POWER UNIT

MOHAMED ABED-ALMAJEED

LOGIC

KHALID AL-NASER

POWER UNIT-JU

http : // POWER UNIT-JU.COM

Logic and Computer Design Fundamentals

Chapter 1 – Digital Systems and Information

Charles Kime & Thomas Kaminski

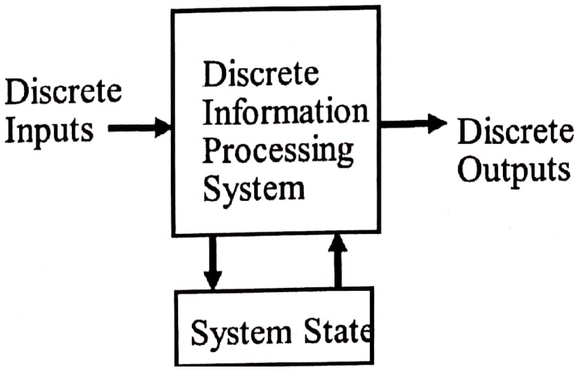
(Hyperlinks are active in View Show mode)

Overview

- Digital Systems, Computers, and Beyond
- Information Representation
- Number Systems [binary, octal and hexadecimal]
- Base Conversion
- Decimal Codes [BCD (binary coded decimal)]
- Alphanumeric Codes
- Parity Bit
- Gray Codes

DIGITAL & COMPUTER SYSTEMS - Digital System

- Takes a set of *discrete* information *inputs* and discrete internal information (*system state*) and generates a set of *discrete* information *outputs*.
- Digits (Latin word for fingers) : Discrete numeric elements
- Logic : Circuits that operate on a set of two elements with values 0 (False), 1 (True)
- ***Computers are digital logic circuits***



Types of Digital Systems

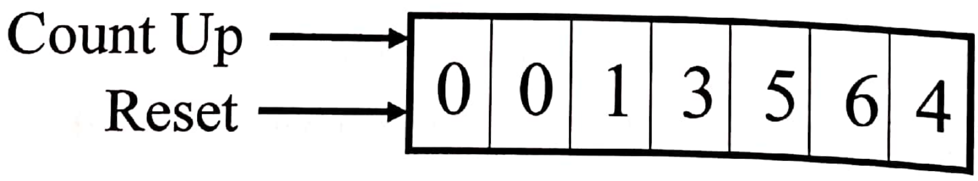
- No state present
  - Combinational Logic System
  - Output = Function(Input)
- State present
  - *Synchronous* Sequential System: State updated at discrete times
  - *Asynchronous* Sequential System: State updated at any time
  - State = Function (State, Input)
  - Output = Function (State) or Function (State, Input)

Moore

Mealy

Digital System Example

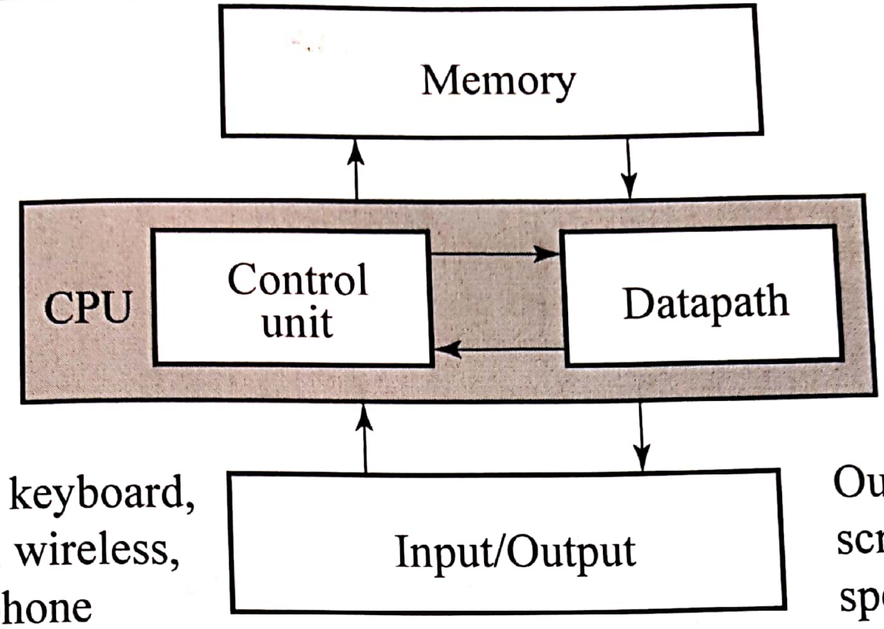
A Digital Counter (e. g., odometer):



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

Synchronous or Asynchronous?

Digital Computer Example



Memory

CPU

Control unit

Datapath

Input/Output

Inputs: keyboard, mouse, wireless, microphone

Outputs: LCD screen, wireless, speakers

Synchronous or Asynchronous?

And Beyond – Embedded Systems

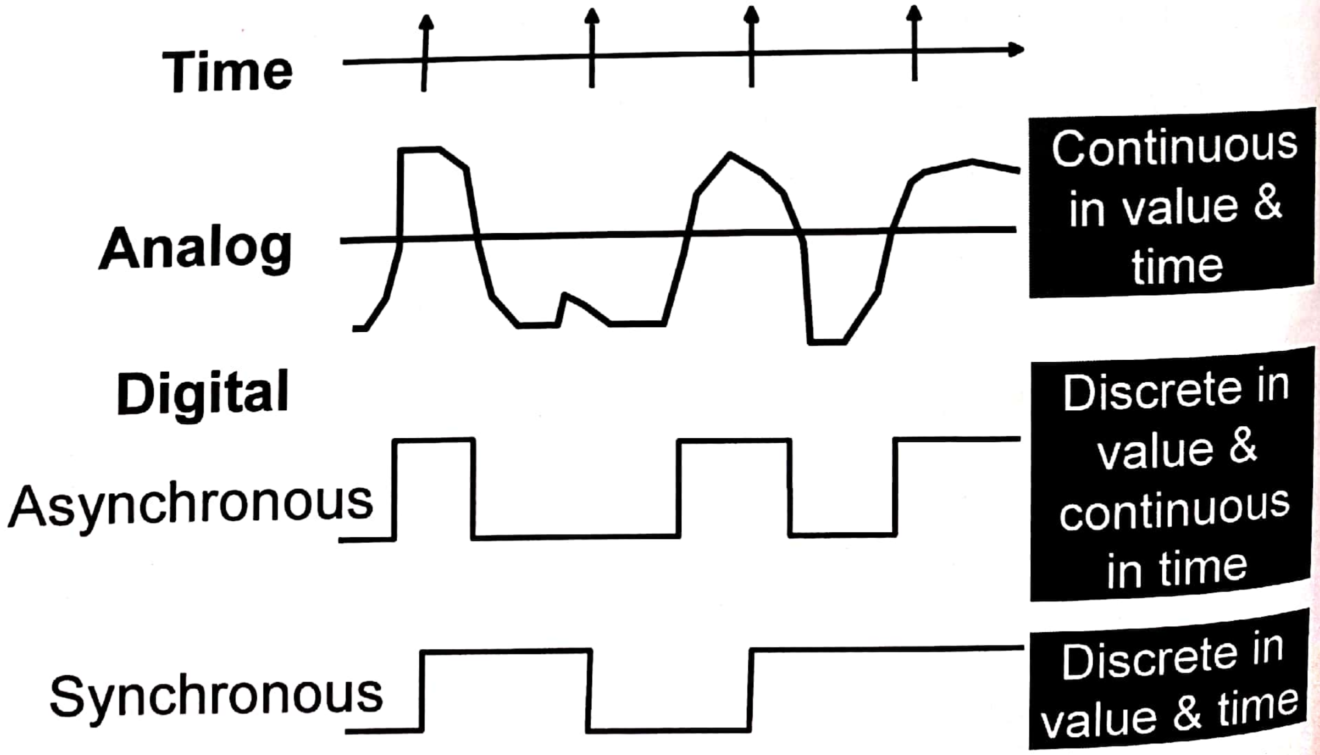
- Computers as integral parts of other products
- Examples of embedded computers
  - Microcomputers
  - Microcontrollers
  - Digital signal processors
- Examples of embedded systems applications

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

Information Representation - Signals

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

Signal Examples Over Time



Time

Analog

Continuous in value & time

Digital

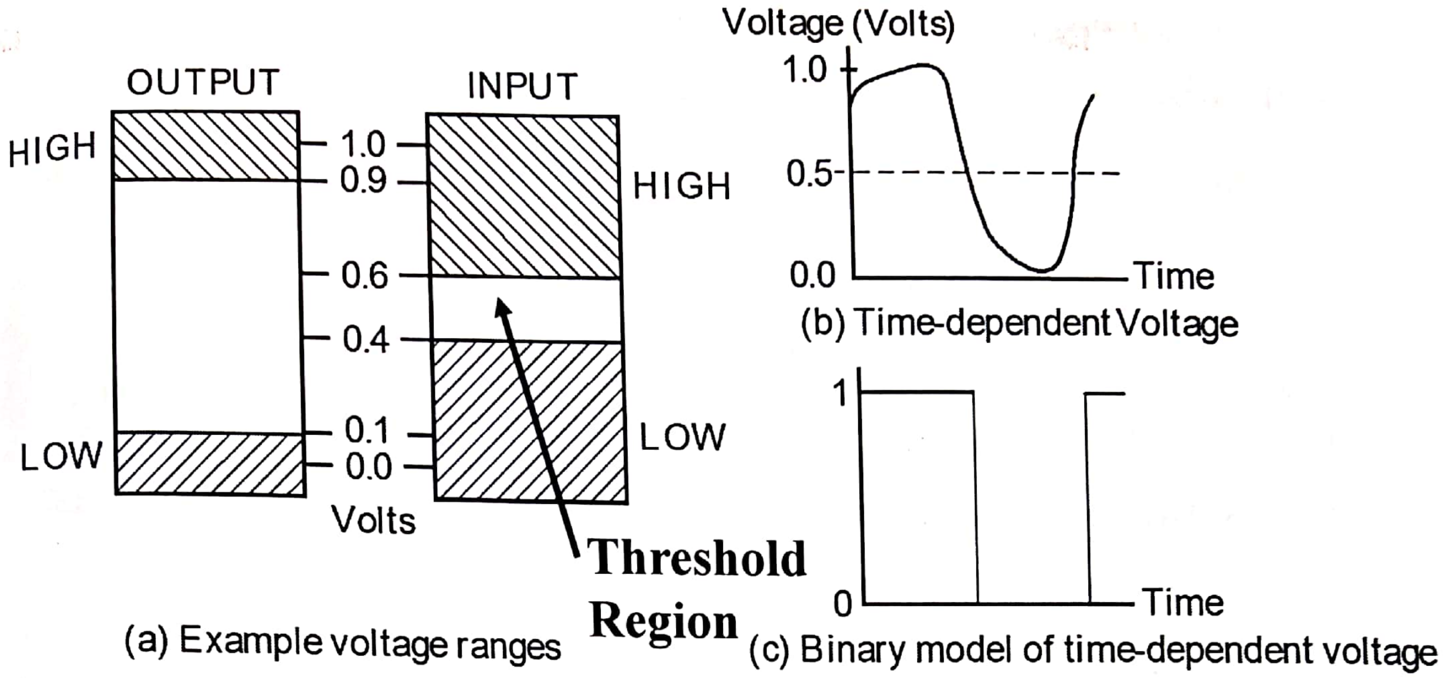
Asynchronous

Discrete in value & continuous in time

Synchronous

Discrete in value & time

Signal Example – Physical Quantity: Voltage



(a) Example voltage ranges

(b) Time-dependent Voltage

(c) Binary model of time-dependent voltage

Binary Values: Other Physical Quantities

- What are other physical quantities represent 0 and 1?
  - CPU → Voltage
  - Disk → Magnetic Field Direction
  - CD → Surface Pits/Light
  - Dynamic RAM → Electrical Charge stored in capacitors

NUMBER SYSTEMS – Representation

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

$$A_{n-1}A_{n-2}\ \cdots\ A_1A_0\ .\ A_{-1}\ A_{-2}\ \cdots\ A_{-m+1}\ A_{-m}$$

in which $0 \leq A_i < r$ and . is the ***radix point***

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

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

Number Systems – Examples

		General	Decimal	Binary
Radix (Base)		r	10	2
Digits		0 => r - 1	0 => 9	0 => 1
	0	r^0	1	1
	1	r^1	10	2
	2	r^2	100	4
	3	r^3	1000	8
Powers of	4	r^4	10,000	16
Radix	5	r^5	100,000	32
	-1	r^{-1}	0.1	0.5
	-2	r^{-2}	0.01	0.25
	-3	r^{-3}	0.001	0.125
	-4	r^{-4}	0.0001	0.0625
	-5	r^{-5}	0.00001	0.03125

Example

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

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

BASE CONVERSION - Positive Powers of 2

- Useful for Base Conversion

Exponent	Value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

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

Special Powers of 2

- 2^{10} (1024) is Kilo, denoted "K"
- 2^{20} (1,048,576) is Mega, denoted "M"
- 2^{30} (1,073, 741,824)is Giga, denoted "G"
- 2^{40} (1,099,511,627,776) is Tera, denoted "T"

Commonly Occurring Bases

النظام
الثنائي

النظام
العشري

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

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

Binary System

- r = 2
- Digits = {0, 1}
- Every binary digit is called a bit
- When a bit is equal to zero, it does not contribute to the value of the number
- Example:
  - $(10011.101)_2 = (1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) + (1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3})$
  - $(10011.101)_2 = (16 + 2 + 1) + \left(\frac{1}{2} + \frac{1}{8}\right) = (19.625)_{10}$

Octal System

- r = 8
- Digits = {0, 1, 2, 3, 4, 5, 6, 7}
- Every digit is represented by 3-bits → More compact than binary
- Example:
  - $(127.4)_8 = (1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0) + (4 \times 8^{-1})$
  - $(127.4)_8 = (64 + 16 + 7) + \left(\frac{1}{2}\right) = (87.5)_{10}$

Hexadecimal System

- r = 16
- Digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
- Every digit is represented by 4-bits
- Example:
  - $(B65F)_{16} = (11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0)$
  - $(B65F)_{16} = (46687)_{10}$

Numbers in Different Bases

- **Good idea to memorize!**

Decimal (Base 10)	Binary (Base 2)	Octal (Base 8)	Hexadecimal (Base 16)
00	00000	00	00
01	00001	01	01
02	00010	02	02
03	00011	03	03
04	00100	04	04
05	00101	05	05
06	00110	06	06
07	00111	07	07
08	01000	10	08
09	01001	11	09
10	01010	12	0A
11	01011	13	0B
12	01100	14	0C
13	01101	15	0D
14	01110	16	0E
15	01111	17	0F
16	10000	20	10

Converting from any Base (r) to Decimal

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

- **Example: Convert 11010_2 to N_{10}:**

Conversion from Decimal to Base (r)

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

Conversion Details

- ***To Convert the Integral Part:***
  - Repeatedly *divide* the number by the new radix and save the *remainders* until *the quotient is zero*
  - The digits for the new radix are the remainders in *reverse order* of their computation
  - If the new radix is > 10, then convert all remainders > 10 to digits A, B, …

- ***To Convert the Fractional Part:***
  - Repeatedly *multiply* the fraction by the new radix and save the *integer digits* of the results until the *fraction is zero or your reached the required number of fractional digits*
  - The digits for the new radix are the integer digits *in order* of their computation
  - If the new radix is > 10, then convert all integers > 10 to digits A, B, …

Example: Convert 46.6875_{10} To Base 2

- Convert 46 to Base 2:

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

Division	Quotient	Remainder	
46/2	23	0 ↑	LSD
23/2	11	1	
11/2	5	1	
5/2	2	1	
2/2	1	0	
1/2	0	1	MSD

- Convert 0.6875 to Base 2:

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

Multiplication	Answer	
0.6875*2	1.375	MSD
0.375*2	0.75	
0.75*2	1.5	
0.5*2	1.0 ↓	LSD

- Join the results together with the radix point:

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

Example: Convert 153.513_{10} To Base 8

- Convert 153 to Base 8:

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

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

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

    $(0.513)_{10} = (0.406)_8$
  - Round:

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

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

- Join the results together with the radix point:

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

Example: Convert 423_{10} To Base 16

Division	Quotient	Remainder	
423/16	26	7 ↑	LSD
26/16	1	10	
1/16	0	1	MSD

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

Converting Decimal to Binary: Alternative Method

- Subtract the largest power of 2 that gives a positive remainder and record the power
- Repeat, subtracting from the prior remainder and recording the power, until the remainder is zero
- Place 1's in the positions in the binary result corresponding to the powers recorded; in all other positions place 0's

طريقة السلة

8 16 --
1 2 4

+
للعشري

0.125 --
0.5 0.25

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

- Convert 46 to Base 2:

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

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

- Convert 0.6875 to Base 2:

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

Subtract	Remainder	Power
0.6875-0.5	0.1875	-1
0.1875-0.125	0.0625	-3
0.0625-0.0625	0	-4

- Join the results together with the radix point:

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

- Easier way to do it:

Power	6	5	4	3	2	1	0	.	-1	-2	-3	-4
	0	1	0	1	1	1	0	.	1	0	1	1

Additional Issue - Fractional Part

- Note that in this conversion, the fractional part can become 0 as a result of the repeated multiplications

- In general, it may take many bits to get this to happen or it may never happen

- Example Problem: Convert 0.65_{10} to N_2
  - 0.65 = 0.1010011001001 …
  - The fractional part begins repeating every 4 steps yielding repeating 1001 forever!

- **Solution: Specify number of bits to right of radix point and *__round__* or *__truncate__* to this number**

Checking the Conversion

- To convert back, sum the digits times their respective powers of r

- From the prior conversion of 46.6875_{10}

$$
\begin{aligned}
101110_2 &= 1\cdot 32 + 0\cdot 16 + 1\cdot 8 + 1\cdot 4 + 1\cdot 2 + 0\cdot 1 \\
&= 32 + 8 + 4 + 2 \\
&= 46
\end{aligned}
$$

$$
\begin{aligned}
0.1011_2 &= 1/2 + 1/8 + 1/16 \\
&= 0.5000 + 0.1250 + 0.0625 \\
&= 0.6875
\end{aligned}
$$

Octal (Hexadecimal) to Binary and Back: Method1

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

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

* من اي نظام للعشري بنضرب ---

* من عشري لـ اي نظام بنقسم

+ و العشرية ـــ بنضرب بـ الاساس الجديد

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

- Octal (Hexadecimal) to Binary:
  - ***Restate*** the octal (hexadecimal) as three (four) binary digits starting at the radix point and going both ways

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

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

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

Examples

- $(673.12)_8 = (110\ 111\ 011\ .\ 001\ 010)_2$
- $(3A6.C)_{16} = (0011\ 1010\ 0110\ .\ 1100)_2$
- $(10110001101011.111100000 1)_2 = (\ ?\ )_8$

  $\mathbf{(10/110/001/101/011.111/100/000/1)_2 = (26153.7404\ )_8}$

- $(10110001101011.111100000 1)_2 = (\ ?\ )_{16}$

  $\mathbf{(10/1100/0110/1011.1111/0000/01)_2 = (2C6B.F04\ )_{16}}$

Octal to Hexadecimal via Binary

- Convert octal to binary
- Use groups of <u>four bits</u> and convert to hexadecimal digits
- Example: Octal to Binary to Hexadecimal

$$\begin{gathered}(635.177)_8\\ \downarrow\\ (110\ 011\ 101\ .\ 001\ 111\ 111)_2\\ \downarrow\\ (1/1001/1101\ .\ 0011/1111/1)_2\\ \downarrow\\ (19D.3F8)_{16}\end{gathered}$$

One last Conversion Example

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

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

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

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

$$r^2 + 2r - 63 = 0$$

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

$$r = 7$$

Binary Numbers and Binary Coding

- Flexibility of representation
  - Within constraints below, can assign any binary combination (called a <u>*code word*</u>) to any data as long as data is uniquely encoded
- Information Types
  - *Numeric*
    - Must represent range of data needed
    - Very desirable to represent data such that simple, straightforward computation for common arithmetic operations permitted
    - Tight relation to binary numbers
  - *Non-numeric*
    - Greater flexibility since arithmetic operations not applied
    - Not tied to binary numbers

Non-numeric Binary Codes

- Given *n* binary digits (called bits), a binary code is a mapping from a set of represented elements to a subset of the 2^n binary numbers.
- Example: A binary code for the seven colors of the rainbow
- Code 100 is not used

استخدمنا ٣ خانات
يكفي لكل لون خيار

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

مش شرط
بالترتيب

(digits)
خانه
Ex. 25 → بحتاج 5 bits

$2^5 = 32 > 25$ ✓

Number of Bits Required

No. of digits
× Base(2) or (8) or ...

* عدد الخيارات اللي
بقدر امثلها

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

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

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

بقربه للاكبر لما

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

$$M = 10$$

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

Number of Elements Represented

- Given n digits in radix r, there are $\underline{r^n}$ distinct elements that can be represented.
- But, you can represent m elements, $\underline{m < r^n}$
- Examples:
  - You can represent 4 elements in radix $r = 2$ with $n = 2$ digits: (00, 01, 10, 11).
  - You can represent 4 elements in radix $r = 2$ with $n = 4$ digits: (0001, 0010, 0100, 1000).
  - This second code is called a "one hot" code.

Ex.

0	0001
1	0010
2	0100
7	1000

ميزته بس Bit واحد اللي يكون رقمه (1) ويختلف موقعه كل مرة .

DECIMAL CODES - Binary Codes for Decimal Digits

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

Decimal	8, 4, 2, 1	Excess 3	8, 4, -2, -1	Gray
0	0000	0011	0000	0000
1	0001	0100	0111	0001
2	0010	0101	0110	0011
3	0011	0110	0101	0010
4	0100	0111	0100	0110
5	0101	1000	1011	0111
6	0110	1001	1010	0101
7	0111	1010	1001	0100
8	1000	1011	1000	1100
9	1001	1100	1111	1101

بس واحد بختلف كلامره

Binary Coded Decimal (BCD)

- Numeric code
- The BCD code is the 8, 4, 2, 1 code
- 8, 4, 2, and 1 are weights → BCD is a *weighted* code
- This code is the simplest, most intuitive binary code for decimal digits and uses the same powers of 2 as a binary number, ***but only encodes the first ten values from 0 to 9***
- Example: 1001 (9) = 1000 (8) + 0001 (1)
- How many "invalid" code words are there?
  - Answer: 6
- What are the "invalid" code words? → مستحيل نشوفهم بال BCD
  - Answer: 1010, 1011, 1100, 1101, 1110, 1111

Ex. $(125)_{10}$

(0001 0010 0101) 8,4,2,1

الوزن

8 4 2 1

Binary نفس

Warning: Conversion or Coding?

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

- $\mathbf{13_{10} = 1101_2}$ **(This is conversion)** → تحويل

- **13 ⇔ 0001|0011 (This is coding)**

Excess 3 Code and 8, 4, –2, –1 Code

- What interesting property is common to these two codes?
  - Answer: Both codes have the property that the codes for 0 and 9, 1 and 8, etc. can be obtained from each other by replacing the 0's with the 1's and vice-versa. Such a code is sometimes called a *complement code.*

Decimal	Excess 3	8, 4, –2, –1
0	0011	0000
1	0100	0111
2	0101	0110
3	0110	0101
4	0111	0100
5	1000	1011
6	1001	1010
7	1010	1001
8	1011	1000
9	1100	1111

1+3=4

2+3=5

Ex: $(125)_{10} =$ 5+8=8

(0100 0101 1000) Excess 3

Ex: 8 4 -2 -1

_ _ _ _

$(125)_{10}$: (0111 0110 1011) 8,4,-2,-1

1 2 5

$(\ )_{BCD} \rightarrow (\ )_8$
نحوله بعدين عن ١٠ ← 8

$*(\ )_{Excess} \rightarrow (\ )_{10}$
نطرح 3 من Excess
بعدين نحول الى ١٠

$(\ )_2 \rightarrow (\ )_{BCD}$
4
نحوله الى ١٠
بعدين نحول لـ BCD

ALPHANUMERIC CODES - ASCII Character Codes

- Non-numeric code
- ASCII stands for American Standard Code for Information Interchange (Refer to Table 1-5 in the text)
- This code is a popular code used to represent information sent as character-based data. It uses 7-bits (i.e. 128 characters) to represent:
  - 94 Graphic printing characters
  - 34 Non-printing characters

ما بعبر اشي
اذا كبسناهم بالكيبورد

Scanned by CamScanner

ASCII Code Table

* الفرق بين حرف Capital و Small نقط bit 5

A (0100 0001)

Ex: (41)16 → (65)10 ←

(0110 0001)

Ex 128

نحتاج ال 7bits لونه

$2^7 = 128$

Least Significant

ASCII Code Chart

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

← معلومه تميز طالب بالامتحان

MSD LSD

(0 - - - - - - -)

نستوعب من 0-7 ←

نستوعب من 0-f

Ex: W → (57)16 → (01010111)2 ← هيك تدخل النظام

ASCII Character Codes

- Graphic printing characters
  - 26 upper case letters (A-Z)
  - 26 lower case letters (a-z)
  - 10 numerals (0-9)
  - 32 special characters (e.g. %, @, $)
- Non-printing characters
  - Format effectors: used for text format (e.g. BS = Backspace, CR = carriage return)
  - Information separators: used to separate the data into paragraphs and pages (e.g. RS = record separator, FS = file separator)
  - Communication control characters (e.g. STX and ETX start and end text areas).

ASCII Properties

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

UNICODE

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

PARITY BIT Error-Detection Codes

عشان اضمن انه يوصل نفس ما نكتب

- Non-numeric

M 101010 → M'

← عدد الـ 1 يتم الاتفاق عليها بين المرسل والمستقبل .

- *Redundancy* (e.g. extra information), in the form of extra bits, can be incorporated into binary code words to detect and correct errors

1010111 1 ← Parity bit (عشان بدي الزوجي يزيد 1)

Data | C → even Parity

إذا كان عدد الـ 1 فردي ← نحط 1 = Parity

" " " زوجيا ← 0 = Parity

- A simple form of redundancy is ***parity***, an extra bit appended onto the code word to make the number of 1's odd or even. Parity can detect all single-bit errors and some multiple-bit errors

عدد الـ 1 بزوجي

- A code word has ***even parity*** if the number of 1's in the code word is even

عدد الـ 1 فردي

- A code word has ***odd parity*** if the number of 1's in the code word is odd

عكس الـ even Parity

4-Bit Parity Code Example

- Fill in the even and odd parity bits:

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

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

GRAY CODE (1)

- Non-numeric code
- For original binary codes (0 through 2^n -1):
  - Copy the leftmost bit as it is
  - Replace each of the remaining bits with the even parity of the bit of the number and the bit to its left
- What special property does the Gray code have in relation to adjacent decimal digits?
  - As we "counts" up or down in decimal, the code word for the Gray code changes in only one bit position including 15 to 0.

Decimal	Binary	Gray
00	0000	0000
01	0001	0001
02	0010	0011
03	0011	0010
04	0100	0110
05	0101	0111
06	0110	0101
07	0111	0100
08	1000	1100
09	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

لـ باستخدام خوارزمية معينة
لكي نصبح بين كل رقم وما قبله وما بعده
فرق واحد فقط (Bit)

GRAY CODE (2)

- For a counting sequence of ***n*** binary code words (***n*** must be even)
  - Replace each of the first ***n/2*** numbers with a code consisting of ***0*** followed by the even parity of each bit of the binary code word and the bit to its left
  - Copy the sequence of numbers formed and copy it in ***reverse order*** with the leftmost bit replaced by 1.

Decimal	BCD	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	1110
6	0110	1010
7	0111	1011
8	1000	1001
9	1001	1000

غيرمطلوب ايجاد ال gray

Logic and Computer Design Fundamentals

Chapter 2 – Combinational Logic Circuits

Part 1 – Gate Circuits and Boolean Equations

Charles Kime & Thomas Kaminski

(Hyperlinks are active in View Show mode)

Combinational Logic Circuits

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

Overview

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

Binary Logic and Gates

- ***Binary variables*** take on one of two values
- ***Logical operators*** operate on binary values and binary variables → العمليات بين ال Variable
- Basic logical operators are the logic functions ***AND***, ***OR*** and ***NOT***
- ***Logic gates*** implement logic functions
- ***Boolean Algebra***: a useful mathematical system for specifying and transforming logic functions
- We study Boolean algebra as a foundation for designing and analyzing digital systems!

Binary Variables → بس ١/٥

- Recall that the two binary values have different names:
  - True/False ١ ٥
  - On/Off ١ ٥
  - Yes/No ١ ٥
  - 1/0
- We use 1 and 0 to denote the two values
- Variable identifier examples:
  - A, B, y, z, or X_1 for now
  - RESET, START_IT, or ADD1 later

Logical Operations

- The three basic logical operations are:
  - AND
  - OR
  - NOT

ما منستخدمها

- AND is denoted by a dot (·) or $(\overset{\times}{\wedge})$

ما منستخدمها

- OR is denoted by a plus (+) or $(\overset{\times}{\vee})$
- NOT is denoted by an over-bar (‾), a single quote mark (') after, or (~) before the variable

Ex: $\bar{A}$ / A' / $\sim A$

Notation Examples

اذا شفتها هيك وعارف انها Variables
معناها اعرف انها X . y.

- Examples:
  - $Z = X \cdot Y = XY = X \wedge Y$: is read "Z is equal to X AND Y"
    - Z = 1 if and only if X = 1 and Y = 1; otherwise, Z = 0

      $Z(X,Y) = XY \rightarrow Z = X \cdot Y$
  - $Z = X + Y = X \vee Y$: is read "Z is equal to X OR Y"
    - Z = 1 if (only X = 1) or if (only Y = 1) or if (X =1 and Y = 1)
  - $Z = \bar{X} = X' = \sim X$: is read "Z is equal to NOT X"
    - Z = 1 if X = 0; otherwise, Z = 0
- Notice the difference between arithmetic addition and logical OR:
  - The statement:

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

    is not the same as

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

Operator Definitions

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

AND
0 . 0 = 0
0 . 1 = 0
1 . 0 = 0
1 . 1 = 1

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

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

Truth Tables

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

- Example: Truth tables for the basic logic operations:

AND		
Inputs		Output
X	Y	Z = X . Y
0	0	0
0	1	0
1	0	0
1	1	1

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

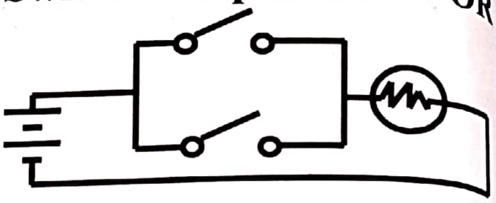
NOT	
Inputs	Output
X	$Z = \overline{X}$
0	1
1	0

Logic Function Implementation

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

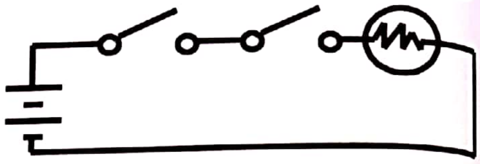
توازي

Switches in parallel => OR

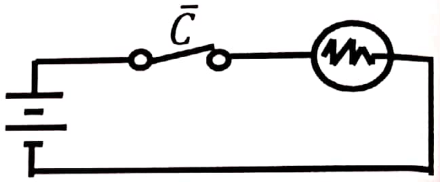


توالي

Switches in series => AND

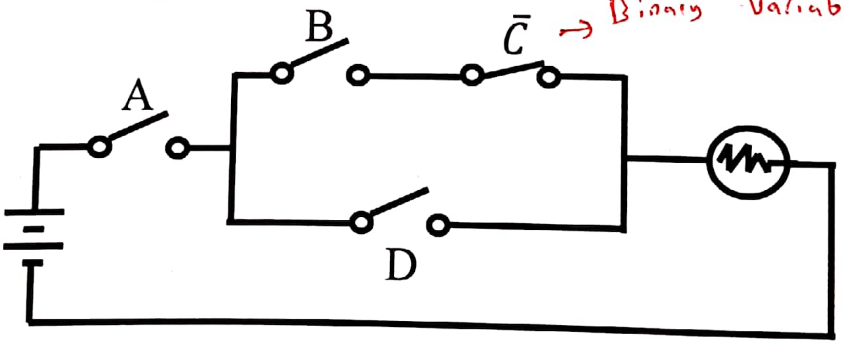


Normally-closed switch => NO



Logic Function Implementation (Continued)

- Example: Logic Using Switches



→ Binary Variable

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

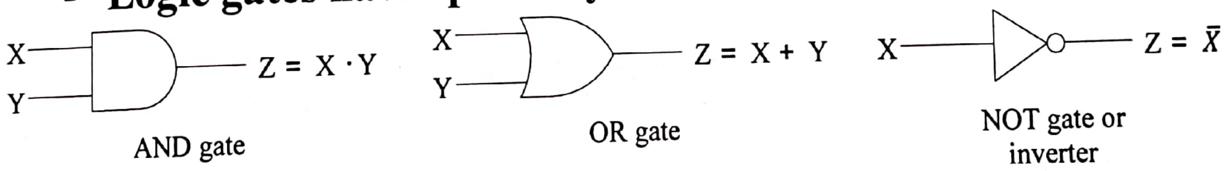
- Useful model for relay circuits and for CMOS gate circuits, the foundation of current digital logic technology

Logic Gates

- In the earliest computers, switches were opened and closed by magnetic fields produced by energizing coils in *relays.* The switches in turn opened and closed the current paths
- Later, *vacuum tubes* that open and close current paths electronically replaced relays
- Today, *transistors* are used as electronic switches that open and close current paths
- Optional: Chapter 6 – Part 1: The Design Space

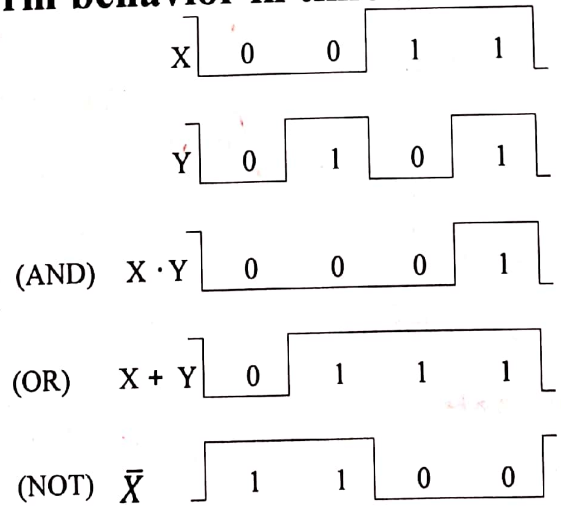
Logic Gate Symbols and Behavior

- **Logic gates have special symbols:**



(a) Graphic symbols

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

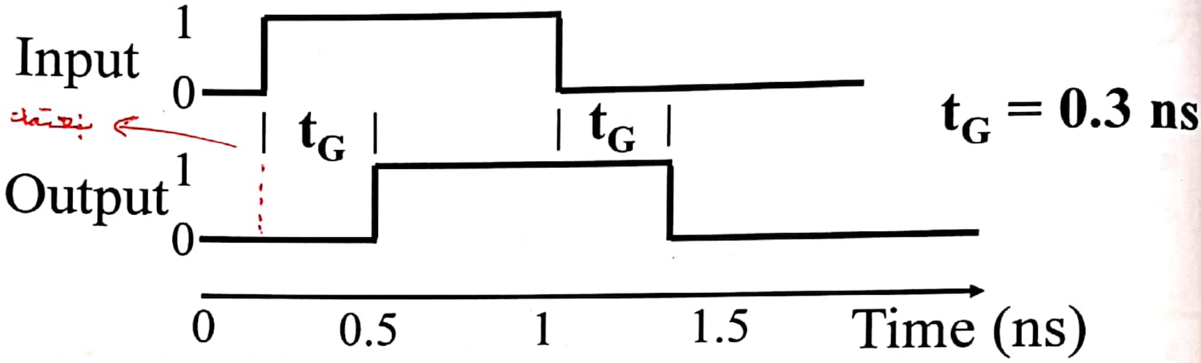


(b) Timing diagram

Gate Delay → بالحل ما بتيجي

- In actual physical gates, if one or more input changes causes the output to change, the output change does not occur instantaneously

- The delay between an input change(s) and the resulting output change is the ***gate delay*** denoted by t_G:



بيعطيك هذا الجواب

t_G = 0.3 ns

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

Logic Gates: Inputs and Outputs

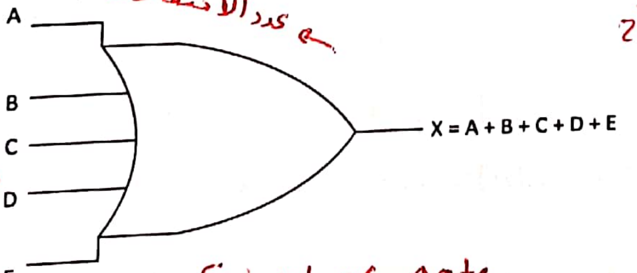
- NOT (inverter) العاكس
  - Always one input and one output

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

$2^5 = 32$

كلهم 1 ما عدا واحد هوه 0 ⇐ $(0+0+0+0)$

عدد الاحتمالات

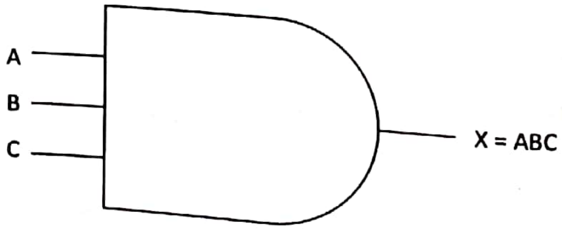


5 input or gate

كلهم 0 ما عدا واحد

١ ⇐ $(1.1.1)$

$2^3 = 8$ عدد الاحتمالات



3 input or gates

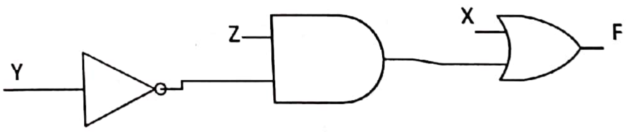
Scanned by CamScanner

Boolean Algebra

- An algebra dealing with binary variables and logic operations
  - Variables are designated by letters of the alphabet
  - Basic logic operations: AND, OR, and NOT
- ***A Boolean expression*** is an algebraic expression formed by using binary variables, constants 0 and 1, the logic operation symbols, and parentheses
  - E.g.: X . 1, A + B + C, (A + B)(C + D)
- ***A Boolean function*** consists of a binary variable identifying the function followed by equals sign and a Boolean expression
  - E.g.: $F = A + B + C,\ L(D, X, A) = DX + \bar{A}$

Logic Diagrams and Expressions

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



3. Truth Table:

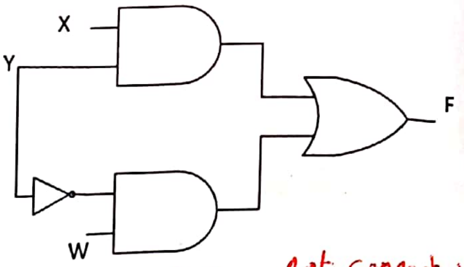
X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

الأولوية:
1 ( )
2 Not
3 and
4 or

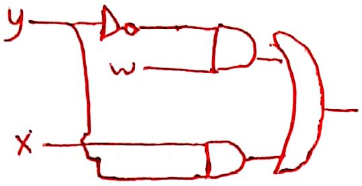
- Boolean equations, truth tables and logic diagrams describe the *same* function!
- Truth tables are *unique*; expressions and logic diagrams are not. This gives flexibility in implementing functions.

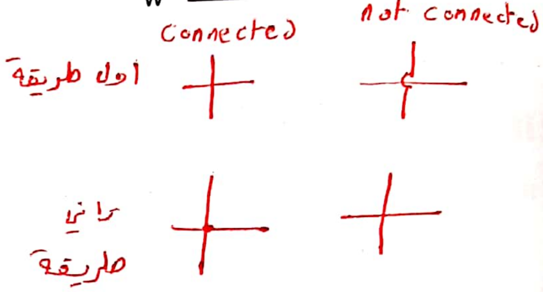
Example

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



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

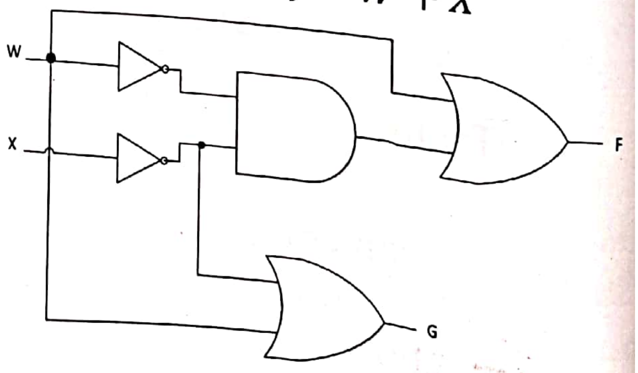




- This example represents a ***Single Output Function***

Example

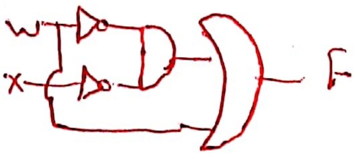
- Draw the logic diagram and the truth table of the following Boolean functions: $F(W,X) = \bar{W}\bar{X} + W, G(W,X) = W + \bar{X}$
- Logic Diagram:
- Truth Table:



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

or

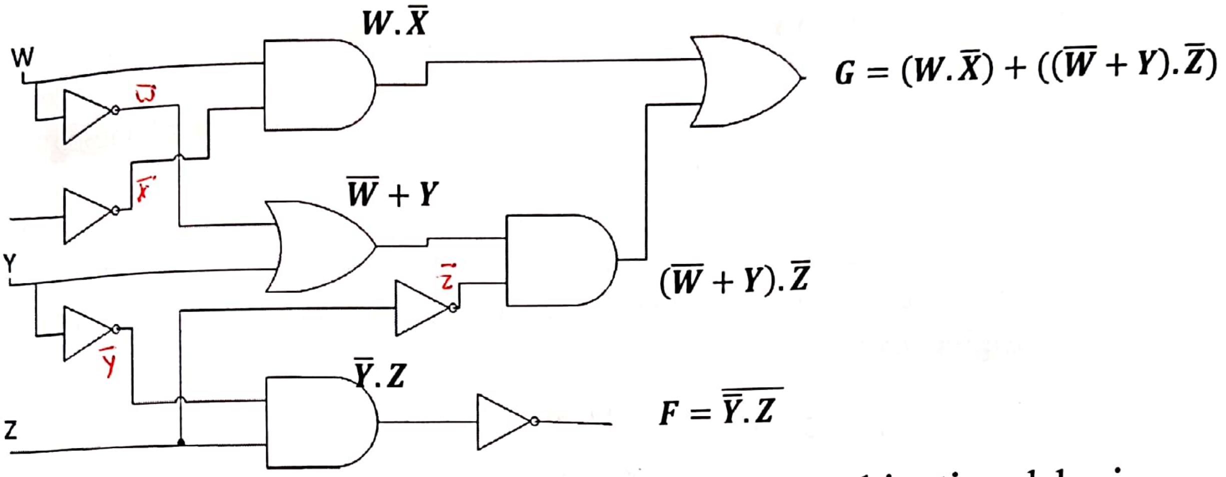
- This example represents a ***Multiple Output Function***



بكمل

Example:

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


$W.\bar{X}$
W
$\bar{w}$
$\bar{x}$
X
$\overline{W}+Y$
Y
$\bar{z}$
$\bar{y}$
$\bar{Y}.Z$
Z
$G = (W.\bar{X}) + ((\overline{W}+Y).\bar{Z})$
$(\overline{W}+Y).\bar{Z}$
$F = \overline{\bar{Y}.Z}$

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

خصائص

Basic Identities of Boolean Algebra

Ex: $\overline{X+(\bar{y}.Z)} = \bar{X}.(Y+\bar{Z}) \rightarrow$ تكامل

ل الأقواس المحصورة

Demorgan's law → الاسماء غير مطلوبة

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

اثباتا

$(x+y)(x+z)$

$x.x + x.z + y.x + x.x$

$x + xz + xy + yz$

$x(1+z+y) + yz$

$x.1+yz = x+yz$

$+ \rightarrow .$

$. \rightarrow +$

$x \rightarrow \bar{x}$

$\bar{x} \rightarrow x$

Some Properties of Identities & the Algebra

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

Dual: ① العمليات

② $X \rightarrow X$

$\bar{X} \rightarrow \bar{X}$

$1 \rightarrow 0$

$0 \rightarrow 1$

$\cdot \rightarrow +$

$+ \rightarrow \cdot$

③ اقتران للعمليات

Some Properties of Identities & the Algebra (Continued)

- Unless it happens to be self-dual, the dual of an expression does not equal the expression itself
- Examples:

نفس الاتي

Ex $((A+\bar{C}).B+1$

$((A.\bar{C})+B).0$

$=0$

  - $F = ((A + \bar{C})\,.\,B) + 0$
    - *Dual* $F = (A\,.\,\bar{C}) + \mathrm{B}\,.\,1 = \mathrm{A}\,.\,\bar{C} + B$
  - $G = \mathrm{XY} + (\overline{W + Z})$
    - *Dual* $G = (X + Y).\overline{WZ} = (X + Y)\,.\,(\bar{W} + \bar{Z})$
  - $H = AB + AC + BC$
    - *Dual* $H = (A + B)(A + C)(B + C) = (A + BC)(B + C)$ $= AB + AC + BC$
- Are any of these functions self-dual?
  - Yes, H is self-dual

اذا عملت للFunction
الDual ورجع نفسه بكون
Self - dual

Boolean Operator Precedence

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

- Consequence: Parentheses appear around OR expressions

- Examples:
  - $F = \overset{3}{\underline{A}}(\overset{2}{\underline{B + C}})(\overset{1}{\underline{C + \bar{D}}})$ 4
  - $F = {\sim}AB = \bar{A}B$
  - $F = AB + C$
  - $F = \underset{2}{A}(\underset{1}{B + C})$

Useful Boolean Theorems

Proof: $x + y1$ $x(1 + y)$ $x.1$ $\boxed{x}$

Proof: $y(x + \bar{x})$ $y.1$ $\boxed{y}$

مش حفظ

Theorem	*Dual*	*Name*
$x.y + \bar{x}.y\ = y$	$(x + y)(\bar{x} + y) = y$	**Minimization**
$x + x.y = x$	$x.(x + y) = x$	**Absorption**
$x + \bar{x}.y = x + y$	$x.(\bar{x} + y) = x.y$	**Simplification**
$x.y + \bar{x}.z + y.z = x.y + \bar{x}.z$ 3		مش مهمين كتير **Consensus**
$(x + y)(\bar{x} + z)(y + z) = (x + y)(\bar{x} + z)$		

Proof: $(x + \bar{x})(x + y)$
$1.(x + y)$
$(x + y)$

باستخدام نظرية:

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

Ex: $X + \bar{X}YZ = (X + \bar{X})(X + YZ)$
$= X + YZ$

$* \; AB + \overline{AB}X = AB + X \rightarrow AB \begin{cases} A\bar{B} \\ \bar{A}\bar{B}X \end{cases}$

Example 1: Boolean Algebraic Proof

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

$A + A \cdot B$	
$= A \cdot 1 + A \cdot B$	$X = X \cdot 1$
$= A \cdot (1 + B)$	*Distributive Law*
$= A \cdot 1$	$1 + X = 1$
$= A$	$X \cdot 1 = X$

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

Example 2: Boolean Algebraic Proofs

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

$AB + \bar{A}C + BC$	
$= AB + \bar{A}C + 1.BC$	$1.X = X$
$= AB + \bar{A}C + (A + \bar{A}).BC$	$X + \bar{X} = 1$
$= AB + \bar{A}C + ABC + \bar{A}BC$	***Distributive Law***
$= AB + ABC + \bar{A}C + \bar{A}BC$	***Commutative Law***
$= AB.1 + AB.C + \bar{A}C.1 + \bar{A}C.B$	$X.1 = X$ ***and Commutative Law***
$= AB(1 + C) + \bar{A}C(1 + B)$	***Distributive Law***
$= AB.1 + \bar{A}C.1$	$1 + X = 1$
$= AB + \bar{A}C$	$X.1 = X$

Proof of Simplification

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

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

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

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

Proof of Minimization

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

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

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

لما يكون عندك
اقواس الأفضل
التوزيع

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

Proof of DeMorgan's Laws (1)

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

	Justification (identity or theorem)
$(X+Y)+X'.Y'$	
$= (X+Y+X')(X+Y+Y')$	*Distributive Law*
$= (1+Y)(X+1)$	$X+\bar{X}=1$
$= 1.1$	$X+1=1$
$= 1$	$X.1=X$

* حل اخر:

$X + y + \bar{x}\,\bar{y}$

$X + y + \bar{y}\,\bar{x}$ عالقانون

$X + y + \bar{x} \rightarrow 1 + y = \boxed{1}$

$y + x + \bar{x} \cdot \bar{y}$

$y + x + \bar{y}$

$1 + x$

$\boxed{1}$

Proof of DeMorgan's Laws (2)

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

$(X+Y).X'.Y'$	
$= (X.X'.Y') + (Y.X'.Y')$	*Distributive Law*
$= (0.Y') + (X'.0)$	$X.\bar{X}=0$
$= 0 + 0$	$X.0=0$
$= 0$	$X+0=X$

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

Example 3: Boolean Algebraic Proofs

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

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

Boolean Function Evaluation

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

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

Expression Simplification

هون انه نبسط

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

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

عدد الأحرف

Complementing Functions

المتممة يعني احسب $\bar{F}$

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

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

اذا اجتني هاي

وبده $F \leftarrow (\bar{F})$ ←

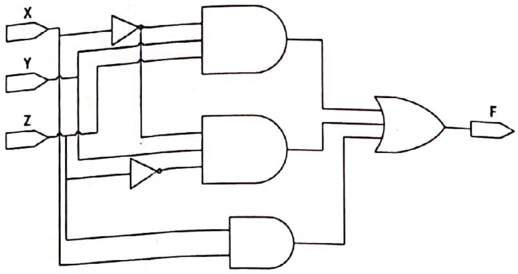
$$F' = (x + y' + z)(x' + y + z)$$

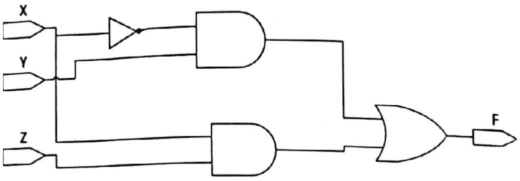
- Example: Complement $G = (a' + bc)d' + e$

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

Example

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





$X'YZ + X'YZ' + XZ$	
$= X'Y(Z + Z') + XZ$	*Distributive law*
$= X'Y.1 + XZ$	$X + X' = 1$
$= X'Y + XZ$	$X.1 = X$

x	y	z	$X'YZ + X'YZ' + XZ$	$X'Y + XZ$
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1
			3 terms and 8 literals	2 terms and 4 literals

بالـ truth table اثبت انه الاولى = الثاني بعد التبسيط

Example

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

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

  - Solution2: Boolean Algebra

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

Examples

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

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

او يختصر ويستخدم ← القانون

$\underbrace{AB}C + \bar{C}$

$\bar{C} + AB$

$AB + \bar{C}$

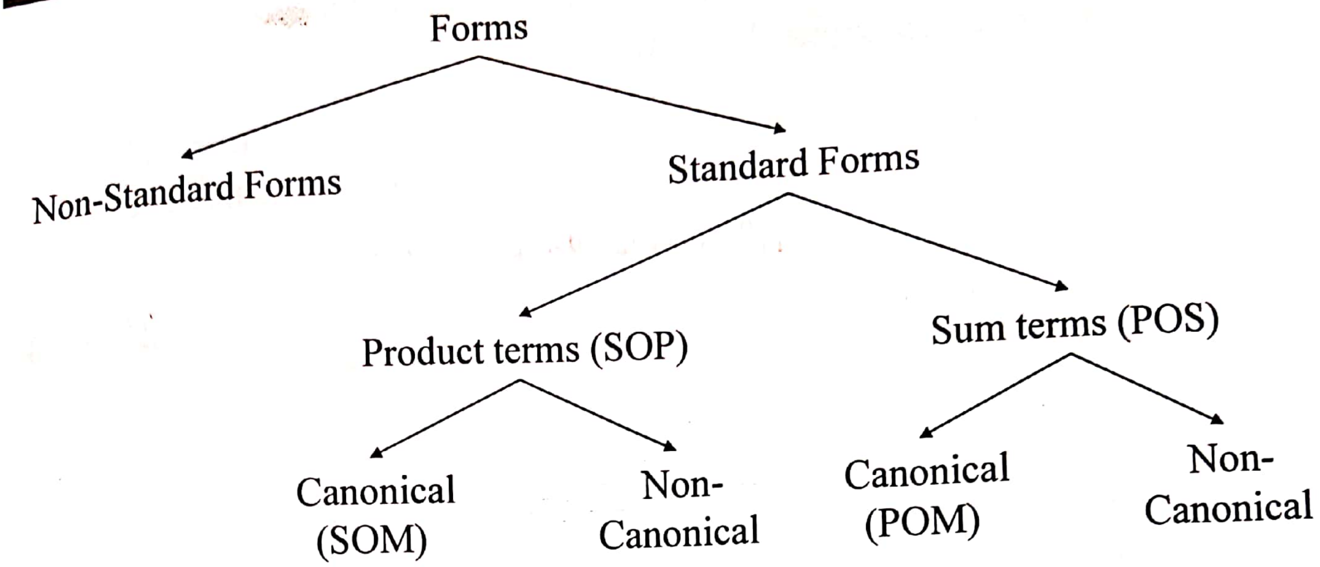
- Find the dual and the complement of $f = wx + y'z.0 + w'z$
  - $Dual(f) = (w + x)(y' + z + 1)(w' + z)$
  - $f' = (w' + x')(y + z' + 1)(w + z')$

Overview – Canonical Forms

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

طرق اني
انا اكتب
الـ Function (

Boolean Representation Forms



Forms

Non-Standard Forms

Standard Forms

Product terms (SOP)

Sum terms (POS)

Canonical (SOM)

Non-Canonical

Canonical (POM)

Non-Canonical

Canonical Forms

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

Minterms

Ex: $+ X \cdot Y +$
$+ \bar{X} \cdot Y + \bar{X} \cdot \bar{Y}$
$+ \bar{Y} \cdot X$

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

حسب عدد الVariables بطلعلي الخيارات 2^n

- Example: Two variables (X and Y) produce $2^2 = 4$ combinations:

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

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

Maxterms

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

حسب عدد ال Variables بطلعلي الخيارات

- Example: Two variables (X and Y) produce $2^2 = 4$ combinations:

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

Maxterms and Minterms

ـه ر

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

يعني بيعطيني $\boxed{1}$ — يعني بعطينا $\boxed{0}$

ترتيب قيم الـ Binary	Index (نفس ←)	Minterm (m)	Maxterm (M)
000	0	$\bar{X}\bar{Y}\bar{Z}$	$X + Y + Z$
001	1	$\bar{X}\bar{Y}Z$	$X + Y + \bar{Z}$
010	2	$\bar{X}Y\bar{Z}$	$X + \bar{Y} + Z$
011	3	$\bar{X}YZ$	$X + \bar{Y} + \bar{Z}$
100	4	$X\bar{Y}\bar{Z}$	$\bar{X} + Y + Z$
101	5	$X\bar{Y}Z$	$\bar{X} + Y + \bar{Z}$
110	6	$XY\bar{Z}$	$\bar{X} + \bar{Y} + Z$
111	7	XYZ	$\bar{X} + \bar{Y} + \bar{Z}$

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

Standard Order

- Minterms and maxterms are designated with a subscript
- The subscript is a number, corresponding to a binary pattern
- The bits in the pattern represent the complemented or normal state of each variable listed in a standard order
- All variables will be present in a minterm or maxterm and will be listed in the same order ***(usually alphabetically)***
- **Example: For variables a, b, c:** $\boxed{0}$
  - **Maxterms:** $\boxed{1}$ $(\overset{0}{a} + \overset{0}{b} + \overset{1}{\bar{c}}), (\overset{0}{a} + \overset{0}{b} + \overset{0}{c})$
  - **Terms:** $(b + a + c)$, $a\bar{c}b$, **and** $(c + b + a)$ **are NOT in standard order.** مش مرتبين
  - **Minterms:** $\boxed{5}$ $a\bar{b}c, abc, \bar{a}\bar{b}c$
  - **Terms:** $(a + c)$, $\bar{b}c$, **and** $(\bar{a} + b)$ **do not contain all variables**

Purpose of the Index

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

- **For Minterms:**
  - بنحط Bar (-) **"0" means the variable is "Complemented"**
  - ما بنحط اشي **"1" means the variable is "Not Complemented"**

- **For Maxterms:**
  - ما بنحط اشي **"0" means the variable is "Not Complemented"**
  - بنحط Bar **"1" means the variable is "Complemented"**

Index Example: Three Variables

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

Index Example: Four Variables

عكس ال mi بال Bar

$\overline{mi} = Mi$

$\overline{m5} = M5$

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

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

Minterm and Maxterm Relationship

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

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

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

Function Tables for Both

- Minterms of 2 variables:

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

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

- Maxterms of 2 variables:

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

- Each column in the maxterm function table is the complement of the column in the minterm function table since M_i is the complement of m_i.

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

$F = m_0 + m_3$

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

SoM

Observations

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

Minterm Function Example

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

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

Minterm Function Example

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

0 0 0 1 0 0 1 0 0 1

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

1 0 0 0 1 1 0 1 1 1

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

$\bar{A}\bar{B}\bar{C}D\bar{E} + \bar{A}B\bar{C}\bar{D}E + A\bar{D}\bar{C}\bar{D}E + A\bar{B}CDE$

Maxterm Function Example

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

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

PoM

Maxterm Function Example

- $F(A, B, C, D) = M_3 \,.\, M_8 \,.\, M_{11} \,.\, M_{14}$
- $$F(A,B,C,D) = (\overset{0}{A} + \overset{0}{B} + \overset{1}{C'} + \overset{1}{D'}) \,.\, (\overset{1}{A'} + \overset{0}{B} + \overset{0}{C} + \overset{0}{D}) \,.\, (\overset{1}{A'} + \overset{0}{B} + \overset{1}{C'} + \overset{1}{D'}) \,.\, (\overset{1}{A'} + \overset{1}{B'} + \overset{1}{C'} + \overset{0}{D})$$

(A + B + C + D) (A + B + C + D) (A + B + C + D)

(A + B + C + D)

Canonical Sum of Minterms

- Any Boolean function can be expressed as a Sum of Minterms (SOM):
  - For the function table, the minterms used are the terms corresponding to the 1's
  - For expressions, expand all terms first to explicitly list all minterms. Do this by "ANDing" any term missing a variable v with a term $(v + \bar{v})$
- Example: Implement $f = x + \bar{x}\bar{y}$ as a SOM?
  1. Expand terms → $f = x(y + \bar{y}) + \bar{x}\bar{y}$
  2. Distributive law → $f = xy + x\bar{y} + \bar{x}\bar{y}$
  3. Express as SOM → $f = m_3 + m_2 + m_0 = m_0 + m_2 + m_3$

(x,y,z)

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

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

$\underset{7}{xyz} + \underset{6}{xy\bar{z}} + \underset{5}{x\bar{y}z} + \underset{4}{x\bar{y}\bar{z}} + \underset{1}{\bar{x}\bar{y}z} + \underset{0}{\bar{x}\bar{y}\bar{z}}$

Another SOM Example

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

نشطب المكرر

- Collect terms (removing all but one of duplicate terms):
  - $F = \underset{\text{١١١}}{ABC} + \underset{\text{١١٠}}{AB\bar{C}} + \underset{\text{١٠١}}{A\bar{B}C} + \underset{\text{١٠٠}}{A\bar{B}\bar{C}} + \underset{\text{٠٠١}}{\bar{A}\bar{B}C}$
- Express as SOM:
  - $F = m_7 + m_6 + m_5 + m_4 + m_1$
  - $F = m_1 + m_4 + m_5 + m_6 + m_7$

Shorthand SOM Form

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

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

$F = m_0 + m_1$ SOM $\Sigma(0,1)$

$F = M_2 \cdot M_3$ POM

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

Canonical Product of Maxterms

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

Ex: $F(x,y,z) =$

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

$x + \bar{y} + v$

$\underset{\bar{A}}{x} + \underset{\bar{B}}{\bar{y}} + \underset{\bar{C}}{z \cdot \bar{z}}$

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

$M_2 \cdot M_3$

$\Pi_M(2,3)$

Another POM Example

- Convert $f(A,B,C) = AC' + BC + A'B'$ to POM?
- Use $x + yz = (x + y) . (x + z)$, assuming $x = AC' + BC$ and $y = A'$ and $z = B'$
  - $f(A,B,C) = (AC' + BC + A') . (AC' + BC + B')$
- Use Simplification theorem to get:
  - $f(A,B,C) = (BC + A' + C') . (AC' + B' + C)$
- Use Simplification theorem again to get:
  - $f(A,B,C) = (\overset{1}{A'} + \overset{0}{B} + \overset{1}{C'}) . (\overset{0}{A} + \overset{1}{B'} + \overset{0}{C}) = M_5 . M_2$
  - $f(A,B,C) = M_2 . M_5 = \boxed{\prod_M(2,5)}$ → ***Shorthand POM form***

(1) $F = \Sigma m(0, 1, 3, 4, 6, 7)$

$\bar{F} = \Pi m(0, 1, 3, 4, 6, 7)$

x y z	F	$\bar{F}$
0 0 0	1	0
0 0 1	1	0
0 1 0	0	1
0 1 1	1	0
1 0 0	1	0
1 0 1	0	1
1 1 0	1	0
1 1 1	1	0

← ماض داريلـ truth table

POM (0)

(1) SOM والباقي

Function Complements

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

$F^{(A,B)} = \Sigma(1,2)$

$F = \Pi m(0,3)$

$F^{(A,B,C)} = \Sigma(1,2)$

$= \Pi m(0,3,4,5,6,7)$

$\bar{F} = \Sigma(0,3,4,5,6,7)$

Conversion Between Forms

- To convert between sum-of-minterms and product-of-maxterm form (or vice-versa) we follow these steps:
  - Find the function complement by swapping terms in the list with terms not in the list.
  - Change from products to sums, or vice versa.
- **Example:**Given F as before: $F(x,y,z) = \sum_m(1,3,5,7)$
  - Form the Complement:

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

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

Important Properties of Minterms

- Maxterms are seldom used directly to express Boolea functions
- Minterms properties:
  - For n Boolean variables, there are 2^n minterms (0 to 2^n -1)
  - Any Boolean function can be represented as a logical sum o minterms (SOM)
  - The complement of a function contains those minterms no included in the original function
  - A function that include all the 2^n minterms is equal to 1

○ + ○ + ○

Standard Forms

شرط كل ال Variables
Som → All term are minterms
→ Relation between term is (+)
(بيكتب ال minterms for.)

Sop عكس

- Standard Sum-of-Products (SOP) form: equations are written as an OR of AND terms
- Standard Product-of-Sums (POS) form: equations are written as an AND of OR terms

تعكس

Ex: $F(x,y) = \underbrace{x}_{x.1} + xy$
(Sop)

- **Examples:**
  - SOP: $ABC + \bar{A}\bar{B}C + B$
  - POS: $(A + B) . (A + \bar{B} + \bar{C}) . C$
- **These "mixed" forms are neither SOP nor POS**
  - $(A\underbrace{B + C}_{x})(A + C)$
  - $ABC\bar{} + \underbrace{AC(A + B)}_{x}$

مش SoP & Pos

(++) . (+++) . (++) (Pos)

(Pom) ماهي كل العناصر زي

Standard Sum-of-Products (SOP)

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

Standard Sum-of-Products (SOP

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

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

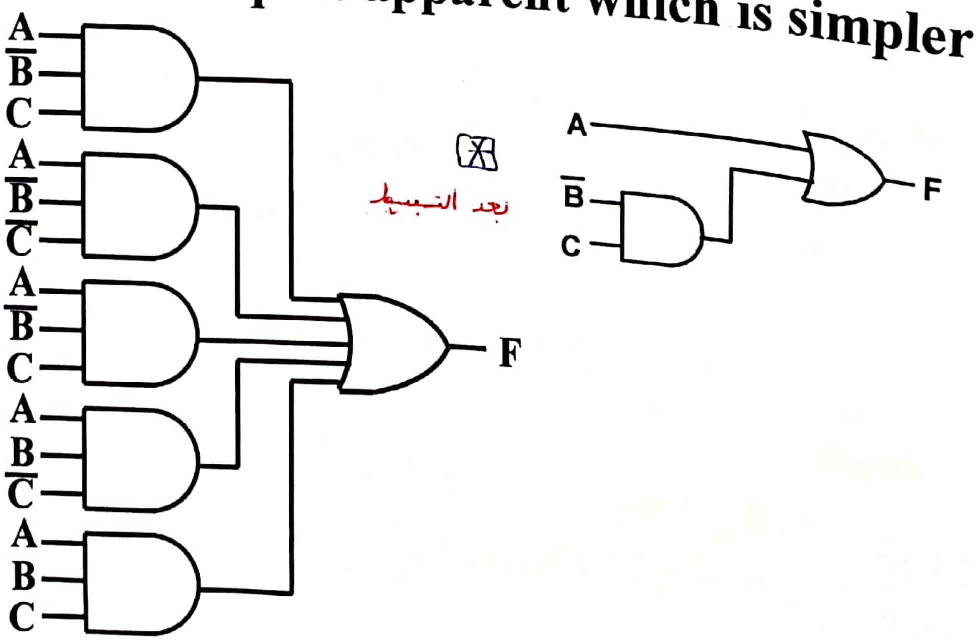
SoP ←

لانه جزء من ال
Variables

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

AND/OR Two-level Implementation of SOP Expression

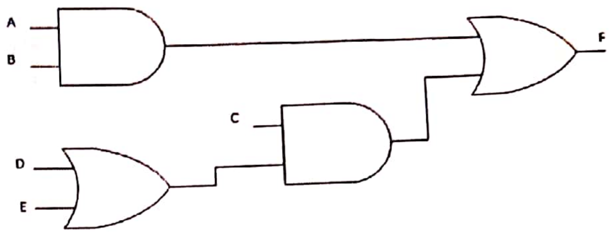
- **The two implementations for F are shown below – it is quite apparent which is simpler!**



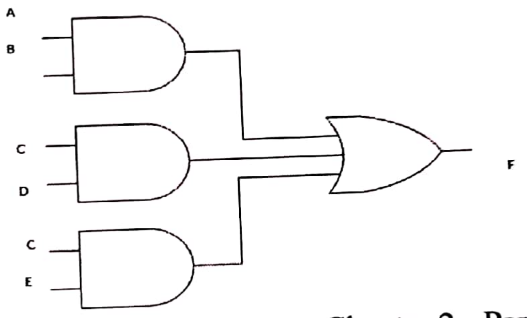
بعد التبسيط

Two-level Implementation

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



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



SOP and POS Observations

- **The previous examples show that:**
  - **Canonical Forms (Sum-of-minterms, Product-of-Maxterms), or other standard forms (SOP, POS) differ in complexity**
  - **Boolean algebra can be used to manipulate equations into simpler forms.**
  - **Simpler equations lead to simpler two-level implementations**
- **Questions:**
  - **How can we attain a "simplest" expression?**
  - **Is there only one minimum cost circuit?**
  - **The next part will deal with these issues.**

Logic and Computer Design Fundamentals

Chapter 2 – Combinational Logic Circuits

Part 2 – Circuit Optimization

Charles Kime & Thomas Kaminski

(Hyperlinks are active in View Show mode)

Overview

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

Circuit Optimization

- Goal: To obtain the simplest implementation for a given function
- Optimization is a more formal approach to simplification that is performed using a specific procedure or algorithm
- Optimization requires a cost criterion to measure the simplicity of a circuit
- Distinct cost criteria we will use:
  - **Literal cost (L)**
  - **Gate input cost (G)**
  - **Gate input cost with NOTs (GN)**

Literal Cost

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

Ex. $f = \underbrace{(A+B)}\,\underbrace{(A+CD)}\,\underbrace{(B+C+\bar{D})}\,\underbrace{(\bar{B}+\bar{C}+D)}$

$L = 11$

$G = 11 + 5 = 16$

Gate Input Cost

- *Gate input cost (G):* the number of inputs to the gates in the implementation corresponding exactly to the given equation or equations. (*G: inverters not counted, GN: inverters counted*)
- For SOP and POS equations, it can be found from the equation(s) by finding the sum of: $G = L + \#\,\text{Terms}$
  - **All literal appearances** لا يحسب المكرر $GN = G + \#\text{of inverters}$
  - **The number of terms excluding single literal terms,(G) and**
  - **optionally, the number of distinct complemented single literals (GN).**
- Examples: لازم نطلع نتيجة كل term اول.

$G = 8 + 3 = \boxed{11}$

  - $F = \underbrace{\overset{T_1}{BD}} + \underbrace{\overset{T_2}{AB'C}} + \overset{T_3}{A}\underbrace{C'D'}$
    - $G = 11\,, GN = 14$ (Minimum cost → Best solution)

$GN = 11 + 3 = 14$

$G = 11 + 4 = \boxed{15}$

  - $F = \underbrace{\overset{T_1}{BD}} + \underbrace{\overset{T_2}{AB'C}} + \underbrace{\overset{T_3}{AB'D'}} + \overset{T_4}{A}\underbrace{BC'}$
    - $G = 15\,, GN = 18$

$GN = 15 + 3 = 18$

$G = 10 + 4 = \boxed{14}$

  - $F = \underbrace{(\overset{T_1}{A} + B)}\underbrace{(\overset{T_2}{A} + D)}\underbrace{(\overset{T_3}{B + C + D'})}(\overset{T_4}{B' + C' + D})$
    - $G = 14\,, GN = 17$

$GN = 14 + 3 = 17$

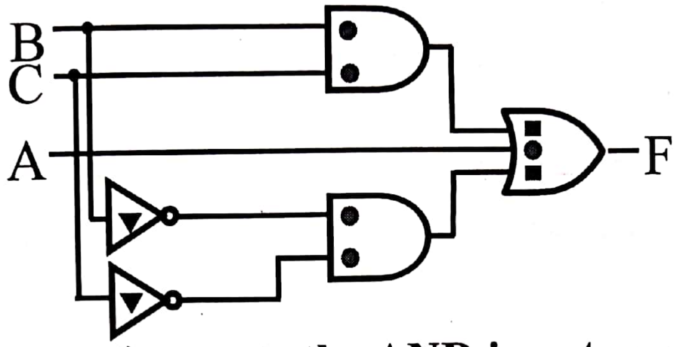
Cost Criteria (continued)

- Example 1:
- $F = A + BC + \bar{B}\bar{C}$

$GN = G + 2 = 9$ ▲

$L = 5$ ●

$G = L + 2 = 7$ ■



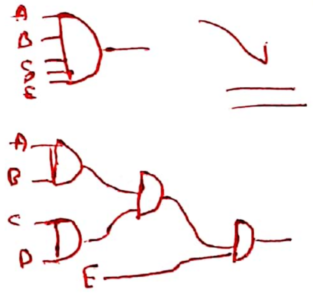
- L (literal count) counts the AND inputs and the single literal OR input.
- G (gate input count) adds the remaining OR gate inputs
- GN(gate input count with NOTs) adds the inverter inputs

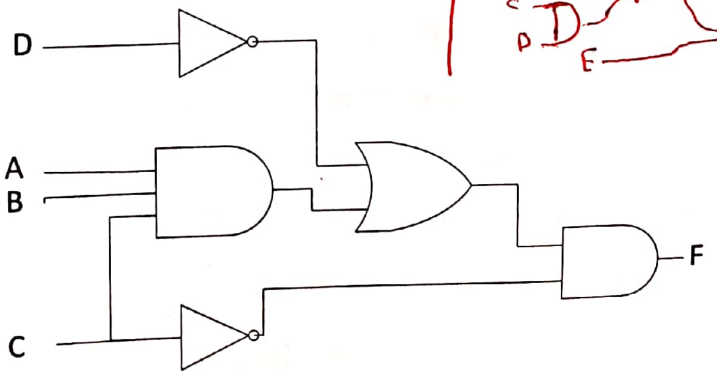
Cost Criteria (continued)

- **Example 2:**
- $F = (A,B,C,D) = (\overbrace{\underbrace{ABC}_{T_1} + D'}^{T_2}).C'$
  - $L = 5$
  - $G = 5 + 2 = 7$
  - $GN = 7 + 2 = 9$

خطأ
دايما اكبر Gate بنستعمل

Ex. ABCDE

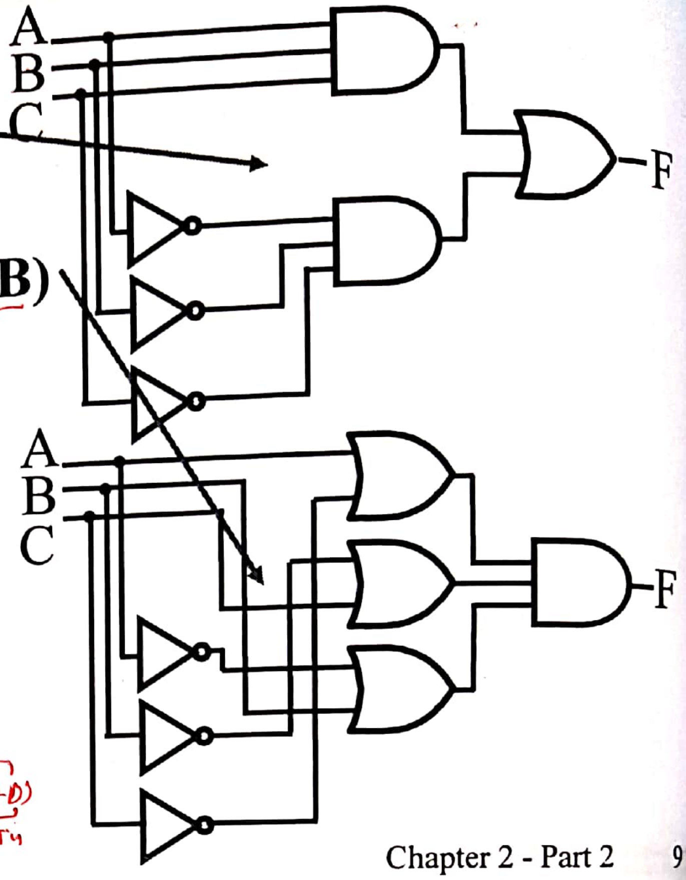




Cost Criteria (continued)

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

احسن لانه
GN
اقل
ن الـCost
اقل



$\text{Ex.}\ \underbrace{\overbrace{(A+\underbrace{\bar{B}C}_{T_1})}^{T_2}D}_{T_3}+\overbrace{E\underbrace{(C+D)}_{T_4}}^{T_5}$

$L = 7$

$G = 7+5 = 12$

$GN = 12+1 = 13$

Boolean Function Optimization

- Minimizing the gate input (or literal) cost of a (a set of) Boolean equation(s) reduces circuit cost
- We choose gate input cost
- Boolean Algebra and graphical techniques are tools to minimize cost criteria values
- Some important questions:
  - When do we stop trying to reduce the cost?
  - Do we know when we have a minimum cost?
- Treat optimum or near-optimum cost functions for two-level (SOP and POS) circuits
- Introduce a graphical technique using Karnaugh maps (K-maps, for short)

Karnaugh Maps (K-map)

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

Some Uses of K-Maps

- Finding optimum or near optimum
  - SOP and POS standard forms, and
  - two-level AND/OR and OR/AND circuit implementations

  for functions with small numbers of variables
- Visualizing concepts related to manipulating Boolean expressions, and
- Demonstrating concepts used by computer-aided design programs to simplify large circuits

Two Variable Maps

xy	F
00	a
01	b
10	c
11	d

	y=0	y=1
x=0	a	b
x=1	c	d

- **A 2-variable Karnaugh Map:**

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

الترتيب مهم

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

K-Map and Truth Tables

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

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

Truth Table

	y = 0	y = 1
x = 0	a	b
x = 1	c	d

K-Map

K-Map Function Representation

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

xy	F
00	0
01	0
10	1
11	1

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

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

x

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

$$F(x, y) = \underbrace{x\bar{y}}_{m2} + \underbrace{xy}_{m3} = \boxed{x}$$

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

الدائرة يا افقية او عامودية

K-Map Function Representation

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

xy	F
00	0
01	1
10	1
11	1

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

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

$\bar{x}y + x$

$\boxed{x + y}$

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

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

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

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

Three Variable Maps

XYZ	
000	a
001	B
010	C
011	D
100	E
101	F
110	g
111	h

- A three-variable K-map:

يجب ان يكون فقط متغير واحد مختلف

	yz = 00	yz = 01	yz = 11	yz = 10
x = 0	m_0 (A)	m_1 (B)	m_3 (D)	m_2 (C)
x = 1	m_4 (E)	m_5 (F)	m_7 (h)	m_6 (g)

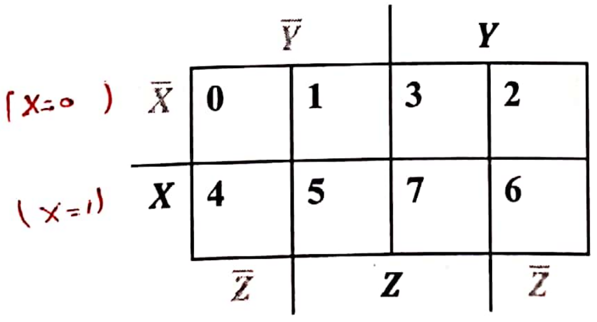
- Where each minterm corresponds to the product terms:

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

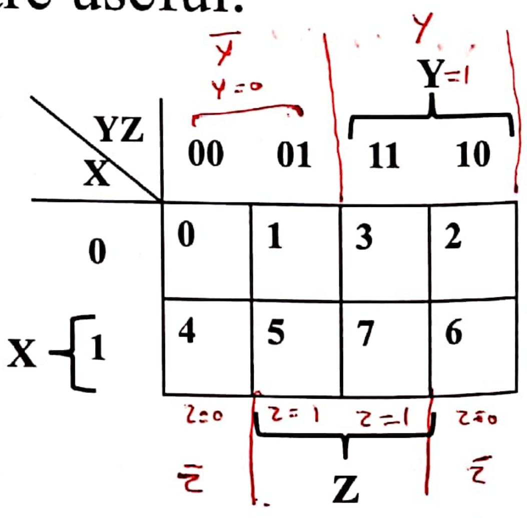
- *Note that if the binary value for an <u>index</u> differs in one bit position, the minterms are adjacent on the K-Map*

Alternative Map Labeling

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

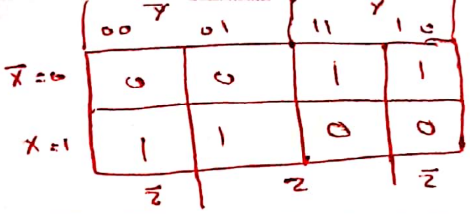


$\bar{X}$	0	1	3	2
X	4	5	7	6



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

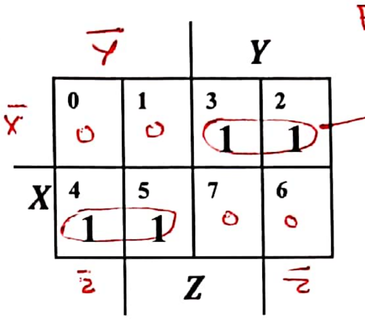
Scanned by CamScanner



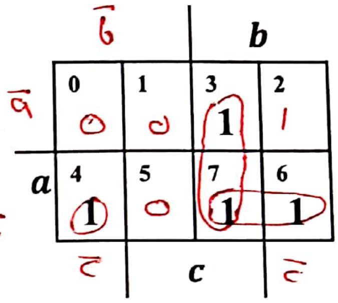
Example Functions

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

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



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

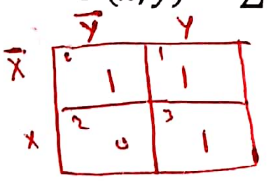


~~$F = \bar{a}b + ab$~~ $F = bc + ab + a\bar{b}\bar{c}$

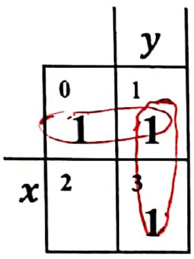
- Learn the locations of the 8 indices based on the variable order shown (X, most significant and Z, least significant) on the map boundaries

Steps for using K-Maps to Simplify Boolean Functions

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



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



$F = \bar{x} + y$

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

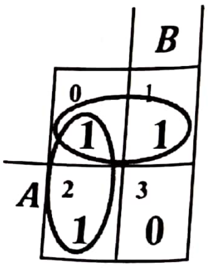
Combining Squares

- By combining squares, we reduce number of literals in a product term, reducing the literal cost, thereby reducing the other two cost criteria
- On a 2-variable K-Map:
  - One square represents a minterm with two variables
  - Two adjacent squares represent a product term with one variable
  - Four "adjacent" terms is the function of all ones (no variables) = 1.
- On a 3-variable K-Map:
  - One square represents a minterm with three variables
  - Two adjacent squares represent a product term with two variables
  - Four "adjacent" terms represent a product term with one variable
  - Eight "adjacent" terms is the function of all ones (no variables) = 1.

Example: Combining Squares

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

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



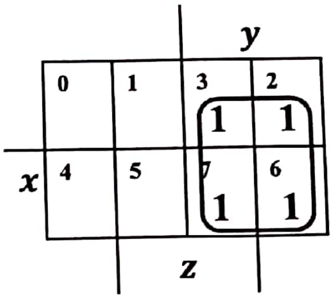
F = $\bar{A} + \bar{B}$

- Using Distributive law
  - $F(A,B) = \bar{A} + A\bar{B}$
- Using simplification theorem
  - $F(A,B) = \bar{A} + \bar{B}$
- **Thus, every two adjacent terms that form a 2×1 rectangle correspond to a product term with one variable**

Example: Combining Squares

- Example: $F(x,y,z) = \Sigma_m(2,3,6,7)$

- $F(x,y,z) = \bar{x}y\bar{z} + \bar{x}yz + xy\bar{z} + xyz$



- Using Distributive law
  - $F(x,y,z) = \bar{x}y + xy$

- Using Distributive law again
  - $F(x,y,z) = \boxed{y}$

- **Thus, the four adjacent terms that form a 2×2 square correspond to the term "y"**

Three-Variable Maps

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

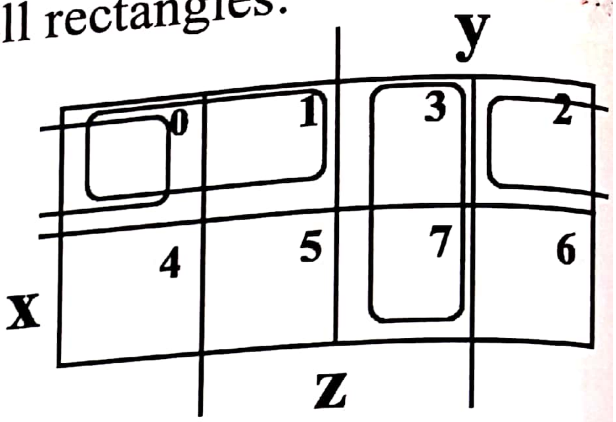
- Rectangles of 2 cells represent 2 adjacent minterms

- Rectangles of 4 cells represent 4 minterms that form a “pairwise adjacent” ring

- Rectangles can contain non-adjacent cells as illustrated by the “pairwise adjacent” ring above

Three-Variable Maps

- Example shapes of 2-cell rectangles:

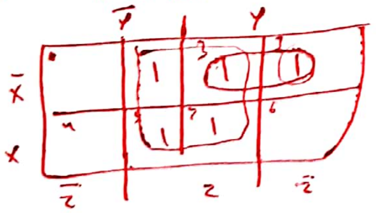


- Read-off the product terms for the rectangles shown:
  - $Rect(0,1) = \bar{X}\bar{Y}$
  - $Rect(0,2) = \bar{X}\bar{Z}$
  - $Rect(3,7) = YZ$

Three-Variable Maps

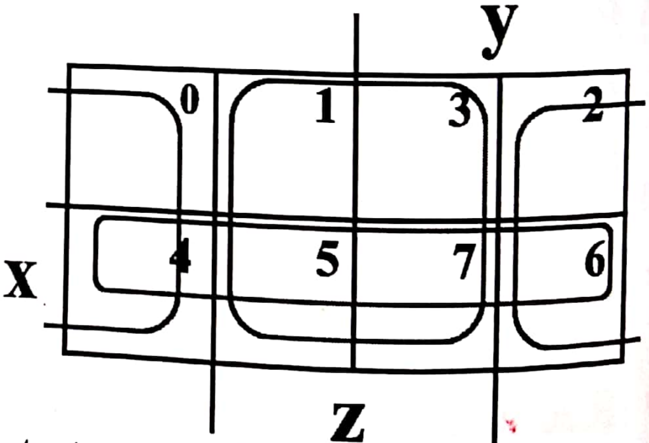
- Example shapes of 4-cell Rectangles:

Ex 1,2,3,5,7



~~$Z + \bar{X}Y\bar{Z}$~~

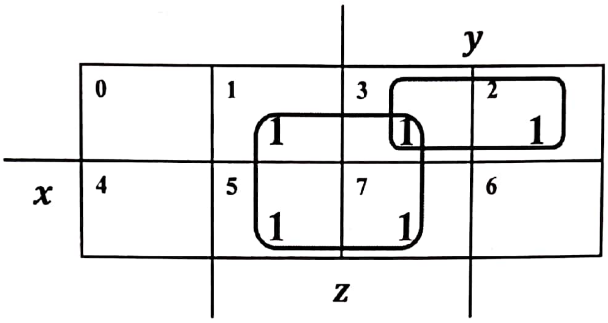
$Z + \bar{X}Y$



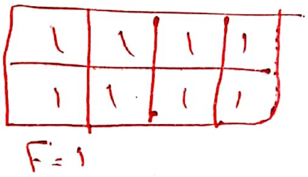
- Read off the product terms for the rectangles shown:
  - $Rect(1,3,5,7) = Z$
  - $Rect(0,2,4,6) = \bar{Z}$
  - $Rect(4,5,6,7) = X$

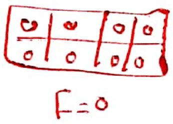
Three Variable Maps

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



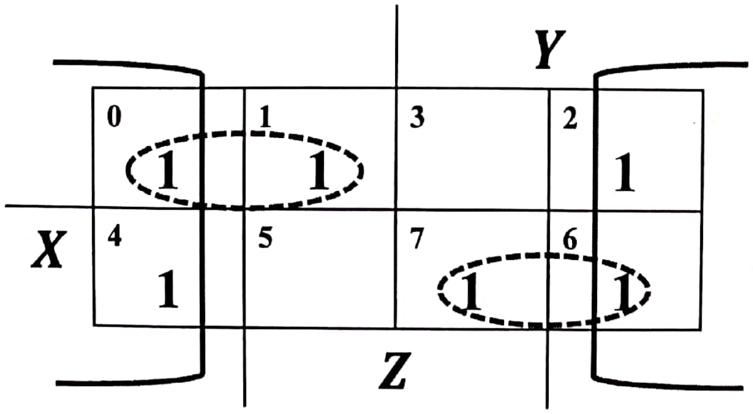
$$F(x, y, z) = z + \bar{x}y$$





Three-Variable Map Simplification

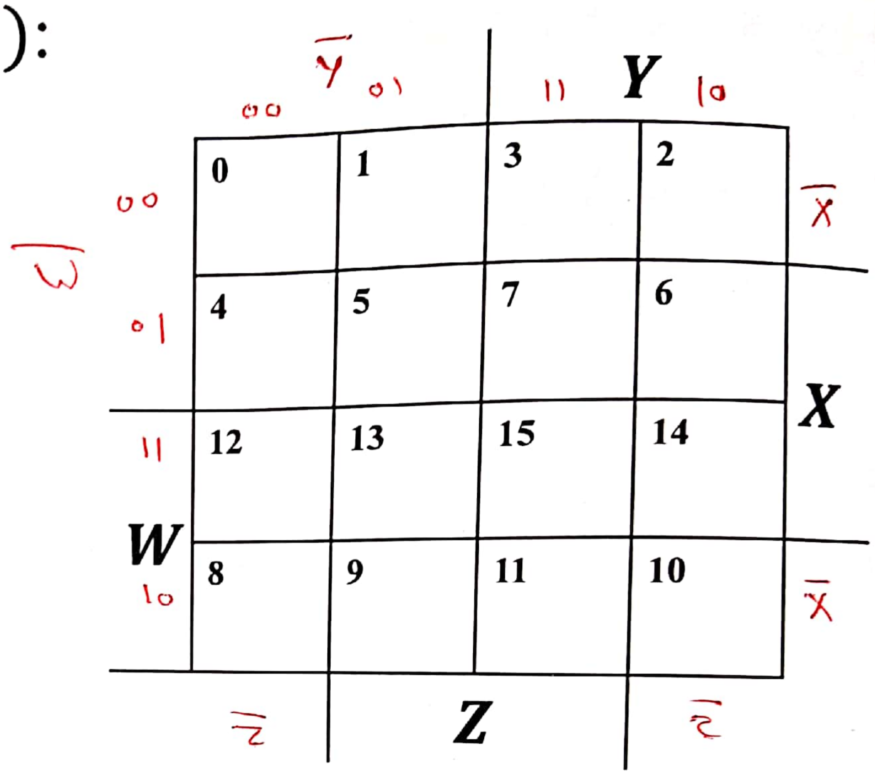
- Use a K-map to find an optimum SOP equation for $F(X, Y, Z) = \sum_m(0,1,2,4,6,7)$



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

Four Variable Maps

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



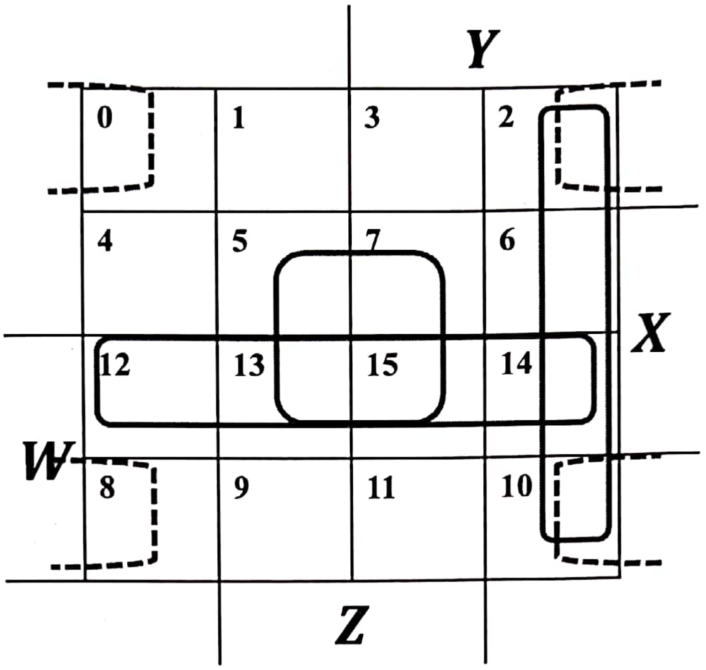
	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

Four Variable Terms

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

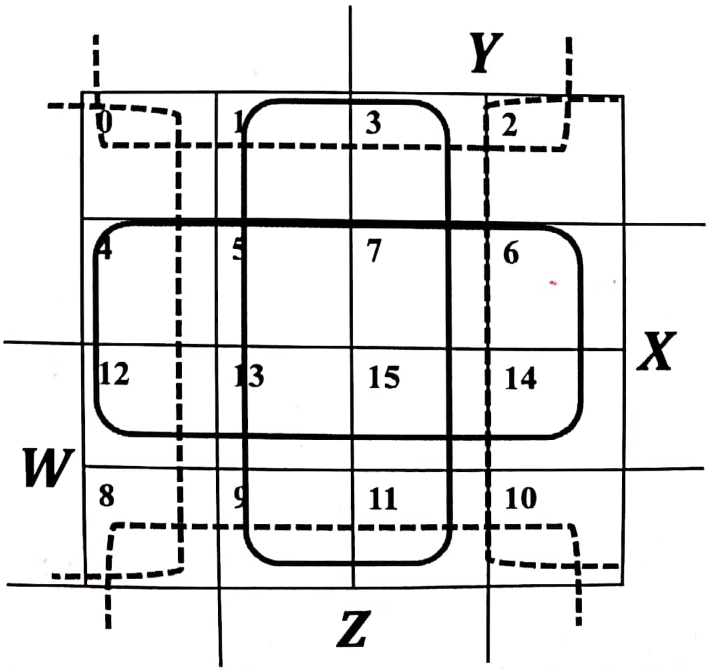
Four-Variable Maps

- Example shapes of 4-cell rectangles:



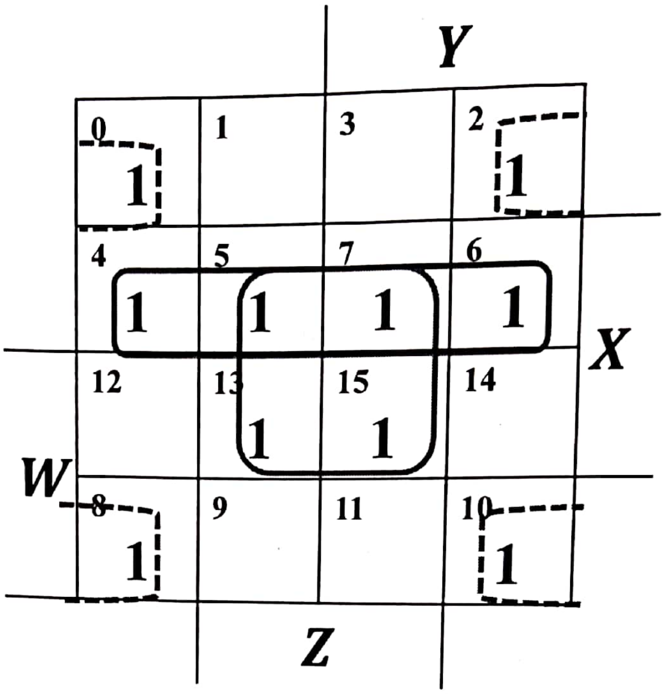
Four-Variable Maps

- Example shapes of 8-cell rectangles:



Four-Variable Map Simplification

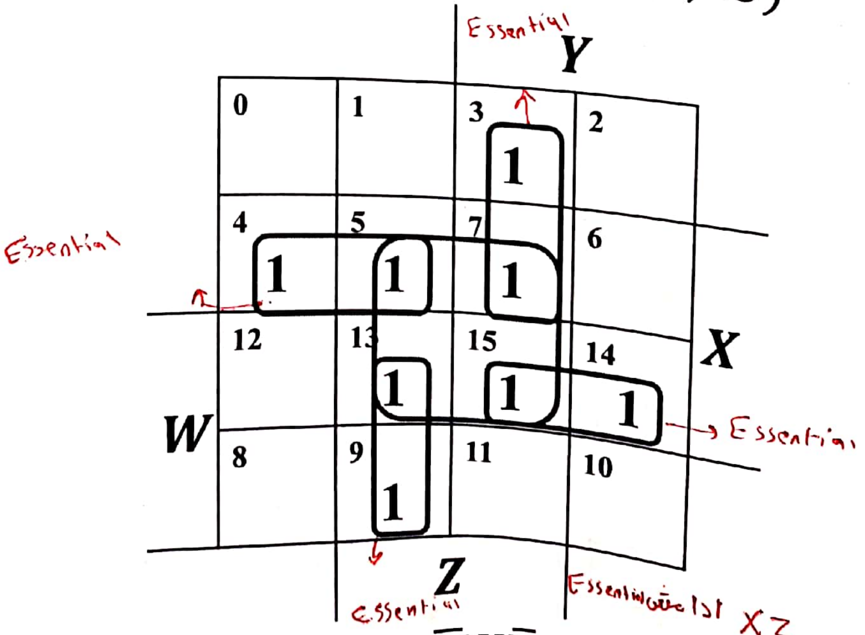
- $F(W,X,Y,Z) = \sum_m(0,2,4,5,6,7,8,10,13,15)$



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

Four-Variable Map Simplification

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



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

Systematic Simplification

اكبر دائرة حولين ال ①

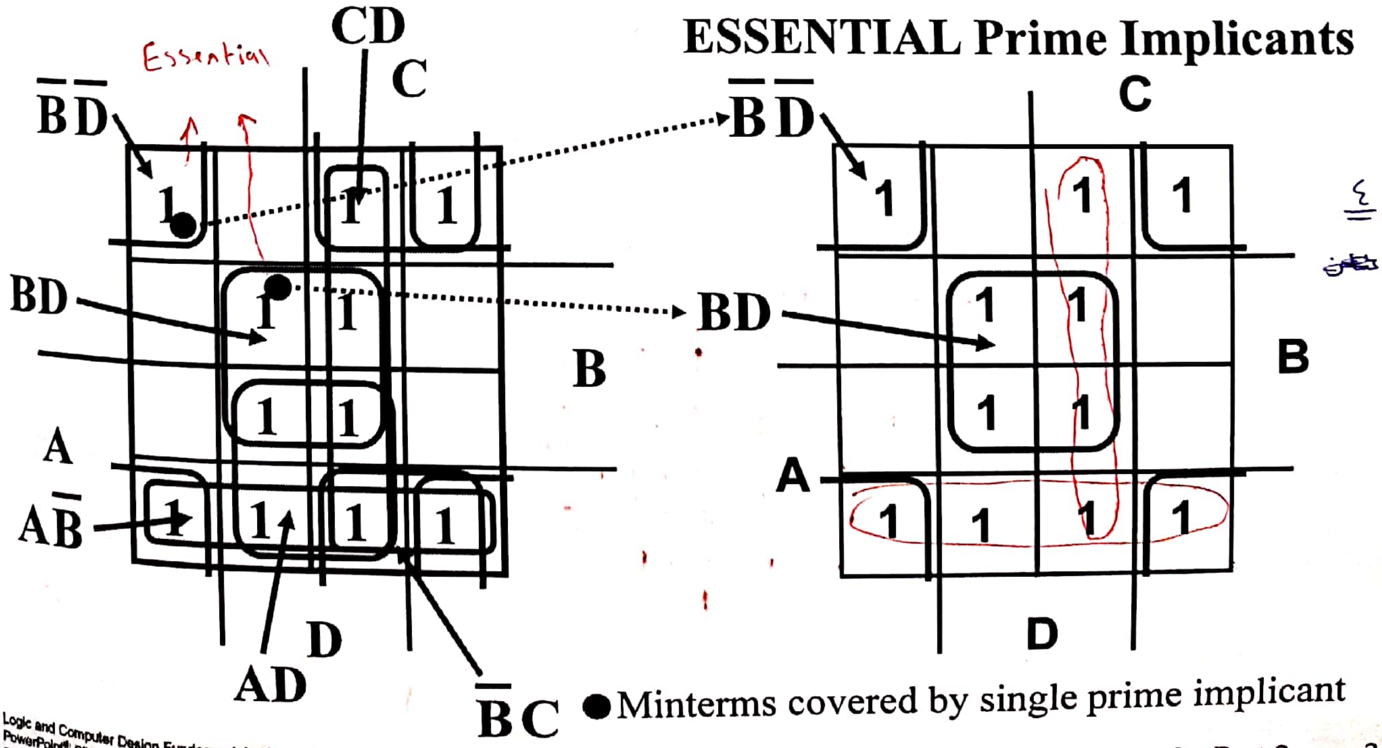
- *Prime Implicant*: is a product term obtained by combining the **maximum** possible number of adjacent squares in the map into a rectangle with the number of squares a power of 2

اكبر دائرة حولين ① + في جواها ①ما حدا بقدر يختاره الا هاي الدائرة.

- A prime implicant is called an ***Essential Prime Implicant*** if it is the **only** prime implicant that covers (includes) one or more minterms
- Prime Implicants and Essential Prime Implicants can be determined by inspection of a K-Map
- A set of prime implicants *"covers all minterms"* if, for each minterm of the function, at least one prime implicant in the set of prime implicants includes the minterm

Example of Prime Implicants

- **Find ALL Prime Implicants**



دائرتين ٤
بنضيف [illegible]

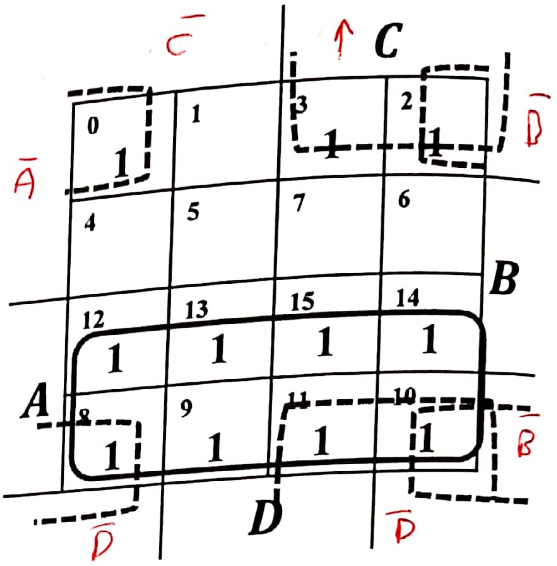
● Minterms covered by single prime implicant

Prime Implicant Practice

- Find all prime implicants for:

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

Essential ③ بسبب



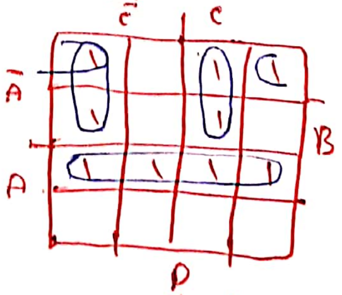
- Prime Implicants:
  - A essential
  - $\bar{B}C$ essential
  - $\bar{B}\bar{D}$ essential $\boxed{0}$ بسبب

Another Example

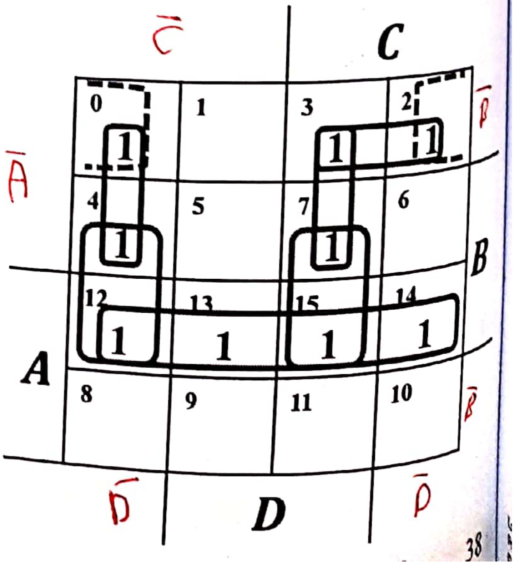
- Find all prime implicants for:

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

- Hint: There are seven prime implicants!
- Prime Implicants:
  - AB + essential ⑬ بسبب
  - BCD
  - $B\bar{C}\bar{D}$
  - $\bar{A}CD$
  - $\bar{A}\bar{C}\bar{D}$
  - $\bar{A}\bar{B}C$
  - $\bar{A}\bar{B}\bar{D}$



في اكثر من حل



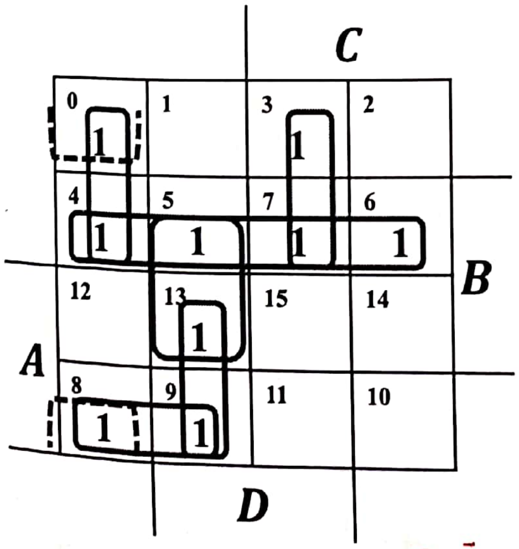
Optimization Algorithm

الـ essential لازم اخدهم

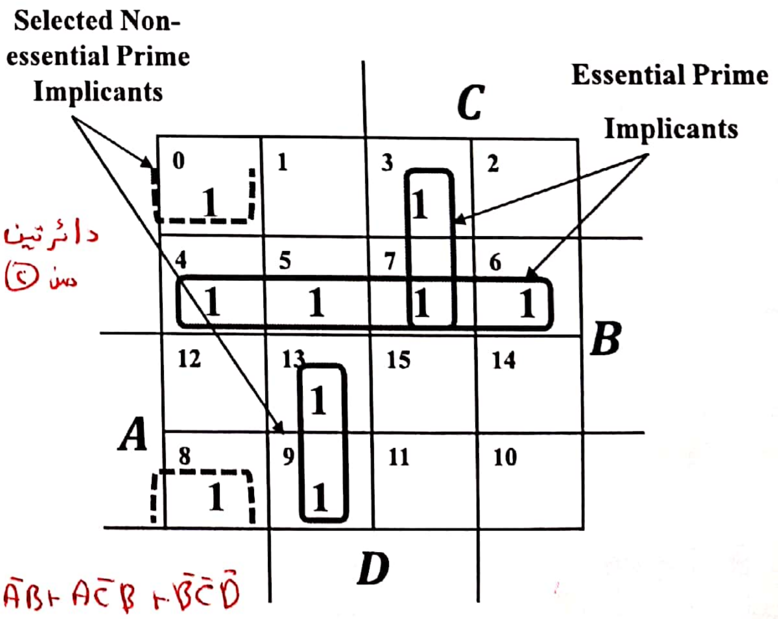
1. Find all prime implicants
2. Include all essential prime implicants in the solution
3. Select a *minimum cost* set of non-essential prime implicants to cover all minterms not yet covered
   - Selection Rule: Minimize the overlap among prime implicants as much as possible. In particular, in the final solution, make sure that each prime implicant selected includes at least one minterm not included in any other prime implicant selected

Selection Rule Example

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



Prime Implicants



Selected Non-essential Prime Implicants

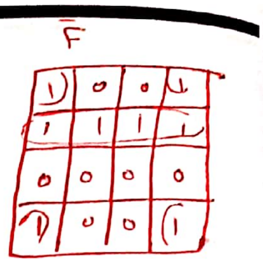
Essential Prime Implicants

$F = \bar{A}CD + \bar{A}B + A\bar{C}B + \bar{B}\bar{C}\bar{D}$

Essential and Selected Non-essential Prime Implicants

Product of Sums Example

Sop (F̄).



- Find the optimum POS solution for:

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

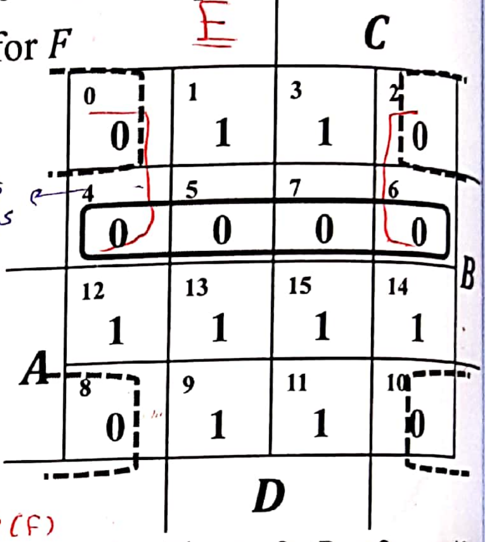
- Solution:
  - Find optimized SOP for $\bar{F}$ by combining 0's in K-Map of F
  - Complement $\bar{F}$ to obtain optimized POS for F

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

كتاب اطلع ←
كل الـ Implicant

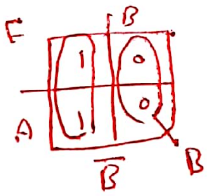
- Using Demorgan's Law:

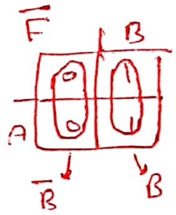
$$F(A,B,C,D) = (A+\bar{B})(B+D)$$



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

K-map (F) → group 1's → Sop(F)
→ group 0's → Sop(F̄)





$\overline{Sop(\bar{F})}$ → Pos(F)

Example

- Find the optimum POS and SOP solution for:

بنعبي ال 0

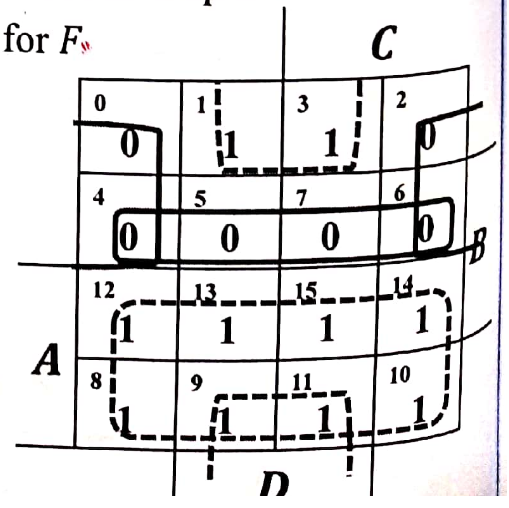
$$F(A,B,C,D) = \prod_{M}(0,2,4,5,6,7)$$

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

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

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

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



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

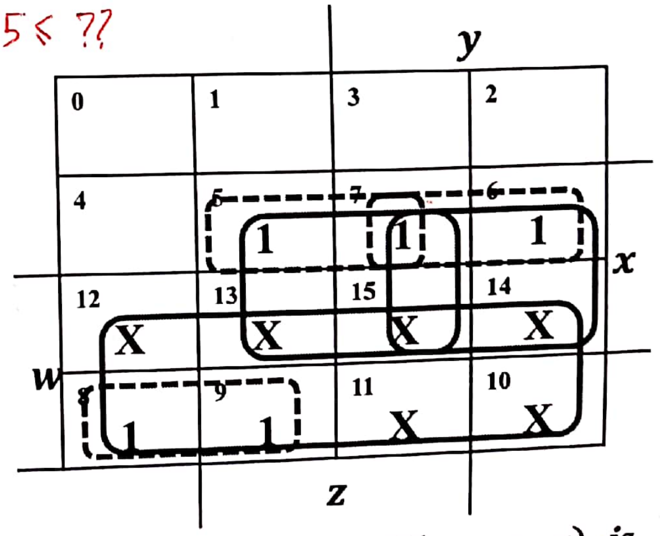
Don't Cares in K-Maps

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

Example: BCD "5 or More"

- The map below gives a function $F(w,x,y,z)$ which is defined as "5 or more" over BCD inputs. With the don't cares used for the 6 non-BCD combinations:

5 ≤ ??



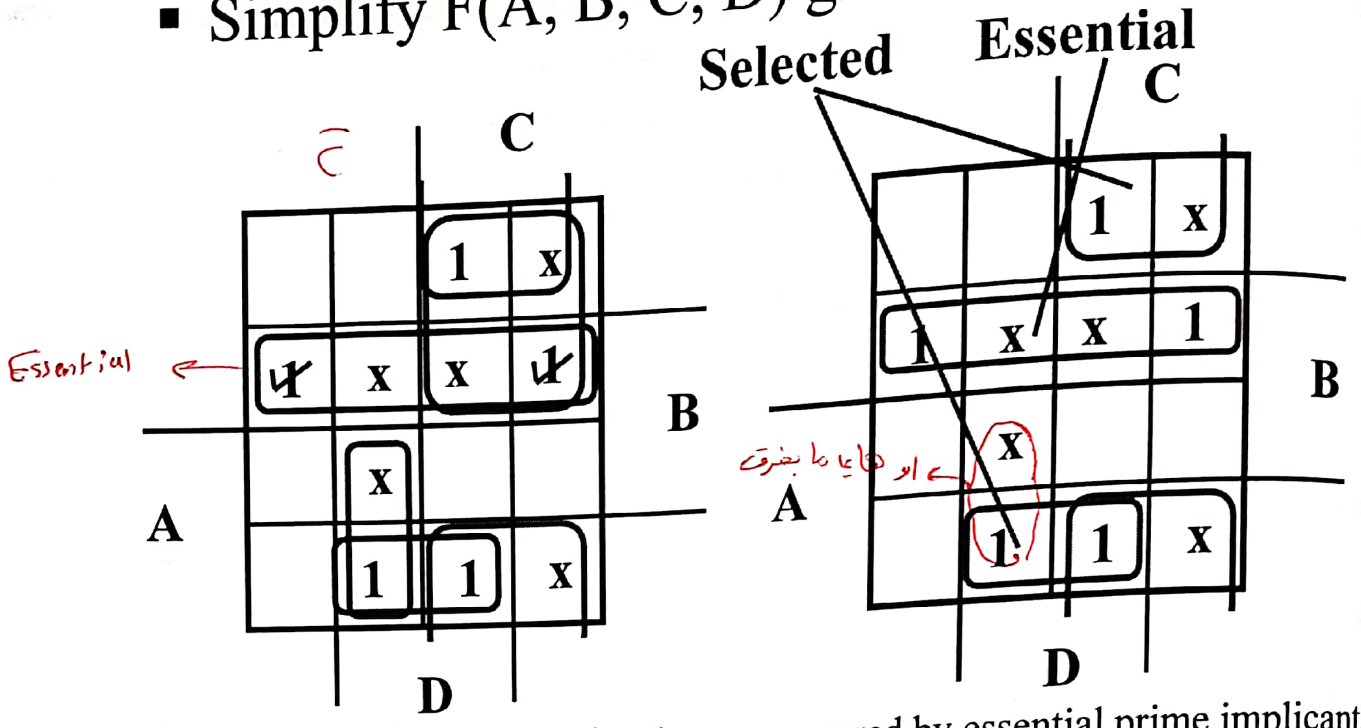
- If don't cares are treated as 1's (Red):
- $F_1(w,x,y,z) = w + xy + xz$
  - $G = 7$
- If don't cares are treated as 0's (Blue):
- $F_2(w,x,y,z) = \bar{w}xz + \bar{w}xy + w\bar{x}\bar{y}$
  - $G = 12$

الـ Don't care نفرضهم ① (0) اذا بصير ابسط (لما اكون اجمع ① (0) نيولين)
ومش شرط اشغليهم كلهم

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

Selection Rule Example with Don't Cares

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



✓ Minterms covered by essential prime implicants

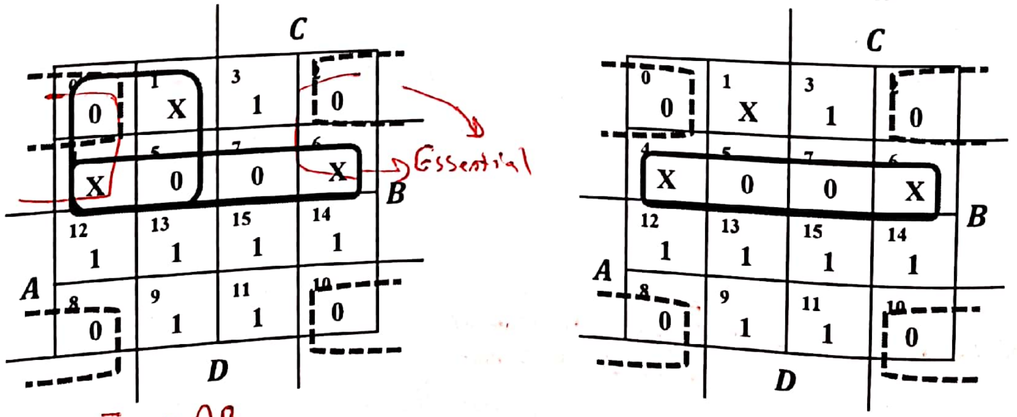
$F = \bar{A}B + \bar{B}C + A\bar{B}b$

Product of Sums with Don’t Care Example

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

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

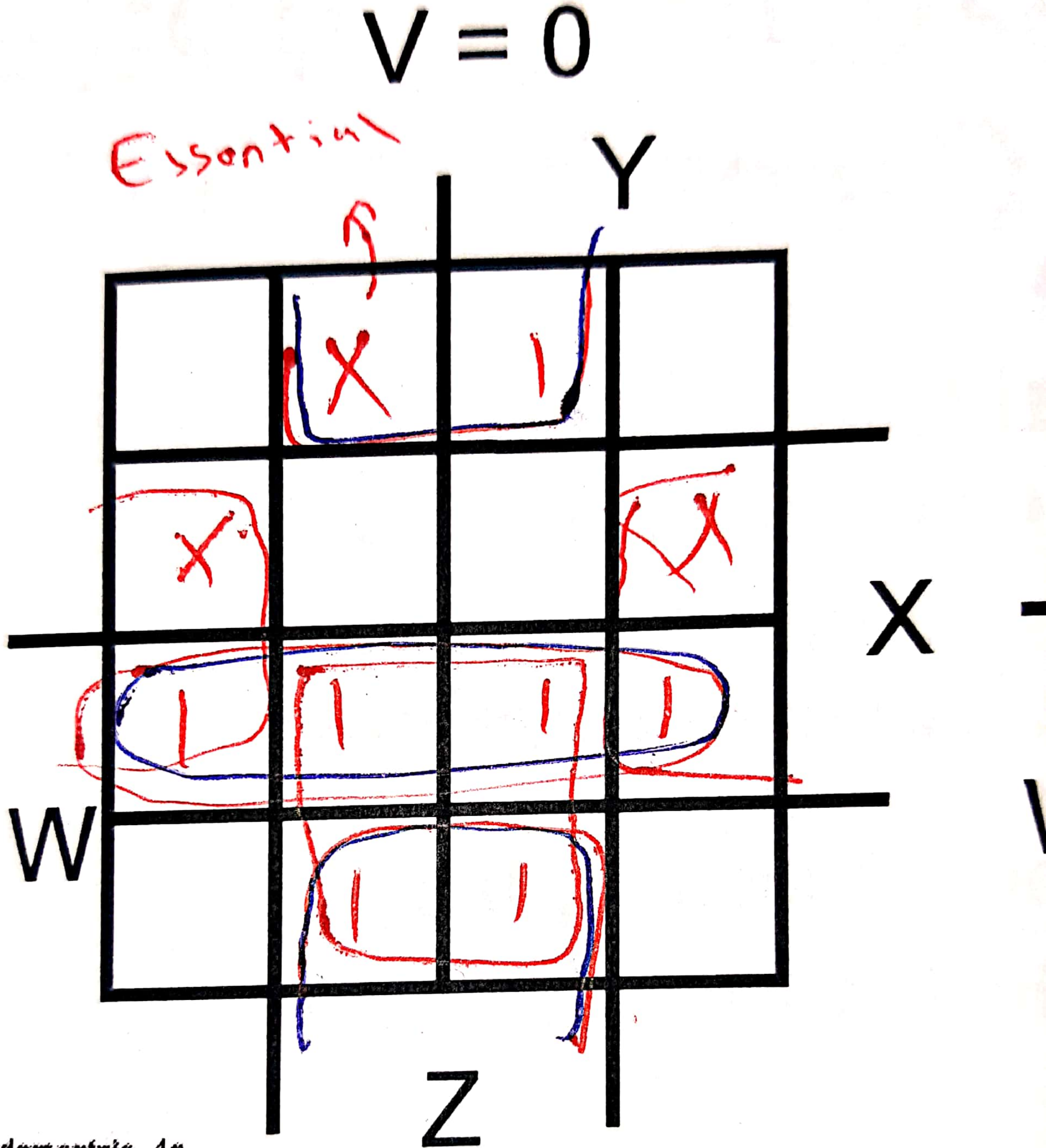
don't care



$F = \bar{B}D + AB$ لل ① حلاب ٤٧

لل ⓪ $\bar{F}(A,B,C,D) = \bar{A}B + \bar{B}\bar{D}$

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



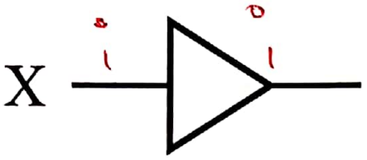
	X	1	
X			X
1	1	1	1
	1	1	

Design Fundamentals, 4e

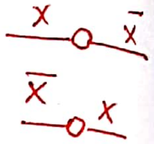
cation, Inc.

Buffer

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



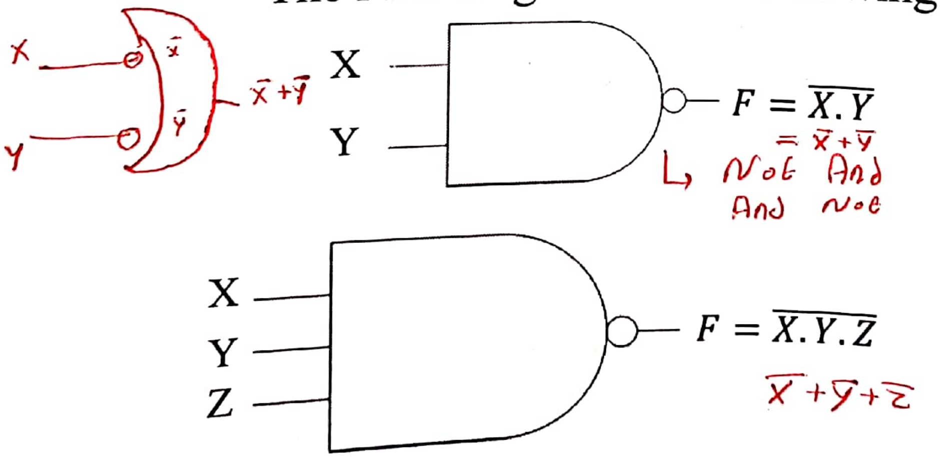
X	*F*
0	*0*
1	*1*



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

NAND Gate

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



$F = \overline{X.Y}$

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

L, Not And

And Not

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

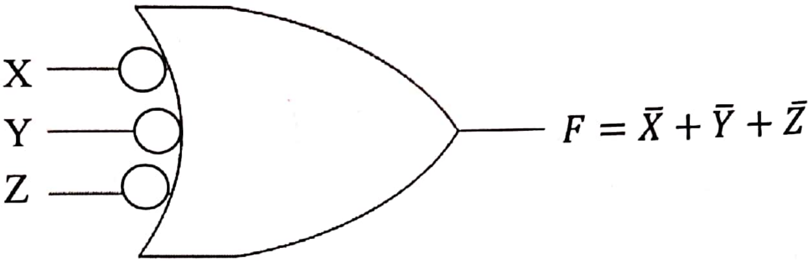
$\bar{X} + \bar{Y} + \bar{Z}$

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

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

NAND Gates (continued)

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



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

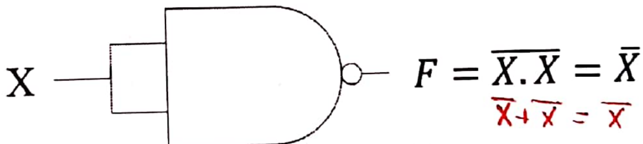
- This NAND symbol is called **Invert-OR**, since inputs are inverted and then ORed together

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

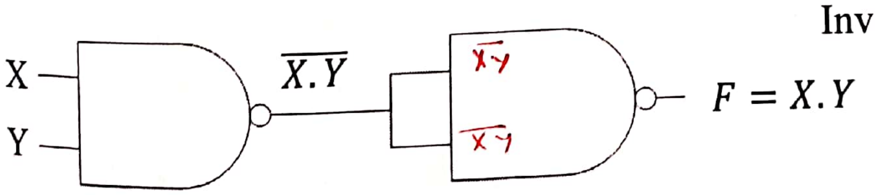
NAND Gates (continued)

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

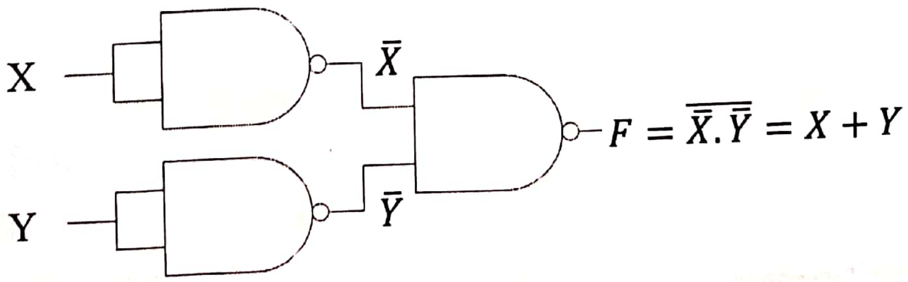
لانه ممكن اعمل منها اي اشي


X
$F = \overline{X.X} = \bar{X}$
$\overline{X}+\overline{X} = \overline{X}$

Inverter using NAND



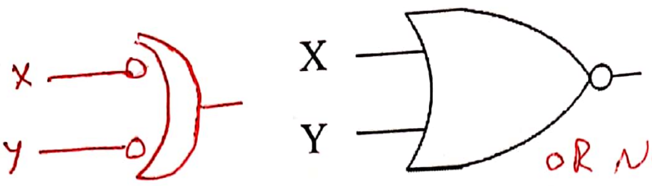
AND using NAND



OR using NAND

NOR Gate

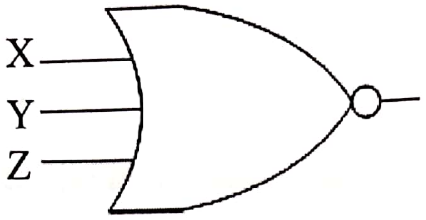
- The NOR gate has the following symbol and truth table:



$F = \overline{X + Y}$

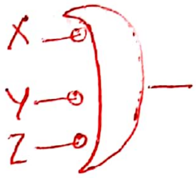
OR Not $\bar{X} \cdot \bar{Y}$

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



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

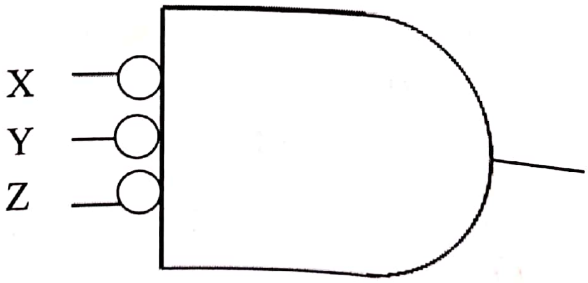
$\bar{x} \cdot \bar{y} \cdot \bar{z}$



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

NOR Gates (continued)

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



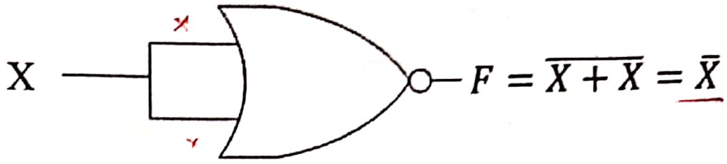
$F = \bar{X}.\bar{Y}.\bar{Z}$

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

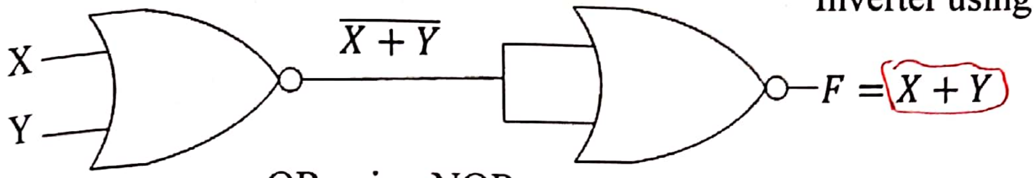
- **OR-Invert** and **Invert-AND** both represent the NOR gate. Having both makes visualization of circuit function easier

NOR Gates (continued)

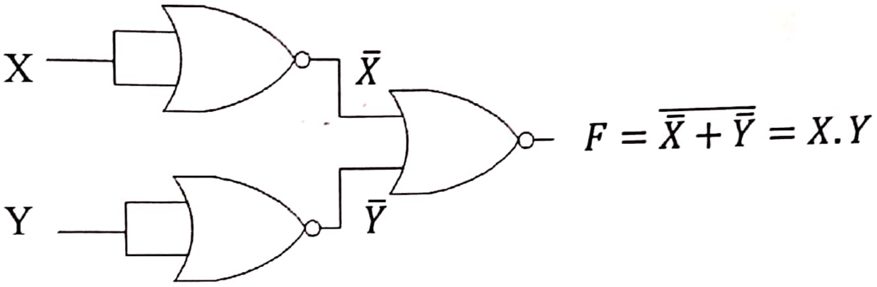
- **The NOR gate is a universal gate:**



Inverter using NOR



OR using NOR



AND using NOR

Hi-Impedance Outputs

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

اذا قطعنا السلك

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

Hi-Impedance Outputs (continued)

- **What is a Hi-Z value?**
  - The Hi-Z value behaves as an open circuit
  - This means that, looking back into the circuit, the output appears to be disconnected
  - It is as if a switch between the internal circuitry and the output has been opened

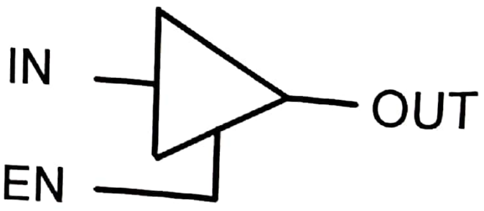
- Hi-Z may appear on the output of any gate, but we restrict gates to ***3-state buffer***

o, 1, Hi-z

Tri-State Buffer (3-State Buffer)

- For the symbol and truth table, **IN** is the **data input**, and **EN** is the **control input**
- For **EN** = **0**, regardless of the value on IN (denoted by X), the output value is Hi-Z
- For **EN** = **1**, the output value follows the input value

Symbol



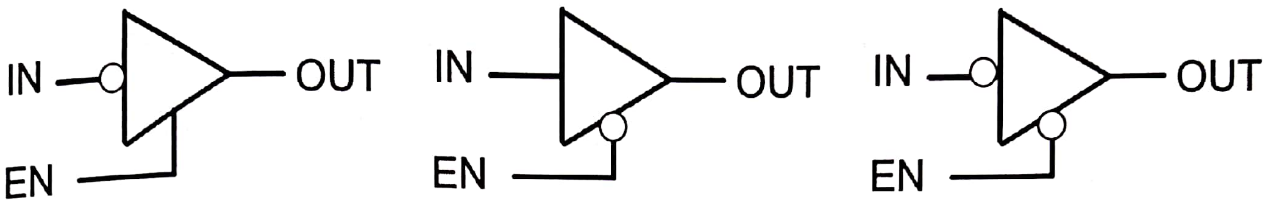
Truth Table

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

ال out نفس ال In

Tri-State Buffer Variations

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



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

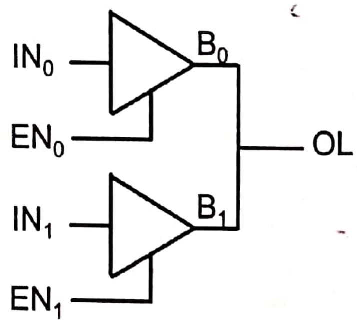
EN	*IN*	*OUT*
0	*0*	*0*
0	*1*	*1*
1	*X*	*Hi-Z*

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

Resolving 3-State Values on a Connection

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

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



لمنع هاد كله يصير
عشان اضمن
ما يصير عندي مشاكل

لما يكونوا ال EN ← 1

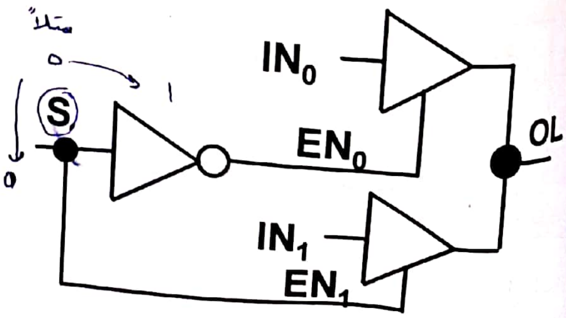
Resolving 3-State Values on a Connection

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

Tri-State Logic Circuit

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

S	EN_1	EN_0	IN_1	IN_0	OL
0	0	1	X	0	0
0	0	1	X	1	1
1	1	0	0	X	0
1	1	0	1	X	1



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

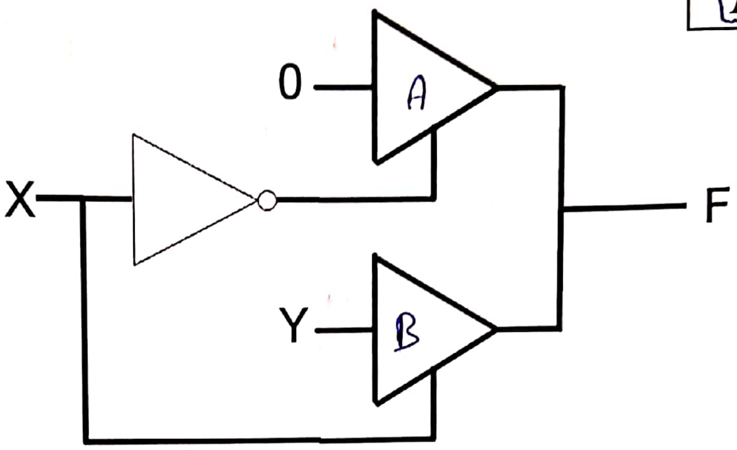
Logic Functions using Tri-State Buffers

- Implement AND gate using 3-State buffers and inverters

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

	X	Y	F
A	0	0	0
	0	1	0
B	1	0	0
	1	1	1

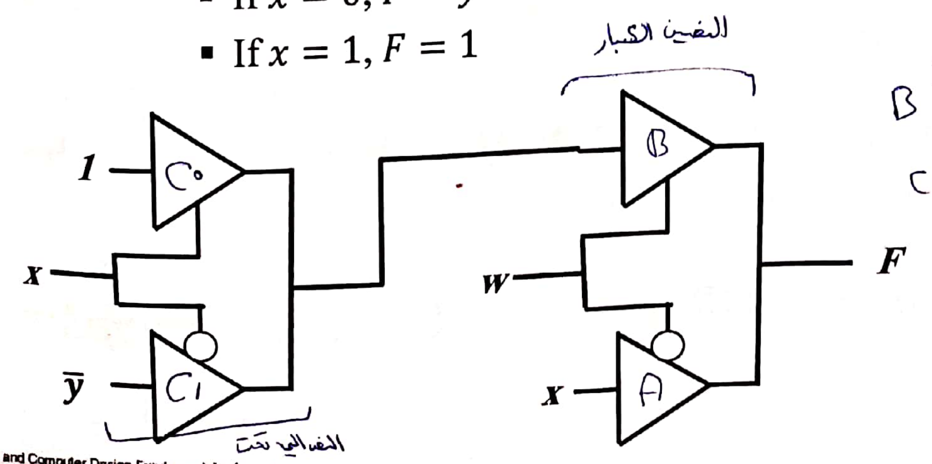
- Use X as control input:
  - When $X = 0$, $F = 0$ regardless of the value of Y
  - When $X = 1$, $F = Y$



Logic Functions using Tri-State Buffers

- Implement the following function using 3-State buffers and inverters: $F(w,x,y) = \overline{w}x + w\overline{y} + xy$
- Use w as control input:
  - When $w = 0$, $F = x$ regardless of the value of Y
  - When $w = 1$
    - If $x = 0$, $F = \overline{y}$
    - If $x = 1$, $F = 1$

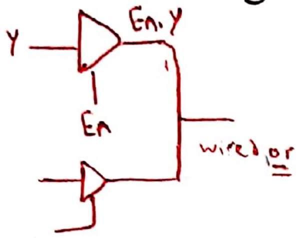
	w	x	y	F
A	0	0	0	0
	0	0	1	0
	0	1	0	1
	0	1	1	1
B C1	1	0	0	1
	1	0	1	0
Co	1	1	0	1
	1	1	1	1

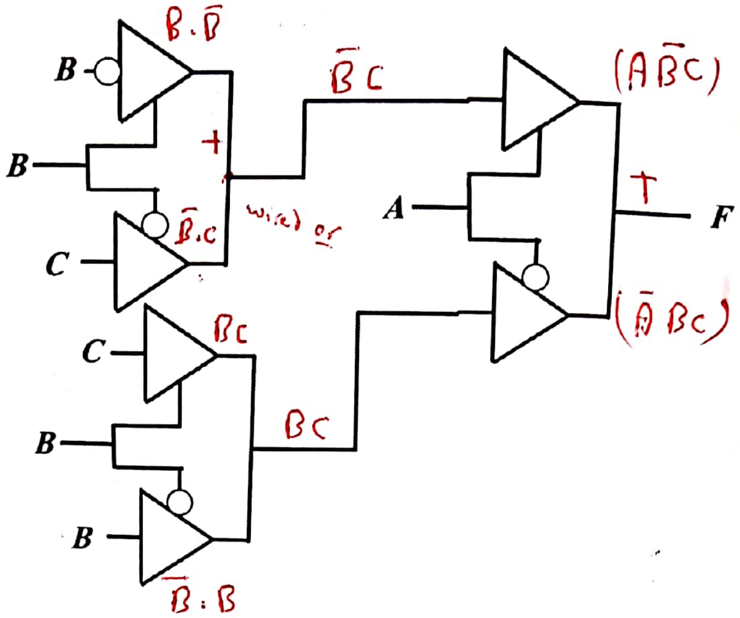


Scanned by CamScanner

Logic Functions using Tri-State Buffers

- Write the Boolean expression of $F(A,B,C)$ given the diagram below:





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

Exclusive OR/ Exclusive NOR

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

Exclusive OR/ Exclusive NOR

- Uses for the XOR and XNORs gate include:
  - Adders/subtractors/multipliers
  - Counters/incrementers/decrementers
  - Parity generators/checkers

XY	$X\oplus Y$	$X\odot Y$
00	0	1
01	1	0
10	1	0
11	0	1

- Definitions
  - The XOR function is: $X \oplus Y = \underline{\overline{X}Y + X\overline{Y}}$
  - The XNOR function is: $X \odot Y = \underline{\overline{X \oplus Y} = XY + \overline{X}\,\overline{Y}}$
- Strictly speaking, XOR and XNOR gates ***do no exist for more than two inputs***. Instead, they are replaced by odd and even functions

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

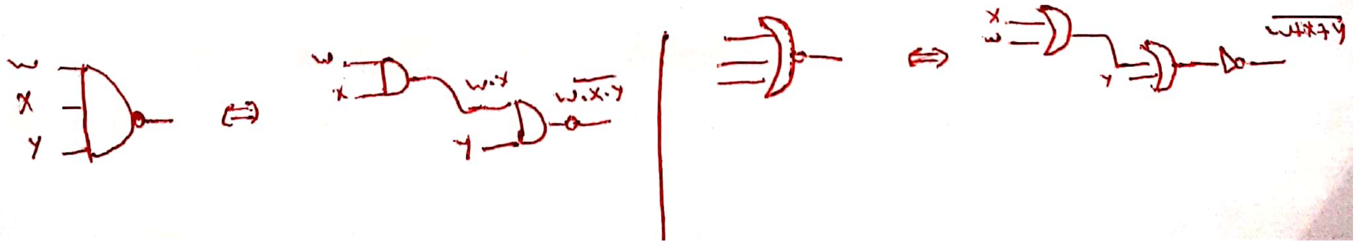
$(\bar{\bar{x}} + \bar{y}) \cdot (\bar{x}\bar{\bar{y}})$

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

$x\bar{x} + \boxed{xy + \bar{x}\bar{y}} + \bar{y} \cdot y$

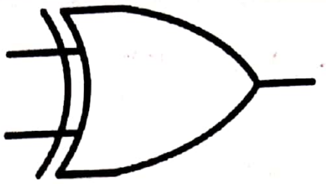
Proof: XNOR is the complement of XOR

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

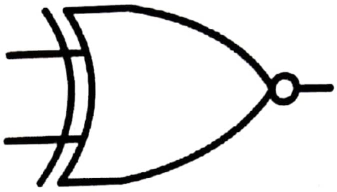


Symbols For XOR and XNOR

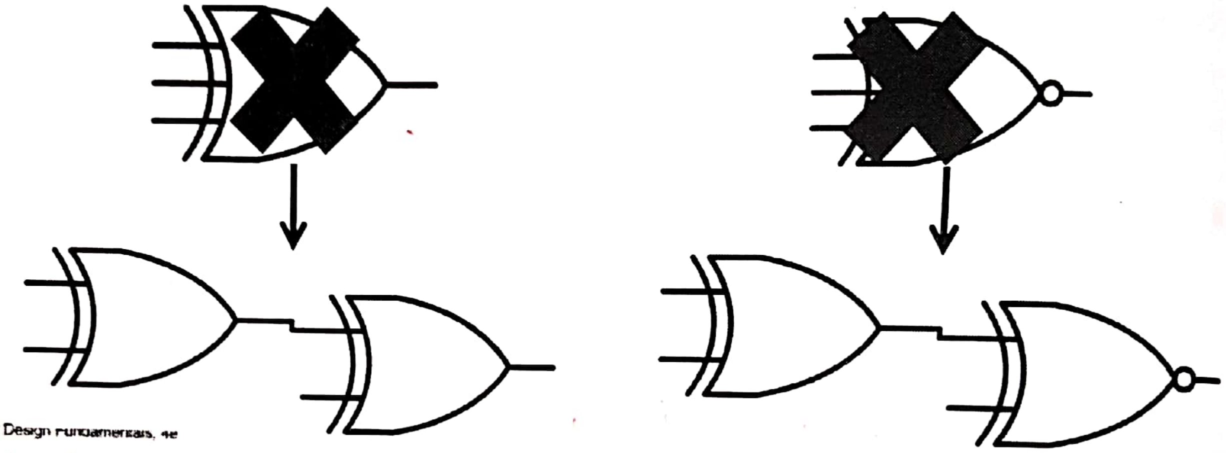
- XOR symbol:



- XNOR symbol:



- *Shaped symbols exist only for two inputs*



Truth Tables for XOR/XNOR

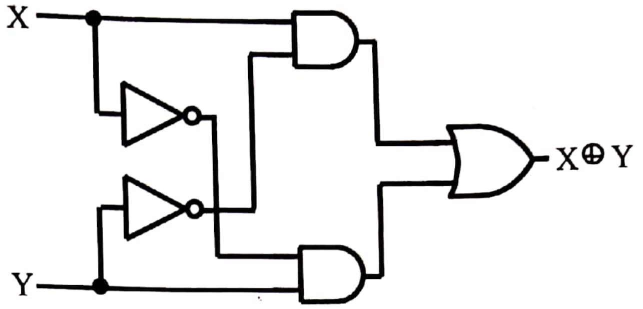
X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

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

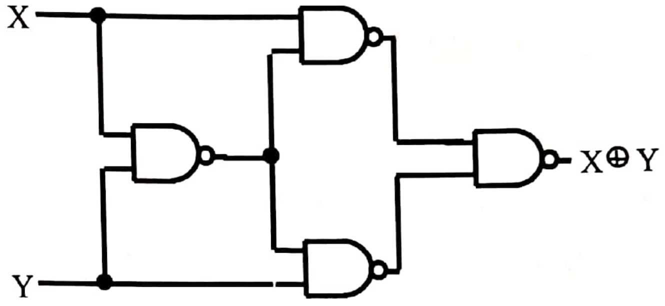
- The XOR function means: ***X OR Y, but NOT BOTH***
- Why is the XNOR function also known as the ***equivalence*** function, denoted by the operator $\equiv$?
  - Because the function equals 1 if and only if $\boldsymbol{X = Y}$

XOR Implementations

- The simple SOP implementation uses the following structure:



- A NAND only implementation is:



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

$XY + \bar{X}\bar{Y} = \bar{X}\bar{Y} + XY$

$XY + \bar{X}\bar{Y} \rightarrow X \odot Y$

$X \oplus Y = X\bar{Y} + \bar{X}Y$

XOR

عدد ال 1 فردي بعطيني 1

0	1	0	1
1	0	1	0

$F = \sum 1, 2, 4, 7$

$\bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ$

- The XOR identities:

$X \oplus 0 = X$	$X \oplus 1 = \bar{X}$
$X \oplus X = 0$	$X \oplus \bar{X} = 1$
$X \oplus \bar{Y} = \overline{X \oplus Y}$	$\bar{X} \oplus Y = \overline{X \oplus Y}$
$X \oplus Y = Y \oplus X$	
$(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z) = X \oplus Y \oplus Z$	

	XYZ	$X \oplus Y \oplus Z$
even	000	0
odd	001	1
odd	010	1
even	011	0
odd	100	1
even	101	0
even	110	0
odd	111	1

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

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

XNOR

1111 0 1111

XNOR → even → 1

XoR → odd → 0

- The XNOR identities:

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

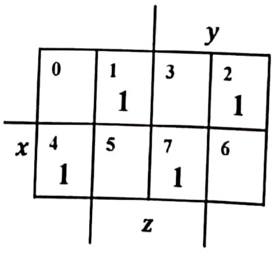
- The XNOR function can be extended to 3 or more variables. For more than 2 variables, it is called an ***even function***, not an XNOR:

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

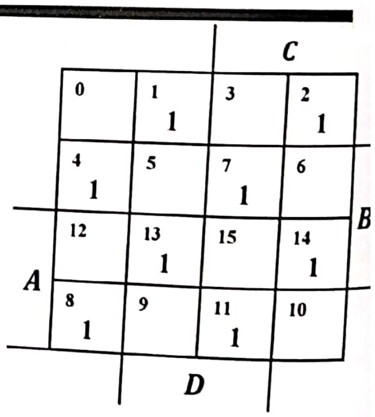
- ***The even function is the complement of the odd function***

Odd and Even Functions

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

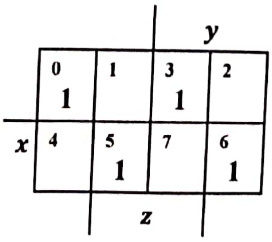


	0	1 1	3	2 1
x	4 1	5	7 1	6

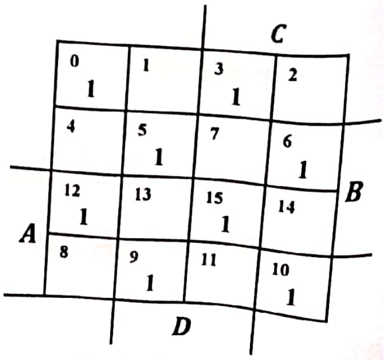


0	1 1	3	2 1
4 1	5	7 1	6
12	13 1	15	14 1
8 1	9	11 1	10

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



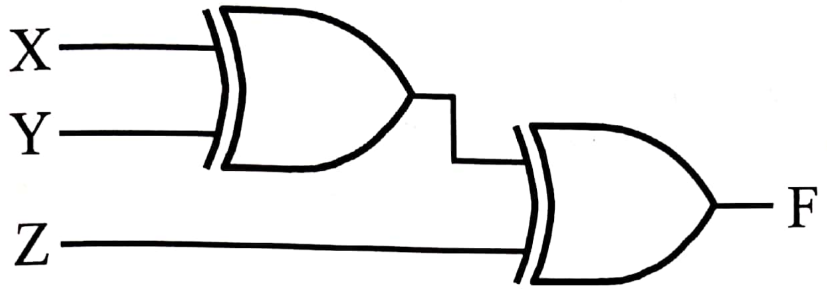
	0 1	1	3 1	2
x	4	5 1	7	6 1



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

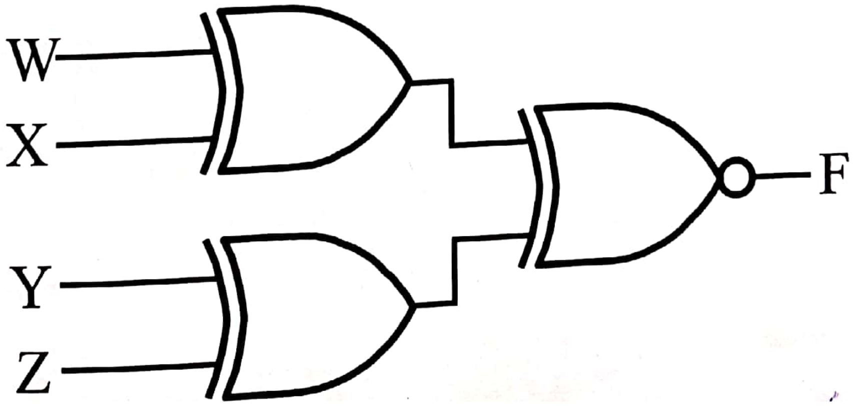
Example: Odd Function Implementation

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



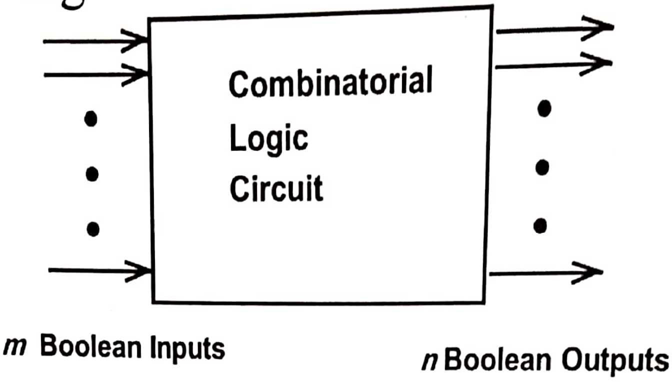
Example: Even Function Implementation

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



Combinational Circuits

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



Design Procedure

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

Design Procedure

4. **Technology Mapping**
   - Map the logic diagram to the implementation technology selected

5. **Verification**
   - Verify the correctness of the final design ***manually*** or using ***simulation***

Design Example1

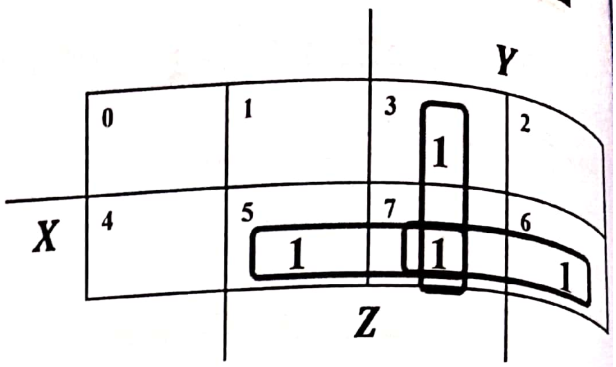
- **Specification:** Design a combinational circuit that has ***3 inputs (X, Y, Z)*** and ***one output F***, such that $F = 1$ when the number of 1's in the input is greater than the number of 0's (i.e. number of 1's ≥ 2)
  - This is called ***majority function*** (i.e. majority of inputs must be 1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

① ⓪	*X*	*Y*	*Z*	*F*
0 < 3	0	0	0	0
1 < 2	0	0	1	0
1 < 2	0	1	0	0
2 > 1	0	1	1	1
1 < 2	1	0	0	0
2 > 1	1	0	1	1
	1	1	0	1
	1	1	1	1

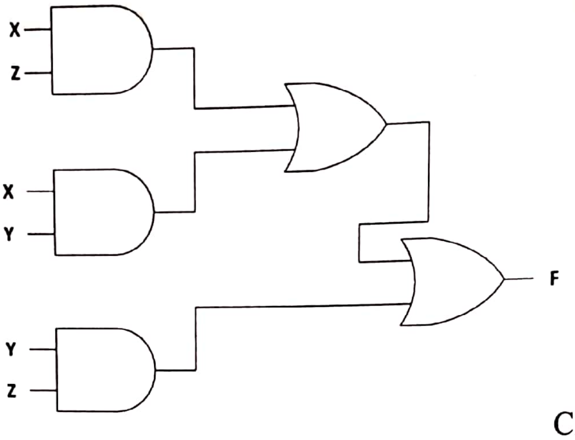
Design Example1 Cont.

- **Optimization:**

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



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



Design Example2

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

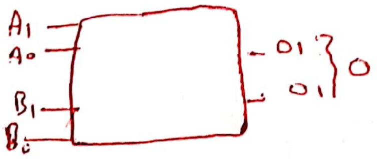
$O_1O_0 = 00$	When $A = B$ *and Both are even*
$O_1O_0 = 01$	When $A < B$
$O_1O_0 = 10$	When $A > B$
$O_1O_0 = 11$	When $A = B$ *and Both are odd*

$2^4 = 16$

لكل O إلها K-map خاصة فيها

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

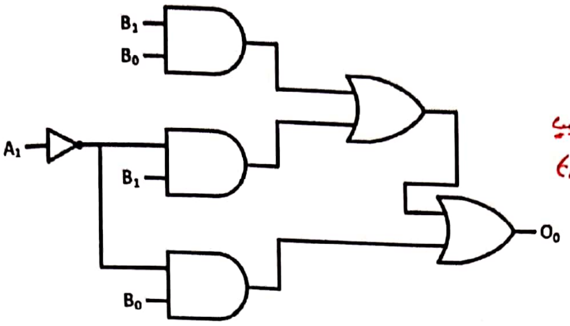
- **Formulation:**

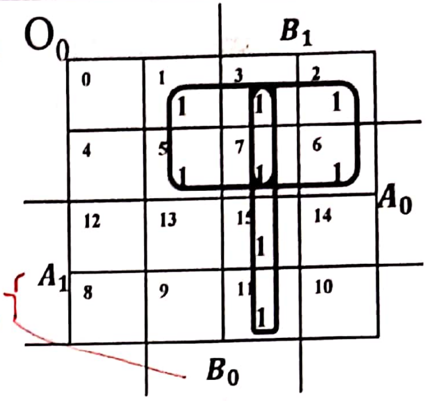


Design Example2 Cont.

- **Optimization and Technology Mapping:**

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

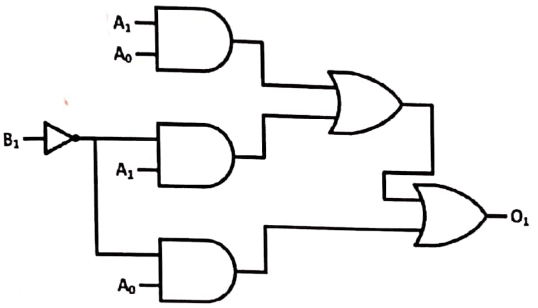


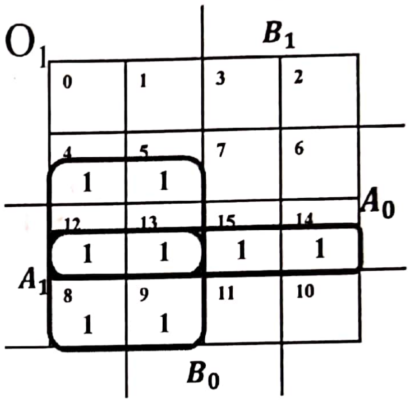


0	1 1	3 1	2 1
4	5 1	7 1	6 1
12	13	15 1	14
8	9	11 1	10

حسب الترتيب بال truth table

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



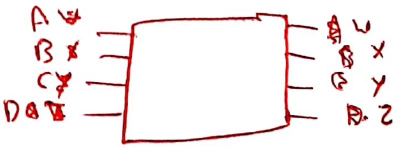


0	1	3	2
4 1	5 1	7	6
12 1	13 1	15 1	14 1
8 1	9 1	11	10

Design Example3

1. Specification

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



Design Example3 Cont.

2. Formulation

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

Valid [illegible]
BCD

Don't
Care

Design Example3 Cont.

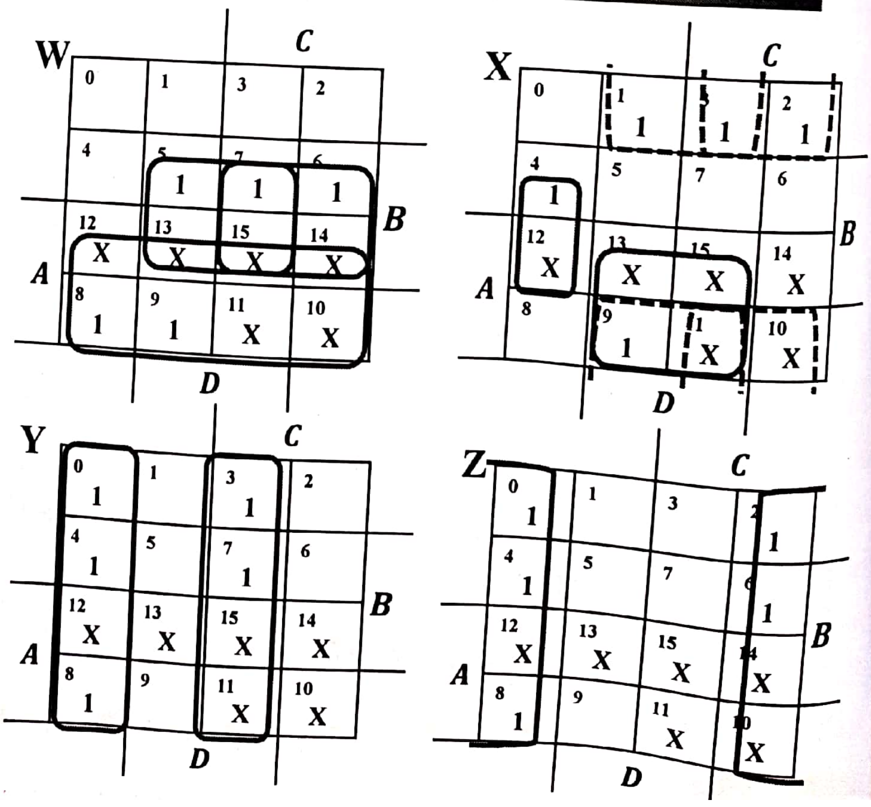
3. Optimization

$W = A + BC + BD$

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

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

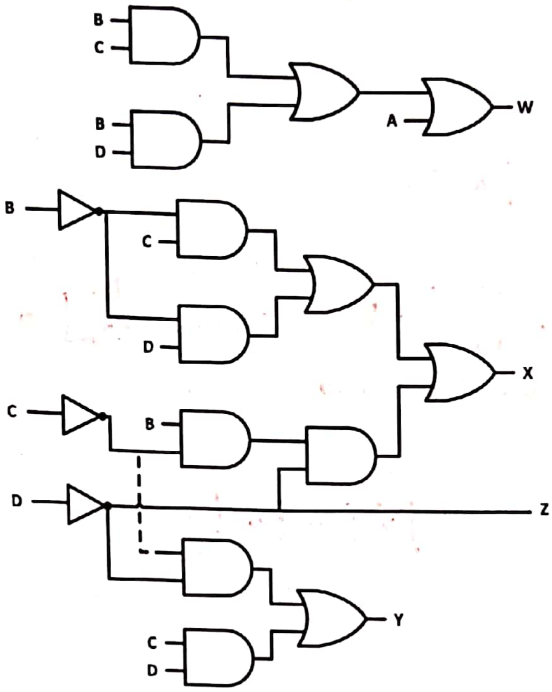
$Z = \bar{D}$



Design Example3 Cont.

4. Technology Mapping

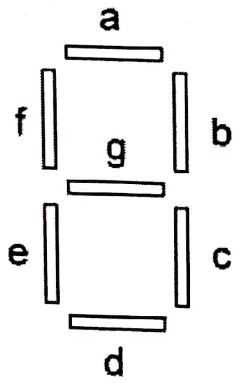
- Mapping with a library containing inverters, 2-input AND, 2-input OR



Homework: BCD to 7-Segment

مش مطلوب

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



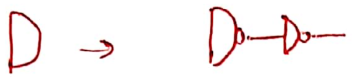
- **Formulation:**

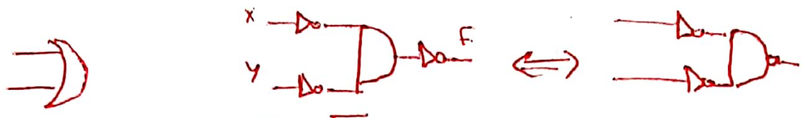
ABCD	*gfedcba*
0000	*0111111*
0001	*0000110*
1001	*1100111*
1010	*0000000*
1111	*0000000*

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

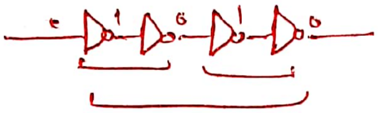
Technology Mapping

- Mapping Procedures
  - To NAND gates
  - To NOR gates





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



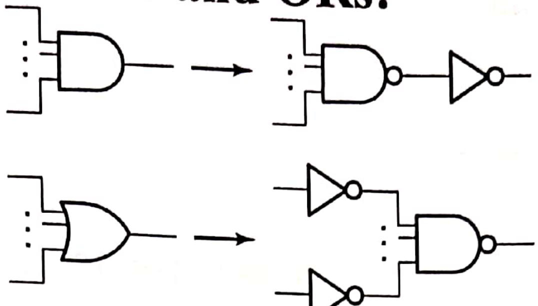
نختصرهم كلهم

Mapping to NAND gates

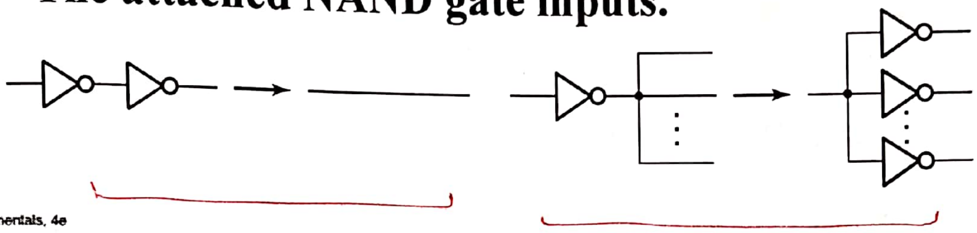
- Assumptions:
  - Gate loading and delay are ignored
  - Cell library contains an inverter and *n*-input NAND gates, *n* = 2, 3, …
  - An AND, OR, inverter schematic for the circuit is available
- The mapping is accomplished by:
  - Replacing AND and OR symbols,
  - Pushing inverters through circuit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

NAND Mapping Algorithm

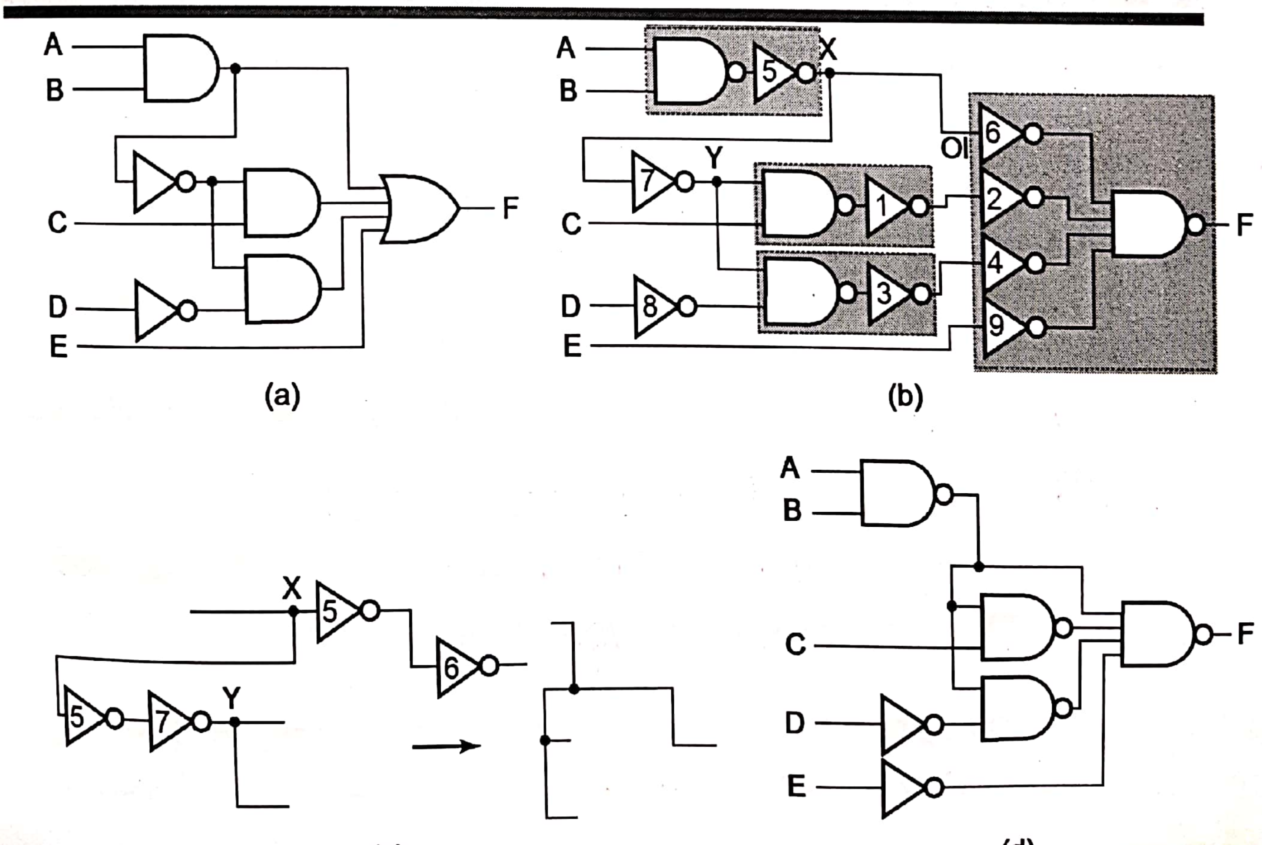
1. **Replace ANDs and ORs:**



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



NAND Mapping Example



(a) (b)

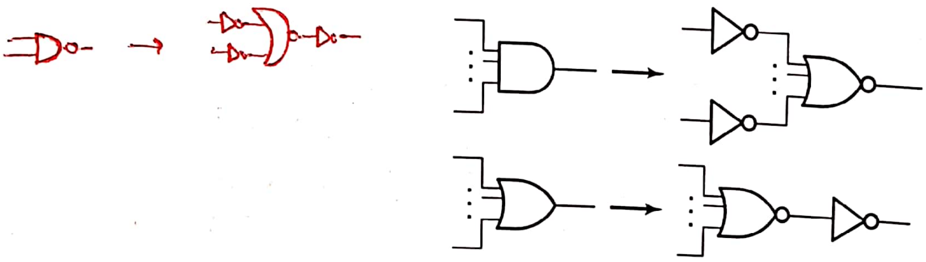
(c) (d)

Mapping to NOR gates

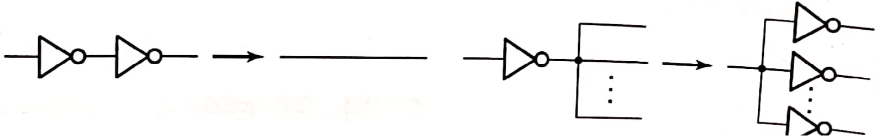
- Assumptions:
  - Gate loading and delay are ignored
  - Cell library contains an inverter and *n*-input NOR gates, *n* = 2, 3, …
  - An AND, OR, inverter schematic for the circuit is available
- The mapping is accomplished by:
  - Replacing AND and OR symbols,
  - Pushing inverters through circuit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

NOR Mapping Algorithm

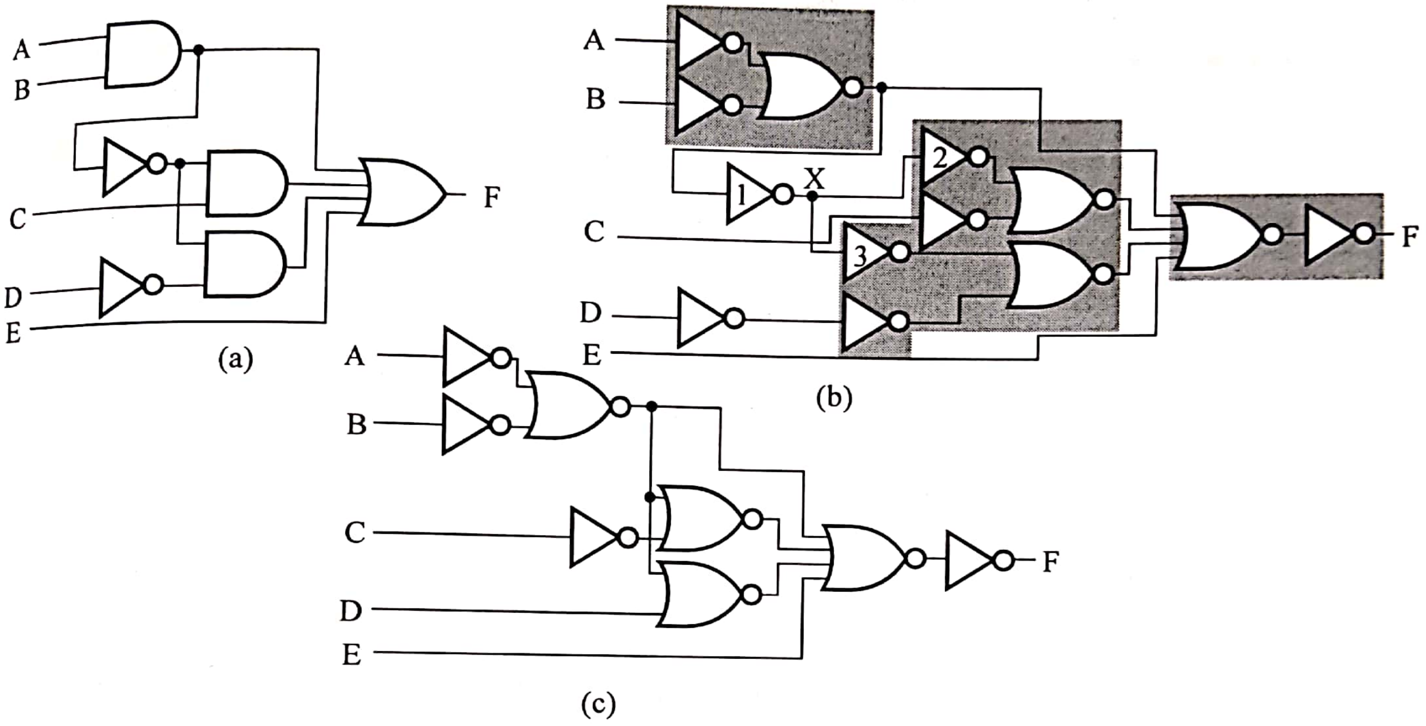
1. Replace ANDs and ORs:



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



NOR Mapping Example



(a)

(b)

(c)

Overview

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

Functions and Functional Blocks

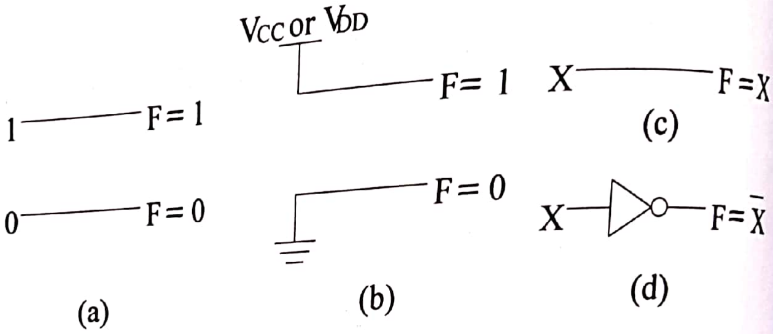
- The functions considered are those found to be very useful in design
- Corresponding to each of the functions is a combinational circuit implementation called a ***functional block***
- In the past, functional blocks were packaged as small-scale-integrated (SSI), medium-scale integrated (MSI), and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

Rudimentary Log

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

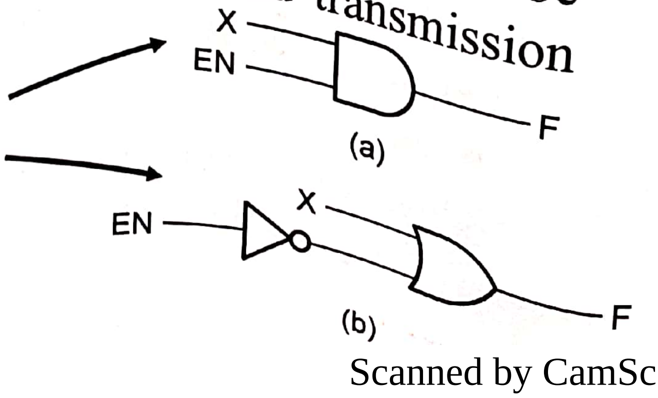
Functions of One Variable

X	$F = 0$	$F = 1$	$F = X$	$F = \overline{X}$
0	0	1	0	1
1	0	1	1	0



Enabling Function

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



Decoding

input →

most

A1,A0

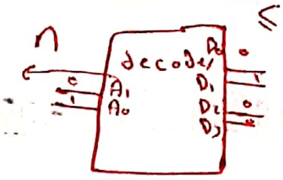
D0 ← 0 0

D1 0 1

D2 1 0

D3 1 1

بيتحط عليها 1



- ***Decoding:*** the conversion of an ***n-bit*** input code to an ***m-bit*** output code with **n ≤ m ≤ 2ⁿ** such that each valid code word produces a unique output code

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

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

1-to-2 Line Decoder

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

- Only one output is active at a time

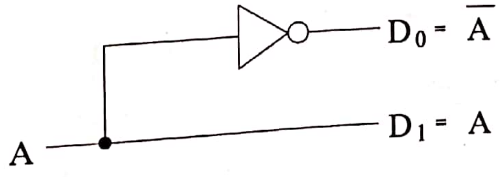
$D_0 = \bar{A}_0$

$D_1 = A_0$

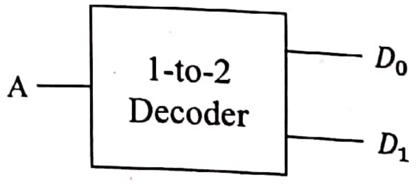
A	D_0	D_1
0	1	0
1	0	1

(a)

معناه إنها عليه 1



(b)

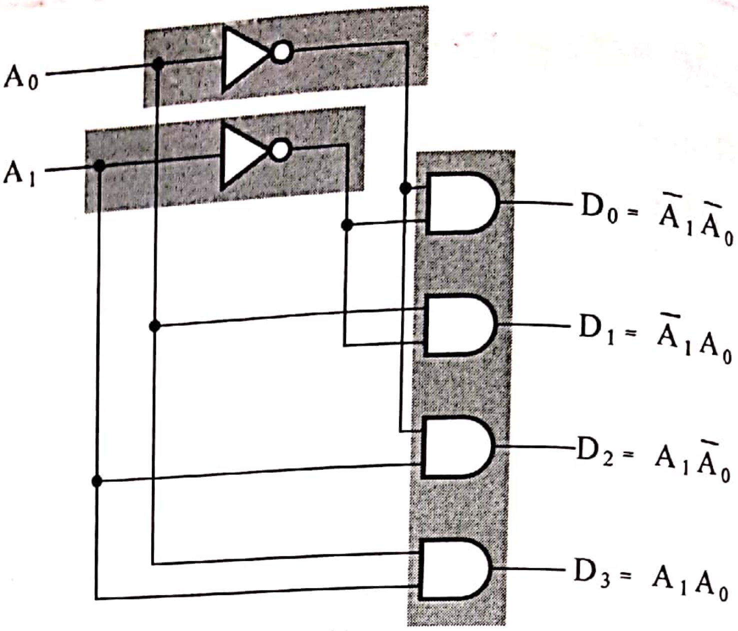


(c)

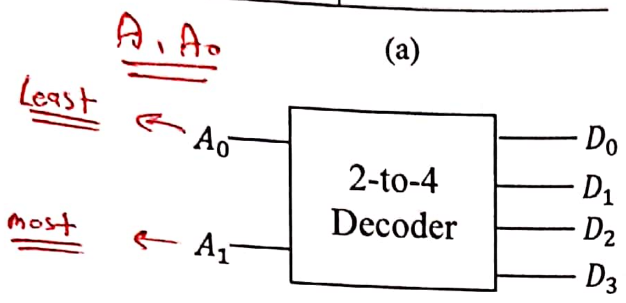
- Decoders are used to control multiple circuits by enabling only one of them at a time

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

(a)

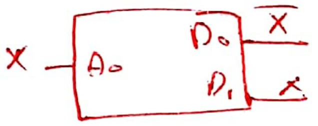


(b)

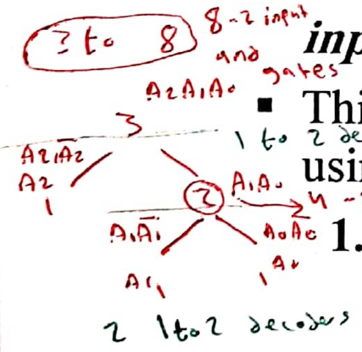


(c)

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



Decoder Expansion



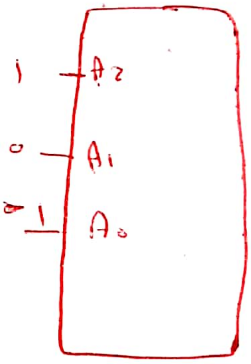
- General procedure given in book for any decoder with *n inputs and 2^n outputs*
- This procedure builds a decoder backward from the outputs using

A_1A_0
$A_1\bar{A}_0$
$\bar{A}_1A_0$
$\bar{A}_1\bar{A}_0$

1. **Let k = n**
2. **We need 2^k 2-input AND gates driven as follows:**
   - **If k is even, drive the gates using two k/2-to-$2^{k/2}$ decoders**
   - **If k is odd, drive the gates using one (k+1)/2-to-$2^{(k+1)/2}$ decoder and one (k-1)/2-to-$2^{(k-1)/2}$ decoder**
3. **For each decoder resulting from step2, repeat step2 until k = 1. For k = 1, use 1-to-2 decoder**

Decoder Expansion - Example 1

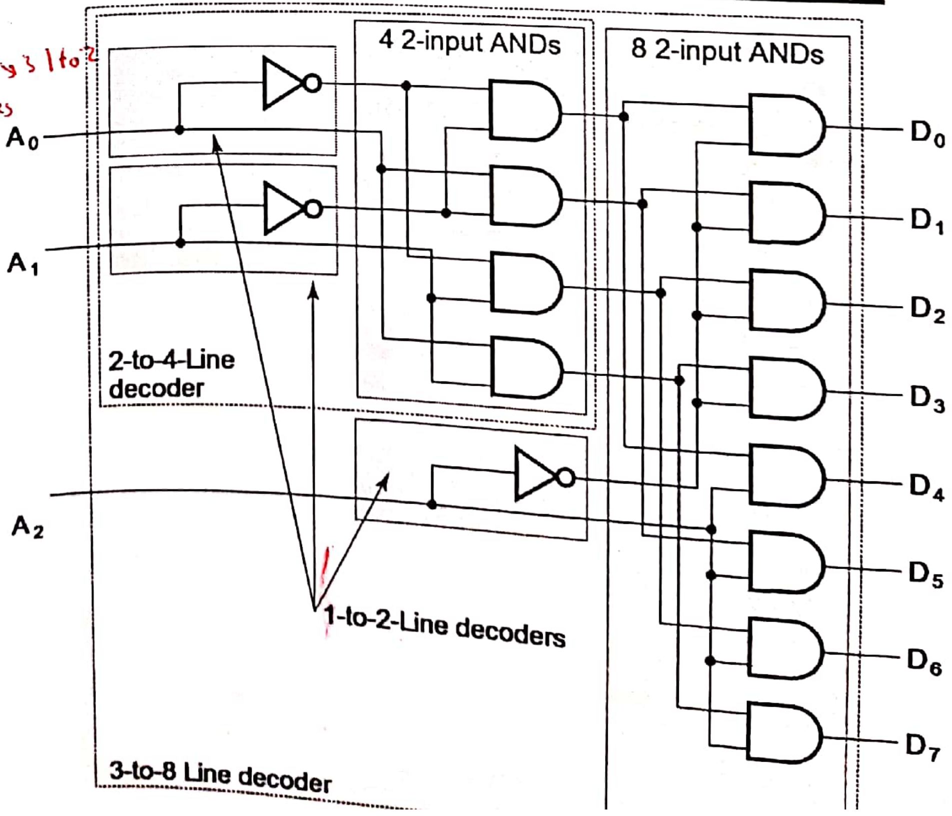
- 3-to-8-line decoder
  - $k = n = 3$
  - We need 2^3(8) 2-input AND gates driven as follows:
  - k is odd, so split to:
    - 2-to-4-line decoder
    - 1-to-2-line decoder
  - 2-to-4-line decoder → $k = n = 2$
    - We need 2^2(4) 2-input AND gates driven as follows:
    - k is even, so split to:
      - Two 1-to-2-line decoder
- See next slide for result



Decoder Expansion - Example 1

8 - 2 input and gates

- $GN = \underline{8} \times 2 + \underline{4} \times 2 + 3$ 3 1to2
- $GN = 27$ 4 2 input And gates
- **Straight forward design has the same GN cost**



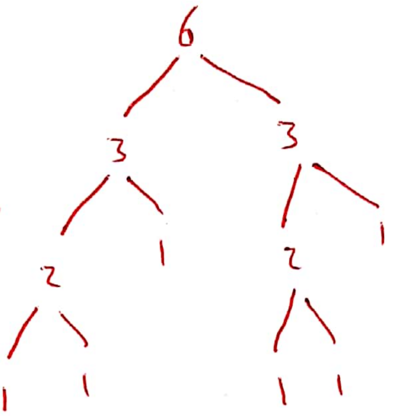
- 6-to-64-line decoder
  - $k = n = 6$
  - We need $2^6(64)$ 2-input AND gates driven as follows:
  - k is even, so split to:
    - Two 3-to-8-line decoders
  - Each 3-to-8-line decoder is designed as shown in Example 1

64 and gat

16 and gate

8 2 input and gate

4 1to 2



128

32

2 1 to 2

2

16

4

182

88 2 input and gate

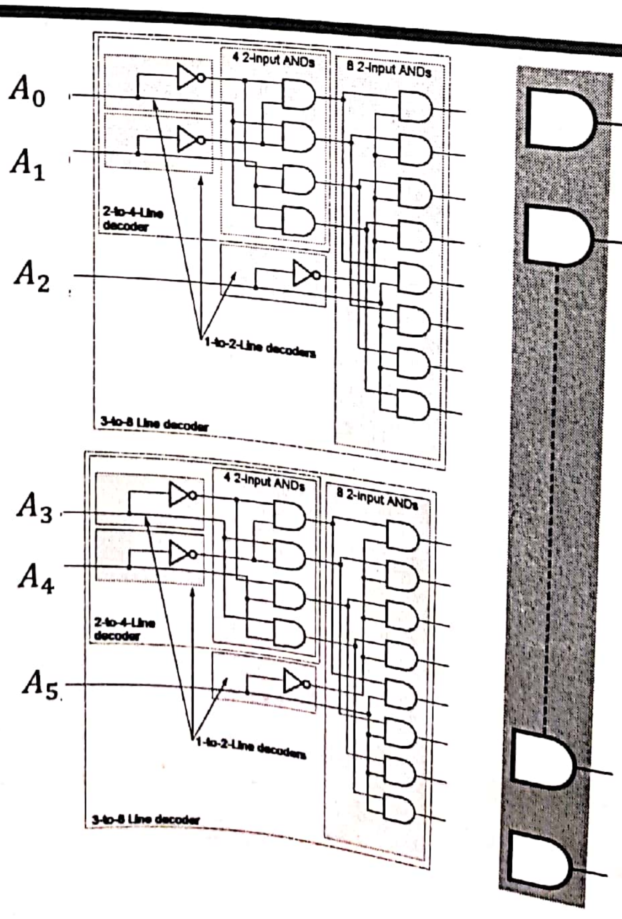
6 1 to 2

Decoder Expansion - Example 2

- $GN = 64 \times 2 + 16 \times 2 + 8 \times 2 + 6$
- $GN = 182$
- Straight forward design has GN cost of 390

↓

$64 \times 6 + 6$



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

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

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

$D_{63} = A_5A_4A_3A_2A_1A_0$

Decoder Expansion - Example 3

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

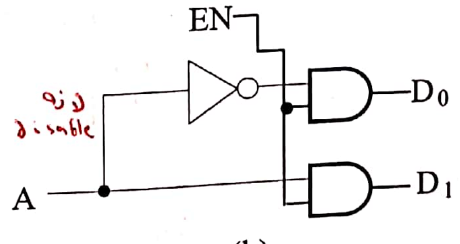
Building Larger Decoders

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

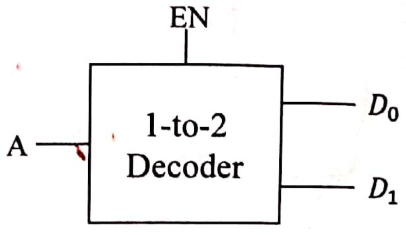
EN	A	D_0	D_1
0	X	0	0
1	0	1	0
1	1	0	1

⊃ [illegible] disable

(a)



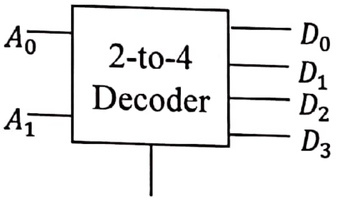
(b)



(c)

2-to-4 Line Decoder with Enable

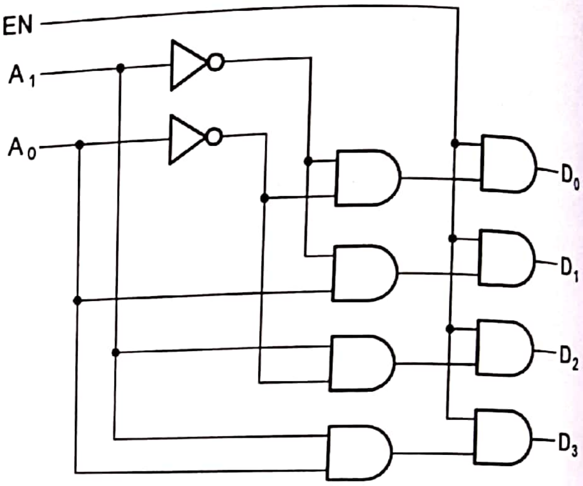
- Attach ***4-enabling*** circuits to the outputs
- See truth table below for function
  - Combination containing two X's represent four binary combinations
- Alternatively, can be viewed as distributing value of signal EN to 1 of 4 outputs
  - In this case, it is called a *Demultiplexer*



(c)

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

(a)



(b)

n 2input and gates ⊕ اذا طلب ال Cost لازم اضيف

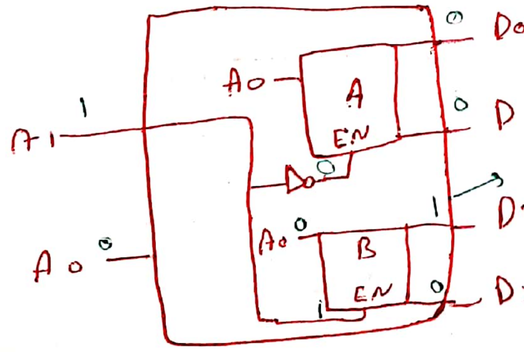
2-to-4 Decoder using 1-to-2 Decoders and Inverters

ال most دايما احسن نستخدمه لل ~~[illegible]~~ Enable

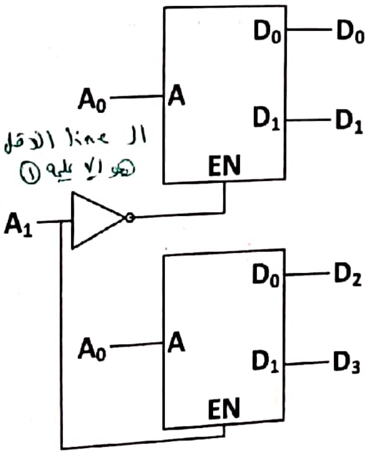
A_1	A_0	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

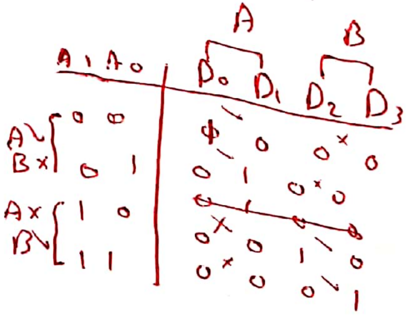
1st 1-to-2 Decoder

2nd 1-to-2 Decoder



ال line الاول هو الي عليه ①



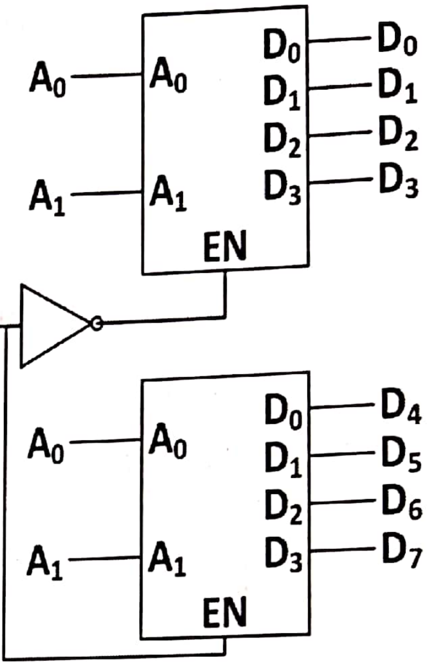


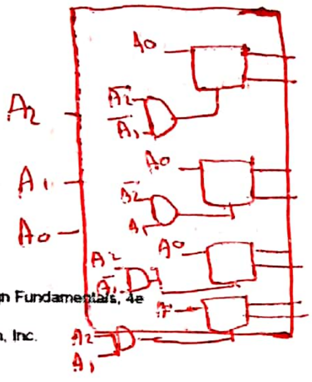
3-to-8 Decoder using 2-to-4 Decoders and Inverters

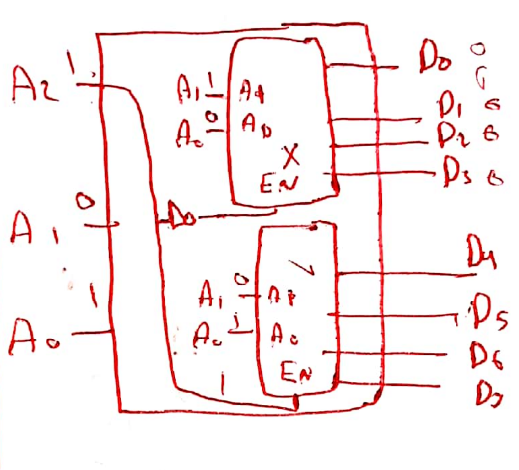
A_2	A_1	A_0	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

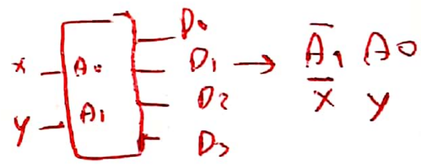
1st 2-to4 Decoder

2nd 2-to4 Decoder

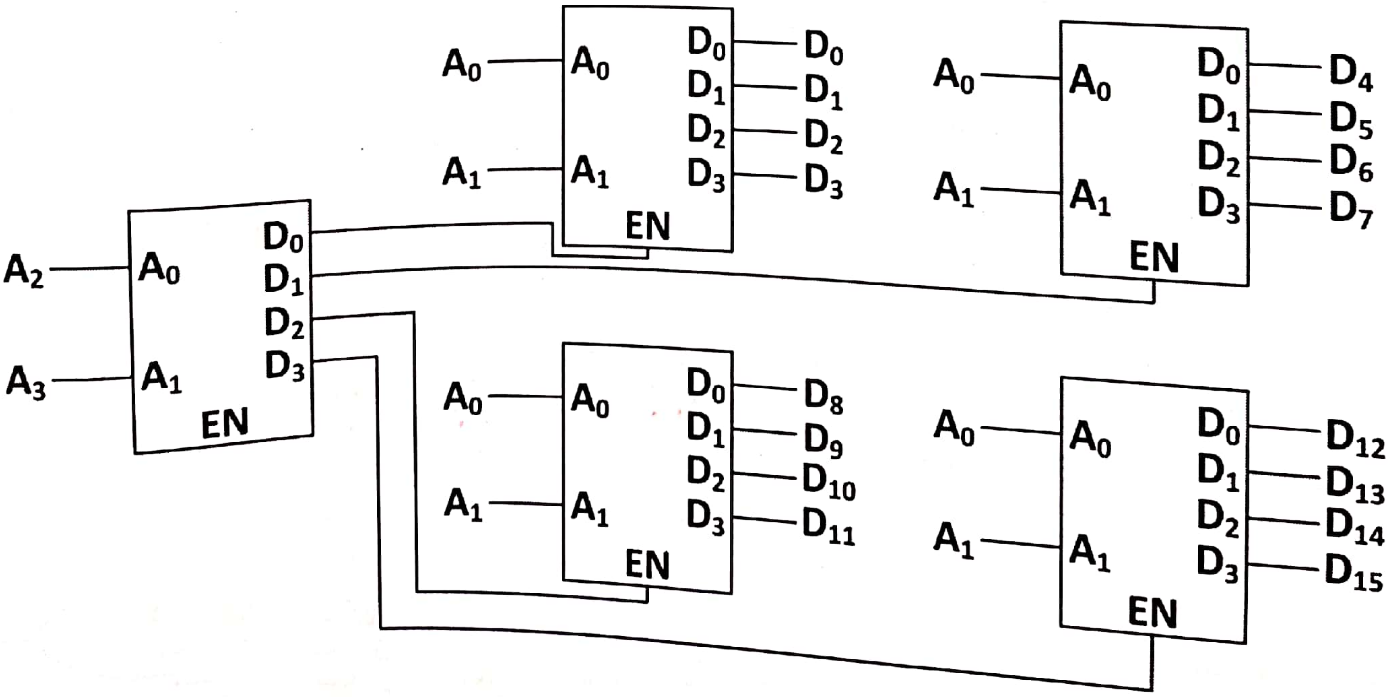








4-to-16 Decoder using Only 2-to-4 Decoders



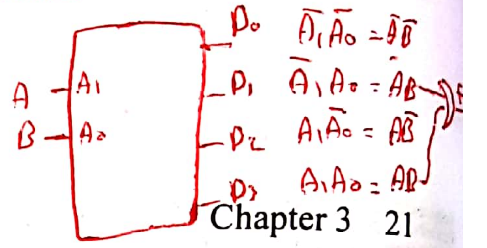
- Decoder and OR Gates

- Implement *m* functions of *n* variables with:
  - Sum-of-minterms expressions
  - One *n*-to-2^n-line decoder
  - *m* OR gates, one for each function
  - For each function, the OR gate has *k* inputs, where *k* is the number of minterms in the function

- **Approach 1:** لازم يكون الInput تاع الDecoder
نفس عدد الInput للFunction
  - Find the truth table for the functions
  - Make a connection to the corresponding OR from the corresponding decoder output wherever a 1 appears in the truth table

$\bar{A}B + AB$

$F(A,B) = \Sigma 1,3$



A
B
A_1
A_0
D_0 $\bar{A}_1\bar{A}_0 = \bar{A}\bar{B}$
D_1 $\bar{A}_1A_0 = \bar{A}B$
D_2 $A_1\bar{A}_0 = A\bar{B}$
D_3 $A_1A_0 = AB$
F

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

للتاكد

AB	F
00	0
01	1
10	0
11	1

Example1

- Implement function *f* using decoder and OR gate:

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

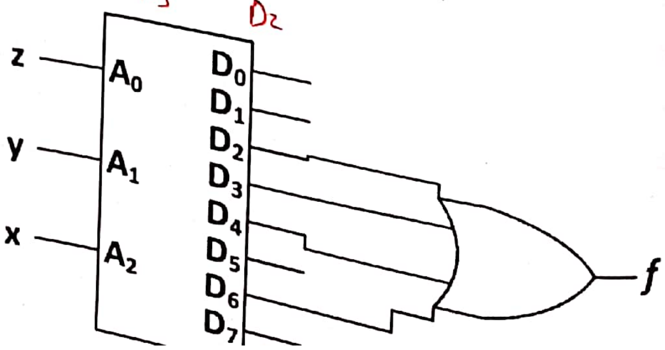
- $n = 3$ variables → ***3-to-8 decoder***
- One function → ***One OR gate***
- Solution: Convert *f* to $\boxed{\text{SOM}}$ format

$\bullet f = x\bar{z}(y+\bar{y}) + \bar{x}y(z+\bar{z}) = xy\bar{z} + x\bar{y}\bar{z} + \bar{x}yz + \bar{x}y\bar{z}$

110 D6 100 D4 011 D3 010 D2

$\bullet f(x,y,z) = \Sigma_m(2,3,4,6)$ → ***4-input OR gate***

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

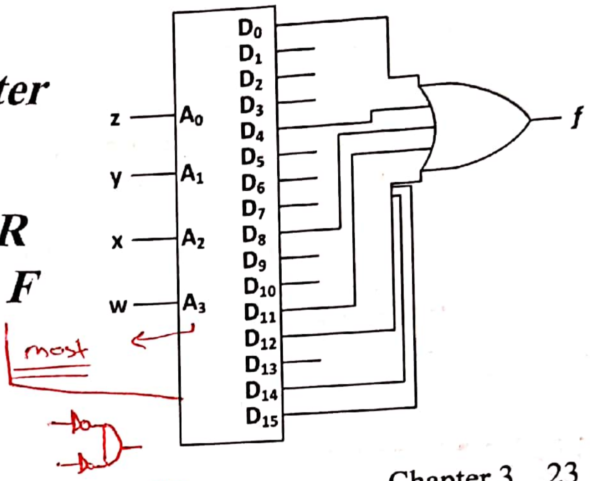


Example2

- Implement function f using decoder and OR gate:

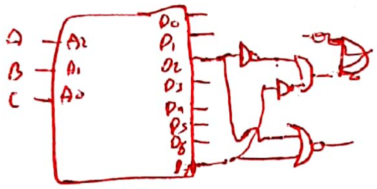
$$f(w,x,y,z) = \sum_{m}(0,4,8,11,12,14,15)$$

- $n = 4$ variables → ***4-to-16 decoder***
- One function with 7 minterms → ***One 7-input OR gate***



- ***If number of minterms is greater than $\frac{2^n}{2}$, then design for complement F ($\overline{\text{F}}$) and use NOR gate instead of OR to generate F***

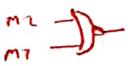
$F(A,B,C) = \Sigma(0,1,3,4,5,6)$



$\overline{m_2}\cdot\overline{m_7}$

$M_2 \cdot M_7$

$= \Pi(2,7)$



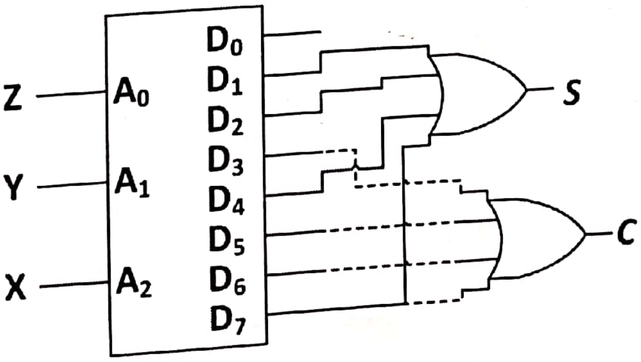
$= \overline{m_2+m_7}$

$\overline{m_2}\cdot\overline{m_7}$

Example3

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

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



Example4

A7 A5 A4
0 0 1
0 1 0
1 0 0
1 1 1

A7 A6 A5 A4
0 0 0 1
0 1 0 1
0 0 1 0
1 0 0 0
1 0 1 1

نفس الاني
P2 د

A7 A6 A4
0 0 1
0 1 0
1 0 0
1 1 1

- Implement the following set of odd parity functions of (A_7, A_6, A_5, A_4)

$P_1 = A_7 \oplus A_5 \oplus A_4$

$P_2 = A_7 \oplus A_6 \oplus A_4$

$P_3 = A_7 \oplus A_6 \oplus A_5$

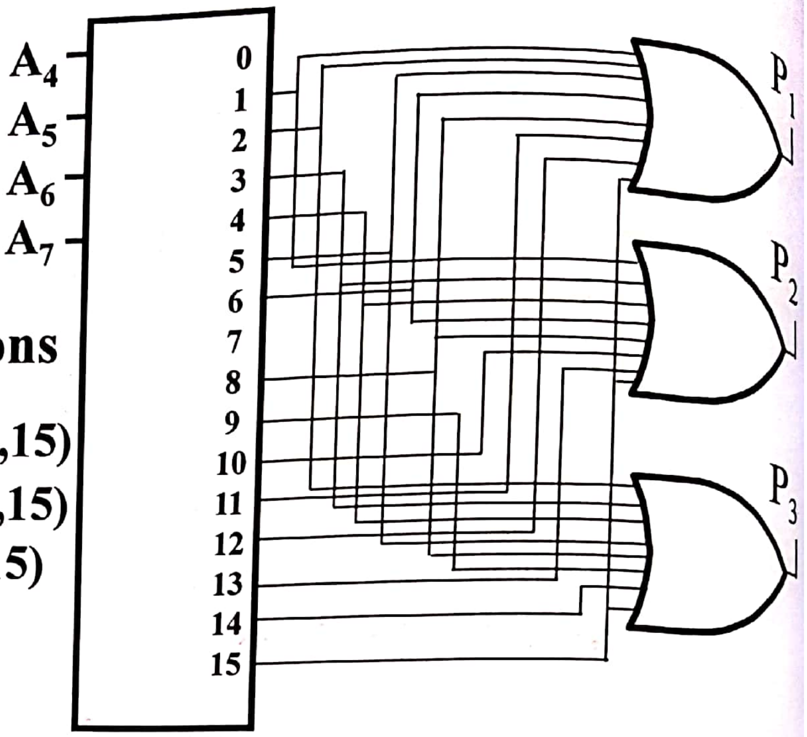
- Finding sum of minterms expressions

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

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

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

- Find circuit
- Is this a good idea?

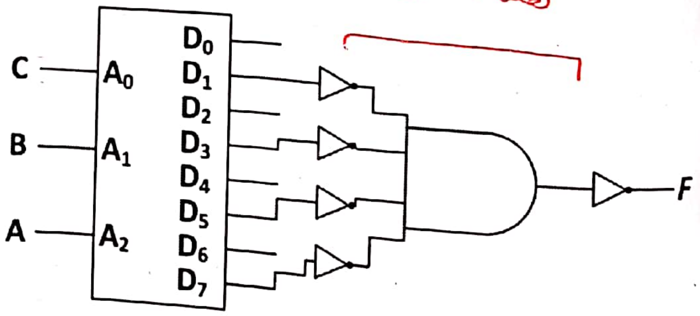


Example5

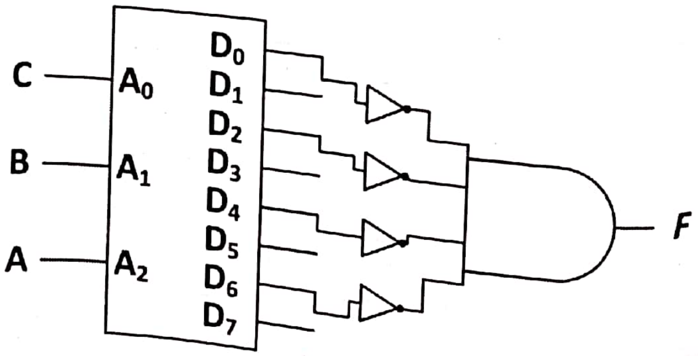
- Implement function F using 3-to-8 decoder, AND gate and inverters: $F(A,B,C) = \Sigma_m(1,3,5,7)$

نفس ال Or

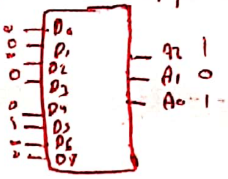
- Solution with 5 inverters:



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



2 to 1 Encoder فيه 1



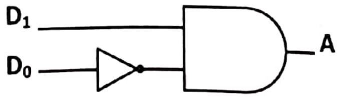
Encoding

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

2-to-1 Encoder & 4-to-2 Encoder

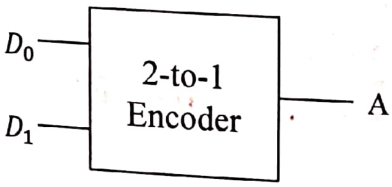
D_1	D_0	A
0	0	Invalid Input
0	1	0
1	0	1
1	1	Invalid Input

(a)



$A = D_1.\overline{D_0}$

(b)



		D_0
	X	0
D_1	①	X

(c)

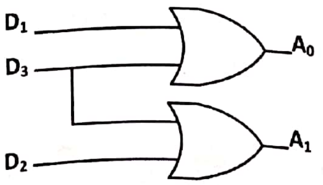
$A_1 = \overline{D_3} D_2 \overline{D_1} \overline{D_0} + D_3 \overline{D_2} \overline{D_1} \overline{D_0}$

$D_2 + D_3$

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

(a)

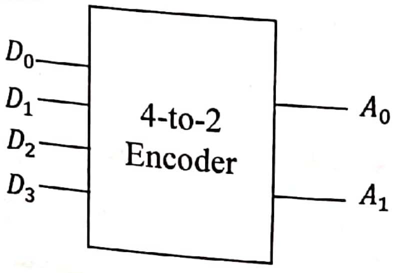
اعطاني ①
لما كان ال ا ع D_2 و D_3



$A_0 = D_1 + D_3$

$A_1 = D_2 + D_3$

(b)

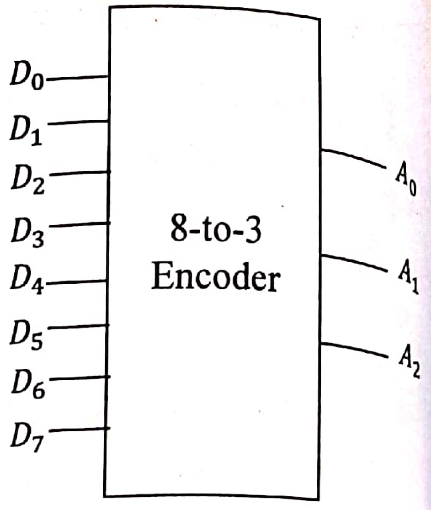


(c)

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

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

(a)



(c)

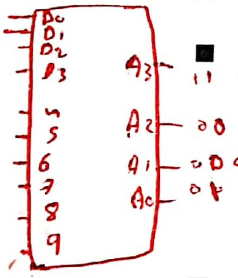
$$A_0 = D_1 + D_3 + D_5 + D_7$$
$$A_1 = D_2 + D_3 + D_6 + D_7$$
$$A_2 = D_4 + D_5 + D_6 + D_7$$

متى بحطهن

$\boxed{1}$

(b)

Decimal-to-BCD Encoder



- ***Inputs:*** 10 bits corresponding to decimal digits 0 through 9, (D_0, …, D_9)

$A_3 = D_8 + D_9$

- ***Outputs:*** 4 bits with BCD codes (A_3, A_2, A_1, A_0)

- ***Function:*** If input bit D_i is a 1, then the output is the BCD code for i

- The truth table could be formed, but alternatively, the equations for each of the four outputs can be obtained directly

Decimal-to-BCD Encoder Cont.

- Input D_i is a term in equation A_j if bit A_j is 1 in the binary value for i

- Equations:

$$A_3 = D_8 + D_9$$
$$A_2 = D_4 + D_5 + D_6 + D_7$$
$$A_1 = D_2 + D_3 + D_6 + D_7$$
$$A_0 = D_1 + D_3 + D_5 + D_7 + D_9$$

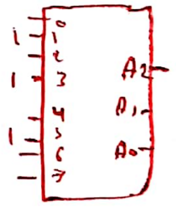
- What happens if two inputs are high simultaneously?
  - For example if D_3 and D_6 are high, then the output is 0111 which indicates that only D_7 is high ???
  - ***Solution: Establish input priority***

Priority Encoder

- If more than one input value is 1, then the encoder just designed does not work

- One encoder that can accept all possible combinations of input values and produce a meaningful result is a ***priority encoder***

- Among the 1s that appear, it selects the most significant input position (or the least significant input position) containing a 1 and responds with the corresponding binary code for that position



High 101

low 001

يعطي الاولوية
لل line
الاقل

  - ***High priority encoder:*** gives priority for the input whose value is 1 and has the highest subscript

بالعكس

  - ***low priority encoder:*** gives priority for the input whose value is 1 and has the lowest subscript

- If all inputs are 0's, what happens?
  - Define an output (V) to encode whether the input is valid or not
  - When all inputs are 0's, V is set to 0 indicating that the input is invalid, otherwise V is set to 1

4-to-2 Low Priority Encoder

لما يكونوا كلهم 0

عدد X
$2^{\#X}$

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

لو اول 1 عند D_0

$\overline{D_0}\,\overline{D_1}D_2$ $\overline{D_0}\,\overline{D_1}\,\overline{D_2}D_3$ (a)

Number of Minterms per Row $= 2^{\#\ of\ don't\ cares}$

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

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

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

(b)

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

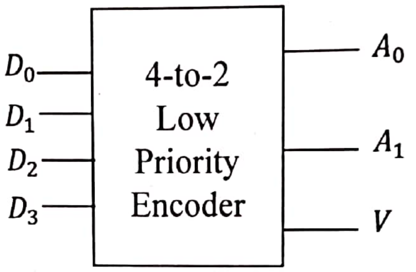
$= 0 \cdot \overline{y} \cdot \overline{x} + 1 \cdot \overline{y} \cdot \overline{x}$

x
y
0
1

وكان بسأل كيف يطلع A_1 ← $\boxed{1}$

A_0 0

A_1 1 ← مابترتبط لانه مستحيل تطلع $\boxed{1}$ من D_2 ← $\boxed{0}$



(c)

ف بترتبط لما يكونوا 1

0
0
0
1

$\boxed{1}$ A_0
← $\boxed{1}$ A_1 → هيك او بنحلها

4-to-2 High Priority Encoder

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

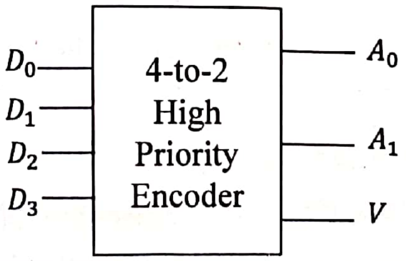
(a)

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

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

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

(b)



(c)

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

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

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

5-input Priority Encoder Cont.

- **Could use a K-map to get equations, but can be read directly from table and manually optimized if careful:**

$$A_2 = D_4$$

$$A_1 = \overline{D}_4 D_3 + \overline{D}_4\overline{D}_3 D_2 = \overline{D}_4(D_3 + D_2)$$

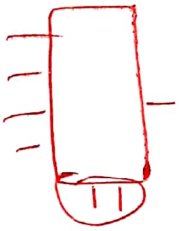
$$A_1 = \overline{D}_4 D_3 + \overline{D}_4 D_2$$

$$A_0 = \overline{D}_4 D_3 + \overline{D}_4\overline{D}_3\overline{D}_2 D_1 = \overline{D}_4(D_3 + \overline{D}_2 D1)$$

$$A_0 = \overline{D}_4 D_3 + \overline{D}_4\overline{D}_2 D_1$$

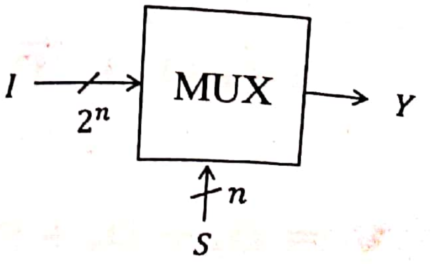
$$V = D_4 + D_3 + D_2 + D_1 + D_0$$

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



Multiplexers (MUX) (Data Selectors)

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



2-to-1-Line MUX

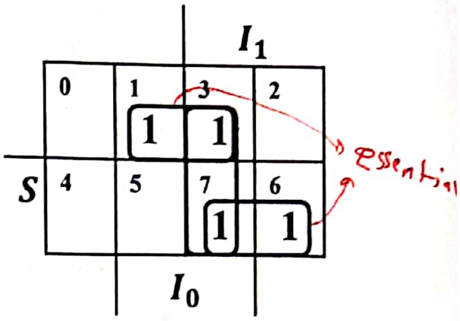
- Since $2 = 2^1$, n = 1
- The single selection variable S has two values:
  - S = 0 selects input I_0
  - S = 1 selects input I_1
- The equation:

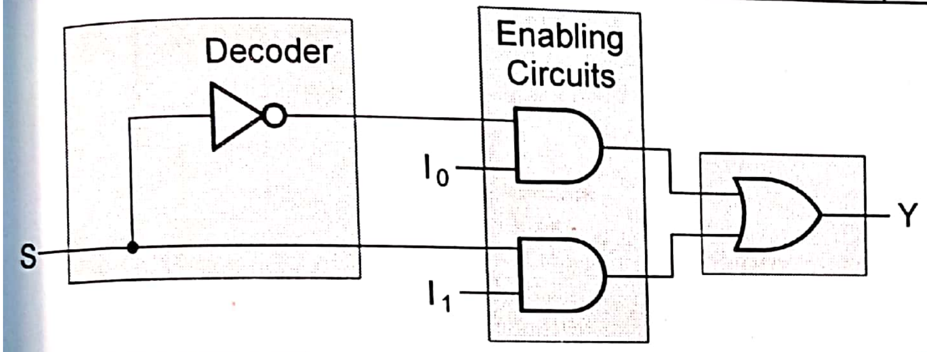
  $Y = \bar{S}I_0 + SI_1$

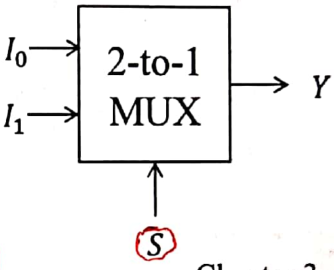
  بنطلعها من K map
- The circuit:

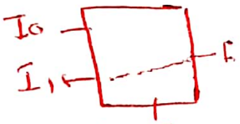
S	I_1	I_0	Y	
0	0	0	0	$Y = I_0$
0	0	1	1	
0	1	0	0	
0	1	1	1	
1	0	0	0	$Y = I_1$
1	0	1	0	
1	1	0	1	
1	1	1	1	

S	Y
0	I_0
1	I_1

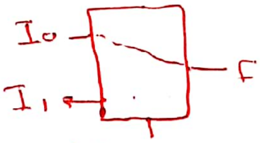








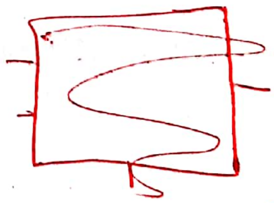
لما يكون S ← 1



لما يكون S ← 0

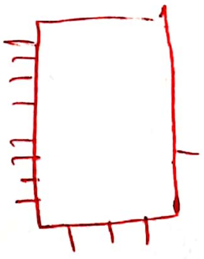
2-to-1-Line MUX Cont.

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



8 to 1

2^3 to 1



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

3 to 8 decode

8 2 input AND

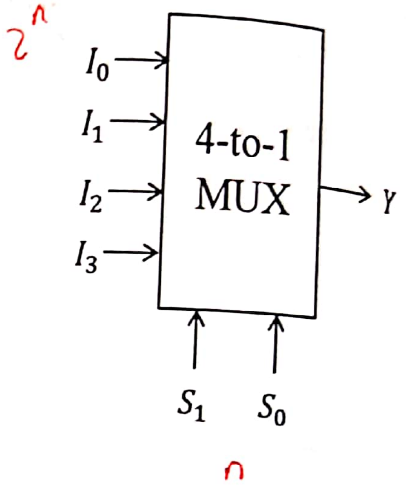
one 8 input OR gate

4-to-1-Line MUX

- Since $4 = 2^2$, n = 2
- There are two selection variables (S_1S_0) and they have four values:
  - S_1S_0 = 00 selects input I_0
  - S_1S_0 = 01 selects input I_1
  - S_1S_0 = 10 selects input I_2
  - S_1S_0 = 11 selects input I_3
- The equation:

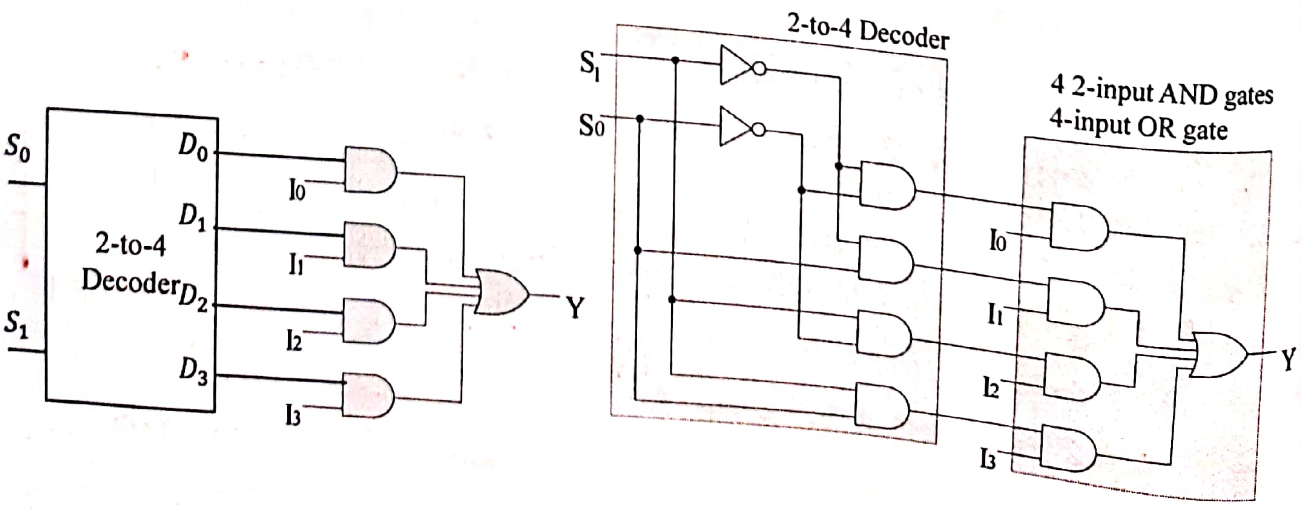
$$Y = \overline{S_1}\,\overline{S_0} I_0 + \overline{S_1}\, S_0 I_1 + S_1 \overline{S_0}\, I_2 + S_1 S_0 I_3$$

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



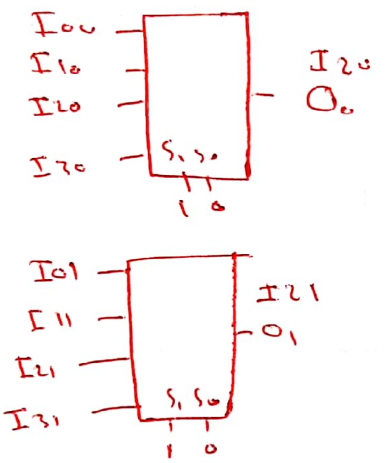
4-to-1-line MUX Cont.

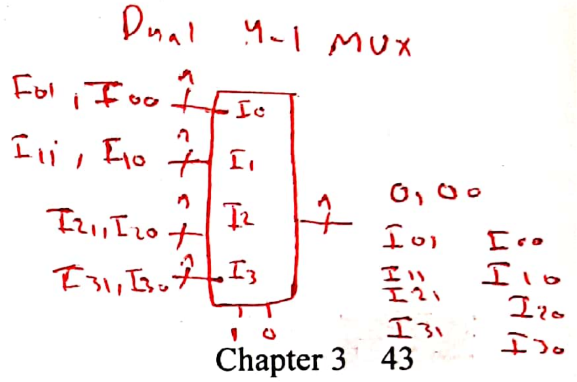
- 2-to-4-line decoder
- 4 2-input AND gates
- 4-input OR gate



Homework

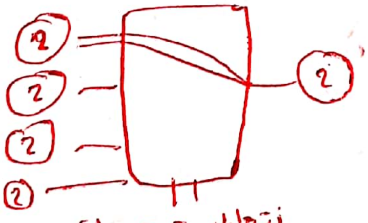
- Implement 8-to-1-Line MUX and 64-to-1 MUX: 2^6 to 1
  - How many select lines are needed? 6
  - Decoder size? 6 to 64
  - How many 2-input AND gates are needed? 64
  - What is the size of the OR gate? 64



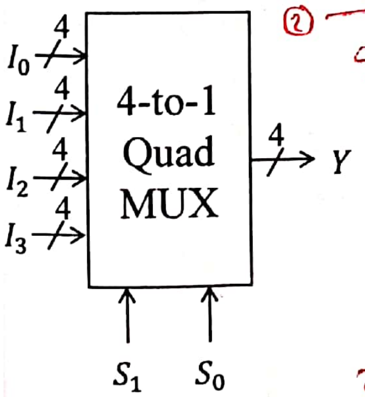


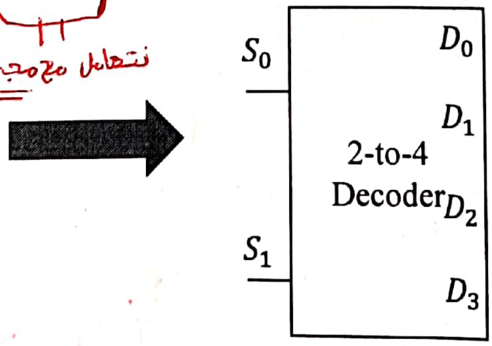
Multiplexer Width Expansion

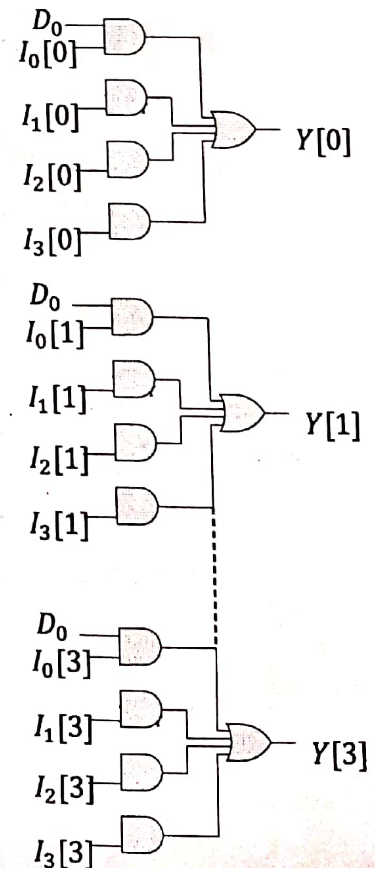
- Select "vectors of bits" instead of "bits"
- Example: *4-to-1-line quad multiplexer*



نتعامل مع مجموعات



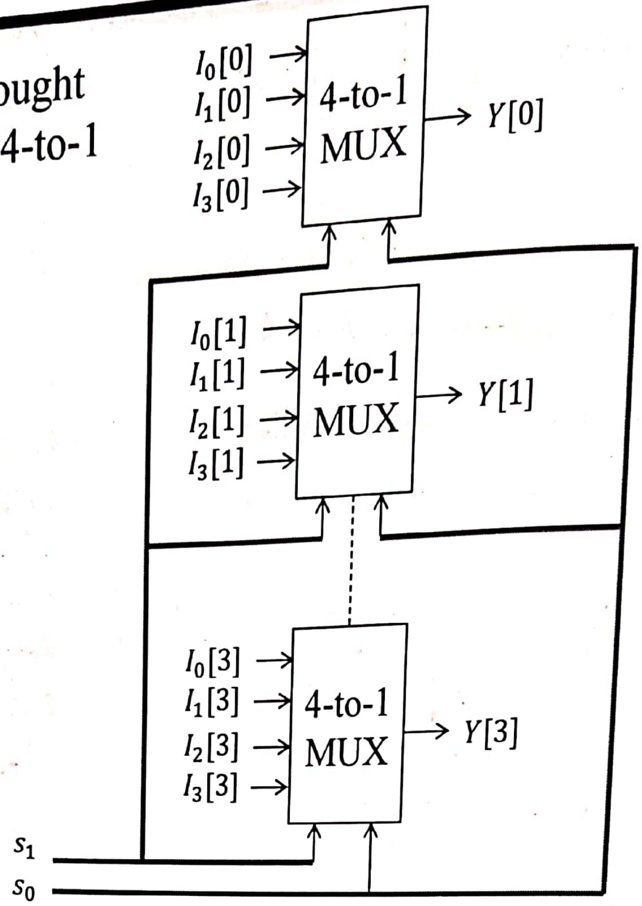




2 bits Dual 4 to 1 Mux
4 bits Quad 4 to 1 MUX

Multiplexer Width Expansion Cont.

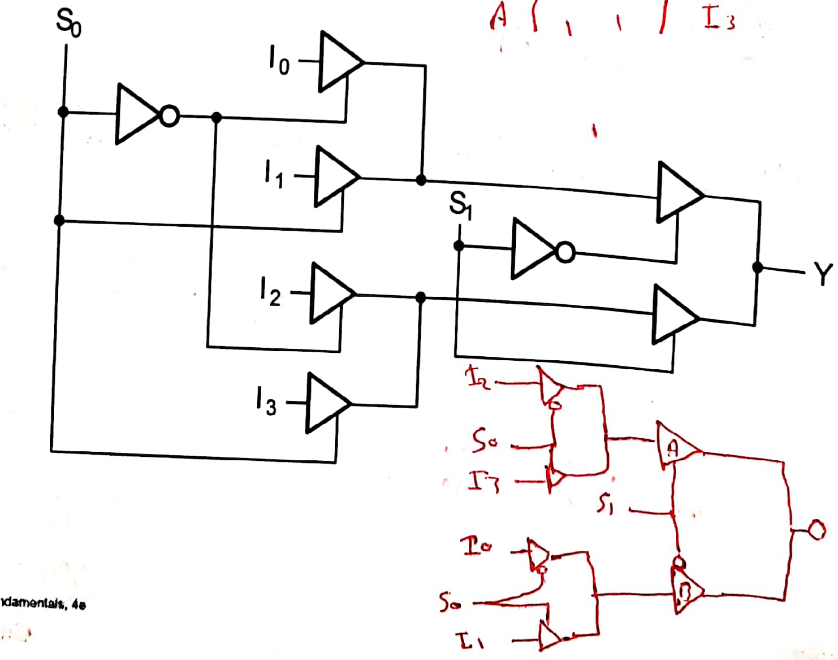
- Can be thought of as four 4-to-1 MUXes:



Other Selection Implementations

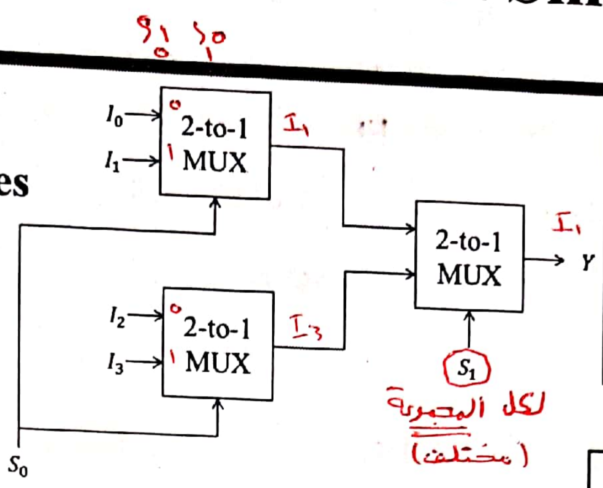
- **Three-state logic**

	S_1	S_0	O
B	0	0	I_0
	0	1	I_1
A	1	0	I_2
	1	1	I_3



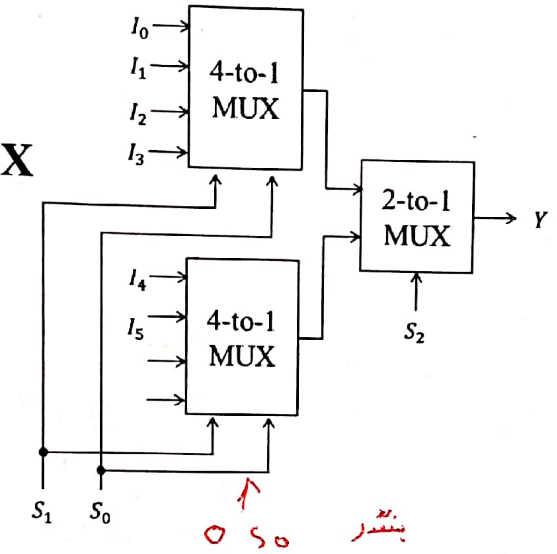
Building Large MUXes from Smaller Ones

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



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

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



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

Homework

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

- Multiplexer Approach 1

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

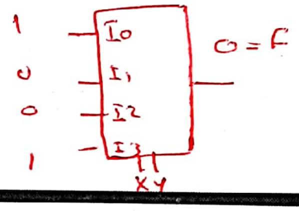
$F(x,y) = xy + \bar{x}\bar{y}$

$O = I_0\bar{S}_1\bar{S}_0 + I_1\bar{S}_1S_0 + I_2S_1\bar{S}_0 + I_3S_1S_0$

$I_0\bar{x}\bar{y} + I_1\bar{x}y + I_2x\bar{y} + I_3xy$

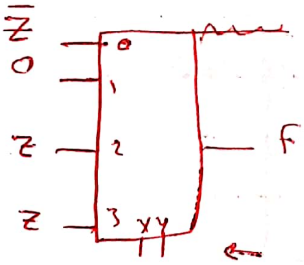
Example1

xy	
0 0	1
0 1	0
1 0	0
1 1	1



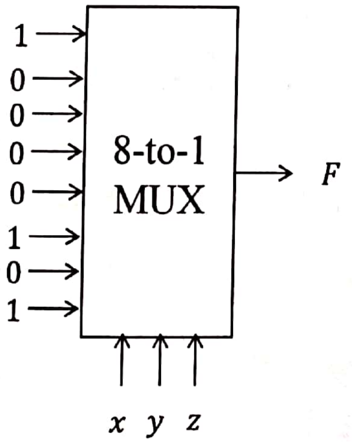
- Implement the following function using a single MUX based on Approach1 : $F(x, y, z) = \sum m(0, 5, 7)$
- Solution:
  - Single function → m = 1
  - 3 variables → n = 3 → 8-to-1 MUX
  - Fill the truth table of *F*

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



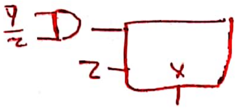
وبزبط
غير x,y

اذا بدي كان عنا
بيسا 4 to 1



2 to 1

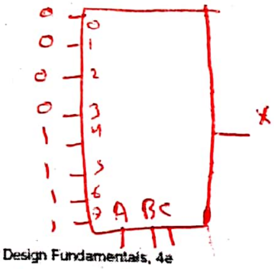
وبزبط تحمله

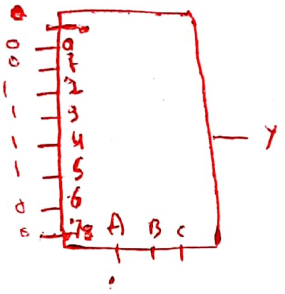


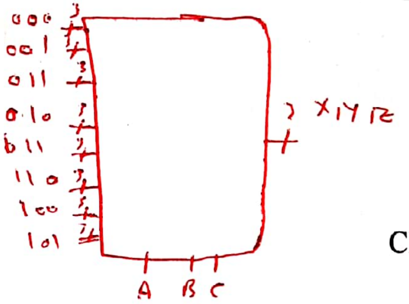
Example2: Gray to Binary Code

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

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







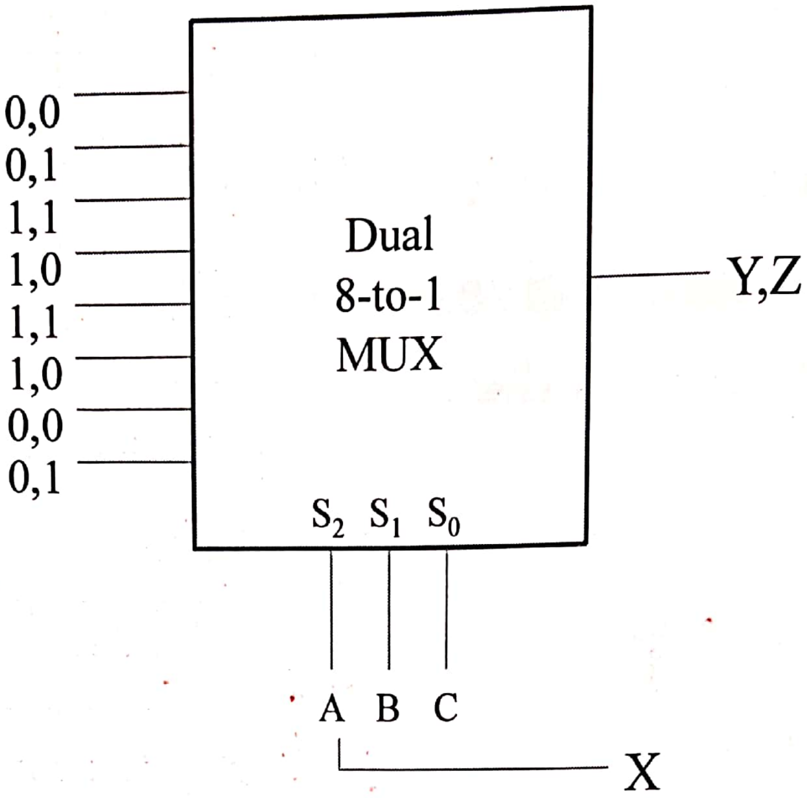
: and Computer Design Fundamentals, 4e
erPoint® Slides
08 Pearson Education, Inc.

Gray to Binary Code Cont.

رتبنا ال table

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

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

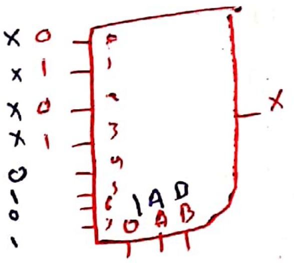


Combinational Logic Implementation - Multiplexer Approach 2

F(A,B) = Σ1,3

00	0
01	1
10	0
11	1

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



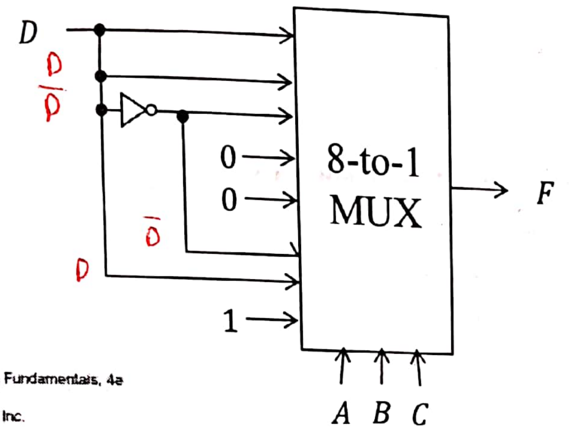
Example1

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

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

- Solution:
  - Single function → m = 1
  - 4 variables → n = 4 → 8-to-1 MUX
  - Fill the truth table of F

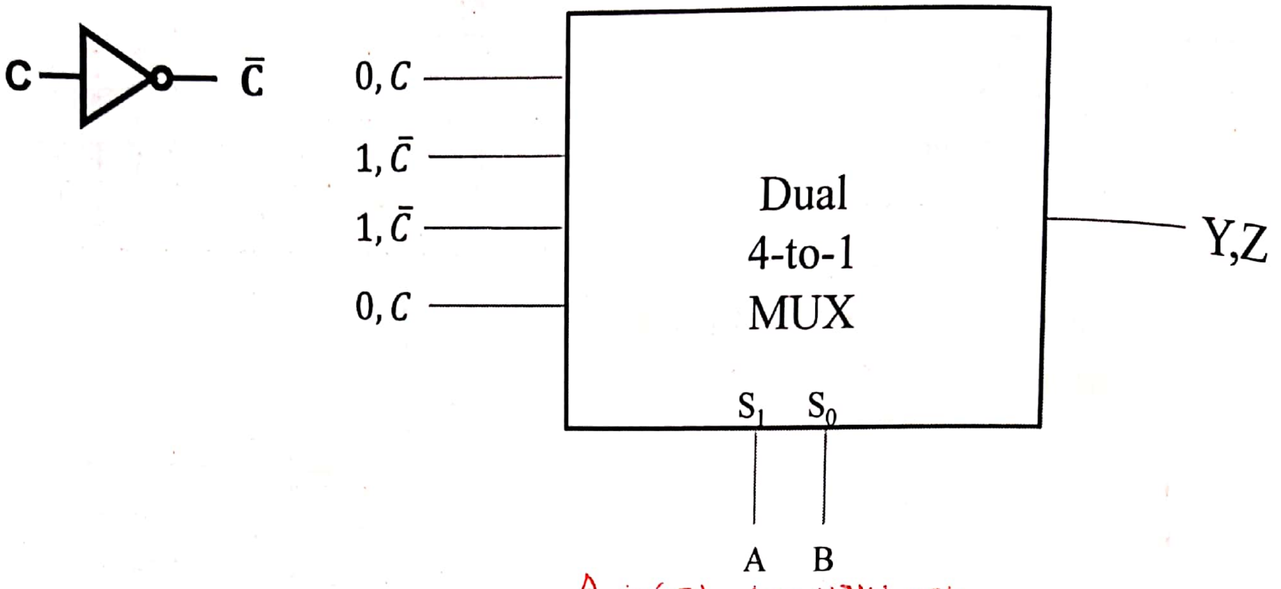
A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = \bar{D}$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	1	$F = \bar{D}$
1	0	1	1	0	
1	1	0	0	0	$F = D$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	



Example2: Gray to Binary Code

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

- Assign the variables and functions to the multiplexer inputs:



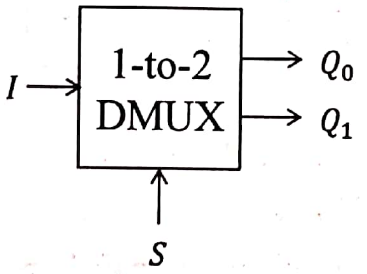
بنقدر نطلع ال x هنا مباشرة من A

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

Demultiplexer (DMUX)

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

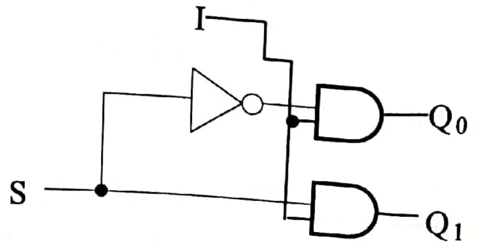
S	Q0	Q1
0	I	X
1	X	I



$$Q_0 = \bar{S}I$$
$$Q_1 = SI$$

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

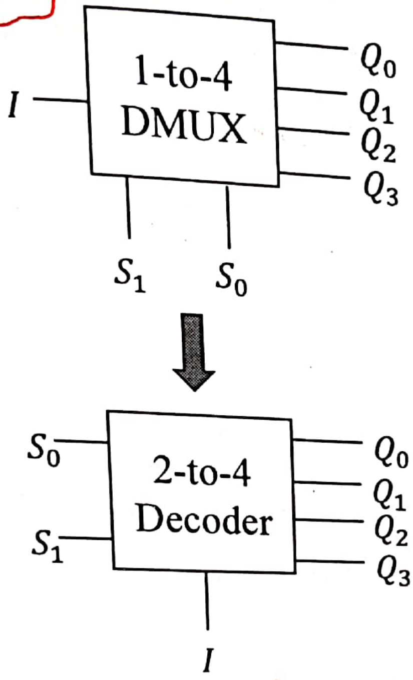
- ***DMUX ≡ Decoder with Enable***

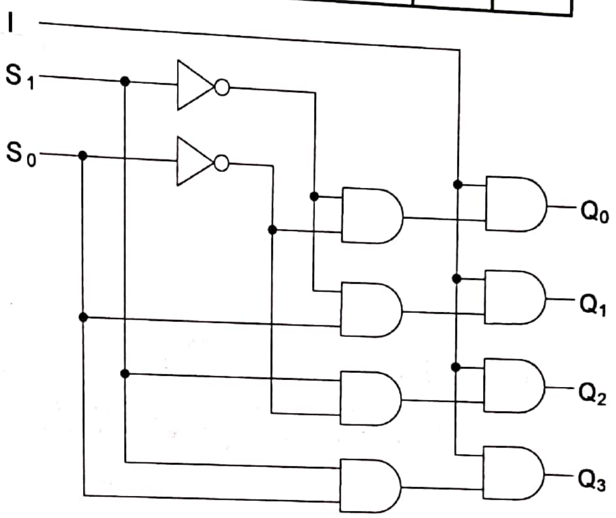


1-to-4 DMUX

- $Q_0 = \bar{S_1}\,\bar{S_0} I$
- $Q_1 = \bar{S_1} S_0 I$
- $Q_2 = S_1 \bar{S_0} I$
- $Q_3 = S_1 S_0 I$

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





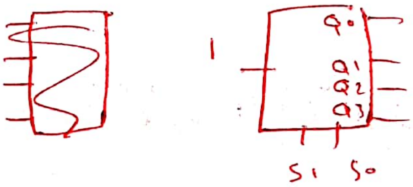
① بنقدر نحول اي DMUX لـ Decoder انه نحط ال input

$Q_0 = 1 \cdot \bar{S_1}\bar{S_0}$

$Q_1 = 1 \cdot \bar{S_1} S_0$

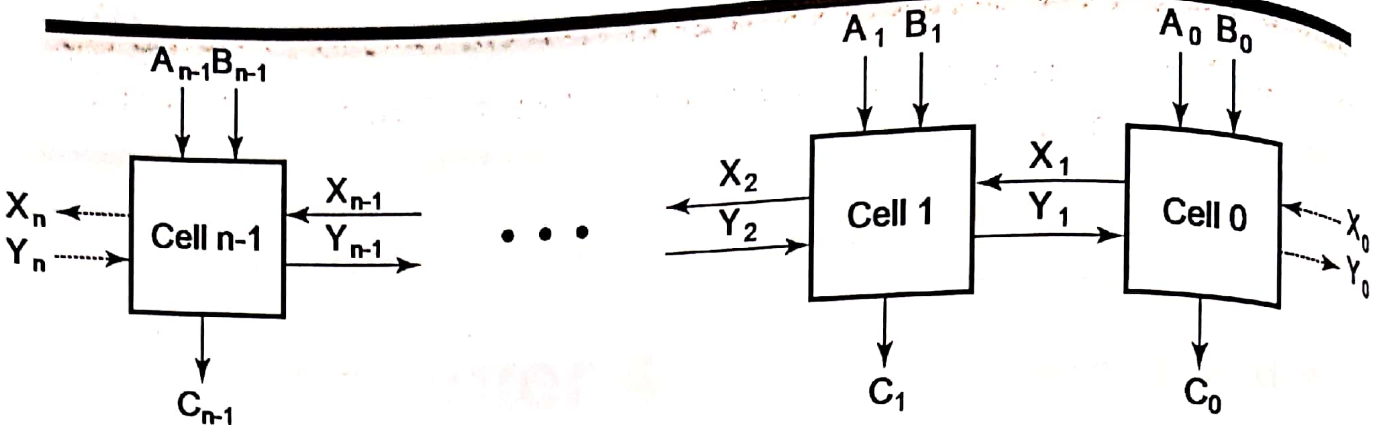
$Q_2 = 1 \quad S_1 \bar{S_0}$

$Q_3 = 1 \quad S_1 S_0 = \phi$



Terms of Use

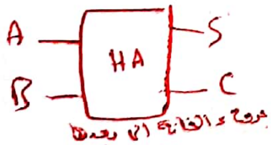
Block Diagram of an Iterative Array



- Example: n = 32
  - Number of inputs = 32*2 + 1 + 1 = 66
  - Truth table rows = 2^{66}
  - Equations with up to 66 input variables
  - Equations with huge number of terms
  - Design impractical!
- Iterative array takes advantage of the regularity to make design feasible

Functional Blocks: Addition

- Binary addition used frequently
- Addition Development:



الي بجمع الخانتين ال بعض

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

Functional Block: Half-Adder

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

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

Ⓒ Ⓢ

ضيعاتنين وحدة وباقي

- A half adder adds two bits to produce a two-bit sum

كم عشرة وا؟ زي $\frac{5+7}{1\;2}$ باقي

- The sum is expressed as a ***sum bit (S)*** and a ***carry bit (C)***

- The half adder can be specified as a truth table for S and C ⇒

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

Logic Simplification and Implementation: Half-Adder

- The K-Map for S, C is:

لما يكون X مختلف عن Y بيعطيني 1

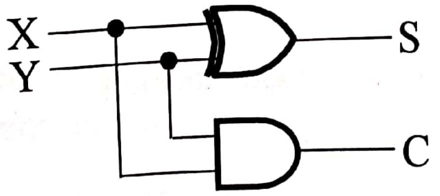
$$S = X \cdot \overline{Y} + \overline{X} \cdot Y = X \oplus Y$$

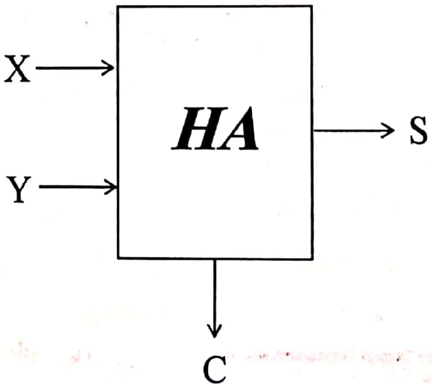
$$C = X \cdot Y$$

S		Y
	$_0$	1$_1$
X	1$_2$	$_3$

C		Y
	$_0$	$_1$
X	$_2$	1$_3$

- The most common half adder implementation is:





Functional Block: Full-Adder

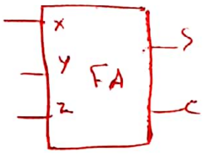
- A full adder is similar to a half adder, but includes a carry-in bit from lower stages. Like the half-adder, it computes a ***sum bit (S)*** and a ***carry bit (C)***

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

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

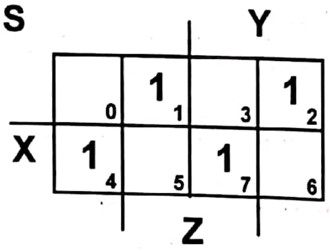
- For a carry- in (Z) of 1:

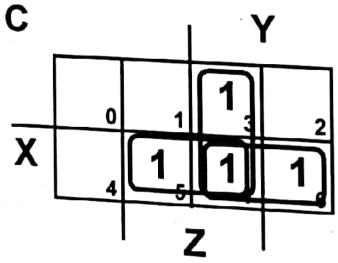
Z	1	1	1	1
X	0	0	1	1
+Y	+0	+1	+0	+1
C S	0 1	1 0	1 0	1 1



Logic Optimization: Full-Adder

- Full-Adder Truth Table:
- Full-Adder K-Map:





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

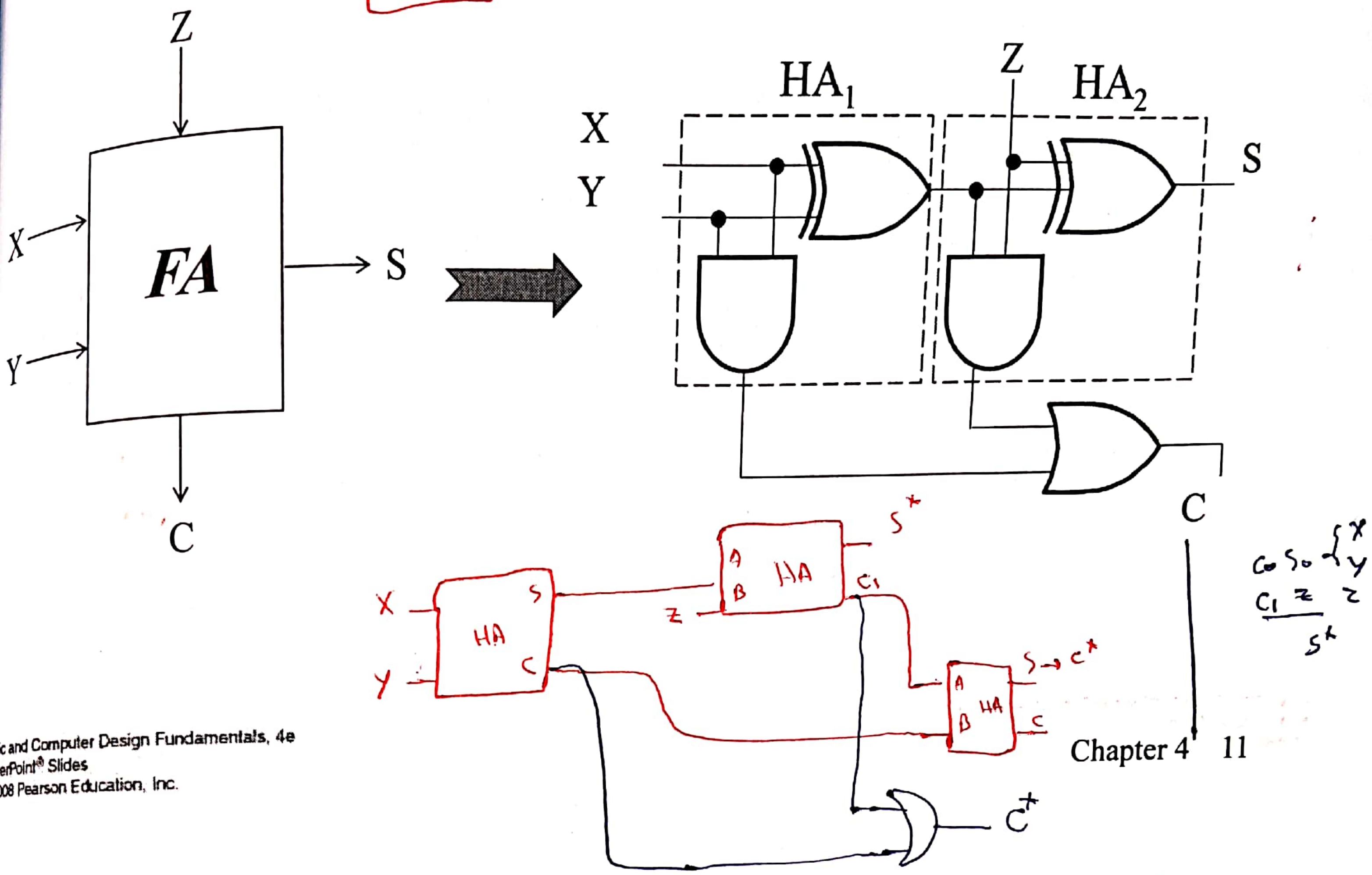
$S = \overline{X}\,\overline{Y}Z + \overline{X}Y\overline{Z} + X\overline{Y}\,\overline{Z} + XYZ$ $\quad C = XZ + XY + YZ$

- The S function is the three-bit XOR function (Odd Function):
  - $S = X \oplus Y \oplus Z$

يكون ال ①
تنتين او اكتر

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

معناها بيطلعوا بيطلع ①
اذا XY مع ①
او X او Y مع ① وتكون Z ①



c and Computer Design Fundamentals, 4e
erPoint® Slides
08 Pearson Education, Inc.

Binary Adders

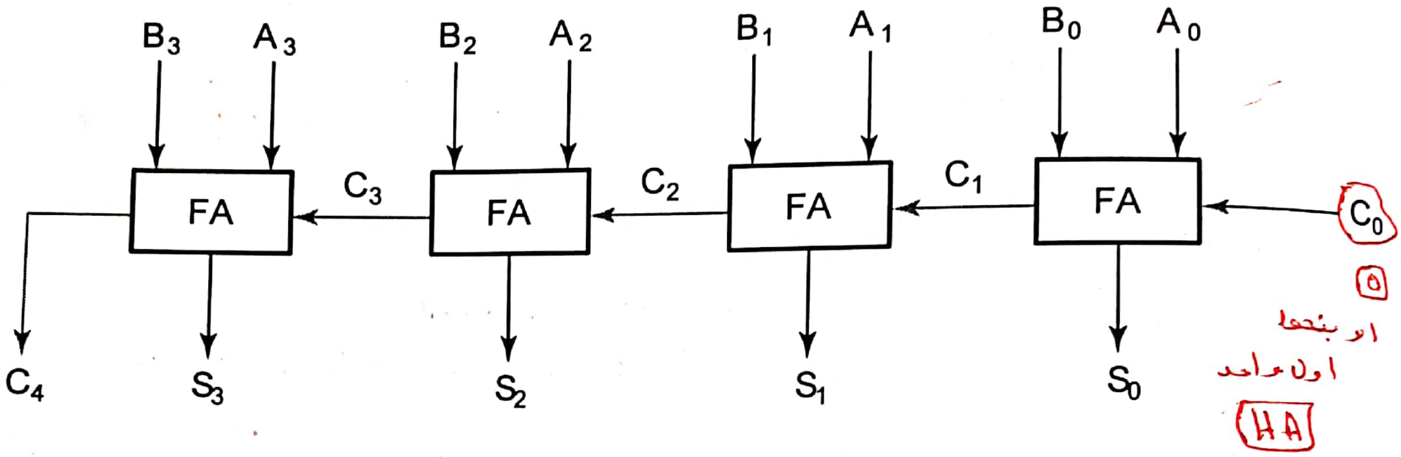
- To add multiple operands, we "bundle" logical signals together into vectors and use functional blocks that operate on the vectors
- Example: ***4-bit ripple carry adder*** adds input vectors A(3:0) and B(3:0) to get a sum vector S(3:0)
- Note: carry-out of ***cell i*** becomes carry-in of ***cell i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

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

$C_3\ C_2\ C_1\ C_0$
$A_3A_2A_1A_0$
$B_3B_2B_1B_0$
$S_3\ S_2\ S_1\ S_0$

4-bit Ripple-Carry Binary Adder

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



Homework

- **Design a 4-bit ripple-carry adder using HA's only?**

زي صفحة ١١

Unsigned Subtraction

D → 10 + 1

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

X	$^{0}0$	$^{1}0$	$^{0}1$	$^{0}1$
− Y	− 0	− 1	− 0	− 1
B D	0 0	1 1	0 1	0 0

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

15 − 7 = 8

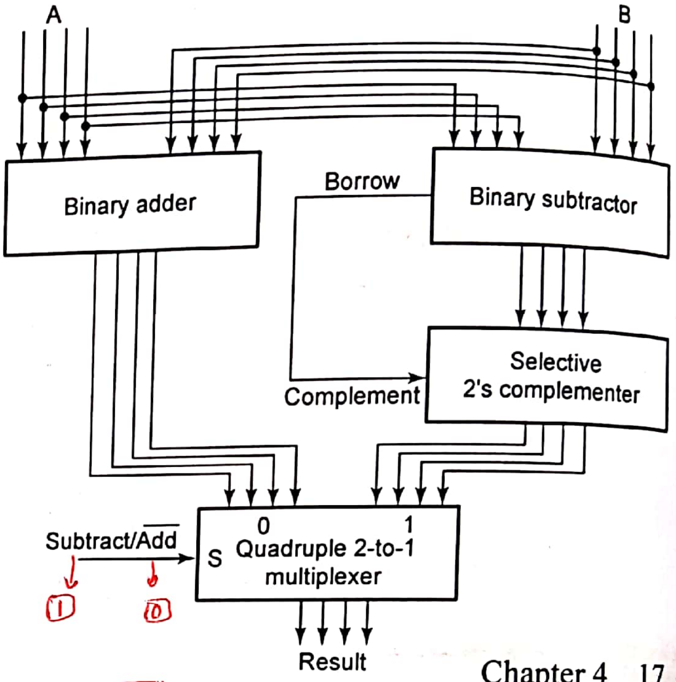
Unsigned Subtraction

- **Examples:**

0	1	1	0	1
1001	0100	10011	10010110	01100100
− 0111	− 0111	− 11110	− 01100100	− 10010110
0010	2^n - (1101)	10101	00110010	11001110
2's comp.	$2^4 = 16$ → 10000	100000		100000000
	− 1101	− 10101		− 11001110
	(−) 0011	(−) 01011		(−) 00110010

Unsigned Subtraction (continued)

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



A
B
Binary adder
Borrow
Binary subtractor
Complement
Selective 2's complementer
Subtract/$\overline{Add}$
1
0
0
1
S
Quadruple 2-to-1 multiplexer
Result

Logic and Computer Design Fundamentals, 4e
PowerPoint® Slides

S=0 add
S=1 Sub

$(8^2-1)-(43)_8$

$((100)_8-1)-(43)_8$

$(77)-(43)_8$

نفس الخانات (اكبر قيمة لل r.)

$\cancel{1}\overset{7}{\cancel{0}}\overset{8}{\cancel{0}}$
$-\ \ 1$
77

Complements

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

$(r^n-1)-N$

$(10^2-1)-65$

$99-65=34$

$(2^3-1)-(101)_2$

$8-1 \rightarrow 1000-1$

$(111)_2-(101)_2=010$

$\cancel{1}\cancel{0}\cancel{0}\overset{1}{\cancel{0}}$
$-\ \ 1$
0111

2	1's Comp
10	9's Comp
8	7's Comp
16	15's Comp

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

$(r^n - 1) - (1011)$

$1111 - 1011 = 0100$

- Example: Find 9's complement of $(45)_{10}$
  - r = 10, n = 2
  - Answer is $(10^2 - 1) - (45)_{10} = (54)_{10}$
  - Notice that $(45)_{10} + (54)_{10} = (99)_{10}$ which is $\underbrace{(10-1)(10-1)}_{\text{2-digits}}$

$99 - 45 = \underline{54}$

- Example: Find 7's complement of $(671)_8$
  - r = 8, n = 3
  - Answer is $(8^3 - 1) - (671)_8 = (106)_8$
  - Notice that $(671)_8 + (106)_8 = (777)_8$ which is $\underbrace{(8-1)(8-1)(8-1)}_{\text{3-digits}}$

Binary 1's Complement

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

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

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

بال Binary بنقدر
بس نعكس كل Bit

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

Radix Complement

- **For number N with n-digit and radix (r):**
  - **If $N \neq 0$, r's complement of $N = r^n - N$**
    - ***r's complement = (r-1)'s complement + 1***
  - **If $N = 0$, r's complement of $N = 0$**
- Example: Find 10's complement of $(92)_{10}$
  - r = 10, n = 2 خانتين
  - Answer is $10^2 - (92)_{10} = (8)_{10}$
  - Notice that 9's complement of $(92)_{10}$ is $(7)_{10}$
    - *10's complement = 9's complement + 1*

0 F F F 16→
~~1~~ ~~0~~ ~~0~~ ~~0~~ 10
3 A E 7 -
$(C519)_{16}$

- Example: Find 16's complement of $(3AE7)_{16}$
  - r = 16, n = 4
  - decimal ←
  - Answer is $16^4 - (3AE7)_{16} = (10000)_{16} - (3AE7)_{16} = (C519)_{16}$
  - 15's complement = $(C518)_{16}$ → 16's complement = $(C518)_{16} + 1 = (C519)_{16}$

Logic and Computer Design Fundamentals, 4e
PowerPoint® Slides

Binary 2's Complement

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

بمشي بين ما الاقي اول واحد
بنزله وبعكس الي بعده
عالاصفار اذا قبل بنزلهم

$$\begin{array}{r} \mathbf{100000000} \\ -\ \underline{\mathbf{01110011}} \\ \mathbf{10001101} \end{array}$$

1 0 1 1 0 0 ←
0 1 0 1 0 0

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

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

- 2's Complement Example:

  10010<u>100</u>

  - Copy underlined bits:

    <u>100</u>

  - and complement bits to the left:

    <u>01101</u>100

$M-N \rightarrow B=0,\ C=1 \quad M>N$

$M-N \rightarrow B=1,\ C=0 \quad , M<N$

$M+2'sN$

$M+(2^n-N)$

$M-N+\underline{2^n}$

$5-3+10=\boxed{1}2$ c

$3-5+10=8$ x c

Subtraction with 2's Complement

- For n-digit, <u>unsigned</u> numbers M and N, find M – N in base 2:
  - Add the 2's complement of the subtrahend N to the minuend M:
    - M – N ⟹ $M + (2^n – N) = M – N + 2^n$
- If M ≥ N, the ***sum*** produces end carry 2^n which is discarded; and from above, M – N remains
- If M < N, the ***sum*** does not produce end carry, and from above, is equal to $2^n – (N – M)$ which is the 2's complement of (N – M)
- To obtain the result – (N – M) , take the 2's complement of the sum and place a "–" to its left

$M-N \rightarrow$

$M+(-N)$

$M+2's\ N$

$\underline{\underline{0-N}}$

$N=101$

1000 ← $\boxed{1}$

000

$\underline{101}$ -

Compl. ← 011

$5-3 \rightarrow 5+(-3)$

$5+7$

$=\boxed{12}$

- Find $01010100_2 - 01000011_2$

$$\boxed{0}\quad \begin{array}{r} 01010100 \\ -\ \underline{01000011} \end{array} \xrightarrow{\text{2's comp}} \begin{array}{r} \overset{c}{\boxed{1}}01010100 \\ +\ \underline{10111101} \\ 00010001 \end{array}$$

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

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

- Find $01000011_2 - 01010100_2$

$$\begin{array}{rr} 1 \rightarrow & 01000011 \\ - & \underline{01010100} \\ \text{Bor} & 11101111 \end{array} \xrightarrow{\text{2's comp}} \begin{array}{r} \boxed{0}\,01000011 \\ +\ \underline{10101100} \\ 11101111 \end{array} \xrightarrow{\text{2's comp}}$$

نضع السالب $\boxed{00010001}$

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

- Result = – (00010001)

Unsig

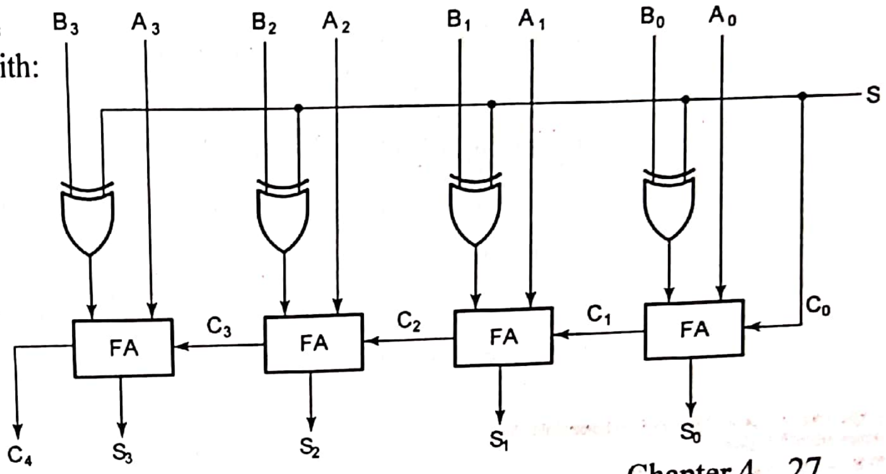
- *Subtraction can be done by addition of the 2's Complement*
  1. Complement each bit (1's Complement)
  2. Add 1 to the result

$A - B = A + 2's\,B$

$A + 1's\,B + 1$

$A + \bar{B} + 1$

- The circuit shown computes A + B and A – B:
- Subtract (S = 1): $A - B = A + (2^n - B) = A + \bar{B} + 1$
  - The 2's complement of B is formed by using XORs to form the 1's complement and adding the 1 applied to C_0
  - If $C_4 = 1$ ($A \geq B$): correct result
  - If $C_4 = 0$ ($A < B$): result = $2^n - (B - A)$
    - Use 2's complement logic **OR**
    - Use Adder/Subtractor again with:
      - A = 0
      - $B = 2^n - (B - A)$
- Add (S = 0): A + B
  - B is passed through unchanged



Signed Integers

- ***Positive numbers and zero*** can be represented by unsigned ***n-digit, radix r*** numbers. ***We need a representation for negative numbers***
- To represent a sign (+ or –) we need exactly one more bit of information (1 binary digit gives $2^1 = 2$ elements which is exactly what is needed).
- Since computers use binary numbers, by convention, ***the most significant bit is interpreted as a sign bit***:

$$\mathbf{s\ a_{n-2} \cdots a_2 a_1 a_0}$$

where:

s = 0 for Positive numbers

s = 1 for Negative numbers

and a_i = 0 or 1 represent the magnitude in some form