Digital Systems ENCS2340
Chapters(1-5)

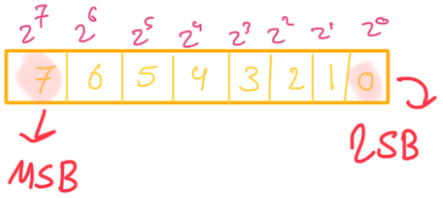
قال تعالى "إِنْ أَحْسَنتُمْ أَحْسَنتُمْ لأَنفُسِكُمْ ۖ وَإِنْ أَسَأْتُمْ فَلَهَا ۚ"

إيمان الغلبان – أصيل قدح

Chapter 1: Digital Systems and Binary numbers:

- **Binary numbers:**
  - → each binary digit (called a bit) is either 1 or 0
  - → Bits can represent:
    - Integers.
    - fractions.
    - characters.

- **Bit numbering:**
  - Least significant bit. (LSB)
  - most significant bit (MSB)



- **Decimal value of Binary Numbers:**
  - → each bit represents a power of 2.
  - → every Binary numbers is a sum of powers of 2.

- التحويل من Binary الى Decimal: $\text{Value} \times (2)^d$ (d = index)
  - Value ← صفر أو واحد.

- example: $(1001111101)_2 = 2^7 + 2^4 + 2^3 + 2^2 + 2^0 = (157)_{10}$
  - 7 6 5 4 3 2 1 0

- التحويل من أي نظام الى Decimal: $\text{Value} \times (\text{radix})^d$
  - ↳ Base.

- example: $(2107)_8 = 2\times8^3 + 1\times8^2 + 0\times8^1 + 7\times8^0 = (1095)_{10}$
  - 3 2 1 0

- التحويل من Decimal الى Binary:
  - ← نقسم على (2) حتى يصبح الناتج = صفر

- example: convert $(37)_{10}$ to Binary:

	الناتج	الباقي	
37/2	18	1	LSB

$(37)_{10} = (100101)_2$

18/2	9	0
9/2	4	1
4/2	2	0
2/2	1	0
1/2	0	1

MSB

• التحويل من عشري إلى أي نظام:
← بقسم على الـ (Base) حتى يصبح الناتج صفر.

• Popular Number Systems:

→ Binary:
- Base (radix) = 2
- Two digit values: 0 and 1
- numbers represented as zeros and ones.

→ Octal:
- Base (radix) = 8
- eight values: 0, 1, 2 ... 7.

• التحويل بين Binary, Octal سهل و Binary, Hexa لأنهم عبارة عن (2) مرفوعة لقوة.

→ Hexadecimal:
- Base (radix) = 16
- sixteen values = 0, 1, 2 ... 9, A ... F.
- A=10, B=11, C=12, D=13, E=14, F=15.

→ Decimal:
- Base (radix) = 10.
- Ten values = 0, 1 ... 9.

• Binary, Octal, Hexadecimal:

• من الـ (LSB) كل 3 منازل Binary = منزلة Octal
• من الـ (LSB) كل 4 منازل Binary = منزلة Hexa

Example: Convert 32-bit number into octal and Hexa:

most ↘ 3 5 3 0 5 5 2 3 6 2 4 ↙ least

1110 1011 0001 0110 1010 0111 1001 0100

E B 1 6 A 7 9 4 Hexa

Example: $(3BA4)_{16} \rightarrow (\quad)_{10}$

$3\times16^3 + 11\times16^2 + 10\times16^1 + 4\times16^0 = (15268)_{10}$

$(7204)_8 \rightarrow (\quad)_{10}$

$7\times8^3 + 2\times8^2 + 0\times8^1 + 4\times8^0 = 3716$

$(422)_{10} \rightarrow (1A6)_{16}$

422/16	26	6
26/16	1	A
1/16	0	1

$(422) \rightarrow (646)_8$

422/8	52	6
52/8	6	4
6/8	0	6

$(821)_8 \rightarrow (\quad)_{10}$

invalid العدد ، 0 → 7 = Range ال

- **Important proprities:**

→ How many possible digits can we have in radix r?

r digits, range of 0 - r-1

r = 8, 8 digits, 0 - 7.

r = 2, 2 digits, 0 - 2.

⋮

→ what is the result of adding (1) to the largest digit in radix r ?

$(9)_{10} + (1)_{10} = (10)_{10}$

$(1)_2 + (1)_2 = (10)_2$

$(7)_8 + (1)_8 = (10)_8$

إذا جمعت (1) لأكبر رقم في ال (radix = r) .
الجواب دائمًا بكون (10) في نفس ال radix

→ what is the Largest value using 3 digits in radix r?

$(111)_2 = 2^3 - 1$

$(777)_8 = 8^3 - 1$

$(999)_{10} = 10^3 - 1$

• أكبر عدد يمكن تمثيله في (n) منزلة في (radix=r) = $r^n - 1$

• Representing Fractions:

3 2 1 0 -1 -2 . . . .

Example: $(2409.87)_{10}$

$2\times10^3 + 4\times10^2 + 0\times10^1 + 9\times10^0 + 8\times10^{-1} + 7\times10^{-2}$

الأعشار بتبلش القوة من (-1)

• Converting Decimal fraction to Binary:

• تحويل الأعشار من Decimal إلى Binary:

← نضرب الأعشار X 2

← نوخذ العدد العشري . ونرجع نضرب لغاية ما نوصل الأعشار = صفر.

Example: Convert 0.6875 to radix 2:

$0.6875 \times 2 = 1.375$

$0.375 \times 2 = 0.75$

$0.75 \times 2 = 1.5$

$0.5 \times 2 = 1$

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

Convert 139.6875 to octal:

139 ← قسمة . 6875 ← ضرب . Radix = 8

$139/8 = 17$, 3 ↑

$17/8 = 2$, 1

$2/8 = 0$, 2

$0.6875 \times 8 = 5.5$ ↓

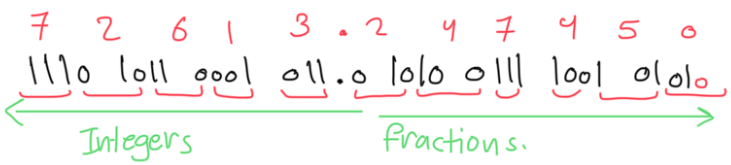
$0.5 \times 8 = 4.0$

$(139.6875)_{10} = (213.54)_8$

• التحويل بين (Octal, Binary) و (Hexa, Binary) في الـ fractions:

← بنلش من النقطة للبسار.

Example: Convert 32 Binary bit to octal:



Important properties of fractions:

→ what is the largest fraction value if m fraction digit is used in radix r?

in general $1 - m^{-r}$

→ How many fraction values exist with m fraction bit ?

Binary ⟶ 2^m (عدد القيم الممكنة (صفر أو واحد

- **Complements of numbers:**
  - → Radix complement (r's complement).
  - → Daminished radix complement (r-1)'s complement).

Daminished radix complement:

Complement = Largest number − N

Example: 9's complement (10-1) of 546700 =

999999 - 546700 = 453299

1's complement (2-1 comp) of 1011000 =

1111111 − 1011000 = 0100111

بخلي الصفر واحد والواحد صفر

Radix Complement:

Complement = $r^n - N$ (n → bits)

Example: 10's complement of 546700 =
1000000 − 546700 = 453300

10's complement of 012398 =
98 7602

- الأصفار زي ما هم.
- أول منزلة غير صفرية بوصلها لـ r^n
- باقي المنازل لـ $r^n - 1$

2's complement of 1011000 =
0101000

- أول منزلة غير صفرية زي ما هي
- اللي بعدها بعكس الصفر واحد والواحد صفر.

2's complement of 0101101 =
1010011

• Sabtraction using Complements:

Subtraction using r's complement:

1- find r's complement for the subtractor.
2- add the complemeat to the minued.

- Two possible cases:

Case 1: Subtractor smaller than minued:

- إذا كان المطروح أقل من المطروح منه، بطلع منزلة زيادة (carry) بهملها.
- الجواب موجب.

Case 2: Subtractor Bigger than the minued:

- إذا كان المطروح أكبر من المطروح منه،
- الجواب سالب (برجع بعمل r's complement).
- فش carry.

Example: Using 10's complement:

72532 − 03250
= 69282

- خليهم نفس عدد الـ Bit

$$
\begin{array}{r}
72532 \\
+\ 96750 \\
\hline
\fbox{1}69282
\end{array}
$$

10's complement for 03250 → 96750

بنطلها

Example: Using 10's complement:

03250 − 72532

= −69282

$$
\begin{array}{r}
03250 \\
+\ 27468 \\
\hline
30718
\end{array}
$$

(2's comp لازم ، carry فش)

(69282) ⊖ بحط سالب.

* Subtraction using (r-1) complement.

1- find (r-1)'s complement for the subtractor.

2- add the complement to the minued.

• Two possible cases:

Case [1]: Subtractor Smaller than minued:

• إذا كان المطروح أقل من المطروح منه ، بطلع منزلة زيادة (carry) بجمعها مع الجواب.

• الجواب موجب.

Case [2]: Subtractor Bigger than the minued:

• إذا كان المطروح أكبر من المطروح منه ،

• الجواب سالب (برجع بحل r-1 complement).

• فش carry.

Example: Using 9's complement:

72532 − 03250

= 69282

• خليهم نفس عدد الـ Bit

• اذا بدي أضيف بضيف عاليسار

9's complement → 72532 + 96749 = (1) 69281 ; + 1 = 69282

Example: Using 9's complement:

03250 - 72532

= -69282

9's complement → 03250 + 27467 = 30717

-(69282) ← 9's complement

Example: Using (2's) complement:

13 - 6 =

- 8 bit binary representation =

0000 1101 −

0000 0110

- convert subtraction to adding:

0000 1101 +

1111 1010

(1) 0000 0111 = 7 (الجواب)

يسقط

- المطروح أصغر من المطروح منه:
  - ← الجواب موجب.
  - ← يحمل الـ carry

Example: Using (2's) complement:

6 − 13

0000 0110 +

1111 0011

1111 1001

2's complement ← (0000 0111) = -7

- المطروح أكبر من المطروح منه.
  - ← الجواب سالب.
  - ← يرجع بشكل r's comp

Example: using (1's) complement:

110101 − 100101 =

110101 +

011010

(1) 001111

1+

010000

كل الحكي كد هيك عن الـ unsigned التعامل مع الـ signed مختلف.

• Signed numbers:

→ Several ways to represent signed numbers:

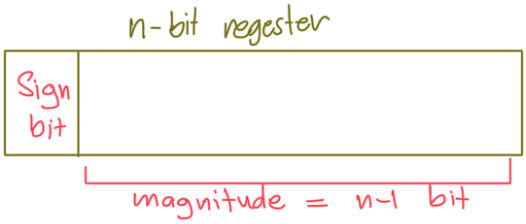
1- 2's complement. (الأكثر استخدامًا)

2- 1's complement.

3- Sign magnitude.

• في الـ unsigned كان الـ Range من صفر لغاية $(r^n - 1)$ هيك الـ Range رح ينقسم لسالب وموجب

1 sign magnitude Representation:



• إذا بدّي أمثل الرّقم في (n-bit) المقدار بمثله في (n-1 bit) و1 bit بحدد الإشارة الرقم.

• اذا Sign bit = 0 يعني الرّقم موجب.

• إذا Sign bit = 1 يعني الرّقم سالب.

• أكبر رقم يمكن تمثيله هو $(r^{n-1} - 1)$

• الـ Range = $-(r^{n-1} - 1)$ to $+(r^{n-1} - 1)$

• يعني الـ Range مقسوم لمنطقتين متساويتين وحدة موجبة و وحدة سالبة. (Symmetric range)

n = number of bits, r = base.

Example:

0 0101101 = + 45

1 0101101 = - 45

مشاكل هاي الطريقة:
← يوجد تمثيلين للصفر +0 , -0.
← صعبة في التعامل والجمع والطرح.

2) 2's complement Representation:

The 2's complement of N = 1's complement + 1

• أو باختصار، نبدل الأصفار والواحد منزلة غير صفرية، والباقي بعكسهم.

← إذا الرقم آخره 0 يكون موجب.
← اذا الرقم آخره 1 يكون سالب، ويرجع يعمل 2's عشان أجيب القيمة الأصلية.

← الـ 2's comp لأي رقم = سالب الرقم.
← إذا جمعت N + 2's complement عندي حالتين: 1- بشكل عام الجواب = 2^n
2- اذا ماحسبت الـ carry الجواب 0

Example: N = 00101100 (44)
-N = 11010100 (-44)
N + -N = 100000000 → in general, Result = 2^n
= 2^8
= 100000000
carry

← يوجد تمثيل واحد فقط للصفر

• Unsigned and signed value:

→ positive numbers:
Signed = unsigned

الاختلاف في القيم يظهر في الأرقام السالبة.

→ negative numbers:
Signed value = unsigned value - 2^{n} (n = number of bits.)

→ negative weight for MSB:

1 0 1 1 0 1 0 0
-128 64 32 16 8 4 2 1

-128 + 32 + 16 + 4 = -76

• Ranges of unsigned / signed Integers:

→ for n bit unsigned integeres : Range is 0 to $(2^n - 1)$
→ for n-bit signed Integeres: Range is -2^{n-1} to $2^{n-1} - 1$
→ Positive integers: 0 to $2^{n-1} - 1$
→ negative integeres: -2^{n-1} to -1

• Arithmetic Addition :

← اذا جمع عددين واحد منهم سالب (2's comp) وطلع مع carry بهملها .
← اذا جمعت رقمين (nbit) وطلع الجواب (n+1)bit يكون overflow .

إذا جمع عددين موجب (آخر bit = 0)
وطلع الجواب سالب (آخر bit = 1)
يعني في overflow.

إذا جمع عددين سالب (آخر bit = 1)
وطلع الجواب موجب (آخر bit = 0)
يعني في overflow.

• بشكل عام عشان أعرف في overflow أو لا بعمل (XOR) بين آخر 2 carry.

Example :

$15 + 8 = 23 \rightarrow$

```
  0000 1111   +
  0000 1000
  ---------
  000 10111
```

carry آخر منزلتين فش
false ⊕ false = false
no overflow + no carry

$79 + 64 = 143 \rightarrow$

```
  1⊕0=1
  (1)(0)
  0100 1111   +
  0 100 0000
  ----------
 (1)000 1111
```

- no carry (فش منزلة زيادة.)
- overflow = 1

↳ جمعت عددين موجب أعطاني الجواب سالب، أو XOR بين الـ carry.

$15 - 8 = 7 \rightarrow$

```
  (1)(1)(1)(1)
  0000 1111   +
  1111 1000
  ---------
 (1)0000 0111
```

carry

- carry = 1
- overflow = 0

1 ⊕ 1 = 0

• Binary codes :

← بقدر أعطي لأي اشي Binary code بشرط يكون unique و استخدم عدد bits مناسب.

Example :

Suppose we want to code (7) colors

$2^3 = (8)$ أقل عدد bit بقدر استخدمه = 3

بقدر أمثل 8 أكواد في 3bit

Red	000
Orange	001
Yellow	010
Green	011
Blue	100
indigo	101
Violet	110

فش اشي بلزمني بالقيم المهم ما يكون اشي مكرر، واستخدم أقل عدد Bit ممكن

• Decimal Codes:

← في أكثر من طريقة code عشان أمثل الـ Decimal بـ Binary

← التحويل بين الأنظمة ≠ الـ coding .

→ Binary coded decimal:

(BCD)

← تمثيل من 0-9 في 4bit binary.

→ BCD is weighted code:

يعني كل منزلة إلها قيمة معينة

Weights are = 8, 4, 2, 1

الـ BCD شبه التحويل لـ Binary من جهة الـ weight لكل منزلة.

Example:

1 = 0001

2 = 0010

3 = 0011

13 = 0001 0011

إذا عندي أكثر من منزلة، كل منزلة بمثلها بـ 4bit كاملها.

→ BCD Arithmetic:

جمع الـ BCD بشبه جمع الـ Binary بس اذا طلع معي قيمة invalid (أكبر من 9) برجع جمع الناتج 6.

Example: 8 + 5 = 13

1000 +

0101

1101 → أكبر من 9

+ 0110

0001 0011

1 3

Example: 2905 + 1897 =

0010 1001 0000 1101 +

0001 1000 1001 0111

0100 10010 1010 1100

جمع أول 4bit واذا بلزم بضيف 6 بعدين بكمل اللي بعده.

0110		0110	0110
0100	1000	0000	0010
4	8	0	2

→ Gray Code:

← الـ Gray code بسمح بتغيير bit واحد بس بين كل عددين متتاليين،
مثل: 1, 2 في الباينري 1=0001 2=0010 ، أما في الـ Gray code
في منزلتين تغيرت قيمتهم 1=0001 2=0011

→ conversion between Binary and gray:

عملية التحويل عبارة عن (XOR).

- from Binary to gray:

Binary 1 0 0 1 1 0
gray 1 0 1 0 1

أول منزلة زي ماهي.
بعدين XOR بين كل منزلتين ورا بعض

- from gray to Binary:

Gray 1 0 1 0 1
Binary 1 0 0 1 1 0

أول منزلة زي ماهي.
XOR بين كل منزلتين قبال بعض.

- Other Decimal codes:
  - → BCD
  - → 5421
  - → 2421
  - → 84-2-1
  - → excess-3

weighted codes.

Self complementary codes.

ممكن كل رقم يكونله أكثر من كود حسب الـ weight

جمع 3 وبعدين Binary

Decimal	BCD	5421	2421	84-2-1	Excess-3
0	0000	0000	0000	0000	0011
1	0001	0001	0001	0111	0100
2	0010	0010	1000	0110	0101

3	0011	0011	1001	0101	0110
4	0100	0100	0100	0100	0111
5	0101	1000	0101	1011	1000
6	0110	1001	1100	1010	1001
7	0111	1010	1101	1001	1010
8	1000	1011	1110	1000	1011
9	1001	1100	1111	1111	1100
unused					

complement

- Self complementry codes:

← اذا اخذت code بال 2421 وجبت ال complement تعكس بنفس ال code
مع يكونوا تنعكس بال excess-3 و 84-2-1

- Binary Logic:

→ Basic operations:

[1] And

Z = x and y
Z = x.y
Z = xy

Truth table

xy	z
00	0
01	0
10	0
11	1

Symbol

[2] OR

Z = x or y
Z = x + y

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

[3] not

Z = not X
Z = $\bar{X}$
Z = X'

x	z
0	1
1	0

Chapter [2]: Boolean Algebra and Logic gates:

Basic Theorems

Table 2.1
Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$		
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

- Example: Prove that $xy + x'z + yz = xy + x'z$.

→ proof: $xy + x'z + yz$ Starting from the left side

$= xy + x'z + 1 \cdot yz$ $\quad yz = yz \cdot 1$

$= xy + x'z + (x + x')(yz)$ $\quad 1 = x + x'$

$= xy + x'z + xyz + x'yz$ Distributive · over +

$= xy + xyz + x'z + x'yz$ associative commutative +

$= xy \cdot 1 + xyz + x'z \cdot 1 + x'yz$ $\quad xy = xy \cdot 1 \quad x'z = x'z \cdot 1$

$= xy(1 + z) + x'z(1 + y)$

$= xy(1) + x'z(1)$ $\quad 1 + z = 1 \quad 1 + y = 1$

$= xy + x'z$

$=$ right side

- Duality Principle:

The dual of boolean expression can obtained by:

1- interchange (And) and (OR) operations.

2- interchanging 0's and 1's

Example: The dual of $x(y + z') = x + (yz)$ The complement doesn't change.

	Property	Dual Property
Identity	$x + 0 = x$	$x \cdot 1 = x$
Complement	$x + x' = 1$	$x \cdot x' = 0$
Distributive	$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$

- if a property is proven to be true, then its dual is also true.

- Demorgan's Theorm.

(بستخدم لتوزيع الـ complement)

→ $(x+y)' = x' \cdot y'$

→ $(xy)' = x' + y'$

• بوزع الـ (')

← بغير الاشارة

← بشبه الـ dual بس هون الـ Complement برضو بتغير.

Truth table:

x	y	x'	y'	x+y	(x+y)'	x'y'	x y	(x y)'	x'+ y'
0	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	0	0	1	1
1	0	0	1	1	0	0	0	1	1
1	1	0	0	1	0	0	1	0	0
						Identical		Identical	

- Boolean functions:

→ could be described using:

1- expression. (Boolean Variables, Boolean constants and Boolean operators)

2- Truth table.

3- logic gates.

- operator precedence:

العمليات حسب الأولوية:

1- Expressions within parentheses. (العمليات داخل الأقواس)

2- Not

3- AND.

4- OR.

- Truth Table:

→ A truth table can represent a Boolean Function.

→ list all possible combinations of 0's and one's

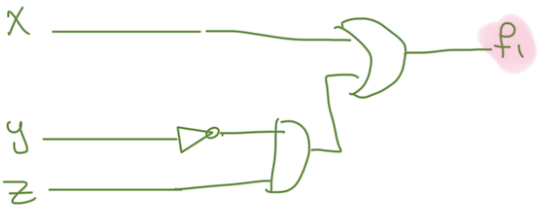
→ if n variables then 2^n rows

$f_1 = x + y'z$ $f_2 = x'y'z + x'yz + xy'$

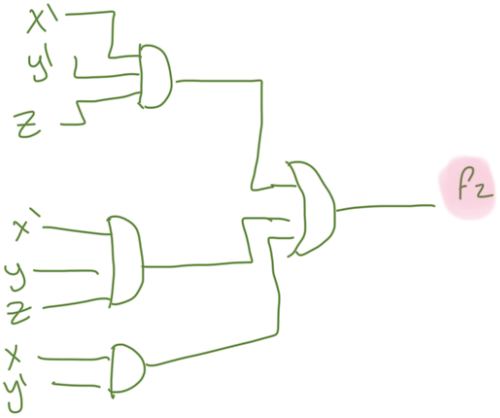
x	y	z	F_1	F_2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0

- circuit diagram

$f_1 = x + y'z$



$f_2 = x'y'z + xy'z + xy'$



- **Algebric manipulation:** تقليل عدد الـ Litrals

→ Litral: a single vorible within a term. كل متغير يسمى Litral سواء مع أو بدون complement

$f = x(x' + y)$ → 3 Litrals

Example:

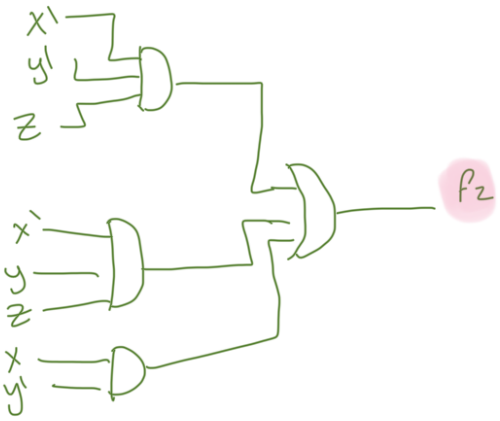
1- $x(x' + y) = xx' + xy = 0 + xy = xy$ — 2 Litrals.

2- $x + x'y =$ (3 Litrals) $(x + x')(x + y) = 1(x + y) = x + y$ — 2 Litrals.

3- $(x + y)(x + y')$ (4 Litrals) $= xx + xy' + xy + yy'$

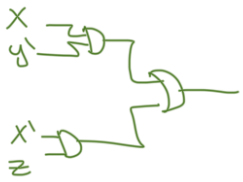
$= x(1 + y' + y)$

$= x(1) = x$ — 1 Litral.



→ After manipulation

$f_2 = xy' + x'z$



- **Complementing Boolean Functions:**

→ using Demorgan theorm.

Example: $f_1 = x'y z' + (xy'z'$ • يوزع النفي على الحدود كلها.

$F_i = (x+y'+z)(x+y+z)$ • يعني المتغيرات وعكس العمليات.

Simplification:

- canonical forms
  - → Sum of minterms.
  - → product of maxterms
- Standard forms
  - → Sum of product.
  - → product of Sum

• combinational Circuit:

الأوتبوت يعتمد على الانبوت الحالي فقط.

• Minterms and Maxterms: (canonical forms)

x	y	index	Minterm (m) (complement ~ يكون د و)	Maxterm (Mi) (complement يكون ا و)
0	0	0	$m_0 = x'y'$	$M_0 = x + y$
0	1	1	$m_1 = x'y$	$M_1 = x + y'$
1	0	2	$m_2 = xy'$	$M_2 = x' + y$
1	1	3	$m_3 = xy$	$M_3 = x' + y'$

- Minterms: And terms, with each variable represented in either true or complement form.
- Maxterms: OR terms, with each variable represented in either true or complement form.
- for n variables there are 2^n Maxterms and 2^n Minterms.
- minterm is complement of maxterm.

- Sum of minterms (SOM), طريقة كتابة f and Product of maxterms (POM)

x	y	z	f

0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	1

→ minterms: where function equals 1.

$f = m_2 + m_3 + m_5 + m_7$

$f = x'y'z' + x'yz + xy'z + xyz$

$f = \Sigma(2,3,5,7)$

→ maxterms: where function equals 0

$f = M_0 + M_1 + M_4 + M_6$.

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

$f = \Pi(0,1,4,6)$.

Examples :

$f(a,b,c,d) = \Pi(1,3,11)$

الترتيب مهم. 0001, 0011, 1011

$f = (a+b+c+d')(a+b+c'+d)(a+b+c'+d)$

عشان أكتب الاكسبريشن:
← بحول الاندكس لبايني.
← بكتبهم بدلالة (a,b,c,d).

$f(a,b,c,d) = \Sigma(2,3,6,10,11)$

0010, 0011, 0110, 1010, 1011

← في SOM و POM
لازم كل تيرم يحتوي على كل المتغيرات.

$f = a'b'cd' + a'b'cd + ab'cd' + ab'cd + abc'd'$.

- Conversions between Canonical forms:

The same Boolean function can be expressed in two ways:

→ Sum of minterms.

→ Product of maxterms.

To convert between them:

→ interchange the symbol (Σ, Π)

→ list the missing numbers.

Example: $f(a,b,c) = \Sigma(1,3,7)$

0-7 احتمالات 8 بين 3 variables $= \Pi(0,2,4,5,6)$

0,2,4,5,6 = الأرقام اللي طلعت ←

← بغير الرمز.

- Function complement :

Two ways to write the complement :

1) interchange the symbol withe Same numbers.
2) Same symbol and the missing numbers.

Example: $F(x,y,z) = \Sigma(0,2,3,5,7)$.
$F'(x,y,z) = \Sigma(1,4,6)$
$F'(x,y,z) = \Pi(0,2,3,5,7)$

- Algebric conversion to (SOM):

← شرط الـ SOM إنّه الـ variable كلهم يكونوا موجودين.
← عشان أحول الـ function لـ SOM بضيف الـ الناقص.

Example: $F(x,y) = x + x'y'$
لازم أكوم عادي عدهم. فش y. وا
$= x(y+y') + x'y'$
قيمتهم = 1 ماخ يأثر على الفنكشن.
$= xy + xy' + x'y'$ (SOM)
$= 11 + 10 + 00 = F = \Sigma(0,2,3)$.

$F(a,b,c) = a + bc$
$= a(b+b')(c+c') + (a+a')bc$
$= abc + abc' + ab'c + ab'c' + abc + a'bc$
$= abc' + abc + ab'c + ab'c' + a'bc$
$= 110 + 111 + 101 + 100 + 011$
$F = \Sigma(3,4,5,6,7)$

- Algebric conversion to (POM):

← يجيب الـ variable الناقص عن طريق الـ distributive law

Example: $F(a,b,c) = (ac' + bc) + a'b'$
ناقص b — a — c

$F(a,b,c) = (ac' + bc + a')(ac' + bc + b')$
$= (c' + a' + bc)(c + b' + ac')$
$ac' + a' = a' + c$
$bc + c' = b + c$
ترتيب المتغيرات تمام $= (a' + b + c')(a + b' + b)$
$= (1\ 0\ 1)\ (0\ 1\ 0)$
$F(a,b,c) = \Pi(2,5)$

- Standard forms:

عشان استخداماتها ← Sum of product. (نفس الـ SOM بس مش شرط تكونوا كل المتغيرات موجودين)

يكون الفنكشن → product of Sum. «نفس الـ POM بس مش شرط يكونوا كل المتغيرات موجودين»
مبسّط (عدد أقل من الـ literals).

Example: $F = A\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C + ABC$

«بقدر أطلع عامل مشترك أكثر من مرة».

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

$= AB' + B'C + AC$

$= SOP.$

Example: $f = A'B'C + AB'C' + AB'C + ABC$ $f = \Sigma(1,4,5,7)$

عشان أكتب f بطريقة (POS) (SOM)

$f' = A'B'C' + A'BC' + \dot{A}BC + ABC'$

1- بكتب f' بطريقة (SOP).

$f' = A'B(C+C') + A'C'(B'+B) + (A'+A)BC'$

2- بعمل complement

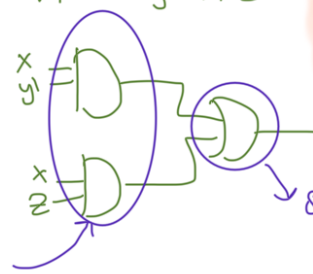
$(f' = A'B + A'C' + BC')'$ SOP

$f = (A+B')(A+C)(B'+C)$ POS

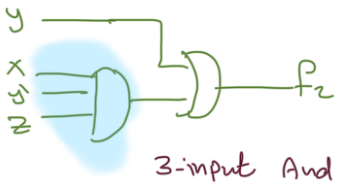
• Two level Gate Implementation:

$f_1 = xy' + xz$

First level: And-OR Implementation (Sum of product) — Second level



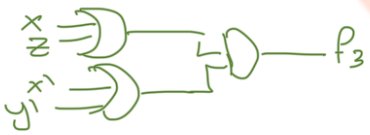
$f_2 = y + xyz$



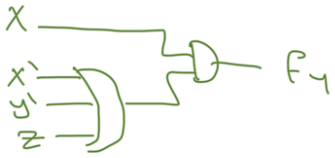
3-input And Gate.

$f_3 = (x+z)(x'+y')$

First level: OR-And Implementation (Product of Sum) — Second level



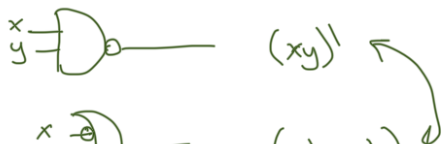
$f_4 = x(x'+y'+z)$



• Additional Logic Gates:

1. NAND Gate: (Not AND):

2 symbols for nand gate:



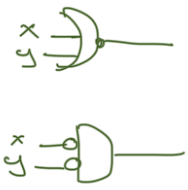
x y	NAND
0 0	1
0 1	1
1 0	1
1 1	0

y —— $(x'+y')'$

2] Nor gate:

(not OR)

2 Symbols for nor gate:



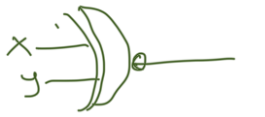
x	y	nor
0	0	1
0	1	0
1	0	0
1	1	0

3] XOR Gate:

Symbol:  $x \oplus y$

x	y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

4] X NOR Gate:

Symbol  $(x \oplus y)'$

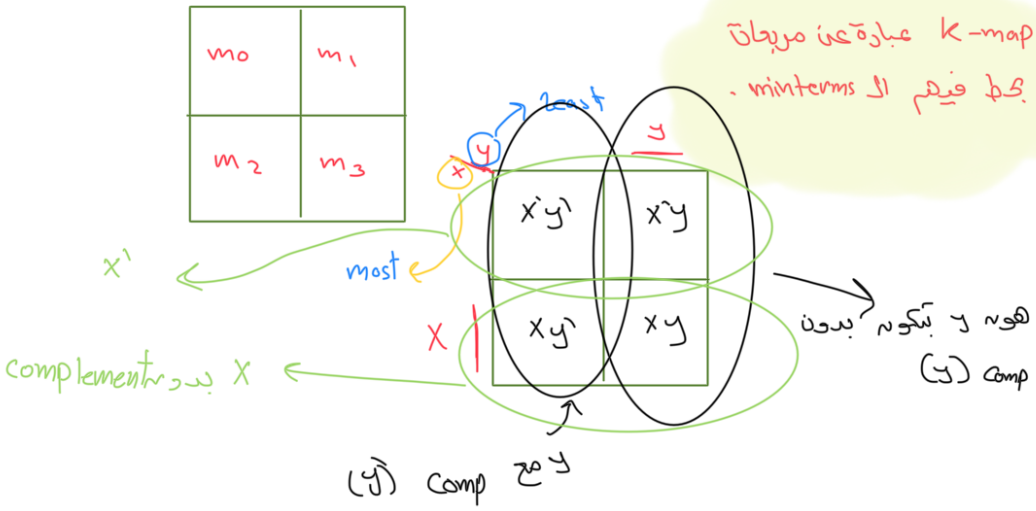
x	y	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

Chapter [3]: Gate - level minimization:

- Karnaugh Map: (k-map)

→ method to minimize the function

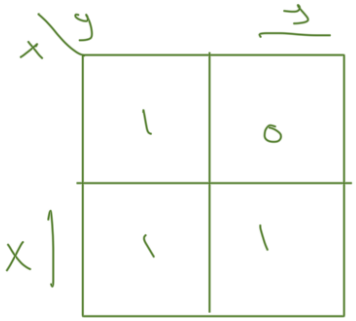
2 variables k-map

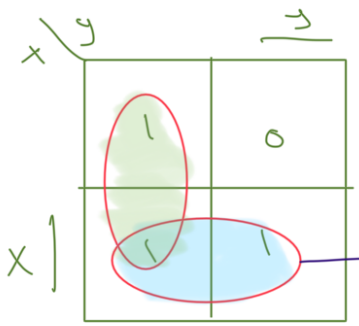


عبارة عن مربعات k-map
بحط فيهم الـ minterms.

Example:

	x	y	f	
محطط من السؤال.	0	0	1	m_0
	0	1	0	m_1
	1	0	1	m_2
	1	1	1	m_3





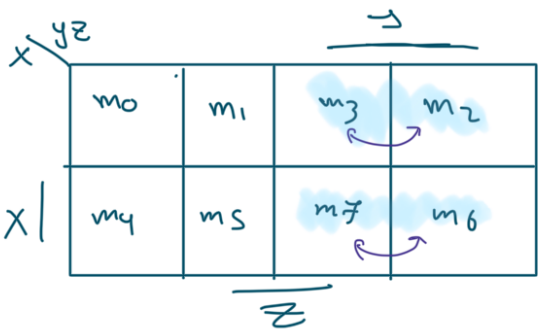
$f = y' + x$

- جمع الواحدات بشرط:
1- يكون عددهم 2 تس ($1, 2, 4, 8, \ldots$).
2- يكونوا adjacent (افقي أو عمودي مش diagonal)
3- كل مجموعة فيها أكبر عدد ممكن من الـ adjacents
4- مجموع الواحدات بدون تكرار

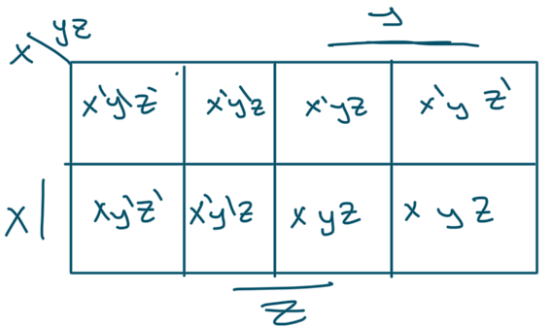
بالنسبة لـ x المجموعة كلها في (x)، بالنسبة لـ y جزء في منطقة y وجزء في منطقة y' فما بكتب y في الـ term.

- لما أكتب الـ function:
← كل مجموعة عبارة عن term.
← بكتب المتغير اللي ما بتغير قيمته.

- Three variables kmap: (x,y,z)



- صار تغيير بالترتيب عشان يضلّوا الـ adjecent جنب بعض، الفرق بين كل adjecent واللي بعده هي متغير واحد (نفس فكرة الـ gray code).



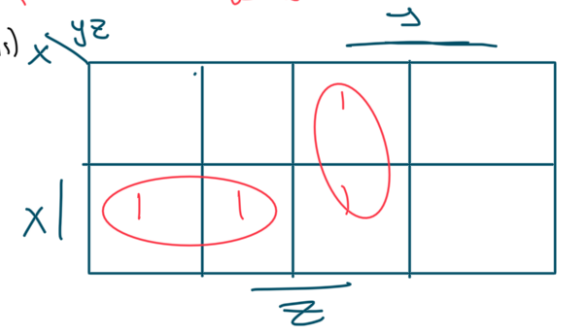
Example:

Simplify the boolean function $F(x,y,z) = \Sigma(3,4,5,7)$

الترتيب مهم — واحدات الفنكشن.

$f = x'yz + xy'z' + xy'z + xyz$ (12 literals)

$f = xy' + yz$ (4 literals)



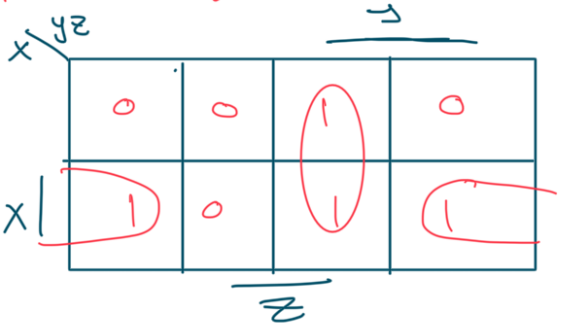
Example:

Simplify the boolean function $F(x,y,z) = \Sigma(3,4,6,7)$

الترتيب مهم — واحدات الفنكشن.

$f = x'yz + xy'z' + xyz' + xyz$ (12 literals)

$f = yz + xz'$ (4 literals)



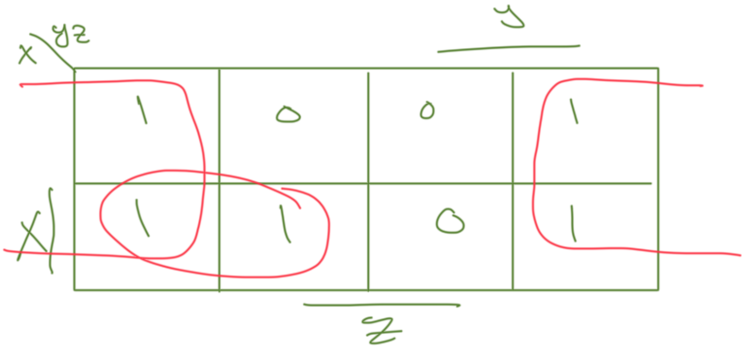
(الزايا عبارة عن adjecent (بقدر أ وخدم بنفس المجموعة))

- في الـ 3 variables kmap:
  - ← اذا أخدت مجموعة فيها 1 واحد، التيرم بكون فيه 3 variables
  - اذا أخدت مجموعة فيها 2 واحدات التيرم بكون فيه 2 variables

اذا أخدنا مجموعة فيها [4] واحدات التيرم بكون فيه 1 variables .

اذا أخدنا مجموعة فيها [8] واحدات التيرم بكون فيه 0 variables . (بتكون قيمته [1])

Example: $f(x,y,z) = \sum(0,2,4,5,6)$



$f = z' + xy'$

• four variables k-map:

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

• في الـ 4 variables Kmap :

← اذا أخدنا مجموعة فيها [1] واحد ، التيرم بكون فيه 4 variables .

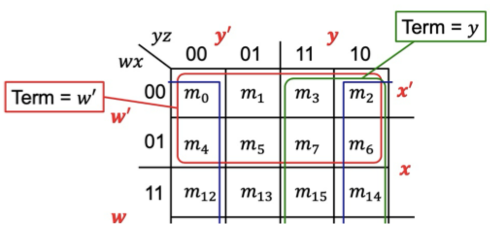
اذا أخدنا مجموعة فيها [2] واحد ، التيرم بكون فيه 3 variables .

اذا أخدنا مجموعة فيها [4] واحدات التيرم بكون فيه 2 variables .

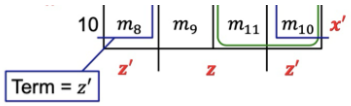
اذا أخدنا مجموعة فيها [8] واحدات التيرم بكون فيه 1 variables .

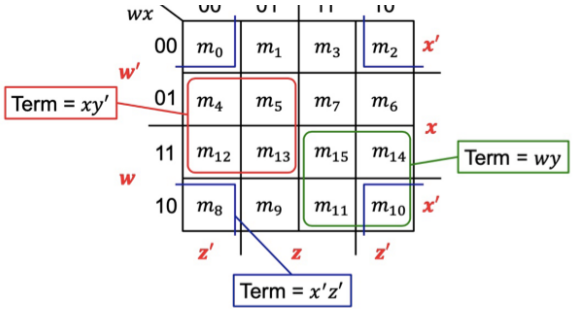
اذا أخدنا مجموعة فيها [16] واحدات التيرم بكون فيه 0 variables . (بتكون قيمته [1])

combining 8 Squares



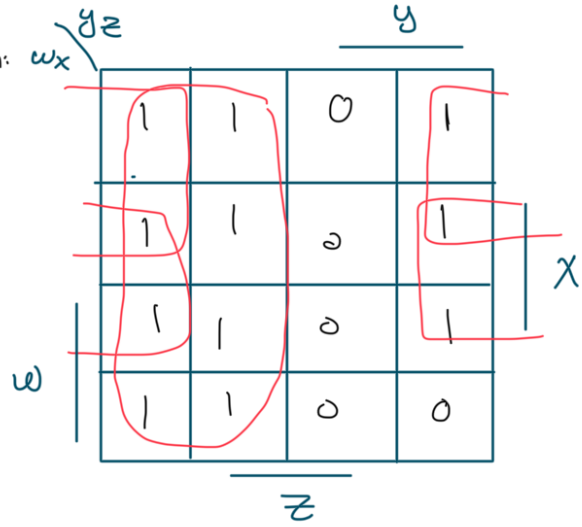
combining 4 Squares.



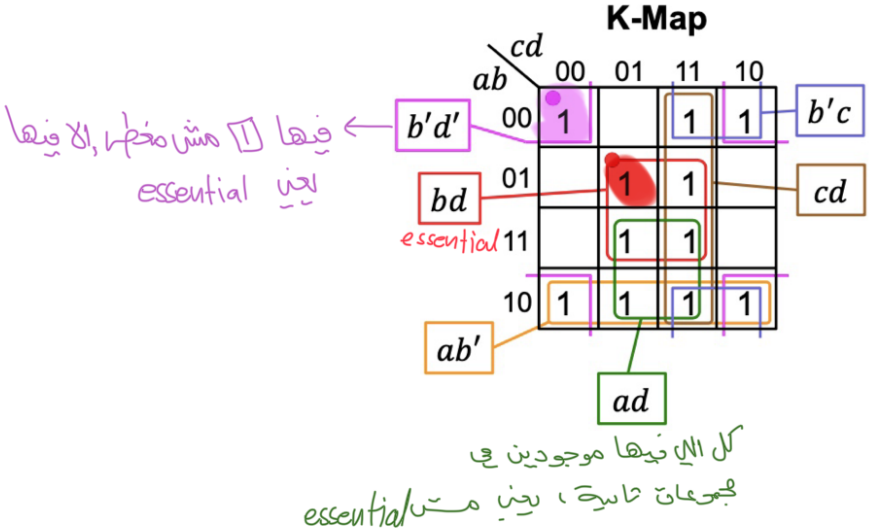


Example: Simplify the function: $f(w,x,y,z) = \Sigma(0,1,2,4,5,6,8,9,12,13,14)$

$f = y' + w'z' + xz'$



- Prime implicants:
  - → prime implicant: مجموعة adjacent
  - → essential prime implicant: مجموعة adjacent بس بتغطي واحدة مش مغطية في مجموعات تانية.



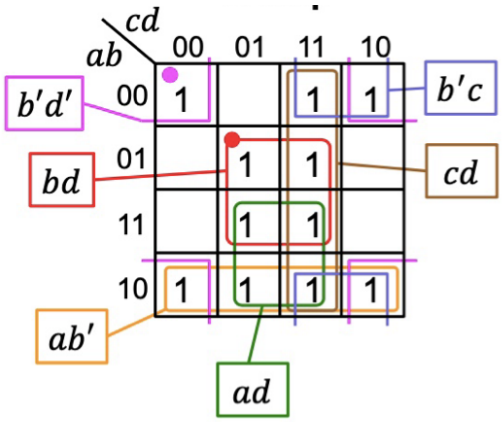
Six Prime Implicants
bd, $b'd'$, ab', ad, cd, $b'c$

Only Two Prime Implicants are essential
bd and $b'd'$

- ممكن يكون في أكثر من Simplified expression بس بشرط الـ essential يكونوا موجودين بكل الـ expressions.

K-Map

Four possible solutions:



Four possible solutions:

$f = bd + b'd' + cd + ad$

$f = bd + b'd' + cd + ab'$

$f = bd + b'd' + b'c + ab'$

$f = bd + b'd' + b'c + ad$

• Product of Sum Simplification.

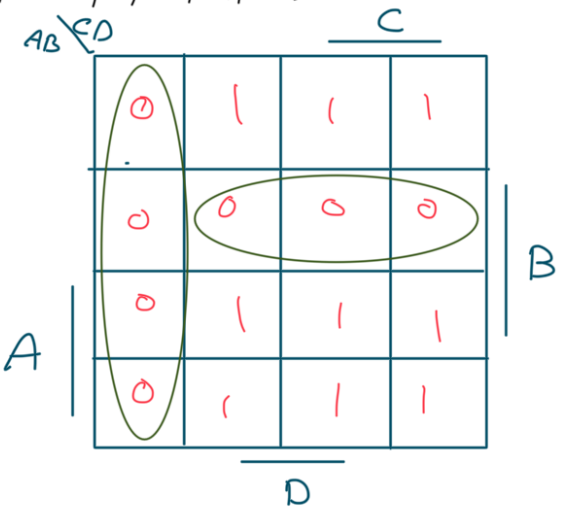
• Sum of product بشكل عام بتطلعلي simplified function بطريقة K-map

• عشان أبسط الفنكشن بس بطريقة product of sum ؛

← جمّع أصفار adjacent بدل واحدات (الناتج بكون F' as SOP

← بجيب complement الناتج عشان أطلع لـ F as POS

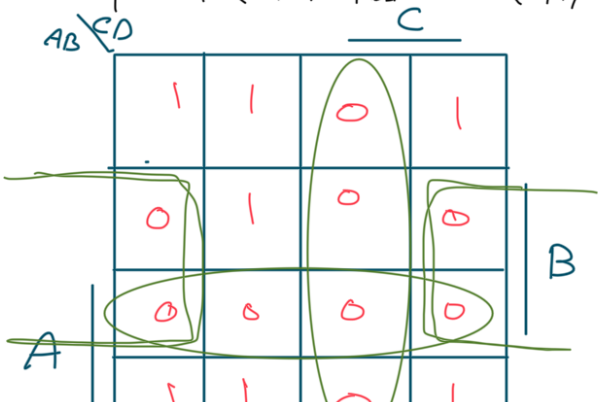
Example: $f(a,b,c,d) = \Sigma(1,2,3,9,10,11,13,14,15)$.



$F' = C'D' + A'B$.

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

Example: $F(A,B,C,D) = \Sigma(0,1,2,5,8,9,10)$



$F' = AB + CD + BD'$

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

- Don't cares:

- في حالتين لـ don't cares :
1- يكون الانبوت don't care ← يعني مش ممكن يكون أوتبوت موجود.
2- يكون الأوتبوت don't care ← يعني أوتبوت غير متوقع.
مكان الـ don't care بخط X ← بمعنى ممكن أحطها 0 or 1

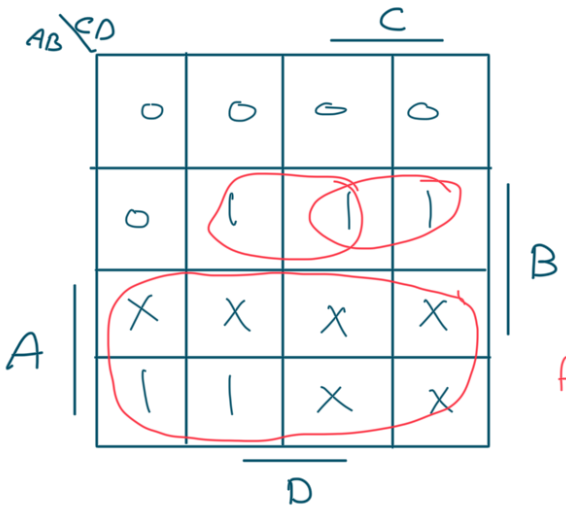
Example:

- Consider a function f defined over BCD inputs
- The function input is a BCD digit from 0 to 9
- The function output is 0 if the BCD input is 0 to 4
- The function output is 1 if the BCD input is 5 to 9
- The function output is X (don't care) if the input is 10 to 15 (not BCD)

Truth table:

a	b	c	d	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

$f = \Sigma(5,6,7,8,9)$
$d = \Sigma(10,11,12,13,14,15)$

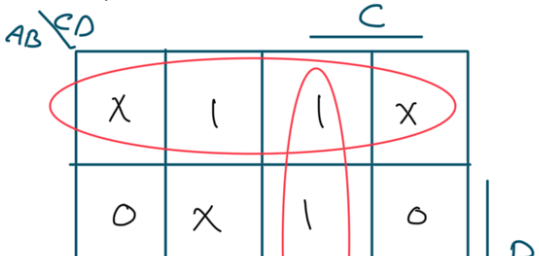


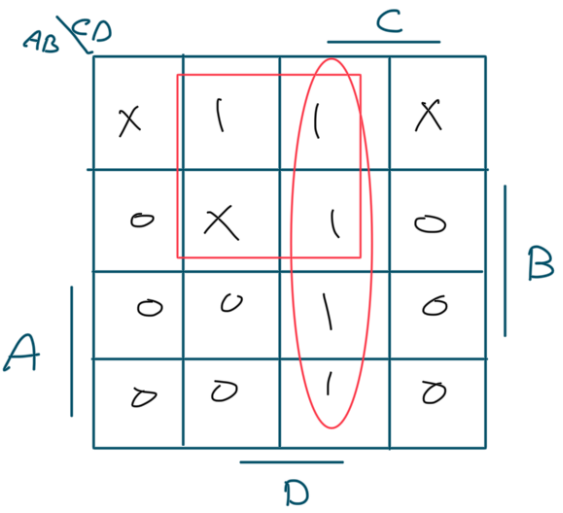
في k-map حسب الـ don't cares حسب شو بجمع يا مهفى يا واحد حسب شو بجمع وحسب اذا بحتاج أجمعهم

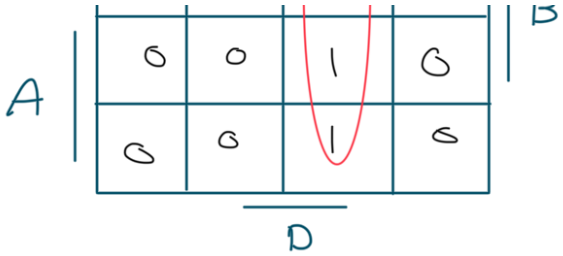
$f = A + A'BD + A'BC$

Example: Simplify $g = \Sigma m(1,3,7,11,15) + \Sigma d(0,2,5)$

First solution:
$g = cd + A'B'$



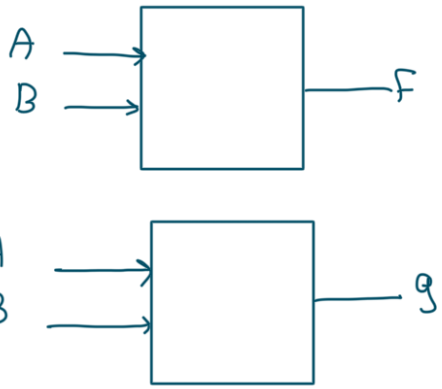


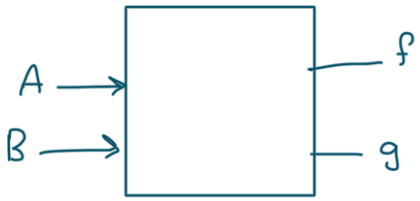


Second solution=

$g = cd + a'd$

- Multiple outputs
  → Same inputs, with different outputs.





2 Separat circuits

one circuit with Two outputs

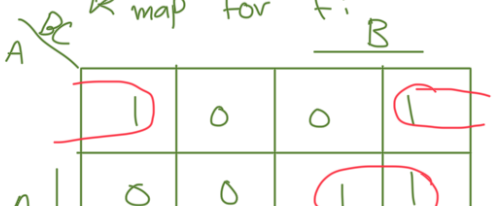
Example:

$f = \Sigma(0,2,6,7) = f(A,B,C)$.

$g = \Sigma(1,3,6,7) = g(A,B,C)$.

→ minimize each function Separatly:

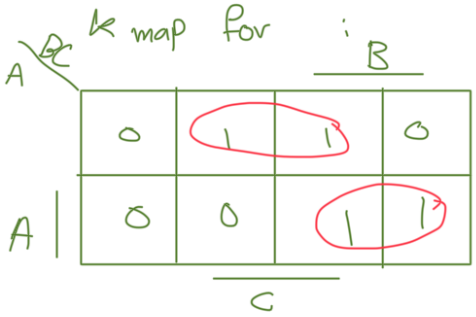
k map for f:



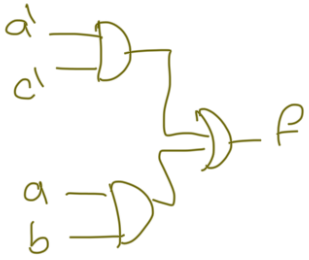
$f = a'c' + ab$

$g = a'c + ab$

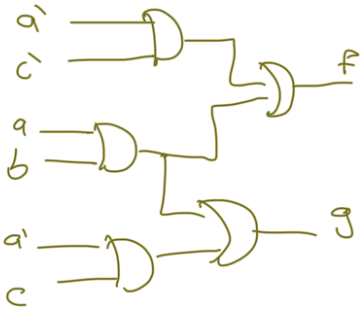
ab is a common term.



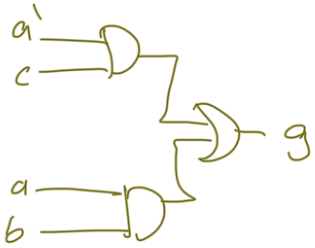
f:



- Using Common Term (one circuit)



g:



- Universal Gates:

← بنقدر نعمل منهم أي Gate ثانية.

1 Nand Gate:

- not  $F = A'$
- And  $A \cdot B$.
- OR  $A + B$

Nand Gate Symbol:

1 

And - invertor

2 

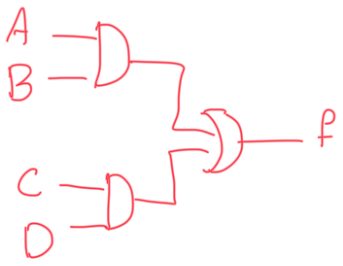
invertor - And.

Two Level Nand Implementation:

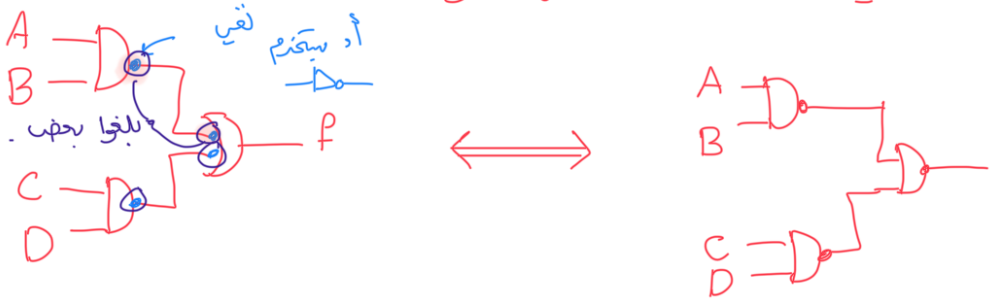
(SoP) And-OR بشرط تكونه Nand ← بقدر أحول أي سيركت لـ

Example:

$f = AB + CD$ Sum of product.

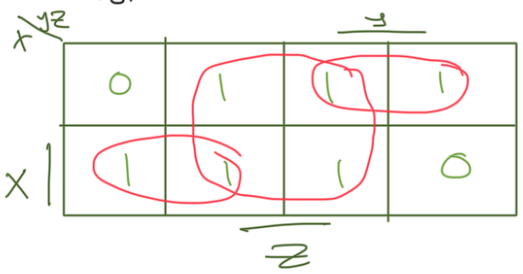


• عشان أحول لـ nand:
← بنفي الاوتبوت الـ and.
← بنفي الانبوت لـ OR.
← كل نفي بضيفه لازم يكون في نفي تاني للجنيه عشان ما يأثر على قيمة الفنكشن الأصلي.



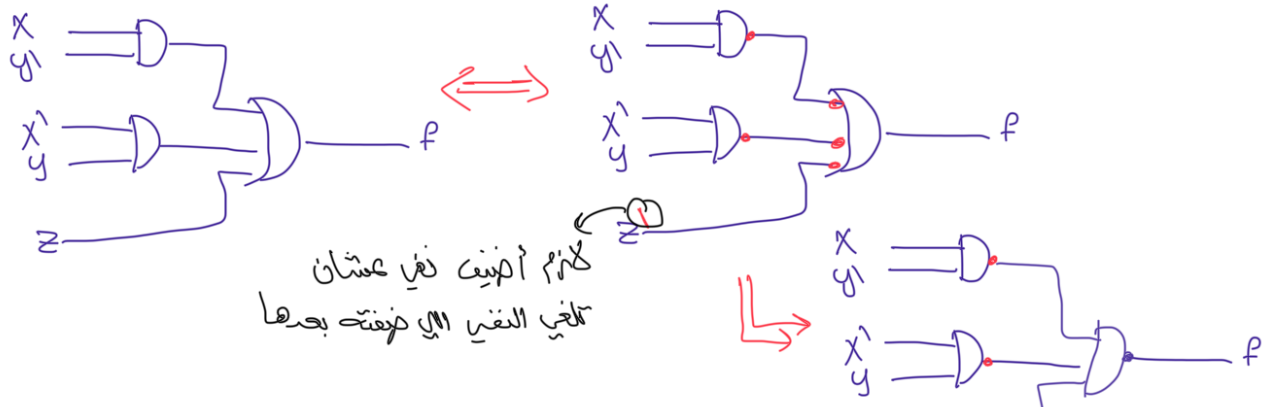
Example:

$F(x,y,z) = (1,2,3,4,5,7)$

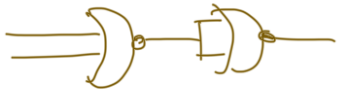


$f = xy' + x'y + z$

SOP.



2 NoR Gate:

- Not  $F = A'$
- OR  $f = A+B.$
- And 

NoR Gate Symbol:

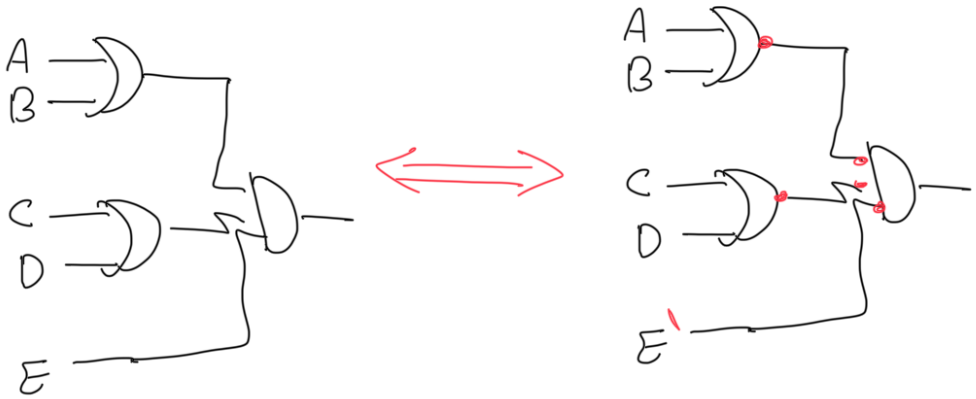
OR – inverter

inverter – And.

Two level NoR Implementation:

(POS) OR-And بشرط تكون NoR ← بقدر أحول اي سيرتك

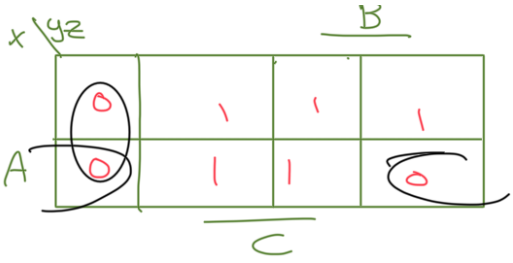
Example:



Example:

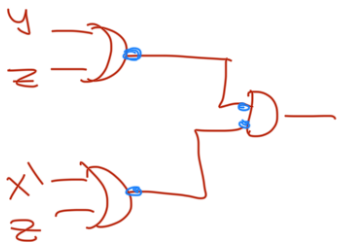
$F = \Sigma(1,2,3,5,7)$ Implement using NoR Gates:

OR – And



P.o.S
نحوّ أصفار

$$f' = y'z' + xz'$$
$$f = (y+z)(x'+z)$$



- Even and odd functions:

→ even function:

يكون الاوتبوت = ١ عندما يكون عدد الواحدات في الانبوت زوجي.

→ odd function:

يكون الاوتبوت =١ عندما يكون عدد الواحدات في الانبوت فردي.

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

1 only if number of one's is odd → odd function

w x y z	feven
0 0 0 0	1
0 0 0 1	0
0 0 1 0	0
0 0 1 1	1
0 1 0 0	0
0 1 0 1	1
0 1 1 0	1
0 1 1 1	0
1 0 0 0	0
1 0 0 1	1
1 0 1 0	1
1 0 1 1	0
1 1 0 0	1
1 1 0 1	0
1 1 1 0	0
1 1 1 1	1

1 only if the number of one's is even
∴ even function

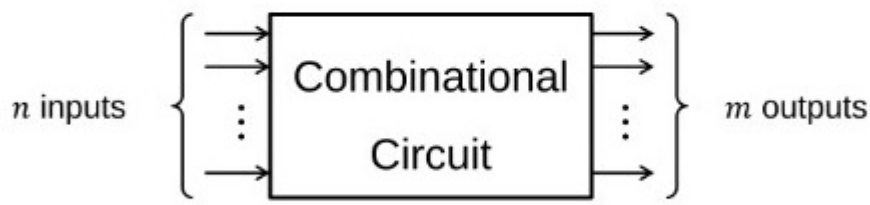
CHAPTER 4 :
COMBINATIONAL LOGIC DESIGN

قال تعالى:"إِنْ أَحْسَنتُمْ أَحْسَنتُمْ لِأَنفُسِكُمْ ۖ وَإِنْ أَسَأْتُمْ فَلَهَا"

إيمان الغلبان.

COMBINATIONAL CIRCUIT: → **ANALYSIS** / **DESIGN**

the output is a function of the input variables.



ANALYSIS:

given a circuit, find out the function.

DESIGN:

given a function, find out the circuit

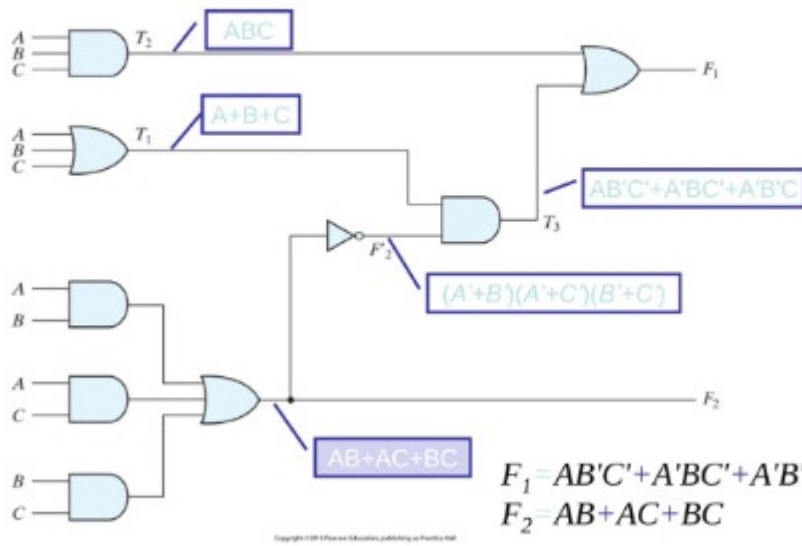
ينقسم التشابتر الى قسمين، القسم الاول هو ،analysis بتكون الدارة موجودة، وانا بطلع الـfunction،

ممكن اطلعه ع شكل اكسبرشن، او تروث تيبل

القسم الثاني هو ،design بتكون الـfunction موجود، وانا بطلع circuit،

ممكن يعطيني الفنكشن ع شكل اكسبرشن، او تروث تيبل

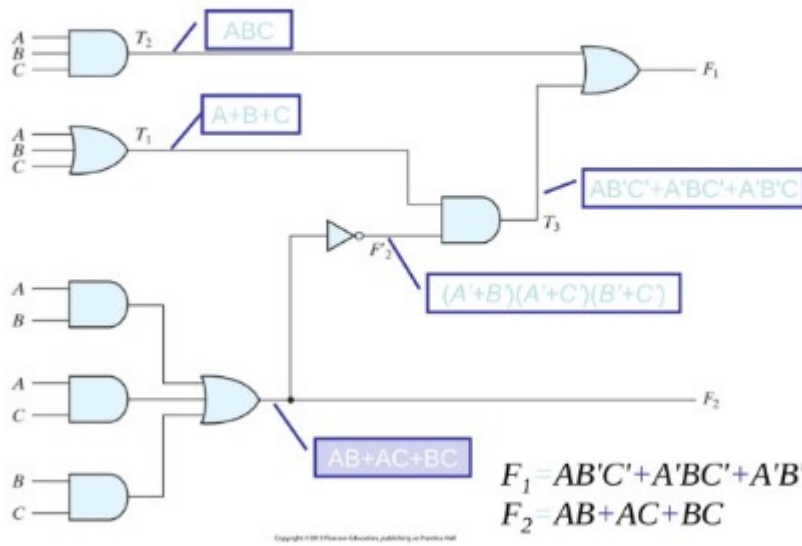
1- ANALYSIS:



$$f_1 = T_3 + T_2$$
$$= (T_1)(F_2') + T_2$$
$$= (A+B+C)(AB+AC+BC)' + T_2$$
$$= AB'C' + A'BC' + A'B'C + ABC$$

$$f_2 = AB + AC + BC.$$

خطوات التحليل:
بتتبع كل **logic gate** شو داخل الها وشو طالع منها، واذا كان عندي اكثر من **level** بقدر استخدم **labels** عشان اسهل ع حالي



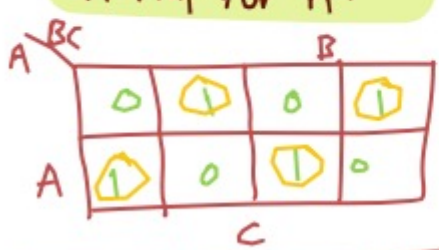
$$F_1 = AB'C' + A'BC' + A'B'C + ABC$$
$$F_2 = AB + AC + BC$$

Truth Table:

A	B	C	F_2	f_1
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

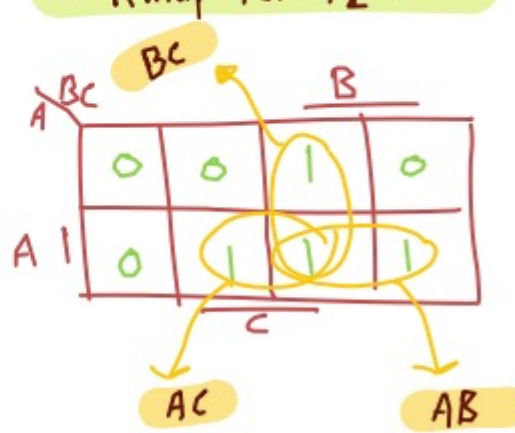
عشان اطلع الـ simplified expression
بقدر اسوي اول اشي الـ truth table ويعدين الـ k-map واوصل بالاخر للـ expression

K-map for f_1:



$$F_1 = AB'C' + ABC + A'B'C + A'BC'$$

Kmap for f_2:



$$F_2 = AC + AB + BC$$

2- DESIGN

خطوات التصميم:

١- افهم شو وصف السيركت المطلوب.
٢- احدد الاوتبوت والانبوت وشو عددهم.
٣- احدد اللوجيك اللي بدي امشي عليه.
٤- تروث تيبل و k map
٥- اصمم السيركت حسب الفنكشن اللي طلع معي.
٦- اتحقق من السيركت اذا شغالة صح

Designing a BCD to Excess-3 Code Converter

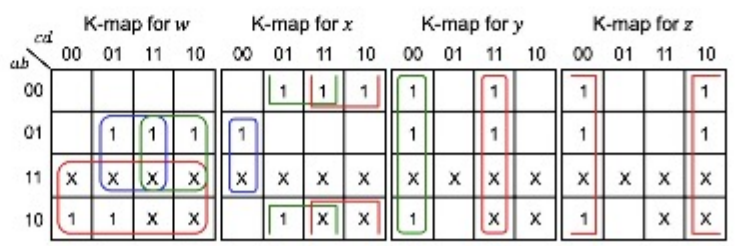
- 4 Bit input
- Range = (0-9)

- 4 Bit output
- Excess-3 = Binary + 3.

- Truth Table:

BCD a b c d	Excess-3 w x y z
0 0 0 0	0 0 1 1
0 0 0 1	0 1 0 0
0 0 1 0	0 1 0 1
0 0 1 1	0 1 1 0
0 1 0 0	0 1 1 1
0 1 0 1	1 0 0 0
0 1 1 0	1 0 0 1
0 1 1 1	1 0 1 0
1 0 0 0	1 0 1 1
1 0 0 1	1 1 0 0
1010 to 1111	X X X X

K-map for each output.



Minimal Sum-of-Product expressions:

$w = a + bc + bd$, $x = b'c + b'd + bc'd'$, $y = cd + c'd'$, $z = d'$

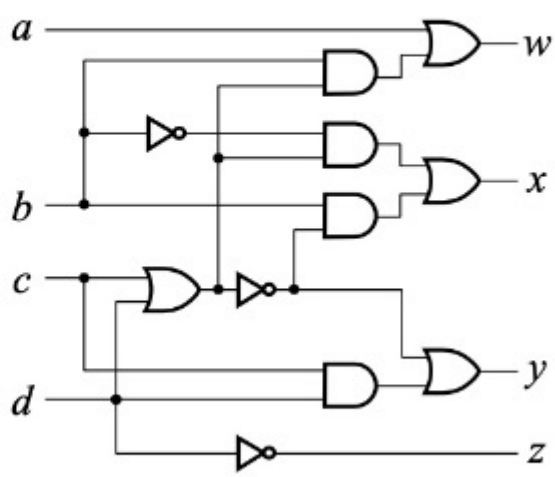
- extract common term (c+d).

$w = a + b(c+d)$

$x = b'(c+d) + b(c+d)'$

$y = cd + (c+d)'$

$z = d'$



2- DESIGN

Hierarchical Design

يعني استخدم دارات بسيطة عشان اوصل دارات معقدة

BINARY ADDER-SUBTRACTOR

عشان اوصل لدارة تجمعلي او تطرحلي n bits انا بحاجة لدارة ابسط تجمع او تطرح 1 bit

- Half Adder
  - Adds 1-bit plus 1-bit → inputs.
  - Produces Sum and Carry → output

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

* ملاحظة:

← عدد الـ Bits في الاوتبوت = عدد الـ Bit في الانبوت +1

2- DESIGN

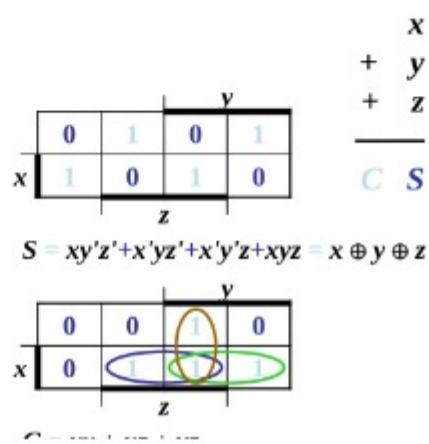
Hierarchical Design

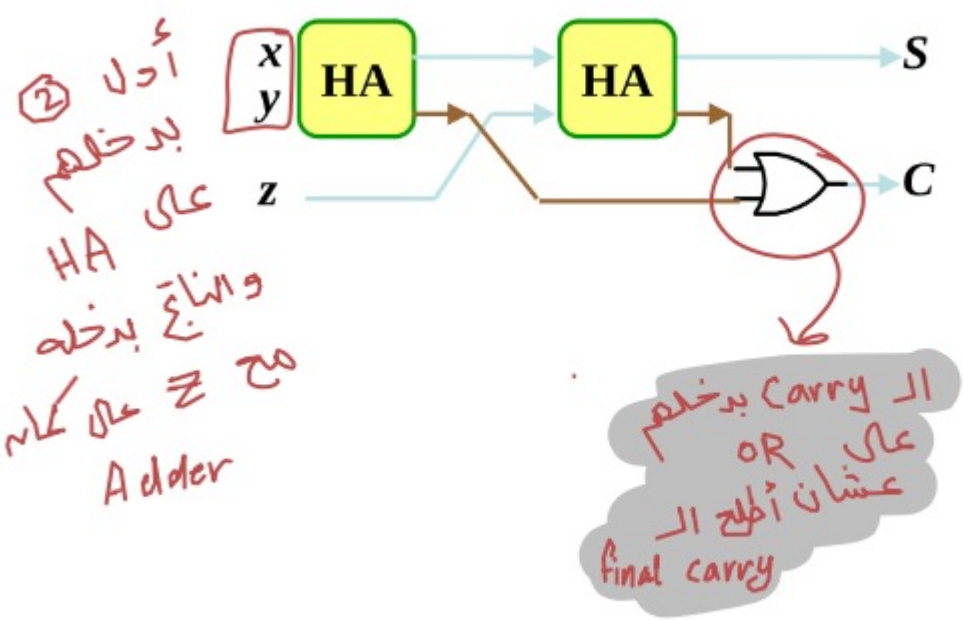
• عشان بقدر استخدم الـ HA عشان أعمل سيركت بجمع أي عدد من الـ bits (Full Adder)

full adder:

- Adds 1-bit plus 1-bit plus 1-bit
- Produces Sum and Carry

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

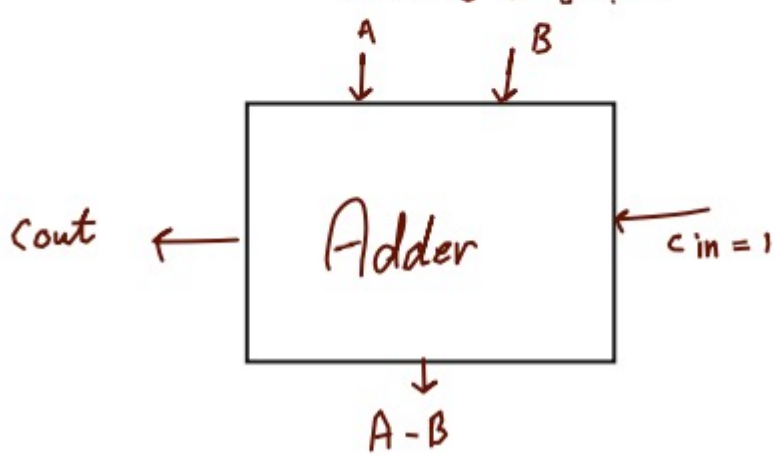




2- DESIGN

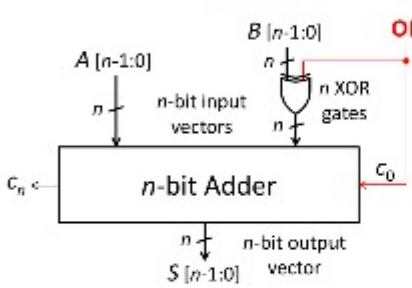
Converting Subtraction into Addition

• بقدر أستخدم الـ Adder عشان أطرح،
← بحوّل الطرح لجمع (بجمع الـ 2's complement)
← بدخّل أول carry = 1



Adder/Subtractor for 2's Complement

❖ Two operations: **OP = 0 (ADD)**, **OP = 1 (SUBTRACT)**



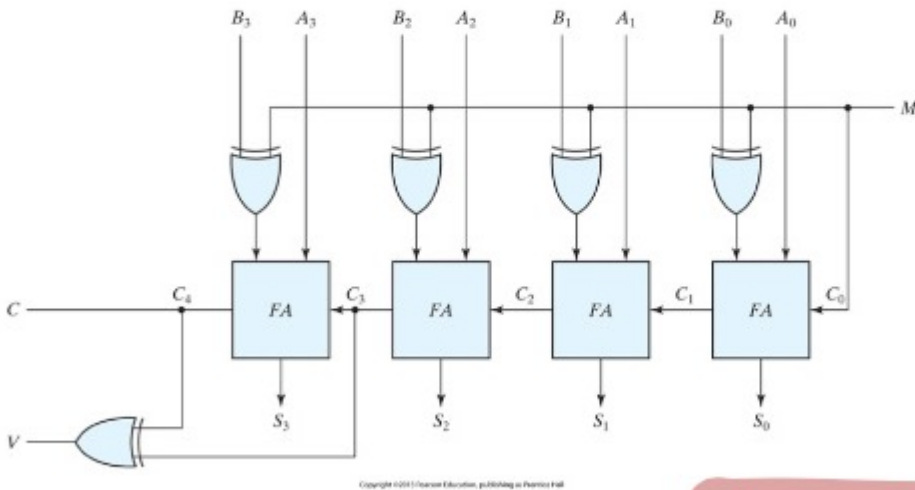
OP = 0 (ADD)
B XOR 0 = B
S = A + B + 0 = A + B

OP = 1 (SUBTRACT)
B XOR 1 = 1's complement of B
S = A + (1's complement of B) + 1
S = A + (2's complement of B)
S = A – B

• عشان استخدم نفس السيركت تجمع وتطرح:
← بزيد ابنوت (OP) بتحكم بالـ carry سبب شو العملية
← بزيد XOR بدخّل عليها (OP) والعدد الثاني.

2- DESIGN

Four-bit adder-subtractor



* مهم جدن جدن

• Subtraction:

→ m = 1

→ Subtraction Result = $CS_3S_2S_1S_0$

working with unsigned numbers

• Addition:

→ m = 0

→ Sum = $CS_3S_2S_1S_0$

• 2's complement:

→ m = 1

→ 2's complement = $S_3S_2S_1S_0$

→ $A_3A_2A_1A_0$ = 0000

→ $B_3B_2B_1B_0$ = الرقم اللي بدي أسويله comp

• باختصار: اللي بدّي أسويله comp بدخله عالـ XOR والبيرة الثاني أصفار

• working with signed:

• بنتبه عال نتيجة الأوفرفلو (V)

2- DESIGN

DECIMAL ADDER (BCD Adder)

• جمع الـ BCD عبارة عن جمع Binary ، بس اذا صار عندي error برجع بجمع 6.

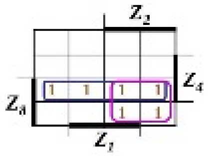
Sum>9 invalid for BCD

carry

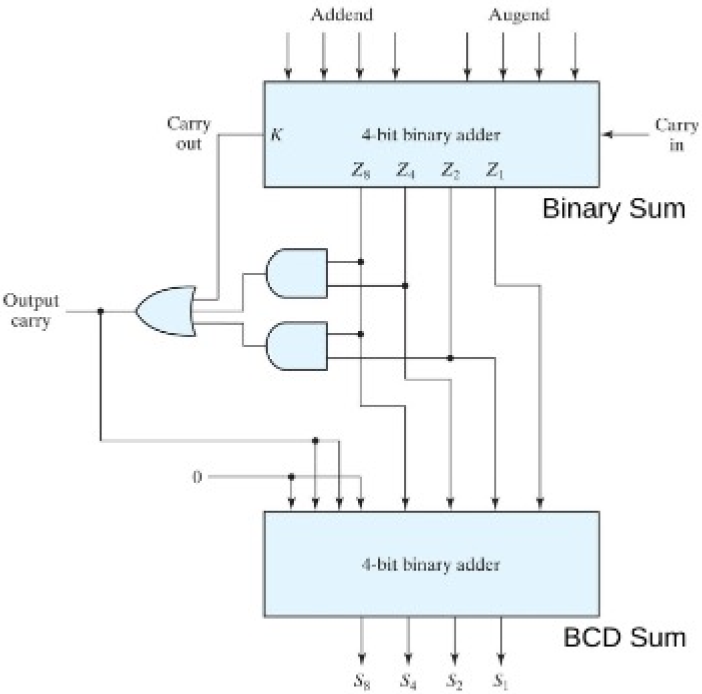
4 bits 0-9

رح يكون عندي 2 Adder

$Z_8 Z_4 Z_2 Z_1$	Err
0 0 0 0	0
1 0 0 0	0
1 0 0 1	0
1 0 1 0	1
1 0 1 1	1
1 1 0 0	1
1 1 0 1	1
1 1 1 0	1
1 1 1 1	1



$$Err = Z_8 Z_4 + Z_8 Z_2$$



2- DESIGN

Two-bit by two-bit binary multiplier

$B_1 \quad B_0$

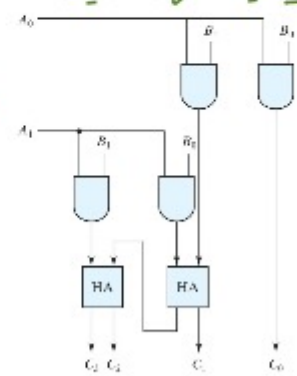
$A_1 \quad A_0$

$A_0B_1 \quad A_0B_0$

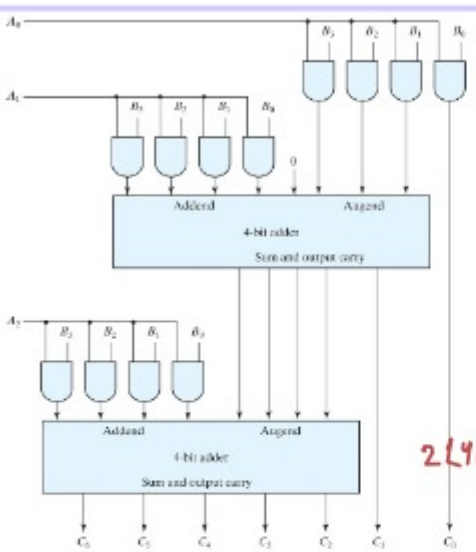
$A_1B_1 \quad A_1B_0$

$A_1B_1 \quad A_0B_1 + A_1B_0 \quad A_0B_0$

طريقة الضرب نفس طريقة الضرب العادي،
في السيركيت في n bit adders و and gates:



Four-bit by three-bit binary multiplier



مصمم جبن جبن جبن:

- اذا أنا شغال في (m x n bits) بقدر أعرف:

1- عدد الـ Bits في الآوتبوت = $Z = m+n$

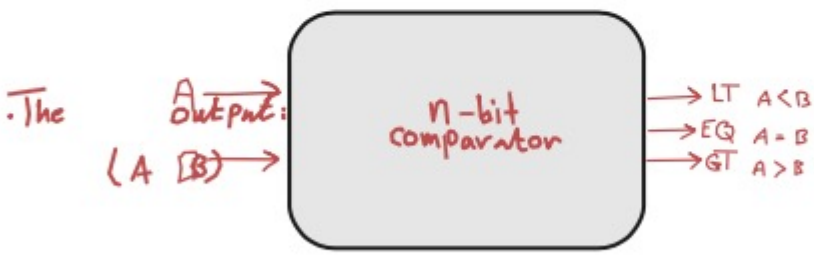
2- عدد الـ And Gates = $12 = m \times n$

3- عدد الـ Adder ونوعهم = 2 (4bit Adder) $= n-1$ (m bit Adder)

2- DESIGN

Magnitude Comparator

- 2 (n-bit) inputs.
- 3 outputs. (only one is 1 and the others are Zeros)



inputs = A = $A_4 A_3 A_2 A_1$
B = $B_4 B_3 B_2 B_1$

* unsigned numbers الحاكي عن

The EQ output: (A=B)

عشان أتقدر أحكم إنهم متساويين
لازم كل منازل A تساوي منازل B

$EQ \leftrightarrow A_3 = B_3$ and $A_2 = B_2$ and $A_1 = B_1$ and $A_0 = B_0$

$EQ = eq_3\ eq_2\ eq_1\ eq_0$

$eq_i = A_i B_i + A_i' B_i'$

The LT output: (A<B)

عشان أحكم إنه A<B :

أول منزلة A_3 أصغر من أقل منزلة B_3 ←

A<B بحكم

إذا أول منزلة متساوية بنارن اللي بعدها ←

$LT = LT_3 + eq_3 LT_2 + eq_3 eq_2 LT_1 + eq_3 eq_2 eq_1 LT_0$

$LT_i = A_i' B_i$
$(A=0, B=1)$

The GT output:

عشان أحكم إنه A>B :

أول منزلة A_3 أكبر من أقل منزلة B_3 ←

A>B بحكم

إذا أول منزلة متساوية بنارن اللي بعد ←

$GT = GT_3 + eq_3 GT_2 + eq_3 eq_2 GT_1 + eq_3 eq_2 eq_1 GT_0$

$GT_i = A_i B_i'$
$(A=1, B=0)$

إذا بدنا نقارن Signed numbers

بنتبع أول اشي بالإشارة (أول منزلة) ←

1 1 1 0

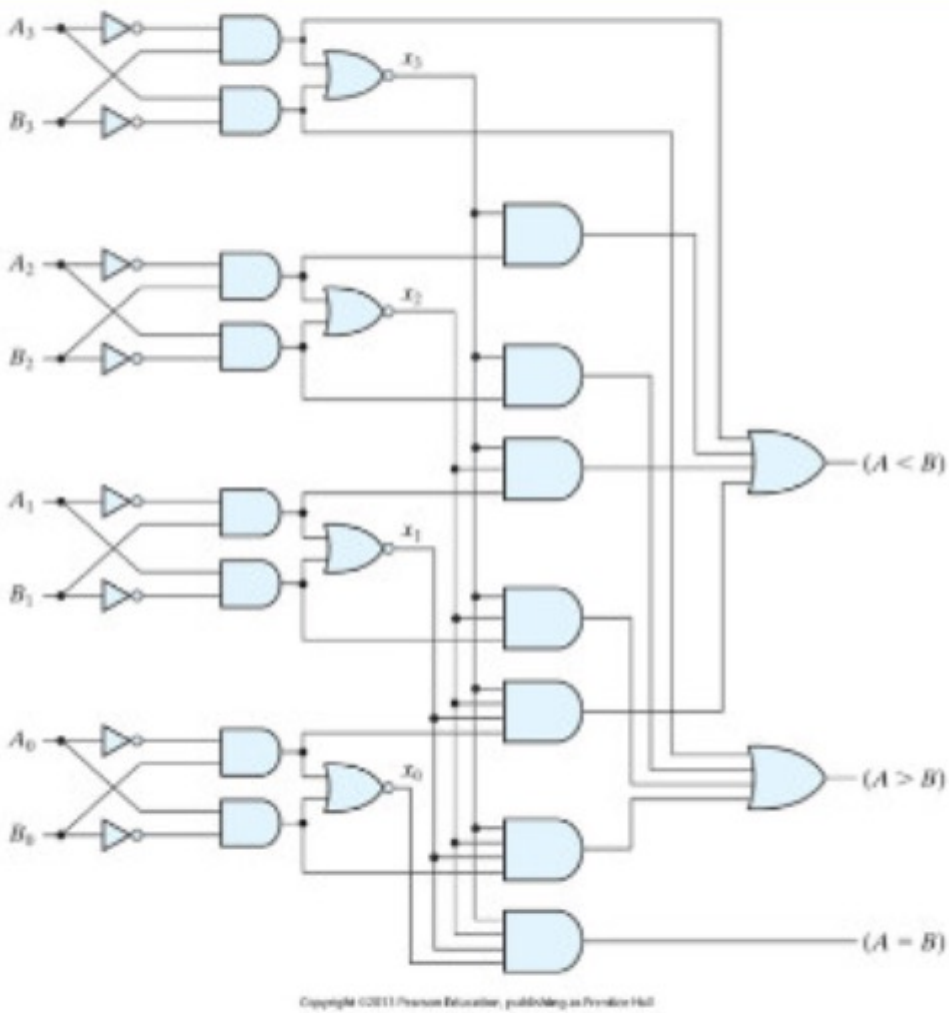
← إذا العددين موجبين بتجي بنفس الموجيك.

← إذا العددين سالبين بتعكس الموجيك

← إذا واحد سالب وواحد موجب، بحكم الموجب أكبر

2- DESIGN

Magnitude Comparator

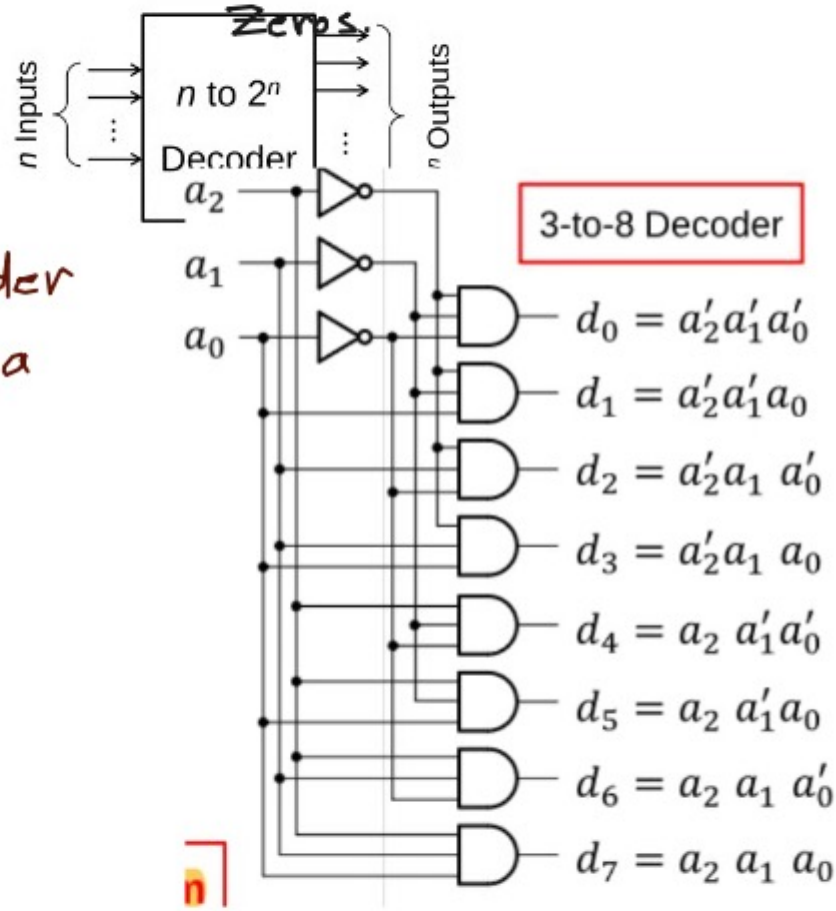


2- DESIGN

Binary Decoders

n inputs, 2^n outputs.

only one is 1 and the others are zeros.



- each decoder output is a minterm.

2- DESIGN

Binary Decoders

active high.
← بشتغل عادي .

active low
← بنعكس الاوتبوت
صفر واحد والباقي واحدات

Using Decoders to Implement Functions

بقدر أسوي implementation لأي فنكشن باستخدام Decoder الـ :
external gate. +

- active high decoder:
  - → OR gate (with minterms)
  - → NOR gate (with maxterms)
- active low decoder:
  - → nand gate (with minterms).
  - → and gate (with maxterms)

* ملاحظة مهمة جدن:
← لازم اختار اللي عدده أقل بين الـ minterms/maxterms

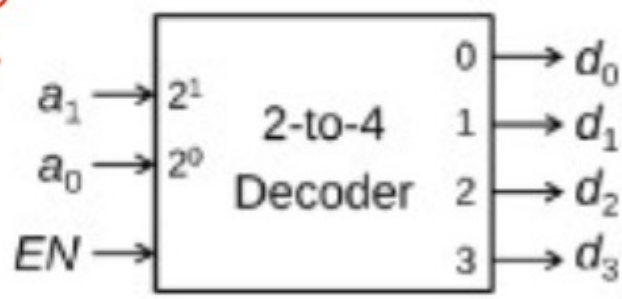
2- DESIGN

2-to-4 Decoder with Enable Input

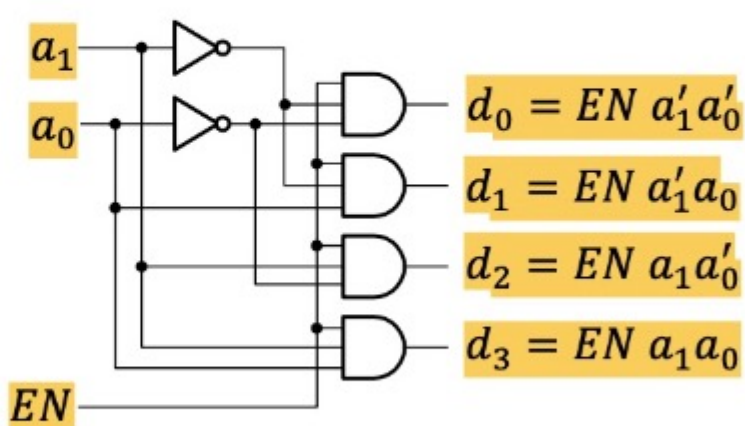
- Additional input (En)

En = 0 output = 000

En = 1 works.



Inputs		Outputs			
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

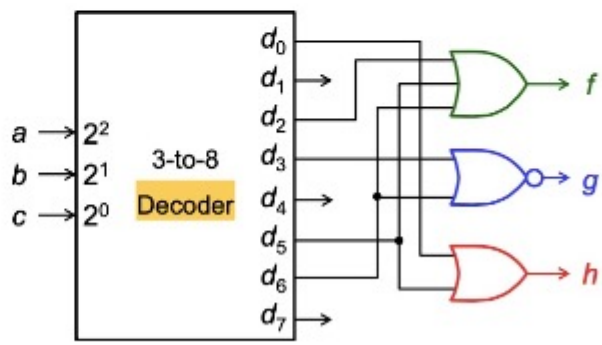


2- DESIGN

Using Decoders to Implement Functions

❖ **Example:** $f = \Sigma(2, 5, 6)$, $g = \Pi(3, 6) \rightarrow g' = \Sigma(3, 6)$, $h = \Sigma(0, 5)$

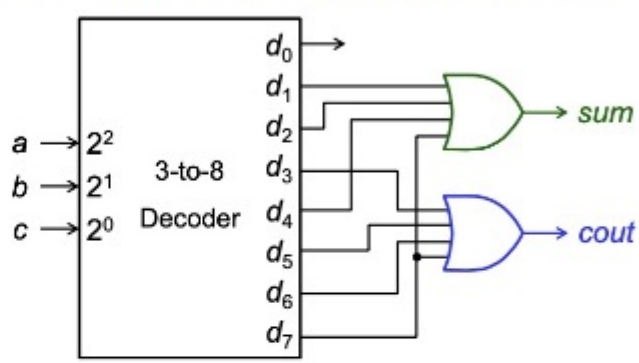
Inputs			Outputs		
a	b	c	f	g	h
0	0	0	0	1	1
0	0	1	0	1	0
0	1	0	1	1	0
0	1	1	0	0	0
1	0	0	0	1	0
1	0	1	1	1	1
1	1	0	1	0	0
1	1	1	0	1	0



Note: the function must not be minimized

❖ **Example:** Full Adder sum = $\Sigma(1, 2, 4, 7)$, cout = $\Sigma(3, 5, 6, 7)$

Inputs			Outputs	
a	b	c	cout	sum
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



2- DESIGN

Building Larger Decoders

❖ A 3-to-8 decoder can be built using:

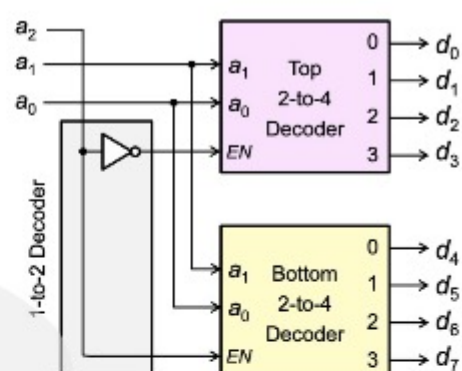
Two 2-to-4 decoders with Enable and an inverter (1-to-2 decoder)

2 (2 to 4 decoders)

Inputs			Outputs							
a_2	a_1	a_0	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

The first works when $a_2 = 0$

The second works when $a_2 = 1$

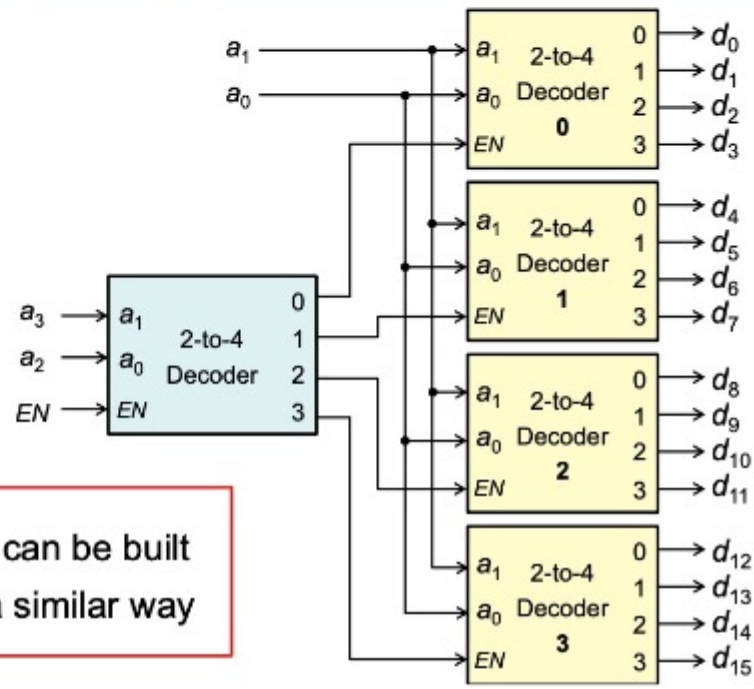


- The most significants are for the enables.
- The least significants are the inputs for the smaller decoder.

2- DESIGN

Building Larger Decoders

A 4-to-16 decoder with enable can be built using **five** 2-to-4 decoders with enables



Larger decoders can be built hierarchically in a similar way

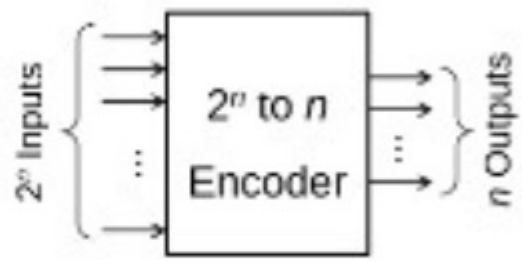
2- DESIGN

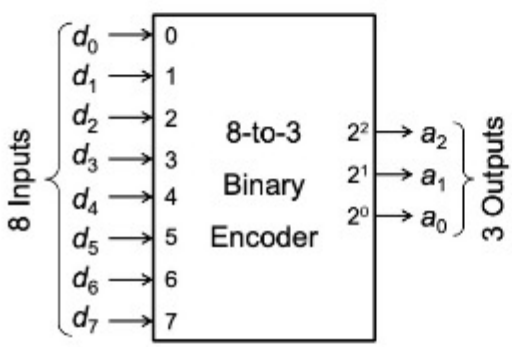
Encoders

بشتغل عكس الـ decoder.

→ 2^n inputs

→ n outputs.

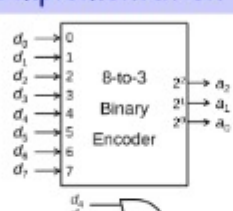




Inputs								Outputs		
d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	a_2	a_1	a_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

8-to-3 Binary Encoder Implementation

Inputs								Outputs		
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



$a_2 = d_4 + d_5 + d_6 + d_7$

$a_1 = d_2 + d_3 + d_6 + d_7$

$a_0 = d_1 + d_3 + d_5 + d_7$

8-to-3 binary encoder implemented using three 4-input OR gates

• إذا كان الـ output فيه أكثر من 1 أو كلّه أصفار بعطيني غلط.

2- DESIGN

Priority Encoder

→ additional output V
to deal with errors

Inputs			L.p	Outputs		
d_3	d_2	d_1	d_0	a_1	a_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

Highest Priority ←

اذا الابيوت فيه على الاقل 1 ← V=1

• بحدد مين الـ Highest Priority

• اذا الـ high = 1 اللي أقل منه بكونوا don't cares

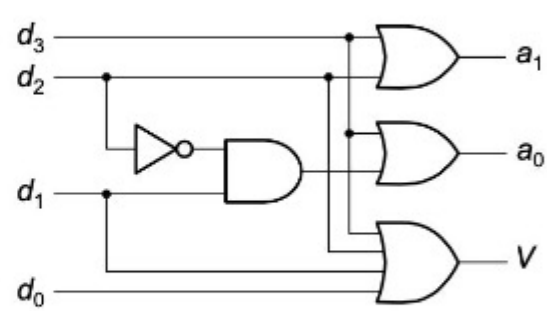
وهو اللي بحدد الاوتبوت.

Output Expressions:

$$a_1 = d_3 + d_2$$

$$a_0 = d_3 + d_1\, d_2'$$

$$V = d_3 + d_2 + d_1 + d_0$$

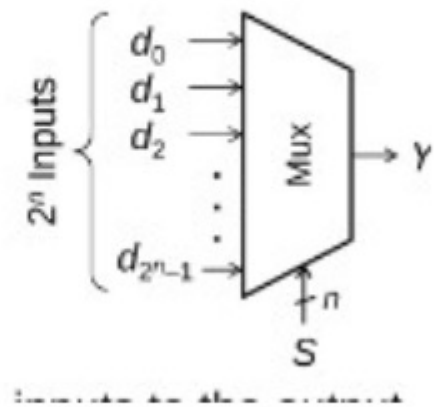


2- DESIGN

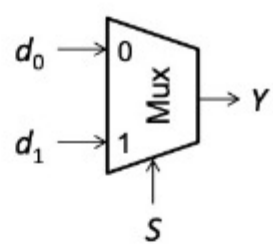
Multiplexers

- 2^n inputs.
- n selection.
- 1 output.

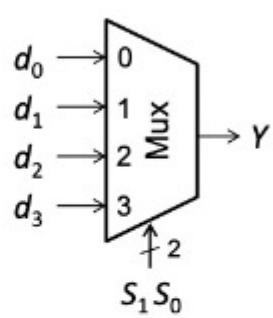
- حسب الـ Selection واحد من الانبوتس يروح عـ الاوتبوت



Inputs			Output
S	d_0	d_1	Y
0	0	X	$0 = d_0$
0	1	X	$1 = d_0$
1	X	0	$0 = d_1$
1	X	1	$1 = d_1$



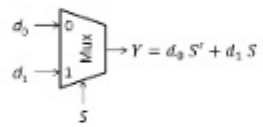
Inputs						Output
S_1	S_0	d_0	d_1	d_2	d_3	Y
0	0	0	X	X	X	$0 = d_0$
0	0	1	X	X	X	$1 = d_0$
0	1	X	0	X	X	$0 = d_1$
0	1	X	1	X	X	$1 = d_1$
1	0	X	X	0	X	$0 = d_2$
1	0	X	X	1	X	$1 = d_2$
1	1	X	X	X	0	$0 = d_3$
1	1	X	X	X	1	$1 = d_3$

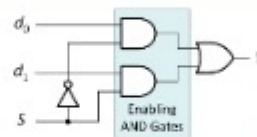


2- DESIGN

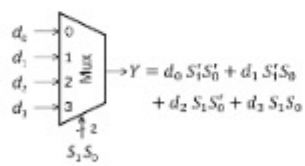
Multiplexers

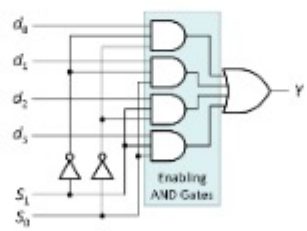
Implementing Multiplexers





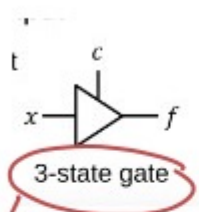
- عدد الـ and = عدد الانبوتس
- OR وحدة عدد الانبوتس الها يساوي عدد الانبوتس للـ mux.





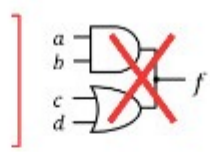
كل ما زاد عدد الانبوتس الدايركت بتلبس بحاجة Gates أبطأ

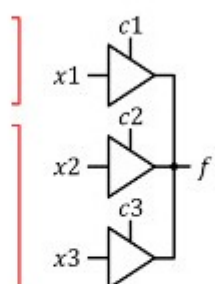
3-State Gate



$c = 0 \quad f = Z$ high Impedence

$c = 1 \quad f = x$





c1	c2	c3	f
0	0	0	Z
1	0	0	x1
0	1	0	x2
0	0	1	x3
0	1	1	Burn
1	0	1	Burn
1	1	0	Burn
1	1	1	Burn

- بقدر أشبك الـ outputs مع بعض بشرط ← واحد منهم بس يكون 1 والباقي يكون أصفار.

2- DESIGN

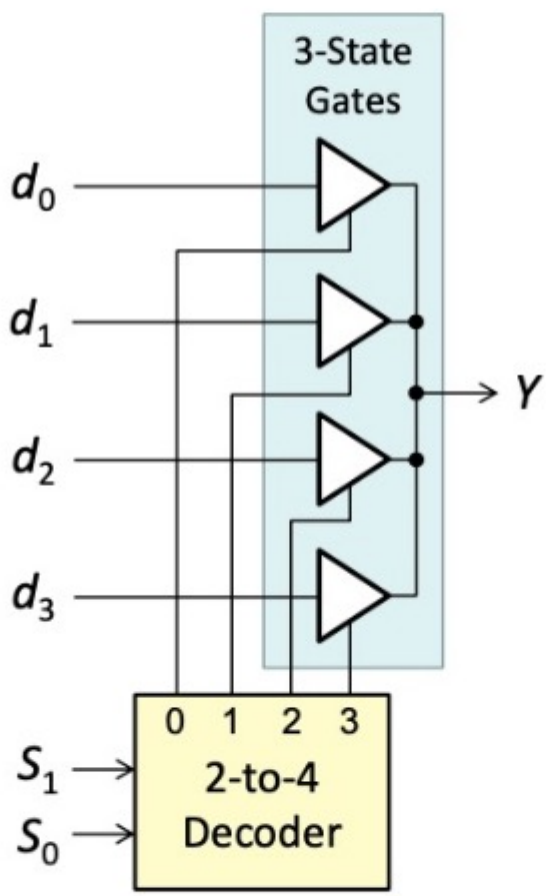
Multiplexers

Implementing Multiplexers with 3-State Gates

A Multiplexer can also be implemented using:

1. A decoder
2. Three-state gates

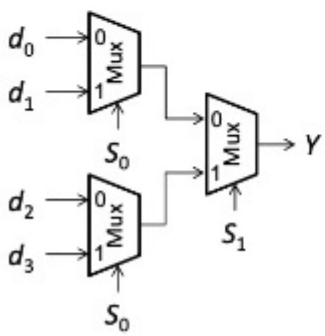
عشان تكون السيركيت أبسط



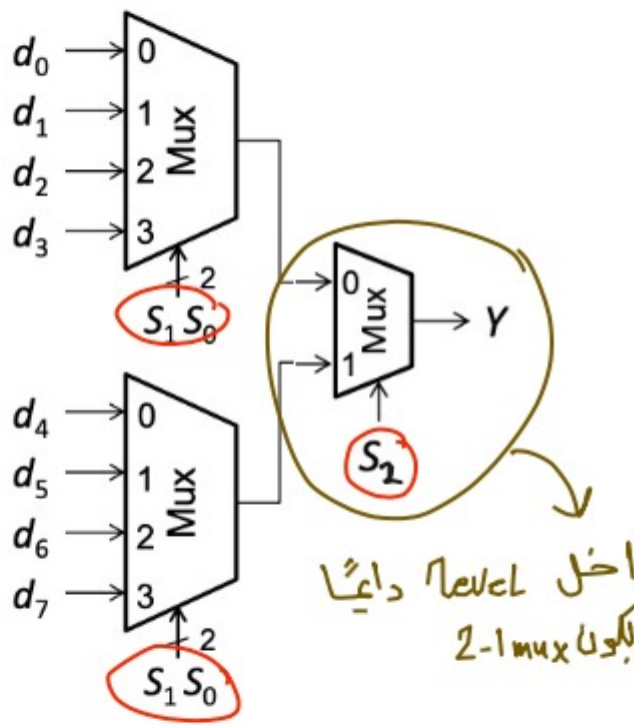
2- DESIGN

Multiplexers

Building Larger Multiplexers



Building 4-to-1 Mux using three 2-to-1 Muxes



اخل level دايمًا بكون 2-1 mux

- الـ least selections بكونوا في أول level
- والـ most selection بكون لآخر level.

2- DESIGN

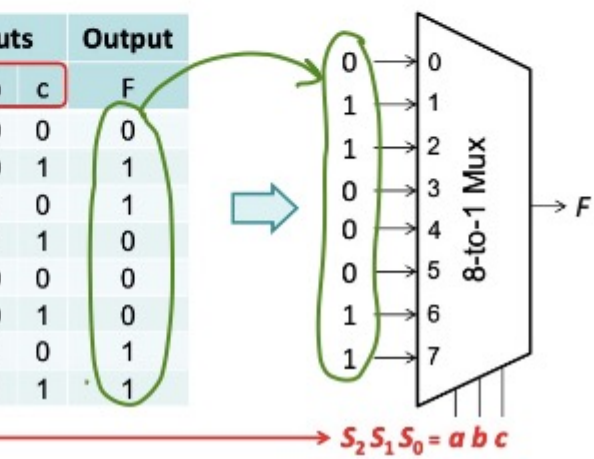
Multiplexers

Implementing a Function with a Multiplexer

❖ **Solution:**

The inputs are used as select lines to a Mux. An 8-to-1 Mux is used because there are 3 variables

Inputs			Output
a	b	c	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



- To implement a function using a mux:
  - → function inputs are the selections for the mux.
  - → function outputs are the mux inputs

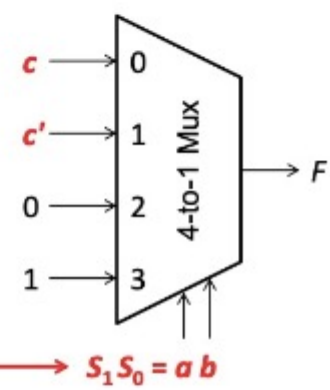
- الطريقة هاي صح بس كل ما زاد انبوتس الـfunction بصير احتاج mux أكبر.
- بيسهل الـ Implementation عن طريق تقليل عدد الـ inputs اللي رح أختارهم Selections.

2- DESIGN

Multiplexers

Implementing a Function with a Multiplexer

Inputs			Output	Comment
a	b	c	F	F
0	0	0	0	F = c
0	0	1	1	
0	1	0	1	F = c'
0	1	1	0	
1	0	0	0	F = 0
1	0	1	0	
1	1	0	1	F = 1
1	1	1	1	



- هون استخدمنا mux أصغر لنفس الـ function عن طريق تقليل عدد الـ Selections.

- عشان أحدد الـ mux inputs: بشوف شو العلاقة بين الـ inputs (اللي ما أختارتهم) والـ output

*** ملاحظة مهمة جدن جدن:**

① فش اشي بلزمني أي inputs أختارهم Selections بس الأسهل واني أدخل الـ most significant

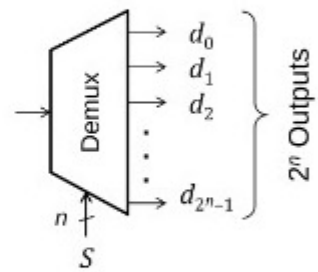
② إذا ما طلب مني mux معيّن بوخذ الـ Selections = inputs - 1

2- DESIGN

Demultiplexer

بشتغل عكس الـ mux.

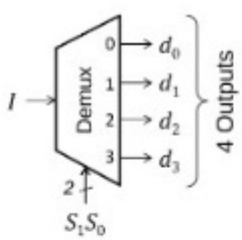
→ 1 input
→ n Selections
→ 2^n outputs

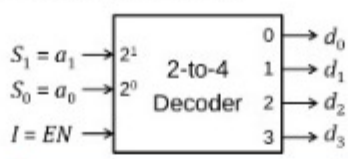


• حسب الـ S الانبوت.. بدخ على أحد الاديبوتس

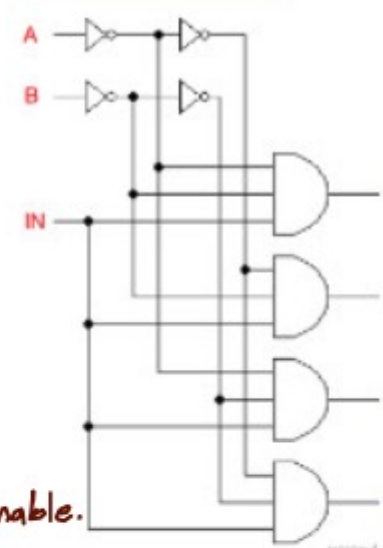
1 - 4

Demux Input *I* is equivalent to Decoder Enable *EN*





Think of a decoder as directing the Enable signal to one output

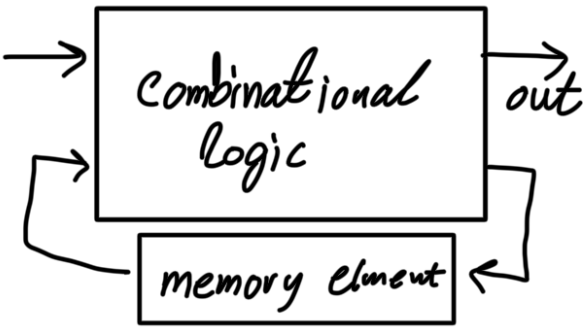


* شغل الـ demux نفس شغل decoder with enable

demux input = decoder's enable.
demux Selections = decoder's inputs.
demux output = decoder outputs.

chapter 5: Synchronous Sequential logic.

- يحتوي على memory element



- الـ output يعتمد على input + previous state.

Sequential Circuit:

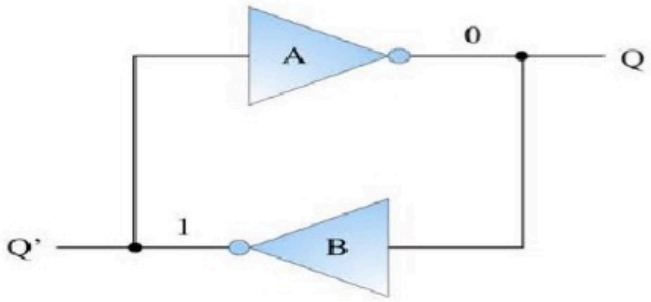
Synchronous.
- additional input (clock) to control the changes in the memory elements.

Asynchronous.
- no additional inputs, so the changes may happen at any time.

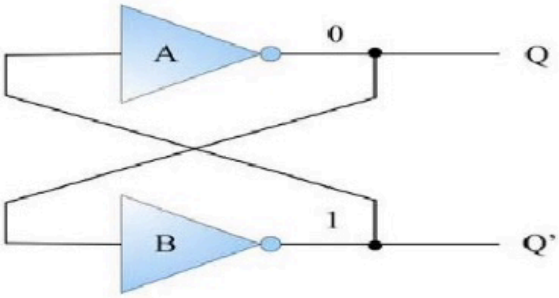
The clock Signal:

- level Sensitive.
  - → positive pulse.
  - → negative pulse.
  - Latch
- edge trigger.
  - → positive edge.
  - → negative edge.
  - flip-flop.

• latches:



(a)

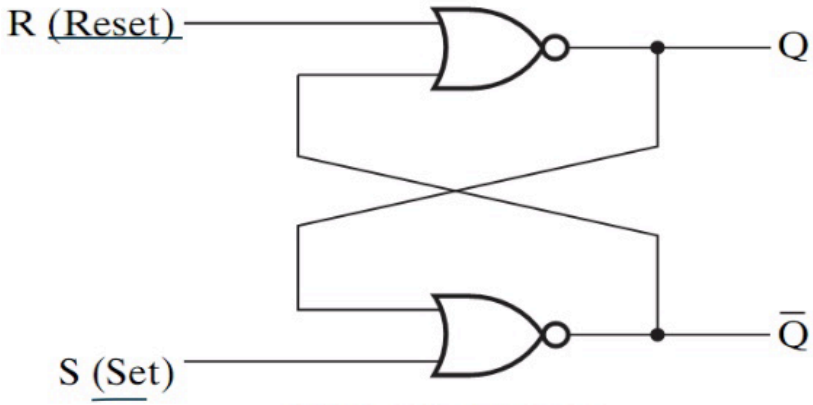


(b)

- أول نوع من الـ Latches عبارة عن 2 Inverters الانبوت لكل واحد عبارة عن الاوبتوت للثّاني .

- مشكلة هاد النّوع إنّه الـ State بتضل ثابتة ما بتتغير

• SR Latches:



(a) Logic diagram

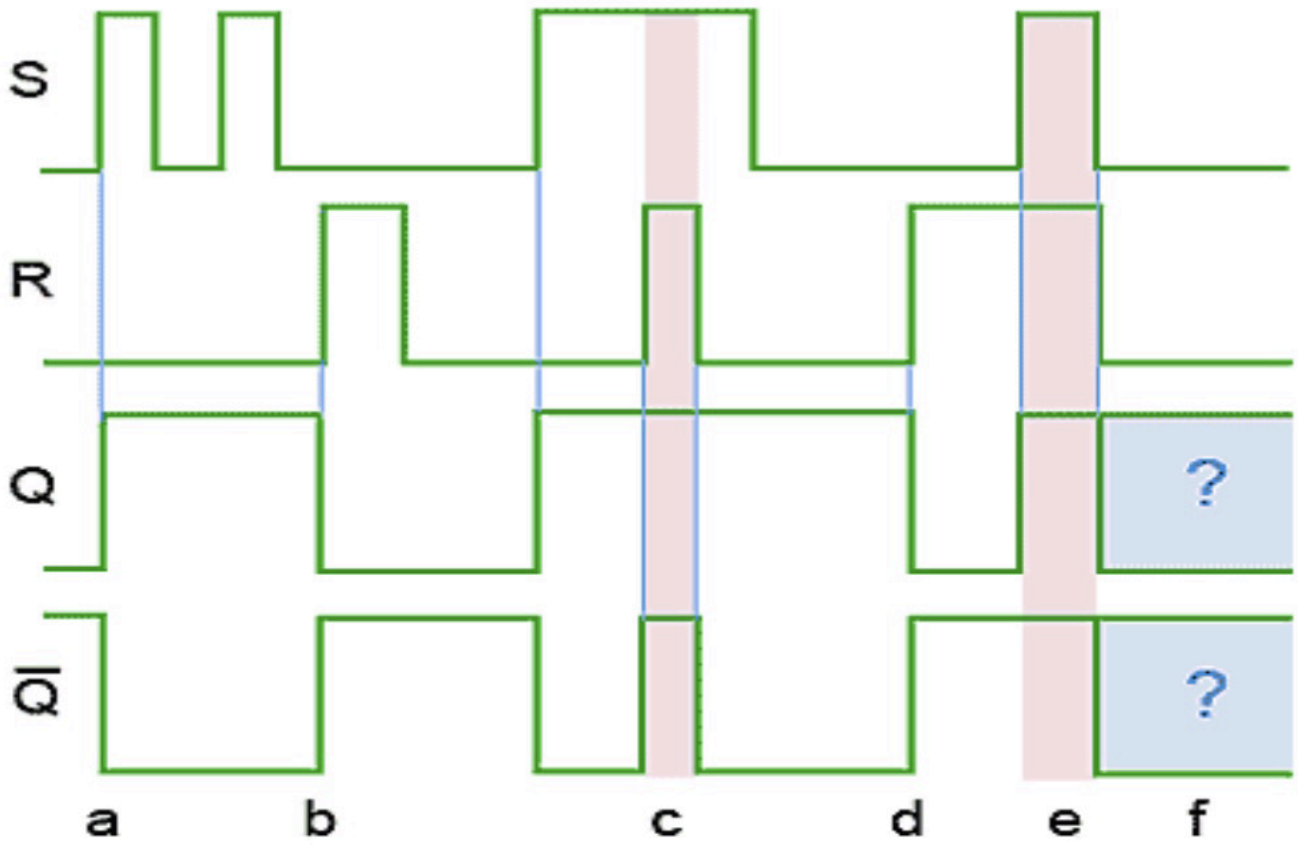
S	R	Q	$\overline{Q}$	
1	0	1	0	Set state
0	0	1	0	
0	1	0	1	Reset state
0	0	0	1	
1	1	0	0	Undefined

(b) Function table

SR latch: 2 nor gates

output: no changes.
Reset.
Set.
undefined.

SR latch مشكلة ← undefined.
وانه اذا كان الانبوت ١١
الاوبتوت يكون

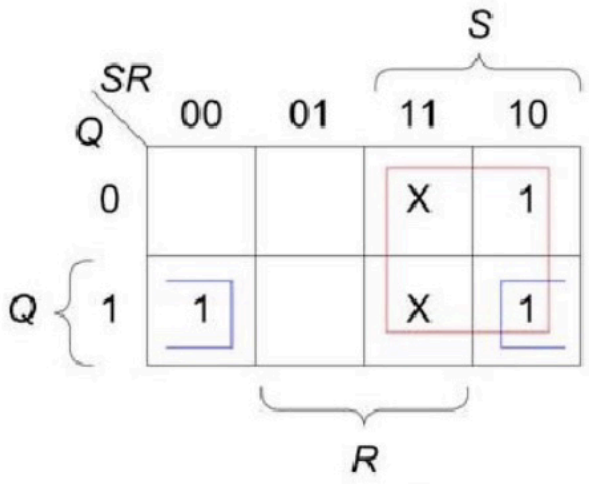


• بما انّ الـ Latch عبارة عن Level Sensitive التغيير على الـ State ممكن يكون بأي لحظة على طول الـ Level

• Characteristic Equation of the SR Latch:

معادلة بتربط بين الـ output والـ Input والـ Present State

Q(t)	S	R	Q(t + 1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	Indeterminate
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	Indeterminate



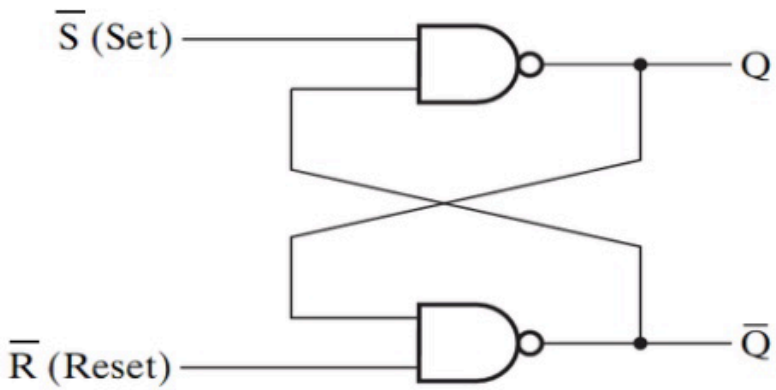
$Q(t+1) = S + R'Q$ $SR = 0$

• S'R' latch with NaNd Gates:

→ Nand Gates.

→ works active low. (when S=1 Q=0).

→ when inputs = 00 → Q is undefined.



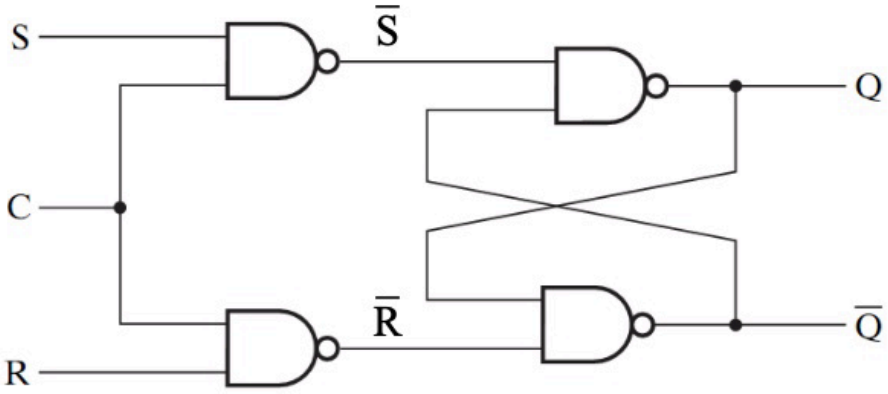
(a) Logic diagram

$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	
0	1	1	0	Set state
1	1	1	0	
1	0	0	1	Reset state
1	1	0	1	
0	0	1	1	Undefined

(b) Function table

if we added a clock Input :

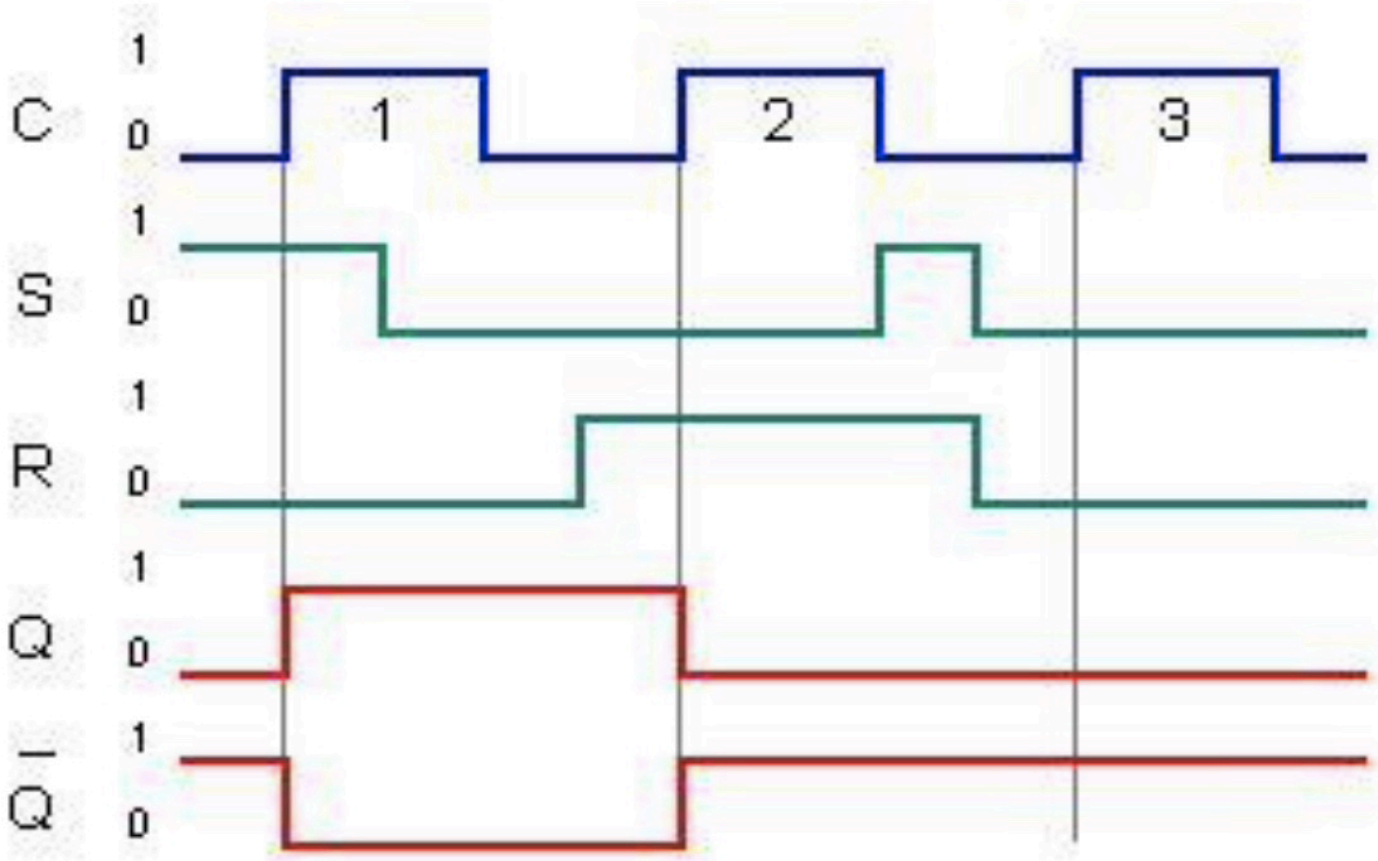
if C = 0 → no changes.
C = 1 → works as normal SR latch
↳ active high



(a) Logic diagram

C	S	R	Next state of Q
0	X	X	No change
1	0	0	No change
1	0	1	Q = 0; Reset state
1	1	0	Q = 1; Set state
1	1	1	Undefined

(b) Function table

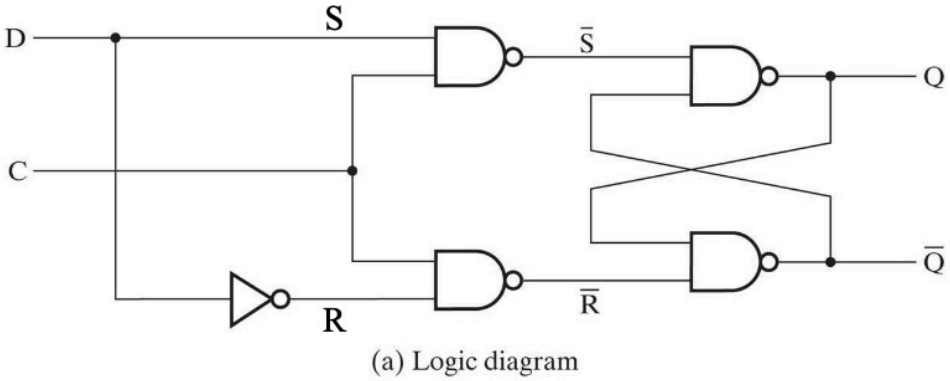


- **D - latch with clock input:**

→ one input with a clock.
→ if C = 0 → no change.
C = 1 → Q(t+1) = D.
↳ D = next State !!

- **Characteristic equation:**

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

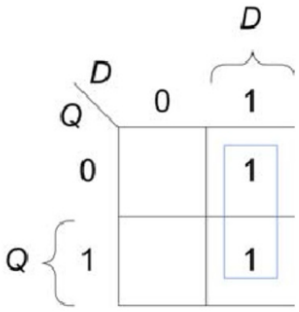


(a) Logic diagram

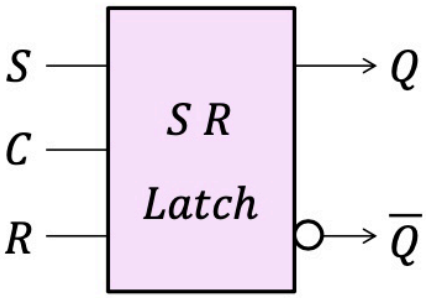
C	D	Next state of Q
0	X	No change
1	0	Q = 0; Reset state
1	1	Q = 1; Set state

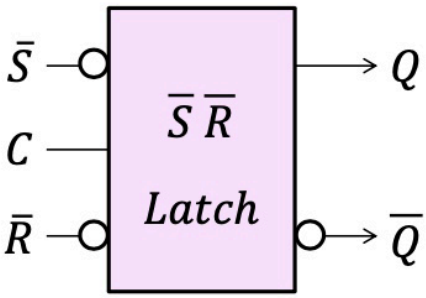
(b) Function table

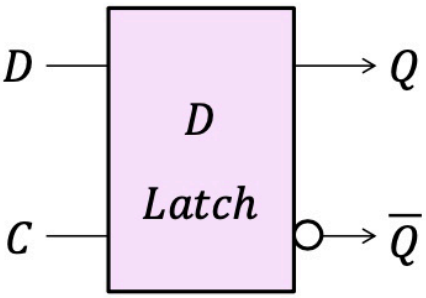
Q(t)	D	Q(t + 1)
0	0	0
0	1	1
1	0	0
1	1	1



$Q(t+1) = D$

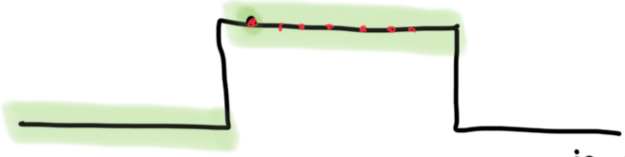






•Problem with Latches:

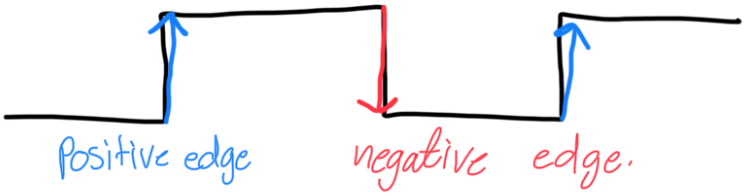
• مشكلة الـ Latches بشكل عام إنه التغيير بيصير على مستوى الـ level فما بكون متزامن (مش دايمًا بكون بنفس اللحظة).



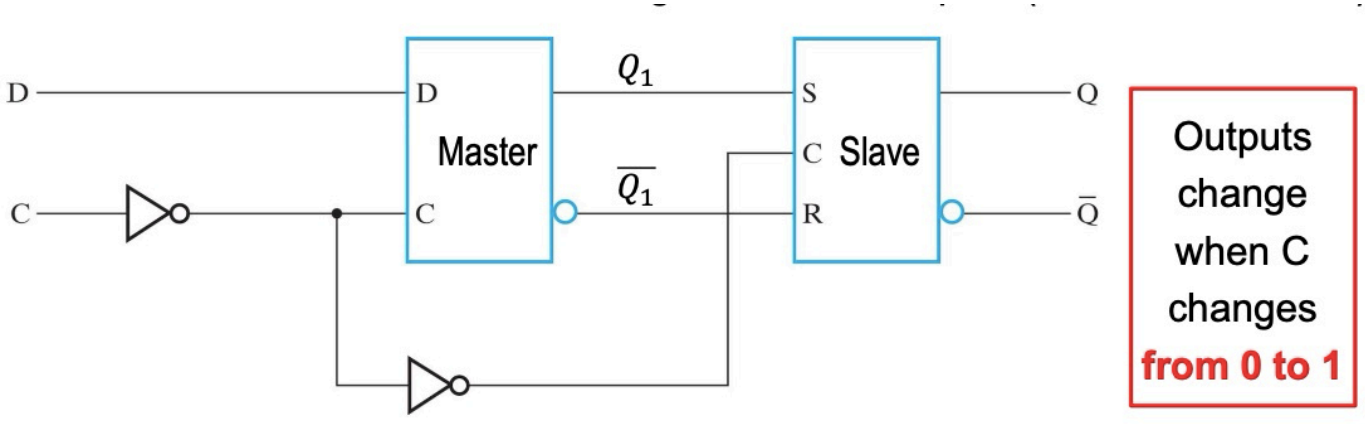
• بحاجة element بكون Synchronous أكتر.

• Flip-Flops:

edge triggered



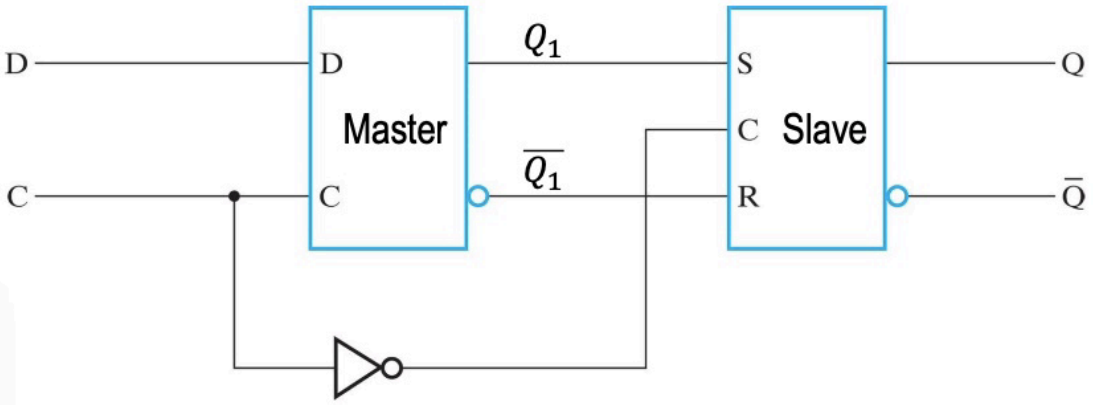
• الـ Flip-Flop بالأساس بتكون من (2latches) حسب الـ clock واحد بشتغل وواحد لا (master-slave).



• Positive Edge -Triggerd D Flip-flop.

- one input (D).
- clock signal.
- D latch (master), SR latch (slave).
- 2 invertors.

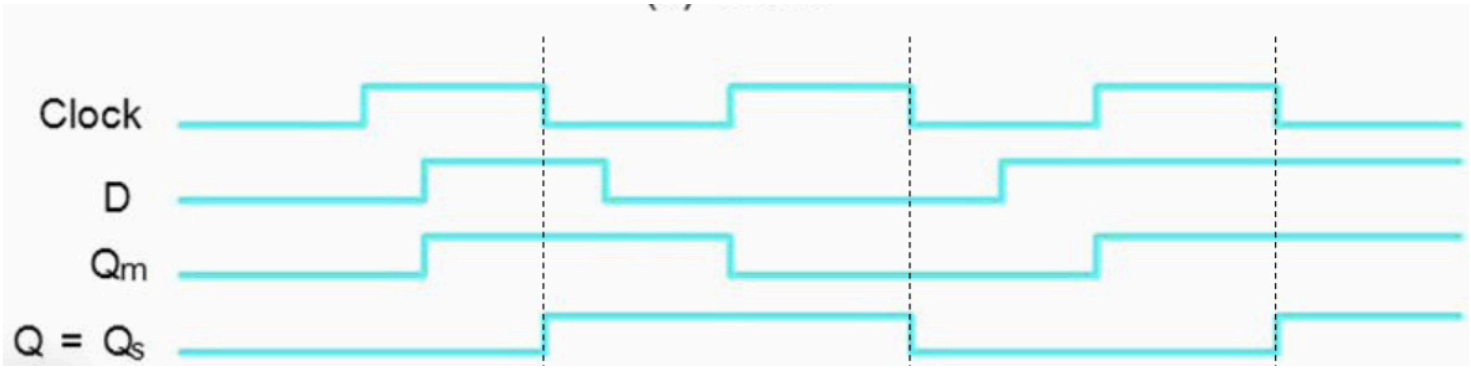
بما إنّه Positive Edge -Triggerd معناته بسمح في التغيير بس عند Rising edge يعني بس في لحظة الانتقال من حبة إلى واحد.



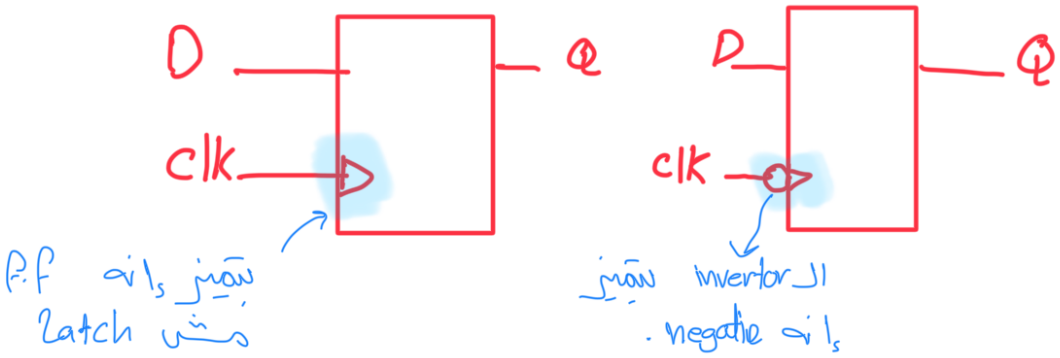
Outputs change when C changes **from 1 to 0**

• negative Edge -Triggerd D Flip-flop.

- نفس الـ positive بس فيه one invertor
- بسمح بالتغيير عند negative edge ، يعني بس لحظة الانتقال من 1 إلى 0.


Clock
D
Qm
Q = Qs

• Graphic Symbols for Flip-flops:

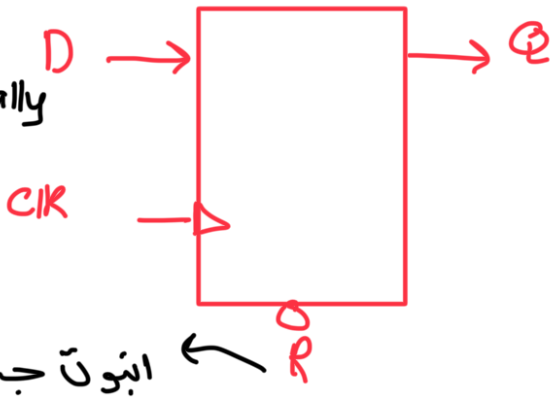


بتميز إنه F.f
Latch مش

بتميز الـ inverter
negative إنه

• D Flip-flop with Asynchronous Reset.

R = 0 → Q = 0
R = 1 → works normally



ابنوت جديد ← R
اذا = 0 ← Q=0
اذا = 1 ← بشتغل D.f.f عادي.

• Jk Flip-flop:

→ 2 inputs (Jk) with clock signal.

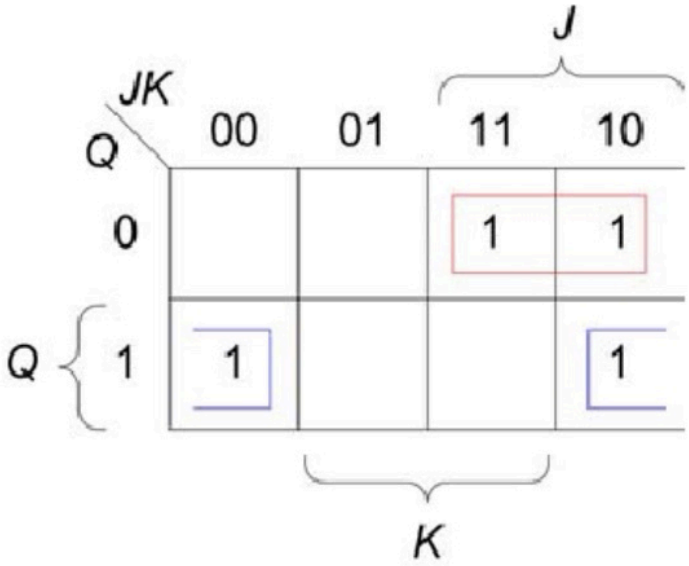
→ 4 possible states:

J	k	Q (t+1)
0	0	no change.
0	1	Reset
1	0	Set
1	1	Q_t' (complement).

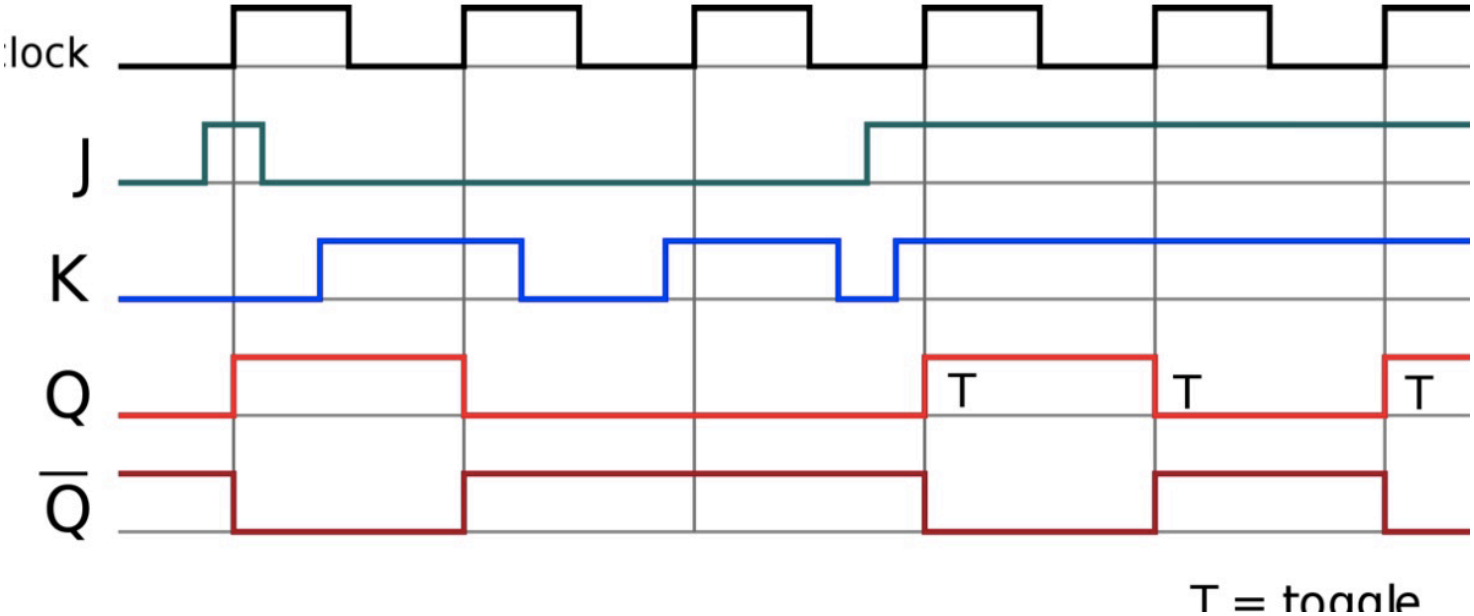
• Jk Flip-flop can be implemented using D.f.f

- Characteristic equation :

Q(t)	J	K	Q(t + 1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



$Q(t+1) = JQ' + K'Q$



T = toggle

• T Flip-flop:

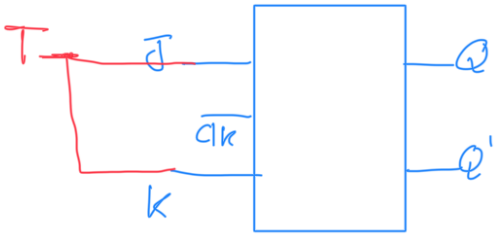
→ one input (T) with clock.

→ 2 possible States:

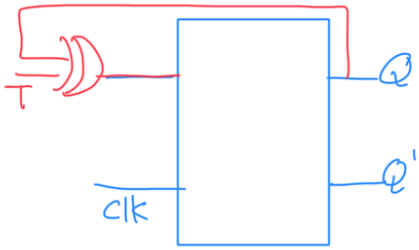
T	$Q(t+1)$
0	no change
1	complement.

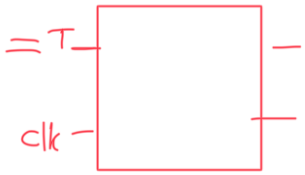
• can be implemented using:

→ JK F.F



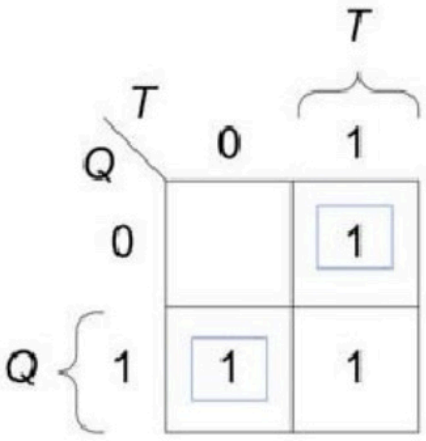
D.F.F



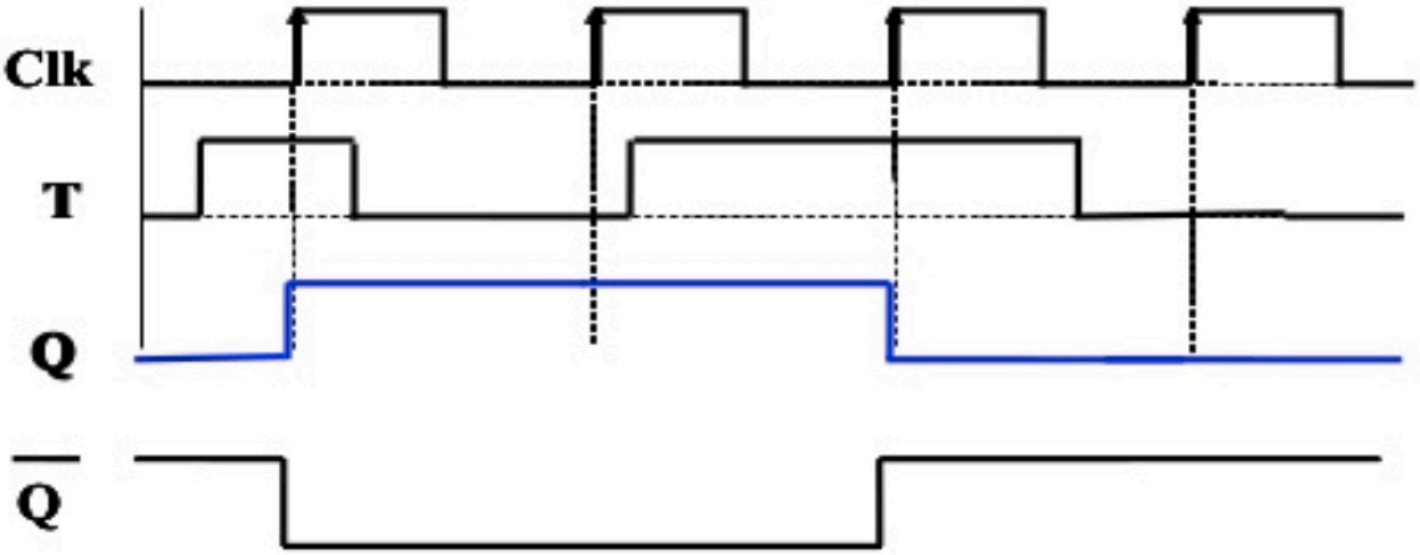


• Characteristic equation:

Q(t)	T	Q(t + 1)
0	0	0
0	1	1
1	0	1
1	1	0



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



• مفهوم جدن جدن :

→ Flip Flops charasteristic equations:

- D Flip Flop: $Q(t+1) = D$
- Jk Flip flop: $Q(t+1) = JQ' + K'Q$.
- T Flip flop: $Q(t+1) = T \oplus Q(t)$

D Flip-Flop			
D	Q(t+1)		
0	0	Reset	
1	1	Set	

JK Flip-Flop			
J	K	Q(t+1)	
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'(t)	Complement

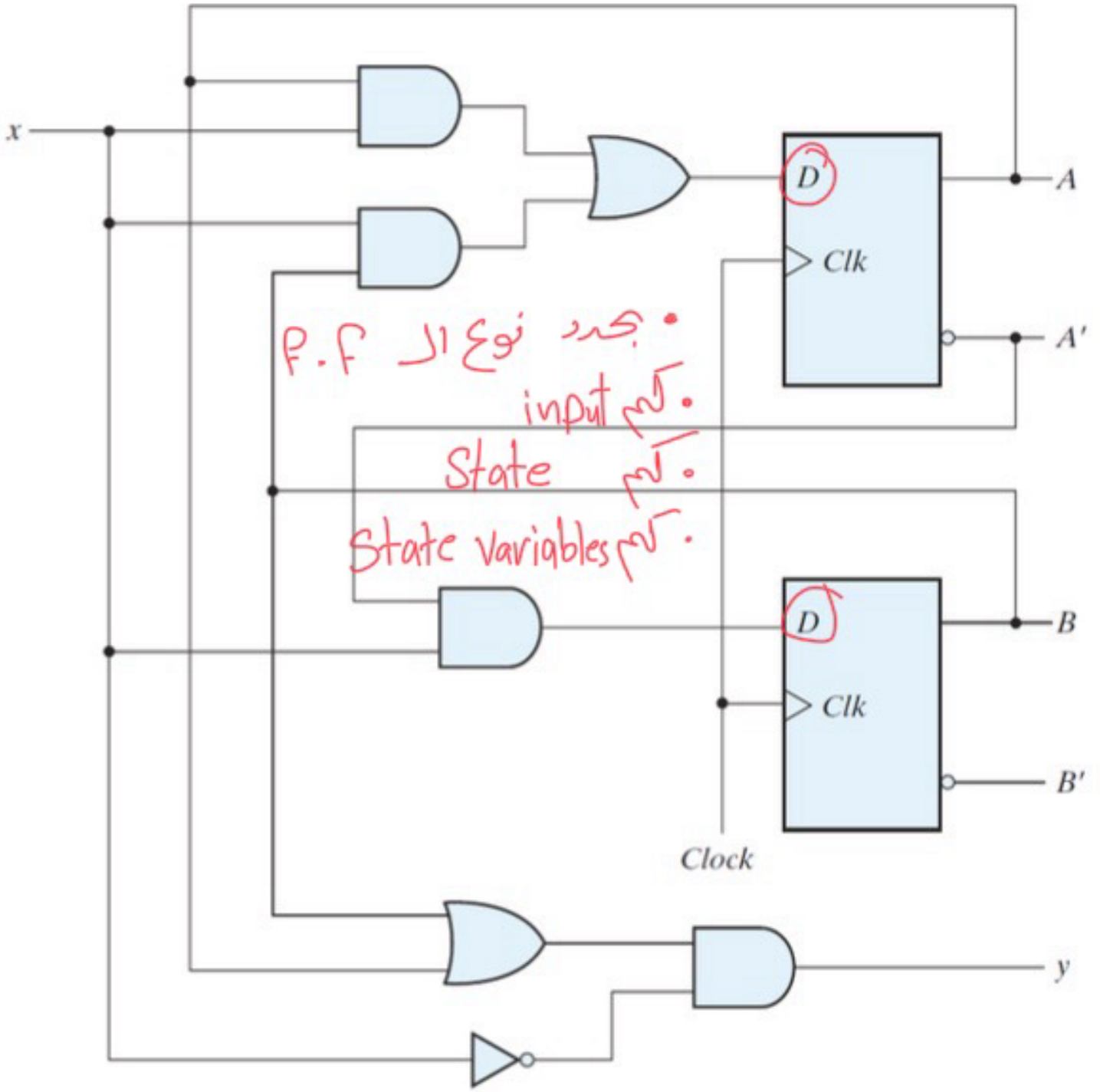
T Flip-Flop		
T	Q(t+1)	
0	Q(t)	No change
1	Q'(t)	Complement

- Analysis of clocked Sequential Circuits:

← الدارة بتكون موجودة ، بدي أوصل للـ State table

• خطوات الـ analysis :
- معادلة الـ inputs ←
- معادلة الـ output ←
- معادلة الـ next State ←
- state table ←
- state diagram. ←

- Analysis Example:



• تحدد نوع ال F.F
• كم input.
• كم State.
• كم State variables.

• مهم جدن جدن:

→ one input (x)
→ 2 (D flip flops).
→ 2 state variables. → State variables = FF نفس عدد
→ 4 possible states. → State = 2^n
→ one output (y)

• باستخدام السيركت والـ Charasteristic eq بدّي أكتب معادلات لل input والـ next state

$0 + 0 = 0$
$(00) + (00)$

• $D_A = Ax + Bx$ (From the circuit)
$D_B = A'x$ $(10) = 0$

$A=0$ $x=0$
$B=0$

• $A(t+1) = D_A$
$B(t+1) = B_B$ (From the charactaristic equation).

• $y = (A+B)x'$ (From the ciraut).
$(0+0)\,1$
$0\,1 = 0$

• State table using the equations:

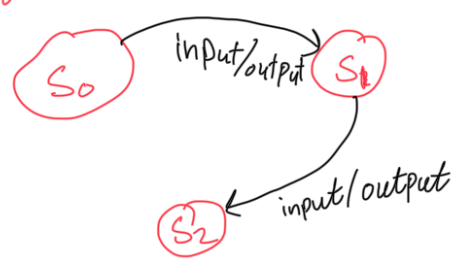
Present State		Input	Next State		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

Another form of the state table

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

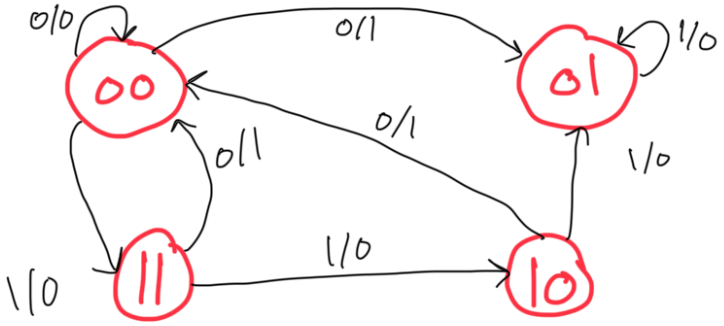
State diagram:

general form:



- كل دائرة بتمثل State.
- سهم من الـ present state لـ next state
- على كل سهم input/output

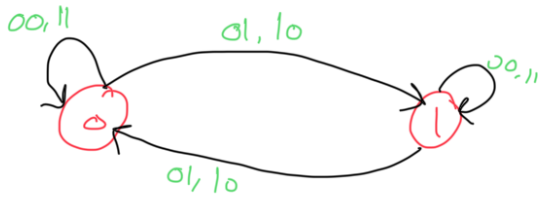
→ State diagram for the example:



- State diagram if there is no output:

← اذا الاوتبوت نفسه الـ state ما بحط الاوتبوت على الـ diagram

present A	inputs x	y	next A
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

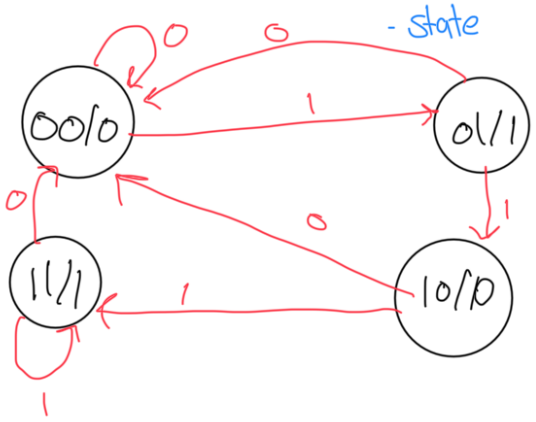


• mealy and moore sequential circuits:

→ **mealy machine:** output depends on present State and inputs.

→ **moore machine:** output depends on present state only. p.s بشكل مباشر بعتمد بس على الـ

• State diagram for moore machine:
لأنه الأوتبوت بعتمد على الـ State بقدر أحط الأوتبوت جوا الدائرة تبعت الـ State.



• State Reduction:

تقليل عدد الـ States عشان أقل عدد الـ f.f أو عشان أبسط الـ combinational circuit

→ Example:

Present	next		output	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

• from the table:
e and g have same output and state, so they are equivalent and I can Replace e by g.

• بدور على equivelent states، بعدها بقدر احذف وحدة منهن و أقلل عدد الـ states.

• بكونوا equivelent اذا الهم نفس الـ output ونفس الـ next State

مهم جدن:
→ Same output and same Q(t+1): eq أكيد
→ different output and Q(t+1): eq أكيد مش
→ Same output but different Q(t+1): لقدرش أحكم لغاية ما أتأكد من الـ state.

Present	next	output

• e,g طلعوا equivalent يعني أخذت وحدة منهم، بعدين برجع أتأكد من باقي الجدول اذا لسا في اشي eq،
* برجع بتأكد من اللي الهم نفس الاوتبوت عشان أقدر أحكم *

	X=0	X=1	X=0	X=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	g	f	0	1
~~e~~	~~g~~	~~f~~	~~0~~	~~1~~
f	g	f	0	1
g	a	f	0	1

d = f
g = e

قللت عدد الـ states من 7 لـ 6

• binary assignment :

بس أوصل للشكل النهائي لازم اسوي binary assignment للـ states، يعني بدل حروف أعطيهم binary code، بختار bits نفس عدد الـ f,f

State:	binary assingment:
a	000
b	001
c	010
f	011
g	100

مهم جدن:
← فش اشي بالزمني شوال code اللي أعطيه لكل state، بس الاسهل أعطيهم حسب binary + الترتيب. المهم استخدم أقل عدد ممكن من الـ Bits.

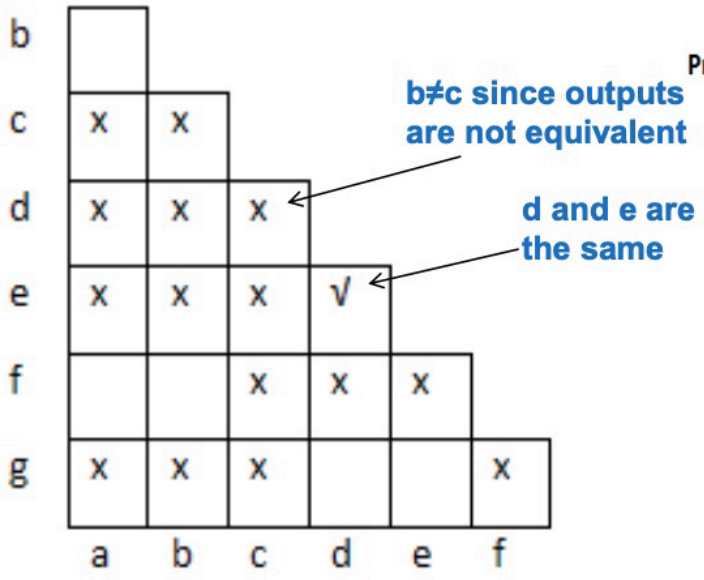
بالنهاية عدد الـ Bits ← هو عدد الـ f,f اللي بدي استخدمه في الـ design

• State Reduction :

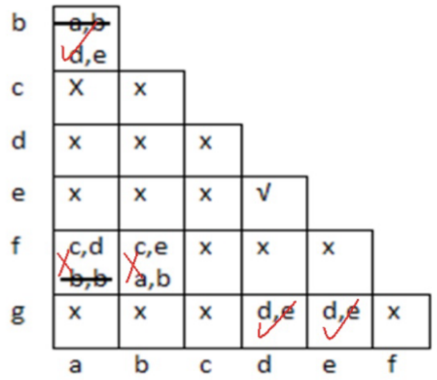
← طريقة ثانية للـ Reduction اي استخدم

Implication chart.

← الـ state اللي متأكد انهم eq بحطلهم (✓)
← اللي متأكد انهم مش eq بحط (x)
← اللي مش متأكد خليهم فاضيين.
← بس أخلص برجع أتأكد من اللي خلوا.



Present State	Next state		Output	
	X=0	X=1	X=0	X=1
a	d	b	0	0
b	e	a	0	0
c	g	f	0	1
d	a	d	1	0
e	a	d	1	0
f	c	b	0	0
g	a	e	1	0



Present State	Next state		Output	
	X=0	X=1	X=0	X=1
a	d	b	0	0
b	e	a	0	0
c	g	f	0	1
d	a	d	1	0
e	a	d	1	0
f	c	b	0	0
g	a	e	1	0

• Sequential Circuit design:

• خطوات ال design :

← من الوصف نعمل State diag ثم State table

← نحدد نوع ال Flip-flop المعدرة ،

← من ال exitation table نعمل معادلات لكل Flip flop input

← circuit

• Flip flop Excitation Tables :

• ال excitation عكس ال characteristic

← يكون عارف ال present state وال next State بس يدي

• أجيب ال flip flop input

Present State	Next State	F.F. Input
$Q(t)$	$Q(t+1)$	D
0	0	0
0	1	1
1	0	0
1	1	1

Present State	Next State	F.F. Input	
$Q(t)$	$Q(t+1)$	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

0 0 (No change)
0 1 (Reset)
1 0 (Set)
1 1 (Toggle)
0 1 (Reset)
1 1 (Toggle)
0 0 (No change)
1 0 (Set)

$Q(t)$	$Q(t+1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

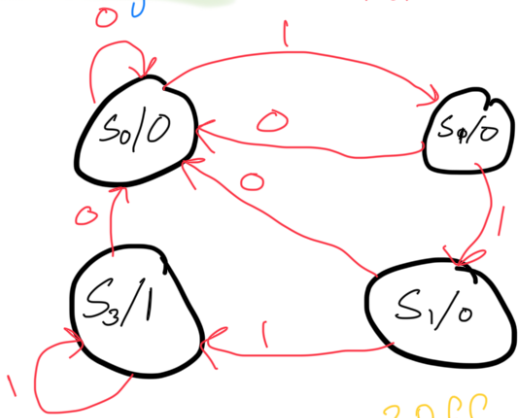
• Design Example:

→ Detect 3 or more consecutive 1's using D Flip Flop.

اذا دخل 3 inputs واحدات ولا بعضن
أو أكتر الاوتبوت بدة يلون 1

• State diagram.

كل مرة بجي 1 بدي أروح عالـ next state
اذا اجا input = 0 بروح لـ State 0



• Binary assignment:

State	A B
S_0	0 0
S_1	0 1
S_2	1 0
S_3	1 1

2Dff يعني ح استخدم ff = Bt لكل

• State Table:

2columns بصيف

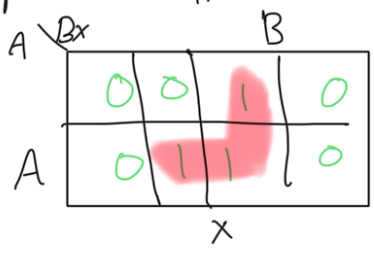
Present A	Present B	input X	next A	next B	output Y	D_A	D_B
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1
0	1	0	0	0	0	0	0
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	0	0	0	0	0
1	1	1	1	1	1	1	1

للانبوتس تبعون الـ f.f ، والقيم من excitation table.

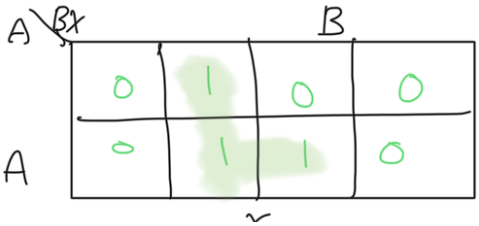
في D.ff القيم بتكون نفس الـ next state

• k maps للي جوفتو

K map for D_A



K map for D_B

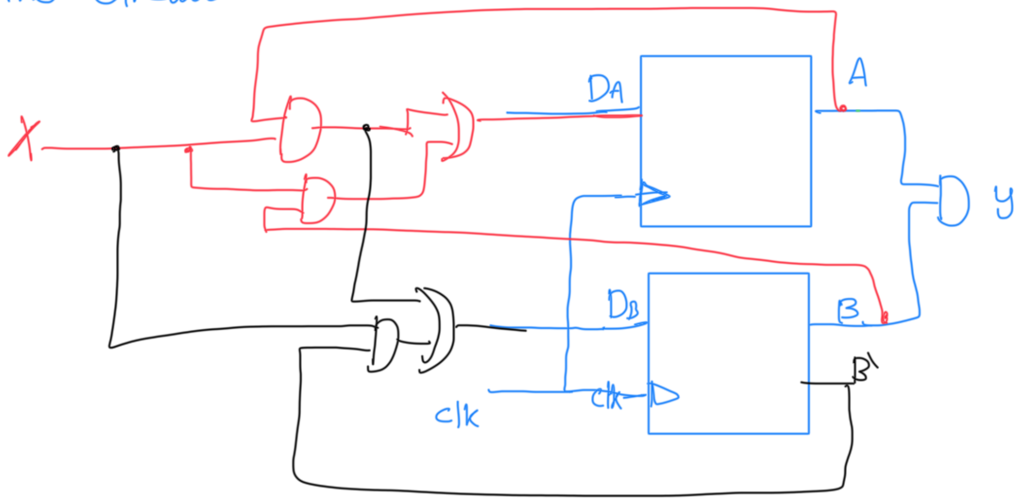


$D_A = Ax + Bx$

$D_B = Ax + B'x$

$y = AB$ (from the table).

• The circuit:



ملاحظة مهمة جدًا :

• كل الأمثلة نفس طريقة الحل ونفس الخطوات بتحتاج الفكرة إني أفهم المطلوب صح و أعمل ال diagram صح وباقي الخطوات بتعتمد عليه.

- State diagram
- binary assignment.
- State table
- equations for f.f inputs, and for the output.
- circuit

• Dealing with Unused States:

← اذا كان عندي states مش مستخدمة في السيركت تبعتي بدي أعرف احدد اذا وصلتلها فشو بدها تكون ال next state

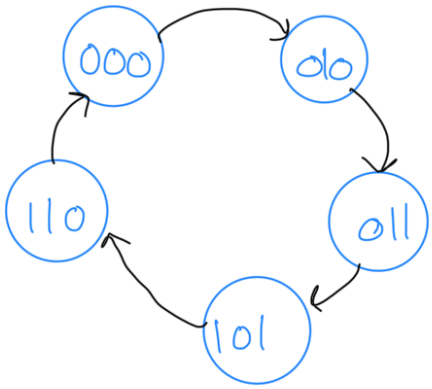
• Example:

design repeats 5 States in Sequence: (using d f.f)

000, 010, 011, 101, 110, 000 (0, 2, 3, 5, 6, 0)

في ال Sequence فش (1, 4, 7) ، ما بحطهم بال diagram
في ال Table بحط ال next State بحط don't care.

• State diagram



• عادة في ال Counters فش inputs ولا outputs
إذا إذا السؤال بده

• State table:

كانه off فش حاجة أو مبين next state

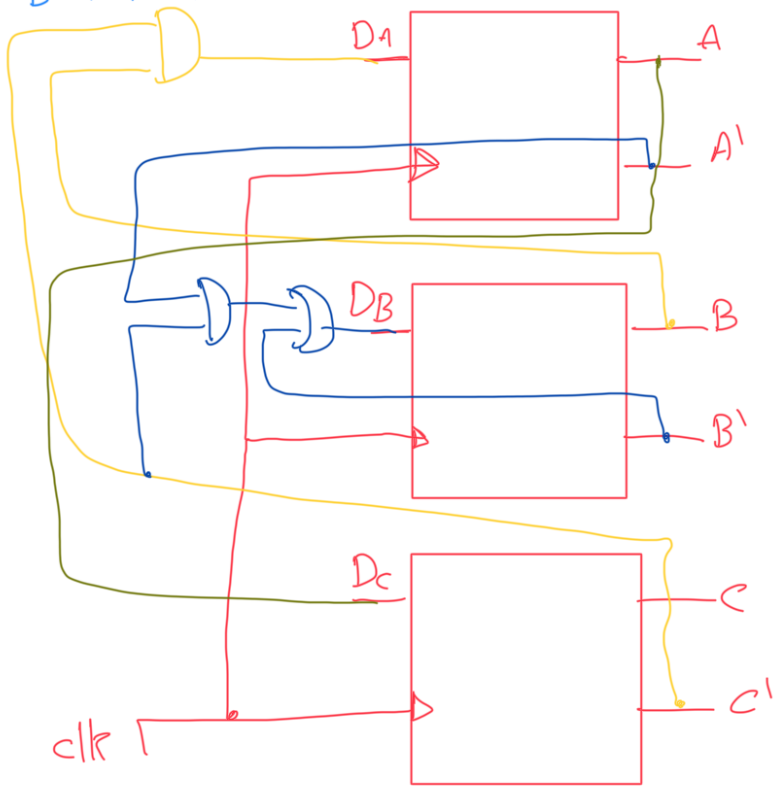
A	B	C	A+	B+	C+
0	0	0	0	1	0
0	0	1	X	X	X
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	X	X	X
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	X	X	X

• next State Functions using K maps:

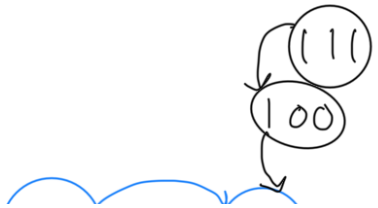
$C+ = A$

$A+ = BC'$

$B+ = B' + A'C'$



• بعد التركيب يرجع بعمل diagram و Table عشان أعرف إذا صار عندي error شو رح تكون ال next state حسب المعادلات



• State table:

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