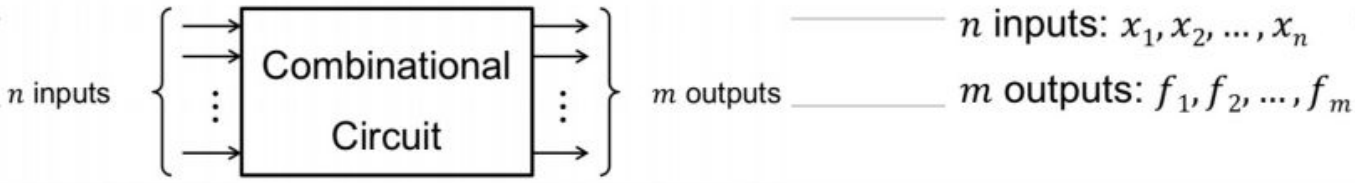
ENCS 2340 Summary Chapter 4

By : Malak Obaid

Combinational circuits

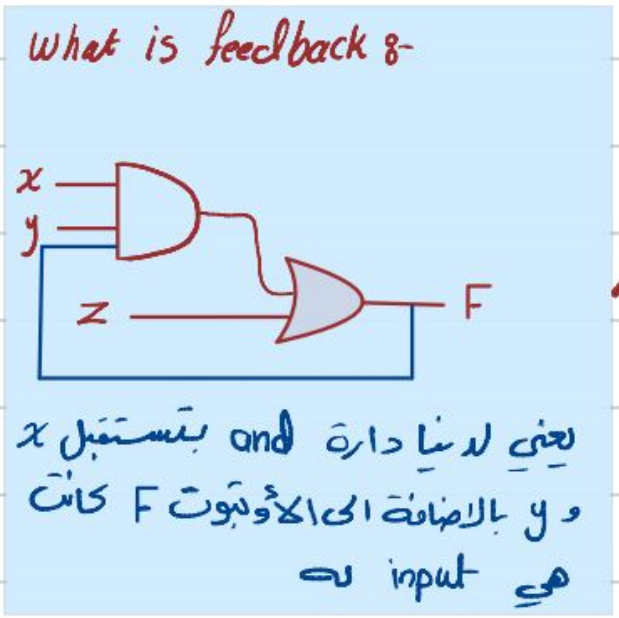
: It's a logic gates with n inputs and m outputs



سيكون لدينا عدة مخرجات ومن ميزاتها انه الاوتبوت عبارة عن فنكشين من الانبوت(input)

الحاليين اي انه اذا تغيرت المدخلات فان الاوتبوت سيكون حسب اخر انبوتس

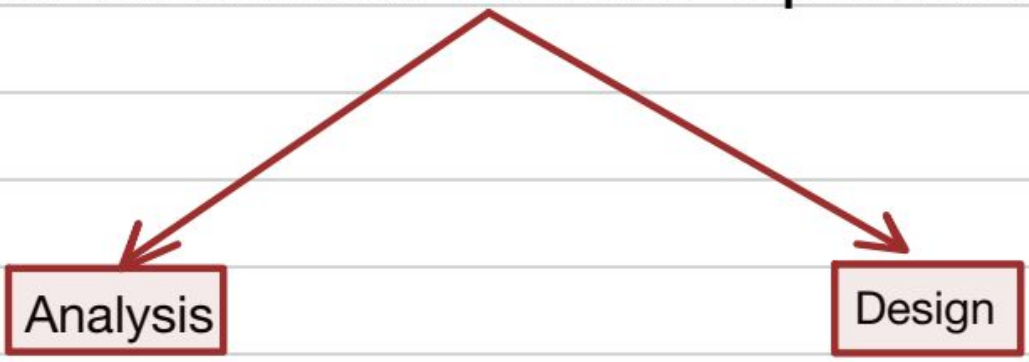
* There's no feedback in combinational circuits

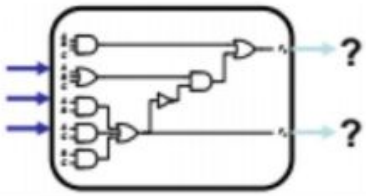


لذا عندما نرى feedback في دارة نعرف انها sequential circuits وليست combinational circuits

* There's no memory in combinational circuits

Combinational Circuits Operation



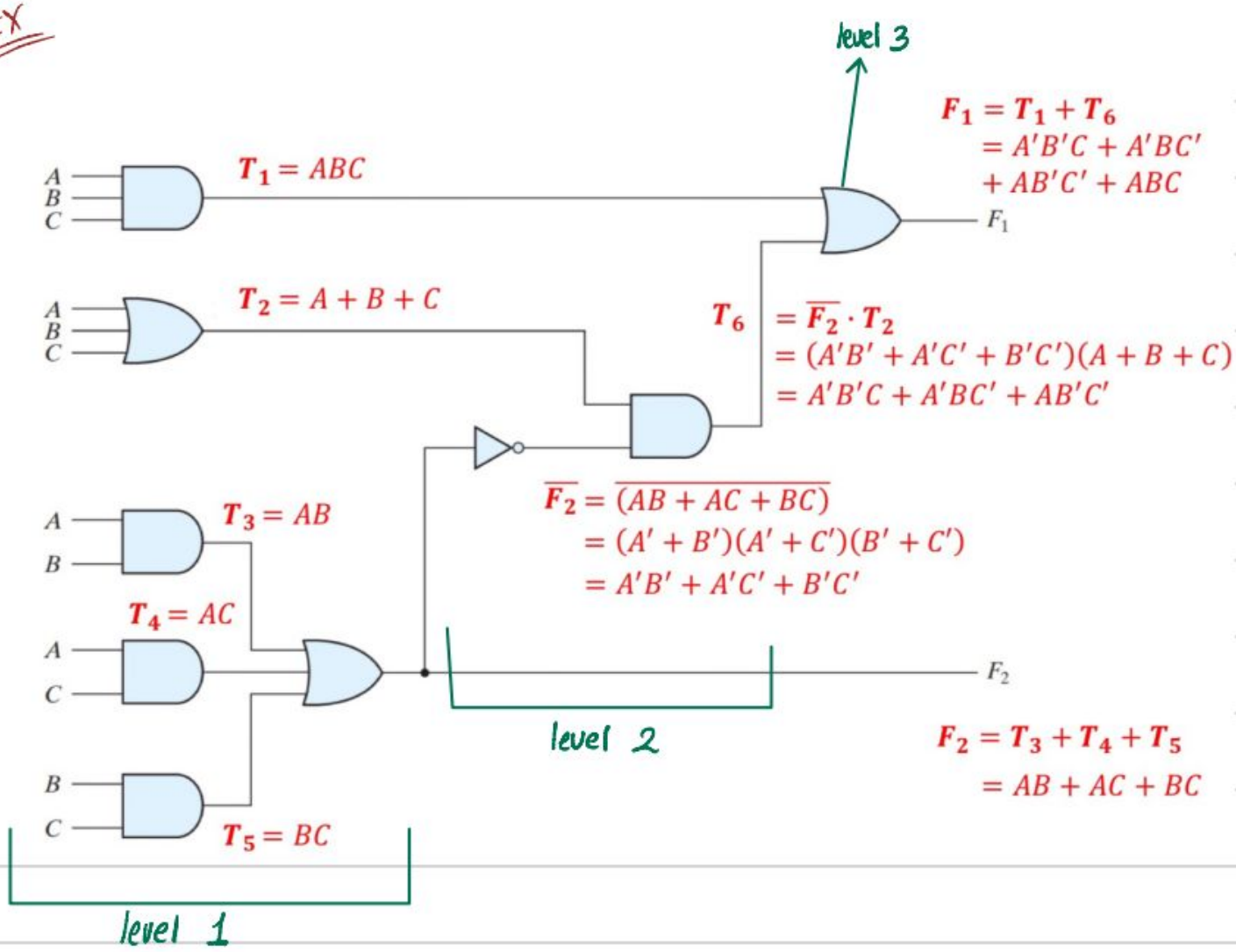


يعطينا دارة ويطلب منا كتابة الاوتبوت لها كما في الصورة

هنا يعطينا الاوتبوت ويطلب منا تصميم دارة بحيث تعطي نفس ال outputs

- Function may be expressed as:
  - Boolean function
  - Truth table

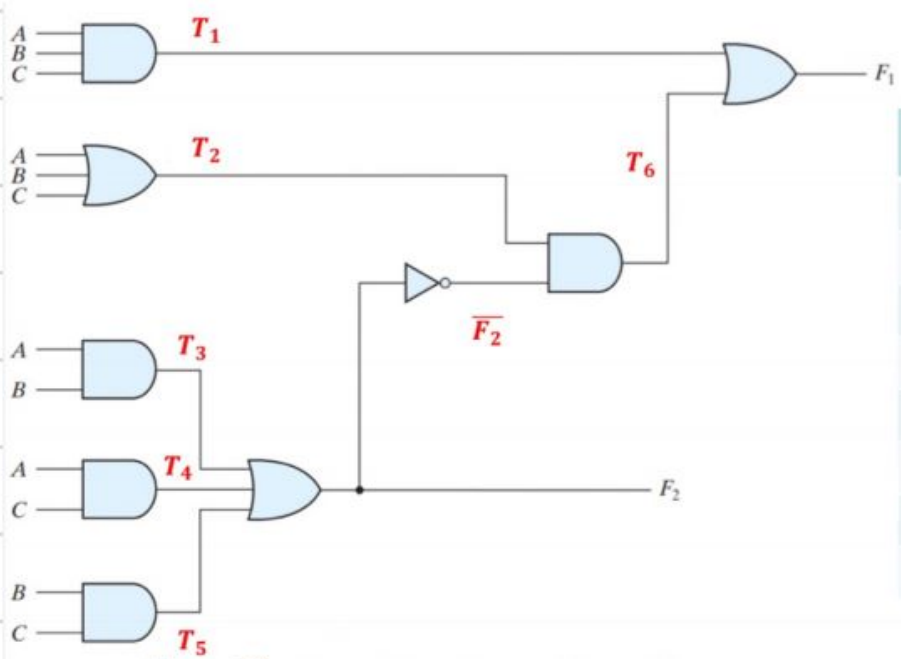
Ex



في هذه الحالة نمشي كل level لحال حتى نصل level 3 وهذه إحدى طرق إيجاد ال Analysis ولدينا طريقة ال truth table لإيجادها.

Truth table method:-

1. نقوم بكتابة عدد ال rows من رموز الدالة من خلال قانون 2^n
2. نقوم بتقسيم وكتابة الخرجات في كل level داخل الجدول وذلك حسب كل دالة كما في المثال الآتي



rows = $2^n = 2^3 = 8$

Truth Table

A	B	C	T_1	T_2	T_3	T_4	T_5	F_2	$\overline{F_2}$	T_6	F_1
0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	0	1	0	0	0	0	1	1	1
0	1	0	0	1	0	0	0	0	1	1	1
0	1	1	0	1	0	0	1	1	0	0	0
1	0	0	0	1	0	0	0	0	1	1	1
1	0	1	0	1	0	1	0	1	0	0	0
1	1	0	0	1	1	0	0	1	0	0	0
1	1	1	1	1	1	1	1	1	0	0	1

3. وحتى نجد ال algebraic expression نستخدم ال k-map للخرجات F_1 و F_2 هكذا

F_2

A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$F_2 = AB + AC + BC$$

F_1

A \ BC	00	01	11	10
0	0	1	0	1
1	1	0	1	0

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

How to Design a Combinational Circuit

1. Specification

يعني من الوصف المعطى بدنا نعرف المطلوب مثل عدد ال inputs وال outputs وشو كل دارة رح رح تعمل بالنسبة للإنبوت والآوتبوت

2. Formulation

يعني نحول اللي عملناه بأول خطوة إلى truth table or logic expressions للأوتبوت

3. Logic Minimization

يعني أقلل عدد الرموز باستخدام ال K-map أو Boolean algebra بالقوانين والخطوات اللي تعلمناها سابقاً

4. Technology Mapping

* نقوم برسم الفنكشن اللي كتبناه من الخطوات السابقة باستخدام دارات AND, OR, inverters
* إذا كان طالب بالسؤال نرسم الفنكشن باستخدام دارات معينة نقوم بعمل المطلوب
* نراعي التكلفة وتقليل عدد الإنبوت والآوتبوت وأقل Delay

5. Verification

في آخر خطوة نتأكد هل اللي عملناه صحيح أم لا إما Using Simulation أو manually

Verification Methods

❖ Manual Logic Analysis

- ✧ Find the logic expressions and truth table of the final circuit
- ✧ Compare the final circuit truth table against the specified truth table
- ✧ Compare the circuit output expressions against the specified expressions
- ✧ Tedious for large designs + Human Errors

❖ Simulation

- ✧ Simulate the final circuit, possibly written in HDL (such as Verilog)
- ✧ Write a test bench that automates the verification process
- ✧ Generate test cases for ALL possible inputs (exhaustive testing)
- ✧ Verify the output correctness for ALL input test cases
- ✧ Exhaustive testing can be very time consuming for many inputs

Example : Designing a BCD to Excess-3 Code Converter

* يجب كتابة الخطوات مع توضيح الكل بكل خطوة كالآتي :

1. Specification:

- ✧ Input: BCD code for decimal digits 0 to 9 (4 bits)
- ✧ Output: Excess-3 code for digits 0 to 9
- ✧ Convert BCD code to Excess-3 code

⟹ هنا كما ذكرنا سابقاً عندما بكتابة ال input & output وشو رح نعمل الدارة

للتذكير :

in BCD code we need 4 bits (numbers from (0-9))
From BCD to Excess-3, we add 3 to BCD num so the numbers in Excess-3 from (3 to 12) and also we need 4 bits, so we represent them using 4 variables.

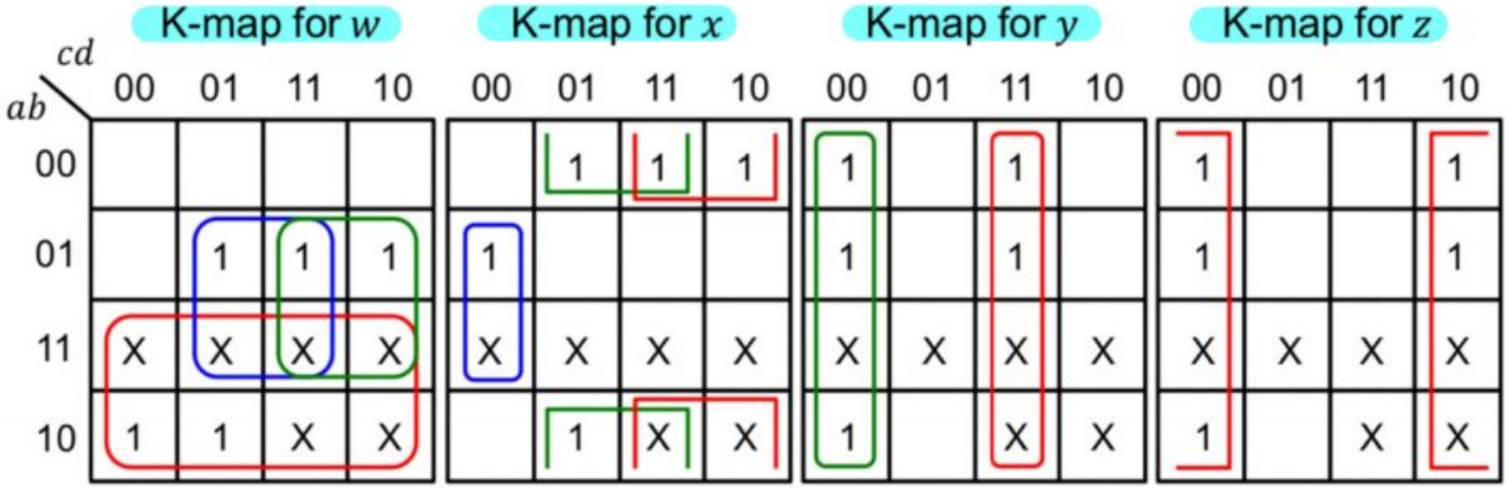
2. Formulation:

- ✧ Done easily with a truth table
- ✧ BCD input: a, b, c, d
- ✧ Excess-3 output: w, x, y, z
- ✧ Output is don't care for 1010 to 1111

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

invalid in BCD so they are dont cares here in Excess-3

3. Logic Minimization using K-maps:



Minimal Sum-of-Products expressions:

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

Additional 3-Level Optimizations: extract common term $(c + d)$

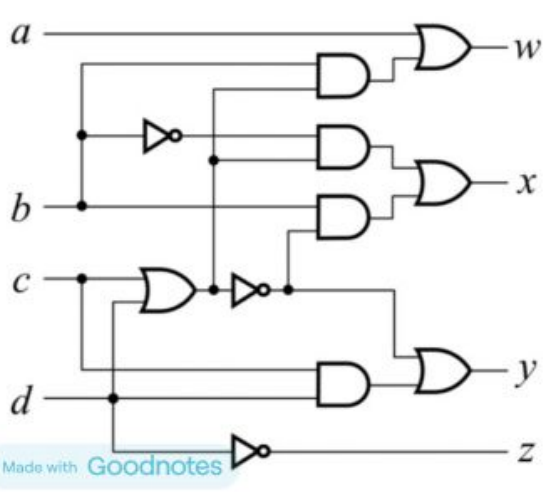
$w = a + b(c + d)$, $x = b'(c + d) + b(c + d)'$, $y = cd + (c + d)'$

* نعمل k-map لكل رمز لوحده حتى نوجد أقل عَمِل بالرموز لهن

4. Technology Mapping:

Draw a logic diagram using ANDs, ORs, and inverters

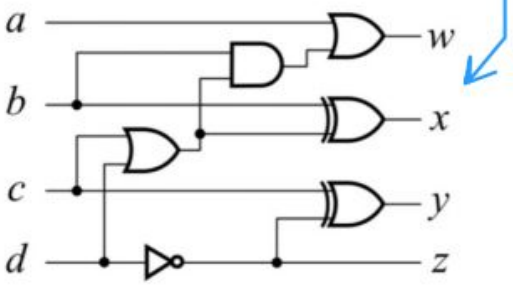
Other gates can be used, such as NAND, NOR, and XOR



Using XOR gates

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

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



Verification methods : manually and simulation .

5. Verification:

Can be done manually

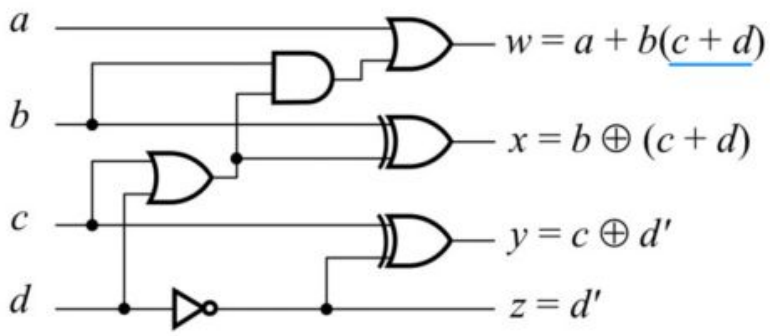
Extract output functions from circuit diagram

Find the truth table of the circuit diagram

Match it against the specification truth table

Verification process can be automated

Using a simulator for complex designs



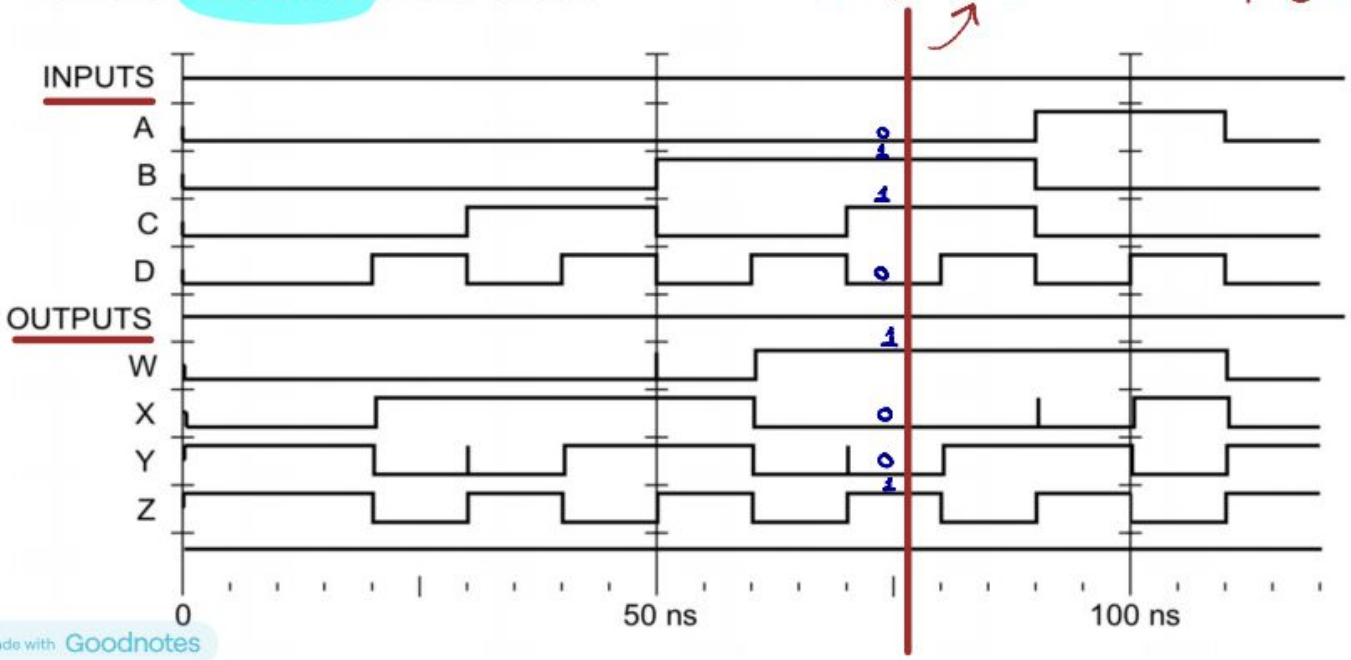
Truth Table of the Circuit Diagram

BCD a b c d	c+d	b(c+d)	Excess-3 w x y z
0 0 0 0	0	0	0 0 1 1
0 0 0 1	1	0	0 1 0 0
0 0 1 0	1	0	0 1 0 1
0 0 1 1	1	0	0 1 1 0
0 1 0 0	0	0	0 1 1 1
0 1 0 1	1	1	1 0 0 0
0 1 1 0	1	1	1 0 0 1
0 1 1 1	1	1	1 0 1 0
1 0 0 0	0	0	1 0 1 1
1 0 0 1	1	0	1 1 0 0

5. Verification:

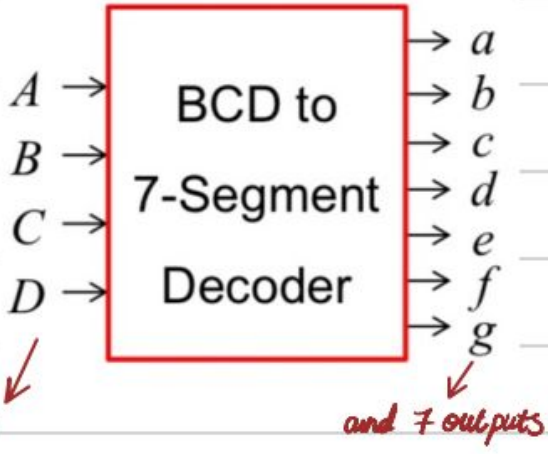
Run the simulation of the circuit

A B C D → W X Y Z (6+3)9 يمثل تكون
0 1 1 0 → 1 0 0 1



7-Segment Decoder

- Made of Seven segments: light-emitting diodes (LED)
- Found in electronic devices: such as clocks, calculators, etc.



Another example : **Designing a BCD to 7-Segment Decoder**

1. Specification:

- Input: 4-bit BCD (A, B, C, D)
- Output: 7-bit (a, b, c, d, e, f, g)
- Display should be OFF for Non-BCD input codes

2. Formulation:

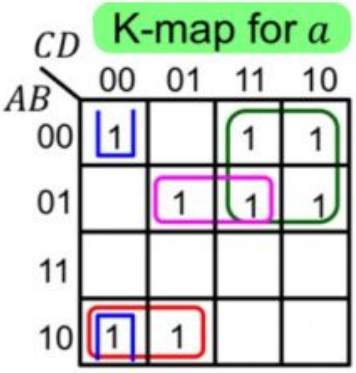
- Done with a truth table
- Output is zero for 1010 to 1111

Truth Table

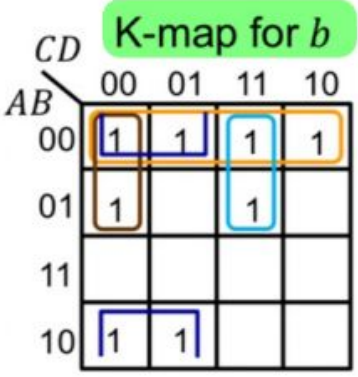
BCD input	7-Segment decoder
A B C D	a b c d e f g
0 0 0 0	1 1 1 1 1 1 0
0 0 0 1	0 1 1 0 0 0 0
0 0 1 0	1 1 0 1 1 0 1
0 0 1 1	1 1 1 1 0 0 1
0 1 0 0	0 1 1 0 0 1 1
0 1 0 1	1 0 1 1 0 1 1
0 1 1 0	1 0 1 1 1 1 1
0 1 1 1	1 1 1 0 0 0 0
1 0 0 0	1 1 1 1 1 1 1
1 0 0 1	1 1 1 1 0 1 1
1010 to 1111	0 0 0 0 0 0 0

3. Logic Minimization Using K-Maps:

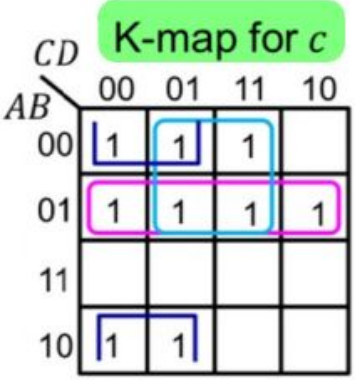
K-map for a



K-map for b



K-map for c



$$a = A'C + A'BD + AB'C' + B'C'D'$$

$$b = A'B' + B'C' + A'C'D' + A'CD$$

$$c = A'B + B'C' + A'D$$

Extracting common terms

Let $T_1 = A'B$, $T_2 = B'C'$, $T_3 = A'D$

Optimized Logic Expressions

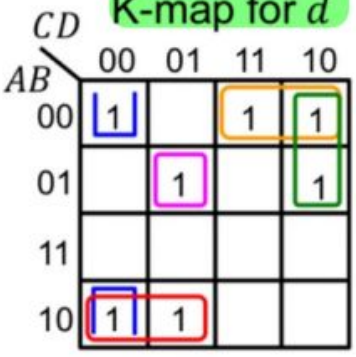
$$a = A'C + T_1 D + T_2 A + T_2 D'$$

$$b = A'B' + T_2 + A'C'D' + T_3 C$$

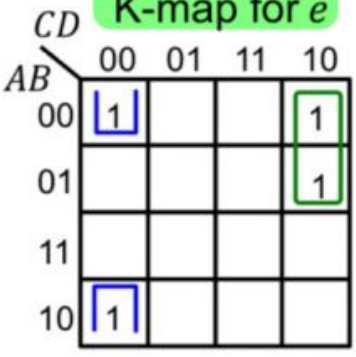
$$c = T_1 + T_2 + T_3$$

T_1, T_2, T_3 are **shared gates**

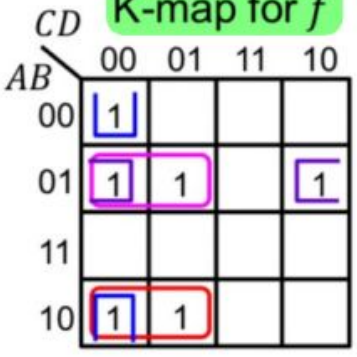
K-map for d



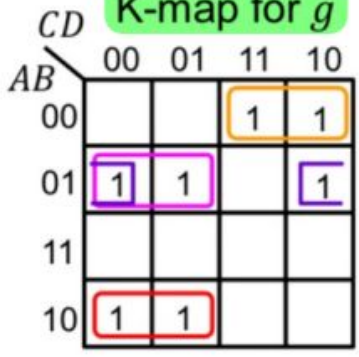
K-map for e



K-map for f



K-map for g



Common AND Terms

➔ Shared Gates

$T_4 = AB'C'$, $T_5 = B'C'D'$

$T_6 = A'B'C$, $T_7 = A'CD'$

$T_8 = A'BC'$, $T_9 = A'BD'$

Optimized Logic Expressions

$$d = T_4 + T_5 + T_6 + T_7 + T_8 D$$

$$e = T_5 + T_7$$

$$f = T_4 + T_5 + T_8 + T_9$$

$$g = T_4 + T_6 + T_8 + T_9$$

4. Technology Mapping:

Many Common AND terms: T_0 thru T_9

$T_0 = A'C$, $T_1 = A'B$, $T_2 = B'C'$

$T_3 = A'D$, $T_4 = AB'C'$, $T_5 = B'C'D'$

$T_6 = A'B'C$, $T_7 = A'CD'$

$T_8 = A'BC'$, $T_9 = A'BD'$

Optimized Logic Expressions

$a = T_0 + T_1\,D + T_4 + T_5$

$b = A'B' + T_2 + A'C'D' + T_3C$

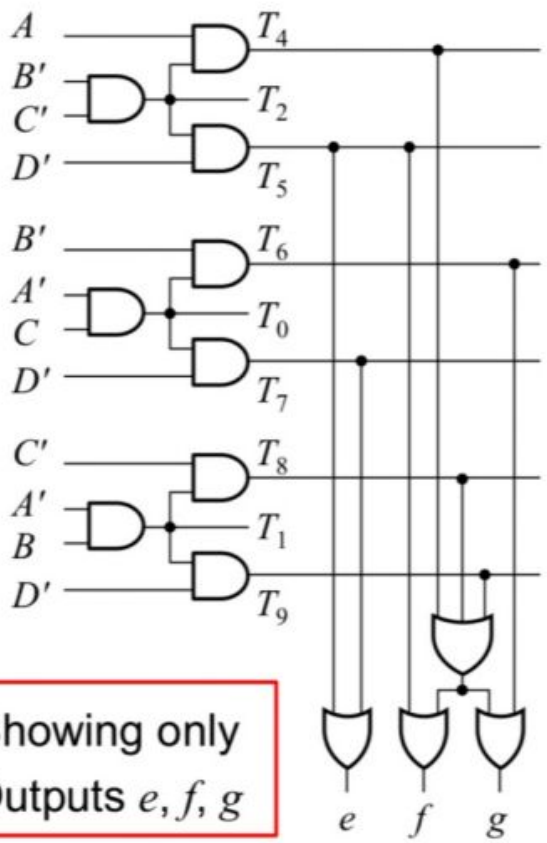
$c = T_1 + T_2 + T_3$

$d = T_4 + T_5 + T_6 + T_7 + T_8\,D$

$e = T_5 + T_7$

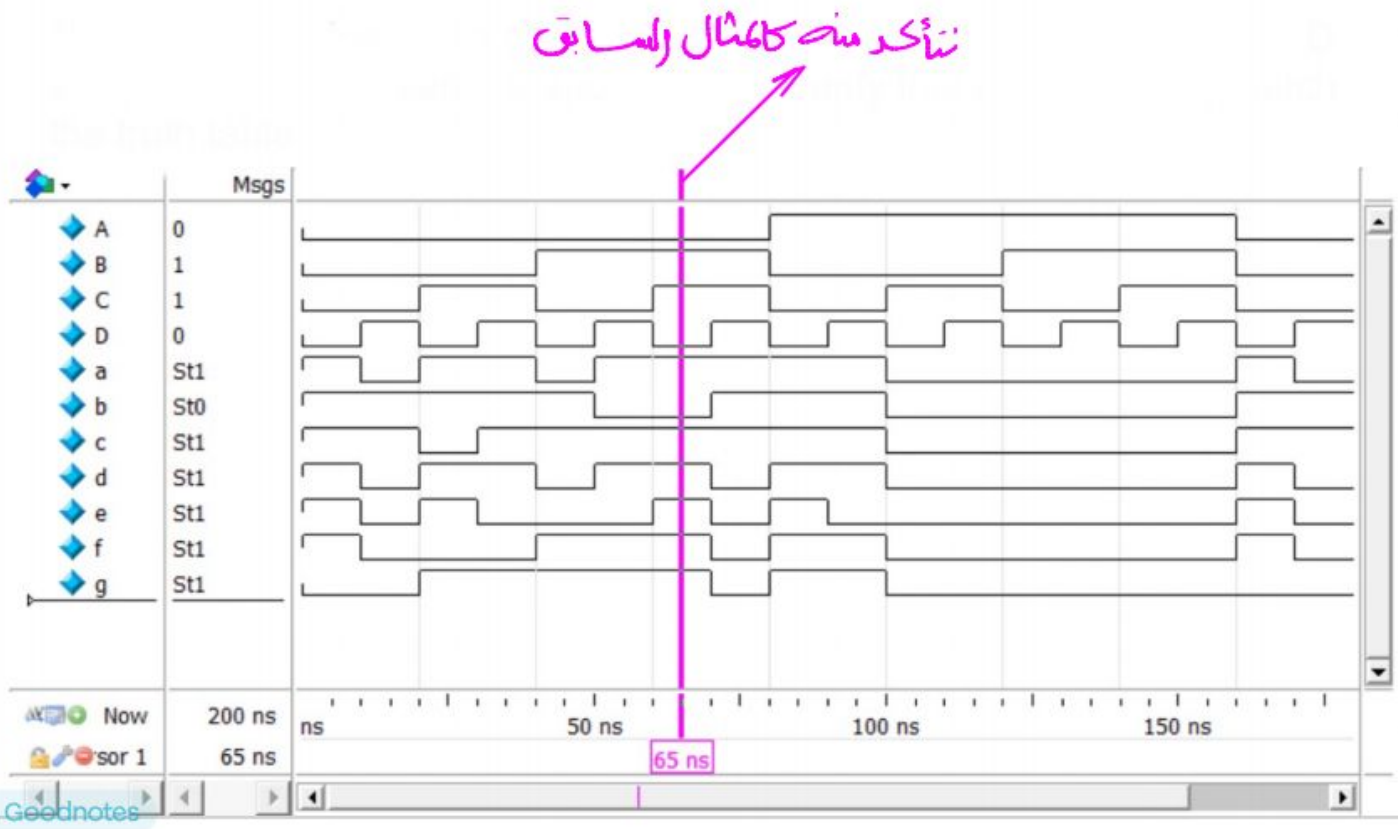
$f = T_4 + T_5 + T_8 + T_9$

$g = T_4 + T_6 + T_8 + T_9$



Showing only Outputs e, f, g

5. Verification:



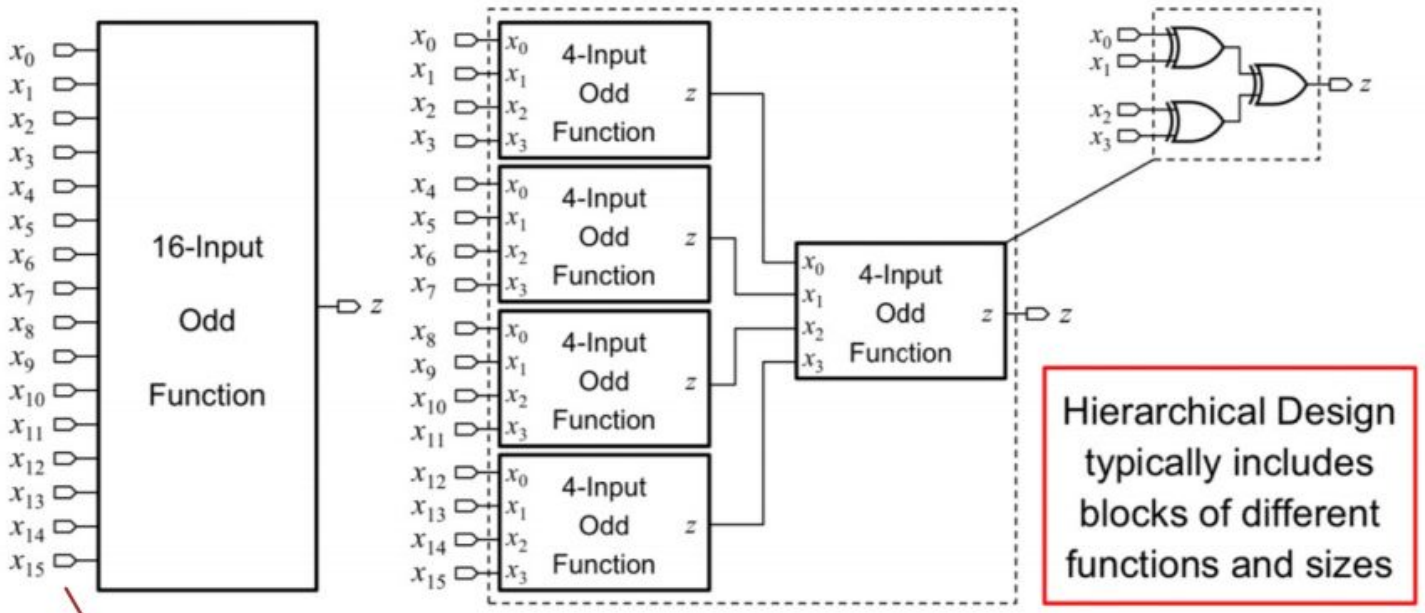
Hierarchical Design

بنعمله لتصميم الدارات المعقدة (complex circuits) حيث نقوم بتقسيم الدارة لأجزاء صغيرة بنسميها (blocks) حتى نبسطها وكل جزء صغير(block) بنرجع نقسمه لاجزاء اصغر والجزء الاصغر اللي بنوصله بنسميه (primitive block)

بعد م وصنا لل primitive block وبسطنا بشكل كبير رح نرجع كانه tree ونبني خطوة خطوة لحتى نرجع نوصل لل complex circuit المطلوبة منا

ومن فوائد هاد الديزاين انه بنقدر نعمل ديزاين لدارات معقدة وكمان بنعمل verification for that primitive block and place them in a library for future use

Example of Hierarchical Design



هان كان عنا 16 إنبوت وقسمناهن لـ 4 مجموعات في كل واحدة 4 إنبوت وعملنا لهن XOR

Testing Hierarchical Design

- Exhaustive testing can be very time consuming (or impossible)
  - For a 16-bit input, there are 2^{16} = 65,536 test cases (combinations)
  - For a 32-bit input, there are 2^{32} = 4,294,967,296 test cases
  - For a 64-bit input, there are 2^{64} = 18,446,744,073,709,551,616 test cases!
- Testing a hierarchical design requires a different strategy
- Test each block in the hierarchy separately
  - For smaller blocks, exhaustive testing can be done
  - It is easier to detect errors in smaller blocks before testing complete circuit
- Test the top-level design by applying selected test inputs
- Make sure that the test inputs exercise all parts of the circuit

Top-Down versus Bottom-Up Design

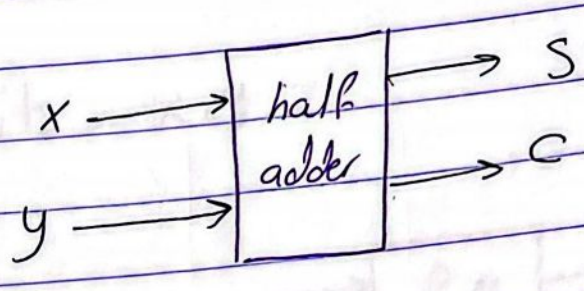
- A **top-down design** proceeds from a high-level specification to a more and more detailed design by decomposition and successive refinement
- A **bottom-up design** starts with detailed primitive blocks and combines them into larger and more complex functional blocks
- Design usually proceeds top-down to a known set of building blocks, ranging from complete processors to primitive logic gates

Ch 4

Half Adder : to add 2 bits, x and y.

- two outputs bits

① Carry bit c
② Sum bit b



$$\begin{array}{r} x \\ +\ y \\ \hline cs \end{array}$$

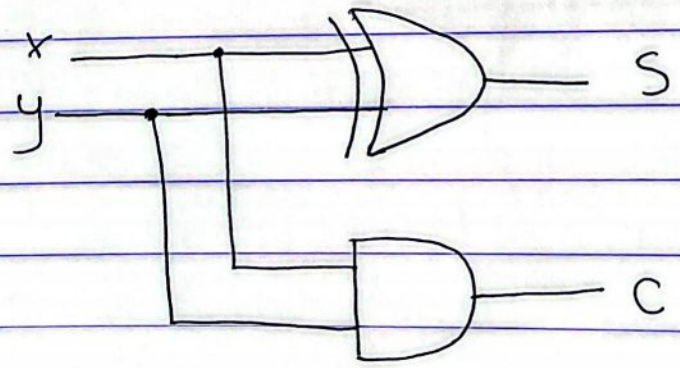
$(10)_2 = 1+1$ ← الجمع يمكن تكون هي $\underset{cs}{(10)_2} = 1+1$

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

ال S = 1 عندما x, y مختلفات
(add func) XOR :

$$S = xy' + x'y$$

ال C=1 عندما فقط x و y = 1
$\therefore C = xy$



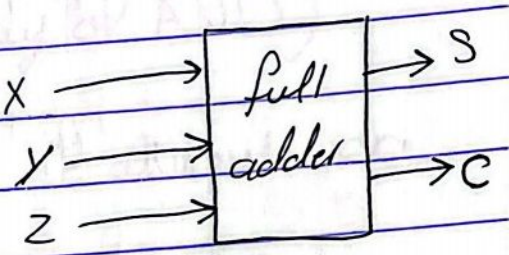
شكل ال half adder
الأساسي.

* For First two bits we need half adder but for more bits we need full adder

Full Adder : Add 3 bits, x, y, z

3 inputs : x, y, z

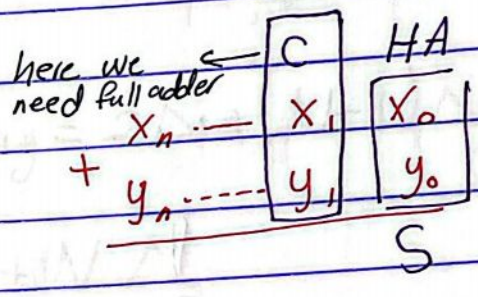
2 outputs :- C (carry), S (sum)



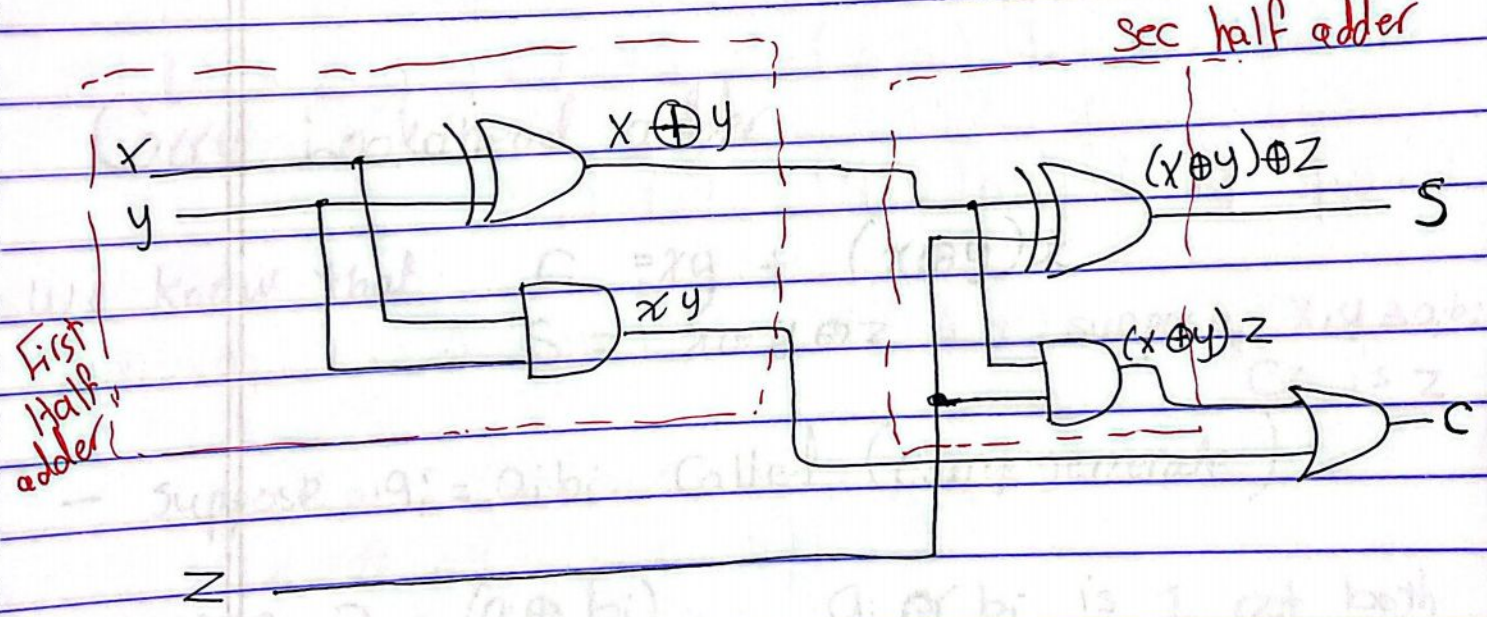
$S = (x \oplus y) \oplus z$

$C = xy + xz + yz = xy + (x \oplus y)z$

here we need full adder



- Full adder Can be implemented with 2 half adders & 1 OR gate



Binary Adder (Ripple Carry Adder)

- Suppose that XOR delay Δ_1 & AND-OR delay is Δ_2 (Delay of XOR > Delay of AND)
- For N-bit ripple-Carry adder if all inputs are present at once

* Most-Significant sum-bit delay $= 2\Delta_1 + (N-1)\Delta_2$
* Final carry-out delay $= \Delta_1 + N\Delta_2$

Carry Lookahead adder

We know that $C = xy + (x \oplus y)z$

$S = x \oplus y \oplus z$ suppose $x, y \Rightarrow a_i, b_i$ C_i is z

- suppose $g_i = a_i b_i$ called (carry generate)
- suppose $P_i = (a_i \oplus b_i)$ a_i or b_i is 1 not both

$\therefore S_i = P_i \oplus C_i$ $\quad C_{i+1} = g_i + P_i C_i$

if a_i and $b_i = 0$ then $g_i = P_i = C_{i+1} = 0$

$$C_{i+1} = g_i + p_i c_i$$

$C_0 \rightarrow$ input carry

$$C_1 = g_0 + p_0 C_0$$

$$C_2 = g_1 + p_1 C_1 = g_1 + p_1 (g_0 + p_0 C_0) = g_1 + p_1 g_0 + p_1 p_0 C_0$$

$$C_3 = g_2 + p_2 C_2 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$$

$$C_4 = g_3 + p_3 C_3 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 C_0$$

Group Generate (GG) $= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$

Group propagate (GP) $= p_3 p_2 p_1 p_0$

$\therefore C_4 = GG + GP C_0$ (Carry doesn't ripple anymore)

- Reduced delay when generating C_1 to C_4 in Parallel

Binary Subtracter

* لما نحسب A−B بناخذ الـ 2's complement لـ B وبنجمع.

$\therefore A - B = A + (2\text{'s complement of } B)$

- Same adder used for both addition & subtraction
- final carry is ignored because:-

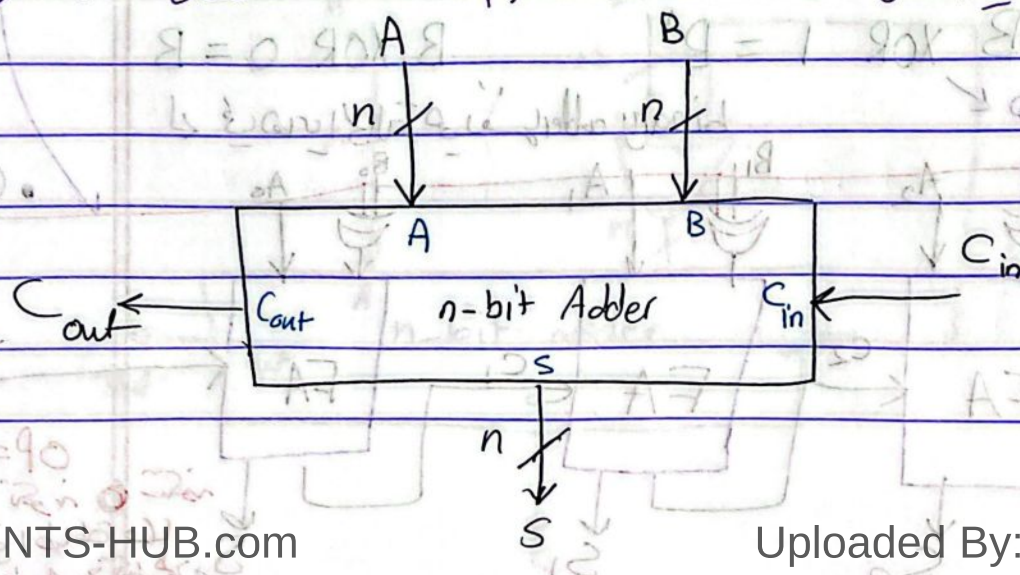
$A + (2\text{'s complement of } B) = A + (2^n - B) = (A - B) + 2^n$

- We have two operations

if operation = 0 then we will calculate (A+B)

but if operation = 1 then we will calculate (A−B)

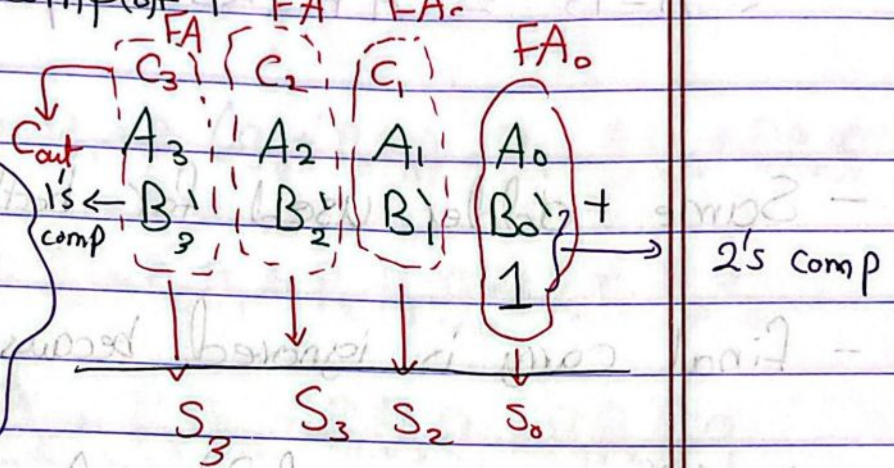
① اذا بدي اعمل الـ Add لازم OP=0 وبكون $C_i=0$



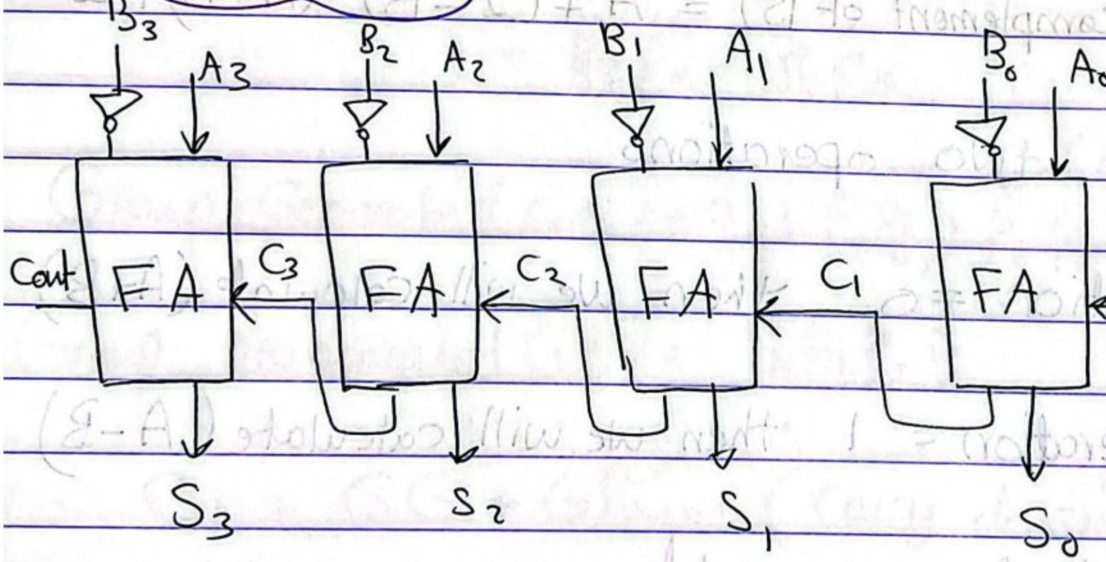
Binary subtractor توضيح لـ

A - B = A + 2's Complement B

= A + 1's comp(B) + 1



B' هي نفس الـ 1's comp
لانو اذا كانت B = 1
فالـ 1's comp = 0 وهكذا



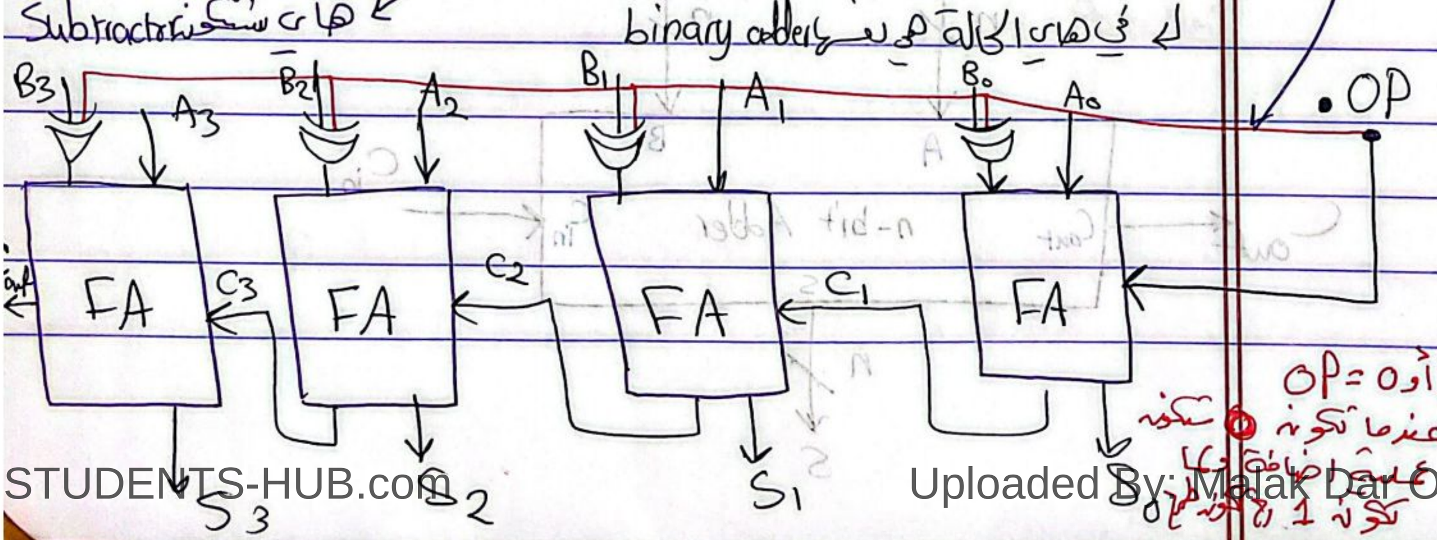
لا نرسمها هكذا! بل مثل رسمة الصفحة التالية

او هكذا

binary | B XOR 1 = B' | B XOR 0 = B

binary subtractor هاي بستخدم ↙

لـ في هاي الحالة هي نفس binary adder

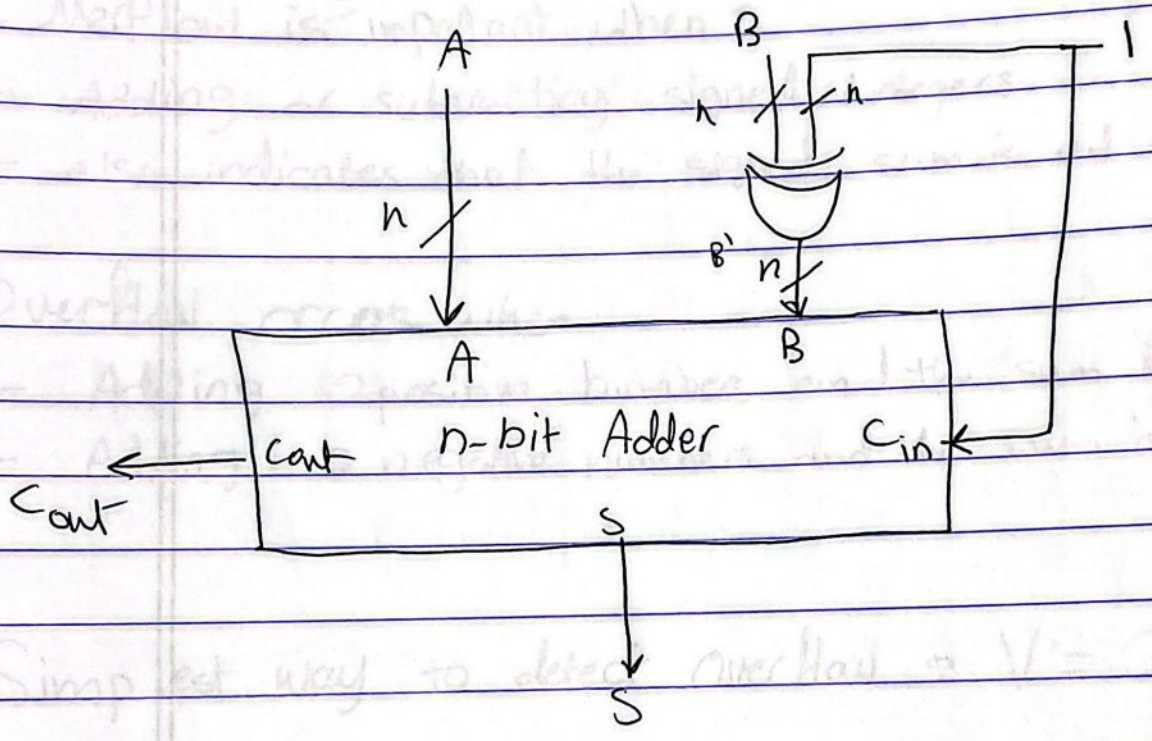


OP = 0 أو 1
عندما تكون 0 ستكون عملية جمع
تكون 1 ...

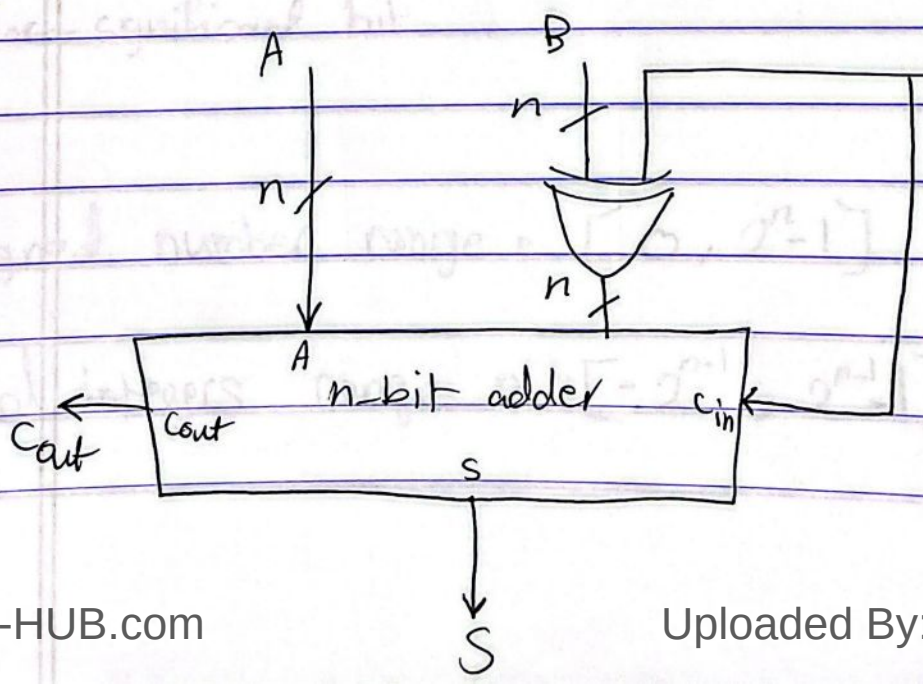
② اذا بدنا نعمل Subtract لازم $C_i = 1$ ، $OP = 1$

* $A - B = A + (1's\ complement\ (B)) + 1$ ← C_{in} بنظلوا منا

حتى نحول B إلى 1's comp بناخدها XOR مع 1



اذا 1 عالـ XOR بين B و Zero راح تظل B فنفقد نختصر الدرجة بأن نكتب OP بدل الـ 1 ونضع صاكة للعمليتين



Carry is important when :-
- Adding unsigned integers
- indecates that the unsigned sum is out of range
- Sum > maximum unsigned n-bit value

Overflow is important when :-
- Adding or subtracting signed integers
- also indicates that the signed sum is out of range

Overflow occurs when :-
- Adding 2 positive numbers and the sum is negative
- Adding 2 negative numbers and the sum is positive

Simplest way to detect overflow : $V = C_{n-1} \oplus C_n$

C_{n-1} and C_n are the carry-in and carry-out of the most-significant bit

Unsigned number range : $[0, 2^n - 1]$

Signed integers range : $[-2^{n-1}, 2^{n-1} - 1]$

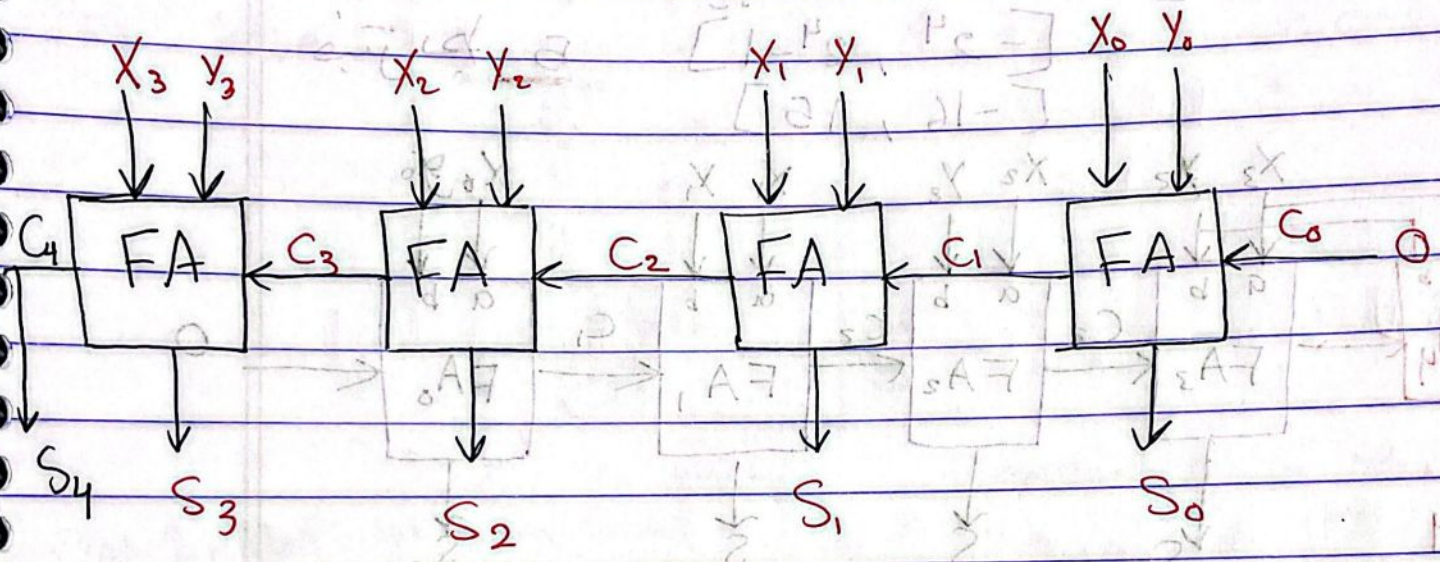
Ex Design a circuit that computes $S = X + Y$ (unsigned x, y)

X and Y are 4-bit unsigned int → range = 0 to 15

min range (S) 0+0 = 0

max range (S) 15 + 15 = 30 → Unsigned S must be 5-bits

we need 4 full adder



⇒ Most-Significant Sum bit S_4 is the Carry bit C_4

Ex 2 Design a circuit that computes

$S = X + Y$ (Signed x, y)

$X[3:0]$ and $Y[3:0]$ are 4-bit signed (range = −8 to 7)

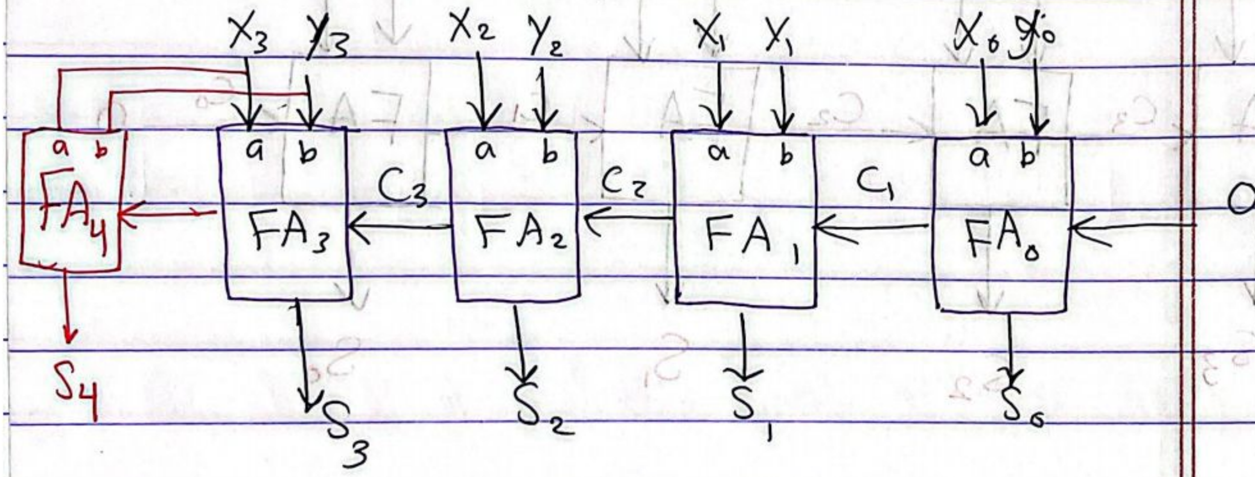
min = $-8 + -8 = -16$ for signed number

max = $7 + 7 = 14$ range $[-2^{n-1}, 2^{n-1}-1]$

بتجرب 4 و 5 حتى نرتبط $[-16, 14]$ عشان نركب عهادا الربيع

$[-2^4, 2^4-1]$ فبنرتبط 5

$[-16, 15]$



$X_3\ X_3\ X_2\ X_1\ X_0$ في 2's comp الكاري نحمله

$+\ Y_3\ Y_3\ Y_2\ Y_1\ Y_0$ عشان 5bits

$S_4\ S_3\ S_2\ S_1\ S_0$

بيتاباه

عشان نصيح

نعمل extint

Decimal Adder & BCD Adder

- Operands (inputs) and result ; 0 to 9

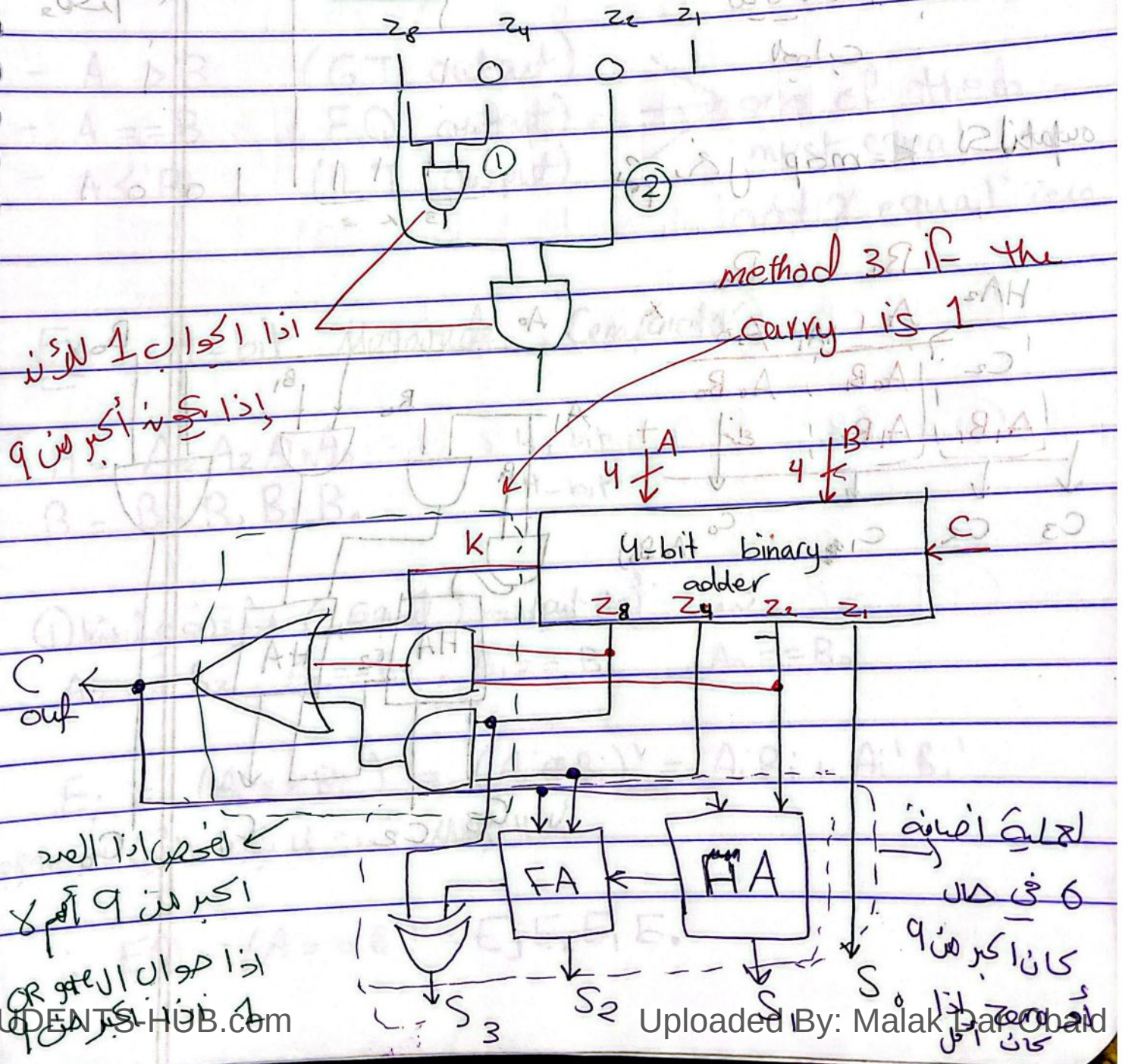
min = 0+0 = 0

max = 9+9+1 = 19

↙ input carry from previous digit

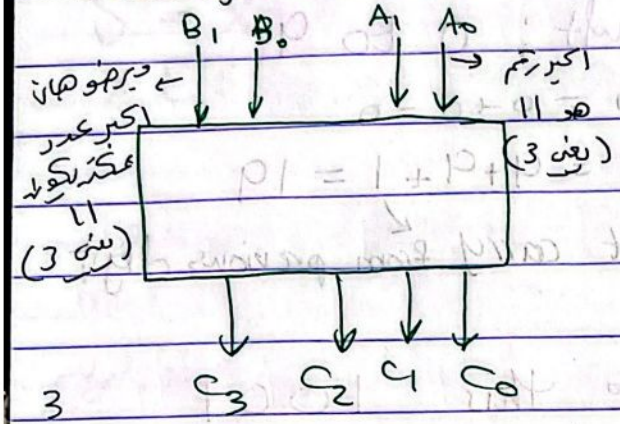
Here we need 5-bits ←

if we have any num like this 1001 we have 3 ways to check if it's >9 or not



Binary Multiplier :-

- design 2 bit x 2 bit binary Multiplier



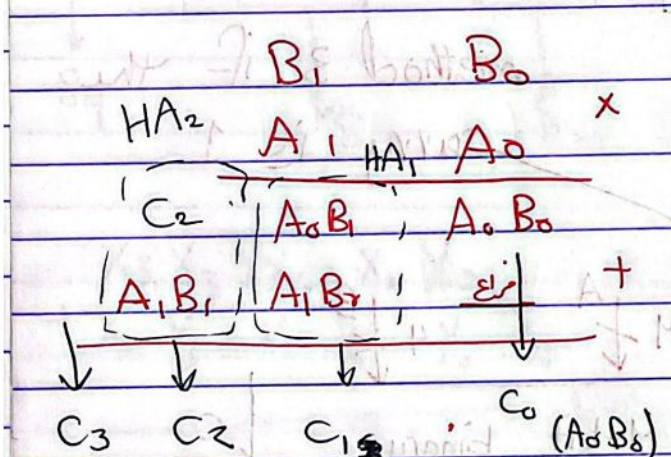
A_1 A_0	B_1 B_0	C_3 C_2 C_1 C_0
0 0	0 0	0 0 0 0
0 0	0 1	0 0 0 0
0 0	1 0	0 0 0 0
⋮	⋮	⋮
1 1	1 1	1 0 0 1

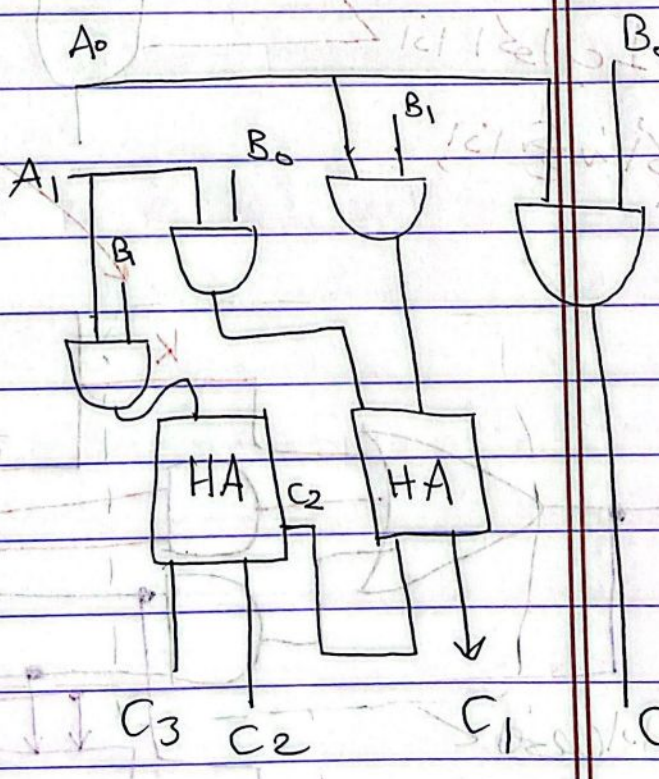
3 x 3

3 × 3 = 9 → يعني بدنا 4 بت للجواب

وهكذا ...

ثم نعمل k-map لكل output

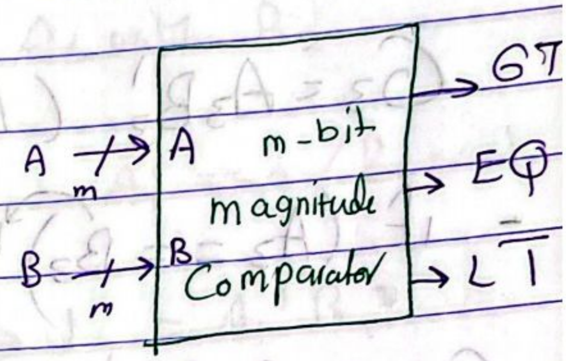




Magnitude Comparator

- Combinational circuit compares 2 unsigned integers

- two inputs
  - Unsigned int A (m = bit)
  - " " B (m - bit)



- Three Outputs

- A > B (GT output)
- A == B (EQ output)
- A < B (LT output)

→ one of them must equal 1 and 2 equal zero

Ex 4-bit Magnitude Comparator

$A = A_3 A_2 A_1 A_0$ 4-bit
$B = B_3 B_2 B_1 B_0$ 4-bit

① نتفقد أول (Equal) output أشي أنه يكون عندنا
$A_3 == B_3$, $A_2 == B_2$, $A_1 == B_1$, $A_0 == B_0$

$E_i = (A_i == B_i) = (A_i \oplus B_i)' = A_i B_i + A_i' B_i'$

↙ ال XNOR بتطلع لما يكونوا الشيئين نفس بعض

$\therefore EQ = (A == B) = E_3 E_2 E_1 E_0$

$A_3\ A_2\ A_1\ A_0$

$B_3\ B_2\ B_1\ B_0$

* GT : greter than ($A > B$)

if $A_3 > B_3$ then $G_T = 1$ irrespective of lower bit of A and B

$G_3 = A_3 B_3'$ ($A_3 == 1$ & $B_3 == 0$)

- if $(A_3 == B_3)$ $\xrightarrow{E_3}$ we compare (A_2 & B_2)

$G_2 = A_2 B_2'$ ($A_2 == 1$ & $B_2 == 0$)

- if $(A_3 == B_3)^{E_3}$ & $(A_2 == B_2)^{E_2}$ then we compare A_1 with B_1

$G_1 = A_1 B_1'$ ($A_1 == 1$ & $B_1 == 0$)

if- $(A_3 = B_3)^{E_3}$ $(A_2 == B_2)^{E_2}$ & $(A_1 == B_1)^{E_1}$ → we compare A_0 & B_0

$G_0 = A_0 B_0'$ ($A_0 == 1$ & $B_0 == 0$)

$\therefore GT = G_3 + E_3 G_2 + E_3 E_2 G_1 + E_3 E_2 E_1 G_0$

* Less than (A < B)

$A_3 A_2 A_1 A_0$
$B_3 B_2 B_1 B_0$

① if $A_3 < B_3$ the LT = 1 نقصد بالمقارنة بين البتات

$L_3 = A_3' B_3$

② if $A_3 == B_3$ (E_3) we compare A_2 with B_2

$L_2 = A_2' B_2$

$A_i == 0 \;/\; B_i == 1$

③ $A_3 == B_3$ & $A_2 == B_2 \longrightarrow L_1 = A_1' B_1$

$(A_3 == B_3)$ & $(A_2 == B_2)$ & $(A_1 == B_1) \longrightarrow L_0 = A_0' B_0$

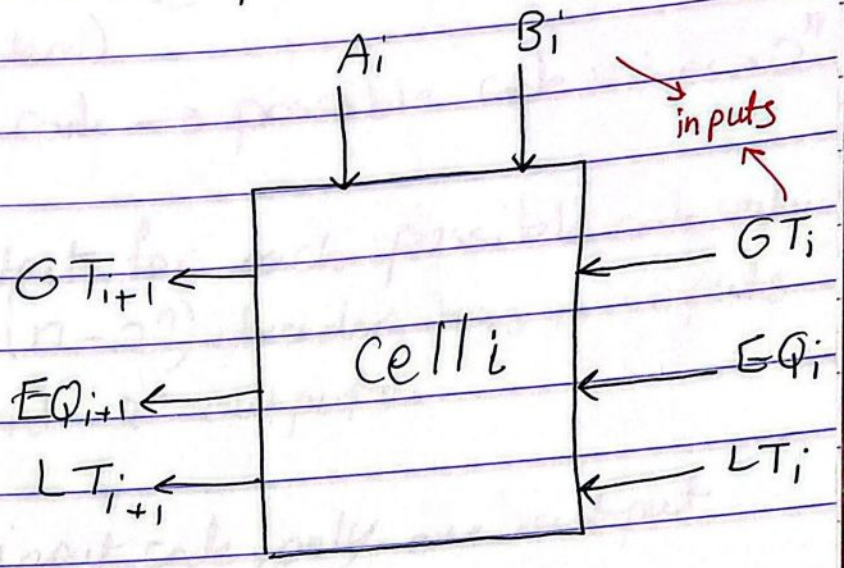
$\therefore LT = L_3 + E_3 L_2 + E_3 E_2 L_1 + E_3 E_2 E_1 L_0$

نوجد
Knowing GT & EQ, we can also derive

$LT = (GT + EQ)'$
$= GT' \cdot EQ'$

Iterative Magnitude Comparator Design

- Output Expression



$$EQ_{i+1} = E_i \cdot EQ_i$$

$$GT_{i+1} = A_i B_i' + E_i \cdot GT_i$$

$$LT_{i+1} = A_i' B_i + E_i \cdot LT_i$$

$$E_i = A_i B_i' + A_i B_i$$

$(A_i = B_i)$ (XNOR)

NOTE

Third output can be produced for first two

$$LT_{i+1} = (EQ_{i+1} + GT_{i+1})' = EQ_{i+1}' \cdot GT_{i+1}'$$

Decoders :-

جهاز فك التشفير

(Binary Decoders)

* Given n-bit binary code → possible code values = 2^n

* decoder has an output for each possible code value which means :- the $(n-2^n)$ decoder has n inputs and 2^n outputs.

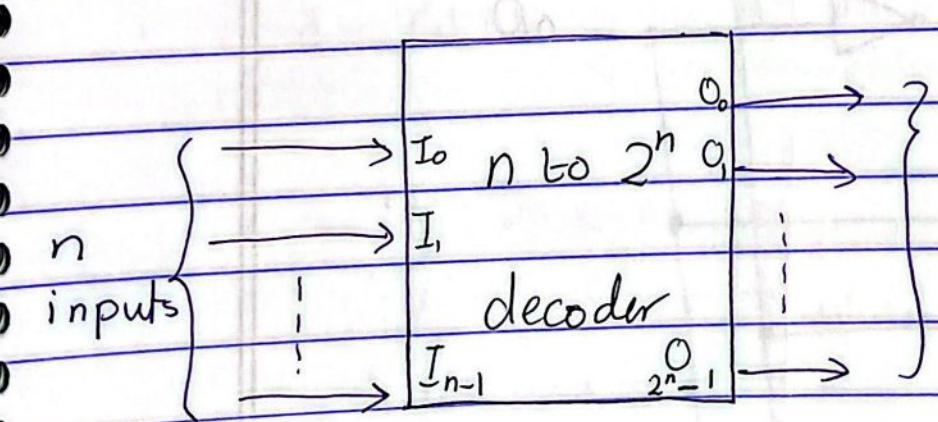
* depending on the input code, only one output is set to logic 1

* the conversion of input to output called decoding

* Unsigned num ليه اول binary بتكون

↓

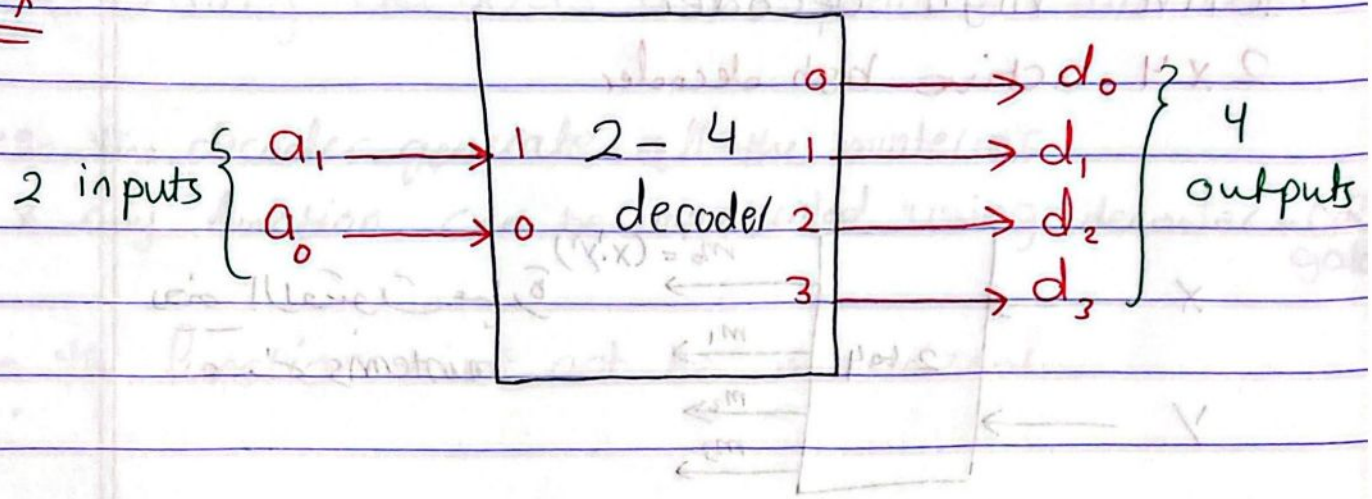
range $[0, 2^n-1]$



NOTE:

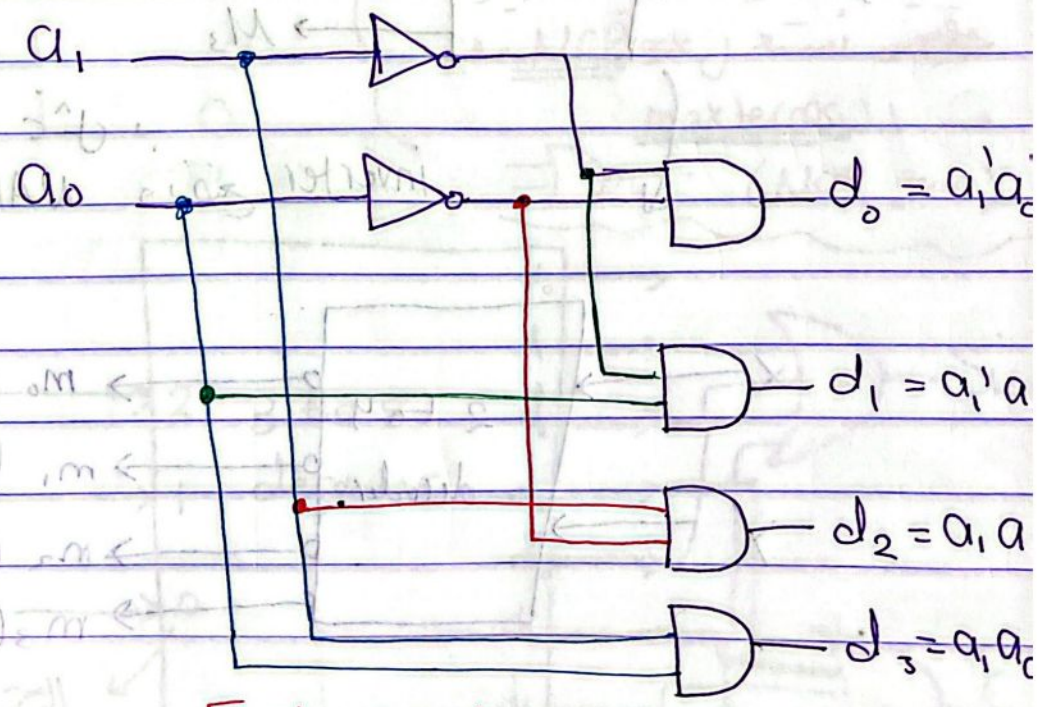
A decoder can have less than 2^n outputs if some input codes are unused

Ex



inputs		outputs				
a_1	a_0	d_0	d_1	d_2	d_3	
0	0	1	0	0	0	$\rightarrow a_1' a_0'$ (d_0)
0	1	0	1	0	0	$\rightarrow a_1' a_0$ (d_1)
1	0	0	0	1	0	$\rightarrow a_1 a_0'$ (d_2)
1	1	0	0	0	1	$\rightarrow a_1 a_0$ (d_3)

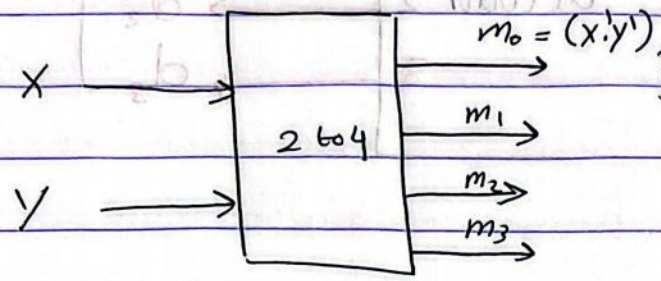
قيمتها في
الديسمال = 3
اذا نكتب اعند d_3
وهكذا



Each decoder output is a minterm

Active high decoder

2 x 4 active high decoder

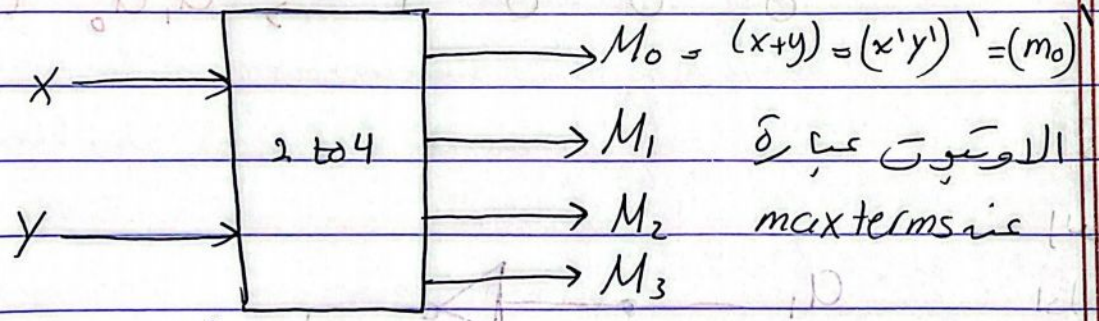


يعني الاوتبوت عبارة عن minterms

* هذا عثلها بـ AND gates

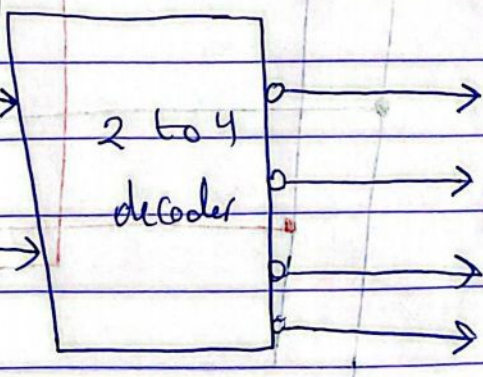
Active low decoder

2 x 4 active low decoder



الاوتبوت عبارة عن maxterms

* لكنه هذا عثل بـ NAND gates



Ex: Using decoders to Implement functions

note:- the decoder generate all the minterms
* any function can be implemented using decoder + OR gate

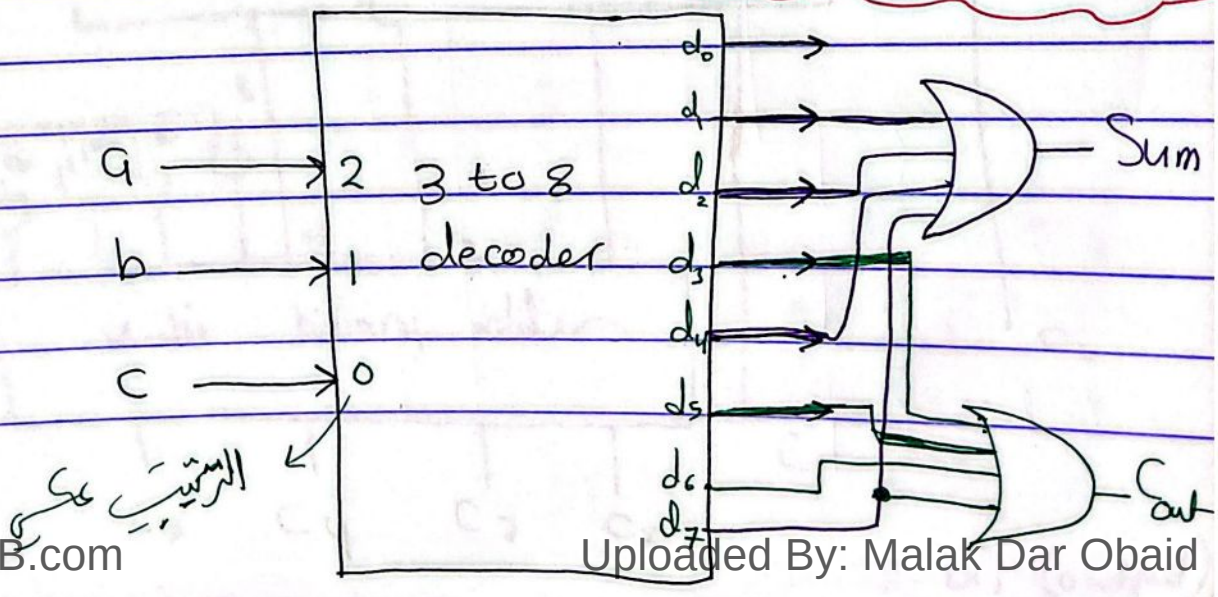
- the function must not be minimized

Ex Full Adder $sum = \Sigma(1,2,4,7)$ $C_{out} = \Sigma(3,5,6,7)$

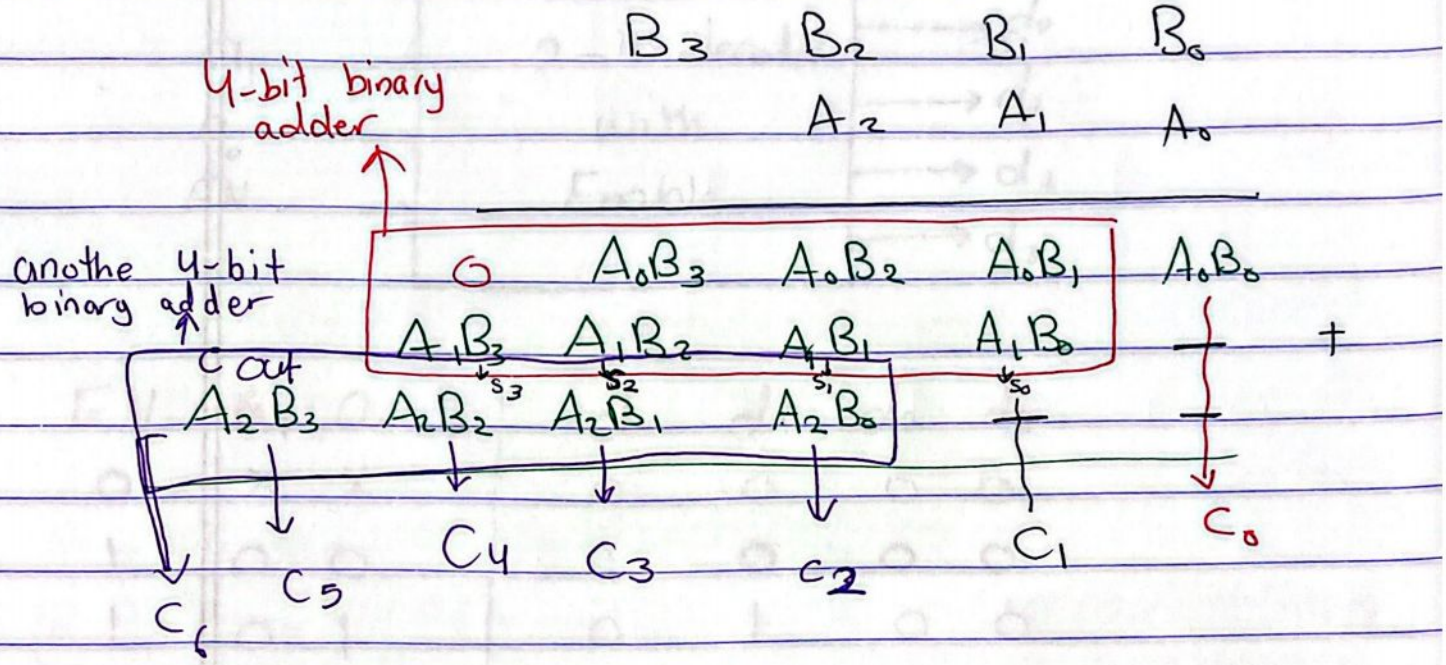
inputs			outputs	
a	b	c	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

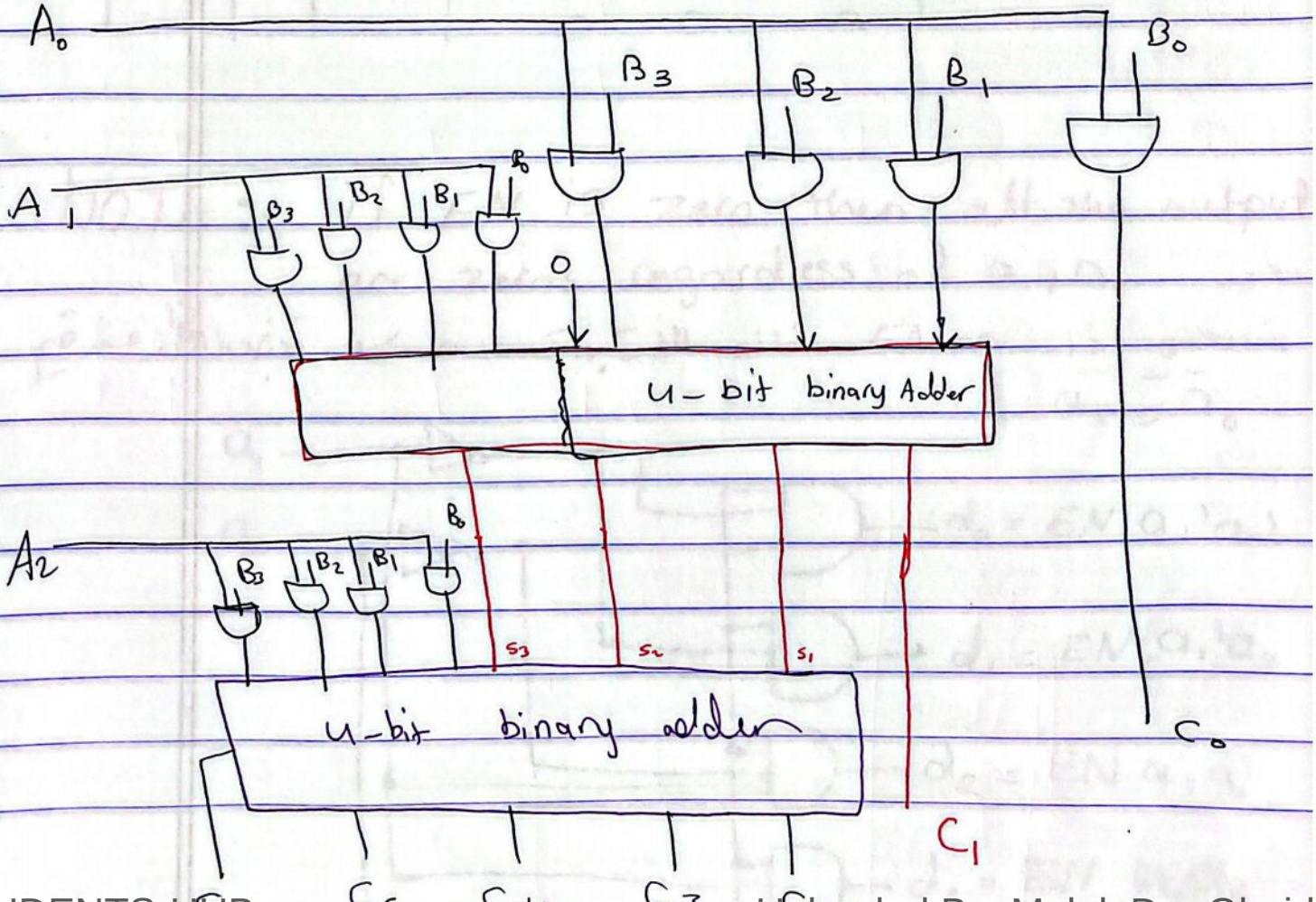
→ نخرج ثم نمثلها ب أبسط شكل

Note:- اذا كان عدد المشروط في الفنكشن كبير نستعمل NOR gate ومع maxterms الـ (NOR gate)



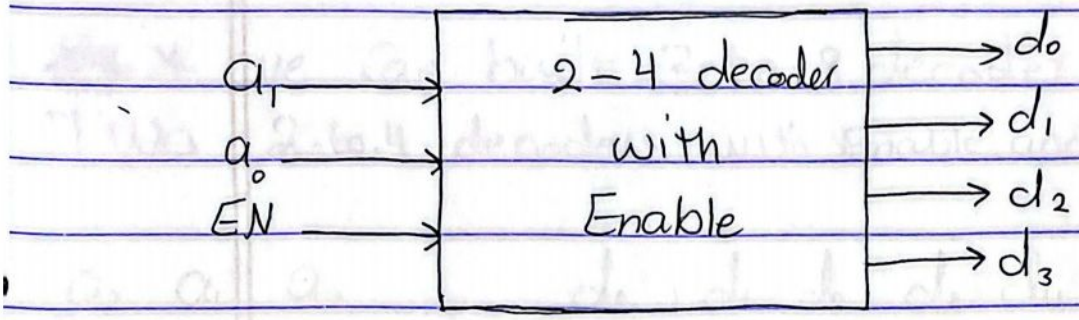
4 bits x 3 bits Multiplier → اكبر رقم ممكن نحصل عليه: 1111 (15) × 111 (7) = 7-bit تحتاج (105)





Decoder with Enable Input

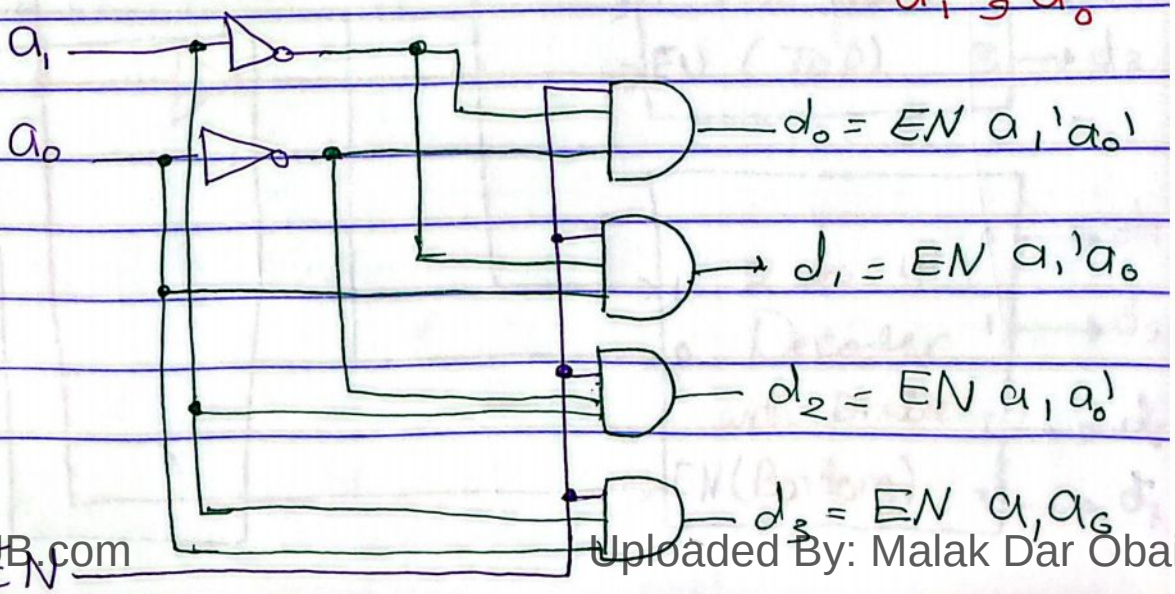
2 to 4 Decoder with enable Input



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

NOTE:- if EN is zero then all the output are zeros regardless of a_1, a_0

يعني اذا EN=0 فان الاوتيبوت = 0 كله بغض النظر عن قيم a_1 و a_0

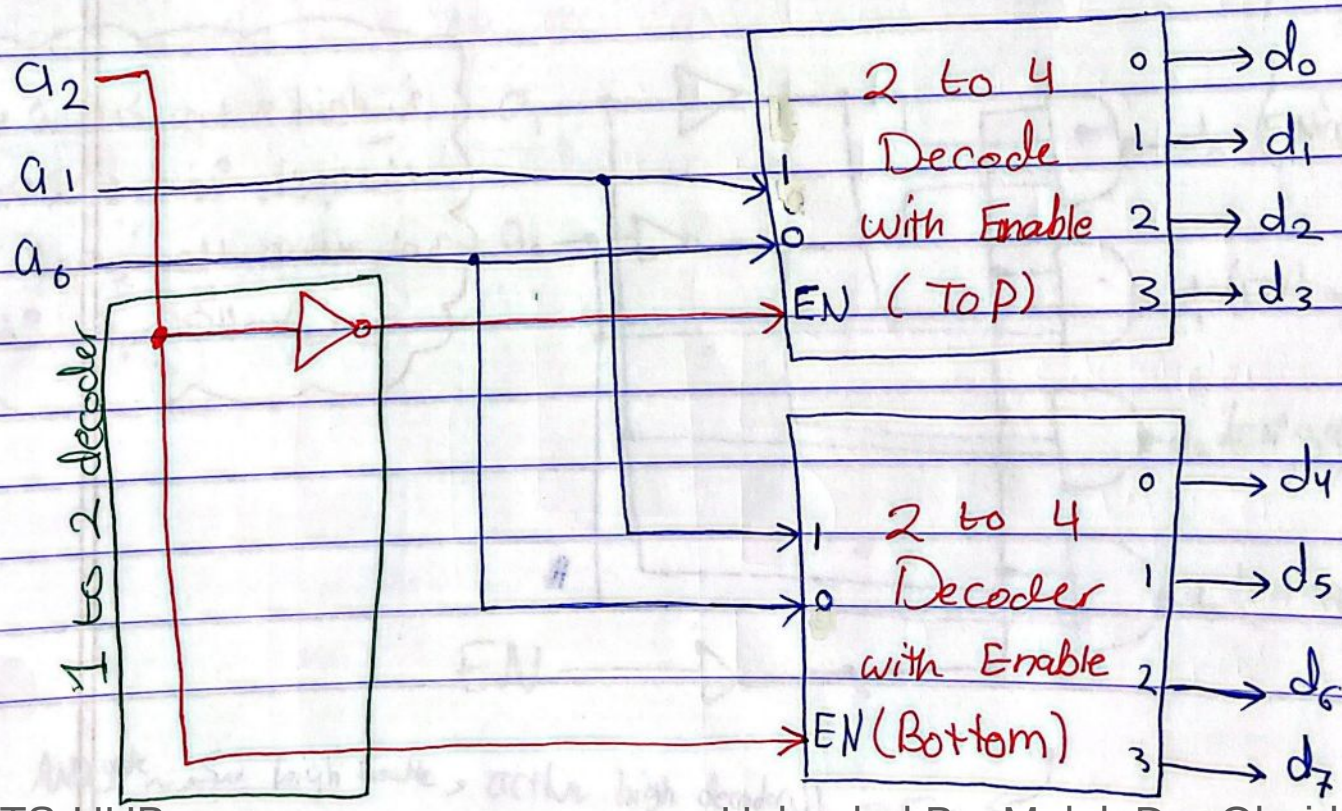


Building larger Decoders

- We can build large decoders using smaller ones

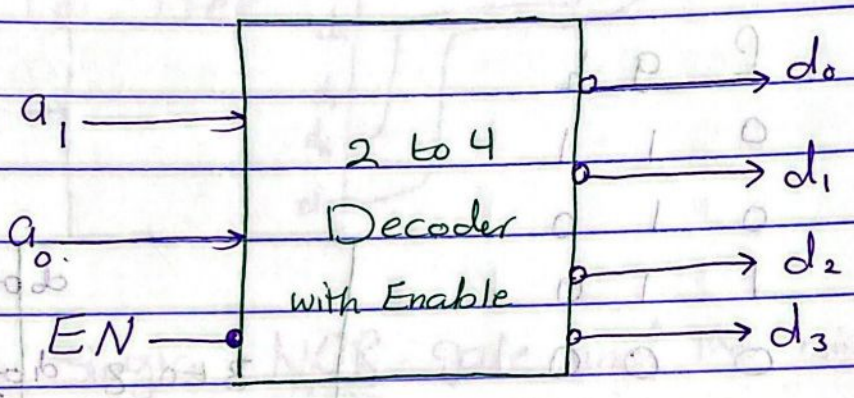
* We can built 3-to-8 decoder using 2-TWO 2-to-4 decoders with Enable and inverter (1 to 2 decoder)

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

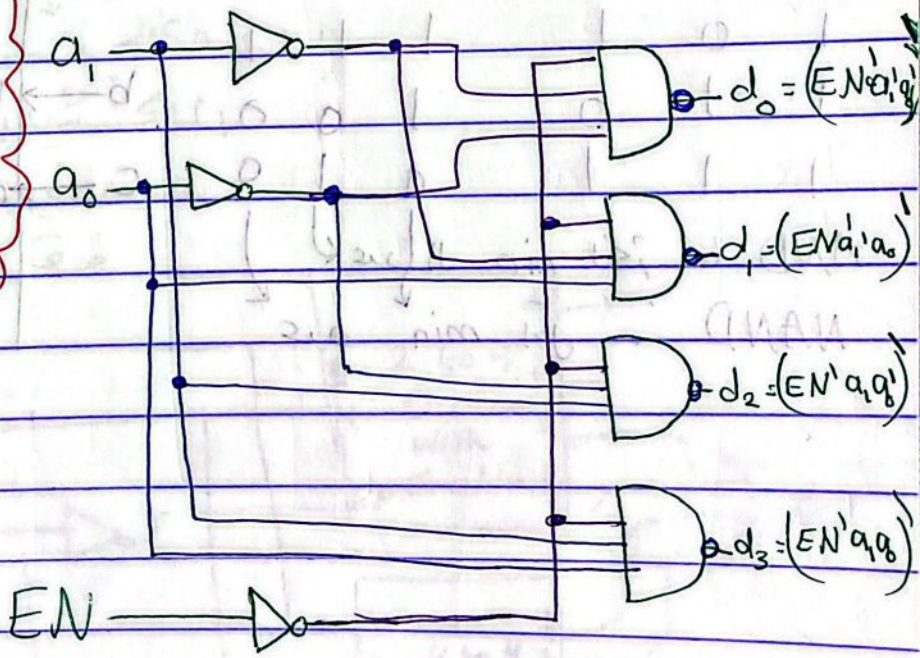


Active low decoder with active low Enable
هون كل مخرج active high بتصير 1 والباقي 0

EN	a_1	a_0	d_0	d_1	d_2	d_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0



في active high كان 1 يرمز انه الجهاز شغال لكنه هنا في active low المخرج هو الذي يرمز إلى انه الجهاز شغال



مع AND gate بتكون high Enable و active high decoder الـ
inverter بدون Enable

Using NAND Decoders

* NAND decoders can be used to implement functions

- Use NAND gates to output the minterms (if fewer ones)

NAND gate تُستخدم maxterms اقل من ال minterms * اذا عدد ال

- Use AND gate to output the maxterms (if fewer ones)

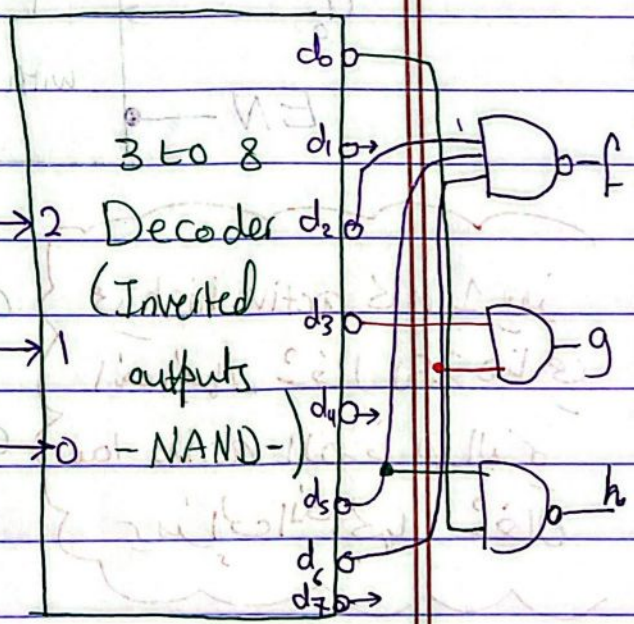
AND gate تُستخدم minterms أقل من ال maxterms * اذا عدد ال

Ex $f = \Sigma(2,5,6)$ $g = \Pi(3,6)$ $h = \Sigma(0,5)$

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

AND ← عدد ال min اكثر

NAND ← عدد min اقل

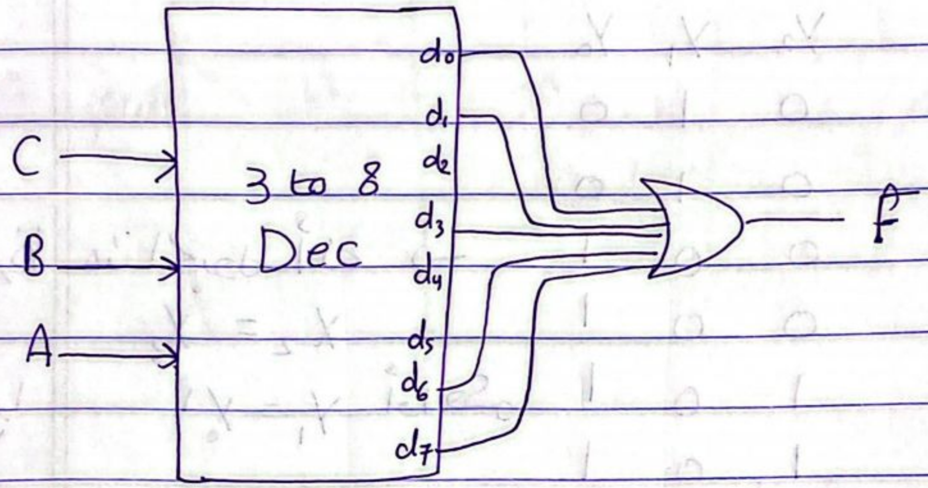


Ex Implement this boolean function $F_{(A,B,C)} = AB + A'C + A'B$

① Using single 3to 8 decoder & OR gate

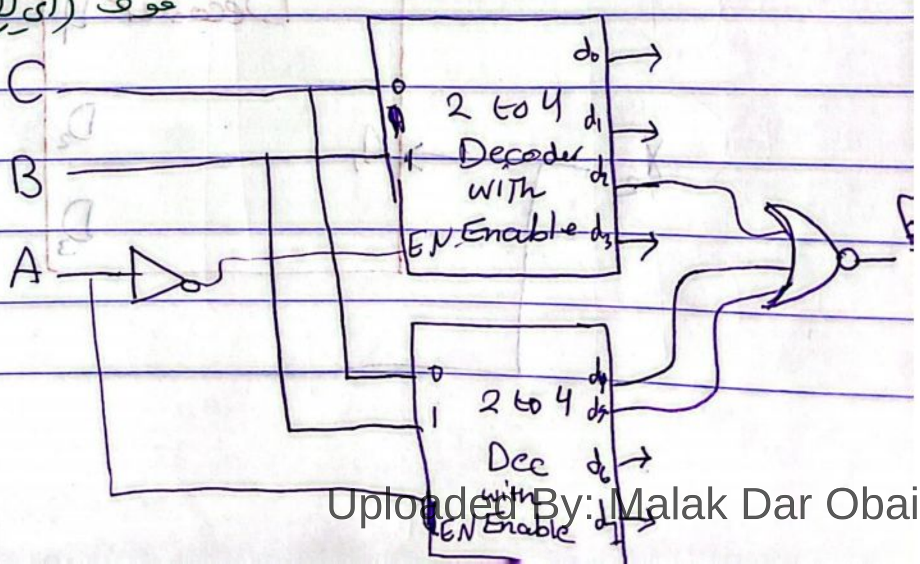
① اول اشي لازم نحل الحدود الناقصة بالطرق السابقة
فنطلع صيغة الـ مِنترات كالآتي

$F = \sum m(0,1,3,6,7)$



② Using single NOR gate and the minimum number of 2 to 4 decoders with Enable

بما إنه حكالي باستخدام NOR ∴ بدنا نستخدم
الـ ماكستيرمز لو ما حكالي نستخدم 2 to 4 decoder
بس نستخدم NOR مع الحدود التي لم نختارها
فوق (اي الـ maxterms)



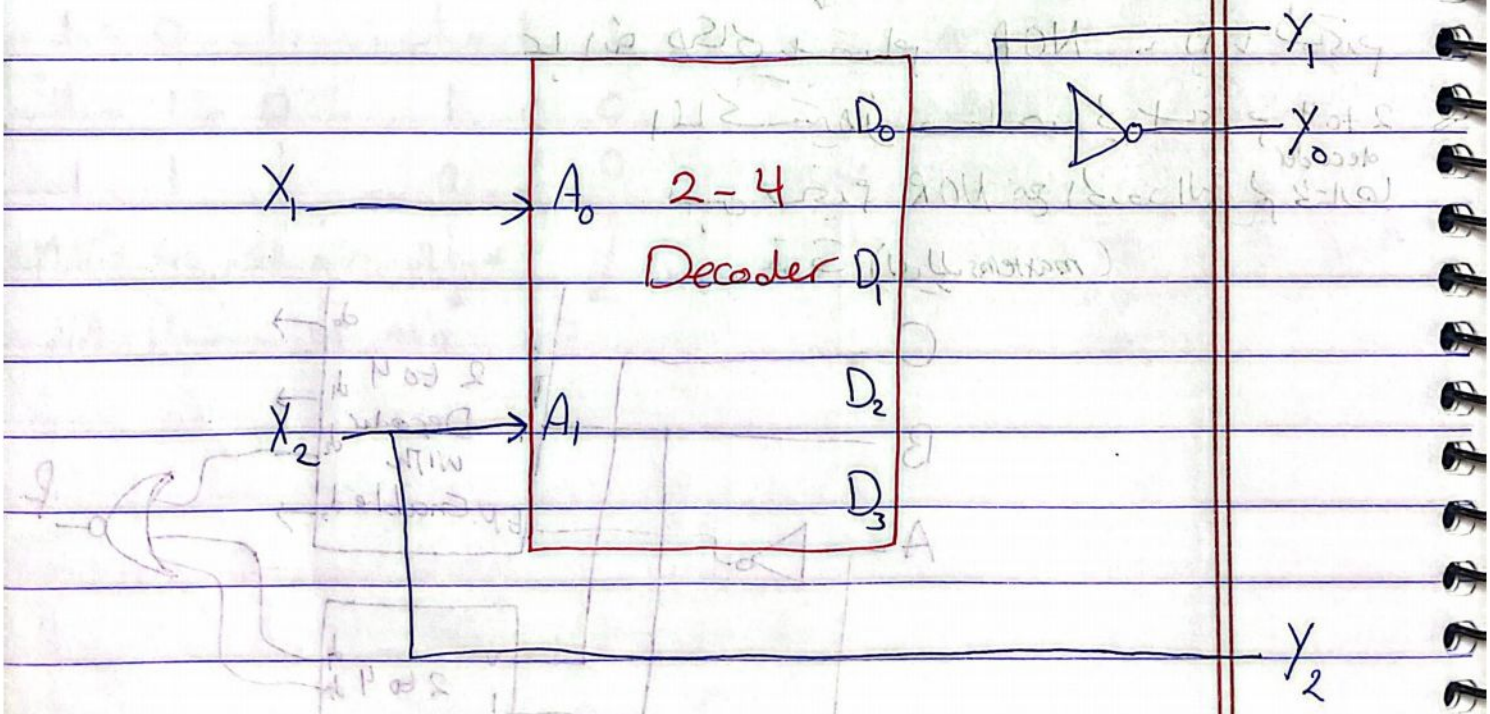
Tricky question: Consider the following truth table in which X_2, X_1 and X_0 are the inputs and Y_2, Y_1 and Y_0 are the outputs. Using a minimum-size decoder and minimum number of additional gates, show how to implement Y_2, Y_1, Y_0. Your additional logic gates must use the smallest possible num of inputs

X_2	X_1	X_0	Y_2	Y_1	Y_0
0	0	0	0	1	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	0	0	1
1	0	0	1	0	1
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	1	0	1

نلاحظ من الجدول أن → $X_2 = Y_2$

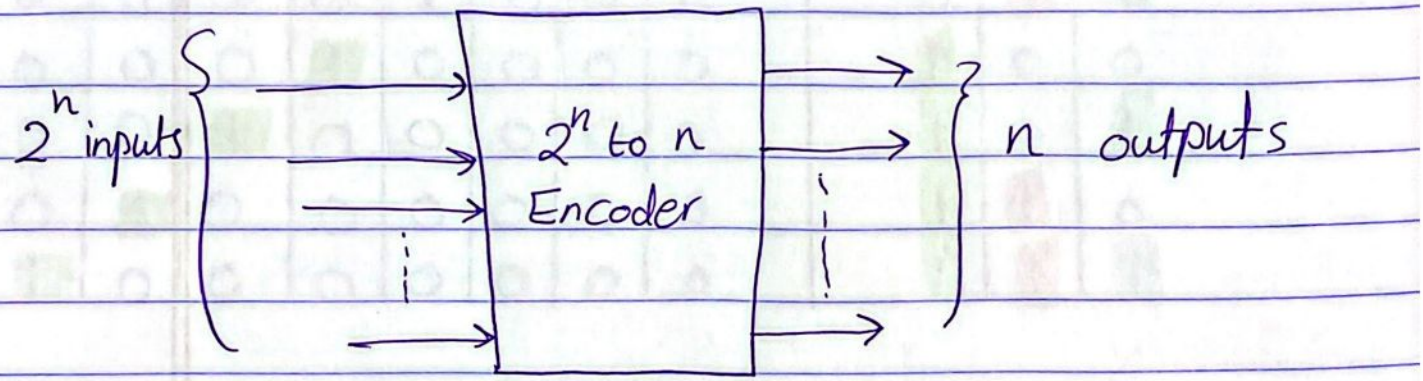
وأيضا أيضا $Y_1 = Y_0$

* بتقدر هان تستخدم 2 to 4 decoder



Encoders :- inverse operation of Decoders

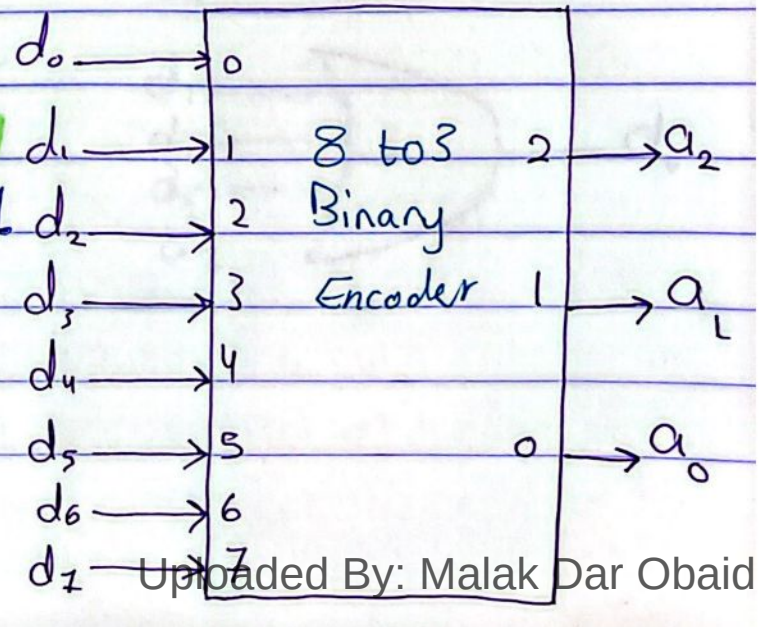
* it converts 2^n input to n-bit output code
* the output indicates which input is active (logic 1)
* One input should be 1 and all others must be zero



Ex 8 to 3 Binary Encoder

* output here is 3 and the input are 8, one input is 1 and all the others are zero's

- Encoder generates the output binary code for the active input
- output is not specified if more than one input is 1

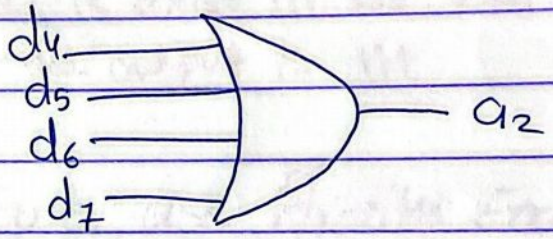


(هذا يسمى truth table لهذا الـ Decoder)

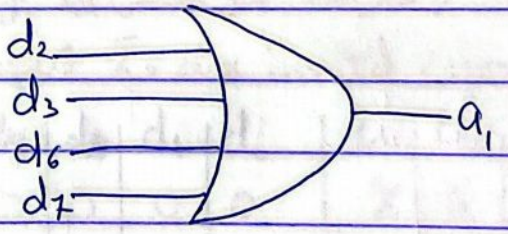
8 to 3 Binary Encoder Implementation

inputs								output		
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

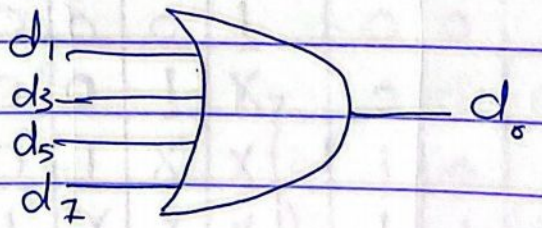
$$a_2 = d_4 + d_5 + d_6 + d_7$$



$$a_1 = d_2 + d_3 + d_6 + d_7$$



$$a_0 = d_1 + d_3 + d_5 + d_7$$



8 to 3 binary encoder can be implemented using 3 4-input OR Gates

Binary Encoder Limitations

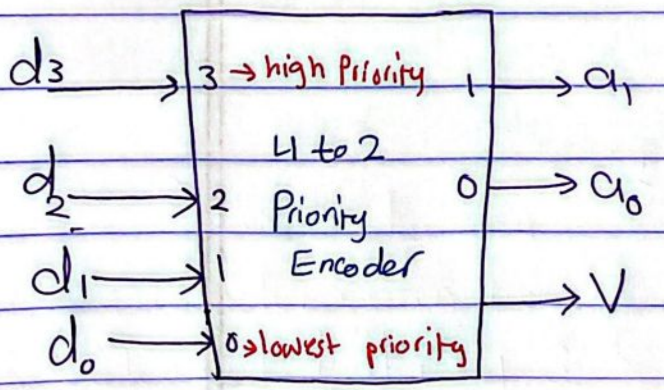
as I mentioned before One input should be 1 and all the other must be Zero

So if more than one input is 1 then the output will be incorrect

Ex if $d_3 = d_6 = 1$

$\therefore a_2 a_1 a_0 = 111$ → incorrect (d_3 exist in a_1 and a_0 also d_6 is exist in a_2 & a_1) So the output is 111

to resolve this problem we use Priority Encoder
(فكرته انو نعطي ال priority لواحد من ال input)



صار invalid لانه ولا input قيمته 1

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

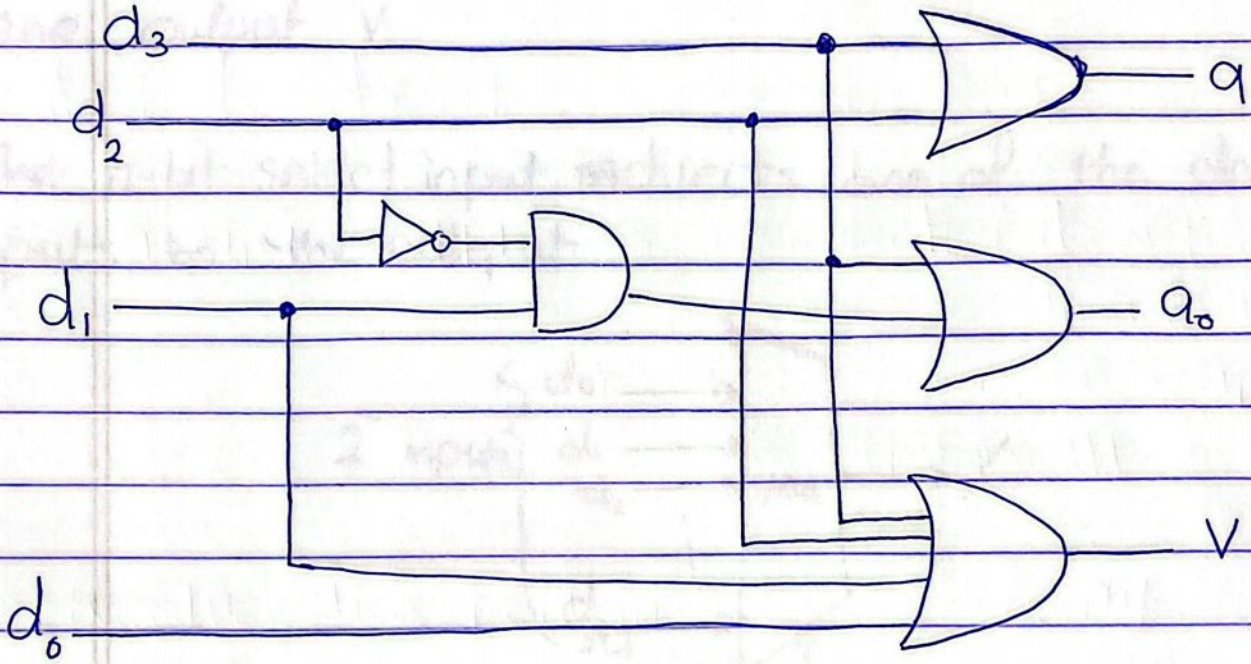
بما انه عنا input=1 نتجاهل قيمة باقي القيم

* if all inputs are Zero's the V (valid) output is zero indicates that all inputs are zeros

بعد عمل k-map لكل output تصبح قيم الـ output كالتالي

$$a_1 = d_3 + d_2$$
$$a_0 = d_3 + d_1 d_2'$$

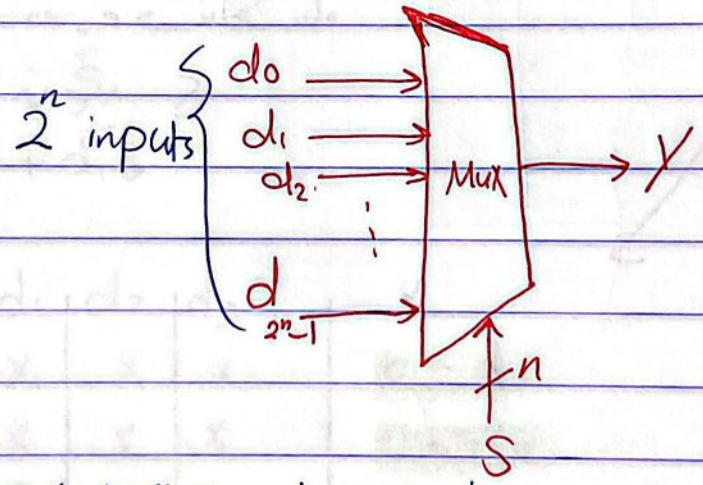
$$V = d_3 + d_2 + d_1 + d_0$$



ممكن يكون مع Encoder و Decoder

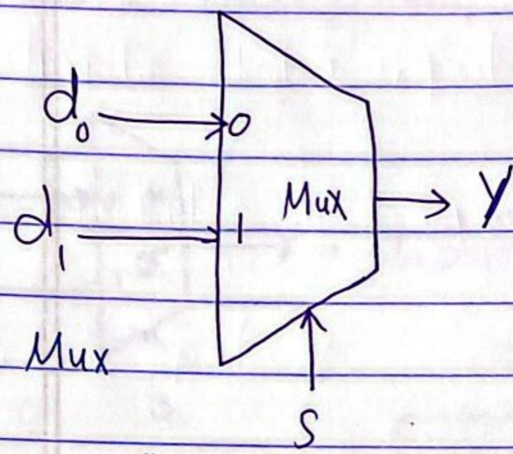
Multiplixers (Mux) : combinitional circuit that has :-

* Multiple data inputs (typically 2^n) to select from
* An n-bit select input S used for control
* One output y
* The n-bit select input directs one of the data inputs to the output

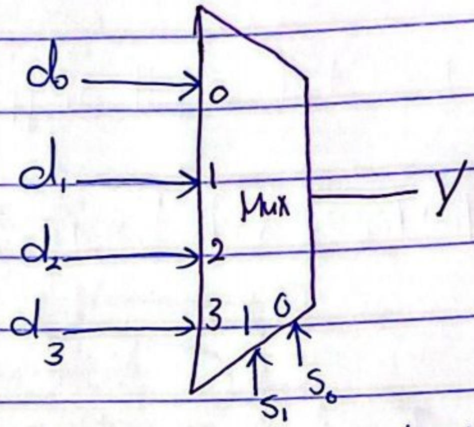


تتحكم فيه مثلاً لو كان 0 فال output سوف يكون مثل d_0 ولو كان 1 الاوتبوت رح يكون d_1 وهكذا

Examples of Multiplexers



2 to 1 Mux



4 to 1 Multiplexer

مثل ما شرحت سابقاً لو $S=0$
الاوتبوت رح يكون d_0
ولو كانت قيمة $S=1$ فالاوتبوت
رح يكون d_1

$$Y = d_0 S' + d_1 S$$

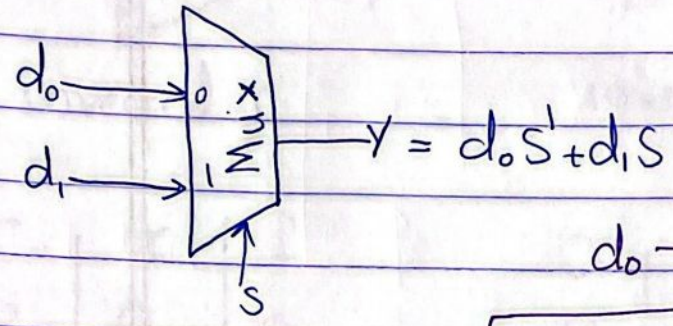
S_1	S_0	d_0	d_1	d_2	d_3	Y
0	0	0	X	X	X	0 = d_0
0	0	1	X	X	X	1 = d_0
0	1	X	0	X	X	0 = d_1
0	1	X	1	X	X	1 = d_1
1	0	X	X	0	X	0 = d_2
1	0	X	X	1	X	1 = d_2
1	1	X	X	X	0	0 = d_3
1	1	X	X	X	1	1 = d_3

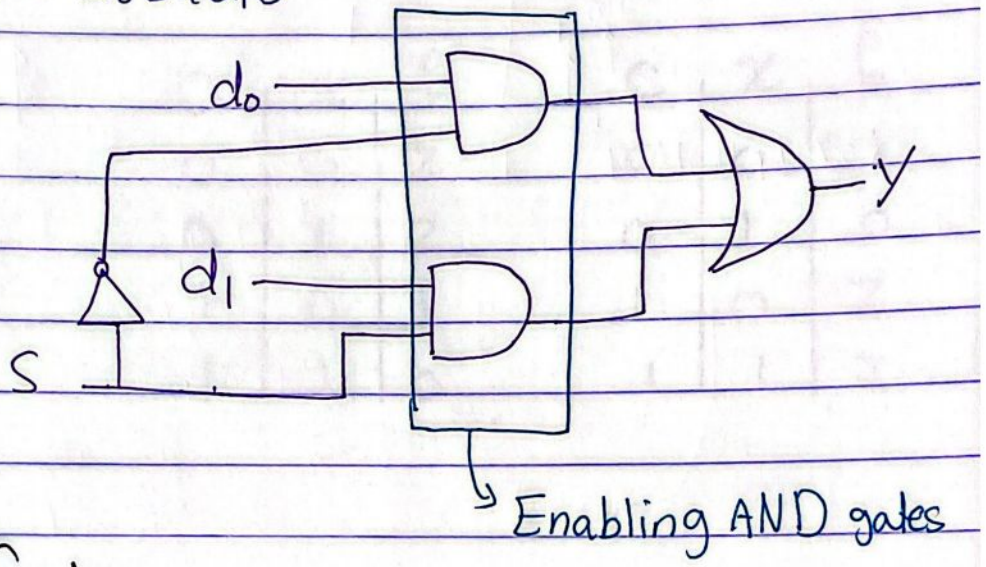
$$Y = d_0 S_1' S_0' + d_1 S_1' S_0 + d_2 S_1 S_0' + d_3 S_1 S_0$$

لـ توضح اللازم تقسيم الطريقة والطرقي
البايزي حسب قيمة d_U

Implementing Multiplixers

* 2 to 1 multiplexer

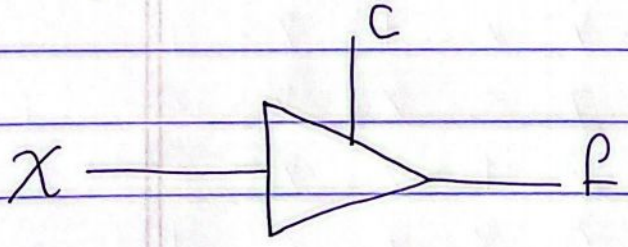




* 3-State Gate

it has 3 possible outputs 0, 1, Z ← hi-Impedance

Z means that the output is disconnected from the input



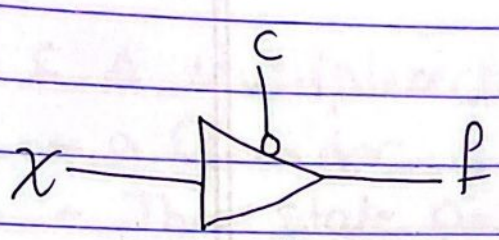
C	x	f
0	0	Z
0	1	Z
1	0	0
1	1	1

اذا $C = 0$ تمنع البوابة من مرور

$f = Z$ و

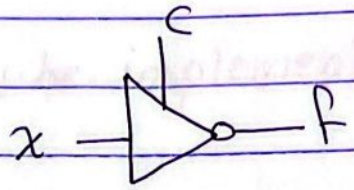
$f = x$ و تسمح بمرور input لكن اذا $C = 1$

Variations of 3-State Gate



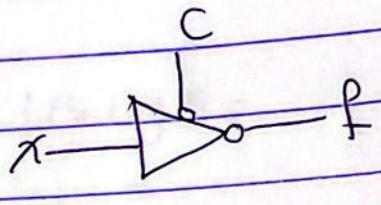
inverted c

c	x	f
0	0	0
0	1	1
1	0	Z
1	1	Z



inverted f

c	x	f
0	0	Z
0	1	Z
1	0	1
1	1	0

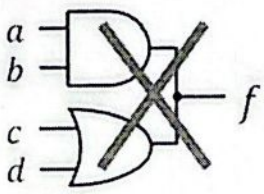


inverted c, f

c	x	f
0	0	1
0	1	0
1	0	Z
1	1	Z

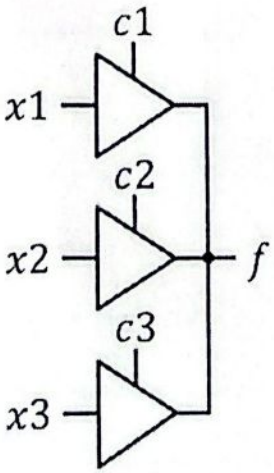
Wired Output

Logic gates with 0 and 1 outputs **cannot** have their outputs wired together



This will result in a **short circuit** that will burn the gates

3-state gates **can wire** their outputs together



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

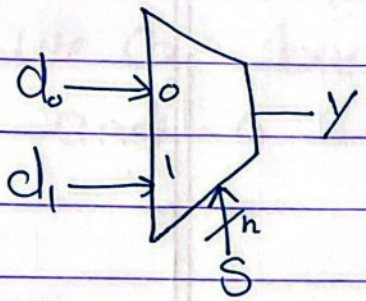
At most one 3-state gate can be enabled at a time

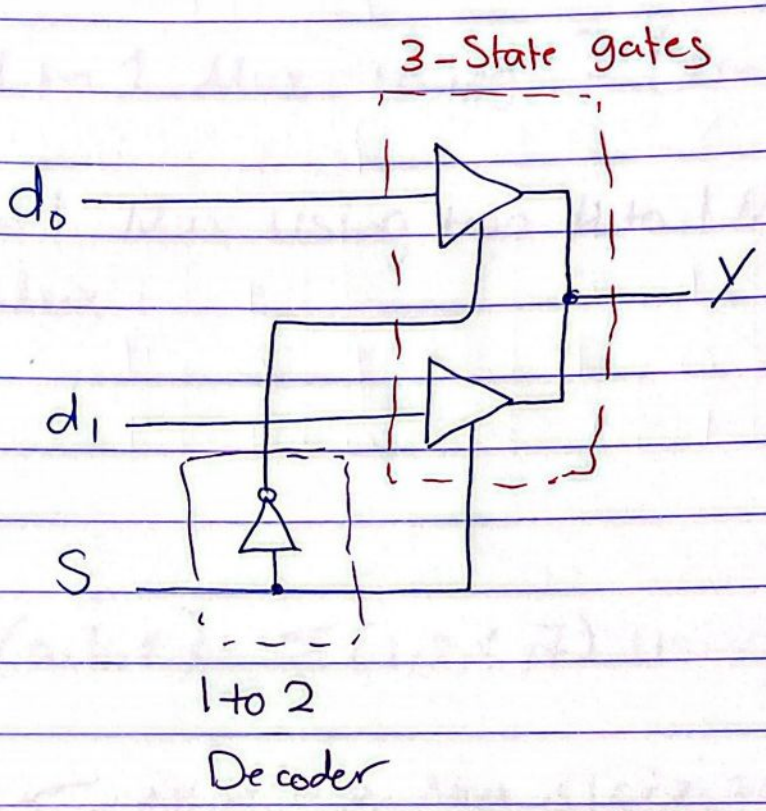
Otherwise, conflicting outputs will burn the circuit

Implementing Multiplexers with 3-State Gates

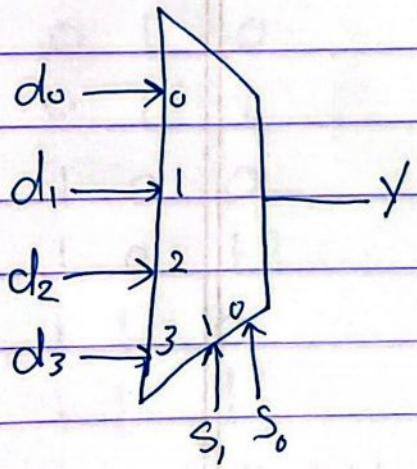
* A Multiplexer can be implemented using :-
  - a Decoder
  - Three state gates

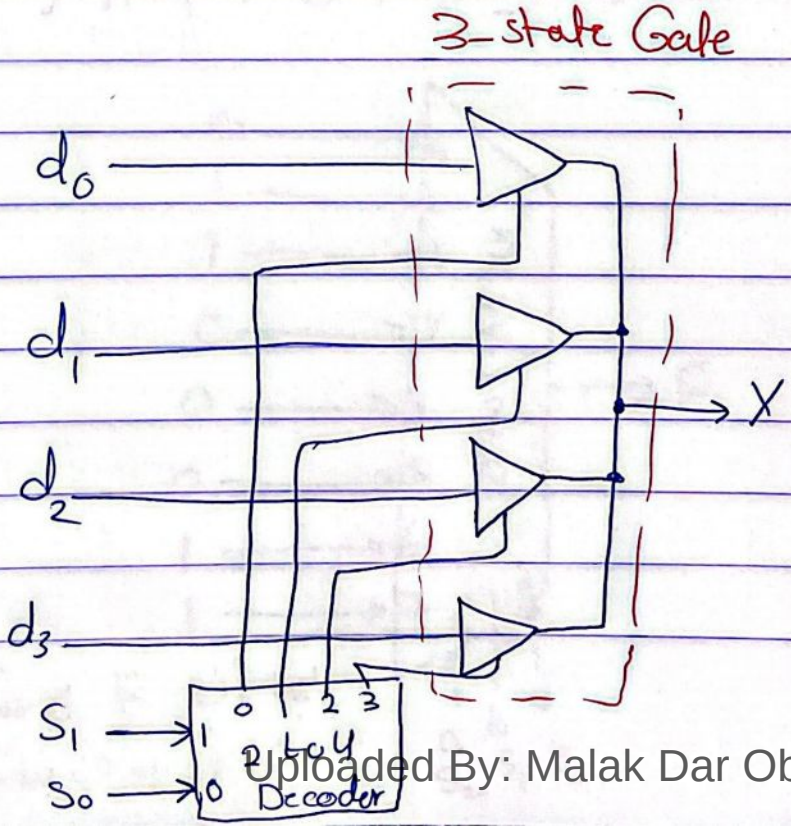
2x1 mux





4x1 Mux





Building Larger Multiplexers

- larger multiplexers can be built hierarchically using smaller ones

- We can build 4 to 1 Mux using 3 (2 to 1 Muxes)

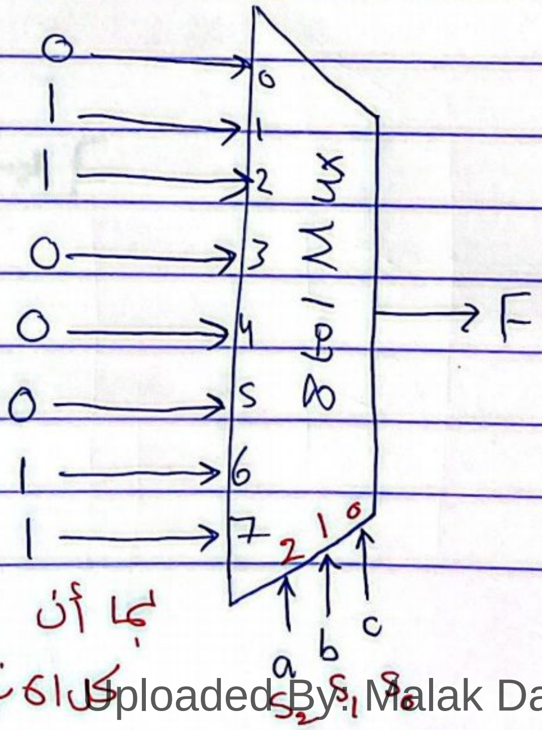
- We can build 8 to 1 Mux using two 4 to 1 Muxes and a 2 to 1 Mux

Examples

① Implement $F(a,b,c) = \Sigma(1,2,6,7)$ using a Mux

* متى نعرف اي Mux رح نرسم لازم نرسم truth table

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

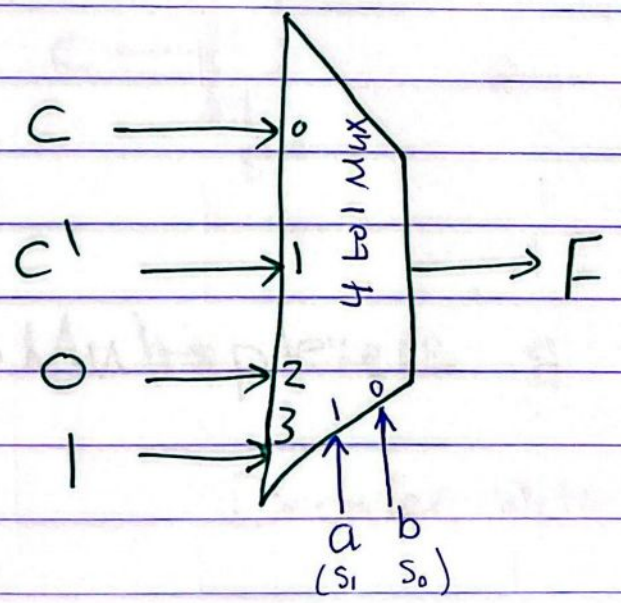


بما أن F لا تساوي a/a'/b/b'/c/c' اذا كل ال3 بتوت بنعتبرهن select ونستخدم 8to1 MUX

* بنقدر نعمل 6 مثال الباقية باستخدام 4 to 1 Mux ويكون عنا 2 selects

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	

حتى نحل 2 select بنقسم الجدول بحيث انه فيه a,b مرة 0 0 ومرة 0 1 و 1 0 و 1 1
ثم نكتب مرة F ← 0 / 1 / c' / c كالآتي
ونفس الطريقة نطبقه لـ 4 رموز وغيره



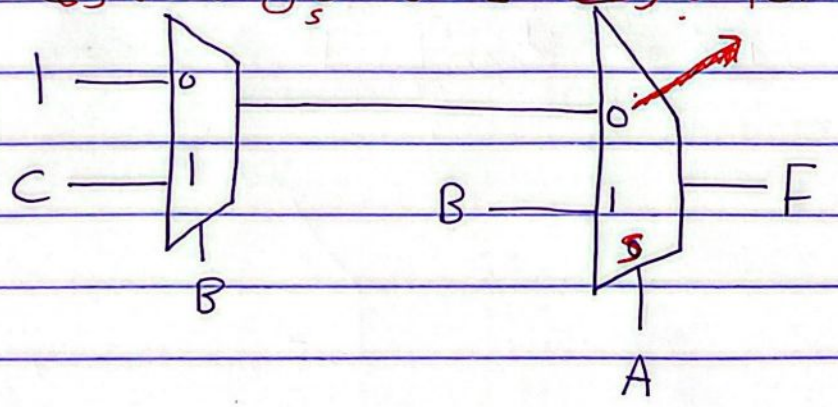
② Implement $F(A,B,C) = AB + A'C + A'B'$
Using 2x1 Mux/Muxes (minimum num of Mux)

$F = \Sigma(0,1,3,6,7)$ اول خطوة نكمل الكود الناقص ونكتب F

ثم نقسم

A	B	C	F	
0	0	0	1	$F = 1$
0	0	1	1	
0	1	0	0	
0	1	1	1	$F = C$
1	0	0	0	
1	0	1	0	
1	1	0	1	$F = B$
1	1	1	1	

هان كا السيليكت (A) بساوي صفر
فان الانبوت اله قيمتين بس ما بنفع فرضنا فرعنا كا السيليكت
صو B لانه اذا بتلاقظ كا B=0 فإن F=1 وكا B=1 فإن F=C



بكتبعنها A=1 فإن كل الانبوت بتكون B

* DeMultiplexers عكس ال Multiplexers

وهو نفسه Decoder With Enable