



---

# Digital Design

**CS / EEE / ECE/ INSTR F215**

**Instructors:** Dr. Ankur Bhattacharjee (L1)  
Prof. Gurunarayanan S (L2)

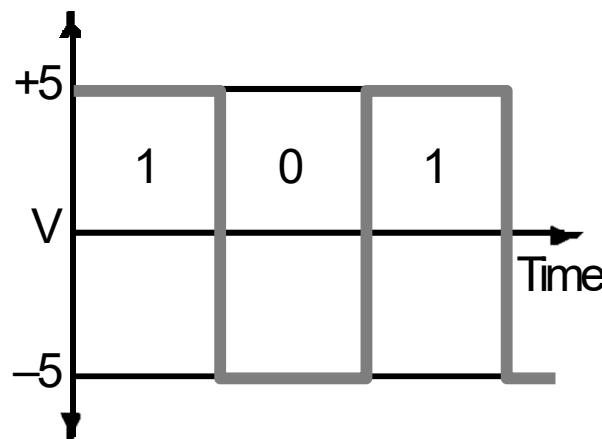
**Contacts:**

[a.bhattacharjee@hyderabad.bits-pilani.ac.in](mailto:a.bhattacharjee@hyderabad.bits-pilani.ac.in)

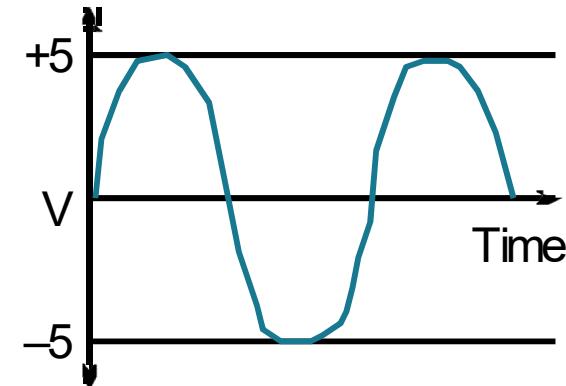
[sguru@pilani.bits-pilani.ac.in](mailto:sguru@pilani.bits-pilani.ac.in)

# Electrical Signal

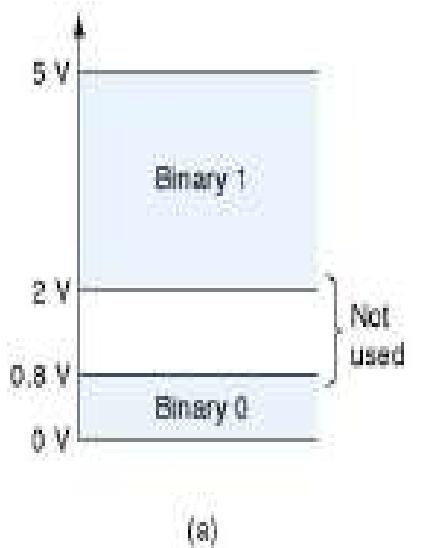
- **Analog Signal:** Output value varies continuously.  
Most of the real world events analog in nature.  
**(Ex: Output from a sensor)**
- **Digital Signal:** Output value varies at discrete levels,  
required for internal analysis



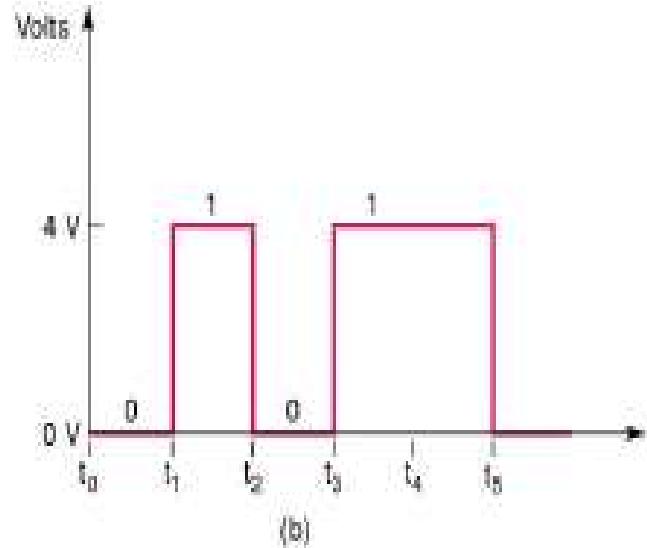
8/21/2021



2



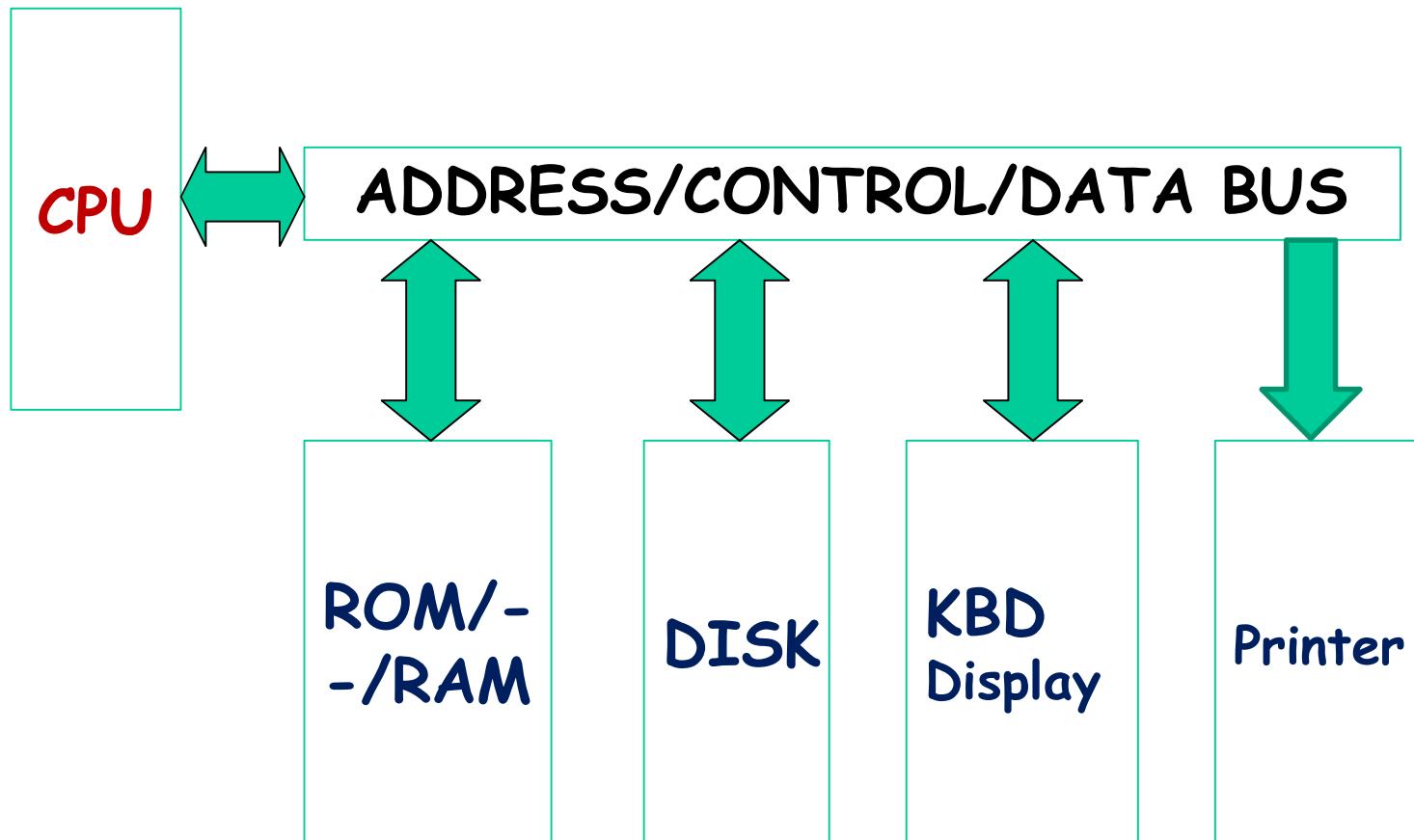
- (a) Typical voltage assignments in digital systems  
(b) Typical digital signal timing diagram



## ADVANTAGES OF DIGITAL SYSTEMS

- Ease of design
- Programmability
- Speed
- Storage
- Cost
- Less Prone to noise

# Ex: Block diagram of a Computer System





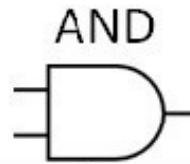
# Flow of Topics

Lecture No.	Learning objectives	Topics to be covered	Chapter in the Text/Ref Book
1	Introduction to Digital Systems and Characteristics of Digital ICs.  Number system	Discussion on course curriculum and evaluation procedure. Advantages and disadvantages of digital systems, Evolution of Digital technology terminologies used in digital systems.  Binary numbers, two's and one's compliment	T1:2.9  T1:1.2-1.9
2	Number system	Addition and subtraction of binary numbers, octal and hexadecimal numbers, binary codes	T1:1.2-1.9
3-4	Boolean algebra and logic gates	Boolean functions, canonical forms, logic gates.	T1: 2.1-2.8
5-6	Simplification of Boolean functions	K-Maps (3,4,5 variables)	T1: 3.1- 3.8
7-8	Simplification of Boolean functions	QM Method	T1: 3.10
9	Simplification of Boolean functions	Mutli-level and Multi-output Circuits  Hazards in in Combinational Logic	R2: 7.1-7.7  R2: 8.4
10-14	Combinational Logic, Arithmetic circuits	Adders, Subtractors, Multipliers	T1: 4.1 – 4.7
15-19	MSI Components	Comparators, Decoders, Encoders, MUXs, DEMUXs	T1: 4.8 - 4.11

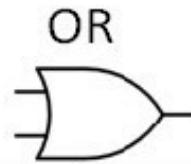


Lect. No.	Learning Objectives	Topics to be covered	Reference to Text/Ref Book
20-23	Sequential Logic circuits	Flip-Flops & Characteristic tables, Latches	T1: 5.1 - 5.4
24-28	Clocked Sequential Circuits	Analysis of clocked sequential circuits, state diagram and reduction	T1: 5.5, 5.7 & 5.8
29-32	Registers & Counters	Shift registers, Synchronous & Asynchronous counters, clock skew & Clock Jitter	T1: 6.1 - 6.5
33-35	Design of Digital Systems	Algorithmic State Machines (ASM)	T1: 8.4
36-38	Memory and PLDs	RAM, ROM, PLA, PAL	T1:7.1 - 7.7
39-40	Digital Integrated Circuits	RTL, DTL,TTL,ECL & CMOS Gates, Implementation of Simple CMOS circuits	T1:10.1 -10.7

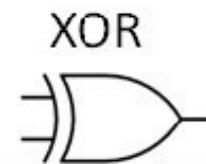
# Basic Binary Logic Gates



INPUT		OUTPUT
A	B	
0	0	0
1	0	0
0	1	0
1	1	1



INPUT		OUTPUT
A	B	
0	0	0
1	0	1
0	1	1
1	1	1



INPUT		OUTPUT
A	B	
0	0	0
1	0	1
0	1	1
1	1	0

**NOT**



INPUT		OUTPUT
A		
0		1
1		0

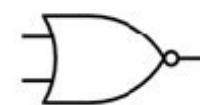
# Basic Binary Logic Gates

NAND



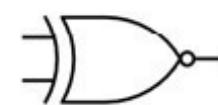
INPUT		OUTPUT
A	B	
0	0	1
1	0	1
0	1	1
1	1	0

NOR



INPUT		OUTPUT
A	B	
0	0	1
1	0	0
0	1	0
1	1	0

XNOR



INPUT		OUTPUT
A	B	
0	0	1
1	0	0
0	1	0
1	1	1



# Evolution of Digital Technology

---

- Diode Transistor Logic : 1959
- Resistor Transistor Logic : 1961
- Transistor Transistor Logic : 1963 Discrete
- Emitter-coupled logic : First Microprocessor 360
- CMOS :
  - 1974 Intel 4004 which had 2000 Transistors  
Channel Length of 10 μm.
  - 2020 AMD 7 nm has billions of Transistors  
Channel Length of 7 nm.



# Digital ICs:

## LEVELS OF INTEGRATION

- SSI
- MSI
- LSI
- VLSI



# Digital LOGIC FAMILIES

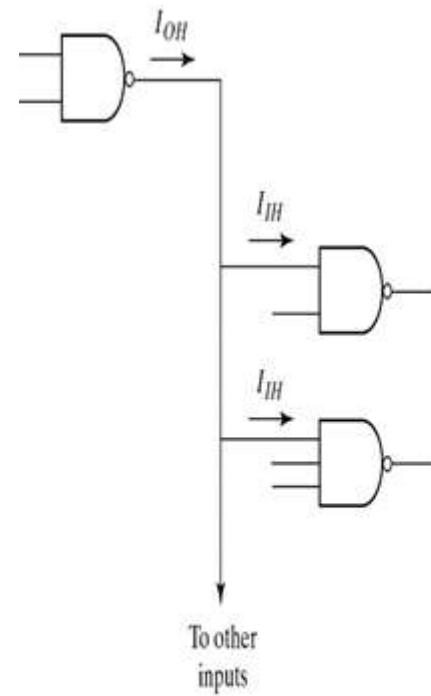
- RTL, DTL
- TTL
- ECL
- NMOS, PMOS
- CMOS



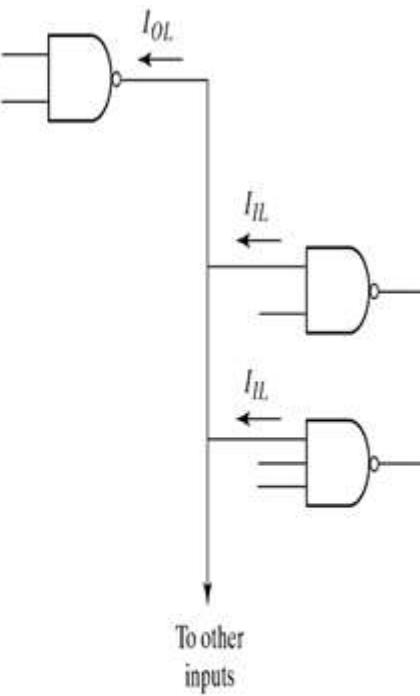
# IC CHARACTERISTICS

---

- Fan Out
- Propagation Delay
- Noise Margin
- Power Dissipation



(a) High-level output



(b) Low-level output

Fan-Out Computation

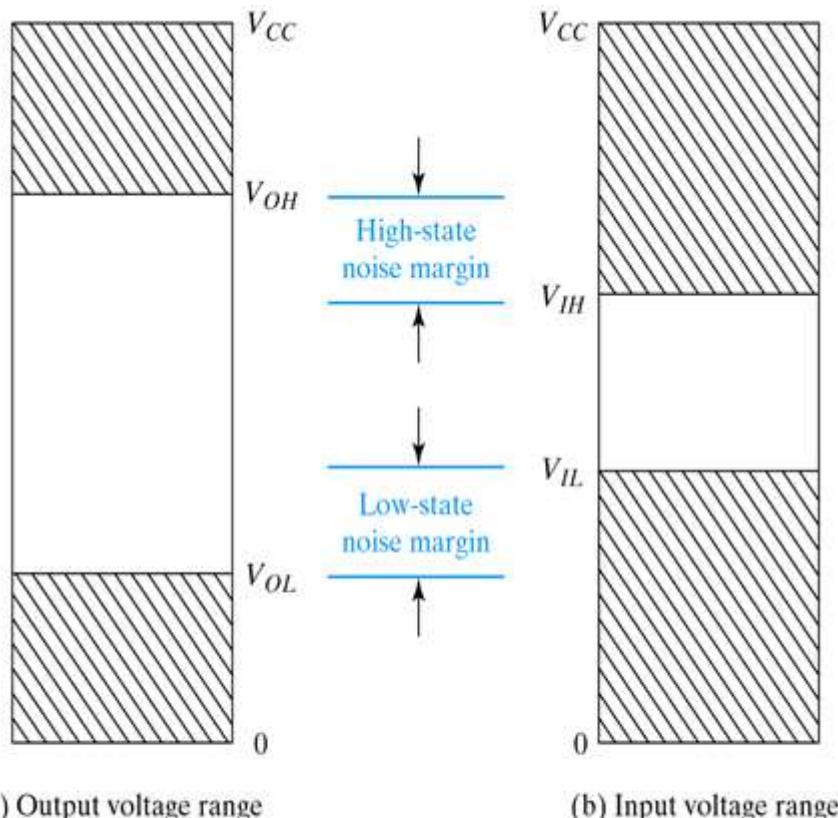
Input

Output

$t_{PHL}$

$t_{PLH}$

Measurement of Propagation Delay



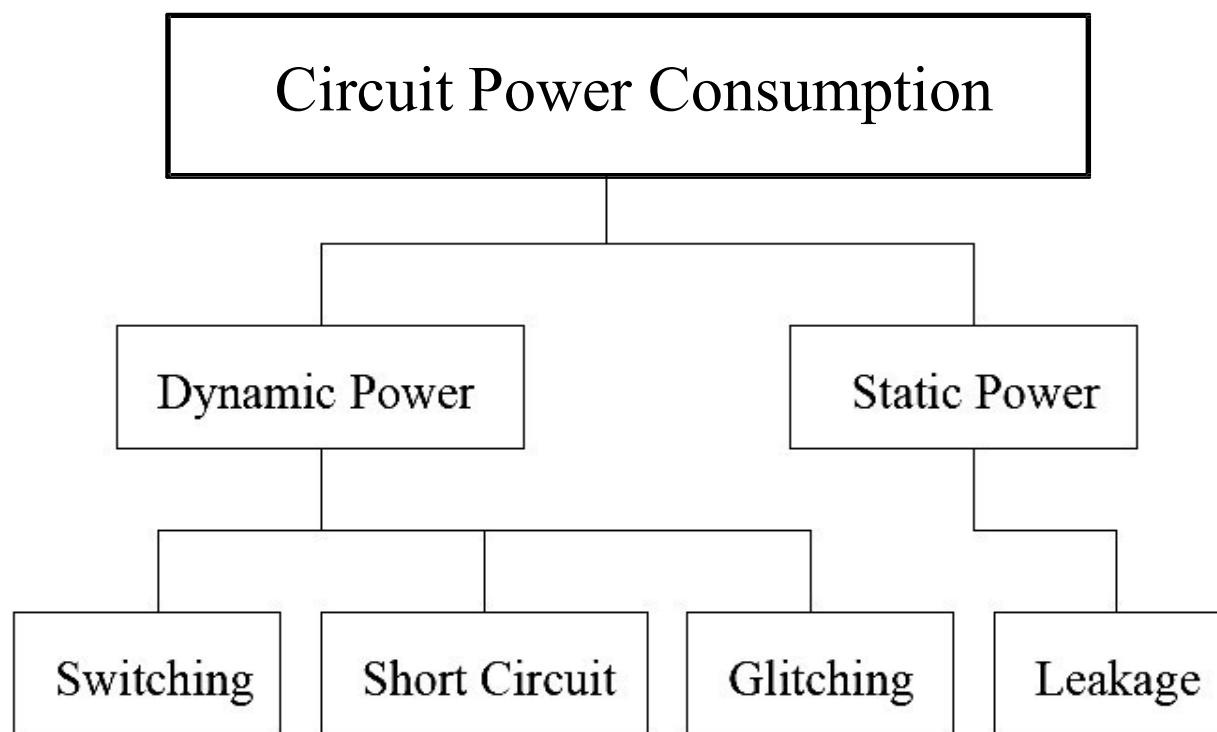
Signals for Evaluating Noise Margin

## Power Dissipation ( $P_D$ )

- Expressed in Milliwatts
- $P_D = V_{CC} * I_{CC}$
- $I_{CC(\text{avg})} = (I_{CCH} + I_{CCL}) / 2$

# Power consumption in Digital Circuits

---





---

# Introduction to Number systems



# Number systems

---

General use Decimal numbers : 8956

Digits used are 0 - 9

$$8956 = 8 \times 10^3 + 9 \times 10^2 + 5 \times 10^1 + 6 \times 10^0$$

Can be generalized to any decimal number

$$a_3 a_2 a_1 a_0 \cdot a_{-1} a_{-2}$$

$$= a_3 \times 10^3 + a_2 \times 10^2 + a_1 \times 10^1 + a_0 \times 10^0 + a_{-1} \times 10^{-1} + a_{-2} \times 10^{-2}$$



# Number systems

---

Decimal number system: Base is 10 Numbers used : 0-9

Base also called radix

Binary number system : Base is 2 Numbers used : 0-1

For example: 101.11

$$= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$$

$$= 4 + 0 + 1 + 0.5 + 0.25$$

$$= 5.75$$



# Number systems

---

For Base - r system  $(a_n a_{n-1} \dots a_1 a_0. a_{-1} a_{-2} \dots a_{-m})_r$

$$a_n \times r^n + a_{n-1} \times r^{n-1} \dots a_1 \times r^1 + a_0 \times r^0 + a_{-1} \times r^{-1} + \dots a_{-m} \times r^{-m}$$

Find the decimal equivalent of

$$(123.4)_8 \text{ [Octal]} = 1 \times 8^2 + 2 \times 8^1 + 3 \times 8^0 + 4 \times 8^{-1} = 83.5$$

$$(B2.4)_{16} \text{ [Hexa decimal]} = 11 \times 16^1 + 2 \times 16^0 + 4 \times 16^{-1} = ??$$

$$(110101)_2 \text{ [Binary]} = 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ = ??$$

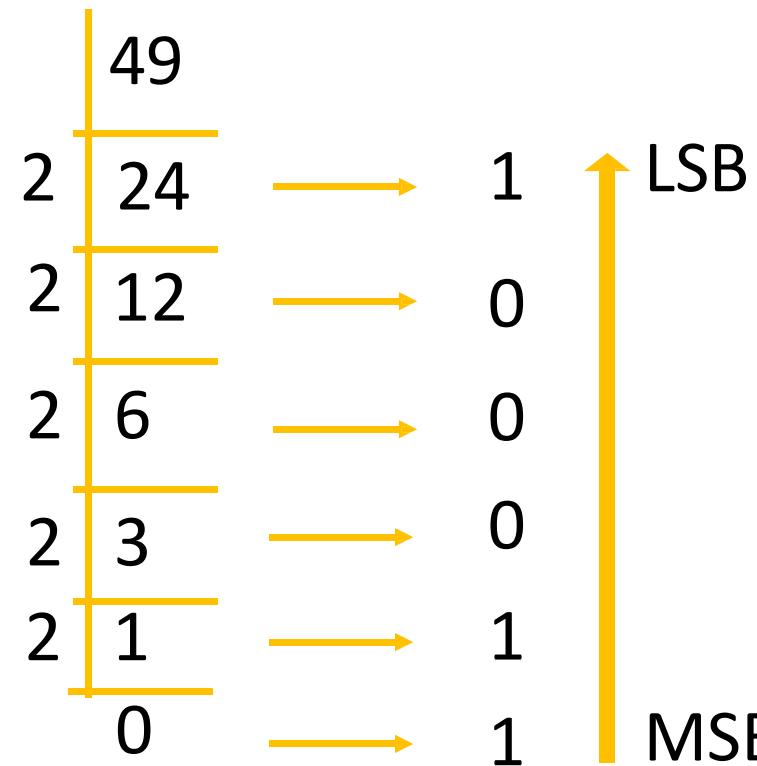
# Number Base conversions

## Typical conversions

Base-10 to Base-r

Convert  $(49)_{10}$  to  $( )_2$

$(110001)_2$





# Number Base conversions

## Typical conversions

Base-10 to Base-r (fraction)

Convert  $(0.125)_{10}$  to  $()_2$

$$0.125 \times 2 = 0.25$$

$$0.25 \times 2 = 0.5$$

$$0.5 \times 2 = 1.0$$

Integer



$$(0.125)_{10} = (0.001)_2$$

Limited to required number of digits



# Number Base conversions

## Typical conversions

Base-r to Base-10

Convert  $(110110)_2$  to  $( )_{10}$

$$=1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

$$=1 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 2^0$$

$$=1 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 2^0$$

$$=(54)_{10}$$



# Exercise

Express the following numbers in decimal

$$(10110.0101)_2$$

$$(1010.1010)_2$$

$$(26.24)_8$$

$$(16.5)_{16}$$

$$(FAFA)_{16}$$

$$(B65F)_{16}$$

# Number Base conversion

## Other conversions

Binary to octal      Octal: base 8 digits used 0 - 7

1110001010101      For Octal-  $2^3$ , 8bit:  $2^3$

001    110    001    010    101  
↓       ↓       ↓       ↓       ↓  
1       6       1       2       5

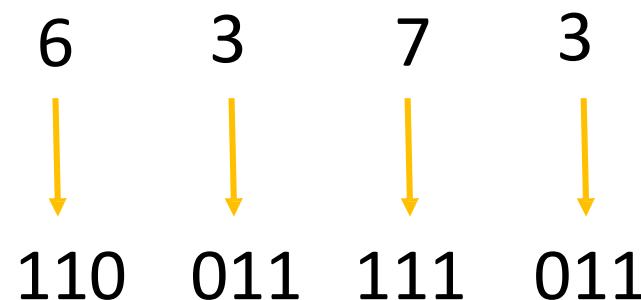
$$(1110001010101)_2 \longrightarrow (16125)_8$$

# Number Base conversion

## Other conversions

octal to binary

Octal: base 8 digits used 0-7



$$(6373)_8 \longrightarrow (110011111011)_2$$

# Number Base conversions

## Other conversions

Binary to Hexadecimal    Hexa: base 16   digits used 0-F

1110001010101    4 bit, then Hexadecimal-  $2^4$

0001 1100 0101 0101

↓              ↓              ↓              ↓  
1              C              5              5

$$(1110001010101)_2 \longrightarrow (1C55)_{16}$$



# Books for Digital Design

---

## ➤ **Text Book:**

M.Moris Mano and Michael D. Ciletti, “Digital Design”, Pearson, 5th Edition, 2013.

## ➤ **Reference Books:**

- R1. Neal S. Widmer, Gregory L. Moss & Ronald J. Tocci, “Digital Systems Principles and Applications” Pearson, 12th Edition, 2018.
- R2. Charles H. Roth, Jr. and Larry L. Kinney “Fundamentals of Logic Design” Cengage Learning 7th Edition, 2013.
- R3: M.Moris Mano and Michael D. Ciletti “ Digital Logic and Computer Design”, Pearson,, e-Book, 2016.



# Evaluation Scheme

Component	Duration	Weightage (%)	Marks allotted	Date & Time	Nature of Component
Assignment(s) /Quiz	-	10%	20	To be announced	Open Book
Mid Semester Examination	90 Minutes	30%	60	20/10/2021 9:00 – 10:30 A.M (Time-Table)	Open Book
Regular Lab	During Lab hours	15%	30	Regular Lab days	-
Final Lab Examination	-	10%	20	To be announced	Open Book
Comprehensive Exam	120 minutes	35%	70	16/12/2021 FN session (Time-Table)	Open Book
<b>Total</b>		<b>100%</b>	<b>200</b>		



# Salient Features of Online Lecture Class

## Notices :

- All notices will be uploaded on **CMS & Google Classroom** (For 'L' section common notice will be uploaded)
- Class record files/links and slides will be uploaded separately for each section (L1 & L2) on **Google classroom**



# *Thank You*



---

# Digital Design

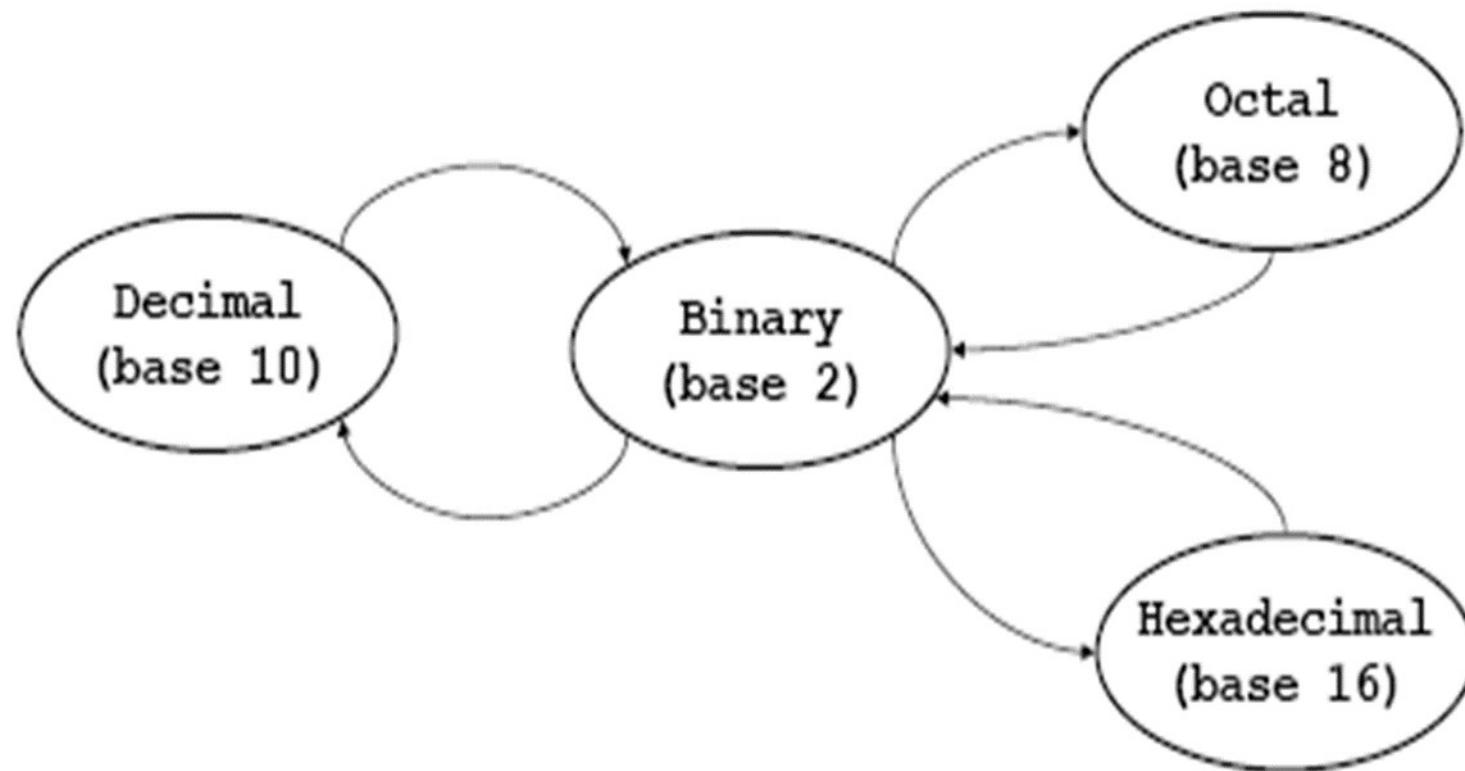
**CS / EEE / ECE/ INSTR F215**

## Lecture 2: Number Systems

**Contd...**

---

## Decimal, Binary, Octal, Hexadecimal





- Signed Magnitude
- Diminished radix complement
- Radix complement



# Representation of Negative Numbers

## ➤ Signed Magnitude

### Limitations

1. Two Zeros

2. Add +2 & -1

0 1 0

1 0 1

1 1 1

3-bit numbers	Signed magnitude
0 0 0	+0
0 0 1	+1
0 1 0	+2
0 1 1	+3
1 0 0	-0
1 0 1	-1
1 1 0	-2
1 1 1	-3

**MSB indicates Sign : 0 indicates positive, 1 indicates negative**



# Complements

## Diminished radix complement

Given a number  $N$  in base  $r$  having  $n$  digits  $(r-1)$ 's complement is defined as  $(r^n - 1 - N)$

In case of decimal it is called 9's complement

9's complement of 865 is:  $10^3 - 1 - 865 = 999 - 865 = 134$

In case of binary it is called 1's complement

1's complement of 1011 is  $2^4 - 1 - 1011 = 1111 - 1011 = 0100$   
(or you can simply use the complement ~ 1 for 0 and 0 for 1)

# Complements

Decimal	S.M.	1's comp.
7	0111	0111
6	0110	0110
5	0101	0101
4	0100	0100
3	0011	0011
2	0010	0010
1	0001	0001
0	0000	0000
-0	1000	1111
-1	1001	1110
-2	1010	1101
-3	1011	1100
-4	1100	1011
-5	1101	1010
-6	1110	1001
-7	1111	1000
-8	-	-

➤ End-around-carry-bit addition

Add 4 & -7

$$\begin{array}{r}
 0100 \\
 1000 \\
 \hline
 1100
 \end{array}$$

Add 4 & -3

$$\begin{array}{r}
 0100 \\
 1100 \\
 \hline
 1\ 0000 \\
 1 \\
 \hline
 0001
 \end{array}$$



# Complements

## Radix complement

Given a number  $N$  in base  $r$  having  $n$  digits  $r$ 's complement is defined as  $(r^n - N)$

In case of decimal it is called 10's complement

$$10\text{'s complement of } 865 \text{ is } 10^3 - 865 = 1000 - 865 = 135$$

$$10\text{'s complement} = 9\text{'s complement} + 1$$

In case of binary it is called 2's complement

$$2^4 - 1011 = 10000 - 1011 = 0101$$

$$2\text{'s complement} = 1\text{'s complement} + 1$$

# Complements

Decimal	S.M.	1's comp.	2's comp.
7	0111	0111	0111
6	0110	0110	0110
5	0101	0101	0101
4	0100	0100	0100
3	0011	0011	0011
2	0010	0010	0010
1	0001	0001	0001
0	0000	0000	0000
-0	1000	1111	—
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	—	—	1000

➤ No End-around-carry-bit addition

Add 4 & -7

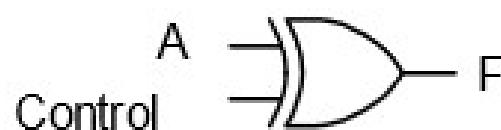
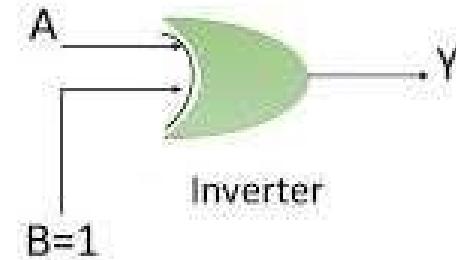
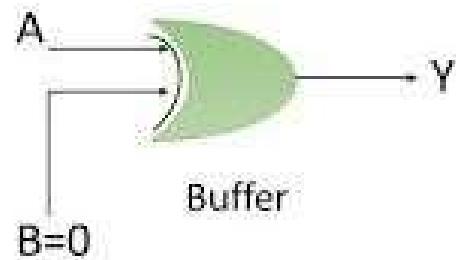
$$\begin{array}{r}
 0100 \\
 1001 \\
 \hline
 1101
 \end{array}$$

Add 4 & -3

$$\begin{array}{r}
 0100 \\
 1101 \\
 \hline
 1\ 0001
 \end{array}$$

# Complements

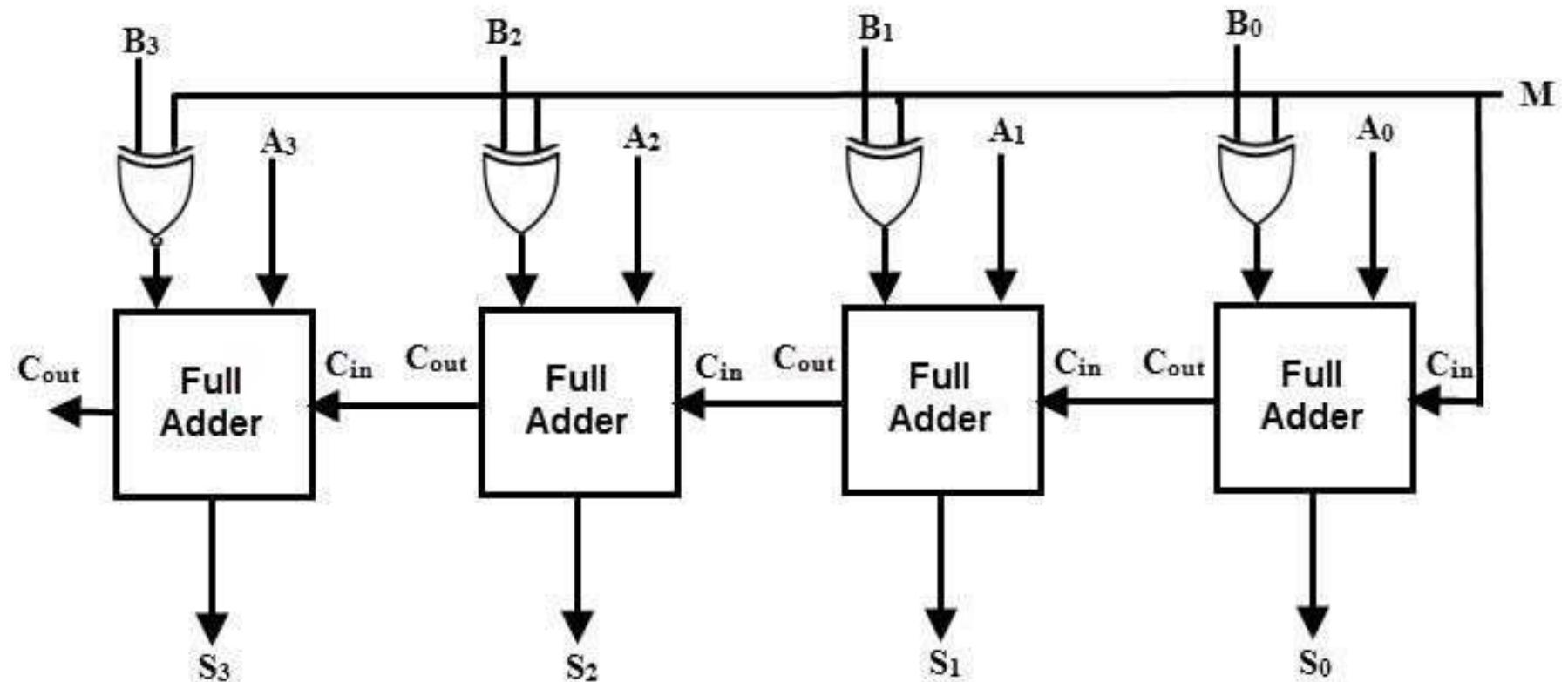
## EX-OR Gate As Buffer and Inverter



Control	A	F	
0	0	0	Pass
0	1	1	
1	0	1	Invert
1	1	0	

# Complements

- Easy Implementation: Adder Subtractor-  
M=0 for 'Adder', M=1 for 'Subtractor'



# Binary Codes - BCD

0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9

Consider an example 7698

7      6      9      8

BCD code



# BCD and Binary relationship

$(185)_{10}$

$\text{BCD} = (0001 \ 1000 \ 0101)$

$\text{Binary} = (10111001)_2$

$\text{BCD} = 12 \text{ bits}, \text{Binary} = 8 \text{ bits}$

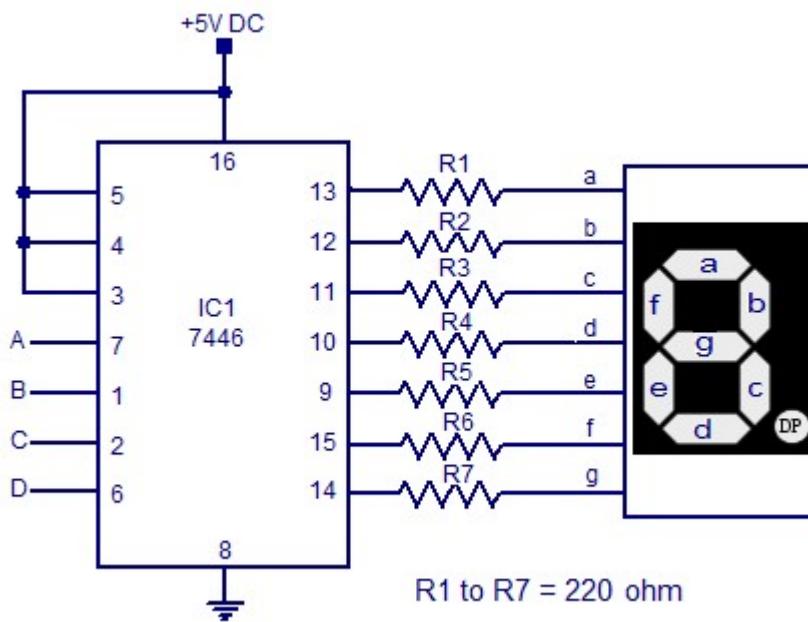
Some systems work directly on BCD (**IBM Power6**)

User enters decimal  $\rightarrow$  BCD i/p  $\rightarrow$  compute in BCD  $\rightarrow$  BCD o/p  $\rightarrow$  Decimal output shown to user

# Binary Codes - BCD

## General digital systems

User enters decimal → BCD i/p → Binary i/p → compute in binary  
→ Binary o/p → BCD o/p → Decimal output shown to user





# Binary Codes - BCD

## BCD addition

$$4 + 5$$

4 0 1 0 0

5 0 1 0 1  
—————

9 1 0 0 1

Expected Result

$$4 + 8$$

4 0 1 0 0

8 1 0 0 0  
—————

1 1 0 0

Is this expected Result ?

Expected answer is BCD of 12      0001 0010



# Binary Codes - BCD

4 + 8

$$\begin{array}{r} 4 \ 0 \ 1 \ 0 \ 0 \\ 8 \ 1 \ 0 \ 0 \ 0 \\ \hline 1 \ 1 \ 0 \ 0 \\ 0 \ 1 \ 1 \ 0 \\ \hline 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \\ \hline 1 \qquad \qquad \qquad 2 \end{array}$$

Greater than 9

1 1 0 0

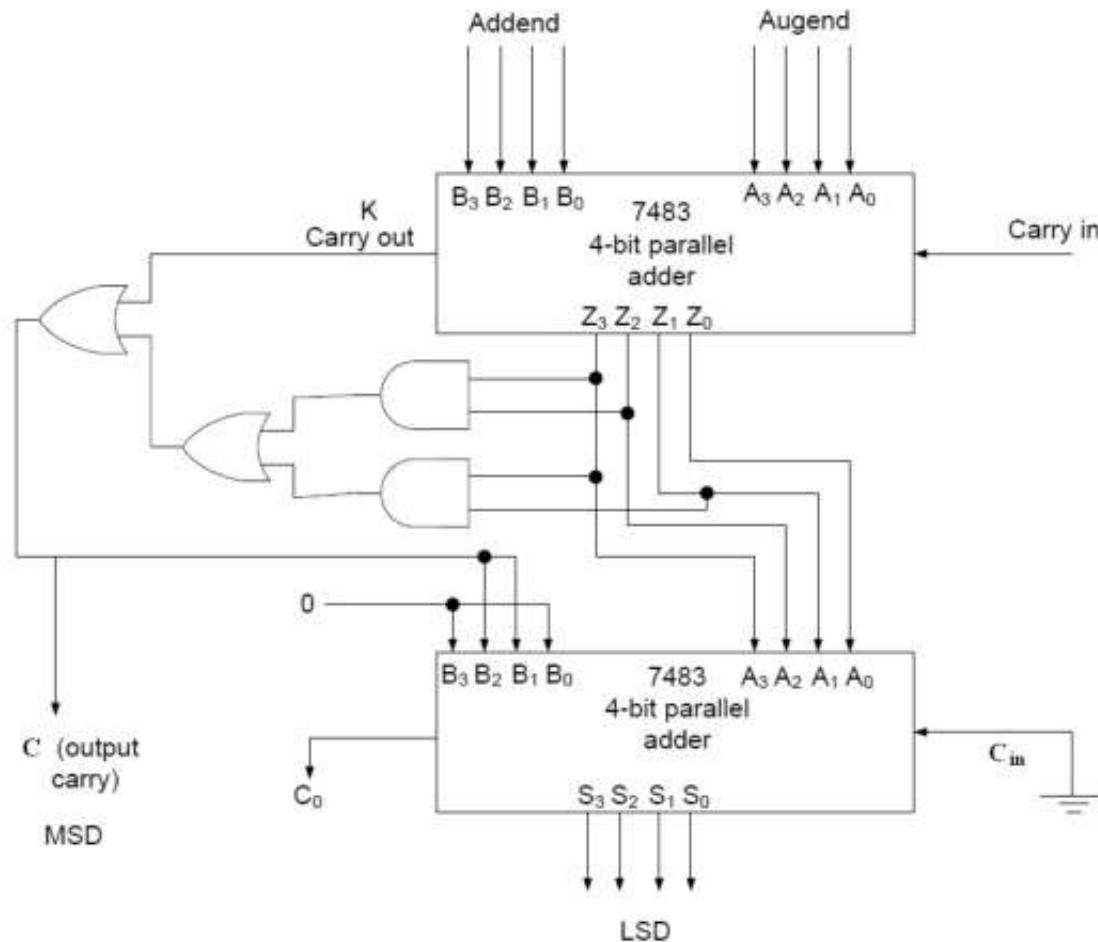
Add correction of +6

0 1 1 0

To skip 6 invalid  
states (10 - 15) BCDs

# Binary Codes - BCD

## BCD addition





# *Thank You*



---

# Digital Design

**CS / EEE / ECE/ INSTR F215**

## **Lecture 3a: Number systems and Boolean Expressions**



# Binary numbers: Mapping

Decimal Digits	8421 (represents BCD )	Excess 3	2421	84-2-1
0	0000	0011	0000	0000
1	0001	0100	0001	0111
2	0010	0101	0010	0110
3	0011	0110	0011	0101
4	0100	0111	0100	0100
5	0101	1000	1011	1011
6	0110	1001	1100	1010
7	0111	1010	1101	1001
8	1000	1011	1110	1000
9	1001	1100	1111	1111



# Gray Code

The advantage of the reflected code over pure binary numbers is that a number in the reflected code changes by only one bit as it proceeds from one number to the next. The reflected code is also known as the *Gray code*

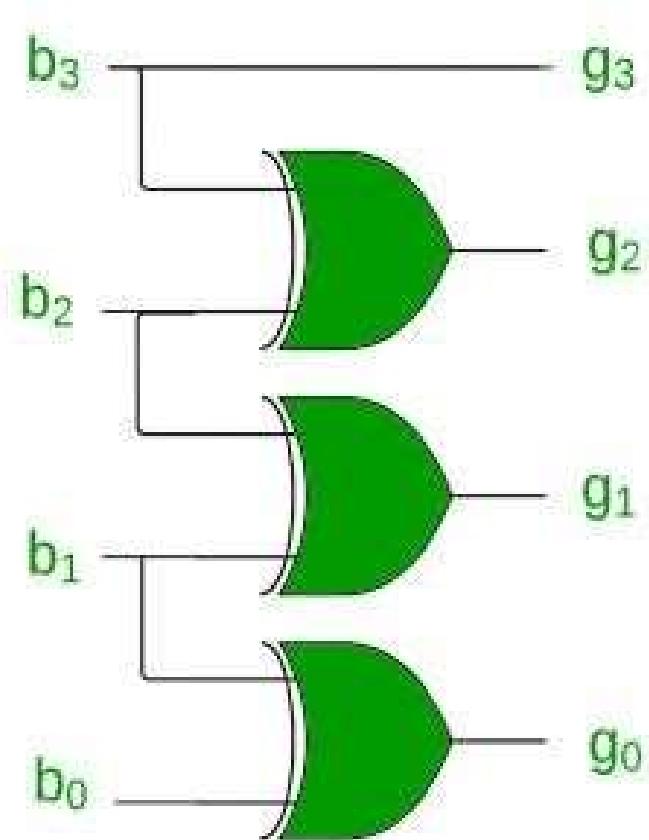


## THE GRAY CODE

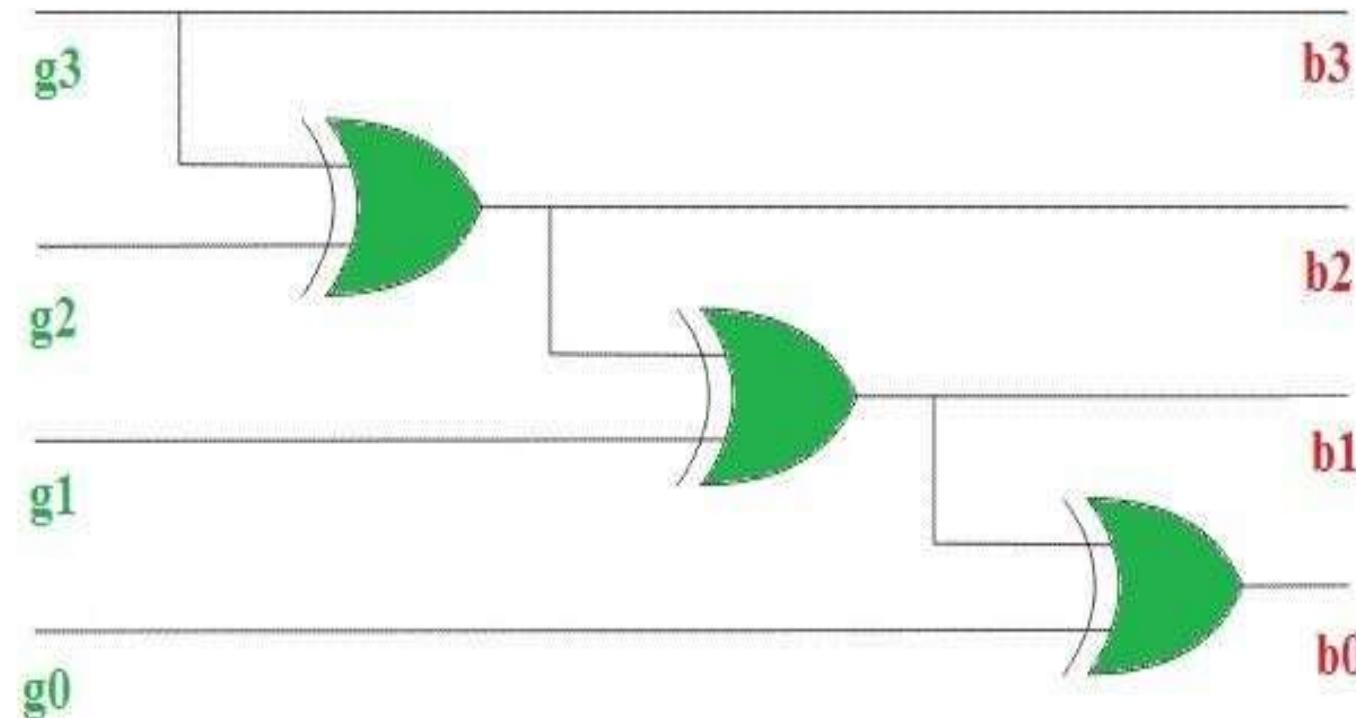
Decimal	Binary	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100

Decimal	Binary	Gray Code
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

# Binary Code – Gray Code



# Gray Code - Binary Code





# Digital Design

CS/EEE/ECE/INSTR F215

Lecture 3b: Boolean Function and Expressions



# Boolean Algebra

Some important properties

$$x + 0 = x$$

$$x \cdot 1 = x$$

$$1 + x = 1$$

$$x + x' = 1$$

$$x \cdot x' = 0$$

$$x \cdot x = x$$

$$x + y = y + x$$

$$x \cdot y = y \cdot x$$

$$x(y + z) = xy + xz$$

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



# Boolean Algebra

## Some important Theorems

$$1. x + x = x \quad x \cdot x = x$$

$$2. (x')' = x$$

$$3. x + (y + z) = (x + y) + z \quad x \cdot (y \cdot z) = (x \cdot y) \cdot z$$

$$4. (x + y)' = x'y' \text{ De-Morgan's Theorem } (x \cdot y)' = x' + y'$$

$$4. x + xy = x \quad x \cdot (x + y) = x$$



# Boolean Algebra

Theorems can be proved in two ways

Example;

-using Postulates

$$x + xy = x$$

$$LHS = x \cdot 1 + xy$$

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

$$= x \cdot 1$$

$$= x$$



# Boolean Algebra

-using Truth tables

$$x + xy = x$$

x	y	xy	x+xy	x
0	0	0	0	0
0	1	0	0	0
1	0	0	1	1
1	1	1	1	1

=

$$LHS = RHS$$



# Boolean Functions

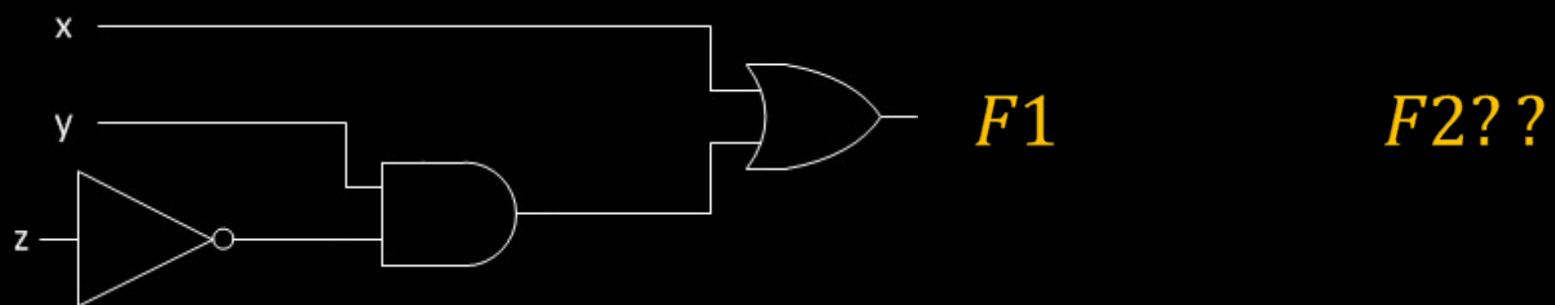
Consists of **binary variables**, **constants** and **Logic symbols**

$$F1 = x + yz'$$

$$F2 = (x + yz')'$$

Single variable in normal or complemented form - **literal**

Can be transformed into **circuit**





# Boolean Functions

Boolean function can be represented by truth table in  
only one way

<b>x</b>	<b>y</b>	<b>z</b>	<b>F1</b>	<b>F2</b>
0	0	0	0	1
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

$$F1 = x + yz'$$

$$F2 = (x + yz')'$$



# Boolean Functions

Boolean function can be represented in many ways in algebraic form

$$F2 = (x + yz')$$

Both represent same function ?

$$F2 = x'yz' + x'yz + xz'$$

Implementation of both same ? Try it out

Need for simplification



# Boolean Functions

Ways of simplification

-Algebraic manipulation

-K Maps

-QM(Quine-McCluskey) Method



# Boolean Functions

Algebraic manipulation/simplification is done using postulates and theorems

For example:  $F1 = x(x' + y)$  1 NOT, 1 OR, 1 AND

$$F1 = xx' + xy$$

$$F1 = xy \quad 1 \text{ AND}$$



CKV

# Thank You



---

# Digital Design

**CS / EEE / ECE/ INSTR F215**

## Lecture 4: Simplification of Boolean Expressions – using 'K' Map



---

Boolean Functions are expressed by  
**Minterms (SOP) and Maxterms (POS)**



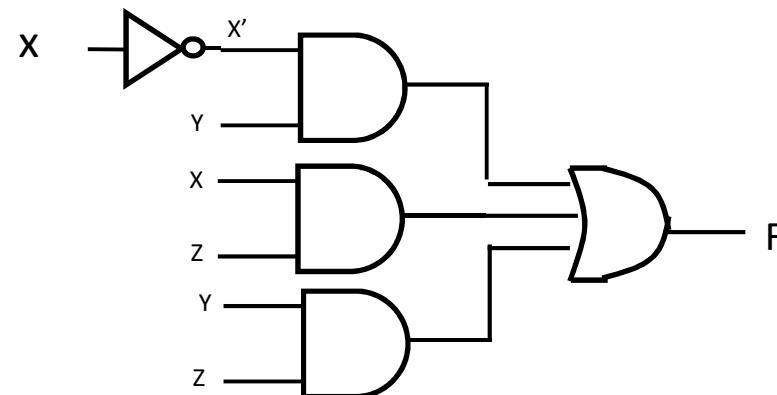
## Sum of the Product (SOP) form:

- A minterm is a special product of literals in which each input variable appears exactly once.
- A function with 'n' input variables has  $2^n$  possible minterms.
- A '*three*' variable function ( $x,y,z$ ) for example has 8 possible minterms  
 $x'y'z'$  ,  $x'y'z$  ,  $x'yz'$  ,  $x'yz$  ,  $xy'z'$  ,  $xy'z$  ,  $xyz'$  ,  $xyz$

minterm	True when	Shorthand
$x'y'z'$	$xyz=000$	$m_0$
$x'y'z$	$xyz=001$	$m_1$
$x'yz'$	$xyz=010$	$m_2$
$x'yz$	$xyz=011$	$m_3$
$xy'z'$	$xyz=100$	$m_4$
$xy'z$	$xyz=101$	$m_5$
$xyz'$	$xyz=110$	$m_6$
$xyz$	$xyz=111$	$m_7$

## Sum of the Product Expression

- 
- There are different ways of representing a function. Some form turns out to be more useful than others.
  - A Sum of Products or SOP expression consists of
    - One or more terms summed ( OR) together
    - Each term is a product term ( AND)
    - Sum of Product expressions can be implemented as a two level expression.



## Sum of minterms expressions

- A **sum of minterms** is a special kind of sum of products.
- Every function can be written as a *unique* sum of minterms expression.
- A truth table for a function can be rewritten as a sum of minterms just by finding the table rows where the function output is 1.

x	y	z	$C(x,y,z)$	$C'(x,y,z)$
0	0	0	0	1
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	0

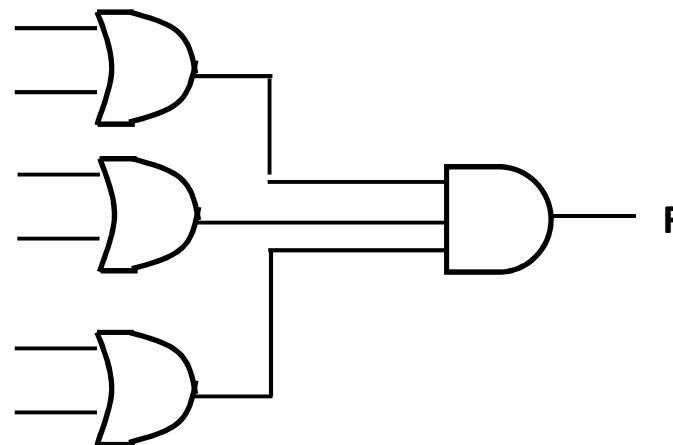
$$\begin{aligned}
 C &= x'y'z + xy'z + xyz' + xyz \\
 &= m_3 + m_5 + m_6 + m_7 \\
 &= \Sigma m(3, 5, 6, 7)
 \end{aligned}$$

$$\begin{aligned}
 C' &= x'y'z' + x'y'z + x'yz' + xy'z' \\
 &= m_0 + m_1 + m_2 + m_4 \\
 &= \Sigma m(0, 1, 2, 4)
 \end{aligned}$$

$C'$  contains all the minterms *not* in  $C$ , and vice versa.

# Product of Sum Expression

- 
- A Product of Sum or POS expression consists of
    - One or more terms multiplied ( AND) together
    - Each term is a sum term ( OR)
    - Product of Sum expressions can be implemented as a two level expression.



## Maxterms

- A **maxterm** is a *sum of literals* where each input variable appears once.
- A function with  $n$  input variables has  $2^n$  possible maxterms.
- For instance, a function with three variables  $x$ ,  $y$  and  $z$  has 8 possible maxterms:

$$\begin{array}{cccc} x + y + z & x + y + z' & x + y' + z & x + y' + z' \\ x' + y + z & x' + y + z' & x' + y' + z & x' + y' + z' \end{array}$$

- Each maxterm is *false* for exactly one combination of inputs.

Maxterm	False when	Shorthand
$x + y + z$	$xyz = 000$	$M_0$
$x + y + z'$	$xyz = 001$	$M_1$
$x + y' + z$	$xyz = 010$	$M_2$
$x + y' + z'$	$xyz = 011$	$M_3$
$x' + y + z$	$xyz = 100$	$M_4$
$x' + y + z'$	$xyz = 101$	$M_5$
$x' + y' + z$	$xyz = 110$	$M_6$
$x' + y' + z'$	$xyz = 111$	$M_7$

## Product of maxterms expressions

- Every function can also be written as a unique **product of maxterms**.
- A truth table for a function can be rewritten as a product of maxterms just by finding the table rows where the function output is 0.

x	y	z	$C(x,y,z)$	$C'(x,y,z)$
0	0	0	0	1
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	0

$$\begin{aligned}
 C &= (x + y + z)(x + y + z') \\
 &\quad (x + y' + z)(x' + y + z) \\
 &= M_0 M_1 M_2 M_4 \\
 &= \prod M(0,1,2,4)
 \end{aligned}$$

$$\begin{aligned}
 C' &= (x + y' + z')(x' + y + z') \\
 &\quad (x' + y' + z)(x' + y' + z') \\
 &= M_3 M_5 M_6 M_7 \\
 &= \prod M(3,5,6,7)
 \end{aligned}$$

$C'$  contains all the maxterms *not* in  $C$ , and vice versa.

- Now we've seen two different ways to write the function  $C$ , as a sum of minterms  $\Sigma m(3,5,6,7)$  and as a product of maxterms  $\prod M(0,1,2,4)$ .
- Notice the product term includes maxterm numbers whose corresponding minterms do not appear in the sum expression.

x	y	z	$C(x,y,z)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$\begin{aligned}
 C &= x'y'z + xy'z + xyz' + xyz \\
 &= m_3 + m_5 + m_6 + m_7 \\
 &= \Sigma m(3,5,6,7)
 \end{aligned}$$

$$\begin{aligned}
 C &= (x + y + z)(x + y + z') \\
 &\quad (x + y' + z)(x' + y + z) \\
 &= M_0 M_1 M_2 M_4 \\
 &= \prod M(0,1,2,4)
 \end{aligned}$$



# Simplification of Boolean Expressions

## using 'K Map'



# KARNAUGH MAP (K-Map)

- Graphical Device used to simplify Boolean expression
- K-map method is simple, straight forward and pictorial understanding gives both SOP and POS forms
- Minimal number of product term and minimal number of literal in each product term (SOP case)
- Leads to minimal two level implementation



---

Row of a Truth Table corresponds to a square in K-map

Adjacent square differ in only one variable

Combine squares with 1's

# Organizing K-Map

x	y	z	minterms
0	0	0	$x'y'z'$ ( $m_0$ )
0	0	1	$x'y'z$ ( $m_1$ )
0	1	0	$x'y z'$ ( $m_2$ )
0	1	1	$x'y z$ ( $m_3$ )
1	0	0	$x y'z'$ ( $m_4$ )
1	0	1	$x y'z$ ( $m_5$ )
1	1	0	$x y z'$ ( $m_6$ )
1	1	1	$x y z$ ( $m_7$ )

For SOP

yz  
x

$x'y'z'$	$x'y'z$	$x'y z$	$x'y z'$
$x y'z'$	$x y'z$	$x y z$	$x y z'$

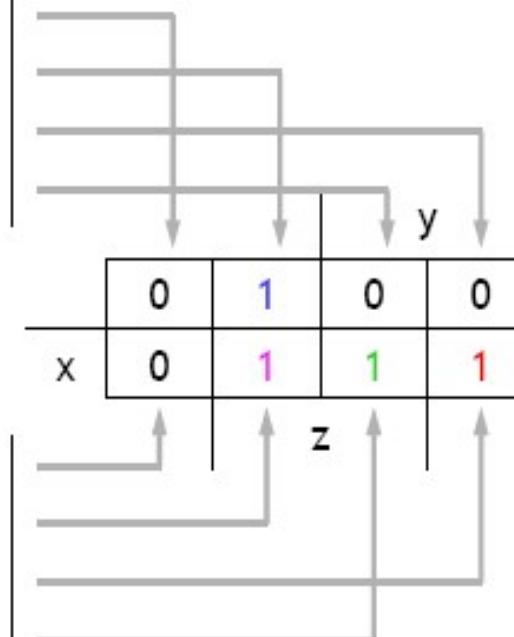
$m_0$	$m_1$	$m_3$	$m_2$
$m_4$	$m_5$	$m_7$	$m_6$

## Filling in the K-map

- Since our labels help us find the correct position of minterms in a K-map, writing the minterms themselves is redundant and repetitive.
- We usually just put a 1 in the K-map squares that correspond to the function minterms, and 0 in the other squares.
- For example, you can quickly fill in a K-map from a truth table by copying the function outputs to the proper squares of the map.

x	y	z	$f(x,y,z)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0

x	y	z	$f(x,y,z)$
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



$$f(x,y,z) = x'y'z + xy'z + xyz' + xyz$$

$x'y'z'$	$x'y'z$		
		$x'yz$	

$x'y'$

$yz$

## Four steps in K-map simplifications

1. Start with a sum of minterms or truth table.  $x'y'z' + x'y'z + xyz + xyz'$

2. Plot the minterms on a Karnaugh map.

			y
x	1	1	0
	0	0	0
	0	1	1

z

3. Find rectangular groups of minterms whose sizes are powers of two. Be sure to include all the minterms in at least one group!

			y
x	1	1	0
	0	0	0
	1	1	1

z

4. Reduce each group to one product term.

$$x'y' + xy$$

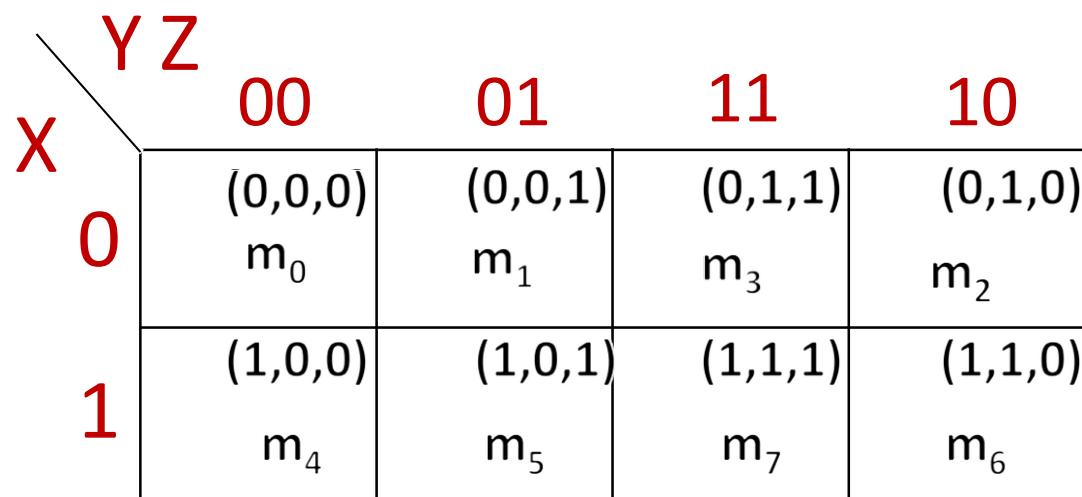


**Maximize the size of the groups**

**Minimizing the number of groups**

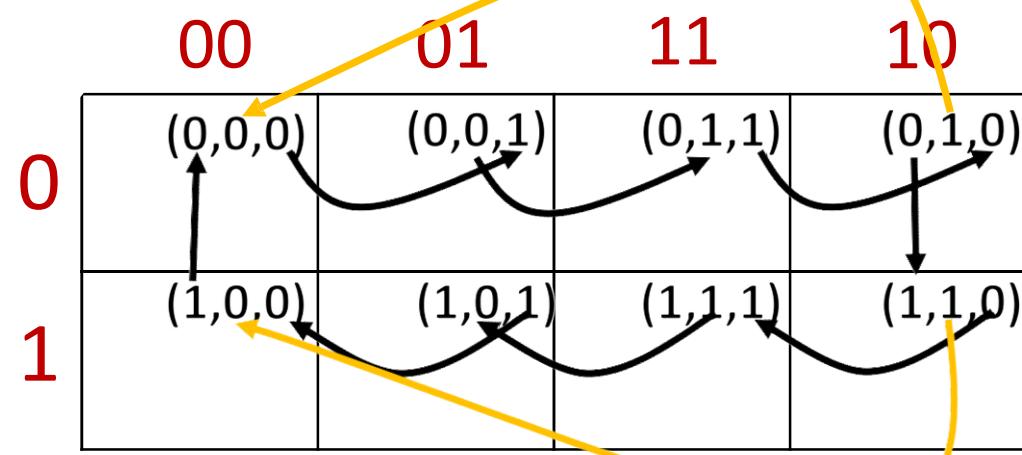
## 3-Variable K-Map

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



# 3-Variable K-Map

Adjacent Cells

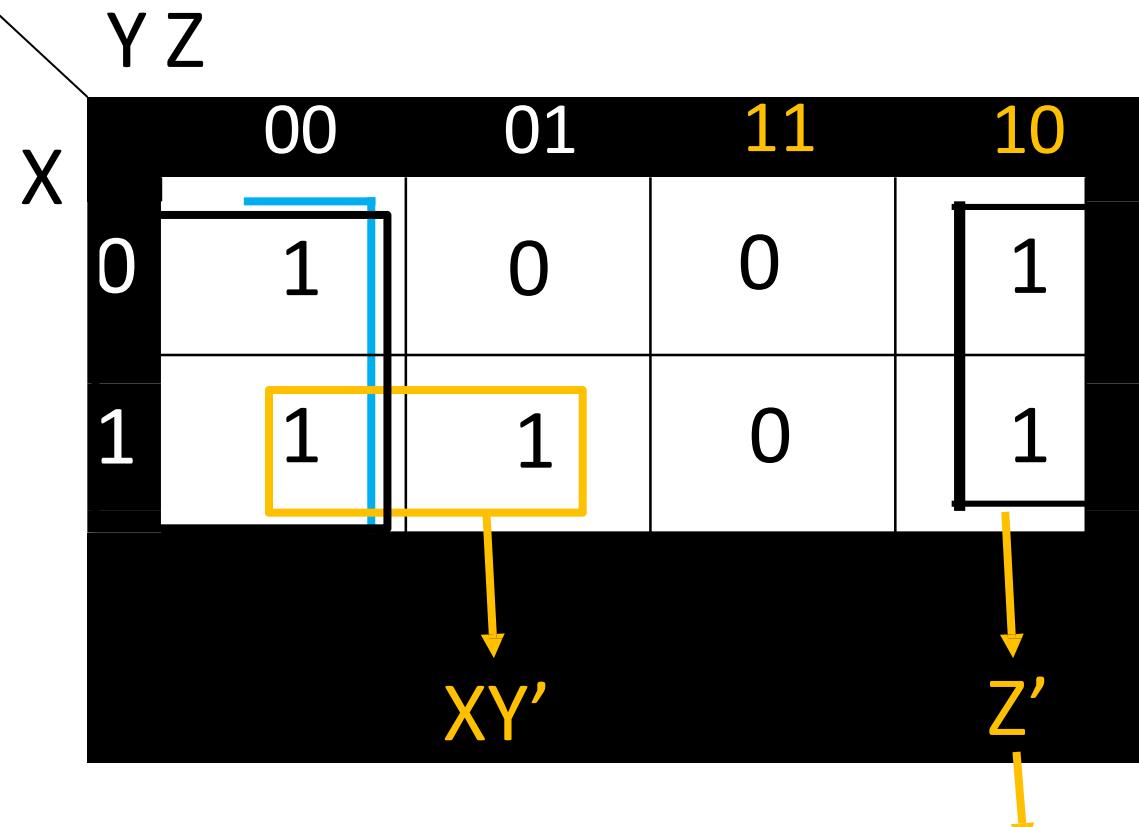


# 3-Variable K-Map : Example

$$F(X,Y,Z) = \sum (0,2,4,5,6)$$

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

2- cells or 4-cells or 8-cells at a time



Final Expression:  $F = XY' + Z'$

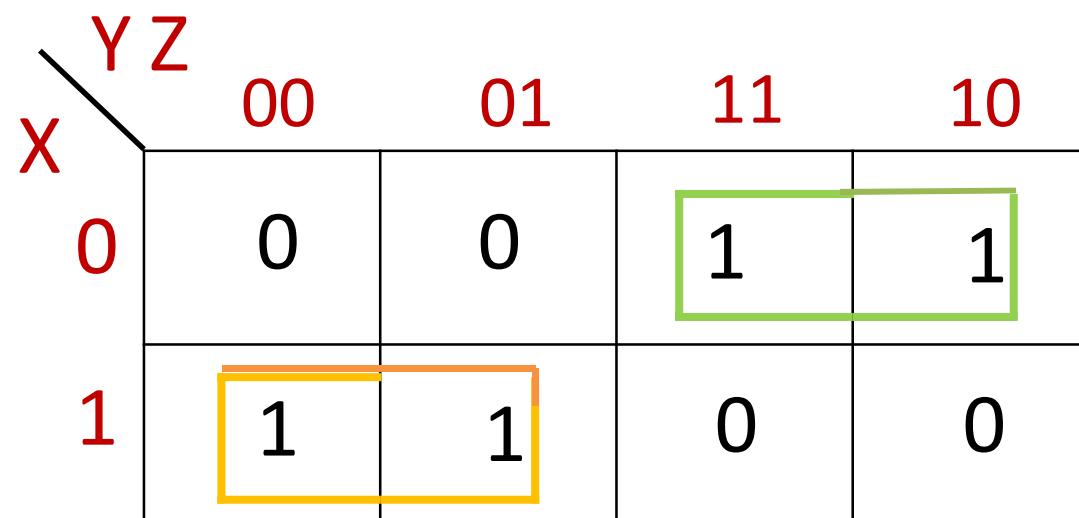
$Z=0$ , constant , X , Y Change

# 3-Variable K-Map

$$F(X,Y,Z) = \sum (2,3,4,5)$$

2- cells or 4-cells or 8-cells at a time

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



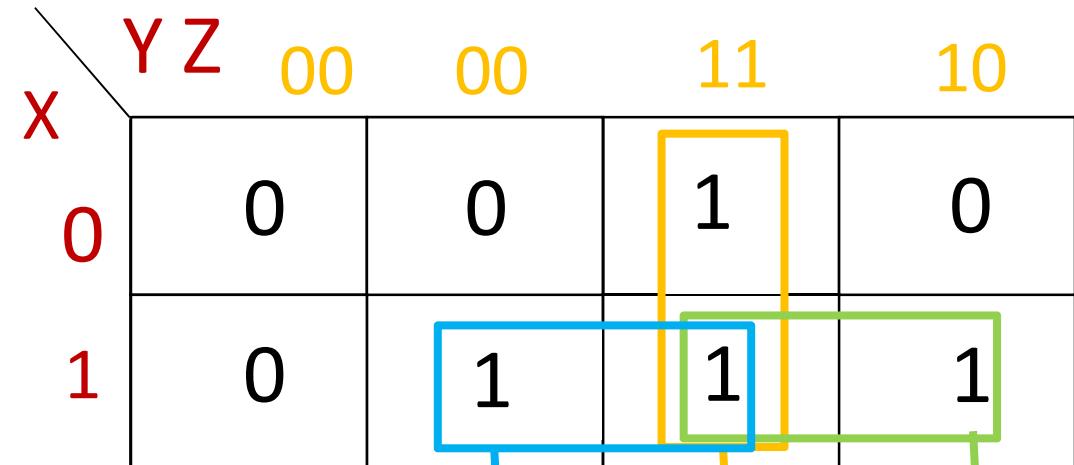
Final Expression:  $F = XY' + X'Y$

# 3-Variable K-Map

$$F(X,Y,Z) = \sum (3, 5, 6, 7)$$

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

2- cells or 4-cells or 8-cells at a time



X=1, Z=1 constant , Y varies

Y=1, Z=1 constant , X varies

X=1, Y=1 constant , Z varies

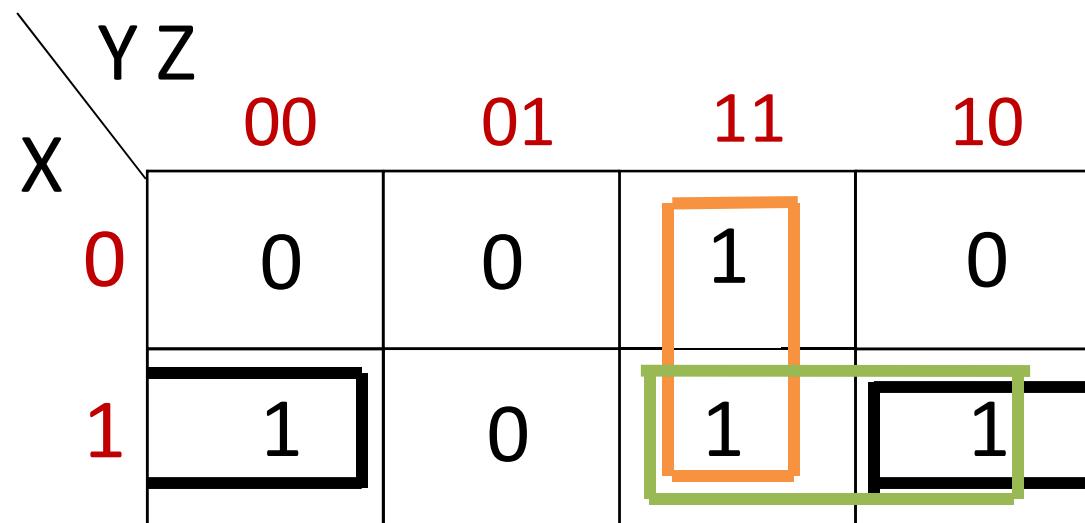
**Final Expression:  $F = XY + YZ + XZ$**

## 3-Variable K-Map

$$F(X,Y,Z) = \sum (3,4,6,7)$$

X	Y	Z	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

2- cells or 4-cells or 8-cells at a time



Is the third grouping (Green) necessary ?

All 'Ones' should get covered atleast once

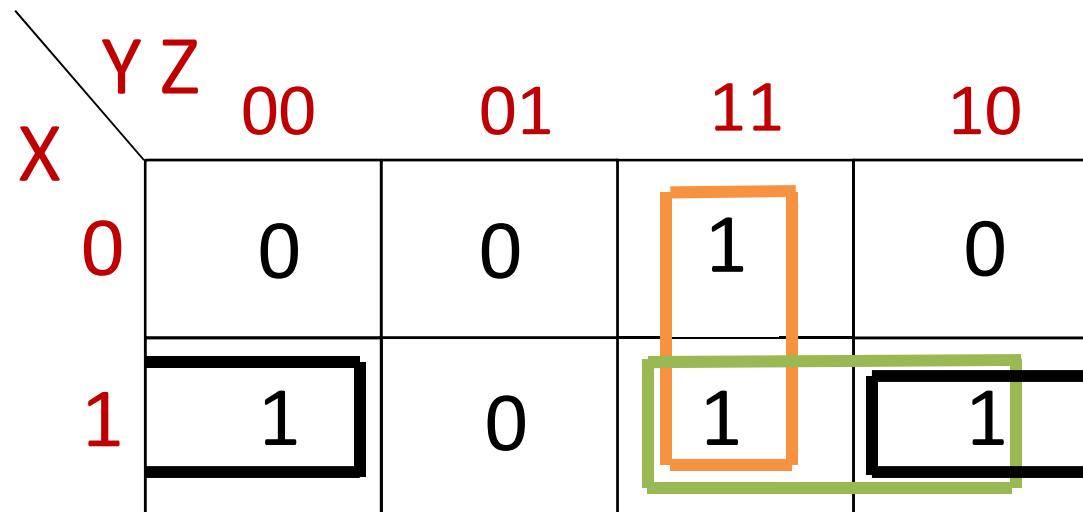
If all of them are covered then there is no need to group them again, overlap is allowed only if it is helping in further reduction of expression

## 3-Variable K-Map

$$F(X,Y,Z) = \sum(3,4,6,7)$$

X	Y	Z	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

2- cells or 4-cells or 8-cells at a time



$$F = YZ + XZ'$$

$$F = YZ + XZ' + XZ'$$

Only the minimized expression will be considered for digital circuit design (Less no. of Logic Gates),  
 \*Green coloured group is finally discarded



## Home assignment:

Simplify the Boolean function using K-map

$$F(x,y,z) = \sum m(1,3,5,6)$$



*Thank You*

# Digital Design

## CS / EEE / ECE/ INSTR F215

### Lecture 5:

#### Simplification of Boolean Expressions – using ‘K’ Map

Examples of 3 variable K-map (SOP, POS expressions)

4 Variable K-map



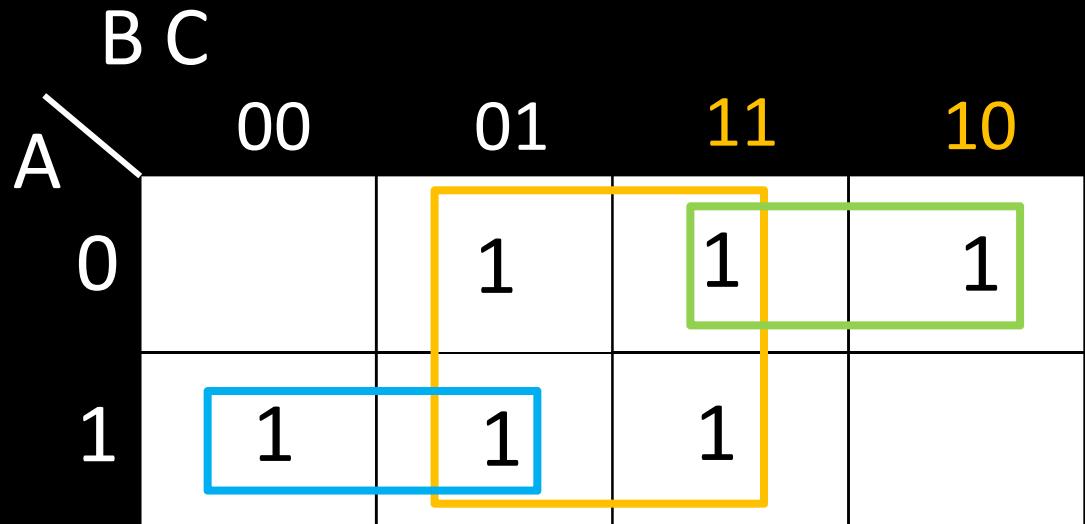
**Birla Institute of Technology & Science, Pilani**  
Hyderabad Campus  
BITS-Pilani, Hyderabad Campus





# Solve Using K-Map

$$F(A,B,C) = A'C + AB'C + BC + AB' + A'BC'$$



$$F = C + A'B + AB'$$

# POS simplification by K-Map

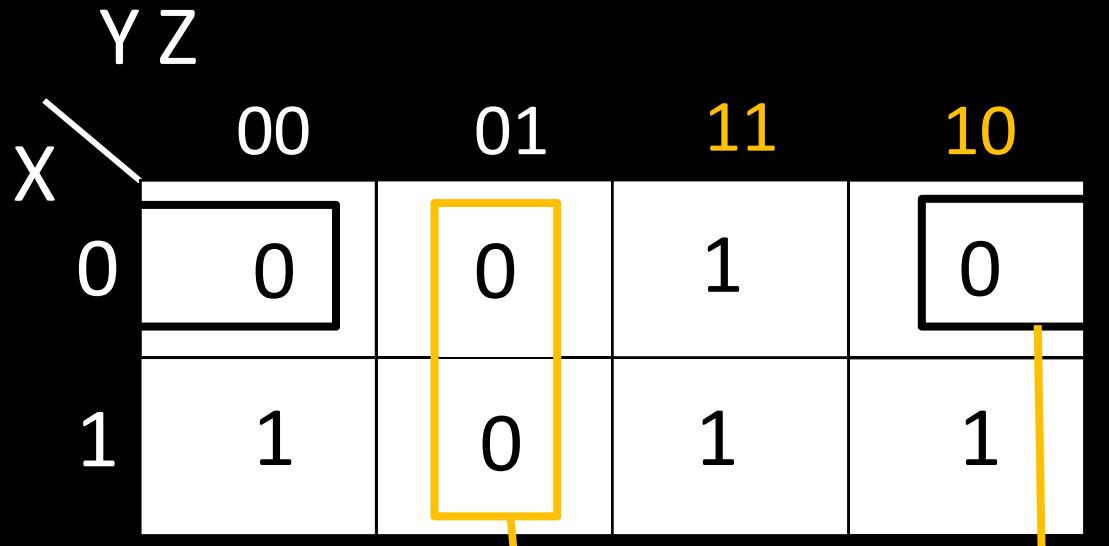


# 3-Variable K-Map

$$F(X,Y,Z) = \pi(0,1,2,5)$$

2- cells or 4-cells or 8-cells at a time

X	Y	Z	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



Y=0, Z=1 constant , X varies  $(Y+Z')$

X=0, Z=0 constant , Y varies

$$F = (Y + Z') (X + Z)$$

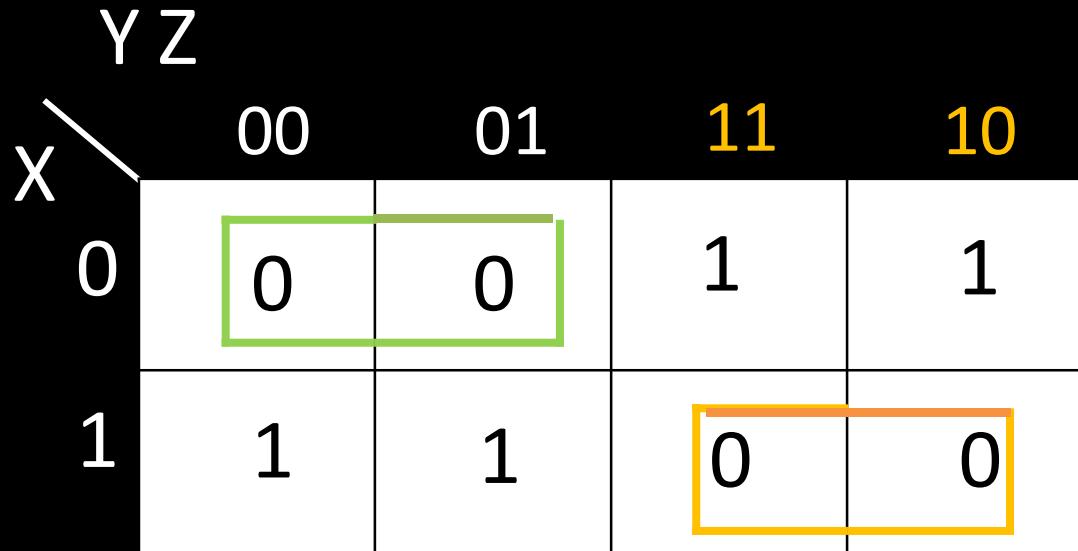


# 3-Variable K-Map

$$F(X,Y,Z) = \pi(0,1,6,7)$$

2-cells or 4-cells or 8-cells at a time

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



$$F = (X+Y)(X'+Y')$$

## **Don't Care Conditions**

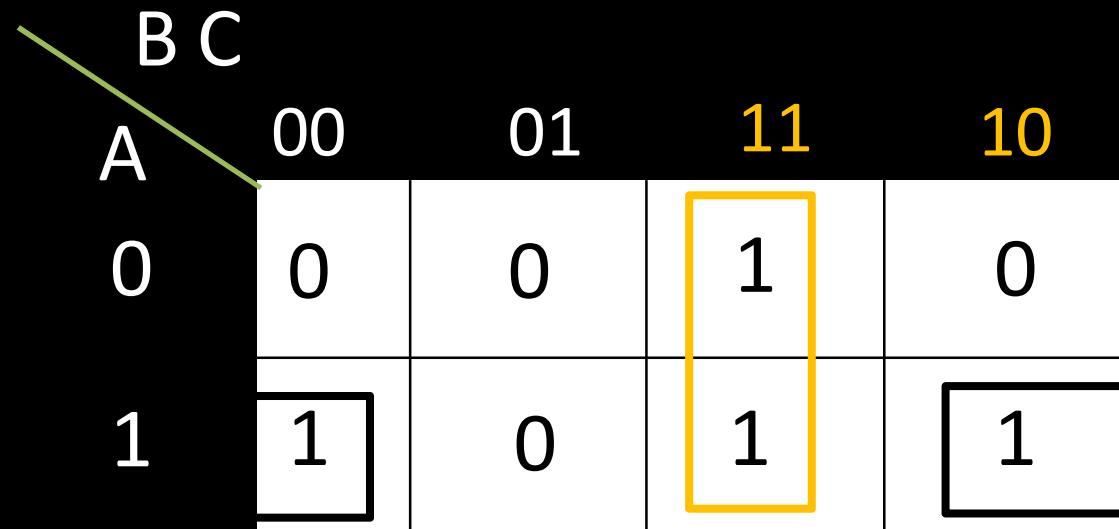
- Unspecified outputs for certain inputs
- Used in Map to provide further simplification
- 'X' is marked inside the square for don't care input
- Choose to include each don't care minterm/maxterm with either 1 or 0



# \*Don't care conditions

$$F(A,B,C) = \sum (3,4,6,7)$$

A	B	C	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

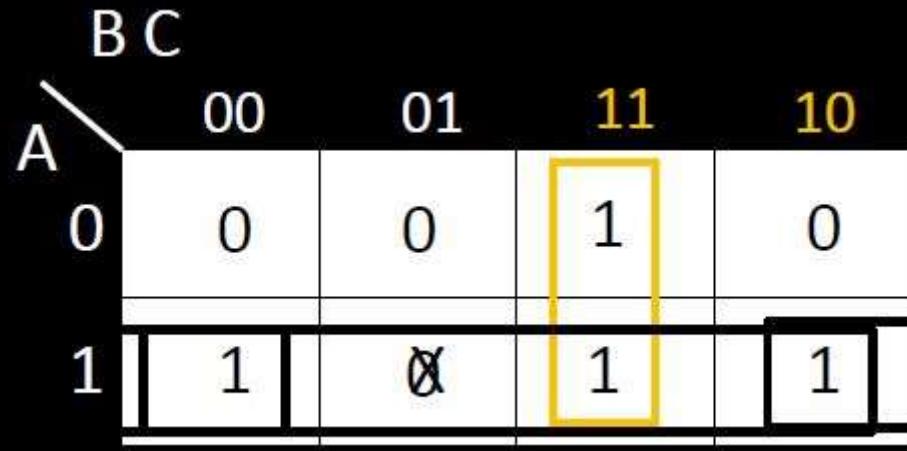




# Don't care conditions

$$F(A,B,C) = \sum (3,4,6,7)$$

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



$$F = BC + A$$

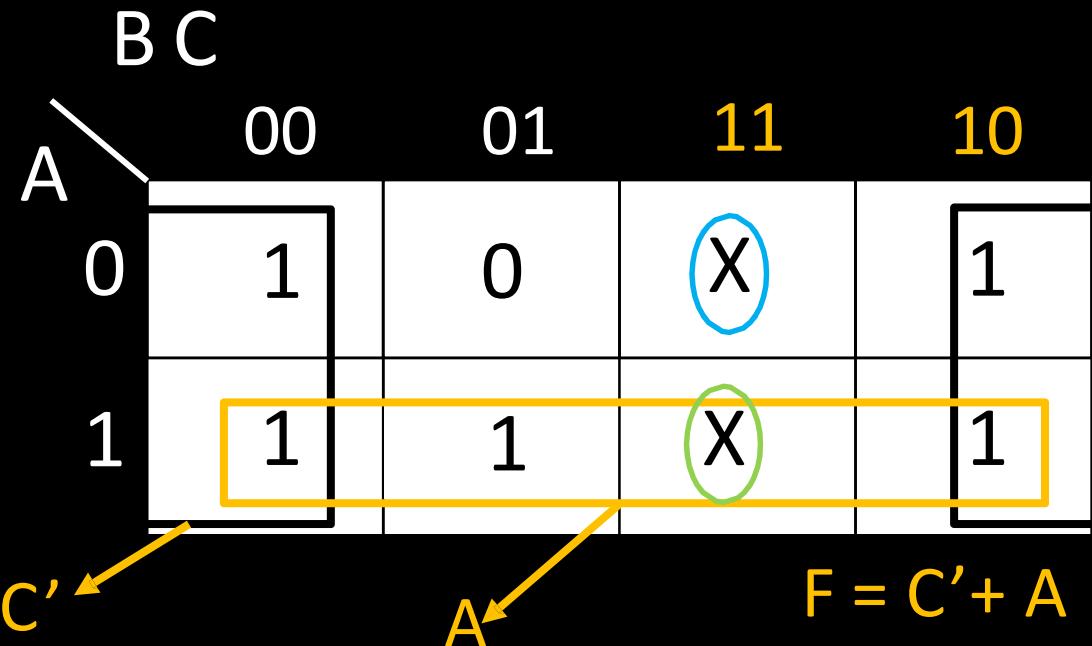
Earlier expression  $F = BC + AC'$



# Don't care conditions

$$F(A,B,C) = \sum (0,2,4,5,6) + d(3,7)$$

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



'X' Considered as 1 if it helps in reducing the terms or literals

'X' taken as 0 and neglected if it does not reduce terms or literals



# 4-Variable K-Map

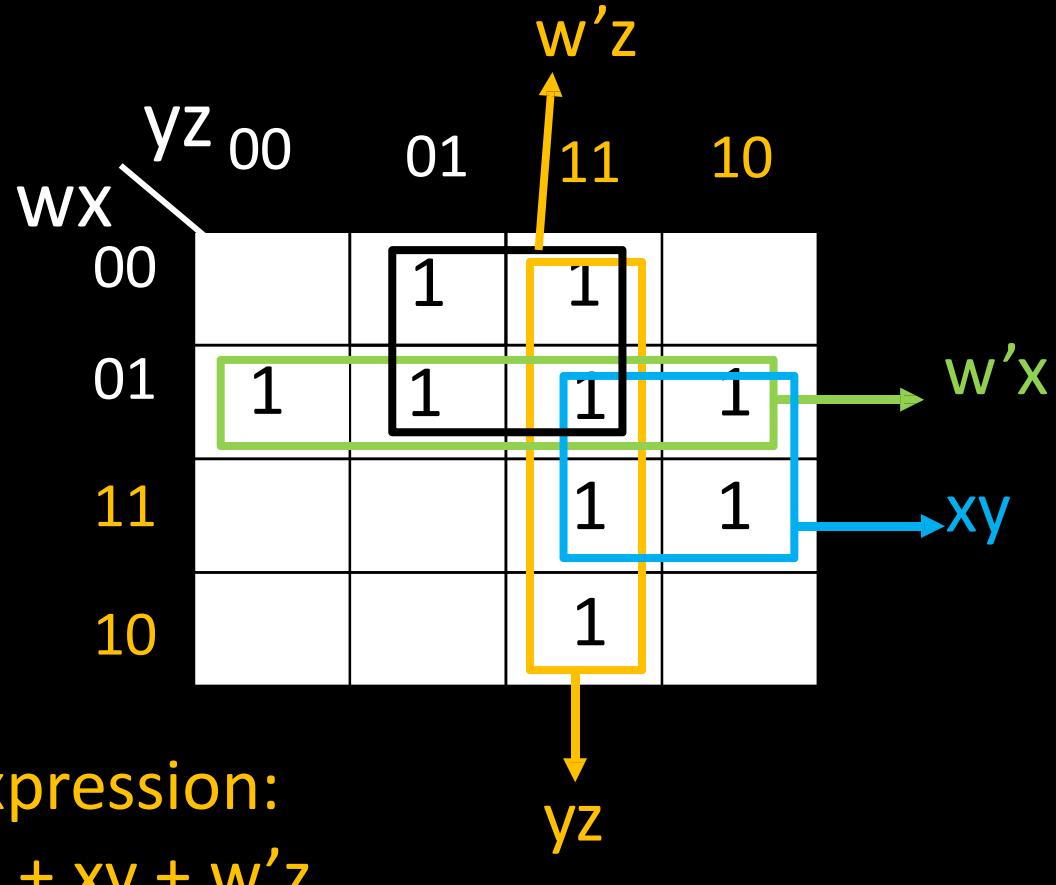
		YZ	00	01	11	10
		WX	00	01	11	10
00	00	(0,0,0,0) $m_0$	(0,0,0,1) $m_1$	(0,0,1,1) $m_3$	(0,0,1,0) $m_2$	
	01	(0,1,0,0) $m_4$	(0,1,0,1) $m_5$	(0,1,1,1) $m_7$	(0,1,1,0) $m_6$	
	11	(1,1,0,0) $m_{12}$	(1,1,0,1) $m_{13}$	(1,1,1,1) $m_{15}$	(1,1,1,0) $m_{14}$	
	10	(1,0,0,0) $m_8$	(1,0,0,1) $m_9$	(1,0,1,1) $m_{11}$	(1,0,1,0) $m_{10}$	

Verify 1-bit variations between adjacent cells



# 4-Variable K-Map

$$F(w,x,y,z) = \sum(1,3,4,5,6,7,11,14,15)$$





# Thank You

# Digital Design

CS / EEE / ECE/ INSTR F215

## Lecture 6: Four and Five Variable K-map

Examples of 4 Variable K-map (Concept of Implicant, prime implicant and essential prime implicant)

5 Variable K-map



**Birla Institute of Technology & Science, Pilani**  
Hyderabad Campus

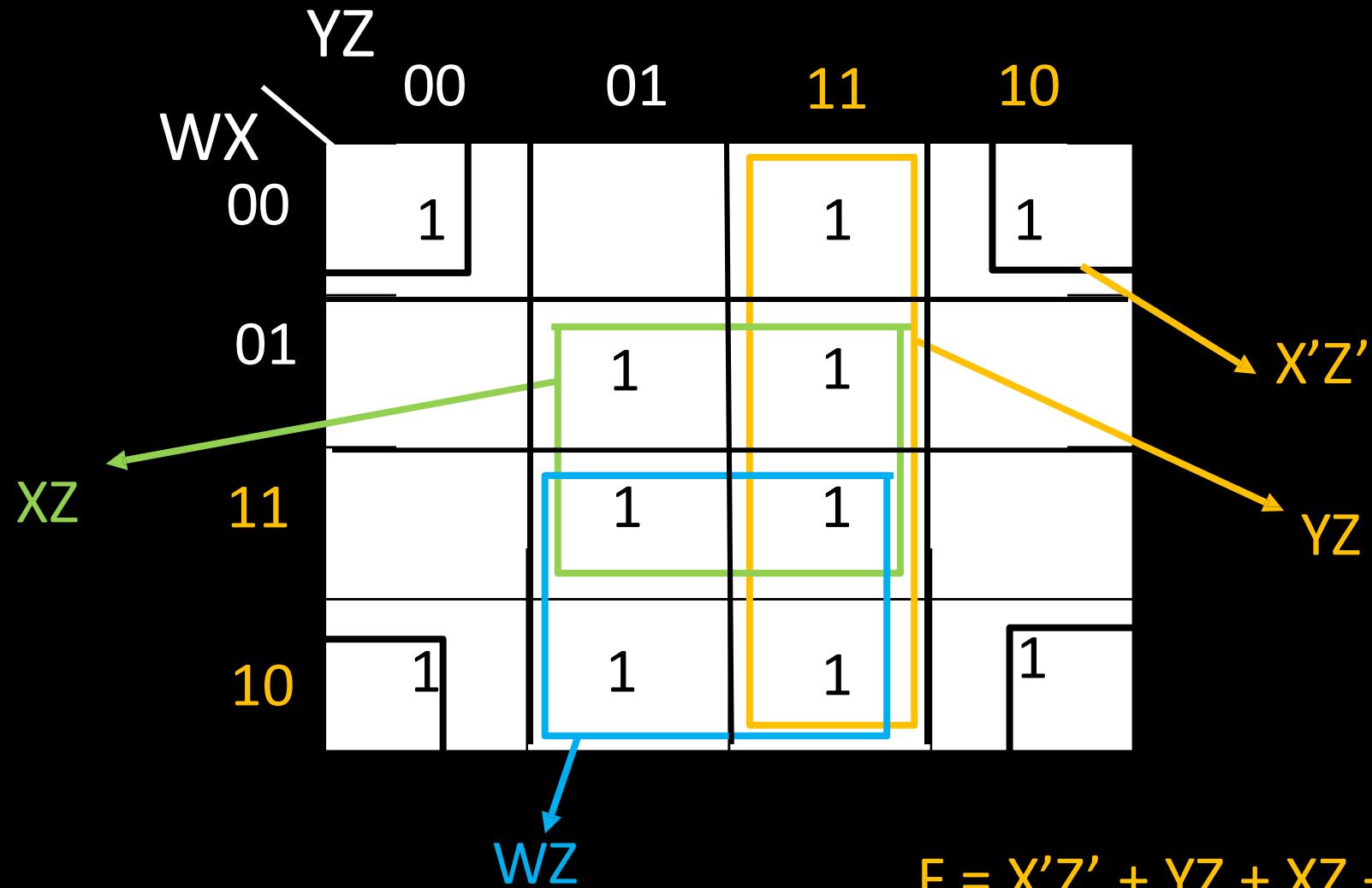


# 4-Variable K-Map: Example

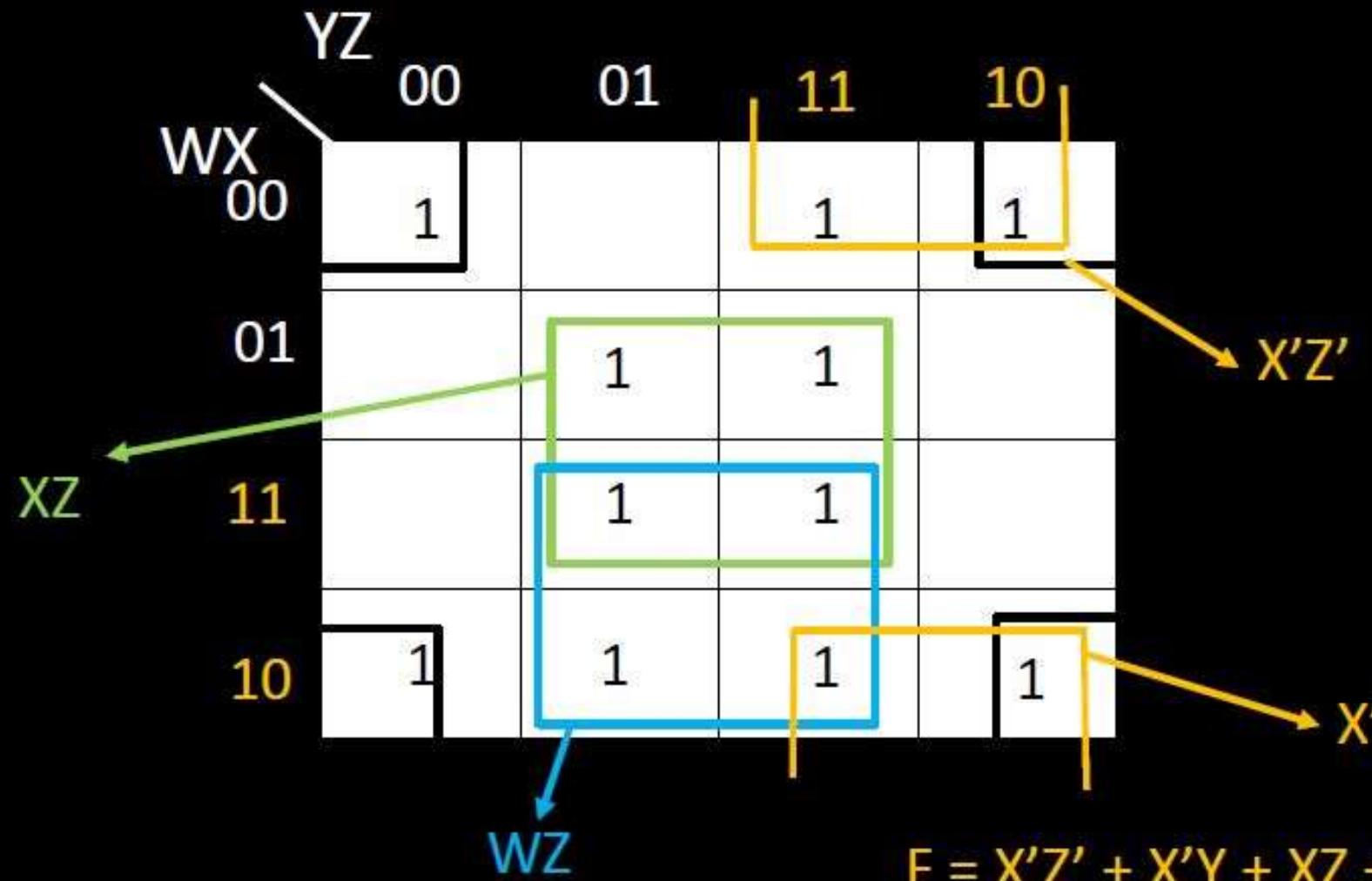
		YZ	00	01	11	10
		WX	00	01	11	10
Y	X	00	1		1	1
		01		1	1	
		11		1	1	
		10	1	1	1	1

$$F(w,x,y,z) = \sum (0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$$

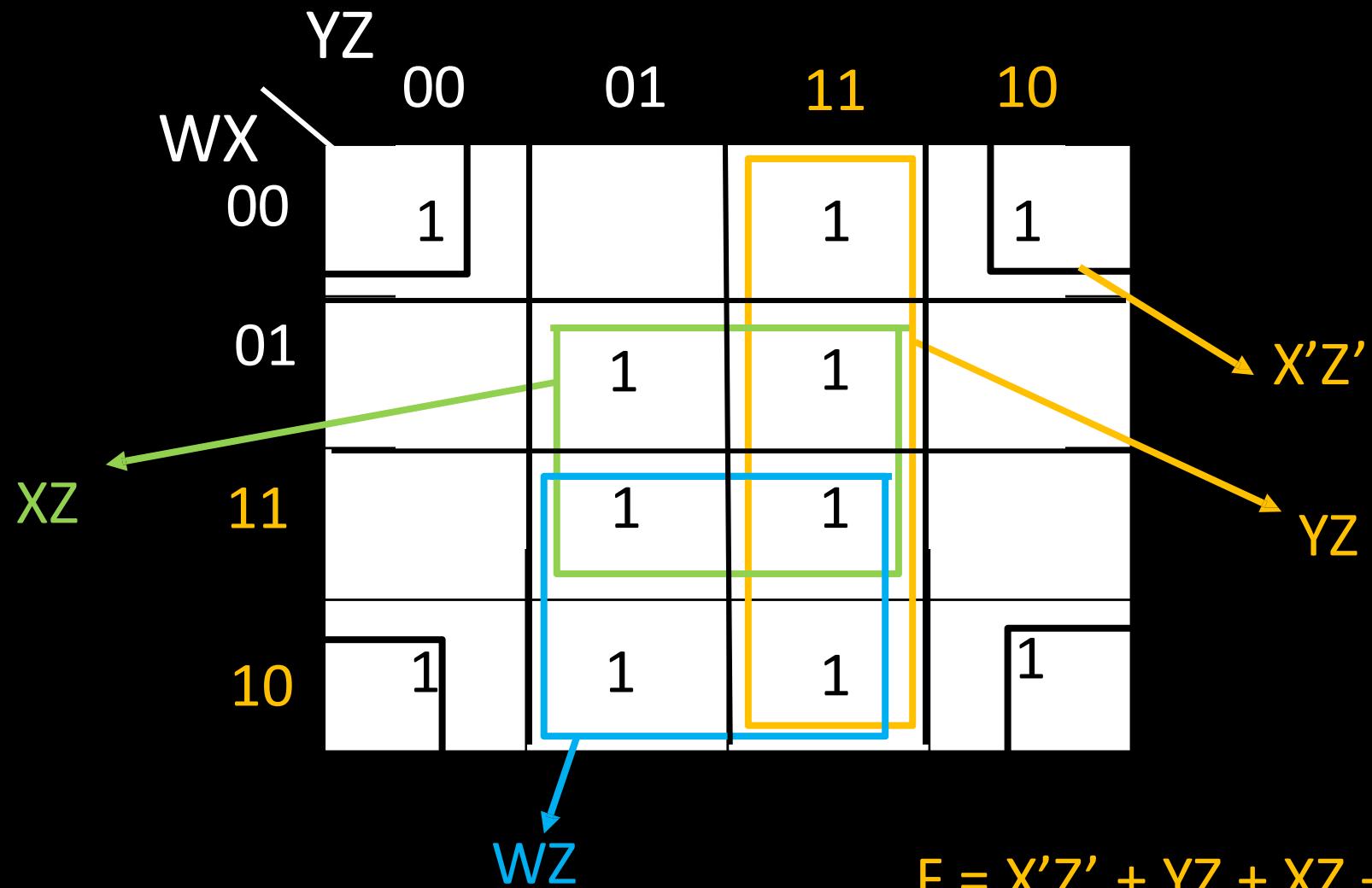
# 4-Variable K-Map



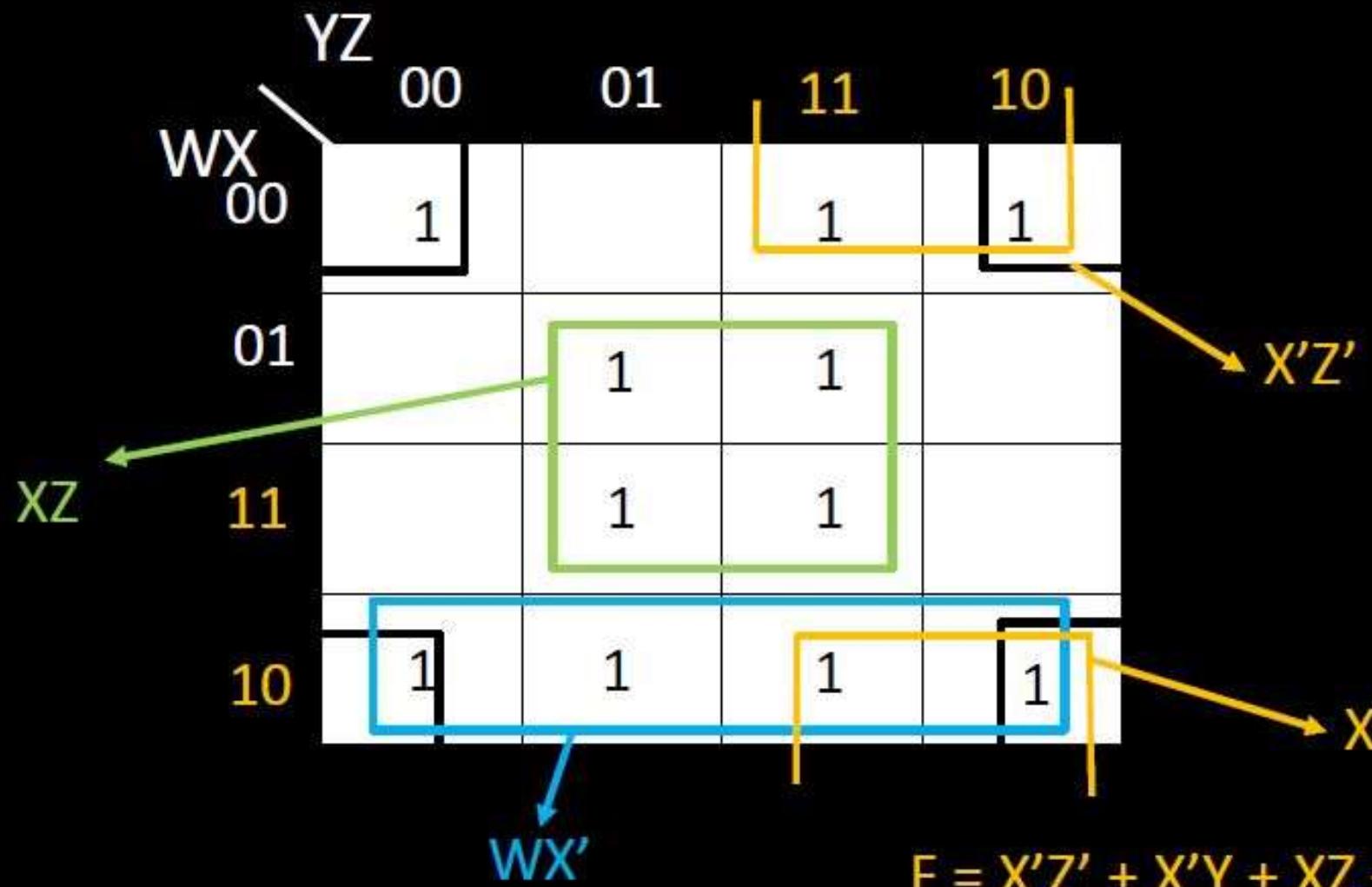
## 4-Variable K-Map



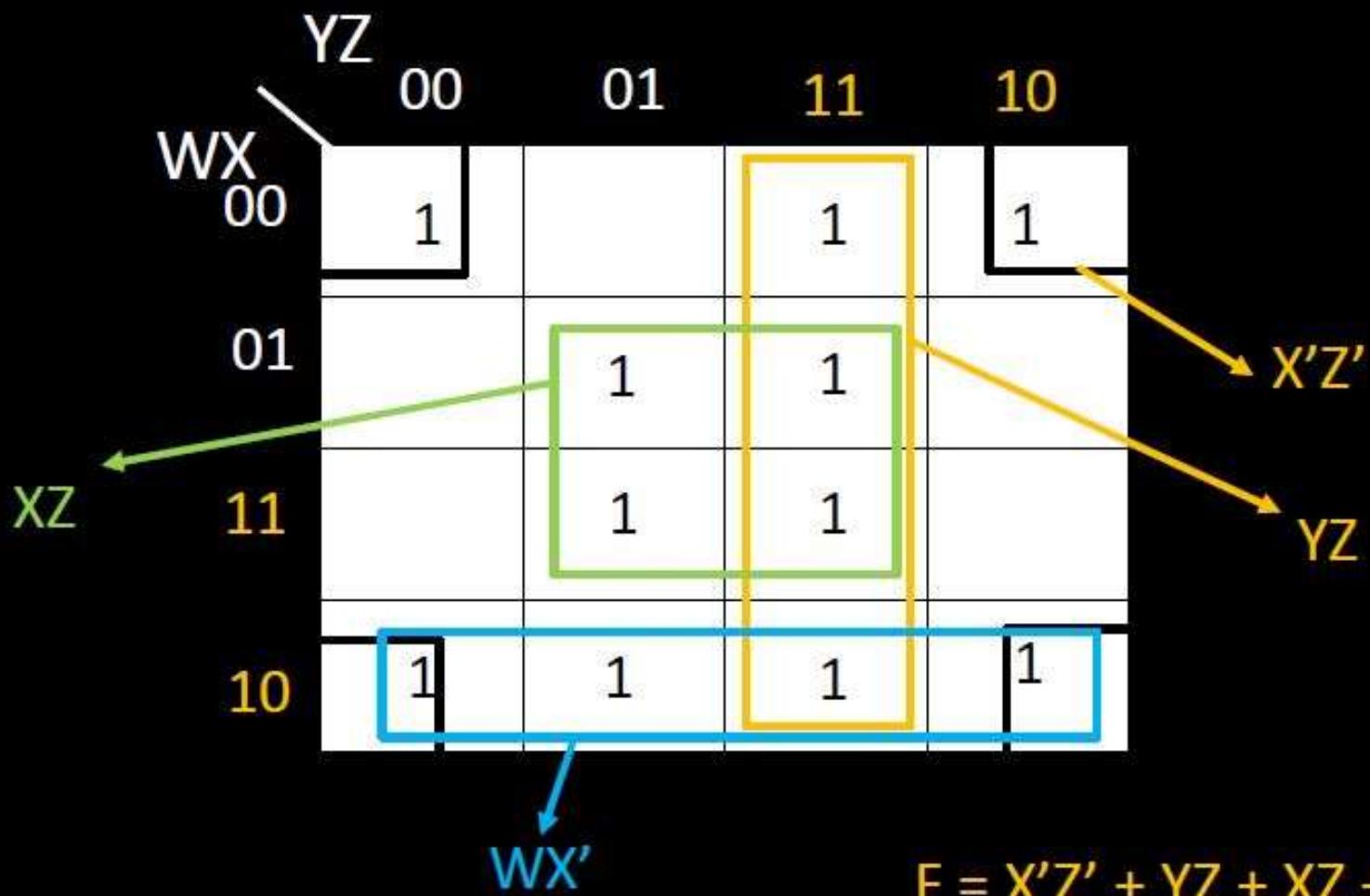
# 4-Variable K-Map



# 4-Variable K-Map



# 4-Variable K-Map



# 4-Variable K-Map

$$F = X'Z' + YZ + XZ + WZ$$

$X'Z'$ ,  $YZ$ ,  $XZ$ ,  $WZ$ ,  $X'Y$ ,  $WX'$  are  
Prime Implicants

$$F = X'Z' + X'Y + XZ + WZ$$

$$F = X'Z' + X'Y + XZ + WX'$$

$$F = X'Z' + YZ + XZ + WX'$$

All are minimized and all are correct solutions

What are the terms common here ??

$X'Z'$  and  $XZ$  appear in all the solutions

$X'Z'$  and  $XZ$  are called Essential Prime Implicants

# 1. Implicant of Functions

Implicant is a product/minterm term in Sum of Products (SOP) or sum/maxterm term in Product of Sums (POS) of a Boolean function.

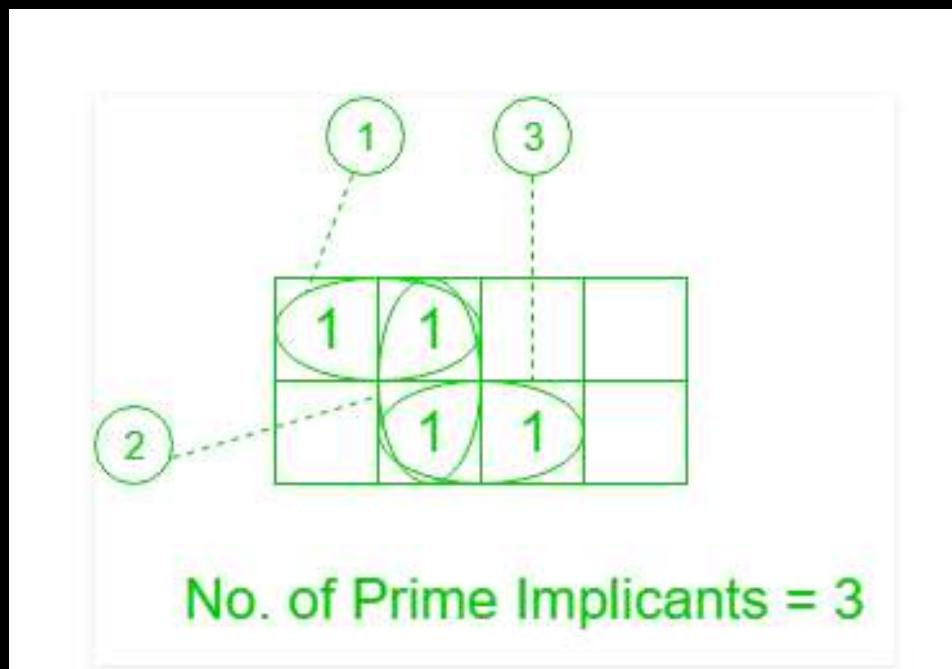
E.g.,

Consider a boolean function,  $F = AB + ABC + BC$ .  
Implicants are AB, ABC and BC.

**Implicant:** For a Boolean function F expressed in SOP form, a product term p is an implicant of the Function F if and only if  $F=1$  for every combination of the input values to the variables of the product term p, for which  $p=1$ .

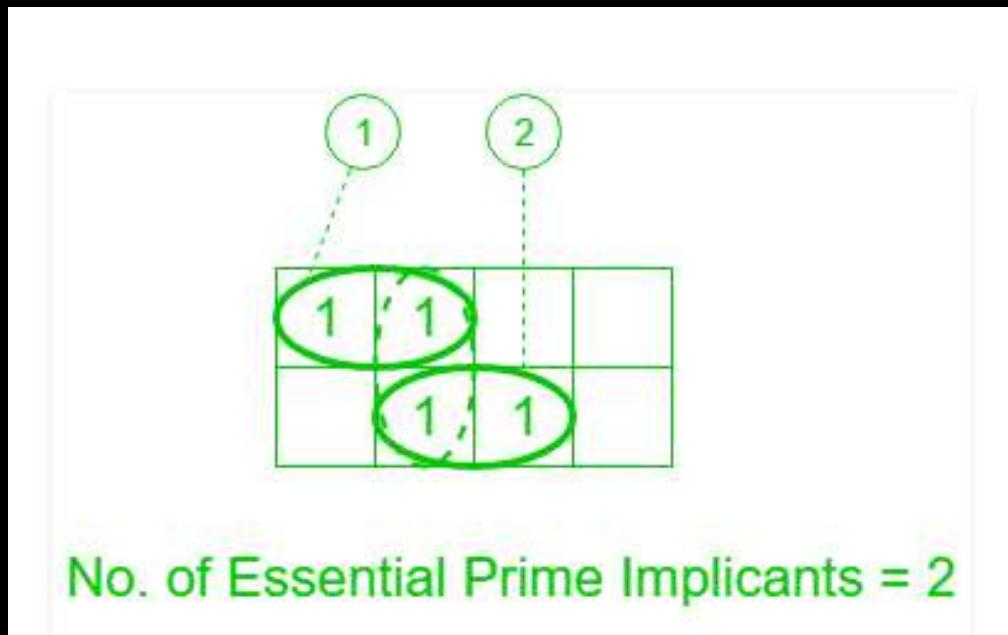
## 2. Prime Implicant of Functions

A group of square or rectangle made up of bunch of adjacent minterms which is allowed by definition of K-Map are called **prime implicants(PI)** i.e. all possible groups formed in K-Map.



### 3. Essential Implicant of Functions

These are those subcubes(groups) which cover atleast one minterm that can't be covered by any other prime implicant. Essential prime implicants(EPI) are those prime implicants which always appear in final solution..



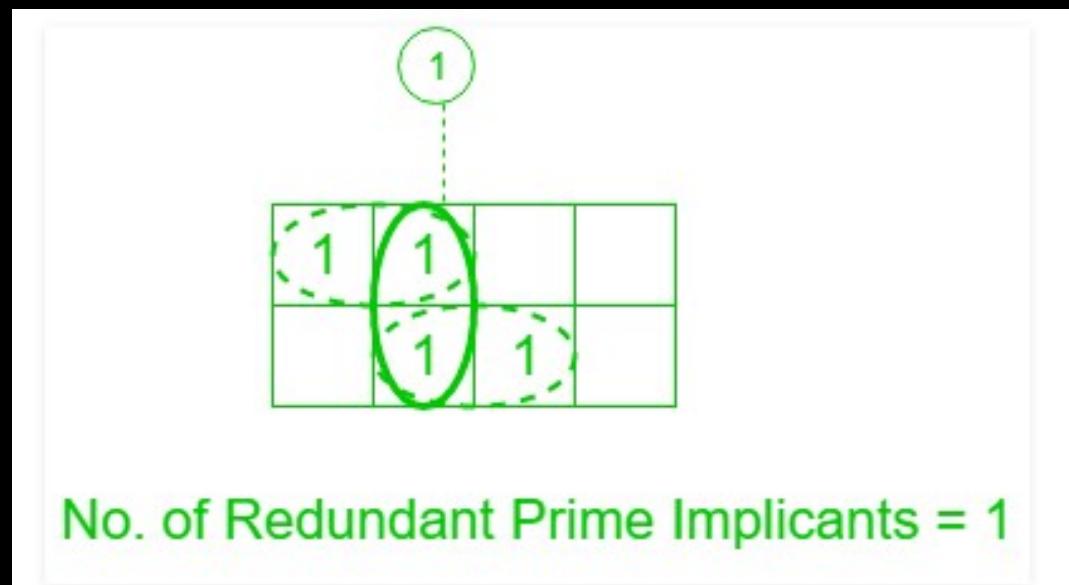
**Implicant:** For a Boolean function  $F$  expressed in SOP form, a product term  $p$  is an implicant of the Function  $F$  if and only if  $F=1$  for every combination of the input values to the variables of the product term  $p$ , for which  $p = 1$ .

**Prime Implicant:** An implicant of a function  $F$  is a prime implicant of  $F$  if it is not completely enclosed in bigger valid group of minterms (or maxterms) in the K-map.

**Essential Prime Implicant:** A prime implicant of a function  $F$  is an essential prime implicant of  $F$  if the loop for prime implicant contains at least one minterm (maxterm) box that is not contained in any other prime implicant loop of  $F$ .

## 4. Redundant Implicant of Functions

The prime implicants for which each of its minterm is covered by some essential prime implicant are redundant prime implicants(RPI). This prime implicant never appears in final solution..



# 4-Variable K-Map

		YZ	00	01	11	10
		WX	00	01	11	10
00	00	1		1	1	
	01		1	1		
	11		1	1		
	10	1	1	1	1	

How to find essential prime Implicants ?

Check how many ways a minterm can be covered ?

Select all minterms that are covered in only one way

Minimized expressions corresponding to minterms that are covered in only one way are essential prime implicants

# 4-Variable K-Map

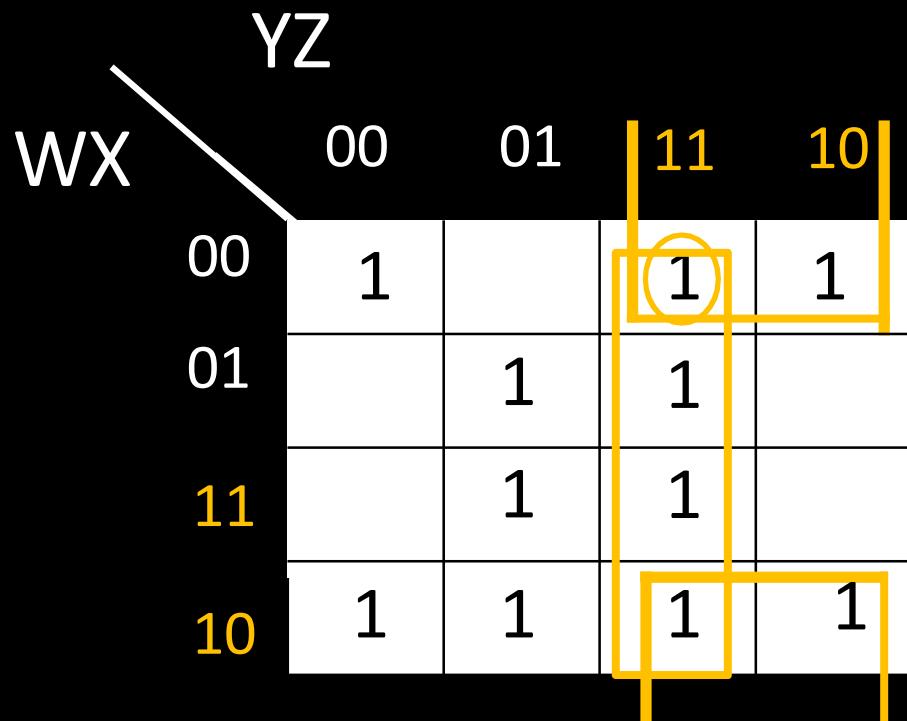
		YZ	00	01	11	10
		WX	00	01	11	10
00	01	00	1		1	1
		01		1	1	
		11		1	1	
		10	1	1	1	1

$m_0$  can be covered in only one way

Corresponding expression  
 $X'Z'$

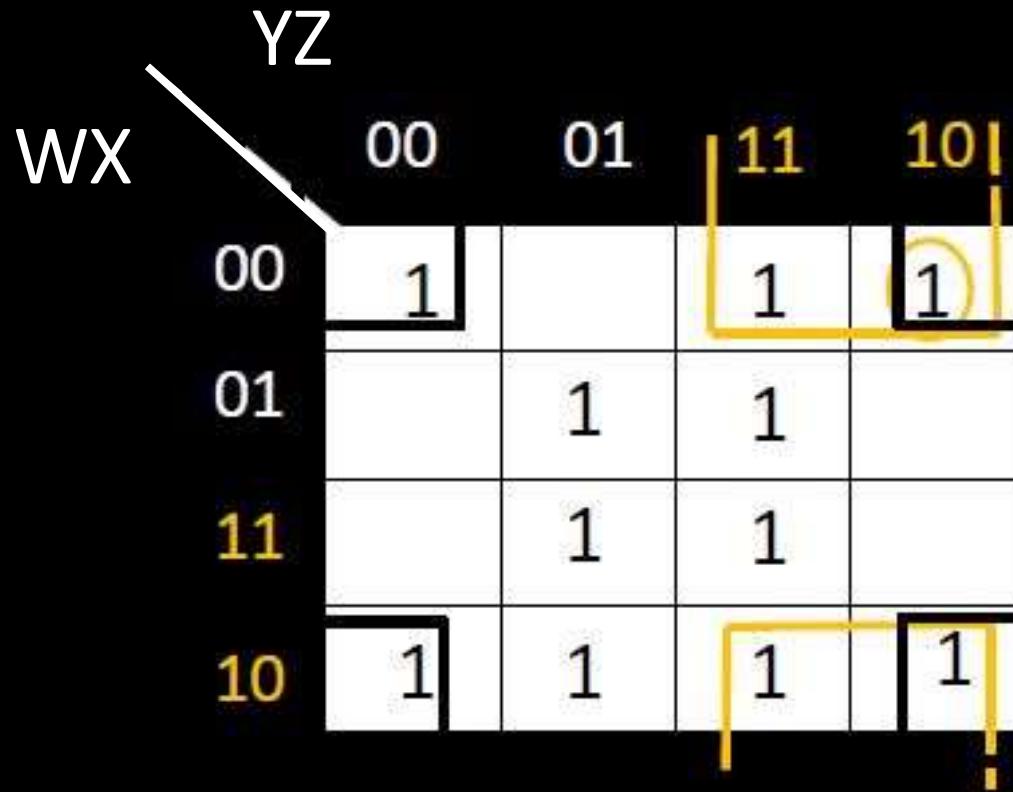
Is there any way that  $m_0$  can be covered ??

# 4-Variable K-Map



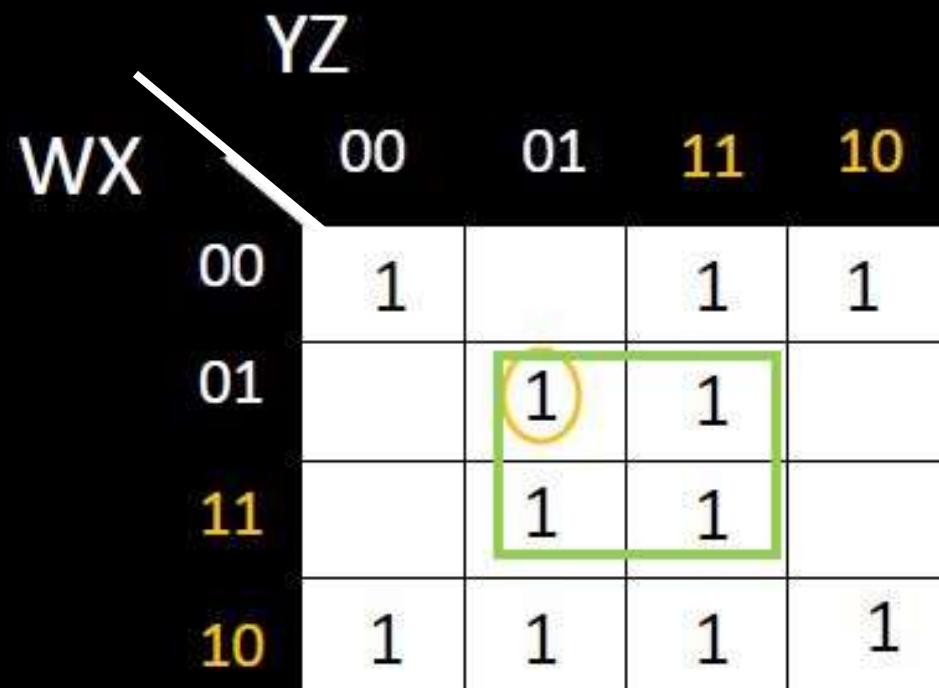
$m_3$  can be covered in multiple ways

# 4-Variable K-Map



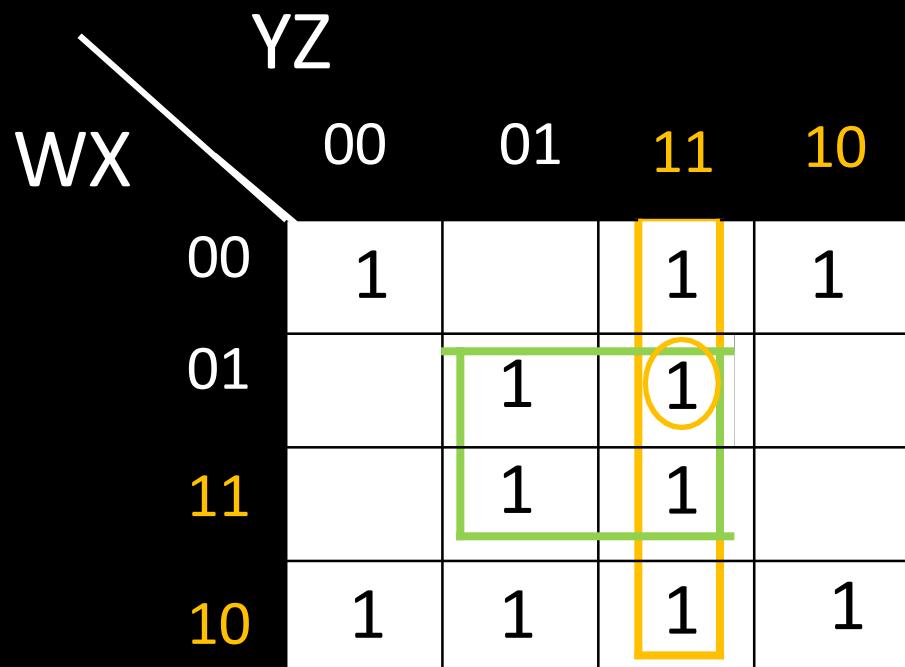
$m_2$  can be covered in multiple ways

# 4-Variable K-Map



Corresponding expression is  $XZ$

# 4-Variable K-Map



Similarly it can be analyzed that  $m_7, m_8, m_9, m_{11}, m_{13}, m_{15}$  can be covered in multiple ways

# 4-Variable K-Map

		YZ	00	01	11	10
		WX	00	01	11	10
00	00	1		1	1	
	01		1	1		
	11		1	1		
	10	1	1	1	1	

Only  $m_0$  and  $m_5$  can be covered in only one way

Hence the corresponding expressions  $XZ$  and  $X'Z'$  are called  
**Essential prime Implicants**



CKV

*Thank You*

# Digital Design

CS / EEE / ECE/ INSTR F215

## Lecture 7: 5 Variable K-map



**Birla Institute of Technology & Science, Pilani**  
Hyderabad Campus



# 5- Variable K –Map

## (Using two 4 variable K-Maps)

The structure of such a K-Map for SOP expression is given below :

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

P=0

F(P,Q,R,S,T)

		ST	00	01	11	10
		QR	16	17	19	18
00						
01			20	21	23	22
11			28	29	31	30
10			24	25	27	26

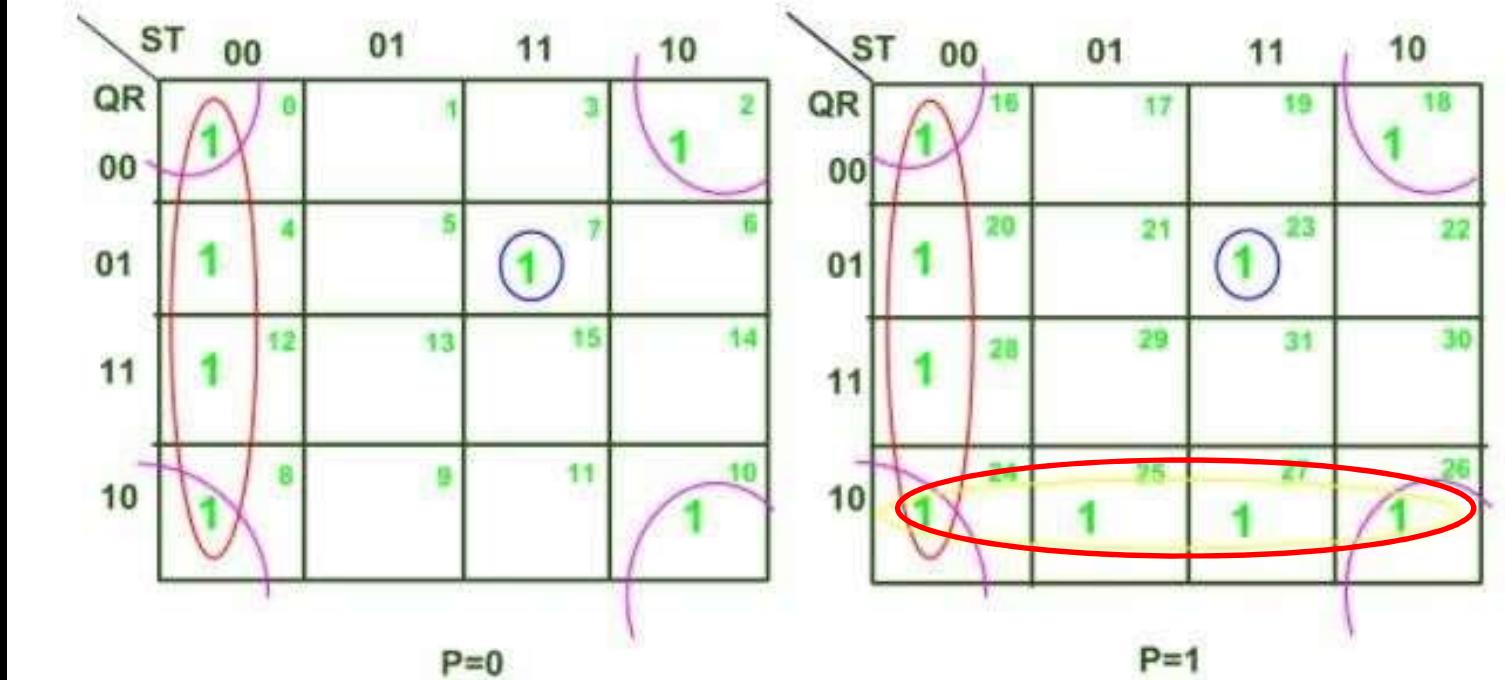
P=1

# Minimization of Boolean Expression by 5 Variable K-Map (SOP form)

## I. Solving SOP function –

For clear understanding, let us solve the example of SOP function minimization of 5 Variable K-Map using the following expression :

$$\sum m(0, 2, 4, 7, 8, 10, 12, 16, 18, 20, 23, 24, 25, 26, 27, 28)$$



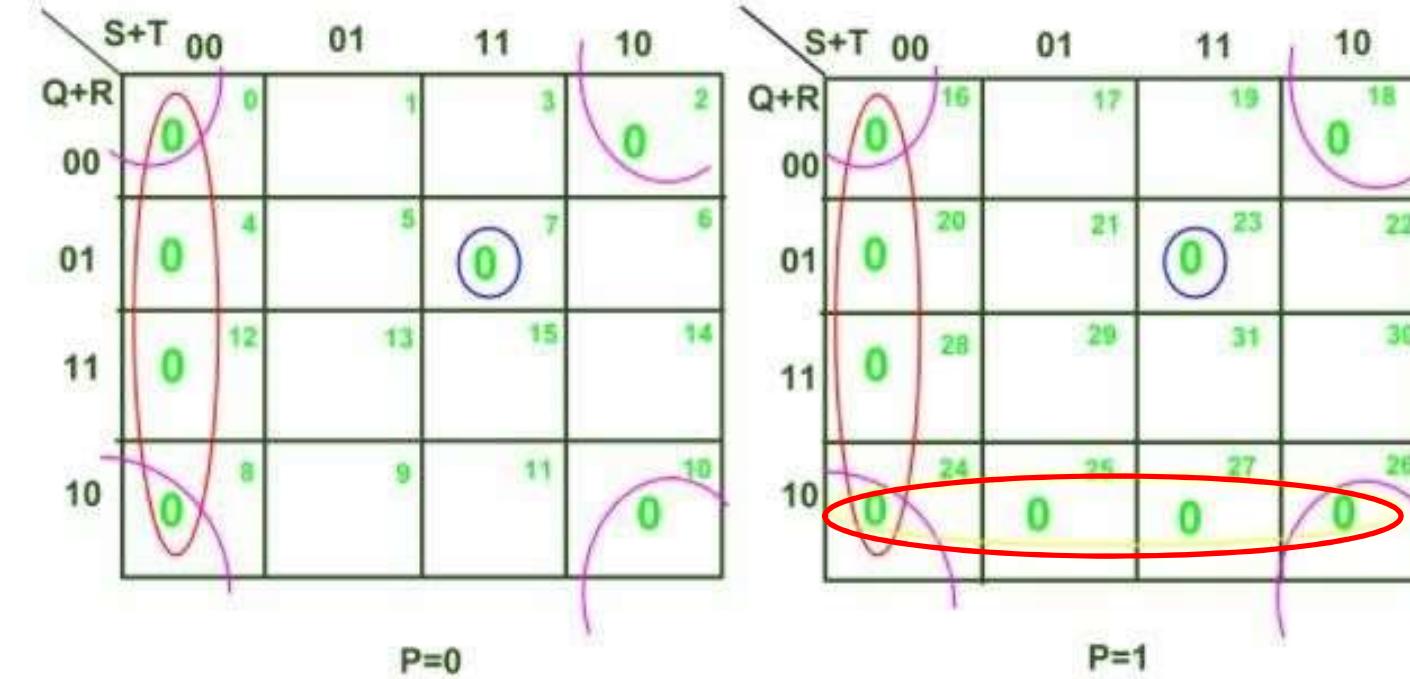
$$F(PQRST) = S'T' + Q' RST + PQR' + R'T'$$

# Minimization of Boolean Expression by 5 Variable K-Map (POS form)

## II. Solving POS function -

Now, let us solve the example of POS function minimization of 5 Variable K-Map using the following expression :

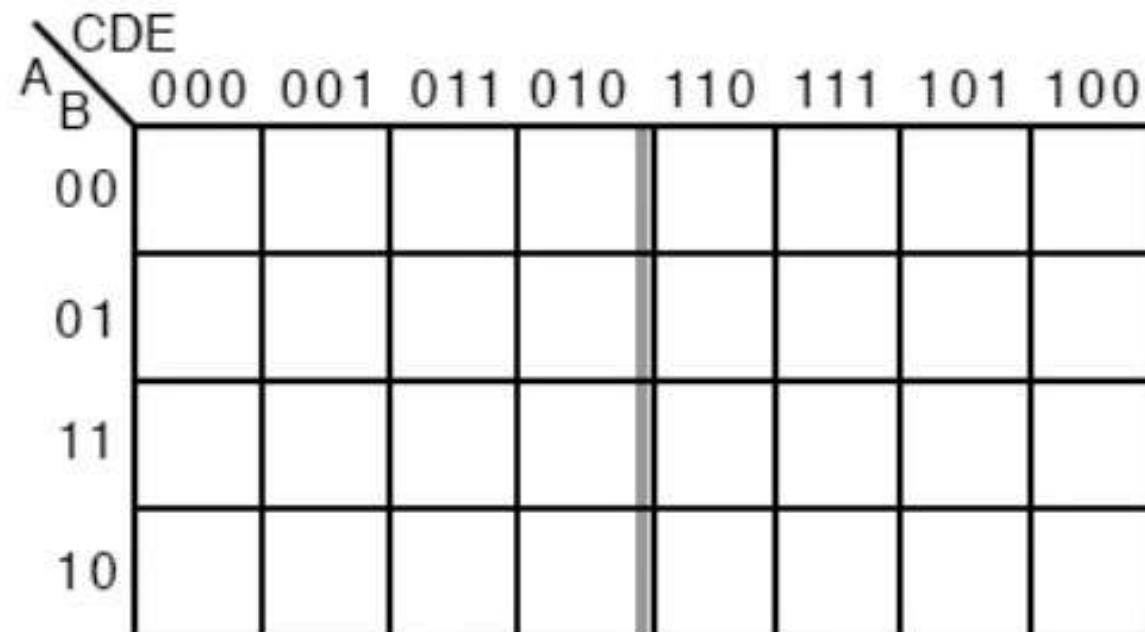
$$\prod M(0, 2, 4, 7, 8, 10, 12, 16, 18, 20, 23, 24, 25, 26, 27, 28)$$



$$f(PQRST) = (S+T) \cdot (R+T) \cdot (Q+R'+S'+T') \cdot (P'+Q'+R)$$

Another approach of 5 Variable K-Map

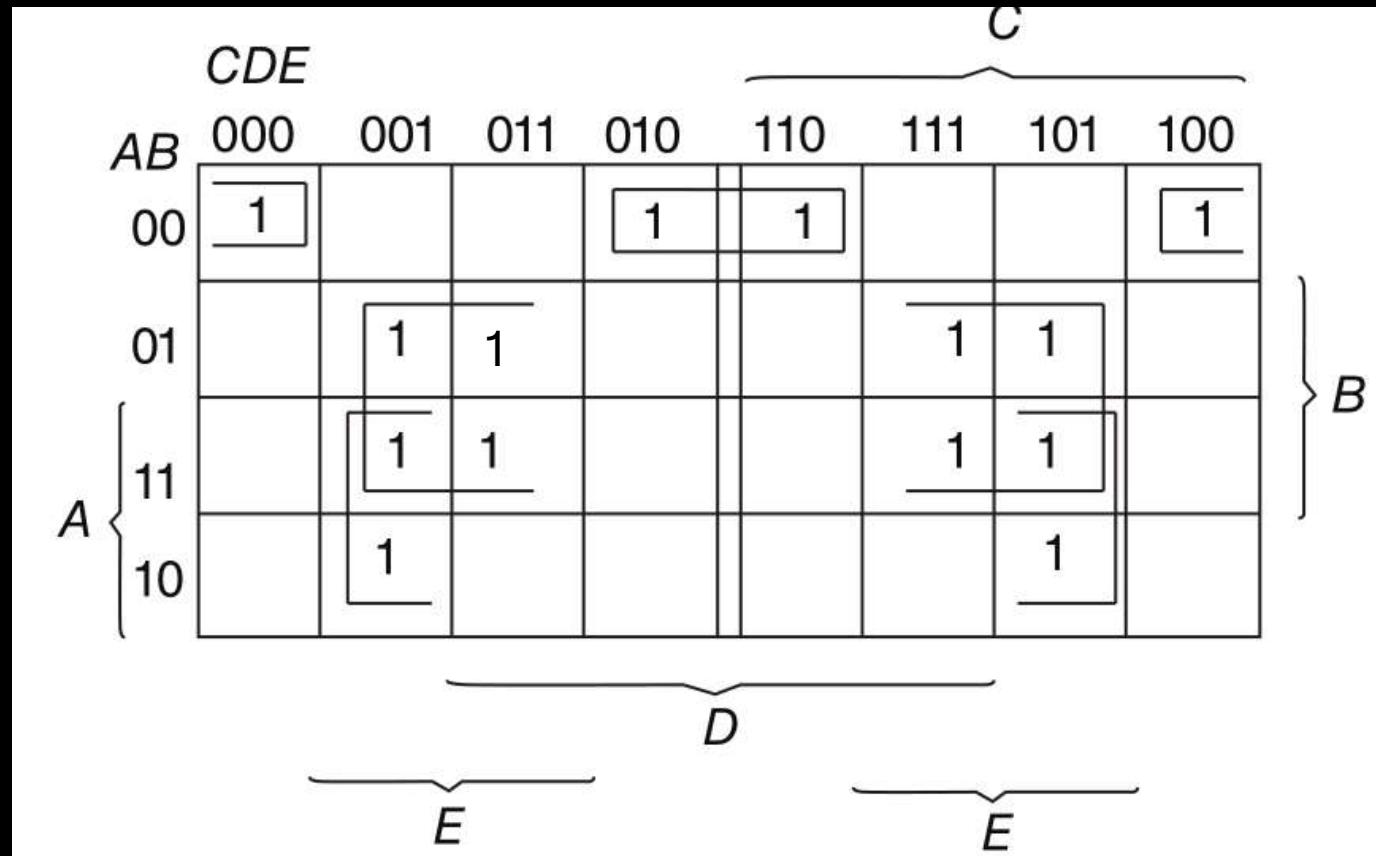
## Five Variable K-map



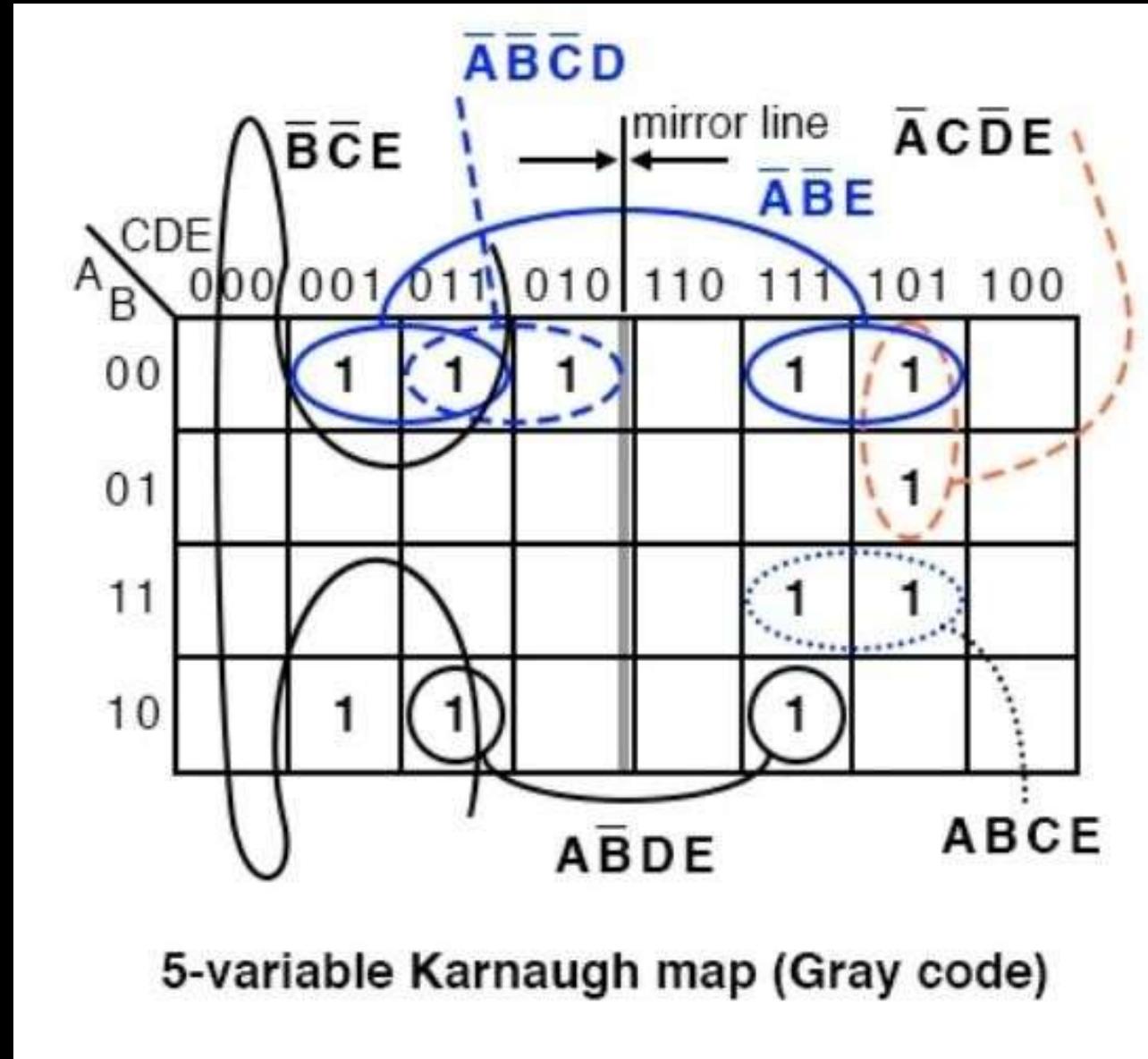
5-variable Karnaugh map (Gray code)



$$F(A, B, C, D, E) = \sum(0, 2, 4, 6, 9, 11, 13, 15, 17, 21, 25, 27, 29, 31)$$

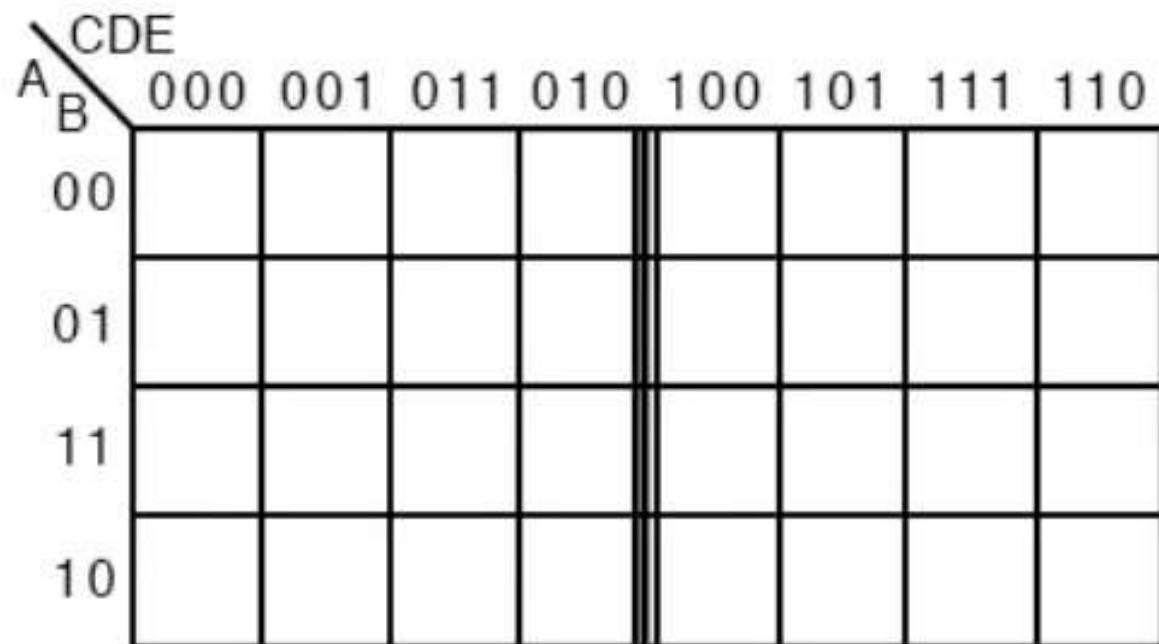


$$BE + AD'E + A'B'E'$$

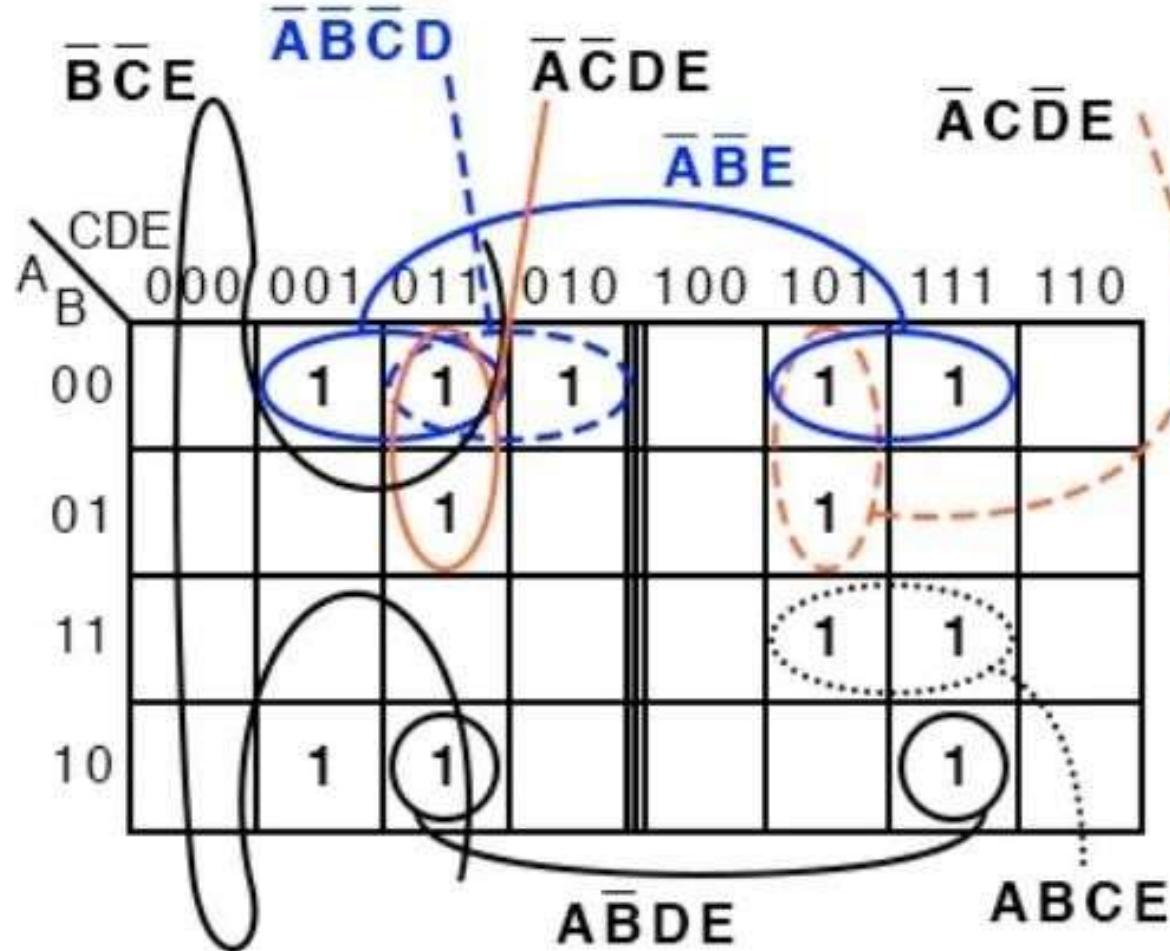




## Overlay Version of the K-map



5-variable Karnaugh map (overlay)



5-variable Karnaugh map (overlay)



# Thank You

# Digital Design

CS / EEE / ECE/ INSTR F215

Lecture 8: QM method for Minimization of Boolean Expressions



**Birla Institute of Technology & Science, Pilani**  
Hyderabad Campus





# Quine-McCluskey (QM) Method

QM Method is effective for increased number of input variables

K-map is a graphical method whereas QM is a Tabular Method



## Steps required for implementing Quine-McCluskey (QM) method

1. Find Prime Implicants of the function
2. Find Essential Prime implicants of the function.
3. Include Essential prime implicants in partial solution and delete them from the prime implicant table
4. Determine and delete dominated rows and dominating columns. Find the (secondary) prime implicants.
5. Repeat steps 3 and 4 as many times as they are applicable until a minimal cover of the function is found



## Example 1

$$f(w, x, y, z) = \sum(0, 1, 2, 5, 6, 7, 8, 9, 10, 14)$$

1. Find all the prime Implicants

group 0	$\begin{array}{r} 0 \ 0000 \\ \hline 1 \ 0001 \\ 2 \ 0010 \\ 8 \ 1000 \end{array}$
group 1	$\left\{ \begin{array}{r} 5 \ 0101 \\ 6 \ 0110 \\ 9 \ 1001 \\ 10 \ 1010 \end{array} \right.$
group 2	$\left\{ \begin{array}{r} 7 \ 0111 \\ 14 \ 1110 \end{array} \right.$
group 3	

Group the Minterms  
according to the  
**number of 1's** in the  
Minterm



Column I

Column II

group 0    0 0000

group 1 {  
  1 0001  
  2 0010  
  8 1000

group 2 {  
  5 0101  
  6 0110  
  9 1001  
  10 1010

group 3 {  
  7 0111  
  14 1110

Combining  
group 0 and  
group 1:



## Combining group 0 and group 1:

Column I

Column II

group 0       $\begin{array}{r} 0 \ 0000 \\ \hline 1 \ 0001 \checkmark \end{array}$        $\rightarrow 0,1 \ 000-$

group 1       $\left\{ \begin{array}{r} 2 \ 0010 \\ 8 \ 1000 \end{array} \right.$

group 2       $\left\{ \begin{array}{r} 5 \ 0101 \\ 6 \ 0110 \\ 9 \ 1001 \\ \hline 10 \ 1010 \end{array} \right.$

group 3       $\left\{ \begin{array}{r} 7 \ 0111 \\ 14 \ 1110 \end{array} \right.$

Find matched pair with only one variable change

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

' - ' indicates absence of literal



## Combining group 0 and group 1:

Column I

group 0	$\begin{array}{r} 0 \ 0000 \\ 1 \ 0001 \\ \hline \end{array}$	✓	0,1 000-
group 1	$\begin{array}{r} 2 \ 0010 \\ 8 \ 1000 \\ \hline \end{array}$	✓	0,2 00-0
group 2	$\begin{array}{r} 5 \ 0101 \\ 6 \ 0110 \\ 9 \ 1001 \\ \hline 10 \ 1010 \end{array}$		
group 3	$\begin{array}{r} 7 \ 0111 \\ 14 \ 1110 \end{array}$		

Column II



## Combining group 0 and group 1:

Column I

group 0	$\begin{array}{r} 0 \ 0000 \checkmark \\ 1 \ 0001 \checkmark \end{array}$	0,1 000-
group 1	$\begin{array}{r} 2 \ 0010 \checkmark \\ 8 \ 1000 \checkmark \end{array}$	0,2 00-0 0,8 -000
group 2	$\begin{array}{r} 5 \ 0101 \\ 6 \ 0110 \\ 9 \ 1001 \\ 10 \ 1010 \end{array}$	
group 3	$\begin{array}{r} 7 \ 0111 \\ 14 \ 1110 \end{array}$	

Column II



## Combining group 1 and group 2:

Column I

group 0	0 0000 ✓
group 1	$\frac{1 \ 0001 \checkmark}{2 \ 0010 \checkmark}$
	8 1000 ✓
	$\frac{5 \ 0101 \checkmark}{6 \ 0110}$
group 2	9 1001
	$\frac{10 \ 1010}{7 \ 0111}$
group 3	14 1110

Column II

$$\begin{array}{r} 0,1 \ 000- \\ 0,2 \ 00-0 \\ 0,8 \ -000 \\ \hline 1,5 \ 0-01 \end{array}$$



# Combining group 1 and group 2:

Column I

group 0    0 0000✓

group 1 { 1 0001✓  
          2 0010✓  
          8 1000✓

group 2 { 5 0101✓

          6 0110  
          9 1001  
          10 1010

group 3 { 7 0111  
          14 1110

Column II

0,1 000-  
0,2 00-0  
0,8 -000  
—————  
1,5 0-01



## Combining group 1 and group 2:

Column I

group 0    0 0000 ✓

group 1 { 1 0001 ✓

2 0010 ✓

8 1000 ✓

group 2 { 5 0101 ✓

6 0110

9 1001 ✓

10 1010

group 3 { 7 0111

14 1110

Column II

0,1 000-

0,2 00-0

0,8 -000

1,5 0-01

1,9 -001



# Combining group 1 and group 2:

Column I

group 0	0 0000 ✓
group 1	$\begin{array}{r} \text{1 } 0001 \checkmark \\ \hline \end{array}$
	2 0010 ✓
	8 1000 ✓
group 2	$\begin{array}{r} \text{5 } 0101 \checkmark \\ \hline \end{array}$
	6 0110
	9 1001 ✓
group 3	$\begin{array}{r} \text{10 } 1010 \\ \hline \end{array}$
	7 0111
	14 1110

Column II

0,1	000-
0,2	00-0
0,8	-000
	$\hline$
1,5	0-01
1,9	-001



# Combining group 1 and group 2:

Column I

group 0	0 0000 ✓
group 1	1 0001 ✓
	2 0010 ✓
	8 1000 ✓
group 2	5 0101 ✓
	6 0110
	9 1001 ✓
	10 1010
group 3	7 0111
	14 1110

Column II

0,1	000-
0,2	00-0
0,8	-000
	—————
1,5	0-01
1,9	-001



## Combining group 1 and group 2:

Column I

group 0	0 0000 ✓
group 1	1 0001 ✓
	2 0010 ✓
	8 1000 ✓
	5 0101 ✓
group 2	6 0110 ✓
	9 1001 ✓
	10 1010
group 3	7 0111
	14 1110

Column II

0,1	000-
0,2	00-0
0,8	-000
	—————
1,5	0-01
1,9	-001
2,6	0-10



# Combining group 1 and group 2:

Column I

group 0	0 0000 ✓
group 1	1 0001 ✓
	2 0010 ✓
	8 1000 ✓
	5 0101 ✓
group 2	6 0110 ✓
	9 1001 ✓
	10 1010
group 3	7 0111
	14 1110

Column II

0,1	000-
0,2	00-0
0,8	-000
	—————
1,5	0-01
1,9	-001
2,6	0-10



## Combining group 1 and group 2:

Column I

group 0    0 0000✓

group 1 { 1 0001✓

      2 0010✓

      8 1000✓

      5 0101✓

group 2 { 6 0110✓

      9 1001✓

      10 1010✓

group 3 { 7 0111

      14 1110

Column II

0,1 000-

0,2 00-0

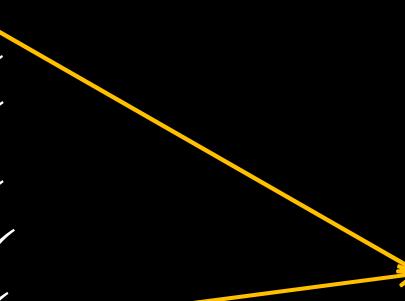
0,8 -000

1,5 0-01

1,9 -001

2,6 0-10

2,10 -010





## Combining group 1 and group 2:

Column I

group 0	0 0000✓
group 1	1 0001✓
	2 0010✓
	8 1000✓
	5 0101✓
group 2	6 0110✓
	9 1001✓
	10 1010✓
group 3	7 0111
	14 1110

Column II

0,1	000-
0,2	00-0
0,8	-000
	—————
1,5	0-01
1,9	-001
2,6	0-10
2,10	-010



## Combining group 1 and group 2:

Column I

group 0    0 0000✓

group 1 { 1 0001✓

2 0010✓

8 1000✓

5 0101✓

6 0110✓

9 1001✓

10 1010✓

group 3 { 7 0111

14 1110

Column II

0,1 000-

0,2 00-0

0,8 -000

1,5 0-01

1,9 -001

2,6 0-10

2,10 -010



## Combining group 1 and group 2:

Column I

group 0    0 0000✓

group 1 { 1 0001✓

2 0010✓

8 1000✓

5 0101✓

6 0110✓

9 1001✓

10 1010✓

group 3 { 7 0111

14 1110

Column II

0,1 000-

0,2 00-0

0,8 -000

1,5 0-01

1,9 -001

2,6 0-10

2,10 -010

8,9 100-



## Combining group 1 and group 2:

Column I

group 0    0 0000✓

group 1 { 1 0001✓

2 0010✓

8 1000✓

5 0101✓

6 0110✓

9 1001✓

10 1010✓

group 3 { 7 0111  
14 1110

Column II

0,1 000-

0,2 00-0

0,8 -000

1,5 0-01

1,9 -001

2,6 0-10

2,10 -010

8,9 100-

8,10 10-0



## Combining group 2 and group 3:

Column I

group 0    0 0000✓

group 1 { 1 0001✓

2 0010✓

8 1000✓

5 0101✓

6 0110✓

9 1001✓

10 1010✓

7 0111✓

14 1110

Column II

0,1 000-

0,2 00-0

0,8 -000

1,5 0-01

1,9 -001

2,6 0-10

2,10 -010

8,9 100-

8,10 10-0

5,7 01-1



## Combining group 2 and group 3:

Column I

group 0    0 0000✓

group 1 { 1 0001✓  
2 0010✓  
8 1000✓

group 2 { 5 0101✓  
6 0110✓  
9 1001✓

group 3 { 10 1010✓  
7 0111✓  
14 1110

Column II

0,1 000-

0,2 00-0

0,8 -000

1,5 0-01

1,9 -001

2,6 0-10

2,10 -010

8,9 100-

8,10 10-0

5,7 01-1



## Combining group 2 and group 3:

Column I

group 0    0 0000✓

group 1 { 1 0001✓

2 0010✓

8 1000✓

5 0101✓

6 0110✓

9 1001✓

10 1010✓

7 0111✓

14 1110

Column II

0,1 000-

0,2 00-0

0,8 -000

1,5 0-01

1,9 -001

2,6 0-10

2,10 -010

8,9 100-

8,10 10-0

5,7 01-1

6,7 011-



## Combining group 2 and group 3:

Column I

group 0    0 0000✓

group 1 { 1 0001✓

2 0010✓

8 1000✓

5 0101✓

6 0110✓

9 1001✓

10 1010✓

7 0111✓

14 1110✓

Column II

0,1 000-

0,2 00-0

0,8 -000

1,5 0-01

1,9 -001

2,6 0-10

2,10 -010

8,9 100-

8,10 10-0

5,7 01-1

6,7 011-

6,14 -110



## Combining group 2 and group 3:

Column I

group 0    0 0000✓

group 1 { 1 0001✓  
2 0010✓  
8 1000✓

group 2 { 5 0101✓  
6 0110✓  
9 1001✓

group 3 { 10 1010✓  
7 0111✓  
14 1110✓

Column II

0,1 000-

0,2 00-0

0,8 -000

1,5 0-01

1,9 -001

2,6 0-10

2,10 -010

8,9 100-

8,10 10-0

5,7 01-1

6,7 011-

6,14 -110



## Combining group 2 and group 3:

Column I

group 0    0 0000✓

group 1 { 1 0001✓  
2 0010✓  
8 1000✓

group 2 { 5 0101✓  
6 0110✓  
9 1001✓

group 3 { 10 1010✓  
7 0111✓  
14 1110✓

Column II

0,1 000-

0,2 00-0

0,8 -000

1,5 0-01

1,9 -001

2,6 0-10

2,10 -010

8,9 100-

8,10 10-0

5,7 01-1

6,7 011-

6,14 -110



## Combining group 2 and group 3:

Column I

group 0    0 0000✓

group 1 { 1 0001✓  
2 0010✓  
8 1000✓

group 2 { 5 0101✓  
6 0110✓  
9 1001✓

group 3 { 10 1010✓  
7 0111✓  
14 1110✓

Column II

0,1 000-

0,2 00-0

0,8 -000

1,5 0-01

1,9 -001

2,6 0-10

2,10 -010

8,9 100-

8,10 10-0

5,7 01-1

6,7 011-

6,14 -110



## Combining group 2 and group 3:

	Column I	Column II
group 0	0 0000✓	0,1 000-
group 1	1 0001✓	0,2 00-0
	2 0010✓	0,8 -000
	8 1000✓	
	5 0101✓	1,5 0-01
group 2	6 0110✓	1,9 -001
	9 1001✓	2,6 0-10
	10 1010✓	2,10 -010
group 3	7 0111✓	8,9 100-
	14 1110✓	8,10 10-0
		5,7 01-1
		6,7 011-
		6,14 -110
		10,14 1-10



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-	
group 1	1 0001✓	0,2 00-0	
	2 0010✓	0,8 -000	
	8 1000✓	1,5 0-01	
	5 0101✓	1,9 -001	
group 2	6 0110✓	2,6 0-10	
	9 1001✓	2,10 -010	
	10 1010✓	8,9 100-	
	7 0111✓	8,10 10-0	
group 3	14 1110✓	5,7 01-1	
		6,7 011-	
		6,14 -110	
		10,14 1-10	



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-	
group 1	1 0001✓	0,2 00-0	
	2 0010✓	0,8 -000	
	8 1000✓	1,5 0-01	
	5 0101✓	1,9 -001	
group 2	6 0110✓	2,6 0-10	
	9 1001✓	2,10 -010	
	10 1010✓	8,9 100-	
group 3	7 0111✓	8,10 10-0	
	14 1110✓	5,7 01-1	
		6,7 011-	
		6,14 -110	
		10,14 1-10	



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-	
group 1	1 0001✓	0,2 00-0	
	2 0010✓	0,8 -000	
	8 1000✓	1,5 0-01	
	5 0101✓	1,9 -001	
group 2	6 0110✓	2,6 0-10	
	9 1001✓	2,10 -010	
	10 1010✓	8,9 100-	
	7 0111✓	8,10 10-0	
group 3	14 1110✓	5,7 01-1	
		6,7 011-	
		6,14 -110	
		10,14 1-10	



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-	
group 1	1 0001✓	0,2 00-0	
	2 0010✓	0,8 -000	
	8 1000✓	1,5 0-01	
	5 0101✓	1,9 -001	
group 2	6 0110✓	2,6 0-10	
	9 1001✓	2,10 -010	
	10 1010✓	8,9 100-	
	7 0111✓	8,10 10-0	
group 3	14 1110✓	5,7 01-1	
		6,7 011-	
		6,14 -110	
		10,14 1-10	



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-	
group 1	1 0001✓	0,2 00-0	
	2 0010✓	0,8 -000	
	8 1000✓	1,5 0-01	
	5 0101✓	1,9 -001	
group 2	6 0110✓	2,6 0-10	
	9 1001✓	2,10 -010	
	10 1010✓	8,9 100-	
group 3	7 0111✓	8,10 10-0	
	14 1110✓	5,7 01-1	
		6,7 011-	
		6,14 -110	
		10,14 1-10	



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-	0,1,8,9 -00-
group 1	1 0001✓	0,2 00-0	
	2 0010✓	0,8 -000	
	8 1000✓		
		1,5 0-01	
group 2	5 0101✓	1,9 -001	
	6 0110✓	2,6 0-10	
	9 1001✓	2,10 -010	
	10 1010✓	8,9 100-	✓
group 3	7 0111✓	8,10 10-0	
	14 1110✓	5,7 01-1	
		6,7 011-	
		6,14 -110	
		10,14 1-10	



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000- ✓	0,1,8,9 -00-
group 1	1 0001✓	0,2 00-0	
	2 0010✓	0,8 -000	
	8 1000✓		
		1,5 0-01	
group 2	5 0101✓	1,9 -001	
	6 0110✓	2,6 0-10	
	9 1001✓	2,10 -010	
	10 1010✓	8,9 100- ✓	
group 3	7 0111✓	8,10 10-0	
	14 1110✓	5,7 01-1	
		6,7 011-	
		6,14 -110	
		10,14 1-10	



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-✓	0,1,8,9 -00-
group 1	1 0001✓	0,2 00-0	
	2 0010✓	0,8 -000	
	8 1000✓	1,5 0-01	
	5 0101✓	1,9 -001	
group 2	6 0110✓	2,6 0-10	
	9 1001✓	2,10 -010	
	10 1010✓	8,9 100-✓	
	7 0111✓	8,10 10-0	
group 3	14 1110✓	5,7 01-1	
		6,7 011-	
		6,14 -110	
		10,14 1-10	



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-✓	0,1,8,9 -00-
group 1	1 0001✓	0,2 00-0	
	2 0010✓	0,8 -000	
	8 1000✓		1,5 0-01
			1,9 -001
group 2	5 0101✓	2,6 0-10	
	6 0110✓	2,10 -010	
	9 1001✓		8,9 100-✓
	10 1010✓		8,10 10-0
group 3	7 0111✓	5,7 01-1	
	14 1110✓	6,7 011-	
		6,14 -110	
		10,14 1-10	



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-✓	0,1,8,9 -00-
group 1	1 0001✓	0,2 00-0	
	2 0010✓	0,8 -000	
	8 1000✓		1,5 0-01
			1,9 -001
group 2	5 0101✓		2,6 0-10
	6 0110✓		2,10 -010
	9 1001✓		8,9 100-✓
	10 1010✓		8,10 10-0
group 3	7 0111✓	5,7 01-1	
	14 1110✓	6,7 011-	
		6,14 -110	
		10,14 1-10	



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-✓	0,1,8,9 -00-
group 1	1 0001✓	0,2 00-0	
	2 0010✓	0,8 -000	
	8 1000✓		1,5 0-01
			1,9 -001
group 2	5 0101✓		2,6 0-10
	6 0110✓		2,10 -010
	9 1001✓		8,9 100-✓
	10 1010✓		8,10 10-0
group 3	7 0111✓	5,7 01-1	
	14 1110✓	6,7 011-	
		6,14 -110	
		10,14 1-10	



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-✓	0,1,8,9 -00-
group 1	1 0001✓	0,2 00-0	
	2 0010✓	0,8 -000	
	8 1000✓		1,5 0-01
			1,9 -001
group 2	5 0101✓		2,6 0-10
	6 0110✓		2,10 -010
	9 1001✓		8,9 100-✓
	10 1010✓		
group 3	7 0111✓	8,10 10-0	
	14 1110✓	5,7 01-1	
		6,7 011-	
		6,14 -110	
		10,14 1-10	



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-✓	0,1,8,9 -00-
group 1	1 0001✓	0,2 00-0✓	0,2,8,10 -0-0
	2 0010✓	0,8 -000	
	8 1000✓	1,5 0-01	
	5 0101✓	1,9 -001	
group 2	6 0110✓	2,6 0-10	
	9 1001✓	2,10 -010	
	10 1010✓	8,9 100-✓	
	7 0111✓	8,10 10-0✓	
group 3	14 1110✓	5,7 01-1	
		6,7 011-	
		6,14 -110	
		10,14 1-10	



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-✓	0,1,8,9 -00-
group 1	1 0001✓	0,2 00-0✓	0,2,8,10 -0-0
	2 0010✓	0,8 -000	
	8 1000✓	1,5 0-01	
	5 0101✓	1,9 -001	
group 2	6 0110✓	2,6 0-10	
	9 1001✓	2,10 -010	
	10 1010✓	8,9 100-✓	
	7 0111✓	8,10 10-0✓	
group 3	14 1110✓	5,7 01-1	
		6,7 011-	
		6,14 -110	
		10,14 1-10	



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-✓	0,1,8,9 -00-
group 1	{ 1 0001✓ 2 0010✓ 8 1000✓	0,2 00-0✓ 0,8 -000✓ 1,5 0-01	0,2,8,10 -0-0 0,8,1,9 -00-
group 2	{ 5 0101✓ 6 0110✓ 9 1001✓ 10 1010✓	1,9 -001✓ 2,6 0-10 2,10 -010 8,9 100-✓	Same as
group 3	{ 7 0111✓ 14 1110✓	8,10 10-0✓ 5,7 01-1 6,7 011- 6,14 -110 10,14 1-10	Need not repeat if already Covered



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-✓	0,1,8,9 -00-
group 1	1 0001✓ 2 0010✓ 8 1000✓	0,2 00-0✓ 0,8 -000✓ 1,5 0-01	0,2,8,10 -0-0
group 2	5 0101✓ 6 0110✓ 9 1001✓ 10 1010✓	1,9 -001✓ 2,6 0-10 2,10 -010 8,9 100-✓	
group 3	7 0111✓ 14 1110✓	8,10 10-0✓ 5,7 01-1 6,7 011- 6,14 -110 10,14 1-10	



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-✓	0,1,8,9 -00-
group 1	1 0001✓ 2 0010✓ 8 1000✓	0,2 00-0✓ 0,8 -000✓ 1,5 0-01	0,2,8,10 -0-0 Already Covered
group 2	5 0101✓ 6 0110✓ 9 1001✓ 10 1010✓	1,9 -001✓ 2,6 0-10 2,10 -010✓ 8,9 100-✓	
group 3	7 0111✓ 14 1110✓	8,10 10-0✓ 5,7 01-1 6,7 011- 6,14 -110 10,14 1-10	



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-✓	0,1,8,9 -00-
group 1	1 0001✓ 2 0010✓ 8 1000✓	0,2 00-0✓ 0,8 -000✓ 1,5 0-01	0,2,8,10 -0-0
group 2	5 0101✓ 6 0110✓ 9 1001✓ 10 1010✓	1,9 -001✓ 2,6 0-10 2,10 -010✓ 8,9 100-✓	
group 3	7 0111✓ 14 1110✓	8,10 10-0✓ 5,7 01-1 6,7 011- 6,14 -110 10,14 1-10	



## Combining group (0,1) and group (1,2):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-✓	0,1,8,9 -00-
group 1	1 0001✓ 2 0010✓ 8 1000✓	0,2 00-0✓ <b>0,8 -000✓</b> 1,5 0-01	0,2,8,10 -0-0
group 2	5 0101✓ 6 0110✓ 9 1001✓ 10 1010✓	1,9 -001✓ 2,6 0-10 2,10 -010✓ 8,9 100-✓	
group 3	7 0111✓ 14 1110✓	<b>8,10 10-0✓</b> 5,7 01-1 6,7 011- 6,14 -110 10,14 1-10	



## Combining group (1,2) and group (2,3):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-✓	0,1,8,9 -00-
group 1	1 0001✓	0,2 00-0✓	0,2,8,10 -0-0
	2 0010✓	0,8 -000✓	
	8 1000✓	1,5 0-01	
		1,9 -001✓	
group 2	5 0101✓	2,6 0-10✓	
	6 0110✓	2,10 -010✓	
	9 1001✓	8,9 100-✓	
	10 1010✓	8,10 10-0✓	
group 3	7 0111✓	5,7 01-1	
	14 1110✓	6,7 011-	
		6,14 -110	
		10,14 1-10✓	



## Combining group (1,2) and group (2,3):

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-✓	0,1,8,9 -00-
group 1	1 0001✓ 2 0010✓ 8 1000✓	0,2 00-0✓ 0,8 -000✓ 1,5 0-01	0,2,8,10 -0-0 2,6,10,14 --10
group 2	5 0101✓ 6 0110✓ 9 1001✓ 10 1010✓	1,9 -001✓ 2,6 0-10✓ 2,10 -010✓ 8,9 100-✓	
group 3	7 0111✓ 14 1110✓	8,10 10-0✓ 5,7 01-1 6,7 011- 6,14 -110✓ 10,14 1-10✓	Already Covered

Any more groupings possible ??



# Prime Implicants

	Column I	Column II	Column III
group 0	0 0000✓	0,1 000-✓	0,1,8,9 -00- ← D = $x'y'$
group 1	1 0001✓	0,2 00-0✓	0,2,8,10 -0-0 ← E = $x'z'$
	2 0010✓	0,8 -000✓	2,6,10,14 --10 ← F = $yz'$
	8 1000✓	1,5 0-01	A = $w'y'z$ ←
	5 0101✓	1,9 -001✓	
group 2	6 0110✓	2,6 0-10✓	
	9 1001✓	2,10 -010✓	
	10 1010✓	8,9 100-✓	
	7 0111✓	8,10 10-0✓	
group 3	14 1110✓	5,7 01-1	B = $w'xz$ ←
		6,7 011-	C = $w'xy$ ←
		6,14 -110✓	
		10,14 1-10✓	

A, B, C, D, E, F are Prime Implicants



# Essential Prime Implicants

## 2. Find Essential Prime Implicants

The Minterms 9 and 14 are covered by single terms

Prime Implicants	Minterms									
	0	1	2	5	6	7	8	9	10	14
(1,5) A=w'y'z		X		X						
(5,7) B=w'xz					X	X				
(6,7) C=w'xy						X	X			
(0,1,8,9) D=x'y'	X	X					X	X		
(0,2,8,10) E=x'z'	X		X				X		X	
(2,6,10,14) F=yz'			X		X			X	X	X

$f(w, x, y, z) = \sum(0, 1, 2, 5, 6, 7, 8, 9, 10, 14)$

$y'z'$  and  $x'y'$

Essential Prime Implicants



# Essential Prime Implicants

Partial Solution: D+F + -----

		0	1	2	5	6	7	8	9	10	14
(1,5)	A=w'y'z		X		X						
(5,7)	B=w'xz					X		X			
(6,7)	C=w'xy							X	X		
(0,1,8,9)	D=x'y'	X	X					X	X		
(0,2,8,10)	E=x'z'	X		X				X		X	
(2,6,10,14)	F=yz'			X		X			X		X

Once the Essential Prime Implicants are part of solution, The columns covered by them can be removed

Columns 0, 1, 8, 9, 2, 6, 10, 14 can be removed



Partial Solution: D+F + -----

		5	7
(1,5)	A=w'y'z	X	
(5,7)	B=w'xz	X	X
(6,7)	C=w'xy		X
(0,2,8,10)	E=x'z'		

Redundant

	5	7	
(1,5)	A=w'y'z	X	
(5,7)	B=w'xz	X	X
(6,7)	C=w'xy		X

Row corresponding to B is  
dominating the other Rows

Including B in the final solution  
both columns get covered

Final Solution: D + F + B = x'y' + yz' + w'xz



## Row Dominance

In Prime Implicant table **dominated rows** can be deleted

Row B is dominating the **rows A and C**

		5	7
(1,5)	$A=w'y'z$	X	
(5,7)	$B=w'xz$	X	X
(6,7)	$C=w'xy$		X

By making **B** part of minimal function both columns 5 and 7 get covered

Dominated rows **A and C** can be deleted (if A and C have higher cost than B)



## Column Dominance

In Prime Implicant table **dominating columns can be deleted**

Column corresponding to 9 dominates column corresponding to 8

		8	9
(8,9,--,--)	A	X	X
(--,9,--,--)	B		X
(--,8,--,9)	C	X	X

Any set of rows that covers dominated column must also cover dominating column

If there is 8 there will always be 9 with it, covering 8 is sufficient

Dominating column 9 can be deleted



## Special Cases

$$F(a,b,c) = \sum m(0, 1, 2, 5, 6, 7)$$

0	000	✓	0,1	00-	
1	001	✓	0,2	0-0	
2	010	✓	1,5	-01	
5	101	✓	2,6	-10	
6	110	✓	5,7	1-1	
7	111	✓	6,7	11-	

		0	1	2	5	6	7
(0,1)	a'b'	X	X				
(0,2)	a'c'	X		X			
(1,5)	b'c		X		X		
(2,6)	bc'			X		X	
(5,7)	ac				X		X
(6,7)	ab					X	X

Essential Prime Implicants ??

Row or Column  
Dominance??



## Special Cases

$$F(a,b,c) = \sum m(0, 1, 2, 5, 6, 7)$$

0	000✓	0,1	00-
1	001✓	0,2	0-0
2	010✓	1,5	-01
5	101✓	2,6	-10
6	110✓	5,7	1-1
7	111✓	6,7	11-

	0	1	2	5	6	7
(0,1)	a'b'	X	X			
(0,2)	a'c'	X		X		
(1,5)	b'c		X		X	
(2,6)	bc'		X			X
(5,7)	ac			X		X
(6,7)	ab			X	X	X

Low cost terms can be included, but all have same cost ??

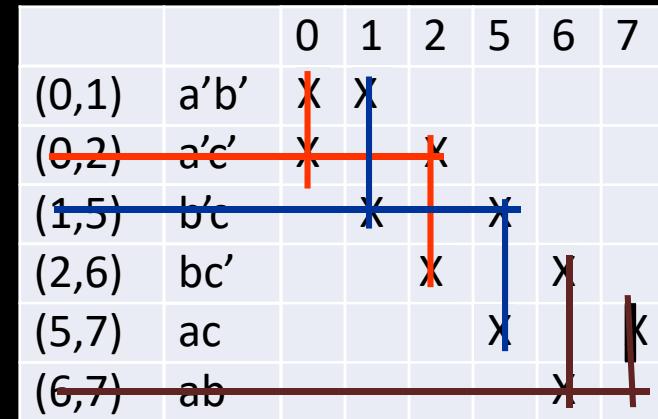
**Solution 1 : a'b' + bc' + ac**



## Special Cases

$$F(a,b,c) = \sum m(0, 1, 2, 5, 6, 7)$$

0	000✓	0,1	00-
1	001✓	0,2	0-0
2	010✓	1,5	-01
5	101✓	2,6	-10
6	110✓	5,7	1-1
7	111✓	6,7	11-



**Solution 2 : a'c' + b'c + ab**



Trial and Error, is it preferred ???

Petrick's Method (If anyone is interested, can look into it later)



*Thank you*

# Digital Design

CS / EEE / ECE/ INSTR F215

## Lecture 9b:

1. Static Hazards in Digital Circuits
2. Optimization of Multi-Output Digital Circuits



**Birla Institute of Technology & Science, Pilani**  
Hyderabad Campus





# Hazard

- A hazard is a momentary unwanted switching transient at a logic function's output (Glitch).
- Hazards/glitches occur due to unequal propagation delays along different paths in a combinational circuit.
- We can take steps to eliminate hazards in digital circuits.
- There are two types of hazards; static and dynamic.



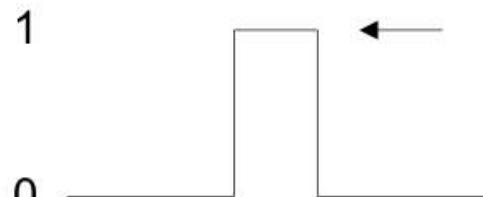
# Static Hazard

## □ Static-0 Hazard:

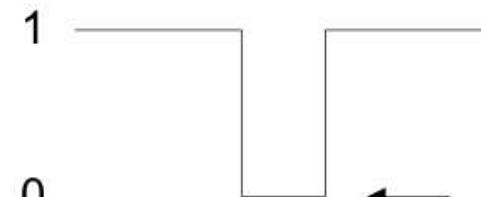
- Occurs when output is 0 and should remain at 0, but temporarily switches to a 1 due to a change in an input.

## □ Static-1 Hazard:

- Occurs when output is 1 and should remain at 1, but temporarily switches to a 0 due to a change in an input.



static-0 hazard (0->0)



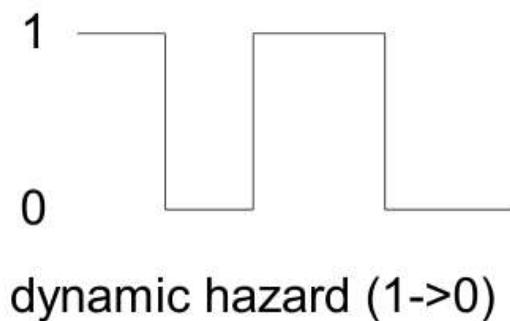
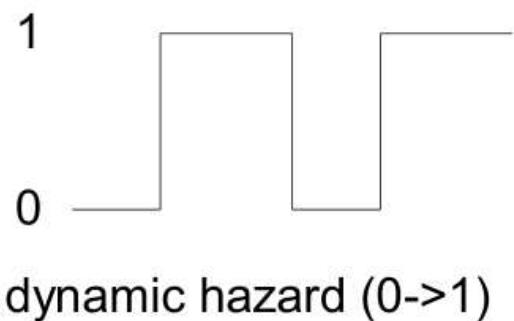
static-1 hazard (1->1)



# Dynamic Hazard

## Dynamic Hazard:

- Occurs when an input changes, and a circuit output should change **0 -> 1 or 1 -> 0**, but temporarily flips between values.





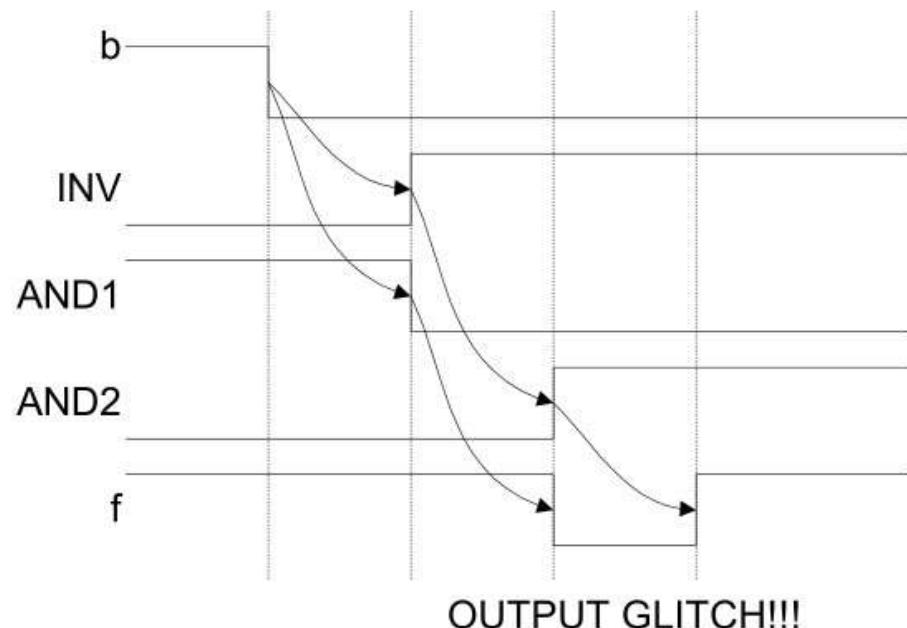
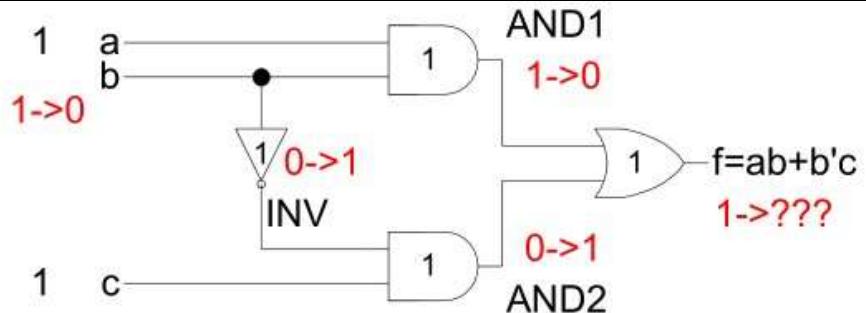
# Static Glitch Example

Consider the following circuit with delays where only one input (input b) changes...

Draw a timing diagram to see what happens at output with delays.

From the logic expression, we see that b changing should result in the output remaining at logic level 1...

Due to delay, the output goes 1->0->1 and this is an output glitch; **we see a static-1 hazard.**





# Static Glitch Elimination

When circuits are implemented as **2-level SOP (2-level POS)**, we can detect and remove hazards by inspecting the K-Map and *adding redundant product (sum) terms*.

		bc	00	01	11	10
		a	0	1	0	0
a	0	0	1			
	1	0	1	1	1	

$$f = ab + b'c$$

Observe that when input b changes from 1->0 (as in the previous timing diagram), that we “jump” from one product term to another product term.

- **If adjacent minterms are not covered by the same product term, then a HAZARD EXISTS!!!**



# Static Glitch Elimination

		bc	00	01	11	10
		a	0	1	0	0
a	0	0	1	0	0	
	1	0	1	1	1	

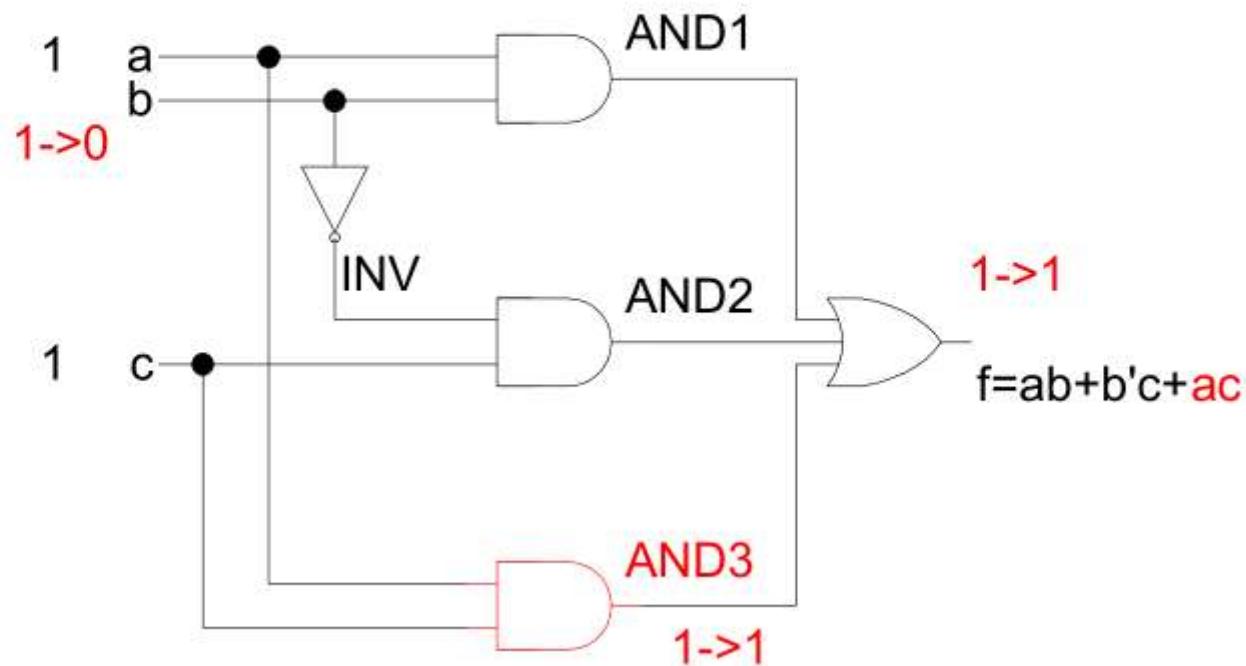
$$f = ab + b'c + ac$$

The extra product term does not include the changing input variable, and therefore serves to prevent possible momentary output glitches due to this variable.



# Static Glitch Elimination

The redundant product term is not influenced by the changing input.





# Static Glitch Elimination

For 2-level circuits, if we remove all static-1 hazards using the K-Map (adding redundant product terms), we are guaranteed that there will be no static-0 hazards or dynamic hazards.

If we work with Product-Of-Sums, we might find static-0 hazards when moving from one sum term to another sum term. We can remove these hazards by adding redundant sum-terms.



# Multi-Output Circuit Optimization

$$F1 = \Sigma(0,2,6,7)$$

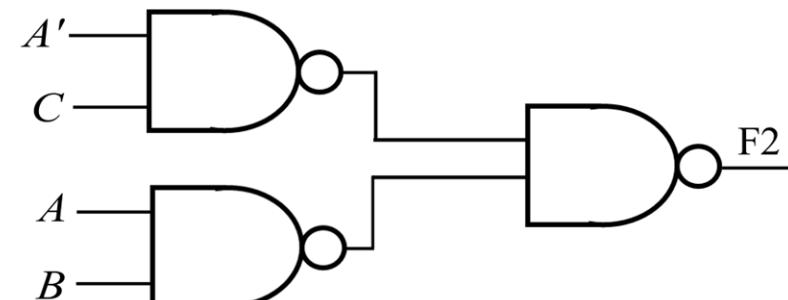
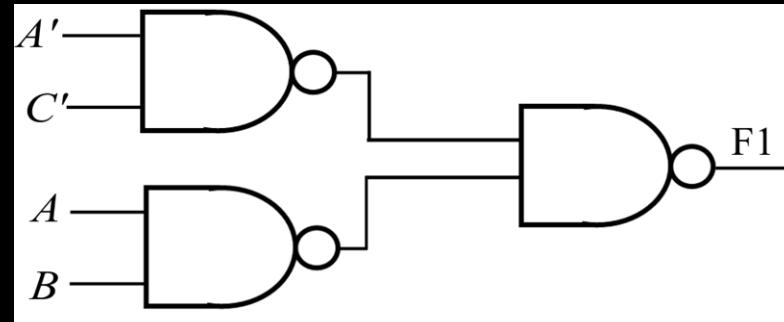
		BC			
		00	01	11	10
A	0	1			1
	1			1	1

$$F1 = A'C' + AB$$

$$F2 = \Sigma(1,3,6,7)$$

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

$$F2 = A'C + AB$$





# Multi-Output Circuit Optimization

$$F1 = \Sigma(0,2,6,7)$$

BC

		00	01	11	10
		0	1		1
A	0				
	1		1	1	

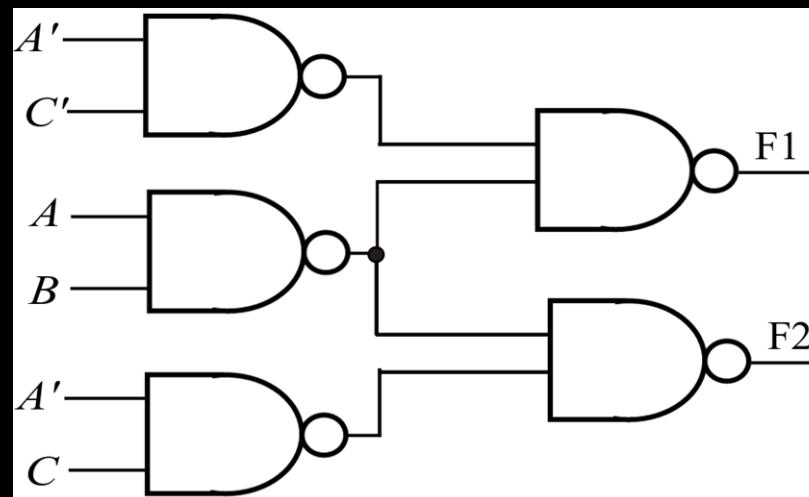
$$F1 = A'C' + AB$$

$$F2 = \Sigma(1,3,6,7)$$

BC

		00	01	11	10
		0	1	1	
A	0				
	1		1	1	1

$$F2 = A'C + AB$$





*Thank you*

# Digital Design

CS / EEE / ECE/ INSTR F215

## Lecture 10: Combinational Circuits



**Birla Institute of Technology & Science, Pilani**  
Hyderabad Campus

9/15/2021





- A combinational circuit consists of logic gates whose outputs at any time are determined by combining the values of the inputs.
- For  $n$  input variables, there are  $2^n$  possible binary input combinations.
- For each binary combination of the input variables, there is one possible output.



## Parity Generation and Checking:

- Parity is used to detect errors in transmitted data caused by noise or other disturbances.
- A parity bit is an extra bit that is added to a data word and can be either odd or even parity.
- In an even parity system, the sum of all the bits (including the parity bit) is an even number
- In an odd parity system the sum of all the bits must be an odd number.
- The circuit that creates the parity bit at the transmitter is called the parity generator.
- The circuit that determines if the received data is correct is the parity checker.
- Parity is good for detecting a single bit error only.
- The parity generator and the parity checker can both be built using Exclusive-OR gates.



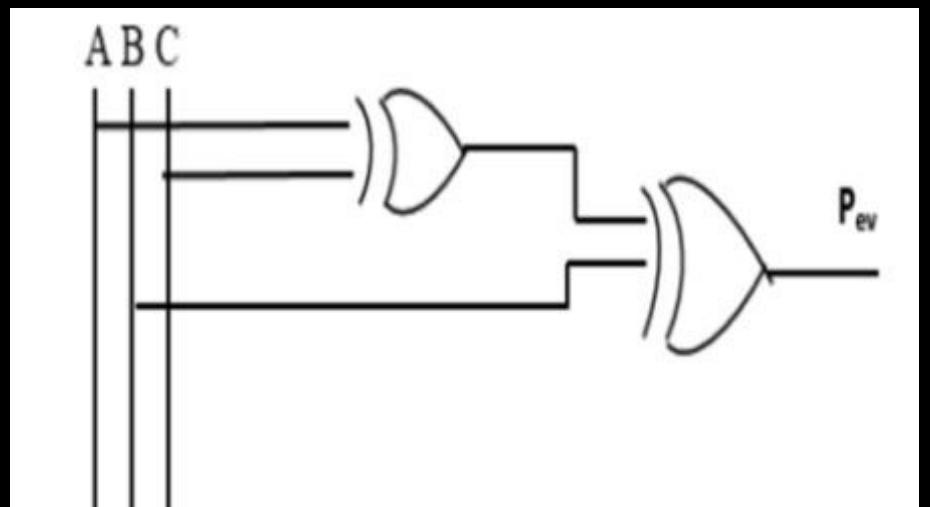
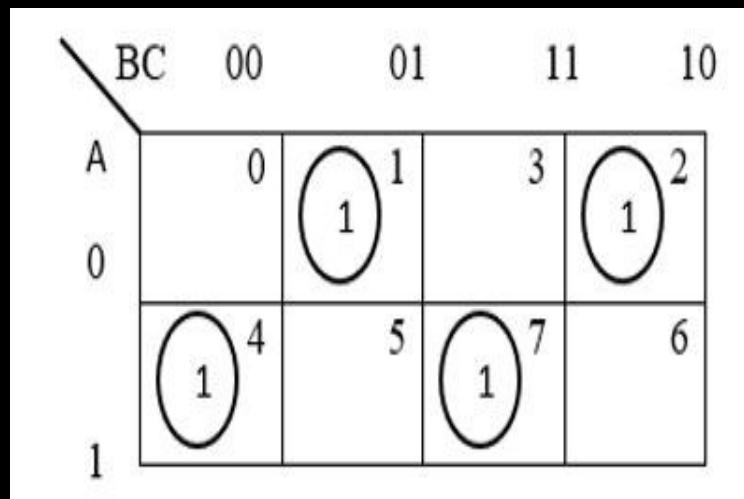
# Even Parity Generator

**Even Parity Generator Truth Table**

A B C	Even Parity
0 0 0	0
0 0 1	1
0 1 0	1
0 1 1	0
1 0 0	1
1 0 1	0
1 1 0	0
1 1 1	1



# K-map for Even Parity generator and its reduced expression for logic circuit design



$$\begin{aligned}P_{ev} &= (A \bar{B} \bar{C} + \bar{A} \bar{B} C) + (A B C + \bar{A} B \bar{C}) \\&= \bar{B} (A \bar{C} + \bar{A} C) + B (A C + \bar{A} \bar{C}) \\&= \bar{B} (A \oplus C) + B (\overline{A \oplus C}) = \bar{B}x + B\bar{x} = B \oplus x = B \oplus A \oplus C\end{aligned}$$



# Odd Parity Generator

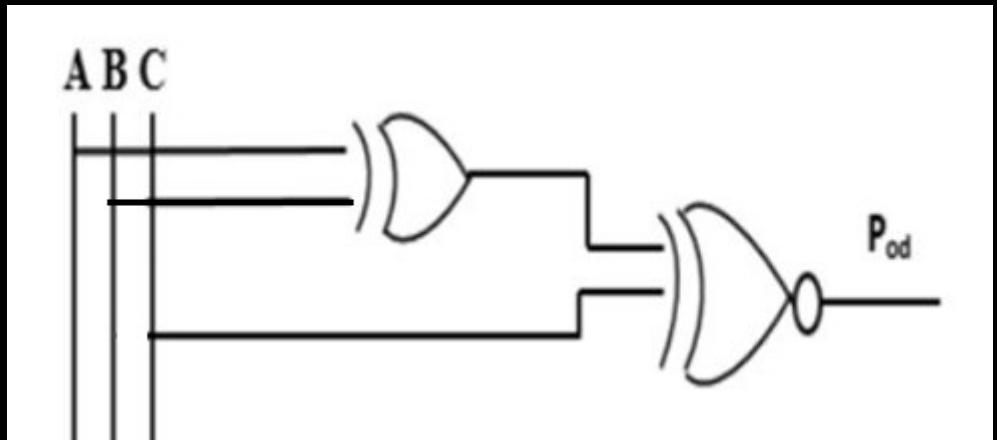
**Odd Parity Generator Truth Table**

A B C	Odd Parity
0 0 0	1
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	0



# K-map for Odd Parity generator and its reduced expression for logic circuit design

		BC	00	01	11	10
		A	0	1	3	2
A	0	1				
	1		4	5	7	6



$$\begin{aligned}P_{od} &= (\bar{A} \bar{B} \bar{C} + A \bar{B} C) + (\bar{A} B C + A B \bar{C}) \\&= \bar{C} (A B + \bar{A} \bar{B}) + B (A \bar{B} + \bar{A} B) \\&= \bar{C} (\overline{A \oplus B}) + C (A \oplus B) \\&= \bar{C} \bar{X} + C X = C \odot X = C \odot A \oplus B\end{aligned}$$



## Even Parity Checker

In even parity checker if the error bit (E) is equal to '1', then we have an error. If error bit E=0 then indicates there is no error.

Error Bit (E) =1, error occurs

Error Bit (E) =0, no error

## Odd Parity Checker

In odd parity checker if an error bit (E) is equal to '1', then it indicates there is no error. If an error bit E=0 then indicates there is an error.

Error Bit (E) =1, no error

Error Bit (E) =0, error occurs



# Example: A 4 bit Even Parity Checker Truth Table

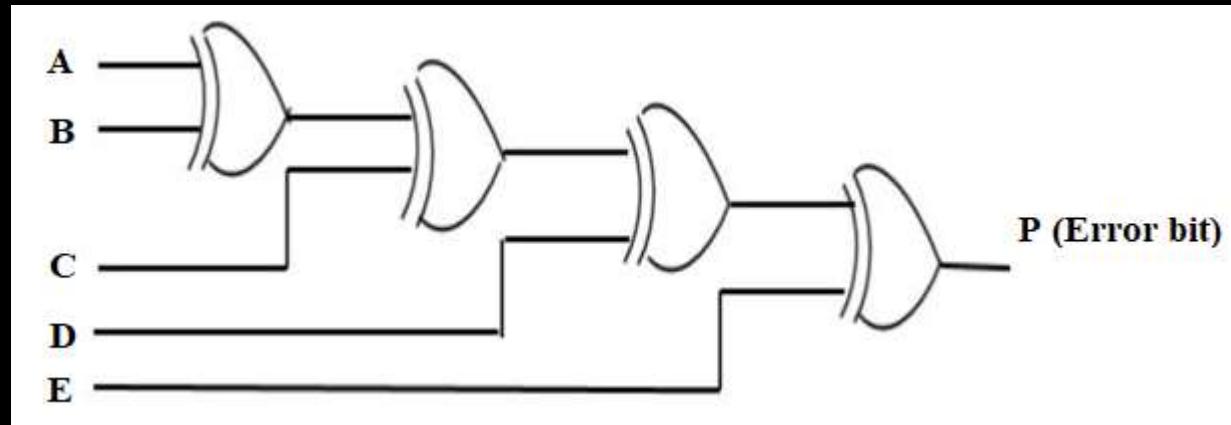
4-bit received message				Parity error check P
A	B	C	D	P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0



$$= (A \oplus B \oplus C \oplus D)$$



# Parity checker Logic Circuit for 5-bit received message

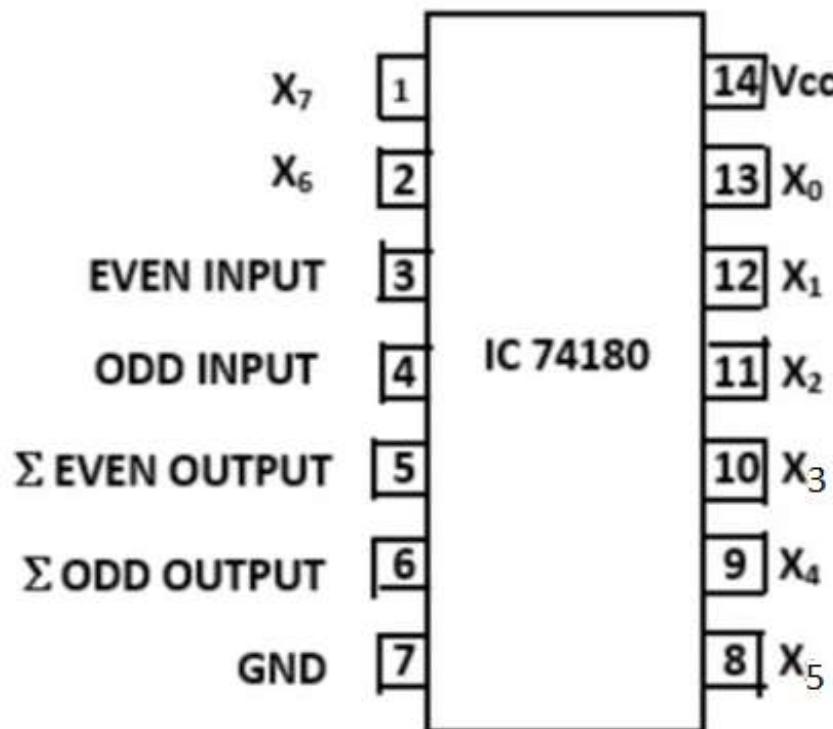


An additional 4<sup>th</sup> EX-OR Gate is introduced



## Parity Generator/Checker using IC's

The IC 74180 does the function of parity generation as well as checking. The 9 bit (8 data bits, 1 parity bit) Parity Generator/Checker is shown in the below figure.





*Thank you*



---

# Digital Design

## Lecture 11: Combinational Logic and Arithmetic Circuits



**Birla Institute of Technology & Science, Pilani**  
Hyderabad Campus



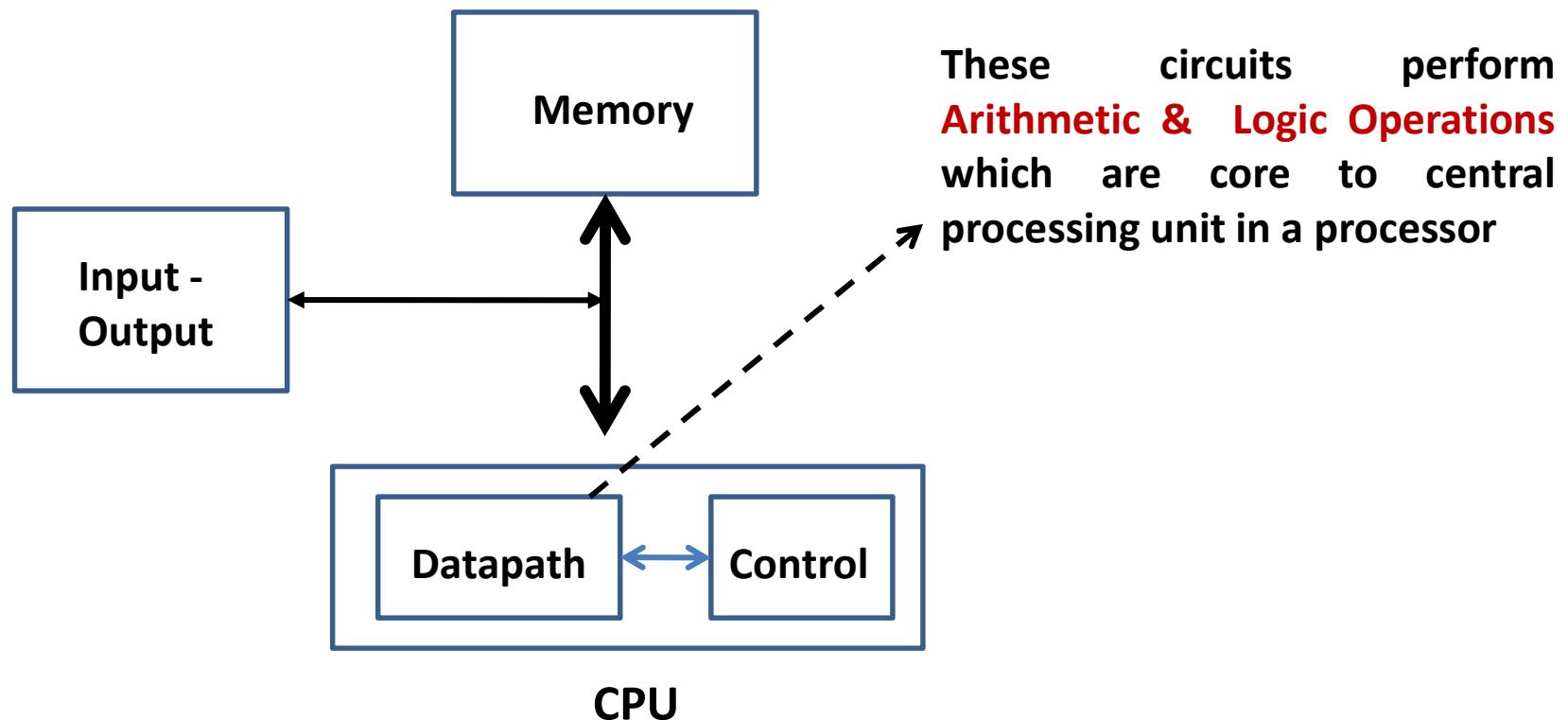


---

# ARITHMETIC LOGIC CIRCUITS DESIGN



## Blocks of a Digital Processor



## Adders : Building blocks

Adding two single-bit binary values, X, Y and produces a sum S and a carry out C\_out

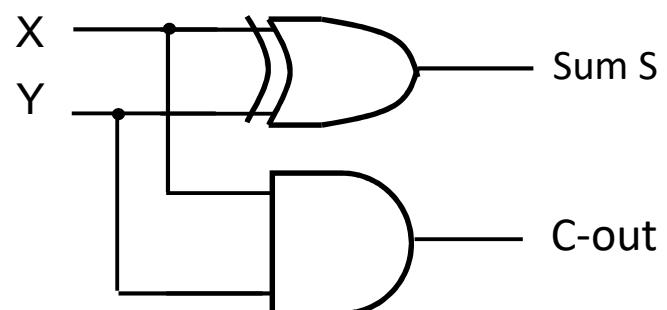
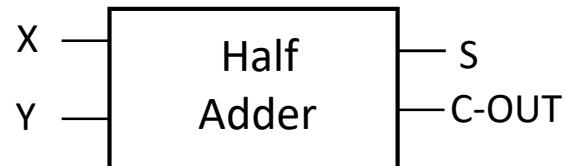
This operation is called half addition – **Half Adder**

**Half Adder Truth Table**

Inputs		Outputs	
X	Y	S	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$\begin{aligned}S(X,Y) &= \Sigma(1,2) \\S &= X'Y + XY' \\S &= X \oplus Y\end{aligned}$$

$$\begin{aligned}C_{\text{out}}(x,y) \\C_{\text{out}} = XY\end{aligned}$$

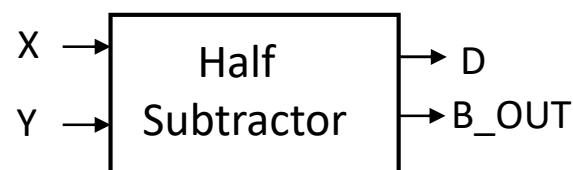


## Building blocks - contd

Subtracting a single-bit binary value Y from another X (i.e.  $X - Y$ ) and producing a difference bit D and a borrow out bit B-out. This operation is called half addition – **Half Subtractor**

**Half Subtractor Truth Table**

Inputs		Outputs	
X	Y	D	B_out
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



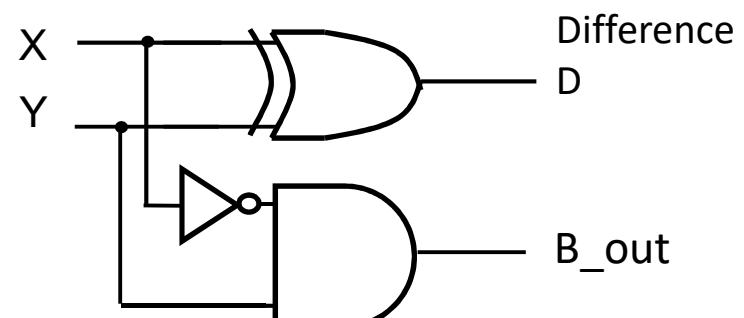
$$D(X,Y) = \Sigma (1,2)$$

$$D = X'Y + XY'$$

$$D = X \oplus Y$$

$$B_{out}(x, y)$$

$$B_{out} = X'Y$$

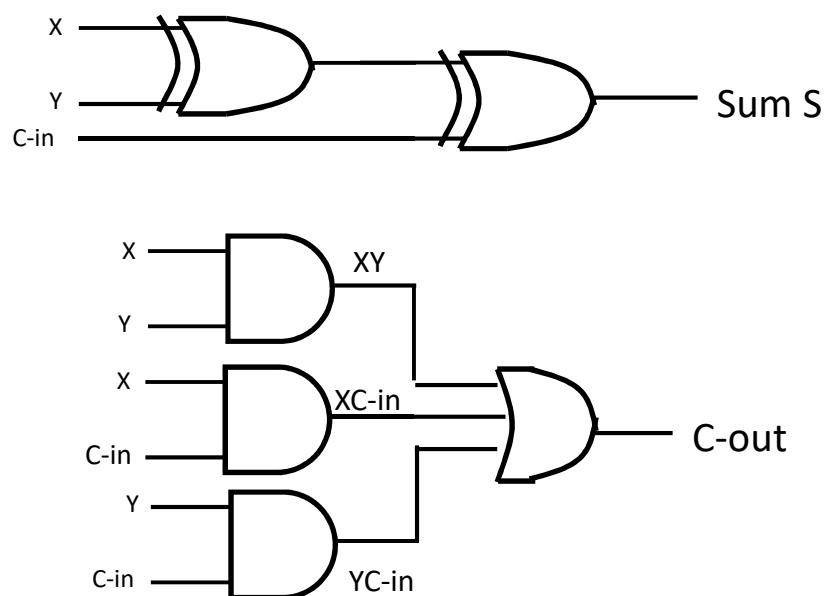


## FULL ADDER

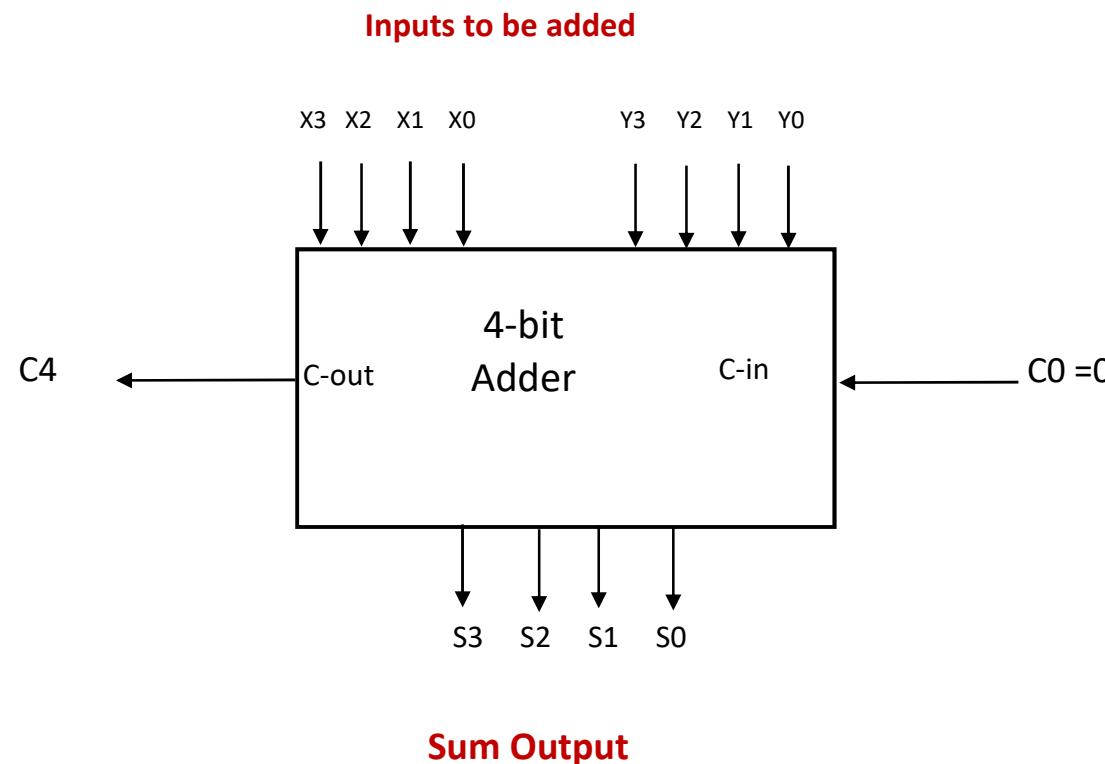
Adding two single-bit binary values, X, Y and a carry input bit C\_in, produces a sum bit S and a carry out C\_out bit.

$$(X + Y + C_{in})$$

X	Y	C-in	S	C-out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

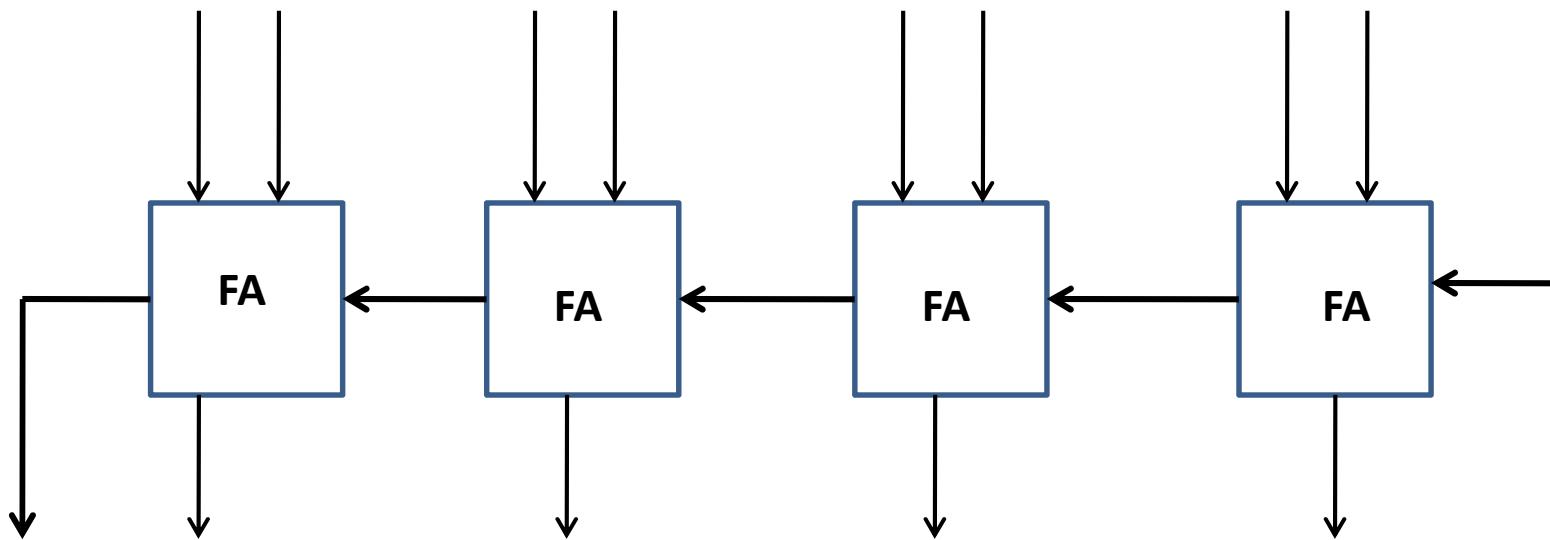


## 4 bit Binary Adder



A 4-bit Adder is used to add two 4-bit binary numbers can be built by connecting in series/cascaded 4 full adders.

## 4 bit Binary Adder



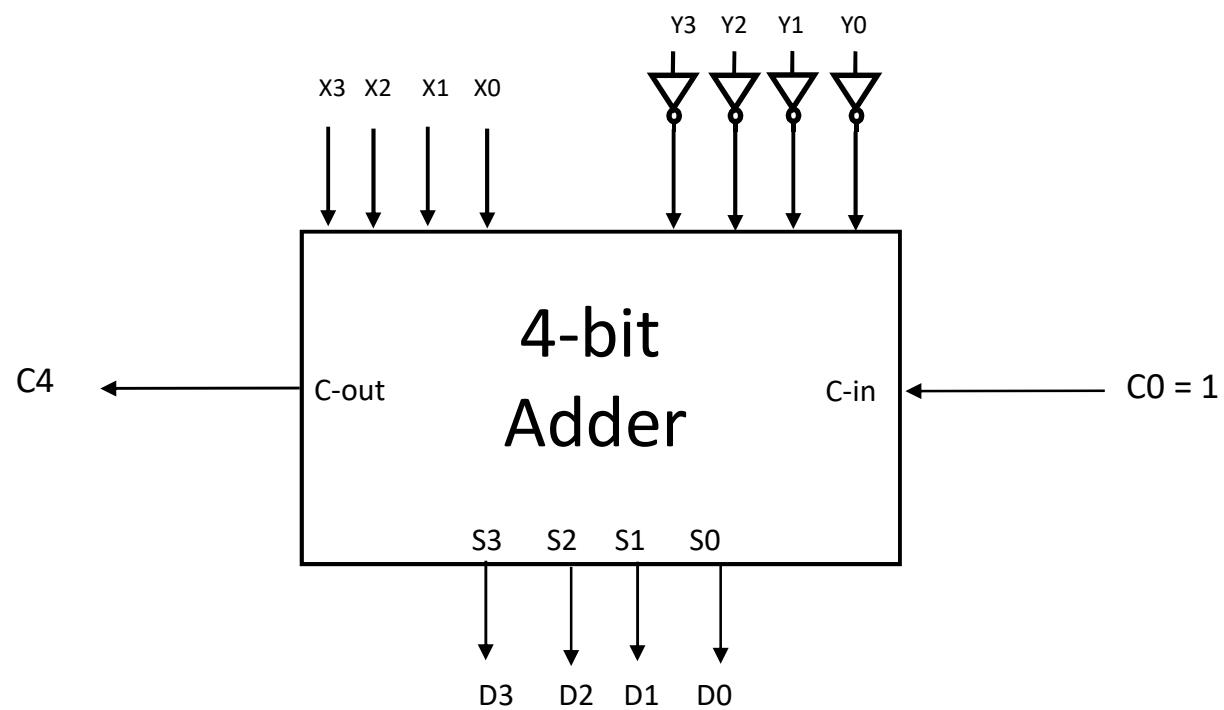
- \*Each full adder represents a bit position ' $j$ ' (from 0 to  $n-1$ ).
- \*Each carry out ( $C_{out}$ ) from a full adder at position  $j$  is connected to the carry in  $C_{in}$  of the full adder at the higher position  $j+1$ .

## Implementation of a Subtractor using Adder

The subtraction  $X-Y$  can be performed by taking the 2's complement of  $Y$  and adding to  $X$ .

$$\begin{aligned}
 X-Y &= X + 2C(Y) \\
 &= X + 1C(Y) + 1 \\
 &= X + Y' + 1
 \end{aligned}$$

Inputs to be subtracted

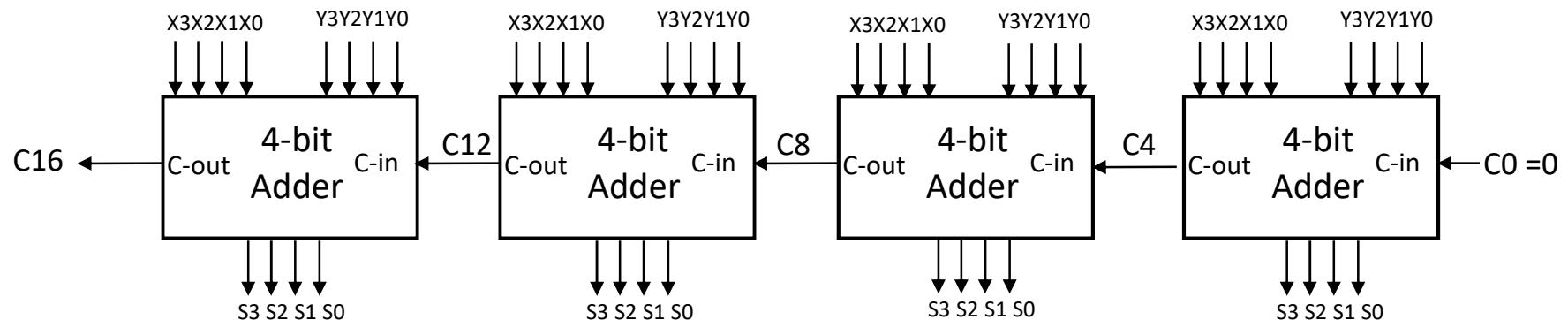


## Larger Adder

### Example: 16-bit adder using 4, 4-bit adders

Adds two 16-bit inputs X (bits X0 to X15), Y (bits Y0 to Y15) producing a 16-bit Sum S (bits S0 to S15) and a carry out C1 from most significant position.

**Data inputs to be added X (X0 to X15) , Y (Y0-Y15)**



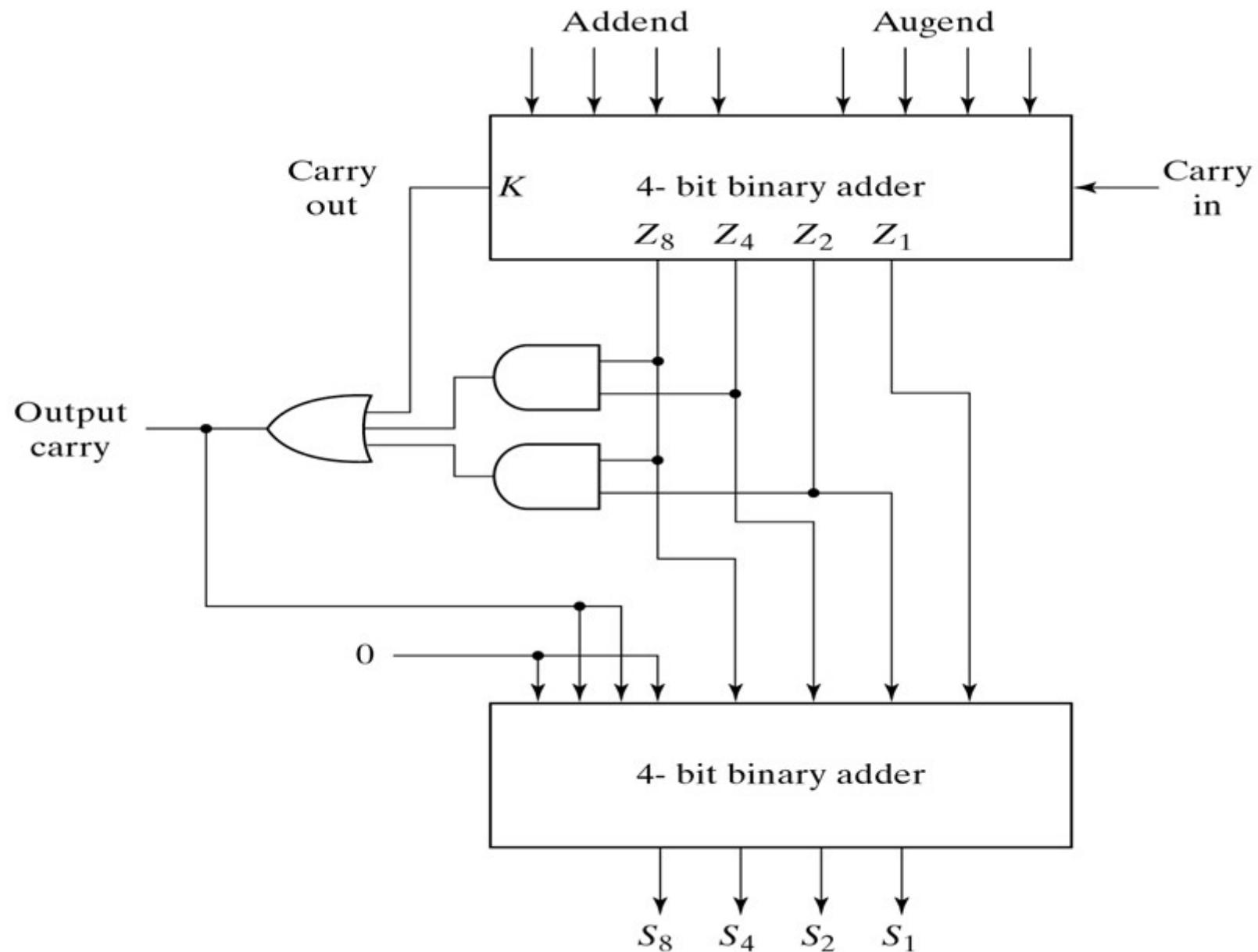
**Sum output S (S0 to S15)**



## BCD Adder

### Principle:

- Is Binary sum less than or equal to 1001 (9 in decimal)?
- For Binary sum more than 1001, add 0110 (6 in decimal) as **correction factor**
- Circuit needs modification accordingly



Block Diagram of a BCD Adder

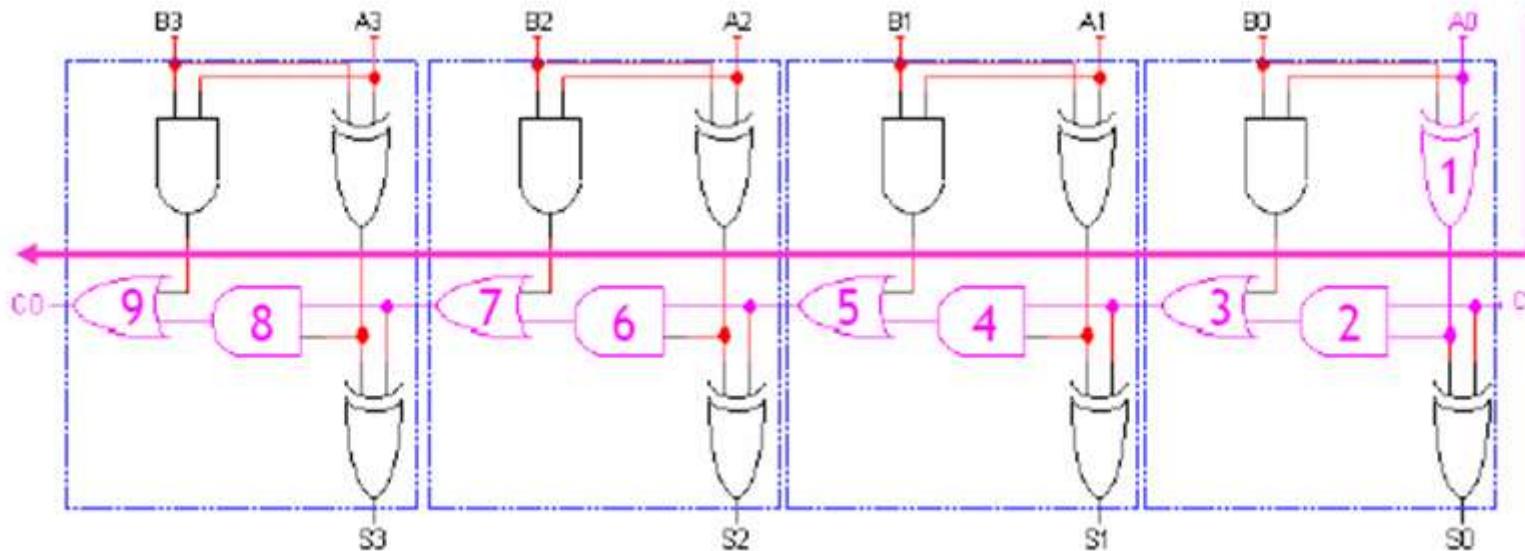


---

# Design of a Carry Look Ahead adder

# Ripple carry delays

- This is called a **ripple carry adder**, because the inputs A0, B0 and Cl "ripple" leftwards until CO and S3 are produced.
- Ripple carry adders are slow!
  - There is a very long path from A0, B0 and Cl to CO and S3.
  - For an  $n$ -bit ripple carry adder, the longest path has  $2n+1$  gates.
  - The longest path in a 64-bit adder would include 129 gates!



## A faster way to compute carry outs

- Instead of waiting for the carry out from each previous stage, we can minimize the delay by computing it directly with a two-level circuit.
- First we'll define two functions.
  - The "generate" function  $G_i$  produces 1 when there *must* be a carry out from position  $i$  (i.e., when  $A_i$  and  $B_i$  are both 1).

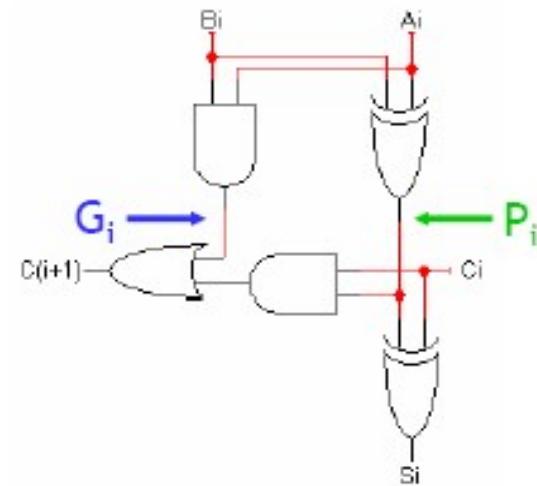
$$G_i = A_i B_i$$

- The "propagate" function  $P_i$  is true when an incoming carry is propagated (i.e, when  $A_i=1$  or  $B_i=1$ , but not both).

$$P_i = A_i \oplus B_i$$

- Then we can rewrite the carry out function.

$$C_{i+1} = G_i + P_i C_i$$

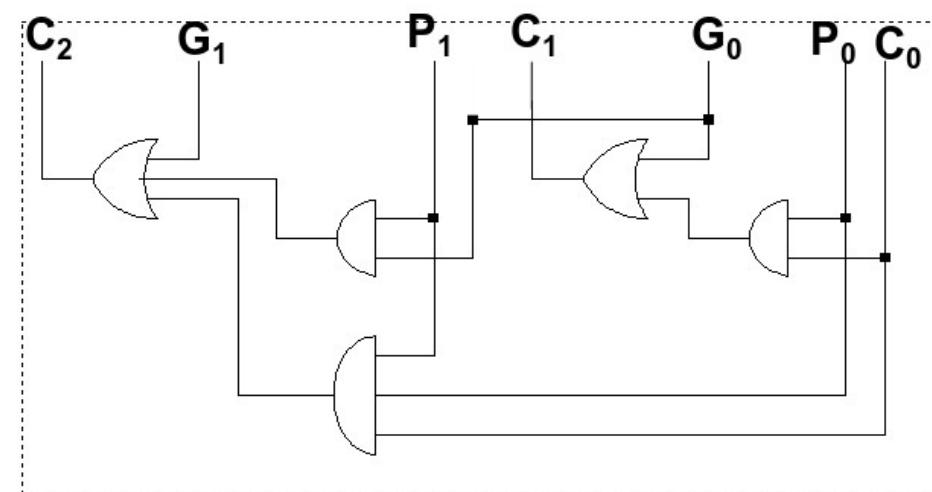
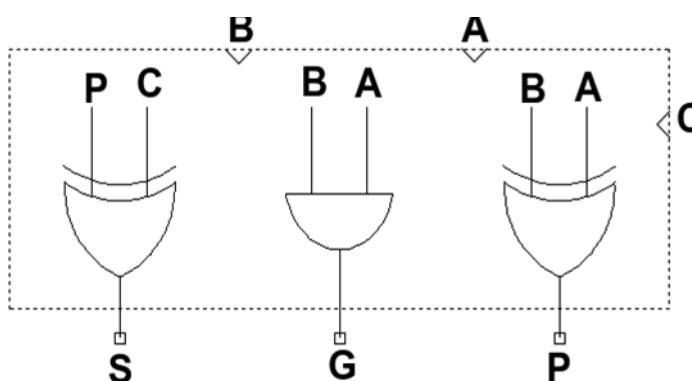


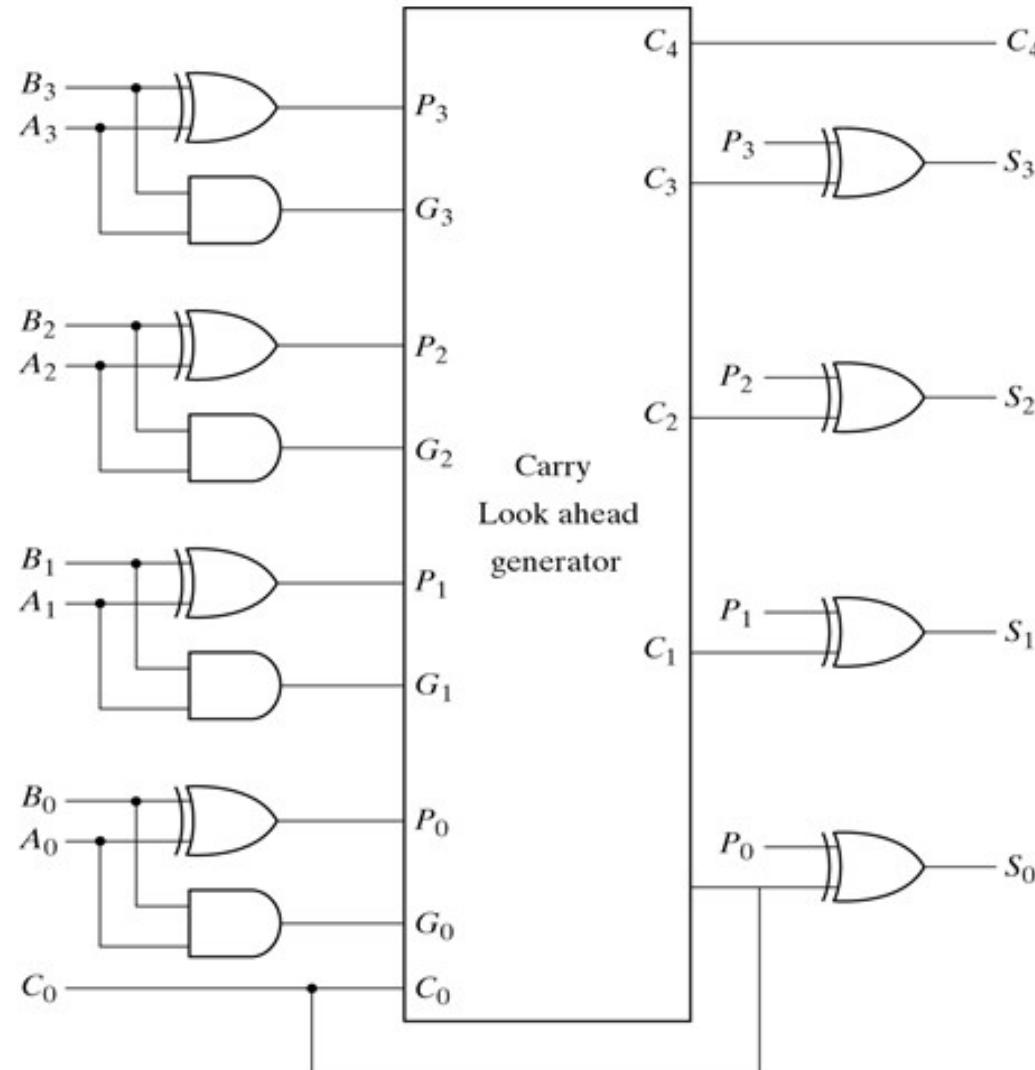
$A_i$	$B_i$	$C_i$	$C_{i+1}$
0	0	0	0
0	0	1	0
0	1	0	0
1	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

- Let's look at the carry out equations for specific bits, using the general equation from the previous page  $C_{i+1} = G_i + P_i C_i$ .

$$C_1 = G_0 + P_0 C_0$$

$$\begin{aligned}C_2 &= G_1 + P_1 C_1 \\&= G_1 + P_1(G_0 + P_0 C_0) \\&= G_1 + P_1 G_0 + P_1 P_0 C_0\end{aligned}$$





4-Bit Adder with Carry Lookahead



---

*Thank you*

# Digital Design

## Lecture 12: Binary Multipliers and Magnitude Comparators

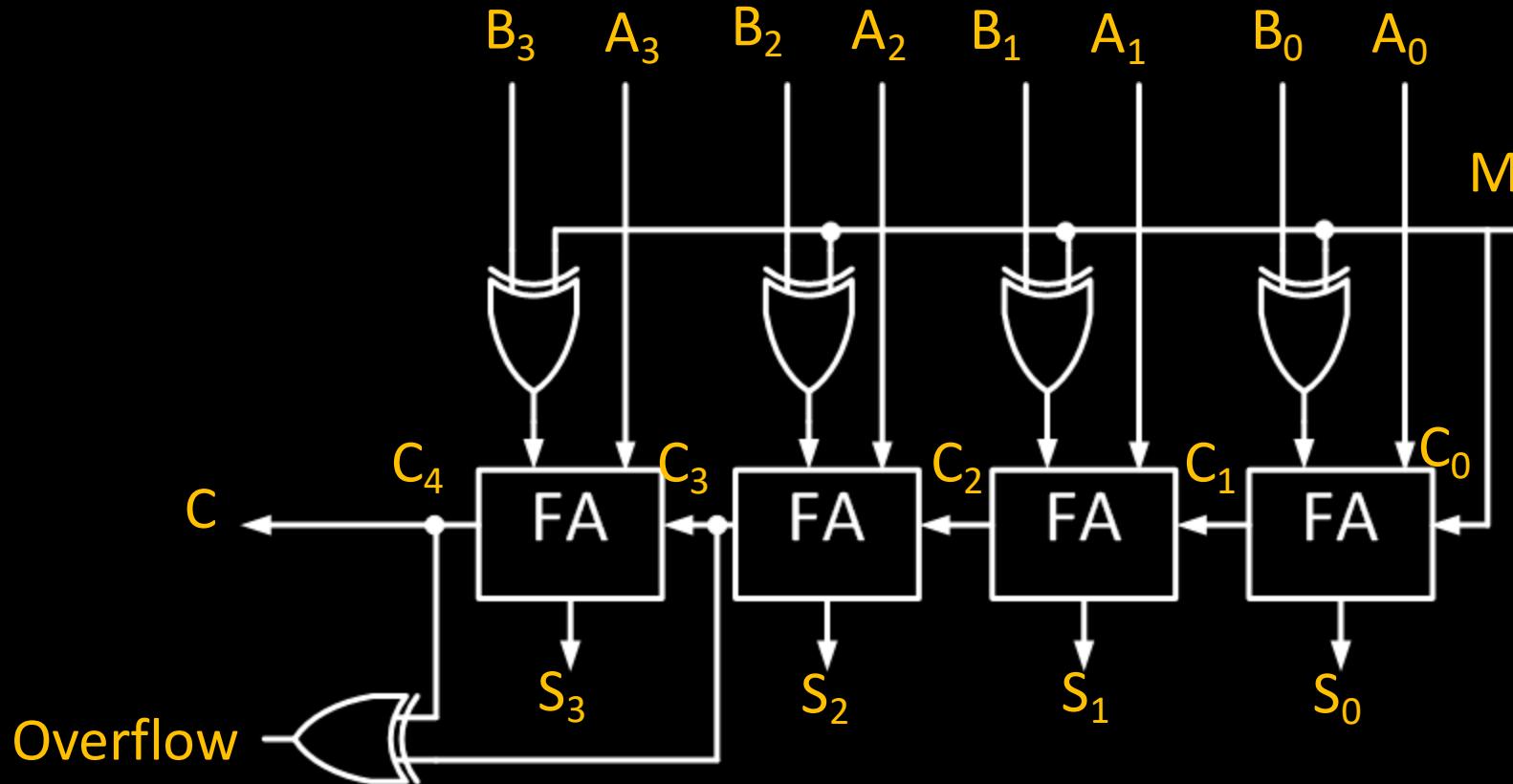


**Birla Institute of Technology & Science, Pilani**  
Hyderabad Campus  
9/18/2021





# 4-bit Adder subtractor with overflow detection





# Multiplier

$$\begin{array}{r} 2 \quad 3 \\ \times \quad 5 \\ \hline \end{array}$$



# Binary Multiplier

$$\begin{array}{r} \boxed{2} \quad \boxed{3} \\ \times \quad \boxed{1} \\ \hline \end{array}$$
$$\begin{array}{r} \boxed{2} \\ \times \quad \boxed{1} \\ \hline \end{array}$$
$$\begin{array}{r} 5 \\ \times \quad 1 \\ \hline 5 \end{array}$$
$$\begin{array}{r} 4 \quad 6 \\ + \quad 1 \quad 5 \\ \hline 1 \quad 1 \quad 9 \quad 6 \end{array}$$



# Binary Multiplier

$$\begin{array}{r} & 2 & 3 \\ & \times & \\ \hline & 5 & 2 \\ & \hline & 4 & 6 \\ & \hline 1 & 1 & 5 \\ & \hline 1 & 1 & 9 & 6 \end{array}$$

$$\begin{array}{r} & 1 & 0 \\ & \times & \\ \hline & 1 & \\ & \hline 1 & 0 \\ & \hline 1 & 0 & 0 \end{array}$$

$$2 \times 3 = 6$$



# Binary Multiplier (2-bit x 2-bit)

$$\begin{array}{r} & \text{B}_1 & \text{B}_0 \\ \times & \text{A}_1 & \text{A}_0 \\ \hline & \text{B}_1\text{A}_0 & \boxed{\text{B}_0\text{A}_0} \\ & \text{B}_1\text{A}_1 & \text{B}_0\text{A}_1 \\ \hline \text{P}_3 & \text{P}_2 & \text{P}_1 & \text{P}_0 \end{array}$$

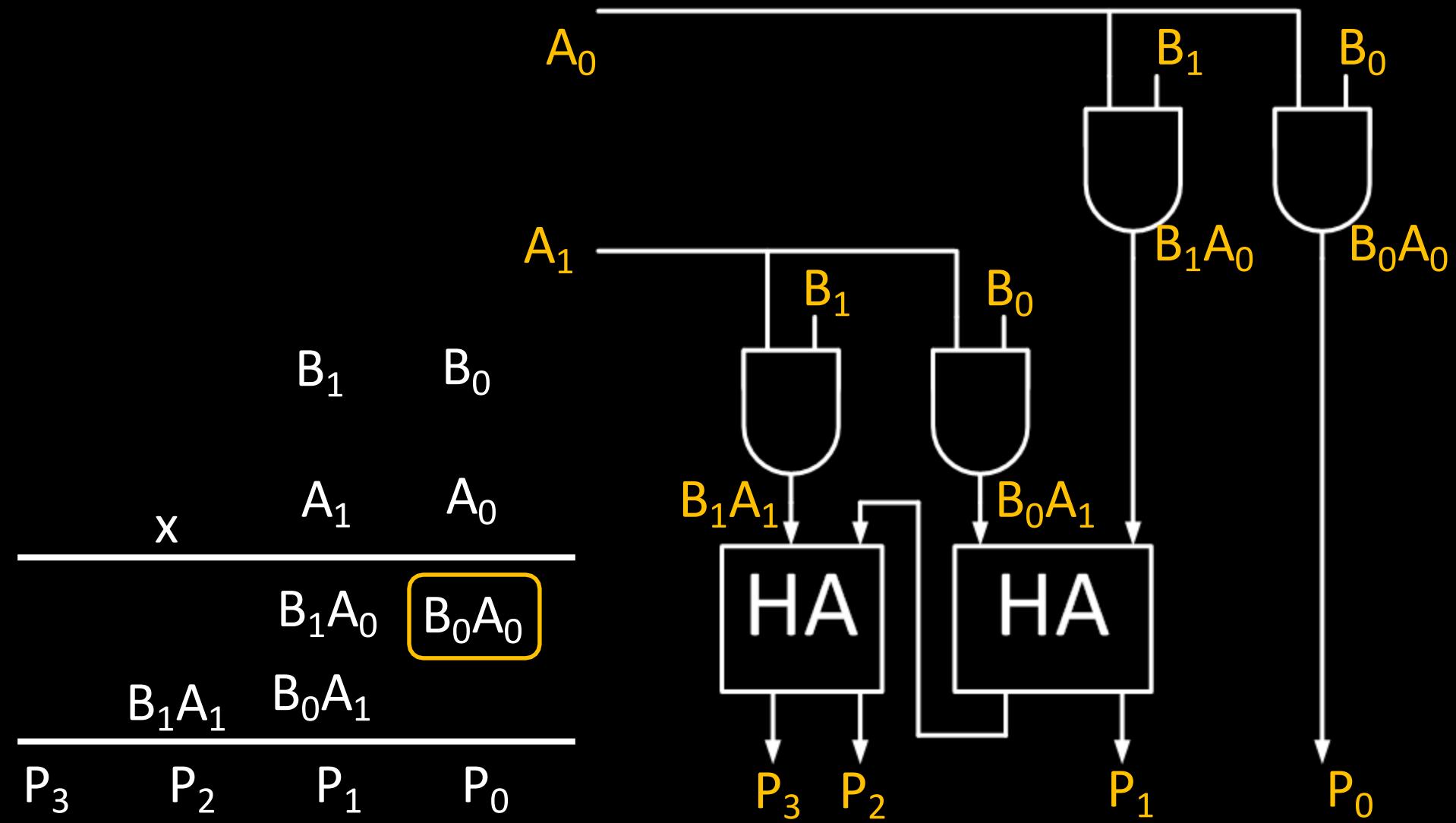
Gate for 1-bit Multiplication ( $\text{B}_0\text{A}_0$ ) ?

$\text{A}_0$	$\text{B}_0$	$F$
0	0	0
0	1	0
1	0	0
1	1	1





# Binary Multiplier Circuit (2-bit x 2-bit)





# Binary Multiplier (4-bit x 4-bit)

(Multiplicand)  $B_3 B_2 B_1 B_0$

(Multiplier)  $A_3 A_2 A_1 A_0$

---

$A_0 B_3 \quad A_0 B_2 \quad A_0 B_1 \quad A_0 B_0$

$A_1 B_3 \quad A_1 B_2 \quad A_1 B_1 \quad A_1 B_0$

$A_2 B_3 \quad A_2 B_2 \quad A_2 B_1 \quad A_2 B_0$

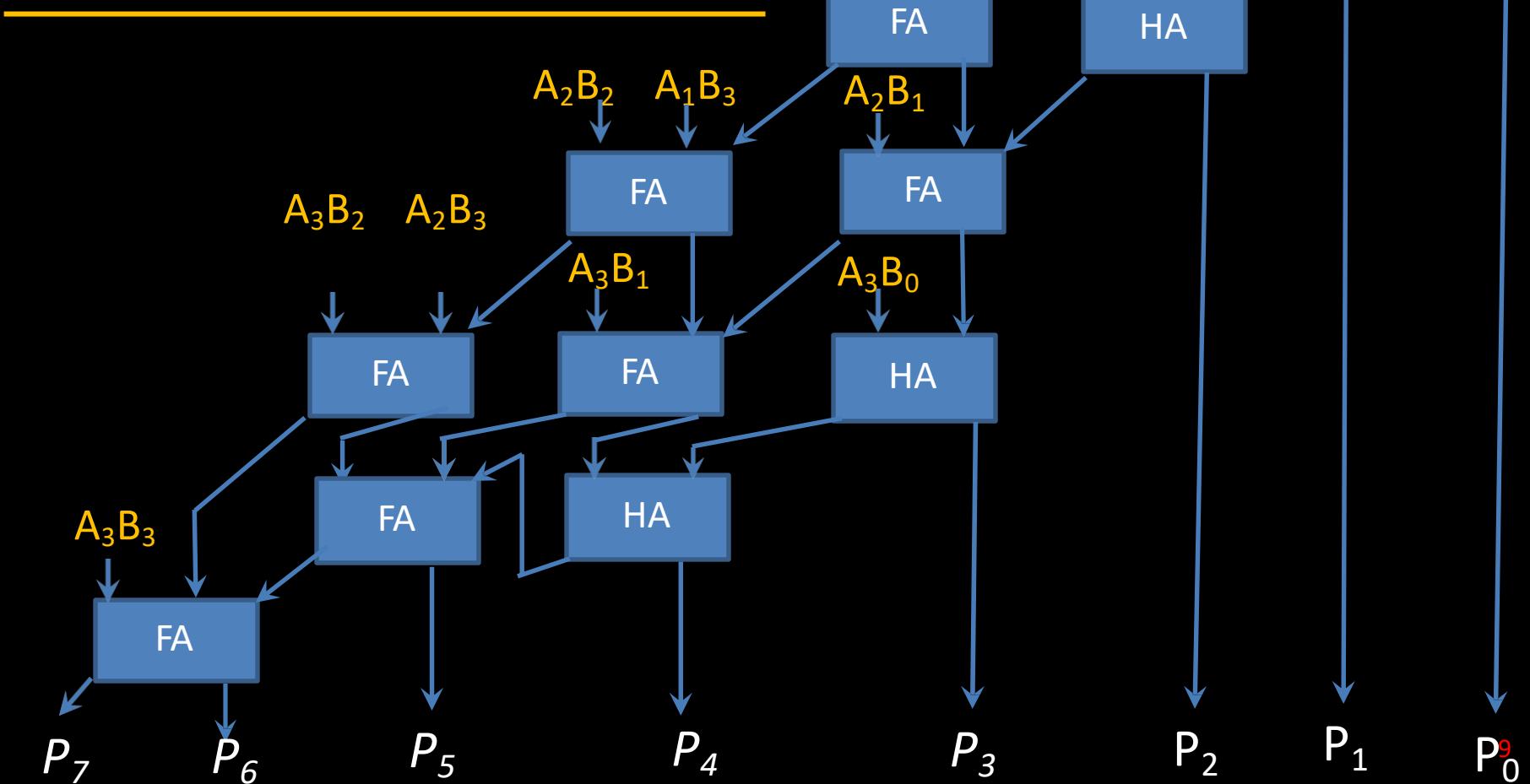
$A_3 B_3 \quad A_3 B_2 \quad A_3 B_1 \quad A_3 B_0$

---

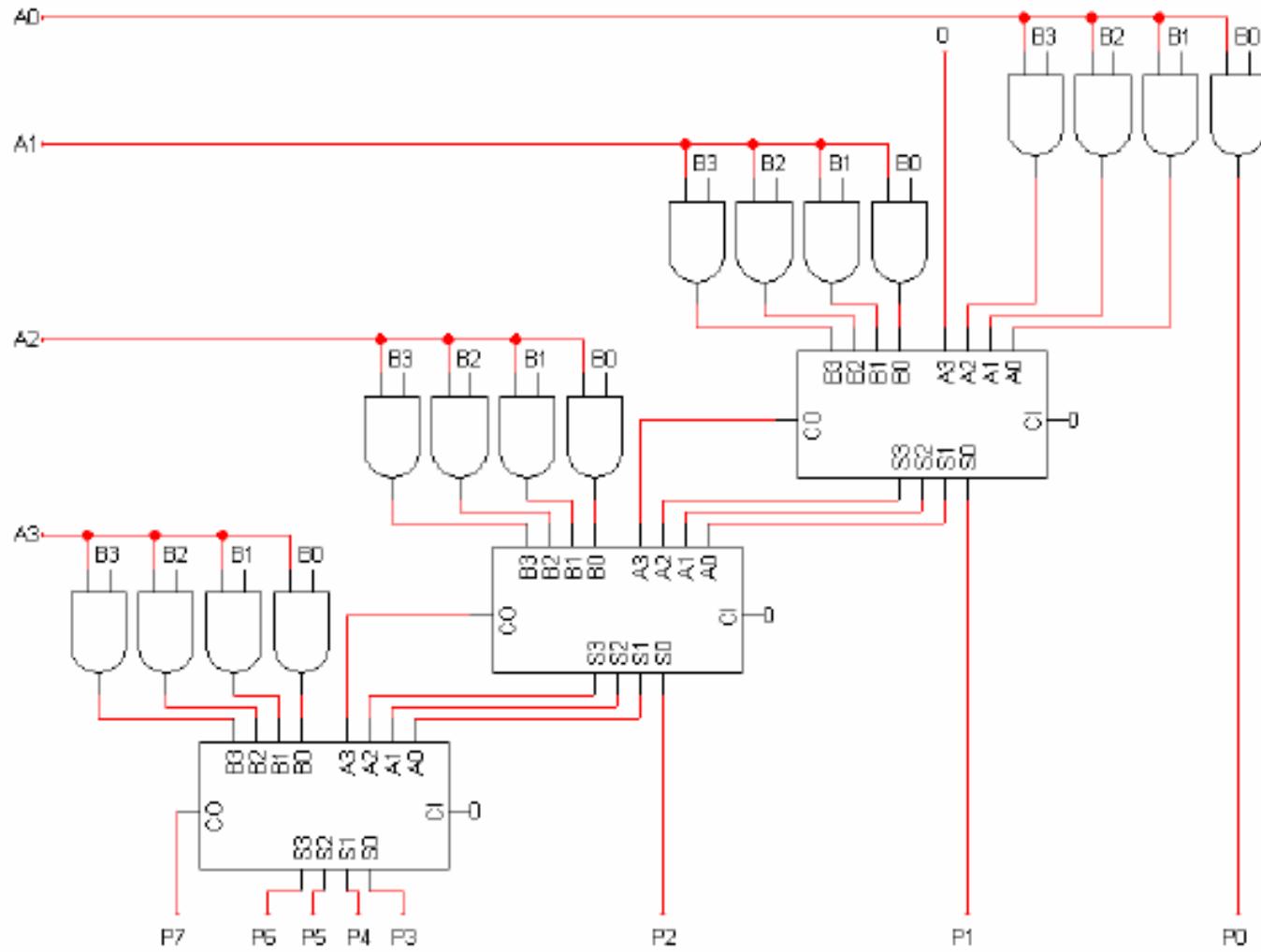


# 4 x 4 bit Binary Multiplier Circuit

	$A_0B_3$	$A_0B_2$	$A_0B_1$	$A_0B_0$
$A_1B_3$	$A_1B_2$	$A_1B_1$	$A_1B_0$	
$A_2B_3$	$A_2B_2$	$A_2B_1$	$A_2B_0$	
$A_3B_3$	$A_3B_2$	$A_3B_1$	$A_3B_0$	



# A $4 \times 4$ binary multiplier





# Magnitude Comparator

A Magnitude comparator is a **combinational circuit** that compares two numbers A and B .

The output of the magnitude comparator are three signals that indicate if **(A>B)**, **(A<B)**, **(A=B)**

Even a 3-bit comparator results in 6 inputs – Too complex to solve using K-maps



# Magnitude Comparator

How do you compare two numbers?

2	3	4
5	6	4

Compare most significant digit

If most significant digit of first number is greater than the most significant digit of second number then

First Number > Second number



# Magnitude Comparator

If most significant digits are equal ??

2	3	4
2	6	4

Compare the next significant digit

If next significant digit of first number is greater than the next significant digit of second number then

First Number > Second number



# Magnitude Comparator

1	0	1
1	1	0



# Magnitude Comparator

1	0	1
1	1	0



# Magnitude Comparator

## 1-Bit Comparator

$A_0$

$B_0$

Let  $G_0$  indicate greater,  $L_0$  indicate Lesser and  $E_0$  indicate equal

$A_0$	$B_0$	$G_0$	$L_0$	$E_0$
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1

$$G_0 = A_0 B_0'$$

$$L_0 = A_0' B_0$$

$$E_0 = (A_0 \oplus B_0)' = (A_0 \ominus B_0) = (G_0 + L_0)'$$



# Magnitude Comparator

1-Bit Comparator

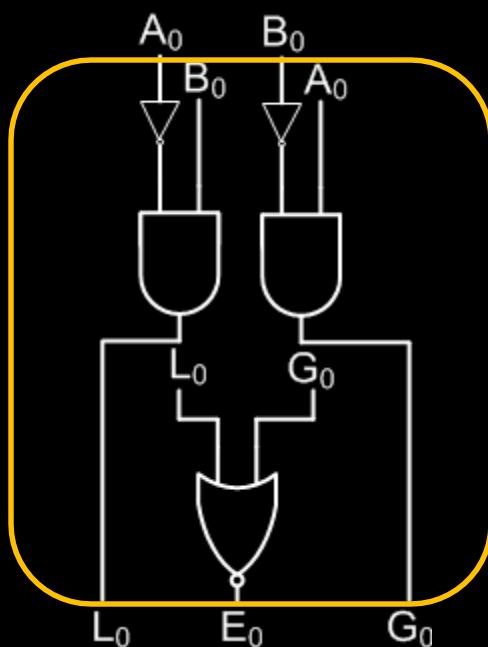
$A_0$

$B_0$

$$G_0 = A_0 B_0'$$

$$L_0 = A_0' B_0$$

$$E_0 = (G_0 + L_0)' = (A_0 \oplus B_0)'$$





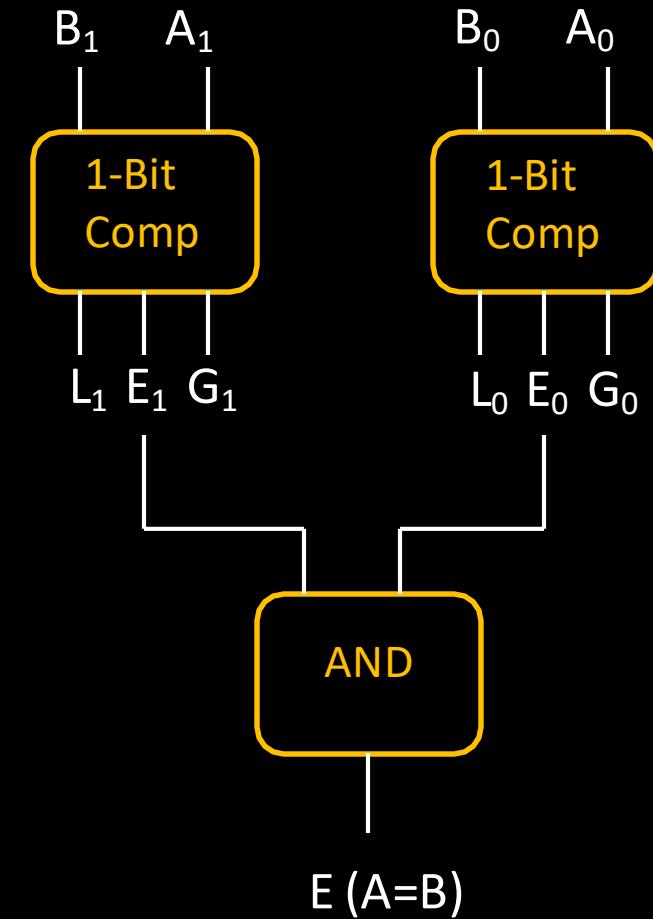
# Magnitude Comparator

## 2-Bit Comparator

A	$A_1 A_0$
B	$B_1 B_0$

$A = B$  When  $A_0 = B_0$  and  $A_1 = B_1$

$$E = E_0 \cdot E_1$$





# Magnitude Comparator

## 2-Bit Comparator

A       $A_1 A_0$

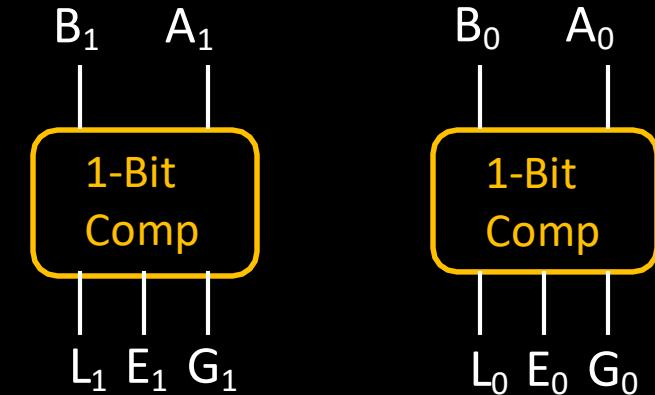
B       $B_1 B_0$



# Magnitude Comparator

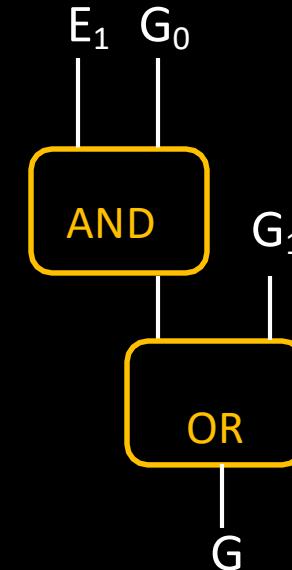
## 2-Bit Comparator

A	$A_1 A_0$
B	$B_1 B_0$



$A > B$  When  $A_1 > B_1$  OR  
When  $A_1 = B_1$  and  $A_0 > B_0$

$$G = G_1 + E_1 \cdot G_0$$

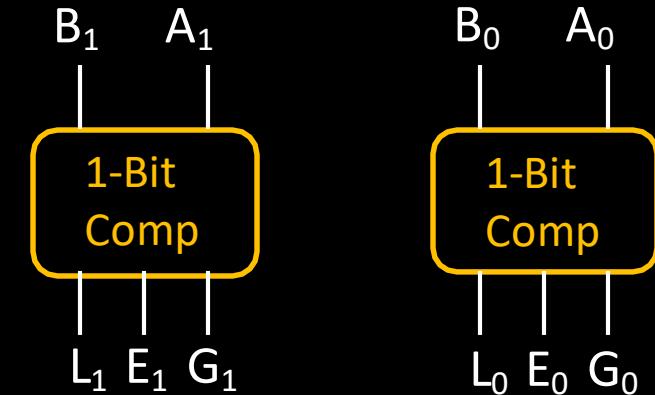




# Magnitude Comparator

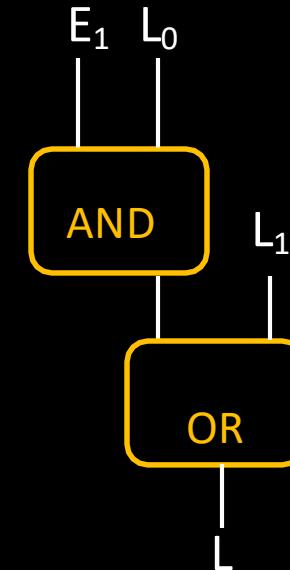
## 2-Bit Comparator

A	$A_1 A_0$
B	$B_1 B_0$



$A < B$  When  $A_1 < B_1$  OR  
When  $A_1 = B_1$  and  $A_0 < B_0$

$$L = L_1 + E_1 \cdot L_0$$





# Magnitude Comparator

## 1 Bit Comparator

$$G = G_0 \quad L = L_0 \quad E = E_0$$

## 2 Bit Comparator

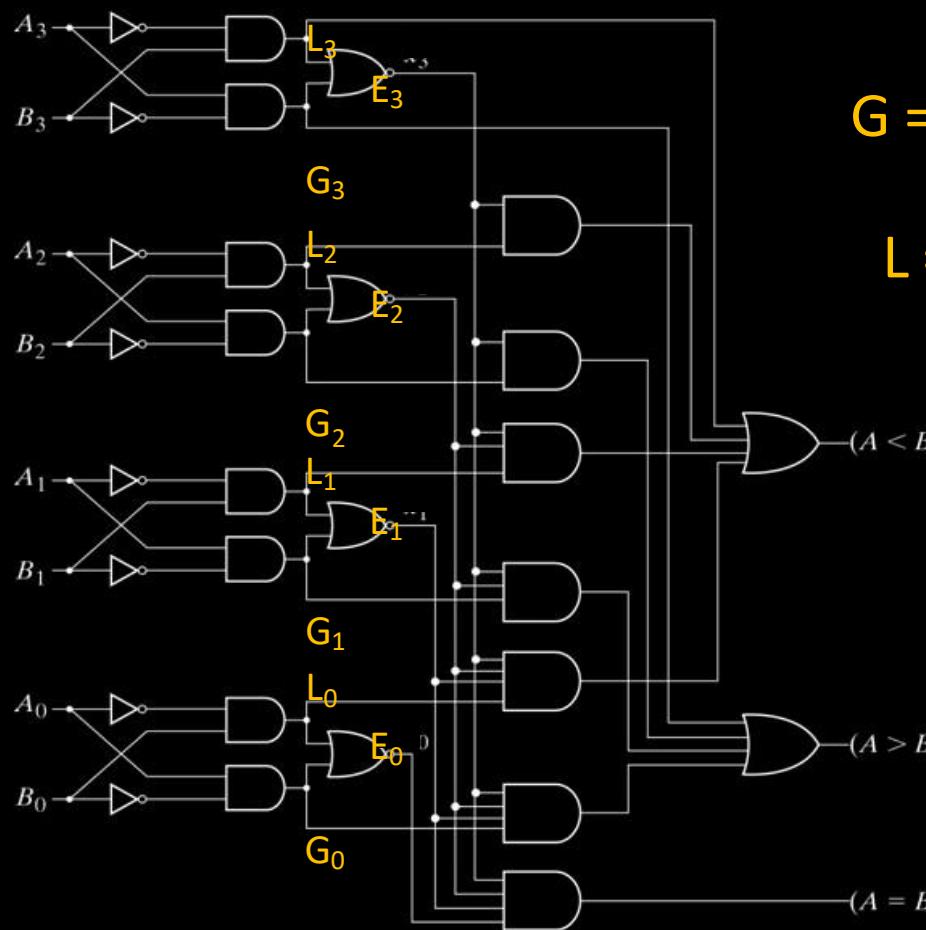
$$G = G_1 + E_1 \cdot G_0 \quad L = L_1 + E_1 \cdot L_0 \quad E = E_1 \cdot E_0$$

## 3 Bit Comparator

$$G = G_2 + E_2 \cdot G_1 + E_2 \cdot E_1 \cdot G_0 \quad E = E_2 \cdot E_1 \cdot E_0$$
$$L = L_2 + E_2 \cdot L_1 + E_2 \cdot E_1 \cdot L_0$$



# Magnitude Comparator



4-Bit Comparator

$$G = G_3 + E_3 \cdot G_2 + E_3 \cdot E_2 \cdot G_1 + E_3 \cdot E_2 \cdot E_1 \cdot G_0$$

$$L = L_3 + E_3 \cdot L_2 + E_3 \cdot E_2 \cdot L_1 + E_3 \cdot E_2 \cdot E_1 \cdot L_0$$

$$E = E_2 \cdot E_1 \cdot E_0$$

Can you draw the circuit ??



# Next Module

Decoders

Encoders

Multiplexers



# Thank You

# Digital Design

## Lecture 13: Decoders and Encoders



**Birla Institute of Technology & Science, Pilani**  
Hyderabad Campus

10/4/2020





## Decoder

A Decoder is a **combinational circuit** that converts binary information from ' $n$ ' input lines to  $2^n$  output lines

$n$  to  $m$  decoders where  $m \leq 2^n$

The name decoder is also used in conjunction with other code converters (e.g. BCD to seven segment decoder)



# Decoder

An Example: 2 to 4 Line Decoder

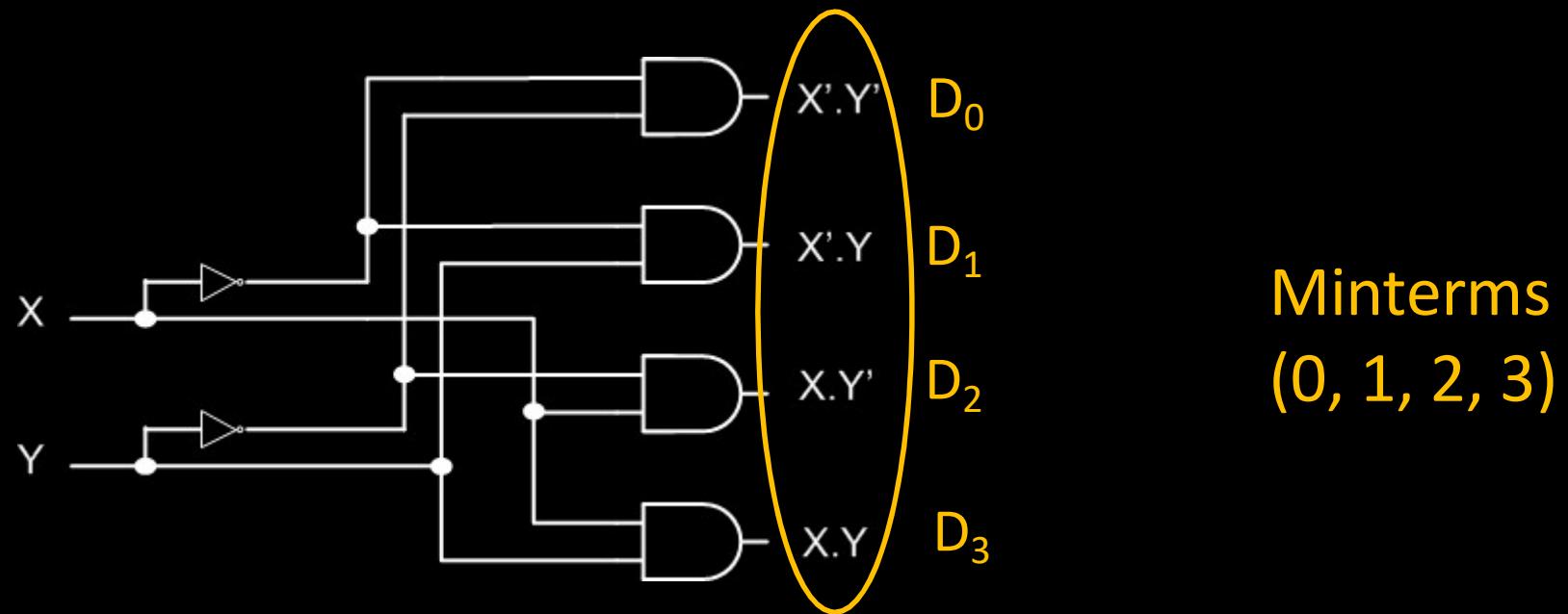
X	Y
0	0
0	1
1	0
1	1

D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1



# Decoder

An Example: 2 to 4 Line Decoder



Minterm Generating circuit



# Decoder

An Example: 2 to 4 Line Decoder with Enable

E	X	Y
1	0	0
1	0	1
1	1	0
1	1	1
0	X	X

D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
0	0	0	0

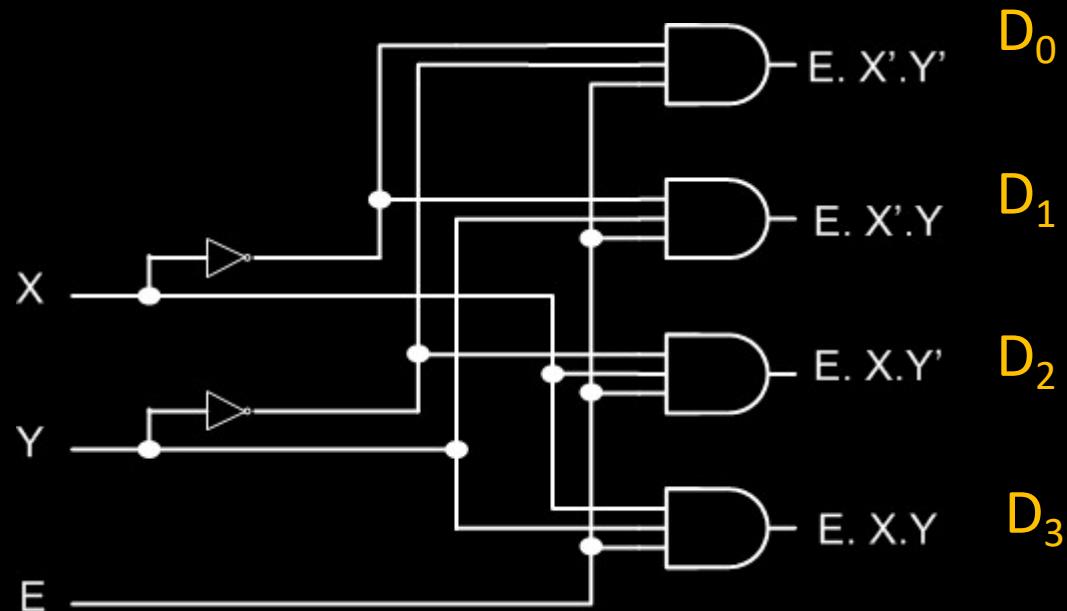
Decoder with enable pin is also called as **Demultiplexer**

X and Y – Select Lines    E- Input line



# Decoder

An Example: 2 to 4 Line Decoder with Enable





# Decoder

An Example: 3 to 8 Line Decoder

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

D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

Direct Implementation – Home Assignment

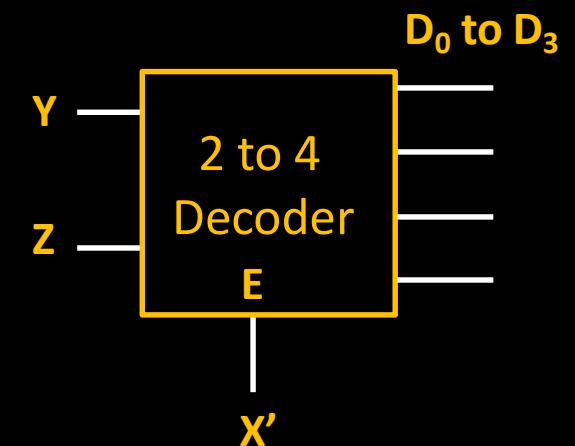


# Decoder

Can you implement a 3 to 8 decoder using two 2 to 4 decoders with enable pin

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

D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1



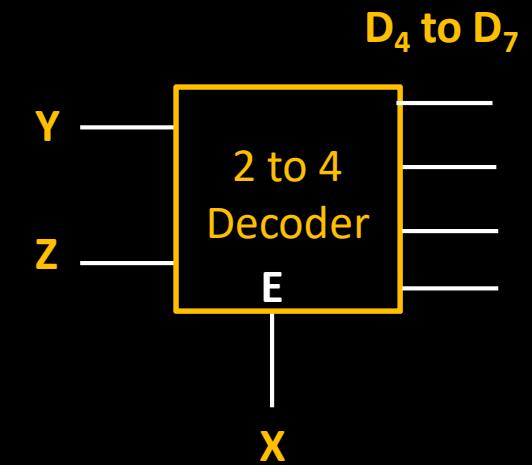


# Decoder

Can implement a 3 to 8 decoder using two 2 to 4 decoders with enable pin

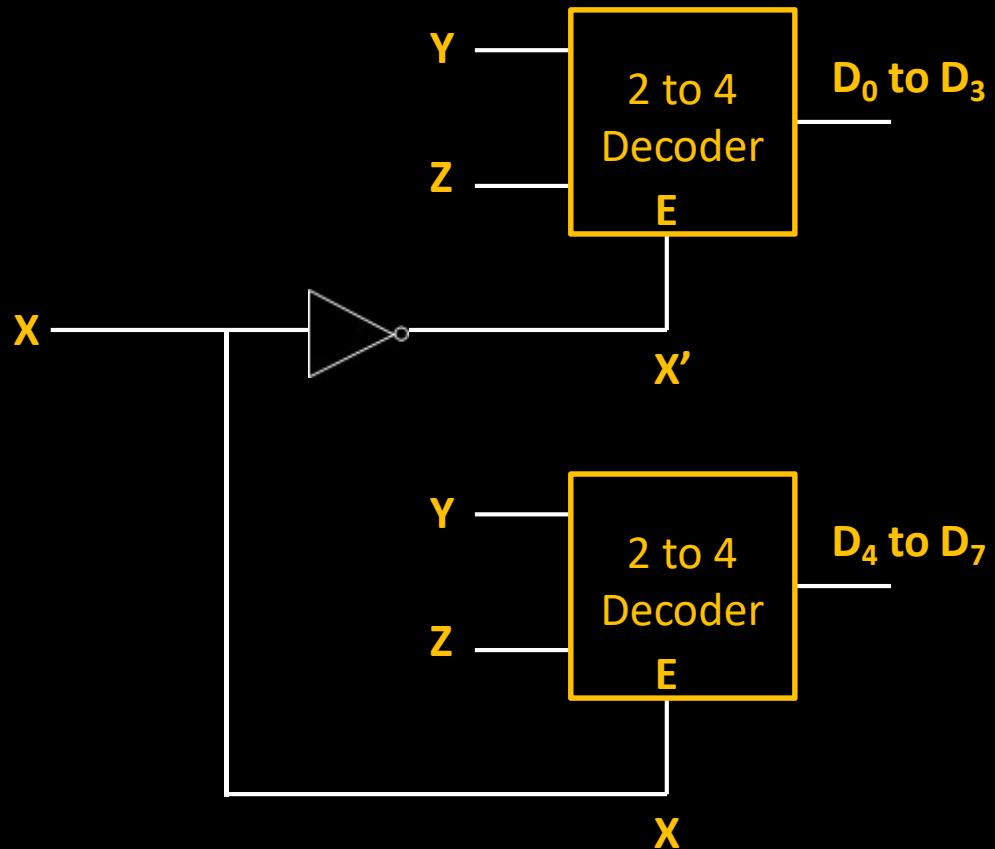
X	Y	Z
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1





## 3 to 8 Decoder using 2nos of 2 to 4 Decoder

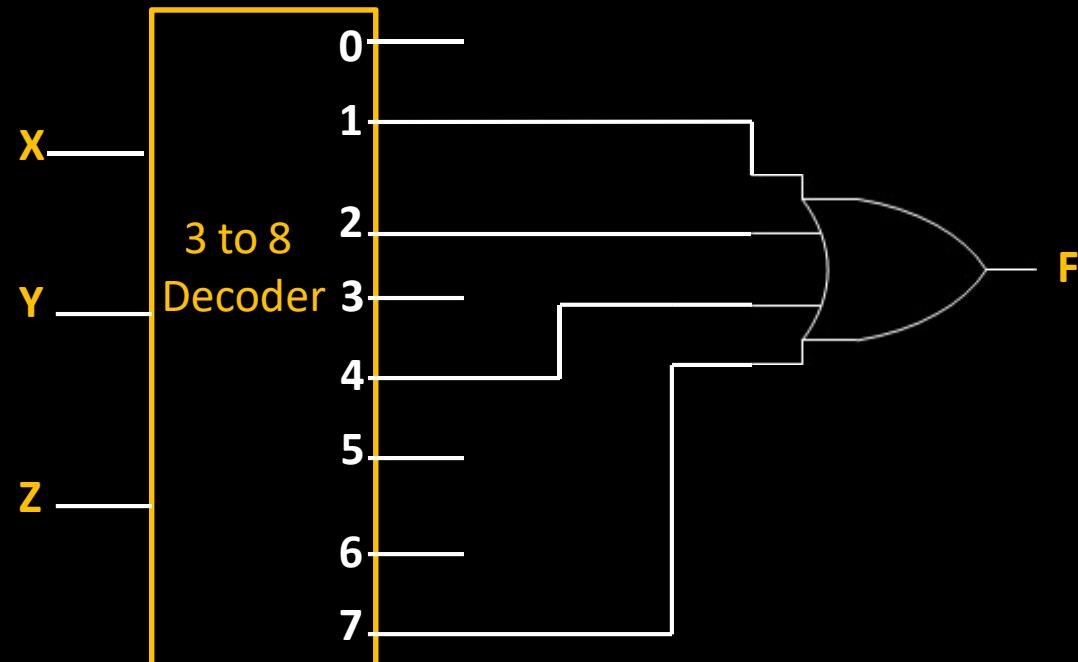




# Decoder

Implementing function using Decoders

$$F = \sum (1, 2, 4, 7) = X'Y'Z + X'YZ' + XY'Z' + XYZ$$





# Encoder

Encoder - combinational circuit that performs inverse operation of decoder

Has  $2^n$  or fewer input lines and n output lines

An Example:

$$X = D_2 + D_3$$

$$Y = D_1 + D_3$$

<b>D<sub>0</sub></b>	<b>D<sub>1</sub></b>	<b>D<sub>2</sub></b>	<b>D<sub>3</sub></b>	<b>X</b>	<b>Y</b>
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1



## Encoder

$$X = D_2 + D_3$$

$$Y = D_1 + D_3$$

$D_0$	$D_1$	$D_2$	$D_3$	$X$	$Y$
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

### Limitations:

If two ones appear i.e. if  $D_1$  and  $D_2$  are '1' then both X and Y will be '1' resulting in output 11

If all the inputs are zeros then output is 00 – which indicates that  $D_0$  is 1



## Priority Encoder

If two ones appear i.e. if  $D_1$  and  $D_2$  are '1' then both X and Y will be '1' resulting in output 11

Encoder circuits must establish **input priority**

E.g. If high priority is given to inputs with higher subscripts  
If both  $D_1$  and  $D_2$  are '1' then resulting o/p should be 10



## Priority Encoder

If all the inputs are zeros then output is 00 – which indicates that  $D_0$  is 1

This ambiguity can be resolved by providing one more output.

Extra Output indicates whether at least one input is equal to 1



# Priority Encoder

D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

X	Y
0	0
0	1
1	0
1	1

Encoder

$$X = D_2 + D_3$$

$$Y = D_1 + D_3$$

D <sub>0</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>
1	0	0	0
X	1	0	0
X	X	1	0
X	X	X	1
0	0	0	0

X	Y	V
0	0	1
0	1	1
1	0	1
1	1	1
X	X	0

Priority Encoder

$$X = D_2 + D_3$$

$$Y = D_1 D_2' + D_3$$

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



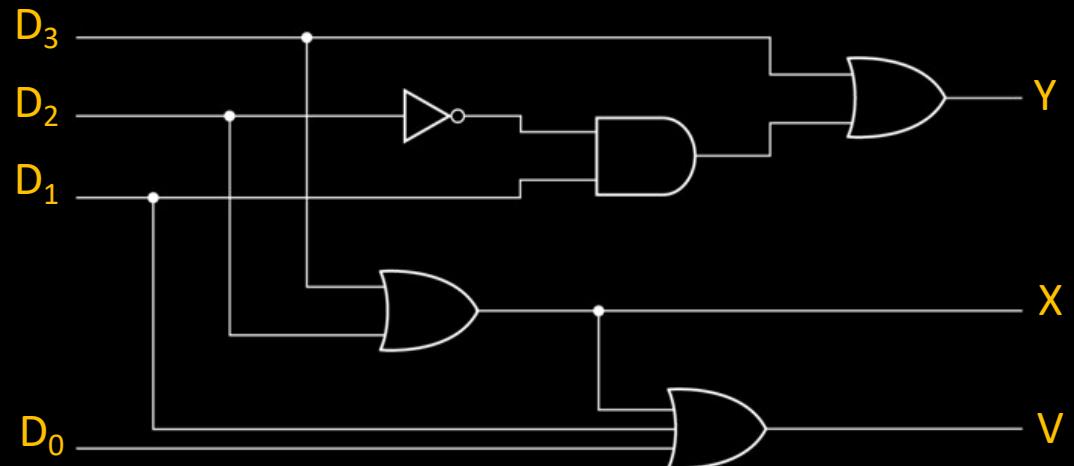
# Priority Encoder

## Priority Encoder

$$X = D_2 + D_3$$

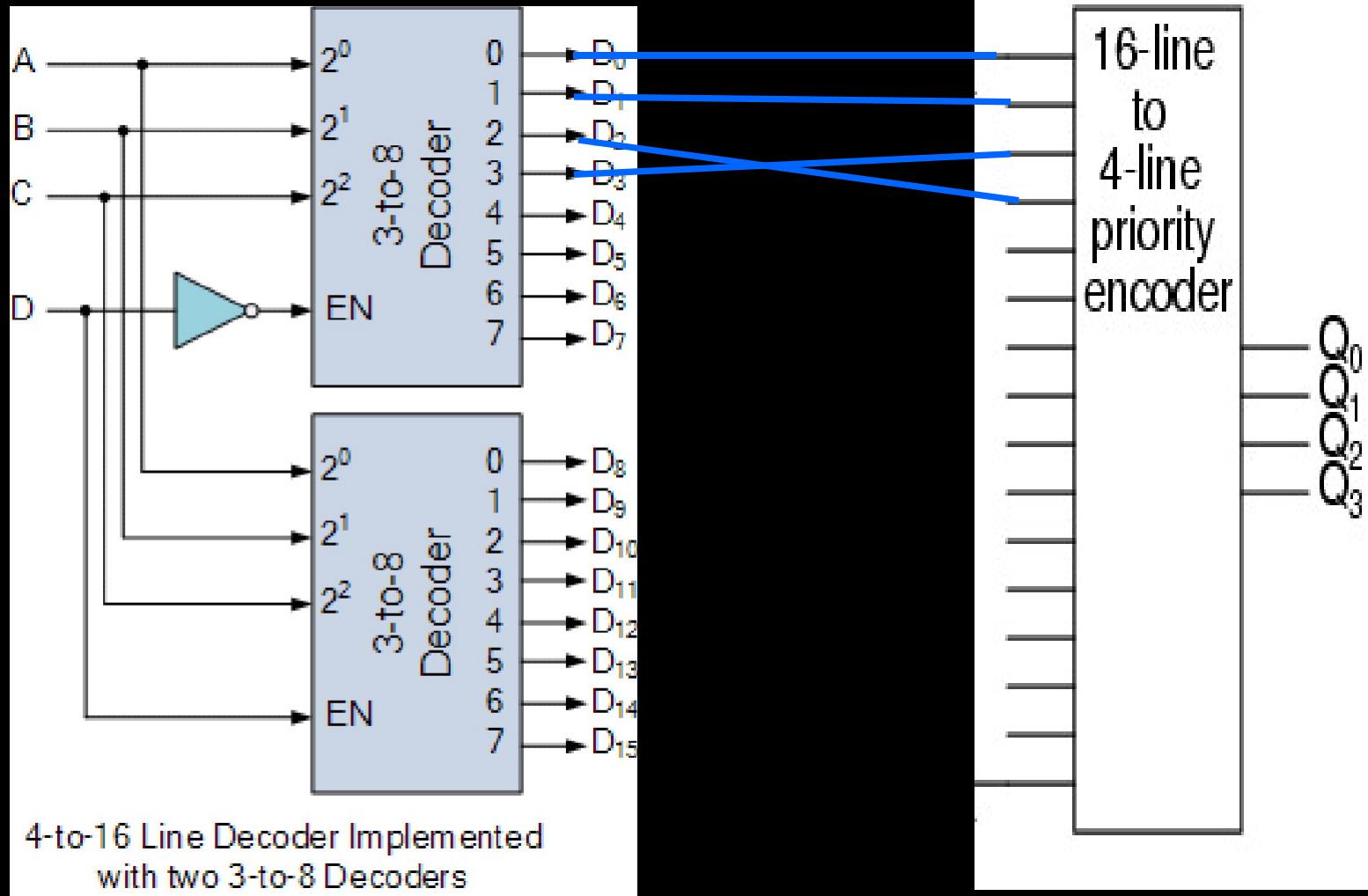
$$Y = D_1D_2' + D_3$$

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



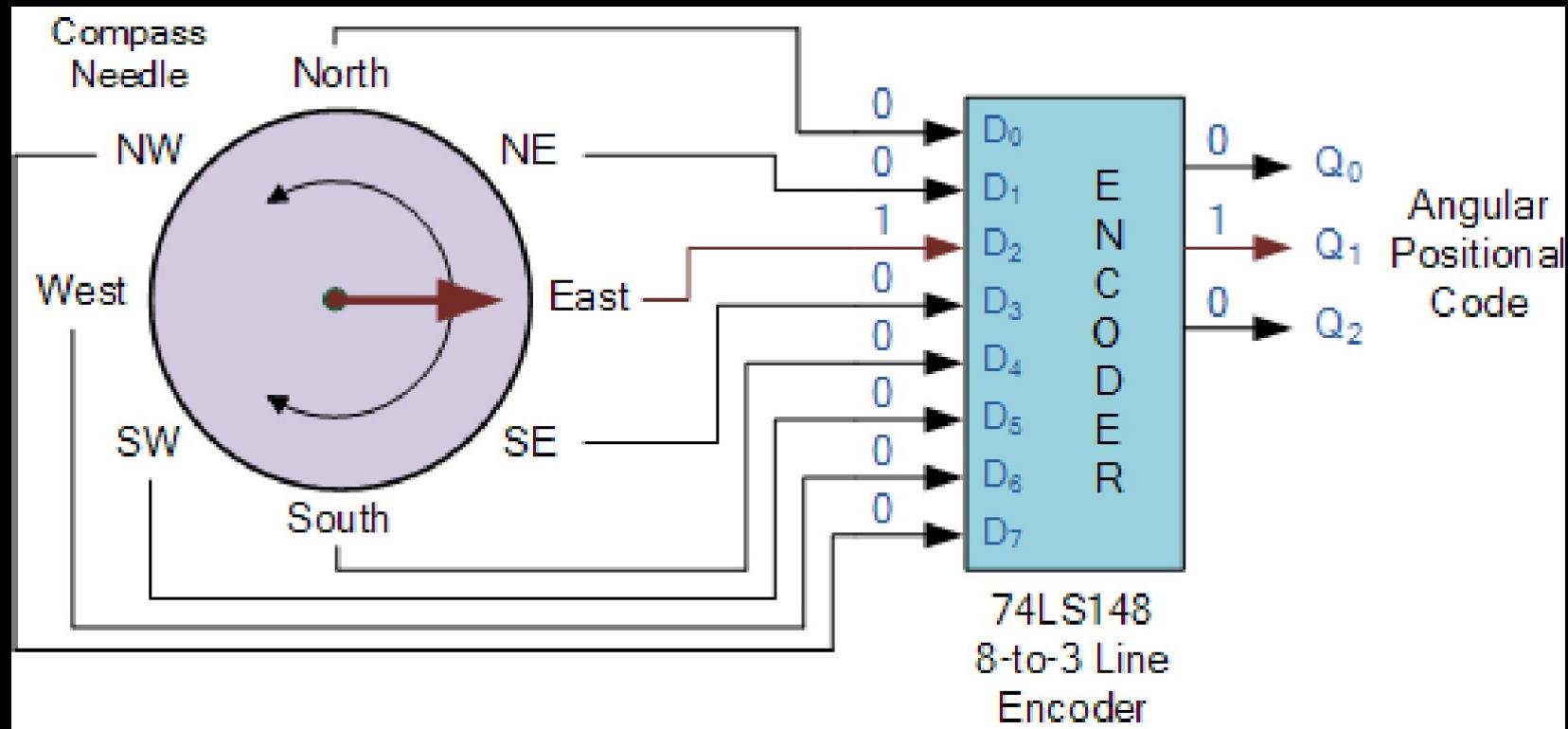


# Binary to Gray Using Encoder-Decoder





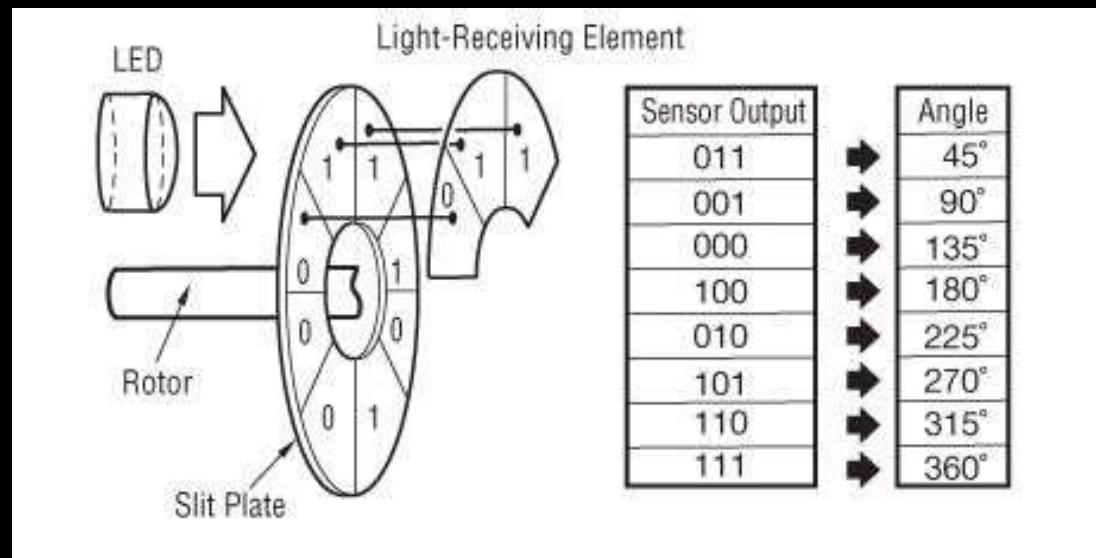
# Encoder Applications





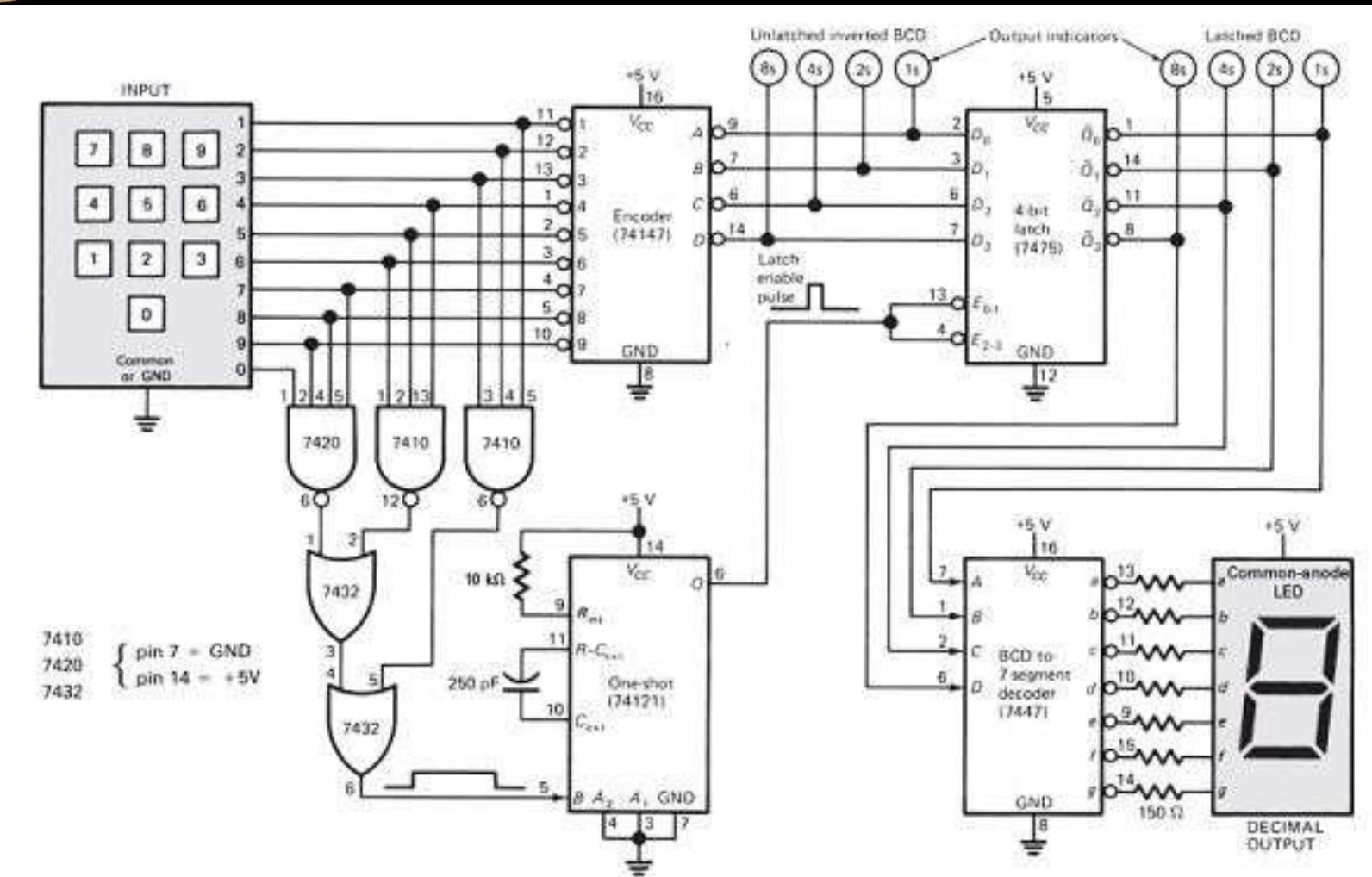
# Encoder Applications :

## Optical Encoder for motor position control



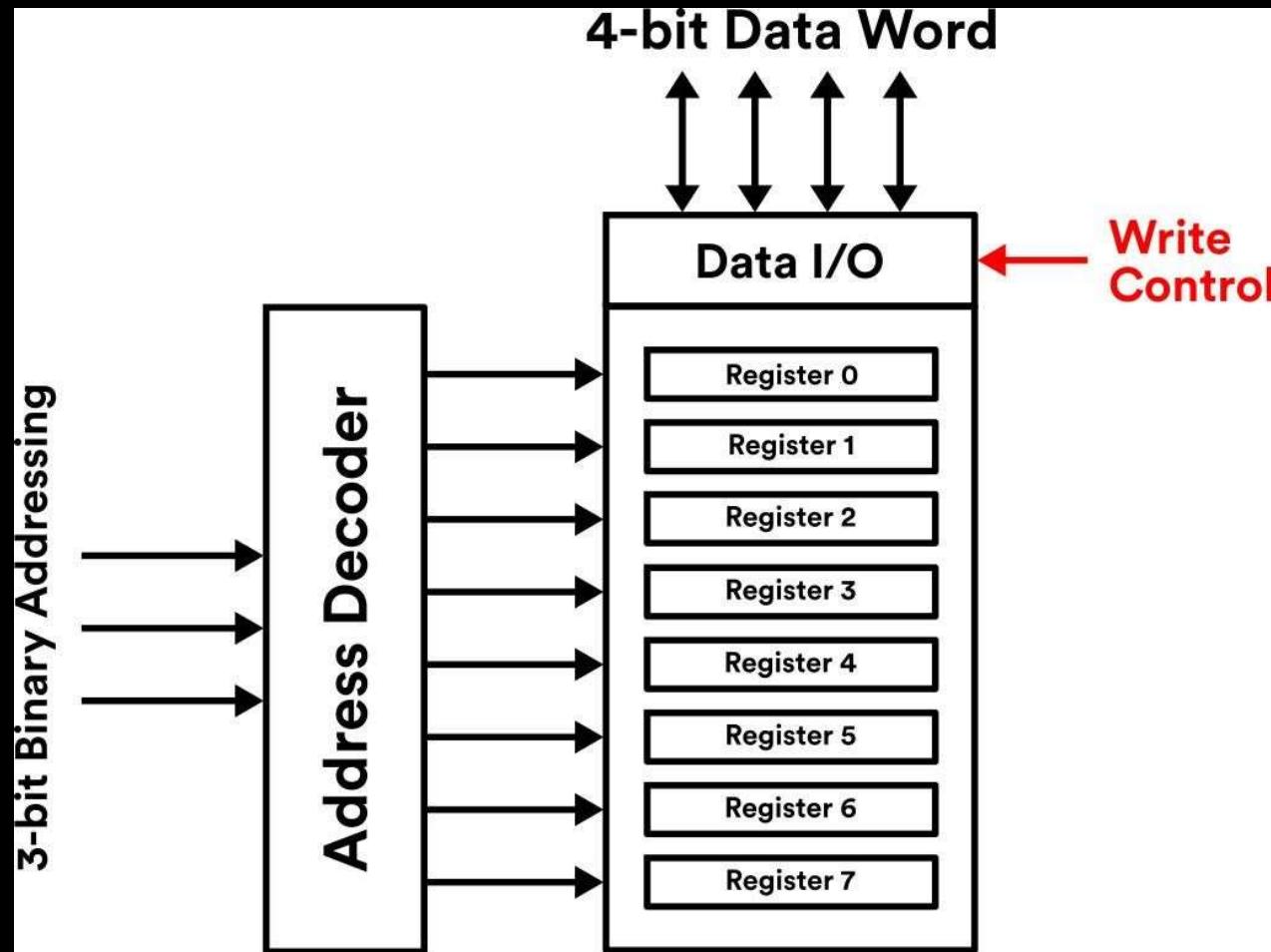


# Encoder Applications





# Decoder Applications





# Thank You

# Digital Design

## Lecture 14: Multiplexer



**Birla Institute of Technology & Science, Pilani**  
Hyderabad Campus

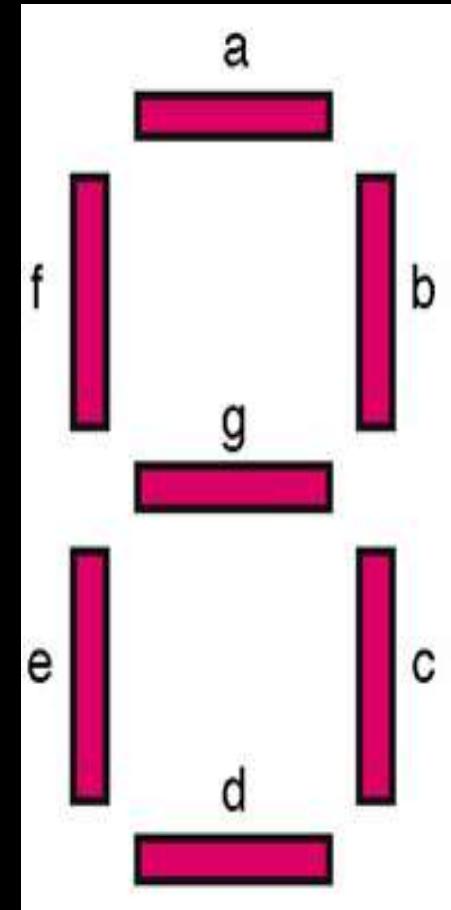
9/23/2021

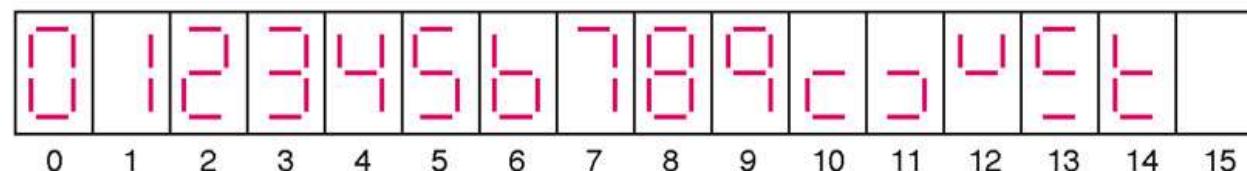
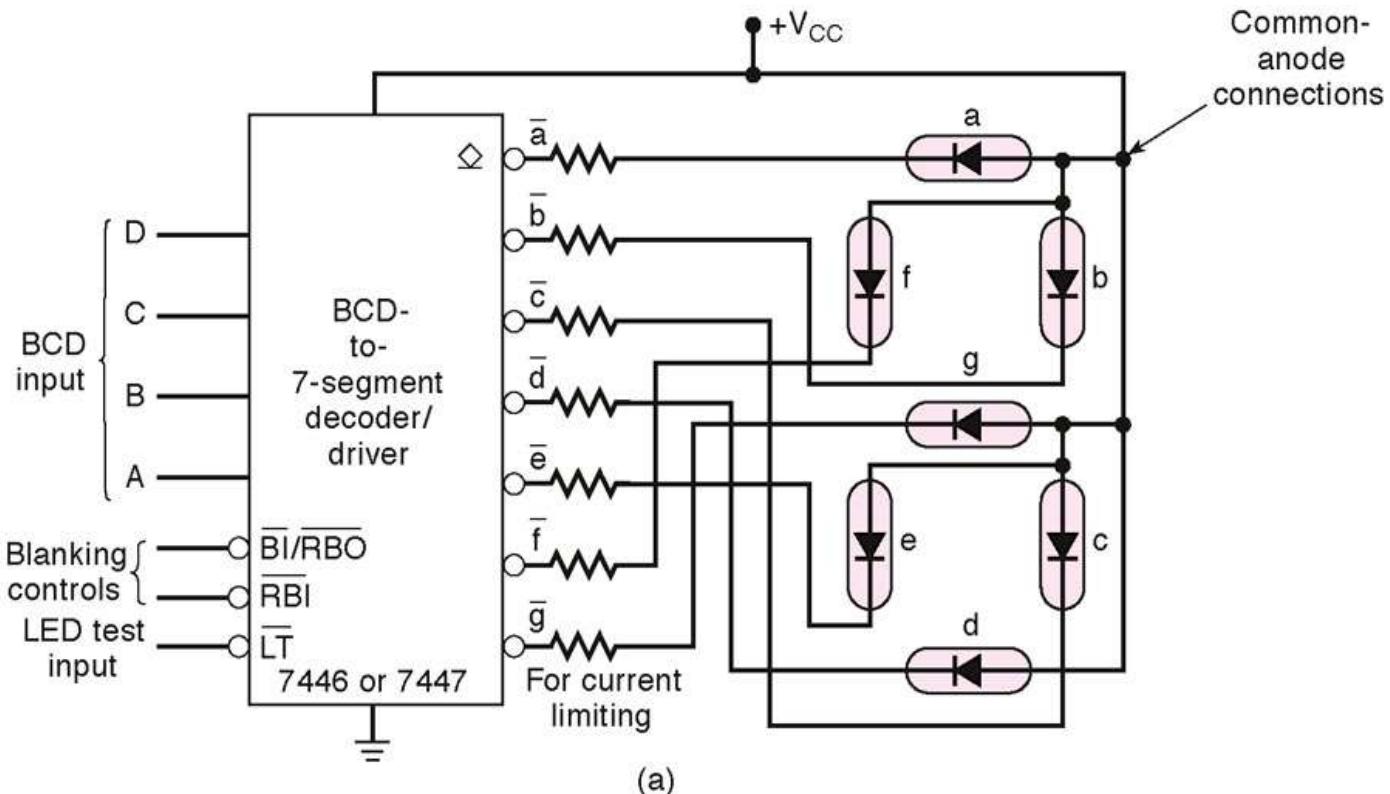


## BCD-7segment display using decoders

Most digital equipment has some means for displaying information in a form that can be understood by the user.

One of the simplest and most popular methods for displaying numerical digits uses a 7-segment configuration to form digital characters 0 to 9 and sometimes the hex characters A to F





(b)



## Multiplexers

Multiplexer - combinational circuit that selects binary information from one of many input lines and directs it to single output line

The selection is controlled by Select lines

In general,  $2^n$  input lines, 'n' selection lines and 1 output



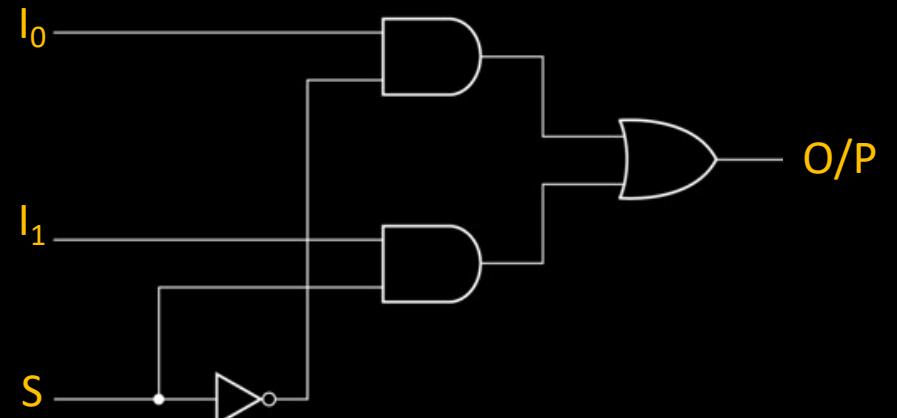
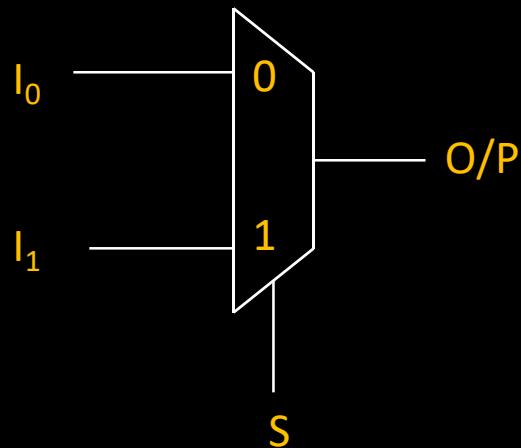
# Multiplexers

## 2:1 Multiplexer – 2:1 MUX

$I_0$  and  $I_1$  are input lines and 'S' is select line

If  $s=0$  then  $o/p = I_0$   
 $s=1$  then  $o/p = I_1$

$$o/p = I_0 \cdot S' + I_1 \cdot S$$

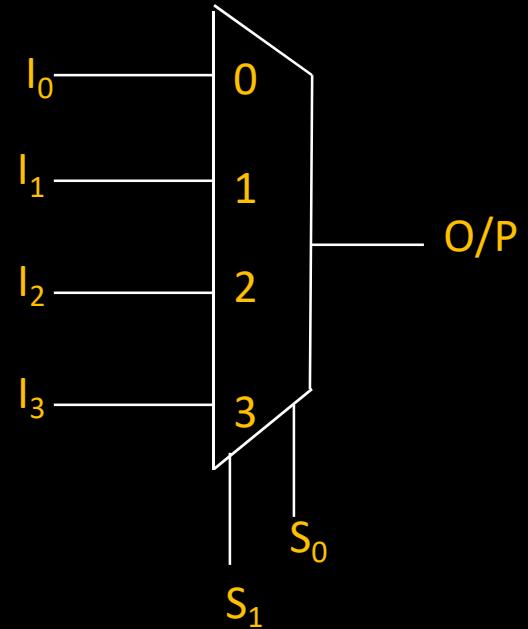




# Multiplexers

## 4:1 Multiplexer – 4:1 MUX

$S_1$	$S_0$	O/P
0	0	$I_0$
0	1	$I_1$
1	0	$I_2$
1	1	$I_3$



$$O/P = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3$$

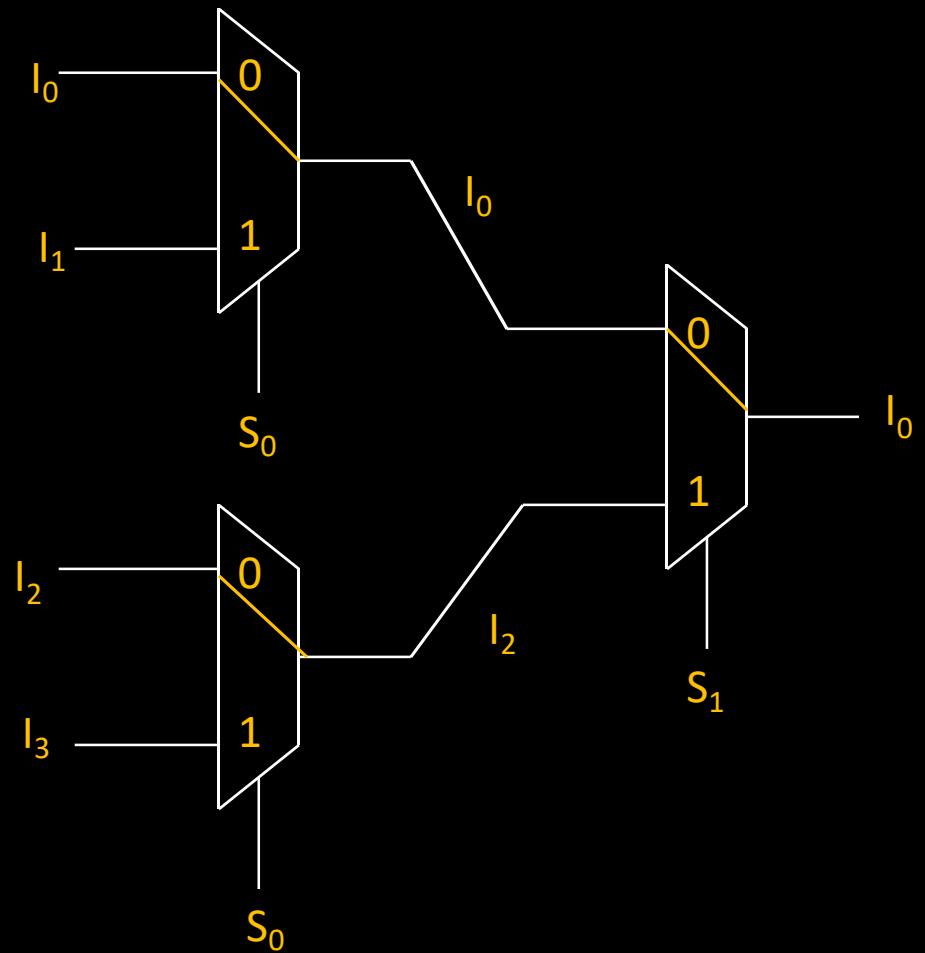
Implement the operation using suitable logic circuit



# Multiplexers

4:1 Multiplexer – 4:1 MUX Using 2:1 MUX

$S_1$	$S_0$	O/P
0	0	$I_0$
0	1	$I_1$
1	0	$I_2$
1	1	$I_3$

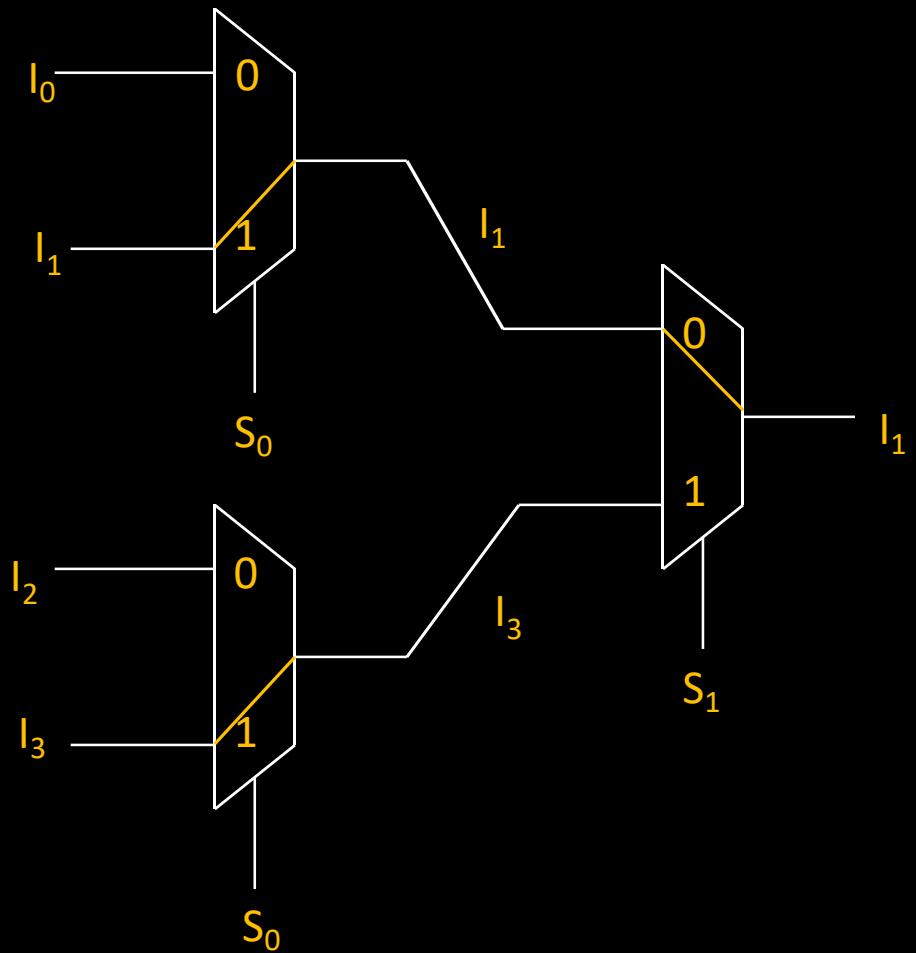




# Multiplexers

4:1 Multiplexer – 4:1 MUX Using 2:1 MUX

$S_1$	$S_0$	O/P
0	0	$I_0$
0	1	$I_1$
1	0	$I_2$
1	1	$I_3$

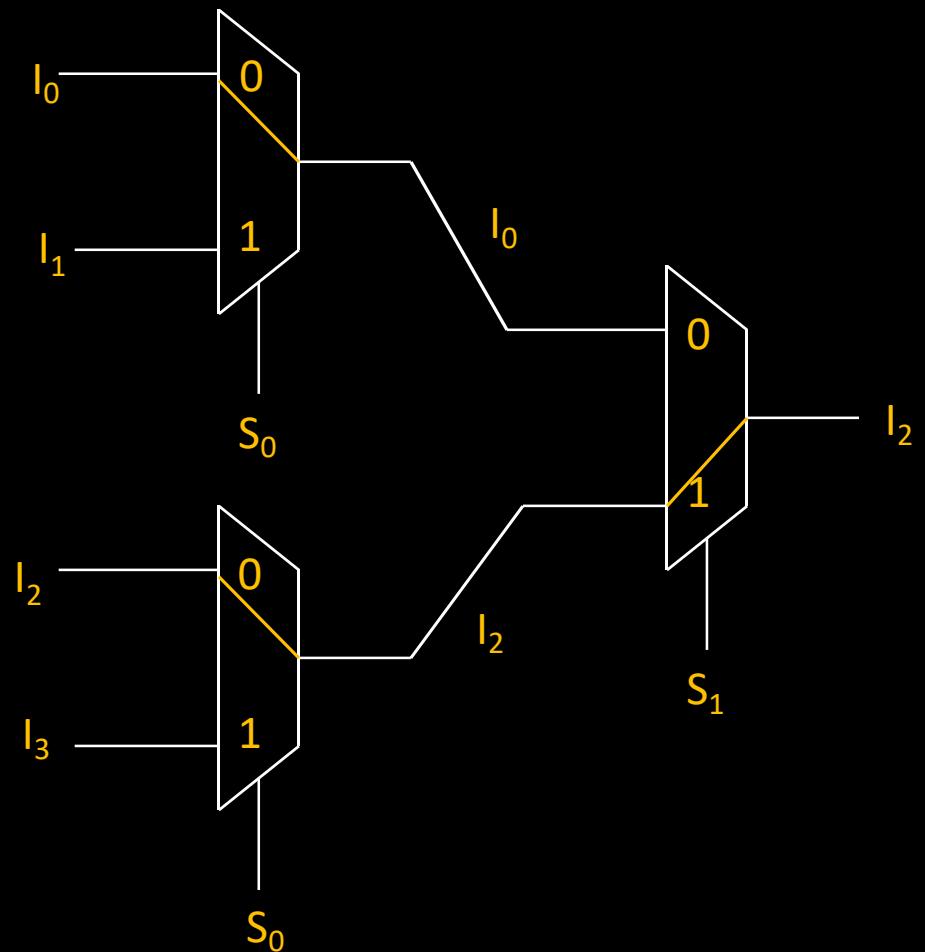




# Multiplexers

4:1 Multiplexer – 4:1 MUX Using 2:1 MUX

$S_1$	$S_0$	O/P
0	0	$I_0$
0	1	$I_1$
1	0	$I_2$
1	1	$I_3$

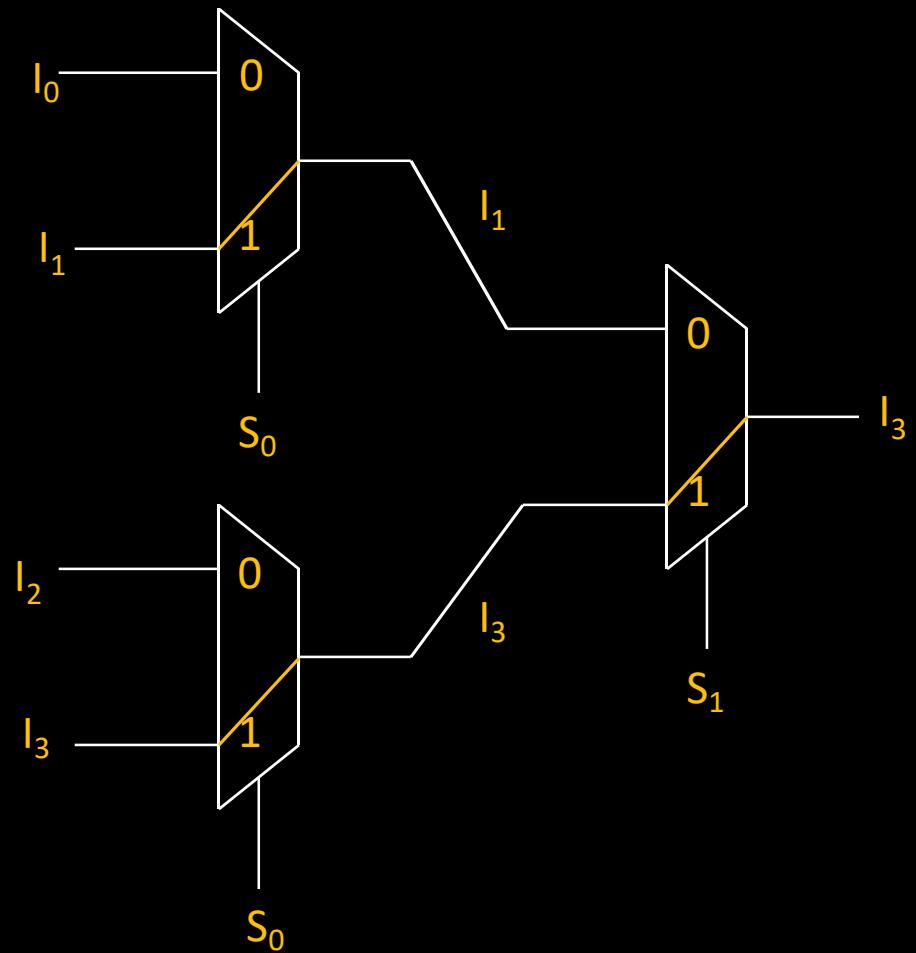




# Multiplexers

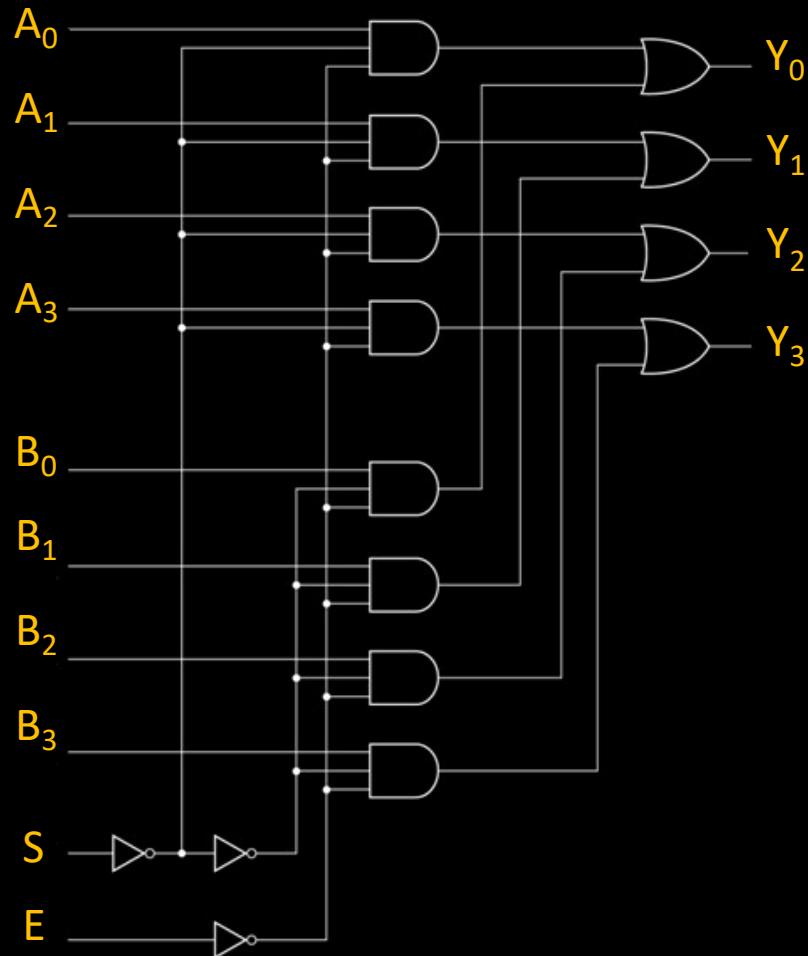
4:1 Multiplexer – 4:1 MUX Using 2:1 MUX

$S_1$	$S_0$	O/P
0	0	$I_0$
0	1	$I_1$
1	0	$I_2$
1	1	$I_3$





# Multiplexers



Multiple Bit Selection Logic

E	S	Output Y
1	0	
1	1	
0	0	
0	1	

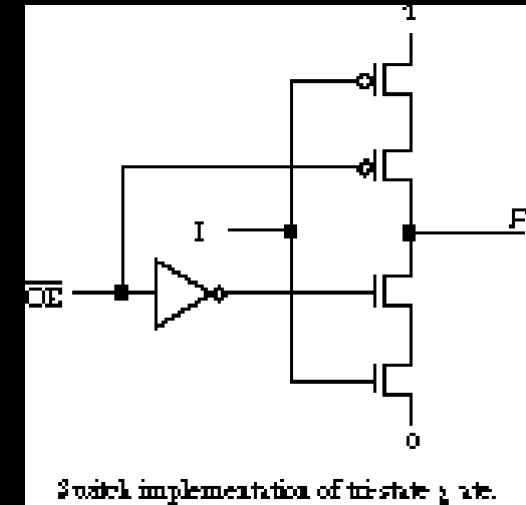
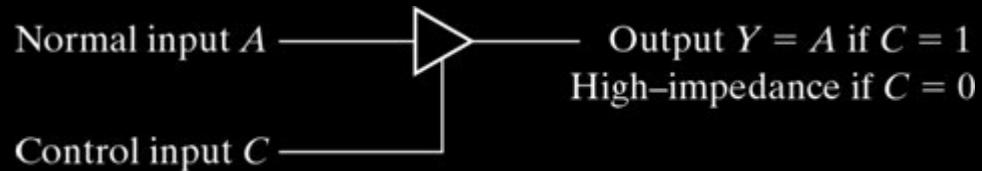
Analyze and Complete the table



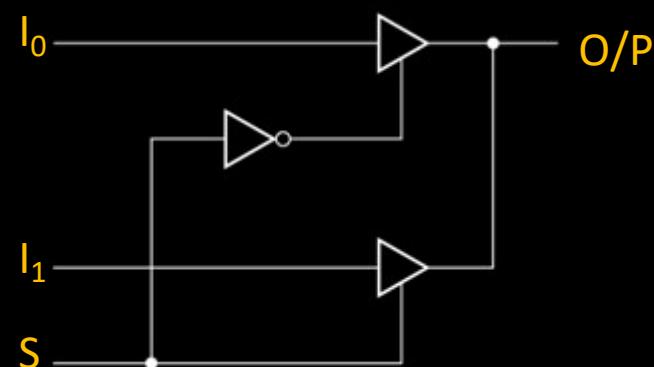
# Multiplexers

Multiplexers can be constructed using 3-state gates

## Tri-State Buffer



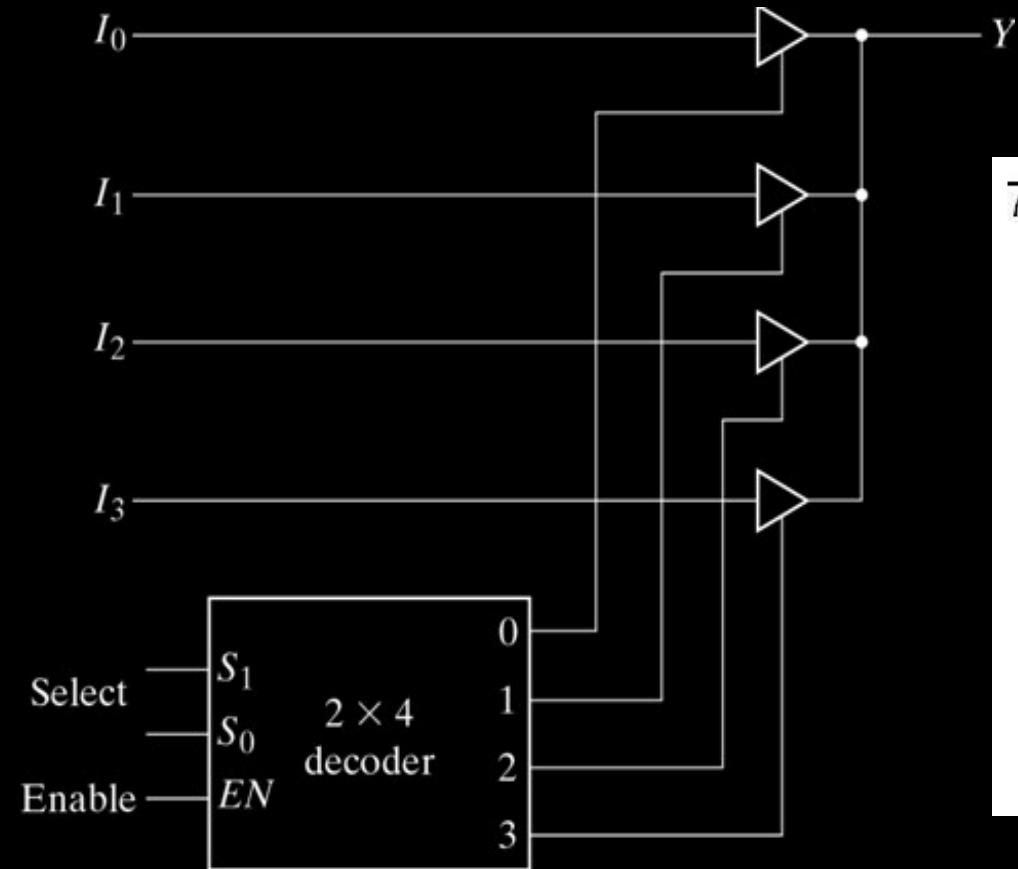
2:1 MUX using Tri-state buffer





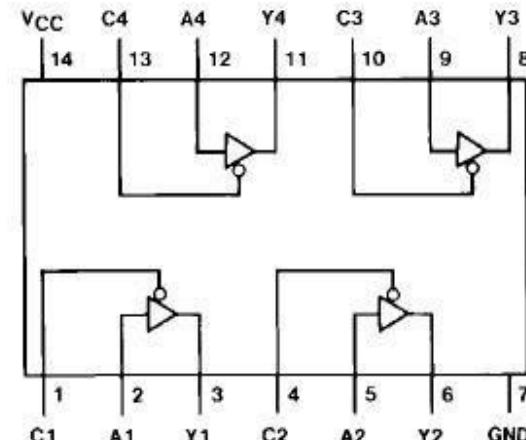
# Multiplexers

4:1 MUX using Tri-state buffers and 2 to 4 Decoder



74HCT125 Quad Tri-State Buffer

LOGIC SYMBOL



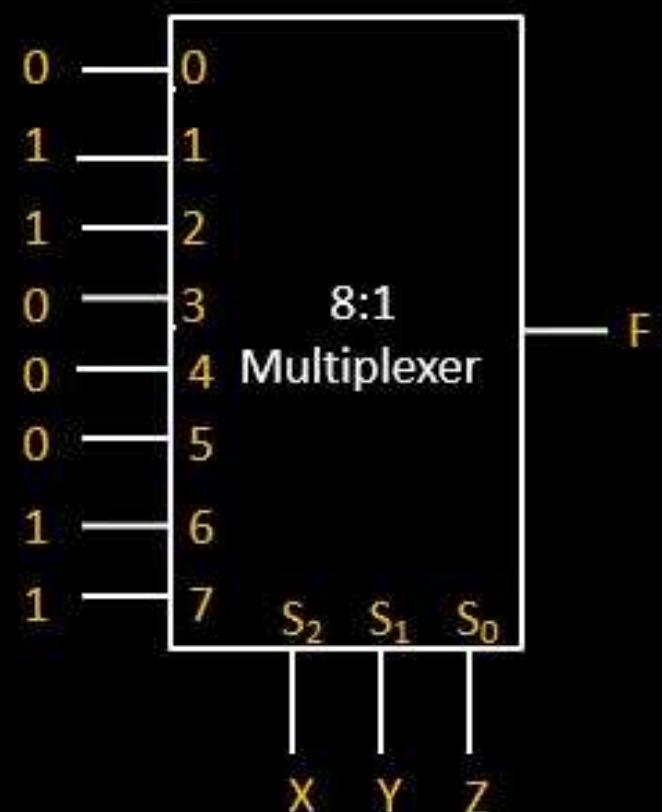


# Multiplexers

Boolean Function implementation

$$F(X,Y,Z) = \sum(1, 2, 6, 7)$$

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





# Multiplexers

Boolean Function implementation

$$F(X,Y,Z) = \sum(1, 2, 6, 7)$$

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

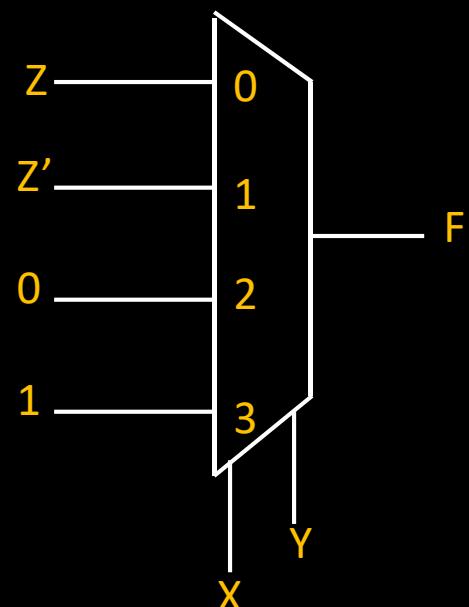
$$F = Z$$

$$F = Z'$$

$$F = 0$$

$$F = 1$$

4:1  
Multiplexer





# Multiplexers

Boolean Function implementation

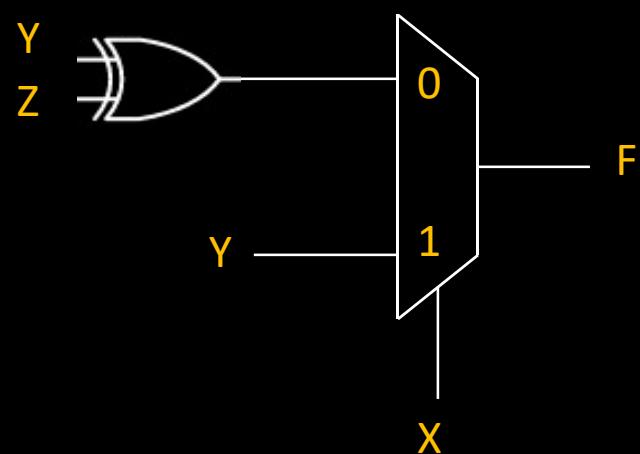
$$F(X,Y,Z) = \sum(1, 2, 6, 7)$$

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

$$F = Y \oplus Z$$

$$F = Y$$

2:1  
Multiplexer

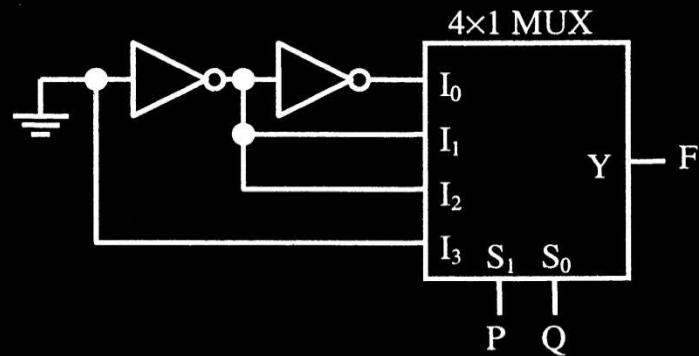




# Multiplexers

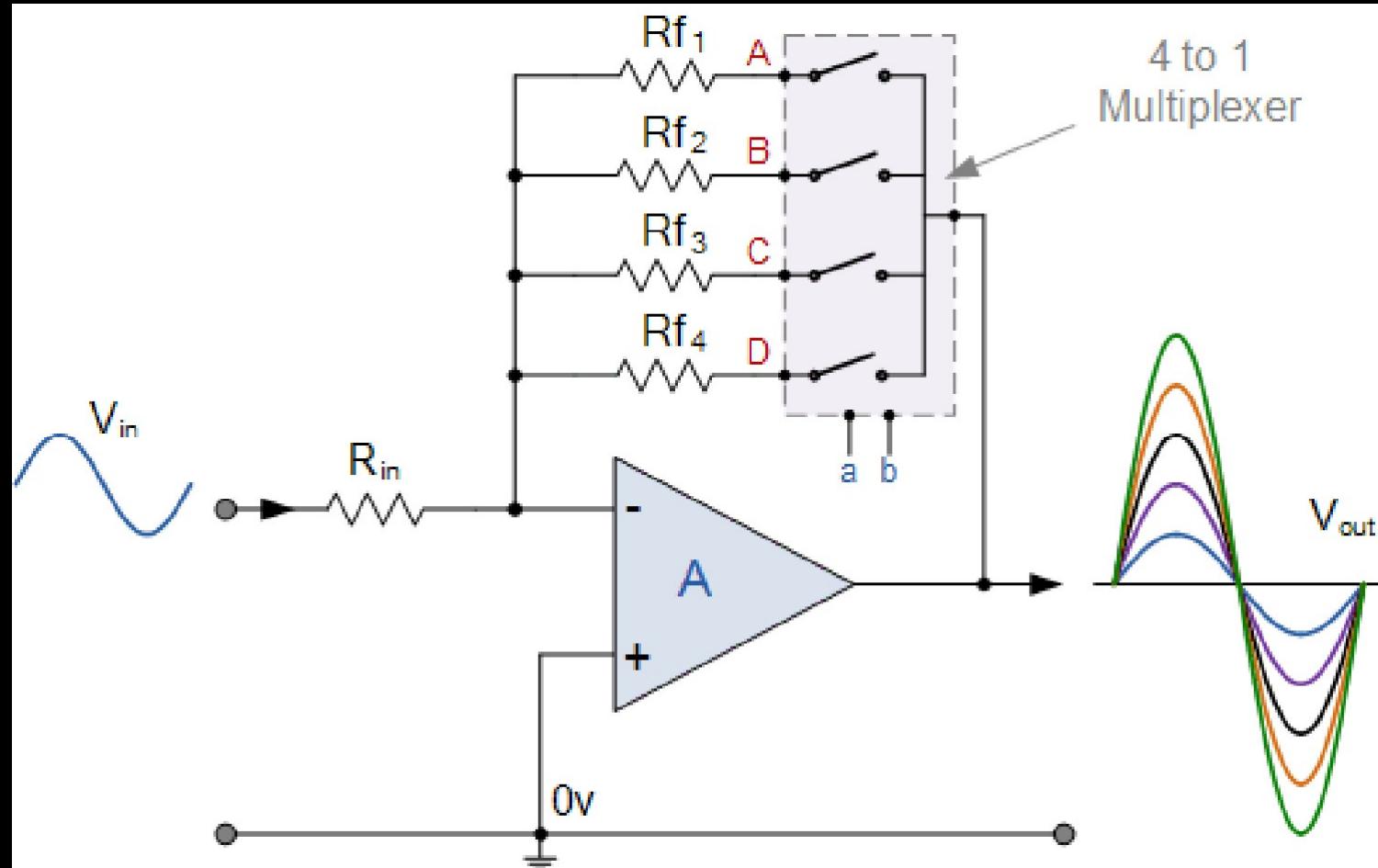
## Boolean Function implementation

1.  $F(X, Y, Z) = \sum (1, 3, 5, 6, 7)$  using 4:1 MUX and 2:1 MUX
2.  $F(A, B, C, D) = \sum (1, 3, 4, 11, 12, 13, 14, 15)$  Using 8:1 MUX and 4:1 MUX
3. The logic function implemented by the circuit below is (Ground implies logic '0')



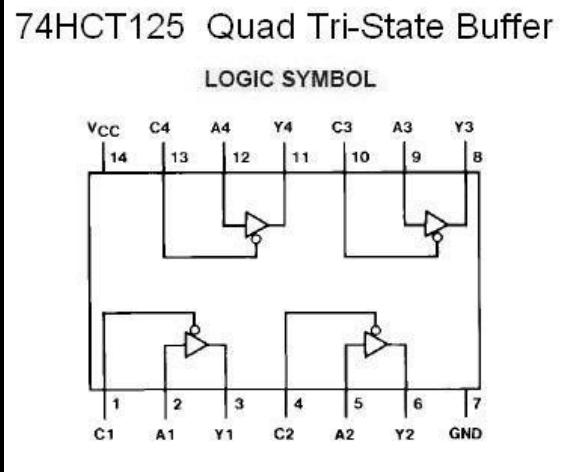


## Multiplexers: Realization using Op-Amp

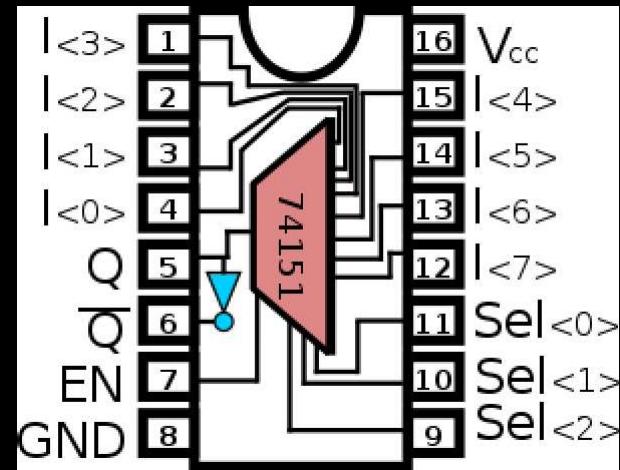




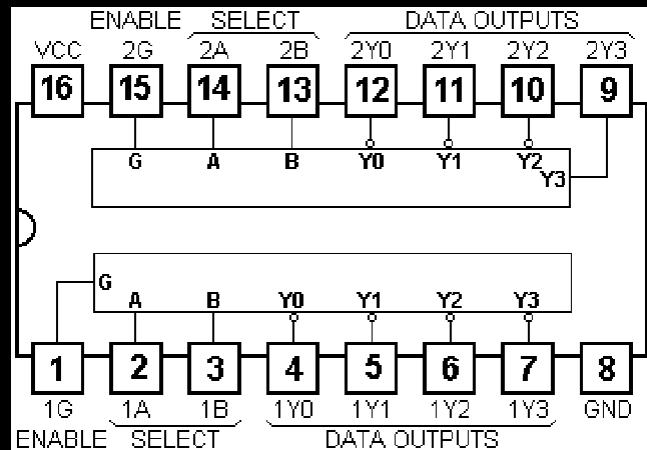
# 74XX series ICs



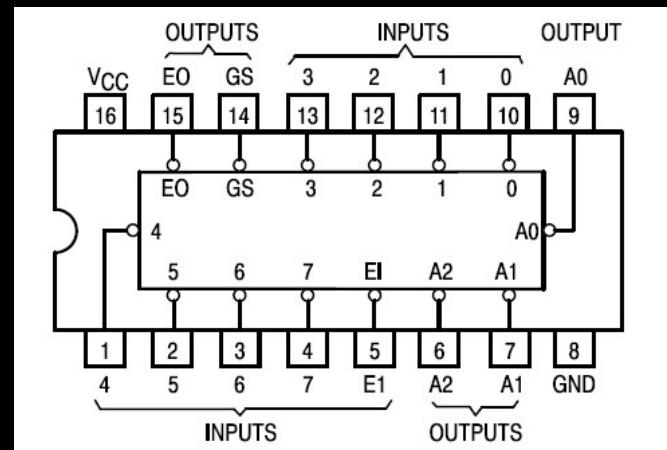
Tri-state Buffer



Mux



Decoder



Encoder



# *Thank You*

# Digital Design

## Lecture 15: Multiplexer and Demultiplexer

Contd....



**Birla Institute of Technology & Science, Pilani**  
Hyderabad Campus

9/23/2021



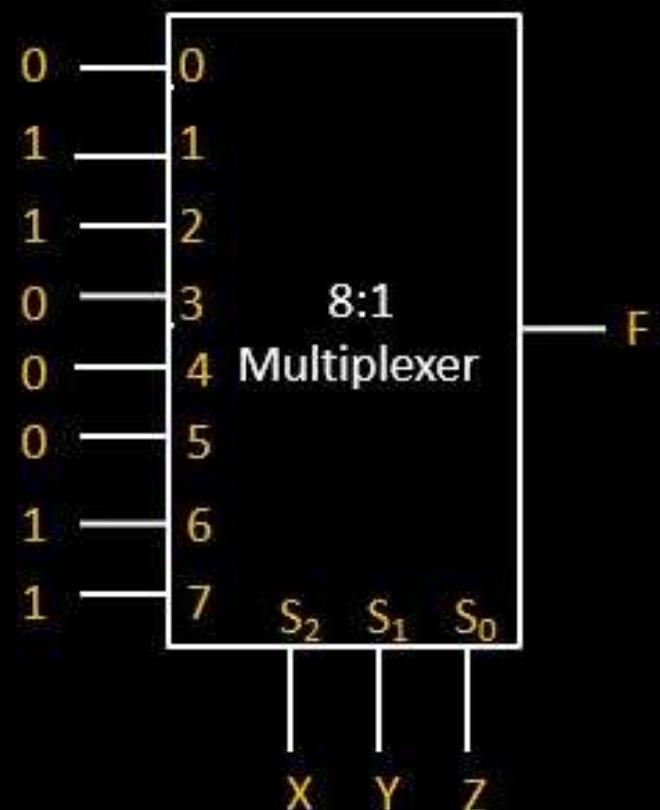


# Multiplexers

Boolean Function implementation using MUX,

$$F(X,Y,Z) = \sum (1, 2, 6, 7)$$

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





# Multiplexers

Boolean Function implementation

$$F(X,Y,Z) = \sum(1, 2, 6, 7)$$

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

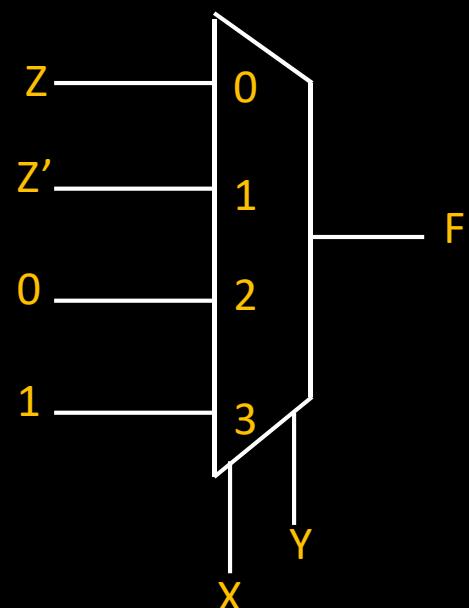
$$F = Z$$

$$F = Z'$$

$$F = 0$$

$$F = 1$$

4:1  
Multiplexer





# Multiplexers

Boolean Function implementation

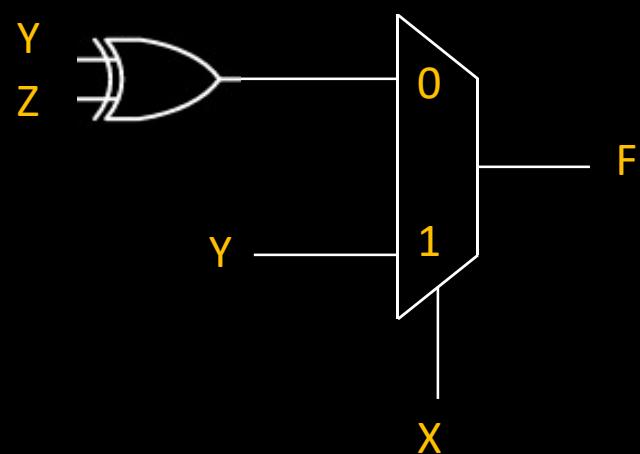
$$F(X,Y,Z) = \sum(1, 2, 6, 7)$$

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

$$F = Y \oplus Z$$

$$F = Y$$

2:1  
Multiplexer



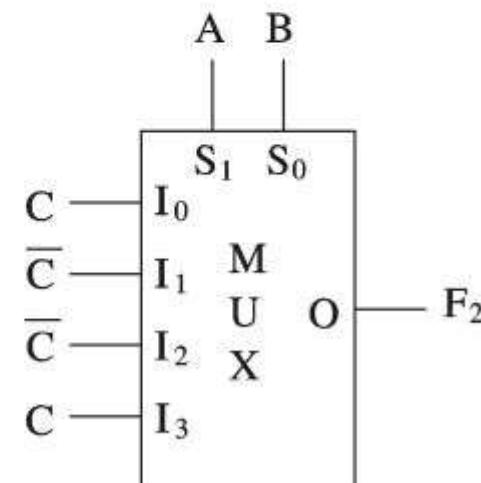
## Efficient implementation: Even-parity function

Original truth table

A	B	C	F <sub>1</sub>
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

New truth table

A	B	F <sub>1</sub>
0	0	C
0	1	$\bar{C}$
1	0	$\bar{C}$
1	1	C

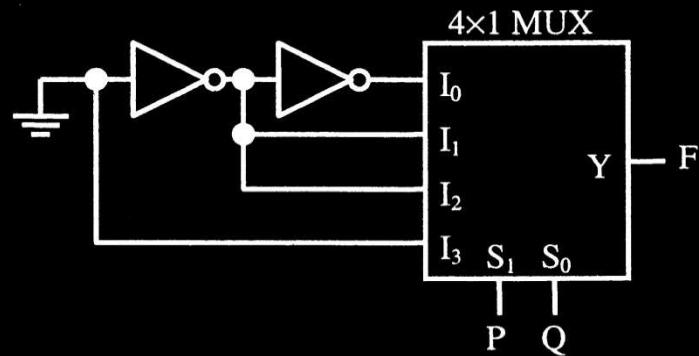




# Multiplexers

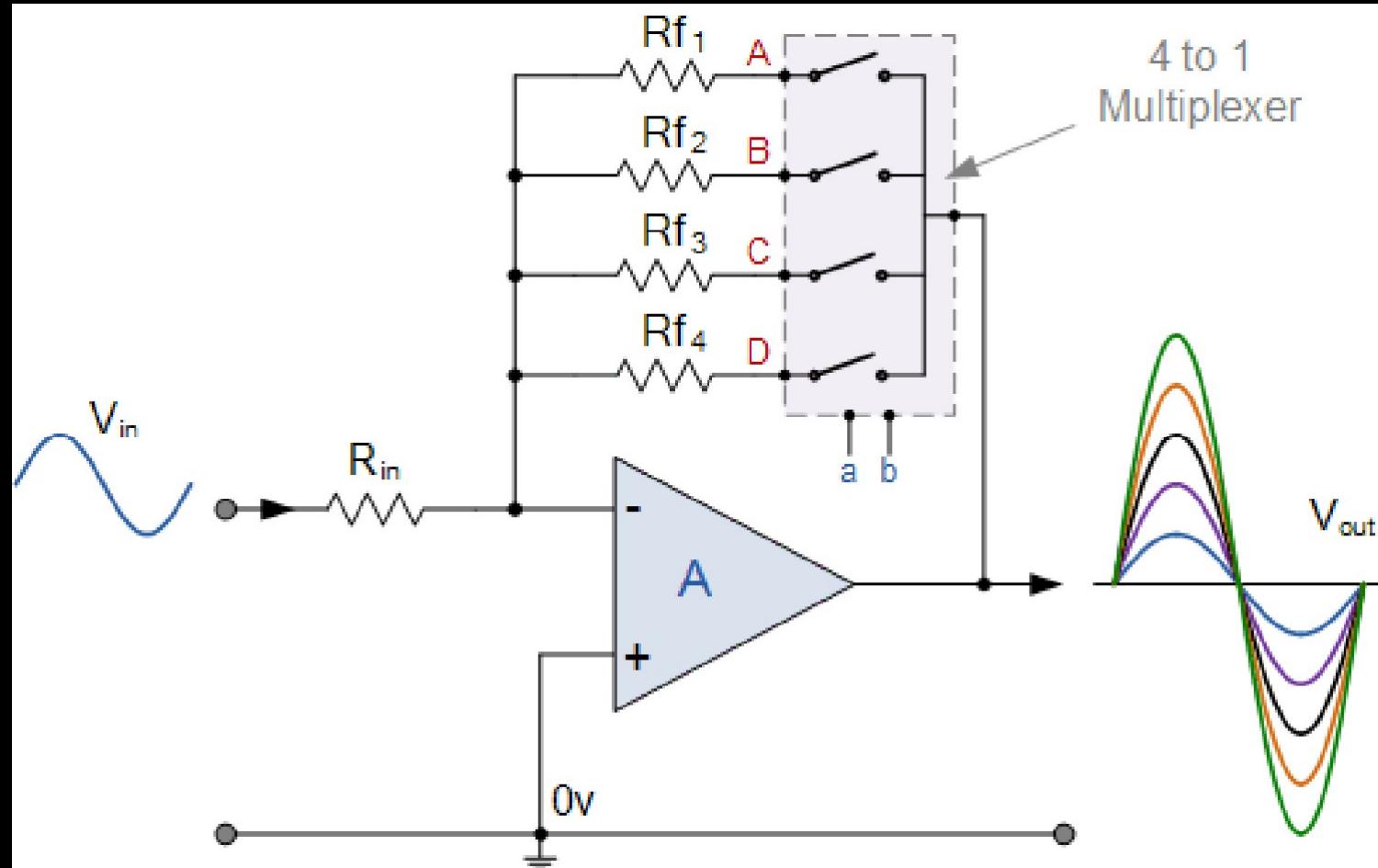
## Boolean Function implementation

1.  $F(X, Y, Z) = \sum (1, 3, 5, 6, 7)$  using 4:1 MUX and 2:1 MUX
2.  $F(A, B, C, D) = \sum (1, 3, 4, 11, 12, 13, 14, 15)$  Using 8:1 MUX and 4:1 MUX
3. The logic function implemented by the circuit below is (Ground implies logic '0')



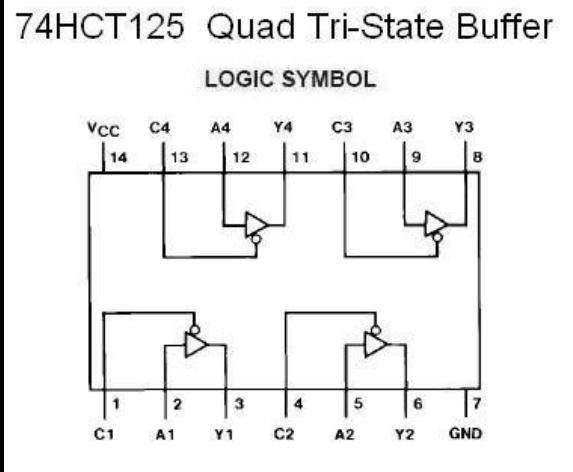


## Multiplexers: Realization using Op-Amp

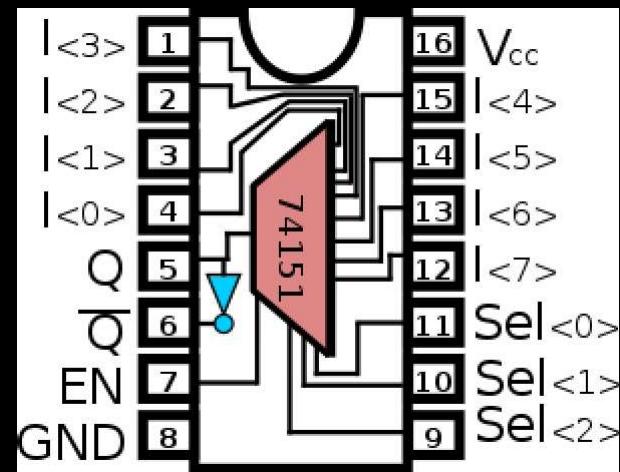




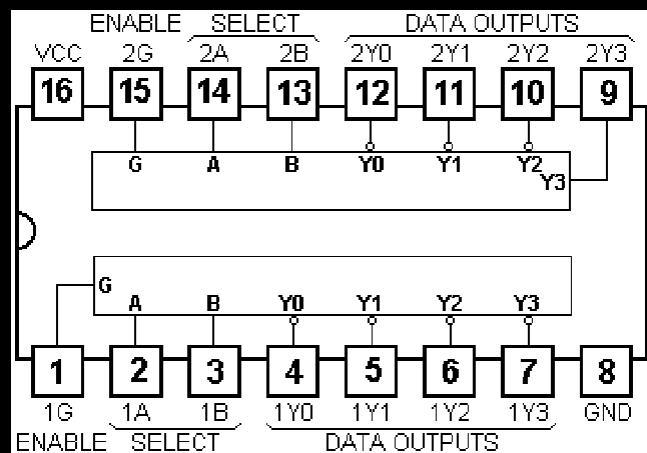
# 74XX series ICs



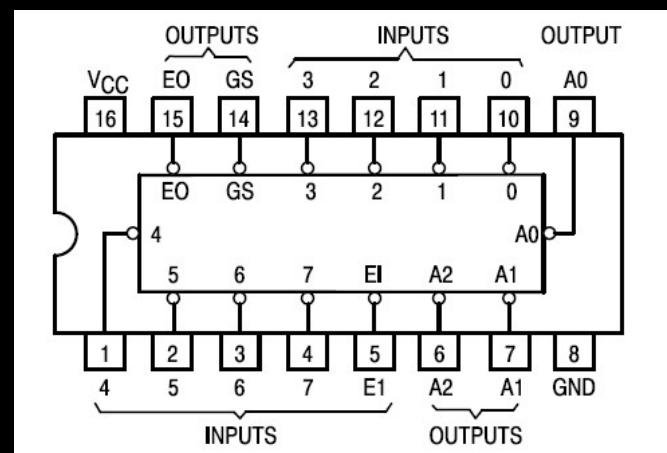
Tri-state Buffer



Mux



Decoder



Encoder



# *Thank You*

# **Digital Design**

## **Lecture No. 16**

### **MULTIPLICATION ALGORITHMS**

# Multiplication

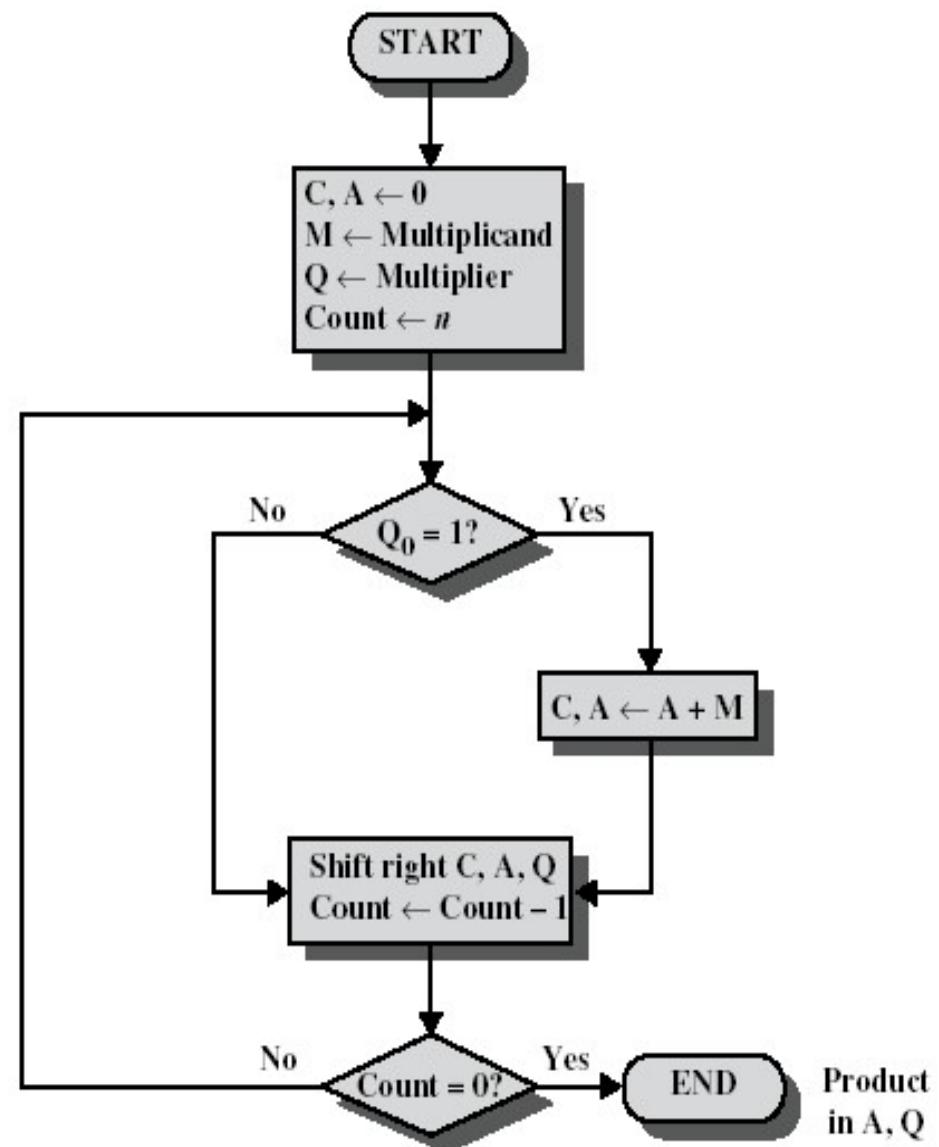
## Some general observations

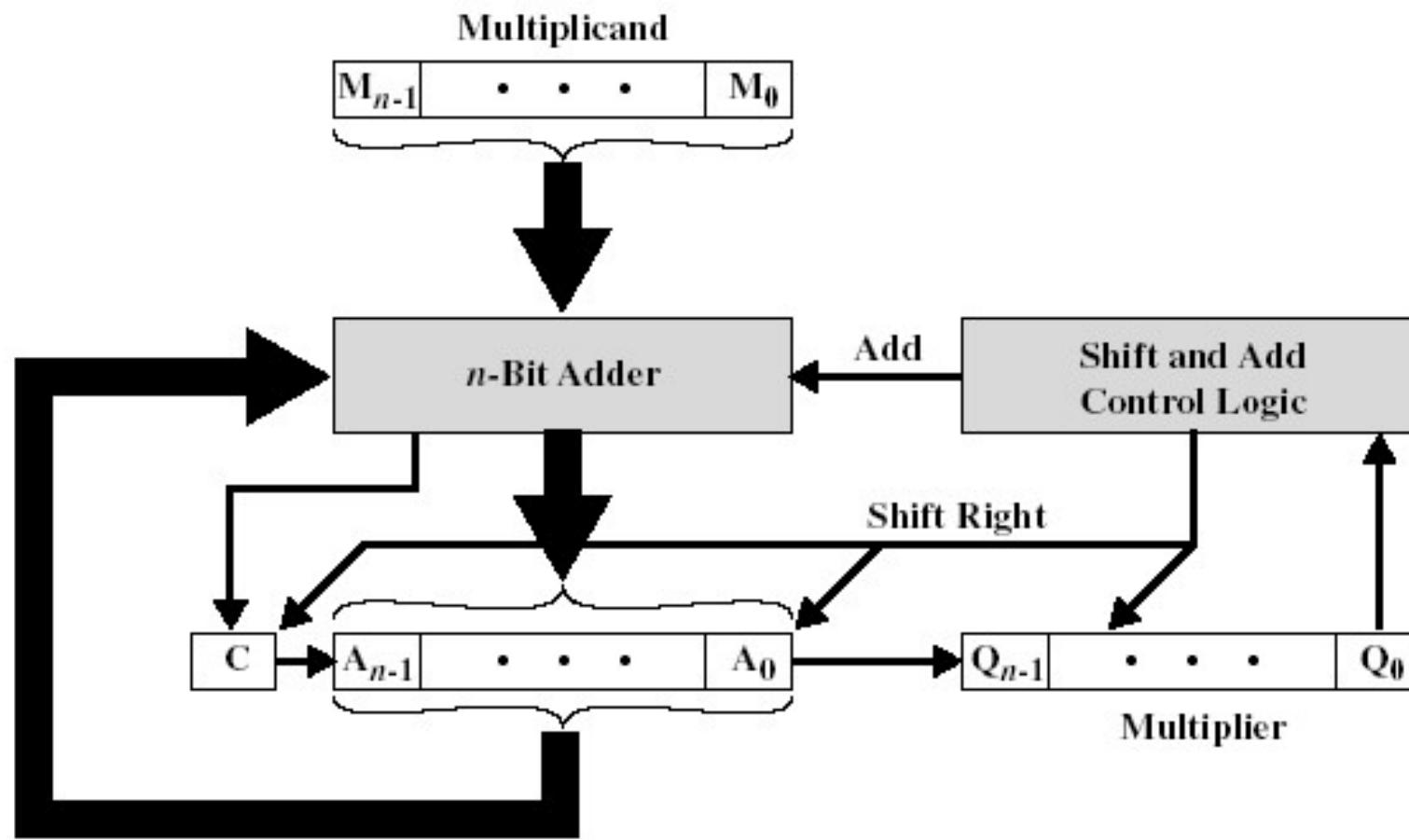
- Multiplication involves the generation of partial products – one for each digit in the multiplier.
- Partial products are summed to produce the final product.
- Partial products are simple to define for binary multiplication. If the digit in multiplier is ‘one’ the partial product is the multiplicand, otherwise the partial product is zero.
- The total product is the sum of the partial products. Each successive partial product is shifted one position to the left.
- The multiplication of two n-bit binary numbers results in a product of up to  $2n$  bits in length.

$$\begin{array}{r} 1011 \\ \underline{1101} \\ 1011 \\ 0000 \\ 1011 \\ \underline{1011} \\ 10001111 \end{array}$$

# Multiplication

1. The Digital system can keep a running product rather than summing at the end.
2. For each ‘1’ in the multiplier we can apply an ‘add’ and a ‘shift’.
3. For each ‘0’ only a shift is needed.





1. Multiplier and multiplicand are loaded into registers  $Q$  and  $M$ .
2. A third register ( $A$ ) is initially set to zero.
3. A one-bit  $C$  register (initialised to zero) holds carry bits.

C	A	Q	M	
0	0000	1101	1011	Initial Values

C	A	Q	M	
0	0000	1101	1011	Initial Values
0	1011	1101	1011	Add } First
0	0101	1110	1011	Shift } Cycle

C	A	Q	M		
0	0000	1101	1011	Initial	Values
0	1011	1101	1011	Add	First
0	0101	1110	1011	Shift	Cycle
0	0010	1111	1011	Shift	Second Cycle

C	A	Q	M			
0	0000	1101	1011	Initial	Values	
0	1011	1101	1011	Add		First
0	0101	1110	1011	Shift	}	Cycle
0	0010	1111	1011	Shift	}	Second
0	1101	1111	1011	Add		Third
0	0110	1111	1011	Shift	}	Cycle
1	0001	1111	1011	Add		Fourth
0	1000	1111	1011	Shift	}	Cycle

**This Approach will not work if numbers  
are signed**

**ALTERNATIVE**

**BOOTH'S MULTIPLIER**

Multiplicand ‘M’ unchanged

The multiplier ‘Q’ to a re-coded value R

Each digit can assume a negative as well as positive and zero values

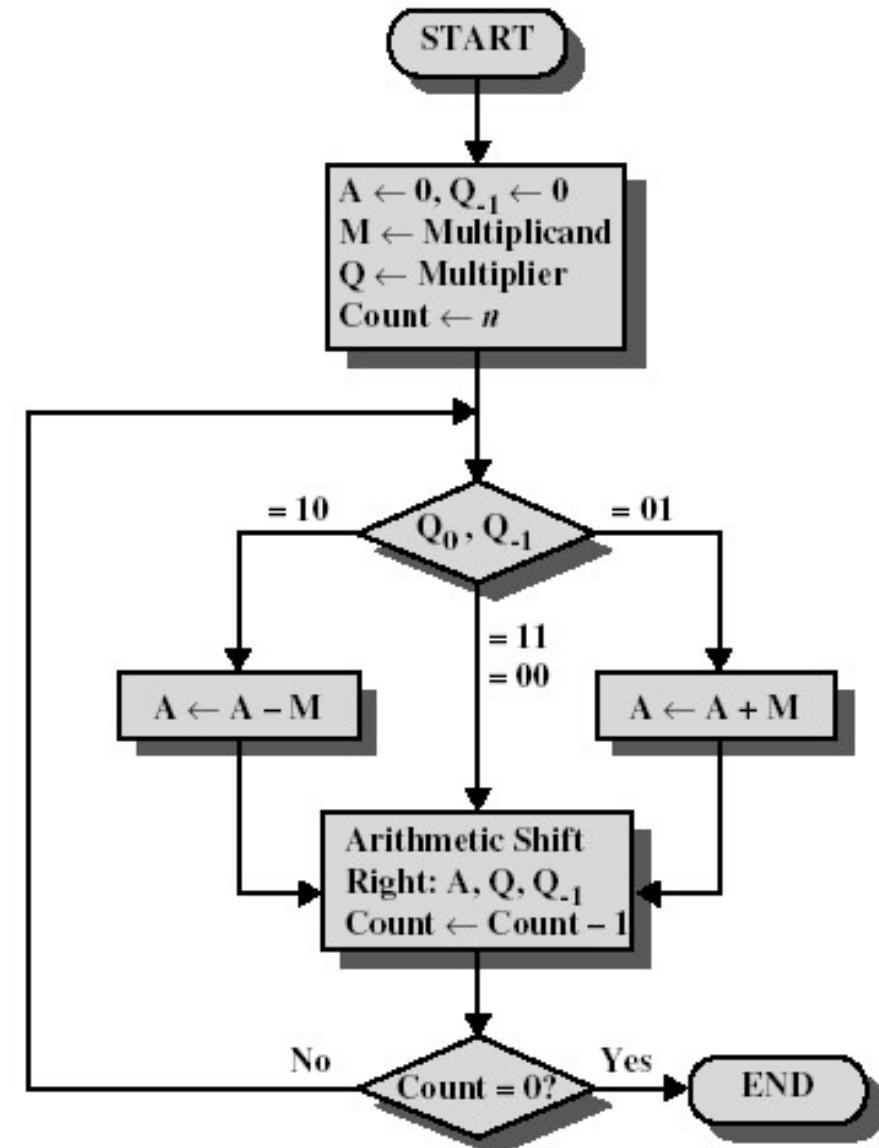
## **Signed Digit ( SD) encoding**

- Booth's algorithm called skipping over one's
- String of 1's replaced by 0's
- **For example:**  $30 = 0011110$
- $= 32 - 2 = 0100000 - 0000010$
- **In the coded form = 01000 $\bar{1}$ 0**

- Booth re-coding Procedure working from LSB to MSB  
retain each 0 until a 1 is reached
- When a 1 is encountered insert  $\bar{1}$  at that position and complement all the succeeding 1's until a 0 is encountered
- Replace that 0 with 1 and continue same
- while multiplying with  $\bar{1}$ ; 2's compliment addition is used

## Booth's Algorithm

1. Multiplier and multiplicand are placed in Q and M registers.
2. A 1-bit register is placed to the right of the least significant bit ( $Q_0$ ) and designated  $Q_{-1}$ .
3. Control logic scans the bits of the multiplier one at a time; but a bit and its bit to the right are examined. If the bits are the same (1-1 or 0-0) then all bits of the A, Q, and  $Q_{-1}$  registers are shifted to the right 1 bit. If the two bits differ, then the multiplicand is added/subtracted depending on whether the bits are 0-1 or 1-0. Addition is followed by a **right arithmetic shift**.



A	Q	Q <sub>-1</sub>	M	
0000	0011	0	0111	Initial Values

A	Q	$Q_{-1}$	M	
0000	0011	0	0111	Initial Values
1001	0011	0	0111	$A \leftarrow A - M$

A	Q	$Q_{-1}$	M	
0000	0011	0	0111	Initial Values
1001	0011	0	0111	$A \leftarrow A - M$ } First
1100	1001	1	0111	Shift } Cycle

A	Q	$Q_{-1}$	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	$A \leftarrow A - M$	} First
1100	1001	1	0111	Shift	} Cycle
1110	0100	1	0111	Shift	} Second Cycle

A	Q	$Q_{-1}$	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	$A \leftarrow A - M$	} First
1100	1001	1	0111	Shift	} Cycle
1110	0100	1	0111	Shift	} Second
0101	0100	1	0111	$A \leftarrow A + M$	

A	Q	$Q_{-1}$	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	$A \leftarrow A - M$	} First Cycle
1100	1001	1	0111	Shift	} Cycle
1110	0100	1	0111	Shift	} Second Cycle
0101	0100	1	0111	$A \leftarrow A + M$	} Third Cycle
0010	1010	0	0111	Shift	} Cycle

A	Q	$Q_{-1}$	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	$A \leftarrow A - M$	} First Cycle
1100	1001	1	0111	Shift	} Second Cycle
1110	0100	1	0111	Shift	} Third Cycle
0101	0100	1	0111	$A \leftarrow A + M$	} Fourth Cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	

Verify the operation of

$$(+7) \times (-3)$$

Multiplicand M = 0111

Multiplier Q = 1101

**A**      **Q**      **Q<sub>-1</sub>**      **M = 0111**

<b>0000</b>	<b>1101</b>	<b>0</b>	<b>initial</b>
<b>1001</b>	<b>1101</b>	<b>0</b>	<b>A ← A - M</b>
<b>1100</b>	<b>1110</b>	<b>1</b>	<b>shift</b>
<b>0011</b>	<b>1110</b>	<b>1</b>	<b>A ← A + M</b>
<b>0001</b>	<b>1111</b>	<b>0</b>	<b>shift</b>
<b>1010</b>	<b>1111</b>	<b>0</b>	<b>A ← A - M</b>
<b>1101</b>	<b>0111</b>	<b>1</b>	<b>shift</b>
<b>1110</b>	<b>1011</b>	<b>1</b>	<b>shift</b>

*Thank you*

# Digital Design

## Lecture 17: Programmable Logic Devices



**Birla Institute of Technology & Science, Pilani**  
Hyderabad Campus

9/30/2021

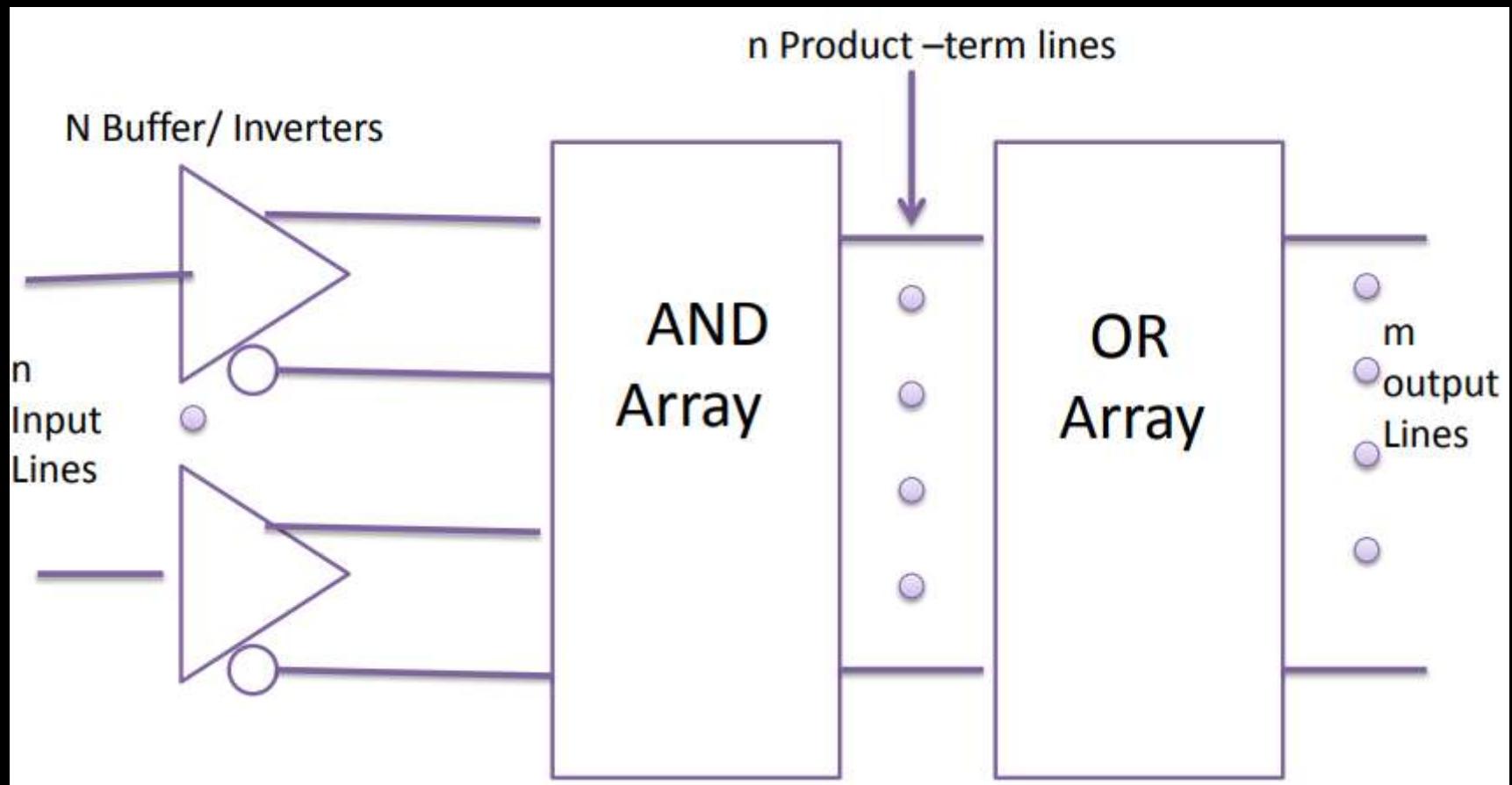
Pilani,  
Hyderabad



# About PLDs

- Large circuit is designed on a single chip
- Can be programmed by the users as per their requirement
- Possible to implement a combinational or sequential circuit using the PLD
- Consist of array of AND, OR, NOT gates
- Any Boolean function can be represented by sum of the product (SOP) form.

# PLD basic Architecture





# Types of PLD

Programmable Read Only Memory (PROM)

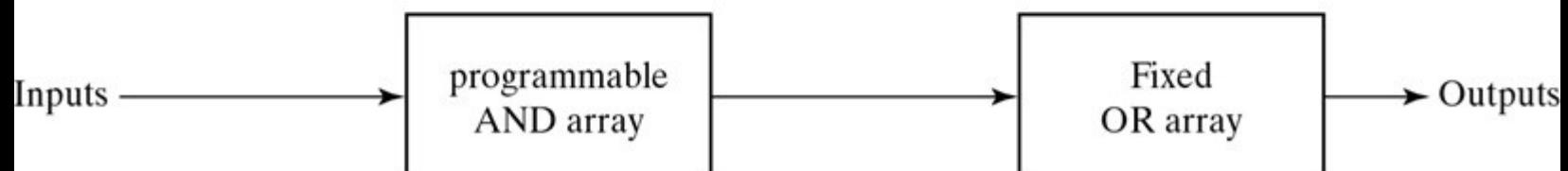
Programmable Logic Array (PLA)

Programmable Array Logic (PAL)

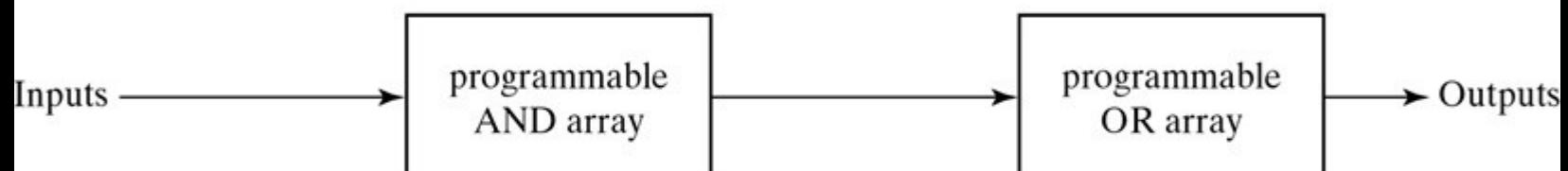
Simple Programmable Logic Device (SPLD)

Complex Programmable Logic Device (CPLD)

Field Programmable Gate Arrays (**FPGAs**)



(b) Programmable array logic (PAL)

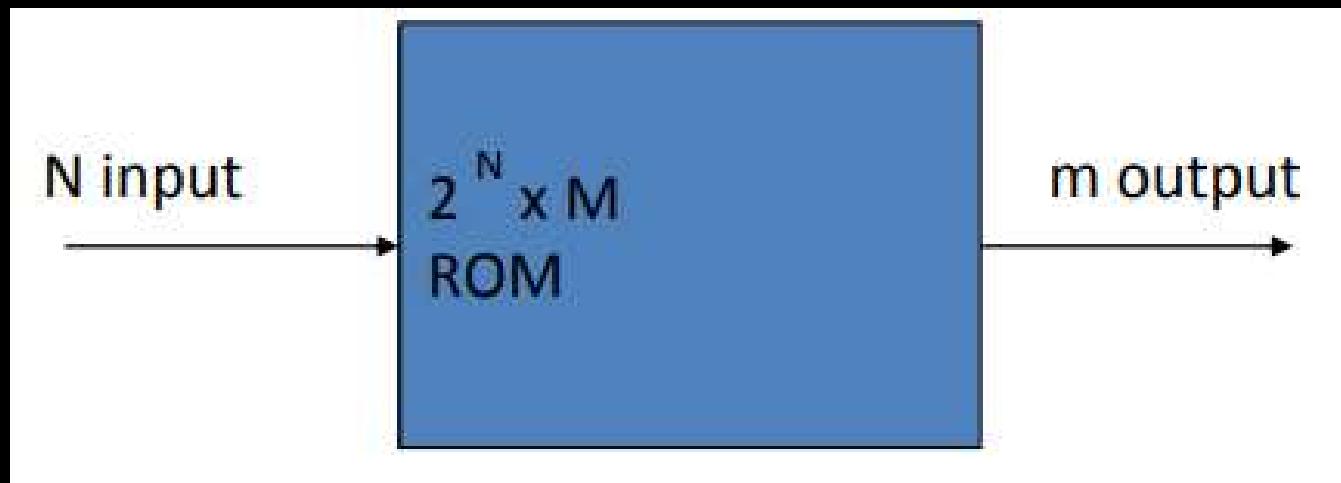


(c) Programmable logic array (PLA)

Basic Configuration of Three PLDs

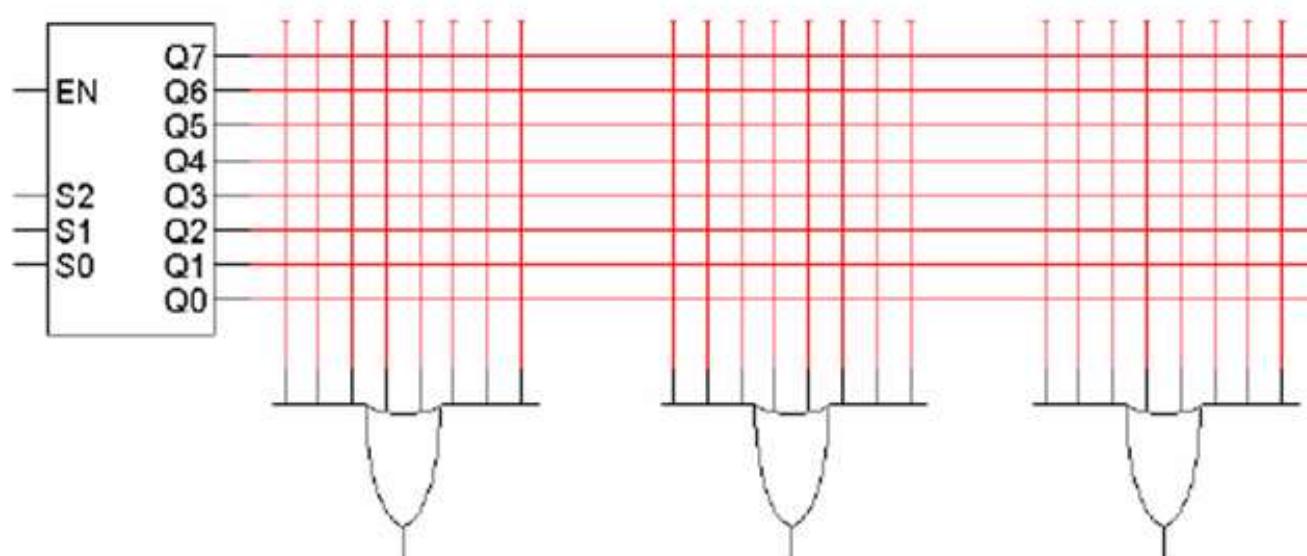
# ROM basics

- A Memory Device in which permanent binary Information is stored
- Used to implement combinational circuits or store binary information
- ROM includes both decoders and OR gates within a single IC



- Address: m bits; data: n bits
- ROM contains  $2^N$  word of M bit each
- The input bits decide the particular word that becomes available on output lines

- Building circuits with decoders is so easy that **programmable logic devices** are often based around decoders.
- The diagram below shows a blank **read-only memory**, or **ROM**, device.
  - It's just a decoder whose outputs may be sent to several OR gates.
  - The connections between the decoder outputs and the OR gate inputs are "programmable," so different functions can be implemented.
- To program a ROM for some specific functions, you just have to make the right connections.



## Boolean Function Implementation using ROM

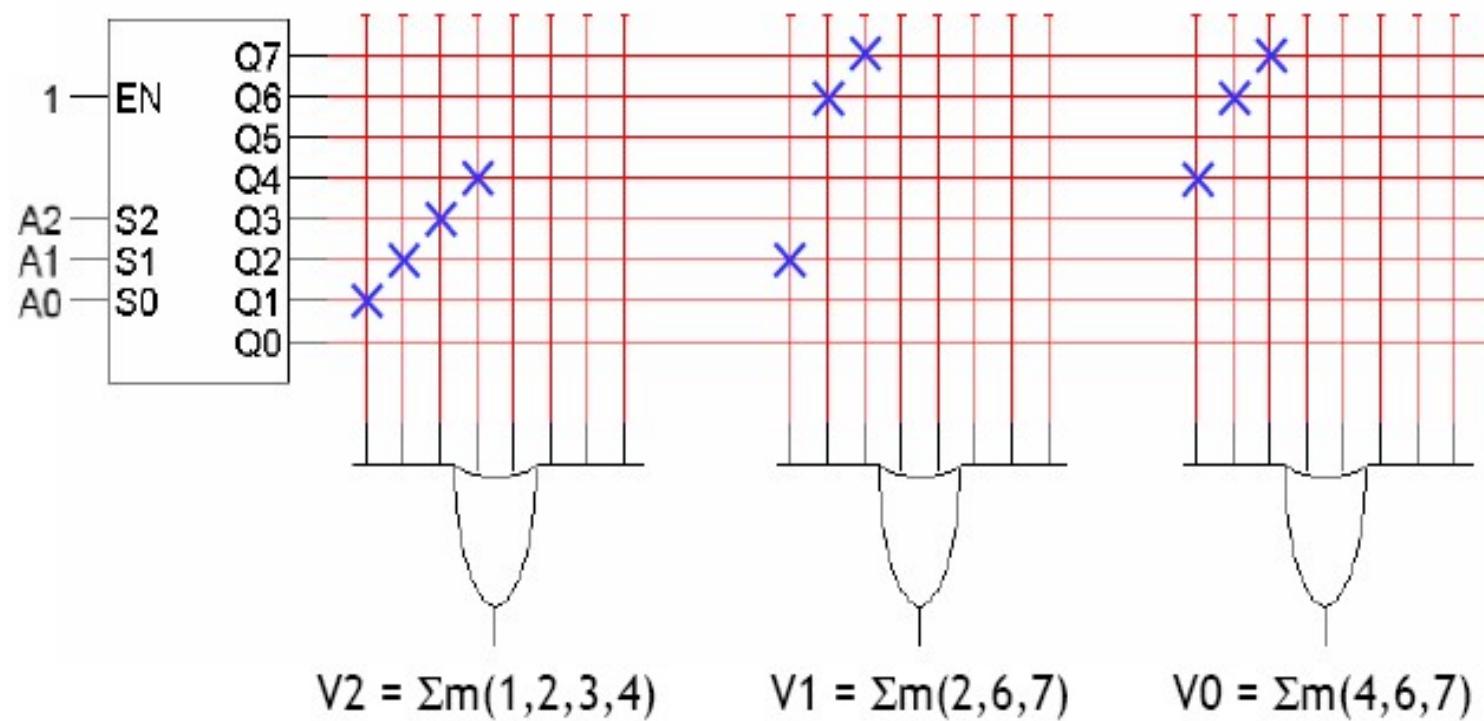
$$V_0 = \sum (4, 6, 7)$$

$$V_1 = \sum (2, 6, 7)$$

$$V_2 = \sum (1, 2, 3, 4)$$

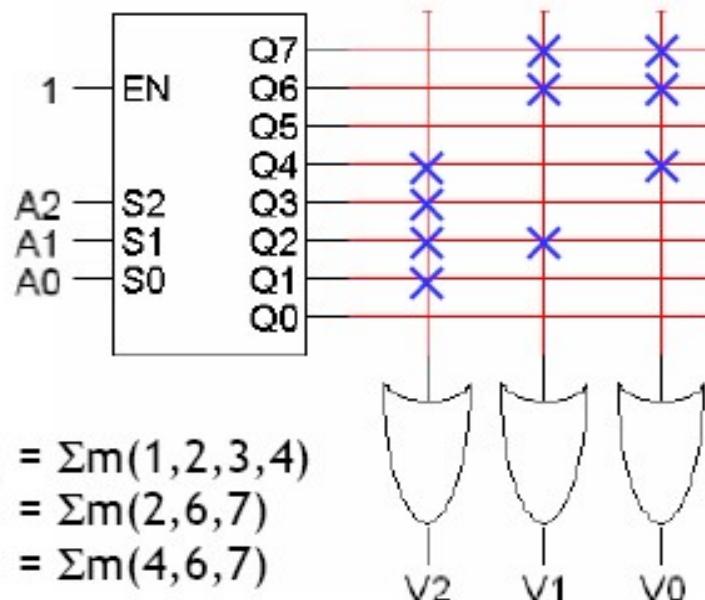
## ROM example

- Here are three functions V2, V1 and V0, implemented with our ROM.
- Blue crosses ( $\times$ ) indicate connections between decoder outputs and OR gates. Empty intersections are not connected.
- This is an  $8 \times 3$  ROM since there are 8 decoder outputs and 3 OR gates.



## The same example again

- Here is an alternative presentation of the same  $8 \times 3$  ROM, but with some simplified OR gates just to make the diagram neater.



- ROM implementation is not efficient for inputs with many don't care conditions
- It uses Decoders and generates all possible 'minterms', thus no circuit minimization is done



# Thank You

# Digital Design

## Lecture 18: Programmable Logic Devices Contd..



**Birla Institute of Technology & Science, Pilani**  
Hyderabad Campus





# Types of PLD

Programmable Read Only Memory (PROM)

Programmable Logic Array (PLA)

Programmable Array Logic (PAL)

Field Programmable Gate Arrays (**FPGAs**)



# PLA

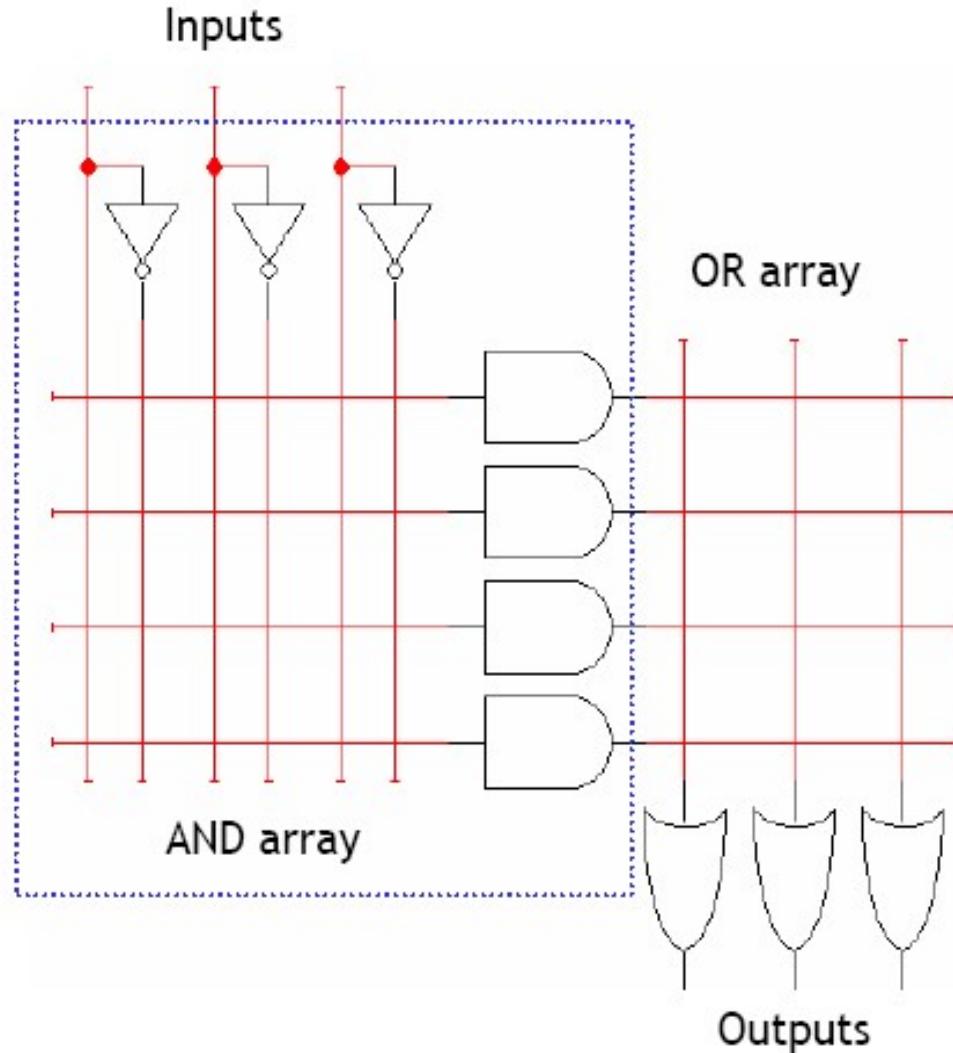
**It's a programmable device capable of implementing functions expressed in SOP.**

- Consists of input buffers and inverters followed by:
- Programmable AND plane, followed by
- Programmable OR plane.
- Can implement ' $m$ ' logic functions of ' $n$ ' variables.



## A blank $3 \times 4 \times 3$ PLA

- This is a  $3 \times 4 \times 3$  PLA (3 inputs, up to 4 product terms, and 3 outputs), ready to be programmed.
- The left part of this diagram replaces the decoder used in a ROM.
- Connections are made within the **AND array** to produce four arbitrary products.
- Those products are then summed in the **OR array**.





## Regular K-map minimization

- The normal K-map approach is to minimize the number of product terms for each *individual* function.
- For our three sample functions, this would result in a total of six different product terms.

$$V_2 = \Sigma m(1,2,3,4)$$

$$V_1 = \Sigma m(2,6,7)$$

$$V_0 = \Sigma m(4,6,7)$$

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

$$V_2 = xy'z' + x'z + x'y$$

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

$$V_1 = yz' + xy$$

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

$$V_0 = xz' + xy$$



## PLA minimization

- But for a PLA, what we really want is to minimize the number of product terms in *all* of the functions.
- We could express V2, V1 and V0 with just *four* total products instead.

$$V_2 = \Sigma m(1, 2, 3, 4)$$

$$V_1 = \Sigma m(2, 6, 7)$$

$$V_0 = \Sigma m(4, 6, 7)$$

		Y	
	0	1	1
X	1	0	0
Z			

$$V_2 = xy'z' + x'z + x'yz'$$

		Y	
	0	0	1
X	0	0	1
Z			

$$V_1 = x'yz' + xy$$

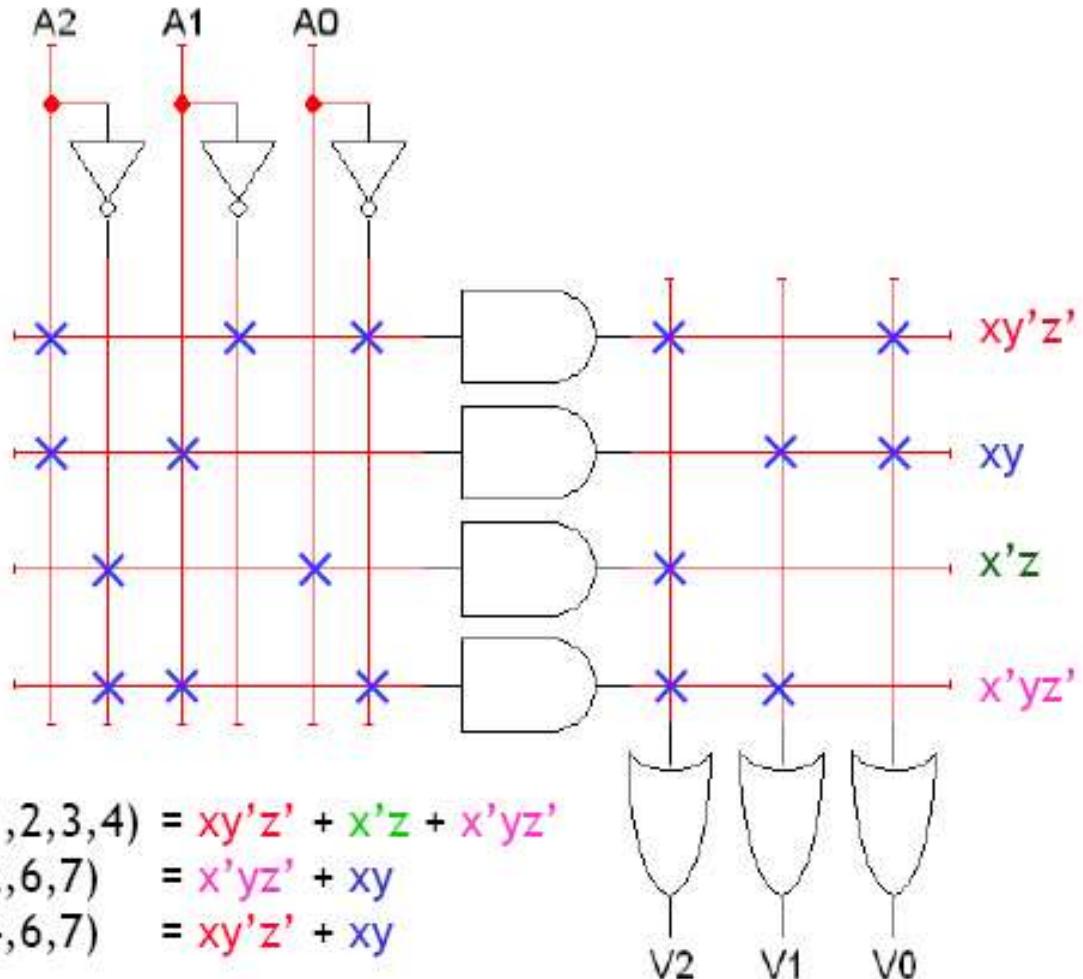
		Y	
	0	0	0
X	1	0	1
Z			

$$V_0 = xy'z' + xy$$



## PLA example

- So we can implement these three functions using a  $3 \times 4 \times 3$  PLA.



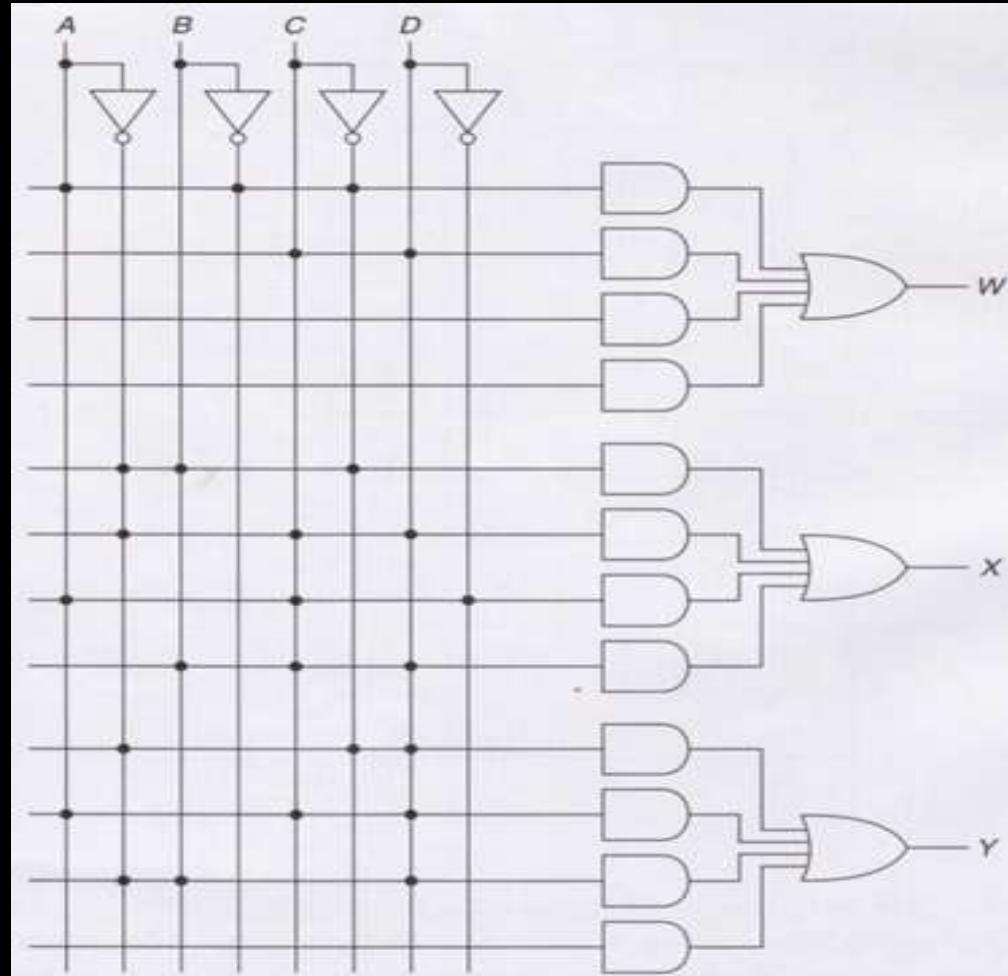


# PAL

- Similar to a PLA, but only has a programmable AND plane.
  - The OR plane is fixed.
- Not as flexible as a PLA since only certain AND gates feed each OR gate, but has fewer things that need programming.



## PAL-based circuit implementation



$$W = AB''C' + CD$$

$$X = A'BC'' + A'CD + ACD' + BCD$$

$$Y = A'C'D + ACD + A'BD$$



## Example:

Implement the combinational circuit described by the following equations, using a PAL with 4 inputs, 4 outputs, and 3-wide AND-OR structure.

$$W(A,B,C,D) = \sum m(2,12,13)$$

$$X(A,B,C,D) = \sum m(7,8,9,10,11,12,13,14,15)$$

$$Y(A,B,C,D) = \sum m(0,2,3,4,5,6,7,8,10,11,15)$$

$$Z(A,B,C,D) = \sum m(1,2,8,12,13)$$



## Solution:

Use function simplification techniques to derive:

$$W = ABC' + A'B'CD'$$

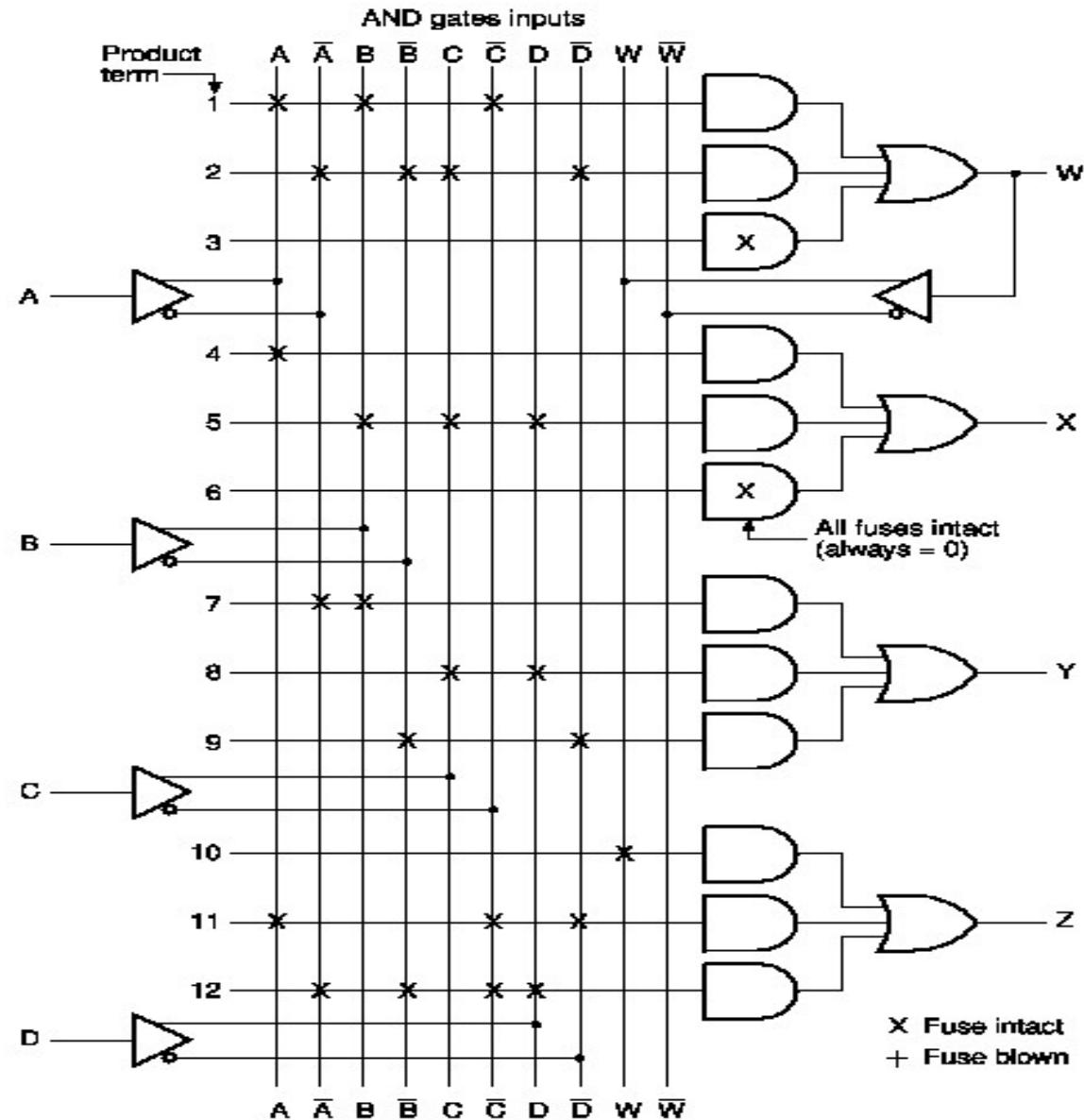
$$X = A + BCD$$

$$Y = A'B + CD + B'D'$$

$$\begin{aligned} Z &= ABC' + A'B'CD' + AC'D' + A'B'C'D \\ &= W + AC'D' + A'B'C'D \end{aligned}$$



**Solution  
Contd..:**





# Tabular Form Specification of interconnection

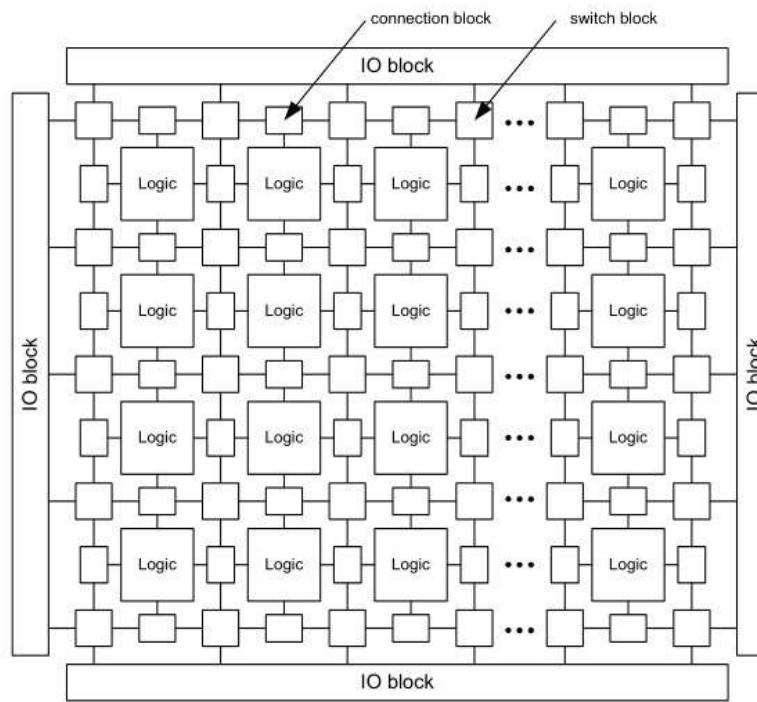
**Solution**  
**Contd.:**

Product term	AND Inputs					Outputs
	A	B	C	D	W	
1	1	1	0	—	—	$W = A B \bar{C}$
2	0	0	1	0	—	$+ \bar{A} \bar{B} C D$
3	—	—	—	—	—	
4	1	—	—	—	—	$X = A$
5	—	1	1	1	—	$+ B C D$
6	—	—	—	—	—	
7	0	1	—	—	—	$Y = \bar{A} B$
8	—	—	1	1	—	$+ C D$
9	—	0	—	0	—	$+ \bar{B} D$
10	—	—	—	—	1	$Z = W$
11	1	—	0	0	—	$+ A \bar{C} D$
12	0	0	0	1	—	$+ \bar{A} \bar{B} C D$



## Field Programmable Gate Array (FPGA)

- Typical FPGA consists of many small logic blocks interconnected by programmable routing resources.





# Thank You