}\overset{4}{1}\overset{2}{1}\overset{1}{0})_2$$

$2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

لعنا كبير فقط: $32 + 8 + 4 + 2 = 32 + 14 = (46)_{10}$

- Convert 0.6875 to Base 2: (radix)

* البسط أقل من المقام إذاً لا تقسم والباقي هو نفسه البسط.

Multiplication	Answer	
0.6875*2	1.3750	MSD
0.375*2	0.75	
0.75*2	1.5	
0.5*2	1.0	LSD

نأخذ الجزء العشري فقط

* في حالة الـ fraction part.

$$(0.6875)_{10} = (0.\overset{2^{-1}}{1}\overset{2^{-2}}{0}\overset{2^{-3}}{1}\overset{2^{-4}}{1})_2$$

$0.625 + 0.0625 = (0.6875)_{10}$

- Join the results together with the radix point:

$$(46.6875)_{10} = (101110.1011)_2$$

Example: Convert 153.513_{10} To Base 8

- Convert 153 to Base 8:

Division	Quotient	Remainder	
153/8	19	1	LSD
19/8	2	3	
2/8	0	2	MSD

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

* can be from (0) to (7) for the octal system

- Convert 0.513 to Base 8: (***Up to 3 digits***)

عدد المنازل المطلوبة بالجواب.

- Truncate: (قطع) تركُ باقي المنازل.

$$(0.513)_{10} = (0.406)_8$$

.406

- Round: التقريب.

$$(0.513)_{10} = (0.407)_8$$

.406 تقريب العدد

Multiplication	Answer	
0.513*8	4.104	MSD
0.104*8	0.832	
0.832*8	6.656	
0.656*8	5.248	LSD

* إذا كان ناتج القسمة دوري ولا ينتهي يحدد لنا بالسؤال عدد المنازل في العدد الكسري.

- Join the results together with the radix point:

$$(153.513)_{10} = (231.407)_8$$

* in the octal system ($\frac{1}{3}$) لا يُقرّب
($\frac{5}{7}$) يُقرّب بينما (4) يقرّب

Example: Convert 423_{10} To Base 16

Division	Quotient	Remainder	
423/16	26	7	LSD
26/16	1	10 (A)	
1/16	0	1	MSD

* انتبه! يجب تحويل الأرقام من (10 ← 15) بالرموز المحصصة في النظام السادس عشري.

$$(423)_{10} = (1A7)_{16}$$

Converting Decimal to Binary: Alternative Method

1. Subtract the largest power of 2 that gives a positive remainder and record the power

2. Repeat, subtracting from the prior remainder and recording the power, until the remainder is zero

3. Place 1's in the positions in the binary result corresponding to the powers recorded; in all other positions place 0's

* Note: $(0.40\cancel{7}5)_8$ octal system. لازم المنازل من (0 ← 7)

نقرّب ولكن ى لتالي ...

$(0.410)_8$

* لأن الـ (٧) لا تصبح (٨) لأنه نظام ثماني لذلك نضيف (١) للمنزلة التالية ونضعها (صفر).

Example: Convert 46.6875_{10} To Base 2 Using Alternative Method

- Convert 46 to Base 2:

$$(46)_{10} = (101110)_2$$

5 4 3 2 1 0

نضع بدل صفر ولا (1) في مكان المنازل والباقي أصفار.

Subtract	Remainder	Power
46-32=2	14	5
14-8=2	6	3
6-4=2	2	2
2-2=2	0	1

- Convert 0.6875 to Base 2:

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

Subtract	Remainder	Power
0.6875-0.500	0.1875	-1
0.1875-0.125	0.0625	-3
0.0625-0.0625	0	-4

- Join the results together with the radix point:

$$(46.6875)_{10} = (101110.1011)_2$$

*ملاحظة: إذا كان الرقم فردي لازم أول منزلة تكون (1) بينما زوجي أول منزلة تكون (0) في النظام الثنائي.

Ex: * 10115 زوجي

* 10111 فردي

- Easier way to do it:

	64	32	16	8	4	2	1		$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$
Power	6	5	4	3	2	1	0	.	-1	-2	-3	-4
	0	1	0	1	1	1	0	.	1	0	1	1

Octal (Hexadecimal) to Binary and Back: Method1

- Octal (Hexadecimal) to Binary:
  1. Convert octal (hexadecimal) to decimal (Slide 23)
  2. Covert decimal to binary (Slide 24 or Slide 29)

- Binary to Octal (Hexadecimal):
  1. Convert binary to decimal (Slide 23)
  2. Covert decimal to octal (hexadecimal) (Slide 24)

Base (r) .
or
octal (r = 8) .
or
hexadecimal (r=16) .
↓ ↑
Decimal (r=10) .
↓ ↑
Binary (r=2).

* r = 8 = 2^3. 3-bits in binary system:

4 2 1
000 : 0
001 : 1
010 : 2
011 : 3
100 : 4
101 : 5
110 : 6
111 : 7

bod

* r = 16 = 2^4. 4-bits in binary system:

8 4 2 1
0000 : 0
0001 : 1
0010 : 2
0011 : 3
0100 : 4
0101 : 5
0110 : 6
0111 : 7
1000 : 8
1001 : 9
1010 : A
1011 : B
1100 : C
1101 : D
1110 : E
1111 : F

bod

Octal (Hexadecimal) to Binary and Back: Method2 (Easier)

octal (r=8) to hexadecimal (r=16).
binary (r=2)

- Octal (Hexadecimal) to Binary:
  - ***Restate*** the octal (hexadecimal) as three (four) binary digits starting at the radix point and going both ways
- Binary to Octal (Hexadecimal):
  - ***Group*** the binary digits into three (four) bit groups starting at the radix point and going both ways, padding (تجزيء) with zeros as needed
  - Convert each group of three (four) bits to an octal (hexadecimal) digit

Octal	0	1	2	3	4	5	6	7
Binary	000	001	010	011	100	101	110	111

Hexadecimal	0	1	2	3	4	5	6	7
Binary	0000	0001	0010	0011	0100	0101	0110	0111
Hexadecimal	8	9	A	B	C	D	E	F
Binary	1000	1001	1010	1011	1100	1101	1110	1111

* ملاحظة :- إذا لم يكونا النظامين المطلوب التحويل بينهما قوّة للأساس (r) نقوم بالتحويل للنظام العشري (decimal) ومن ثم نحوّل للنظام المطلوب.

- Given that $(365)_r = (194)_{10}$, compute the value of r?

$$3 \times r^2 + 6 \times r^1 + 5 \times r^0 = 194$$

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

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

$$r^2 + 2r - 63 = 0$$ * عبارة تربيعية ---- تحلل

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

* نأخذ الـ (radix) الموجب فقط ونهمل السالب.

$$r = 7$$

Binary Numbers and Binary Coding

⊛ نلاحظ : دائماً كتابة الـ (codes) باستخدام الـ (binary system).

- Flexibility of representation
  - Within constraints below, can assign any binary combination (called a *code word*) to any data as long as data is uniquely encoded

* لمعرفة عدد الـ (codes) اللازمة :

- Information Types

n-bits. $\longrightarrow 2^n$. (using binary system, r=2).

  - *Numeric* . رقميّة / . عدديّة
    - Must represent range of data needed
    - Very desirable to represent data such that simple, straightforward computation for common arithmetic operations permitted
    - Tight relation to binary numbers
  - *Non-numeric* . غير رقميّة / . غير عدديّة
    - Greater flexibility since arithmetic operations not applied
    - Not tied to binary numbers

Non-numeric Binary Codes

- Given n binary digits (called bits), a binary code is a mapping from a set of represented elements to a subset of the 2^n binary numbers.

  [n-bits. → 2^n.] ← * لمعرفة عدد الـ (Codes) اللازمة :

- Example: A binary code for the seven colors of the rainbow
- Code 100 is not used * متاحة ولكن لم تستخدم عادي

Color	Binary Number
Red	000
Orange	001
Yellow	010
Green	011
Blue	101
Indigo	110
Violet	111

Number of Bits Required

(Number of elements).

- Given ***M*** elements to be represented by a binary code, the minimum number of bits, ***n***, needed, satisfies the following relationships:

if $\boxed{n=3}$. $2^3 \geq M > 2^2$

$(8 \geq 7 > 4)$.

$$2^n \geq M > 2^{n-1}$$

$n = \lceil log_2 M \rceil$, where $\lceil x \rceil$ is called the *ceiling function,* is the integer greater than or equal to x.

عدد البت المطلوبة:-

- Example: How many bits are required to represent decimal digits with a binary code?

*if: $r=2$. $n=7$.

$M = 2^7 = 128$.

(maximum possible number of elements):

* also: $r = (*)$

$M = (*)^n$. (نفس الـ radix)

$$M = 10$$

$$n = \lceil log_2 10 \rceil = \lceil 3.33 \rceil = 4$$

تقريب للعدد الأكبر

Number of Elements Represented

- Given n digits in radix r, there are r^n distinct elements that can be represented.
- But, you can represent m elements, $m \leq r^n$

Examples:

- You can represent 4 elements in radix $r = 2$ with $n = 2$ digits: (00, 01, 10, 11).
- You can represent 4 elements in radix $r = 2$ with $n = 4$ digits: (0001, 0010, 0100, 1000). كل مرة يكون العدد (1) في منزلة معينة.
- This second code is called a "<u>one hot</u>" code.

* if we have elements and we want to represent them using minimum number of digits of base "r", we use the form:

$n = \lceil \log_r M \rceil$.

* Ex: $n = \lceil \log_8 7 \rceil = \lceil \log_8 8^{.0-} \rceil = \lceil .0- \rceil = 1$.

* n: minimum number of bits to represent (M).

Ex: $n = \lceil \log_2 365 \rceil = \lceil \log_2 2^{8.-} \rceil = \lceil 8.- \rceil = \boxed{9}$

* يمكن أيضاً معرفة أكبر قيمة (value) نصل لها / نمثلها :

if $r = 10$, $n = 2$ } then $r^n - 1 \Rightarrow r^{10} - 1 = 100 - 1 = 99$. the biggest value.

* دائماً نقرب لأكبر بعدد واحد. (+1)

DECIMAL CODES - Binary Codes for Decimal Digits (BCD).

- There are over 8,000 ways that you can chose 10 elements from the 16 binary numbers of 4 bits. A few are useful:

slide(44)

Decimal	8, 4, 2, 1	Excess 3	8, 4, -2, -1 (weights)	Gray
0	0000	0011	0000	0000
1	0001	0100	0111	0001
2	0010	0101	0110	0011
3	0011	0110	0101	0010
4	0100	0111	0100	0110
5	0101	1000	1011	1110
6	0110	1001	1010	1010
7	0111	1010	1001	1011
8	1000	1011	1000	1001
9	1001	1100	1111	1000

(*) if we have a digital system:

- $M = 10$: عدد الـ elements
  $r = 2$
  n: must be (4)
  because $2^4 = 16$
  10 تستخدم — 6 لا تستخدم وتبقى متاحة.

(6) codes will be unused or available.

0000 ← 0
0001 ← 1
0010 ← 2
0011 ← 3
⋮
1001 ← 9

} these are used codes but codes from (1010 → 1111) will not be used.

(*) $(19)_{10} \rightarrow (10011)_2$
∴ convert (تحويل)

(*) $(19)_{10} \rightarrow (0001\ 1001)$
∴ BCD coding (ترميز)

* أحياناً يتم استخدام الـ codes لكي نخفي القيمة الحقيقية للرقم حيث نعطي الرقم قيمة تختلف عن قيمته الحقيقية.

Binary coded decimal

* Ex: 1 : 0111 : 4 − 3 = 1 (8 4 −2 −1)
... وهكذا ... 9 : 1111 : 12 − 3 = 9

Binary Coded Decimal (BCD)

- Numeric code
- The BCD code is the 8, 4, 2, 1 code
- 8, 4, 2, and 1 are weights → BCD is a *weighted* code
- This code is the simplest, most intuitive binary code for decimal digits and uses the same powers of 2 as a binary number, ***but only encodes the first ten values from 0 to 9***
- Example: 1001 (9) = 1000 (8) + 0001 (1)

غير مستخدم (متاحة)

- How many "invalid" code words are there?
  - Answer: 6
- What are the "invalid" code words?
  - Answer: 1010, 1011, 1100, 1101, 1110, 1111
    (10) (11) (12) (13) (14) (15)

التحويل. و الترميز

Warning: Conversion or Coding?

- Do NOT mix up ***conversion*** of a decimal number to a binary number with ***coding*** a decimal number with a BINARY CODE.

- $13_{10} = 1101_2$ **(This is conversion)** تحويل بين الأنظمة

  8 4 2 1

  12+1 = 13

  كقيمة.

Excess 3 ← - **13 ⇔ 0001|0011 (This is coding)** ترميز.

  8 4 2 1 8 4 2 1

  1 3

  كأعداد

ALPHANUMERIC CODES - ASCII Character Codes

- Non-numeric code ⊛ ASCII code:-

  if: $n=7$, $r=2$. } → then: maximum number of elements $= 2^7 = 128$.

- ASCII stands for American Standard Code for Information Interchange (Refer to Table 1-5 in the text)

- This code is a popular code used to represent information sent as character-based data. It uses 7-bits (i.e. 128 characters) to represent:
  - 95 Graphic printing characters
  - 33 Non-printing characters

ASCII Code Table (c++)

Least Significant

ASCII Code Chart

Most Significant

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

$W = (77)_{16} = (0111\ 0111)_2$

$W = (57)_{16} = (0101\ 0111)_2$.

row صف — column عمود — الاختلاف بنفس الـ(bit)

$A = (41)_{16} = (0100\ 0001)_2$

$a = (61)_{16} = (0110\ 0001)_2$

- ASCII has some interesting properties:
  - Digits 0 to 9 span Hexadecimal values 30_{16} to 39_{16}
  - Upper case A-Z span 41_{16} to $5A_{16}$
  - Lower case a-z span 61_{16} to $7A_{16}$
  - Lower to upper case translation (and vice versa) occurs by flipping bit 6 (0 1 0 0 1 1 1 1)

UNICODE

- UNICODE extends ASCII to 65,536 universal characters codes:
  - Non-numeric
  - For encoding characters in world languages
  - Available in many modern applications
  - 2 byte (16-bit) code words

⊛ if: $n = 16$.
$r = 2$.
$2^{16} = 65{,}000$.

PARITY BIT Error-Detection Codes

(ECC)

* تبحث عن الأخطاء الفردية فقط و تجد الخطأ فقط لا تقوم بتصليحه. (تحدّد مكان الخطأ فقط).

detect odd number of error (تحدّد)

- Non-numeric
- ***Redundancy*** (e.g. extra information), in the form of extra bits, can be incorporated into binary code words to detect and correct errors
- A simple form of redundancy is ***parity***, an extra bit appended onto the code word to make the number of 1's odd or even. Parity can detect all single-bit errors and some multiple-bit errors
  - A code word has ***even parity*** if the number of 1's in the code word is even
  - A code word has ***odd parity*** if the number of 1's in the code word is odd

4-Bit Parity Code Example

- Fill in the even and odd parity bits:

Even Parity Message	Odd Parity Message
0000 → parity bit.	0001
0011	0010
0101	0100
0110	0111
1001	1000
1010	1011
1100	1101
1111	1110

* if we have:
0011
parity bit نضيف
parity=0 even parity 00110 عدد الواحدات زوجي لذلك ال parity زوجية.
parity=1 odd parity 00111 عدد الواحدات فردي لذلك ال parity فردية.

*Ex: 0001
even parity 00011 لكي يصبح عدد (1) زوجي.
odd parity 00010 لكي يبقى عدد (1) فردي.

- The code word "1111" has even parity and the code word "1110" has odd parity. Both can be used to represent the same 3-bit data

* Note to know:
write the gray codes for:
0 → 00
1 → 01
2 → 11
3 → 10
يحدث اختلاف بين بت وبت واحد فقط بين كل code و الأخرى

Combinational Logic Circuits

- Digital (logic) circuits are hardware components that manipulate binary information.
- Integrated circuits: transistors and interconnections.
  - Basic circuits is referred to as ***logic gates***
  - The outputs of gates are applied to the inputs of other gates to form a digital circuit
- Combinational? Later…

Binary Logic and Gates

- ***Binary variables*** take on one of two values
- ***Logical operators*** operate on binary values and binary variables
- Basic logical operators are the logic functions ***AND***, ***OR*** and ***NOT***

  ⊛ Basic operator gates:
  ① AND : · , ∧ ③ NOT : ~/ , / – .
  ② OR : + , ∨ .
- ***Logic gates*** implement logic functions
- ***Boolean Algebra***: a useful mathematical system for specifying and transforming logic functions
- We study Boolean algebra as a foundation for designing and analyzing digital systems!

Notation Examples

- Examples:
  - $Z = X \cdot Y = XY = X \wedge Y$: is read "Z is equal to X AND Y"
    - Z = 1 if and only if X = 1 (both) and Y = 1; otherwise, Z = 0 غير ذلك. ⊛ if: $Z(x,y) = xy$
  - $Z = X + Y = X \vee Y$: is read "Z is equal to X OR Y" XANDY = X·y = تقاطع
    - Z = 1 if (only X = 1) or (one of them) if (only Y = 1) or (or both) if (X =1 and Y = 1)
  - $Z = \bar{X} = X' = \sim X$: is read "Z is equal to NOT X"
    - Z = 1 if X = 0; otherwise, Z = 0 (opposite always).
- Notice the difference between arithmetic addition and logical OR: البوابة المنطقية (OR) العملية الجمع. (+)
  - The statement:

    1 + 1 = 2 (read "one plus one equals two") (10)2

    *is not the same as difference.

    1 + 1 = 1 (read "1 or 1 equals 1")

Operator Definitions

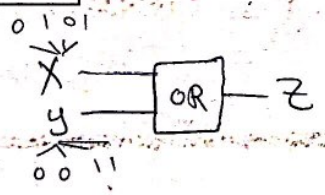
- Operations are defined on the values "0" and "1" for each operator: معرّف الاحتمالات.

AND
$0 \cdot 0 = 0$
$0 \cdot 1 = 0$
$1 \cdot 0 = 0$
$1 \cdot 1 = 1$

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

NOT
$\overline{0} = 1$
$\overline{1} = 0$

$r = 2$, $n = 2$ } → then $2^2 = 4$ احتمالات.



Truth Tables

جدول الحقيقة

يشمل كل الاحتمالات.

- ***Truth table*** - a tabular listing of the values of a function for all possible combinations of values on its arguments

- Example: Truth tables for the basic logic operations:

	AND		
	Inputs		Output
	X	Y	Z = X . Y
0 ←	0	0	0
1 ←	0	1	0
2 ←	1	0	0
3 ←	1	1	1

العدّ بالترتيب

عدد المدخلات 2

$2^2 = 4$ احتمالات

OR		
Inputs		Output
X	Y	Z = X + Y
0	0	0
0	1	1
1	0	1
1	1	1

$2^2 = 4$ احتمالات

NOT	
Inputs	Output
X	$Z = \overline{X}$
0	1
1	0

عدد المدخلات = 1

$2^1 = 2$ احتمال فقط.

most significant.

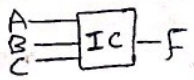
least significant

$2^3 = 8$

MSbit inputs: LSbit

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

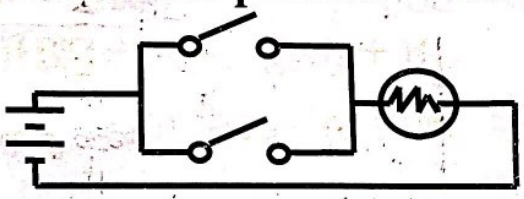
Draw:-



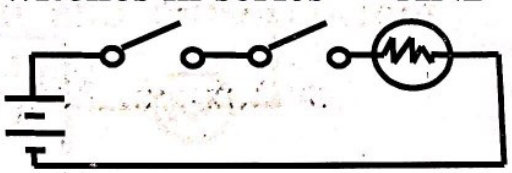
Logic Function Implementation

- Using Switches
  - For inputs:
    - logic 1 is switch closed
    - logic 0 is switch open
  - For outputs:
    - logic 1 is light on
    - logic 0 is light off
  - NOT uses a switch such that:
    - logic 1 is switch open
    - logic 0 is switch closed

Switches in parallel => OR

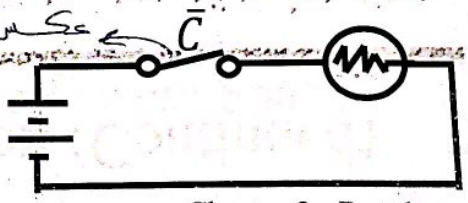


Switches in series => AND



Normally-closed switch => NOT

عكس القيمة



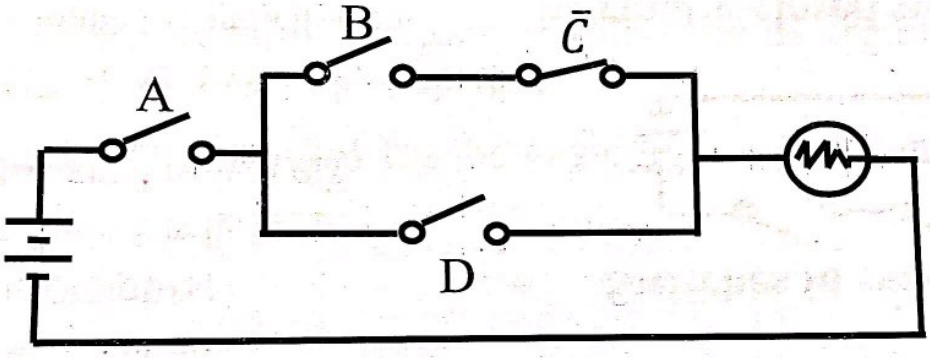
* عندما يكون المفتاح مغلق يعني الدارة (0)
* عندما يكون المفتاح مفتوح يعني الدارة (1)

(1 ← 0)
(0 ← 1)

عكس الواقع لأن البوابة (not)

Logic Function Implementation (Continued)

- Example: Logic Using Switches



وصف الـ circuit ← by ← Formula.

- Light is
  ON (L = 1) for $L\ (A, B, C, D) = A\,.\,(B\bar{C} + D) = AB\bar{C} + AD$
  and ***OFF*** (L = 0), otherwise.

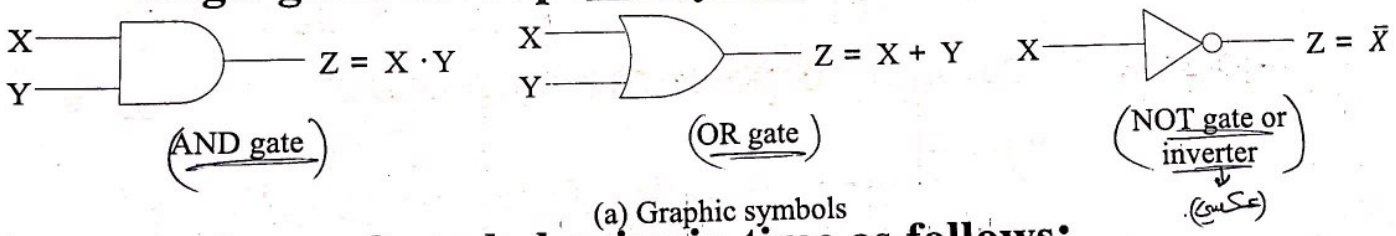
  نفتحها

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

- Useful model for relay circuits and for CMOS gate circuits, the foundation of current digital logic technology

Logic Gate Symbols and Behavior

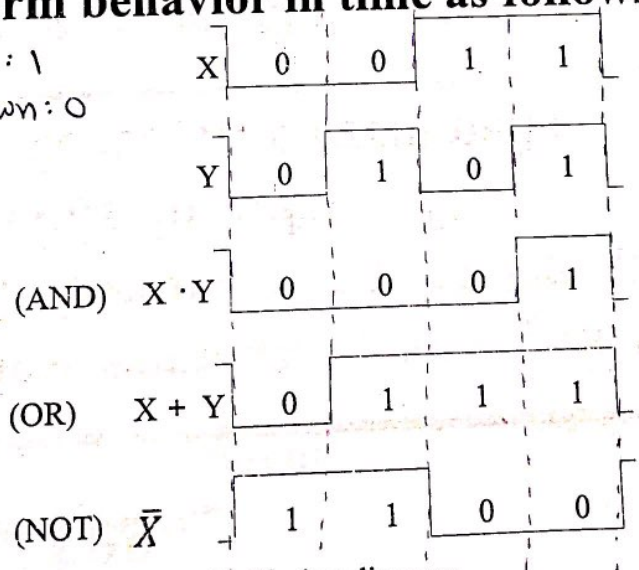
- **Logic gates have special symbols:** أشكال خاصة



(a) Graphic symbols

- **And waveform behavior in time as follows:**

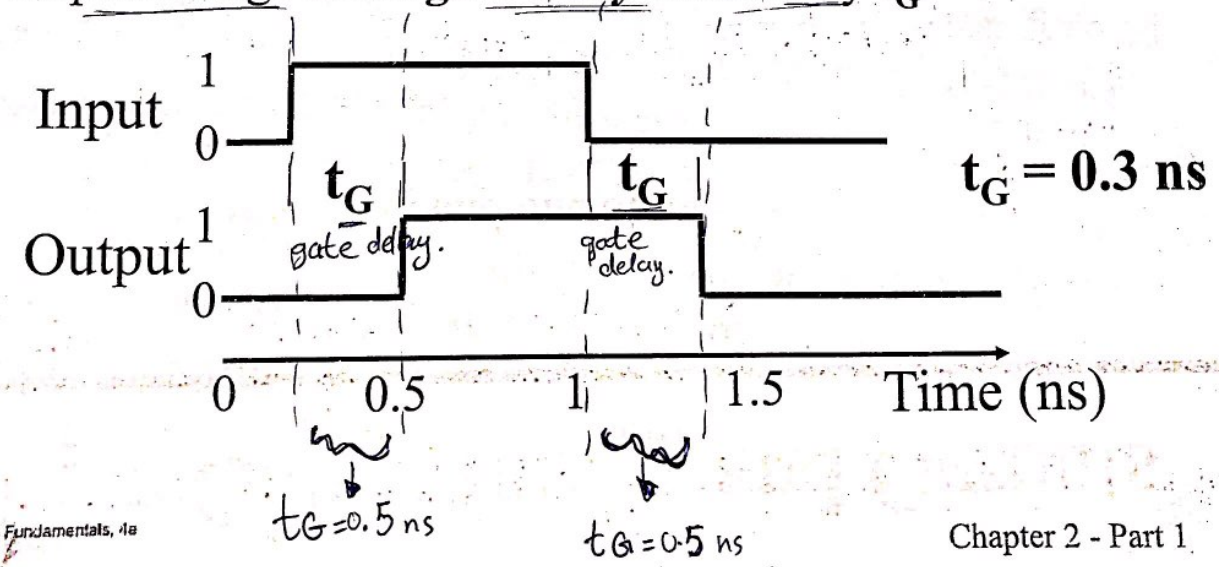
* up: 1
* down: 0



(b) Timing diagram

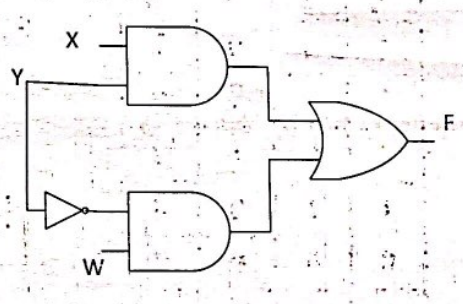
Gate Delay (التأخير)

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



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

عدد المتغيرات
عدد الاحتمالات = ٢

∴ ٢^٣ = ٨ = ٤/٢ = ٢/٢ ،

نوزّع القيم بالأول ٤ ← أصفار.
٤ ← واحدات.

ثم ٤/٢ = ٢ ، ٢ ← صفر.
٢ ← واحد.
وهكذا

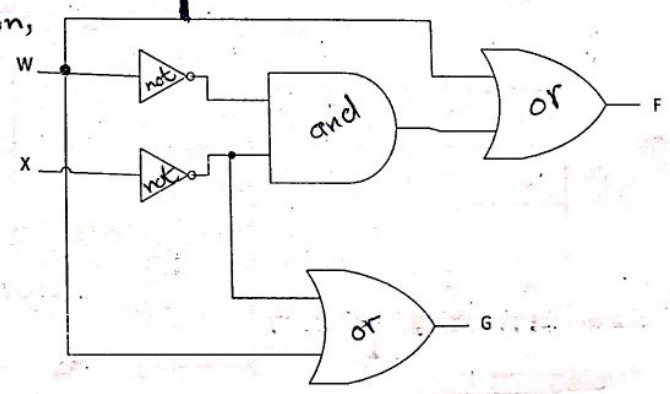
- This example represents a ***Single Output Function***

Example

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

as a function, of(w, y)

- Logic Diagram:



- Truth Table:

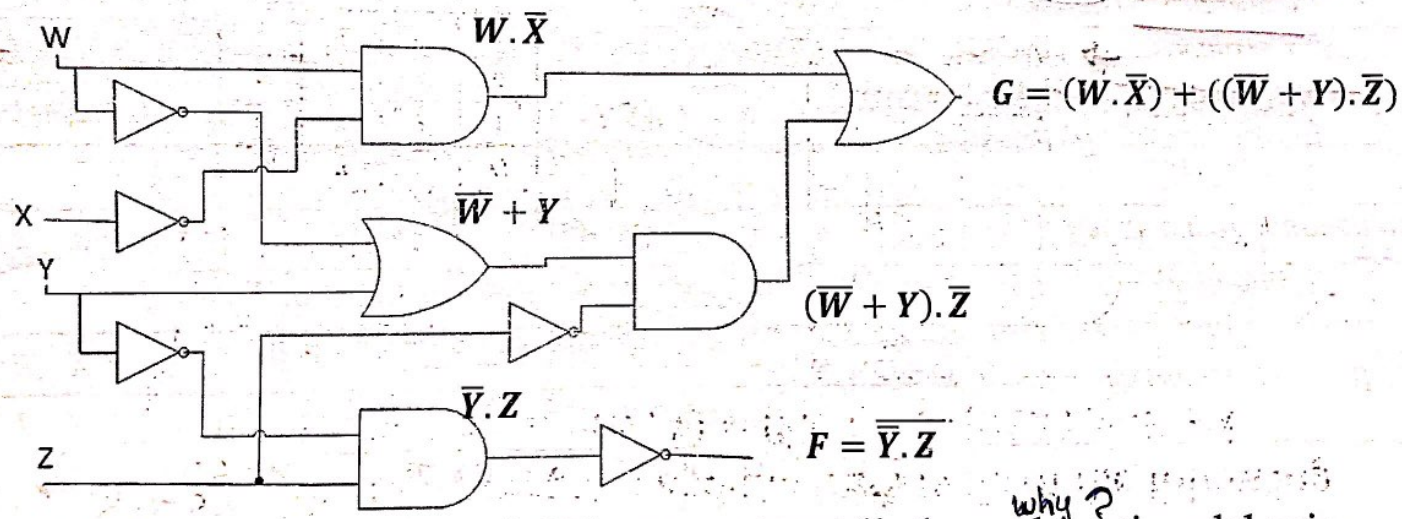
W	X	F	G
0	0	1	1
0	1	0	0
1	0	1	1
1	1	1	1

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

Example:

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

المعادلة المقابلة التي تمثل الشكل التالي



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

why?

answer:-

Basic Identities of Boolean Algebra

* The distributive law even though more than one variable
*Ex: $AB.(x+y+z) = AB.x + AB.y + AB.z$

* $0.0 = 0$
$0.1 = 0$
$(A+B+C).0 = 0$
بعض النظر عنه.

* complement = not = bar = $\bar{X}$. (نفس الحقل)

* $X.1 = X$, $0.1 = 0$, $1.1 = 1$

1. $X + 0 = X$	2. $X . 1 = X$	*Existence of 0 and 1*
3. $X + 1 = 1$	4. $X . 0 = 0$ ($1.0 = 0$, $0.0 = 0$)	
5. $X + X = X$	6. $X . X = X$	*Idempotence*
7. $X + \bar{X} = 1$	8. $X . \bar{X} = 0$	*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*

قانون الترتيب للمتغيرات (تبادلي العملية).

ترتيب المتغيرات عند تساوي العملية بينهم لا يهم

شرط هنا القانون أن تكون العملية بين المتغيرات نفسها.

Even though more than two variables
*Ex: $\overline{X+y+z} = \bar{X}.\bar{y}.\bar{z}$

توزيع ما خارج القوس على القوس وعمل العملية نفسها بين المتغيرات.

نوزع الـ (not) ونعكس العملية (and ⇄ or).

Some Properties of Identities & the Algebra

- If the meaning is unambiguous, we leave out the symbol "." → * AB = A.B = A AND B.

  خالي من الشكوك.

- 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**

    * تعكس الـ (0 ⇄ 1) والعملية (or ⇄ and) ولكن (X) تبقى كما هي.

    تغيير / تبديل

    عبارات متضادة

  - 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.

    الجواب للعبارتين المتضادتين نفسه.

عبارة صحيحة : $X + 0 = X$

عبارة صحيحة أخرى : $X \cdot 1 = X$

تعتبره متغير واحد.

* Note:- $\overline{X + (y \cdot z)} = \overline{X} \cdot \overline{y \cdot z}$ →

$\overline{X} \cdot (\overline{y} + \overline{z})$. نضع أقواس لكي نحافظ على الأولوية.

Boolean Operator Precedence

- The order of evaluation in a Boolean expression is: أولويات العمليات:
  1. Parentheses الأقواس
  2. NOT بوابة not
  3. AND بوابة and
  4. OR بوابة or

- Consequence: Parentheses appear around OR expressions

  * لكي يتم تنفيذه قبل and أو not حسب الحاجة. *

- Examples:
  - $F = A(B + C)(C + \overline{D})$
  - $F = \sim A B = \overline{A} B$
  - $F = A B + C$
  - $F = A(B + C)$ $= A \cdot B + A \cdot C$.

Useful Boolean Theorems

النظريات:

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

$* F(A,B) = \sim A.B \longrightarrow$ not A ① , A and B ②

بينما $* F(A,B) = \sim(A.B) \longrightarrow$ A and B ① , not(A and B) ②

Example 1: Boolean Algebraic Proof

- $A + A \cdot B = A$ (Absorption Theorem)

كأنها $(A \cdot 1)$ فنأخذ (A) عاملاً مشتركاً.

$A + A \cdot B$		dual:-
$= A \cdot 1 + A \cdot B$	$X = X \cdot 1$	$A \cdot (A + B) = A$
$= A \cdot (1 + B)$	Distributive Law	$A \cdot A + A \cdot B$
$= A \cdot 1$	$1 + X = 1$	$A + AB$
$= A$	$X \cdot 1 = X$	$A(1 + B)$
	كأنها متغير واحد	$A \cdot 1 = A$

*even though: $X + X \cdot (ABCDE) = X$

- 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

even though more than one variables $X \cdot yz + \bar{X} \cdot w + wyz = Xyz + \bar{X} \cdot w$.

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

كأنّ (BC) غير موجودة أي لا تؤثر.

* حقيقة: $A + \bar{A} = 1$ دائماً.

$AB + \bar{A}C + BC$	كأن BC هي (BC.1)
$= AB + \bar{A}C + 1.BC$	$1.X = X$
$= AB + \bar{A}C + (A + \bar{A}).BC$	$X + \bar{X} = 1$
$= AB + \bar{A}C + ABC + \bar{A}BC$	*Distributive Law* توزيع القوس
$= AB + ABC + \bar{A}C + \bar{A}BC$	*Commutative Law* ترتيب الحدود
$= AB.1 + AB.C + \bar{A}C.1 + \bar{A}C.B$	$X.1 = X$ *and Commutative Law*
$= AB(1 + C) + \bar{A}C(1 + B)$	*Distributive Law* إخراج عامل مشترك
$= AB.1 + \bar{A}C.1$	$1 + X = 1$
$= AB + \bar{A}C$	$X.1 = X$

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

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

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

البرهان

- $A.B + \bar{A}.B = B$ نظرية الاختزال (التقليص) (Minimization Theorem)

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

* حقيقة: $A+\bar{A}=1$.
* حقيقة: $A.\bar{A}=0$.

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

$(A + B)(\bar{A} + B)$	B: عامل مشترك
$= B + (A.\bar{A})$	*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 $\boxed{A + A' = 1}$ and $\boxed{A.A' = 0}$ with $A = X + Y$ and $A' = X'.Y'$. This proves that $X'.Y' = \overline{X+Y}$.

slide 32 * $\overline{X+y} = \bar{X}\cdot\bar{y}$.

$A = X+y$.

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

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

$A+\bar{A} = 1$.

- Part 1: Show $X + Y + X'.Y' = 1$

slide 31:

* $\overline{X+y} = \bar{X}\cdot\bar{y}$.

$A + \bar{A} = (X+y) + \bar{X}\cdot\bar{y}$

$= (X+y+\bar{X})\cdot(X+y+\bar{y})$

$= (1+y)\cdot(X+1) =$

$= 1\cdot 1 = 1$.

$(X+Y)+X'.Y'$	توزيع
$= (X+Y+X')(X+Y+Y')$	*Distributive Law*
$= (1+Y)(X+1)$	$X+\bar{X}=1$
$= 1.1$	$X+1=1$
$= 1$	$X.1 = X$

* $\overline{X\cdot y} = \bar{X}+\bar{y}$.

$A\cdot\bar{A} = (X+y)\cdot\bar{X}\cdot\bar{y}$.

$= \bar{X}\cdot\bar{y}\cdot(X+y)$. ترتيب

$= Xy\bar{X} + Xyy$.

$= Xy + Xy =$

$= 0 + 0 = 0$.

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* (not) توزيع
$= Y'(X'Z + X)$	*Distributive law* عامل مشترك
$= Y'(X + X'Z)$	*Commutative law* ترتيب الحدود
$= Y'(X + Z)$	*Simplification Theorem* $A + \bar{A}.B = A+B$.

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

* في مثل هذه الأسئلة ، نحدّد متى يكون قيمة الناتج النهائي (١) أو (٥) حسب العبارة وحالة المسألة أفضل من حل جميع العبارات لجميع القيم وذلك للسرعة و لتسهيل الحلّ.

Expression Simplification

- An application of Boolean algebra
- Simplify to contain the smallest number of literals (complemented and uncomplemented variables). have (not) gate.
  تختصر عدد المتغيرات في التعبير لأقل عدد ممكن
- Example: Simplify the following Boolean expression
  - $AB + A'CD + A'BD + A'CD' + ABCD$

$AB + A'CD + A'BD + A'CD' + ABCD$	
$= AB + ABCD + A'CD + A'CD' + A'BD$	*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*
$= B(A + D) + A'C \rightarrow$ *5 Literals*	*Simplification Theorem*

Complementing Functions

(not) وضع

نفي التعبير

- 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

- 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

*Note: عند وجود جميع احتمالات التعبير خلال الاقتران هذا يعني بأن الجواب دائماً (1).

الاحتمالات جميعها هي: $\bar{x}\bar{y}$ ، $\bar{x}y$ ، $x\bar{y}$ ، xy

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

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

Note* : هذا السؤال لكي تميز بين الـ complement والـ dual لأي function.

Minterms

- ***Minterms*** are AND terms with ***every variable*** present in either true or complemented form
  (X) إيجاب (X̄) منفية
  تحتوي على كل المتغيرات مرتبة بالترتيب الأبجدي.
- 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:
  عدد الاحتمالات.
  1. $X.Y$ (both normal)
  2. $X.\bar{Y}$ (X normal, Y complemented)
  3. $\bar{X}.Y$ (X complemented, Y normal)
  4. $\bar{X}.\bar{Y}$ (both complemented)
- Thus there are ***four minterms*** of two variables

Maxterms and Minterms

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

Index	Minterm (m)	Maxterm (M)
0	$\bar{X}.\bar{Y}.\bar{Z}$	$X+Y+Z$
1	$\bar{X}.\bar{Y}.Z$	$X+Y+\bar{Z}$
2	$\bar{X}.Y.\bar{Z}$	$X+\bar{Y}+Z$
3	$\bar{X}.Y.Z$	$X+\bar{Y}+\bar{Z}$
4	$X.\bar{Y}.\bar{Z}$	$\bar{X}+Y+Z$
5	$X.\bar{Y}.Z$	$\bar{X}+Y+\bar{Z}$
6	$X.Y.\bar{Z}$	$\bar{X}+\bar{Y}+Z$
7	$X.Y.Z$	$\bar{X}+\bar{Y}+\bar{Z}$

* لكل minterm و maxterm اسم معين
minterm 0 : $\bar{x}\bar{y}\bar{z}$
maxterm 0 : $x+y+z$
minterm 1 : $\bar{x}\bar{y}z$
وهكذا

* نلاحظ : في الـ minterm نبدأ بنفي جميع المتغيرات أي أن كل الـ variables يكونوا complement . بينما في الـ maxterm نبدأ من المتغيرات كما هي في حالة الـ true .

- **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)*** ترتيب أبجدي
- **Example: For variables a, b, c:**
  - **Maxterms:** $(a + b + \bar{c})$, $(a + b + c)$
  - حدّ **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**

* الـ minterm : هو تعبير يحتوي جميع الـ Variables (متغيرات) في السؤال ويكون فيه جميع المتغيرات بالترتيب الأبجدي .

Index Example: Three Variables

* for minterm: 0 → complemented.
1 → true.

* for maxterm: 0 → true.
1 → complemented.

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	$m_7 = XYZ$	$M_7 = \bar{X} + \bar{Y} + \bar{Z}$

* الفرق فقط أنه تم زيادة عدد المتغيرات .

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	$abcd$	$\bar{a}+\bar{b}+\bar{c}+\bar{d}$

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:

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

* عكس بعضهم دائماً:

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

Function Tables for Both

- Minterms of 2 variables:

xy	m_0	m_1	m_2	m_3
00	1	0	0	0
01	0	1	0	0
10	0	0	1	0
11	0	0	0	1

* كل مرّة يكون الرقم (1) موجود بحالة وحدة والباقي أصفار (0).
* نلاحظ بأن الرقم (1) قليل كثيرًا لذلك minterm.

عكس بعض تمامًا

- Maxterms of 2 variables:

xy	M_0	M_1	M_2	M_3
00	0	1	1	1
01	1	0	1	1
10	1	1	0	1
11	1	1	1	0

* كما قلنا سابقًا بأنّ $M_i = \overline{m_i}$ حيث أنّهم عكس بعض.
* نلاحظ بأن الرقم (1) سائد كثيرًا لذلك maxterm.

- Each column in the maxterm function table is the complement of the column in the minterm function table since M_i is the complement of m_i. $M_i = \overline{m_i}$ / $\overline{M_i} = m_i$

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)*** $\rightarrow f(x,y) = x\bar{y} + xy.$
  - ***Product of Maxterms (POM)***

Minterm Function Example

- **Example: Find $F_1 = m_1 + m_4 + m_7$**
- $F_1(x,y,z) = x'y'z + xy'z' + xyz$ → هدول الذين يكون جوابهم (1) بالـ minterm و جوابهم (0) بالـ maxterm (Some of minterm) (SOM)

$\underbrace{0\,0\,1}_{m_1} + \underbrace{1\,0\,0}_{m_4} + \underbrace{1\,1\,1}_{m_7}$

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	$0 + 0 + 1 = 1$

Minterm Function Example

- $F(A,B,C,D,E) = m_{\underline{\underline{2}}} + m_{\underline{\underline{9}}} + m_{1\underline{\underline{7}}} + m_{2\underline{\underline{3}}}$

أرقام الـ(maxterm) الذين قيمتهم =(صفر) وهم (٢، ٩، ١٧، ٢٣).

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

* ملاحظة: أرقام الـ(maxterms) تخبرنا عن الأماكن التي يكون فيها الـ (function) = صفر، (0).

وكذلك أرقام الـ (minterms) تخبرنا عن الأماكن التي يكون فيها الـ (function) = واحد، (1).

Maxterm Function Example

- **Example: Implement F1 in maxterms:**
- $F_1 = M_0 \,.\, M_2 \,.\, M_3 \,.\, M_5 \,.\, M_6$ (0,2,3,5,6) هي أماكن الأصفار: بالـ truthtable لأنه maxterms
- $F_1 = (x + y + z) \,.\, (x + y' + z) \,.\, (x + y' + z') \,.\, (x' + y + z') \,.\, (x' + y' + z)$

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

Canonical Sum of Minterms

- Any Boolean function can be expressed as a <u>Sum of Minterms (SOM)</u>:
  - For the function table, the <u>minterms</u> used are the terms corresponding to the 1's
  - For expressions, <u>expand</u> all terms first to explicitly list all minterms. Do this by "ANDing" any term missing a variable v with a term $(v + \bar{v})$

(y) ينقصه

- Example: Implement $f_{(x,y)} = x + \bar{x}\bar{y}$ as a SOM? not(SOM) but (SOP): (Some of products)

توزيع

1. Expand terms → $f_{(x,y)} = x(y + \bar{y}) + \bar{x}\bar{y}$
2. Distributive law → $f = xy + x\bar{y} + \bar{x}\bar{y}$
3. Express as SOM → $f = m_3 + m_2 + m_0 = m_0 + m_2 + m_3$

(3) 11 (2) 10 (0) 00

ترتيبهم تصاعدي ليس شرط!

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*:**
  - $F(A, B, C) = \sum_m(1,4,5,6,7)$

(Equal) (sum).

- **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 <u>Product of Maxterms (POM)</u>:
  - 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

$X \cdot (v + \bar{v}) = X$

$X + (v \cdot \bar{v}) = X$

- 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***

(multiply) (ضرب) الصورة المختصرة

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?

  $f(x,y,z) = m_1 + m_3 + m_5 + m_7.$

  - $\bar{F}(x,y,z) = \sum_m(0,2,4,6)$ (SOM)
  - $\bar{F}(x,y,z) = \prod_M(1,3,5,7)$ (POM)

$f(x,y,z) = \sum_m(1,3,5,7) = \prod_M(0,2,4,6).$

$\bar{f}(x,y,z) = \sum_m(0,2,4,6) = \prod_M(1,3,5,7).$

Conversion Between Forms

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

  نضع الأرقام الـ minterms في الـ complements عكس الموجودين بالـ (minterm) نفسه.

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

    8 variables, means: $2^3 = 8$, means from (0 → 7) هي رتبة الـ (minterms).

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

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

* لكي تحوّل من (minterm) إلى (maxterm)

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

لـ function معيّن: مثلاً لدينا function بدلالة الـ (minterm)

ونريد تحويله لـ (maxterm) نقوم بإيجاد الـ complement للـ function، ونبدّل الـ minterm بدلالة الـ minterm بدل الـ maxterm.

* مع مراعاة عدد الـ (variables). لأن عدد الـ (maxterm/minterm) هو 2^n. حيث n: عدد المتغيرات.

Important Properties of Minterms

- Maxterms are seldom used directly to express Boolean functions (نادرة)
  * Minterms are used more than Maxterms.

  * في (function) معيّن الـ(complement) هو نفسه الـ(maxterm) لذلك الـ(function) (كقيمة)

- Minterms properties:
  * إذا كان كل الـ(minterms) موجودين بالـ(function) معناه قيمته (1)
  * إذا كان كل الـ(maxterms) موجودين بالـ(function) معناه قيمته (0)
  - For n Boolean variables, there are 2^n minterms (0 to 2^n -1)
  - Any Boolean function can be represented as a logical sum of minterms (SOM)
  - The complement of a function contains those minterms not included in the original function
    * النفي للاقتران يحتوي على الحدود الغير موجودة بالاقتران نفسه (مكمّلة لها).
  - A function that include all the 2^n minterms is equal to 1

$f(x,y) = \Sigma m(0,1,2,3)$ ⇒ The function contains all(2^n) then it equals to(1).

* عندما تحتوي الـ(minterm) على جميع عناصر الـ(function) يكون الـ(function) = (1) ويكون جواب الـ(maxterms) لا يوجد
و الـ(complement) لـ(function) هو (0) دائماً.

Standard Forms

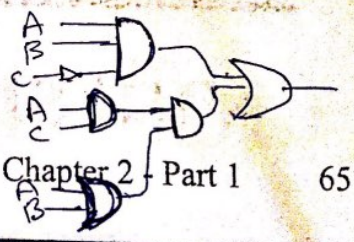
- Standard Sum-of-Products (SOP) form: equations are written as an OR of AND terms → (SOM) $f(x,y) = (x \cdot y) + (\bar{x} \cdot y) + (x \cdot \bar{y}) + (\bar{x} \cdot \bar{y})$.
- Standard Product-of-Sums (POS) form: equations are written as an AND of OR terms $f(x,y) = (x+y) \cdot (\bar{x}+y) \cdot (x+\bar{y}) \cdot (\bar{x}+\bar{y})$ (POM)

  * لا يشترط فيه احتواء جميع الحدود على جميع المتغيرات.
- **Examples:**
  - SOP: $ABC + \bar{A}\bar{B}C + B$ → not (SOM)
  - POS: $(A + B) \,.\, (A + \bar{B} + \bar{C}) \,.\, C$ → not (POM)
- **These "mixed" forms are neither SOP nor POS**
  - $(AB + C)(A + C)$
  - $AB\bar{C} + AC(A + B)$

  (OR) (and)

  * محايد لا (SOP) أو (POS) → تمثيله

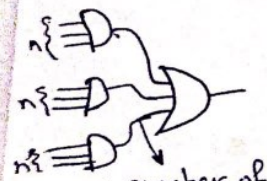
  بالتوزيع يصبح (POS) أو (SOP)
  ولكن كشكله هكذا يعتبر محايد.



Standard Sum-of-Products (SOP)

- A sum of minterms form for *n* variables can be written down directly from a truth table
- Implementation of this form is a two-level network of gates such that:
  - The first level consists of *n*-input AND gates, and
  - The second level is a single OR gate (with fewer than 2^n inputs)
- This form often can be simplified so that the corresponding circuit is simpler

* التمثيل لل (SOP) / شكل ال (design).



* number of minterms in the function = عدد ال (and gates)

* عدد ال (and gates) يساوي عدد ال (terms)

* عدد ال (inputs) = n = عدد ال Variables.

Standard Sum-of-Products (SOP)

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

* نكتب الـ(function) بدلالة الـ (minterm) لكي يسهّل (-- $m_0 + m_1 + m_3$ --) علينا تعبئة الـ(truth table) ولكي يسهّل التبسيط (simplify).

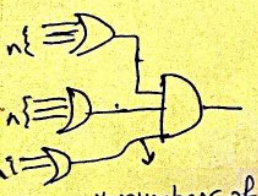
- Simplifying using boolean Algebra:

$A'B'C + AB'C' + AB'C + ABC' + ABC$	$m_1 + m_4 + m_5 + m_6 + m_7$
$= A'B'C + AB'(C' + C) + AB(C' + C)$	*Distributive law* * (use the laws & theorems)
$= A'B'C + AB' + AB$	$X + X' = 1$
$= A'B'C + A(B' + B)$	*Distributive law*
$= A'B'C + A \cdot 1$	*Simplification Theorem*
$= A + B'C$	* التمثيل بالرسم في (slide 68)

* تحويل من (SOM) إلى (SOP) بتقليل عدد الـ (inputs)

* التمثيل (pos/pom):

* شكل الـ(design):



* number of Maxterms in the function.

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

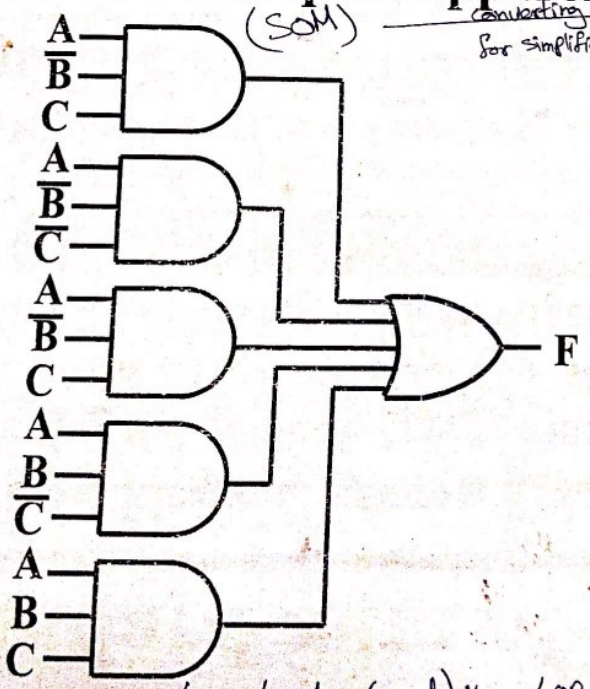
* في حالة الـ (pos): عدد الـ inputs لـ (OR gate) هو (n) = عدد الـ Variable. بينما عدد الـ inputs لـ (and gate) هو (number of maxterms).

AND/OR Two-level Implementation of SOP Expression

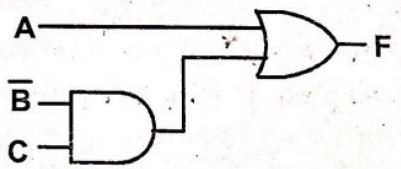
- **The two implementations for F are shown below – it is quite apparent which is simpler!**

* لتبسيط شكل الـ (circuit) فقط

(SOM) $\xrightarrow[\text{for simplification}]{\text{converting}}$ (SOP)



two levels (and gate) then (OR gate)

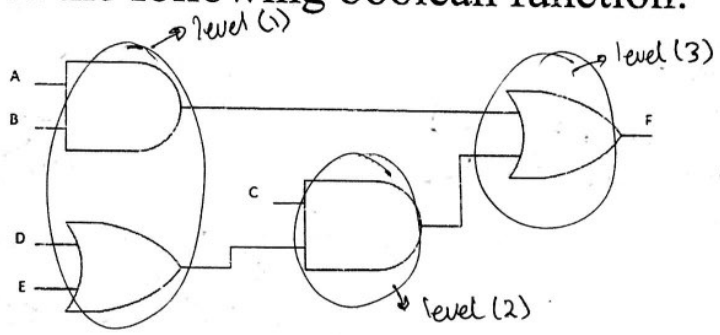


two levels (and gate) $\xrightarrow{\text{then}}$ (OR gate)

ولكن هذه مبسّطة أكثر.
وعدد البوابات أقل لتقليل
الـ (cost)

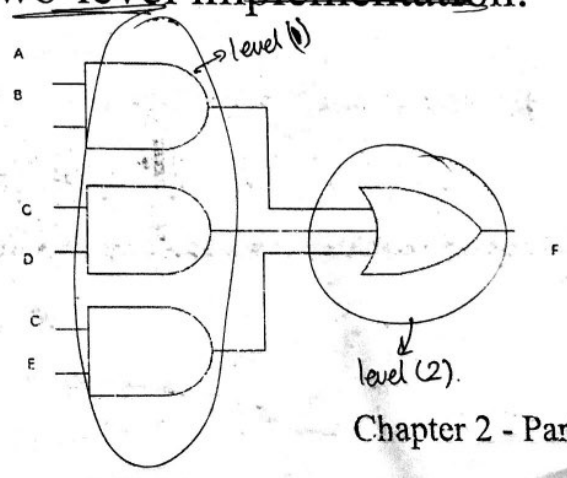
Two-level Implementation

- Draw the logic diagram of the following boolean function:
  - $f = AB + C(D + E)$



* شرط 8 – التمثيل باستخدام ال (gates) على مرحلتين.

- Represent the function using two-level implementation:
  - $f = AB + CD + CE$ → SOP



SOP and POS Observations

- The previous examples show that:
  - Canonical Forms (Sum-of-minterms, Product-of-Maxterms), or other standard forms (SOP, POS) differ in complexity
  - Boolean algebra can be used to manipulate equations into simpler forms.
  - Simpler equations lead to simpler two-level implementations

* عند التحويل من : SOP ⇄ SOM
POS ⇄ POM
تقلل الـ (cost).

- Questions:
  - How can we attain a "simplest" expression?
  - Is there only one minimum cost circuit?
  - The next part will deal with these issues.

Literal Cost (L)

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

*L :- عدد الأمثلة هو نفسه عدد الـ Variables

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:

  * G = L + number of terms excluding single Variable ter (باستثناء)

  - **All literal appearances (L)**
  - **The number of terms excluding single literal terms,(G) and**
  - **optionally, the number of distinct complemented single literals (GN).**

  * GN = L + number of terms + number of inverters

- Examples:
  - $F = BD + AB'C + AC'D'$ (number of terms = 3, L = 11)
    - $G = 11$, $GN = 14$ (Minimum cost → Best solution)

      L = 8, number of terms = 3

      number of variables complemented (inverters) = 3. لا نذكر المتغير المتكرر (تحسبه مرة وحدة فقط).
  - $F = BD + AB'C + AB'D' + ABC'$
    - $G = 15$, $GN = 18$
  - $F = (A + B)(A + D)(B + C + D')(B' + C' + D)$
    - $G = 14$, $GN = 17$

    * 4 terms.
    * 10 number of variables
    * Complemented variables: 3

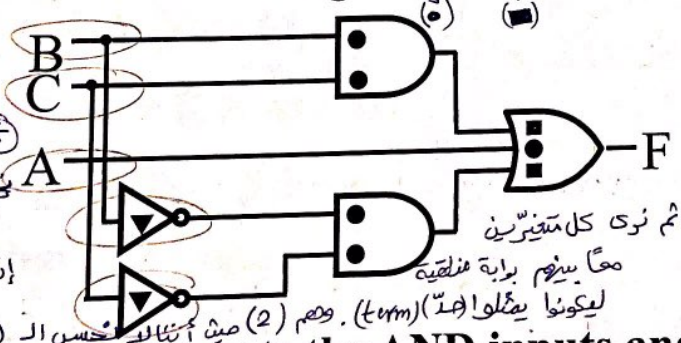
Cost Criteria (continued)

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

GN = G + 2 = 9

L = 5

G = L + 2 = 7



(L)

(*) من الشكل نرى الـ (input variables)

نجد أنّ عددهم (5) وهم $(\overline{B}, B)$ $(\overline{C}, C)$ (A)

إذاً (L = 5).

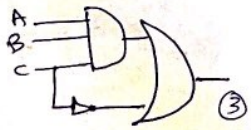
(*) ثم نرى كل متغيّرين معاً بينهم بوابة منطقية ليكونوا يمثلوا (حدّ) (term). وهم (2) حيث أننا لا نحسب الـ (term) الذي يحتوي متغير واحد.

(G = 2+5).

- **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**

*Example:-

$ABC + \overline{C}$.



① L = 4. (ABC) هو الحدّ (term) → $(\overline{C})$ لا تعتبر term

② G = 5 = 4 + 1

③ GN = 6 = 5 + 1 ($\overline{C}$)

(*) ثم نعدّ عدد الـ (complemented variables) ونجمعها لنجد (GN) وهي (7+2 = GN)

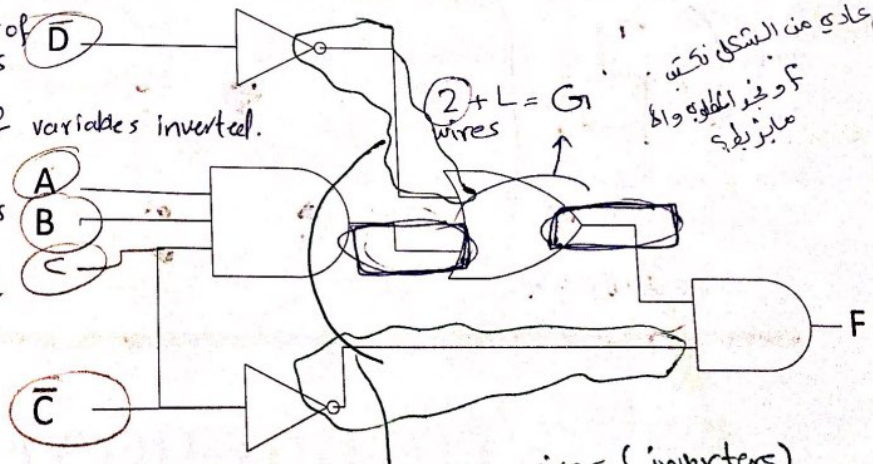
Cost Criteria (continued)

- **Example 2:**
- $F = (A, B, C, D) = (ABC + D').C'$
  - $L = 5$
  - $G = 5 + 2 = 7$ → because distributive law: $\underbrace{AB\bar{C}C}_{term①} + \underbrace{\bar{D}\bar{C}}_{term②}$.
  - $GN = 7 + 2 = 9$

* L = variables = number of inputs

* GN = G + number of variables inverted.

* G = L + number of terms that contains more than one variable.

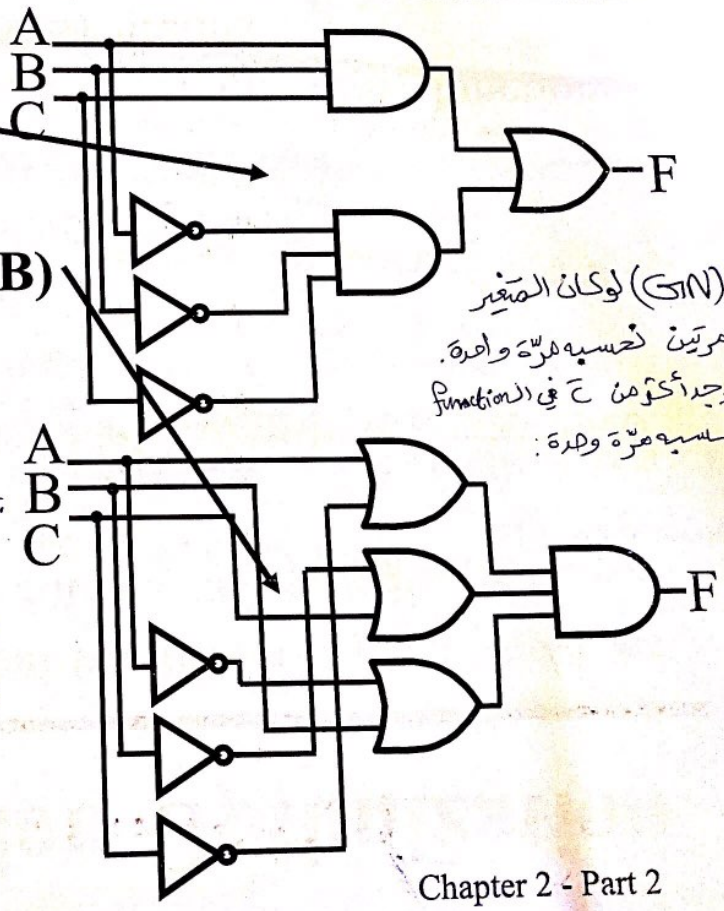


عادي من الشكل نكتب F وجد الحلول واذا مابزبط

GN = G + 2

Cost Criteria (continued)

- **Example 3:**
- **$F = A\,B\,C + \bar{A}\bar{B}\bar{C}$**
- **L = 6, G = 8, GN = 11**
- **$F = (A + \bar{C})(\bar{B} + C)(\bar{A} + B)$**
- **L = 6 , G = 9, GN = 12**
- **<u>Same</u> function and <u>same</u> literal cost** (L) . same variables number.
- **But first circuit has <u>better</u> gate input count** (G) **and <u>better</u> gate input count** (GN) **with NOTs**
- **Select it!**



* في (GN) لو كان المتغير منفي مرتين نحسبه مرّة واحدة.
يوجد أكثر من $\bar{C}$ في الـ function نحسبه مرّة واحدة:

Karnaugh Maps (K-map)

- A K-map is a collection of squares 2^n = عدد المربعات ، حيث n: عدد المتغيرات
  - Graphical representation of the truth table
  - Each square represents a minterm, or a maxterm, or a row in the truth table
  - For n-variable, there are 2^n squares
  - The collection of squares is a graphical representation of a Boolean function
  - Adjacent squares differ in the value of one variable
  - Alternative algebraic expressions for the same function are derived by recognizing patterns of squares

* يتم تجميع الـ (ones) أو (zeros) لكتابة الـ function بدلالة (minterm) / (maxterm) عكسيا.

* بين كل مربعين متجاورين يختلف الـ (code) بـ (one variable) فقط. 001 000 010 هكذا يشبه الـ (gray code)

أي بينهم ضلع مشترك. المربعات المتجاورة (متلامسة)

Two Variable Maps

- **A 2-variable Karnaugh Map:**
  - Note that minterm m_0 and minterm m_1 are "adjacent" and differ in the value of the variable y.

متجاورين

*كل مربعين متجاورين يختلفون (variable) في واحد.

x \ y	y = 0	y = 1
x = 0	$m_0 = \bar{x}\bar{y}$	$m_1 = \bar{x}y$
x = 1	$m_2 = x\bar{y}$	$m_3 = xy$

  - Similarly, minterm m_0 and minterm m_2 differ in the x variable
  - Also, m_1 and m_3 differ in the x variable as well
  - Finally, m_2 and m_3 differ in the value of the variable y

K-Map and Truth Tables → 2^n = عدد ال rows and columns.

2^n = عدد المربعات

- The K-Map is just a different form of the truth table

implementation of a function: Truth table. → K-maps.

- Example: Two variable function
  - We choose a,b,c and d from the set {0,1} to implement a particular function, $F(x, y)$

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

Truth Table

احتمالات (y)

احتمالات (x)

x \ y	y = 0	y = 1
x = 0	00 a	01 b
x = 1	10 c	11 d

K-Map

Same implementation for the function.

K-Map Function Representation

- Example: ① $F(x, y) = x$

$2^2 = 4$ squares and 4 اختصارات

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

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

نجمع

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

$x \cdot 1 = \boxed{x}$ جواب الـ function

* Example ②: $F(x,y) = y$.

$\sum_m(1,3)$ → مرتبة الذين جوابهم (1) بالـ function

$F(x,y)=y$	$y=0$	$y=1$
$x=0$	$\bar{x}\bar{y} = m_0$ ⓪	$\bar{x}y = m_1$ ①
$x=1$	$x\bar{y} = m_2$ ⓪	$\bar{x}\bar{y} = m_3$ ①

K-Map Function Representation

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

$F(x,y) = \Sigma_m(1,2,3) \rightarrow$

$\bar{x}y + x\bar{y} + xy \rightarrow$

$\bar{x}y + x(y + \bar{y})(1) \rightarrow$

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

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

المشتركة بينهم هو (y)

المشتركة بينهم هو (x)

* تجميع ال (1) يكون هو جواب ال function يعني : جواب هذا ال function هو (x+y)

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

يتم دمجهم .

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

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

m1 (01), m3 (11): $\bar{x}y + xy \rightarrow y(\bar{x} + x) = y \cdot 1 = y$

m2 (10), m3 (11): $x\bar{y} + xy \rightarrow x(\bar{y} + y) = x \cdot 1 = x$

Three Variable Maps

- **A three-variable K-map:** تم تبديل أماكنهم ؛ لكي يختلف كل مربع عن المربع المجاور له بـ (1 bit)

	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:**

$F(x,y,z) = \prod M(0,1,3,4) \rightarrow$ zeros

maxterms.

$F(x,y,z) = \sum m(2,5,6,7) \rightarrow$ ones

minterms.

	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	$xy\bar{z}$

كذلك (m_7 و m_6) (m_3 و m_2)

- ***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:

* نذكر الرموز المشتركة فيهم فقط.

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

by defult : تلقائيًا

X \ YZ	00	01	11	10
$\bar{X}$ 0	0	1	3	2
X 1	4	5	7	6

$\bar{y}$ by defult

$\bar{x}\bar{y}\bar{z} = 000$

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

$\bar{x}yz = 011$

$\bar{x}y\bar{z} = 010$

Y

Z

$\bar{Z}$

Example Functions

- By convention, we represent the minterms of F by a "1" in the map and leave the minterms of $\overline{F}$ blank (0) بدل
- Example:
  - $F(x, y, z) = \sum_m(2,3,4,5)$

    $\overline{F}$ = زيادة → $\prod M$ (0) = 6, 7).

			Y	
	0	1	3	2
	0	0	1	1
X	4	5	7	6
	1	1	0	0
			Z	

most significant = X

Z = least significant.

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

(zero's by defult)

			b	
	0	1	3	2
	0	0	1	0
a	4	5	7	6
	1	0	1	1
			c	

$2^3 =$ 3 Variables.

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

* ترتيب الـ Variables مهم بالـ function لكي نعرف ترتيبهم بالـ (K-map)

Functions

تجميع ال (ones)

- Enter the function on the K-Map
  - Function can be given in truth table, shorthand notation, SOP,…etc

    مثلاً $\Sigma m(2,3,4)$
  - 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

		y
	0 1	1 1
x	2 0	3 1

$2^2 = 4$ squares the (k-map) will be.

- Combining squares for simplification
  - Rectangles that include power of 2 squares {1, 2, 4, 8, …} 5 6 7 مضروب 8
  - Goal: Fewest rectangles that cover all 1's → as large as possible

    8 أفضل من 4

    4 أفضل من 2

    وهكذا
- Determine if any rectangle is not needed (not essential)
- Read-off the SOP terms

نكتب ما يطلع معنا (الناتج) بدلالة m_0 m_1 وهكذا

Combining Squares

- By combining squares, we reduce number of literals in a product term, reducing the literal cost, thereby reducing the other two cost criteria

- On a 2-variable K-Map: $F(x,y)$.
  - One square represents a minterm with two variables
  - Two adjacent squares represent a product term with one variable
  - Four "adjacent" terms is the function of all ones (no variables) = 1. $2^2=4$

- On a 3-variable K-Map: $F(x,y,z)$
  - One square represents a minterm with three variables
  - Two adjacent squares represent a product term with two variables
  - Four "adjacent" terms represent a product term with one variable
  - Eight "adjacent" terms is the function of all ones (no variables) = 1. $2^3=8$

Vimp

Example: Combining Squares

- Example: $F(x, y, z) = \sum_m(2,3,6,7)$
- $F(x, y, z) = \bar{x}y\bar{z} + \bar{x}yz + xy\bar{z} + xyz$
- Using Distributive law
  - $F(x, y, z) = \bar{x}y + xy$ ← By Boolean Expressions using theorems.

			y	
0	1	3	2	
		1	1	
4	5	7	6	x
		1	1	
		z		

* when we have a function with (2 variables)

① عندما يكون لدينا (1) ولا تجمّع مع (1) غيرها ← يكون تمثيله بمتغيّرين. (2variables)

② عندما يكون لدينا (1) مجمّعة مع (1) أخرى. ← يكون تمثيله بمتغيّر واحد. (1variable)

③ عندما يكون لدينا (1) مجمّعة مع (2 وامدات) أخرى. ← يكون تمثيله برقم ثابت (عدد) أي (0variables)

- Using Distributive law again
  - $F(x, y, z) = y$ ← by the k-map with combining the (ones together)
- **Thus, the four adjacent terms that form a 2×2 square correspond to the term "y"**

* عند تجميع الـ (ones) معًا يقلّ كل مرّة عدد الـ variables بمقدار واحد. (1)

Three-Variable Maps

- Reduced literal product terms for SOP standard forms correspond to rectangles on K-maps containing cell counts that are powers of 2

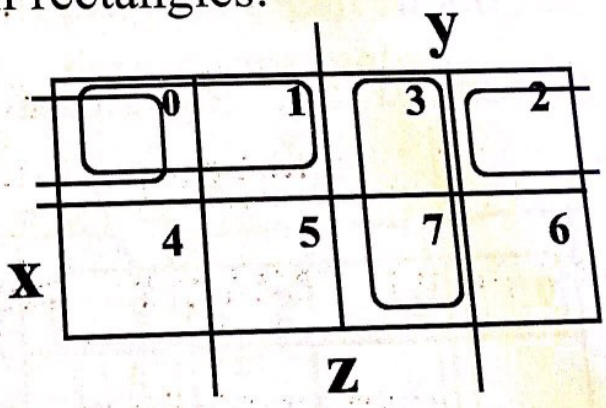
* عدد ال (ones) الذي يتم تجميعهم يجب أن يكون 2^n حول للعدد (2)

- Rectangles of 2 cells (squares) represent 2 adjacent minterms
- Rectangles of 4 cells represent 4 minterms that form a "pairwise adjacent" ring
- Rectangles can contain non-adjacent cells as illustrated by the "pairwise adjacent" ring above

Three-Variable Maps

- Example shapes of 2-cell rectangles:

* مكان الـ (x, y, z) ثابت دائماً
حسب الـ (mast significant) و (least significant).



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

نضع المتغير بالأول حسب الترتيب الأبجدي ووجوده في الـ function

Four Variable Terms

- Four variable maps can have rectangles corresponding to:
  - A single 1: 4 variables (i.e. Minterm) $F = (4)$ variables.
  - Two 1's: 3 variables * كل مرة يقل مقدار التمثيل بمقدار (1) (متغير واحد). $F = (3)$ variables.
  - Four 1's: 2 variables $F = (2)$ variables
  - Eight 1's: 1 variable $F = (1)$ variable
  - Sixteen 1's: zero variables (function of all ones) $F = 1$.

which means:

مهم *

* دائماً يتم تجميع الـ (1) بـ $2^4 / 2^3 / 2^2 / 2^1 / 2^0$ يعني power of (2)

16 8 4 2 1

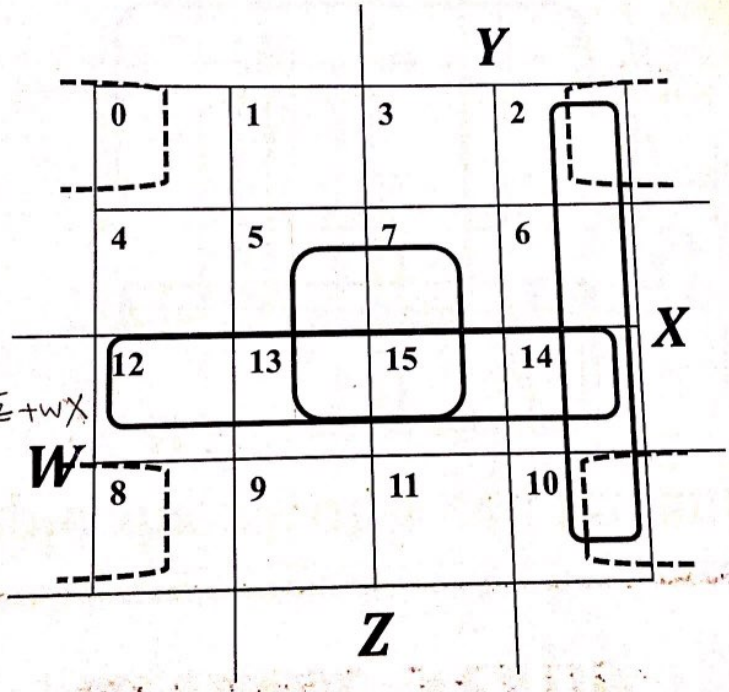
عدد الـ cells أو الـ rectangles.

Four-Variable Maps

- Example shapes of 4-cell rectangles:

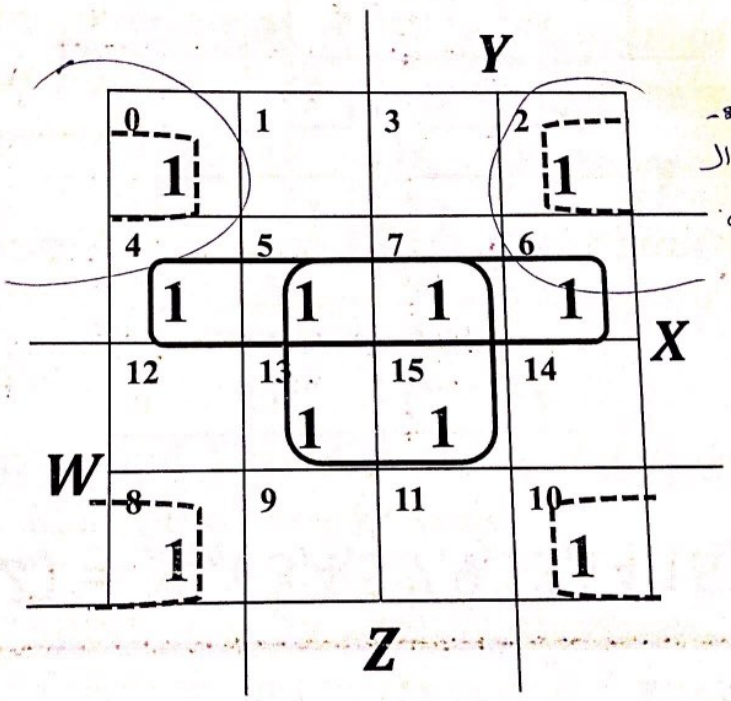
rec(5,7,13,15) : XZ.
rec(0,2,8,10) : $\bar{X}\bar{Z}$.
rec(2,6,10,14) : $Y\bar{Z}$.
rec(12,13,14,15) : WX.

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



Simplification

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

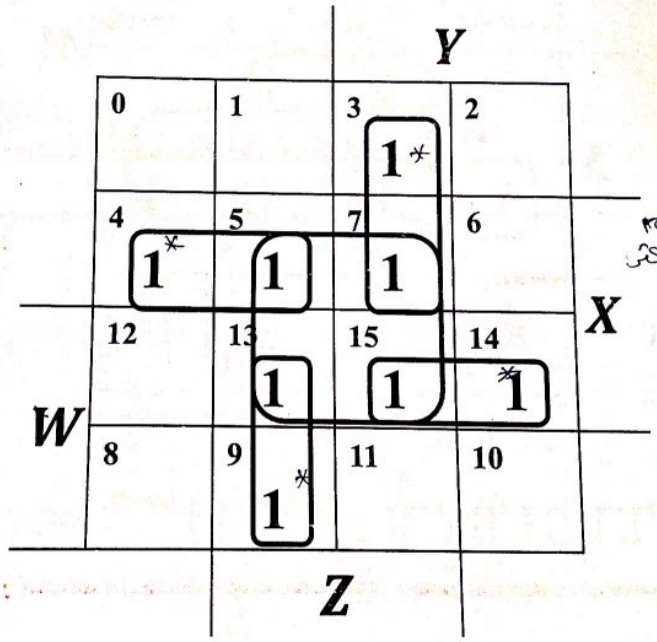


-8 (prime implicant) الـ
هو أن نجمّع أكبر عدد من الـ
(ones) معًا بأكبر مربّع.

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

Four-Variable Map Simplification

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



* Essential Implicants:-

* rec(9,13)
* rec (3,7)
* rec (4,5)
* rec (14,15)

5 - prime implicants.

rec(5,7) → prime implicant لا يكون

لأنه يوجد rec(5,7,13,15) وهي أكبر

لذلك تعتبر prime implicant

* Non - Essential :-

لأنه rec(5, 7, 13, 15) مكرر وال (ones) فيه مغطيين بأكثر من مربع آخر.

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

Systematic Simplification

term. * Implicant ≡ term.

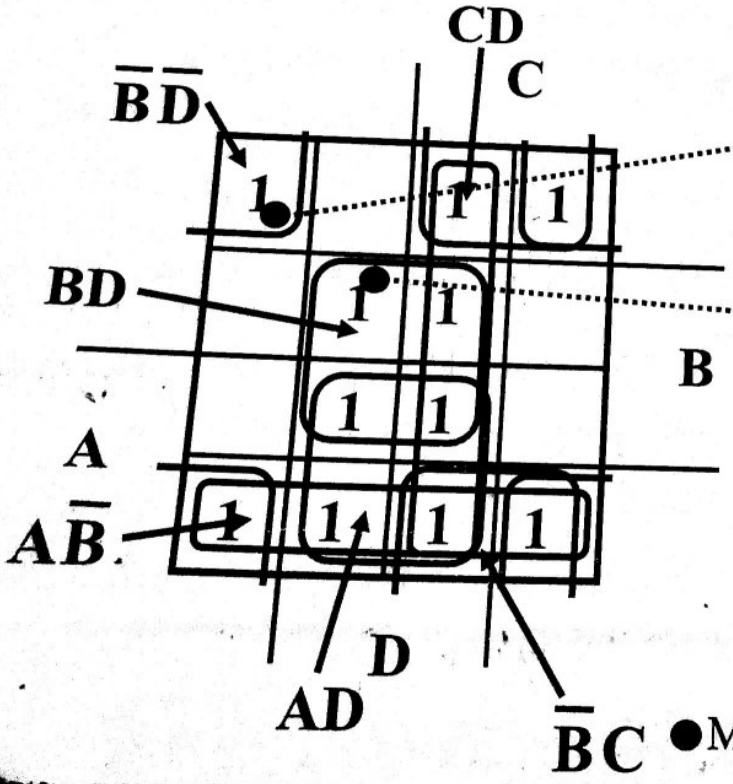
Essential.

Non-Essential.

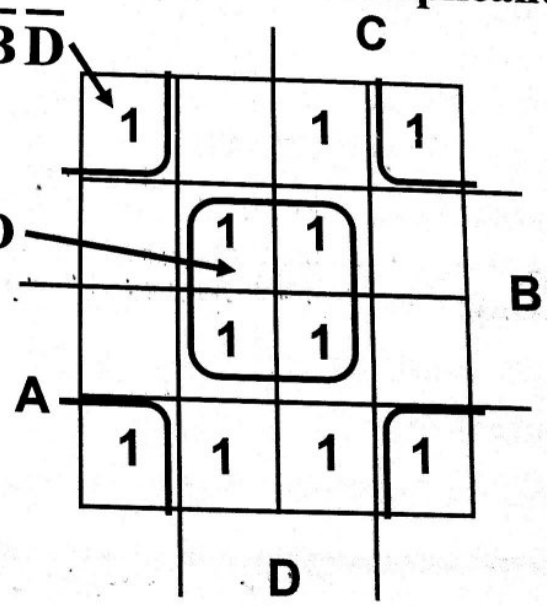
- ***Prime Implicant:*** is a product term obtained by combining the **maximum** possible number of adjacent squares in the map into a rectangle with the number of squares a power of 2 * the maximum possible number of ajacent terms.
  * لكل مجموعة أكبر امكان لتجميع الواحدات (ones).
- A prime implicant is called an ***Essential Prime Implicant*** if it is the **only** prime implicant that covers (includes) one or more minterms
  أساسي
- 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

Example of Prime Implicants

- **Find ALL Prime Implicants**



ESSENTIAL Prime Implicants



● Minterms covered by single prime implicant

من مربع (one) لأنه يغطيه أكثر من مربع آخر

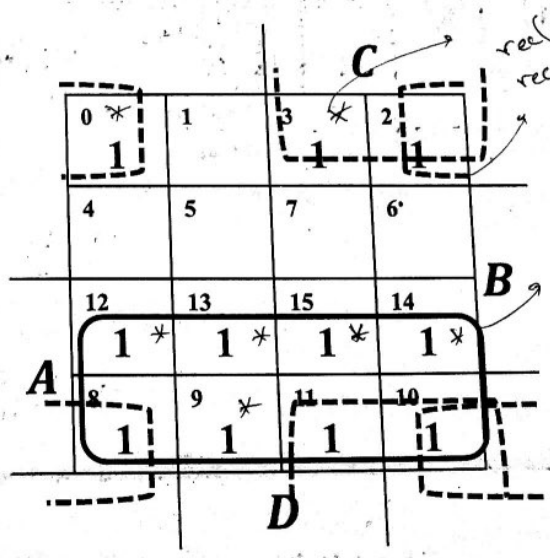
* انتبه! $A\overline{B}$ ليس prime implicant

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: [Essentials].
  - A
  - $\bar{B}C$
  - $\bar{B}\bar{D}$



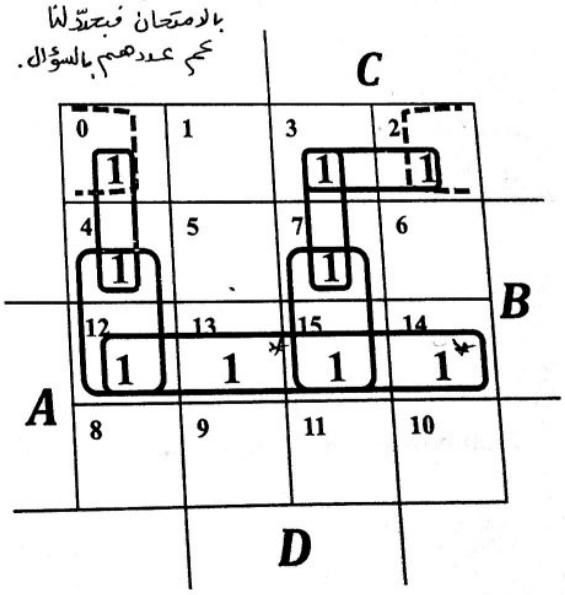
Another Example

- Find all prime implicants for:

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

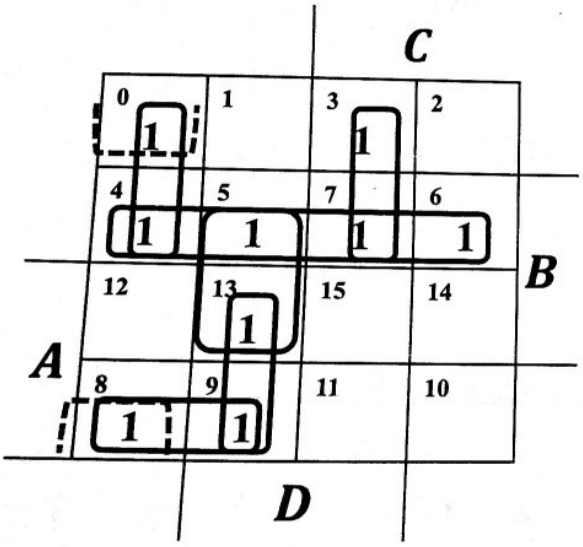
- Hint: There are seven prime implicants! لأن عددهم كثير بالامتحان فبحدّد لنا كم عددهم بالسؤال.
- Prime Implicants:
  - AB → The only one (Essential).
  - BCD
  - $B\bar{C}\bar{D}$
  - $\bar{A}CD$
  - $\bar{A}\bar{C}\bar{D}$
  - $\bar{A}\bar{B}C$
  - $\bar{A}\bar{B}\bar{D}$

* كل (one) مغطّى بأكثر من مرّة لذلك يكونوا (non-essential)

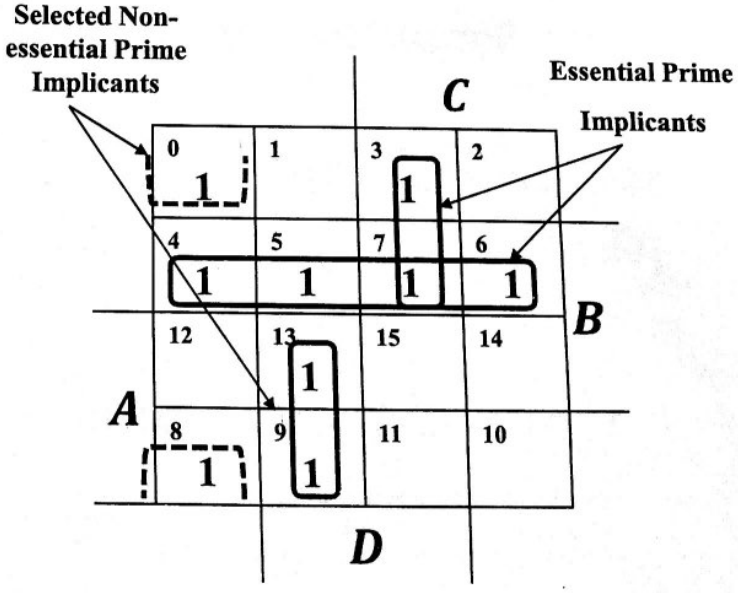


Selection Rule Example

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

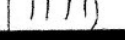


Prime Implicants



Essential and Selected Non-essential Prime Implicants

* ال (prime implicants) الأفضل أن يكون عدد الـ(ones) أقل


→ not prime implicant.

→ a prime implicant.

Product of Sums Example

- Find the optimum POS solution for: * أوجد ناتج الـfunction بدلالة (POS) لمصغر (SOP) كالعادة.

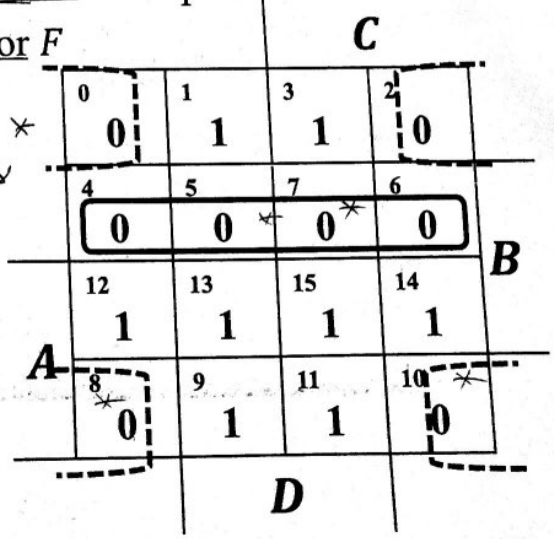
$$F(A,B,C,D) = \sum_{m}(1,3,9,11,12,13,14,15)$$

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

- $\bar{F}(A,B,C,D) = \bar{A}B + \bar{B}\bar{D}$ * نجمع الـzeros بالنفي لأنهم (one's) منفية

- Using Demorgan's Law:

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



Example

- Find the optimum POS and SOP solution for:

$\bar{F}$ وناخذ (F) لل(Zeros) نجمع.

(F) لل(Ones) نجمع.

$$*F(A,B,C,D) = \prod (0,2,4,5,6,7)$$

$*F(A,B,C,D) = \Sigma_m(1,3,8,9,10,11,12,13,14,15)$

M maxterms (Zero's)

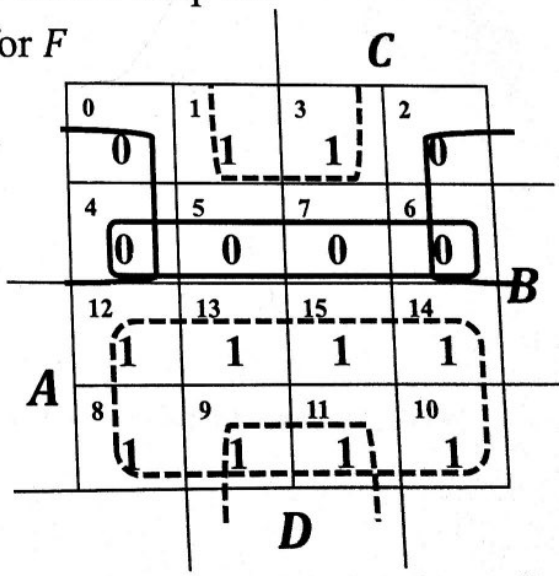
minterms (one's)

- 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

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

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

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

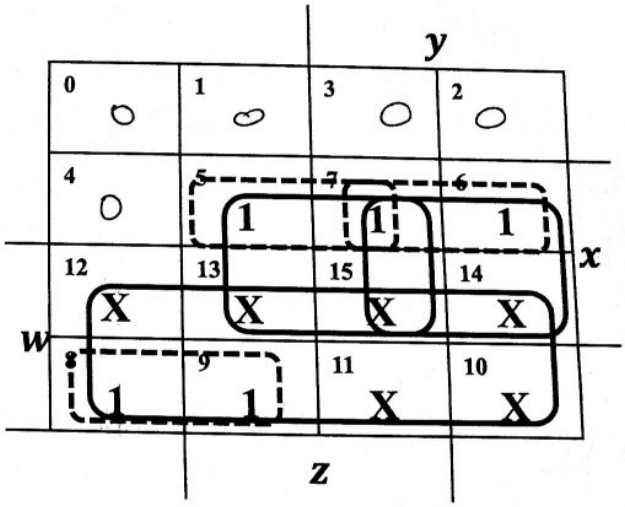


Don't Cares in K-Maps

- 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):
- $F_1(w,x,y,z) = w + xy + xz$
  - $G = 7$ $L=5$.
- If don't cares are treated as 0's (Blue):
- $F_2(w,x,y,z) = \bar{w}xz + \bar{w}xy + w\bar{x}\bar{y}$
  - $G = 12$ $L=9$

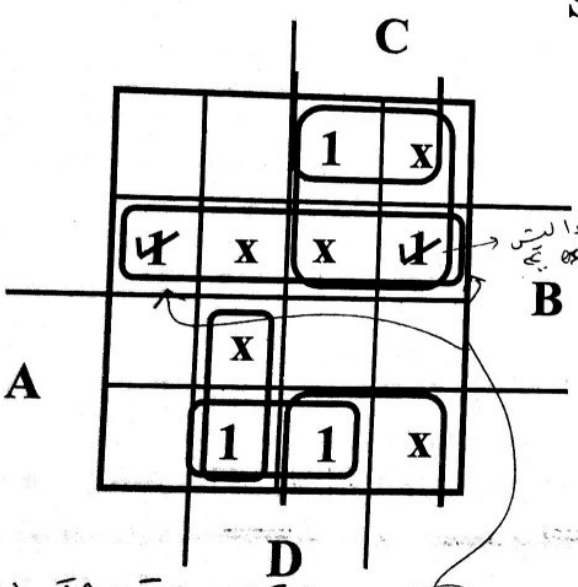
كيف؟!

* عند عدم تضمّن الـ (ones) تكون قيمتها في الـ (function) (صفر)

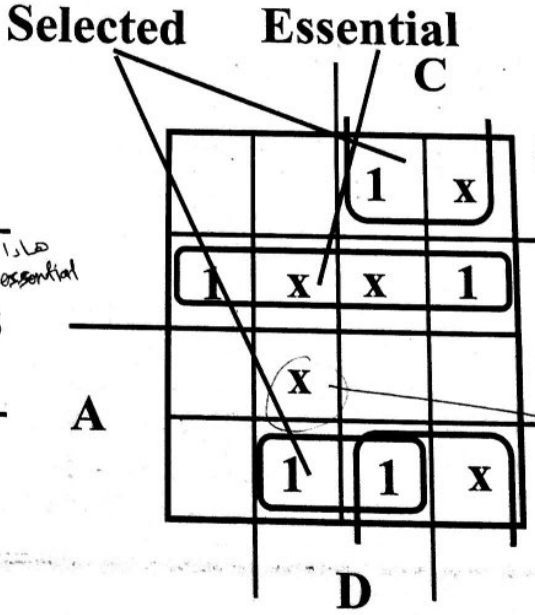
- ***For this particular function, cost G for the POS solution for F(w, x, y, z) is not changed by using the don't cares***
  - ***Choose the one less inverters (i.e. less GN)***

Selection Rule Example with Don't Cares

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



هادا ليش بيه essential



* لاحظ: هنعتبر الـ(X) هنا كأنها قيم (1) لكي نختصر أكبر عدد من الـ(ones) ولكن إذا لم يتم شمله (x) في مربعات لا يهم (عادي)

* $F(A,B,C,D) = \bar{A}B + \bar{B}C + A\bar{B}D$.
$G = 10$. $GN = 18$.

✓ Minterms covered by essential prime implicants

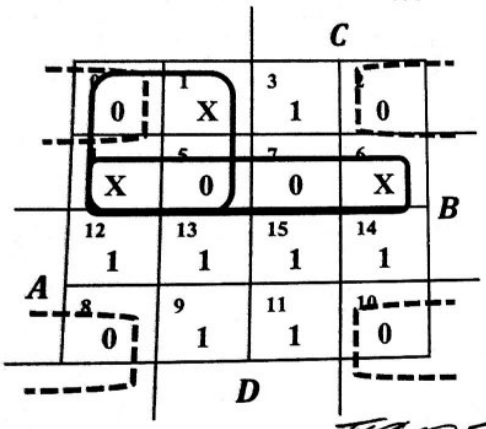
* $F(A,B,C,D) = \bar{A}B + \bar{B}C + A\bar{B}D$
$G = 10$. $GN = 12$. → (inverters) أقل لأنه أقل عدد

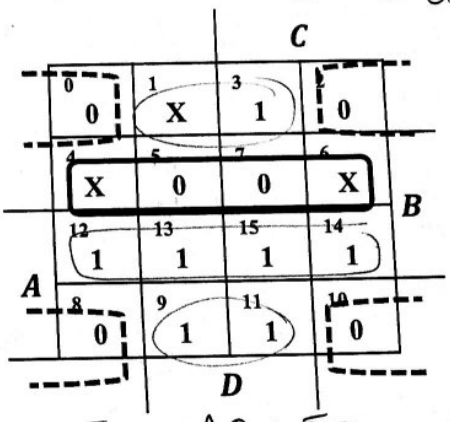
Product of Sums with Don't Care Example

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

$$F(A,B,C,D) = \sum_{m}(3,9,11,12,13,14,15) + \sum_{d}(1,4,6)$$

→ don't cares (x)





* $F_{sop} = AB + \bar{B}D$. → عكس ال (ones)

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

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

Five Variable or More K-Maps

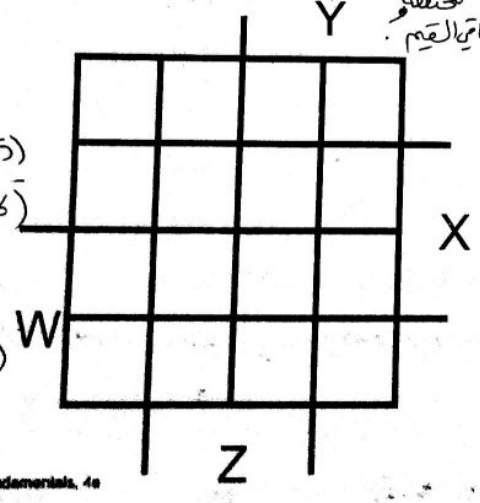
- For five variable problems, we use ***two adjacent*** *K-maps.* It becomes harder to visualize adjacent minterms for selecting PIs. You can extend the problem to six variables by using four K-Maps.

$2^5 = 32 =$ two K-maps
4 variables in each → 16
16

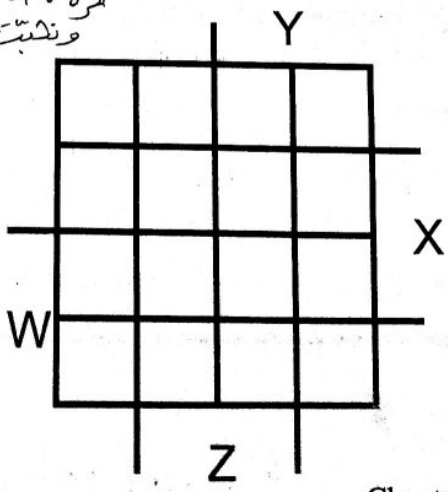
VWXYZ :
00000 (0)
⋮
01111 (15)
10000 (16)
⋮
11111 (31)

V = 0

* شرطنا كل مرة نأخذ قيمة مختلفة ونثبت باقي القيم.



V = 1



Other Gate Types

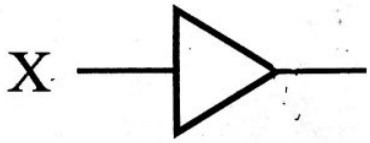
- **Why?**
  - Implementation feasibility and low cost
  - Power in implementing Boolean functions
  - Convenient conceptual representation
- **Gate classifications:**
  - بسيط **_Primitive gate:_** a gate that can be described using a single primitive operation type (AND or OR) plus an optional inversion(s). (NOT) (and & not ≡ OR & not). (AND, OR, buffer)
  - معقد **_Complex gate:_** a gate that requires more than one primitive operation type for its description (XOR, XNOR) (and & or)

Buffer

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

كأنه سلك لا يغير في قيمة الـ (inputs).

دائماً : inputs = outputs.



$X \quad \bar{X} \quad \bar{\bar{X}} = X = F$

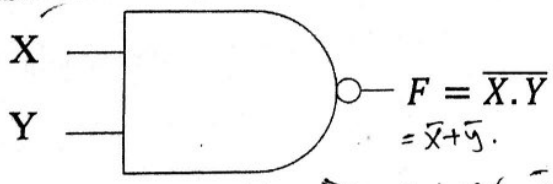
X	F
0	0
1	1

- In terms of Boolean function, a buffer is the same as a connection! (wire) سلك ماضي لا يغير قيمة الـ (inputs)
- **So why use it?**
  - A buffer is an electronic amplifier used to improve (1) circuit voltage levels and increase (2) the speed of circuit operation (signal مقوي)
  - Protection and isolation between circuits

NAND Gate

* يتم تنفيذ ال (and) أولاً ثم ال (not)

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

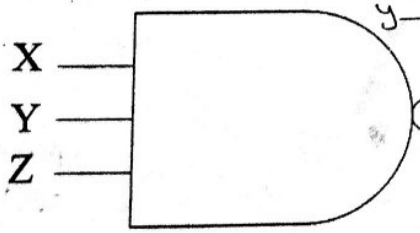


$F = \overline{X.Y}$

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

تساوي : F

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



$F = \overline{X.Y.Z}$

using demorgan's theorem:

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

- **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

NAND Gates (continued)

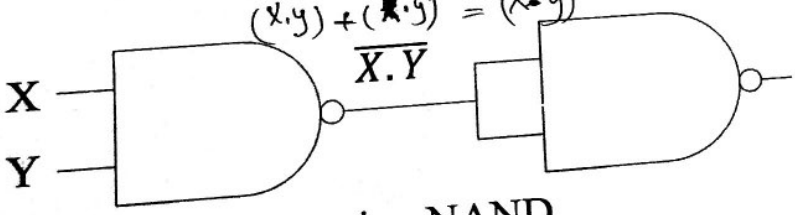
- ***Universal gate***: a gate type that can implement (تنفيذ / تمثيل) any Boolean function. **The NAND gate is a universal gate:** → * نستطيع أن نبني أي بوابة منطقية من (NAND) لوحدها فقط.

$F = \overline{(\overline{X \cdot y}) \cdot (\overline{x \cdot y})}$

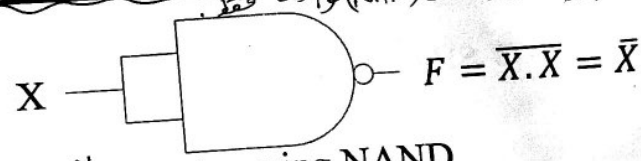
Demorgan's law:

$(\overline{\overline{X \cdot y}}) + (\overline{\overline{x \cdot y}}) =$

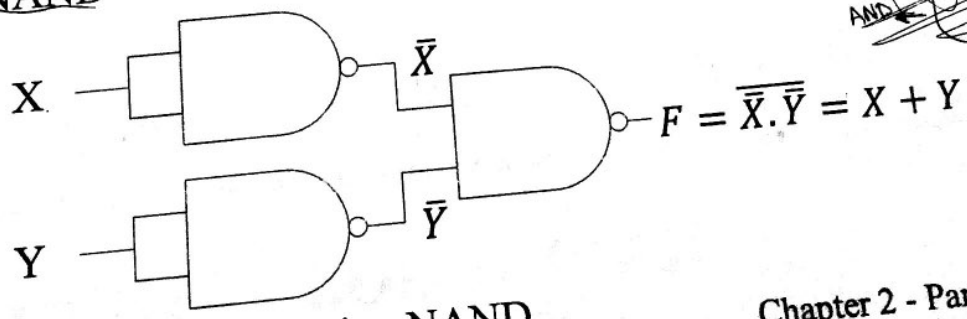
$(X \cdot y) + (x \cdot y) = (X \cdot y)$



* AND using NAND



* Inverter using NAND

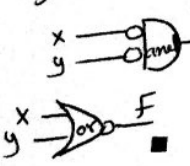


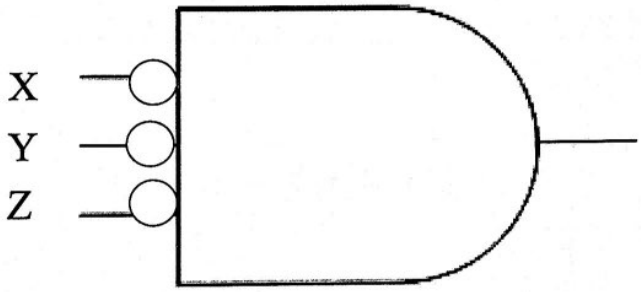
* OR using NAND

NOR Gates (continued)

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

* عندما نغيّر مكان ال inverter من ال output إلى ال input نعكس نوع ال gate.





$F = \overline{X+Y+Z} \rightarrow \bar{X}\cdot\bar{Y}\cdot\bar{Z}$

$F = \bar{X}.\bar{Y}.\bar{Z}$

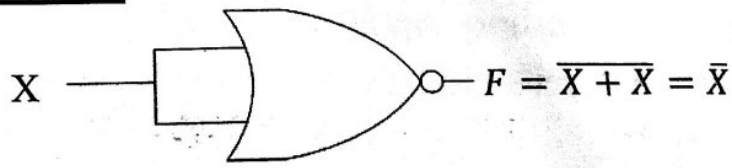
- This NOR symbol is called **Invert-AND**, since inputs are inverted and then ANDed together

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

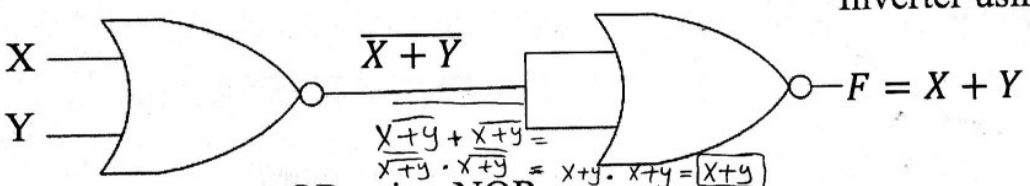
تصوّر

NOR Gates (continued)

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

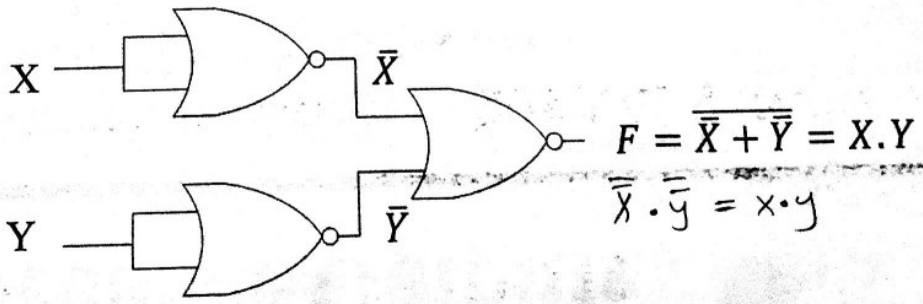


Inverter using NOR



$\overline{\overline{X+y}} + \overline{\overline{X+y}} =$
$\overline{\overline{X+y}} \cdot \overline{\overline{X+y}} = X+y \cdot X+y = \boxed{X+y}$

OR using NOR



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

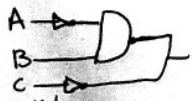
AND using NOR

(NAND) بدل (OR) نفس

Hi-Impedance Outputs (buffer) مرتبط بال

(primitive gate) التي هي

- Logic gates introduced thus far (2-output) لدينا *
  - have 1 and 0 output values, Values مختلفين
  - cannot have their outputs connected together, and the circuit is damage. مشبوكين مع بعض.
  - transmit signals on connections in only one direction



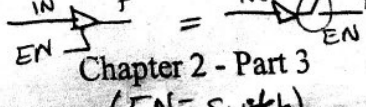
not(0) and not(1) then is a third new value.

- Three-state logic adds a third logic value, **Hi-Impedance (Hi-Z)**, giving three states 0, 1, and Hi-Z on the outputs. binary gates: { AND, OR, NOT }
- ***Hi-Z can be also denoted as Z or z*** لديهم احتمالين فقط للـ outputs (وهم: 0، 1).
- 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"

*Two-state buffers- IN → F=IN.

EN	IN	F
0	0	Z
0	1	Z
1	0	0
1	1	1

*Tri-state buffer (3 state buffer)



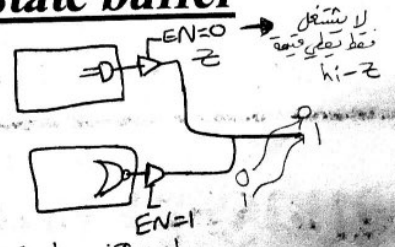
(EN ≡ Switch)

Hi-Impedance Outputs (continued)

- **What is a Hi-Z value?**
  - The Hi-Z value behaves as an open circuit
  - This means that, looking back into the circuit, the output appears to be disconnected
  - It is as if a switch (مفتاح) 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***

تقييد.



لا يشتغل فقط يعطي قيمة hi-Z

لما يشتغل ويعطي قيمة مشابهة لقيم ال inputs.

Tri-State Buffer (3-State Buffer)

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

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

  لا يشتغل

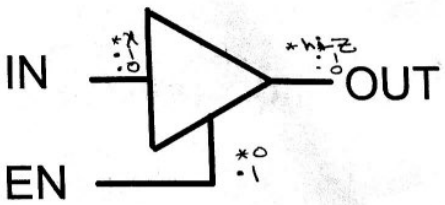
  * نغض النظر عن قيمة ال (inputs)

  عندما يكون ال EN ← (0) فإن ال (output) تكون (Hi-Z) يشتغل

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

  * عندما يكون ال EN ← (1) فإن ال (output) تكون مشابهة لل (input) أي أنها تصرّف مثل (buffer).

Symbol



Truth Table

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

don't care ← لا يهمّ (لا يؤثر)

لا تشتغل ال switch

تشتغل ال switch

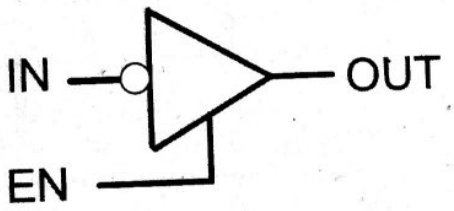
Same

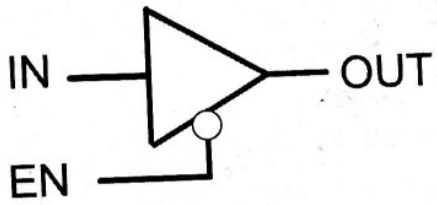
حيث (X) ال (input) يكون function ناتج ال

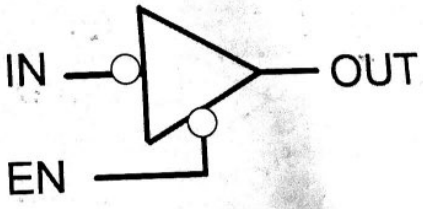
Tri-State Buffer Variations

0 1 hi-z

- By adding "bubbles" to signals:
  - Data input, IN, can be inverted (by bubbles)
  - Control input, EN, can be inverted (by bubble).







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

دائماً نغض النظر عن ال (input)

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

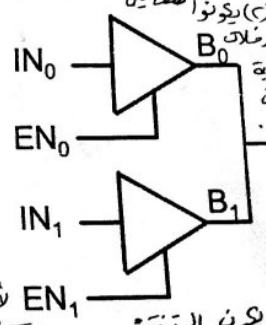
EN	IN	OUT
0	0	1
0	1	0
1	X	Hi-Z

يكون (D) صوال (ENable) لهما و نحدد ال (input) حسب

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
[redacted]						



Resolving 3-State Values on a Connection

- **Resulting Rule: At least one buffer output value must be Hi-Z, Why?** *At least one tri-state buffer must be inactive (hi-z)

  - 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 (Fire)

- **How many valid buffer output combinations exist?**
  - 5 valid output combination

  # of legal outputs. = 5. → the maximum number of outputs. يعني ممكن أقل من (5) يكون
  number عدد الـ EN عندما (n=2) → القاعدة - (2n+1).

- **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.

*at most 1 tri-state buffer must be active

slide

*Ex:- n=2 (1) must be hi-z , n=4 (3) must be hi-z

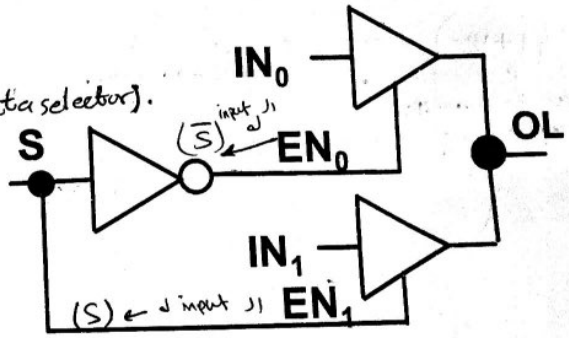
(1) at most buffer must be active (1)

Tri-State Logic Circuit

*طريقة التي تقلل مساحة الرسمة.

- **Data Selection Function:** If s = 0, OL = IN_0, else OL = IN_1
- Performing data selection with tri-state buffers:

$\bar{s}$	EN_1	EN_0	IN_1	IN_0	OL
0	0	1	X (hi-z) لأنه don't care	0	0
0	0	1	X	1	1
1	1	0	0	X (hi-z) لأنه don't care	0
1	1	0	1	X	1



بما أن وحدة من (EN0) ال input له $(\bar{s})$ والثاني (EN1) ال input له (s) إذا لا يكونوا مختلفين دائماً (0,1) أو (1,0) بالتالي هاد فقط دوري

- **Since $EN_0 = \bar{s}$ and $EN_1 = s$, one of the two buffer outputs is always Hi-Z.**

$(\bar{s}) \leftarrow 0$ 1
$(s) \leftarrow 1$ 0

Logic Functions using Tri-State Buffers

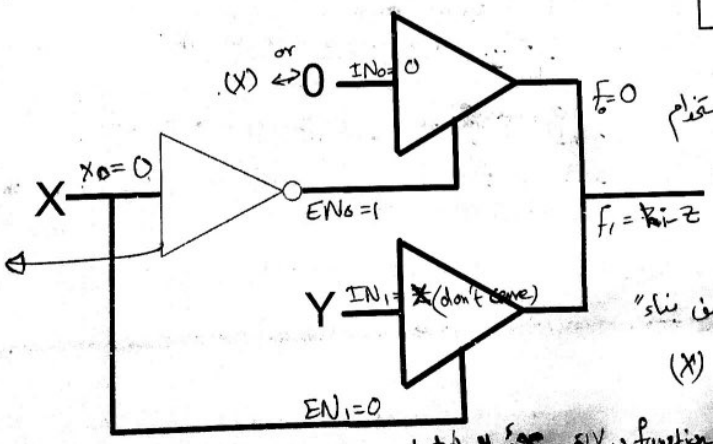
- Implement AND gate using 3-State buffers and inverters Tristate buffers. + inverters to implement any gate we want.

$$F(X,Y) = X.Y$$

- Use **X** as control input: (* 2input AND gate) * this implementaion is :

  - When $X = 0$, $F = 0$ regardless of the value of Y → or $F = X$
  - When $X = 1$, $F = Y$

X	Y	F	
0	0	0	F = constant. F = 0 أو
0	1	0	F = X
1	0	0	F = y
1	1	1	F ≠ constant



constant أو y أو x
↔ 0 أو 1

* ملاحظة: ① دائماً أن الـ outputs مشبوكين معاً في سلك واحد بالأخير

② يجوز بدل ▷ كاملة وضع (o) bubble على المكان لـ buffer مباشرة أي لا يلزم كتابة إشارة الـ inverter كاملة فقط الـ bubble على الـ buffer

* الخطوات لتمثيل أي gate باستخدام → inverters + Tri-state buffer

① نرسم الـ (Truth table) للـ (function) المراد تمثيله.

② نقسم الـ (truth table) من النصف بناءً على الـ most significant variable وهي الـ (X)

③ بعد ما نقسم الجدول نحاول نطلع الـ function برا الـ X أو y

Logic Functions using Tri-State Buffers

- Implement the following function using 3-State buffers and inverters: $F(w, x, y) = \bar{w}x + w\bar{y} + xy$

نبحث عن يكون جوابهم (1) بالـ function . والباقي (غير ذلك) جوابهم (0) يكون بلا function .

$\bar{w}x \rightarrow \begin{array}{ccc} 0 & 1 & 0 \\ 0 & 1 & 1 \end{array} + w\bar{y} \begin{array}{ccc} 1 & 0 & 0 \\ 1 & 1 & 0 \end{array} + xy \begin{array}{ccc} 1 & 1 & 1 \\ 1 & 1 & 1 \end{array}$

(الأرقام مختلفة) بذلك نضع كل احتمالاته

- Use w as control input:
  - When $w = 0$, $F = x$ regardless of the value of Y
  - When $w = 1$
    - If $x = 0$, $F = \bar{y}$
    - If $x = 1$, $F = 1$

* Note :- *

x : Variable(x)

X : don't care value (0 أو 1)

* إذا أعطاني بالسؤال الرسمة وطلب كتابة الـ function الممثل لها : الخطوات:

① عند كل مثلث buffer نكتب الـ input الـ Enable مع anded . سواء كانوا true أو inverted .

② ثم بين مثلثين الـ buffer نقل تاج الـ buffer الأول مع ORed تاج الـ buffer الثاني .

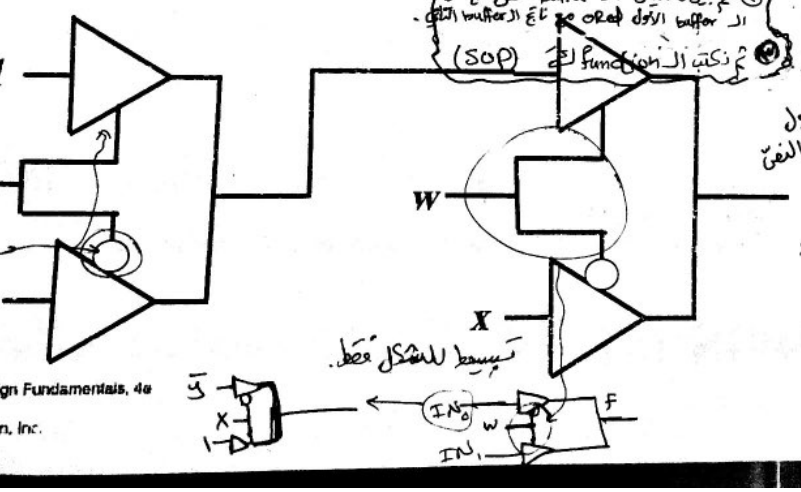
③ ثم نكتب الـ function الـ (SOP)

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

$F = 1$ (cons

قسمنا الجدول بالنص بناءً على (w)

قسمنا الجدول مرة ثانية بالنص بناءً على (x) التالية لأعلى قيمة the next most significant variable.



* Note :- لا يفرق مكان وضع الـ bubble لل inverter فوق أو تحت و لكن المهم أن تكون بمكان واحد منهم موالتين أو ولا واحد

تبسيط للشكل فقط.

Logic Functions using Tri-State Buffers

* مهم جدًا أكيد جاي عليه سؤال !

- Write the Boolean expression of $F(A,B,C)$ given the diagram below: $F(A,B,C) = A\bar{B}C + \bar{A}BC$

* لحل هذا السؤال إذا أعطاني الـfunction وطلب الـimplement :-

① نرسم الـ(truth table) وذلك

2^n : n : عدد المتغيرات . $2^3 = 8$

$\frac{8}{2} = \frac{4}{2} = \frac{2}{2} = 1$

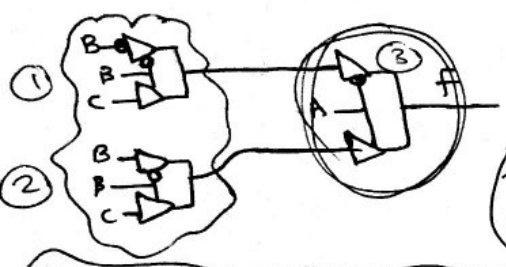
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

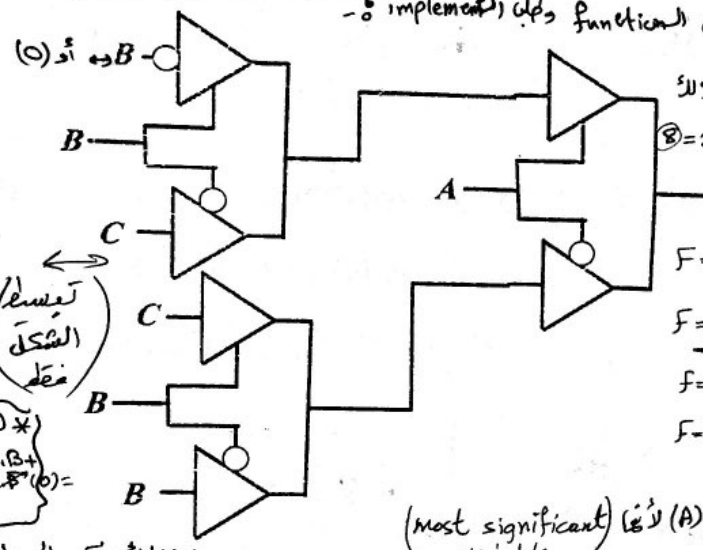
$A\bar{B}C$: 101
$\bar{A}BC$: 011

جوابهم (1) بالـfunction وما سواهم جوابهم (0)

F = B
F = C
F = C
F = $\bar{B}$

* مقطع الرسم :-





تبسيط الشكل مقطع

* (كتابة الـ function الـ implementation) نتبع التالي :

① $\bar{B}\cdot B$ → $\bar{B}B + \bar{B}\cdot C = \bar{B}\cdot C$ (0)
$\bar{B}\cdot C$

② $C\cdot B$ → $C\cdot B + B\cdot\bar{B}$ (0) = $C\cdot B$
$B\cdot\bar{B}$

③ $A\cdot\bar{B}\cdot C + A\cdot B\cdot C$ هو المطلوب

② نقسم الجدول بالنصّ بالنسبة للـ(A) لأنها (most significant variable) و لكنه لا يربط لذلك نقسم الجدول مرّة أخرى بالنسبة لـ (B) لأنها the next most significant variable.

③ نحاول كتابة النصف العلوي بدلالة $F = X$ أو $F = Y$ أو $F = 0$ و لكنه لا يربط لذلك نقسم الجدول مرّة أخرى بالنسبة لـ (B) لأنها the next most significant variable.

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

④ نبدأ الرسمة من الـ output (F) ثم نرجع للوراء ونرسم (2-buffers) ونضع (A) هي الـ (ENable) لها و وحدة مهم عليها (bubble) في .

⑤ نبدأ نحدّد الـ (inputs) للـ (buffers) وصمم بناءً على (ENable) (A) كالتالي نرسم (2 buffers) لكل (input) .

⑥ يكون (B) صوال (ENable) لهما و نحدّد الـ (inputs) حسب ما تم كتابة الـ function بدلالة : أي $\bar{B} = F$ وهكذا

$C \rightarrow = C = F$

The complex gates

Exclusive OR/ Exclusive NOR

- The ***eXclusive OR*** (***XOR***) function is an important Boolean function used extensively in logic circuits

بشكل واسع

- The XOR function may be:
  - implemented directly as an electronic circuit (truly a gate) or
  - implemented by interconnecting other gate types (used as a convenient representation)
- The ***eXclusive NOR*** (***XNOR***) function is the complement of the XOR function
- By our definition, XOR and XNOR gates are ***complex gates***

Proof: XNOR is the complement of XOR

(معقدة)(XOR) (بسيطة)(OR)

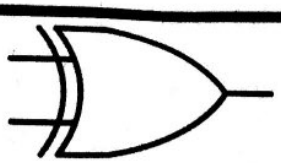
- $\overline{X \oplus Y} = \overline{\overline{X}Y \oplus X\overline{Y}}$

* انه لا يجوز استخدام (demorgan's law) بل يجب تحويل الـ (complex gates) إلى (primitive gates)

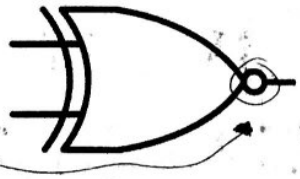
- $\overline{X \oplus Y} = \overline{\overline{X}Y}.\overline{X\overline{Y}}$
- $\overline{X \oplus Y} = (X + \overline{Y}).(\overline{X} + Y)$ distributive law.
- $\overline{X \oplus Y} = \cancel{X.\overline{X}}^{(0)} + XY + \overline{X}\overline{Y} + \cancel{Y.\overline{Y}}^{(0)}$
- $X \odot Y = \boxed{\overline{X \oplus Y}} = \boxed{XY + \overline{X}\overline{Y}}$

Symbols For XOR and XNOR

- XOR symbol:



- XNOR symbol:



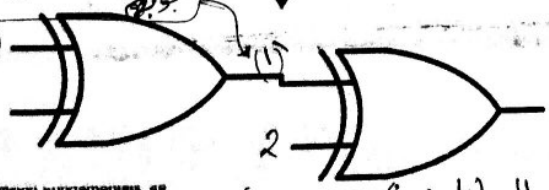
- ***Shaped symbols exist only for two inputs***

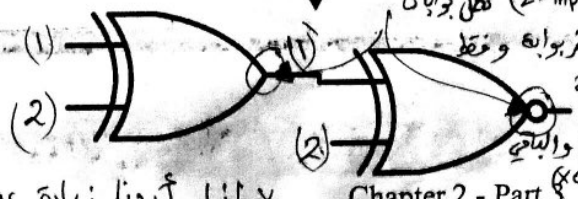
* مهم جداً : فقط يسمح لهم بـ (2-IN)

wrong! 3-inputs. ←

$(1)' = 1 \oplus 2$ → ناتج جوابهم

Right! 2-inputs only





* انتبه! : عند عمل بوابة (X NOR) لأكثر من (2-inputs) نعمل بوابات قبل آخر بوابة وفقط (XOR) البوابة الأخيرة تكون (X NOR) والباقي (XOR) تحويل عليه

* إذا أردنا زيادة عدد الـ (inputs) عن (two) نقوم بإضافة (gates) أخرى.

Truth Tables for XOR/XNOR

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

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

من ال equation $(\bar{x}y + x\bar{y})$

(1) function * يجب أن يكونوا (x,y) مختلفين ليكون ناتج ال

دليل أنهم complement to each other

(1) * يجب أن يكونوا (x,y) متشابهين ليكون الناتج

من ال equation $(\bar{x}\bar{y} + xy)$.

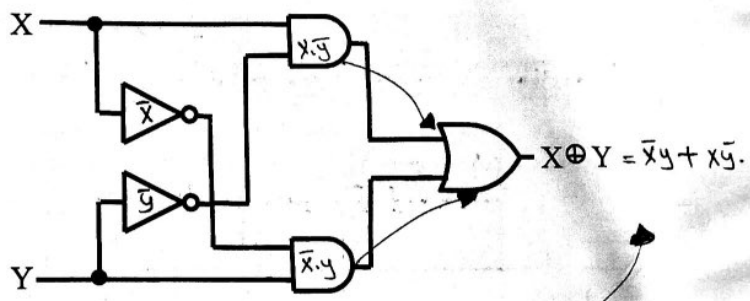
- The XOR function means: ***X OR Y, but NOT BOTH***

- Why is the XNOR function also known as the ***equivalence*** function, denoted by the operator ≡?
  - Because the function equals 1 if and only if $\boxed{X = Y}$

(XNOR Gate)
لأنها تتأكد من أن (x,y) قيمتين متساويتين.

XOR Implementations

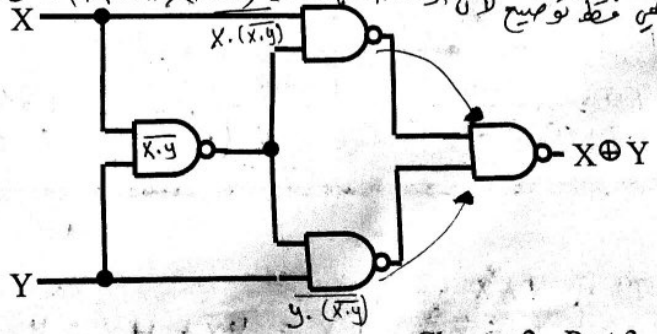
- **The simple SOP implementation uses the following structure:**



- **A NAND only implementation is:**

** غير مطلوب حفظ هاي الرسمات ولكن هي فقط توضيح لأن الـ equations لـ (XOR)(XNOR) صحيحات.

* نريد الحصول على (XOR gate) باستخدام (NAND gates) فقط كالتالي :-



XOR

⑤ 2- variables:
A = $\bar{y}$ نفرض ثم نكتب معادلة الـ (XOR) ثم نرجع قيمة (A)

$X \oplus A : \bar{X}A + X\bar{A} = \bar{X}\cdot\bar{y} + X\cdot\bar{\bar{y}} = \bar{X}\cdot\bar{y} + X\cdot y = X \odot y = \overline{X \oplus y}$

كأننا عملنا X (X NOR) y

- The XOR identities: هدولا حفظ + مهم استخدامهم. (مسلّمات ثابتة)

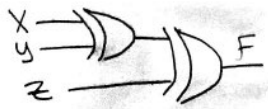
① $X \oplus 0 = X$	③ $X \oplus 1 = \bar{X}$
② $X \oplus X = 0$	④ $X \oplus \bar{X} = 1$
⑤ $X \oplus \bar{Y} = \overline{X \oplus Y}$	⑥ $\bar{X} \oplus Y = \overline{X \oplus Y}$
$X \oplus Y = Y \oplus X$ لا يهمّ إذا تم تبديل أحدهما كمنهم. (خاصيّة تبديليّة)	
$(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z) = X \oplus Y \oplus Z$ لا يهمّ تجميع أول حدّين أو ثاني حدّين (لا تفرق) (خاصيّة تجميعيّة)	

① مختلفين $1 \oplus 0 = 1$, $0 \oplus 0 = 0$

② لأنهم متشابهين $0 \oplus 0 = 0$, $1 \oplus 1 = 0$

③ حسب القيم التي بعدها يختلفان $1 \oplus 1 = 0$, $0 \oplus 1 = 1$

④ لأنهم مختلفين إذاً دائماً الناتج (1) $1 \oplus 0 = 1$, $0 \oplus 1 = 1$



- The XOR function can be extended to 3 or more variables. * مهمّة

For more than 2 variables, it is called an ***odd function*** or ***modulo 2 sum*** (***Mod 2 sum***), not an XOR:

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

001 + 010 + 100 + 111

* فردي (1's) ← (number) معناها

* نستطيع كتابة الـ Boolean function بدون رسم truth table عن طريق معرفة أي الصفوف يكون مجموع الـ (1's) فيها فردي يكون ناتج الـ function (1) وبالتالي نكتبه كحدّ.

* إذا كان عدد الـ (ones) في الـ (input) فردي (odd) يكون ناتج الـ function هو (1).

* بينما إذا كان عدد الـ (ones) في الـ (input) زوجي (even) يكون ناتج الـ function هو (0).

* يعني قاعدة الـ (2-variables) حالة خاصّة من قاعدة الـ (3 variables وأكثر).

* بينما إذا كان عدد المتغيرات (2) نعتمد القاعدة بأن إذا كان الـ (inputs) مختلفين يكون ناتج الـ function (1) بينما متشابهين يكون ناتج الـ function (0)

في حالة كان عدد المتغيرات أكبر من (2) نعمل الـ (XOR)

XNOR

*ليس شرط فقط هذه الـ(Identities) حيث لو عوّضنا القيم (x,y) بالامتحان نستطيع أن نجد الجواب ببساطة.

- The XNOR identities:

① $X \odot 0 = \overline{X}$	② $X \odot 1 = X$
③ $X \odot X = 1$	④ $X \odot \overline{X} = 0$
$X \odot Y = Y \odot X$ (خاصيّة تبديليّة)	
$X \odot Y \odot Z = (X \oplus Y) \odot Z = X \odot (Y \oplus Z)$ (خاصيّة تجميعيّة)	

① $X \odot 0 = \overline{X}$: $1 \odot 0 = 0$ (متخالفين), $0 \odot 0 = 1$ (متشابهين)

② $X \odot 1 = X$: $1 \odot 1 = 1$ (متشابهين), $0 \odot 1 = 0$ (متخالفين)

③ $0 \cdot 0 = 1$, $1 \cdot 1 = 1$ دائماً متشابهين إذاً الناتج دائماً (1).

④ $1 \odot 0 = 0$, $0 \odot 1 = 0$ دائماً مختلفين إذاً الناتج دائماً (0).

- 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:

*ملاحظة: عندما يكون الحدّ (minterm): 000 يكون عدد الـ (1's) زوجي لأنه لا يوجد (1's) إذاً Number of ones = even number of ones.

*إذا كان عدد الـ (ones) في الـ (input) زوجي يكون ناتج الـ function (0) في الـ (XOR) و (1) في الـ (XNOR)

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

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

Reason: *السبب: نقلب (inverter) فقط على آخر (gate)

Odd and Even Functions

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

		y	
	0: 0	1: 1	3: 0	2: 1
x	4: 1	5: 0	7: 1	6: 0
		z	

(K-maps)

		C	
	0: 0	1: 1	3: 0	2: 1
	4: 1	5: 0	7: 1	6: 0
A	12: 0	13: 1	15: 0	14: 1
	8: 1	9: 0	11: 1	10: 0
		D	

(نبدأ (0) →)

* شكلهم مميز مربع (1) و مربع جنبه (0) ونلاحظ عدم قدرتنا على تجميع الـ(ones) أبداً. ولكن من الشكل نعرف أنه even / odd function.

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

		y	
	0: 1	1	3: 1	2
x	4	5: 1	7	6: 1
		z	

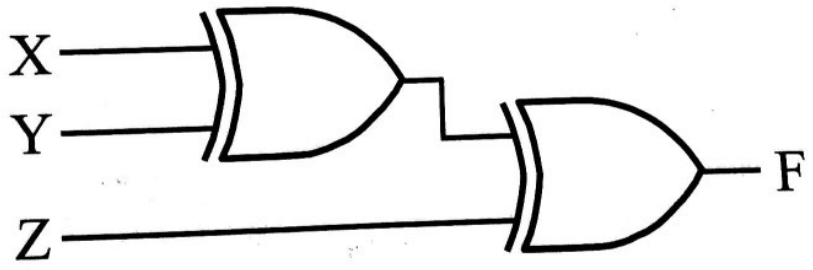
		C	
	0: 1	1	3: 1	2
	4	5: 1	7	6: 1
A	12: 1	13	15: 1	14
	8	9: 1	11	10: 1
		D	

(نبدأ (1) →)

Example: Odd Function Implementation

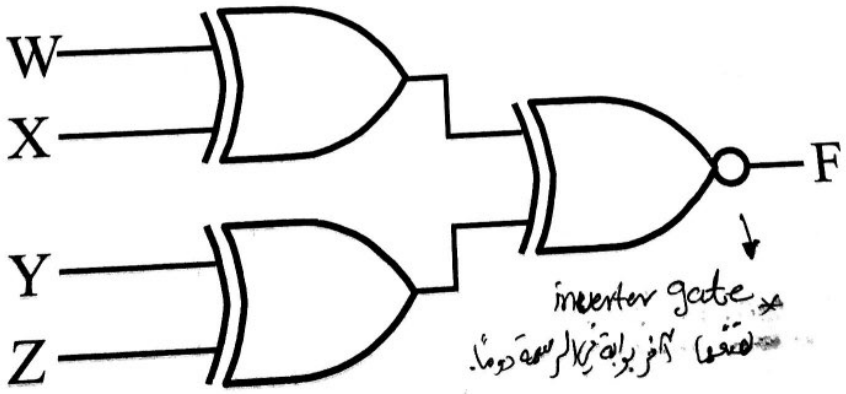
سؤال بجي بالامتحان مثله

- 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:



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:



Parity Generators and Checkers

- In Chapter 1, a parity bit added to n-bit code to produce an n+1 bit code:

* Example:- 0110 ← علينا إضافة (bit)
0101 ← زيادة ولكن تكون even parity مثلاً إذا
علينا المحافظة على عدد
ال (1) even .

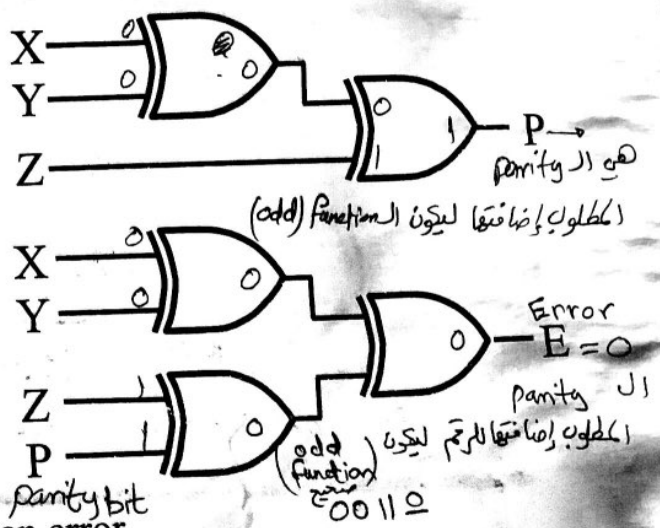
- Example: n = 3. **Generate even parity** code words of length four with **odd function (XOR)**:

بينما لو كانت odd parity يكون even function

- **Check even parity** code words of length four with **odd function (XOR)**:

- Operation: (X,Y,Z) = (0,0,1) gives (X,Y,Z,P) = (0,0,1,1) and E = 0. No error!
If Y changes from 0 to 1 between generator and checker, then E = 1 indicates an error

there is error!



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

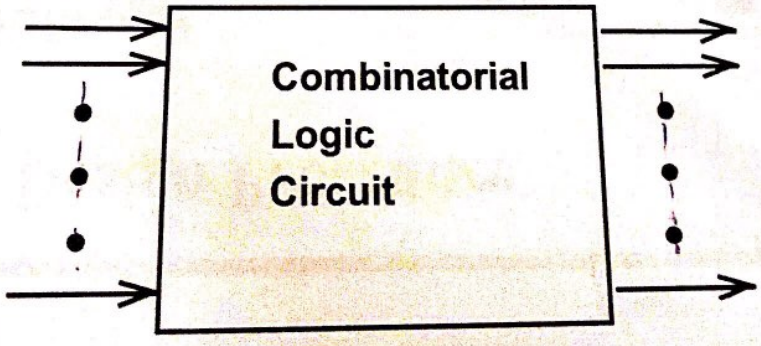
* نريد استخدام التالي في تصميم دوائر كهربائية تقوم بوظيفة معيّنة.

Combinational Circuits (doesn't have memory).

- 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***

* يكون هناك أكثر من input وبالتالي أكثر من output ولكن كل output يعتمد على ال input الذي يخصه فقط.

- A block diagram:



***m* Boolean Inputs** ***n* Boolean Outputs**

طريقة التصميم :-

Design Procedure

* خطوات لكي نقلل من احتمال الخطأ في تصميم الـ (circuit):

1. Specification (الوصف)

- Write a specification for the circuit if one is not already available. ***What does the circuit do? Including names or symbols for inputs and outputs***

2. Formulation (التمثيل)

- Derive a ***truth table*** or ***initial Boolean equations*** that define the required relationships between the inputs and outputs, if not in the specification

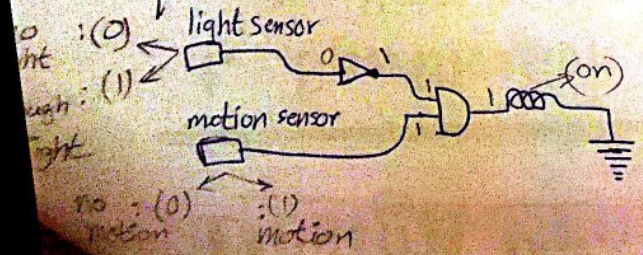
3. Optimization (الاختصار) / (التقليص)

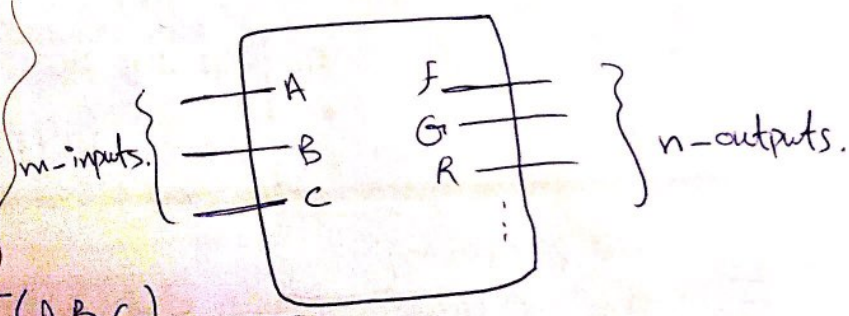
- Apply 2-level optimization using K-maps
- Draw a logic diagram for the resulting circuit using ANDs, ORs, and inverters

Design Procedure

4. **Technology Mapping** . (التمثيل بالرسم)
   - Map the logic diagram to the implementation technology selected
5. **Verification** (using lab logic).
   - Verify the correctness of the final design ***manually*** or using ***simulation*** → (محاكاة)

* quick Examples





$F(A,B,C,\ldots) = \_\_\_$

$L(A,B,\ldots)$ = light & $G(A,B,C,\ldots) = \_\_\_$
motion.

Design Example1

الوصف

- **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)

  الجملة التي توصف متى يعطي الـ function (١) ومتى (٥).

  * شرط متى يكون ناتج (F) هو (١)

  (معظم)

  الأغلبية، الأكثرية:

  - This is called ***majority function*** (i.e. majority of inputs must be 1 for the function to be 1)

- **Formulation:**

  نعمل الـ (truth table)

* specification the table: نعبّي الجدول: مكان الـ (١) عن طريق الشرط المعطى بالامتحان.

	X	Y	Z	F
m0	0	0	0	0
m1	0	0	1	0
m2	0	1	0	0
m3	0	1	1	1
m4	1	0	0	0
m5	1	0	1	1
m6	1	1	0	1
m7	1	1	1	1

عدد الأصفار أكبر من عدد الواحدات

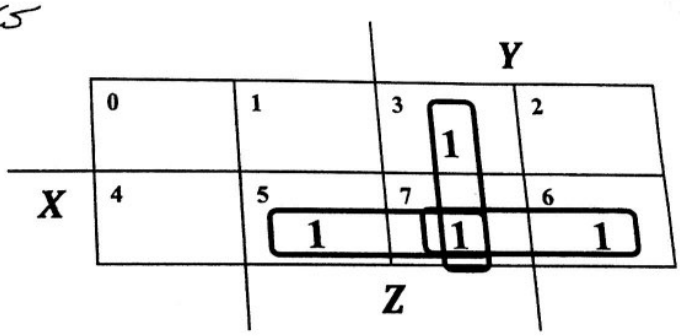
عدد الواحد أكبر من عدد الأصفار.

Design Example1 Cont.

(cost) لتقليل ال

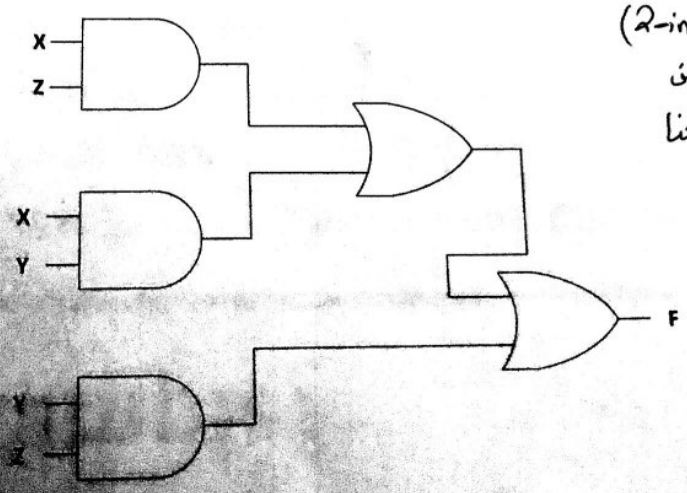
- **Optimization:** كتابة ال function بأبسط صورة using (k-maps)

$$F(X,Y,Z) = XY + XZ + YZ$$



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

* شرط فقط (2-inputs) لذلك نعمل أكثر من بوابة لكي يربط معنا ذلك.



Design Example2

* معناها :-
(2 binary numbers and each number consists of 2 bits.)

- **Specification:** Design a combinational circuit that compares ***2-bit*** Binary number ***(A, B)*** and produce two outputs (O_1, O_0), such that: *(one two bits output) decimal بال even (0)

الوصف — تقارن

$O_1O_0 = 00$	**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**

$A(A_1A_0)$		$B(B_1B_0)$	$O(O_1O_0)$
even 00	A=B	00	00
00	A<B	01	01
00		10	01
00		11	01
01	A>B	00	10
odd 01	A=B	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

odd بال decimal (1)

لكل bit output بال (kmap) نعمل له.

- **Formulation:** (truth table) تعبئة ال (decimal بال B أكبر من A)

* Note : نعبّي الجدول كأنهم (4-variable inputs) يعني لو دخل لأن A ← كمتغير لها (2bits) نعتبرها متغيرين.

(decimal بال A أكبر من B)

decimal بال odd (3) ← (0) ← (kmap) لل (0) ← (kmap)

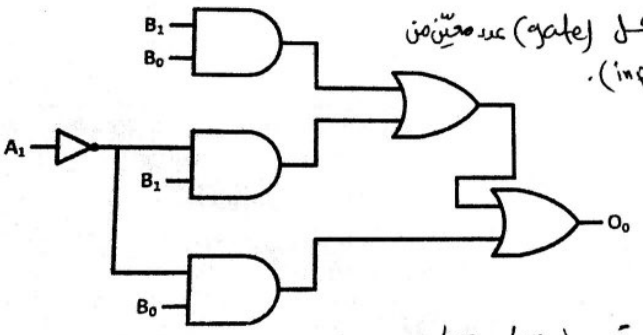
Design Example2 Cont.

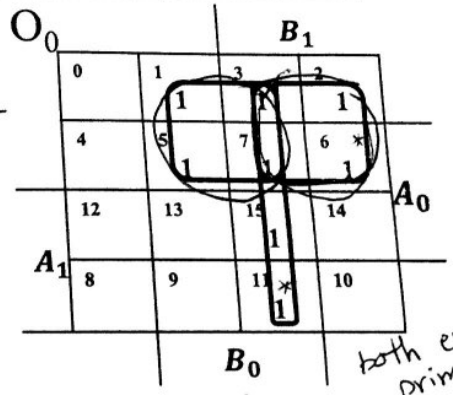
* لكل (output) نعمل (k-map) منفصلة . وحجمها (16 مربع) لأن $2^4=16$. 4: متغيرات على اعتبار A_1 ، A_0 ، B_1 ، B_0 (هي 4 متغيرات).

- **Optimization and Technology Mapping:**

$$O_0 = B_1B_0 + \overline{A_1}B_1 + \overline{A_1}B_0$$

* نأخذ عدد كبير من ال (gates) لأن لكل (gate) عدد معين من ال (inputs).

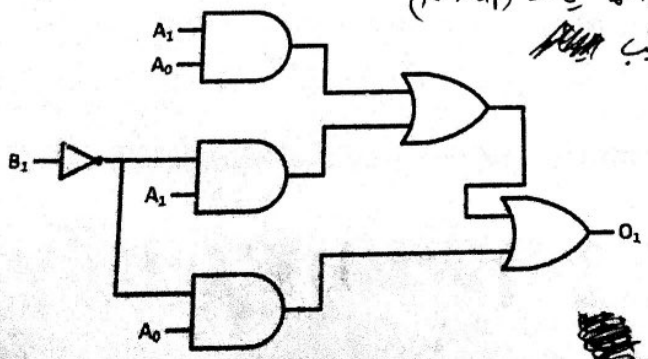


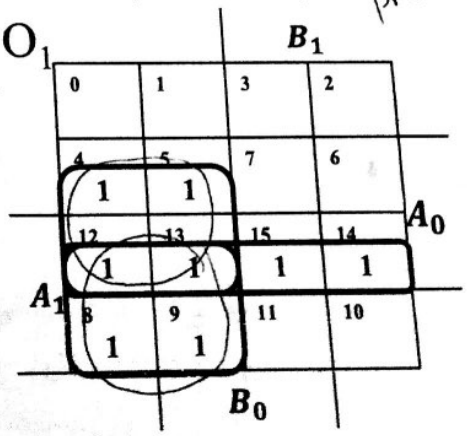


both essent
prime imi

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

* يتم أخذ قيم (O_1 / O_0) من الجدول ووضعها في ال (k-map) حسب الترتيب





Design Example3

1. Specification

- ***BCD to Excess-3 code converter*** (+3) (Excess 3) ← (BCD) الـ(circuit) تحوّل من
- Transforms BCD (4-bits) code (4-bits) 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 BCD : (0 → 9)
  نغطّي لغاية (9) فقط. حيث تكون في الـ(Excess3) هي (12)
- 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

ABCD	*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

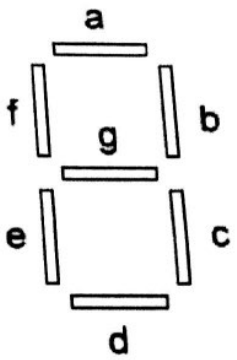
don't care

* لم يخبرني بالسؤال ماذا نفعل بال (illegal outputs) وهم من (10 ← 15) حيث أنهم غير موجودين بال (BCD) لأنه من (0 ← 9) فقط.

لأنه مكوّن من (٧ قطع).

Homework: BCD to 7-Segment

- **Specification:**
  - **Inputs:** ***(A, B, C, D)*** BCD code from 0000-to-1001
  - **Outputs:** ***(g, f, e, d, c, b, a)***



- **Formulation:**

ABCD	*gfedcba*
(0) *0000*	يشتغلو *0111111* يطفي
(1) *0001*	*0000110*
(2)	
1001	*1101111*
1010 ← لا يوجد (١٥) بال (BCD)	*0000000*
1111 ← لأنه مش موجود بال (BCD) (15) لذلك نضع (zeros) (don't care)	*0000000*

decimal

إذا كان ال output (١) : يضيء الضو.
بينما ال output (٥) : يطفي الضو.

- **Optimization:**
  - **How many K-maps?**

← نفس عدد المتغيرات في ال outputs
⊛ أي لكل output (k map) خاصة به.

Mapping to NAND gates

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

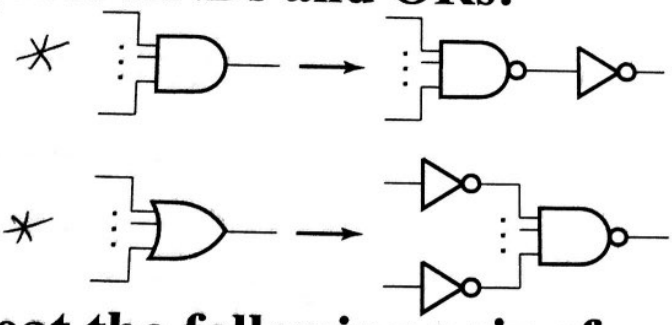
عندما يكون عندنا

(2-inverters) ورا بعض نلغيهم مع بعض

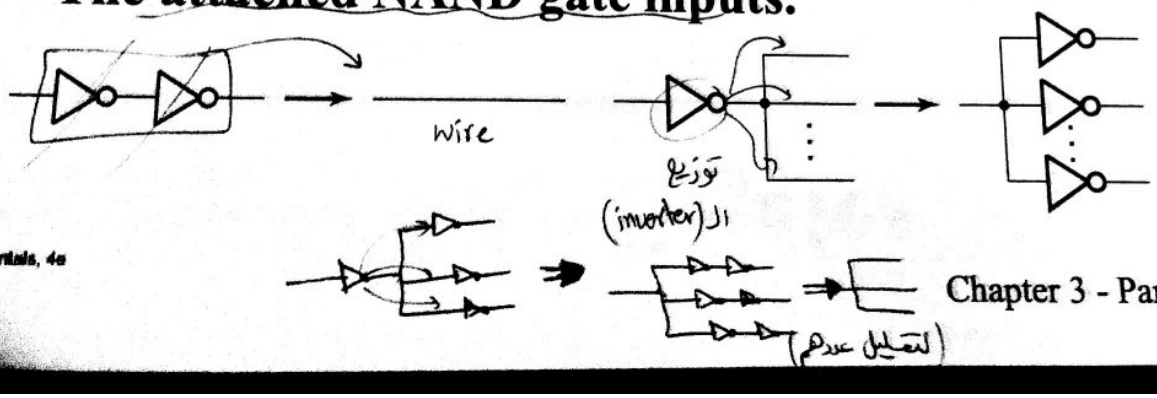
$\bar{X} \cdot X = \bar{X}$

NAND Mapping Algorithm

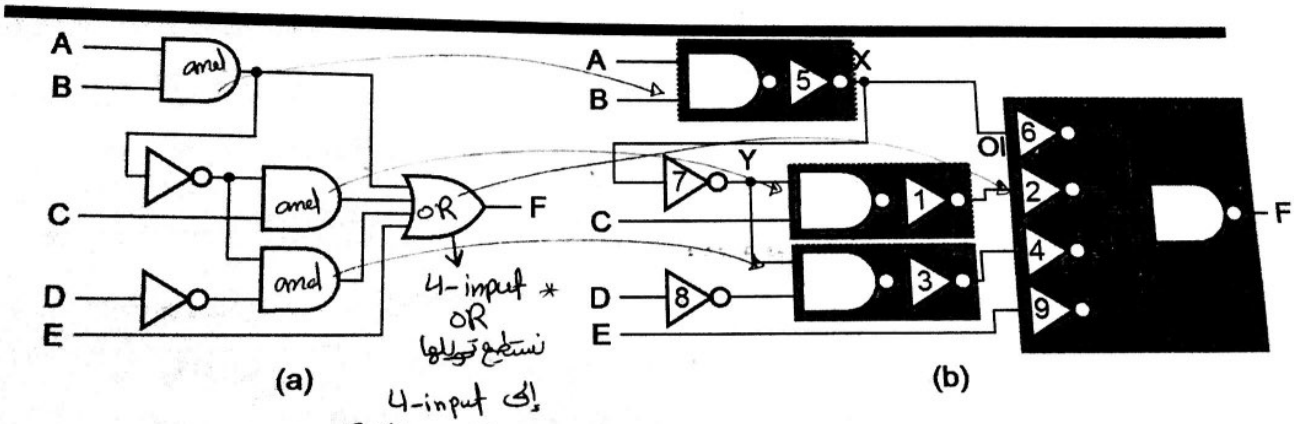
1. **Replace ANDs and ORs:**

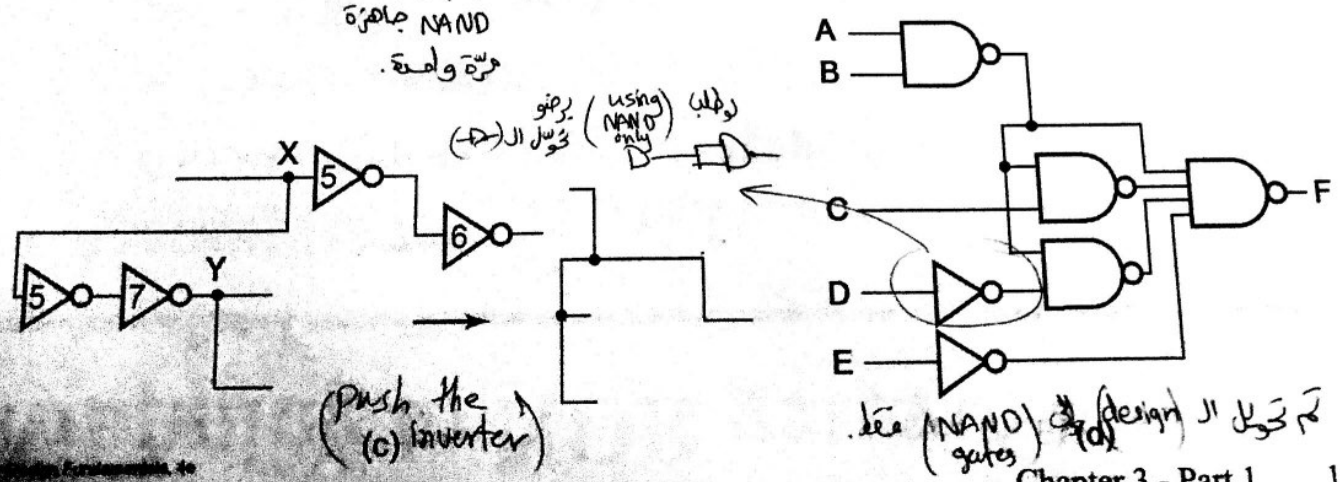


2. **Repeat the following pair of actions until there is at most one inverter between :**
   a. **A circuit input or driving NAND gate output, and**
   b. **The attached NAND gate inputs.**



NAND Mapping Example





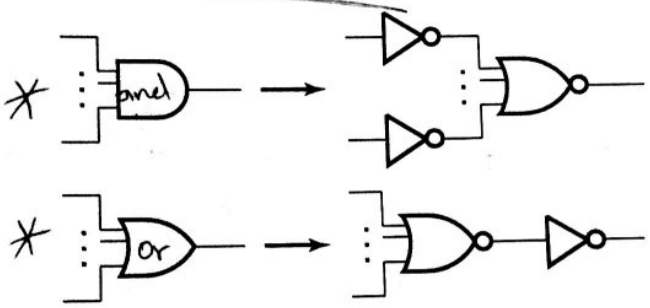
تم تحويل ال design (a) إلى (NAND gates) فقط.

* to reduce the number of inverters.

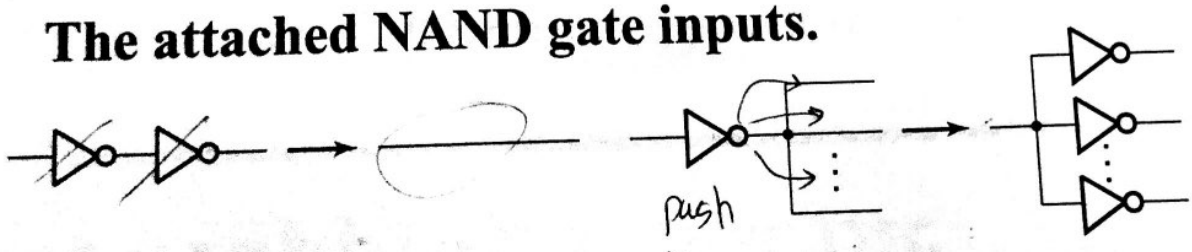
NOR Mapping Algorithm

1. **Replace ANDs and ORs:**

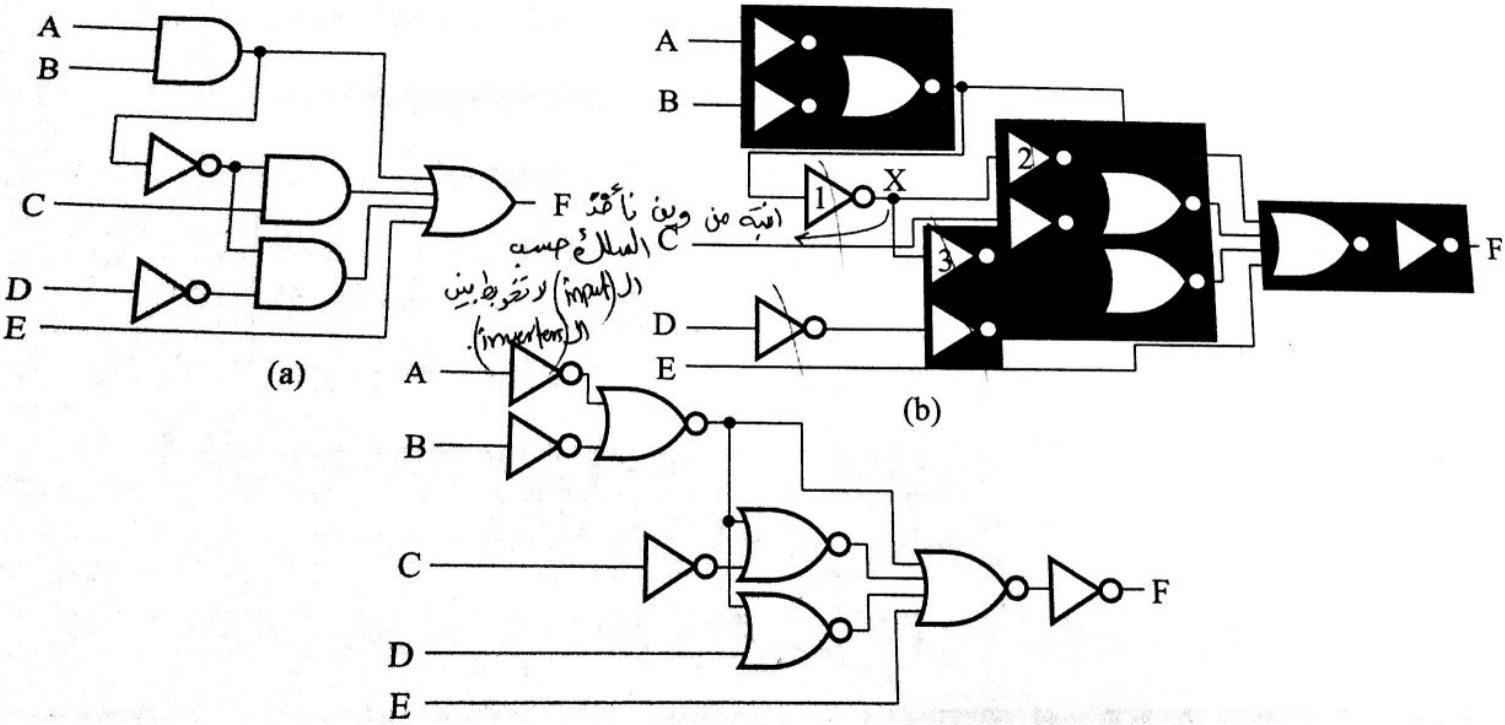
* نختلف عن ال (NAND) هنا:



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.**



NOR Mapping Example



انبه من وين ناخذ C المسلك حسب الـ(input) لا تخوط بين الـ(inverters).

نختصر الـ(inverters) المكررة وراء بعض لنقلل عددهم.

Overview

- **Part 2 – Combinational Logic** (without memory).
  input $\xrightarrow{\text{gives}}$ output.
  - Functions and functional blocks
  - Rudimentary logic functions
  - * Decoding using Decoders (blockes).
    - Implementing Combinational Functions with Decoders
  - * Encoding using Encoders (blockes)
  - Selecting using Multiplexers
    - Implementing Combinational Functions with Multiplexers

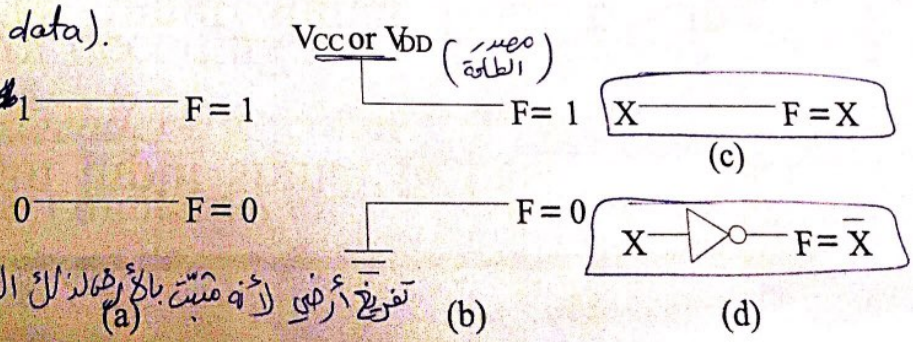
Basic Functions:

Rudimentary 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

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

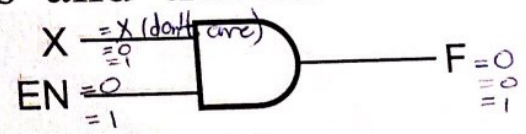
- ***Value fixing* : a, b** (output يأخذ وصف الـ (F) ويحلله ويخرجه)
- ***Transferring* : c** (same data). (نقل)
- ***Inverting* : d** (invert the data)
- ***Enabling* : next slide**



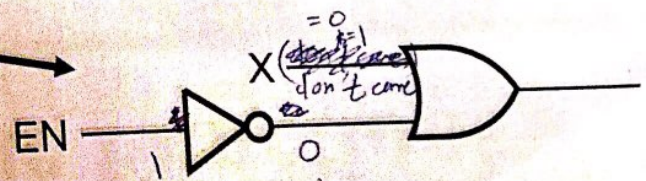
تفريغ أرضي لأنه مثبت بالأرض لذلك القيمة لـ (F) دائماً (0)

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*** using (AND)
- When disabled, ***1 output*** using (OR)



(a)



(b)

Decoding

- ***Decoding:*** the conversion (تحويل) of an ***n-bit*** input code to an ***m-bit*** output code with $\mathbf{n \le m \le 2^n}$ such that each valid code word produces a unique output code

  كل input له output مختلف.
  والعلاقة هي عدد inputs ≤ عدد الـ outputs ≤ $2^{\text{عدد الـ inputs}}$.

- Circuits that perform decoding are called ***decoders***

- Functional blocks for decoding are
  - called ***n-to-m line decoders,*** where $\mathbf{m \le 2^n}$, and $(m = 2^n)$ ← غالباً
  - generate $\mathbf{2^n}$ (or fewer) minterms for the *n* input variables

1-to-2 Line Decoder

(n) when $\boxed{n=1}$

- 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

⊛ مميزة الـ (decoder): أنّه فقط (output) واحد هو (active) في وقت معيّن.

- Only one output is active at a time

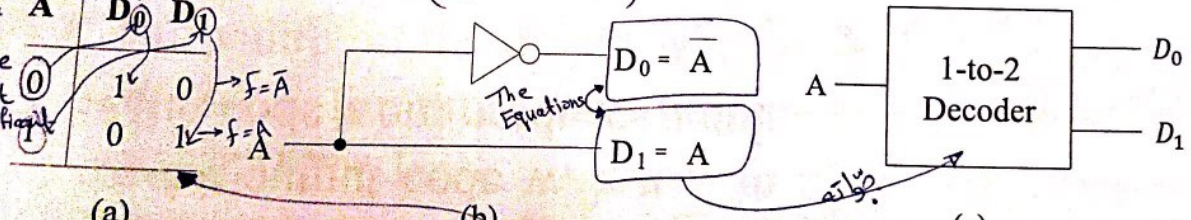
input → A; outputs → D_0, D_1

* ترتيب الـ truth table: least significant → most significant

A	D_0	D_1	
0	1	0	$\rightarrow f=\bar{A}$
1	0	1	$\rightarrow f=A$

(a)

Design for (1 to 2 Decoder):



(b) (c)

⊛ عندما يكون الـ input (0) يحوّلها من (binary) إلى (decimal) فيكون جوابها (0) لذلك نبحث عن الـ output الذي الـ (index) له نفس قيمة الـ input فيكون ناتجه (1) وكذلك الـ (1) لـ (input)

⊛ ميّزة الـ (decoder) فقط الـ (output) الذي يتشابه الـ (index) تاعه للـ (input) يكون (active) وجوابه (1) فقط والباقي (Zeros)

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

2-to-4 Line Decoder

(n) when $n = 2$

انتبه! مهم

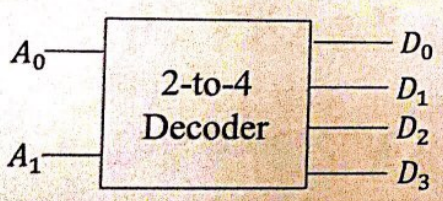
* ترتيب ال inputs (most → least)

* ترتيب ال output عشوائي لكن الأفضل (least → most)

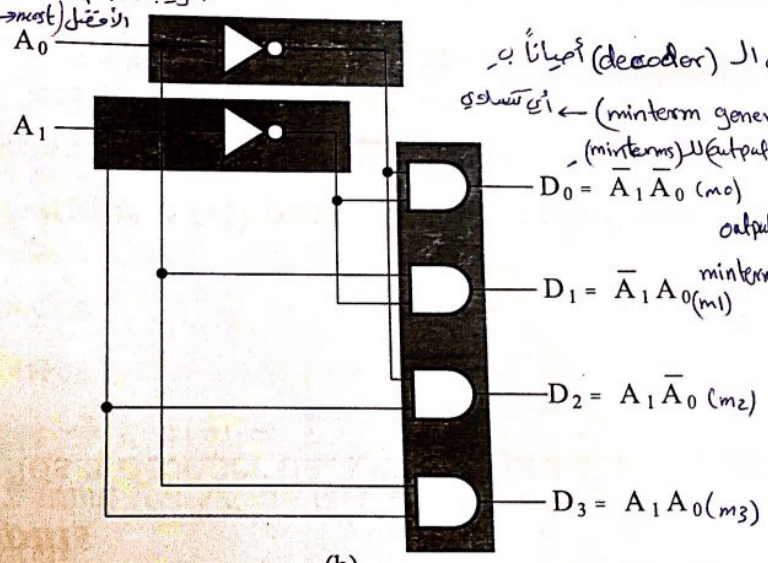
A_1	A_0	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

decimal: 0, 1, 2, 3

(a)



(c)



(b)

* يسمى ال (decoder) أحياناً بـ (minterm generator) ← أي تساوي ال (outputs) لل (minterms)

* نلاحظ أن ال output هم نفس ال minterms

- No more optimization is possible

which means (2 inverters)

- ***Note that the 2-to-4 line decoder is made up of two 1-to-2- line decoders and 4 AND gates***

2-input

Decoder Expansion

- General procedure given in book for any decoder with ***n inputs and 2^n outputs***
- This procedure builds a decoder backward from the outputs using

1. **Let k = n**

2. **We need 2^k 2-input AND gates driven as follows:**
   - **If k is even, drive the gates using two k/2-to-$2^{k/2}$ decoders**
   - **If k is odd, drive the gates using one (k+1)/2-to-$2^{(k+1)/2}$ decoder and one (k-1)/2-to-$2^{(k-1)/2}$ decoder**

3. **For each decoder resulting from step2, repeat step2 until k = 1. For k = 1, use 1-to-2 decoder** (loop) until we reach the simplest decoder.

Decoder Expansion - Example 1

- 3-to-8-line decoder

  * نبدأ رسم الحركة من اليمين (النهاية) لليسار (البداية).

  *• $k = n = 3$ = number of inputs.

  (Loop)

  *• We need 2^3 (8) 2-input AND gates driven as follows: 2^n = (عدد الـ 2-input and gates)

  *• k is odd, so split to:

  حتى نصل لأبسط (decoder) وهو عندما $\boxed{k=1}$

  (k) is odd (differs)

  - 2-to-4-line decoder — $\frac{k+1}{2}$ to $2^{\frac{k+1}{2}}$ decoder
  - 1-to-2-line decoder — $\frac{k-1}{2}$ to $2^{\frac{k-1}{2}}$ decoder

  *• 2-to-4-line decoder → $k = n = 2$

  (k) is even (same)

  - We need 2^2 (4) 2-input AND gates driven as follows: — $\frac{k}{2}$ to $2^{\frac{k}{2}}$ decoder
  - k is even, so split to: — $\frac{k}{2}$ to $2^{\frac{k}{2}}$ decoder
    - Two 1-to-2-line decoder

- See next slide for result

Decoder Expansion - Example 1

- $GN = 8 \times 2 + 4 \times 2 + 3$ (AND gates. / AND gates / عدد ال inverters)
- $GN = 27$ (كل وحدة 2-inputs / كل وحدة 2-inputs)
- **Straight forward design has the same GN cost**

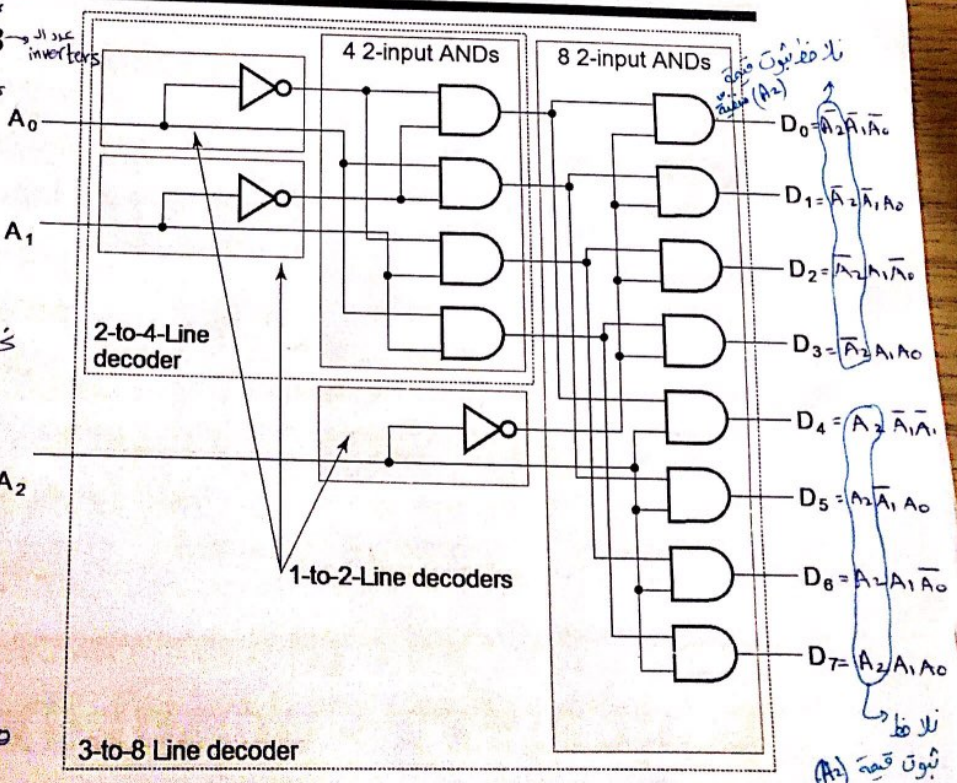
* من ناحية (cost) متساويات ولكن من ناحية (delay) يختلفوا.

* لكي نسهّل الرسم نبدأ من اليمين ثم نحدد شكل ال output لكل (and gate) من حيث D_0 ← هي $\bar{A}_2\bar{A}_1\bar{A}_0$ وهكذا لكل المخرجات

* يجوز تبديل (A_2 بـ A_0) أو غيرها أي أنّ أي ترتيب لا يهمّ والمهم فقط أن ترتيب ال outputs يكون عليه صح.

$0 \equiv \bar{A}_2\bar{A}_1\bar{A}_0$

$1 \equiv \bar{A}_2\bar{A}_1A_0$



Decoder Expansion - Example 2

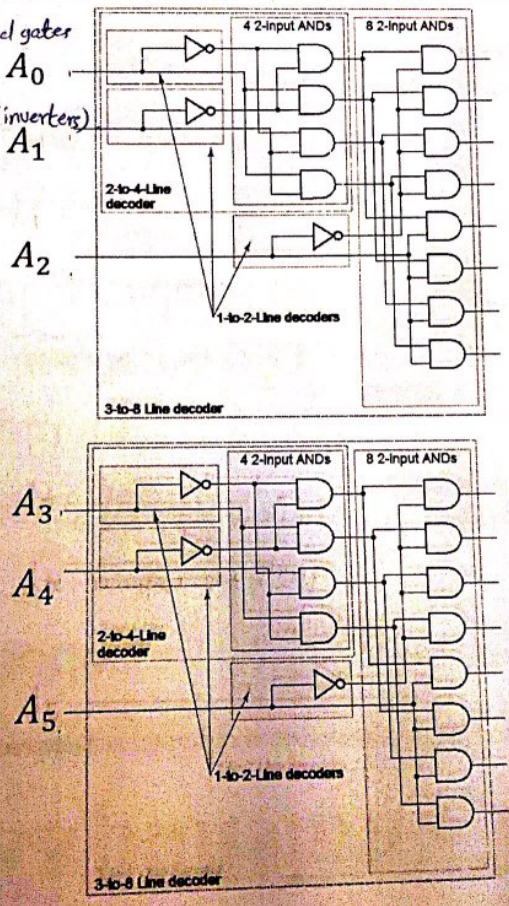
- 6-to-64-line decoder
  - $k = n = 6$ → The number of inputs.
  - We need 2^6(64) 2-input AND gates driven as follows: (2^n = number of AND gates).
  - k is even, so split to:
    - Two 3-to-8-line decoders
  - Each 3-to-8-line decoder is designed as shown in Example 1

or continue the loop!:

$k=n=3$ → odd → (1 to 2) decoder.
→ (2 to 4) decoder. → continue the loop! → $n=k=2$ even → 1 to 2 decoder.
→ 1 to 2 decoder.

Decoder Expansion - Example 2

- $GN = 64 \times 2 + 16 \times 2 + 8 \times 2 + 6$ (2^6 and gates. 2^2 and gates 2^4 and gates. (inverters))
- $GN = 182$
- **Straight forward design has GN cost of 390**



$D_0 = \overline{A_5}\,\overline{A_4}\,\overline{A_3}\,\overline{A_2}\,\overline{A_1}\,\overline{A_0}$

$D_1 = \overline{A_5}\,\overline{A_4}\,\overline{A_3}\,\overline{A_2}\,\overline{A_1}A_0$

$D_{62} = A_5A_4A_3A_2A_1\overline{A_0}$

$D_{63} = A_5A_4A_3A_2A_1A_0$

Decoder Expansion - Example 3

- **7-to-128-line decoder**
  - $k = n = 7$
  - We need 2^7(128) 2-input AND gates driven as follows:
  - k is odd, so split to:
    - 4-to-16-line decoder
    - 3-to-8-line decoder
  - 4-to-16-line decoder
    - $k = n = 4$
    - We need 2^4(16) 2-input AND gates driven as follows:
    - k is even, so split to:
      - Two 2-to-4-line decoders
  - Complete using known 3-8 and 2-to-4 line decoders
- $GN = \underbrace{128}_{\text{2-input and gate}} \times 2 + \underbrace{16}_{\text{2-input and gates}} \times 2 + 8 \times 2 + 12 \times 2 + \underbrace{7}_{\text{inputs}} = 335$
- Compare to straight forward design with GN cost of 903

using Enables.

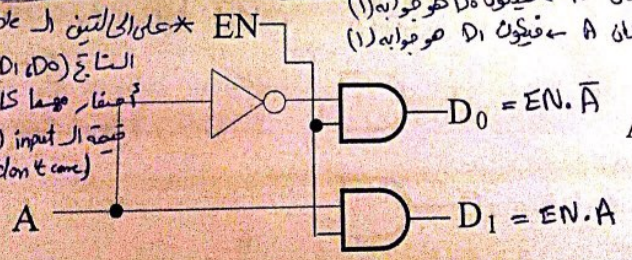
Building Larger Decoders

- **Method_1:** Decoder Expansion (الطريقة السابقة)
- **Method_2:** Using Small ***Decoders with Enable input*** (الطريقة الحالية):
- Example: * 1-to-2 line decoder with enable *
  - In general, attach ***m-enabling*** circuits to the outputs
  - See truth table below for function
    - Note use of X's to denote both 0 and 1
    - Combination containing two X's represent two binary combinations
- Alternatively, can be viewed as distributing value of signal EN to 1 of 2 outputs
  - In this case, it is called a *Demultiplexer* (decoder + Enable)

* متى ما كان ال Enable ← (0) يكون D_0 و D_1 ← صفر.
* متى ما كان ال Enable ← (1) يكون حسب ال input لل D_0 و D_1.
* على الحالتين ال Enable لازم يكون (1) حيث إذا كان (0) يكون الناتج (D_1 و D_0) أصفار مهما كانت قيمة ال input (X) (don't care).
← حيث إذا كان $\bar{A}$ ← فيكون D_0 هو جوابه (1)
وإذا كان A ← فيكون D_1 هو جوابه (1)

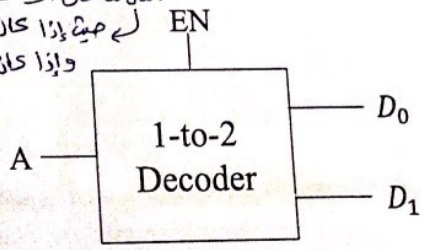
EN	A	D_0	D_1
0	X	0	0
1	0	1	0
1	1	0	1

(a)



(b)

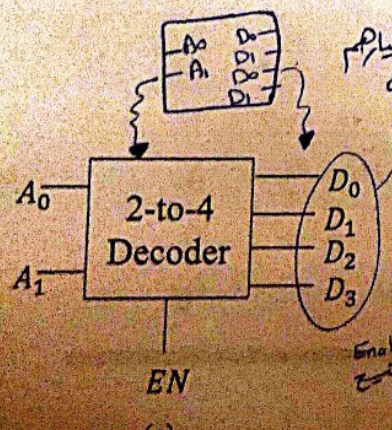
نفس التصميم القديم ولكن تضيف ال Enable على ال D_0 و D_1 مرتبط معه بـ (and gate)



(c)

2-to-4 Line Decoder with Enable

- 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*



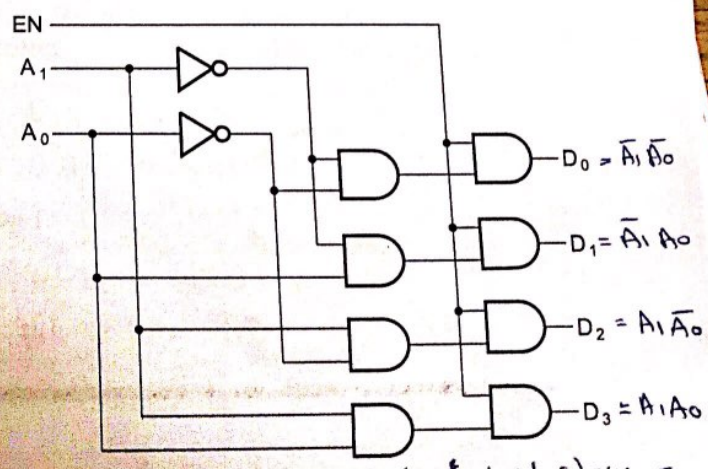
(c)

* الأفضل اختيارهم بالترتيب

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

Enable منسوخ

(a)



(b)

* تم إضافة (4 بوابات and أخرى) لأن كل بوابة منفصل (2 inputs)

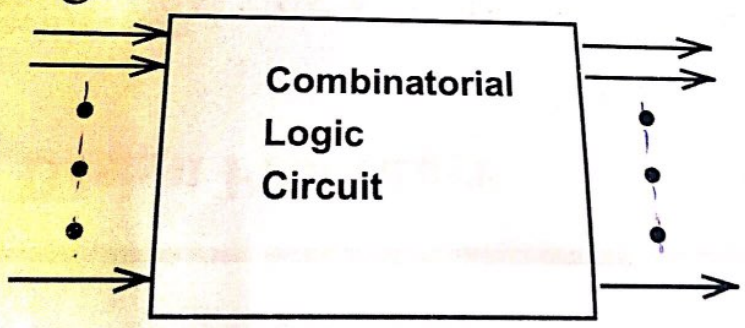
* غالباً ال (Enable) يكون هو ال (most significant digit)

Combinational Circuits (doesn't have memory).

- 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***

(Functions) المقصود ال

- A block diagram:



m Boolean Inputs

n Boolean Outputs

* يكون هناك أكثر من input وبالتالي أكثر من output ولكن كل output يعتمد على ال input الذي يخصه فقط.

طريقة تصميم

هذا التحليل نتكلم مناطقة الظاهر

طريقة تصميم

Design Procedure

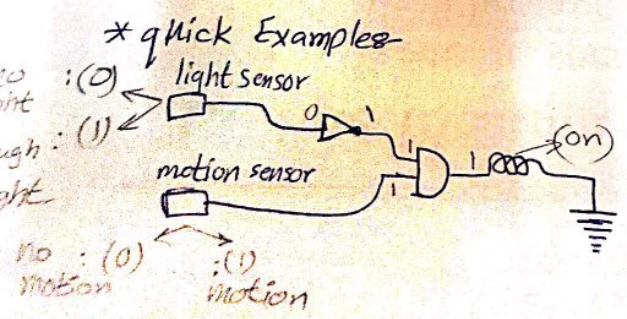
خطوات لكي نقلل من احتمال الخطأ في تصميم الـ (circuit):

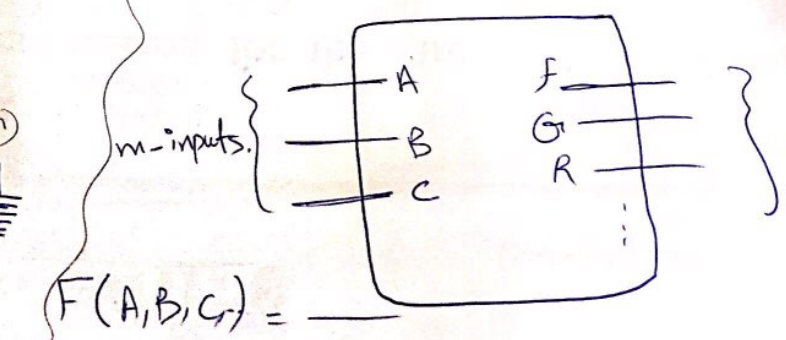
1. **Specification** (الوصف)
   - Write a specification for the circuit if one is not already available. ***What does the circuit do? Including names or symbols for inputs and outputs***
2. **Formulation** (التمثيل)
   - Derive a ***truth table*** or ***initial Boolean equations*** that define the required relationships between the inputs and outputs, if not in the specification
3. **Optimization** (الاختصار) / (التقليص)
   - Apply 2-level optimization using K-maps
   - Draw a logic diagram for the resulting circuit using ANDs, ORs, and inverters (basic gates)

Design Procedure

4. **Technology Mapping**. (التمثيل بالرسم)
   - Map the logic diagram to the implementation technology selected
5. **Verification** (using lab logic).
   - Verify the correctness of the final design ***manually*** or using ***simulation*** → (محاكاة)

*quick Examples





$F(A,B,C) =$ _____

Design Example1

Design Example1

- **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)

  الوصف

  الجملة التي توصف متى يعطي الـ function (1) ومتى (0).

  * شرط متى يكون ناتج (F) هو (1) (معظم)

  - This is called ***majority function*** (i.e. majority of inputs must be 1 for the function to be 1)

  الأغلبية، الأكثرية

- **Formulation:**

  نعمل الـ (truth table)

	X	Y	Z	F
m_0	0	0	0	0
m_1	0	0	1	0
m_2	0	1	0	0
m_3	0	1	1	1
m_4	1	0	0	0
m_5	1	0	1	1
m_6	1	1	0	1
m_7	1	1	1	1

عدد الأصفار أكبر من عدد الواحدات

عدد الواحد أكبر من عدد الأصفار.

* specification the table: نعبّي الجدول: مكان الـ (1) عن طريق الشرط المعطى بالامتحان.

Design Example1 Cont.

(cost) ← لتقليل ال

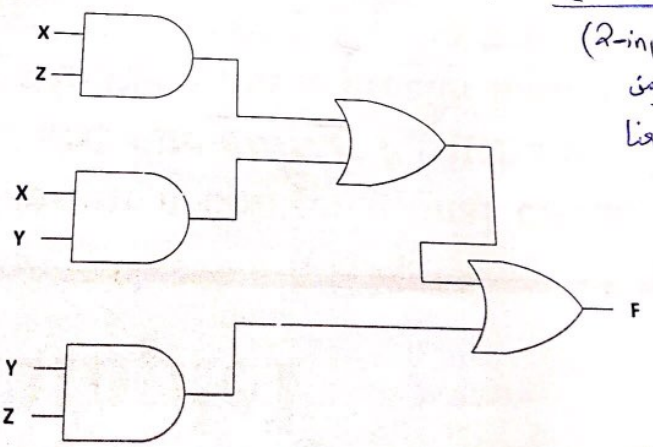
- **Optimization:** كتابة ال function بأبسط صورة using (k-maps)

$$F(X,Y,Z) = XY + XZ + YZ$$

as a sum of minterms: $\sum m(3,5,6,7)$.

				Y
	0	1	3	2
			1	
X	4	5	7	6
		1	1	1
			Z	

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



* شرط فقط (2-inputs) لذلك نعمل أكثر من بوابة لكي يربط معنا ذلك.

Design Example2

* معناها:- (2 binary numbers and each number consists of 2 bits.)

- **Specification:** Design a combinational circuit that compares **2-bit** Binary number **(A, B)** and produce two outputs **(O_1, O_0)**, such that: *(one two bits output) decimal بال even (0)

الوصف — تقارن

$O_1O_0 = 00$	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

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

A=B, A<B, A>B, A=B

odd بال decimal (1)

لكل bit output بال (kmap) نعمل له.

- **Formulation:** (truth table) تعبئة ال (decimal بال B أكبر من A)

* Note: نعبّي الجدول كأنهم (4-variable inputs) يعني لا دخل لأن A ← كمتغير لها (2 bits) نعتبرها متغيرين.

(decimal بال A أكبر من B)

decimal بال odd (B) ← (0,1) ← (kmap) لل (0,1) ← (kmap) لل

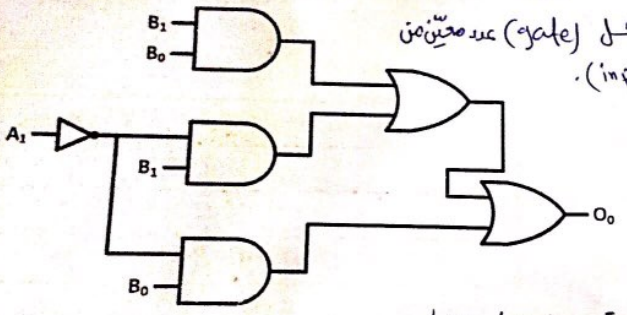
Design Example2 Cont.

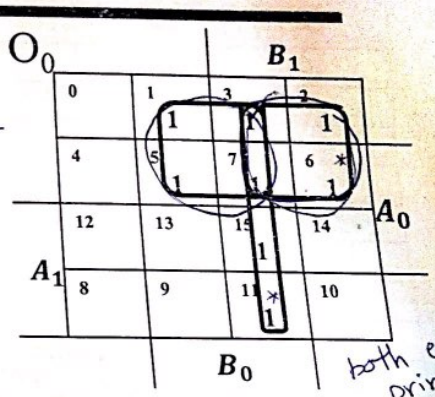
* لكل (output) نعمل (k-map) منفصلة . وحجمها (16 مربع) لأن $2^4=16$. الـ متغيرات على اعتبار A_1 ، A_0 ، B_1 ، B_0 (هي 4 متغيرات).

- **Optimization and Technology Mapping:**

$$O_0 = B_1B_0 + \overline{A_1}B_1 + \overline{A_1}B_0$$

* نأخذ عدد كبير من الـ (gates) لأن لكل (gate) عدد معين من الـ (inputs).

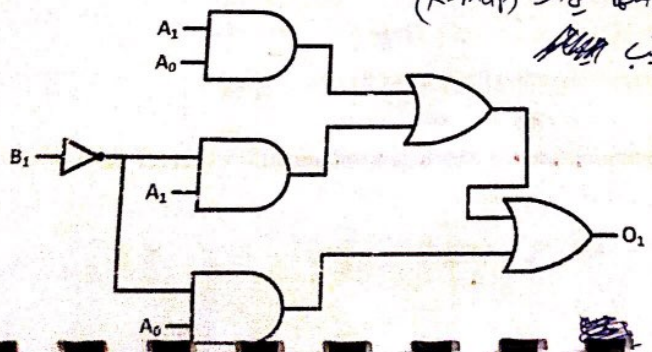


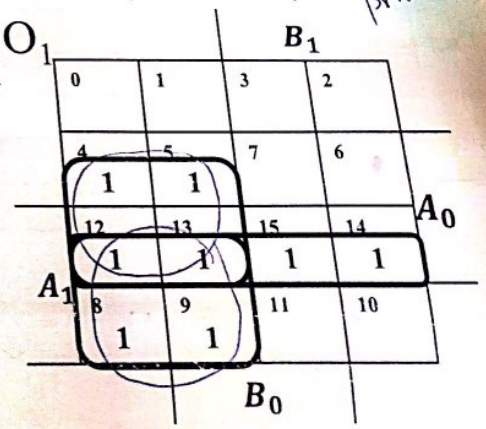


both essential prime implicants

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

* يتم أخذ قيم (O_1 / O_0) من الجدول ووضعها في الـ (K-map) حسب الترتيب





Design Example3

Design Example3

1. Specification

- ***BCD to Excess-3 code converter*** (+3) (Excess 3) ← (BCD) تحويل من (circuit) عمل
- Transforms BCD (4-bits) code for the decimal digits to Excess-3 (4-bits) code for the decimal digits
- BCD code words for digits 0 through 9: 4-bit patterns 0000 to 1001, respectively BCD : (0 → 9)
  نقطتين لغاية (9) فقط. حيث تكون في الـ(Excess3) هي (12)
- 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

ABCD	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

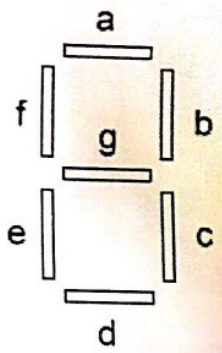
don't care.

* لم يخبرني بالسؤال ماذا نفعل بال (illegal outputs) وهم من (10 ← 15) حيث أنهم غير موجودين بال (BCD) لأنه من (0 ← 9) فقط.

Design Example3 Cont.

Homework: BCD to 7-Segment

- **Specification:**
  - **Inputs:** ***(A, B, C, D)*** BCD code from 0000-to-1001
  - **Outputs:** ***(g, f, e, d, c, b, a)***



- **Formulation:**

ABCD	gfedcba
(0) 0000	0111111 يطفي (يشتغلو)
(1) 0001	0000110
(2) ⋮	⋮
1001	1100111
1010 ← لا يوجد (10) بال (BCD)	0000000
⋮	⋮
1111 ← لأنه مش موجود بال (BCD) (15) لذلك نضع (zeros) (don't care)	0000000

decimal

* إذا كان ال output (1) : يضيء الضو.
بينما ال output (0) : يطفي الضو.

- **Optimization:**
  - **How many K-maps?** ← نفس عدد المتغيرات في ال outputs

⊗ أي لكل output خاصة به (kmap).

Technology Mapping

- **Mapping Procedures**
  - **To NAND gates**
  - **To NOR gates**

* Better to make a circuit only with one gate type (using one type universal gate)

* NAND Gate:

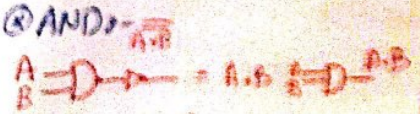
* NOR Gate:

@ NOT:

$\overline{A \cdot A} = \overline{A}$

@ AND:

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



@ OR:

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

@ NOT:

$\overline{B + B} = \overline{B}$

@ OR:

$\overline{\overline{A + B}}$

@ AND:

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

Mapping to NAND gates

Mapping to NAND gates

- **Assumptions:**
  - **Gate loading and delay are ignored**
  - **Cell library contains an inverter and *n*-input NAND gates, *n* = 2, 3, …**
  - **An AND, OR, inverter schematic for the circuit is available** تخطيطي

- **The mapping is accomplished by:** يتحقق.
  - **Replacing AND and OR symbols,** → to NAND
  - **Pushing inverters through circuit fan-out points, and**
  - **Canceling inverter pairs** → to NAND.

عندما يكون عندنا

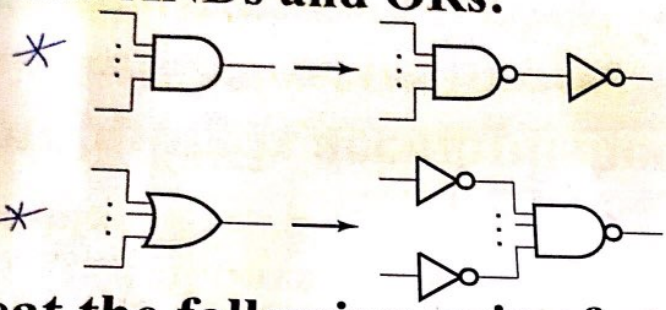
وراء بعض نلغيهم مع (2-inverters)

بعض

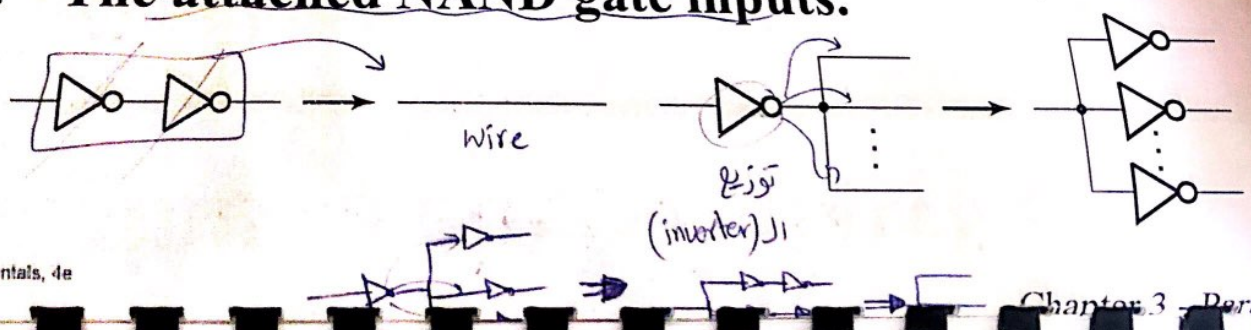
$\bar{X} \cdot X = \bar{X}$

NAND Mapping Algorithm

1. **Replace ANDs and ORs:**

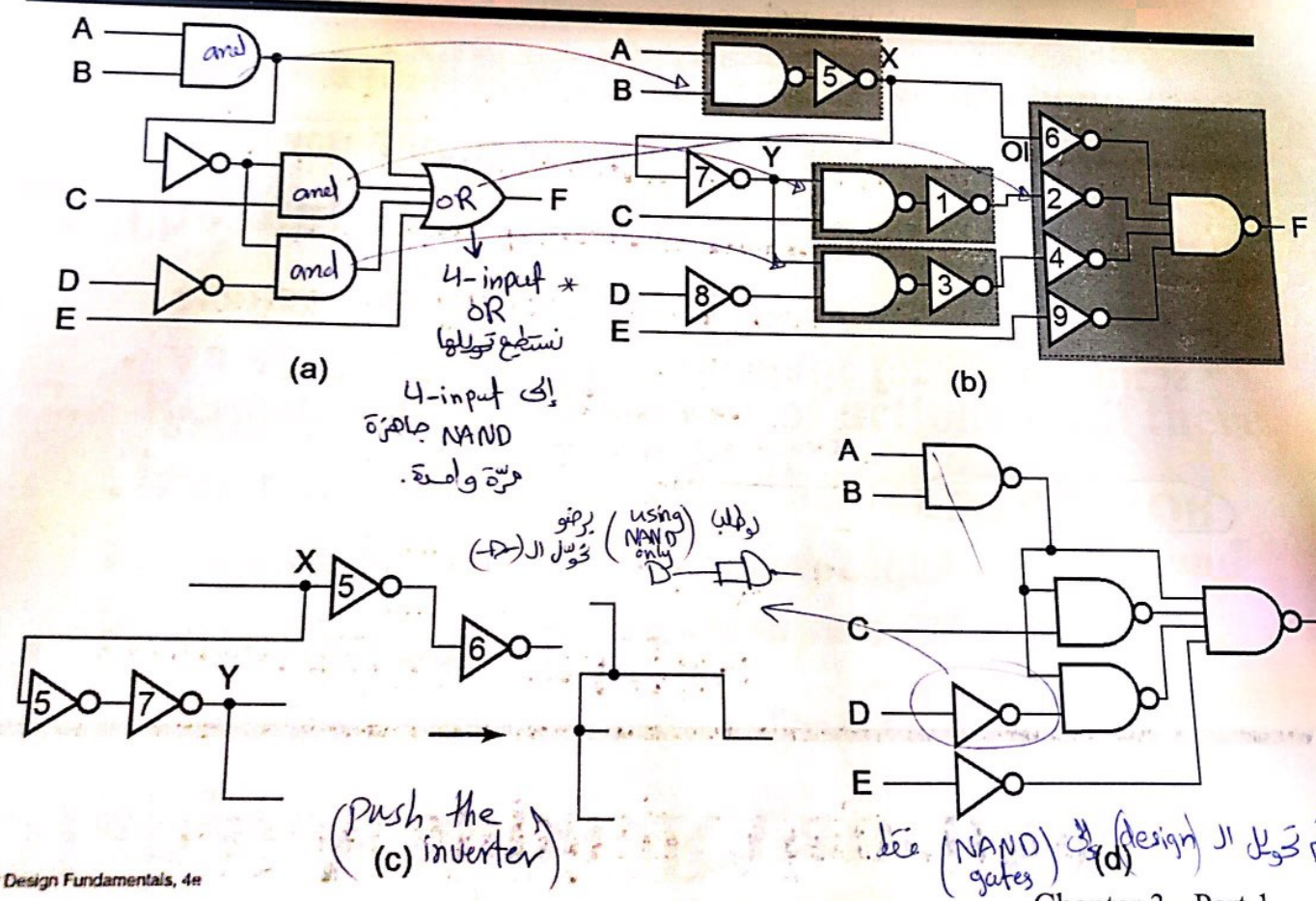


2. **Repeat the following pair of actions until there is at most one inverter between :**
   a. **A circuit input or driving NAND gate output, and**
   b. **The attached NAND gate inputs.**



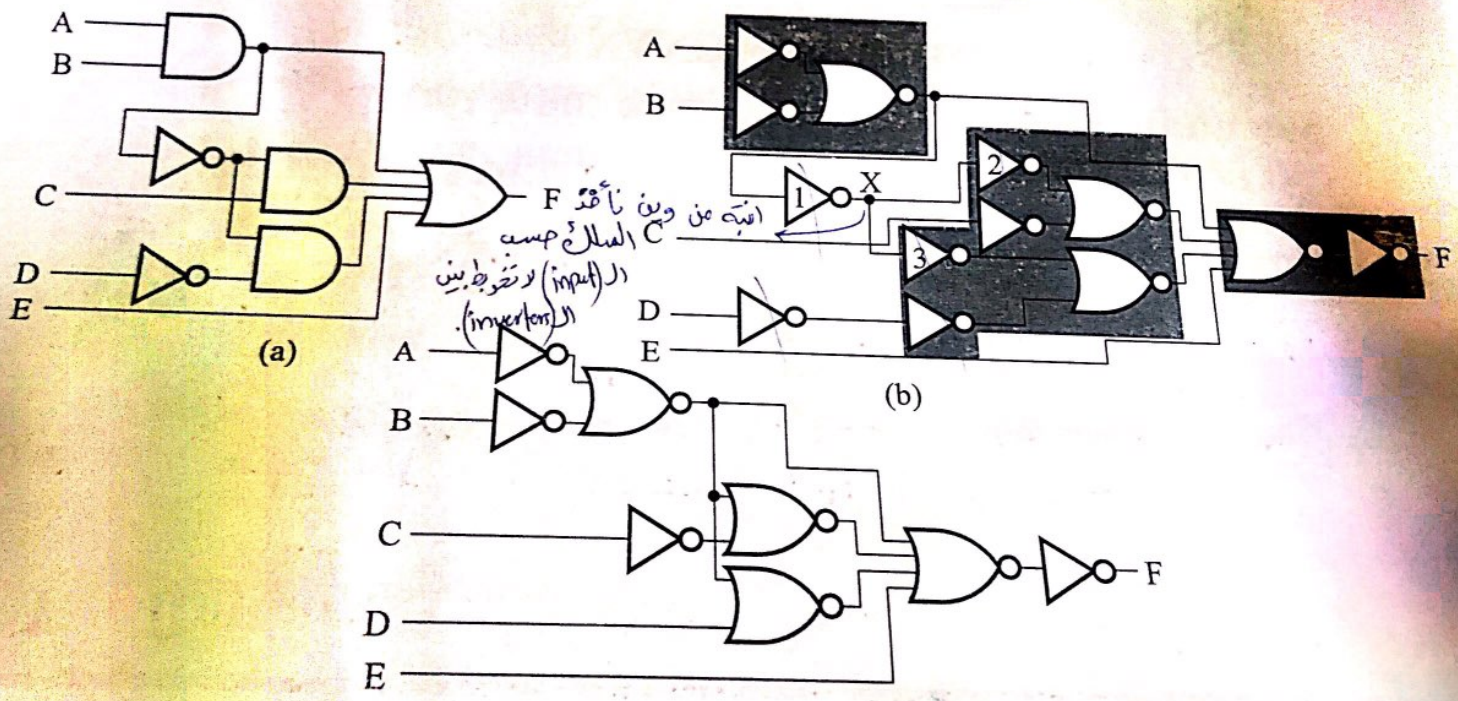
NAND Mapping Example

NAND Mapping Example



*to reduce the number of inverters.

NOR Mapping Example



انتبه من وين نأخذ C المسألة حسب الـ(input) لا تخلط بين الـ(inverter).

نختصر الـ(inverters) المكررة وراء بعض لنقلل عددهم

Rudimentary Logic Functions

Basic 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

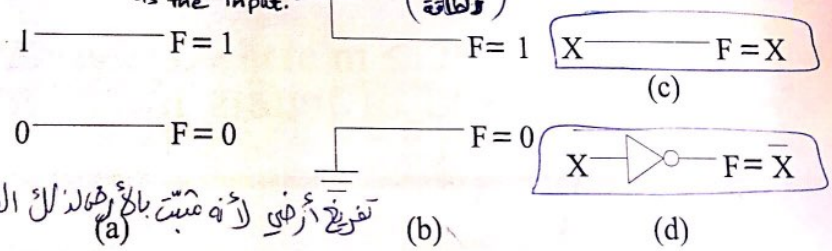
Value fixing. transferring. inverting.

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

- ***Value fixing*** **: a, b** (output) يأخذ وصف الـ (F) ويحلله ويخرجه
- ***Transferring*** **: c** (same data). → the output is as same Vcc or VDD (logic high) (مصدر الطاقة)
- ***Inverting*** **: d** (invert the data)
- ***Enabling*** **: next slide**

VDD ⇒ 1
⏚ ⇒ 0

يأخذ قيمة ثابتة ومعيّنة حسب المفهوم (الحالة).



(a) (b) (c) (d)

تفريغ أرضي لأنه مثبت بالأرضي لذلك القيمة لـ (F) دائماً (0)

Enabling Function

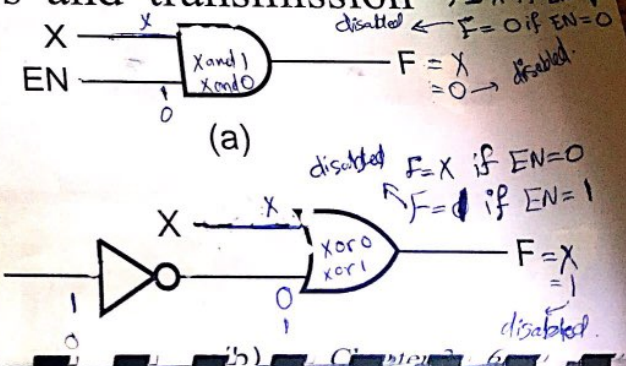
- ***Enabling*** permits an input signal to pass through to an output
- ***Disabling*** blocks an input signal from passing through to an output, replacing it with a fixed value
- The value on the output when it is disable can be ***Hi-Z*** (as for three-state buffers and transmission gates), 0 , or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

*when Enable = 0. then the function have a fixed value = 0

* when Enable = 1 then the function = X. wether X=0 or X=1

using (AND)

using (OR)



Decoding

8 combenational logic block.

يختلف عن سابقه (٣) بـ (G) (١)
بأن ال inputs هي (m) و ال

Decoding

8 combenational logic block.

يختلف عن شابتر (٣) بار (١) بأن ال inputs هي (m) و ال outputs هي (n) فقط.

- ***Decoding:*** the conversion of an ***n-bit*** input code to an ***m-bit*** output code with $\mathbf{n \le m \le 2^n}$ such that each valid code word produces a unique output code

تحويل

كل input له output مختلف.

والعلاقة هي عدد ال inputs ≤ عدد ال outputs ≤ $2^{\text{عدد ال inputs}}$.

- Circuits that perform decoding are called ***decoders***

only one of the outputs is one (1) and the rest are zero's (0)

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

غالباً ← $(m = 2^n)$ وأكبر قيمة له هي (2^n) كعدد outputs.

* Decoder :- n-bit input $\xrightarrow{\text{convert}}$ m-bit output.

* Relation : $n \le m \le 2^n$ ، $2^n \ge$ عدد ال outputs غالباً متساوي $\ge$ عدد ال inputs

يجوز ولكن يمكن أقل بينما أكبر مستحيل

1-to-2 Line Decoder when $n=1$

(n)

* A: input : Address
* D : output : Data.

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

قيمة (A) بال (decimal) هي ال (index) لل output الذي قيمته (1).

(*) مميزة ال (decoder): أنه فقط (output) واحد هو (active) في وقت معين.

(*) vimp :

- Only one output is active at a time

Design for (1 to 2 Decoder):

* ترتيب ال truth table

A	D_0	D_1	
0	1	0	$\to f=\bar{A}$
1	0	1	$\to f=A$

least significant → most significant

inputs → outputs

(a)

[inverting rudimentary function].

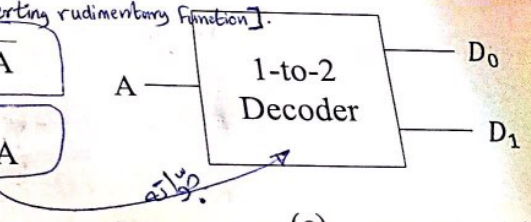
The Equations

$D_0 = \bar{A}$

$D_1 = A$

[transfaring rudimentary function].

(b)



(c)

(*) عندما يكون ال input (0) بحوّلها من (binary) إلى (decimal) فيكون جوابها (0) لذلك نبحث عن ال output الذي ال (index) له نفس قيمة ال input فيكون ناتجه (1) وكل ال (1) ك (input) جوابه

(*) ميزة ال (decoder) فقط ال (output) الذي يتشابه ال (index) تاعه لل (input) يكون (active) وجوابه (1) فقط والباقي (Zeros)

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

2-to-4 Line Decoder

when $n = 2$

(n)

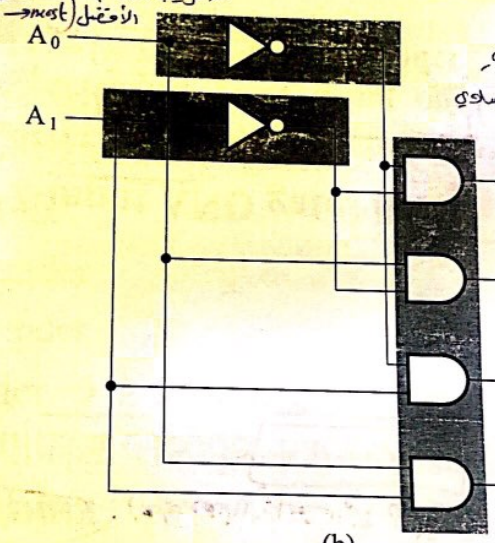
انتبه! * ترتيب ال inputs (most → least)

* ترتيب ال output هو الي اكن الأقفل (least → most)

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

decimal (0)10 (1)10 (2)10 (3)10

(a)



* يسمى ال (decoder) أحياناً بـ (minterm generator) أي تساوي ال (outputs) لل (minterms)

$D_0 = \bar{A}_1 \bar{A}_0$ (m0)

$D_1 = \bar{A}_1 A_0$ (m1)

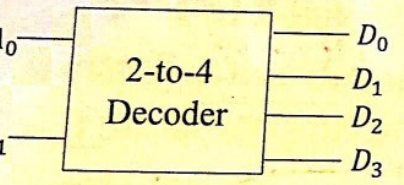
$D_2 = A_1 \bar{A}_0$ (m2)

$D_3 = A_1 A_0$ (m3)

* نلاحظ أن ال output هم نفس ال minterms

(b)

* كيف نمثل ال Truth table لل decoder بأنه وحده يكون (1) في كل سطر مرة عند (D) معينة تختلف عن الأخرى.



(c)

- No more optimization is possible

which means (2 inverters)

- ***Note that the 2-to-4 line decoder is made up of two 1-to-2- line decoders and 4 AND gates***

2-input

* موشرط عدد ال outputs اجباري أن يكون (2^n) يمكن أقل من ذلك أي $\boxed{\text{output} \leq 2^n}$

Decoder Expansion (less cost).

- General procedure given in book for any decoder with ***n inputs and 2^n outputs*** نبدأ من آخر الشكل (الناتج للأمام)
- This procedure builds a decoder backward from the outputs using

1. **Let k = n**

2. **We need 2^k 2-input AND gates driven as follows:**
   - **If k is even, drive the gates using two k/2-to-$2^{k/2}$ decoders**
   - **If k is odd, drive the gates using one (k+1)/2-to-$2^{(k+1)/2}$ decoder and one (k-1)/2-to-$2^{(k-1)/2}$ decoder**

3. **For each decoder resulting from step2, repeat step2 until k = 1. For k = 1, use 1-to-2 decoder** (loop) until we reach the simplest de

Decoder Expansion - Example 1

simplest decoder.

Decoder Expansion - Example 1

- 3-to-8-line decoder

* نبدأ رسم الركمة من اليمين (النهاية) لليسار (البداية).

  - * $k = n = 3$ = number of inputs.
  - * We need 2^3(8) 2-input AND gates driven as follows: 2^n = (عدد الـ 2-input and gates)
  - * k is odd, so split to:
    - 2-to-4-line decoder
    - 1-to-2-line decoder

(k) is odd (differs)

$\frac{k+1}{2}$ to $2^{\frac{k+1}{2}}$ decoder

$\frac{k-1}{2}$ to $2^{\frac{k-1}{2}}$ decoder

(Loop) حتى نصل لأبسط (decoder) وهو عندما $\boxed{k=1}$

  - * 2-to-4-line decoder → $k = n = 2$
    - We need 2^2(4) 2-input AND gates driven as follows:
    - k is even, so split to:
      - Two 1-to-2-line decoder

(k) is even (same)

$\frac{k}{2}$ to $2^{\frac{k}{2}}$ decoder

$\frac{k}{2}$ to $2^{\frac{k}{2}}$ decoder.

- See next slide for result

Decoder Expansion - Example 1

- $GN = 8 \times 2 + 4 \times 2 + 3$ → عدد الـ inverters
  AND gates. AND gates
  كل واحدة 2-inputs كل واحدة 2-inputs
- $GN = 27$
- **Straight forward design has the same GN cost**

نفس الـ and gates (8) لكل بوابة 2 (2-input) + 2 inverter

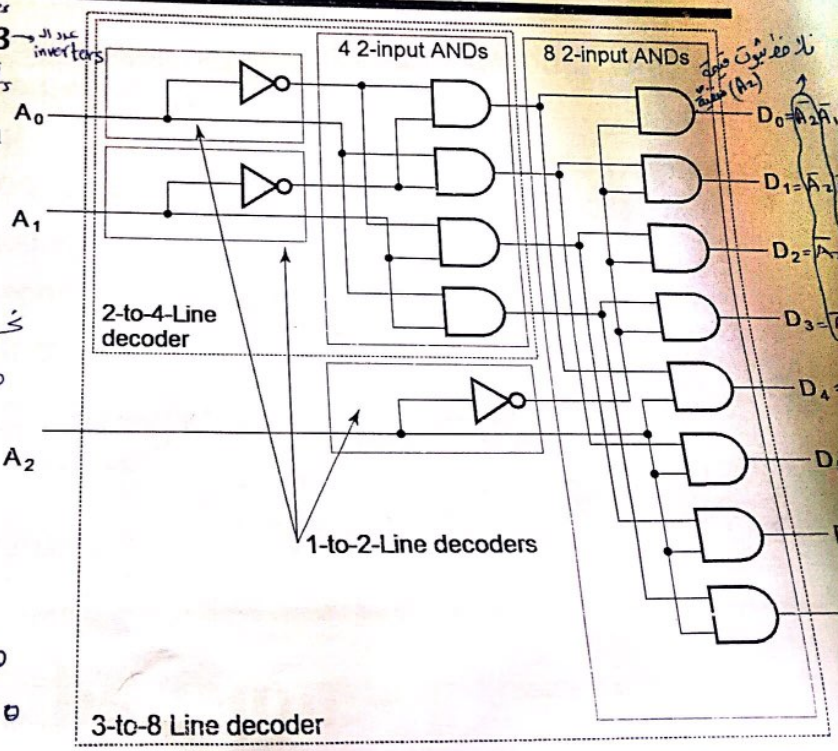
* من ناحية (cost) متساويات ولكن من ناحية (delay) مختلفوا.

* لكي نسهّل الرسم نبدأ من اليمين ثم نحدد شكل الـ output لكل (and gate) من حيث D_0 ← هي $\bar{A}_2\bar{A}_1\bar{A}_0$ وهكذا لكل المخرجات

* يجوز تبديل (A_2 بـ A_0) أو غيرها أي أنّ أي ترتيب لا يهمّ والمهم فقط أن ترتيب الـ outputs يكون عليه صح.

$0 \equiv \bar{A}_2\bar{A}_1\bar{A}_0$

$1 \equiv \bar{A}_2\bar{A}_1 A_0$



Decoder Expansion - Example 2

Decoder Expansion - Example 2

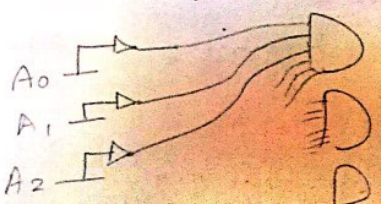
- 6-to-64-line decoder
  - $k = n = 6$ → The number of inputs.
  - We need 2^6(64) 2-input AND gates driven as follows: (2^n = number of AND gates).
  - k is even, so split to:
    - Two 3-to-8-line decoders
  - Each 3-to-8-line decoder is designed as shown in Example 1

or continue the loop!:

$k=n=3$ → odd → (1 to 2) decoder.
→ (2 to 4) decoder. → continue the loop! → $n=k=2$ even → 1 to 2 decoder.
→ 1 to 2 decoder.

Decoder Expansion - Example 2

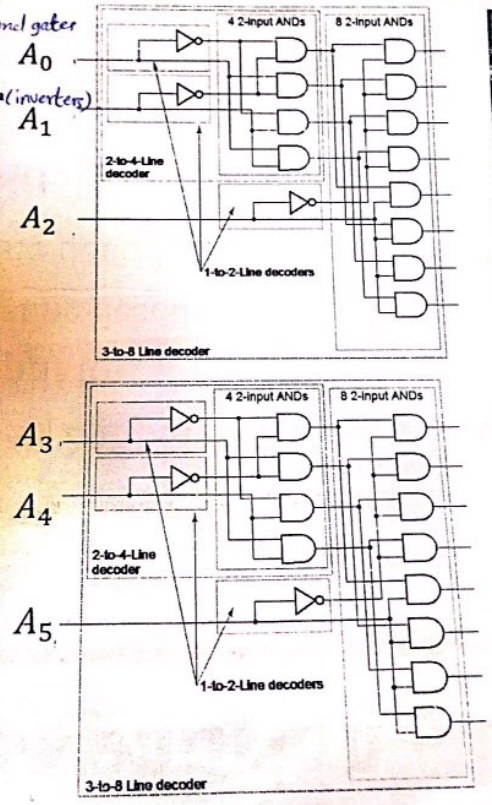
- $GN = 64 \times 2 + 16 \times 2 + 8 \times 2 + 6$
  (2^6) and gates. (2^2) and gates (2^4) and gates. (inverters)
- $GN = 182$
- **Straight forward design has GN cost of 390**



لكل بوابة (and) (6-inputs) وبوجد (6-inverters)

$64 \times 6 + 6 =$ الكلي

$\boxed{390}$



$D_0 = \overline{A_5}\,\overline{A_4}\,\overline{A_3}\,\overline{A_2}\,\overline{A_1}\,\overline{A_0}$

$D_1 = \overline{A_5}\,\overline{A_4}\,\overline{A_3}\,\overline{A_2}\,\overline{A_1}A_0$

For the cost decoder Expansion is better.
but for the delay the straight forward design is better because it has one level implementation

$D_{62} = A_5A_4A_3A_2A_1\overline{A_0}$

$D_{63} = A_5A_4A_3A_2A_1A_0$

Decoder Expansion - Example 3

- **7-to-128-line decoder**
  - $k = n = 7$
  - We need 2^7(128) 2-input AND gates driven as follows:
  - k is odd, so split to:
    - 4-to-16-line decoder
    - 3-to-8-line decoder
  - 4-to-16-line decoder
    - $k = n = 4$
    - We need 2^4(16) 2-input AND gates driven as follows:
    - k is even, so split to:
      - Two 2-to-4-line decoders
  - Complete using known 3-8 and 2-to-4 line decoders
- $GN = 128 \times 2 + 16 \times 2 + 8 \times 2 + 12 \times 2 + 7 = 335$
- Compare to straight forward design with GN cost of 903

2-input and gate. 2-input and gates. inputs

(1-2 decoders) عدد الـ = (inputs) عدد الـ = (inverter) عدد الـ

* write the procedure for the Decoder Expansion:

using Enables.

Building Larger Decoders

- **Method_1:** Decoder Expansion (الطريقة السابقة)
- **Method_2:** Using Small ***Decoders with Enable input*** (الطريقة الحالية):
- Example: 1-to-2 line decoder with enable
  - In general, attach ***m-enabling*** circuits to the outputs
  - See truth table below for function
    - Note use of X's to denote both 0 and 1
    - Combination containing two X's represent two binary combinations
- Alternatively, can be viewed as distributing value of signal EN to 1 of 2 outputs
  - In this case, it is called a *Demultiplexer*

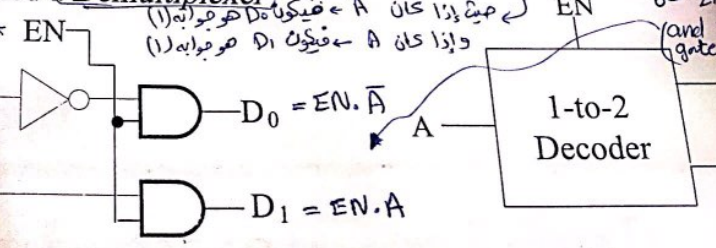
DMUX = (decoder + Enable)

* متى ما كان ال Enable ← (0) يكون D_0 و D_1 ← صفر.
* متى ما كان ال Enable ← (1) يكون حسب ال input لل D_0 و D_1.
* نفس التصميم القديم ولكن نضيف Enable على ال inputs مرتبط بـ (and gate)
* على الاثنين ال Enable لازم يكون (1) حيث إذا كان (0) يكون التابع (D_1 و D_0) أصفار مهما كانت قيمة ال input (X) (don't care)

EN	A	D_0	D_1
0	X	0	0
1	0	1	0
1	1	0	1

← لا يشتغل.
← يشتغل.

(a)



* حيث إذا كان $\bar{A}$ ← فيكون D_0 هو جوابه (1)
وإذا كان A ← فيكون D_1 هو جوابه (1)

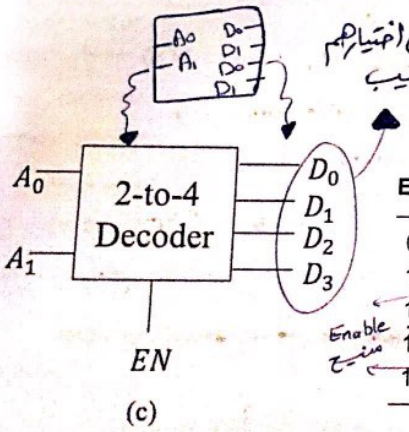
(b) (c)

When: $E_n = 0$
When: $E_n = 1$

2-to-4 Line Decoder with Enable

2-to-4 Line Decoder with Enable

- Attach ***4-enabling*** circuits to the outputs by an and gates
- See truth table below for function
  - Combination containing two X's represent four binary combinations (00, 01, 10, 11) ⊗don't cares represent الاحتمالات الـ4
- Alternatively, can be viewed as distributing value of signal EN to 1 of 4 outputs
  - In this case, it is called a *Demultiplexer*



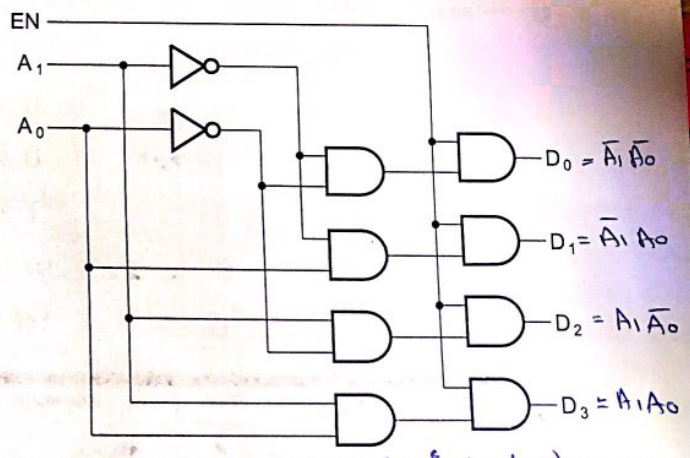
(c)

*الأفضل اختيارهم بالترتيب

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

Enable منشط

(a)



(b)

*غالباً الـ (Enable) يكون هو الـ (most significant digit)

* تم إضافة (4 بوابات and أخرى) لأن لكل بوابة منطق (2 inputs)

$n = 2$ = inputs

$m = 2^n$ = outputs = number of 2-input AND gates.

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

(two 1-2 decoder and one inverter) طريقة الأسئلة بالامتحان

③ ثم نحدد الـ output كيف بيكون شكلهم بمعلومية الـ inputs ونكمل توصيل الأسلاك.

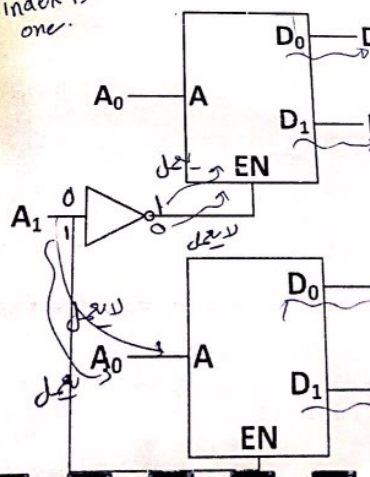
* لمعرفة كيفية حل السؤال:
① يجب علينا رسم T.T (Truth table) للـ decoder المطلوب اختصاره لـ decoder أبسط منه.
② نقسم الجدول ونحدد الـ most significant digit حسب الـ Enable.

A_1	A_0	Decimal	D_0	D_1	D_2	D_3
0	0	(0)	1	0	0	0
0	1	(1)	0	1	0	0
1	0	(2)	0	0	1	0
1	1	(3)	0	0	0	1

1st 1-to-2 Decoder — 2nd 1-to-2 Decoder

the index is one.

* فقط واحد (decoder) يشتغل ويكون جوابه (1) والباقي الـ (decoders) يكون جوابهم (0)
* وجود الـ Enable هو فقط لجعل واحد من الـ (decoders) تعطيه (1) أي شغال والباقي لا يعمل لكي لا يختلط قيم. فيكون الـ output الخارج قيمة و متعارضة.
* لاحظ: غالباً الـ (Enable) هو الـ (most significant digit.)



$D_0 = EN \cdot \bar{A}_0 \rightarrow \bar{A}_1 \bar{A}_0$

$D_1 = EN \cdot A_0 \rightarrow \bar{A}_1 A_0$

* نأخذ سطر عامل:

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

$EN \cdot \bar{A}_0 = A_1 \cdot \bar{A}_0$

نلاحظ أن: A_1 هو الـ Enable نعمله invert لكي يشتغل وعندما يعمل يعطي قيمة (1) عند الـ (D) التي رتبتها (0) وهي D_2 والباقي أصفار.

Logic and Computer Design Fundamentals, 4e PowerPoint® Slides

Chapter 2 - 10

على الأقل نحتاج عدد (5) 2to4 decoders لكي نمثله

3-to-8 Decoder using 2-to-4 Decoders and Inverters

على الأقل نحتاج عدد (2) 2to4 decoders لكي نمثله.

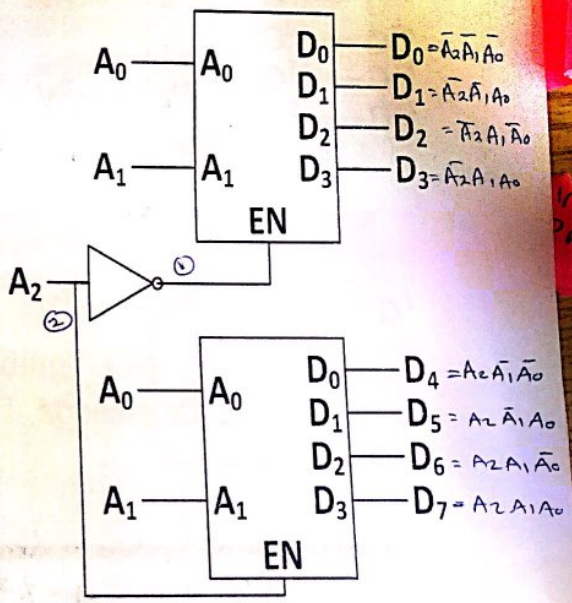
3-to-8 Decoder using 2-to-4 Decoders and Inverters

$n = 3$ = inputs.

$m = 8$ = outputs $= 2^n$ = number of 2-input AND gates.

	A_2	A_1	A_0	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{EN}$	0	0	0	1	0	0	0	0	0	0	0
	0	0	1	0	1	0	0	0	0	0	0
①	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
② EN	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



على الأقل نحتاج decoder(2) ← عشان (2to4)

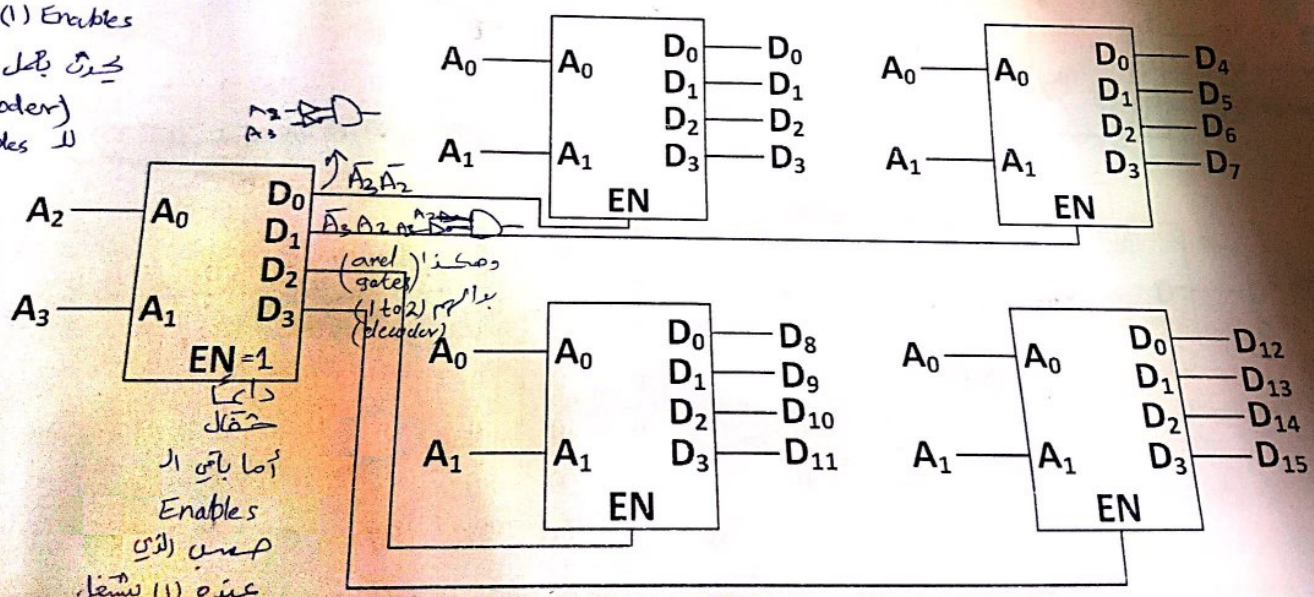
ننتج decoder(1) ← (3to8)

$n = 4$ = inputs.
$m = 16$ = outputs $= 2^n$ = number of AND gates.

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

عملياً الأقل نحتاج عدد (٤) 2 to 4 decoders لتشيل. → $4 * X = 16$ → $\boxed{X = 4}$ number of decoders.

* الكلي نقسمه وحدة من ال Enables (1) وذلك يحدث بجعل (decoder) لل Enables



دائماً شغال

وهكذا (and gates) بدالهم (1 to 2) (decoder)

أما باقي ال Enables حسب الذي عنده (1) يشتغل ويطلع الـ outputs حسب ذلك

Functional Logic Implementation

Example1

- Implement function f using decoder and OR gate:

$$f(x, y, z) = x\bar{z} + \bar{x}y$$ → from Boolean Equation to Some of minterms.

$m = 1$ number of functions.

- $n = 3$ variables → ***3-to-8 decoder***
- One function → ***One OR gate***
- Solution: Convert f to SOM format

or using truth table.

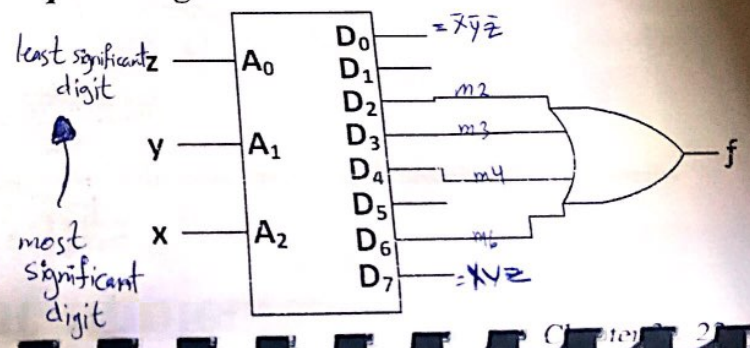
$x\bar{z}$ → $xy\bar{z}$ or $x\bar{y}\bar{z}$
110 100

وهكذا يكون جوابهم (1)
ونفرق ماهم ال minterms بس.

- $f = x\bar{z}(y + \bar{y}) + \bar{x}y(z + \bar{z}) = xy\bar{z} + x\bar{y}\bar{z} + \bar{x}yz + \bar{x}y\bar{z}$
- $f(x, y, z) = \sum_m(2,3,4,6)$ → ***4-input OR gate***

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

مهم كيف
ترتيبهم



Example2

Example2

- Implement function f using decoder and OR gate:

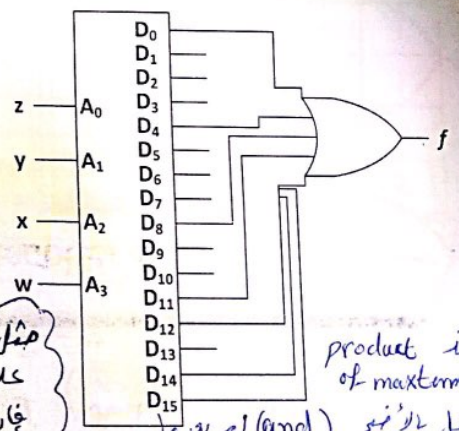
$$f(w,x,y,z) = \sum_{m}(0,4,8,11,12,14,15)$$

- $n = 4$ variables → ***4-to-16 decoder***
- One function with 7 minterms → ***One 7-input OR gate***

one (OR) gate

Note:*

- ***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***

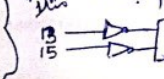


* أحياناً قد يكون الـ Function يحتوي على أكثر من نصف الـ (minterms) أي أن عددهم كبير ولا يجوز جعلهم جميعاً (inputs) لل (OR gate) لذلك نأخذ الـ ($\bar{F}$) ونجعل بآخر شكل الـ (OR gate) فتصبح (inverter) ، (gate NOR) حيث يكون قل عدد الـ (inputs).

مثلاً أن F تحتوي على (minterm :١٤) فإن $\bar{F}$ تحتوي على (minterm : ٢) ، ..... وذلك أفضل

أو نأخذ product of maxterms وذلك نعمل بالأخير (and) لهم بعدها نعمل لكل (maxterm) inverter

مثلاً



* لو كان عنّا (2functions) وبختلفوا في عدد ال inputs

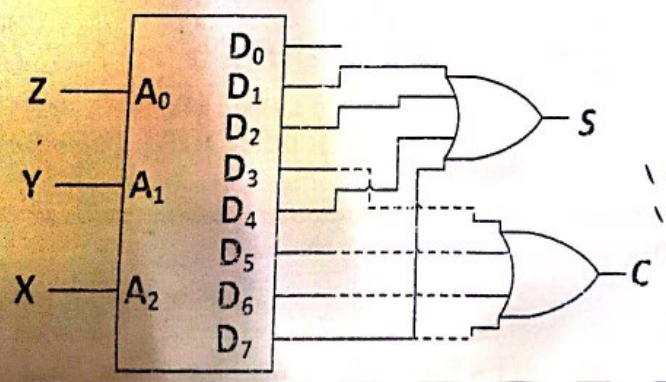
Example3

نعمل للـ (function الناقص) مع and $(W+\bar{W})$ [الحد الناقص]

وبكتب ال (minterms) لكلا الأمتراتين بدلالة العدد الأكبر من المتغيرات. حيث أن (W) لا تؤثر في قيمة ال (minterms) بال (function الناقص).

- Implement functions *C and S* using decoder and OR gates: ← هيك بيأتي السؤال بالكتاب.
- $n = 3$ 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	m0
0	0	1	0	1	m1
0	1	0	0	1	m2
0	1	1	1	0	m3
1	0	0	0	1	m4
1	0	1	1	0	m5
1	1	0	1	0	m6
1	1	1	1	1	m7



* نرى متى يكون ال Function $C = 1$ ، $S = 1$ ونكتبه كشكل مجموع minterms ونوصلهم بـ (OR gate)

Example4 A7 A6 A5 A4

Example5

- Implement function F using 3-to-8 decoder, AND gate and (4) inverters: $F(A,B,C) = \sum_m(1,3,5,7)$ 4 minterms + 4 maxterms.

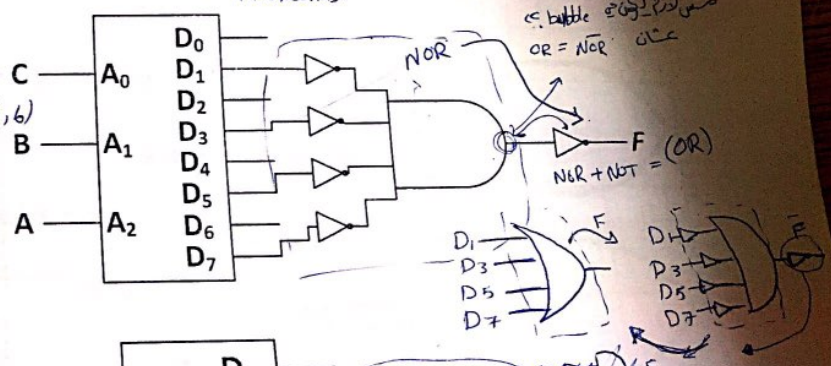
- Solution with 5 inverters:

$\sum_m F$ مثل $\prod_M F$

$\sum_m F = \overline{\prod_M F}$

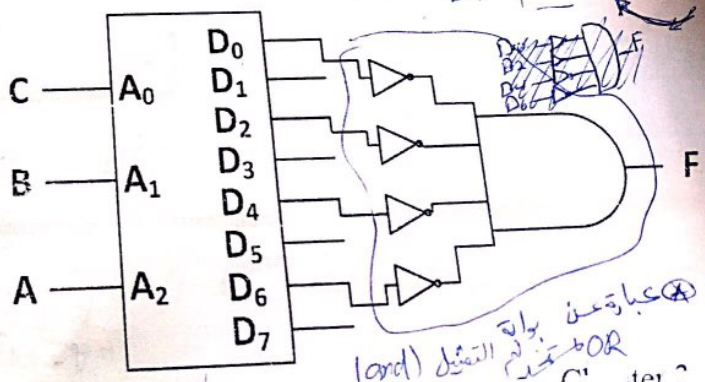
$\prod_M F = \sum_m \overline{F}$

$(\overline{F}) = \sum_m(0,2,4,6)$ as a sum of minterms



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

$F \rightarrow \prod_M (0,2,4,6)$



(and) عبارة عن بوابة OR تستخدم التمثيل

Encoding

(one input = 1)
m → inputs / n → outputs.

decoder:- (one output = 1)
n → inputs / m → outputs.

- ***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***

  inputs → D → Data.
  outputs → A → Address.

- 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

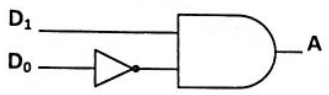
تحويل ال (input) الذي يحتوي على (1)
إلى (index) ووضعه (1) في ال (output)

2-to-1 Encoder & 4-to-2 Encoder

D_1	D_0	A
0	0	Invalid input
0	1	0 (decimal)
1	0	1 (decimal)
1	1	Invalid input

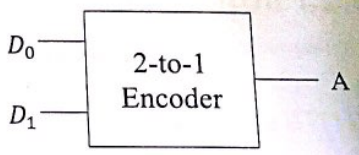
(a)

decimal بال
كمية ولكن نمثله
binary بال
يعني لو كان ما (2) (*) رقم ال (index) الذي جوابه (1) بال input حوال
$(10)_2 =$



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

(b)



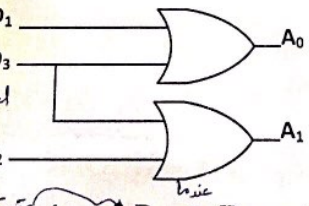
(c)

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)

حوال لي
output
بال
decimal

تمثيل (0) بال decimal
تمثيل (1)
تمثيل (2)
تمثيل (3)

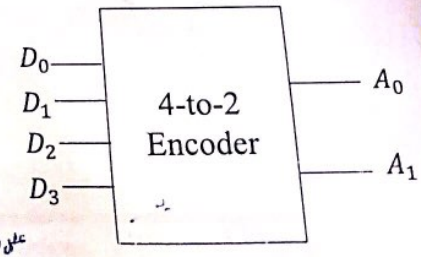


A_0 متى تكون (1) تساوي $A_0 = D_1 + D_3$ عندما

A_1 متى تكون (1) تساوي $A_1 = D_2 + D_3$

بدي نوا على (1) منهم (1) والباقي by default (0)

(b)



(c)

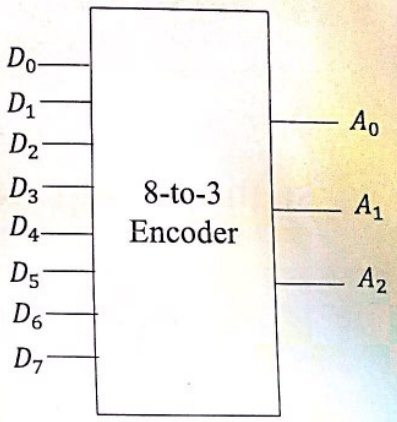
8-to-3 Encoder (Octal-to-Binary Encoder)

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

كأنّه تحويل من octal ← binary

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)



$$A_0 = D_1 + D_3 + D_5 + D_7$$
001 001 010 101 111

$$A_1 = D_2 + D_3 + D_6 + D_7$$
010 011 110 111

$$A_2 = D_4 + D_5 + D_6 + D_7$$
100 101 110 111

(b)

(c)

$A_2A_1A_0$
000 → D_0
001 → D_1

* لمّا ال bit التي ترتيبها نفس ال index تاع (A) يكون هو (1)

ممكن يكون (A2) (1) في أي حالة ↑ نضعها بدل ما نعمل truth table

Priority Encoder

- If more than one input value is 1, then the encoder just designed does not work
- One encoder that can accept all possible combinations of input values and produce a meaningful result is a ***priority encoder***

  يعطي الأولوية لرقم له شروط معينة
- Among the 1s that appear, it selects the most significant input position (or the least significant input position) containing a 1 and responds with the corresponding binary code for that position
  - ***High priority encoder:*** gives priority for the input whose value is 1 and has the highest subscript

    يعطي الأولوية للرقم الأكبر ليكون عنده (1) بال inputs
  - ***low priority encoder:*** gives priority for the input whose value is 1 and has the lowest subscript

    يعطي الأولوية للرقم الأقل ليكون عنده (1) بال input
- If all inputs are 0's, what happens?

  وظيفة ال (Valid bit) ال function فقط هي لمعرفة إذا ما كان كامل "صفر أو لا"
  - Define an output (V) to encode whether the input is valid or not
  - When all inputs are 0's, V is set to 0 indicating that the input is invalid, otherwise V is set to 1

    ولكن لا نعلم كم وجود (1)=input

4-to-2 Low Priority Encoder

4-to-2 Low Priority Encoder

#_of_Minterms/ Rows	D_3	D_2	D_1	D_0	A_1	A_0	V
$2^0=1$	0	0	0	0	X	X	0
$2^3=8$	X	X	X	1	0	0	1
$2^2=4$	X	X	1	0	0	1	1
$2^1=2$	X	1	0	0	1	0	1
$2^0=1$	1	0	0	0	1	1	1

(a)

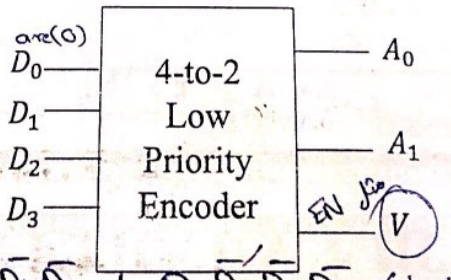
Number of Minterms per Row = 2^k (k = number of inputs)

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

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

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

(b)



* valid bit = 0 . when all inputs are(0)
* valid bit = 1 . when one or two or more than input = (1)

$$A_2 = D_2\overline{D_1}\,\overline{D_0} + D_3\overline{D_2}\,\overline{D_1}\,\overline{D_0} \text{ (simplification theorem)}$$
$$A_2 = D_2\overline{D_1}\,\overline{D_0} + D_3\overline{D_1}\,\overline{D_0}$$

4-to-2 High Priority Encoder

يعطي priority للـ input الذي رقمه أعلى.

لـ الـ Truth table عكس الـ low priority بالزبط

#_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)

(b)

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

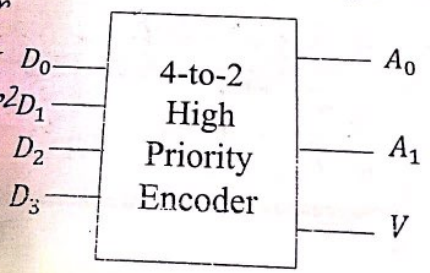
$$A_1 = D_3 + \overline{D_3}D_2$$
$$A_1 = D_3 + D_2$$

simplification theorem.

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

valid bit equation.

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



(c)

To write the equation we don't need to make the whole (T.T). If we have a 4 to 2 high priority encoder and we want the equation for:

$V_{alid} = V = D_3 + D_2 + D_1 + D_0$

$A_0 = D_1\overline{D_2}\,\overline{D_3} + D_3 \xrightarrow{\text{simplification theorem}} D_1\overline{D_2} + D_3$

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

5-input Priority Encoder

$2^2 = 4$ not 5 x $2^3 = 8$ أقل

5-input Priority Encoder

$2^2 = 4$ not 5 ✗ $2^3 = 8$ → أقل عدد لازم

- 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	X	X	X	0
1	0	0	0	0	1	0	0	0	1
2	0	0	0	1	X	0	0	1	1
4	0	0	1	X	X	0	1	0	1
8	0	1	X	X	X	0	1	1	1
16	1	X	X	X	X	1	0	0	1

(inputs) السفرتية هي اللي تكون A0 لها = (1) وهي بـ D0 | D3 وحسب نوع الـ proinity خود الـ (D) العنفية ال complemented.

لازم يكون صفر عشان يكون الـ (one) اللي قبلها priority أعلى

don't cares → موجود بين على D high priority عيش الجدول لازم

- 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 minterms are present in the product terms in the table

5-input Priority Encoder Cont.

$2^3 = 8$ لكل (encoder) عدد أعلى من ال (inputs) يسمح بتمثيله مثلاً 8 يجوز

8 to 3 أقل منه ولكن أكثر

6 to 3 لا يجوز

7 to 3

$2^4 = 16$

9 to 4

12 to 4

13 to 4

16 to 4

$2^5 = 32$

30 to 5

32 to 5

18 to 5

- Could use a K-map to get equations, but can be read directly from table and manually optimized if careful:

$$A_2 = D_4$$

$$A_1 = \overline{D}_4 D_3 + \overline{D}_4 \overline{D}_3 D_2 = \overline{D}_4 (D_3 + D_2)$$

$$A_1 = \overline{D}_4 D_3 + \overline{D}_4 D_2$$

$$A_0 = \overline{D}_4 D_3 + \overline{D}_4 \overline{D}_3 \overline{D}_2 D_1 = \overline{D}_4 (D_3 + \overline{D}_2 D1)$$

$$A_0 = \overline{D}_4 D_3 + \overline{D}_4 \overline{D}_2 D_1$$

$$V = D_4 + D_3 + D_2 + D_1 + D_0$$

Selecting

Multiplexers (MUX) (Data Selectors)

لا يوجد فيه (Hi-Z) وظيفته تختار وتحمل (select) لأحد الـ (inputs) وتظهره على الـ (output)

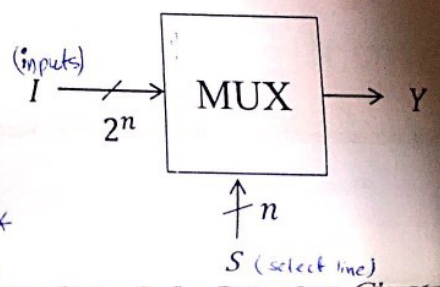
- A multiplexer selects information from an input line and directs the information to an output line
- A typical multiplexer has ***n* control inputs** ($S_{n-1}, \ldots S_0$) called ***selection inputs***, **2^n information inputs** ($I_{2^n-1}, \ldots I_0$), and **one output Y**
- A multiplexer can 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

عدد الـ inputs أقل أو تساوي $2^{(n)}$ عدد الـ variables

2 to 1 mux
4 to 1 mux
8 to 1 mux

(one output دائماً)

* الـ (decoder) يكون الـ output له يساوي الـ (address) التي له (1) فيما (binary).
* بينما الـ (mux) يكون الـ output هو الذي نختار ظهوره 0 أو 1



Logic and Computer Design Fundamentals, 4e
PowerPoint® Slides

2-to-1-Line MUX

2-to-1-Line MUX

وهو الذي يحدد من ال (inputs) جنس ال (select line) يُعتبر ال

وهو الذي يحدد من ال (inputs) الذي منتصه تطبع على ال (outputs)

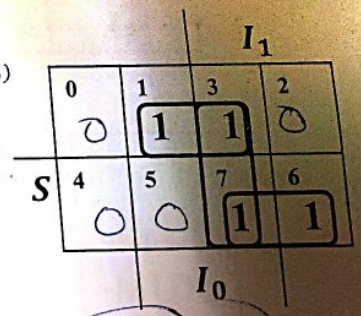
- Since $2 = 2^1$, $n = 1$
- The single selection variable S has two values:
  - S = 0 selects input I_0
  - S = 1 selects input I_1
- The equation:

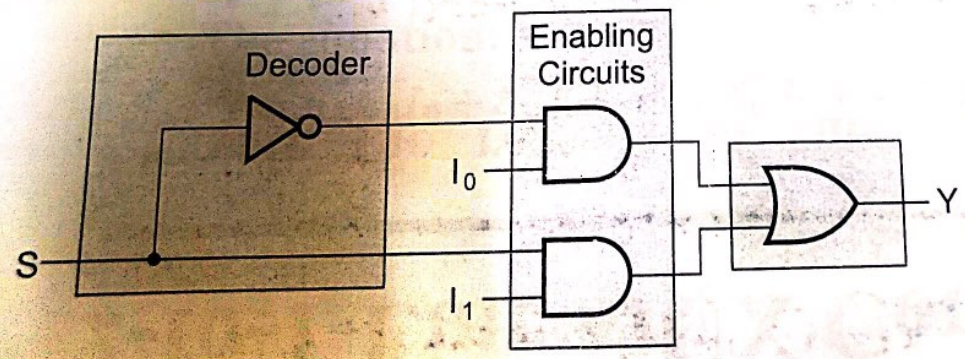
$$Y = \bar{S}I_0 + SI_1$$

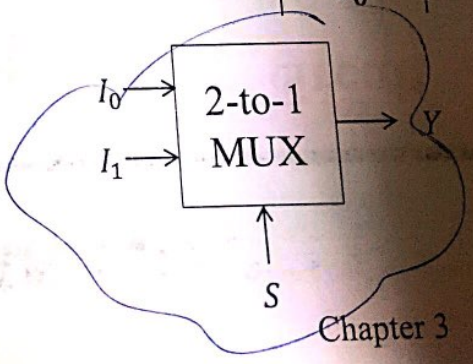
- The circuit:

S	I_1	I_0	Y	
0	0	0	0	y = I_0 when S = 0 بشرط ($\bar{S}$) $Y = I_0$ selected (I_0)
0	0	1	1	
0	1	0	0	
0	1	1	1	
1	0	0	0	y = I_1 when S = 1 بشرط (S) $Y = I_1$ selected (I_1)
1	0	1	0	
1	1	0	1	
1	1	1	1	

S	Y
0	I_0
1	I_1







4-to-1-Line MUX

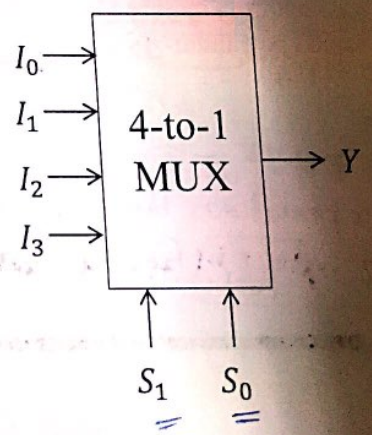
Short truth table بدل
وضع كل ال inputs بدل (64) سطر

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

- Since $4 = 2^2$, n = 2
- There are two selection variables (S_1S_0) and they have four values:
  - $S_1S_0 = 00$ selects input I_0 (0)10
  - $S_1S_0 = 01$ selects input I_1 (1)10
  - $S_1S_0 = 10$ selects input I_2 (2)10
  - $S_1S_0 = 11$ selects input I_3 (3)10
- The equation:

$$Y = \overline{S_1}\,\overline{S_0}I_0 + \overline{S_1}\,S_0I_1 + S_1\overline{S_0}\,I_2 + S_1S_0I_3$$

اهتمام اظهار I_0
وهكذا



2-select lines.

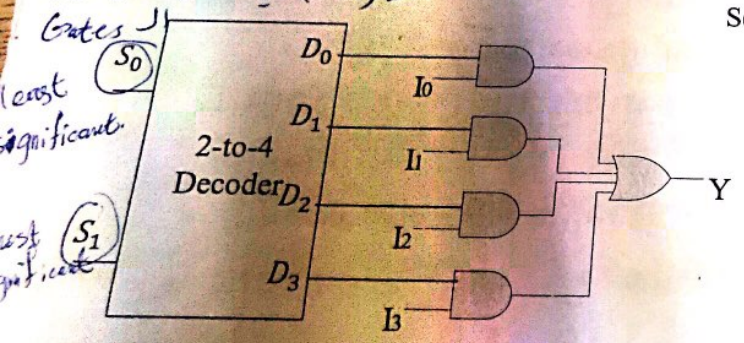
Design a 4-to-1-line MUX Cont. using only

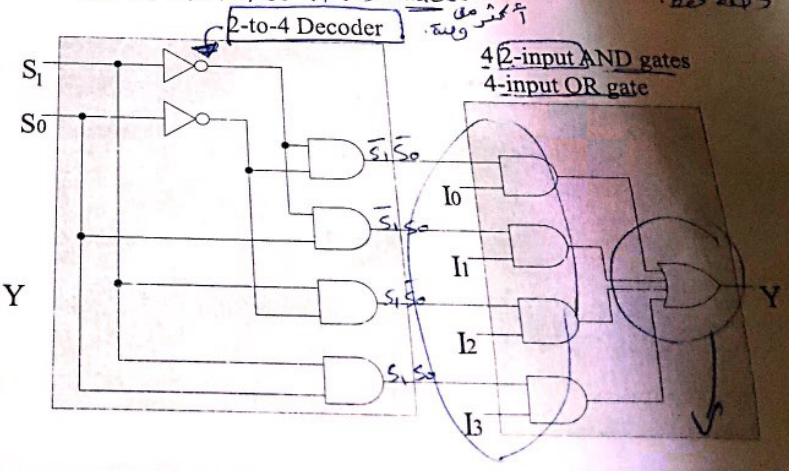
- 1 ▪ 2-to-4-line decoder ($n = 2$) number of select lines.
- 2 ▪ 4 2-input AND gates
- 3 ▪ 4-input OR gate

In General if we are building (2^n) to 1 mux we will use the following components:-

① n to 2^n decoder.
② 2^n - 2 input AND Gates. أكثر من واحدة
③ 2^n - input OR Gate. واحدة فقط

* يتم استخدام ال (decoder) في (design) ال (mux) لكي نختصر ونقلل عدد ال Gates





2^n - 2input and gate
4 - 2input and gate

2^2 - input OR gate
4 - input OR gate

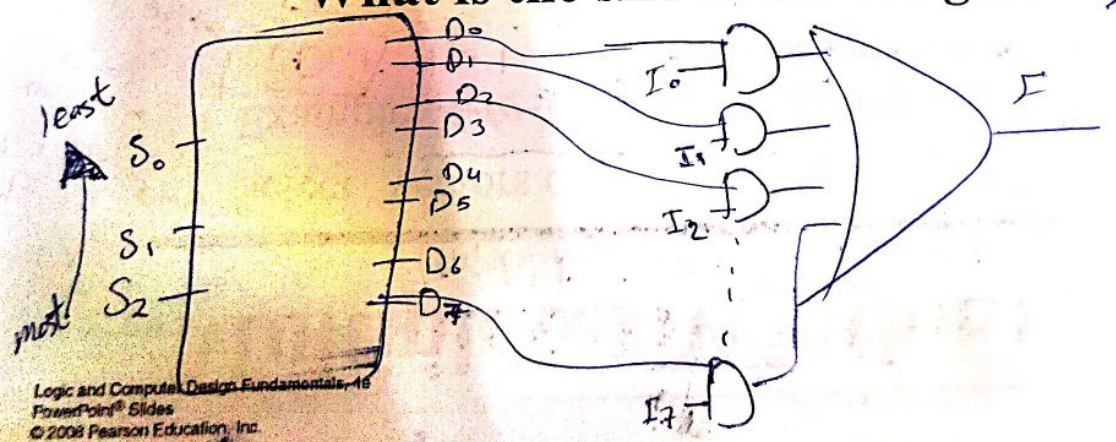
Homework

Homework

- **Implement 8-to-1-Line MUX and 64-to-1 MUX:**
  - **How many select lines are needed?** n = Select lines.
  - **Decoder size?** n to 2^n decoder.
  - **How many 2-input AND gates are needed?** 2^n - 2 input AND Gates
  - **What is the size of the OR gate?** 2^n - input OR Gate.

* n = 3. * 8-input OR gate
* 3 to 8 decoder.
* 8 - 2 input and gates

* n = 4.
* 4 to 16 decoder.
* 16 - 2 input and gates
* 16-input OR gate.



عدد ال bits

Multiplexer Width Expansion

عدد ال (bits) لل (output) = عدد ال (bits) لل (inputs)

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

ال bits فيهم 3, 2 مش بس (1, 0)

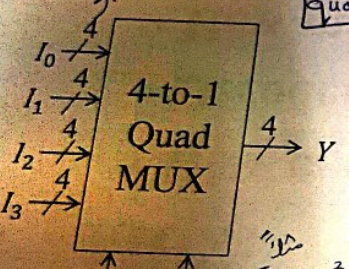
ال (bits) لهم (0, 1) فقط

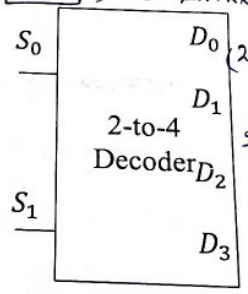
Dual 4 to 1 - mux. → 2 bit, 4 to 1 - mux

tri 4 to 1 - mux → 3 bit, 4 to 1 - mux

input with (4-bits.)

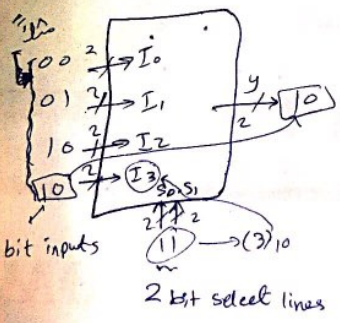
quad 4 to 1 mux → 4 bit, 4 to 1 mux





ال (2-input bits) سيظهروا حسب ال (2-select lines) يعني $\bar{S}_1\bar{S}_0 = I_0[0]I_0[1]$

أصبح input I_0 يتكون من (2-bits) وهما $I_0[0]$ و $I_0[1]$

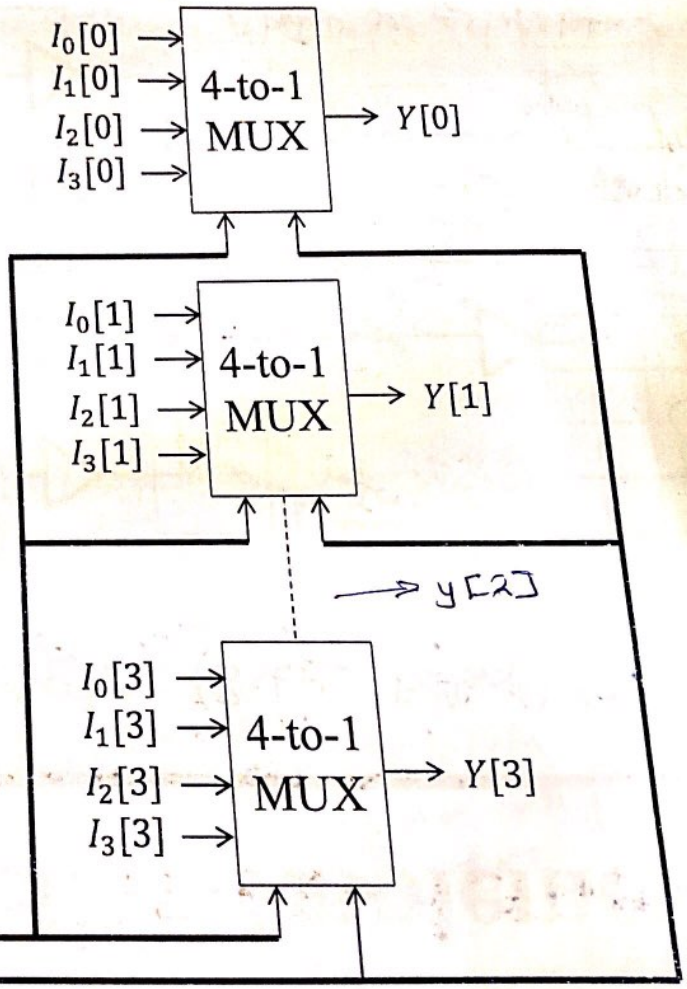


دائما نفس ال (input) ناخذ نفس رقمه ولكن الذي يختلف هو فقط ال (bits)

عدد ال (select lines) لتحديد على ال (inputs) ليس على ال (width) أي عدد ال (bits) للإدخال

Multiplexer Width Expansion Cont.

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



quad . 4 to 1 mux.

4-bits .

لنقطع العدد Y كامل
قيمة ك (2bits)

نشبكهم ال (2select lines) على

نفس ال (input) مع

لكي نختار ال (Value) من فوق مع من تحت

Other Selection Implementations

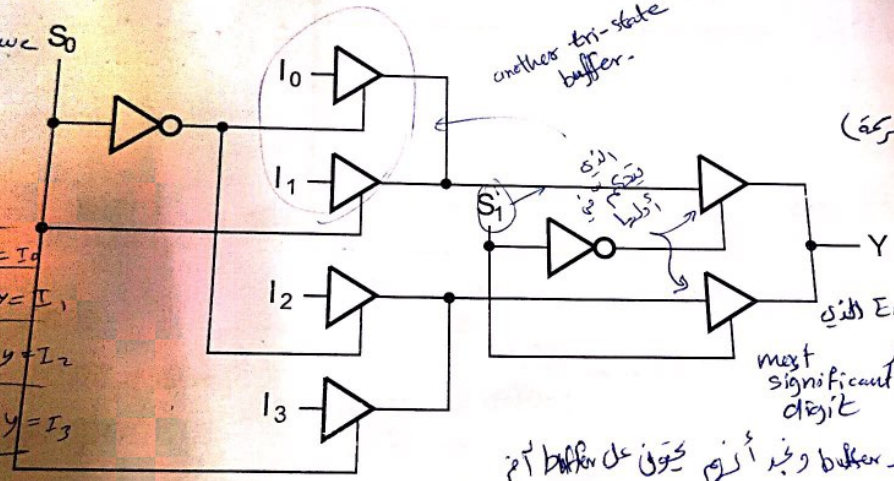
* مثل طريقة الـ Tri-state buffers السابقة نقسم الجدول بالنصف ونكمل الشكل على اليمين

- **Three-state logic** (3 state buffers) (Tri-state buffers).

It is 4 to 1 MUX

because we have 2 select lines and 4 inputs

S_1	S_0	y
0	0	I_0 $y = I_0$
0	1	I_1 $y = I_1$
1	0	I_2 $y = I_2$
1	1	I_3 $y = I_3$

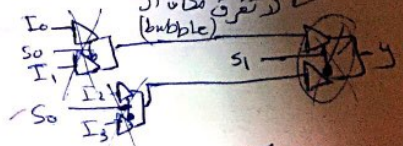


* نبدأ الرسمة من اليمين (آخر الرسمة)

ثم نريد تحديد الـ Enable الذي يتحكم حسب الـ (select line) الـ most significant digit

ثم نبدأ نكتب الـ inputs الـ buffer ونجد أنه يكون على buffer آخر

غلطة! مكان الـ (bubble) انتبه لازم عند I_2 / I_0 ← لا تفرق مكان الـ (bubble)



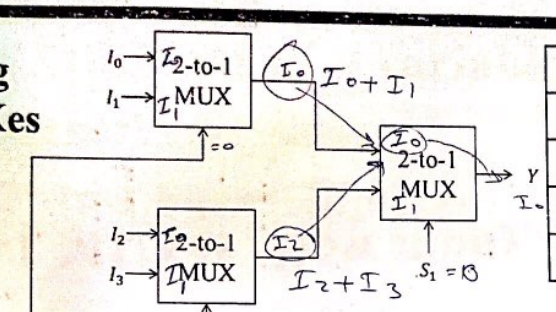
I_0, I_1, I_2, I_3 (inputs) وكذلك الـ (most significant) لأنه التالي (select line) حكم Enable من نوع واحد هو

Building Large MUXes from Smaller Ones

Building Large MUXes from Smaller Ones

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

على الأقل نحتاج (2) من (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**

متعوّدين (4 to 1) أو (8 to 1)

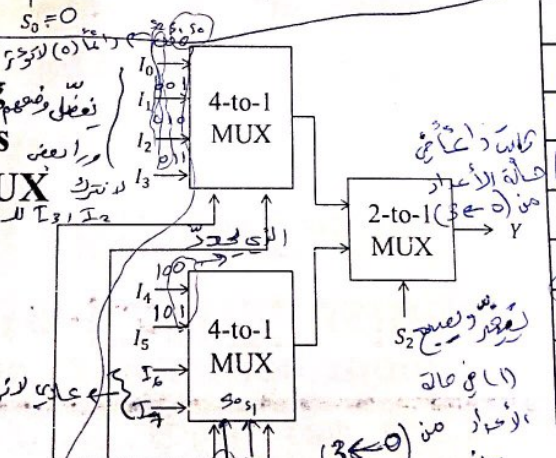
عادة يجوز (6 to 1) عشان شرط عدد الـ outputs 2^n معنى أقل منه

3 select lines

لأنه (2)

$2^2 = (4)$ لا يكفوا

$2^3 = (8)$ يسلم



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

$= \overline{S_2}\,\overline{S_1}$

$= \overline{S_2}\,\overline{S}$

$= S_2\,\overline{S}$

$= S_2$

ال (S_2) كل مرة يثبت على قيمة معيّنة و ياخذ منها جميع قيم (S_0) اللي هي تحدد ماذا يطلع على ال output بعدها

قيمة (S_2) ⟸ 0 ، 1

لأنه نختارهم

هو اللي يؤثر فيهم (S_1) وقيمته (0) لأن قيمهم عند I_7 / I_6 مش

نخليه (0) لأنه I_7 / I_6 مش

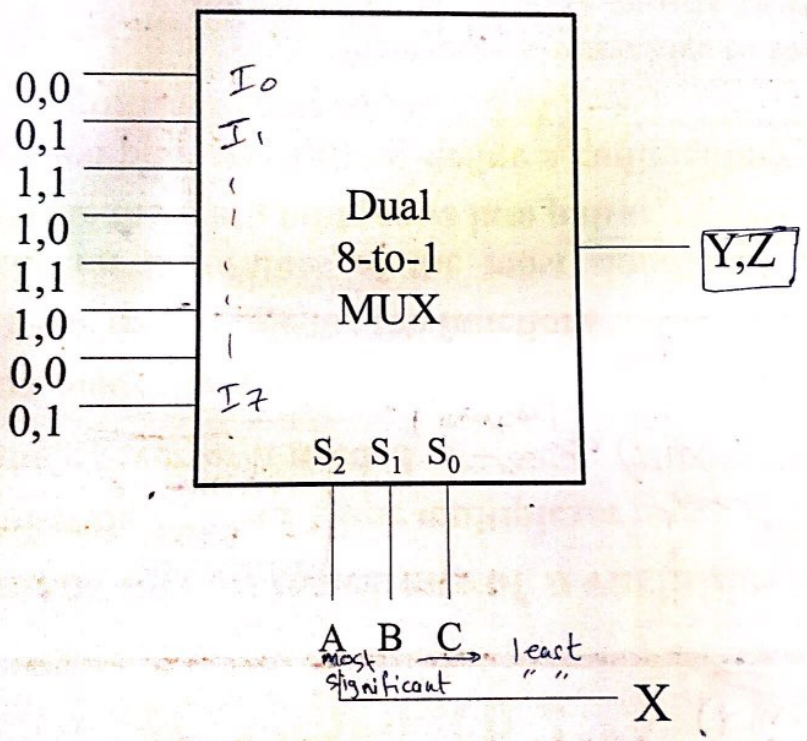
Gray to Binary Code Cont.

- 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.

Gray to Binary Code Cont.



هنا (A = X) نقلل ال width
بدل (2 bits ← 3 bits)

Combinational Logic Implementation - Multiplexer Approach 2

لتصغير ال (MUX)

- **Implement any *m* functions of *n* variables by using:**
  - **An m-wide $2^{(n-1)}$-to-1-line multiplexer** تصغير ال
  - **A single inverter if needed** ← وإضافة (MUX). (inverter)
- **Design:**
  - **Find the truth table for the functions** الأصل لبعده
  - **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 $\overline{X}$**

Example1

لأنه لم يعطى له مهمة ستجد الfunction بدلالة the relation.

Example1

لأنه لم يعمل له مهمة ستجعل الـFunction بدلالة the relation.

* لو كان (2 select lines) نضعهم (A,B) ونضع جدول Function بدلالة (C D) ونضيع نأخذ ٤ سطور مع بعض

The select lines

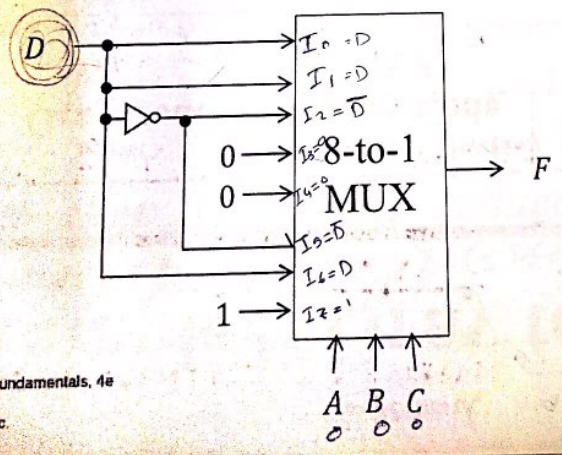
- **Implement the following function using a single MUX and an inverter (if needed) based on Approach2 :**

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

$2^{(n-1)}$ to 1 →

- Solution:
  - Single function → m = 1
  - 4 variables → n = 4 → 8-to-1 MUX
  - Fill the truth table of F

2^3 to 1 = (8 to 1) بدل (16 to 1) ويمكن استخدام (4 to 1) ولكن سنحتاج لـ logic gates أكثر ليس فقط زيادة (inverter) عليهم



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	$F = 0$ constant
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	

implement a circuit to convert from

Example2: Gray to Binary Code

(3-bits)

* 3 functions:
X (A, B, C)
Y (A, B, C)
Z (A, B, C)

* لكي تسهّل ال (implementation) كتابة ال (T.T) بالترتيب الصحيح.

inputs Gray Code ABC مرتبين لانو جرمتاكي	output (3 Function) Binary Code XYZ	Rudimentary Functions of C for Y	Rudimentary Functions of C for Z
000 (0)10	000	$Y = 0$	$Z = C$
001 (1)10	001		
010 (2)10	011	$Y = 1$	$Z = \overline{C}$
011 (3)10	010		
100 (4)10	111	$Y = 1$	$Z = \overline{C}$
101 (5)10	110		
110 (6)10	100	$Y = 0$	$Z = C$
111 (7)10	101		

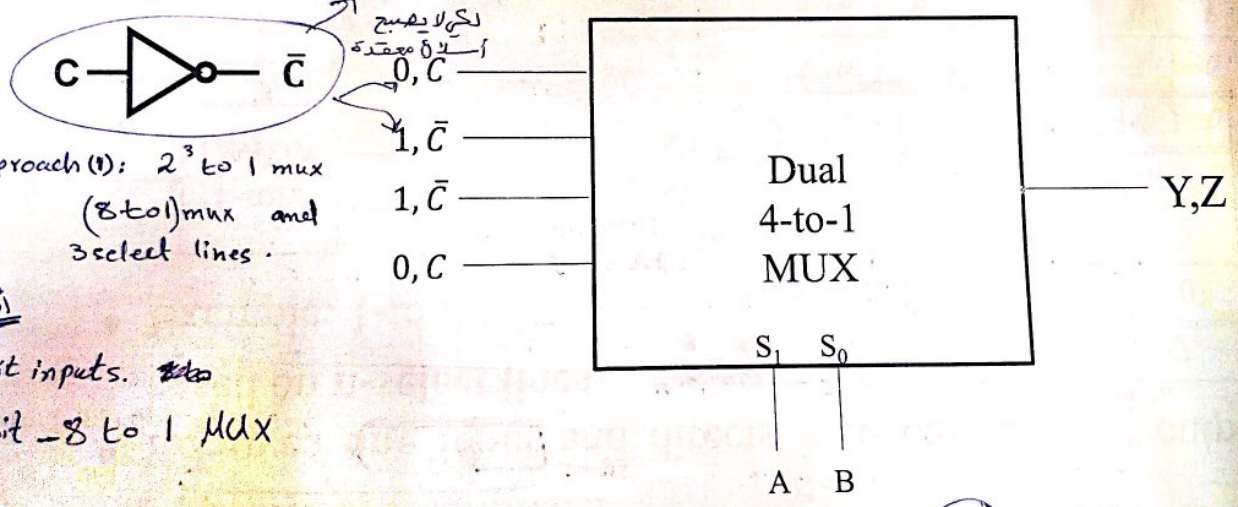
نقارن ونكتب ال Function بدلالة (C)

Gray to Binary Code Cont.

... the variables and functions to the multiplexer

Gray to Binary Code Cont.

- **Assign the variables and functions to the multiplexer inputs:** تسهيل للشكل الرسمة



(*) Approach (1): 2^3 to 1 mux (8 to 1) mux and 3 select lines.

أو

3 bit inputs.

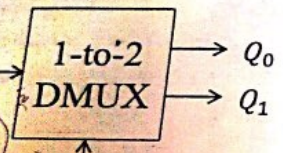
3 bit _ 8 to 1 MUX

- ***Note that Approach2 reduces the cost by almost half compared to Approach1*** approach (2) is better than approach (1)

Demultiplexer (DMUX)

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

4 outputs → 2 select lines

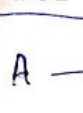


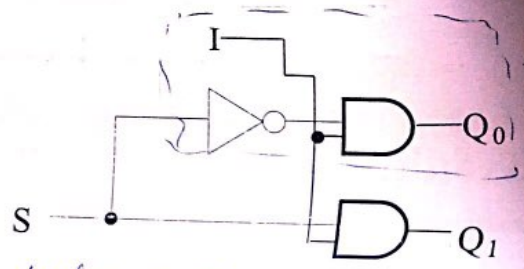
$$Q_0 = \bar{S}I$$
$$Q_1 = SI$$

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

(1 to 2 decoder with enable)

- ***DMUX ≡ Decoder with Enable***





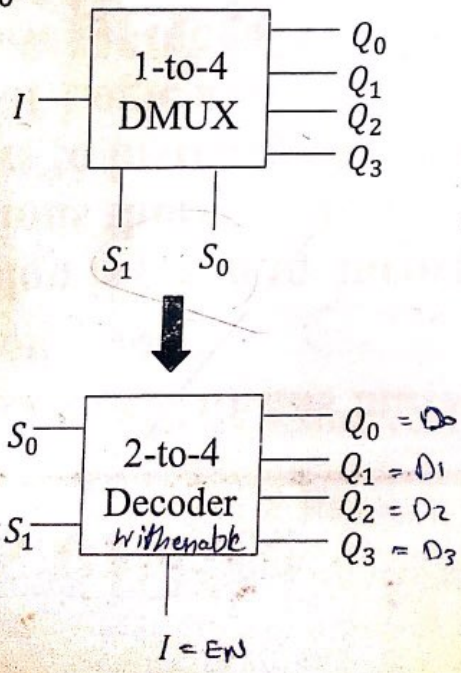
decoder with Enable

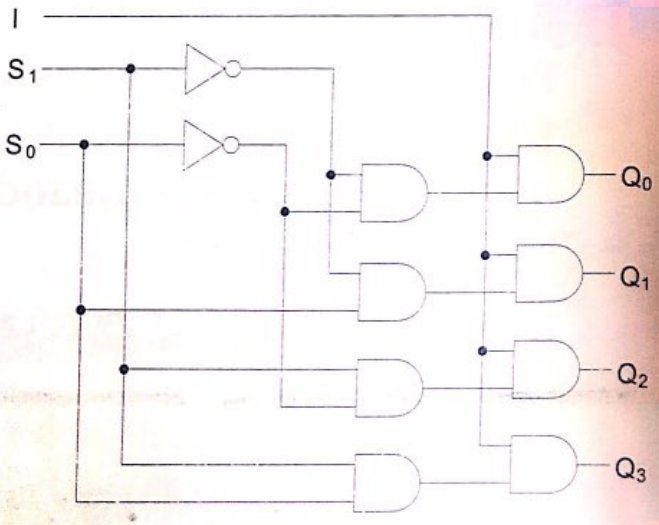
1-to-4 DMUX

- $Q_0 = \overline{S_1}\,\overline{S_0} I$
- $Q_1 = \overline{S_1} S_0 I$
- $Q_2 = S_1 \overline{S_0} I$
- $Q_3 = S_1 S_0 I$

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

$S_1 = 0$, $S_0 = 0$ } تفعّل الـ input على Q_0 وهكذا





* يجوز عمل (width-Expansion) بزيادة عدد الـ bits للـ inputs والـ outputs وعدد الـ select lines يبقى ثابت اعتماداً على العدد للـ (inputs).

* الـ (width-Expansion) فقط ينطبق على الـ DMUX / MUX لا ينطبق على Encoder / Decoder

Raghad Jaber.

Logic and Computer Design Fundamentals

Chapter 4 – Arithmetic Functions

operation

العمليات الحسابية

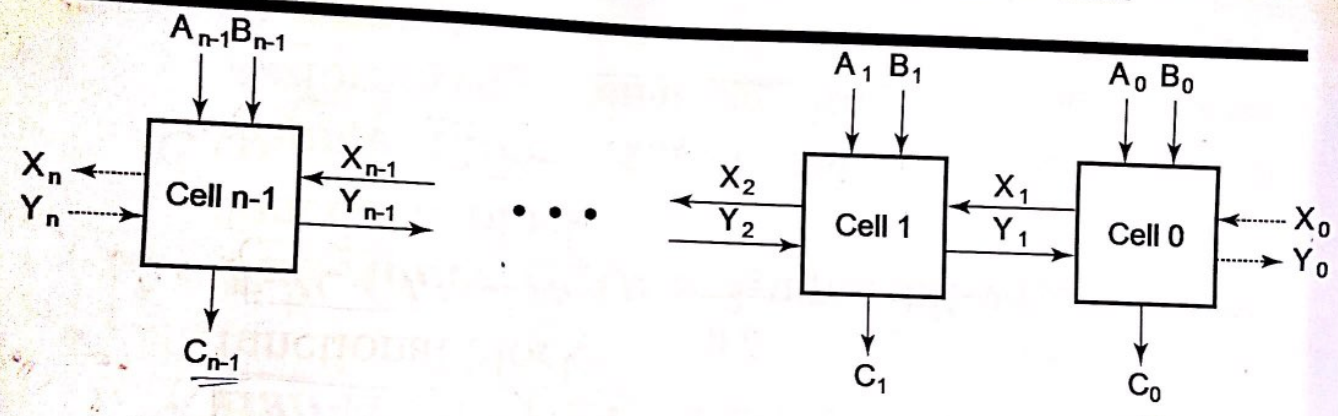
addition subtraction.

Charles Kime & Thomas Kaminski

(Hyperlinks are active in View Show mode)

Block Diagram of an Iterative Array

تكرار



- Example: n = 32 Total number of cells is (n).
  - Number of inputs = 32*2 + 1 + 1 = 66 مثال رقمي
  - Truth table rows = 2^{66}
  - Equations with up to 66 input variables
  - Equations with huge number of terms
  - Design impractical! غير عملي!

*number of bits in the inputs: $\overset{A}{n}+\overset{B}{n}+\overset{y\ x}{2}$ $= \boxed{2n+2}$ total number of bits

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

مجموعة ال (cells) مشبوكين مع بعض.

* Functional Blocks: Addition

عملية الجمع

- Binary addition used frequently

- Addition Development:
  - ① ***Half-Adder (HA):*** a 2-input bit-wise addition functional block
  - ② ***Full-Adder (FA):*** a 3-input bit-wise addition functional block
  - ③ ***Ripple Carry Adder:*** an iterative array to perform vector binary addition

اسم ال (cell) بإيه" وحجمها صغير.

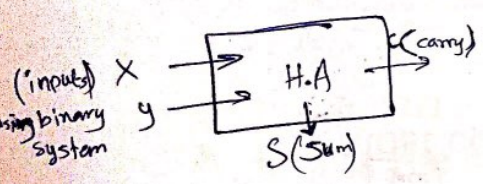
تم تغيير اسم وحجم ال (cell)

مجموعة من ال (full adders) مع بعض.

Half Adder (HA)

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
	carry sum	carry sum	carry sum	carry sum

1+1=(2)10

(10)2 = (2)10

- 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 ⇒

(inputs) X Y	(outputs) 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: $2^2 = 4$ sized k-map.

the sum (S) output = (1) when X = 1 or Y = 1 (or) both

(Sum) $S = X \cdot Y + X \cdot Y = X \oplus Y$

the boolean for X $\boxed{xor}$ y

(carry) $C = X \cdot Y$ → both x and y should be (1)

and gate

so the output (C) would be (1)

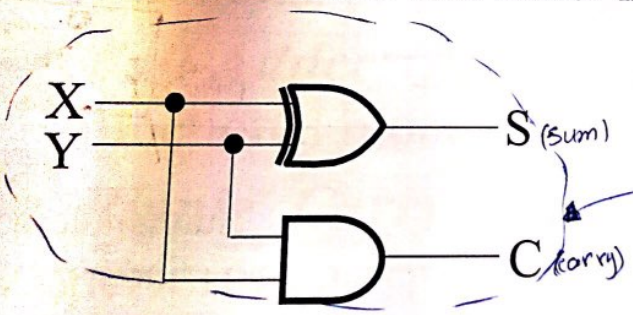
S

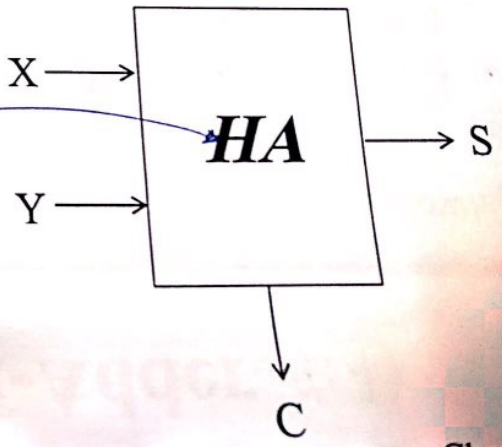
		Y	
		0	1 1
X		1 2	3

C

		Y	
		0	1
X		2	1 3

- The most common half adder implementation is:





The design inside the (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)***

الـ (HA) يأخذ قيمتين جمعهم فقط.
بينما الـ (FA) يأخذ ٣ قيم ← قيمتين inputs
+ ال carry bit من ال input السابقة.

- 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:

نثبت قيمة (Z) ونأخذ معها كل احتمالات (X, Y)

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

carry Sum

$1+1+1=(3)_{10}$
$=(11)_2$

Logic Optimization: Full-Adder

- Full-Adder Truth Table:
- Full-Adder K-Map:

the implement for (XOR) gate $X \oplus y \oplus z$

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

S

X \ Y		
	0 (0)	1 (1)
	1 (4)	0 (5)
	1 (7)	0 (6)
	0 (3)	1 (2)

Z

C

X \ Y		
	0 (0)	0 (1)
	0 (4)	1 (5)
	1 (7)	1 (6)
	0 (3)	1 (2)

Z

يجوز / نربط تجميع ال (ones)

odd Function
Then number of ones must be (odd) < 3 or

majority Function :
يعني ال output يكون (1) إذا كان الأكثر من ال (1=input) حيث اثنين منهم على الأقل = (1)

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

- 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:
  - $C = XY + (X \oplus Y)Z$

واحد منهم يكون (1)

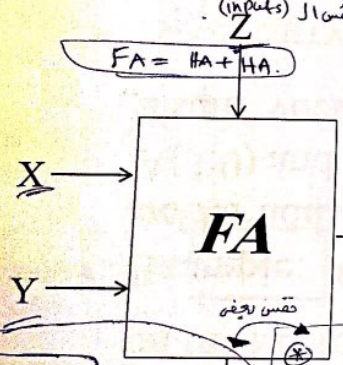
Implementation: Full Adder

Implementation: Full Adder

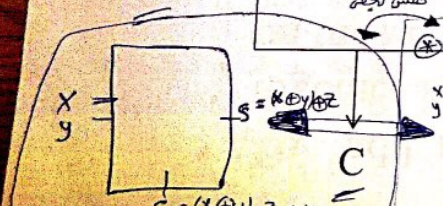
(XOR) gate 2-input . معروف

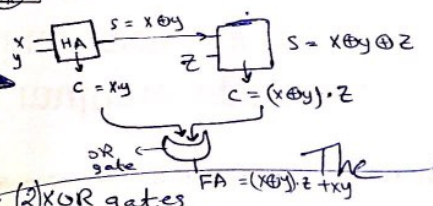
HA : and/ XOR يحتوي
تنقص ال (inputs) .

FA = HA + HA.



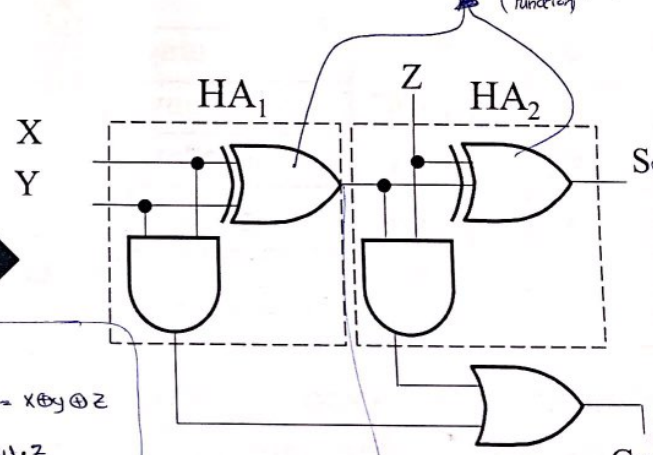
تقص بجهد





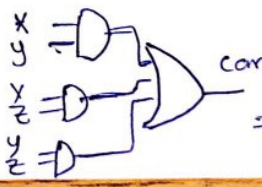
عدد ال (inputs) معروف (٢) الايجوز تغييره.

(2) XOR gates
(3) 2-input and gates
(1) 2-input OR gate.

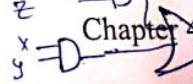


We need 2 XOR gates لأن
ال (XOR) هي (odd function) ولأنه فقط (2-inputs)

The carry would be:



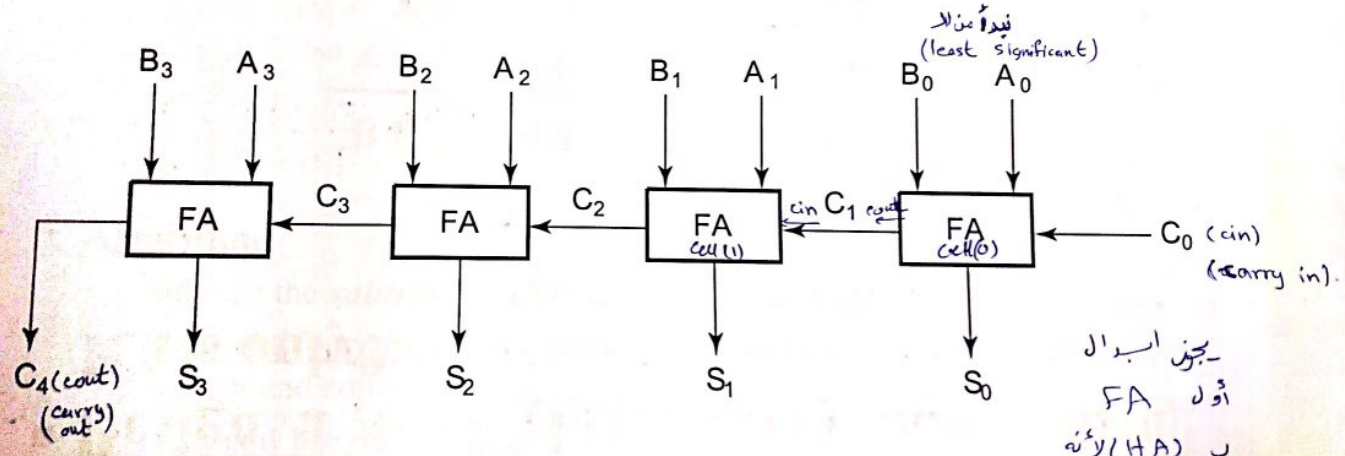
another implementation
as a majority function
يعني (2 input) على الأقل يكونوا (1)



4-bit Ripple-Carry Binary Adder

صورة: كل FA يعتمد على القيمة من الذي قبله. RCA

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



يجوز استبدال أول FA بـ (HA) لأنه الـ (carry) له دائماً = (0) حيث يكون الـ (Adder) هنا صوبح (2 bits only)

Unsigned Subtraction

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

بدل (Sum) بدل (Carry)

(M) ← X	0_0	1_{0}^{10}	0_1	0_1
(N) ← Y	− 0	− 1	− 0	− 1
B D	0 0	1 1	0 1	0 0
		B D	B D	B D

لأننا استعرنا (مُعلّتان) إذاً 1=B

* عندما يكون الناتج يطلع عادي بدون استخدام ال (Borrow) يكون الناتج النهائي صحيح وأن (n<m).

- **Algorithm:**
  - Subtract the ***subtrahend (N)*** from the ***minuend (M)***
  - If **no end borrow occurs**, then M ≥ 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

إذا كان ال borrow = صفر نطرح عليه الرقم الكبير - الصغير

إذا كان ال borrow = 1 نطرح الرقم من ال 2^n $(M-N+2^n) = -(N-M)$ نطرح الكبير - الصغير مع السالب

وزن ال (borrow)

Unsigned Subtraction

$M - N + 2^n$

- **Examples:** $2^n - (M - N + 2^n) = \cancel{2^n} - M + N - \cancel{2^n} = \boxed{N - M}$

لأنها capital
كقيمة صحيح ونضع أمامها
إشارة (−)

نأخذ (borrow) من الخارج

0	1	1	0	1
1001	0100	10011	10010110	01100100
−0111	−0111	−11110	−01100100	−10010110
0010	1101	$\boxed{1}$10101	00110010	11001110
	10000	100000		100000000
	−1101	−10101		−11001110
	(−) 0011	(−) 01011		(−) 00110010

لا يوجد (borrow) خارجية لأن (M>N) إذاً الناتج الذي طلع معنا هو الجواب الصحيح.

2^n = 16

2^n = 32

الناتج − 2^n

$(32)_{10} \Rightarrow (100000)_2$

borrow خارجية. سنأخذ n=5

borrow خارجية مرة أخرى.

for the sign (−) we always ignore the second carry bit.

نعكس الجواب ونضع له سالب

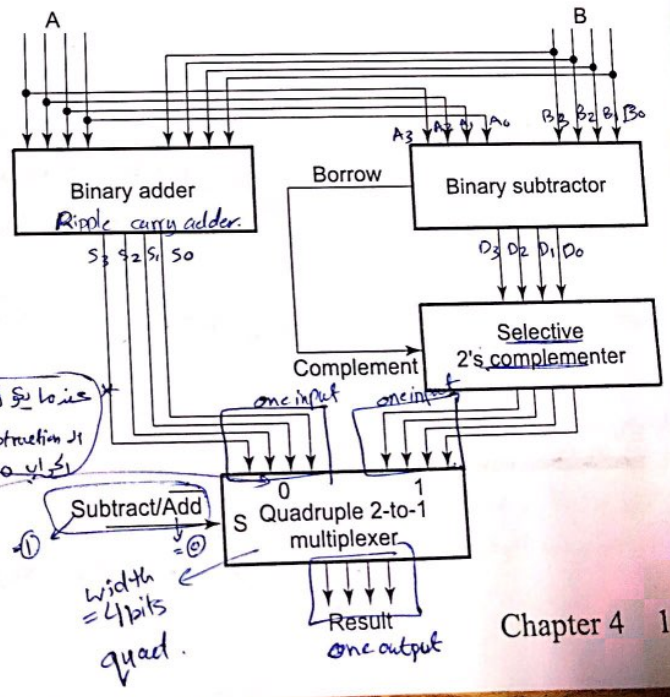
Logic and Computer Design Fundamentals, 4e
PowerPoint® Slides

Unsigned Subtraction (continued)

Unsigned Subtraction (continued)

n: عدد ال digits للعدد (D)

- The subtraction, $2^n - D$, is taking the ***2's complement of D***
- To do both unsigned addition and unsigned subtraction requires: تتطلب:
  - 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



عندما يكون ال select = 1
ال subtraction تعمل وإذا كان
الجواب 0 ال Add

كم تابع شكل
ال MUX
4 bits 4 bits
inputs + output (width 4bits) 5
2 to 1 mux.
4 bits

Complements (2 Types of complements):-

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

Diminished Radix Complement

- If N is a ~~number of n-digits with radix~~ (r), then
  - $N + (r-1)'s\ complement\ of\ N = \underbrace{(r-1)(r-1)(r-1)\ldots(r-1)}_{\text{n-digits}}$

    نعمل ال N

    $(r-1)^n - N$ لدينا عدد (r-1) digit(n)
  - ***The (r – 1)'s complement can be computed by subtracting each digit from (r – 1)***
- Example: Find 1's complement of $(1011)_2$ (binary = base = (2 = r))
  - r = 2, n = 4 (radix, عدد ال bits) → so we know that we need diminished radix complement.
  - Answer is $(2^4 - 1)_{10} - (1011)_2 = (0100)_2$
  - Notice that $(1011)_2 + (0100)_2 = (1111)_2$ which is $\underbrace{(2-1)\,(2-1)\,(2-1)\,(2-1)}_{\text{4-digits}}$

    $(15)_{10} = (1111)_2 - (1011)_2 = (0100)_2 = (4)_{10}$

    هذا الرقم يطلع معنا بال (binary) نحوله لل (decimal)
- 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 $\underbrace{(10-1)\,(10-1)}_{\text{2-digits}}$ → $(10-1)$ قيمة ال digit * عدد ال digits = 99

    * الطريقة السهلة هي $(r-1)(r-1)$ n نأخذ قيمة الرقم المعطى بالسؤال
- 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 $\underbrace{(8-1)\,(8-1)\,(8-1)}_{\text{3-digits}}$

    $n(r-1) \rightarrow$

    $r-1 = 7$, $n = 3$ $(777)_8 - N = (\text{Answer})_8$

    * للـ 1's complement فقط نعكس كل (bit) من 0 ← 1 أو 1 ← 0 فقط بدل ما نوجد كل مرة قيمة الرقم حسب الفرق اللي أخذناها فوق.

Binary 1's Complement

* هي عبارة عن أخذ كل (bit) وعكسها من (0 ← 1) و (1 ← 0) أي نعمل complement لل (digit).

- 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:

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

$(01100111)_2 \rightarrow (1001100)_2$

- 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

- **For number N with n-digit and radix (r):**
  - **If N ≠ 0, r's complement of N = $r^n - N$** القاعدة
  - هي صيغة مختلفة لها ***r's complement = (r-1)'s complement + 1***
  - **If N = 0, r's complement of N = 0**

for the binary number → it is called 2's complement.
for a decimal it's called 10's complement.

- Example: Find 10's complement of $(92)_{10}$
  - r = 10, n = 2
  - 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 (decimal) يطلع ناتجها بال
  - Answer is $16^4 - (3AE7)_{16} = (10000)_{16} - (3AE7)_{16} = (C519)_{16}$ تحوله لل hexadecimal
  - 15's complement = $(C518)_{16}$ → 16's complement = $(C518)_{16} + 1 = (C519)_{16}$

الأصل نجد ال (r-1) complement بالطريقة السهلة ثانية و هي (FFFF) لل 4 digits و نطرح كل رقم عليها (1)

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

r's complement = (r-1)'s complement +1

$r^n - N = r^n - 1 - N + 1$

Binary 2's Complement

لها معاملة (خاصة)

- 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: $2^n - N$.

الطريقة الأسهل:

*نحسبه (2's complement) وذلك بعكس ال bits (1 ← 0, 0 ← 1) ونضيف بالأخير له (1).

$$\begin{array}{r} 100000000 \\ -\ \underline{01110011} \\ 10001101 \end{array}$$

$(10001100)_2 + 1 = (10001101)_2$

- ***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 $(000..00)_2$***

(4bits) $(0000)_2 \rightarrow$ 2's complement

أولاً نحو 1's complement

$(1111)_2 + 1 \rightarrow$

$$\begin{array}{r} 1111 \\ +1 \\ \hline 10000 \end{array}$$

ignore it

تعني عدد ال (4digits) لذلك إذا (N=0) يبقى العدد (0) حتى بال complement.

- ***Complement of a complement restores the number to its original value:***
  - ***The Complement of complement*** $N = \boxed{2^n - (2^n - N) = N}$

$\bar{\bar{N}}$ هو الرقم نفسه.

Alternate 2's Complement Method

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**

- 2's Complement Example:

  10010100

  - **Copy underlined bits:**

    100

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

    01101100

(Logic subtraction) نريد استخدام في عمل (Logic Addition) نفس

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

2's complement لـ(N) — رجعنا لجمع — ترتيب — الرقم الثاني — والرقم الأول

الأعداد مرتبة كبير-صغير وإذا الناتج يكون موجب. (carry عكس borrow)

- If M > N, the ***sum*** produces end carry 2^n which ~~is discarded~~; and from above, M – N remains

عنهما. — الجواب يكون صحيح.

(carry = 1) (borrow = 0)

- 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)

M – N + 2^n → ① نحسب الـ 2's complement → ② ثم نضع إشارة (–) للناتج

الجواب لا يكون صحيح — لازم نحوله

- 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:

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

- Find $01010100_2 - 01000011_2$

عملية طرح عادية. بإستخدام (logic الـ subtraction) 1

عملية طرح بإستخدام الـ (logic Addition)

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

نجمع الـ 2's complement له.

نعم نجمع

نعم عكس التالي لها

الـ carry=1 إذاً الـ borrow=0

ignored

الـ (borrow=1) الخارجية

والـ (carry=0) إذاً M>N والجواب الأول طلع معنا صحيح ولا يحتاج لأي تعديل.

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

حالة (1) في السلايد السابق:

① بإستخدام (logic الـ Addition) نجمع الـ 2's complement الرقم الثاني وتكون الـ carry=1 أي أن الـ borrow=0

② بإستخدام (logic الـ Subtraction) عادي نطرح العدد M (الأكبر) - العدد N (الأصغر)

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

- Find $01000011_2 - 01010100_2$

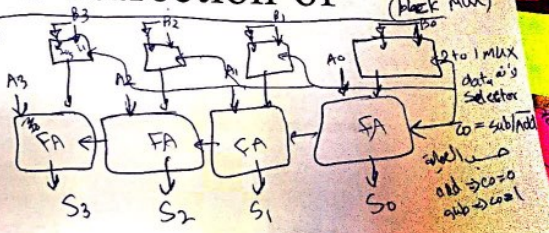
1 = borrow
carry = 0

$$\begin{array}{r} 01000011 \\ -\ 01010100 \end{array} \xrightarrow{\text{2's comp}} \begin{array}{r} 0\ 01000011 \\ +\ 10101100 \\ \hline 11101111 \end{array} \xrightarrow{\text{2's comp}} 00010001$$

carry = 0
borrow = 1

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

- Result = $-(00010001)$



2's Complement Adder/Subtractor for Unsigned Numbers

2's Complement Adder/Subtractor for Unsigned Numbers

ارجع للسلايد السابقة بالأخضر عشان فرايد
Add / Sub يتكون عن (a Ripple carrier) معاها ركة وهدة.

- ***Subtraction can be done by addition of the 2's Complement***
  1. Complement each bit (1's Complement)
  2. Add 1 to the result
- The circuit shown computes A + B and A – B:
- Subtract (S = 1): $A - B = A + (2^n - B) = A + \bar{B} + 1$ (نفس التكملة)
  2's complement = 1's complement + 1
  - The 2's complement of B is formed by using XORs to form the 1's complement and adding the 1 applied to C_0
  - If $C_4 = 1$ ($A \geq B$): correct result
  - If $C_4 = 0$ ($A < B$): result = $2^n - (B - A)$
    - Use 2's complement logic **OR**
    - Use Adder/Subtractor again with:
      - A = 0
      - $B = 2^n - (B - A)$ هي (0-B)
      using the same hardware to implement the 2's comp
- Add (S = 0): A + B
  - B is passed through unchanged

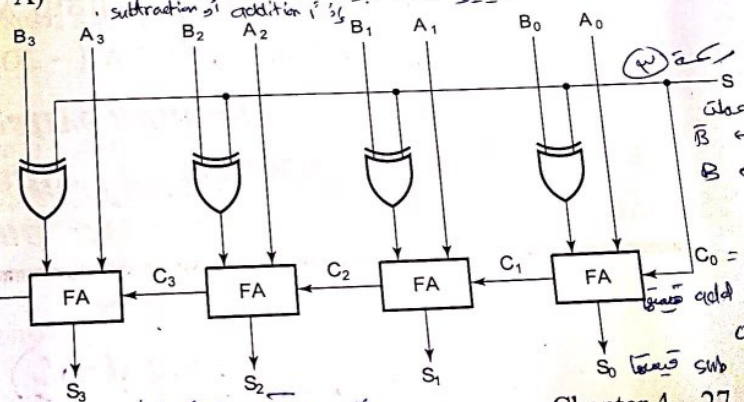
Addition: في حالة ال
ركة ①
$C_0 = 0$
في حالة subtraction
$C_0 = 1$
الاختلافات فقط.

نعمل بعد الركة للناتج الحاكي $S_3S_2S_1S_0$ وتم تنفيذ (بيت) block

$X \oplus 0 = X$ ⊛ تواعد then

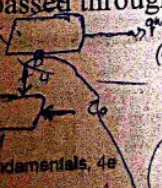
$X \oplus 1 = \bar{X}$

نشبك ال (A) على سلك لحال لأنه دائماً نزل كما هي القيمة فلا يحدث تغيير عليها. بينما ال (B) تتغير حسب العملية التي نعملها إذا addition أو subtraction.



addition: $A+B+C_0$

subtraction: $A+\bar{B}+1$



quad 2 to 1 MUX (4bit)

Zoom!

4bit Ripple Carry Adder

نضيف Add عشان لا يكون معكوس نعمل عليه تجمع يختار ال (block) الذي على اليسار (B3 to B0)

Signed Integers (الأرقام مع الإشارة)

موجبة سالبة

- ***Positive numbers and zero*** can be represented by unsigned ***n-digit, radix r*** numbers. ***We need a representation for negative numbers***
- To represent a sign (+ or –) we need exactly one more bit of information (1 binary digit gives $2^1 = 2$ elements which is exactly what is needed).
- Since computers use binary numbers, by convention, ***the most significant bit is interpreted as a sign bit.***

$$s\ a_{n-2} \ldots a_2 a_1 a_0$$

where:

s = 0 for Positive numbers

s = 1 for Negative numbers

and a_i = 0 or 1 represent the magnitude in some form

Signed Integer Representations

- ***Signed-Magnitude:*** here the (n – 1) digits are interpreted as a positive magnitude
  - Max = + (2^{n-1} – 1) → أكبر رقم موجب
  - Min = – (2^{n-1} – 1) → أصغر رقم سالب
  - Two representation for zero (i.e. ± 0) 0 000 = +0 1 000 = -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

-6 ⇒ 1110 (-6)

بدل له

العدد مطلوب مثلاً (-8) نحول الـ (8) و نأخذ لها (1's comp)

0110 (+6) → we need -6.

(1001) (-6)

تعتبر مشكلة

الحل الأفضل

8 4 2 1
0110 → 2's com → -6 = 1010

→ +6 =

n = 4 bits
+ $2^3 - 1$ = +7
– 2^3 = -8

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

نعكس ال sign bit فقط

نعكس كل ال bits

نعمل 2's complement فقط

- **Represent the number -9 using 8-bits**
  - Sign-Magnitude = 10001001 (0 0 0 0 1 0 0 1) → +9 → نحول bit (sign) السالبة فقط ال most significant bit.
  - 1's complement = 11110110 (نعكس كل bit)
  - 2's complement = 11110111 نعمل 2's com

2's Complement Signed Numbers

2's Complement Signed Numbers

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

عند حساب 2's comp للعدد السالب يصبح موجب.
وكذلك عند حساب 2's comp للعدد الموجب يصبح سالب.

- 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

المنزلة قبل الأخيرة

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

وزن آخر منزلة * آخر منزلة
(سالب) لأنها تكون (1)

قيم مقارن (وزنهم) bits مجموع الـ
الضائعة

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

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

011001
32 16 8 4 2 1
16+8+1 = 25 → نرى قيمته ثم نضع سالب.

Signed-2's Complement Arithmetic

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

- **Subtraction:** $A+(-B)$ عكس اشارته عن طريق ال 2's comp $= A+\bar{B}+1$
  - 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 Addition

(8 bit addition)

```
(+6)   00000110
+      +
(+13)  00001101
       00010011 (+19)
```

نجمع عادي

+19

```
(-6)   11111010
+      +
(+13)  00001101
       100000111 (+7)
```

Carry-out is ignored (نهملها)

لأن العملية addition

```
(+6)   00000110
+      +
(-13)  11110011
       11111001 (-7)
```

نجمع عادي

نحولها لـ 2's comp

الجواب سالب

حسب لها 2's comp لأن الناتج سالب

حسب الفرق السابق

2's comp

(11110011) = -13

+13 → 00001101

```
(-6)   11111010
+      +
(-13)  11110011
       111101101 (-19)
```

Carry-out is ignored

قيمة الجواب السالب حسب الطريقة: 2's comp → نجد 00000111 → (7) ونضع اشارة (-)

or

128 + نجمع القيم الأرقام الموجبة

128 + 64 + 32 + 16 + 8 + 1 = (-7)

Signed 2's Complement Subtraction

```
(+6)   00000110
     +
(+13)  11110011
       11111001 (-7)
```

فرضها هون.

2's comp.

هون جمع فقط.

```
(-6)   11111010
     +
(+13)  11110011
       111101101 (-19)
```

Carry-out is ignored

```
(+6)   00000110
     +
(-13)  00001101
       00010011 (+19)
```

```
(-6)   11111010
     +
(-13)  00001101
       100000111 (+7)
```

Carry-out is ignored

2's Complement Adder/Subtractor for Signed Numbers

- ***Subtraction can be done by addition of the 2's Complement***
  1. Complement each bit (1's Complement)
  2. Add 1 to the result
- The circuit shown computes A + B and A – B:

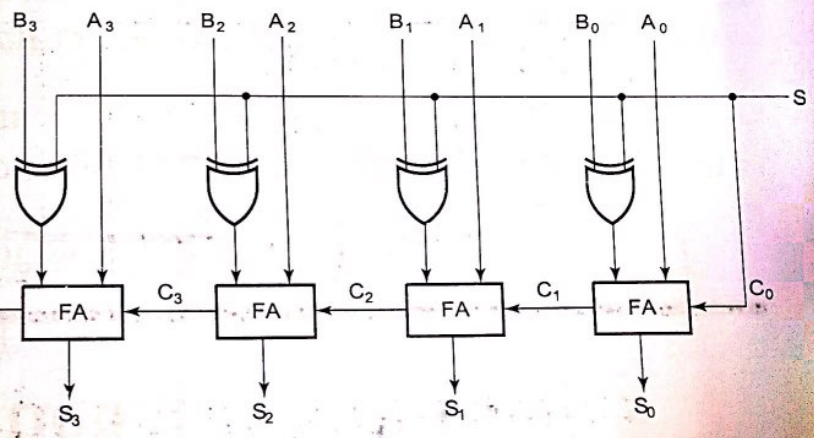
$C_0 = 1$

- Subtract (S = 1): $A - B = A + (2^n - B) = A + \bar{B} + 1$
  - The 2's complement of B is formed by using XORs to form the 1's complement and adding the 1 applied to C_0

$C_0 = 0$

- Add (S = 0): A + B
  - B is passed through unchanged
- ***Same Hardware for Signed and Unsigned numbers***

2's complement



Overflow Detection

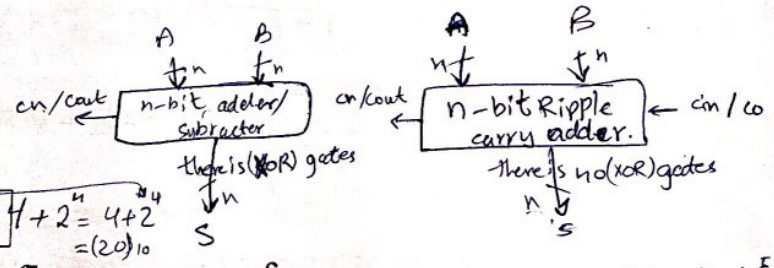
- In computers, the number of bits is fixed (محدد)
- ***Overflow*** occurs if $n + 1$ bits are required to contain the result from an n-bit addition or subtraction
- ***Unsigned number overflow*** is detected from the **end carry-out** when ***adding*** two unsigned numbers

  * حتى الحالة الوحيدة لل unsigned في حالة ال (addition) أن يكون له overflow

  - **Overflow is *impossible* (مستحيل) for unsigned subtraction** لأنه دائماً تصغير للأرقام

(8)	1000	$(8)_{10}$
+	+	
(12)	1100	$(12)_{10}$
	10100 (4)	

(cout) (S)



Carry-out = 1 → Overflow $4 + 2^4 = 4 + 2^4 = (20)_{10}$

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

حيث أن ال (n) bits للناتج (S) لا يكفي لتمثيله بعد عملية الجمع للعددين الأصليين.

Signed-number Overflow Detection

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

(*) تنظر الآخر (digit) فقط:-

0 0 (cout, cin)	0 0	1 1	1 1
0	0	1	1
+ 0	− 1	− 0	+ 1
0 (s)	0	1	1

جمع عددين موجبين < والجواب (الناتج) موجب

إذاً لا يوجد overflow

No overflow X

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

0 1 (cout, cin)	0 1	1 0	1 0
0	0	1	1
+ 0	− 1	− 0	+ 1
1 (s)	1	0	0

اختلاف cin و cout

يعني الـ (overflow=1)

وتمثيله بالـ (XOR gate)

(*) حالتي الطرح اللتين يصير فيهم overflow.

جمع عددين موجبين < والناتج سالب إذاً يوجد (overflow)

overflow ✓

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

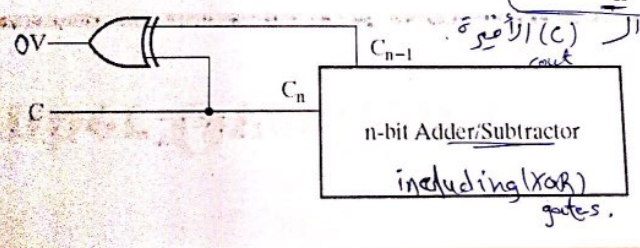
(XOR gate)

الـ (c) الأخيرة

الـ (c) الرافعة عليها قبلها (Cin)

(*) في حالة الـ (unsigned):

الـ $\boxed{\text{cout} = \text{OV}}$



Signed-number Overflow Examples

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

(2's complement)

-2^{n-1} $2^{n-1}-1$

(+70) 01000110 + + (+80) 01010000 10010110 (-106) $V = C_7 \oplus C_8 = 1 \oplus 0 = 1$	(-70) 10111010 + + (-80) 10110000 101101010 (+106) $V = C_7 \oplus C_8 = 0 \oplus 1 = 1$
(+70) 01000110 - + (-80) 01010000 10010110 (-106) $V = C_7 \oplus C_8 = 1 \oplus 0 = 1$	(-70) 10111010 - + (+80) 10110000 101101010 (+106) $V = C_7 \oplus C_8 = 0 \oplus 1 = 1$

Other Arithmetic Functions

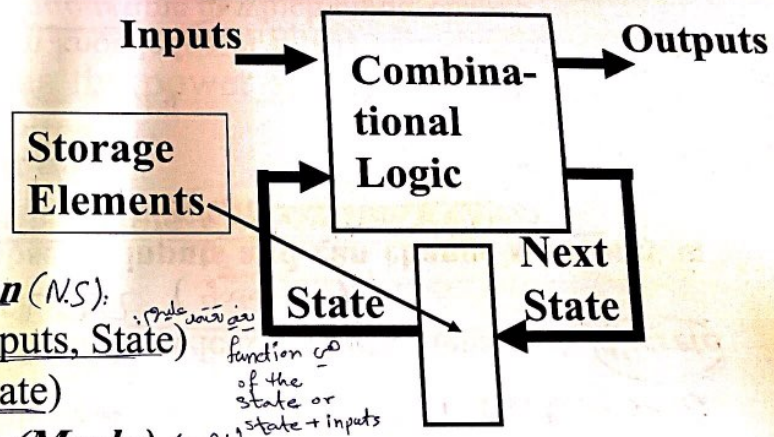
- Incrementing . زيادة بمقدار معين
- Decrementing . نقصان بمقدار معين
- Multiplication by Constant
- Division by Constant
- Zero Fill and Extension

Zero Fill

- ***Zero fill:*** filling an *m*-bit operand with 0s to become an *n*-bit operand with $n > m$
- Filling usually is applied to the MSB end of the operand, but can also be done on the LSB end
- **Example: 11110101 filled to 16 bits**
  - **MSB end: 0000000011110101 <u>(Zero Extension)</u>**
  - **LSB end: 1111010100000000**

(Zeros) الأصفار
على ال
most
significant
digit.
على اليسار

Introduction to Sequential Circuits



- **Combinatorial Logic**
  - ***Next state function*** (N.S):
    ① Next State = f(Inputs, State)
    ② *OR* Next State = f(State)
  - ***Output function (Mealy)*** (o.f.L)
    Outputs = g(Inputs, State)
  - ***Output function (Moore)***
    Outputs = g(State)
- **Output function type depends on specification and affects the design significantly**

يعني تعتمد عليهم:
function هي
of the
state or
state + inputs

منفصلين عن
بعض لا يعتمدوا
على بعض حيث
ممكن لا
output
يكون
f(state)
و لا Next state
f(input, state)

الفرق هو:
لا يشترط
التشابه

Types of Sequential Circuits

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

Flip Flop ← **Synchronous** (clock signal) يعتمد على ال (Storage elements) بوقت معين تغير ال

- Behavior defined from knowledge of its signals at ***discrete*** instances of time (clock signal شكل).
- **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***

latches ← **Asynchronous** لا تلتزم بوقت معين.

- 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***

Storage Elements

latches :-
flip-flops :-

- 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

    التصرف
- Latch:
  - Asynchronous (*) لا تلتزم بتوقيت معين
  - Although difficult to design, we discuss latches first because they are the building blocks for flip-flops

(Flip-flops) made of ~~contain from~~ (2latches).

(simplest)

Basic (NOR) SR Latch using (NOR) gates:

(X) يشبه ال

(signal active high) أي أن ال signal تقوم بوظيفتها عندما تكون ال (input) عاليا (1)

- **Cross-coupling** two NOR gates

حيث أن ال (output) لل (NOR) هو ال (input) لبوابة (NOR) الثانية.

R	S	Q	$\bar{Q}$	Comment
0	0	Q	$\bar{Q}$	Hold, no change
0	1	1	0	Set تخزين قيم (1)
1	0	0	1	Reset تخزين (0)
1	1	0	0	Not allowed (unstable).

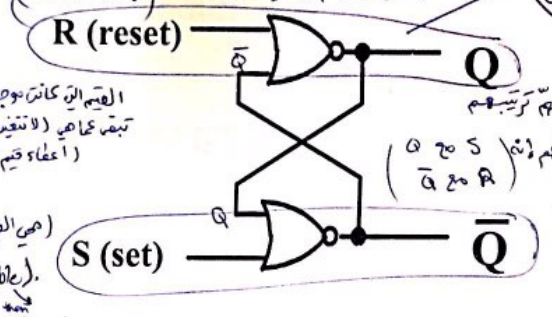
القيم التي كانت بوجود تبقى كما هي (لا تتغير) (اعطاء قيم)

تصفير ال system (هي القيم)

not allowed

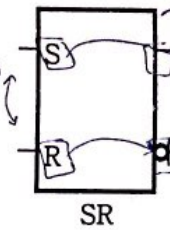
يخالف التعريف لأن $\bar{Q} \neq Q$.

انتبه إيا توا مش في حالة (NOR) SR latch



(logic diagram)

لازم ترتيبهم المهم إنه (Q مع S، $\bar{Q}$ مع R)



(block)

انتبه لشكل ال block!

- **Time sequence behavior:**

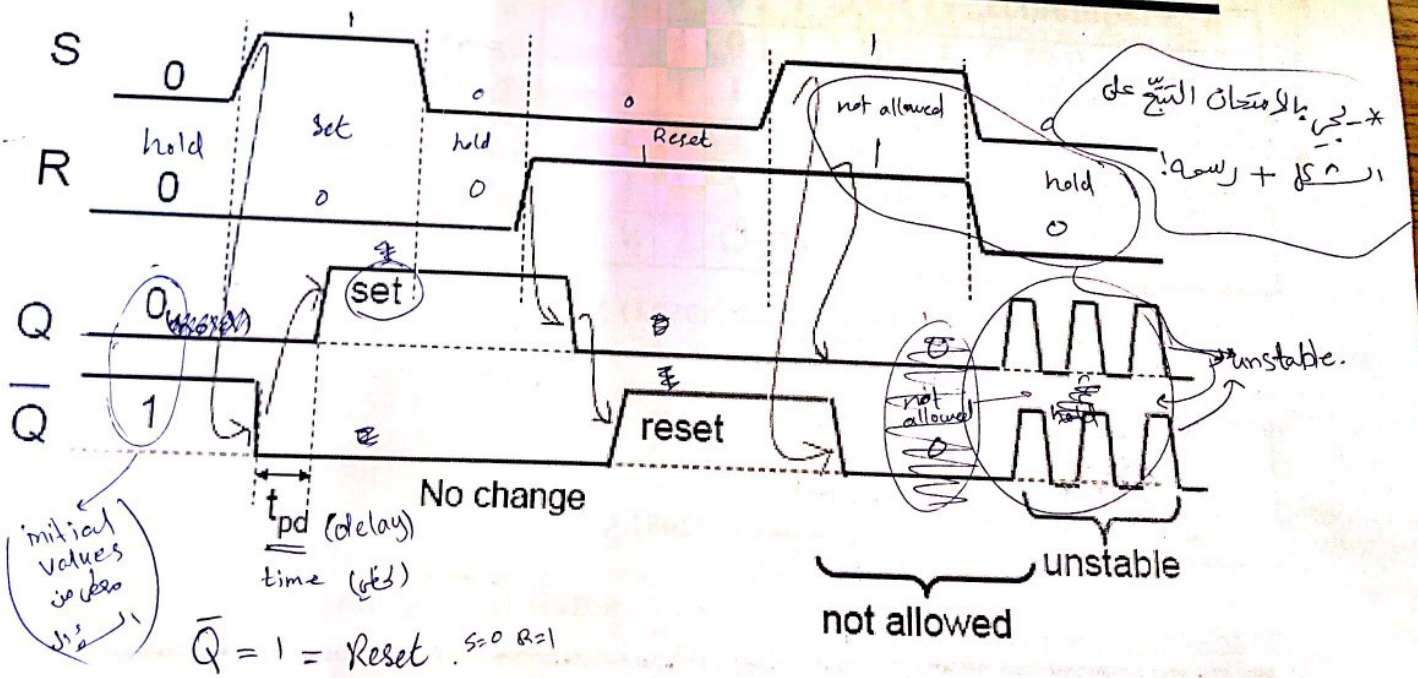
Time

R	S	Q	$\bar{Q}$	Comment	
0	0	?	?	~~Stored state unknown~~	(hold)
0	1	1	0	"Set" Q to 1	
0	0	1	0	Now Q "remembers" 1	(hold)
1	0	0	1	"Reset" Q to 0	
0	0	0	1	Now Q "remembers" 0	(hold)
1	1	0	0	Both go low	
0	0	?	?	Unstable! حالة يطلع وينزل لا يثبت	

- **S = 1, R = 1 is forbidden as input pattern**

Timing Waveform of NOR SR Latch

Timing Waveform of NOR SR Latch



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

$Q = 1 =$ set. S=1 R=0

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

(signal active low) أي أن ال (signal) تعمل بوظيفتها عندما تكون ال (input) عليها (0)

- **Cross-coupling two NAND gates** (bars)
- ***Active low inputs***

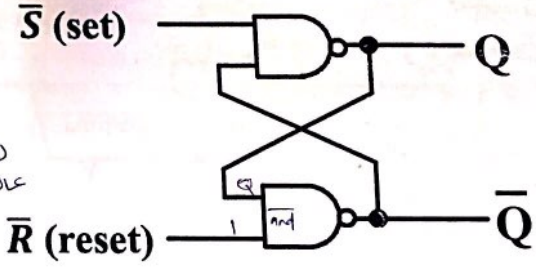
$\overline{R}$	$\overline{S}$	Q	$\overline{Q}$	Comment
0	0	1	1	Not allowed
0	1	0	1	Reset
1	0	1	0	Set
1	1	Q	$\overline{Q}$	Hold, no change

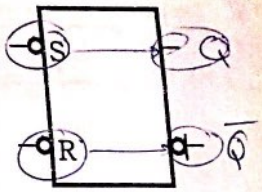
reset (set)

set (reset)

لا نحتاج لنرجع لمرحلة عالسريع $\overline{S}=0$ إذا الأكثيفة هو (S) فيكون set يعني $Q=1$ $\overline{Q}=0$

$\overline{S}=0$ $\overline{R}=1$ نفس الشي $\overline{R}=0$ $\overline{S}=1$ وبكرا $\overline{R}=0$ $\overline{S}=1$ $\overline{S}=0$





- **Time sequence behavior:**

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!

عكس ال (T.T) السابقة.

- **$\overline{S}=0, \overline{R}=0$ is forbidden as**

$\overline{S} \Rightarrow \overline{1}=0$

$\overline{R} \Rightarrow \overline{0}=1$

S R → Reset ($Q=0$, $\overline{Q}=1$)

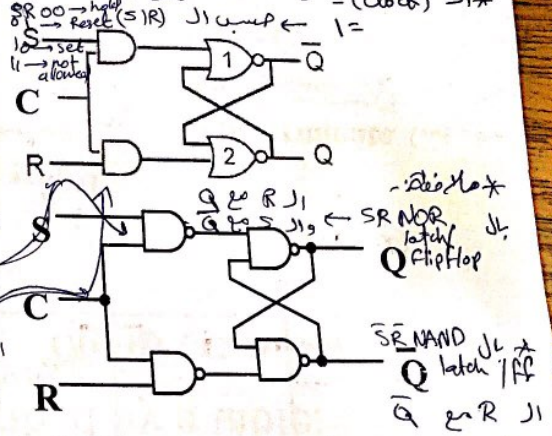
1	1	?	?	Unstable!

Clocked SR Latch (Pulse-triggered Latch)

(SR NANA / SR NOR) تعديل على ال

- 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**

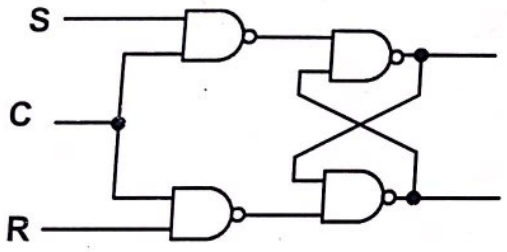
C	R	S	Q	$\overline{Q}$	Comment
0	x	x	Q	$\overline{Q}$	Hold, no change
1	0	0	Q	$\overline{Q}$	Hold, no change
1	0	1	1	0	Set
1	1	0	0	1	Reset
1	1	1	Not allowed		



- 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"

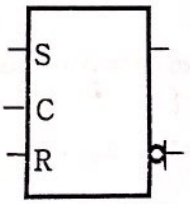
Clocked SR Latch (continued)

- 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 (not allowed)
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)

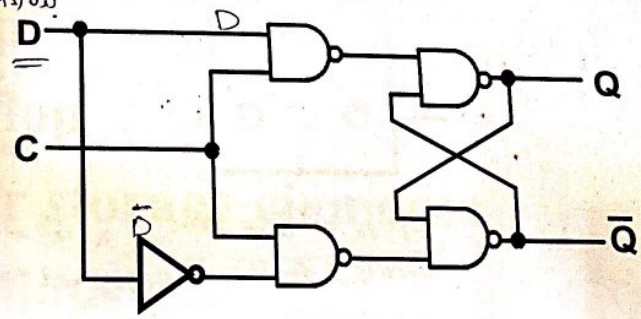


Clocked SR

D Latch (التعديل لحلّ مشكلة ال not allowed)

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

(بدل S_R = 2-inputs)



التخلّص من مشكلة ال (Not allowed)

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

حسب ال clock (0 hold)

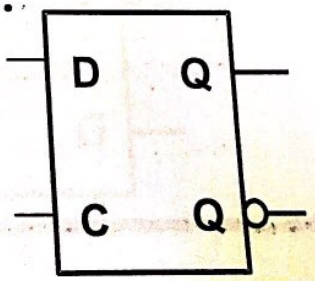
(change) حسب ال D

set ← (D)

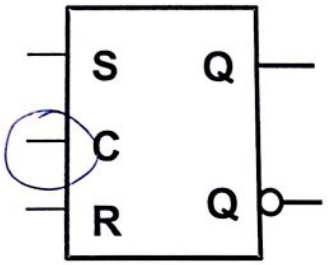
reset ← ($\bar{D}$)

وعلى أنّه ($\bar{D}$/D) يستحيل أن يكونوا (1) مع بعض بنفس الوقت بالتالي تم حلّ مشكلة ال (not allowed).

The graphic symbol for a D Latch is:



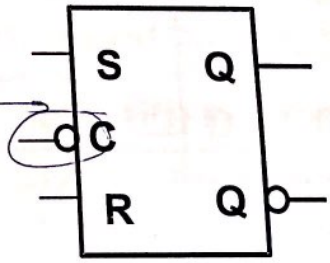
Variations of Clocked SR and D Latches



*الفرق :

(not allowed) يكون

(not allowed) بينما لا يكون



+ve pulse-triggered SR latch

* ***C = 0 → Hold***
* ***C = 1 → Change***

-ve pulse-triggered SR latch

C = 0 → Change
C = 1 → Hold

(clock) وضع inverter على ال

* يعني عندما c=1 يعمل hold بينما c=0 يعمل change

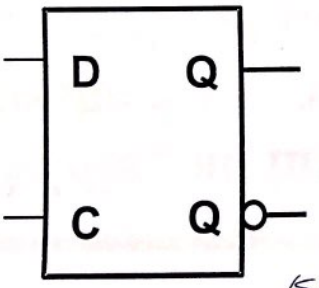
c=1 يعني عندما يعمل change بينما c=0 يعمل hold صعب

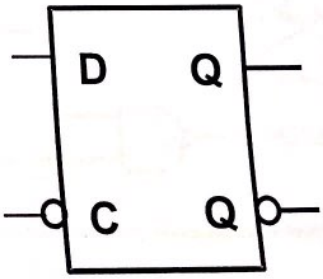
SR 01 (Reset)

SR 10 (set)

SR 10 (set)

SR 01 (reset)





+ve pulse-triggered D latch → بدل (SR)

C=0 hold
C=1 change صعب D

-ve pulse-triggered D latch

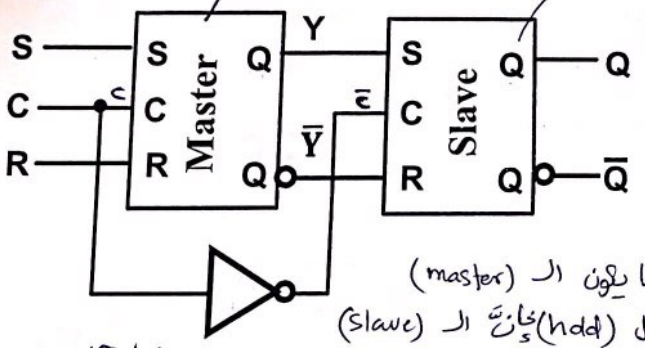
C=0 change → صعب (D)
C=1 hold

SR Master-Slave Flip-Flop

C = 1 · change. → حسب (S/R)
C = 0 · hold.

حسب (S/R)
C = 0. change.
C = 1 · hold.

- Consists of two clocked SR latches in series with the clock on the second latch inverted



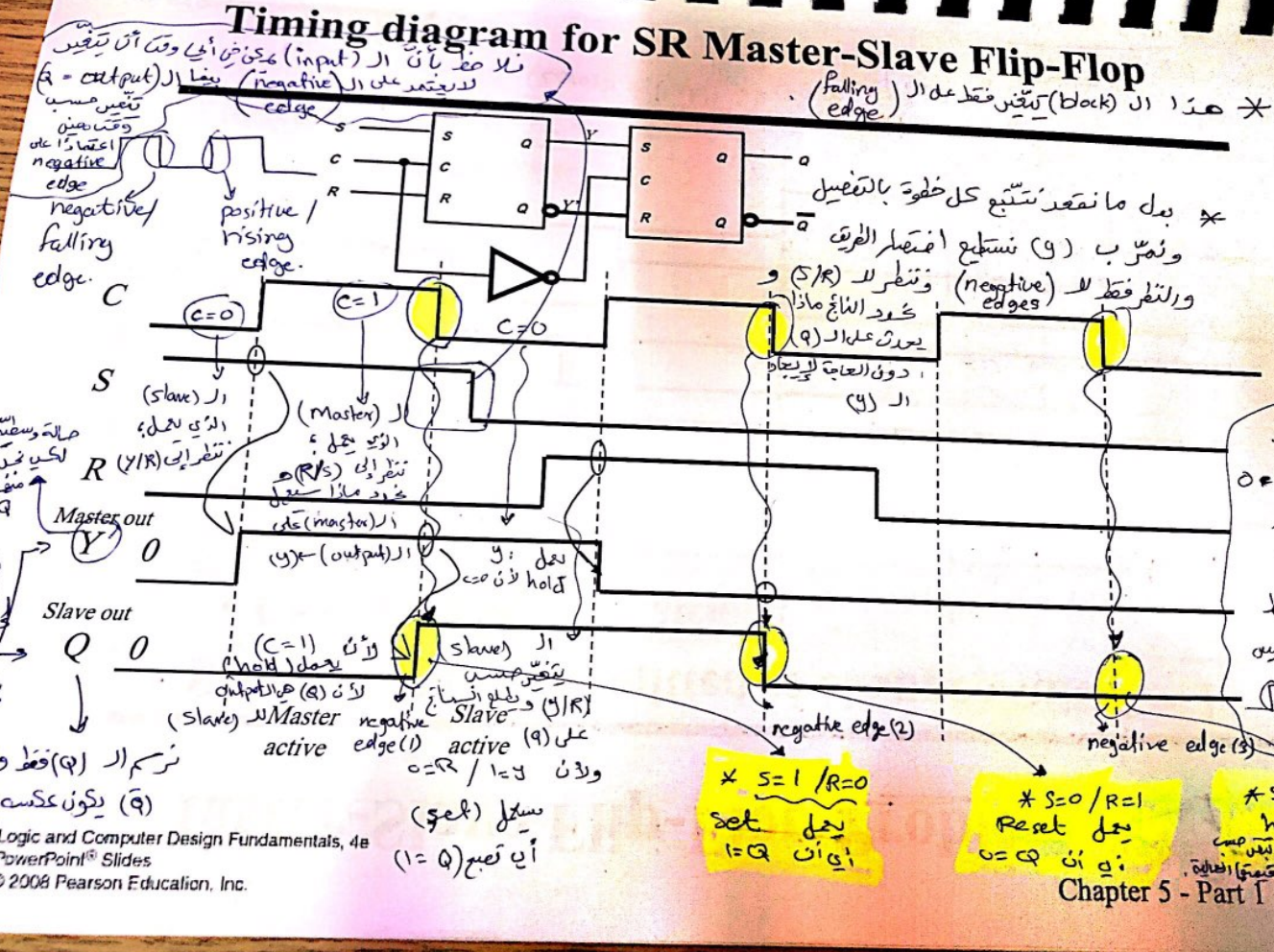
* عندما يكون ال (master) عامل (hold) فإن ال (Slave) يكون هو الشغال ويعمل (change) حسب ال (R/S)

- The input is observed by the first latch with C = 1 → because (C)

- The output is changed by the second latch with C = 0 → because ($\overline{C}$)

- *The path from input to output is broken by the difference in clocking values (C = 1 and C = 0)*

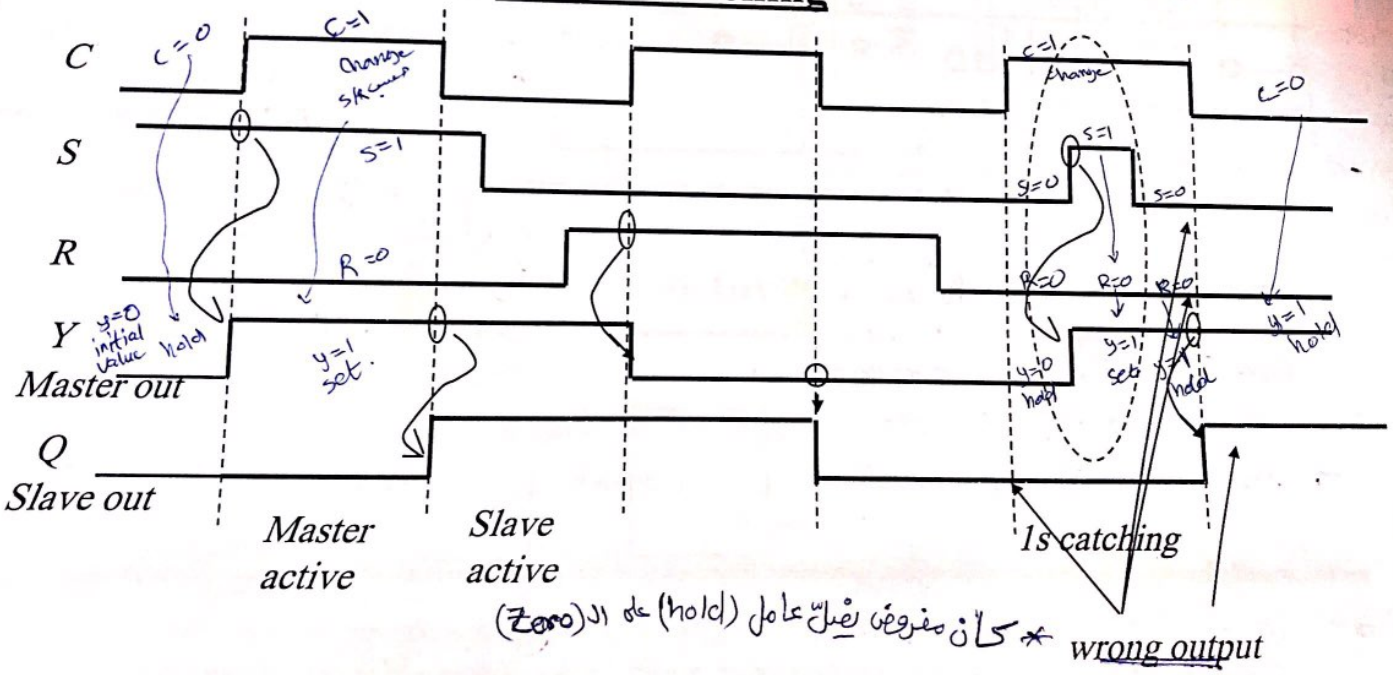
Timing diagram for SR Master-Slave Flip-Flop



Master-Slave Flip-Flop Problem

→ 0 catching
→ 1 catching

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

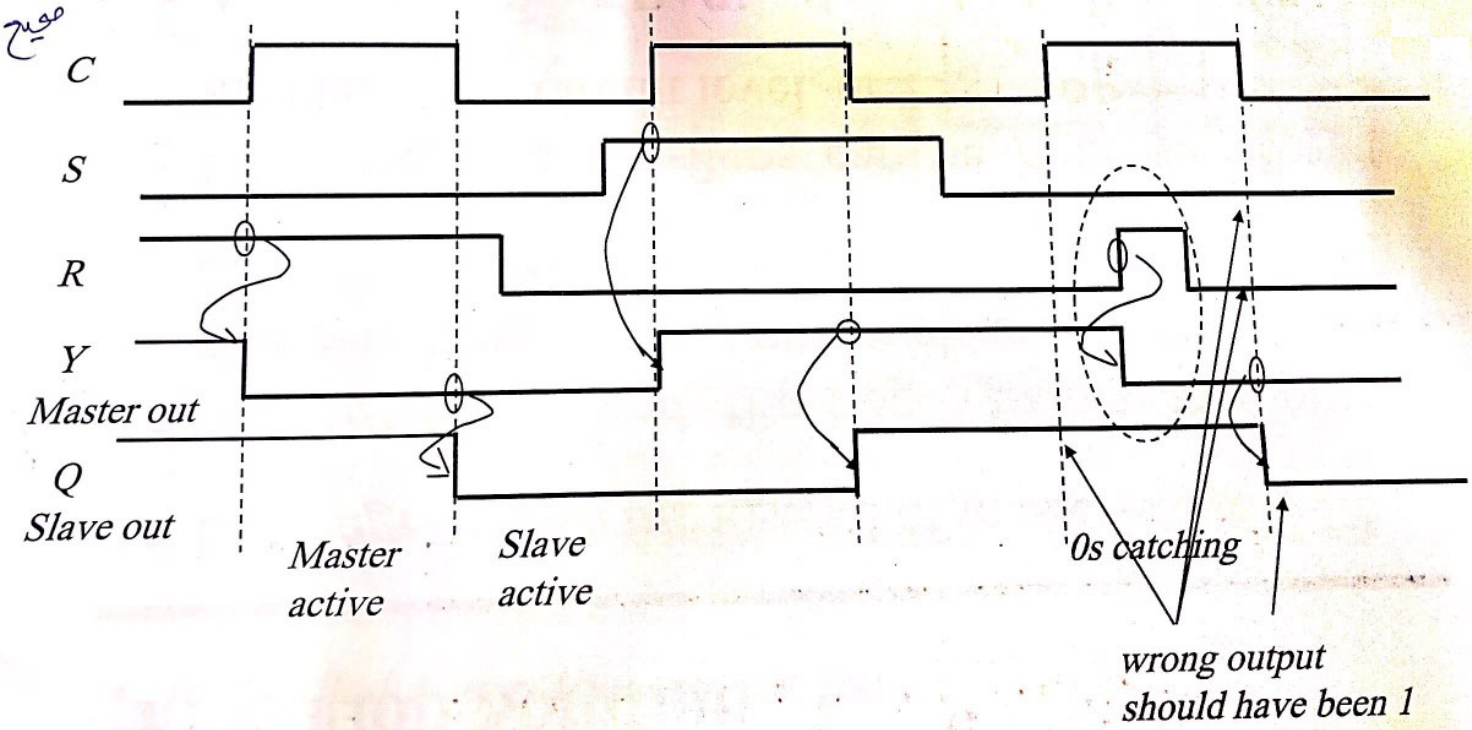


* كأن مفروض نفصل عامل (hold) على الـ (Zero)

wrong output should have been 0

0s Catching

* مش شرط أن يبيّن دائماً أنّه حدث (1/0 catching)، ممكن يحدث الـ(catching) مرّتين ويرجع الناتج النهائي يكون صحيح.



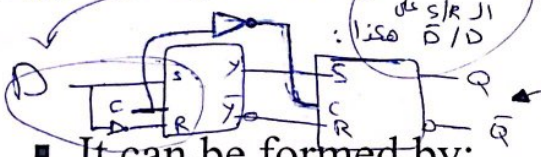
Edge-Triggered D Flip-Flop

الحل لمشكلة الـ (0/1 catching)

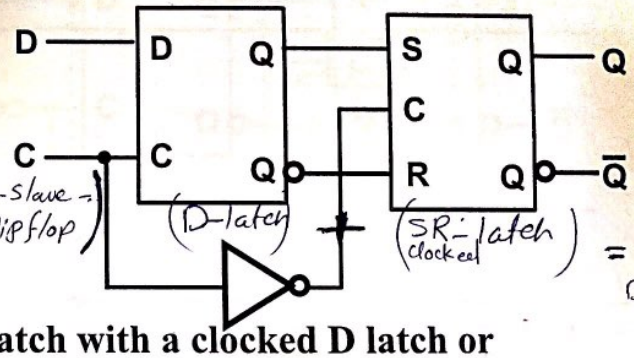
- ***The edge-triggered D flip-flop is the same as the master-slave D flip-flop***

D - latch = SR - latch

ولكن بشوكين ال S/R على D/D هكذا:



(Master-slave D-flipflop)



(D-latch) (SR-latch clocked) = edge triggered D-flipflop.

- It can be formed by:
  - **Replacing the first clocked SR latch with a clocked D latch or**
  - **Adding a D input and inverter to a master-slave SR flip-flop**
- The 1s and 0s catching behaviors are not present with D replacing S and R inputs
- The change of the D flip-flop output is associated with the negative edge at the end of the pulse
- ***It is called a negative-edge triggered flip-flop***

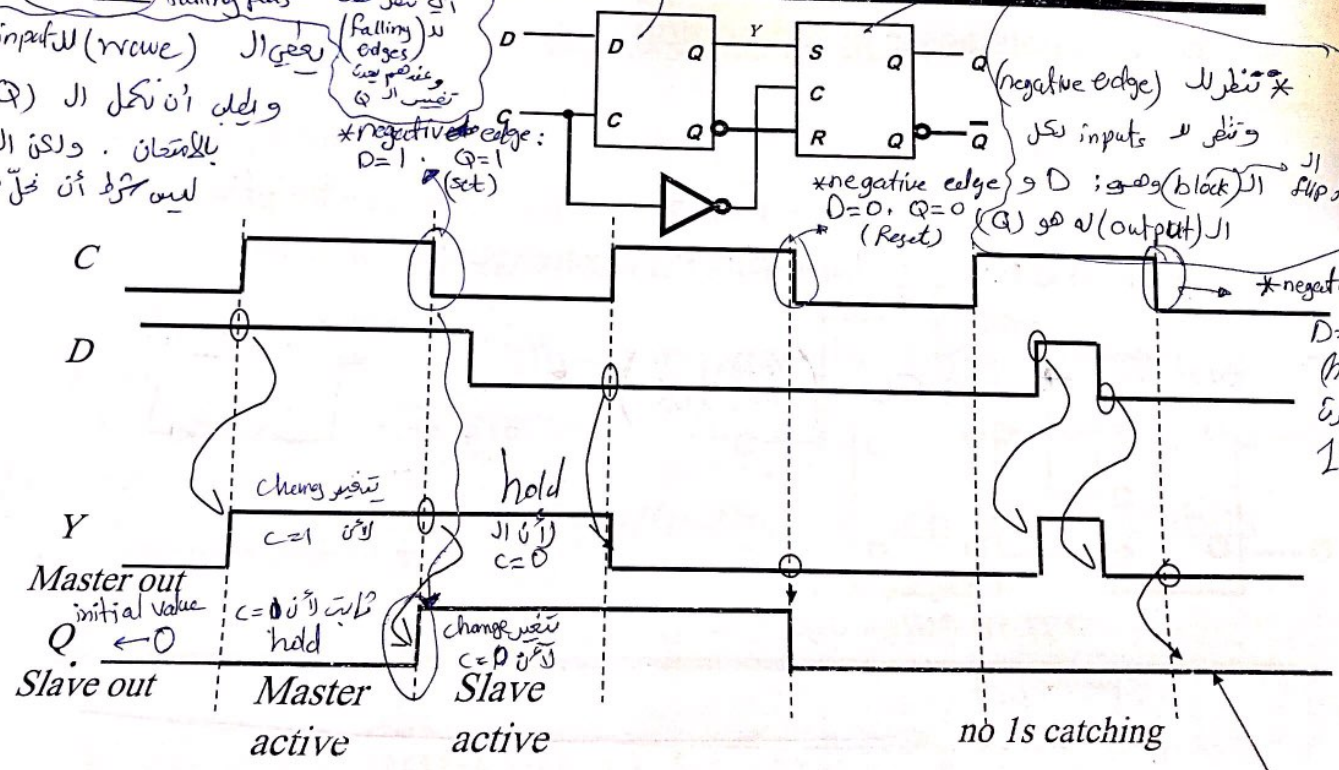
No 1s catching in the edge-triggered D Flip-Flops

* لمعرفة ال (puls) التي يعمل عليها التغيير. تقول ال (master) يعمل على ال (positive) وال (slave) على ال (negative) إذاً ال puls هي

master — slave falling puls → أي تنظر عند ال (falling edges) وعندهم يحدث تغيير ال Q

active at positive puls.

active at negative puls.



* negative edge: D=1 . Q=1 (set)

* negative edge D=0, Q=0 (Reset)

* negative edge D=0 . Q=0 (hold) ولم يحدث 1's catching

* تنظر لل (negative edge) وتنظر لل inputs لكل ال (block) وهو: D و ال (output) له هو (Q) ال flip flop.

يعطي ال input (wave) ويجب أن تكمل ال (Q) بالامتحان . ولكن ال (Y) ليس شرط أن نحل بها

Positive-Edge Triggered D Flip-Flop

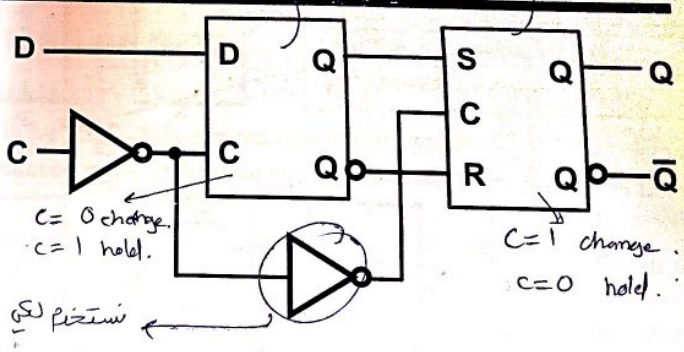
master slave ← ليصبح ال (pals) عالتالي:

active · يجب على negative ال

active يجب على positive ال

- Formed by adding inverter to clock input

عشان يكون ال (master) ← يعمل على clock=0
و ال (slave) يعمل على clock=1



نستخدم لكي نعالج فكرة ال (delay) حيث أنه بذلك يكون لدينا في لحظة على ال (two blockes) = 1 أي يكونوا ال (two blockes) active at the same time.

- Q changes to the value on D applied at the positive clock edge → لأن ال master يعمل على clock=0 وال slave يعمل على clock=1 لذلك:

positive edge triggered — master — slave

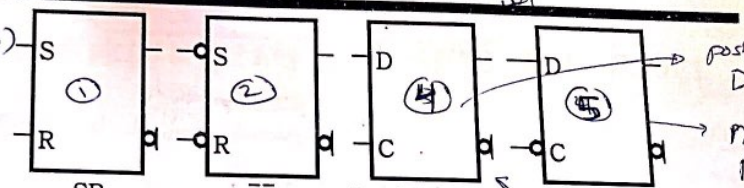
- Our choice as the standard flip-flop for most sequential circuits

Standard Symbols for Storage Elements

(Clocked SR-latch)

- **Latches:** (Asynchronous)

(input) كل ما تغيّر ال
تغيّر القيمة المخزّنة على
ال (output)



SR latch (NOR) — SR latch (NAND) — D with 1 Control (clocked D latch) — D with 0 Control

(a) Latches

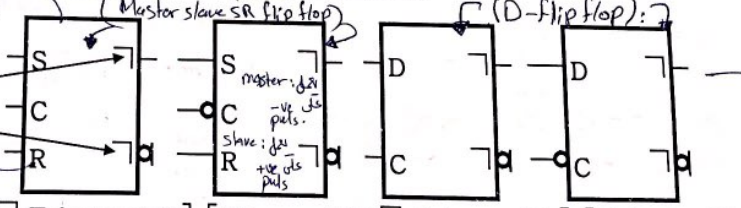
- **Master-Slave:** **Postponed output indicators**

master: يعمل على +ve puls
slave: يعمل على -ve puls

(Master slave SR flip flop) — (D-flip flop):

يدل على أنه يعمل على ال (negative puls) وكان يحدث فيه 0/1 catching.

يدل على أنه (Master-slave SR-flip flop)



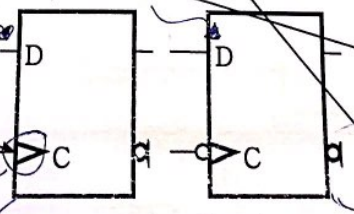
لا يحدث فيه 0/1 catching.

Triggered SR — Triggered SR — Triggered D — Triggered D

(b) Master-Slave Flip-Flops

- **Edge-Triggered:** **Dynamic indicator**

(D-flip flop) تصميم آخر لل



Master active when C = 1
Slave active when C = 0

Master active when C = 0
Slave active when C = 1

يدل على أنه تصميم كوحدة واحدة (separate electronic circuit)

positive edge triggered D-ff

Triggered D — Triggered D

negative edge triggered D-FF.

(c) Edge-Triggered Flip-Flops

positive edge triggered D-ff

Triggered D Triggered D

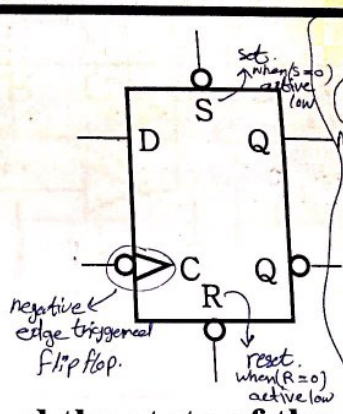
(c) Edge-Triggered Flip-Flops

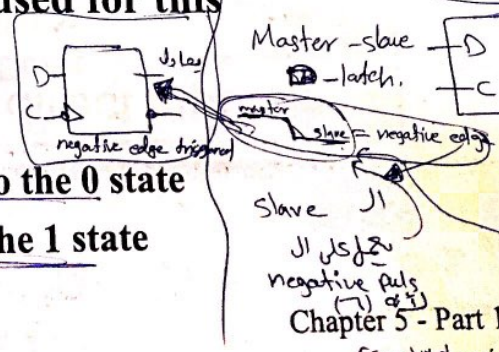
negative edge triggered, D-FF.

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., <u>asynchronously</u>
- 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

يعمل على ال active low ($\bar{S}/\bar{R}$) لو موجود

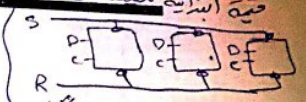




Direct inputs

وظيفتهم يعملوا (initializing) لل (flip flops) يعني يعطوه قيمة ابتدائية

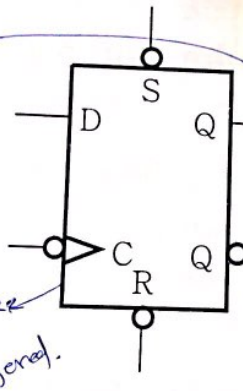
* لو كان لدينا عدد كبير من (f.f) متوصلهم معا مع (R/S) عشان نعطيهم عليهم قيمة ابتدائية معينة كالتالي:



لكن تكون قيمهم الابتدائية جميعاً (0) أي (Q=0) نفتح (R=1, S=0) ⇐ وإذا أردنا كل (f.f) يعمل حسب قيمة (D/C) الخاصة به نفتح (R=S=1)

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

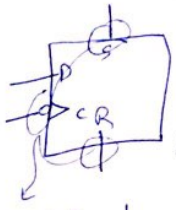
ال (direct inputs) أقوى من ال (clock) طوال ما همّة (active) يكون ال (output) يعتمد عليهم فقط ولو كانوا من active يصير ال output يعتمد على ال (D/C)



negative edge triggered.

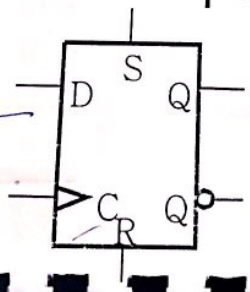
Direct inputs S	R	C	D	Q	Q'	
0 (active)	1	x	x	1	0	(set)
1	0 (active)	x	x	0	1	(Reset)
1	1	↓	0	0	1	(حسب قيمة D)
1 (both not active)	1	↓	1	1	0	

negative edge



ليس شرط يكون مع $\overline{S}/\overline{R}$ negative edge triggered عادي التبديل

- Active high direct inputs:



positive edge triggered.

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

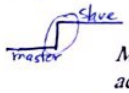
positive edge (clock) ال

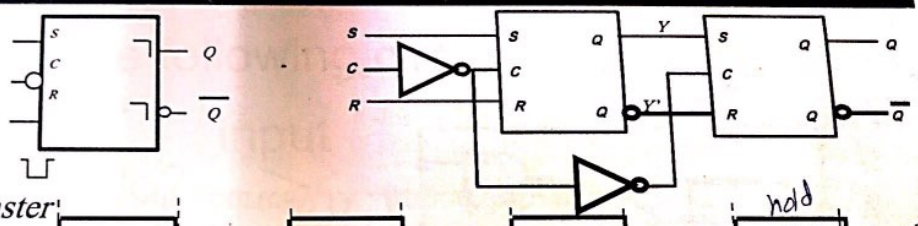
negative edge لازم يكون ال (clock)

ال (latch) يتبع (0 = Q, 1 = D)

Timing diagram of A SR Master-Slave Flip-Flop

لأن على ال (clock) يوجد
(Bar) لذلك نحن ننظر لل
(Positive edge) وذلك حسب :





*- بجي سؤال : هل صدث في هذا ال (form) (Zero's or one's catching) ؟؟ كيف نعرف : بالتالي ننظر في ال (clock) على ال (positive edge) ونقارنه مع ال (Q) و نرى متى تختلف القيمة - يكون عل catch 0/1

Chapter * مذكرة (1) بال

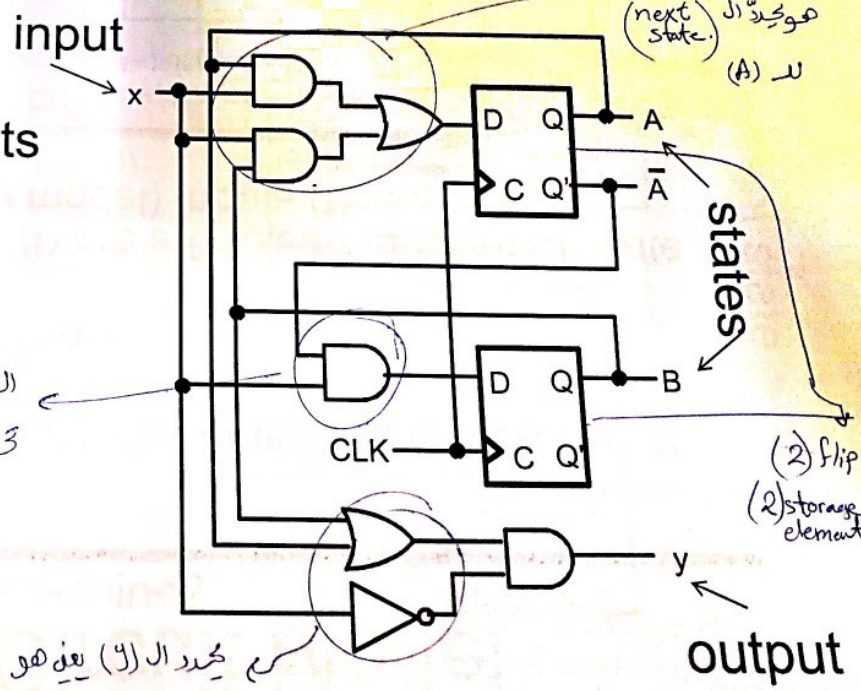
5-4 Sequential Circuit Analysis

التحليل :

- Consider the following circuit:

➢ What does it do?

➢ How do the outputs change when an input arrives?



هذا ال logic diagram هو يحدد ال (next state) لل (A)

ال (and gate) يحدد ال (next state) لل ($\bar{A}$)

(2) flip flops يعني (2) storage elements

يحدد ال (y) يعني هو output logic function.

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).

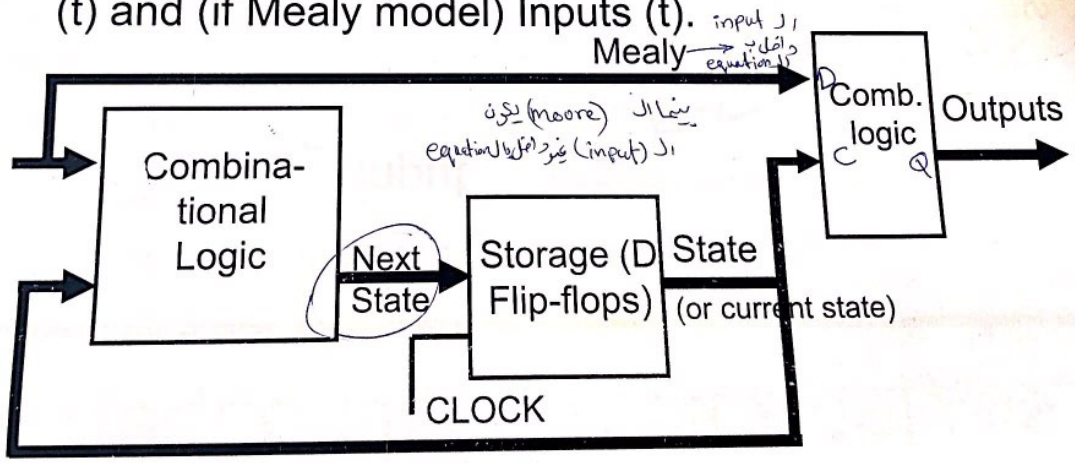
* Q : هي ال current state.

* D : هي ال next state.

* يقسم ال combinational logic إلى قسمين :-

① قسم المسؤول عن ال output (OFL)

② قسم مسؤول عن ال next state (NSL)

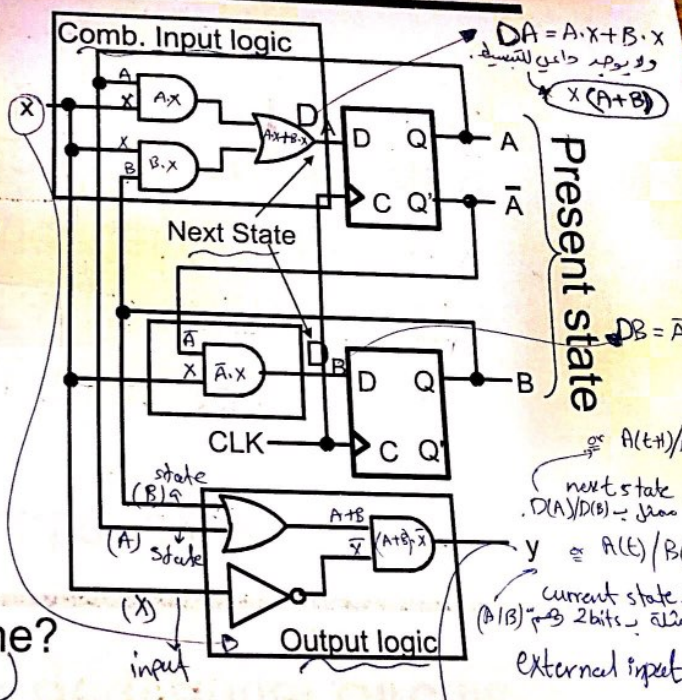


بينما ال (moore) يكون ال (input) غير داخل ال equation

Previous Example (from Fig. 5-15)

Previous Example (from Fig. 5-15)

- Input: X
- Output: Y
- State: Present/Current States (A(t), B(t))
  Example: (AB)= (01), (10)
- Next State:
  $(D_A(t), D_B(t))$
  or
  $= (A(t+1), B(t+1))$



Is this a Moore or Mealy machine?

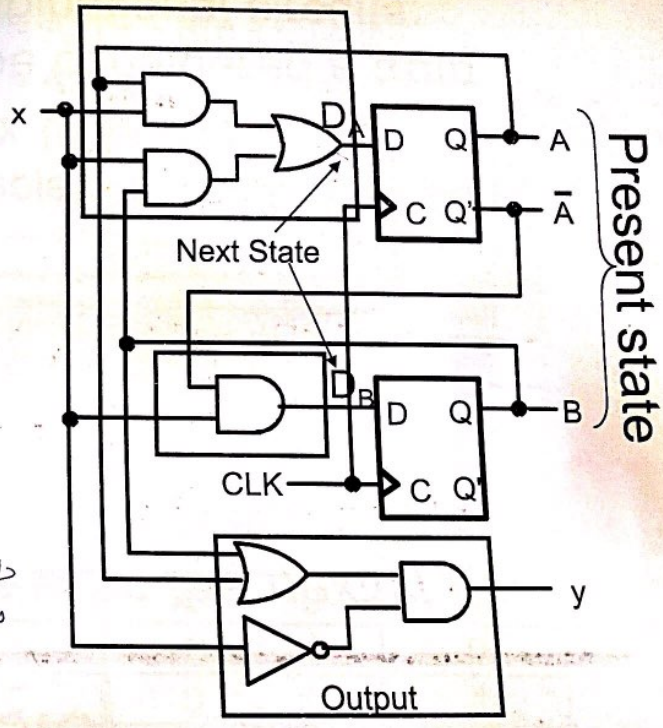
تنظر للـ (Output) ← وهو (y) ← وهو اقتران في ال (mealy) لذلك هو (state وال input)

Step 1: Input and output equations

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

لها عناصر سابقاً.

- Also can be written as
  - $A(t+1) = D_A = A(t)X + B(t)X$
  - $B(t+1) = D_B = \overline{A}(t)X$ دلالات مختلفة
  - $Y = \overline{X}(A(t) + B(t))$ بس نفس المعنى مجموعهم.



Step 2: State Table

يشبه (Truth Table) ولكن يختلف الترتيب شوي

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

حفظ الترتيب (مهم).

Inputs of the combinational circuit		Outputs of the table	
Present State	*Input*	Next State	Output

part I — part II

- 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*

مثل في ال Truth Table

State Table For The Example

and the outputs are *Next State* and *Output*

State Table For The Example

- For the example: $A(t+1) = A(t)\,x + B(t)\,x$

  $B(t+1) = A'(t)\,x$

  $Y(t) = X'\,(B(t) + A(t))$

* نلاحظ أنه كل (row) يعطي transsestion من حالة لحالة أخرى أي نتنقل من state 00 إلى state 01 وهكذا
و يجوز أن نبقى بال state نفسها مش شرط الانتقال

Inputs of the table | Outputs of the table

Present State		Input	Next State		Output
A(t)	B(t)	X	A(t+1)	B(t+1)	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

كينية قراءة ال (row)
نقول لو كنا في ال current state العالية وهي (0,0) وجاء على ال input ال (x) (0) وتكون ال nextstate (0,0) أي نبقى بنفس ال state و الجواب أي ال output يكون = (0).

جميع الاحتمالات نعبيها عادي مثل (T,T) $2^3 = \frac{8}{2} = \frac{4}{2} = 2$

نجدهم من خلال ال equation التي كتبناهم بالتعويض بقيم ال inputs وهكذا

e State Table

(مختصر)
expantion in two dimensions.

1-dimensional table) can become quite lengthy
(m=no. of flip-flops; n=no. of inputs)

2-dimensional table has the present state in the
row

لازم نعرف كيف نقرأ من المعلومات
-جي بالاختبار بس موشرط نحل عليه

* ج: m=3 . n=2 . → use the Alternate state table and the state table to make it (implement it):

m: state elements : A, B, C.
n: inputs : x, y.
Outputs: F, G.

*One dimensional state table

A B C	x y	D(A) D(B) D(C) F G
0 0 0	0 0	
0 0 0	0 1	
⋮	⋮	
1 1 1	1 1	

$2^5 = 32$ rows

ونجي الجدول من اليمين حسب ال (equations) من اليسار

* Two dimensional state table :-

inputs A B C	next state D(A) D(B) D(C) x=0 y=0	x=1 y=0	x=0 y=1	x=1 y=1

$2^3 = 8$ rows

وبالمثال زاد عدد ال (inputs)

dimension (2)

A(t) B(t)	A(t+1) B(t+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

وهي (00) وهاد ال input=1 نركب ل state (01) و لو كان ال x=0 نركب ل state (00) وهكذا

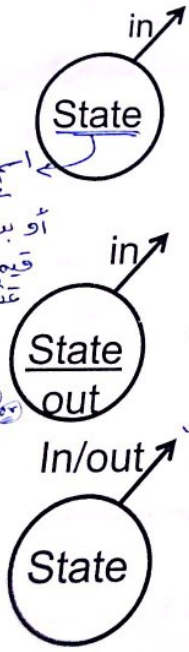
→ next state x=0 عندما

output x=0 عندما

Step 3: State Diagrams

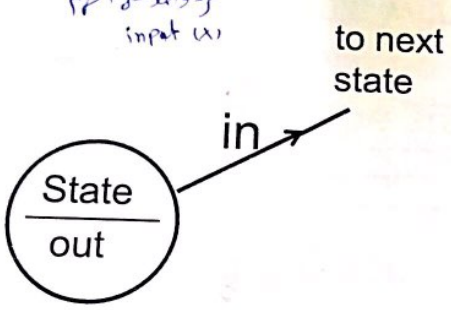
بعد ال State table.

- 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.

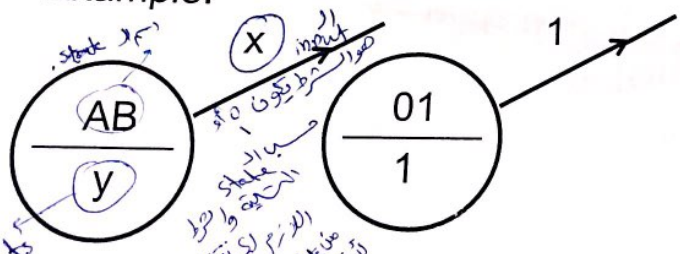


State Diagram Convention

Moore Machine:

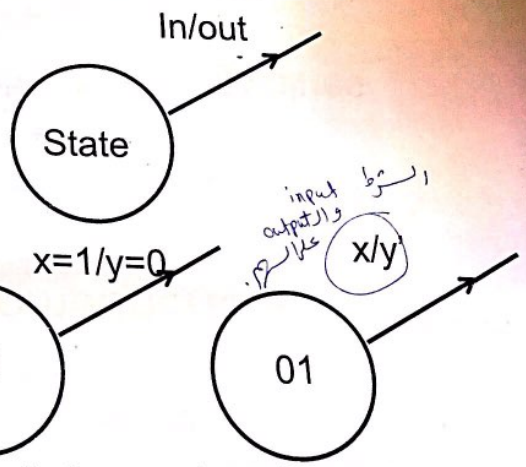


Example:



Moore type output depends only on state

Mealy Machine:



Mealy type output depends on state and input

State Diagram For The Example

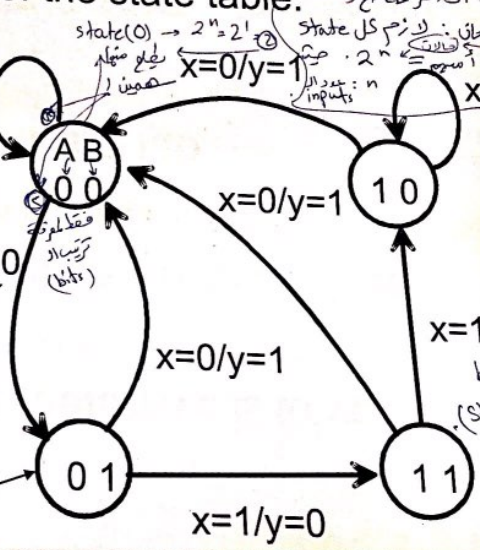
- Graphical representation of the state table:

2^n = outgoing arrows.

* لكي نعرف أن الرسمة صح وكاملة
جمع الرسم بالامتحان: لازم كل state
يطلع منها عدد أسهم = 2^n . حيث
عدد الـ inputs : n

state(0) → $2^n = 2^1$ ②
يطلع منها
سهمين

* ال next state و ال output
يعتمدوا على ال state
وعلى ال input :



state	—
S_0	00
S_1	01
S_2	10
S_3	11

يُفضّل أن يكون ال (index) هو نفسه الرقم ال binary في حالة تسمية ال (state).

* a Mealy design: لأن ال output يعتمد على ال input.

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

لديهم احتمالات 4 لأنك (4 states)

* كيف عرفنا أنو (mealy) لأن ال output يتغير مع تغير ال (input)

* فائدة ال (diagram) هو تسهيل قراءة ال (system) فقط.

Example2

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

* inputs : x, y. (2)
* outputs : Z . (1)

it's fine to have the output as same as the current state.

A = Z → current state = output.

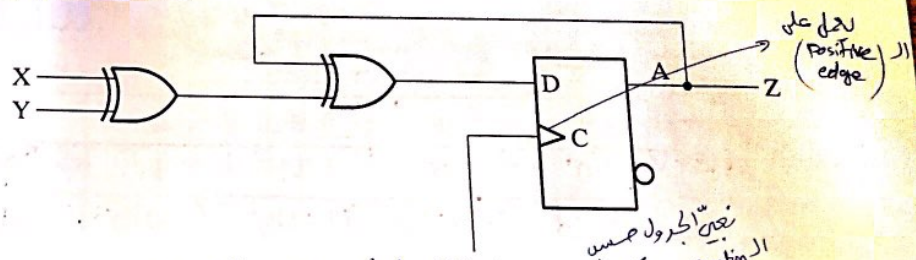
* the state elements : A. (→ number of flip flops we have.)
* the next state : D(A).

→ Writing the Equations:-

① input equation: $D(A) = x \oplus y \oplus A$. (next state)

② output equation: $Z = A$ → a $\boxed{\text{moore design}}$

لأن ال output هو function of the state لا يعتمد على ال inputs (x و y)



نعبي الجدول حسب ال equation

* One dimension state table:-

present state A	inputs x, y	next state D(A)	output Z	Z = A
0	0 0	0	0	نفس القيمة للـ A
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	
	odd function	$x \oplus y \oplus A$		

عدد ال (1) فردي

Example2 Cont.

- State Table:

$2^2 = 4$ احتمالات columns للجدول

* لـ (XOR) عدد ال ones فردي الجواب (1)
* عدد ال ones زوجي الجواب (0)

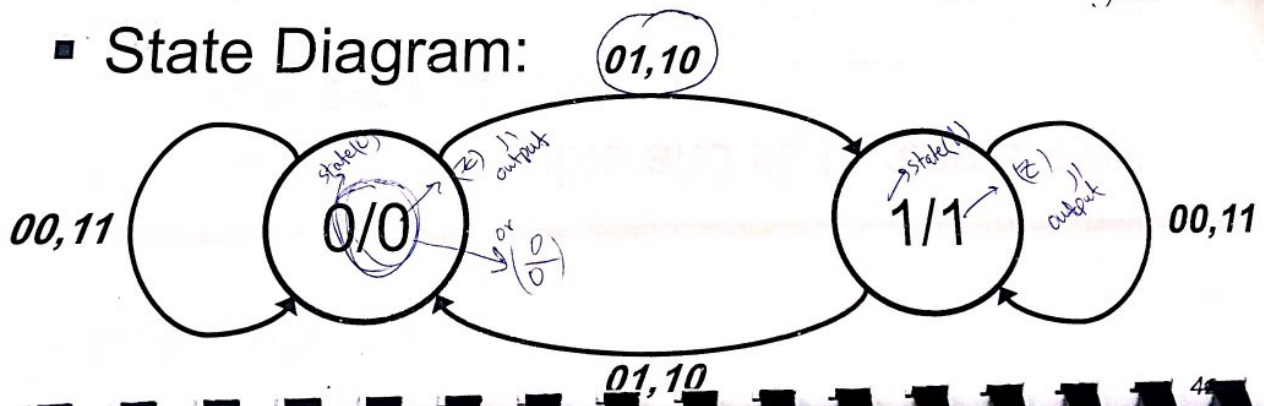
$2^1 = 2$ احتمالات

Preset State A(t)	Next State XY = 00 A(t+1)	XY = 01 A(t+1)	XY = 10 A(t+1)	XY = 11 A(t+1)	Z
0	0	1	1	0	0
1	1	0	0	1	1

current state — next state — output

* ال output فقط يعتمد على ال A(t) لذلك لا يعتمد على ال inputs (xy) يكفيه عمود واحد.

- State Diagram:



Example3

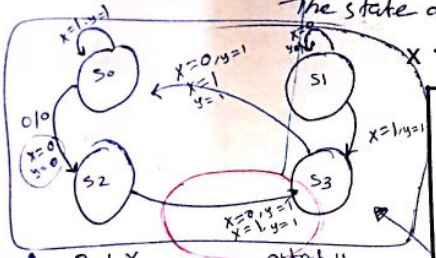
* للفصل بين input و output المكتوبين على السهم نضع (backslash)

بدل ما نرسم (2 arrows) ولا داعي لكتابة X=0 , X=1

* للفصل بين ال 2 conditions على السهم نضع (,) فقط لأن X = don't care فقط y=1 أي سنذهب لل state ③

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

The state diagram:-



* inputs: x
* state elements: $A, \bar{A}, B, \bar{B}$.
* output: y.

* Equations:- $y = \overline{\bar{A} \cdot \bar{B} \cdot \bar{x}} = A + B + X$ → output ال

inputs ال (X) يعتمد على

so it is a mealy design

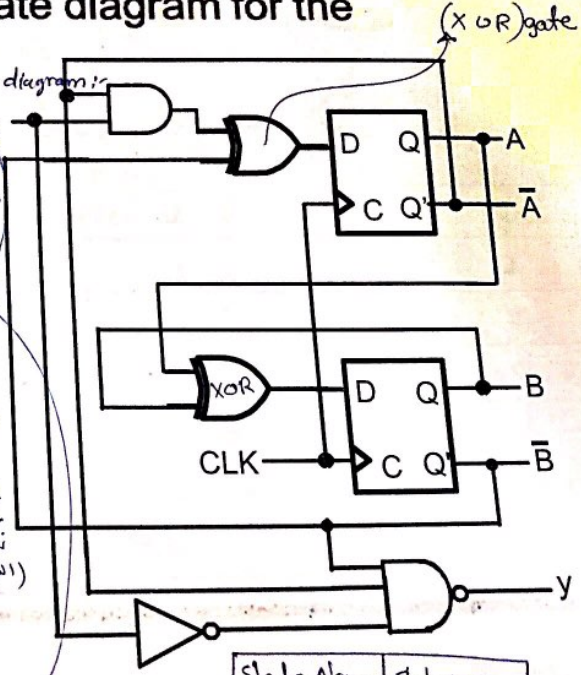
flip flop ① : $D(A) = (\bar{A} \cdot X) \oplus \bar{B}$.

flip flop ② : $D(B) = B \oplus A$

(من المعادلات)

Present state		inputs	Next state		outputs
A	B	X	D(A)	D(B)	Y
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	0	0	1
1	1	0	0	0	1
1	1	1			

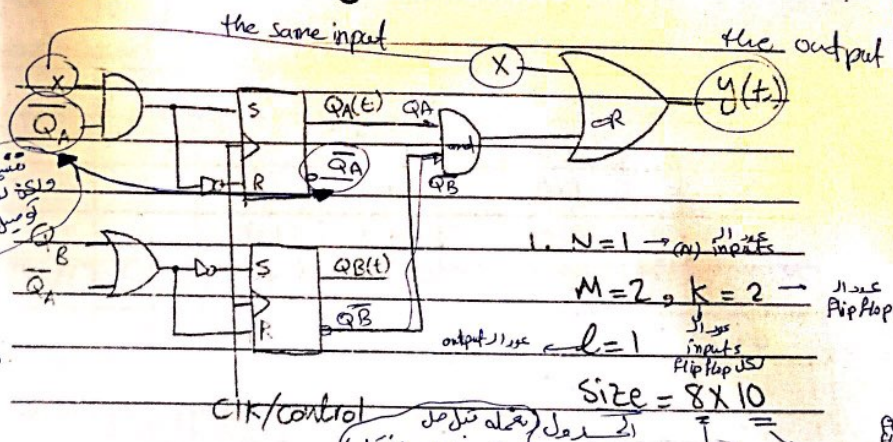
لا يوجد output بالرسمة لل state (الأولى)



State Name	State value
S0	00
S1	01
S2	10
S3	11

Example4 [we will use an SR(Master Slave flip flops)]

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



* نلاحظ أنّ الـ(inputs) واحدة هنا
عكس بعض بوابة موجود عليها inverter موجودة
لا لذلك سيستعمل هنا مدخل مشكلة
(0/1) catching

تشبيها
ولكن لكي لا يصبح
وسيط الإشارات معقد

1. N=1 → عدد الـ inputs (A)

M=2 , K=2 → عدد الـ Flip flops (A)

L=1 → عدد الـ output

عدد الـ inputs لكل Flip flop

Size = 8X10

الجدول (نعمله قبل حل السؤال بالاختبار)

(rows) عدد الـ 2^{n+m}

$2^3 = 8$

$2^m + N + L + 2$ (Flip flops)

$2^2 + 4 + 1 + 1 = 4 + 4 + 2 = 10$

* Write the inputs/outputs equation for this sequential circuit:-

* لكل (flip flop) نلاحظ وجود (2-input equations)

* The input Equations:-

وهم S/R :-

$S(A) = X \cdot \overline{QA}$

$R(A) = \overline{X} + QA$ → جاي من $\overline{X \cdot \overline{QA}} = \overline{X} + QA$

$S(B) = \overline{QB + \overline{Q}A} = \overline{QB} \cdot QA$

$R(B) = QB + \overline{QA}$

* The output Equations:

$y(t) = y = (QA \cdot \overline{QB}) + X$

(Mealy design)

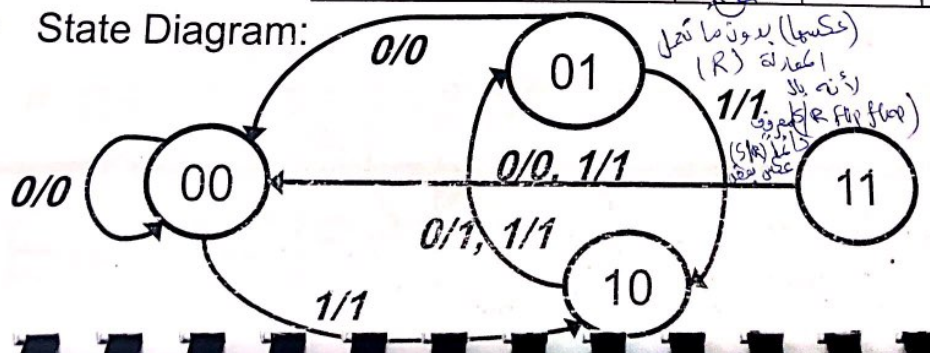
لأنه يوجد input بمعادلة الـ output

Example4 Cont.

- 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	0 0	0
0 0	1	1 0	0 1	1 0	1
0 1	0	0 1	0 1	0 0	0
0 1	1	1 0	0 1	1 0	1
1 0	0	0 1	1 0	0 1	1
1 0	1	0 1	1 0	0 1	1
1 1	0	0 1	0 1	0 0	0
1 1	1	0 1	0 1	0 0	1

- State Diagram:



Exercise: Derive the state diagram of the following Circuit (The last Example - important).

- Logic Diagram:

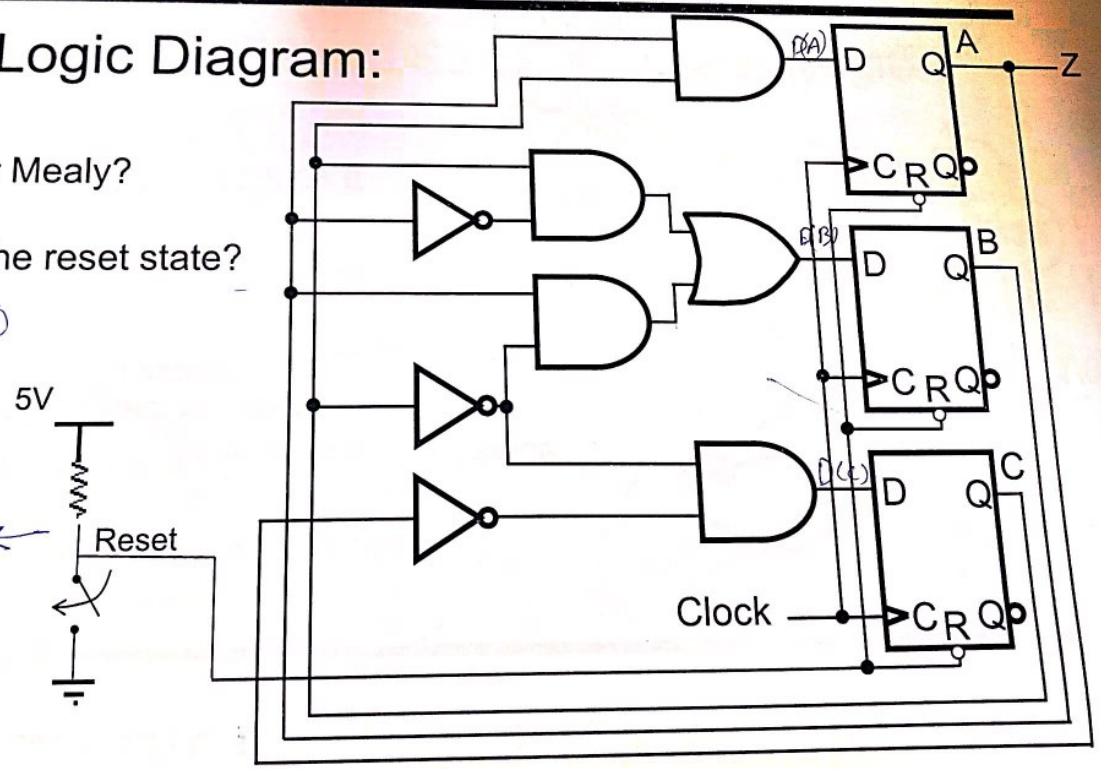
Moore or Mealy?

What is the reset state?

(*) نغلق ال (switch) فيعطي قيمة ابتدائية لكل ff وهي صفر ثم نفتح ال (switch) فيبدأ يعمل كل مسب ال (ff) الخاص به

مشبوكين عليه ال (Reset)

ويعمل على ال (active low)



Step1: Flip-Flop Input Equations

- Variables
  - Inputs: None
  - Outputs: Z
  - State Variables: A, B, C
- Initialization: Reset to (0,0,0)
- Equations
  - $A(t+1) = BC$ $\qquad Z = A$
  - $B(t+1) = B'C + BC' = B \oplus C$
  - $C(t+1) = A'C'$

* كتابة الـ(Equations) لـ (inputs) :

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

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

* output Eqn. : $Z = A$ (Moore)
لا يعتمد على الـ (inputs).

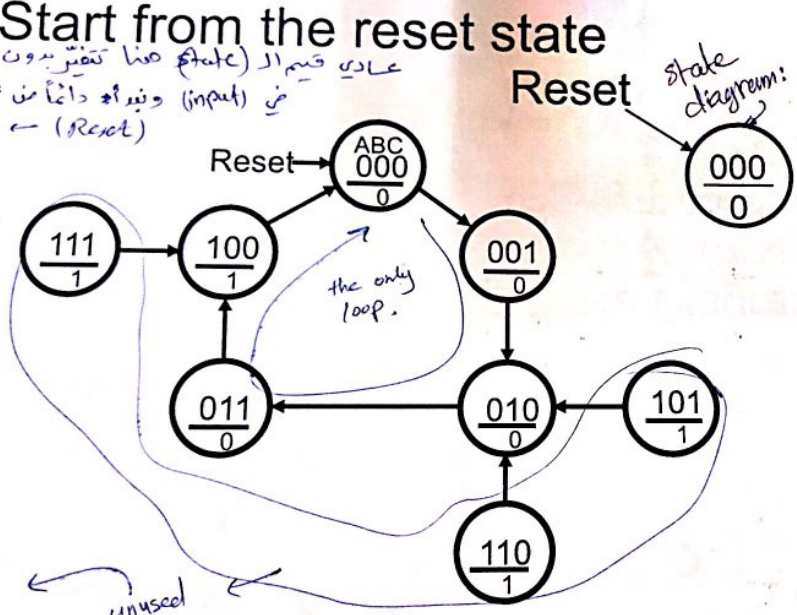
* ملاحظة : لا يوجد أي external input
لهذه الجدول

Step 3: State Diagram

(state table)

Start from the reset state

عادي قيم ال (state) هنا تتغير بدون ما يكون في (input) ونبدأ دائماً من عند ال (Reset) ← initial state ← (000) عندما يكون ال Reset active نبدأ من (state 000)



state diagram:

A B C	$A^+B^+C^+$	Z
0 0 0	0 0 1	0
0 0 1	0 1 0	0
0 1 0	0 1 1	0
0 1 1	1 0 0	0
1 0 0	0 0 0	1
1 0 1	0 1 0	1
1 1 0	0 1 0	1
1 1 1	1 0 0	1

the system won't use them in the normal path but if something went wrong they will be used.

- Are all states used? Which ones?

* فكرة (٣) بال chapter

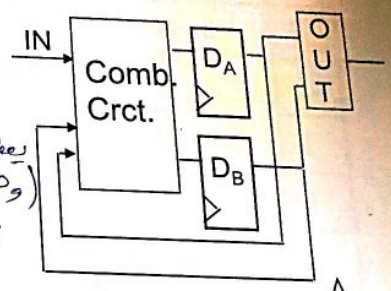
5-5 Sequential Circuit Design

Design procedure

?

- Word description (يعطيني وصف)
- State Diagram (نعمل منه جدول رسمة)
- State Table (نعمل منه جدول)
- *State encoding*
- Select type of Flip-flop (ثم نرسم ال FF)
- Input equations to FF, output eq.
- Verification

(تعبئة عادي وترسمه)



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.

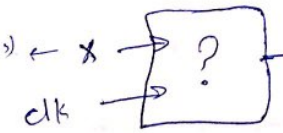
  شيء حدث بالماضي و مهم و مفيد نحتاج للحفاظ على القيمة السابقة لأنه يهمنا في الحالة التالية.

  الجهاز المستخدم للتخزين. (التصميم المخطط للتخزين) ← يتكون من binary inputs

- 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.

لازم يدخل التسلسل (1101) ← x

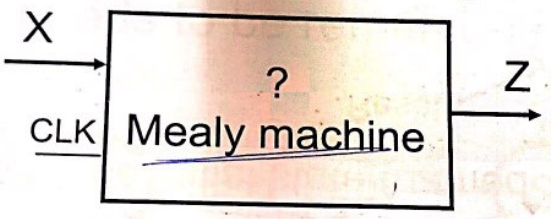


→ Z → عندما يكون التسلسل المدخل (1101)

يكون ال Z output = (1)

Sequence Detector Example: 1101

Sequential design system:
Mealy — Moore.



التسلسل الصحيح

over lapping sequence : 1101 مشتركة

Input X:	00111001101011011010011110111 → sequence : تسلسل
Output Z:	00000000001000001001000000000100

آخر وحدة بعدما تحقق من الشرط يطلع ال (output = 1)

Overlapping sequences are allowed

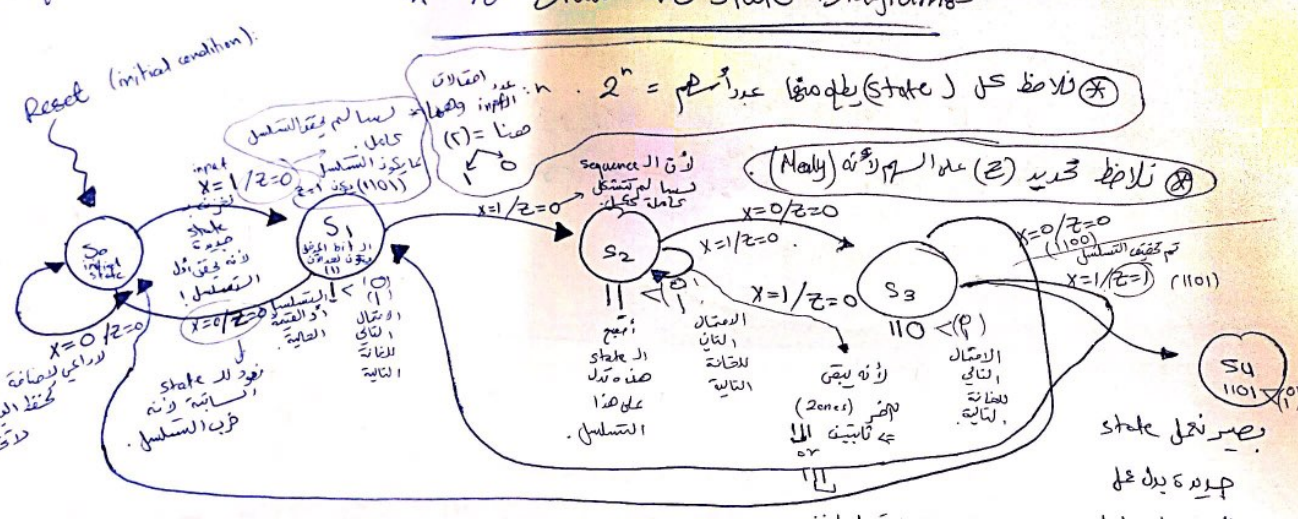
If the sequence is (0011):

input x : 00111001101011011010011110111

output Z : 00001000000000000000000000000

The sequence (1101):-

* To Draw the state Diagrams



* نلاحظ كل (state) يمثل عدد أسهم = 2^n . n: عدد الاحتمالات الـ input وهما صفر = (0) ، 1

* نلاحظ تحديد (z) على السهم لأنه (Mealy).

لأن الـ sequence لسه لم تتشكل كاملة كخطوة

لسه لم يحقق التسلسل كامل. عايزين التسلسل (1101) يكون 1+2

input

نفس الـ state جديدة لأنه لحقق أول التسلسل!

الاحتمال الثاني للخانة التالية

التسلسل أو القيمة الحالية

نعود للـ state السابقة لأنه خرب التسلسل.

لأنه الـ state هذه تدل على هذا التسلسل.

الاحتمال الثاني للخانة التالية

لأنه يبقى له معنى ثابتين (2ones) الـ

الاحتمال الثاني للخانة التالية.

تم تحقيق التسلسل

بصير نعمل state جديدة بدل عن الرجوع لـ state (1) حيث لو عملنا state رجع لـ 0 ← 11010

state رجع لـ 2 ← 11011

يبقى لها نفس معنى (state2) لأنه الـ 2bit لسه تم (11)

أن يكون الـ (bit) السابقة بداية لتسلسل جديد.

* لو كان الـ (over lapping) غير مسموح ← نرجع لـ state (0) دائماً عندما (x=0 / x=1)

يعني كل الـ arrows يعودوا لـ (S0)

* عندما نكون نريد تخزين القيمة السابقة نضيف (state) جديدة

* بينما عندما نكون لا نريد تخزين القيمة السابقة (أي لا تهم) في الـ sequence لا نضيف شيء ونبقى في الـ state الأولية

أحياناً نعود للـ state السابقة

نعود لنحول الرسمة (state diagram) إلى جدول (state table)

⊗ عادي قراءة الأسهم.

present state	inputs (X)	next state	outputs
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

* بعد ما عملنا الجدول نكتب المعادلات ولكي تكون (design optimised) نستخدم الـ (K-map)

0	0	1	0
1	1	0	0

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

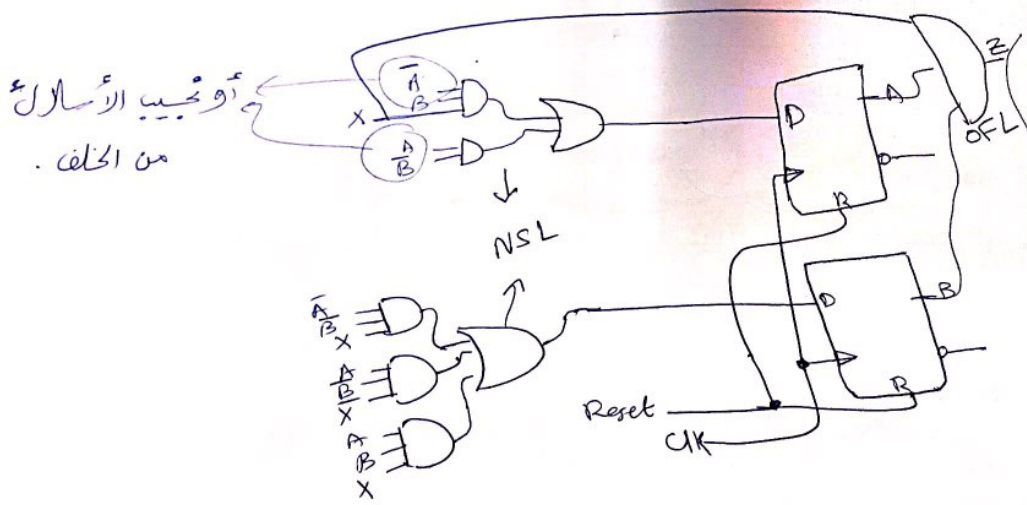
ونفس الشي لـ (D_B)

لنا (4 states) S_0, S_1, S_2, S_3 يجب نحول الـ (states) إلى (binary bits) لكي نجد الـ Equations والـ gates المستخدمة فيهم.

ولازم لكل (state) أن يكون له (binary code) مميز عن غيرها من الـ (states) لأن أقل عدد من (bits) يلزم هو (2) bits.

state encoding:
$S_0 = 00$
$S_1 = 01$
$S_2 = 10$
$S_3 = 11$

* نريد نرسم الـ (design) بعد ما عملنا الـ (k-maps) :



* اعمال (k-map) لـ (DB) :

		B	
0	1	0	0
1	0	1	0

A { 1 } لا تجمع !

X

$$DB = \bar{A}\bar{B}X + A\bar{B}\bar{X} + ABX.$$

* عمل (k-map) لـ output

		B	
0	0	0	0
0	0	1	0

A { 0 }

X

$$Z = ABX$$

G For (DA) = 7

G For (DB) = 12

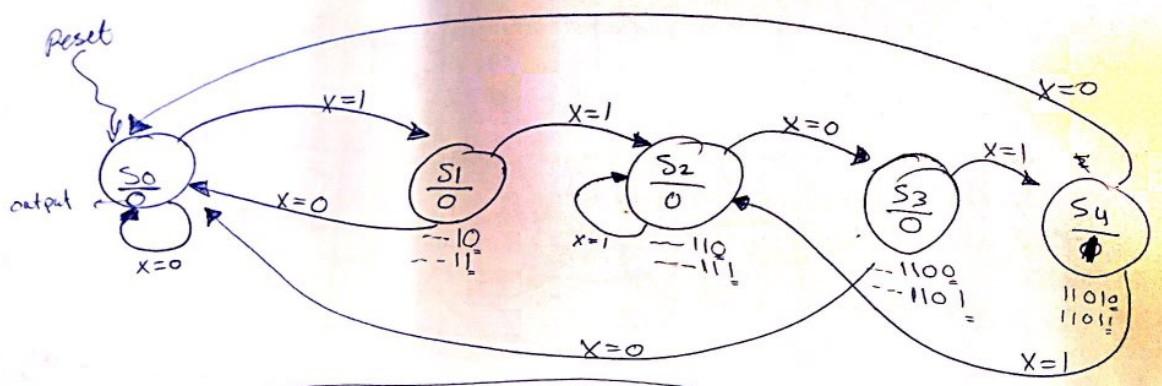
G for (Z) = 3

Total G = 22

Reset active heigh → بدون pupple (1)

Reset active low → موجود pupple (0)

* The state diagram as a Moore Machine:-



State table

Present state	input	next state	output
S0 (000)	0	000	0
S0	1	001	0
S1 (001)	0	000	0
S1	1	010	0
S2 (010)	0	011	0
S2	1	010	0
S3 (011)	0	000	0
S3	1	100	0
S4 (100)	0	000	1
S4	1	010	1

We have (5 states) so, we need at least (3) bits.

* size of the table

$2^4 = (16)$

$D(Q_A) :- Q_A^+$

0	0	0	0
0	0	1	0
0	0	0	0
0	0	0	0

Q_C, Q_B, Q_A, input (X)

unused states.

$Q_A^+ = D_{Q_A} = \bar{Q}_A Q_B Q_C X$

$D(Q_B) :- Q_B^+$

0	0	1	0
1	1	0	0
0	0	0	0
0	1	0	0

Q_C, Q_B, Q_A, X

$Q_B^+ = Q_A \bar{Q}_B \bar{Q}_C X + \bar{Q}_A Q_B \bar{Q}_C + \bar{Q}_A \bar{Q}_B Q_C X.$

$D(Q_C) :- Q_C^+ :-$

0	1	0	0
1	0	0	0
0	0	0	0
0	0	0	0

Q_C, Q_B, Q_A, X

$Q_C^+ :- \bar{Q}_A Q_B \bar{Q}_C \bar{X} + \bar{Q}_A \bar{Q}_B \bar{Q}_C X.$

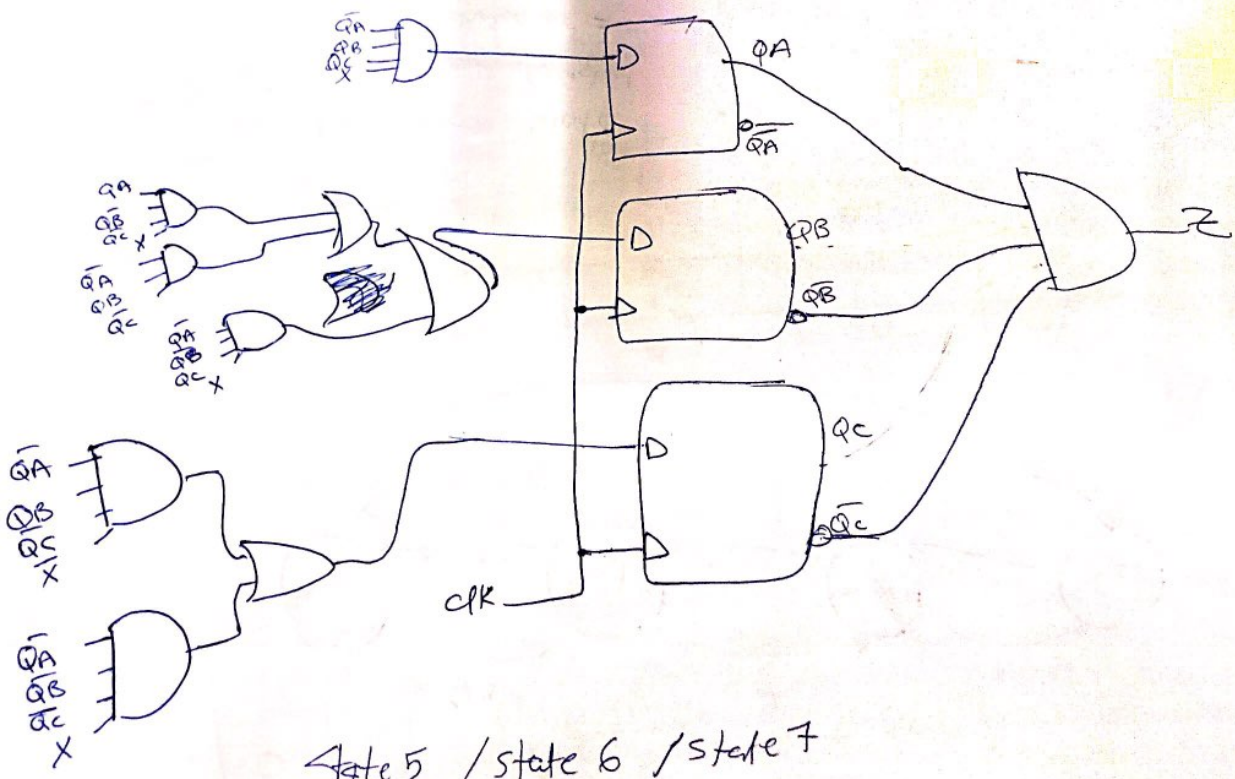
$Z :-$

0	0	0	0
0	0	0	0
0	0	0	0
1	1	0	0

$Z = Q_A \bar{Q}_B \bar{Q}_C$

07977706608

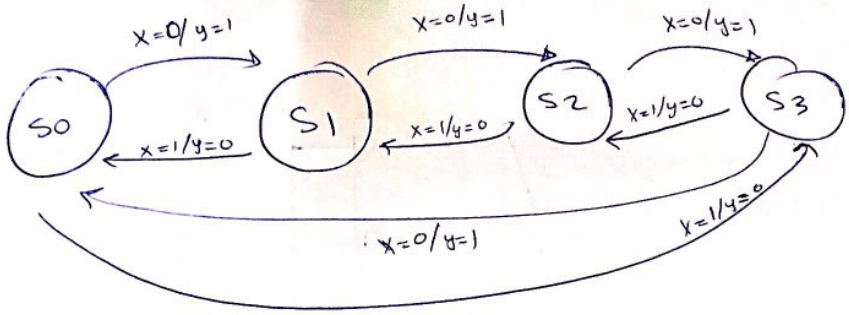
The logic diagram:-



State 5 / state 6 / state 7

(unused)

up / down counter:-



$x=1, y=0 \rightarrow$ down counter.

$x=0, y=1 \rightarrow$ up counter.

2 → inputs $(x, y) \rightarrow$ state لكل

(4 conditions) لطلوعها

= عدد الاحتمالات.

⊛ لـ (Up counter) :- نعمل (State table) جدول (1-input)

Resent state QA QB	inputs X	next state DA DB	No outputs
00	0	00	the state = the output.
00	1	01	
01	0	01	
01	1	10	
10	0	10	
10	1	11	
11	0	11	
11	1	00	

⊛ نعمل (K-map) :-

DA:

0	0	1	0
1	1	0	1

$DA = A\bar{B} + AB\bar{X} + \bar{A}BX$.

DB

0	1	0	1
0	1	0	1

$DB = \bar{B}X + B\bar{X}$.

For the sequential circuits:-

analysis: التحليل

... خطوات الحل ...

① نكتب المعادلات. y = output
DA = next states
DB = — .

② نرسم (state table) ونضع الاحتمالات فيه كالمات
$2^{(n)}$ عدد ال inputs + present states

③ نرسم (state diagram) حسب ال design
moore / mealy

(*) نبدأ من logic diagram ← نحول ل state diagram

Design: التصميم

① يكون لدينا specification (word description) (وصف).

② نرسم منه ال (state diagram).

③ نحول منه ال (state table).

④ نحل (K-maps) لل y (output)
DA (next state)
DB

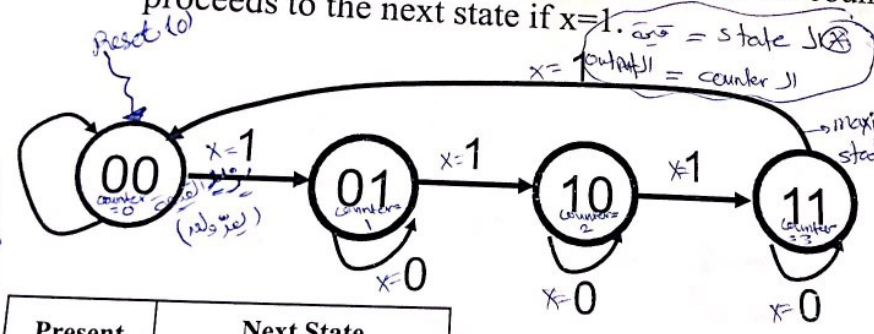
⑤ نجمع ونكتب ال equations ولازم يكونوا optimized (مختصرين / مبسطين).

⑥ بعد ال (Equations) نرسم الشكل حسب ال (gates) التي لدينا بالمعادلات.

(*) نبدأ من state diagram ← نحول لل logic diagram

Exercise

- Use (D Flip-Flops) design a counter that counts 00,01,10,11,00,01,10,11, ..etc.
- The counter also has an input x such that the counter pauses if x=0 and proceeds to the next state if x=1.



Present State Q_1Q_0	Next State X = 0 $Q_1^+Q_0^+$	Next State X = 1 $Q_1^+Q_0^+$
00	00	01
01	01	10
10	10	11
11	11	00

Q_0^+	Q_1		
0	1	3	2
4	5	7	6

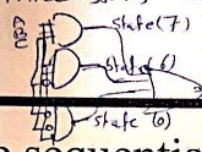
$Q_0^+ = X \oplus Q_0$

Q_1^+	Q_1		
0	1	3	2
4	5	7	6

$Q_1^+ = \bar{X}Q_1 + Q_1\overline{Q_0} + X\overline{Q_1}Q_0$

Exercise

* The Three states which we don't use are :-



state 7 : 111
state 6 : 110
state 0 : 000

if we are in one of the unused states. (0, 6, 7)

- Use D-FFs to design the sequential circuit that implements the following state table. Note that there are three unused states (000, 110 and 111).

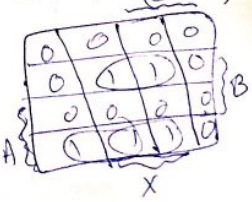
$2^4 = 16$ cells ; Kmap

* ال unused نعبي بدالهم أصفار في ال (K-map)

Present State → 3 bits			Input → 1 bit	Next State		
A	B	C	X	A	B	C
0	0	1	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	0	1	1
0	1	0	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	1	0	0
1	0	0	0	1	0	1
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	0	1	1	1	0	0

State (0, 6, 7) are unused states.
state 0
state 2
state 3
state 4
state 5

* We have (3 Kmaps) :-



DA = 16 cell (K-map)
DB = 16 cell (K-map)
DC = 16 cell (K-map)

Solution

* استخدام هذا ال (combinadional logic) هو عمل detiction (تحري) للدخول مع أحد من ال (unused states)

* تم نستخدم ال (Signal F) لعمل (Reset / set) ال (unused states)

(*) إذا حدث (Error) في النظام يدخل في ال unused states فنحن نريد إجباره على العودة لأحد ال used states.

- Asynchronously change the state to "011"

ال (FF) بشكل (asynchronous) وانتبه في الانتباه ال active (P) S low / high (1)

A_{t+1}

	C		
0	1	3	2
4	5	7	6
	1	1	
12	13	15	14
8	9	11	10
1	1	1	

A, B, X

B_{t+1}

	C		
0	1	3	2
		1	
4	5	7	6
1			
12	13	15	14
8	9	11	10

A, B, X

C_{t+1}

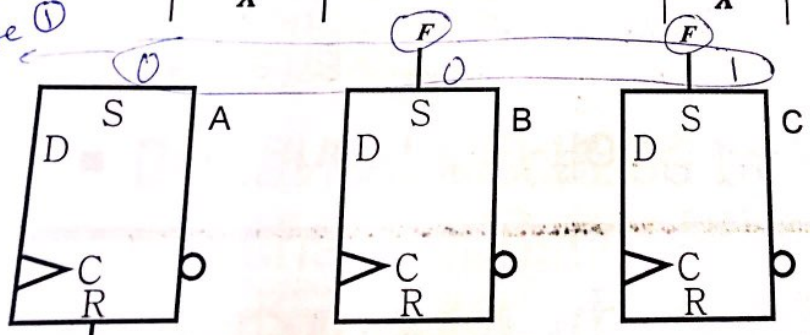
	C		
0	1	3	2
			1
4	5	7	6
1			1
12	13	15	14
8	9	11	10
1			1

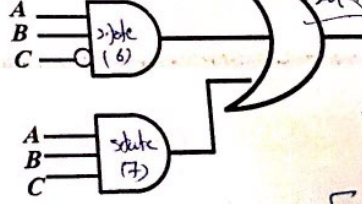
A, B, X

مثال :- state (4) 100 Reset set Reset وهكذا

(*) حسب رقم و تمثيل ال (state) المراد الدخول إليها نضع Reset / set حيث Reset ← 0 , set ← 1

from unused state to 0 state (1)





if one of them is the unused state.

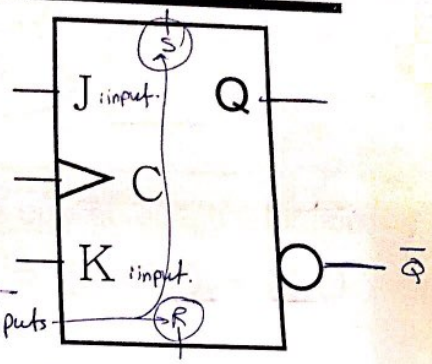
F

unused states (0,6,7) عند ظهور أحدى ال state(1) بأن يذهب ال (system) نريد إجبار ال

1 J-K Flip-flop

→ It has (2-inputs) يشبه (SR-FF)

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



It may has direct inputs

- Behavior described by the characteristic table (function table):

يوضح ال behavior لل (JK-FF) ← يعني ننظر (inputs J, K) وال (current state) ومنه نحسب ال (next state)

J (S)	K (R)	Q(t+1) القيمة المضرية	
0	0	Q(t)	no change (hold) يحتفظ بنفس القيمة الموجودة.
0	1	0	reset (تصفير)
1	0	1	set (تخزين)
1	1	$\overline{Q(t)}$	toggle (complement القيمة الحالية)

(not allowed) ← [بدل]

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

* بتناد (J,K) FF من (D FF)

Design a sequential circuit using (1) D-type flipflops and has 2-inputs (J, K) such that when

State table of a JK FF:

لأنه (1) D type Flip flop (1bit) إذا

Present state Q(t)	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) $\bar{0}=1$
1	0	0	1 (hold)
1	0	1	0 (Reset)
1	1	0	1 (set)
1	1	1	0 (toggle) $\bar{1}=0$

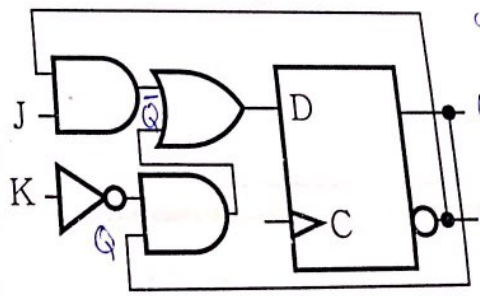
$Q(t+1)=D_A$

(next state) الـ هي (1bit) لأنه (1k-map) لونا

Q	J=0, K=0	J=0, K=1	J=1, K=0	J=1, K=1
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 → لأنها ما فرقة صا characteristic table



J-K Flip-Flop Excitation Table

عكس ال Characteristic table

الجدول معكوس: من ال (Present / Next state) نجد ال (inputs)

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

(hold) 0 0 (0)
(Reset) 0 1 (0)
0 X → don't care.

10 (set)
$\bar{0}=1 \rightarrow$ 11 (toggle)

Reset 01
toggle 11 → X1

10 (set)
00 (toggle).
X0 →

T Flip-Flop → It has (1-input) الـ (D-FF)

The external inputs are important here because they make (Set/Reset)

- Behavior described by its characteristic table:

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

- 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

* ممكن يكون الـ (T-FF) direct inputs (S/R) و كذلك (JK-FF) وفي حالة (TFF) ضروريين (S/R) لأنه بدونهم لا نعمل intialization (Reset/Set)

Characteristic equation:

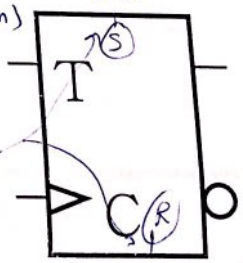
$$Q(t+1) = \overline{T}Q(t) + T\overline{Q}(t)$$

$$= T \oplus Q(t)$$

(0) (1) xor

من الـ (Equation) يتم استنتاج الـ (implementation) تبع الـ (T/JK) FF من الـ (D.FF)

it may has direct inputs



Characteristic table:

T	Q(t)	Q(t+1)
0	0	0 hold
0	1	1 hold
1	0	1 complement
1	1	0 toggle

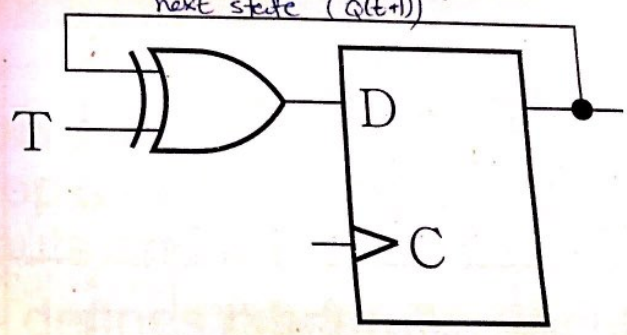
$T \oplus Q(t)$

T Flip-Flop Realization

- Using a D Flip-flop: $D = T \oplus Q(t)$ → present state.

next state ($Q(t+1)$)

(D.FF) من (T.FF) بناء



- Cannot be initialized to a known state using the T input
  - Reset (asynchronous or synchronous) essential

direct inputs → set S=1 R=0
→ Reset R=1 S=0

T Flip-Flop Excitation Table

الجدول العكسي :-

next state = current state → T = 0

next state = complement of current state → T = 1

Q(t)	Q(t+1)	T	
0	0	0	No change (hold)
0	1	1	toggle.
1	0	1	toggle.
1	1	0	hold.

Q(t)	Q(t +1)	T	Operation (اختصاره)
	$Q(t)$	0	No change (hold)
	$\overline{Q}(t)$	1	Complement (toggle)
			~~(toggle)~~
			~~No change (hold)~~

Flip-Flops Characteristics

For analysis

- *Characteristic table* (or (Chara --- Equation)) - defines the next state of the flip-flop in terms of flip-flop inputs and current state

  we have logic Diagram → we want to build the state Diagram

- *Characteristic equation* - defines the next state of the flip-flop as a Boolean function of the flip-flop inputs and the current state.

For design

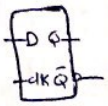
- *Excitation table* - defines the flip-flop input variable values as function of the current state and next state. In other words, the table tells us what input is needed to cause a transition from the current state to a specific next state.

  we have the state diagram → we want to build the logic diagram.

D Flip-Flop Descriptors

- Characteristic Table

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



Q(t)	D	Q(t+1)	
0	0	0	→ hold نقط تتقد
0	1	1	→ set على (D)
1	0	0	→ reset قيمتها
1	1	1	→ hold

⊗ لا ننظر لـ Q(t) فقط لـ (D)

- Characteristic Equation

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

Q(t+1) = D.
Characteristic Equation

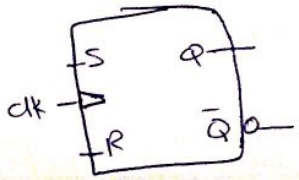
- Excitation Table

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

S-R Flip-Flop Descriptors

- Characteristic Table

	S	R	Q(t+1)	Operation
$Q(t).\bar{S}\bar{R}$	0	0	Q(t)	No change (hold)
	0	1	0	Reset
$1.S.\bar{R}$	1	0	1	Set
	1	1	?	Undefined (not allowed) X دنشيلها!



- Characteristic Equation

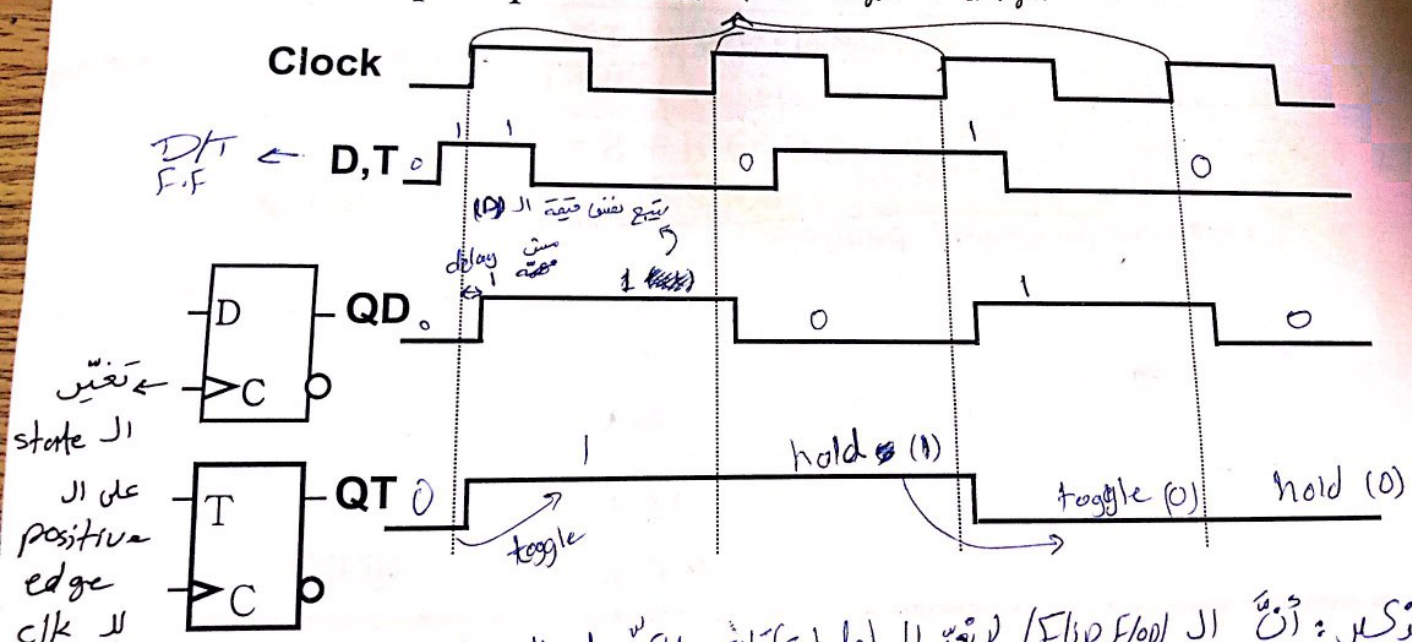
$$Q(t+1) = S + \bar{R}\,Q,\ S \cdot R = 0 \xleftrightarrow{\text{تبسيط (من جدول)}} Q(t).\bar{S}.\bar{R} + S.\bar{R}.1.$$

- Excitation Table

Q(t)	Q(t+1)	S	R	Operation
0	0	0	X	No change (00) / Reset (01)
0	1	1	0	Set
1	0	0	1	Reset
1	1	X	0	No change (00) / set (10)

Flip-flop Behavior Example

- Use the characteristic tables to find the output waveforms for the flip-flops shown: positive edges. → changes.

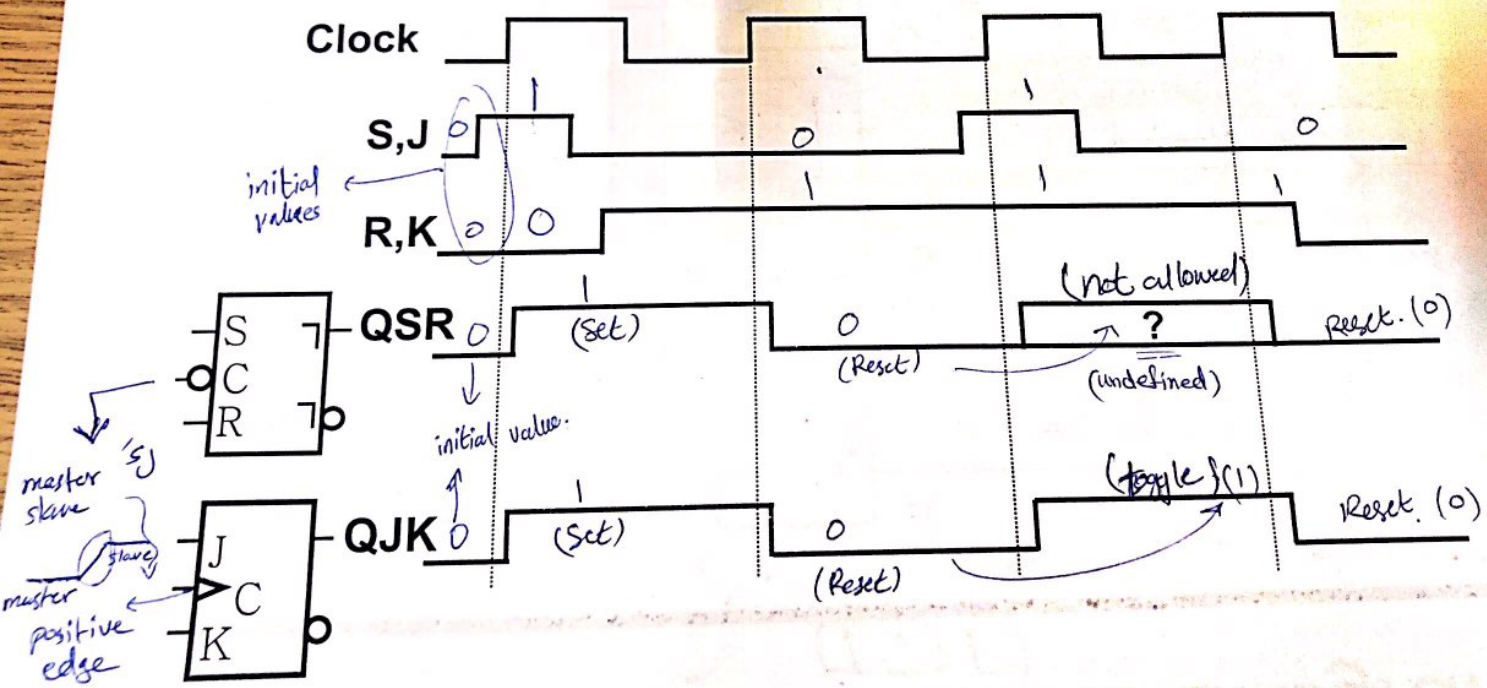


* تذكير: أنّ ال (Flip Flop) لا يتغيّر ال (state) تابعته إلا على ال positive لا يوجد (clk) ele bubble
أو
يوجد (clk) ele bubble ← negative edge

Flip-flop Behavior Example

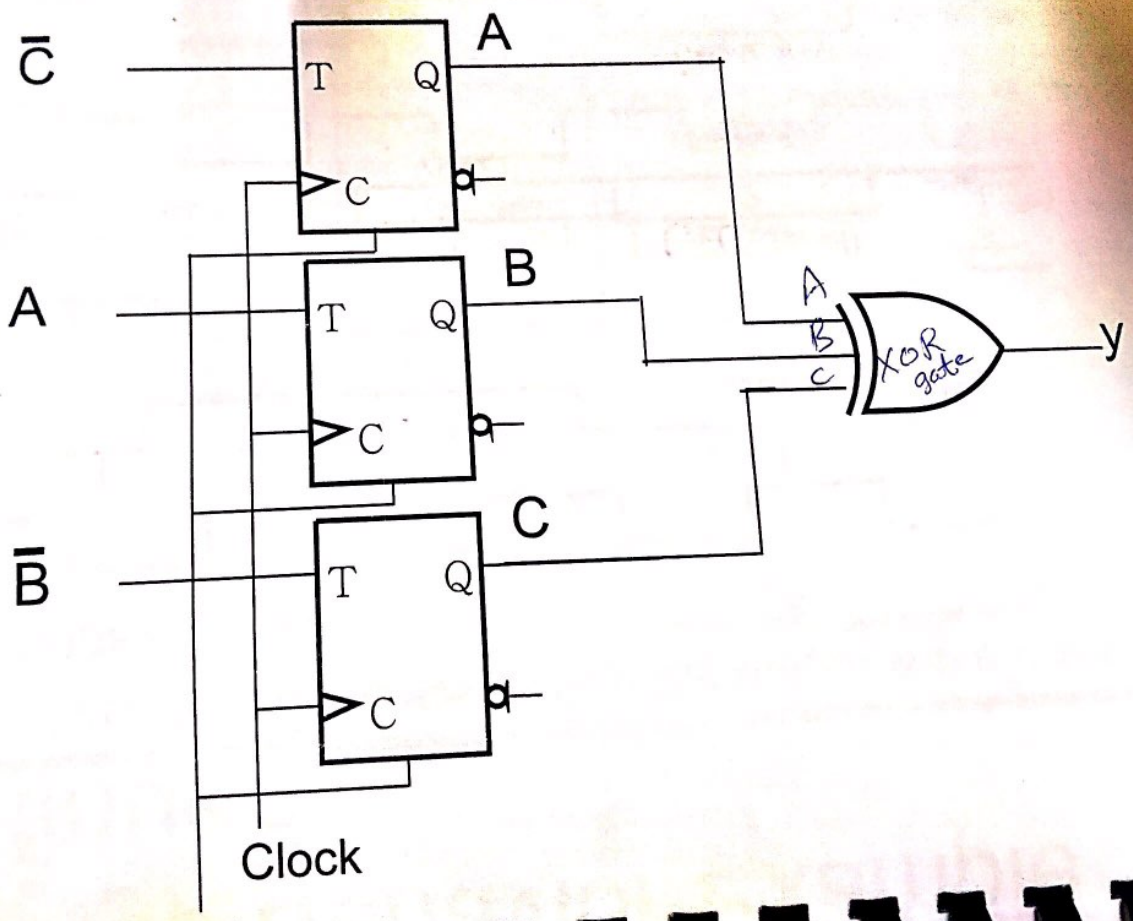
Flip-Flop Behavior Example (continued)

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



Exercise: Find State Diagram

(T. FF)



* Slide (88) :- Exercise

⊗ --- خطوات الحل --- :

① نكتب ال (outputs / inputs Equations)

② نعبّي ال (state table)

③ نرسم ال (state diagram)

(*) قبل كل ذلك
يجب أن نحدد
إذا ال system
Moore machine ⟷ Mealy machine
ومن هذا المثال
نستنتج أن التصميم
(Moore) لأنه لا يحوي
أي (inputs)

1. (*) n(inputs) / (one) output $y = A \oplus B \oplus C$.

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

} → for Flip flops

The input Equations are straight forward (مباشرة)

2. ⊛

نعبّي الاحتمالات عادي مثل الـ Truth Table

present State A	B	C	no-inputs	T(A) = $\bar{C}$	T(B) = A	T(C) = $\bar{B}$	next State A^+	B^+	C^+	outputs Y
0	0	0		1 (toggle)	0	1	1	0	1	0
0	0	1		0 (hold) (no change)	0	1	0	0	0	1
0	1	0		1	0	0	1	1	0	1
0	1	1		0	0	0	0	1	1	0
1	0	0		1	1	1	0	1	1	1
1	0	1		0	1	1	1	1	0	0
1	1	0		1	1	0	0	0	0	0
1	1	1		0	1	0	1	0	1	1

⊛ لكي نعبّي الـ (C^+ / B^+ / A^+)

نرى الـ T(A)/T(B)/T(C) :

بناءً عليه نحدد الذي يحدث

إذا ما كان ← toggle .

→ no change (hold)

⊛ النظر للقيمة الطبيعية (A / B / C)

1 = T(A)
1 = T(B)
1 = T(C)

T(A) = 0
T(B) = 0
T(C) = 0

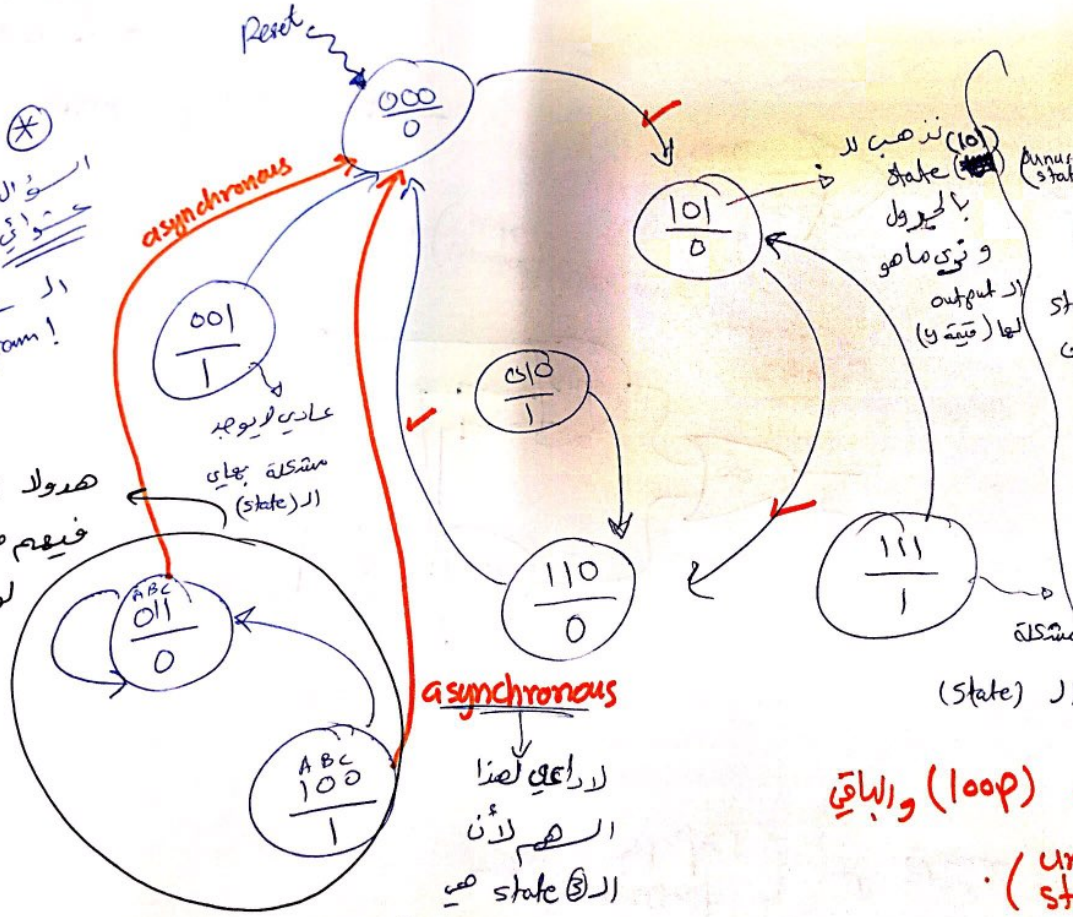
⊛ نعبّيه بأنه (XOR function) يعني (odd) ← عدد الـ (1) فردي

← يكون ناتجه (1) غير ذلك (0)

⊛ ملاحظة : مصنوع نكتب الـ (next state) ← لأن T(A) T(B) T(C) لا يمثلون الـ (state) .

3. ⊛

السؤال هنا عشوائي بشكل ال state diagram!



⊛ لمعرفة من هي ال (unused states) نرسم (state diagram) ونبدأ من ال (Reset) ونرى ال state التي لا ندخل عليها تكون هي (unused state).

نذهب لل state (101) بالجدول ونرى ما هو ال output لها (قيمة 0)

عادي لا يوجد مشكلة بهاي ال (state)

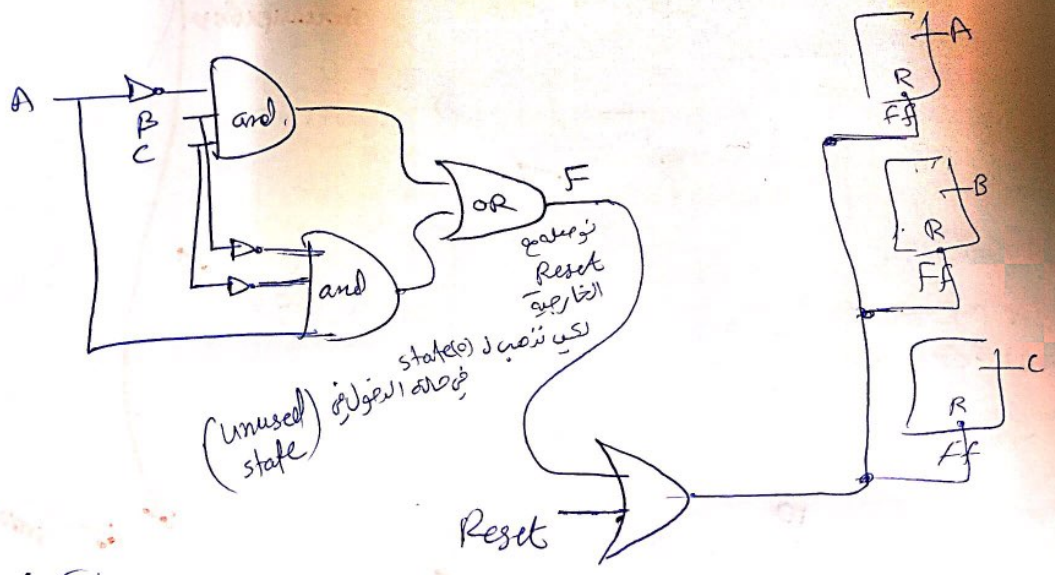
عادي لا يوجد مشكلة بهاي ال (state)

هدولا ال (states) فيهم مشكلة إنو لو دخلنا فيهم بالغلط! منعلق لذلك الحل هو: السهم التالي

لا داعي لهذا السهم لأن ال state هي التي يحدث عندها التعليق.

✓ states: ضمن ال (loop) والباقي يكون (unused states).

⊛ الحل لمشكلة الـ (unused states) :-



← لكي نرجع ل state(0) في حالة الدخول بال (unused state)