Digital Logic

By: Raad Abu Al soud

Summary

Chapter3

POWER UNIT

Chapter 3 : combinational logic design.

Part 1

* كل ماسبقة هو تحليل للدوائر الرقمية , هذا الفصل يتحدث عن تصميم الدوائر و وصف آلية عملها.

* خطوات تصميم أي دارة :-

1- معرفة الهدف من هذه الدارة و تحديد وصف لها (إذا لم يكن موجوداً) ---- Specification

2- استخراج الـ Truth Table من الوصف أو الـ Booleanequations والذي يحدد العلاقة بين الـ inputs والـ output ---- Formalation

3- كتابة الـ logic function و ذلك عن طريق تمثيل الـ Truth.T بـ K-map optimization

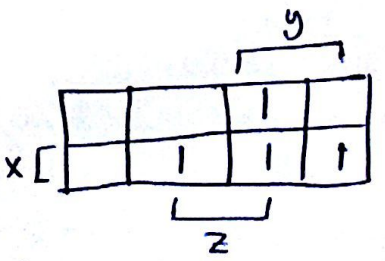
4- رسم logic diagram من الـ function's باستخدام NAND/ Technology Mapping الـ

Ex : design a circuit with 3 inputs, the output is "1" is the number of 1's in the input value is grater than number of 0's.

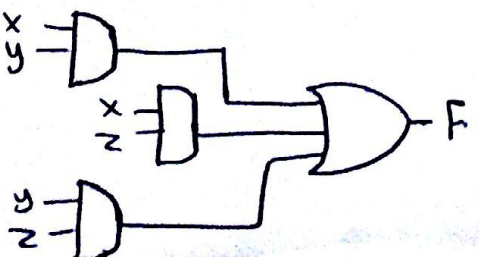
الوصف:-
* دارة بـ 3 input و 1 output
لكل output يتم رسم Kmap و Function
* الـ output هو "1" اذا كان عدد الـ 1's في الـ input اكبر من عدد الـ 0's
* الـ output هو "0" اذا كان عكس ذلك

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

منفلان عدد الـ 1's



$F(x,y,z) = xy + xz + yz$



Ex: Design of a BCD-to-Excess-3 code converter.

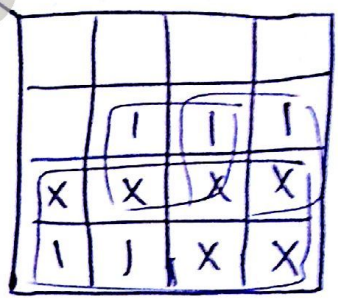
كود يزايد ٣ ارقام على الـ BCD

BCD → input 4 Variable

E3 → out put 4 Variable Function (w, x, y, z)

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
⋮	
1 0 0 1	1 1 0 0
1 0 1 0	x x x x
⋮	x x x x
1 1 1 1	x x x x

w للـ ←

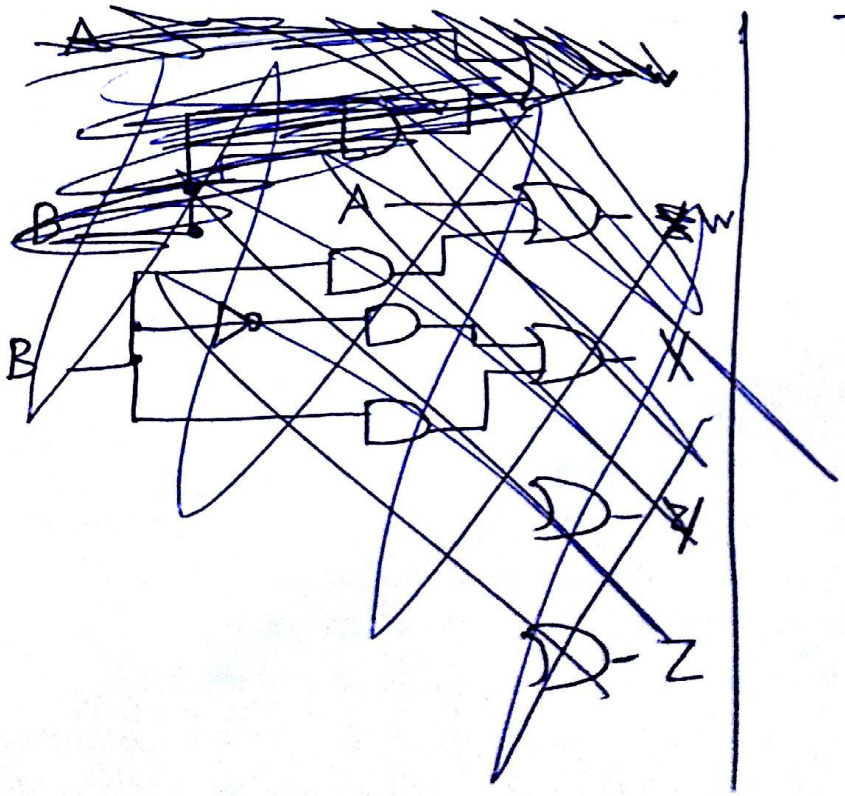


وتعمل لـ x, y, z

$$w = A + BC + BD$$
$$x = \bar{B}C + \bar{B}D + B\bar{C}\bar{D}$$
$$y = CD + \bar{C}\bar{D}$$
$$z = \bar{D}$$

* نلاحظ ان الـ input المشتركة بين الـ function هي C, D

(اقل عدد)

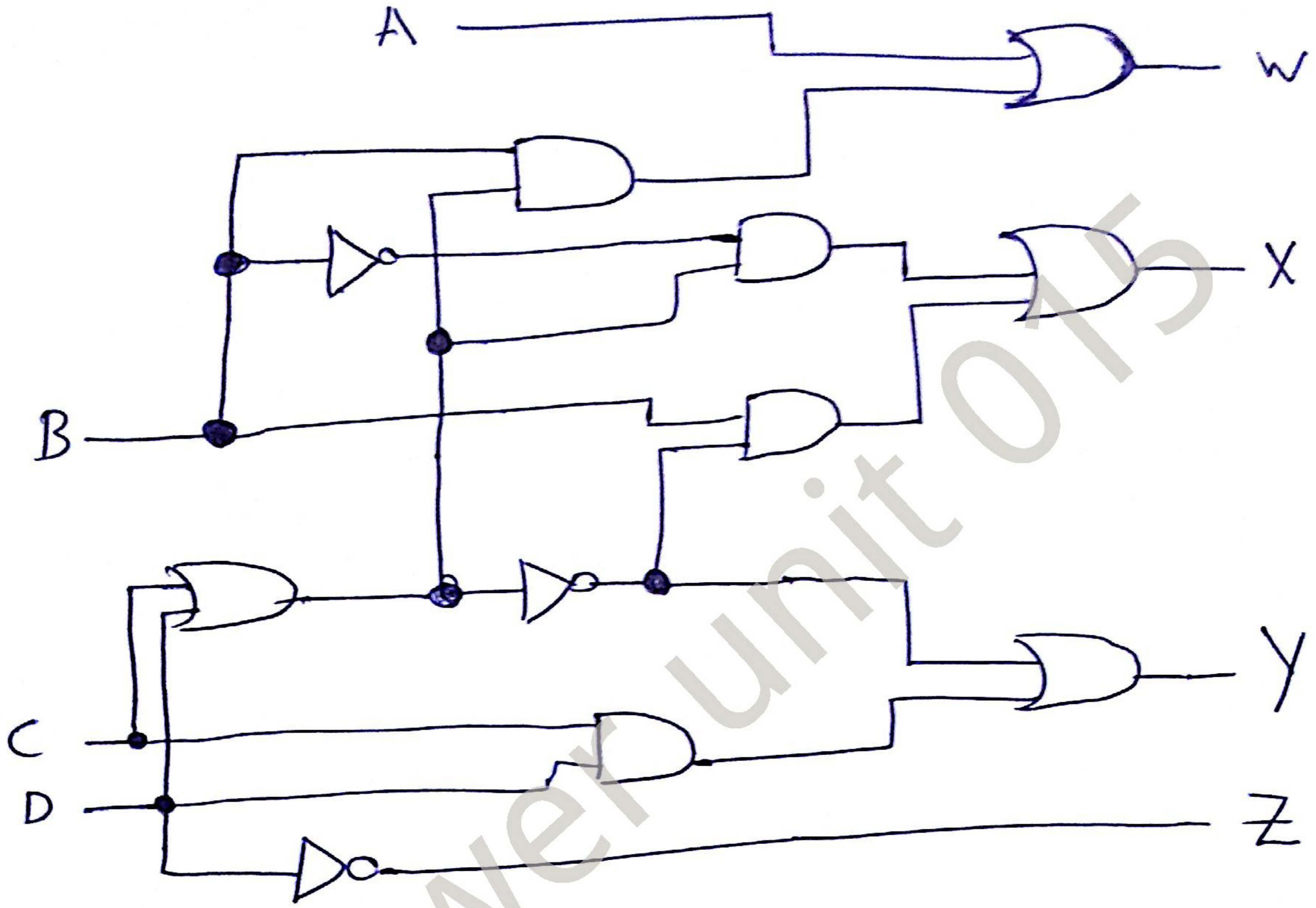


* نفرض أن $T_1 = C + D$

وعليه

$$w = A + BT_1$$
$$x = \bar{B}T_1 + B\bar{T}_1$$
$$y = CD + \bar{T}_1$$
$$z = \bar{D}$$

هاد بس لتقليل الـ cost



Design a simple (4-Bit) even parity checker that gives an output = 1 when the number of it is even

w	x	y	z	F
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	[illegible]

even parity

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

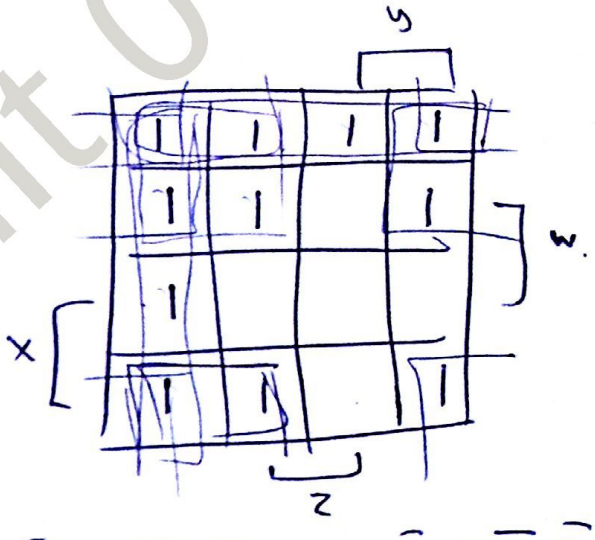
F = ----------

Ex: Design a 4 Bit minority fun with out put = 1 with the number of 1's Less or equal number of 0's

- input = w x y z
- output = F

الوصف [* اذا كان عدد الـ 1's اقل او يساوي الـ 0's فان القيمة في 1

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



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

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

حيث: $1 + \bar{w} = 1$

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

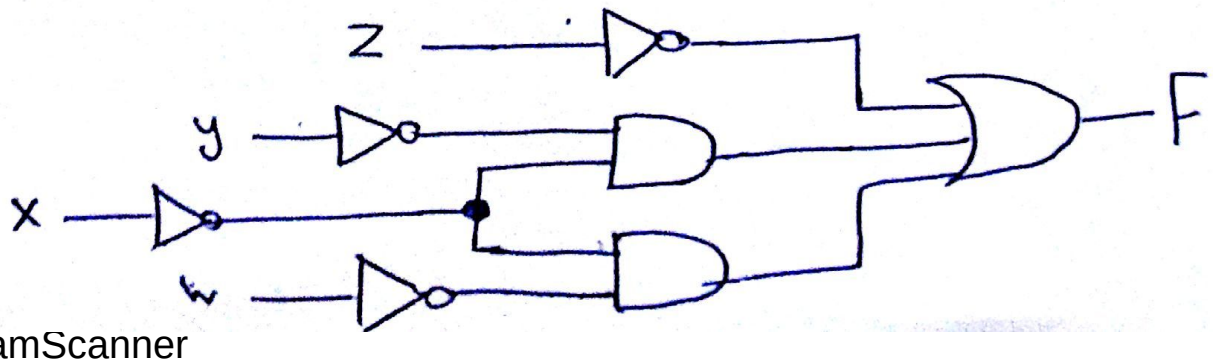
$$= \bar{z} + \bar{y}(\bar{x} + \bar{w}) + \bar{w}\bar{y}$$

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

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

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

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



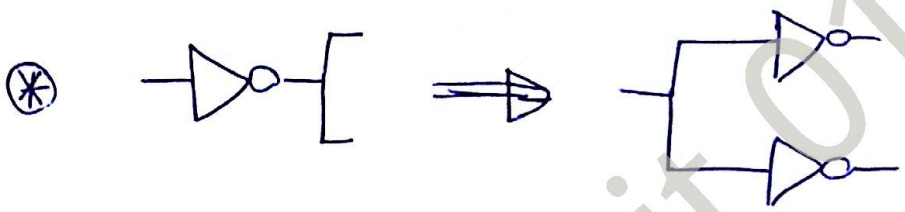
Technology mapping

* تعني تحويل الدارة من (And-or-inverter) الى AOL ← فقط NAND أو NOR

1. replace every "AOL" with it's equivalent NAND or NOR

الخطوات

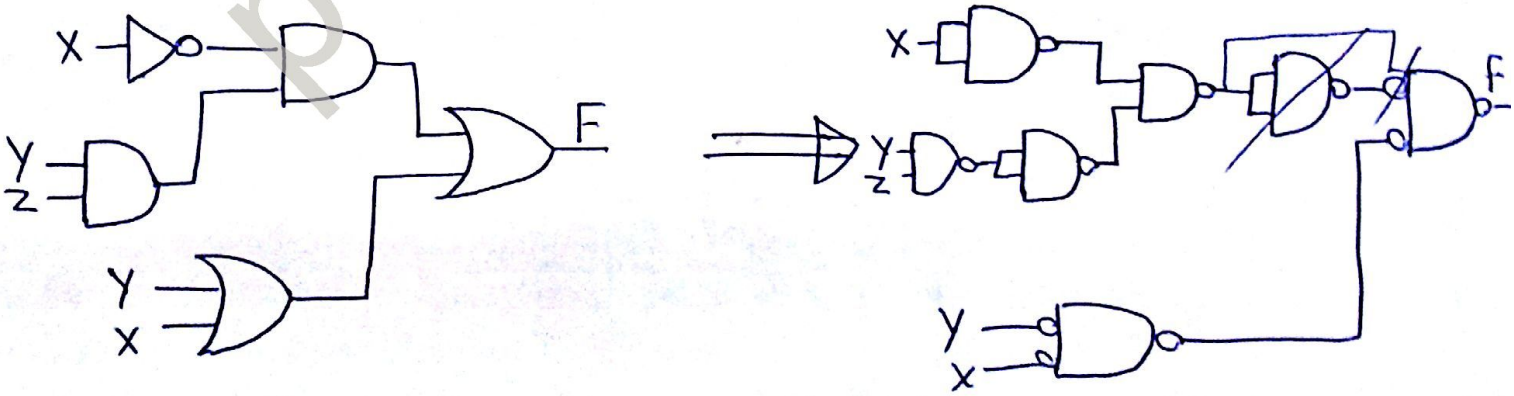
2. if :



Push inverters thrugh circuit fan-out point

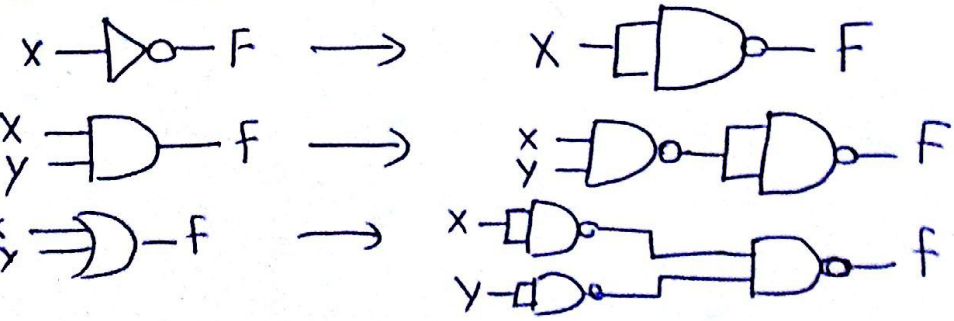
cancel pairs of inverters

Ex : using technology mapping with NAND gate only

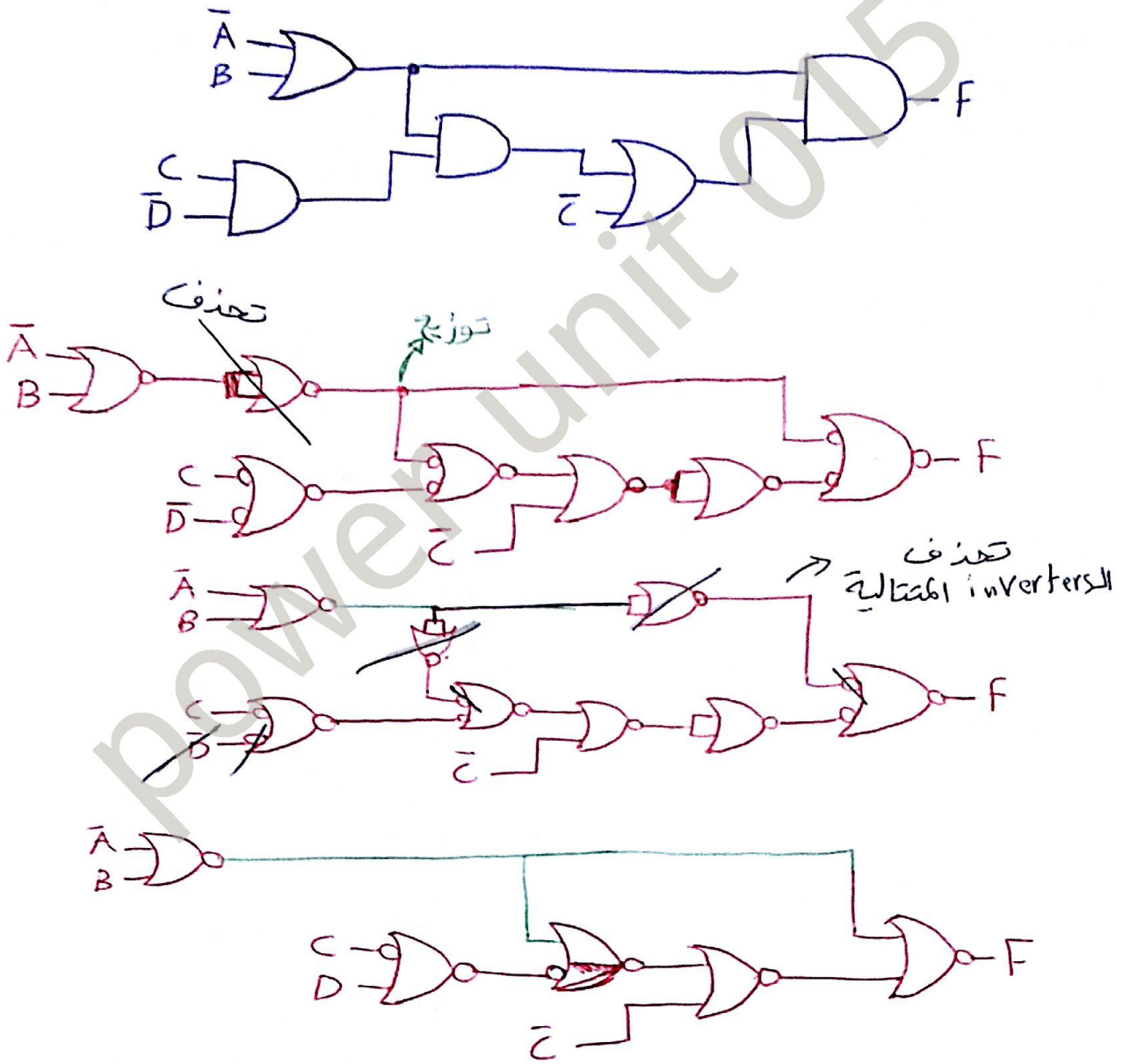


Remember

NAND

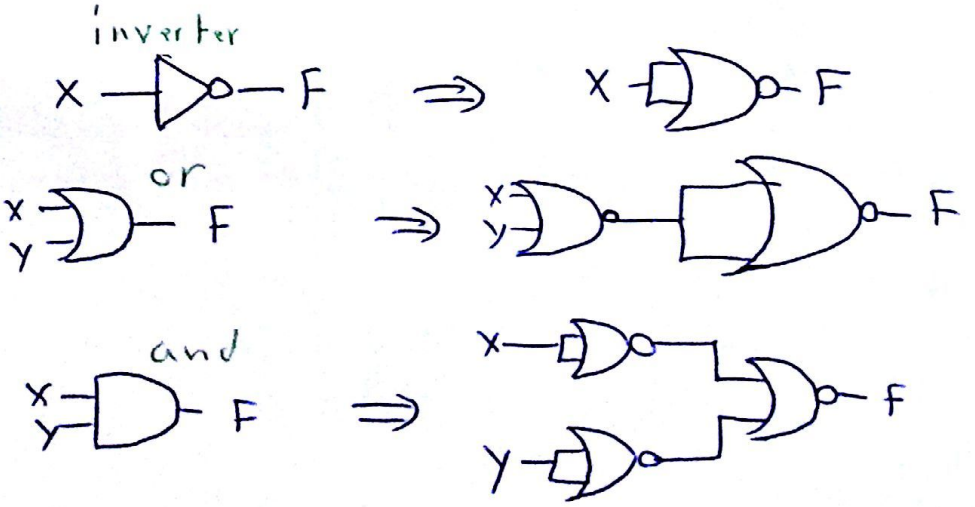


Ex: using tech-mapping with NOR gates only.



Remark

تحويل الى NOR



Chapter3 Part 2

"Value fixing"

* يوجد نوعان من ال single bit function

Volt (V_{cc}/V_{DD})

- F = 1 / 1 —wire— F = 1
- F = 0 / 0 — F = 0
  - ↳ No Volt

- X —wire— F = X ⇐ transfer *
- X —▷o— F = $\bar{X}$ ⇐ invert *

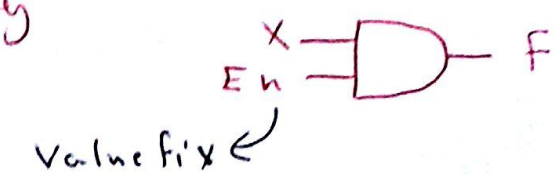
→ Value fixing

- ال transfer يعني انه ينقل ~~البيانات~~ المعلومات كما هي دون تغيير
- ال invert " " " " " ولكن يعكسها

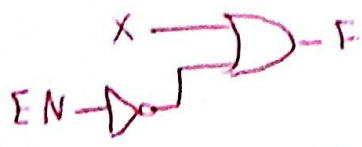
ال Value fixing ⇐ تعني ان ال output ثابت دائماً وهو القيمة إما 0 أو 1

استخدام ال Value fix

* for Enabling



X pass if EN = 1, otherwise, if EN = 0
the output is fixed 0



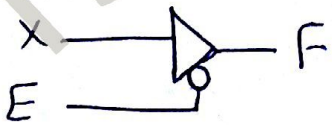
X pass if EN = 1, otherwise, if EN = 0
the output is fixed 1

Tri-state Buffer "Enable"

* there are Two type of it:

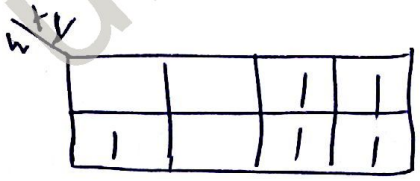
① Active high : enable when E = 1

② Active low : " " " = 0



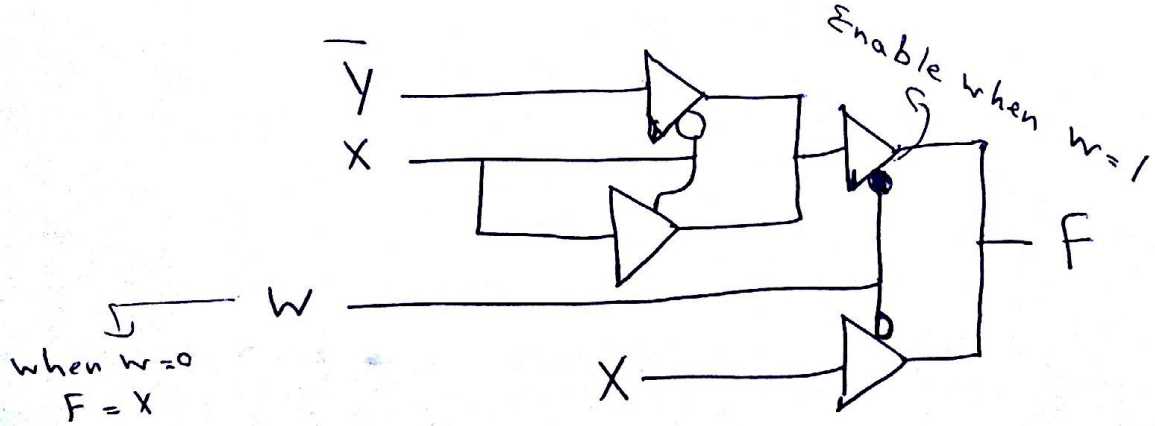
Ex: implement the function $F(w,x,y) = \bar{w}x + w\bar{y} + xy$ using tri-state buffers only.

① طلع الـ truth table
من الـ K-map



	w	x	y	F
when w=0 F=x	0	0	0	0
	0	0	1	0
	0	1	0	1
	0	1	1	1
when w=1 and x=0 ⇓ F=ȳ	1	0	0	1
	1	0	1	0
when w=1 x=1 ⇓ F=x or F=1	1	1	0	1
	1	1	1	1

* قارن قيمة المتغير الأول بـ قيمة الـ output وعليه نخرج العلاقة بينهم



Functional Blocks

* من الصعب احياناً بناء دوائر رقمية باستخدام الأسلاك والـ gates بسبب كبر حجم الدارة وتعقيدها لذلك نستخدم الـ Blocks الجاهزة التي تقوم بوظيفة معينة ومنها

1) Decoder 2) Encoder 3) multiplexor (MUX)
4) Demultiplexor (Demux)

* احياناً تحوي هذه الـ Blocks على خط إضافي هو (Enable) وهو يتحكم في عمل الـ Block من ناحية on/off

I) Decoder

* يستخدم في عملية توسيع البيانات : اي فك ضغطها :- حيث تنقل البيانات على شكل مضغوط ثم نحتاج الى فك ضغطها وذلك من خلال الـ Decoder

للـ Decoder
1) n input (binary-code)
2) $m = 2^n$ output (also binary code)

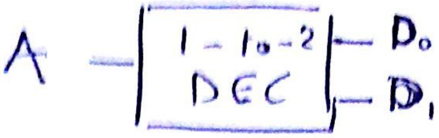
حيث n هي عدد الاسلاك أو المتغيرات

$n \le m \le 2^n$ (n: input, m: output)

* انواع الـ Decoder
* 4-to-16
* 3-to-8
* 2-to-4
* 1-to-2

حيث ان الشق الاول هو عدد الـ inputs بينما الشق الثاني هو عدد الـ outputs

⊛ 1-to-2 decoder



* في هذا الـ Decoder يكون الـ output تبعته واحد يعمل (أو على 1) والباقي لا يعمل (أو 0)
ويتم معرفة ذلك عن طريق قيمة الـ Decimal الـ input حيث يكون D0 على 1 إذا كانت قيمة ~~Decimal~~ الـ Decimal الـ input هي 0 والعكس لـ D1

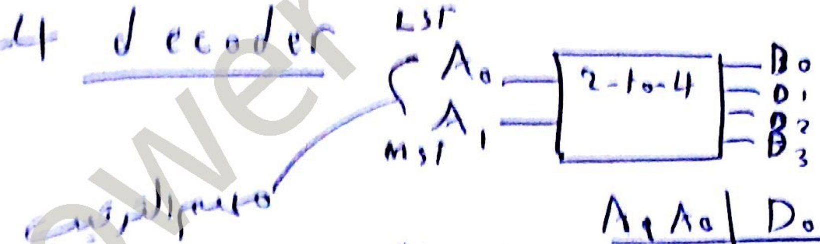
A	D_0	D_1
0	1	0
1	0	1

$D_0 = \bar{A}$

$D_1 = A$

---

* 2-to-4 decoder



قيمة الـ Decimal	A_1	A_0	D_0	D_1	D_2	D_3
0	0	0	1	0	0	0
1	0	1	0	1	0	0
2	1	0	0	0	1	0
3	1	1	0	0	0	1

⊛ نفس القصة عشان نعرف قيمة الـ output

$D_0 = \bar{A}_0 \bar{A}_1$ $D_2 = \bar{A}_0 A_1$

$D_1 = A_0 \bar{A}_1$ $D_3 = A_0 A_1$

3-to-8, 4-to-16] نفس الطريقة لمعرفة قيمة الـ output

⊛ لا يعمل إلا مخرج واحد فقط

* Building larger Decoder *

1 Decoder expansion

لبناء اي Decoder نتبع مايلي ← n-to-2^n

١- افرض $K = n$

٢- اذا "K" زوجي even اقسم "K" على 2

٣- استعمل بوابات And بعدد يساوي 2^k (عدد الـ output المطلوب) في المرحلة الاولى ثم بعدد يساوي $2^{\frac{k}{2}}$ في الثانية افرضي

وهكذا حتى تصبح K تساوي 1

أما إذا

٤- اذا "K" فردي odd اجعل $(K+1)$ واقسم على 2

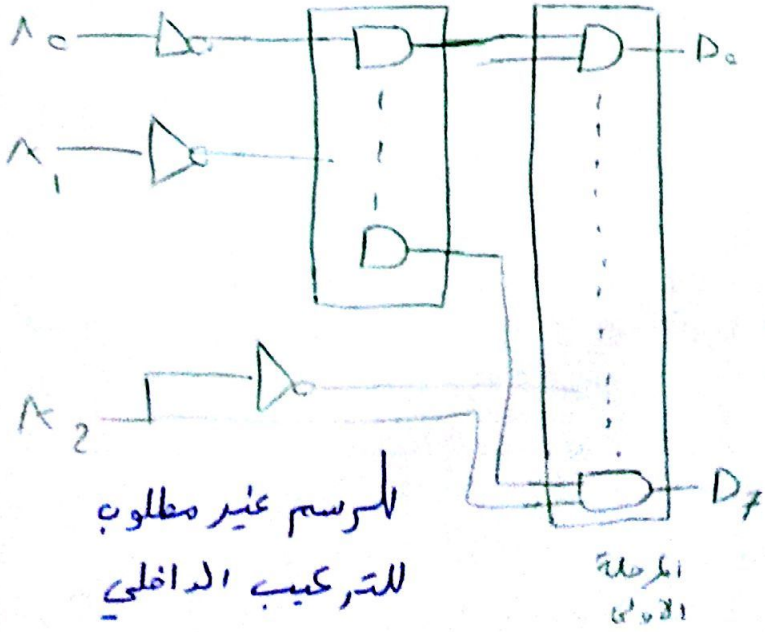
ثم استعمل بوابات And بعدد يساوي $2^{(K+1)}$ ثم

$2^{\frac{(K+1)}{2}}$ - - - -

٥- آخر خطوة عندما تصبح $K = 1$ استعمل

1-to-2 decoder ⇐

3-to-8



الرسم غير مطلوب للتركيب الداخلي

المرحلة الاولى

$K = n = 3$ *

$(K+1) = 4$ *

عدد And = 2^4 *

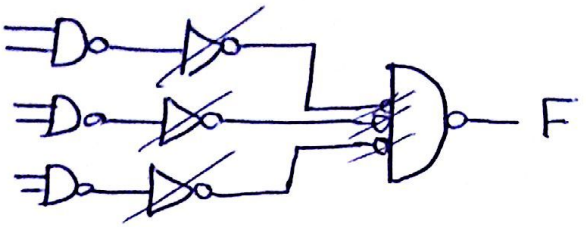
8 =

عدد $(And)_2 = \frac{4}{2} = 2$ *

عدد $(And)_3 = \frac{2}{2} = 1$ X *

لا نضع And وإنما نضع 1-to-2 decoder

then By T. Mapping / NAND



Ex: design a circuit that compares two (2-bit) numbers, A & B, and the output two bits; E_1 & E_0 such

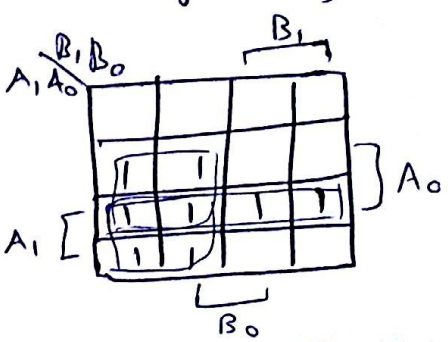
$$E_1 E_0 = \begin{cases} 0\,0, & \text{if } A = B \text{ \& Both are even} \\ 0\,1, & \text{if } A < B \\ 1\,0, & \text{if } A > B \\ 1\,1, & \text{if } A = B \text{ \& Both are odd} \end{cases}$$

الوصف ⊛ الدارة لها 4 inputs حيث كل رقم مكون من منزلتين ولها 2 outputs

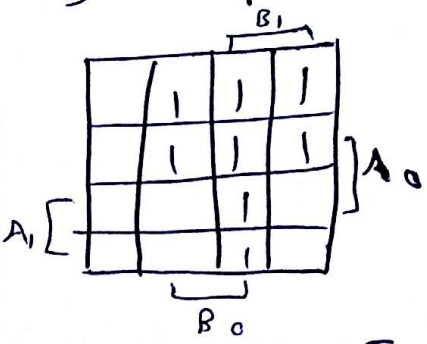
⊛ تقوم بالمقارنة بين الرقمين (قيمة Decimal) وتخرج الـ output بالترتيب المذكور

$A_1 A_0$	$B_1 B_0$	$E_1 E_0$
0 0	0 0	0 0
0 0	0 1	0 1
0 0	1 0	0 1
0 0	1 1	0 1
0 1	0 0	1 0
0 1	0 1	1 1
0 1	1 0	0 1
0 1	1 1	0 1
1 0	0 0	1 0
1 0	0 1	1 0
1 0	1 0	0 0
1 0	1 1	0 1
1 1	0 0	1 0
1 1	0 1	1 0
1 1	1 0	1 0
1 1	1 1	1 1

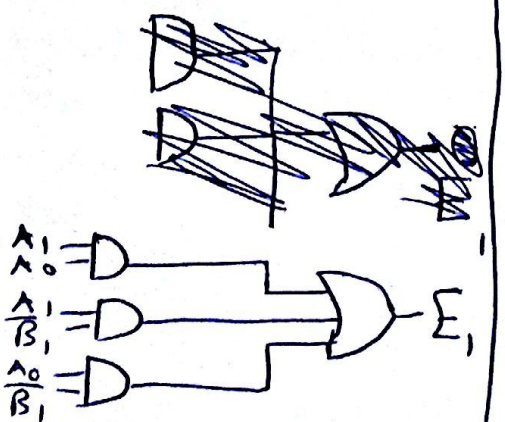
⊛ سوف نأخذ كل output على حدة ونرسم الـ K-map والـ Function والـ Diagram

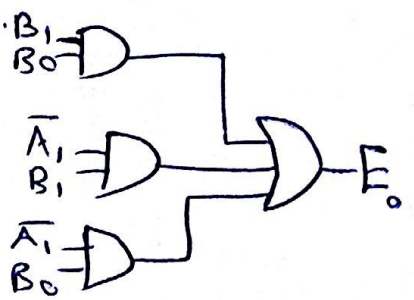


$$E_1 = A_1 A_0 + \bar{A}_1 \bar{B}_1 + A_0 \bar{B}_1$$



$$E_0 = B_1 B_0 + \bar{A}_1 B_1 + \bar{A}_1 B_0$$





Building large decoder from smaller ones:

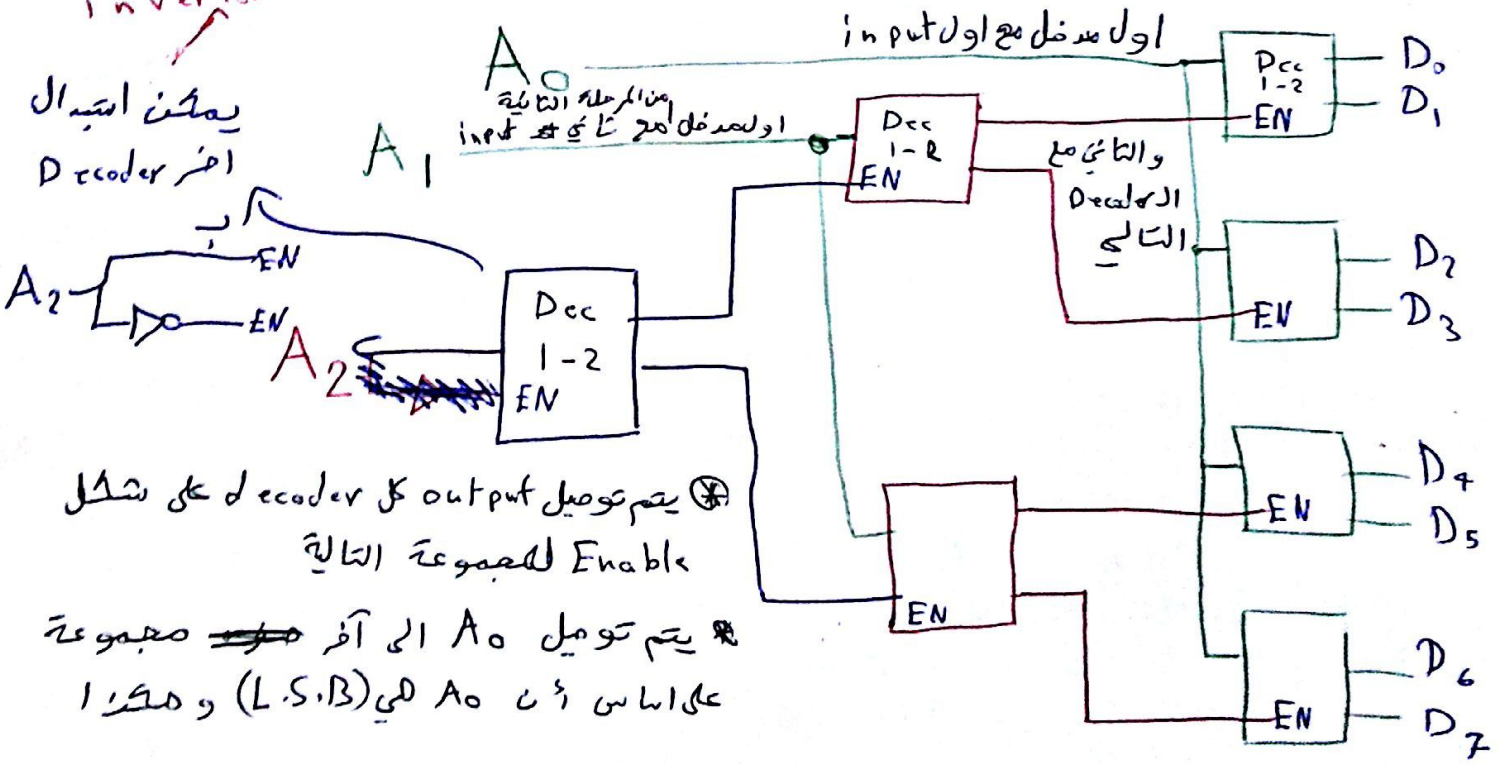
1- تصميم large decoder من نوع واحد فقط | الفرق عما سبق هو استخدام Enable

* Build 3 to 8 decoder using 1-to-2 decoder with enable

* بما انه نوع واحد فاننا نقسم عدد الـ outputs للـ Decoder الكبير على عدد output للصغير $(8/2=4)$ ← يعني 4 Decoder من نوع 1-to-2

* نرسم decoder بعدد الناتج و نبدأ من اليمين لليسار ومن ثم نعيد القسمة مرة اخرى $(4/2=2)$ ونعيد نفس الخطوة ، ونرسم الناتج على يسار المجموعة الاولى حتى يصبح الناتج صفر

اذا طلب استعمال inverter يمكن استبدال اخر Decoder



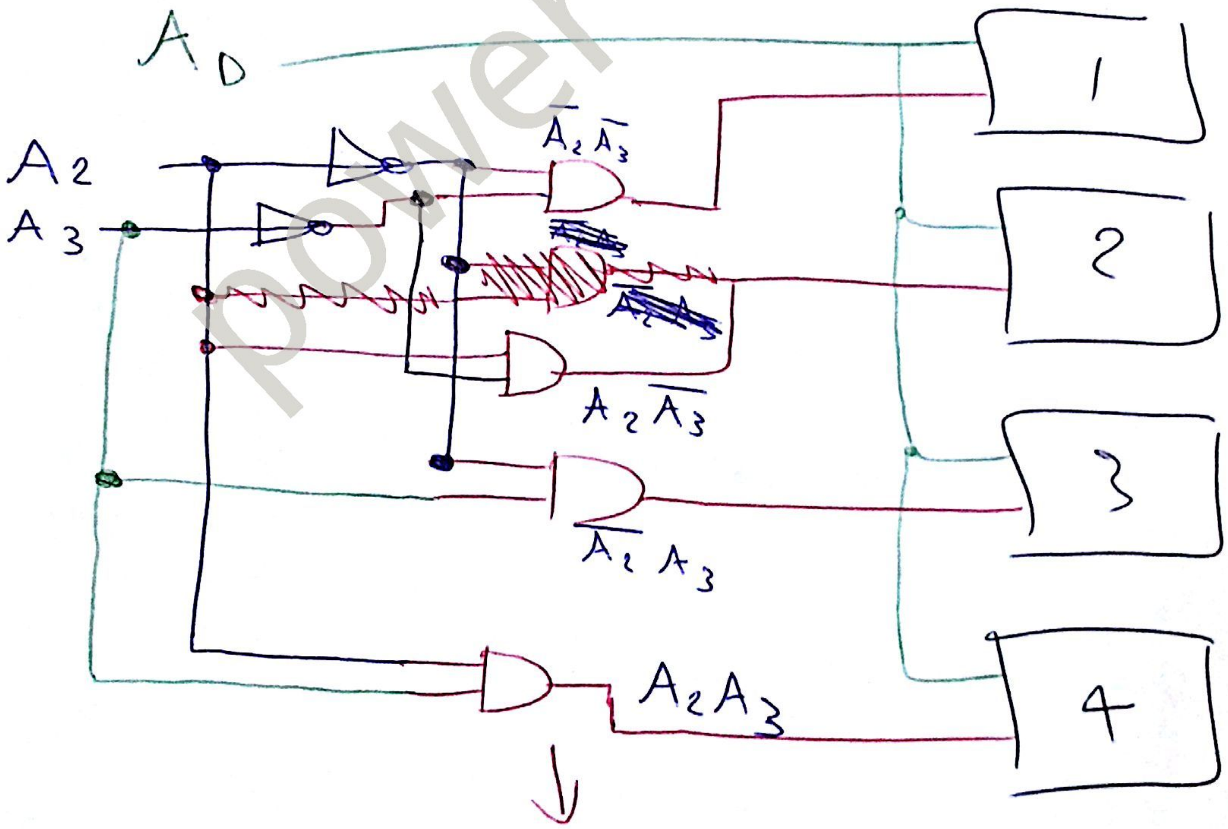
* يتم توصيل output كل decoder على شكل Enable للمجموعة التالية

* يتم توصيل A0 الى آخر مجموعة على اساس انه A0 هي (L.S.B) وهكذا

* اذا طلب في السؤال استعمال inverter فانا نستبدل
اخر Decoder بـ AND و inverter

Build 4-to-16 from 2-to-4 and inverters

$$\frac{16}{4} = 4 \Rightarrow \frac{4}{4} = 1$$



بوابات And بدل ال Decoder الاخير

2 - تصميم Decoder من اكثر من نوع

⊛ اذا طلب اكثر من نوع لا نستطيع استعمال القسمة لذلك نستخدم الـ truth table

⊛ لا تهم قيمة الـ output اكثر شي الـ input و يجب ان نجد علاقة بين الـ inputs

Build 4 to 16 decoder using

3 (a) 2-to-4 decoder with enable

1 (b) 3 to 8 decoder with enable

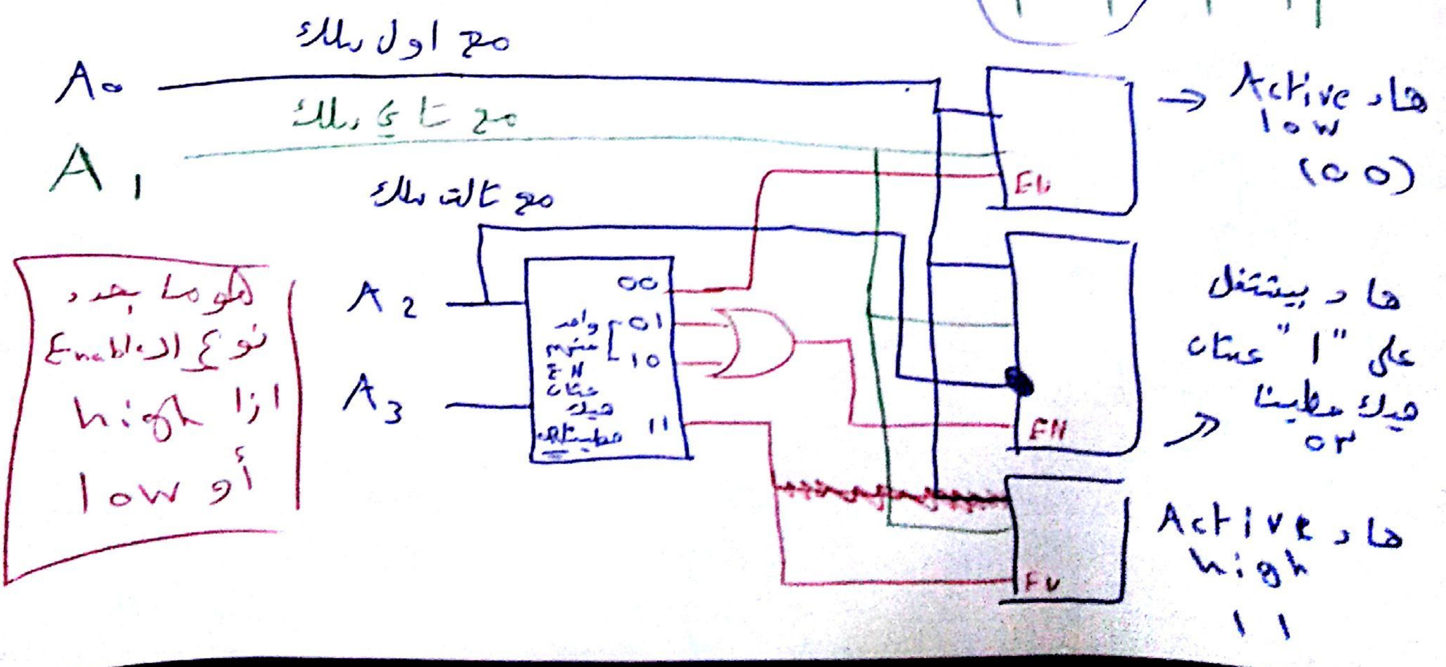
1 (c) or-Gate

العلاقات في اول ٤ حالات وآخر اربعة تكون $A_3 = A_2$ (يعني اول ٤ وآخر اربعة بكون A_1, A_3 هم الـ Enable الهم) يعني بنستخدم 2-to-4 في اول المجموعة وآخرها بنستعمل 2 منهم

في الوسط A_3A_2 بتتبدل من (01) الى (10) بس على ٤ مرات يعني بنقدر نستعمل 3-to-8

A_3	A_2	A_1	A_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

$A_3 = A_2$ (first four rows) — $A_3 = A_2$ (last four rows) — output نفس الترتيب



Implementing Function using decoder

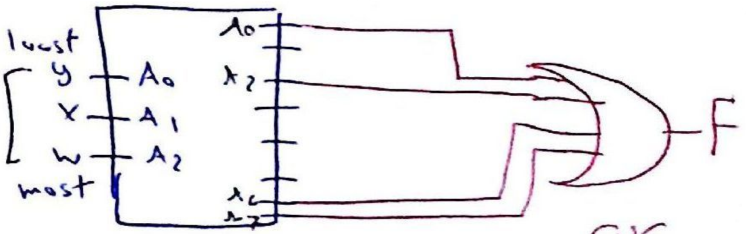
⊛ يمكن تبسيط اي معادلة باستخدام
1) n-to-2^n decoder
2) NAND gates / AND gate / or gate

⊛ تستخدم ال AND
و ال NAND
في حالة كان ال Enable
active low
يعني بيشتغل على قيمة 0

⊛ تستخدم ال OR
في حالة كان ال Enable
active high
يعني بيشتغل على 1

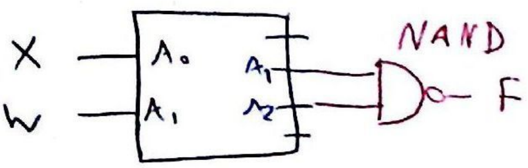
Ex: Implement F(w,x,y) = $\sum_m(0,2,6,7)$ using 3-to-8 decoder with (active high) output
(least: y)

LSB [redacted] A_0 تمثل ال
يتم ترتيب المتغيرات و يقابلها y



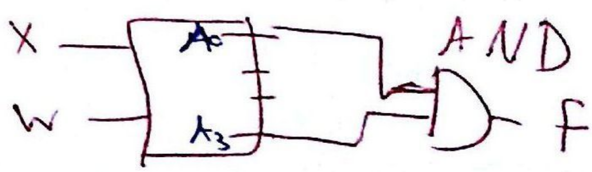
نأخذ فقط output التي تقابل ال minterm

Ex: Implement F(w,x) = $\sum_m(1,2)$ usin 2-to-4 with (active low)



اذا اخذنا minterm
نستعمل NAND

اذا طالبك $\prod_m(0,3)$



اذا اخذنا Maxterm
نستعمل AND

2 Endecoder

* تعمل عمل عكسي الـ Decoder حيث تحول البيانات اي حجم أقل

- يحتوي الـ Block على $\begin{cases} \text{input} : 2^n \\ \text{output} : n \end{cases}$ حيث n عدد المتغيرات

* انواع الـ Endecoder

1- الـ Normal Endecoder

* 2-to-1

يقوم على إخراج القيمة الـ Binary للـ input الذي يحوي قيمة 1 ولكن لا يخرج قيمة اذا كان اكثر من 1 input على 1

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

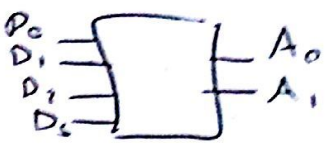
00 أو 11 لا يخرج قيمة لانه لا يستطيع تحديد اي من الـ input هو المطلوب

D_0 D_1

فتنتج القيمة بالـ Binary

---

* 4-to-2



* يجب ان يكون مدخل واحد فقط عليه 1
* الناتج هو رقم المدخل بالـ Binary

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

النوع الثاني : priority Endecoder

* يقوم هذا النوع على تحديد الـ (L.S.B) أو (M.S.B) في الـ inputs اذا ظهر عليه 1 (يعني حتى لو ظهر 1 على اكثر من مدخل إصنا بنحدد اذا بدنا ياه يكون المدخل الـ L.S.B أو M.S.B)

4 to 2 endecoder

مثال

(high priority)

* يعني اذ بياخد قيمة اعلى رقم لـ inputs يلي ظهر عليهم 1 والباقي بيحول الـ 1's لـ X's

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

(low priority)

* هي الحالة هي العكس تماما اللي قبل يعني بياخذ اقل رقم للأسلاك يلي ظهر عليها 1 ويحول الباقي لـ X's

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

ملاحظات (*)

* اذا كل الـ inputs كانو (0's) بيكون عنا اشي اضافي اسمو 1 Bit (Additional Bit) ويكون قيمتو 1 واسمو INV

low

D_1	D_0	A_1	INV	
0	0	x	1	بس اذا كانو (00)
1	0	1	0	
x	1	0	0	

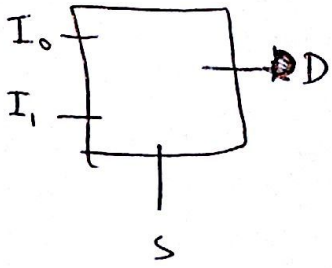
3 Multiplexars (MUX)

- يتميز هذا الـ Block بوجود جزء إضافي غير الـ input والـ output يسمى (selection line)
- وظيفة الـ selectionline هو اختيار ما هو الـ (input) الذي سوف يخرج الى الـ output
- يحتوى هذا الـ Block على

selection lines: n
inputs: 2^n
output: 1

نلاحظ ان الـ output دائماً واحد فقط

2-to-1 (MUX)



S	I_1	I_0	D
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

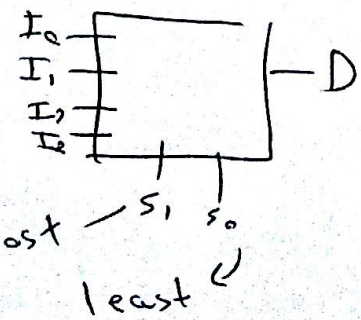
* لما تكون قيمة "S" هي "0" يخرج لدينا الـ output بقيمة I_0 بغض النظر عن قيمة I_1
* لما تكون قيمة "S" هي "1" يخرج لدينا الـ output I_1

S	D
0	I_0
1	I_1

* يعني قيمة "S" بال Binary هي يلي بتحدد اي input رح يطلع على الـ output

---

4-to-1



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

* to Build a (Mux) we need :-

1 - n to 2^n decoder
2 - 2^n (2input AND) ⟹ ⇐ عدد من 2^n
3 - 1 (2^n input or) ⇐ عدد من 1

2^n

---

* يتم ادخال الـ Selection الـ Decoder
* يتم ادخال الـ inputs على Mux الـ inputs الـ AND
* الـ output الخاص بـ AND يكون الـ Decoder
* يتم ادخال كل الـ output الـ AND على الـ OR gate

مثال

4-to-1 Mux

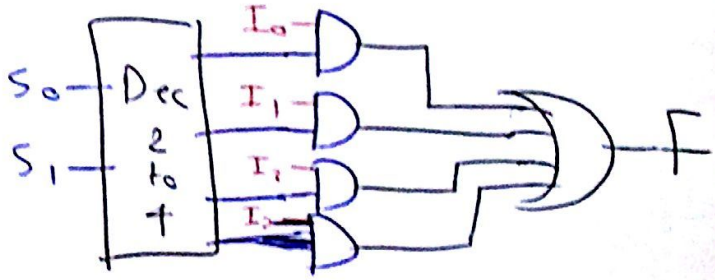
n = 2

يعني بدنا

2 to 4 decoder

(4) 2-input AND gate

(1) 4 input or gate



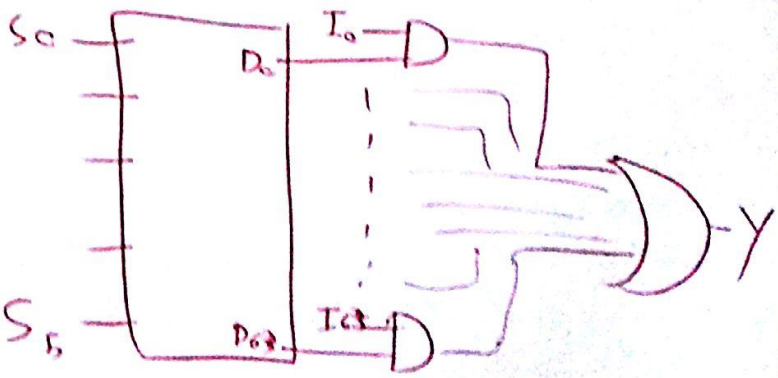
64-to-1 Mux

n = 6

6-to-64 decoder

(64) 2-input AND

(1) 64-input or



Building larger Mux From smaller one

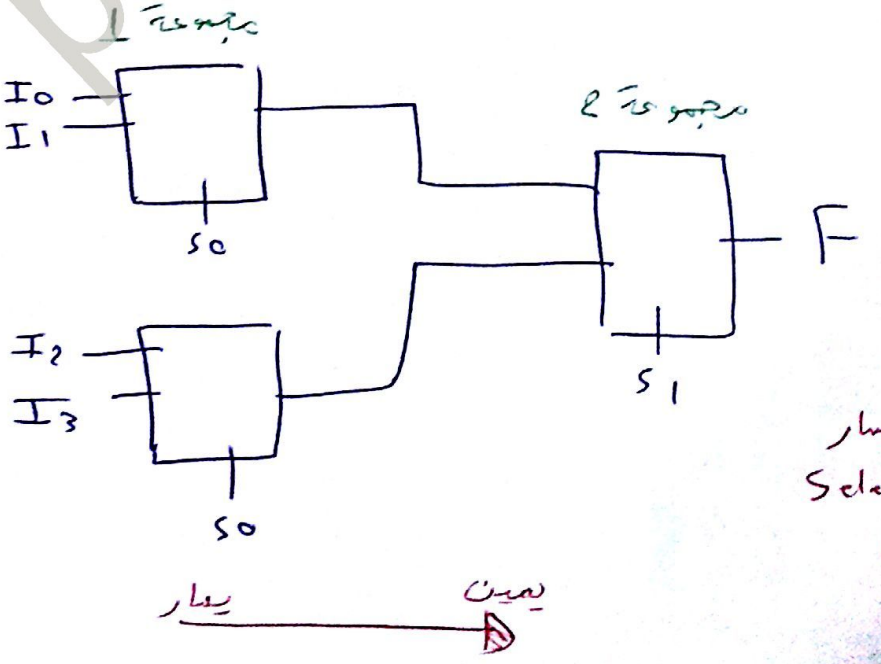
* عندنا نوعين من البناء

الاول Normal Mux كلهم نفس النوع

* بنقسم الـ input الـ Mux الكبير على input الـ Mux الصغير و يستمر ذلك حتى يصبح الجواب صفر و يترسم من اليسار لليمين

Build 4-to-1 Mux using 2-to-1 Muxels

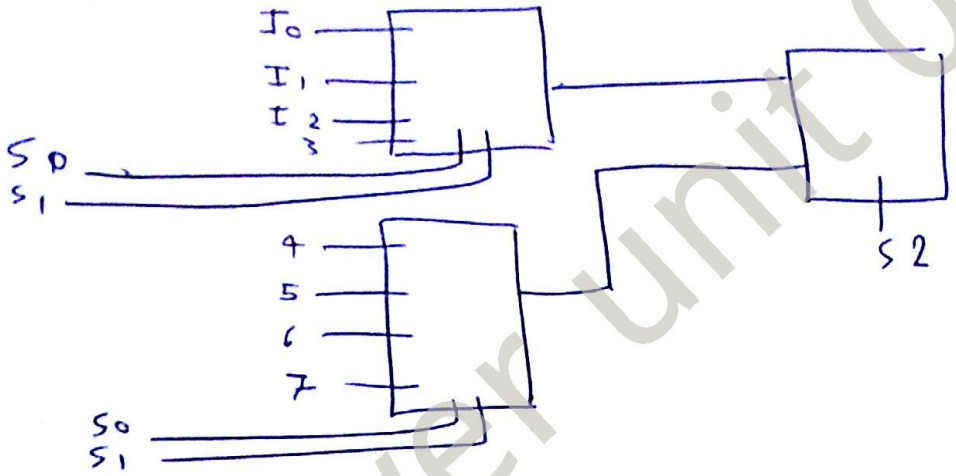
$$\frac{4}{2} = 2 \Rightarrow \frac{2}{2} = 1 \Rightarrow \frac{1}{2} = 0$$



* اول مجموعة من اليسار بتاخد اول Selection line كـ LSB

* اذا اما كانو من نفس النوع (انت وحظك تلاقي طريقة للحل)

Build 8-to-1 Mux by using 4-to-1 and 2-to-1



⊛ Implementig Function By Mux :-

⊛ يوجد طريقتان للتبسيط

1) First Appreach

⊛ اذا كان عدد minterms نفس عدد الـ input لـ Mux أو اقل منها

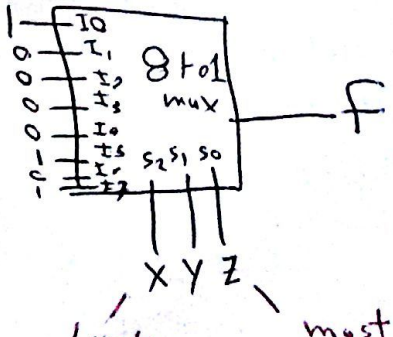
$$F(x,y,z) = \sum_m (0,5,7)$$

1) بتوصل الـ Variable مع الـ selection

2) كل minterm مع الـ input يلي بقابل قيمة حيث تنتج 1 عندما

نظر الى اكبر minterm قيمة كتابة اعلى قيمة للـ input يلي مسموح

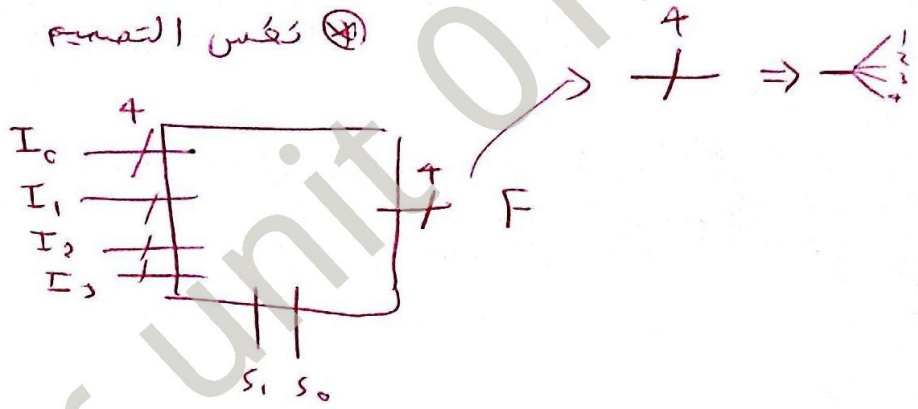
مثال



4-to-1 line Quad mux

* هاد بيكون كل input (I) فيه أربع
مداخل داخلية و الـ output فيه 4 كمان

* نعكس التصميم



⊛ اذا كان عدد الـ minterm اكبر من عدد inputs الـ Mux

(Second Approach)

مثال

$$F(w,x,y) = \sum_m (3,5,6) \text{ By } 4\text{-to-}1$$

اكبر من عدد ال input

~~⊛ في هاي الحالة بنقدر نستخدم~~

① نأخذ المتغيرات ما عدا الـ (LSB) يعني بناخذ (w,x) ونضعهم selection line في Mux

② يتبقى (L.S.B) يلي هو (y) نبسط الـ Function بالنسبة الـ (بـ لا y)

y=0	m_0	m_1	m_2	(m_3)
y=1	m_4	(m_5)	(m_6)	m_7
F	0	y	y	$\bar{y}$

⊛ نضع دوائر على الـ minterm المطلوبة

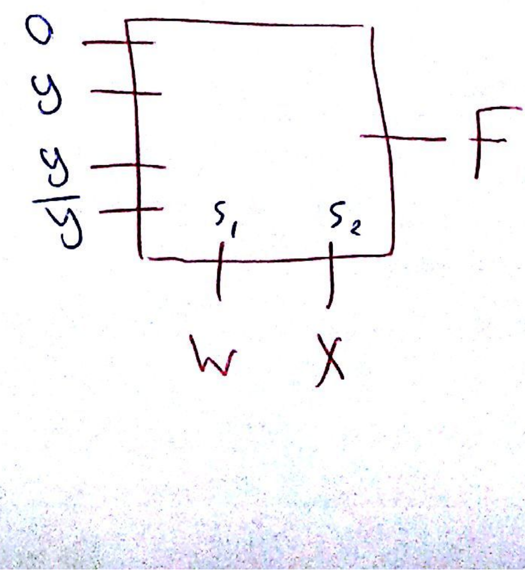
⊛ اذا لم يكن هناك minterm نضع الجواب 0

⊛ اذا كان هناك دائرة على احد الـ minterm يلي فوق بعضهم نأخذ قيمته يلي بتقابله بال y

قيمة F

⊛ اذا كان هناك دائرتين فوق بعض بتكون القيمة 1

⊛ تصبح قيم y هي قيم الـ input

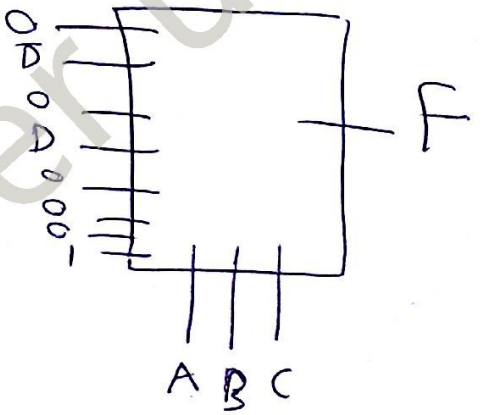


مثال

$$F(\underbrace{A,B,C}_{S},D) = \Sigma_m(1,7,11,15)$$

using 8-to-1

D = 0	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
D = 1	8	m_9	m_{10}	m_{11}	m_{12}	m_{13}	m_{14}	m_{15}
F	0	$\bar{D}$	0	D	0	0	0	1



4) De-Multiplexers (DeMux):

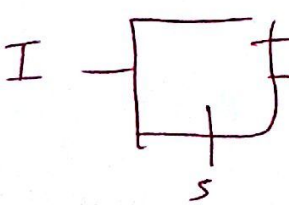
في عكس الـ Mux تصوير Selection lines

output : 2^n

input : 1

* تقوم بإخراج قيمة input على output ولكن بطريقة معينة

1-to-2 Demux



S	I	D_1 D_0
0	0	0 0
0	1	0 1
1	0	0 0
1	1	1 0

- ننظر الى قيمة الـ Binary الـ Selection حتى نحدد على اي output
- نخرج قيمة الـ Input كما هي بدون تغيير

يخرج I على نفس قيمة S واما الـ D التالية تبقى Q

ملاحظة ال DeMux

هو Decoder مع enable

يعني بشكل عام بنقدر نبني اي Demux من

enable + Decoder

* بيكون ال input لل Dmux هو هو مع ال enable

لل Decoder

* بتكون ال selection هو هو مع ال inputs لل

Decoder

1-to-m Demux — [n-to-m decoder, enable]

---

1 to 4 DMux By 2 to 4 decoder