



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



Switching Theory and Logic Design

ARI-209

By: Dr. Divya Agarwal



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT- II** 10 HOURS
- **Sequential Logic Circuits:** Latches and Flip Flops- SR, D, T and MS-JK Flip Flops, Asynchronous Inputs.
- **Counters and Shift Registers:** Design of Synchronous and Asynchronous Counters- Binary, BCD, Decade and Up/Down Counters, Shift Registers, Types of Shift Registers, Counters using Shift Registers- Ring Counter and Johnson Counter.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAJAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-III** **08 HOURS**
- **Integrated circuits:** TTL and CMOS logic families and their characteristics. Brief introduction to RAM and ROM
- **Synchronous Sequential Circuits:** State Tables State Equations and State Diagrams, State Reduction and State Assignment, Design of Clocked Sequential Circuits using State Equations.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-IV** 06 HOURS
- **Finite state machine:** capabilities and limitations, Mealy and Moore models-minimization of completely specified and incompletely specified sequential machines, Partition techniques and merger chart methods, concept of minimal cover table.
- **Algorithmic State Machine:** Representation of sequential circuits using ASM charts synthesis of output and next state functions, Data path control path partition-based design.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



Text Books:

1. Leach and Malvino (2011). *Digital principles and Applications*. Tata McGraw-Hill Education
2. Mano, M. M. (2017). *Digital logic and computer design*. Pearson Education India.
3. Jain, R. P. (2003). *Modern digital electronics*. Tata McGraw-Hill Education.

Reference Books:

1. A Anand Kumar. (2016) *Fundamentals of Digital Logic Circuits*, PHI
2. Taub, H., & Schilling, D. L. (1977). *Digital integrated electronics*. McGraw-Hill College.



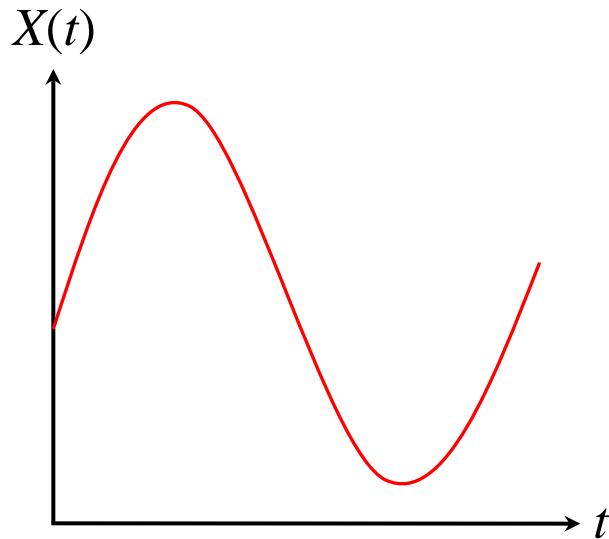
■ UNIT- I

14 HOURS

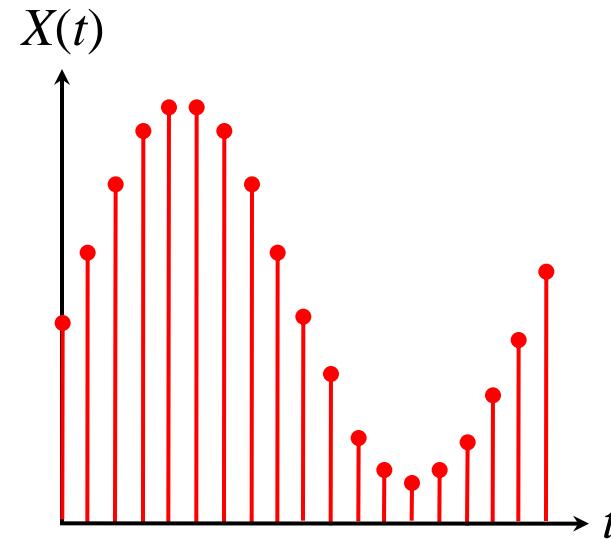
- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.

Analog and Digital Signal

- Analog system
 - ◆ The physical quantities or signals may vary continuously over a specified range.
- Digital system
 - ◆ The physical quantities or signals can assume only discrete values.
 - ◆ Greater accuracy



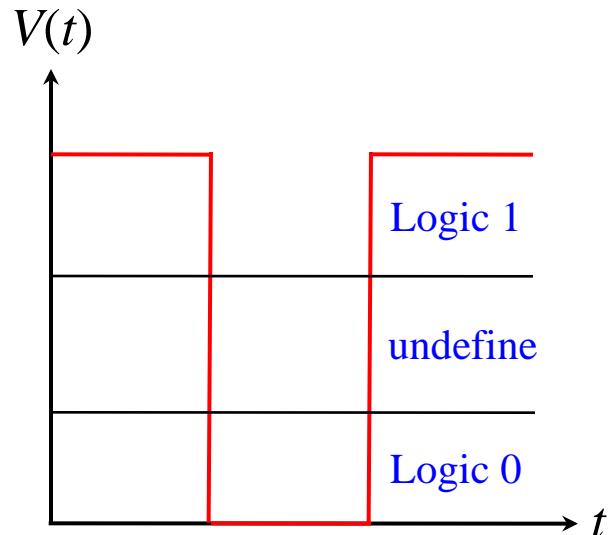
Analog signal



Digital signal

Binary Digital Signal

- An information variable represented by physical quantity.
- For digital systems, the variable takes on discrete values.
 - ◆ Two level, or binary values are the most prevalent values.
- Binary values are represented abstractly by:
 - ◆ Digits 0 and 1
 - ◆ Words (symbols) False (F) and True (T)
 - ◆ Words (symbols) Low (L) and High (H)
 - ◆ And words On and Off
- Binary values are represented by values or ranges of values of physical quantities.



Binary digital signal

Analog Vs Digital

Analog Signal

- Continuous
- Infinite range of values
- More exact values, but more difficult to work with

Digital Signal

- Discrete
- Finite range of values (2)
- Not as exact as analog, but easier to work with

Example:

A digital thermostat in a room displays a temperature of 72° . An analog thermometer measures the room temperature at 72.482° . The analog value is continuous and more accurate, but the digital value is more than adequate for the application and significantly easier to process electronically.

Example of Analog Vs Digital System



Digital
advantages:

Battery life

Programmability

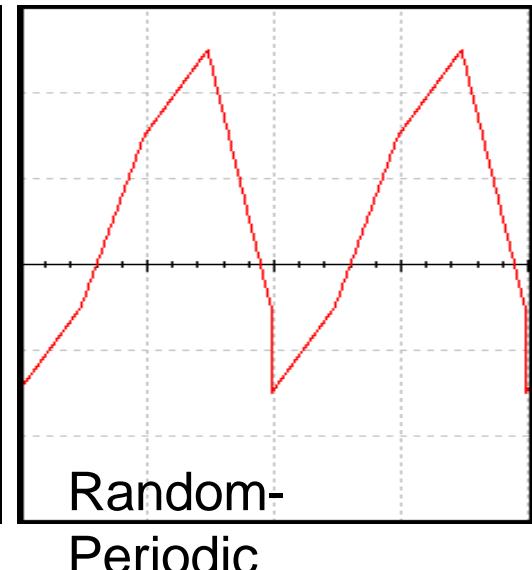
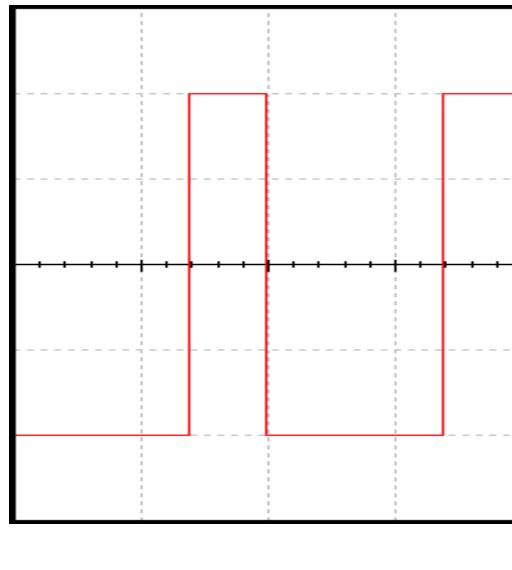
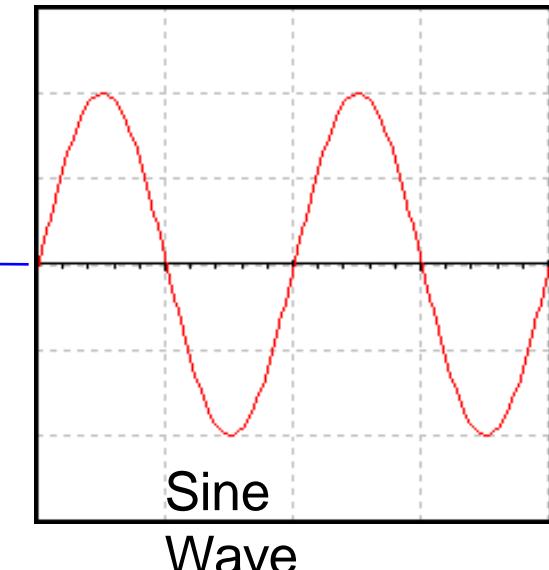
Accuracy

The World is Analog

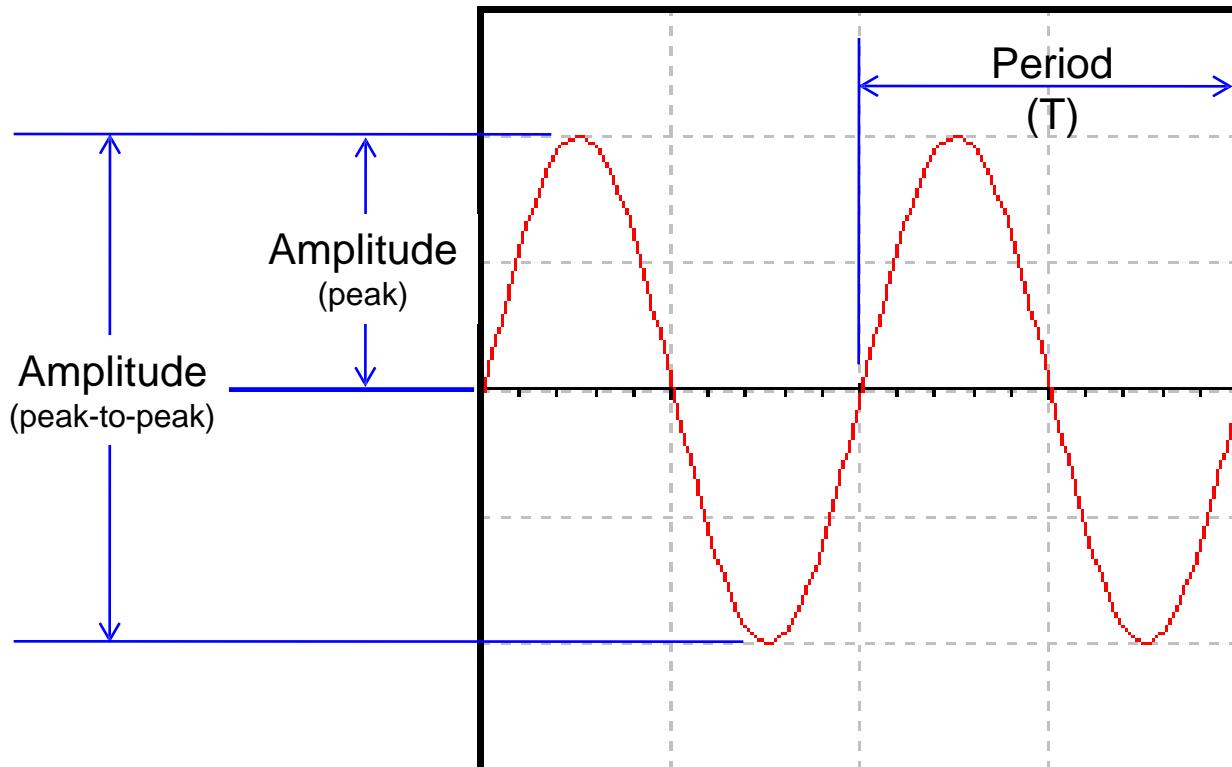
- ✓ The world we live in is analog.
- ✓ We are analog.
- ✓ Any inputs we can perceive are analog.
- ✓ For example,
 - sounds are analog signals; they are continuous time and continuous value.
 - Our ears listen to analog signals and we speak with analog signals.
 - Images, pictures, and video are all analog at the source and our eyes are analog sensors.
 - Measuring our heartbeat, tracking our activity, all requires processing analog sensor information.

Examples of Analog Signal

- ✓ An analog signal can be any time-varying signal.
- ✓ Minimum and maximum values can be either positive or negative.
- ✓ They can be periodic (repeating) or non-periodic.
- ✓ Sine waves and square waves are two common analog signals.
- ✓ Note that this square wave is not a digital signal because its minimum value is negative.
- ✓ Video and Audio



Parts of Analog Signal



Frequency:

$$F = \frac{1}{T} \text{ Hz}$$

Pros and Cons Analog Signal

➤ Advantages

- ✓ Major advantages of the Analog signal is infinite amount of data.
- ✓ Density is much higher.
- ✓ Easy processing.

➤ Disadvantages

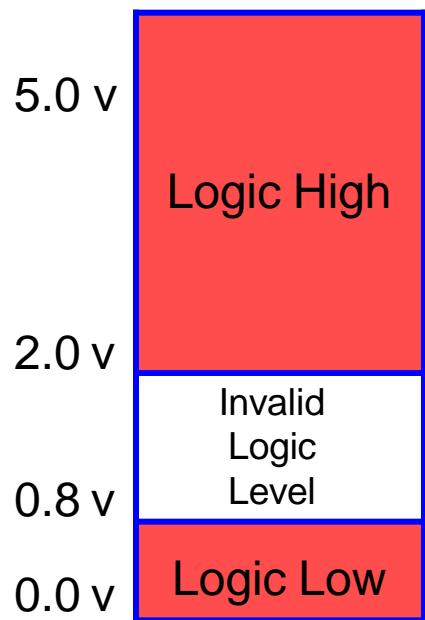
- ✓ Unwanted noise in recording.
- ✓ If we transmit data at long distance then unwanted disturbance is there.
- ✓ Generation loss is also a big con of analog signals.

Logic Levels

Before examining digital signals, we must define logic levels.

A logic level is a voltage level that represents a defined digital state.

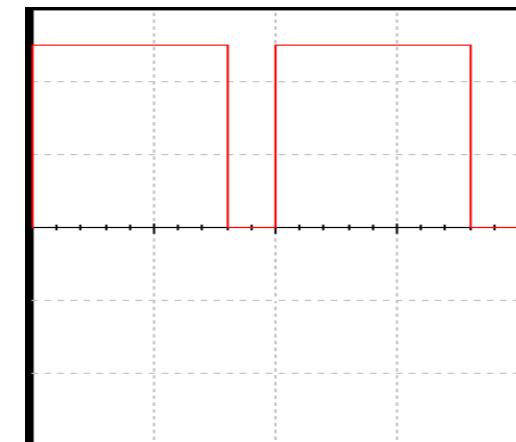
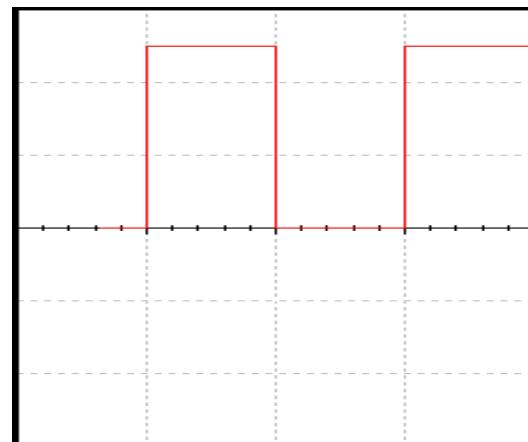
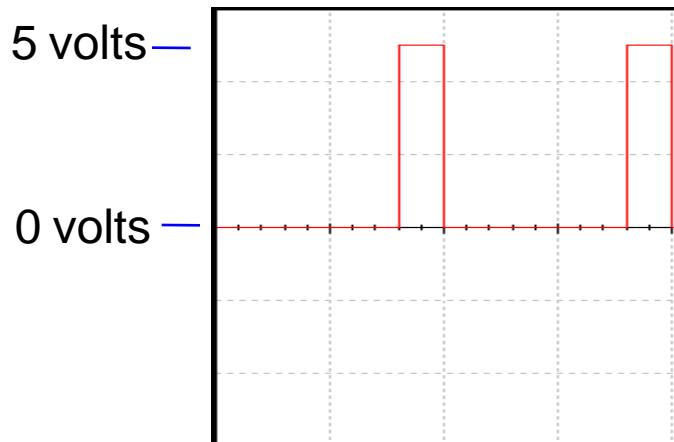
- ✓ Logic HIGH: The higher of two voltages, typically 5 volts
- ✓ Logic LOW: The lower of two voltages, typically 0 volts



Logic Level	Voltage	True/False	On/Off	0/1
HIGH	5 volts	True	On	1
LOW	0 volts	False	Off	0

Examples of Digital Signal

- ✓ Digital signal are commonly referred to as square waves or clock signals.
- ✓ Their minimum value must be 0 volts, and their maximum value must be 5 volts.
- ✓ They can be periodic (repeating) or non-periodic.
- ✓ The time the signal is high (t_H) can vary anywhere from 1% of the period to 99% of the period.
- ✓ Text and Integers.



Parts of Digital Signal

Amplitude:

For digital signals, this will ALWAYS be 5 volts.

Period:

The time it takes for a periodic signal to repeat. (seconds)

Frequency:

A measure of the number of occurrences of the signal per second. (Hertz, Hz)

Time High (t_H):

The time the signal is at 5 v.

Time Low (t_L):

The time the signal is at 0 v.

Duty Cycle:

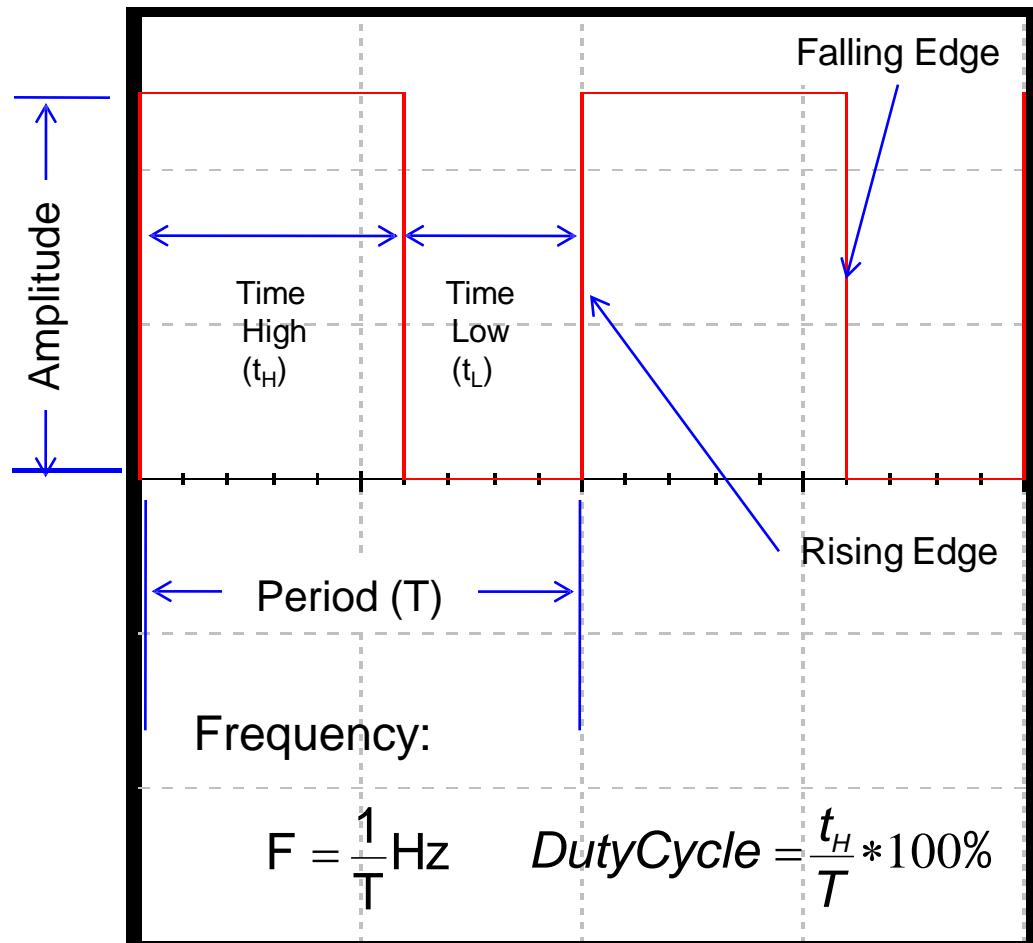
The ratio of t_H to the total period (T).

Rising Edge:

A 0-to-1 transition of the signal.

Falling Edge:

A 1-to-0 transition of the signal.



Pros and Cons Digital Signal

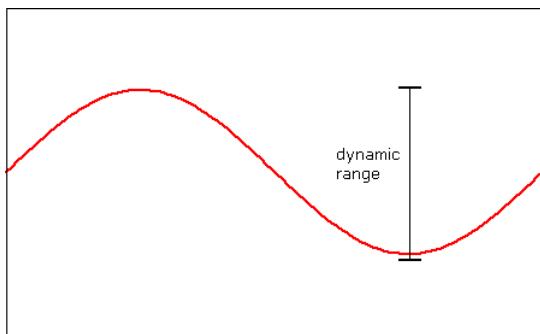
➤ Advantages

- ✓ Because of their digital nature they can travel faster in over digital lines.
- ✓ Ability to transfer more data as compared to analog.

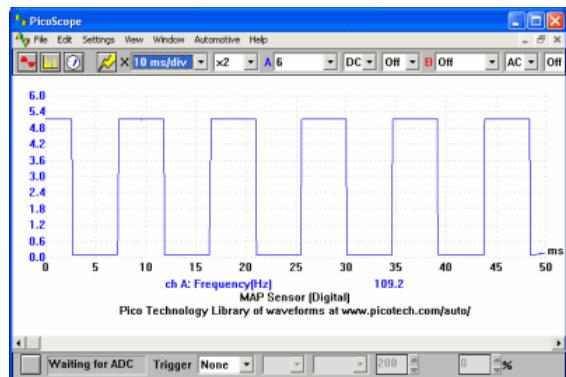
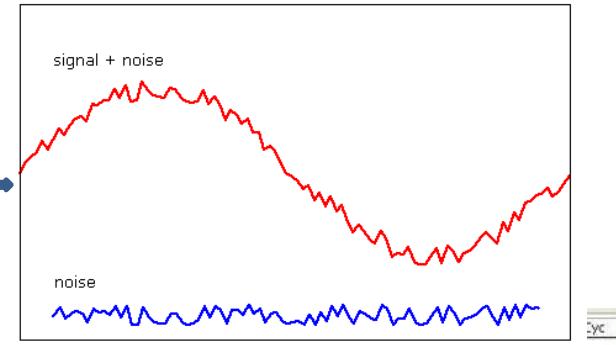
➤ Disadvantages

- ✓ Greater bandwidth is essential.
- ✓ Systems and processing is more complex.

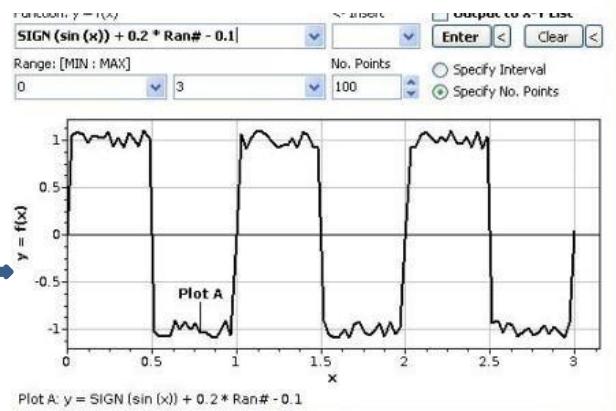
Example of using digital over analog:



Noisy channel



Noisy channel



- ✓ Digital systems are less sensitive to noise
- ✓ As long as 0 is distinguishable from 1

Advantages of Digital System over Analog System

- ✓ Digital Systems are easier to design
- ✓ Information storage is easy
- ✓ Accuracy & Precision are greater
- ✓ Digital systems are more versatile
- ✓ Digital circuits are less affected by noise
- ✓ More digital circuitry can be fabricated on IC chips
- ✓ Reliability is more

What is Signal?

- ✓ A signal is a physical quantity (sound, light, voltage, current) that carries **information**
 - The power cable supplies power but no information (not a signal)
 - A USB cable carries information (files)
- ✓ Examples of quantities used as digital information signals
 - Voltage: 5V (logic 1), 0V (logic 0) in digital circuits
 - Magnetic field orientation in magnetic hard disks
 - Pits and lands on the CD surface reflect the light from the laser differently, and that difference is encoded as binary data

Number System

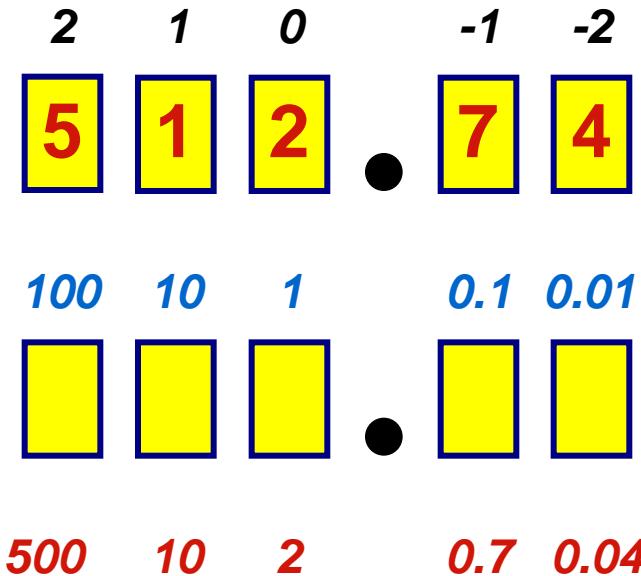
- ✓ A number system defines a set of values used to represent quantity.

Different Number Systems

- ✓ Decimal Number System - Base/Radix 10
- ✓ Binary Number System - Base/Radix 2
- ✓ Octal Number System - Base/Radix 8
- ✓ Hexadecimal Number System - Base/Radix 16

Decimal Number System

- Base (also called radix) = 10
 - ◆ 10 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
- Digit Position
 - ◆ Integer & fraction
- Digit Weight
 - ◆ Weight = $(Base)^{Position}$
- Magnitude
 - ◆ Sum of “Digit x Weight”
- Formal Notation



$$d_2 * B^2 + d_1 * B^1 + d_0 * B^0 + d_{-1} * B^{-1} + d_{-2} * B^{-2}$$

$$(512.74)_{10}$$

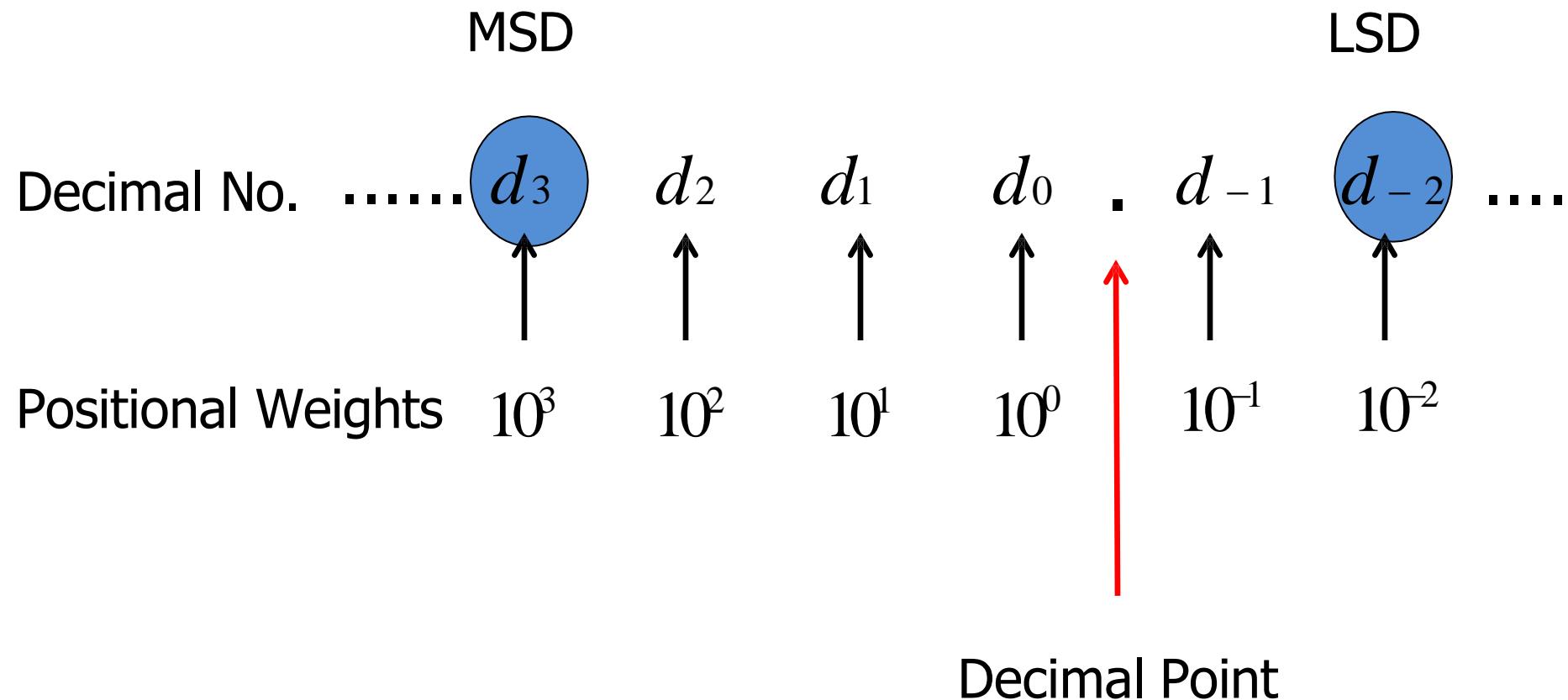


Decimal Number System

- ✓ Decimal number system contains ten unique symbols 0,1,2,3,4,5,6,7,8 and 9
- ✓ Since counting in decimal involves ten symbols, we can say that its base or radix is ten.
- ✓ It is a positional weighted system
- ✓ In this system, any number (integer, fraction or mixed) of any magnitude can be represented by the use of these ten symbols only
- ✓ Each symbols in the number is called a “Digit”

Decimal Number System

Structure:



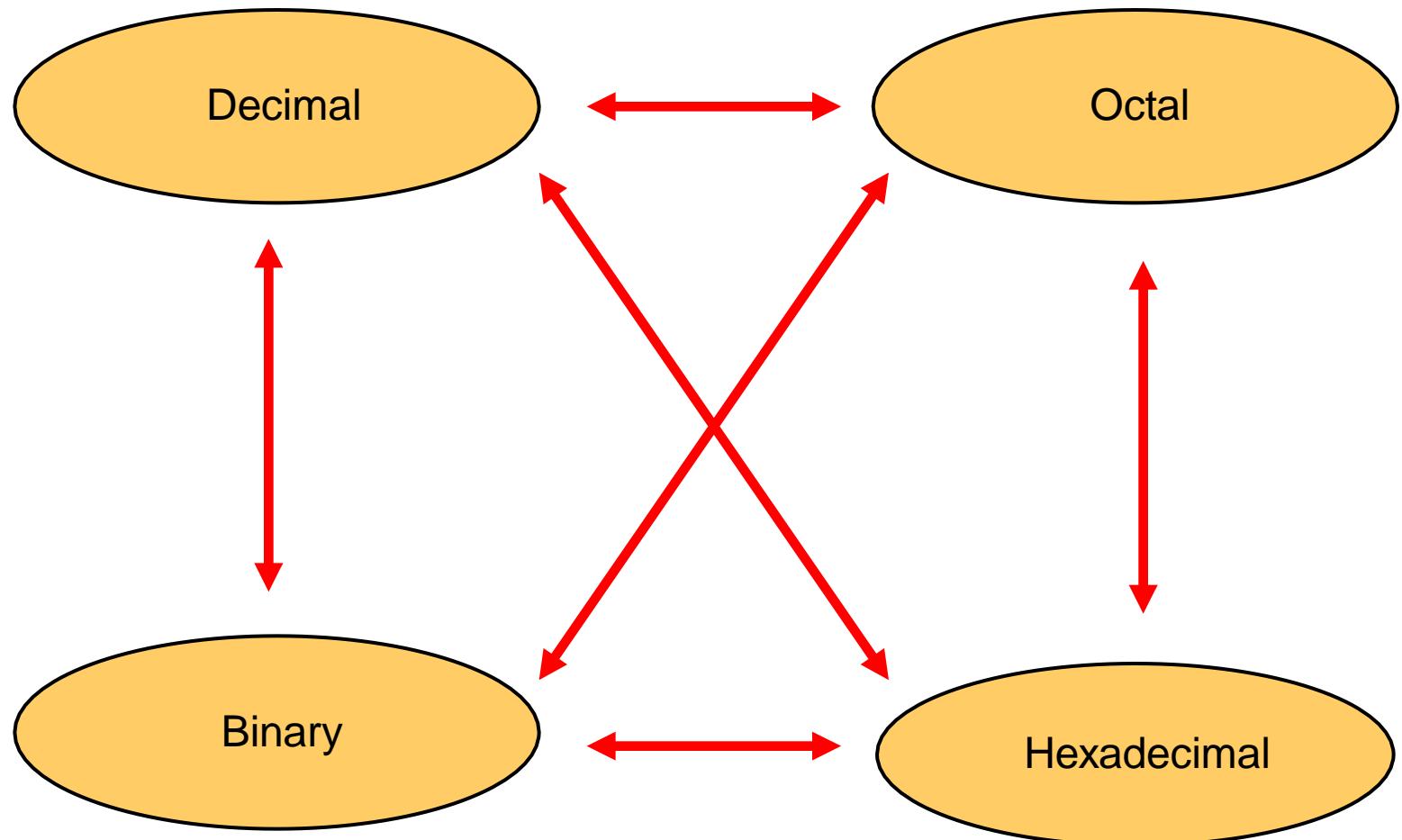
Decimal Number System

- ✓ **MSD:** The leftmost digit in any number representation, which has the greatest positional weight out of all the digits present in that number is called the “Most Significant Digit” (MSD)

- ✓ **LSD:** The rightmost digit in any number representation, which has the least positional weight out of all the digits present in that number is called the “Least Significant Digit” (LSD)

CONVERSION AMONG BASES

Possibilities

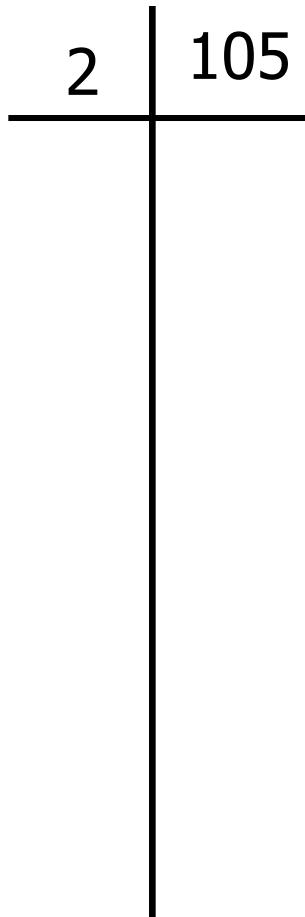


CONVERSION OF DECIMAL NUMBER INTO BINARY NUMBER (INTEGER NUMBER)

Procedure:

1. Divide the decimal no by the base 2, noting the remainder.
2. Continue to divide the quotient by 2 until there is nothing left, keeping the track of the remainders from each step.
3. List the remainder values in reverse order to find the number's binary equivalent

**EXAMPLE: CONVERT 105 DECIMAL NUMBER IN
TO IT'S EQUIVALENT BINARY NUMBER.**



**EXAMPLE: CONVERT 105 DECIMAL NUMBER IN
TO IT'S EQUIVALENT BINARY NUMBER.**

$$\begin{array}{r} 105 \\ \hline 2 | 52 \quad 1 \\ \hline 2 | 26 \\ \hline 2 | 13 \\ \hline 2 | 6 \\ \hline 2 | 3 \\ \hline 2 | 1 \\ \hline \end{array}$$

**EXAMPLE: CONVERT 105 DECIMAL NUMBER IN
TO IT'S EQUIVALENT BINARY NUMBER.**

2	105	
2	52	1
2	26	0

EXAMPLE: CONVERT 105 DECIMAL NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

2	105	
2	52	1
2	26	0
2	13	0

EXAMPLE: CONVERT 105 DECIMAL NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

2	105	
2	52	1
2	26	0
2	13	0
2	6	1

EXAMPLE: CONVERT 105 DECIMAL NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

2	105	
2	52	1
2	26	0
2	13	0
2	6	1
2	3	0

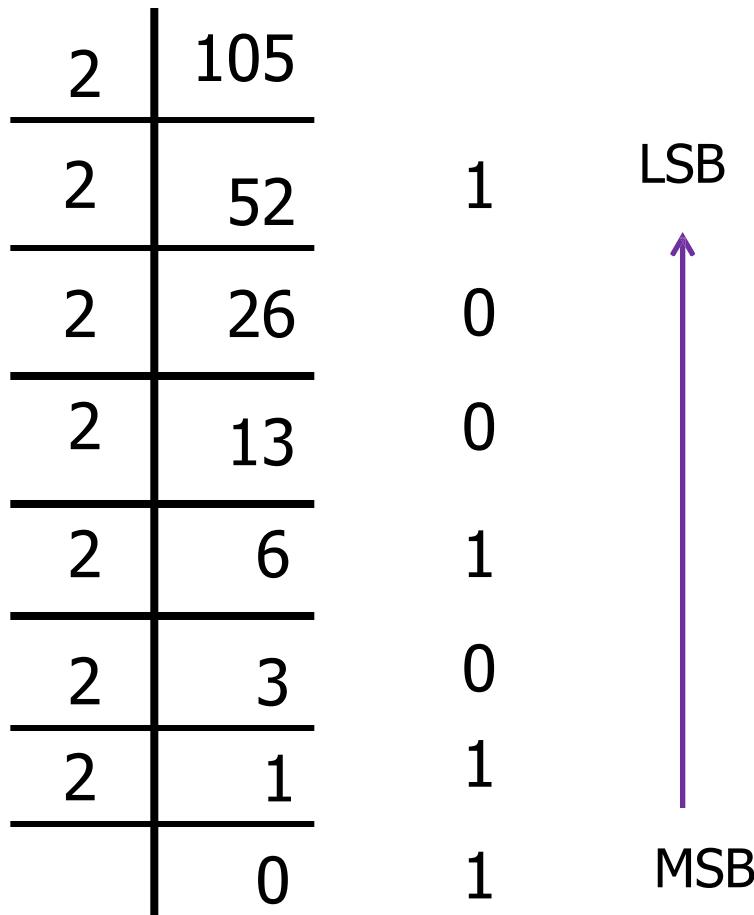
EXAMPLE: CONVERT 105 DECIMAL NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

2	105	
2	52	1
2	26	0
2	13	0
2	6	1
2	3	0
2	1	1

EXAMPLE: CONVERT 105 DECIMAL NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

2	105	
2	52	1
2	26	0
2	13	0
2	6	1
2	3	0
2	1	1
	0	1

EXAMPLE: CONVERT 105 DECIMAL NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.



$$(105)_{10} = (1101001)_2$$

CONVERSION OF DECIMAL NUMBER INTO BINARY NUMBER (FRACTIONAL NUMBER)

Procedure:

1. Multiply the given fractional number by base 2.
2. Record the carry generated in this multiplication as MSB.
3. Multiply only the fractional number of the product in step 2 by 2 and record the carry as the next bit to MSB.
4. Repeat the steps 2 and 3 up to 5 bits. The last carry will represent the LSB of equivalent binary number

EXAMPLE: CONVERT 0.42 DECIMAL NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

$$0.42 \times 2 = 0.84 \quad 0$$



EXAMPLE: CONVERT 0.42 DECIMAL NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

$$0.42 \times 2 = 0.84 \quad 0$$

$$0.84 \times 2 = 1.68 \quad 1$$

EXAMPLE: CONVERT 0.42 DECIMAL NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

$$0.42 \times 2 = 0.84 \quad 0$$

$$0.84 \times 2 = 1.68 \quad 1$$

$$0.68 \times 2 = 1.36 \quad 1$$

EXAMPLE: CONVERT 0.42 DECIMAL NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

$$\begin{array}{rcl} 0.42 \times 2 & = & 0.84 \quad 0 \\ & & \text{---} \\ 0.84 \times 2 & = & 1.68 \quad 1 \\ & & \text{---} \\ 0.68 \times 2 & = & 1.36 \quad 1 \\ & & \text{---} \\ 0.36 \times 2 & = & 0.72 \quad 0 \end{array}$$

EXAMPLE: CONVERT 0.42 DECIMAL NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

$$\begin{array}{rcl} 0.42 \times 2 & = & 0.84 \quad 0 \\ & & \text{---} \\ 0.84 \times 2 & = & 1.68 \quad 1 \\ & & \text{---} \\ 0.68 \times 2 & = & 1.36 \quad 1 \\ & & \text{---} \\ 0.36 \times 2 & = & 0.72 \quad 0 \\ & & \text{---} \\ 0.72 \times 2 & = & 1.44 \quad 1 \end{array}$$

EXAMPLE: CONVERT 0.42 DECIMAL NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

$$\begin{array}{rcl} 0.42 \times 2 & = & 0.84 \quad 0 \\ 0.84 \times 2 & = & 1.68 \quad 1 \\ 0.68 \times 2 & = & 1.36 \quad 1 \\ 0.36 \times 2 & = & 0.72 \quad 0 \\ 0.72 \times 2 & = & 1.44 \quad 1 \end{array}$$

MSB

LSB

$$(0.42)_{10} = (0.01101)_2$$

Exercise

- Convert following Decimal Numbers in to its equivalent Binary Number:

$$1. (1248.56)_{10} = (?)_2$$

$$2. (8957.75)_{10} = (?)_2$$

$$3. (420.6)_{10} = (?)_2$$

$$4. (8476.47)_{10} = (?)_2$$

Exercise

- Convert following Decimal Numbers in to its equivalent Binary Number:

$$1. \ (1248.56)_{10} = (10011100000.10001)_2$$

$$2. \ (8957.75)_{10} = (1000101111101.11)_2$$

$$3. \ (420.6)_{10} = (110100100.10011)_2$$

$$4. \ (8476.47)_{10} = (10000100011100.01111)_2$$

Binary Number System

- Base = 2

- ◆ 2 digits { 0, 1 }, called *binary digits* or “*bits*”

- Weights

- ◆ Weight = $(Base)^{Position}$

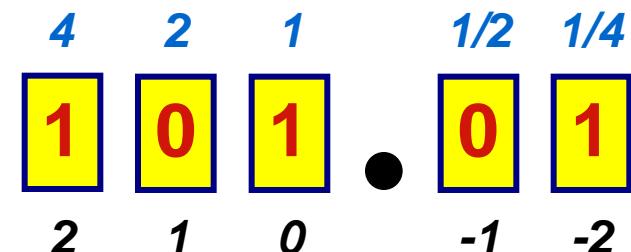
- Magnitude

- ◆ Sum of “*Bit x Weight*”

- Formal Notation

- Groups of bits 4 bits = *Nibble*

8 bits = *Byte*



$$1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2}$$

$$= (5.25)_{10}$$
$$(101.01)_2$$

1 0 1 1

1 1 0 0 0 1 0 1

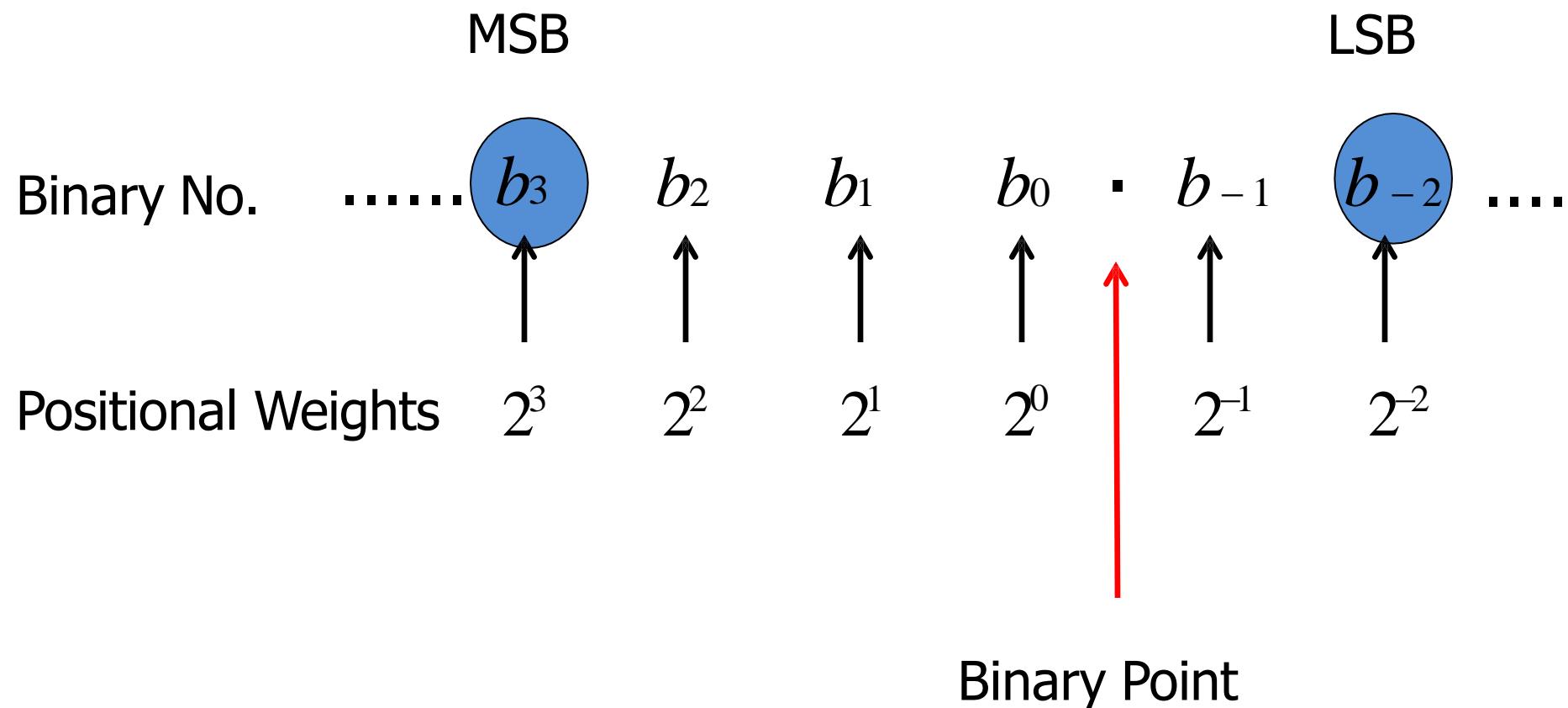


Binary Number System

- ✓ Binary number system is a positional weighted system
- ✓ It contains two unique symbols 0 and 1
- ✓ Since counting in binary involves two symbols, so its base or radix is two.
- ✓ A binary digit is called a “Bit”
- ✓ A binary number consists of a sequence of bits, each of which is either a 0 or a 1.
- ✓ The binary point separates the integer and fraction parts

Binary Number System

Structure:



BINARY NUMBER SYSTEM

✓ **MSB:** The leftmost bit in a given binary number with the highest positional weight is called the “Most Significant Bit” (MSB)

✓ **LSB:** The rightmost bit in a given binary number with the lowest positional weight is called the “Least Significant Bit” (LSB)

BINARY NUMBER SYSTEM

Decimal No.	Binary No.
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Decimal No.	Binary No.
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Terms related to Binary Numbers

✓ **BIT:** The binary digits (0 and 1) are called bits. Single unit in binary digit is called “Bit”

- Example 1, 0

✓ **NIBBLE:** A nibble is a combination of 4 binary bits.

- Example 1110

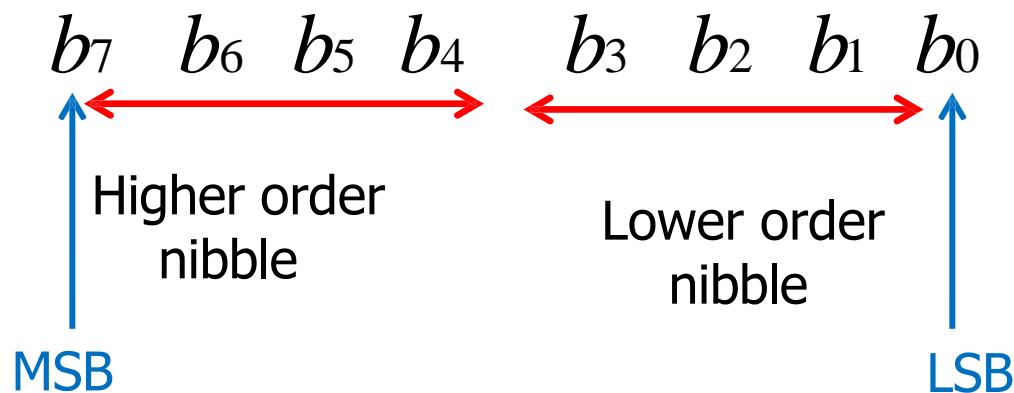
0000

1001

0101

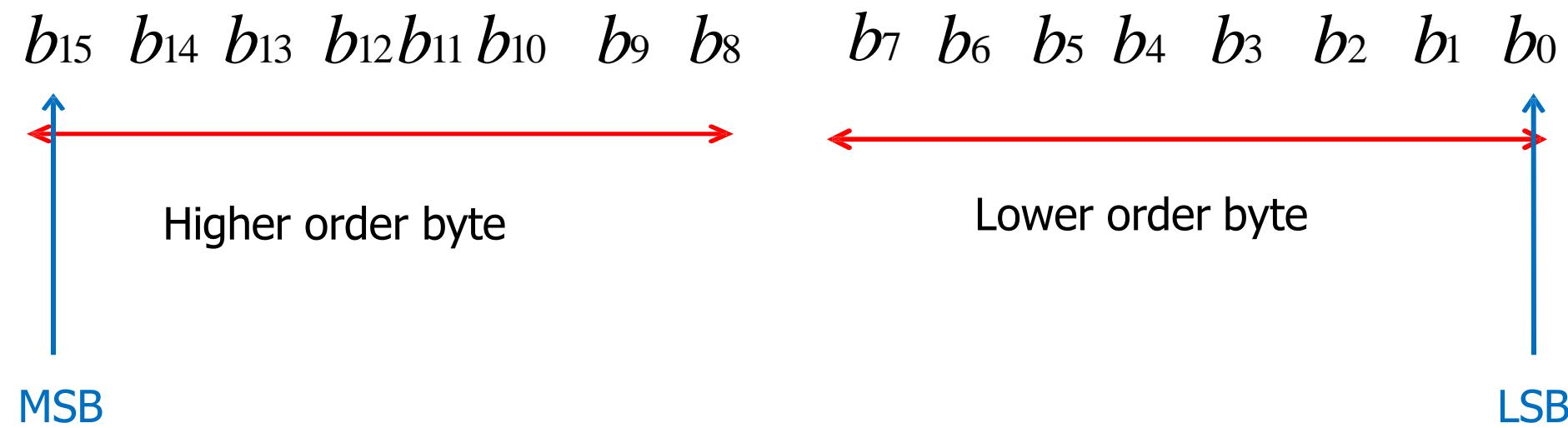
TERMS RELATED TO BINARY NUMBERS

- ✓ **BYTE:** A byte is a combination of 8 binary bits. The number of distinct values represented by a byte is 256 ranging from 0000 0000 to 1111 1111.



Terms related to Binary Numbers

✓ **WORD:** A word is a combination of 16 binary bits. Hence it consists of two bytes.



✓ **DOUBLE WORD:** A double word is exactly what its name implies, two words. It is a combination of 32 binary bits.

Conversion of Binary Number into Decimal Number

PROCEDURE:

1. Write down the binary number.
2. Write down the weights for different positions.
3. Multiply each bit in the binary number with the corresponding weight to obtain product numbers to get the decimal numbers.
4. Add all the product numbers to get the decimal equivalent

Example: Convert 1011.01 binary number in to it's equivalent decimal number.

Binary No.

1 0 1 1 • 0 1

Example: Convert 1011.01 binary number in to it's equivalent decimal number.

Binary No.	1	0	1	1	•	0	1
Positional Weights	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}

```
graph TD; A[1] --> B["23"]; C[0] --> D["22"]; E[1] --> F["21"]; G[1] --> H["20"]; I["•"] --> J["2-1"]; K[0] --> L["2-1"]; M[1] --> N["2-2"]
```

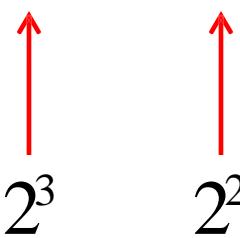
Example: Convert 1011.01 binary number in to it's equivalent decimal number.

Binary No.

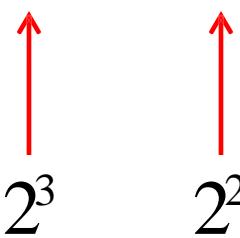
1 0 1 1 • 0



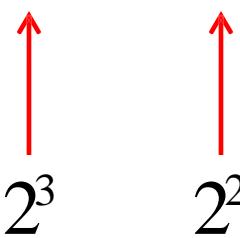
2³



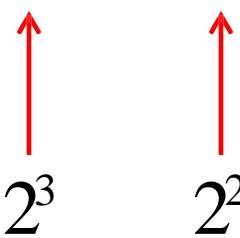
2²



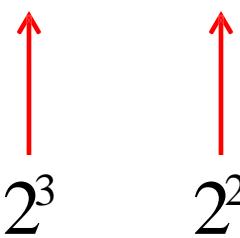
2¹



2⁰



2⁻¹



2⁻²

1

2⁻²

$$=(1*2^3)+(0*2^2)+(1*2^1)+(1*2^0).(0*2^{-1})+(1*2^{-2})$$

Example: Convert 1011.01 binary number into its equivalent decimal number.

Binary No.	1	0	1	1	·	0	1
Positional Weights	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}

$$\begin{aligned}
 &= (1*2^3) + (0*2^2) + (1*2^1) + (1*2^0). (0*2^{-1}) + (1*2^{-2}) \\
 &= 8 + 0 + 2 + 1 . 0 + 0.25
 \end{aligned}$$

Example: Convert 1011.01 binary number in to it's equivalent decimal number.

Binary No.	1	0	1	1	•	0	1
Positional Weights	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}

$$= (1*2^3) + (0*2^2) + (1*2^1) + (1*2^0). (0*2^{-1}) + (1*2^{-2})$$

$$= 8 + 0 + 2 + 1 . 0 + 0.25$$

$$= 11.25$$

Example: Convert 1011.01 binary number in to it's equivalent decimal number.

Binary No.	1	0	1	1	•	0	1
Positional Weights	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}

$$\begin{aligned} &= (1*2^3) + (0*2^2) + (1*2^1) + (1*2^0). (0*2^{-1}) + (1*2^{-2}) \\ &= 8 + 0 + 2 + 1 . 0 + 0.25 \\ &= 11.25 \end{aligned}$$

$$(1011.01)_2 = (11.25)_{10}$$

Exercise

- Convert following Binary Numbers in to its equivalent Decimal Number:

$$1. \ (1101110.011)_2 = (?)_{10}$$

$$2. \ (1101.11)_2 = (?)_{10}$$

$$3. \ (10001.01)_2 = (?)_{10}$$

Exercise

- Convert following Binary Numbers in to its equivalent Decimal Number:

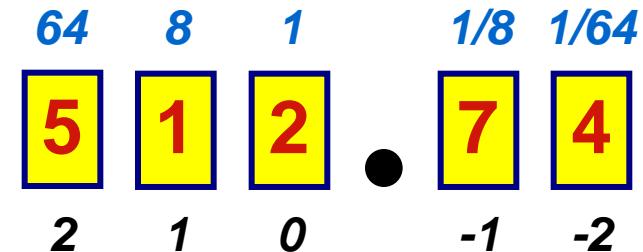
$$1. \ (1101110.011)_2 = (110.375)_{10}$$

$$2. \ (1101.11)_2 = (13.75)_{10}$$

$$3. \ (10001.01)_2 = (17.25)_{10}$$

Octal Number System

- Base = 8
 - ◆ 8 digits { 0, 1, 2, 3, 4, 5, 6, 7 }
- Weights
 - ◆ Weight = $(Base)^{Position}$
- Magnitude
 - ◆ Sum of “Digit x Weight”
- Formal Notation



$$5 * 8^2 + 1 * 8^1 + 2 * 8^0 + 7 * 8^{-1} + 4 * 8^{-2}$$

$$= (330.9375)_{10}$$
$$(512.74)_8$$

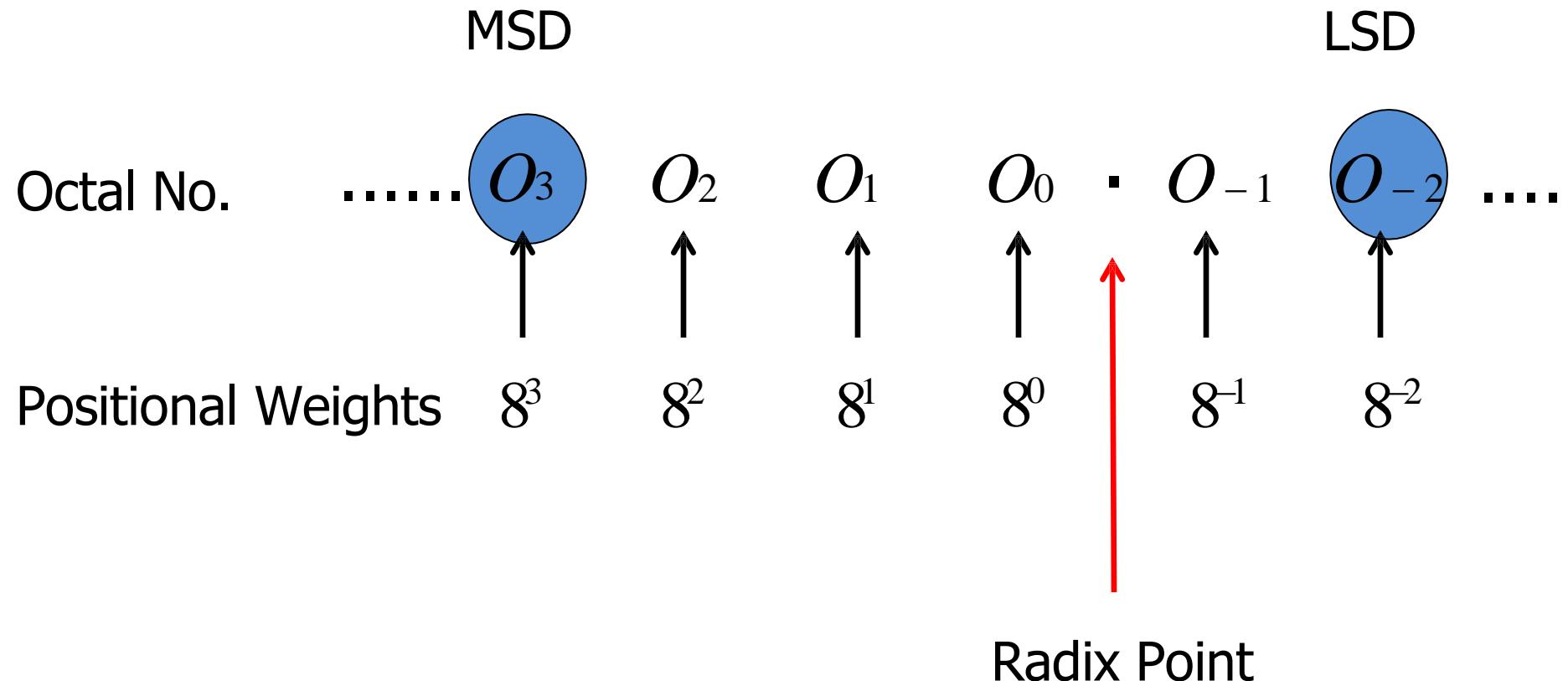


Octal Number System

- ✓ Octal number system is a weighted positional system
- ✓ It contains eight unique symbols 0,1,2,3,4,5,6 and 7
- ✓ Since counting in octal involves eight symbols, we can say that its base or radix is eight.
- ✓ Largest value of a digit in octal system will be 7.
- ✓ That means the octal number higher than 7 will not be 8, instead of that it will be 10.
- ✓ Since its base $8 = 2^3$, every 3 bit group of binary can be represented by an octal digit.
- ✓ An octal number is thus $1/3^{\text{rd}}$ the length of the corresponding binary number

Octal Number System

Structure:



Octal Number System

Decimal No.	Binary No.	Octal No.
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	10
9	1001	11
10	1010	12
11	1011	13
12	1100	14
13	1101	15

Conversion of Octal Number into Decimal Number

PROCEDURE:

1. Write down the octal number.
2. Write down the weights for different positions.
3. Multiply each bit in the binary number with the corresponding weight to obtain product numbers to get the decimal numbers.
4. Add all the product numbers to get the decimal equivalent

Example: Convert 365.24 octal number in
to it's equivalent decimal number.

Octal No.

3 6 5 • 2 4

Example: Convert 365.24 octal number in to it's equivalent decimal number.

Octal No.	3	6	5	•	2	4
	↑	↑	↑		↑	↑
Positional Weights	8^2	8^1	8^0		8^{-1}	8^{-2}

Example: Convert 365.24 octal number in to it's equivalent decimal number.

Octal No.	3	6	5	•	2	4
	↑	↑	↑		↑	↑
Positional Weights	8^2	8^1	8^0		8^{-1}	8^{-2}

$$=(3*8^2)+(6*8^1)+(5*8^0).(2*8^{-1})+(4*8^{-2})$$

Example: Convert 365.24 octal number in to it's equivalent decimal number.

Octal No.	3	6	5	•	2	4
	↑	↑	↑		↑	↑
Positional Weights	8^2	8^1	8^0		8^{-1}	8^{-2}

$$= (3*8^2) + (6*8^1) + (5*8^0). (2*8^{-1}) + (4*8^{-2})$$

$$= 192 + 48 + 5 . 0.25 + 0.0625$$

Example: Convert 365.24 octal number in to it's equivalent decimal number.

Octal No.	3	6	5	•	2	4
	↑	↑	↑		↑	↑
Positional Weights	8^2	8^1	8^0		8^{-1}	8^{-2}

$$= (3*8^2) + (6*8^1) + (5*8^0). (2*8^{-1}) + (4*8^{-2})$$

$$= 192 + 48 + 5 . 0.25 + 0.0625$$

$$= 245.3125$$

Example: Convert 365.24 octal number in to it's equivalent decimal number.

Octal No.	3	6	5	•	2	4
	↑	↑	↑		↑	↑
Positional Weights	8^2	8^1	8^0		8^{-1}	8^{-2}
$= (3*8^2) + (6*8^1) + (5*8^0). (2*8^{-1}) + (4*8^{-2})$						
$= 192 + 48 + 5 . 0.25 + 0.0625$						
$= 245.3125$						

$$(365.24)_8 = (245.3125)_{10}$$

Exercise

- Convert following Octal Numbers into its equivalent Decimal Number:

$$1. \ (3006.05)_8 = (?)_{10}$$

$$2. \ (273.56)_8 = (?)_{10}$$

$$3. \ (6534.04)_8 = (?)_{10}$$

Exercise

- Convert following Octal Numbers into its equivalent Decimal Number:

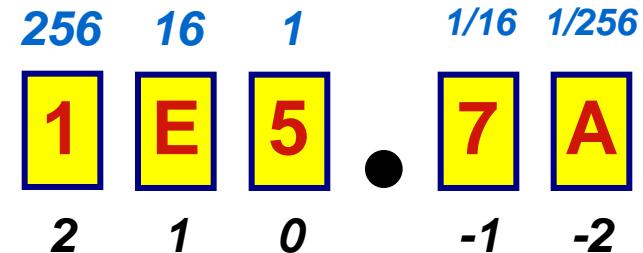
$$1. \ (3006.05)_8 = (1542.078125)_{10}$$

$$2. \ (273.56)_8 = (187.71875)_{10}$$

$$3. \ (6534.04)_8 = (3420.0625)_{10}$$

Hexadecimal Number System (HEX)

- Base = 16
 - ◆ 16 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }
- Weights
 - ◆ Weight = $(Base)^{Position}$
- Magnitude
 - ◆ Sum of “Digit x Weight”
- Formal Notation



$$\begin{aligned} & 1 * 16^2 + 14 * 16^1 + 5 * 16^0 + 7 * 16^{-1} + 10 * 16^{-2} \\ & = (485.4765625)_{10} \end{aligned}$$

$(1E5.7A)_{16}$

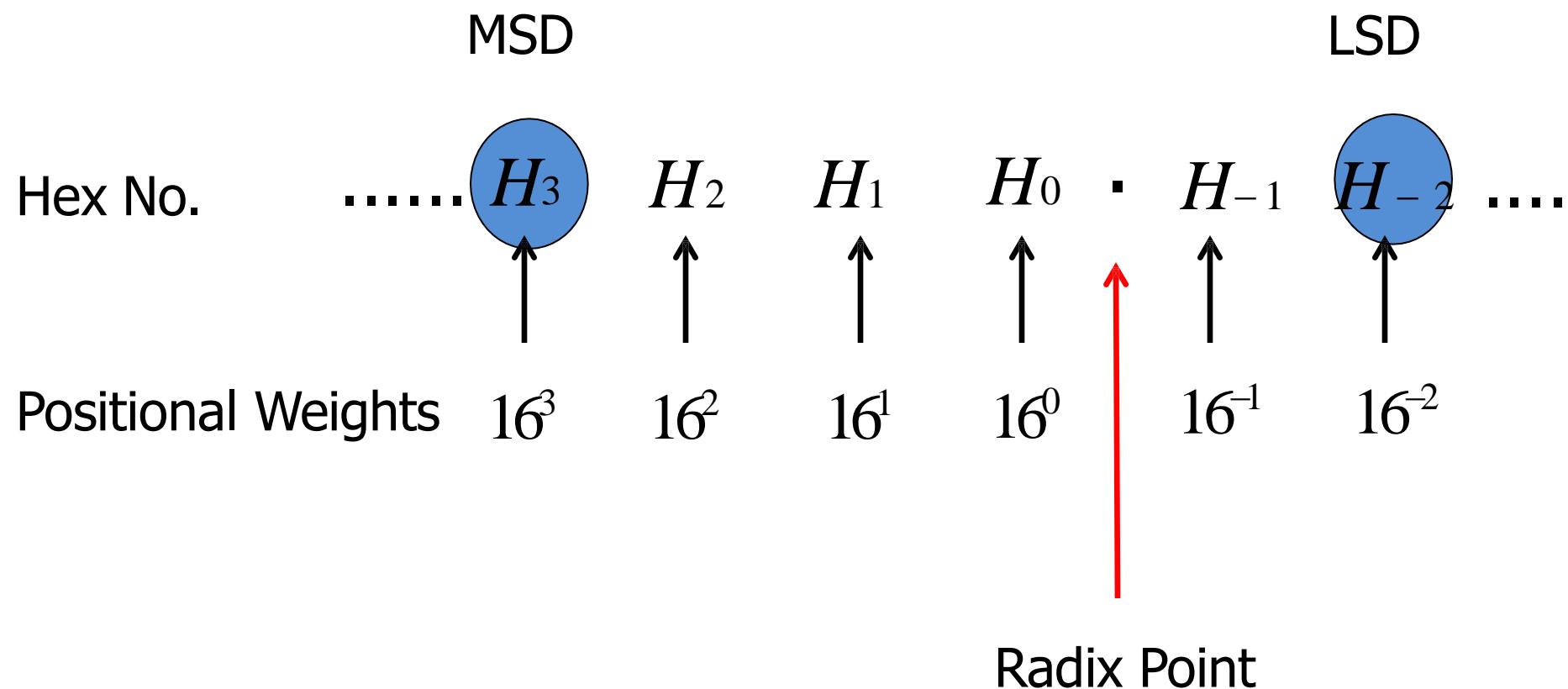


HEXADECIMAL NUMBER SYSTEM (HEX)

- ✓ Binary numbers are long. These numbers are fine for machines but are too lengthy to be handled by human beings. So there is a need to represent the binary numbers concisely.
- ✓ One number system developed with this objective is the hexadecimal number system (or Hex)
- ✓ Hex number system is a weighted positional system
- ✓ Contains 16 unique symbols 0 – 9 and A - F.
- ✓ Since counting in hex involves sixteen symbols, we can say that its base or radix is sixteen.
- ✓ Since its base $16 = 2^4$, every 4 bit group of binary can be represented by an hex digit.
- ✓ An hex number is thus $1/4^{\text{th}}$ the length of the corresponding binary number
- ✓ Hex system is particularly useful for human communications with computer

Hexadecimal Number System (HEX)

Structure:



HEXADECIMAL NUMBER SYSTEM (HEX)

Decimal No.	Binary No.	Hex No.
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

Decimal No.	Binary No.	Hex No.
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Conversion of Hexadecimal Number into Decimal Number

PROCEDURE:

1. Write down the hex number.
2. Write down the weights for different positions.
3. Multiply each bit in the binary number with the corresponding weight to obtain product numbers to get the decimal numbers.
4. Add all the product numbers to get the decimal equivalent

Example: Convert 5826 hex number in to it's equivalent decimal number.

Hex No.

5 8 2 6

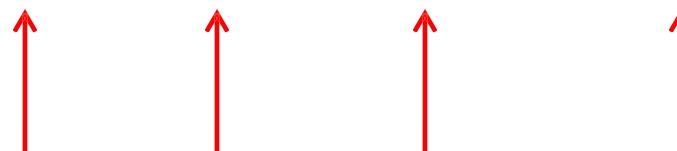
Example: Convert 5826 hex number in to it's equivalent decimal number.

Hex No.	5	8	2	6
	↑	↑	↑	↑
Positional Weights	16^3	16^2	16^1	16^0

Example: Convert 5826 hex number in to it's equivalent decimal number.

Hex No.

5 8 2 6



Positional Weights

16^3 16^2 16^1 16^0

$$= (5*16^3) + (8*16^2) + (2*16^1) + (6*16^0)$$

Example: Convert 5826 hex number in to it's equivalent decimal number.

Hex No.

5 8 2 6



Positional Weights

16^3 16^2 16^1 16^0

$$= (5*16^3) + (8*16^2) + (2*16^1) + (6*16^0)$$

$$= 20480 + 2048 + 32 + 6$$

Example: Convert 5826 hex number in to it's equivalent decimal number.

Hex No.

5 8 2 6

↑ ↑ ↑ ↑

Positional Weights

16^3 16^2 16^1 16^0

$$= (5*16^3) + (8*16^2) + (2*16^1) + (6*16^0)$$
$$= 20480 \quad \quad + 2048 \quad \quad + \quad \quad 32$$

$$\quad \quad \quad + \quad \quad \quad 6$$

$$= 22566$$

Example: Convert 5826 hex number in to it's equivalent decimal number.

Hex No.

5 8 2 6



Positional Weights

16^3 16^2 16^1 16^0

$$= (5*16^3) + (8*16^2) + (2*16^1) + (6*16^0)$$

$$= 20480 + 2048 + 32 + 6$$

$$= 22566$$

$$(5826)_{16} = (22566)_{10}$$

EXERCISE

- Convert following Hexadecimal Numbers in to its equivalent Decimal Number:

$$1. \ (4056)_{16} = (?)_{10}$$

$$2. \ (6B7)_{16} = (?)_{10}$$

$$3. \ (8E47.AB)_{16} = (?)_{10}$$

EXERCISE

- Convert following Hexadecimal Numbers in to its equivalent Decimal Number:

$$1. \ (4056)_{16} = (16470)_{10}$$

$$2. \ (6B7)_{16} = (1719)_{10}$$

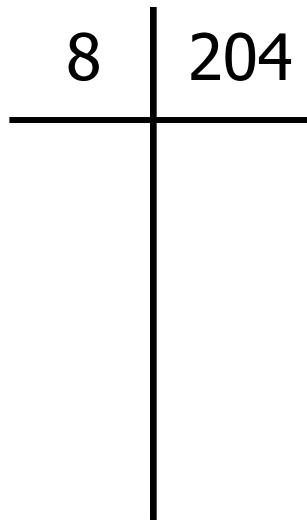
$$3. \ (8E47.AB)_{16} = (36423.66796875)_{10}$$

Conversion of Decimal Number into Octal Number (Integer Number)

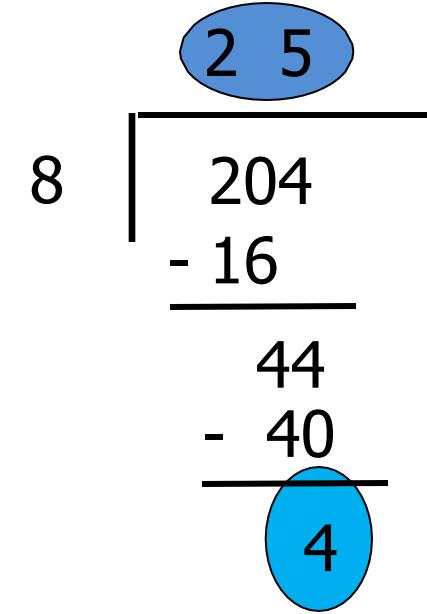
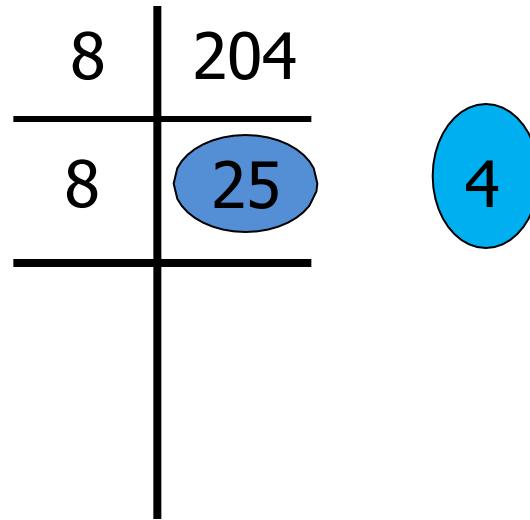
PROCEDURE:

1. Divide the decimal no by the base 8, noting the remainder.
2. Continue to divide the quotient by 8 until there is nothing left, keeping the track of the remainders from each step.
3. List the remainder values in reverse order to find the number's octal equivalent

Example: Convert 204 decimal number in to it's equivalent octal number.



EXAMPLE: CONVERT 204 DECIMAL NUMBER IN TO IT'S EQUIVALENT OCTAL NUMBER.



EXAMPLE: CONVERT 204 DECIMAL NUMBER IN TO IT'S EQUIVALENT OCTAL NUMBER.

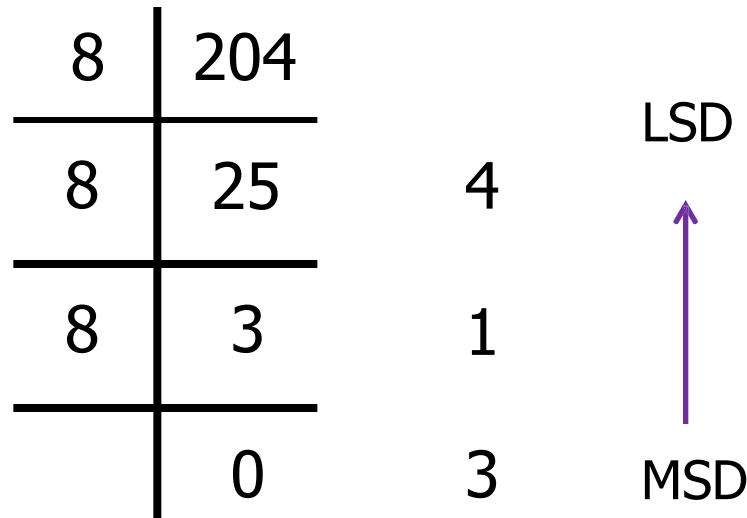
$$\begin{array}{r} 8 | 204 \\ \hline 8 | 25 \qquad 4 \\ \hline 8 | 3 \qquad 1 \end{array}$$

$$\begin{array}{r} 8 | 25 \\ \hline - 24 \\ \hline 1 \end{array}$$

EXAMPLE: CONVERT 204 DECIMAL NUMBER IN TO IT'S EQUIVALENT OCTAL NUMBER.

8	204	
8	25	4
8	3	1
8	0	3

EXAMPLE: CONVERT 204 DECIMAL NUMBER IN TO IT'S EQUIVALENT OCTAL NUMBER.



$$(204)_{10} = (314)_8$$

CONVERSION OF DECIMAL NUMBER INTO OCTAL NUMBER (FRACTIONAL NUMBER)

Procedure:

1. Multiply the given fractional number by base 8.
2. Record the carry generated in this multiplication as MSD.
3. Multiply only the fractional number of the product in step 2 by 8 and record the carry as the next bit to MSD.
4. Repeat the steps 2 and 3 up to 5 bits. The last carry will represent the LSD of equivalent octal number

**EXAMPLE: CONVERT 0.6234 DECIMAL NUMBER
IN TO IT'S EQUIVALENT OCTAL NUMBER.**

$$0.6234 \times 8 = 4.9872 \quad 4$$



EXAMPLE: CONVERT 0.6234 DECIMAL NUMBER IN TO IT'S EQUIVALENT OCTAL NUMBER.

$$\begin{array}{rcl} 0.6234 \times 8 & = & 4.9872 \\ & & \downarrow \\ 0.9872 \times 8 & = & 7.8976 \end{array}$$

EXAMPLE: CONVERT 0.6234 DECIMAL NUMBER IN TO IT'S EQUIVALENT OCTAL NUMBER.

$$0.6234 \times 8 = 4.9872 \quad 4$$

$$0.9872 \times 8 = 7.8976 \quad 7$$

$$0.8976 \times 8 = 7.1808 \quad 7$$

EXAMPLE: CONVERT 0.6234 DECIMAL NUMBER IN TO IT'S EQUIVALENT OCTAL NUMBER.

$$0.6234 \times 8 = 4.9872 \quad 4$$

$$0.9872 \times 8 = 7.8976 \quad 7$$

$$0.8976 \times 8 = 7.1808 \quad 7$$

$$0.1808 \times 8 = 1.4464 \quad 1$$

EXAMPLE: CONVERT 0.6234 DECIMAL NUMBER IN TO IT'S EQUIVALENT OCTAL NUMBER.

$$\begin{array}{rcl} 0.6234 \times 8 & = & 4.9872 \quad 4 \\ & & \text{---} \\ 0.9872 \times 8 & = & 7.8976 \quad 7 \\ & & \text{---} \\ 0.8976 \times 8 & = & 7.1808 \quad 7 \\ & & \text{---} \\ 0.1808 \times 8 & = & 1.4464 \quad 1 \\ & & \text{---} \\ 0.4464 \times 8 & = & 3.5712 \quad 3 \end{array}$$

EXAMPLE: CONVERT 0.6234 DECIMAL NUMBER IN TO IT'S EQUIVALENT OCTAL NUMBER.

$$\begin{array}{rcl} 0.6234 \times 8 & = & 4.9872 \quad 4 \quad \text{MSD} \\ & & \swarrow \\ 0.9872 \times 8 & = & 7.8976 \quad 7 \\ & & \swarrow \\ 0.8976 \times 8 & = & 7.1808 \quad 7 \\ & & \swarrow \\ 0.1808 \times 8 & = & 1.4464 \quad 1 \\ & & \swarrow \\ 0.4464 \times 8 & = & 3.5712 \quad 3 \quad \text{LSD} \end{array}$$

$$(0.6234)_{10} = (0.47713)_8$$

Exercise

- Convert following Decimal Numbers in to its equivalent Octal Number:

$$1. (1248.56)_{10} = (?)_8$$

$$2. (8957.75)_{10} = (?)_8$$

$$3. (420.6)_{10} = (?)_8$$

$$4. (8476.47)_{10} = (?)_8$$

Exercise

- Convert following Decimal Numbers in to its equivalent Octal Number:

$$1. \ (1248.56)_{10} = (2340.43656)_8$$

$$2. \ (8957.75)_{10} = (21375.6)_8$$

$$3. \ (420.6)_{10} = (644.463146)_8$$

$$4. \ (8476.47)_{10} = (20434.360507)_8$$



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



Switching Theory and Logic Design

ARI-209

By: Dr. Divya Agarwal



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT- II** 10 HOURS
- **Sequential Logic Circuits:** Latches and Flip Flops- SR, D, T and MS-JK Flip Flops, Asynchronous Inputs.
- **Counters and Shift Registers:** Design of Synchronous and Asynchronous Counters- Binary, BCD, Decade and Up/Down Counters, Shift Registers, Types of Shift Registers, Counters using Shift Registers- Ring Counter and Johnson Counter.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-III** **08 HOURS**
- **Integrated circuits:** TTL and CMOS logic families and their characteristics. Brief introduction to RAM and ROM
- **Synchronous Sequential Circuits:** State Tables State Equations and State Diagrams, State Reduction and State Assignment, Design of Clocked Sequential Circuits using State Equations.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-IV** 06 HOURS
- **Finite state machine:** capabilities and limitations, Mealy and Moore models-minimization of completely specified and incompletely specified sequential machines, Partition techniques and merger chart methods, concept of minimal cover table.
- **Algorithmic State Machine:** Representation of sequential circuits using ASM charts synthesis of output and next state functions, Data path control path partition-based design.



Text Books:

1. Leach and Malvino (2011). *Digital principles and Applications*. Tata McGraw-Hill Education
2. Mano, M. M. (2017). *Digital logic and computer design*. Pearson Education India.
3. Jain, R. P. (2003). *Modern digital electronics*. Tata McGraw-Hill Education.

Reference Books:

1. A Anand Kumar. (2016) *Fundamentals of Digital Logic Circuits*, PHI
2. Taub, H., & Schilling, D. L. (1977). *Digital integrated electronics*. McGraw-Hill College.



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.

How come hexadecimal number system be used instead of Binary number system for communicating with computer

- ✓ The hexadecimal number system provides a condensed way of representing large binary numbers stored and processed inside the computer.
- ✓ One such example is in representing addresses of different memory locations.
- ✓ Let us assume that a machine has 64K of memory. Such a memory has 64K ($= 2^{16} = 65\,536$) memory locations and needs 65 536 different addresses.
- ✓ These addresses can be designated as 0 to 65 535 in the decimal number system and 00000000 00000000 to 11111111 11111111 in the binary number system.
- ✓ The decimal number system is not used in computers and the binary notation here appears too cumbersome and inconvenient to handle.
- ✓ In the hexadecimal number system, 65 536 different addresses can be expressed with four digits from 0000 to FFFF. Similarly, the contents of the memory when represented in hexadecimal form are very convenient to handle.

Conversion of Decimal Number into Hexadecimal Number (Integer Number)

Procedure:

1. Divide the decimal no by the base 16, noting the remainder.
2. Continue to divide the quotient by 16 until there is nothing left, keeping the track of the remainders from each step.
3. List the remainder values in reverse order to find the number's hex equivalent

Example: Convert 2003 decimal number in to it's equivalent Hex number.

$$\begin{array}{r} 16 \mid 2003 \\ \hline \end{array}$$

EXAMPLE: CONVERT 2003 DECIMAL NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.

$$\begin{array}{r} 16 \mid 2003 \\ \hline 16 \quad 125 \\ \hline \end{array}$$

3

3

$$\begin{array}{r} 125 \\ \hline 16 \mid 2003 \\ - 16 \\ \hline 40 \\ - 32 \\ \hline 83 \\ - 80 \\ \hline 3 \end{array}$$

EXAMPLE: CONVERT 2003 DECIMAL NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.

$$\begin{array}{r} 16 \Big| 2003 \\ \hline 16 \quad 125 \\ \hline 16 \quad \textcircled{7} \end{array}$$

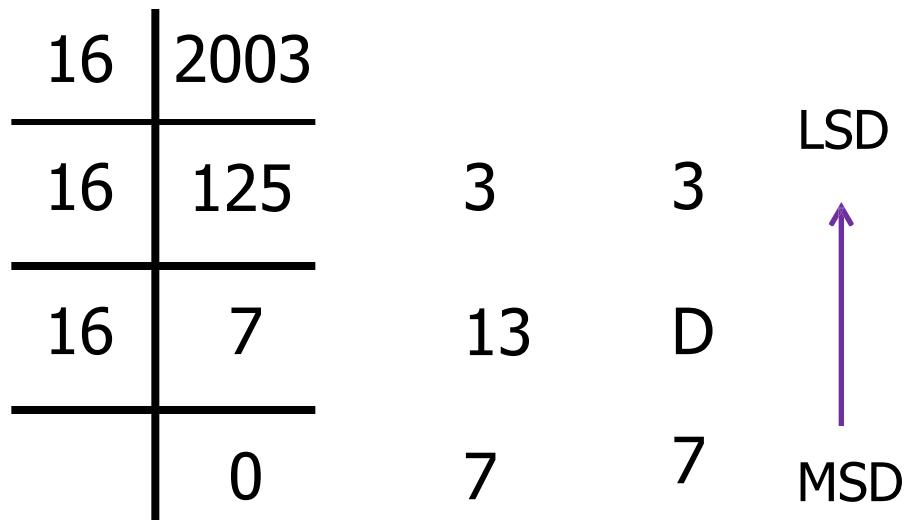
3 3
 \textcircled{13} D

$$\begin{array}{r} 16 \Big| 125 \\ \hline - 112 \\ \hline \textcircled{13} \end{array}$$

EXAMPLE: CONVERT 2003 DECIMAL NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.

16	2003		
16	125	3	3
16	7	13	D
16	0	7	7

EXAMPLE: CONVERT 2003 DECIMAL NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.



$$(2003)_{10} = (7D3)_{16}$$

CONVERSION OF DECIMAL NUMBER INTO HEXADECIMAL NUMBER (FRACTIONAL NUMBER)

Procedure:

1. Multiply the given fractional number by base 16.
2. Record the carry generated in this multiplication as MSD.
3. Multiply only the fractional number of the product in step 2 by 16 and record the carry as the next bit to MSD.
4. Repeat the steps 2 and 3 up to 5 bits. The last carry will represent the LSD of equivalent hex number

EXAMPLE: CONVERT 0.122 DECIMAL NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.

$$0.122 \times 16 = 1.952$$


1 1

EXAMPLE: CONVERT 0.122 DECIMAL NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.

$$\begin{array}{rcl} 0.122 \times 16 & = & 1.952 & 1 \\ & & \downarrow & \\ 0.952 \times 16 & = & 15.232 & 15 \\ & & \downarrow & \\ & & & F \end{array}$$

EXAMPLE: CONVERT 0.122 DECIMAL NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.

$$\begin{array}{rcl} 0.122 \times 16 & = & 1.952 & 1 & 1 \\ 0.952 \times 16 & = & 15.232 & 15 & F \\ 0.232 \times 16 & = & 3.712 & 3 & 3 \end{array}$$

EXAMPLE: CONVERT 0.122 DECIMAL NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.

$$\begin{array}{rcl} 0.122 \times 16 & = & 1.952 \quad 1 \quad 1 \\ 0.952 \times 16 & = & 15.232 \quad 15 \quad F \\ 0.232 \times 16 & = & 3.712 \quad 3 \quad 3 \\ 0.712 \times 16 & = & 11.392 \quad 11 \quad B \end{array}$$

EXAMPLE: CONVERT 0.122 DECIMAL NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.

$$\begin{array}{rcl} 0.122 \times 16 & = & 1.952 \quad 1 \quad 1 \\ 0.952 \times 16 & = & 15.232 \quad 15 \quad F \\ 0.232 \times 16 & = & 3.712 \quad 3 \quad 3 \\ 0.712 \times 16 & = & 11.392 \quad 11 \quad B \\ 0.392 \times 16 & = & 6.272 \quad 6 \end{array}$$

EXAMPLE: CONVERT 0.122 DECIMAL NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.

$0.122 \times 16 =$	1.952	1	1	MSD
$0.952 \times 16 =$	15.232	15	F	
$0.232 \times 16 =$	3.712	3	3	
$0.712 \times 16 =$	11.392	11	B	
$0.392 \times 16 =$	6.272	6	6	LSD

$$(0.122)_{10} = (0.1F3B6)_{16}$$

Exercise

- Convert following Decimal Numbers in to its equivalent Hex Number:

$$1. (1248.56)_{10} = (?)_{16}$$

$$2. (8957.75)_{10} = (?)_{16}$$

$$3. (420.6)_{10} = (?)_{16}$$

$$4. (8476.47)_{10} = (?)_{16}$$

Exercise

- Convert following Decimal Numbers in to its equivalent Hex Number:

$$1. \ (1248.56)_{10} = (4E0.8F5C28)_{16}$$

$$2. \ (8957.75)_{10} = (22FD.C)_{16}$$

$$3. \ (420.6)_{10} = (1A4.99999)_{16}$$

$$4. \ (8476.47)_{10} = (211C.7851E)_{16}$$

Conversion of Binary Number into Octal Number

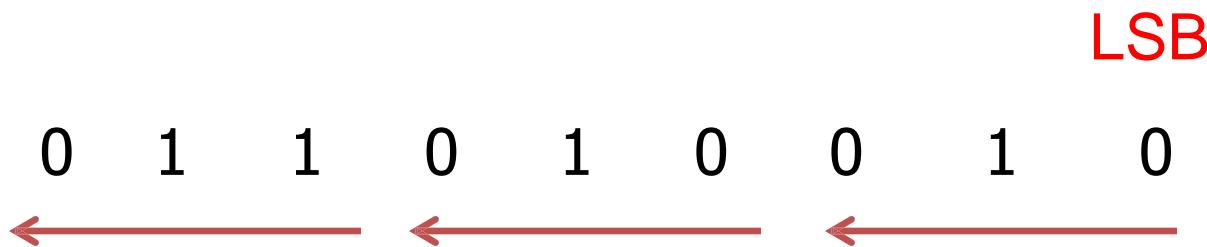
Procedure:

- 1. GROUP THE BINARY BITS INTO GROUPS OF 3 STARTING FROM LSB.**
- 2. Convert each group into its equivalent decimal. As the number of bits in each group is restricted to 3, the decimal number will be same as octal number**

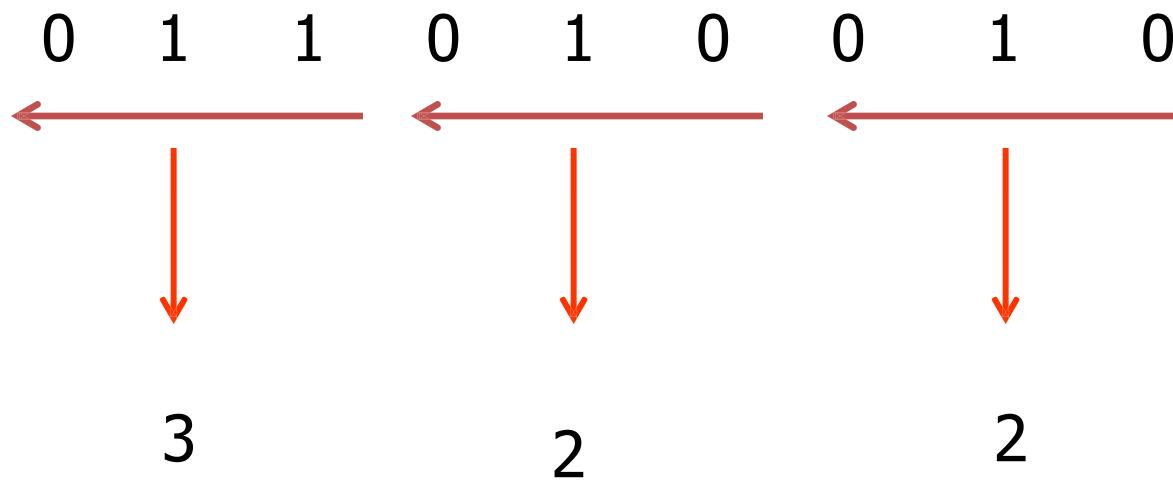
**EXAMPLE: CONVERT 11010010 BINARY
NUMBER IN TO IT'S EQUIVALENT OCTAL
NUMBER.**

0 1 1 0 1 0 0 1 0

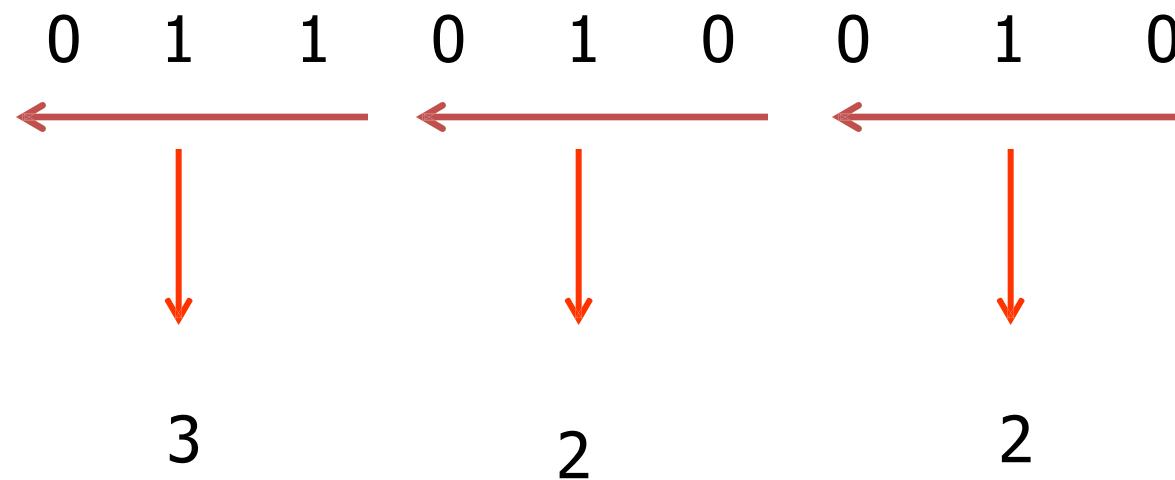
**EXAMPLE: CONVERT 11010010 BINARY NUMBER
IN TO IT'S EQUIVALENT OCTAL NUMBER.**



**EXAMPLE: CONVERT 11010010 BINARY NUMBER
IN TO IT'S EQUIVALENT OCTAL NUMBER.**



EXAMPLE: CONVERT 11010010 BINARY NUMBER IN TO IT'S EQUIVALENT OCTAL NUMBER.



$$(11010010)_2 = (322)_8$$

Exercise

- Convert following Binary equivalent

number into its Octal Number:

$$1. \ (1101110.011)_2 = (?)_8$$

$$2. \ (1101.11)_2 = (?)_8$$

$$3. \ (10001.01)_2 = (?)_8$$

Exercise

- Convert following Binary equivalent

number into its Octal Number:

$$1. \ (1101110.011)_2 = (156.3)_8$$

$$2. \ (1101.11)_2 = (15.6)_8$$

$$3. \ (10001.01)_2 = (21.2)_8$$

Conversion of Binary Number to Hexadecimal Number

Procedure:

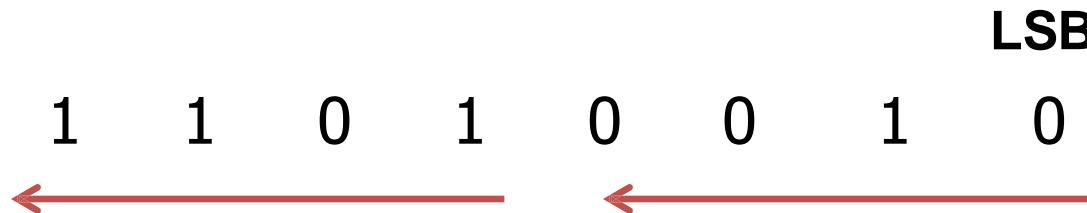
- 1. GROUP THE BINARY BITS INTO GROUPS OF 4 STARTING FROM LSB.**
- 2. Convert each group into its equivalent decimal. As the number of bits in each group is restricted to 4, the decimal number will be same as hex number**

EXAMPLE: CONVERT 11010010 BINARY NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.

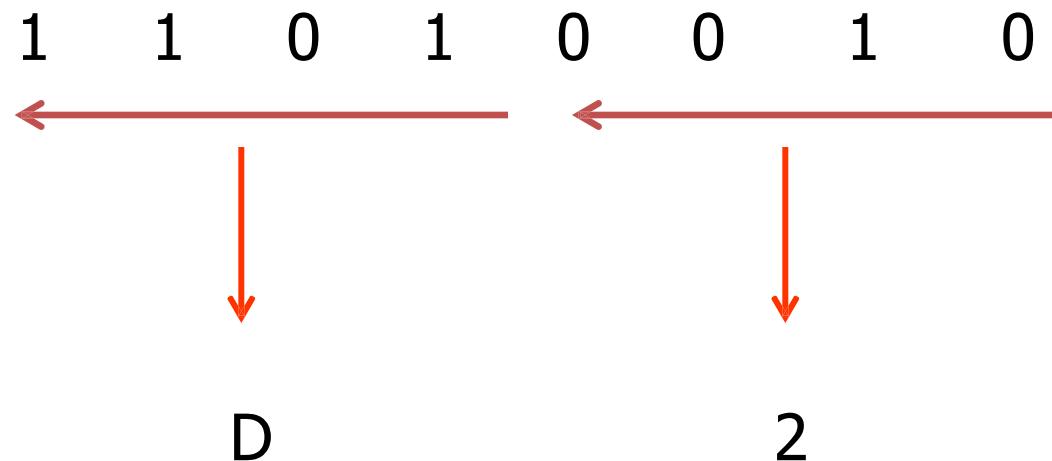
LSB

1	1	0	1	0	0	1	0
---	---	---	---	---	---	---	---

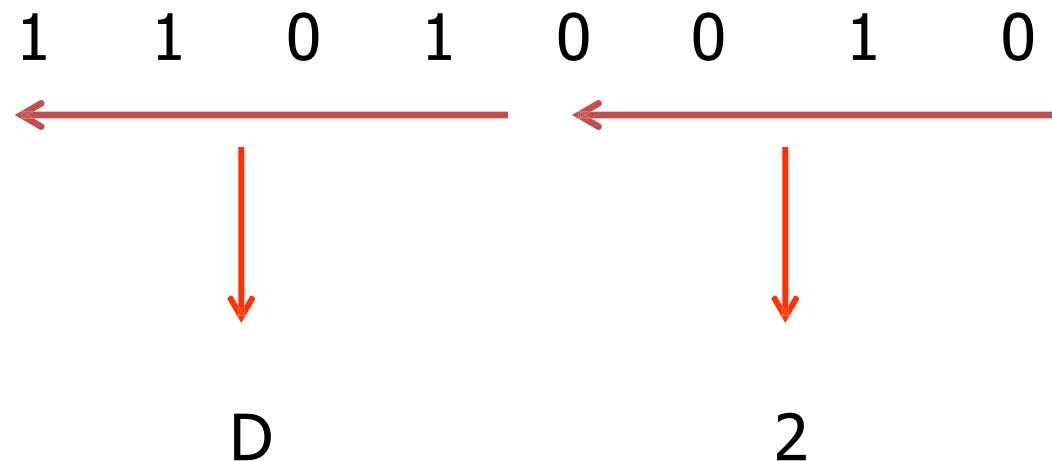
**EXAMPLE: CONVERT 11010010 BINARY NUMBER
IN TO IT'S EQUIVALENT HEX NUMBER.**



**EXAMPLE: CONVERT 11010010 BINARY NUMBER
IN TO IT'S EQUIVALENT HEX NUMBER.**



EXAMPLE: CONVERT 11010010 BINARY NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.



$$(11010010)_2 = (\text{D}2)_{16}$$

Exercise

- Convert following Binary Numbers into its equivalent Hexadecimal Number:

$$1. \ (1101110.011)_2 = (?)_{16}$$

$$2. \ (1101.11)_2 = (?)_{16}$$

$$3. \ (10001.01)_2 = (?)_{16}$$

Exercise

- Convert following Binary Numbers into its equivalent Hexadecimal Number:

$$1. \ (1101110.011)_2 = (6E.6)_{16}$$

$$2. \ (1101.11)_2 = (D.C)_{16}$$

$$3. \ (10001.01)_2 = (11.4)_{16}$$

Conversion of Octal Number into Binary Number

- ✓ To get the binary equivalent of the given octal number we have to convert each octal digit into its equivalent 3 bit binary number

EXAMPLE: CONVERT 364 OCTAL NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

3

6

4

EXAMPLE: CONVERT 364 OCTAL NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

3 6 4
↓ ↓ ↓
 100

EXAMPLE: CONVERT 364 OCTAL NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

3 6 4
↓ ↓ ↓
110 110 100

EXAMPLE: CONVERT 364 OCTAL NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

3	6	4
↓	↓	↓
011	110	100

EXAMPLE: CONVERT 364 OCTAL NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

3	6	4
↓	↓	↓
011	110	100

$$(364)_8 = (011110100)_2$$

OR

$$(364)_8 = (11110100)_2$$

Exercise

- Convert following Octal Numbers into its equivalent Binary Number:

$$1. \ (3006.05)_8 = (?)_2$$

$$2. \ (273.56)_8 = (?)_2$$

$$3. \ (6534.04)_8 = (?)_2$$

Exercise

- Convert following Octal Numbers into its equivalent Binary Number:

$$1. \ (3006.05)_8 = (11000000110.000101)_2$$

$$2. \ (273.56)_8 = (10111011.10111)_2$$

$$3. \ (6534.04)_8 = (110101011100.0001)_2$$

Conversion of Hexadecimal Number into Binary Number

- ✓ To get the binary equivalent of the given hex number we have to convert each hex digit into its equivalent 4 bit binary number

EXAMPLE: CONVERT AFB2 HEX NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

A

F

B

2

EXAMPLE: CONVERT AFB2 HEX NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

A



F



B



2



0010

EXAMPLE: CONVERT AFB2 HEX NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

A



F



B



2



1011

0010

EXAMPLE: CONVERT AFB2 HEX NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

A



F



1111

B



1011

2



0010

EXAMPLE: CONVERT AFB2 HEX NUMBER IN TO IT'S EQUIVALENT BINARY NUMBER.

A	F	B	2
↓	↓	↓	↓
1010	1111	1011	0010

$$(AFB2)_{16} = (101011110110010)_2$$

Exercise

- Convert following Hexadecimal Numbers in to its equivalent Binary Number:
 1. $(4056)_{16} = (?)_2$
 2. $(6B7)_{16} = (?)_2$
 3. $(8E47.AB)_{16} = (?)_2$

Exercise

- Convert following Hexadecimal Numbers in to its equivalent Binary Number:
 - $(4056)_{16} = (100000001010110)_2$
 - $(6B7)_{16} = (11010110111)_2$
 - $(8E47.AB)_{16} = (1000111001000111.10101011)_2$

Conversion of Octal Number into Hexadecimal Number

✓ To get hex equivalent number of given octal number, first we have to convert octal number into its 3 bit binary equivalent and then convert binary number into its hex equivalent.

EXAMPLE: CONVERT 364 OCTAL NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.

3 6 4 Octal Number

EXAMPLE: CONVERT 364 OCTAL NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.

3	6	4	Octal Number
↓	↓	↓	
			100
			Binary Number

EXAMPLE: CONVERT 364 OCTAL NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.

3	6	4	Octal Number
↓	↓	↓	
	110	100	Binary Number

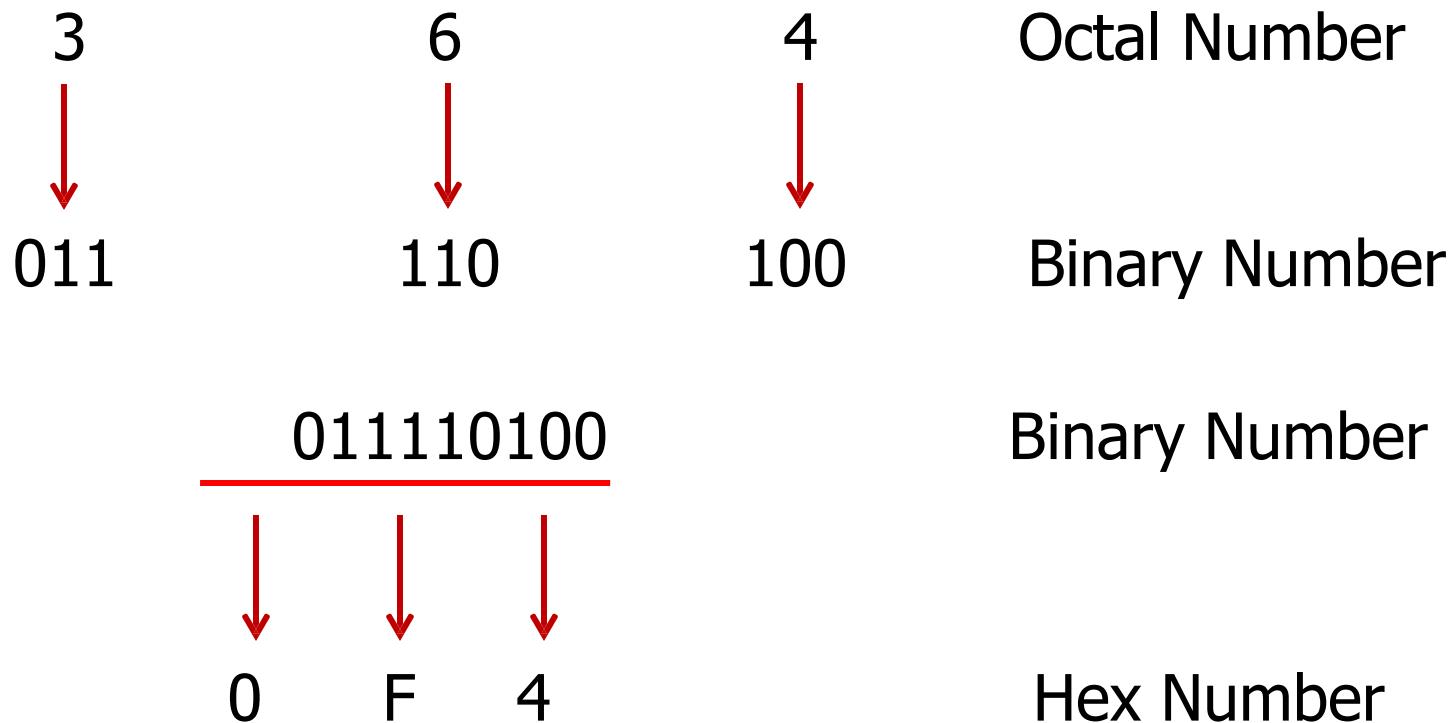
EXAMPLE: CONVERT 364 OCTAL NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.

3	6	4	Octal Number
↓	↓	↓	
011	110	100	Binary Number

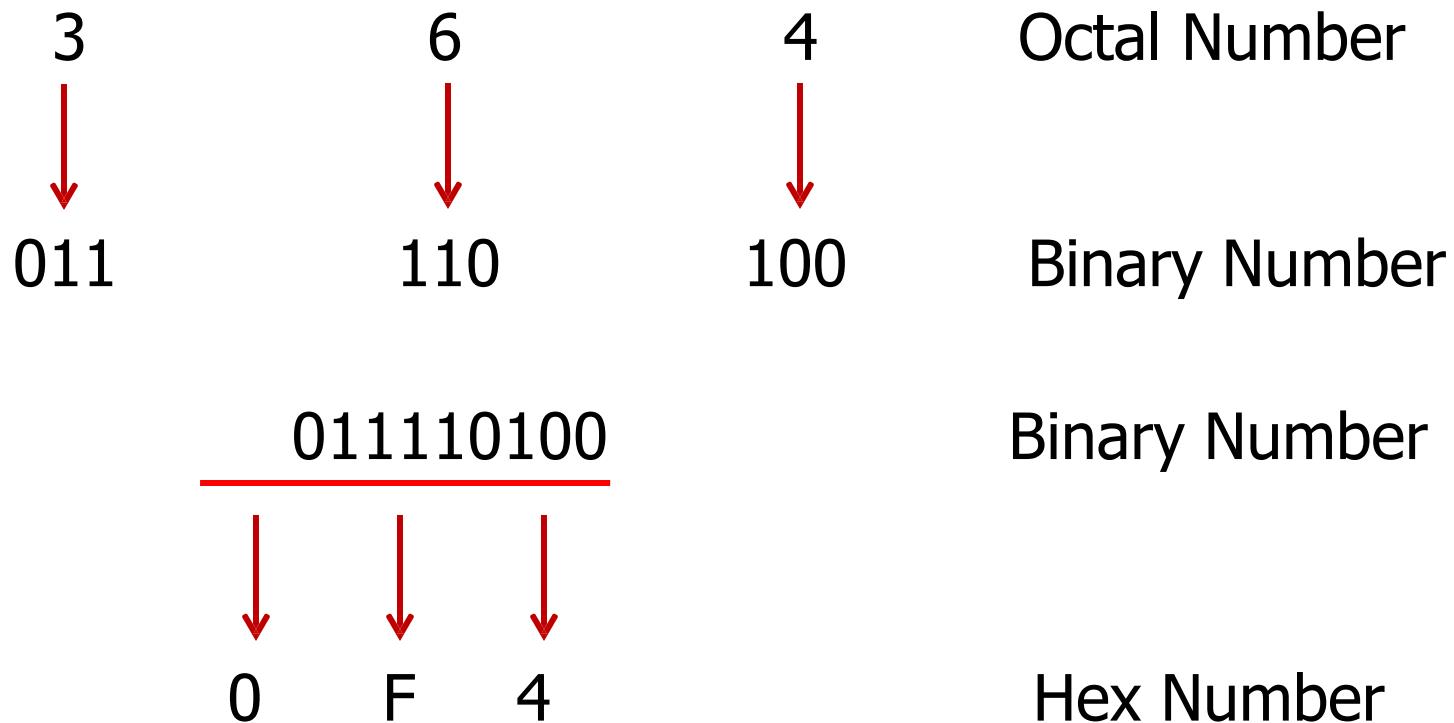
EXAMPLE: CONVERT 364 OCTAL NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.

3	6	4	Octal Number
↓	↓	↓	
011	110	100	Binary Number
011110100			Binary Number

EXAMPLE: CONVERT 364 OCTAL NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.



EXAMPLE: CONVERT 364 OCTAL NUMBER IN TO IT'S EQUIVALENT HEX NUMBER.



$$(364)_8 = (\text{F}4)_{16}$$

EXERCISE

- Convert following Octal Numbers in to its equivalent Hex Number:

$$1. \ (3006.05)_8 = (?)_{16}$$

$$2. \ (273.56)_8 = (?)_{16}$$

$$3. \ (6534.04)_8 = (?)_{16}$$

EXERCISE

- Convert following Octal Numbers in to its equivalent Hex Number:

$$1. \ (3006.05)_8 = (606.14)_{16}$$

$$2. \ (273.56)_8 = (BB.B8)_{16}$$

$$3. \ (6534.04)_8 = (D5C.1)_{16}$$

CONVERSION OF HEXADECIMAL NUMBER INTO OCTAL NUMBER

- ✓ To get octal equivalent number of given hex number, first we have to convert hex number into its 4 bit binary equivalent and then convert binary number into its octal equivalent.

EXAMPLE: CONVERT 4CA HEX NUMBER IN TO IT'S EQUIVALENT OCTAL NUMBER.

4	C	A	Hex Number
---	---	---	------------

EXAMPLE: CONVERT 4CA HEX NUMBER IN TO IT'S EQUIVALENT OCTAL NUMBER.

4	C	A	Hex Number
			
		1010	Binary Number

EXAMPLE: CONVERT 4CA HEX NUMBER IN TO IT'S EQUIVALENT OCTAL NUMBER.

4	C	A	Hex Number
↓	↓	↓	
	1100	1010	Binary Number

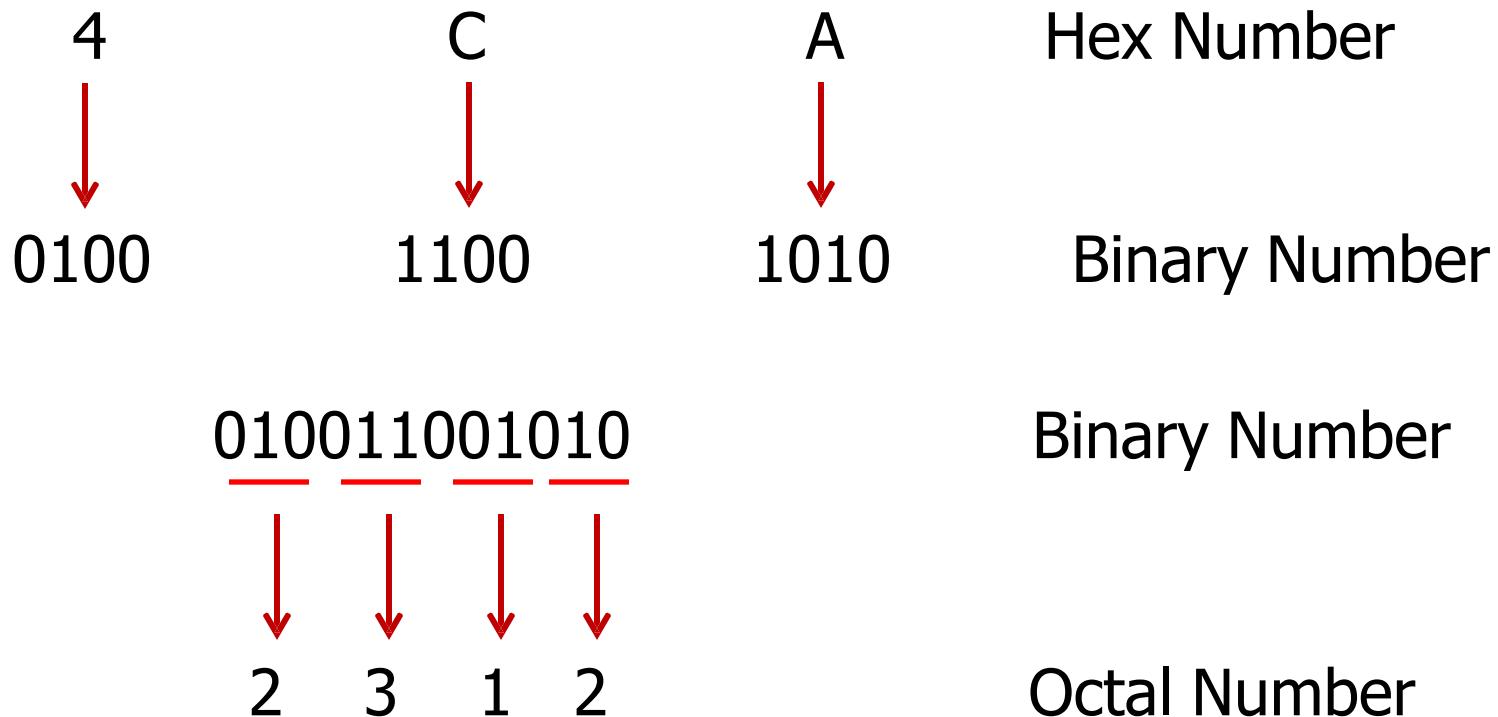
EXAMPLE: CONVERT 4CA HEX NUMBER IN TO IT'S EQUIVALENT OCTAL NUMBER.

Hex Number	Binary Number
4	0100
C	1100
A	1010

EXAMPLE: CONVERT 4CA HEX NUMBER IN TO IT'S EQUIVALENT OCTAL NUMBER.

Hex Number	Binary Number
4	0100
C	1100
A	1010
	010011001010
	Binary Number

EXAMPLE: CONVERT 4CA HEX NUMBER IN TO IT'S EQUIVALENT OCTAL NUMBER.



EXAMPLE: CONVERT 4CA HEX NUMBER IN TO IT'S EQUIVALENT OCTAL NUMBER.

4	C	A	Hex Number
0100	1100	1010	Binary Number

010011001010	Binary Number
<u> </u> <u> </u> <u> </u> <u> </u>	
2 3 1 2	Octal Number

$$(4CA)_{16} = (2312)_8$$

Exercise

- Convert following Hexadecimal Numbers in to its equivalent Octal Number:
 1. $(4056)_{16} = (?)_8$
 2. $(6B7)_{16} = (?)_8$
 3. $(8E47.AB)_{16} = (?)_8$

Exercise

- Convert following Hexadecimal Numbers in to its equivalent Octal Number:
 1. $(4056)_{16} = (40126)_8$
 2. $(6B7)_{16} = (3267)_8$
 3. $(8E47.AB)_{16} = (107107.526)_8$

CONVERSION OF ANY RADIX NUMBER INTO DECIMAL NUMBER

- ✓ To get decimal equivalent number of any radix/base number, we have to :
 1. Write down the any radix number.
 2. Write down the weights for different positions.
 3. Multiply each digit in the any radix number with the corresponding weight to obtain product numbers to get the decimal numbers.
 4. Add all the product numbers to get the decimal equivalent

Example: Convert 231.23 4 radix number in to it's equivalent decimal number.

Base 4 No.

2 3 1 • 2 3

Example: Convert 231.23 4 radix number in to it's equivalent decimal number.

Base 4 No.

2 3 1 • 2 3
↑ ↑ ↑ ↑ ↑

Positional Weights

4^2 4^1 4^0 4^{-1} 4^{-2}

Example: Convert 231.23 4 radix number in to it's equivalent decimal number.

Base 4 No.

2 3 1 • 2 3
↑ ↑ ↑ ↑ ↑

Positional Weights

4^2 4^1 4^0 4^{-1} 4^{-2}

$$= (2*4^2) + (3*4^1) + (1*4^0) \cdot (2*4^{-1}) + (3*4^{-2})$$

Example: Convert 231.23 (four) 4 radix number in to it's equivalent decimal number.

Base 4 No.

2 3 1 • 2 3
↑ ↑ ↑ ↑ ↑

Positional Weights

4^2 4^1 4^0 4^{-1} 4^{-2}

$$== (2*4^2) + (3*4^1) + (1*4^0). (2*4^{-1}) + (3*4^{-2})$$

$$= \quad 32 \quad + \quad 12 \quad + \quad 1 \quad . \quad 0.5 \quad + 0.1875$$

$$= 48.6875$$

Example: Convert 231.23 (four) 4 radix number in to it's equivalent decimal number.

Base 4 No.

2 3 1 • 2 3
↑ ↑ ↑ ↑ ↑

Positional Weights

4^2 4^1 4^0 4^{-1} 4^{-2}

$$= (2*4^2) + (3*4^1) + (1*4^0). (2*4^{-1}) + (3*4^{-2})$$

$$= 32 + 12 + 1 . 0.5 + 0.1875$$

$$= 48.6875$$

$$(231.23)_4 = (48.6875)_{10}$$

Exercise

- Convert following any radix Numbers in to its equivalent Decimal Number:
 1. $(48A6)_{12} = (?)_{10}$
 2. $(614.15)_7 = (?)_{10}$
 3. $(8C47.AB)_{13} = (?)_{10}$

Exercise

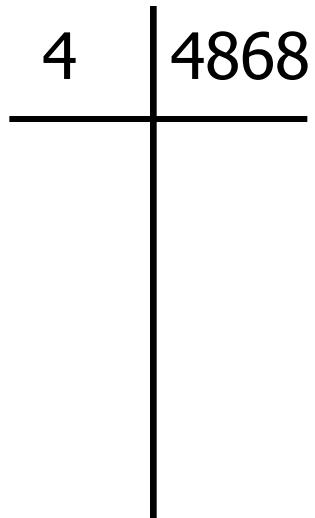
- Convert following any radix Numbers in to its equivalent Decimal Number:
 1. $(48A6)_{12} = (8190)_{10}$
 2. $(614.15)_7 = (305.24489)_{10}$
 3. $(8C47.AB)_{13} = (19663.834319)_{10}$

Conversion of Decimal Number into Any Radix Number (Integer Number)

Procedure:

1. Divide the decimal no by the radix number, noting the remainder.
2. Continue to divide the quotient by radix number until there is nothing left, keeping the track of the remainders from each step.
3. List the remainder values in reverse order to find the number's decimal equivalent

Example: Convert 4868 decimal number in to it's equivalent **four radix** number.



EXAMPLE: CONVERT 4868 DECIMAL NUMBER IN TO IT'S EQUIVALENT FOUR RADIX NUMBER.

$$\begin{array}{r} 4 | 4868 \\ \hline 4 | 1217 \quad 0 \end{array}$$

$$\begin{array}{r} 1217 \\ \hline 4 | 4868 \\ - 4 \\ \hline 8 \\ - 8 \\ \hline 6 \\ - 4 \\ \hline 8 \\ - 8 \\ \hline 0 \end{array}$$

EXAMPLE: CONVERT 4868 DECIMAL NUMBER IN TO IT'S EQUIVALENT FOUR RADIX NUMBER.

$$\begin{array}{r} 4 | 4868 \\ \hline 4 | 1217 \\ \hline 4 | 304 \end{array} \quad \begin{array}{l} 0 \\ 1 \end{array}$$

$$\begin{array}{r} 304 \\ \hline 4 | 1217 \\ - 12 \\ \hline 17 \\ - 16 \\ \hline 1 \end{array}$$

**EXAMPLE: CONVERT 4868 DECIMAL NUMBER IN
TO IT'S EQUIVALENT **FOUR RADIX** NUMBER.**

$$\begin{array}{r} 4 | 4868 \\ \hline 4 | 1217 & 0 \\ \hline 4 | 304 & 1 \\ \hline 76 & 0 \end{array}$$

**EXAMPLE: CONVERT 4868 DECIMAL NUMBER IN
TO IT'S EQUIVALENT **FOUR RADIX** NUMBER.**

$$\begin{array}{r|l} 4 & 4868 \\ \hline 4 & 1217 \quad 0 \\ \hline 4 & 304 \quad 1 \\ \hline 4 & 76 \quad 0 \\ \hline & 19 \quad 0 \end{array}$$

**EXAMPLE: CONVERT 4868 DECIMAL NUMBER IN
TO IT'S EQUIVALENT **FOUR RADIX** NUMBER.**

$$\begin{array}{r|l} 4 & 4868 \\ \hline 4 & 1217 \quad 0 \\ \hline 4 & 304 \quad 1 \\ \hline 4 & 76 \quad 0 \\ \hline 4 & 19 \quad 0 \\ \hline & 4 \quad 3 \end{array}$$

**EXAMPLE: CONVERT 4868 DECIMAL NUMBER IN
TO IT'S EQUIVALENT **FOUR RADIX** NUMBER.**

$$\begin{array}{r|l} 4 & 4868 \\ \hline 4 & 1217 & 0 \\ \hline 4 & 304 & 1 \\ \hline 4 & 76 & 0 \\ \hline 4 & 19 & 0 \\ \hline 4 & 4 & 3 \\ \hline & 1 & 0 \\ & & 1 \end{array}$$

EXAMPLE: CONVERT 4868 DECIMAL NUMBER IN TO IT'S EQUIVALENT **FOUR RADIX** NUMBER.

4	4868
4	1217
4	304
4	76
4	19
4	4
	1

0 LSD
1
0
0
0
3
0
1 MSD

$$(4868)_{10} = (1030010)_4$$

CONVERSION OF DECIMAL NUMBER INTO ANY RADIX NUMBER (FRACTIONAL NUMBER)

Procedure:

1. Multiply the given fractional number by radix number.
2. Record the carry generated in this multiplication as MSD.
3. Multiply only the fractional number of the product in step 2 by radix number and record the carry as the next bit to MSD.
4. Repeat the steps 2 and 3 up to 5 bits. The last carry will represent the LSD of equivalent radix number

EXAMPLE: CONVERT 0.6875 DECIMAL NUMBER IN TO IT'S EQUIVALENT FOUR RADIX NUMBER.

$$0.6875 \times 4 = 2.75 \quad 2$$


EXAMPLE: CONVERT 0.6875 DECIMAL NUMBER IN TO IT'S EQUIVALENT FOUR RADIX NUMBER.

$$\begin{array}{rcl} 0.6875 \times 4 & = & 2.75 \quad 2 \\ & & \swarrow \\ 0.75 \times 4 & = & 3.0 \quad 3 \end{array}$$

EXAMPLE: CONVERT 0.6875 DECIMAL NUMBER IN TO IT'S EQUIVALENT FOUR RADIX NUMBER.

$$\begin{array}{rcl} 0.6875 \times 4 & = & 2.75 \quad 2 \\ 0.75 \times 4 & = & 3.0 \quad 3 \end{array}$$

↓

MSD

LSD

$$(0.6875)_{10} = (0.23)_4$$

Exercise

- Convert following any radix Numbers in to its equivalent Decimal Number:
 1. $(8190)_{10} = (?)_{12}$
 2. $(305.24489)_{10} = (?)_7$
 3. $(63.8343)_{10} = (?)_{13}$

Exercise

- Convert following any radix Numbers in to its equivalent Decimal Number:
 1. $(8190)_{10} = (48A6)_{12}$
 2. $(305.24489)_{10} = (614.1466)_7$
 3. $(63.8343)_{10} = (4B.AACC5)_{13}$

Binary Addition

□ Column Addition

$$\begin{array}{r} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & = 61 \\ + & 1 & 0 & 1 & 1 & 1 & = 23 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 0 & = 84 \end{array}$$

$\geq (2)_{10}$





BINARY ADDITION

- Following are the four most basic cases for binary addition

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10 \text{ i.e. } 0 \text{ with carry } 1$$

Binary Addition

Example: Perform $(10111)_2 + (11001)_2$

BINARY ADDITION

Example: Perform $(10111)_2 + (11001)_2$

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

BINARY ADDITION

Example: Perform $(10111)_2 + (11001)_2$

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

BINARY ADDITION

Example: Perform $(10111)_2 + (11001)_2$

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

BINARY ADDITION

Example: Perform $(10111)_2 + (11001)_2$

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

BINARY ADDITION

Example: Perform $(10111)_2 + (11001)_2$

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

BINARY ADDITION

Example: Perform $(10111)_2 + (11001)_2$

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

$$(10111)_2 + (11001)_2 = (110000)_2$$

Binary Addition

Example: Perform $(1101.101)_2 + (111.011)_2$

BINARY ADDITION

Example: Perform $(1101.101)_2 + (111.011)_2$

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

$$(1101.101)_2 + (111.011)_2 = (10101.000)_2$$

Exercise

- Perform Binary Addition of following:

$$1. \ (11011)_2 + (1101)_2$$

$$2. \ (1011)_2 + (1101)_2 + (1001)_2 + (1111)_2$$

$$3. \ (1010.11)_2 + (1101.10)_2 + (1001.11)_2 + (1111.11)_2$$

$$4. \ (10111.101)_2 + (110111.01)_2$$

Exercise

- Perform Binary Addition of following:

$$1. \ (11011)_2 + (1101)_2 = \mathbf{101000}$$

$$2. \ (1011)_2 + (1101)_2 + (1001)_2 + (1111)_2 = \mathbf{110000}$$

$$3. \ (1010.11)_2 + (1101.10)_2 + (1001.11)_2 + (1111.11)_2$$

$$= \mathbf{110001.11}$$

$$4. \ (10111.101)_2 + (110111.01)_2 = \mathbf{1001110.110}$$

Binary Subtraction

- Borrow a “Base” when needed

$$\begin{array}{r} & \overset{1}{\cancel{0}} & & \overset{2}{\cancel{0}} & & & \\ & 0 & \cancel{2} & 2 & 0 & 0 & 2 \\ -1 & 0 & 0 & \cancel{1} & \cancel{1} & 0 & 1 \\ - & & & 1 & 0 & 1 & 1 & 1 \\ \hline & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ & & & & & & & = 54 \end{array}$$

$= (10)_2$



BINARY SUBTRACTION

- Following are the four most basic cases for binary subtraction

Subtraction	Borrow
0 - 0 = 0	0
0 - 1 = 1	1
1 - 0 = 1	0
1 - 1 = 0	0

Binary Subtraction

Example: Perform $(1010.010)_2 - (111.111)_2$

BINARY SUBTRACTION

Example: Perform $(1010.010)_2 - (111.111)_2$

$$\begin{array}{r} & \overset{1}{\cancel{1}} & \overset{1}{\cancel{0}} & \overset{1}{\cancel{1}} \\ 0 & \overset{10}{\cancel{10}} & 0 & \overset{10}{\cancel{10}} & & & \overset{1}{\cancel{10}} & \overset{10}{\cancel{0}} & \overset{10}{\cancel{10}} \\ 1 & 0 & 1 & 0 & . & 0 & 1 & 0 \\ \hline - & & & & . & 1 & 1 & 1 \\ \hline & 0 & 0 & 1 & 0 & . & 0 & 1 & 1 \end{array}$$

$$(1010.010)_2 - (111.111)_2 = (0010.011)_2$$

Exercise

- Perform Binary Subtraction of following:
 1. $(1011)_2 - (101)_2$
 2. $(1100.10)_2 - (111.01)_2$
 3. $(10110)_2 - (1011)_2$
 4. $(10001.01)_2 - (1111.11)_2$

Exercise

- Perform Binary Subtraction of following:

$$1. \ (1011)_2 - (101)_2 = \mathbf{110}$$

$$2. \ (1100.10)_2 - (111.01)_2 = \mathbf{101.01}$$

$$3. \ (10110)_2 - (1011)_2 = \mathbf{1011}$$

$$4. \ (10001.01)_2 - (1111.11)_2 = \mathbf{1.10}$$

BINARY MULTIPLICATION

- Following are the four most basic cases for binary multiplication

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

Binary Multiplication

- Bit by bit

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



BINARY MULTIPLICATION

Example: Perform $(1001)_2 * (1000)_2$

BINARY MULTIPLICATION

Example: Perform $(1001)_2 * (1000)_2$

$$\begin{array}{r} 1 \quad 0 \quad 0 \quad 1 \\ \times \quad 1 \quad 0 \quad 0 \quad 0 \\ \hline 0 \quad 0 \quad 0 \quad 0 \quad 0 \\ + \quad 0 \quad 0 \quad 0 \quad 0 \quad x \\ 0 \quad 0 \quad 0 \quad 1 \quad x \quad x \\ \hline 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \end{array}$$

$$(1001)_2 * (1000)_2 = (1001000)_2$$

Exercise

- Perform Binary Multiplication of following:
 1. $(1101)_2 \times (101)_2$
 2. $(1101.11)_2 \times (101.1)_2$
 3. $(11001)_2 \times (10)_2$
 4. $(10110)_2 \times (10.1)_2$

Exercise

- Perform Binary Multiplication of following:
 1. $(1101)_2 \times (101)_2 = \textcolor{red}{1000001}$
 2. $(1101.11)_2 \times (101.1)_2 = \textcolor{red}{11111.101}$
 3. $(11001)_2 \times (10)_2 = \textcolor{red}{110010}$
 4. $(10110)_2 \times (10.1)_2 = \textcolor{red}{0110111.0}$

BINARY DIVISION

Example: Perform $(110110)_2 / (101)_2$

BINARY DIVISION

Example: Perform $(110110)_2 / (101)_2$

$$\begin{array}{r} 1 \ 0 \ 1 \ 0 \\ \hline 101 \overline{)1 \ 1 \ 0 \ 1 \ 1 \ 0} \\ -1 \ 0 \ 1 \\ \hline 0 \ 0 \ 1 \ 1 \\ -0 \ 0 \\ \hline 1 \ 1 \ 1 \\ -1 \ 0 \ 1 \\ \hline 0 \ 1 \ 0 \ 0 \\ -0 \ 0 \ 0 \\ \hline 1 \ 0 \ 0 \end{array}$$

Exercise

- Perform Binary Division of following:
 1. $(1010)_2$ by $(11)_2$
 2. $(11110)_2$ by $(101)_2$
 3. $(11011)_2$ by $(101)_2$
 4. $(1101111)_2$ by $(101)_2$

Exercise

- Perform Binary Division of following:

1. $(1010)_2$ by $(11)_2 = \text{011 Remainder : 1}$
2. $(11110)_2$ by $(101)_2 = \text{0110}$
3. $(11011)_2$ by $(10)_2 = \text{01101 Remainder : 1}$
4. $(1101111)_2$ by $(101)_2 = \text{010110 Remainder : 1}$



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



Switching Theory and Logic Design

ARI-209

By: Dr. Divya Agarwal



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT- II** 10 HOURS
- **Sequential Logic Circuits:** Latches and Flip Flops- SR, D, T and MS-JK Flip Flops, Asynchronous Inputs.
- **Counters and Shift Registers:** Design of Synchronous and Asynchronous Counters- Binary, BCD, Decade and Up/Down Counters, Shift Registers, Types of Shift Registers, Counters using Shift Registers- Ring Counter and Johnson Counter.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAJAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-III** **08 HOURS**
- **Integrated circuits:** TTL and CMOS logic families and their characteristics. Brief introduction to RAM and ROM
- **Synchronous Sequential Circuits:** State Tables State Equations and State Diagrams, State Reduction and State Assignment, Design of Clocked Sequential Circuits using State Equations.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-IV** 06 HOURS
- **Finite state machine:** capabilities and limitations, Mealy and Moore models-minimization of completely specified and incompletely specified sequential machines, Partition techniques and merger chart methods, concept of minimal cover table.
- **Algorithmic State Machine:** Representation of sequential circuits using ASM charts synthesis of output and next state functions, Data path control path partition-based design.



Text Books:

1. Leach and Malvino (2011). *Digital principles and Applications*. Tata McGraw-Hill Education
2. Mano, M. M. (2017). *Digital logic and computer design*. Pearson Education India.
3. Jain, R. P. (2003). *Modern digital electronics*. Tata McGraw-Hill Education.

Reference Books:

1. A Anand Kumar. (2016) *Fundamentals of Digital Logic Circuits*, PHI
2. Taub, H., & Schilling, D. L. (1977). *Digital integrated electronics*. McGraw-Hill College.



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.

Complements

- There are two types of complements for each base- r system: the radix complement and diminished radix complement.

- **Diminished Radix Complement - $(r-1)$'s Complement**

- ◆ Given a number N in base r having n digits, the $(r-1)$'s complement of N is defined as:

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

- **Example for 6-digit decimal numbers:**

- ◆ 9's complement is $(r^n - 1) - N = (10^6 - 1) - N = 999999 - N$
 - ◆ 9's complement of 546700 is $999999 - 546700 = 453299$

- **Example for 7-digit binary numbers:**

- ◆ 1's complement is $(r^n - 1) - N = (2^7 - 1) - N = 1111111 - N$
 - ◆ 1's complement of 1011000 is $1111111 - 1011000 = 0100111$

- **Observation:**

- ◆ Subtraction from $(r^n - 1)$ will never require a borrow
 - ◆ Diminished radix complement can be computed digit-by-digit
 - ◆ For binary: $1 - 0 = 1$ and $1 - 1 = 0$

Complements

□ Example 1

- ◆ Using 10's complement, subtract $72532 - 3250$.

$$\begin{array}{rcl} M & = & 72532 \\ \text{10's complement of } N & = & +\underline{96750} \\ \text{Sum} & = & 169282 \\ \text{Discard end carry } 10^5 & = & -\underline{100000} \\ \text{Answer} & = & 69282 \end{array}$$

□ Example 2

- ◆ Using 10's complement, subtract $3250 - 72532$.

$$\begin{array}{rcl} M & = & 03250 \\ \text{10's complement of } N & = & +\underline{27468} \\ \text{Sum} & = & 30718 \end{array}$$



There is no end carry.



Therefore, the answer is $-(10\text{'s complement of } 30718) = -69282$.

1's Complement

- The 1's complement of a number is obtained by simply complementing each bit of the number that is by changing all 0's to 1's and all 1's to 0's.
- This system is called as 1's complement because the number can be subtracted from 1 to obtain result

1'S COMPLEMENT

Example: Obtain 1's complement of the 1010

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

1's complement of the 1010 is 0101

1'S COMPLEMENT

Sr. No.	Binary Number	1's Complement
1	1101 0101	0010 1010
2	1001	0110
3	1011 1111	0100 0000
4	1101 1010 0001	0010 0101 1110
5	1110 0111 0101	0001 1000 1010
6	1011 0100 1001	0100 1011 0110
7	1100 0011 0010	0011 1100 1101
8	0001 0010 1000	1110 1101 0111

SUBTRACTION USING 1'S COMPLEMENT

- In 1's complement subtraction, add the 1's complement of subtrahend to the minuend.
- If there is carry out, bring the carry around and add it to LSB.
- Look at the sing bit (MSB), if this is 0, the result is positive and is in its true binary form.
- If the MSB is 1(whether there is a carry or no carry at all), the result is negative & is in its 1's complement form. So take 1's complement to obtain result.

SUBTRACTION USING 1'S COMPLEMENT

Example: Perform using 1' complement $(9)_{10} - (4)_{10}$

SUBTRACTION USING 1'S COMPLEMENT

Example: Perform using 1' complement $(9)_{10} - (4)_{10}$

Step 1: Take 1' complement of $(4)_{10} = (0100)_2$
 $= 1011$

Step 2: Add 9 with 1' complement of 4

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

final carry → 1 Result

Step 3: If carry is generated add final carry to the result

EXAMPLE

Continue

$$\begin{array}{r} & 1 & 0 & 0 & 1 \\ + & 1 & 0 & 1 & 1 \\ \hline & 1 & 0 & 1 & 0 & 0 & \text{Result} \\ \text{final carry} \longrightarrow & & & & & & \\ & \downarrow & & & & & \\ & & & & & & \longrightarrow 1 \\ \hline & 0 & 1 & 0 & 1 & & \text{Final Result} \end{array}$$

When the final carry is produced the answer is positive and is in its true binary form

Exercise

- Perform Binary Subtraction using 1's Complement method

$$1. \ (52)_{10} - (17)_{10}$$

$$2. \ (84)_{10} - (46)_{10}$$

$$3. \ (63.75)_{10} - (17.50)_{10}$$

Exercise

- Perform Binary Subtraction using 1's Complement method

$$1. \ (52)_{10} - (17)_{10} = \mathbf{0100011} = (35)_{10}$$

$$2. \ (84)_{10} - (46)_{10} = \mathbf{0100110} = (38)_{10}$$

$$3. \ (63.75)_{10} - (17.50)_{10} = \mathbf{101110.01} = (46.25)_{10}$$

2'S COMPLEMENT

- ✓ The 2's complement of a number is obtained by adding 1 to the 1's complement of that number

2's Complement

Example: Obtain 2's complement of the 1010

2'S COMPLEMENT

Example: Obtain 2's complement of the 1010

$$\begin{array}{r} 1 & 1 & 1 & 1 \\ \underline{-} \\ 1 & 0 & 1 & 0 \\ \hline + & 0 & 1 & 0 & 1 & \text{---1's complement} \\ \hline & 0 & 1 & 1 & 0 & \text{----2's complement} \end{array}$$

2's complement of the 1010 is 0110

Complements

□ Example 3

- Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction (a) $X - Y$; and (b) $Y - X$, by using 2's complement.

(a)

$$\begin{array}{r} X = 1010100 \\ 2\text{'s complement of } Y = +0111101 \\ \hline \text{Sum} = 10010001 \\ \text{Discard end carry } 2^7 = -10000000 \\ \hline \text{Answer. } X - Y = 0010001 \end{array}$$

(b)

$$\begin{array}{r} Y = 1000011 \\ 2\text{'s complement of } X = +0101100 \\ \hline \text{Sum} = 1101111 \end{array}$$



There is no end carry.
Therefore, the answer is $Y - X = -(2\text{'s complement of } 1101111) = -0010001$.

2'S COMPLEMENT

Sr. No.	Binary Number	1's Complement	2's Complement
1	1101 0101	0010 1010	0010 1011
2	1001	0110	0111
3	1011 1111	0100 0000	0100 0001
4	1101 1010 0001	0010 0101 1110	0010 0101 1111
5	1110 0111 0101	0001 1000 1010	0001 1000 1011

SUBTRACTION USING 2'S COMPLEMENT

- ✓ In 2's complement subtraction, add the 2's complement of subtrahend to the minuend.
- ✓ If carry is generated then the result is positive and in its true form.
- ✓ If the carry is not produced, then the result is negative and in its 2's complement form.

***Carry is always to be discarded**

SUBTRACTION USING 2'S COMPLEMENT

Example: Perform using 2' complement $(9)_{10} - (4)_{10}$

SUBTRACTION USING 2'S COMPLEMENT

Example: Perform using 2' complement $(9)_{10} - (4)_{10}$

Step 1: Take 2' complement of $(4)_{10} = (0100)_2$

$$= 1011 + 1 = 1100$$

Step 2: Add 9 with 2' complement of 4

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

final carry
Discard

Final Result

If Carry is generated, discard carry. The result is positive and its true binary form

Exercise

- Perform Binary Subtraction using 2's Complement method

$$1. (46)_{10} - (19)_{10}$$

$$2. (27)_{10} - (75)_{10}$$

$$3. (125.3)_{10} - (46.7)_{10}$$

$$4. (36.75)_{10} - (89.5)_{10}$$

Exercise

- Perform Binary Subtraction using 2's Complement method

$$1. (46)_{10} - (19)_{10} = \textcolor{red}{011011}$$

$$2. (27)_{10} - (75)_{10} = \textcolor{red}{-0110000}$$

$$3. (125.3)_{10} - (46.7)_{10} = \textcolor{red}{1001110.10011}$$

$$4. (36.75)_{10} - (89.5)_{10} = \textcolor{red}{-0110100.11}$$

BCD OR 8421 CODE

- ✓ The smallest BCD number is (0000) and the largest is (1001). The next number to 9 will be 10 which is expressed as (0001 0000) in BCD.
- ✓ There are six illegal combinations 1010, 1011, 1100, 1101, 1110 and 1111 in this code i.e. they are not part of the 8421 BCD code

DECIMAL TO BCD CONVERSION

Sr. No.	Decimal Number	BCD Code
1	8	1000
2	47	0100 0111
3	345	0011 0100 0101
4	99	1001 1001
5	10	0001 0000

GRAY CODE

- ✓ The gray code is non-weighted code.
- ✓ It is not suitable for arithmetic operations.
- ✓ It is a cyclic code because successive code words in this code differ in one bit position only i.e. unit distance code
- ✓ This code is also called reflected code.
- ✓ In general the n least significant bits for 2^n to $2^{n+1} - 1$ are the mirror images of those of 0 to $2^n - 1$
- ✓ The gray coded no corresponding to the decimal no $2^n - 1$ for any n differs from gray coded 0 in one bit position only.

Ex: If n = 2, 3, 4 etc. See that

$$2^2 - 1 = 3 - 1 = 3_{10} = 0010 \text{ in gray}$$

$$\cdot \quad 2^3 - 1 = 7_{10} = 0100$$

$$2^4 - 1 = 15_{10} = 1000$$

GRAY CODE

Decimal

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

Gray code.

0000

0001

0011

0010

0110

0111

0101

0100

1100

1101

1111

1110

1010

1011

1001

1000

BINARY TO GRAY CODE CONVERSION

- ✓ If an n bit binary number is represented by B_n, B_{n-1}, \dots, B_1 and its gray code equivalent by G_n, G_{n-1}, \dots, G_1 then gray code bits are obtained from the binary code as follows;

$G_n = B_n$	$G_{n-1} = B_n \oplus B_{n-1}$	$G_{n-2} = B_{n-1} \oplus B_{n-2}$	$G_1 = B_2 \oplus B_1$
-------------	--------------------------------	------------------------------------	-------	------------------------

where B_n and G_n are the MSBs,

*where the symbol represents Exclusive-OR operation

Binary to Gray Code Conversion

Example 1: Convert 1011 Binary Number into Gray Code

BINARY TO GRAY CODE CONVERSION

Example 1: Convert 1011 Binary Number into Gray Code

Binary Number

1

0

1

1

Example 1:

CONTINUE

Binary Number

1

0

1

1



Gray Code

1

EXAMPLE 1:

Continue

Binary Number

1 →⊕← 0

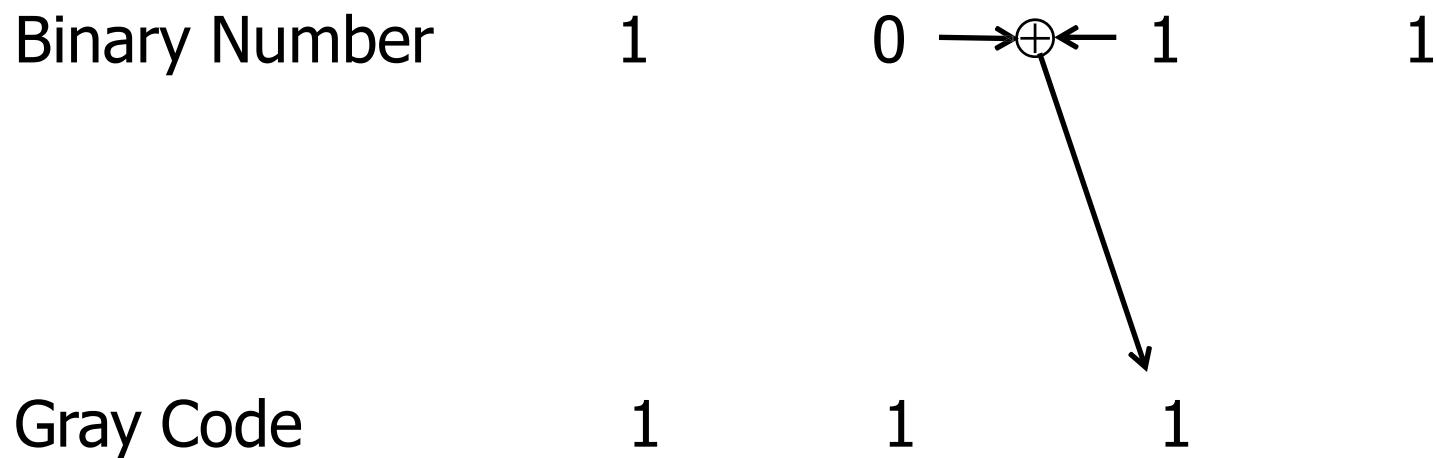
1 1

Gray Code

1 1

EXAMPLE 1:

Continue



EXAMPLE 1:

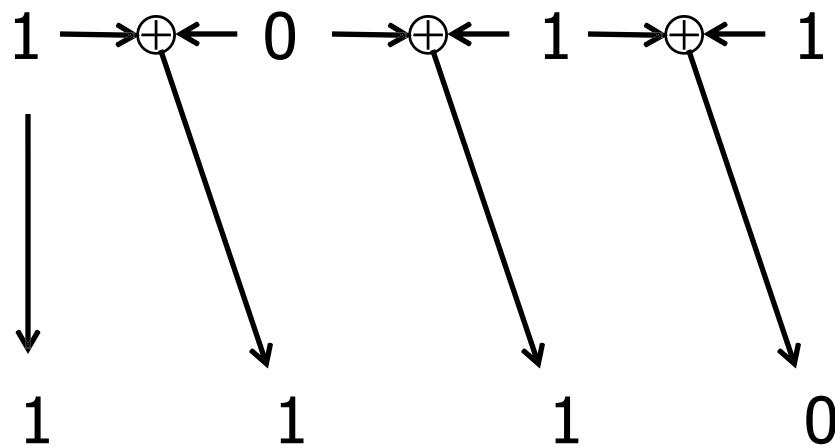
Continue

Binary Number	1	0	1 →⊕← 1
Gray Code	1	1	1 0

EXAMPLE 1:

Continue

Binary Number



Gray Code

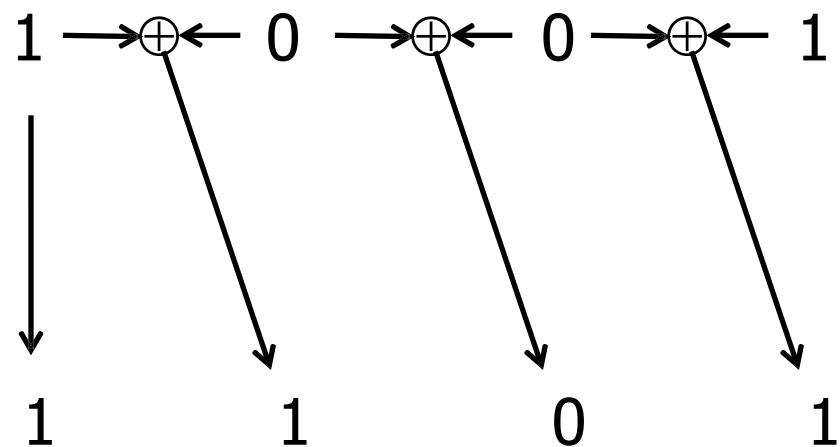
Binary to Gray Code Conversion

Example 2: Convert 1001 Binary Number into Gray Code

BINARY TO GRAY CODE CONVERSION

Example 2: Convert 1001 Binary Number into Gray Code

Binary Number



Gray Code

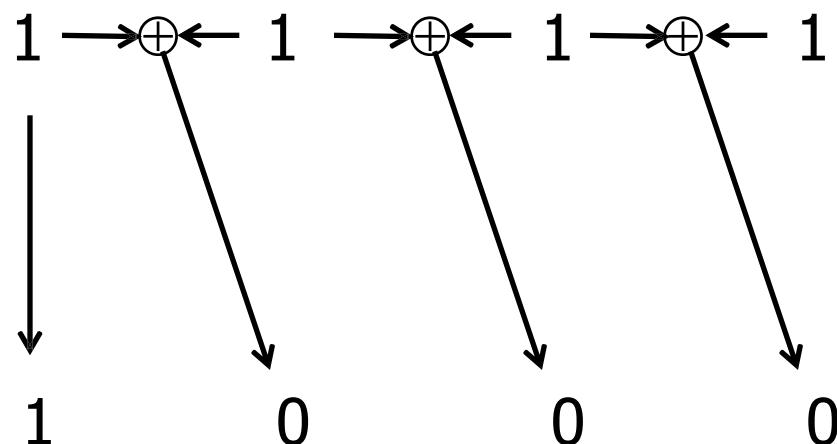
Binary to Gray Code Conversion

Example 3: Convert 1111 Binary Number into Gray Code

BINARY TO GRAY CODE CONVERSION

Example 3: Convert 1111 Binary Number into Gray Code

Binary Number



Gray Code

Binary to Gray Code Conversion

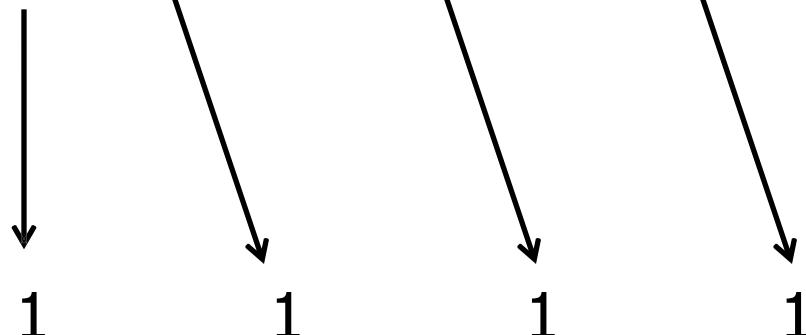
Example 4: Convert 1010 Binary Number into Gray Code

BINARY TO GRAY CODE CONVERSION

Example 4: Convert 1010 Binary Number into Gray Code

Binary Number

1 →⊕← 0 →⊕← 1 →⊕← 0



Gray Code

BINARY AND CORRESPONDING GRAY CODES

Decimal No.	Binary No.	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
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Exercise

- Convert following Binary Numbers into Gray Code

1. $(1011)_2$

2. $(110110010)_2$

3. $(101010110101)_2$

4. $(100001)_2$

Exercise

- Convert following Binary Numbers into Gray Code

$$1. \ (1011)_2 = (1110)_{\text{GRAY}}$$

$$2. \ (110110010)_2 = (101101011)_{\text{GRAY}}$$

$$3. \ (101010110101)_2 = (111111101111)_{\text{GRAY}}$$

$$4. \ (100001)_2 = (110001)_{\text{GRAY}}$$

Gray Code to Binary Conversion

✓ If an n bit gray code is represented by G_n, G_{n-1}, \dots, G_1 and its binary equivalent is represented by B_n, B_{n-1}, \dots, B_1 then binary bits are obtained from gray bits as follows:

$$B_n = G_n \quad | \quad B_{n-1} = B_n \oplus G_{n-1} \quad | \quad B_{n-2} = B_{n-1} \oplus G_{n-2} \quad | \quad \dots \quad | \quad B_1 = B_2 \oplus G_1$$

*where the symbol represents Exclusive-OR operation

Gray Code to Binary Conversion

Example 1: Convert 1110 Gray code into Binary Number.

GRAY CODE TO BINARY CONVERSION

Example 1: Convert 1110 Gray code into Binary Number.

Gray Code

1 1 1 0

Example 1:

CONTINUE

Gray Code

1 1 1 0



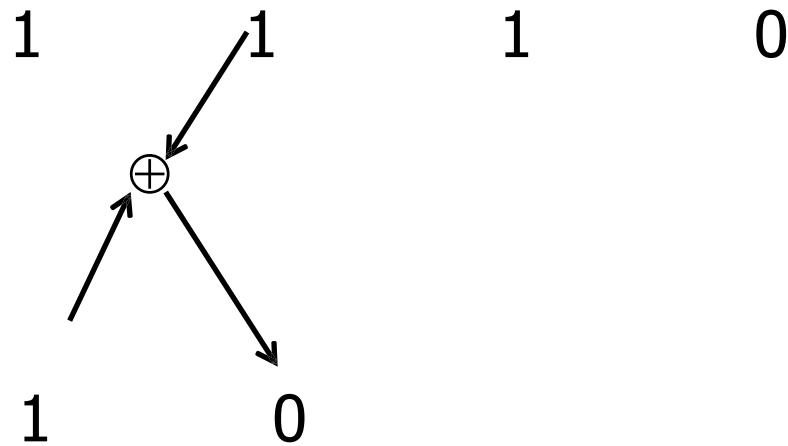
Binary Number

1

EXAMPLE 1:

Continue

Gray Code



Binary Number

EXAMPLE 1:

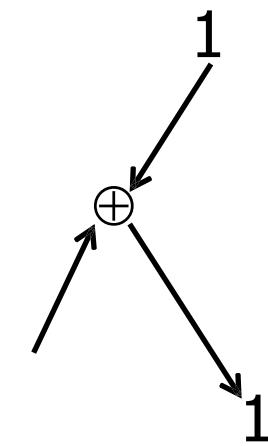
Continue

Gray Code

1 1 1 0

Binary Number

1 0 1 0



EXAMPLE 1:

Continue

Gray Code

1

1

1

0

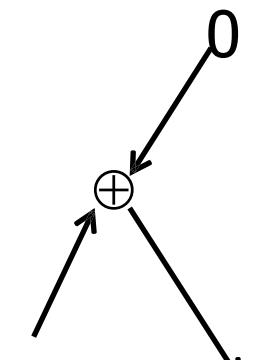
Binary Number

1

0

1

1

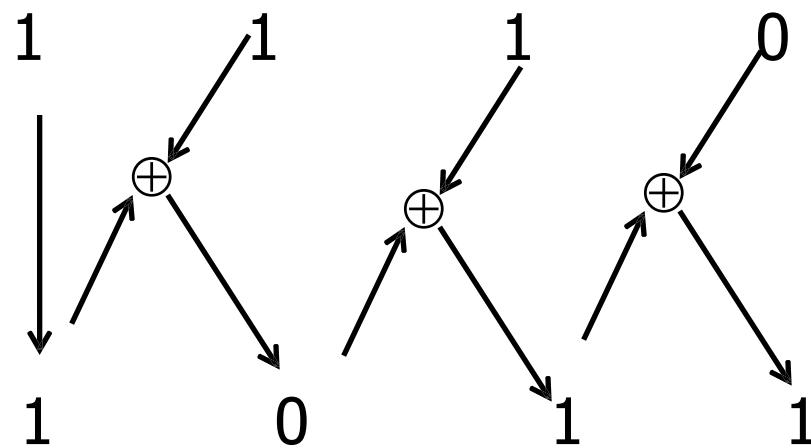


EXAMPLE 1:

Continue

Gray Code

Binary Number



Gray Code to Binary Conversion

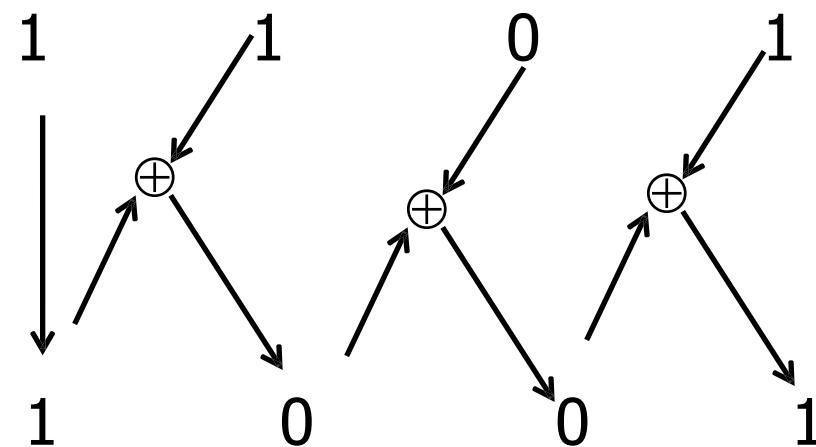
Example 2: Convert 1101 Gray code into Binary Number.

GRAY CODE TO BINARY CONVERSION

Example 2: Convert 1101 Gray code into Binary Number.

Gray Code

Binary Number



Gray Code to Binary Conversion

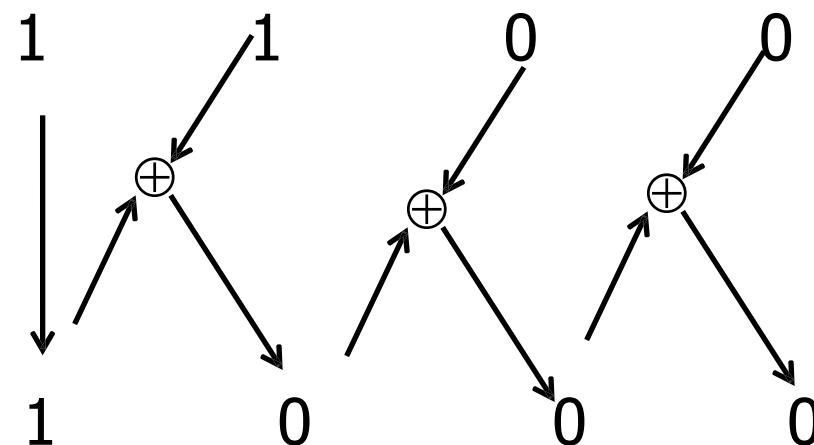
Example 3: Convert 1100 Gray code into Binary Number.

GRAY CODE TO BINARY CONVERSION

Example 3: Convert 1100 Gray code into Binary Number.

Gray Code

Binary Number



Exercise

- Convert following Gray Numbers into Binary Numbers
1. $(1111)_{GRAY}$
 2. $(101110)_{GRAY}$
 3. $(100010110)_{GRAY}$
 4. $(11100111)_{GRAY}$

Exercise

- Convert following Gray Numbers into Binary Numbers

$$1. (1111)_{\text{GRAY}} = (1010)_2$$

$$2. (101110)_{\text{GRAY}} = (110100)_2$$

$$3. (100010110)_{\text{GRAY}} = (111100100)_2$$

$$4. (11100111)_{\text{GRAY}} = (10111010)_2$$

Excess-3 Code (XS-3)

- ✓ The XS-3 is non-weighted BCD code.
- ✓ This code derives its name from the fact that each binary code word is the corresponding 8421 code word plus 0011.
- ✓ It is a sequential code & therefore can be used for arithmetic operations.
- ✓ It is a self complementing code

EXCESS-3 CODE (XS-3)

Decimal No.	BCD Code	Excess-3 Code= BCD + Excess-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Excess-3 Code (XS-3)

Example 1: Obtain Xs-3 Code for 428 Decimal

EXCESS-3 CODE (XS-3)

Example 1: Obtain Xs-3 Code for 428 Decimal

$$\begin{array}{ccc} & 4 & \\ & 2 & \\ & 8 & \\ \\ \text{+} & 0100 & 0010 & 1000 \\ & 0011 & 0011 & 0011 \\ \hline & 0111 & 0101 & 1011 \end{array}$$

Exercise

- Convert following BCD Numbers into Excess- 3 Code
 1. $(40)_{BCD}$
 2. $(88)_{BCD}$
 3. $(64)_{BCD}$
 4. $(23)_{BCD}$

Exercise

- Convert following BCD Numbers into Excess- 3 Code

$$1. \ (40)_{BCD} = [(0100\ 0000) + (0011+0011)] = (73)_{XS3}$$

$$2. \ (88)_{BCD} = [(1000\ 1000) + (0011+0011)] = (BB)_{XS3}$$

$$3. \ (64)_{BCD} = [(0110\ 0100) + (0011+0011)] = (97)_{XS3}$$

$$4. \ (23)_{BCD} = [(0010\ 0011) + (0011+0011)] = (56)_{XS3}$$

ASCII Codes

- ✓ The **American Standard Code for Information Interchange** is a character-encoding scheme originally based on the English alphabet.
- ✓ ASCII codes represent text in computers, communications equipment, and other devices that use text.
- ✓ Most modern character-encoding schemes are based on ASCII, though they support many additional characters.

ASCII Codes

- ✓ ASCII developed from telegraphic codes.
- ✓ Its first commercial use was as a seven-bit tele-printer code promoted by Bell data services.
- ✓ The first edition of the standard was published during 1963.
- ✓ ASCII includes definitions for 128 characters: 33 are non-printing control characters (many now obsolete) that affect how text and space is processed and 95 printable characters, including the space (which is considered an invisible graphic)

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	1100000	140	`
1	1	1	1	[START OF HEADING]	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	[START OF TEXT]	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[END OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	[CANCEL]	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	[END OF MEDIUM]	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	[SUBSTITUTE]	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	[ESCAPE]	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	[FILE SEPARATOR]	76	4C	1001100	114	L	124	7C	1111100	174	
29	1D	11101	35	[GROUP SEPARATOR]	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	[RECORD SEPARATOR]	78	4E	1001110	116	N	126	7E	1111110	176	~
31	1F	11111	37	[UNIT SEPARATOR]	79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
32	20	100000	40	[SPACE]	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	\$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(88	58	1011000	130	X					
41	29	101001	51)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011101	135]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

EBCDIC Codes

- ✓ EBCDIC stands for **Extended Binary Coded Decimal Interchange Code** is eight bits, or one byte, wide.
- ✓ Coding system used to represent characters-letters, numerals, punctuation marks, and other symbols in computerized text.
- ✓ A character is represented in EBCDIC by eight bit.
- ✓ Used on IBM mainframe and IBM mid-range computer operating systems.
- ✓ Each byte consists of two nibbles, each four bits wide

EBCDIC Codes

- ✓ First four bits define class of character, while second nibble defines specific character inside that class.
- ✓ EBCDIC is different from, and incompatible with, ASCII character set used by all other computers.
- ✓ It allows for 256 different characters.
- ✓ For personal computers, however, ASCII is the standard.
If you want to move text between your computer and a mainframe, you can get a file conversion utility that will convert between EBCDIC and ASCII.

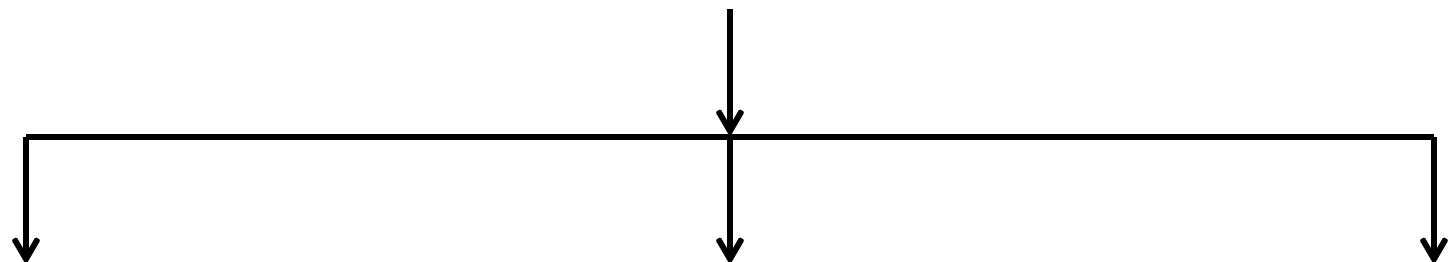
EBCDIC Code Table

BCD ADDITION

- ✓ The BCD addition is performed by individually adding the corresponding digits of the decimal number expressed in 4 bit binary groups starting from LSD.
- ✓ If there is no carry & the sum term is not an illegal code, no correction is needed.
- ✓ If there is a carry out of one group to the next group or if the sum term is an illegal code then 6 i.e. 0110 is added to the sum term of that group and resulting carry is added to the next group.
- ✓ This is done to skip the six illegal states.

BCD ADDITION

Addition of two BCD numbers



Sum \leq 9, Carry=0



Answer is correct.
No correction
required.

Sum \leq 9, Carry=1



Add 6 to the sum
term to get the
correct answer

Sum $>$ 9, Carry=0



Add 6 to the sum
term to get the
correct answer

BCD Addition

Example: Perform in BCD $(57)_{10} + (26)_{10}$

BCD ADDITION

Example: Perform in BCD $(57)_{10} + (26)_{10}$

$$\begin{array}{r} 57 \\ + 26 \\ \hline 83 \end{array} \quad + \quad \begin{array}{r} 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \end{array}$$

Final Carry 0 Valid BCD Code Invalid BCD Code

Thus we have to add 0110 in illegal BCD code

EXAMPLE

Continue

A binary addition diagram showing the sum of two binary numbers. The top number is 01111101 (57₁₀). The bottom number is 00000110 (26₁₀). A plus sign (+) is placed to the left of the bottom number. A horizontal line separates the addends from the sum. The sum is 10001111 (83₁₀). A red horizontal line is drawn under the 5th column from the left (the 4th column of the sum).

	0	1	1	1	1	1	0	1
+	0	0	0	0	0	1	1	0
<hr/>								
	1	0	0	0	0	0	1	1

Add 0110 in
only invalid
code

$$(57)_{10} + (26)_{10} = (83)_{10}$$

Exercise

- Perform BCD Addition
- 1. $(275)_{10} + (493)_{10}$
- 2. $(109)_{10} + (778)_{10}$
- 3. $(88.7)_{10} + (265.8)_{10}$
- 4. $(204.6)_{10} + (185.56)_{10}$

Exercise

- Perform BCD Addition
- 1. $(275)_{10} + (493)_{10} = (768)_{10}$
- 2. $(109)_{10} + (778)_{10} = (887)_{10}$
- 3. $(88.7)_{10} + (265.8)_{10} = (354.5)_{10}$
- 4. $(204.6)_{10} + (185.56)_{10} = (390.16)_{10}$



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.

Thank You



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



Switching Theory and Logic Design

ARI-209

By: Dr. Divya Agarwal



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT- II** 10 HOURS
- **Sequential Logic Circuits:** Latches and Flip Flops- SR, D, T and MS-JK Flip Flops, Asynchronous Inputs.
- **Counters and Shift Registers:** Design of Synchronous and Asynchronous Counters- Binary, BCD, Decade and Up/Down Counters, Shift Registers, Types of Shift Registers, Counters using Shift Registers- Ring Counter and Johnson Counter.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-III** **08 HOURS**
- **Integrated circuits:** TTL and CMOS logic families and their characteristics. Brief introduction to RAM and ROM
- **Synchronous Sequential Circuits:** State Tables State Equations and State Diagrams, State Reduction and State Assignment, Design of Clocked Sequential Circuits using State Equations.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-IV** 06 HOURS
- **Finite state machine:** capabilities and limitations, Mealy and Moore models-minimization of completely specified and incompletely specified sequential machines, Partition techniques and merger chart methods, concept of minimal cover table.
- **Algorithmic State Machine:** Representation of sequential circuits using ASM charts synthesis of output and next state functions, Data path control path partition-based design.



Text Books:

1. Leach and Malvino (2011). *Digital principles and Applications*. Tata McGraw-Hill Education
2. Mano, M. M. (2017). *Digital logic and computer design*. Pearson Education India.
3. Jain, R. P. (2003). *Modern digital electronics*. Tata McGraw-Hill Education.

Reference Books:

1. A Anand Kumar. (2016) *Fundamentals of Digital Logic Circuits*, PHI
2. Taub, H., & Schilling, D. L. (1977). *Digital integrated electronics*. McGraw-Hill College.



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.

Basics.....

- ✓ Logic gates are the fundamental building blocks of digital systems.
- ✓ The name logic gate is derived from the ability of such devices to make decisions, in the sense that it produces one output level when some combinations of input levels are present

➤ Inputs & Outputs for Logic Circuits

- ✓ Input & Output of logic gates can occur only in two levels.

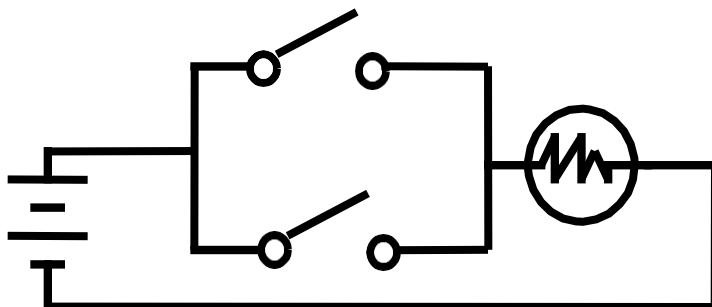
HIGH	LOW
True	False
ON	OFF
1	0

Basics.....

➤ Truth Table

- ✓ A table which lists all the possible combinations of input variables and the corresponding outputs is called a “Truth Table”.
- ✓ It shows how the logic circuits output responds to various combinations of logic levels at the inputs

Switches in parallel => OR



Switch 1	Switch 2	Output
OFF	OFF	OFF
OFF	ON	GLOW
ON	OFF	GLOW
ON	ON	GLOW

Basics.....

➤ Logic

- ✓ A logic in which the voltage levels represent logic 1 and logic 0.
- ✓ Level logic may be Positive or Negative.
- ✓ A “**Positive Logic**” is the one which the higher of the two voltage levels represents the logic 1 and the lower of the two voltage level represents the logic 0.
- ✓ A “**Negative Logic**” is the one which the lower of the two voltage levels represents the logic 1 and the higher of the two voltage level represents the logic 0.

Basics.....

➤ Logic

✓ Positive Logic

Logic 0 (LOW)=0V

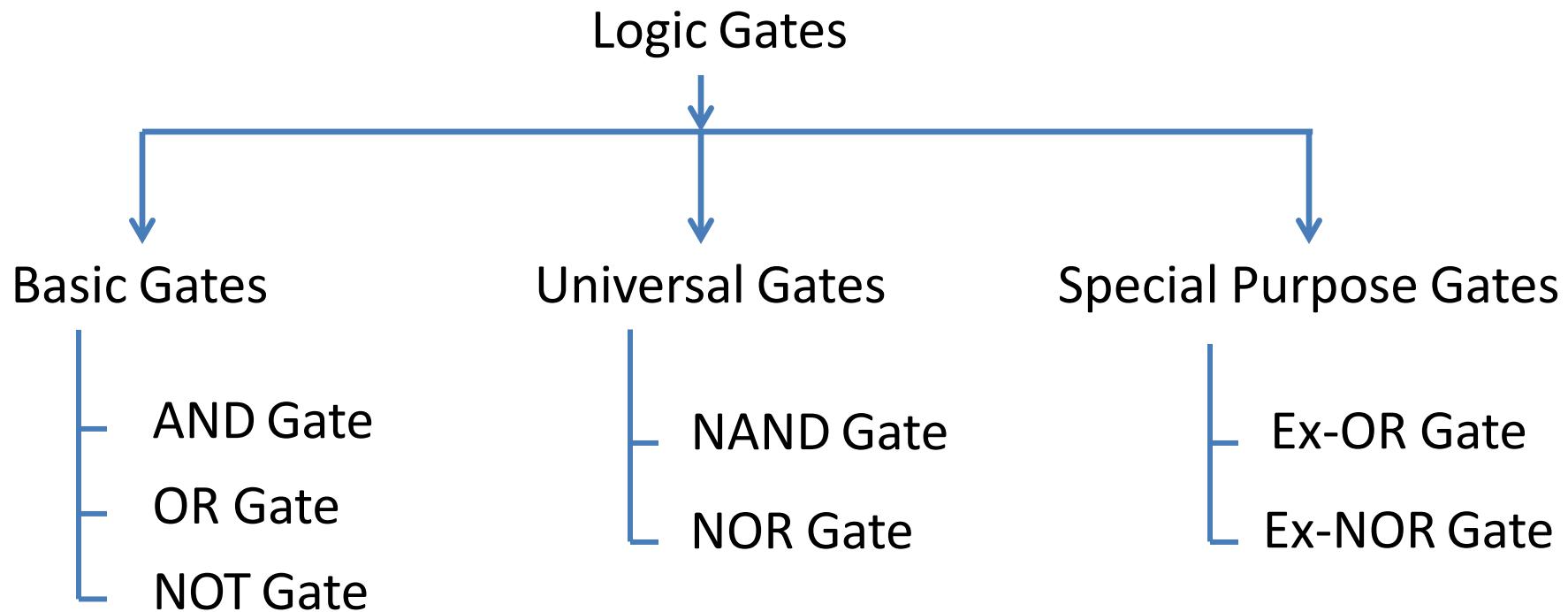
Logic 1 (HIGH)=+5V

✓ Negative Logic

Logic 0 (LOW)=+5V

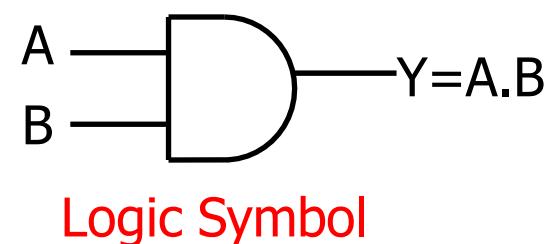
Logic 1 (HIGH)=0V

Logic Gates

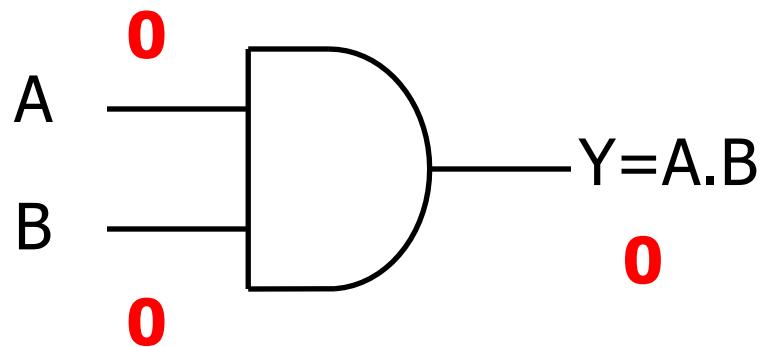


AND Gate

- ✓ An AND gate has two or more inputs but only one output.
- ✓ The output assumes the logic 1 state, when both inputs are at logic 1 state.
- ✓ The output assumes the logic 0 state even if one of its inputs is at logic 0 state.

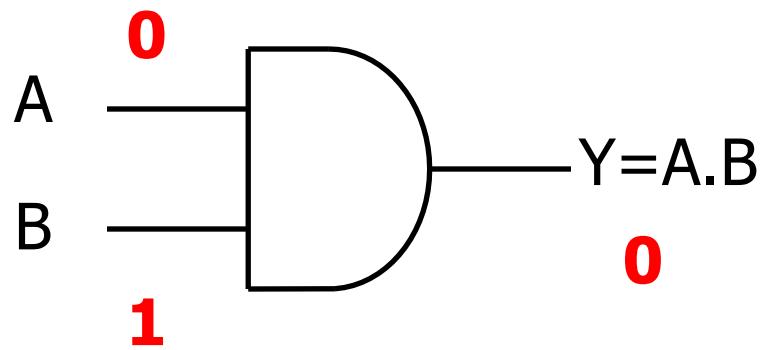


AND Gate



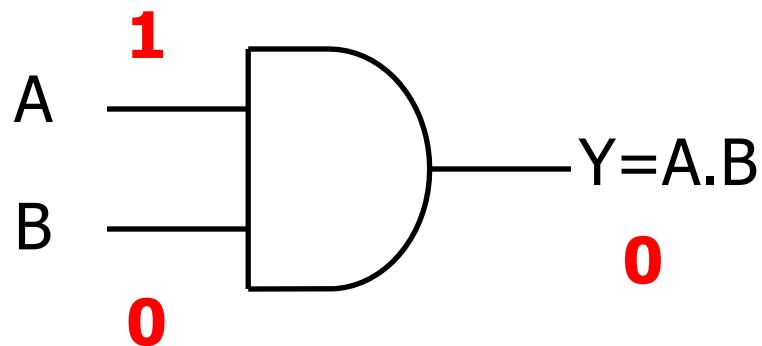
Inputs		Output
A	B	$Y=A \cdot B$
0	0	0

AND Gate



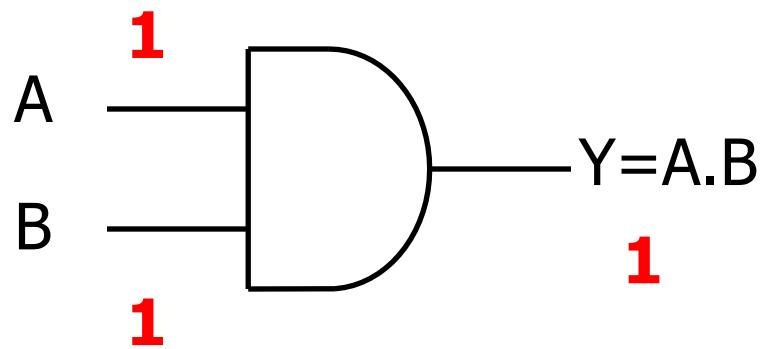
Inputs		Output
A	B	$Y=A \cdot B$
0	0	0
0	1	0

AND Gate



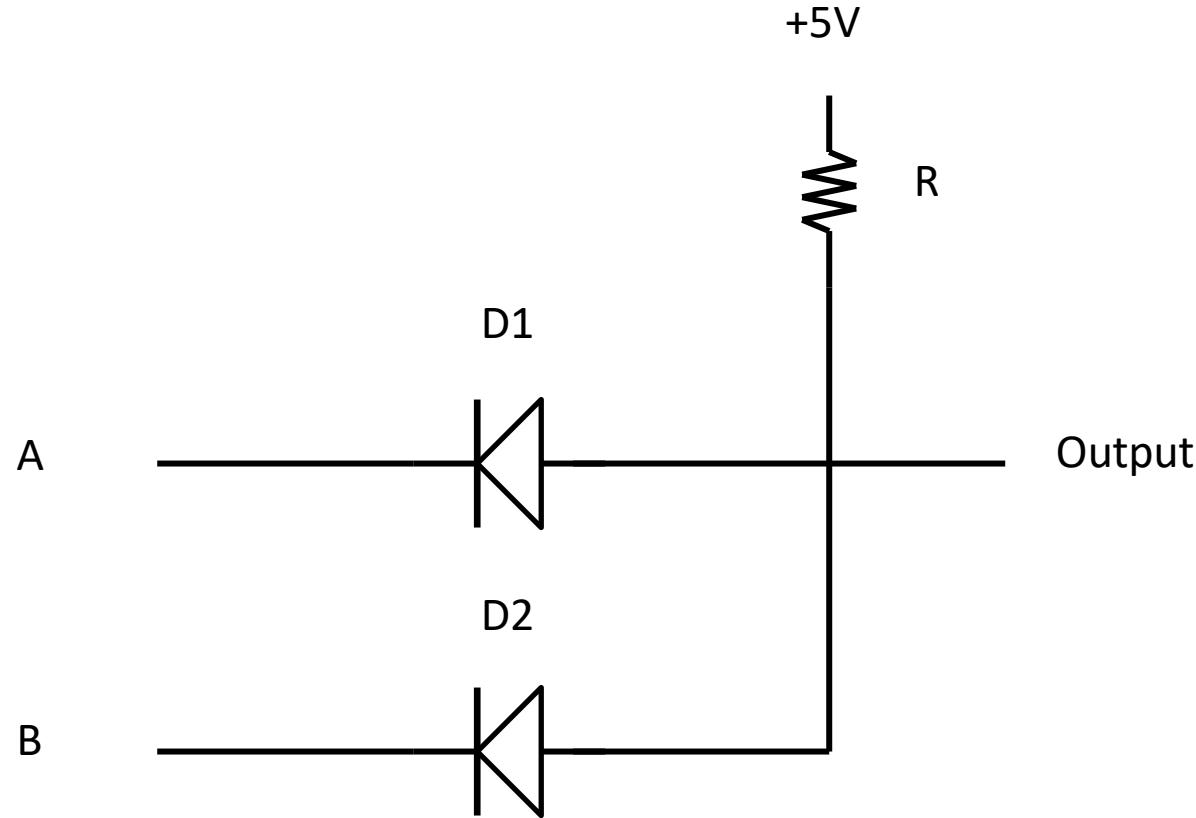
Inputs		Output
A	B	$Y=A \cdot B$
0	0	0
0	1	0
1	0	0

AND Gate



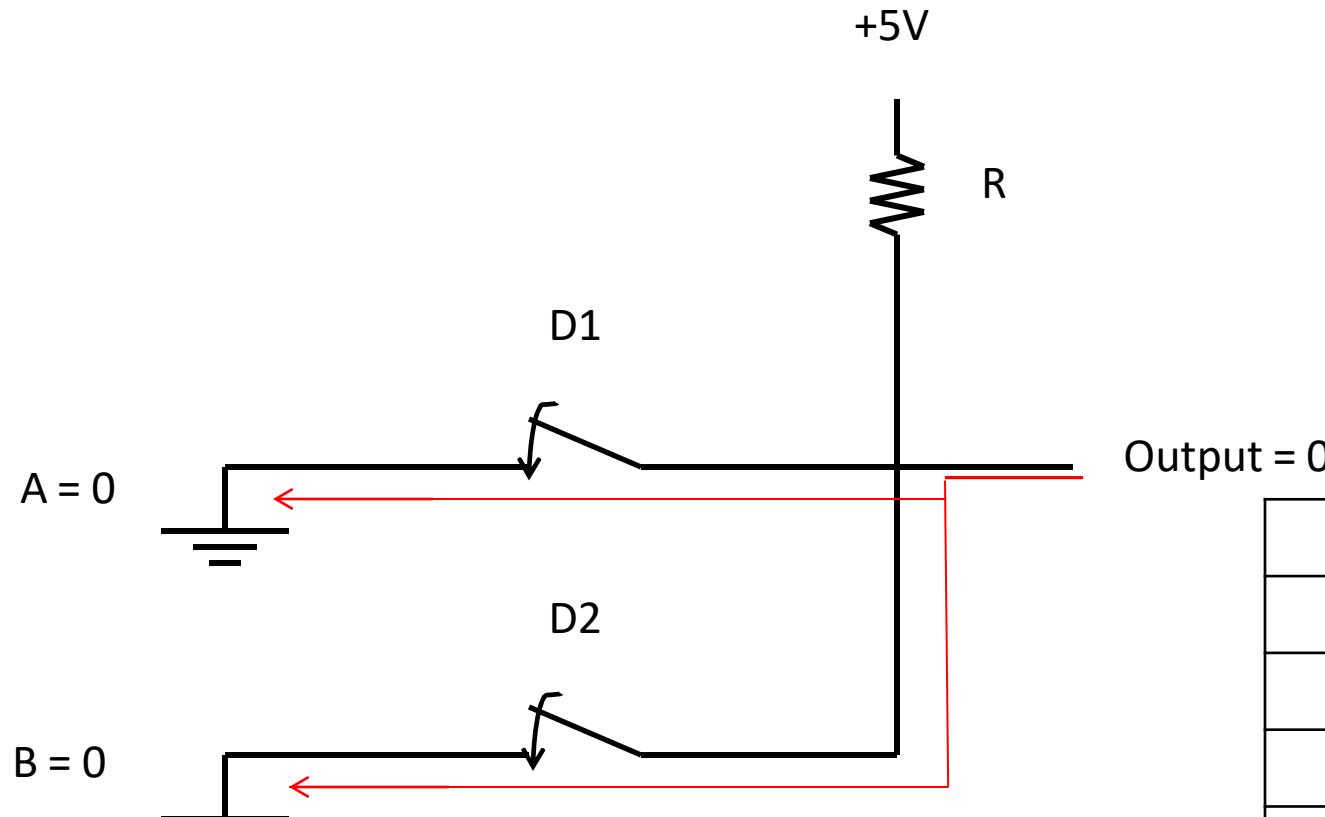
Inputs		Output
A	B	$Y=A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

Realization of AND Gate using Diode



Realization of AND Gate using Diode

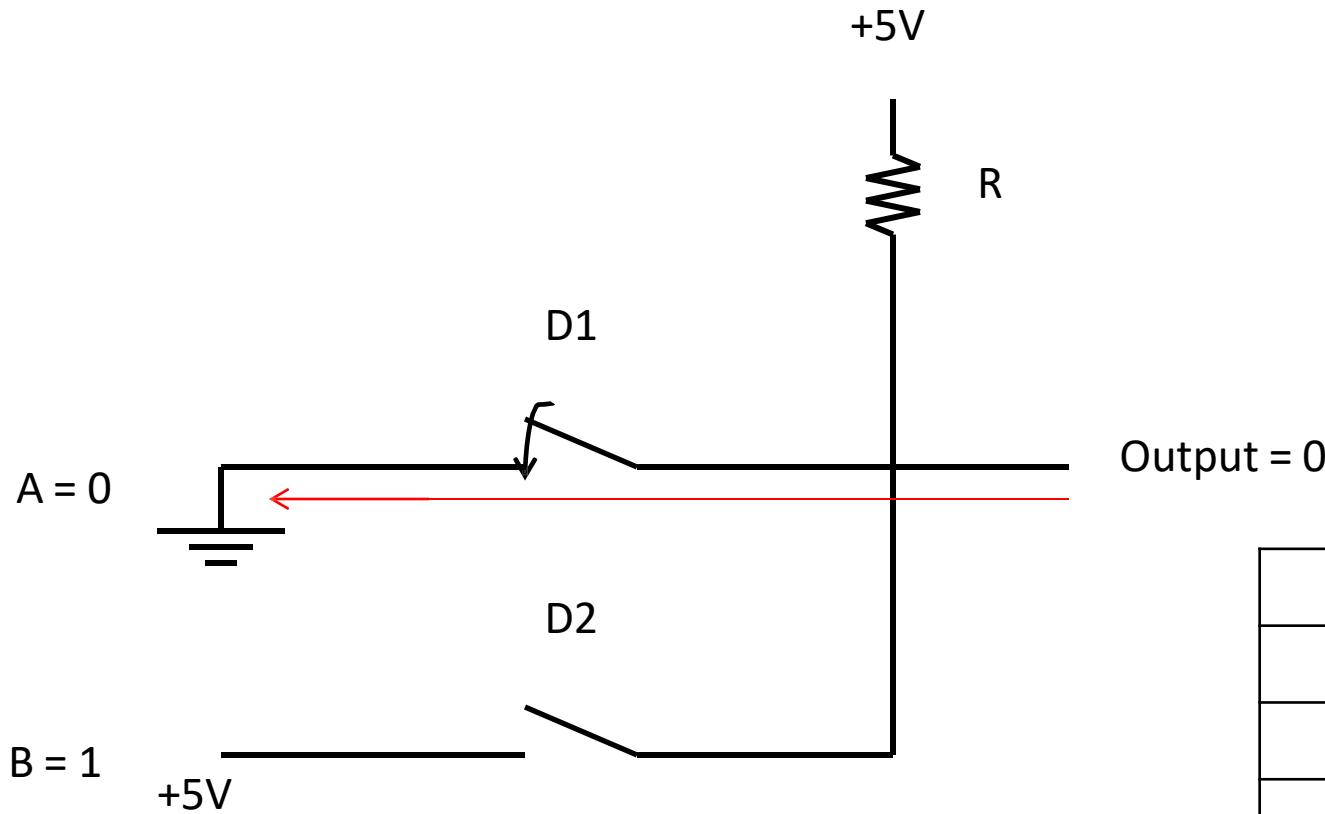
Case I: When A = 0 and B = 0



Inputs		Output
A	B	$Y = A \cdot B$
0	0	0

Realization of AND Gate using Diode

Case II: When A = 0 and B = 1

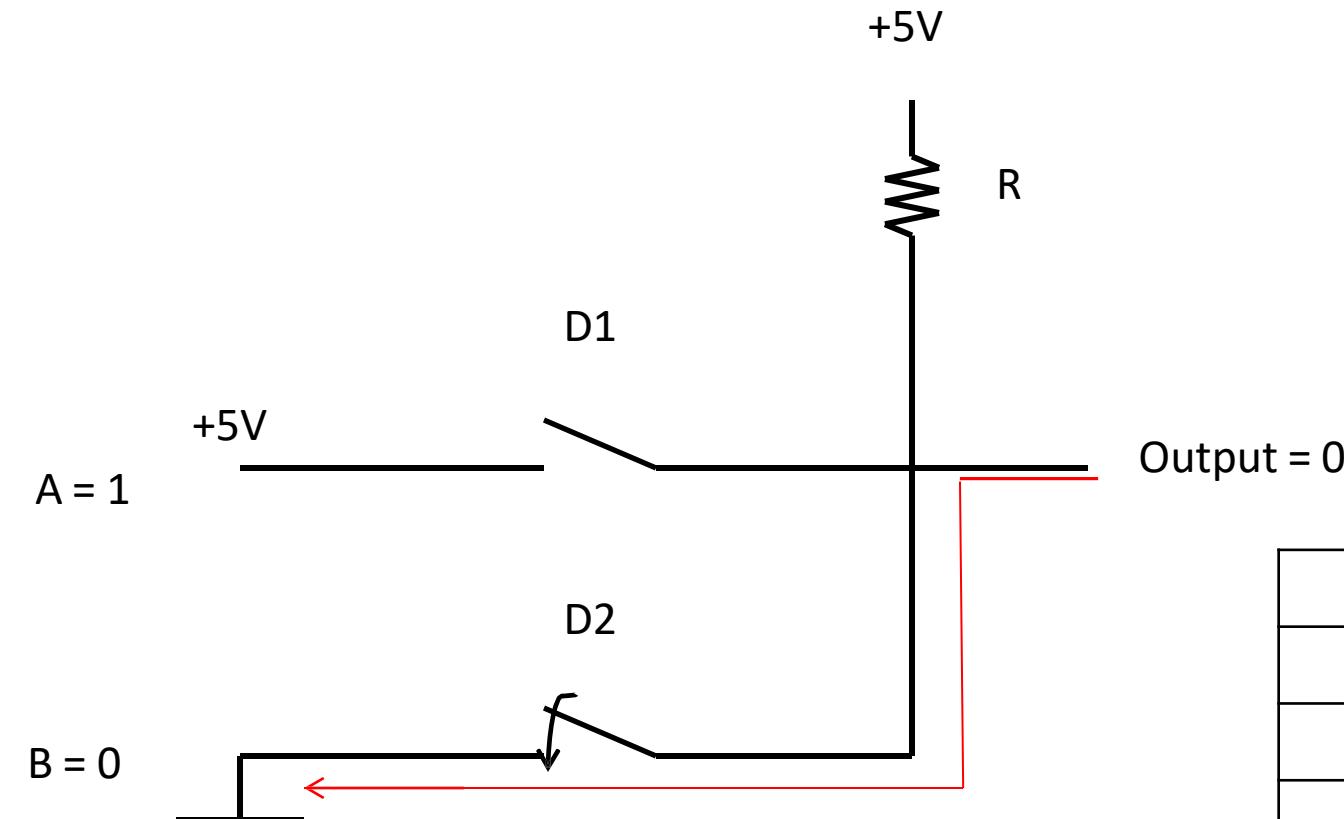


Output = 0

Inputs		Output
A	B	$Y=A \cdot B$
0	0	0
0	1	0

Realization of AND Gate using Diode

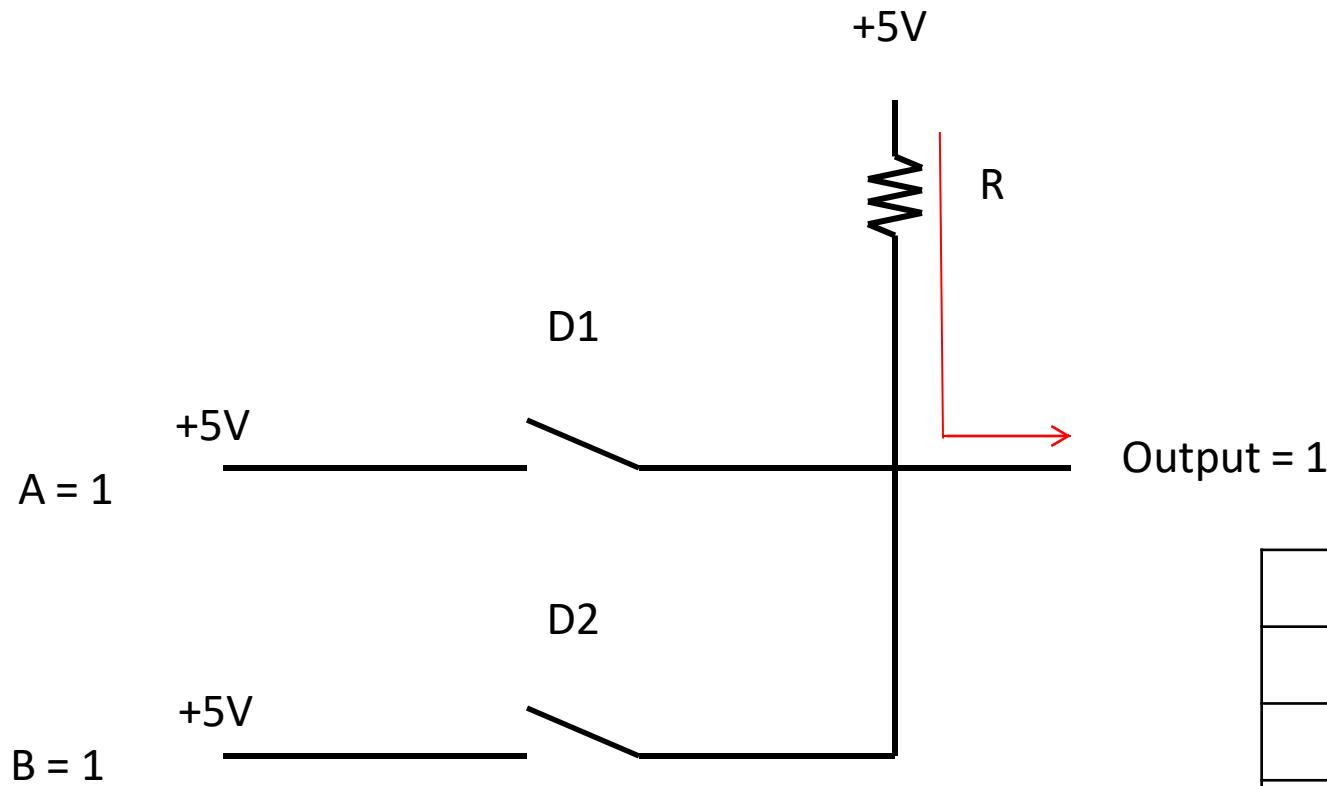
Case III: When A = 1 and B = 0



Inputs		Output
A	B	$Y=A.B$
0	0	0
0	1	0
1	0	0

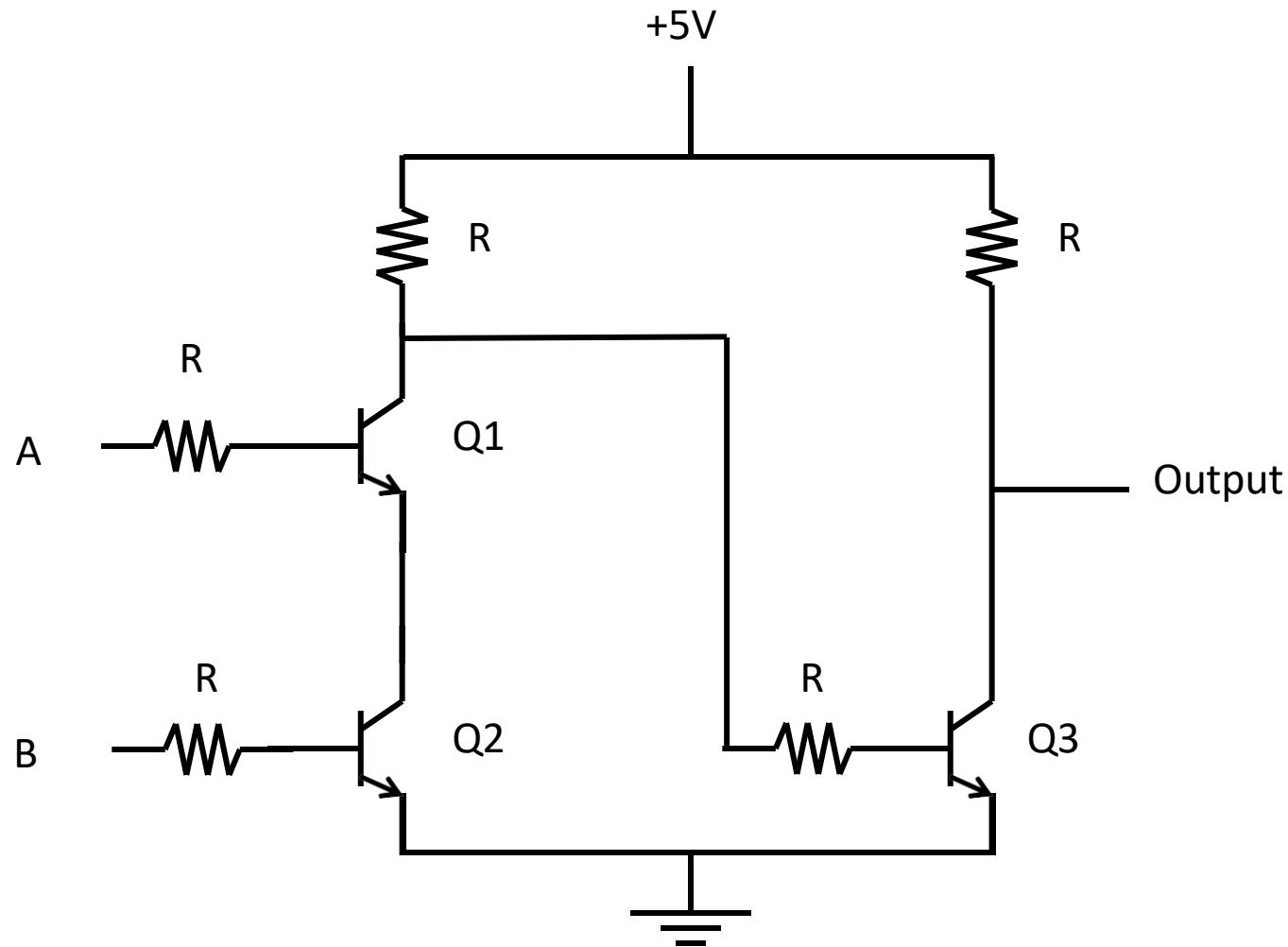
Realization of AND Gate using Diode

Case IV: When A = 1 and B = 1



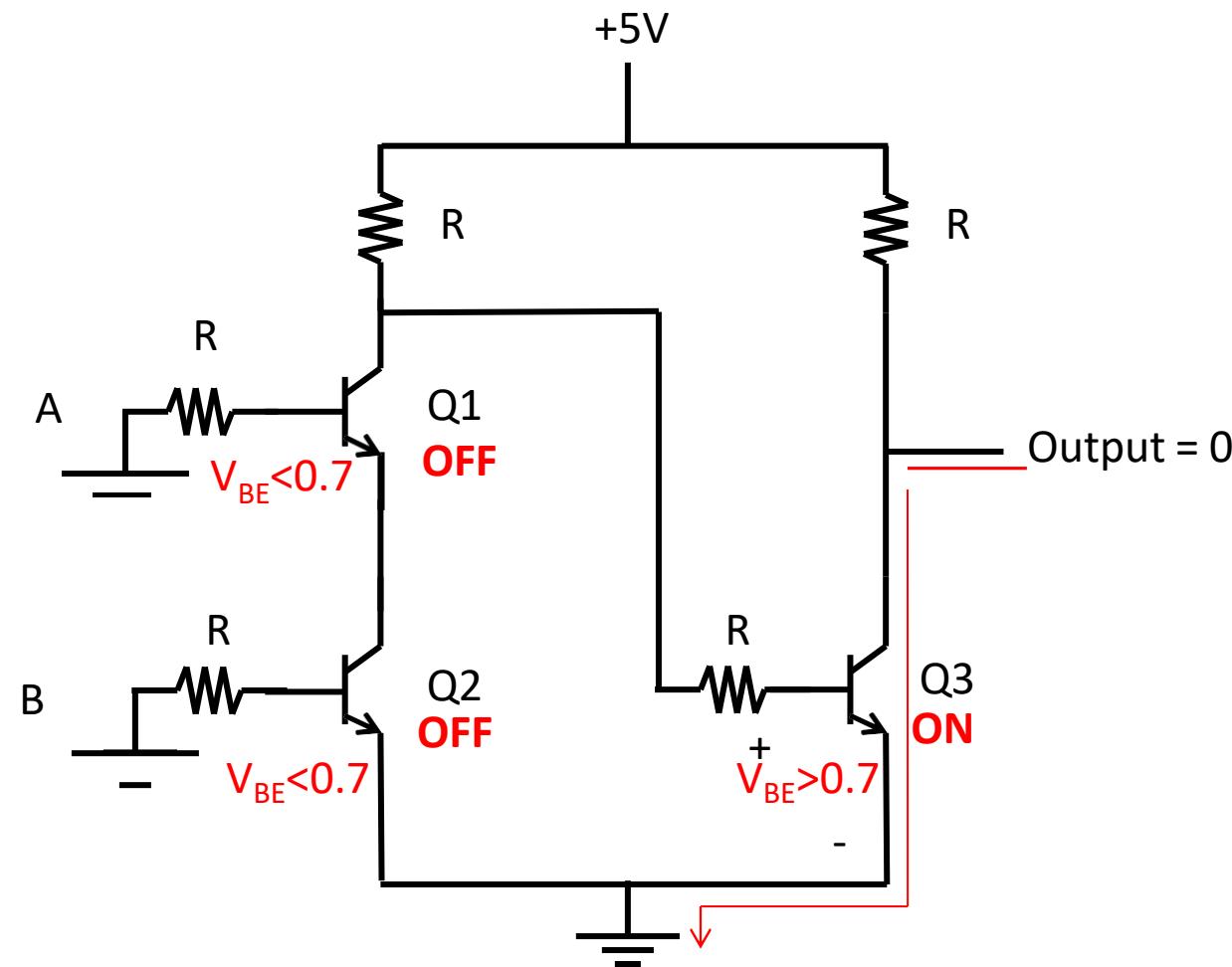
Inputs		Output
A	B	$Y=A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

Realization of AND Gate using Transistor



Realization of AND Gate using Transistor

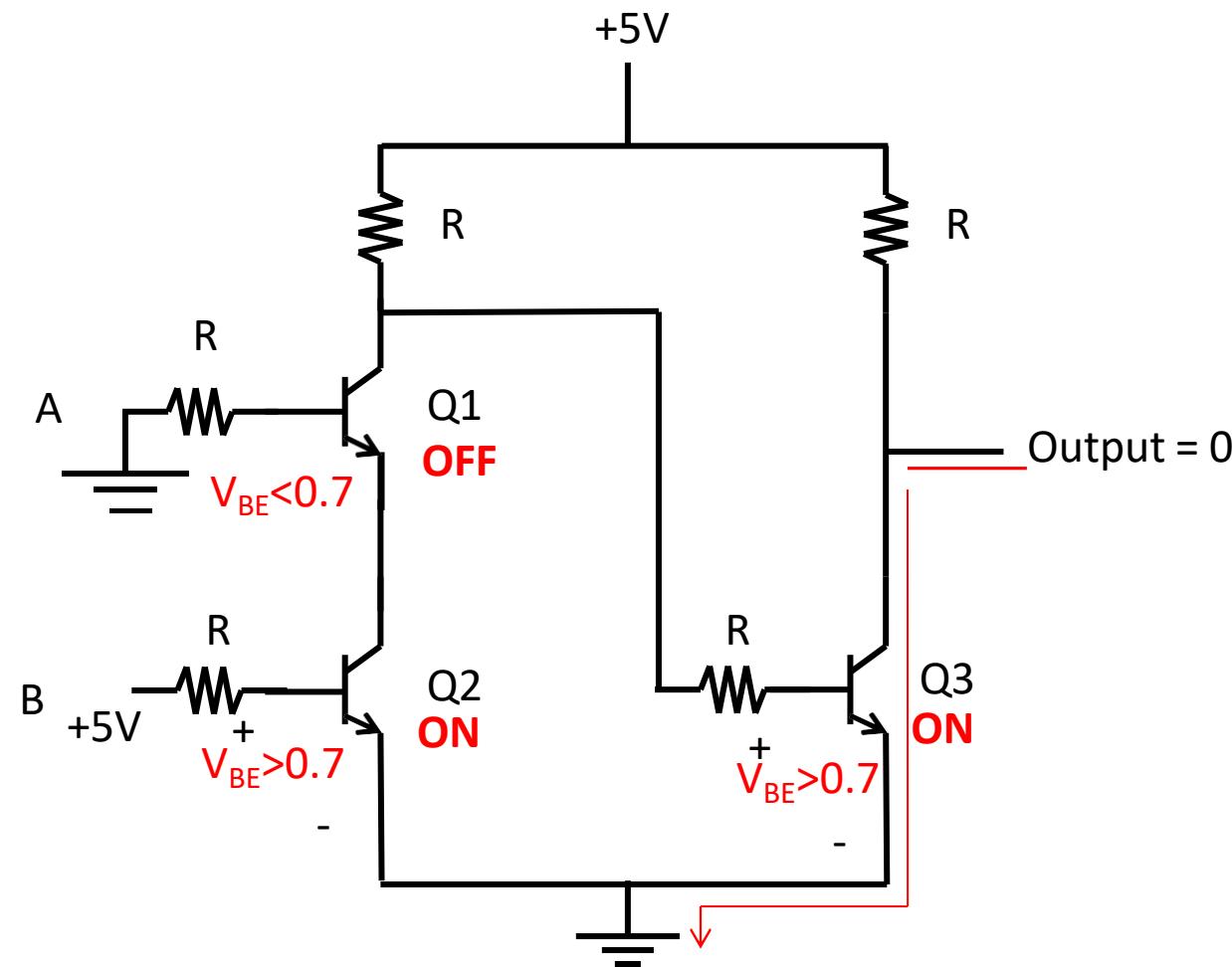
Case I: When A = 0 and B = 0



Inputs		Output
A	B	$Y=A.B$
0	0	0

Realization of AND Gate using Transistor

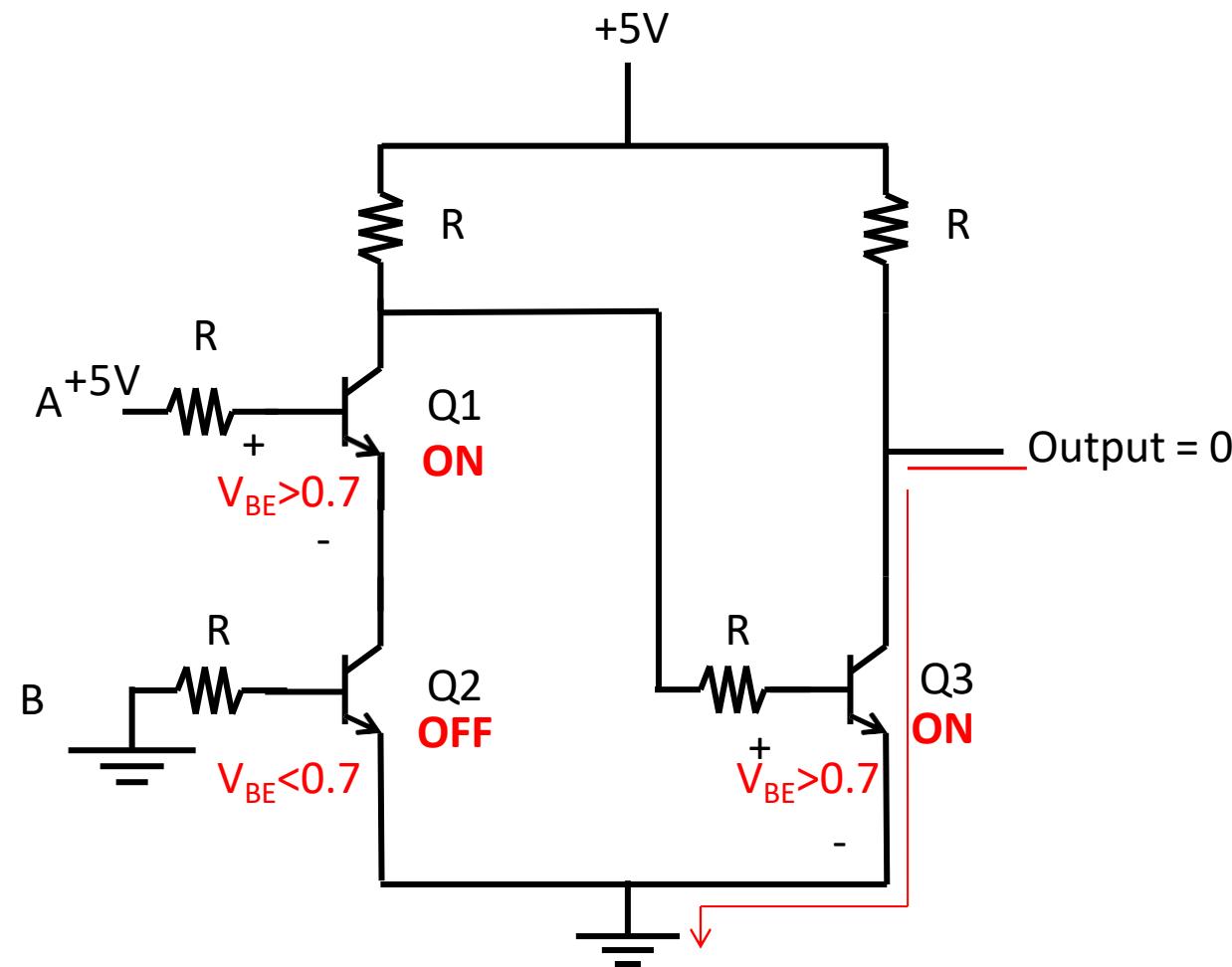
Case II: When A = 0 and B = 1



Inputs		Output
A	B	$Y = A \cdot B$
0	0	0
0	1	0

Realization of AND Gate using Transistor

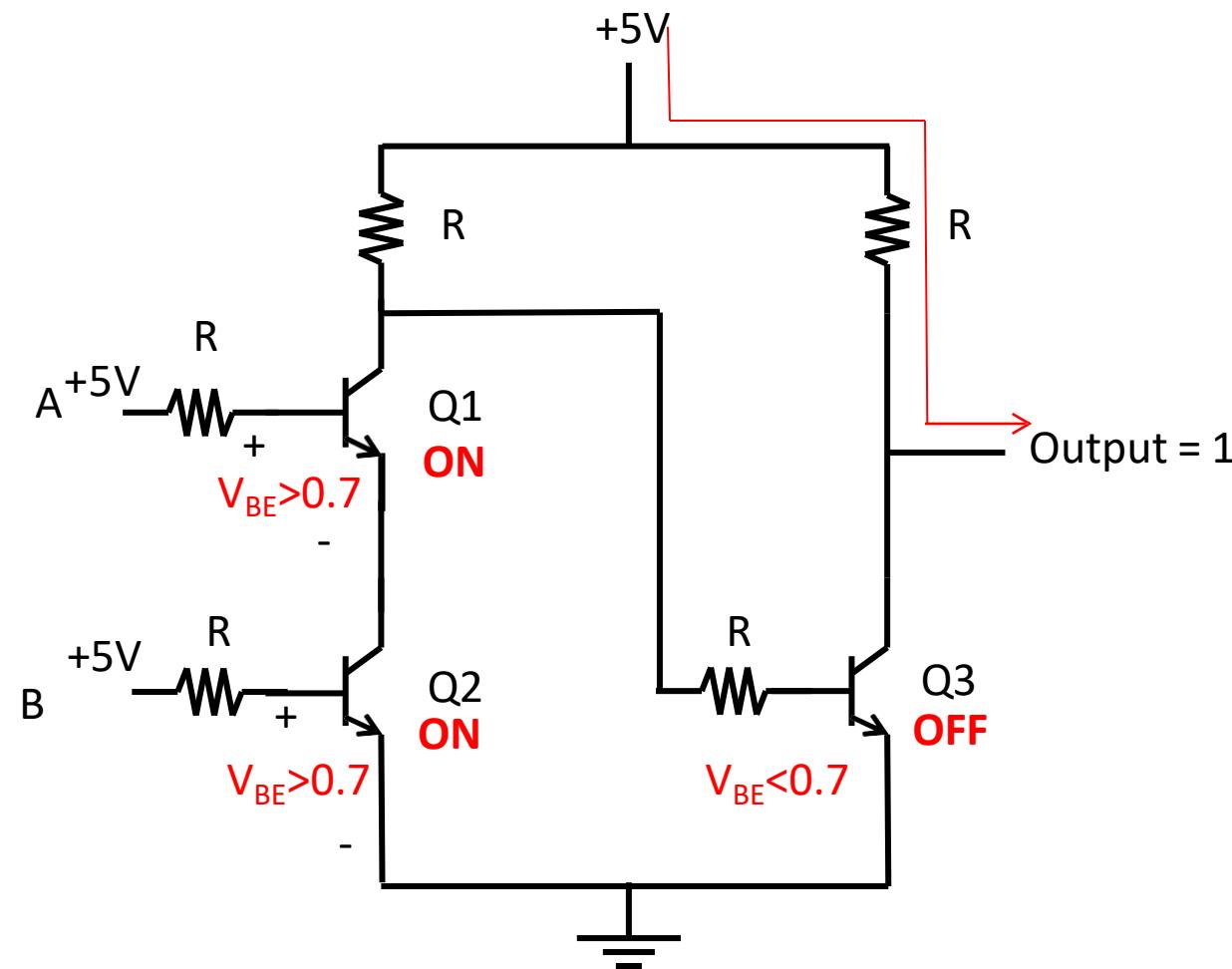
Case III: When A = 1 and B = 0



Inputs		Output
A	B	$Y=A.B$
0	0	0
0	1	0
1	0	0

Realization of AND Gate using Transistor

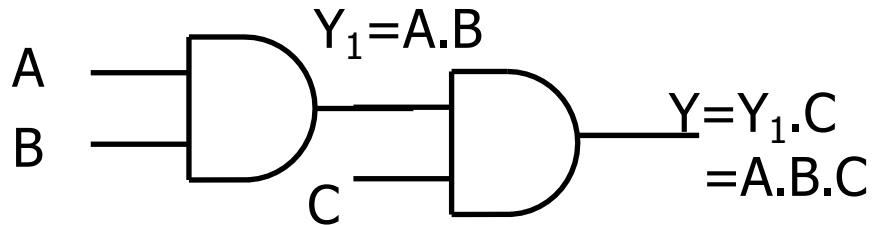
Case IV: When A = 1 and B = 1



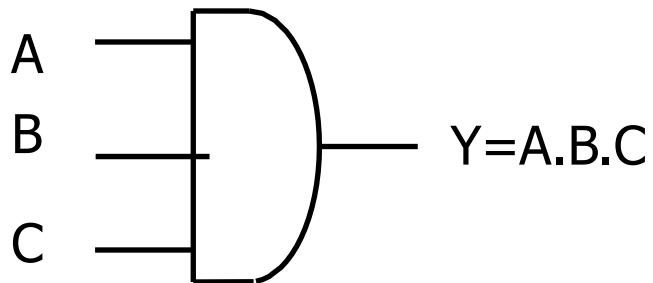
Inputs		Output
A	B	$Y=A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

3 - Input AND Gate

3 – Input AND Gate using 2 – Input AND Gate



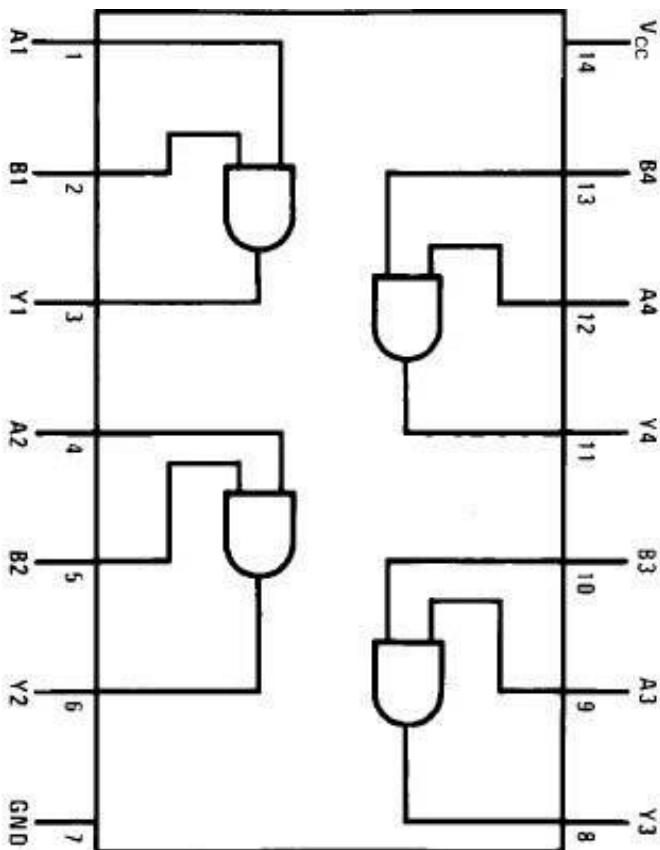
Symbol : 3 – Input AND Gate



Truth Table : 3 – Input AND Gate

Input			Output $Y = A \cdot B \cdot C$
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

AND Gate IC – IC 7408 (Quad 2 I/P AND Gate)



Pin Configuration of IC 7408

- ✓ This device contains four independent gates each of which performs the logic AND function.

$$Y = AB$$

Inputs		Output
A	B	Y
L	L	L
L	H	L
H	L	L
H	H	H

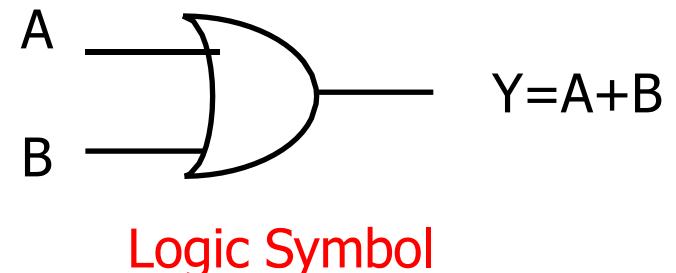
H = High Logic Level

L = Low Logic Level

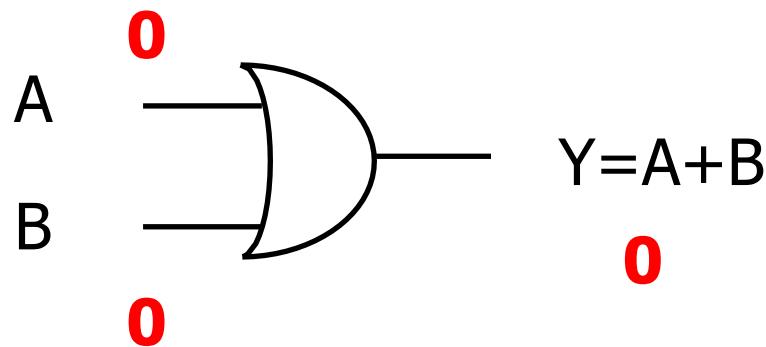
Function Table of IC 7408

OR Gate

- ✓ An OR gate has two or more inputs but only one output.
- ✓ The output assumes the logic 1 state, when even if one of its inputs is in logic 1 state.
- ✓ The output assumes the logic 0 state only when both the inputs are in logic 0 state.

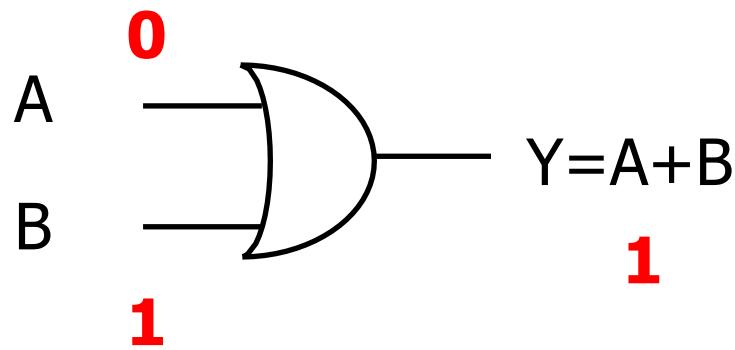


OR Gate



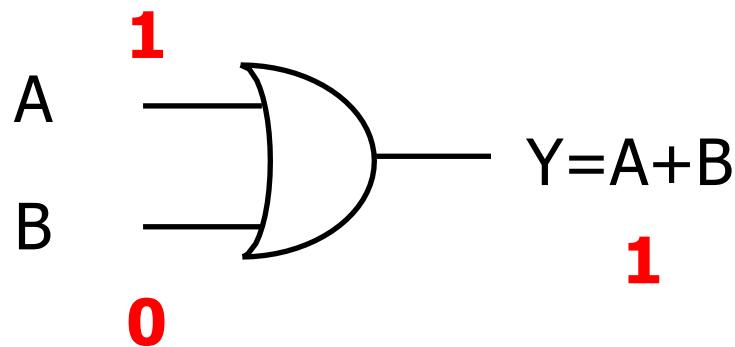
Inputs		Output
A	B	$Y = A + B$
0	0	0

OR Gate



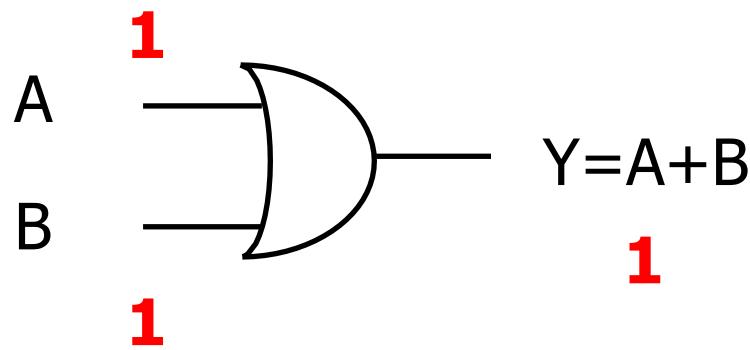
Inputs		Output
A	B	$Y = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

OR Gate



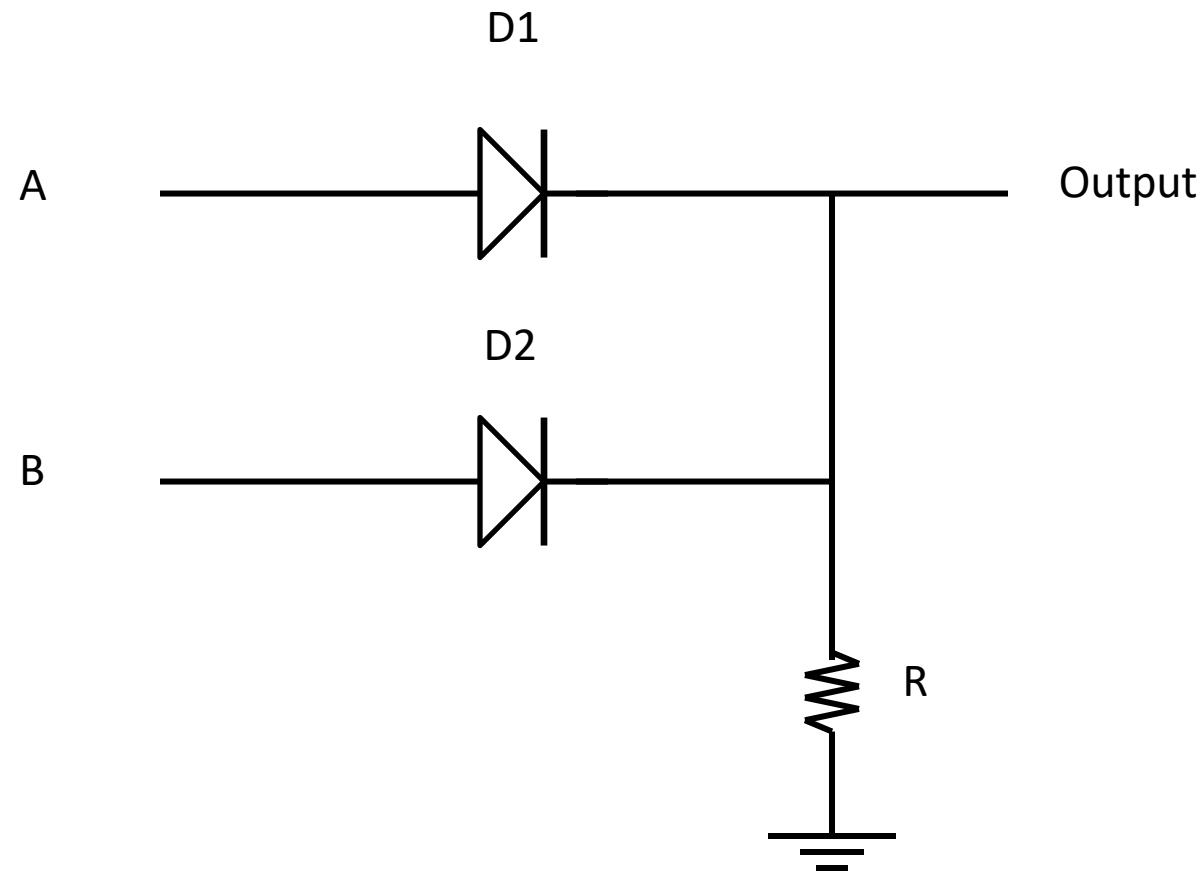
Inputs		Output
A	B	$Y = A + B$
0	0	0
0	1	1
1	0	1

OR Gate



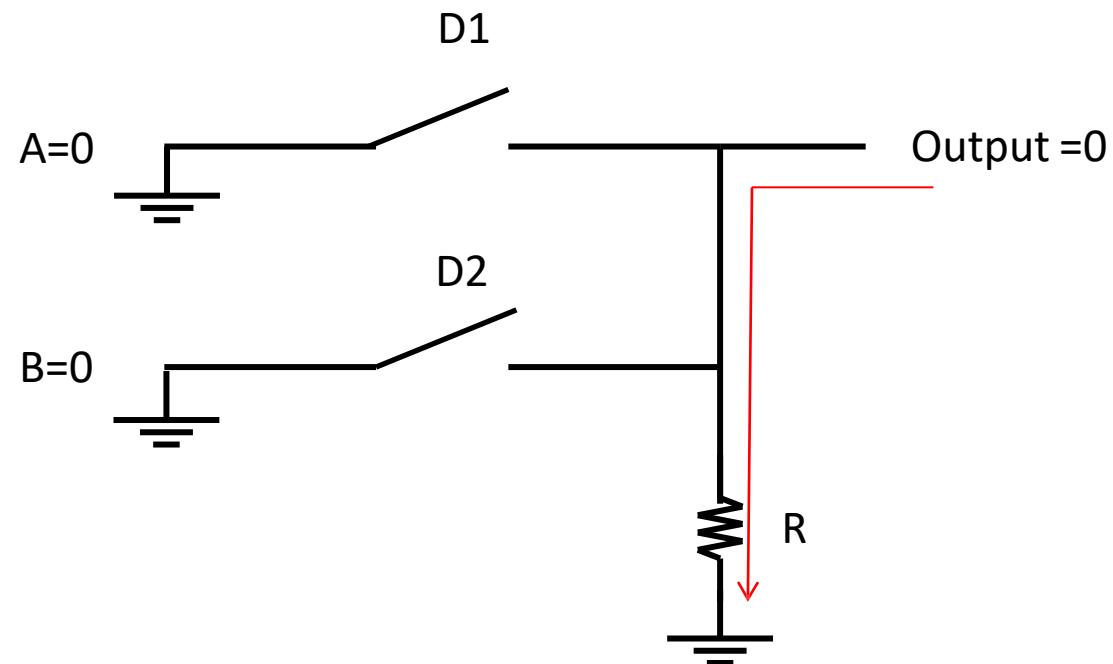
Inputs		Output
A	B	$Y = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

Realization of OR Gate Using Diodes



Realization of OR Gate Using Diodes

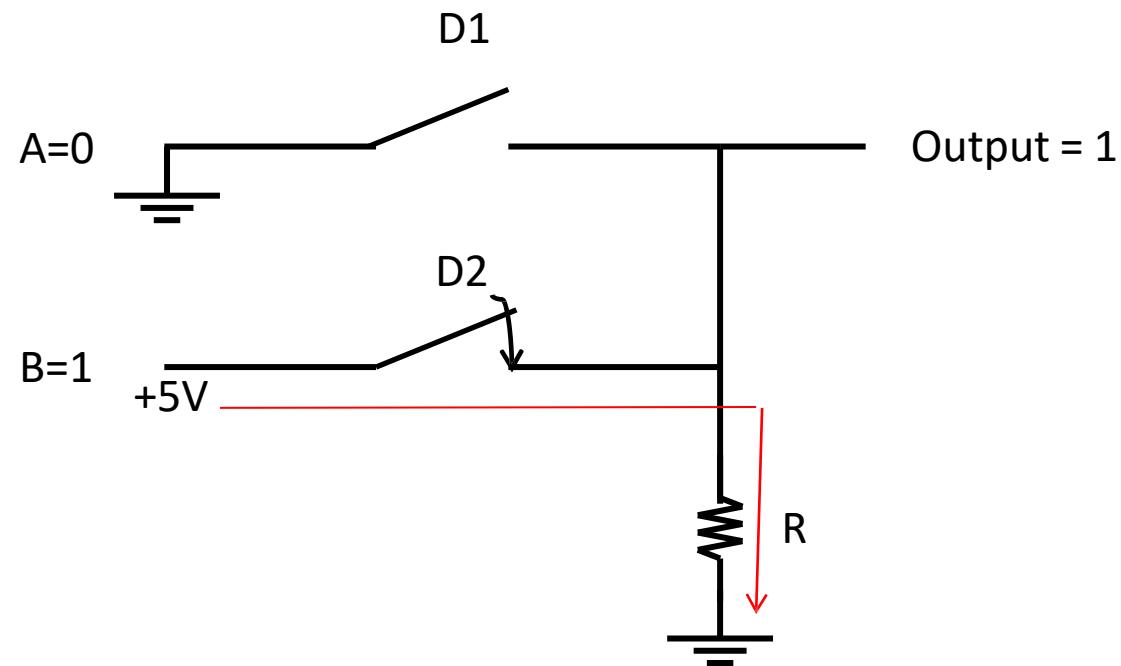
Case I: When A = 0 and B = 0



Inputs		Output
A	B	$Y=A+B$
0	0	0

Realization of OR Gate Using Diodes

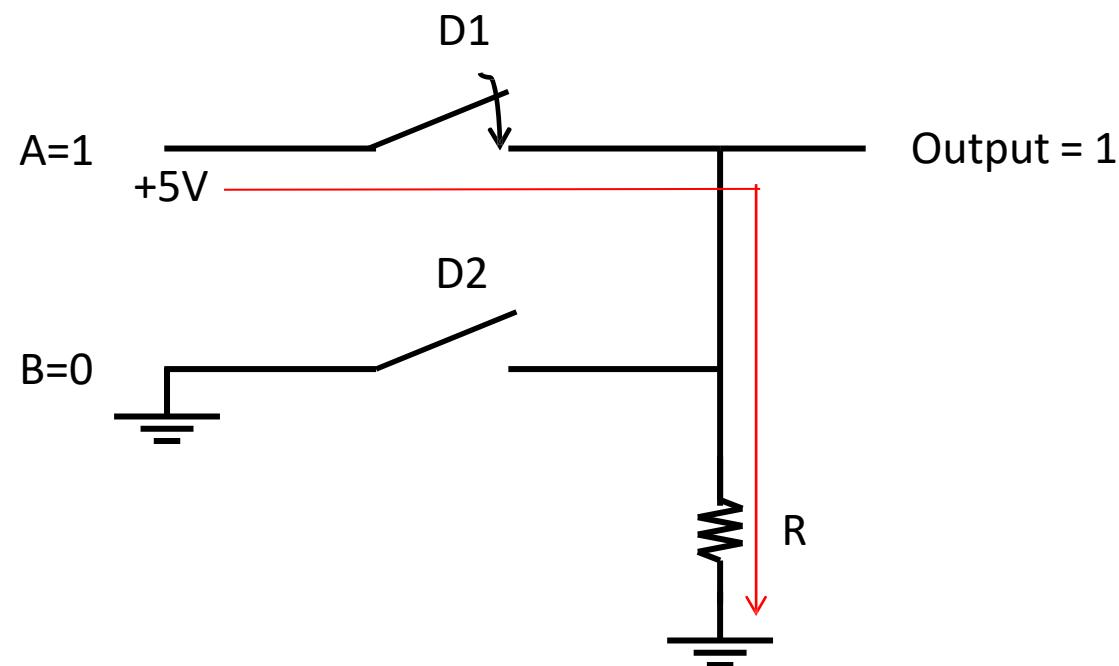
Case II: When A = 0 and B = 1



Inputs		Output
A	B	$Y=A+B$
0	0	0
0	1	1

Realization of OR Gate Using Diodes

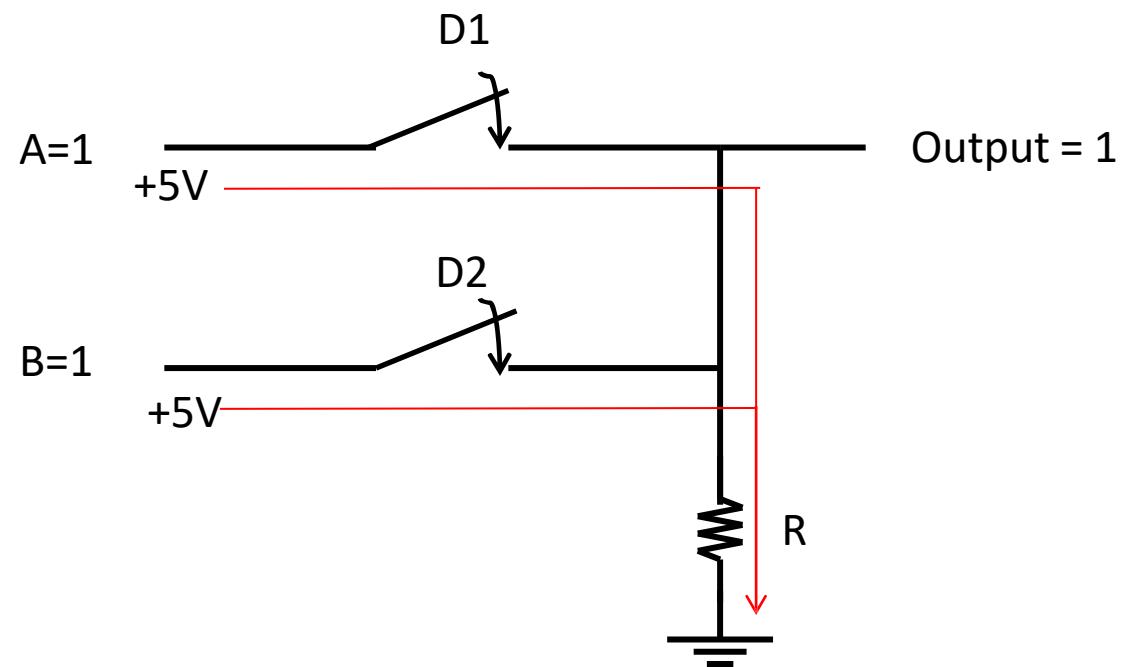
Case III: When A = 1 and B = 0



Inputs		Output
A	B	$Y=A+B$
0	0	0
0	1	1
1	0	1

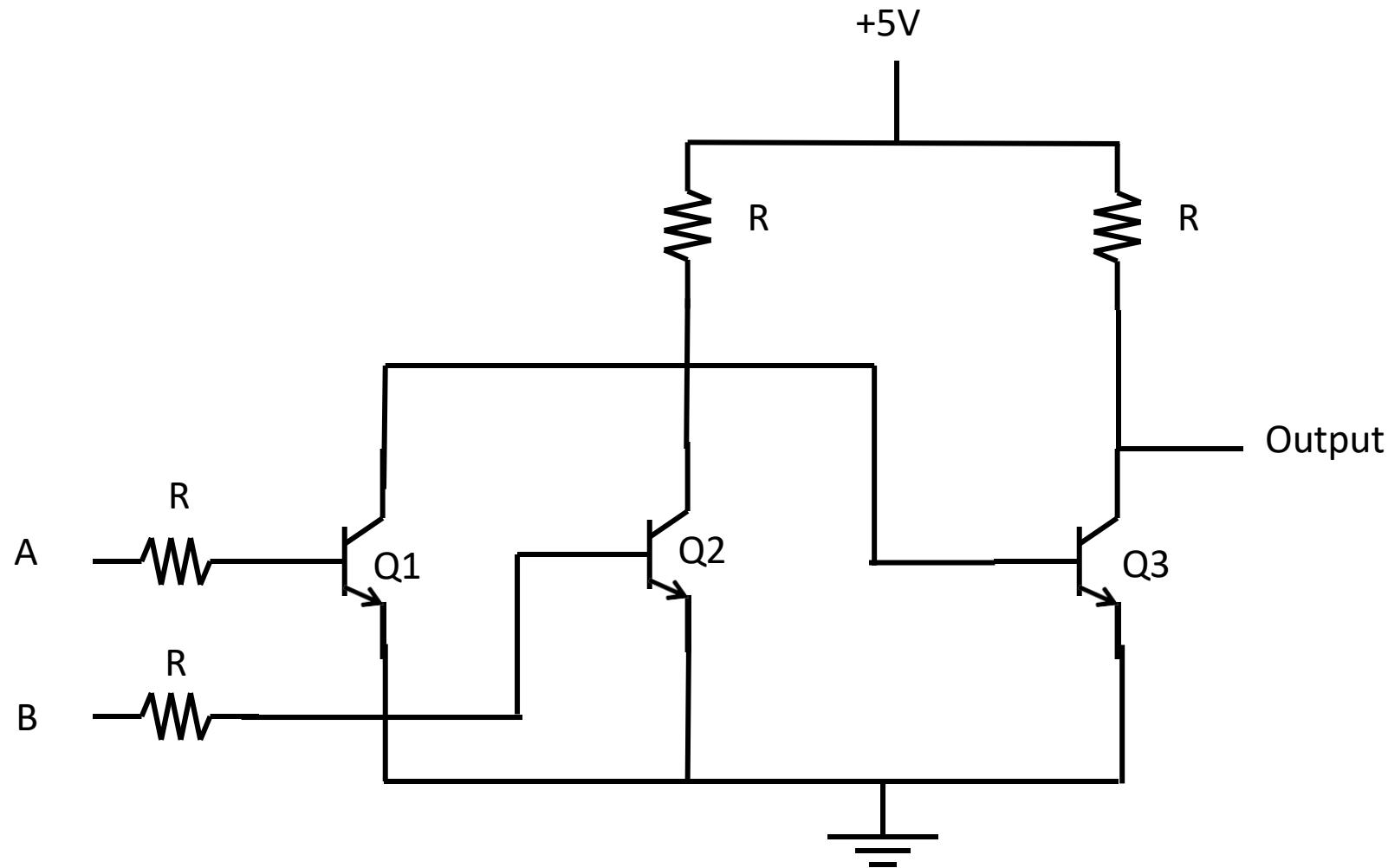
Realization of OR Gate Using Diodes

Case IV: When A = 1 and B = 1



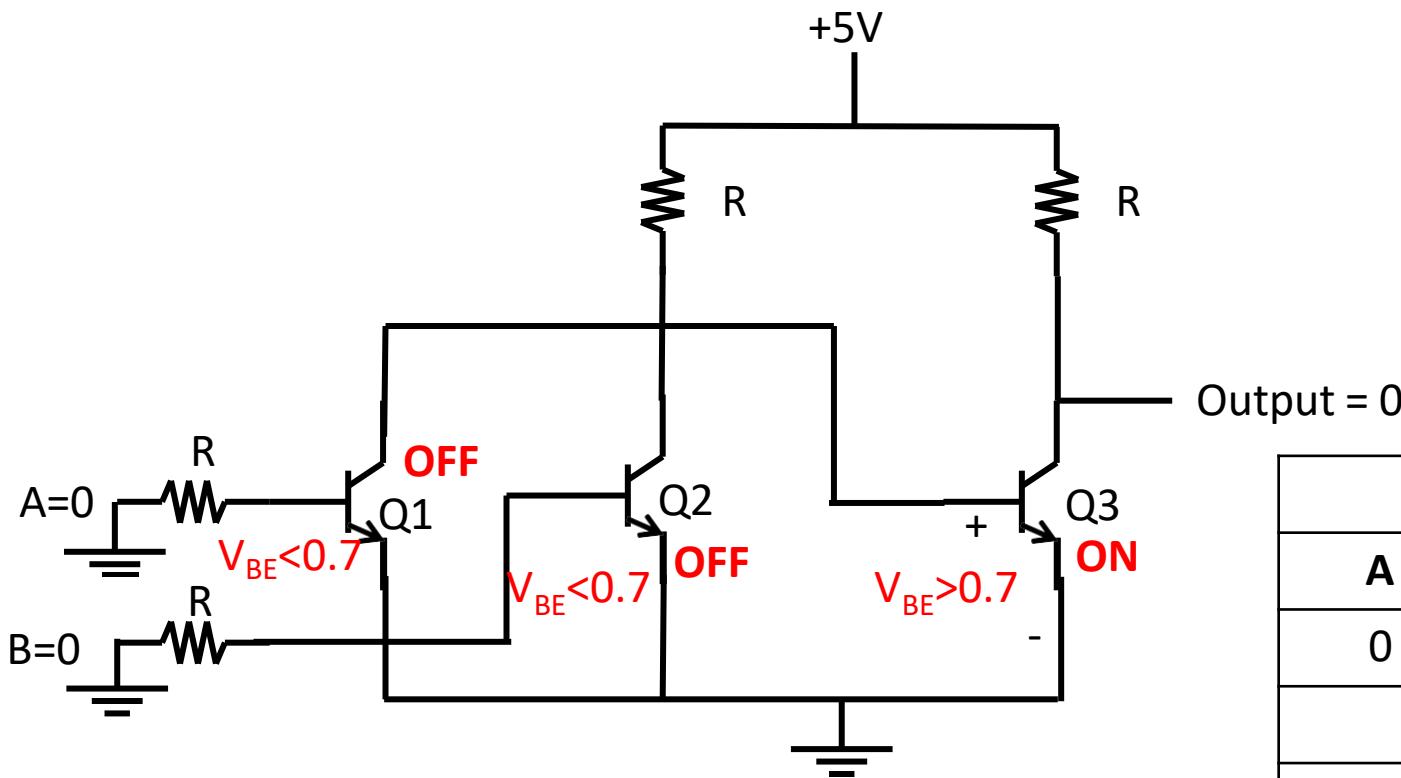
Inputs		Output
A	B	$Y=A+B$
0	0	0
0	1	1
1	0	1
1	1	1

Realization of OR Gate Using Transistors



Realization of OR Gate Using Transistors

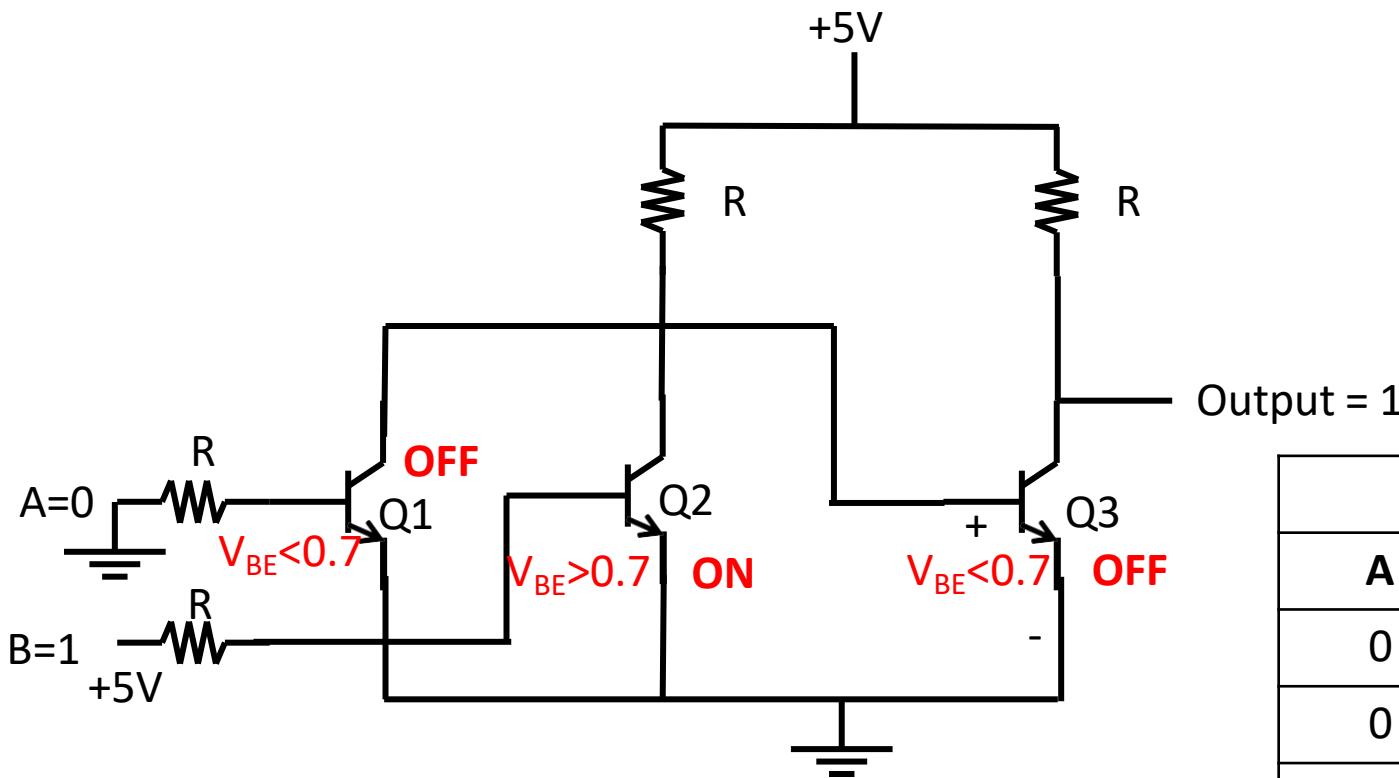
Case I: When A = 0 and B = 0



Inputs		Output
A	B	$Y=A.B$
0	0	0

Realization of OR Gate Using Transistors

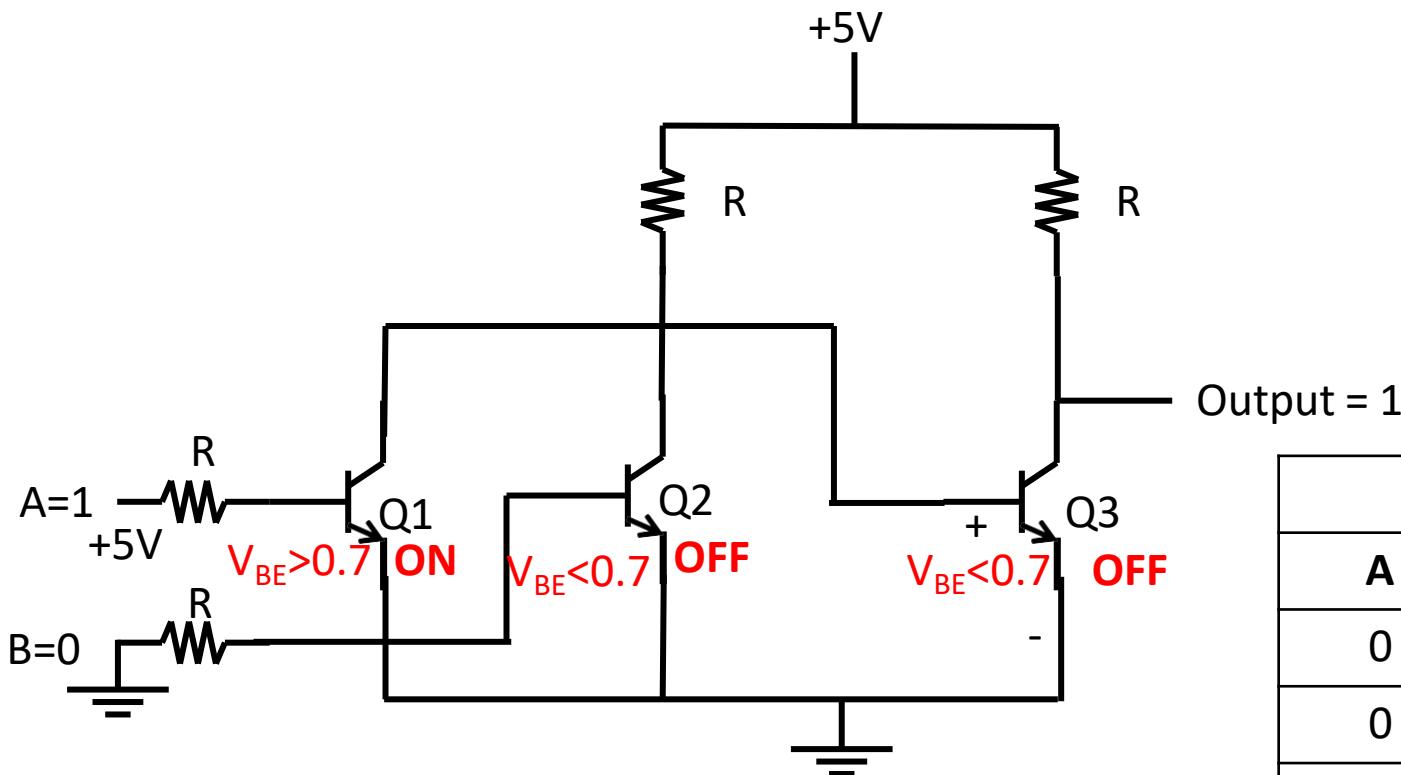
Case II: When A = 0 and B = 1



Inputs		Output
A	B	$Y=A.B$
0	0	0
0	1	1

Realization of OR Gate Using Transistors

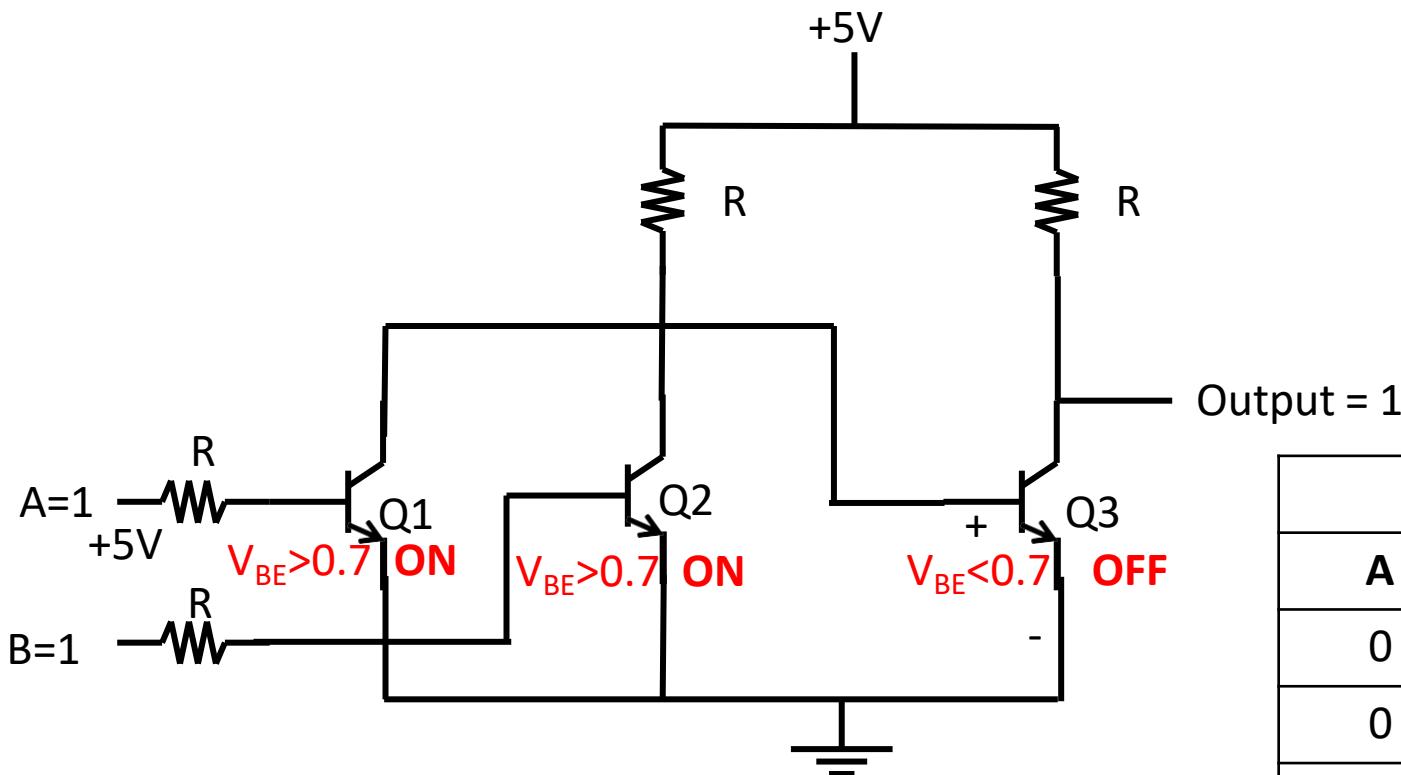
Case III: When A = 1 and B = 0



Inputs		Output
A	B	Y=A.B
0	0	0
0	1	1
1	0	1

Realization of OR Gate Using Transistors

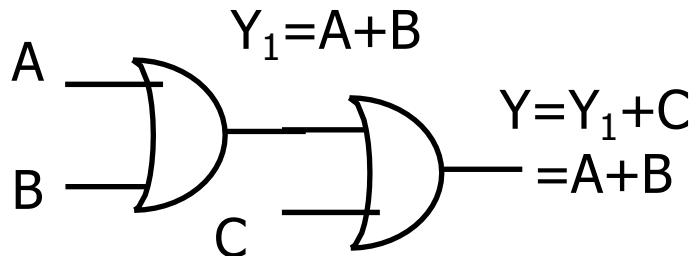
Case IV: When A = 1 and B = 1



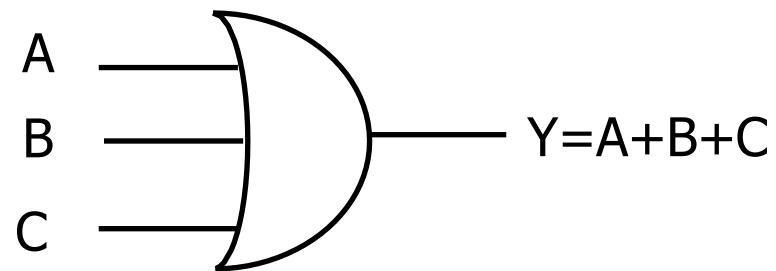
Inputs		Output
A	B	$Y=A.B$
0	0	0
0	1	1
1	0	1
1	1	1

Three Input OR Gate

3 – Input OR Gate using 2 – Input OR Gate



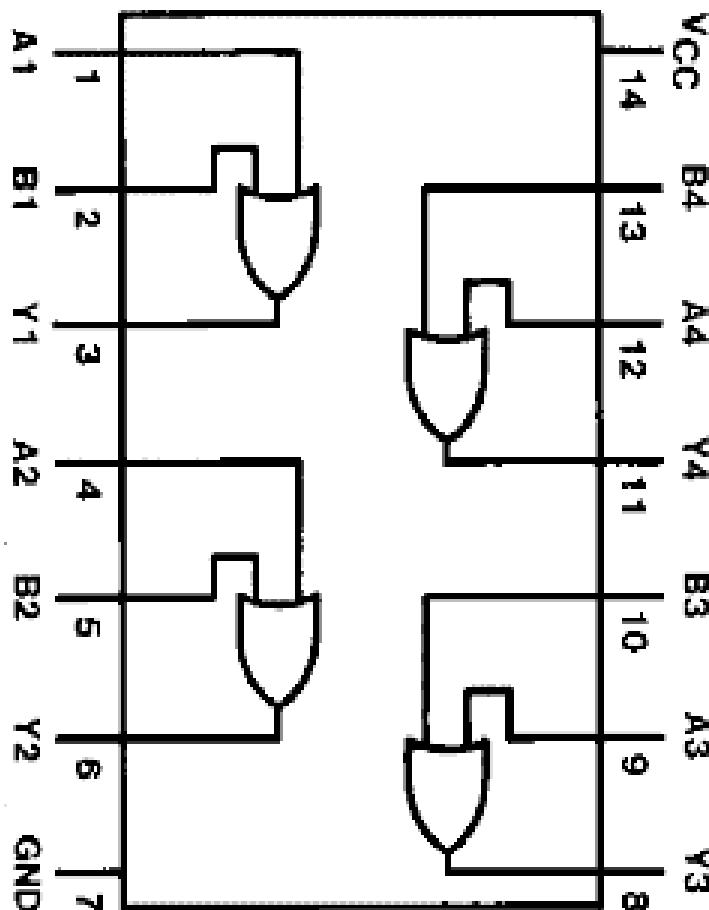
Symbol : 3 – Input OR Gate



Truth Table : 3 – Input OR Gate

Input			Output $Y=A+B+C$
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

OR Gate IC – IC 7432 (Quad 2 I/P OR Gate)



Pin Configuration of IC 7432

✓ This device contains four independent gates each of which performs the logic OR function

$$Y = A + B$$

Inputs		Output
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	H

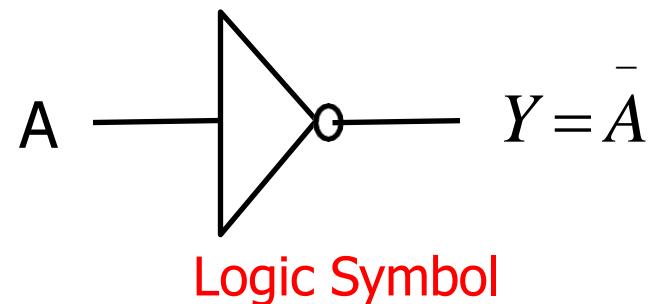
H = High Logic Level

L = Low Logic Level

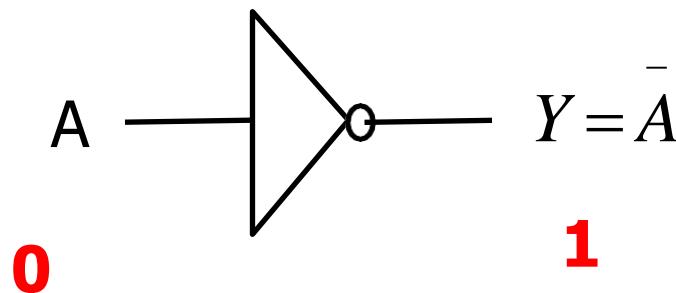
Function Table of IC 7432

NOT Gate (Inverter)

- ✓ A NOT gate, also called inverter, has only one input and of course only one output.
- ✓ It is a device whose output is always the complement of its input.
- ✓ That is, the output of a NOT gate assumes the logic 1 state when its input is in logic 0 state and vice versa.

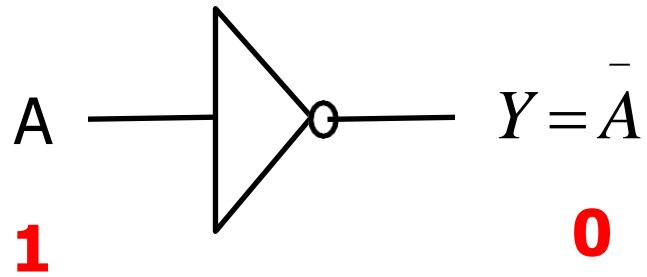


NOT Gate



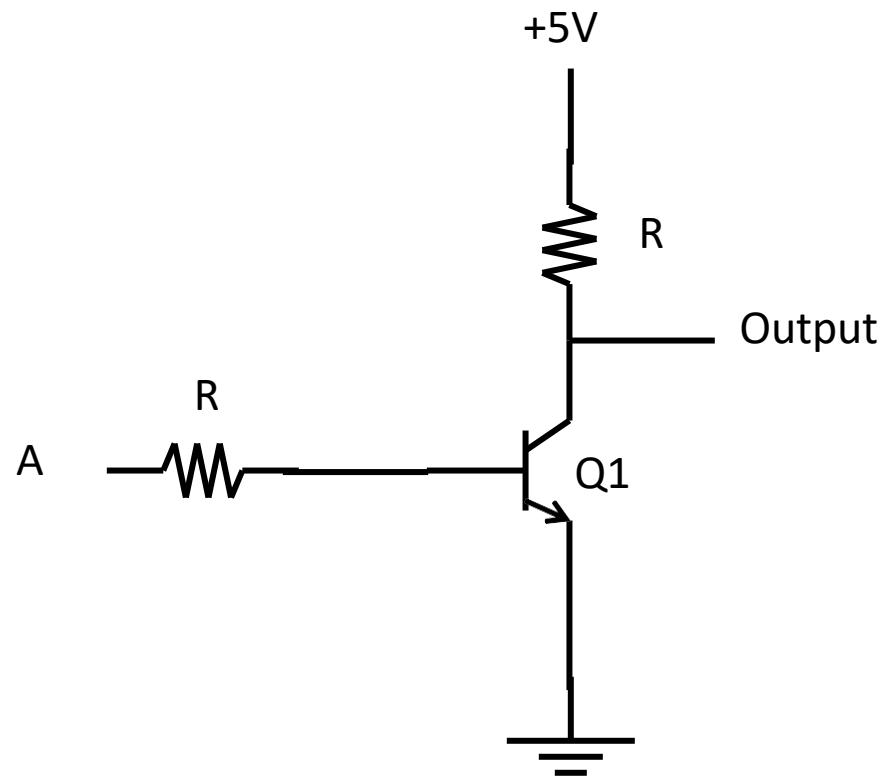
Input	Output
A	$Y = \bar{A}$
0	1

NOT Gate



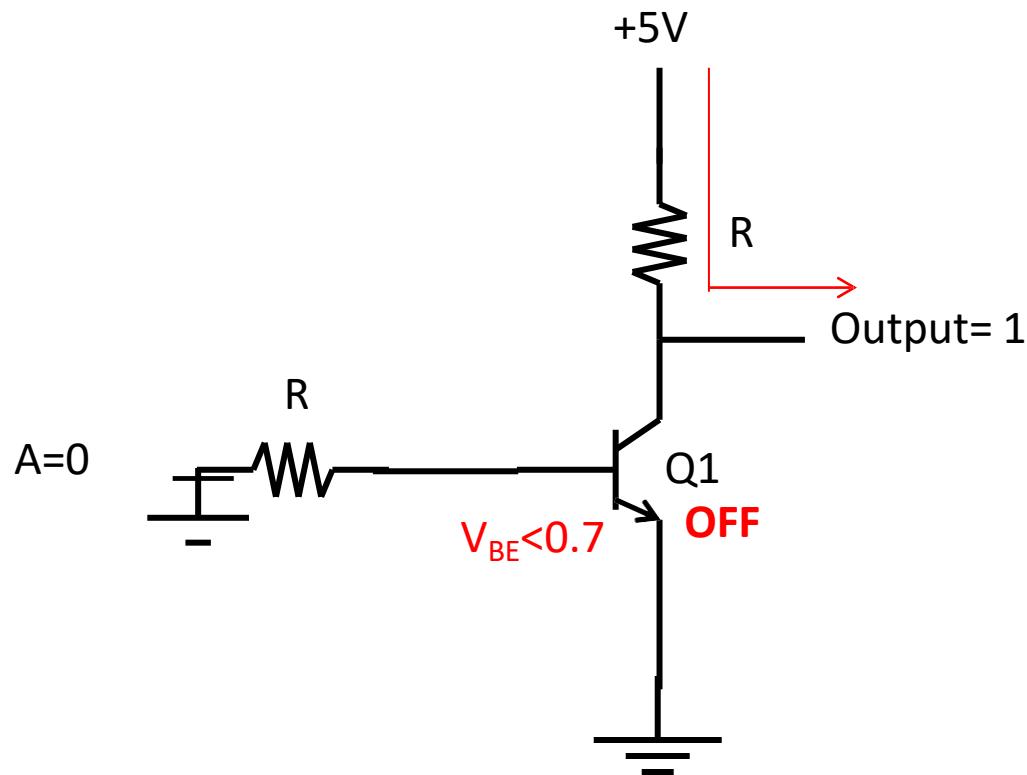
Input	Output
A	$\bar{Y} = \bar{A}$
0	1
1	0

Realization of NOT Gate using Transistor



Realization of NOT Gate using Transistor

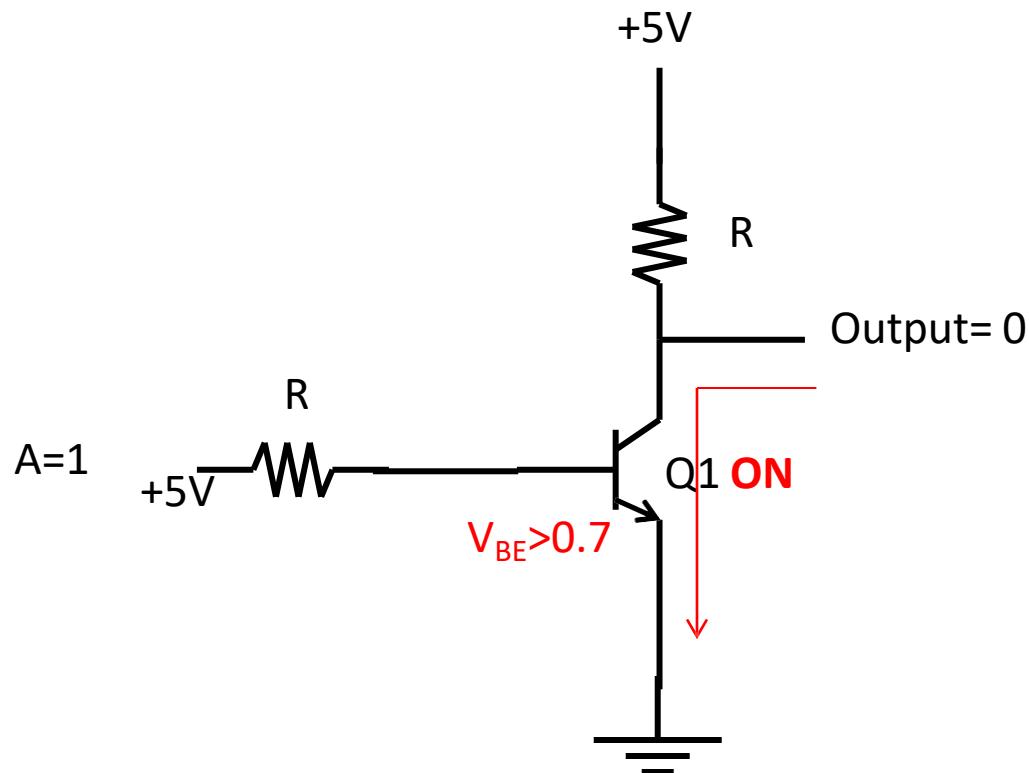
Case I: When $A = 0$



Input	Output
A	$Y = \bar{A}$
0	1

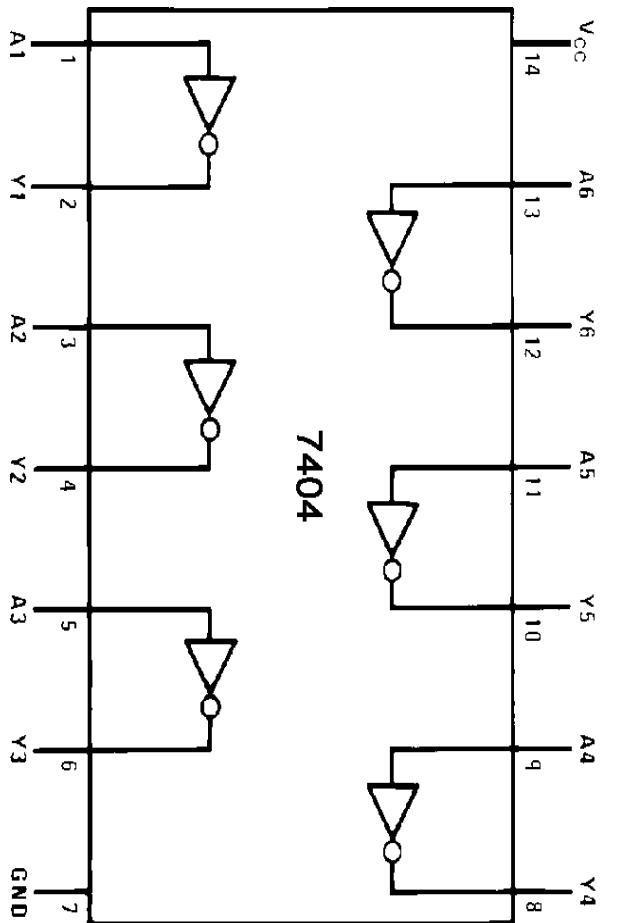
Realization of NOT Gate using Transistor

Case II: When $A = 1$



Input	Output
A	$Y = \bar{A}$
0	1
1	0

NOT Gate IC – IC 7404 (Hex Inverters)



✓ This device contains six independent inverters

INPUT A	OUTPUT Y
H	L
L	H

Function Table of IC 7404

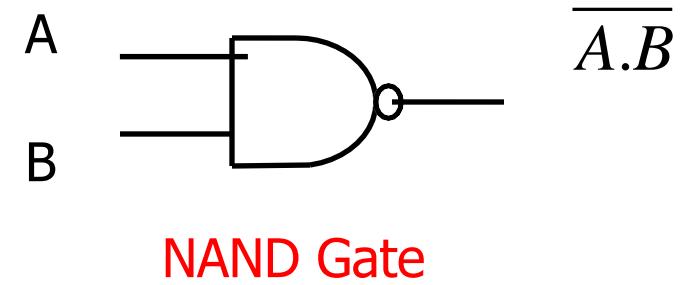
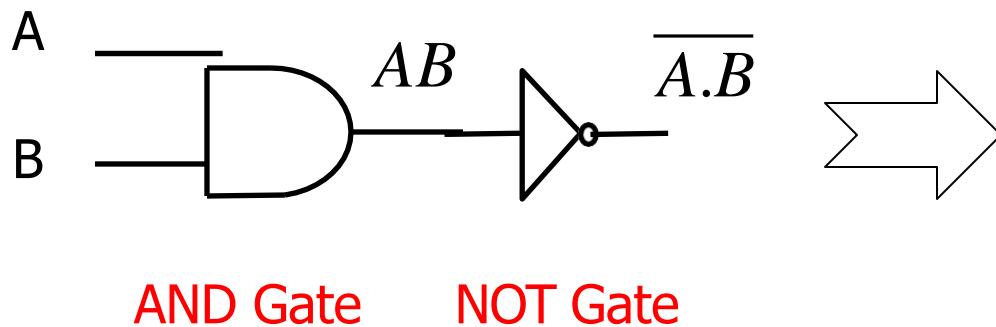
Pin Configuration of IC 7404

Universal Gates (NAND and NOR Gate)

- ✓ NAND and NOR gates are Universal Gates.
- ✓ Both NAND and NOR gates can perform all the three basic logic functions (AND, OR and NOT).
- ✓ Therefore, AOI logic can be converted to NAND logic or NOR logic

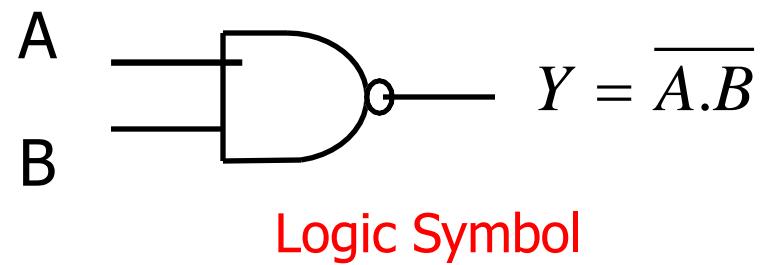
NAND Gate

- ✓ NAND means NOT AND i.e. AND output is inverted.
- ✓ So NAND gate is a combination of an AND gate and a NOT gate.

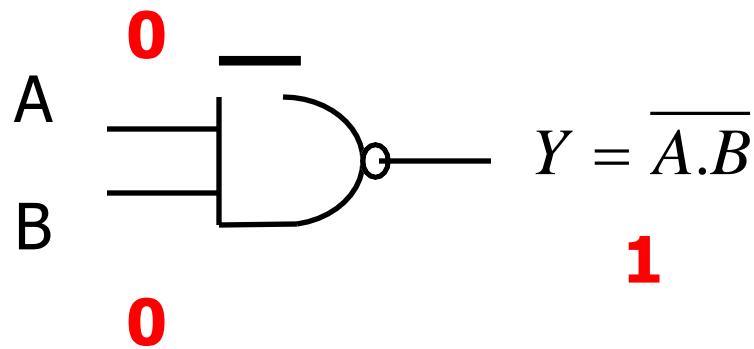


NAND Gate

- ✓ The output is logic 0 level, only when all the inputs are logic 1 level.
- ✓ For any other combination of inputs, the output is a logic 1 level.

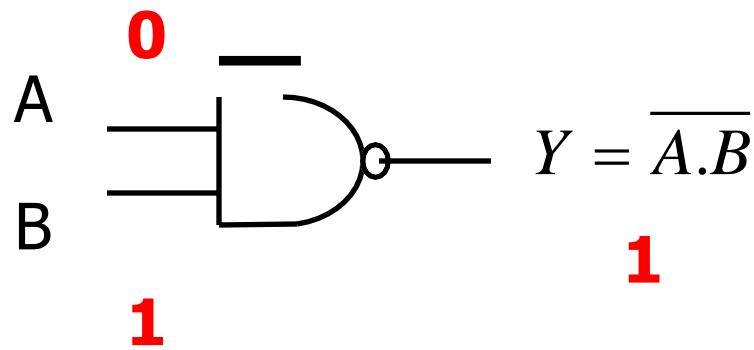


NAND Gate



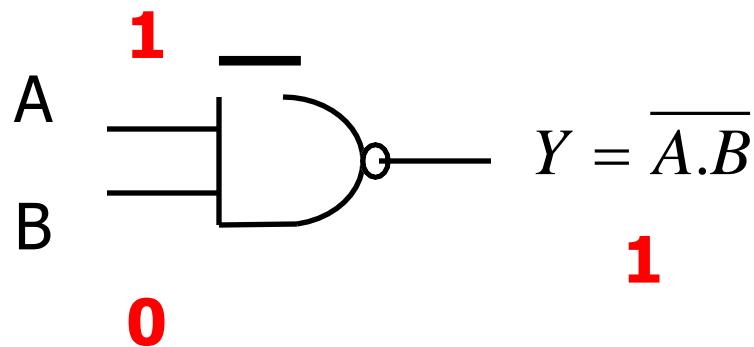
Inputs		Output
A	B	$Y = \overline{A.B}$
0	0	1

NAND Gate



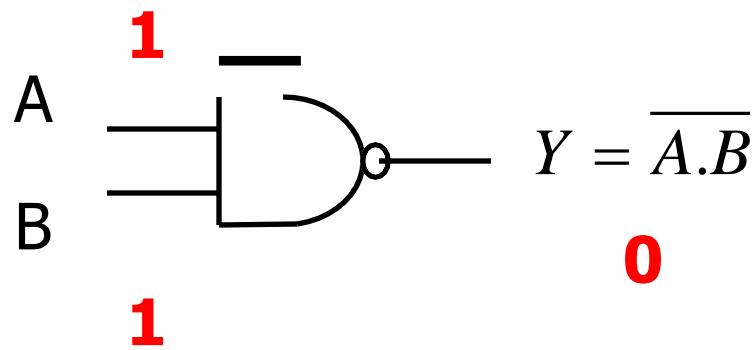
Inputs		Output
A	B	$Y = \overline{A \cdot B}$
0	0	1
0	1	1

NAND Gate



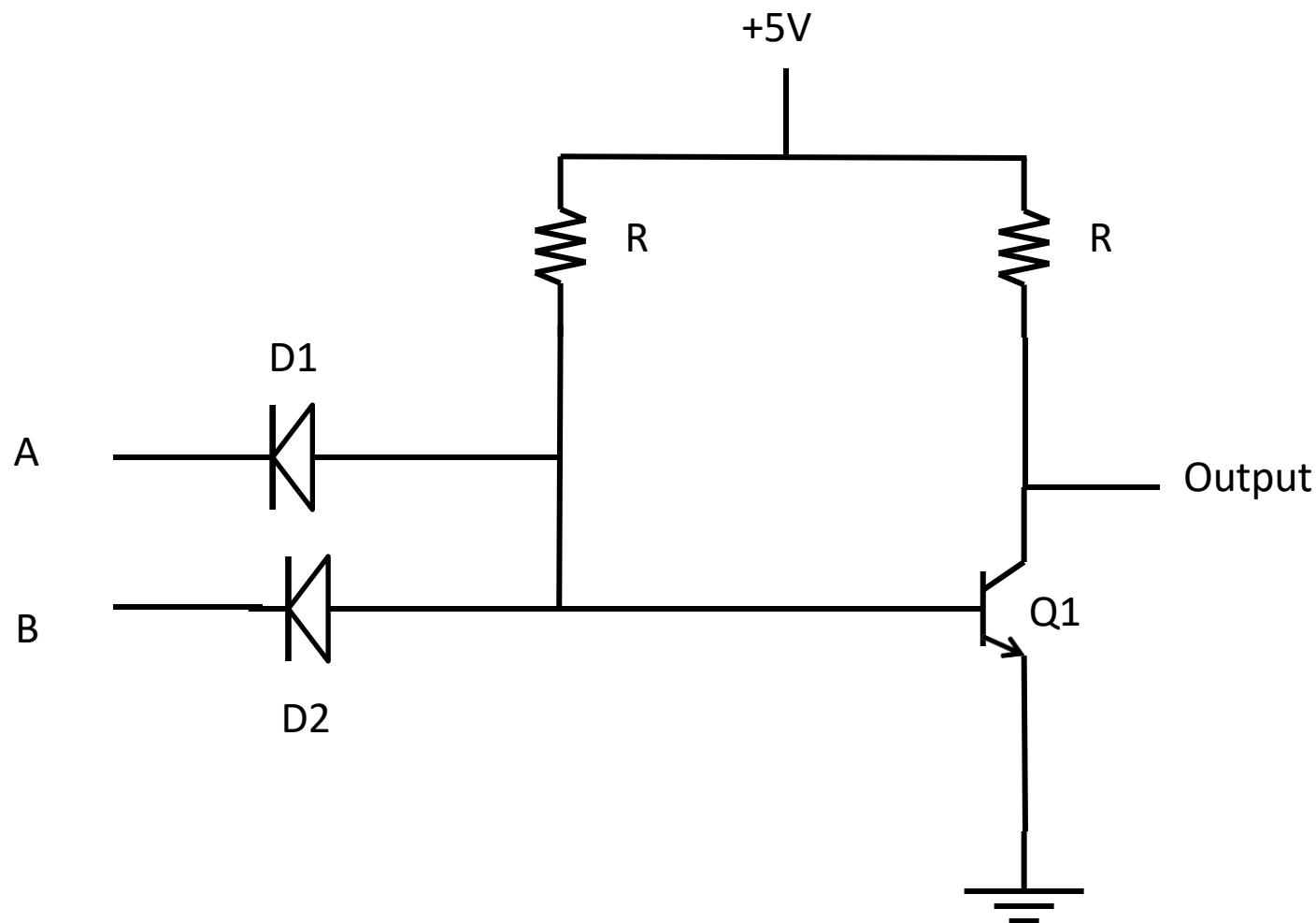
Inputs		Output
A	B	$Y = \overline{A.B}$
0	0	1
0	1	1
1	0	1

NAND Gate

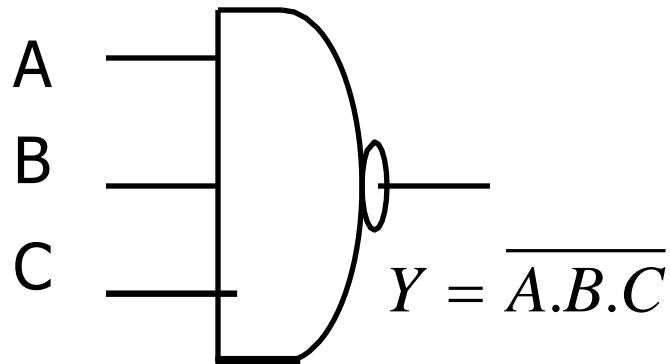


Inputs		Output
A	B	$Y = \overline{A} \cdot \overline{B}$
0	0	1
0	1	1
1	0	1
1	1	0

Realization of NAND Gate using Transistor

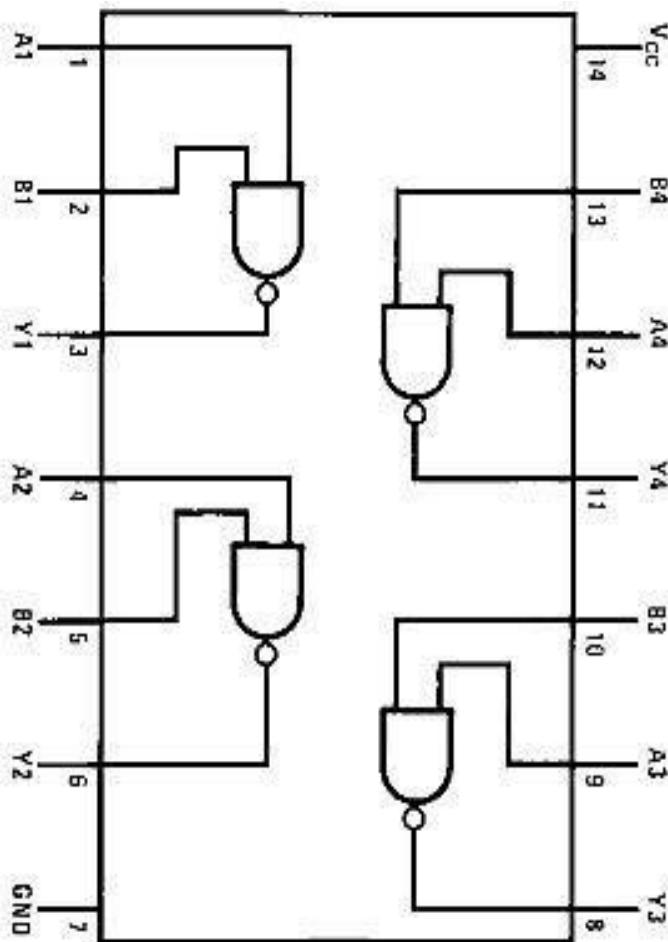


3 - Input NAND Gate



INPUT			OUTPUT
A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

NAND Gate IC – IC 7400 (Quad 2 I/P NAND Gate)



Pin Configuration of IC 7400

- ✓ This device contains four independent gates each of which performs the logic NAND function.

$$Y = \overline{AB}$$

Inputs		Output
A	B	Y
L	L	H
L	H	H
H	L	H
H	H	L

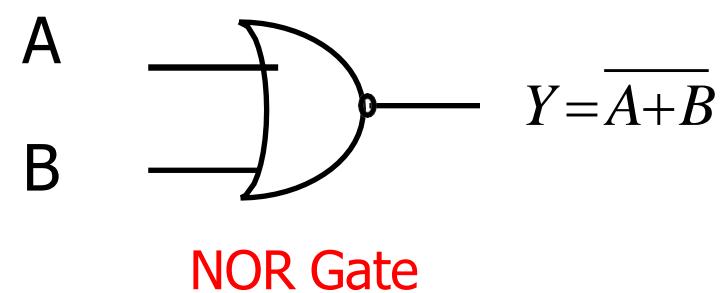
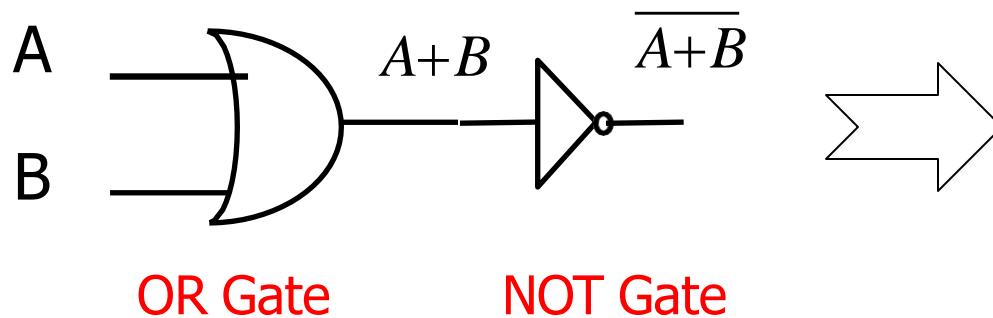
H = HIGH Logic Level

L = LOW Logic Level

Function Table of IC 7400

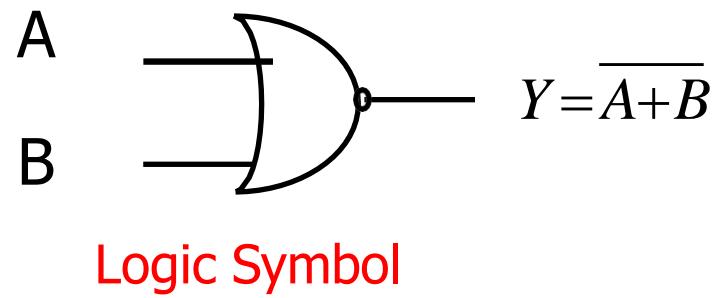
NOR Gate

- ✓ NOR means NOT OR i.e. OR output is inverted.
- ✓ So NOR gate is a combination of an OR gate and a NOT gate.

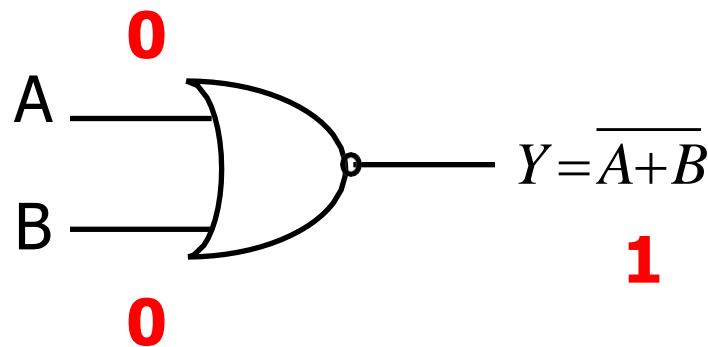


NOR Gate

- ✓ The output is logic 1 level, only when all the inputs are logic 0 level.
- ✓ For any other combination of inputs, the output is a logic 0 level.

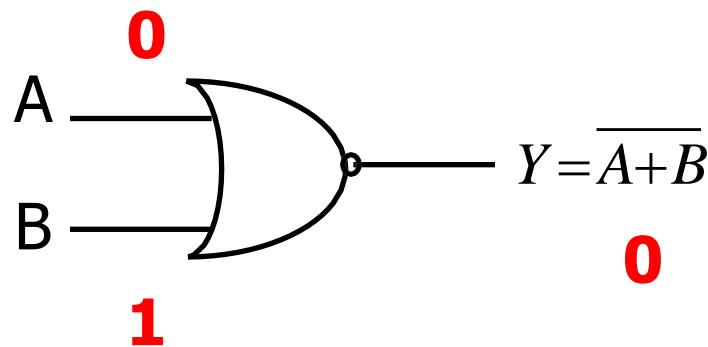


NOR Gate



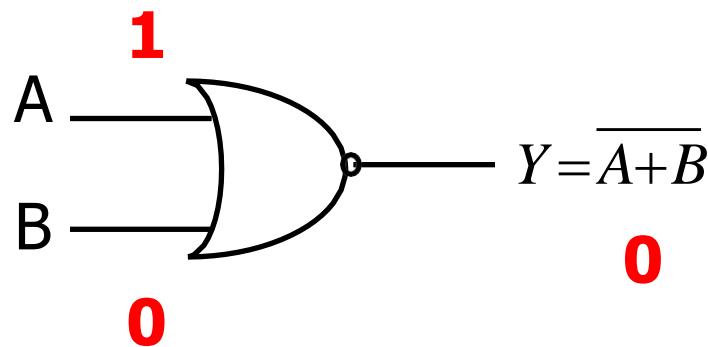
Inputs		Output
A	B	$Y = \overline{A+B}$
0	0	1

NOR Gate



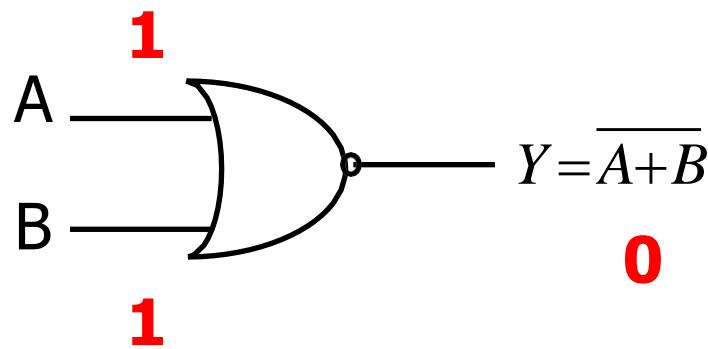
Inputs		Output
A	B	$Y = \overline{A+B}$
0	0	1
0	1	0

NOR Gate



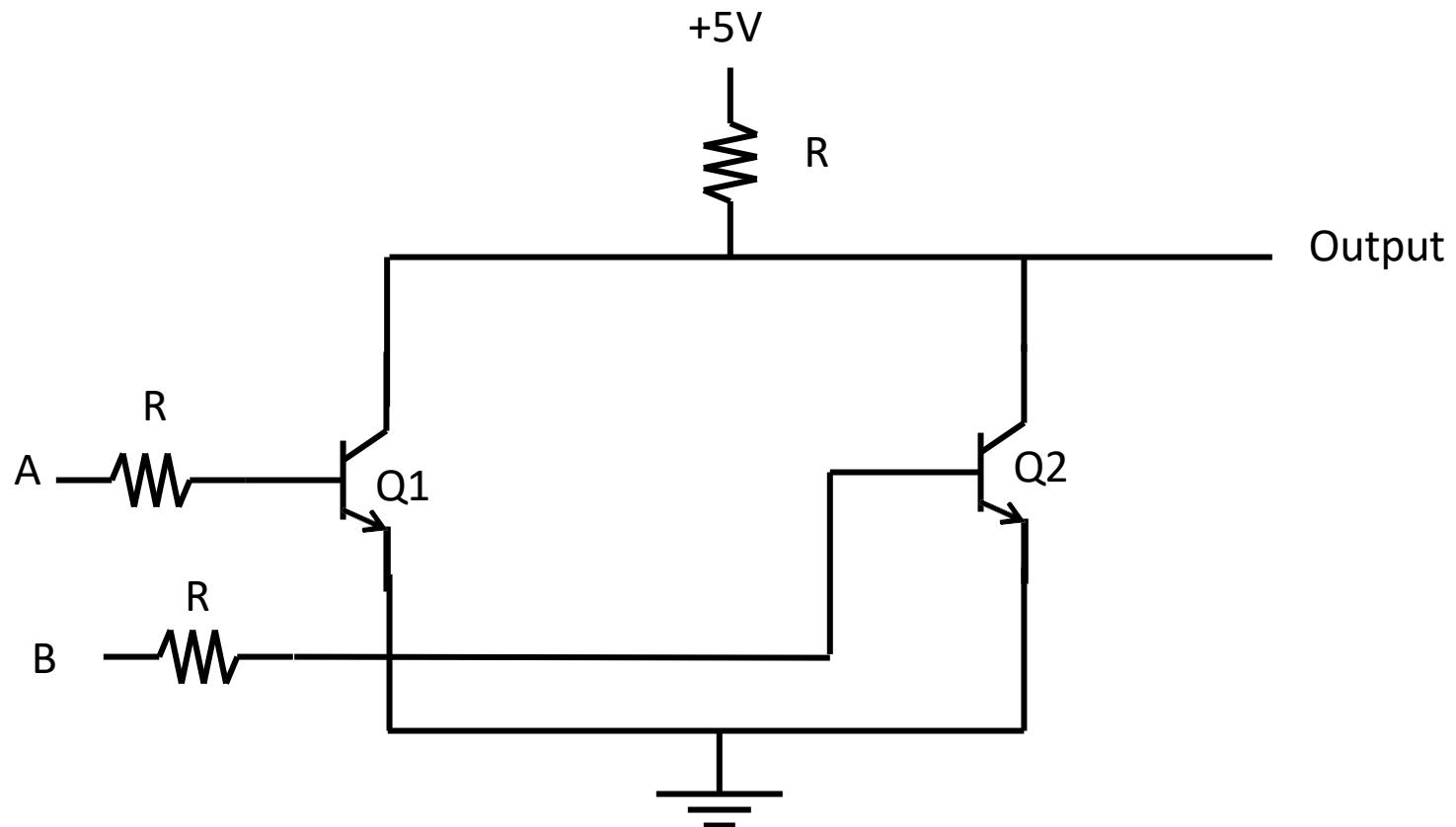
Inputs		Output
A	B	$Y = \overline{A+B}$
0	0	1
0	1	0
1	0	0

NOR Gate

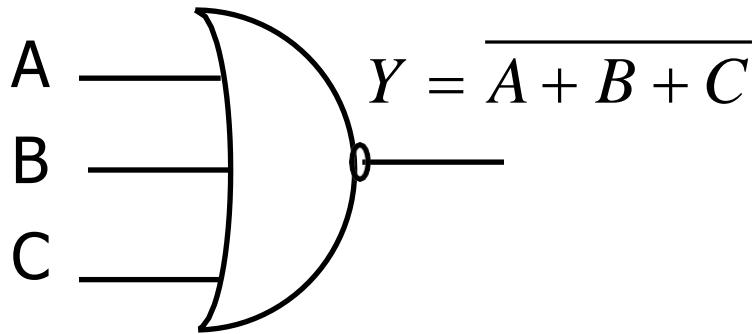


Inputs		Output
A	B	$Y = \overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Realization of NOR Gate using Transistor

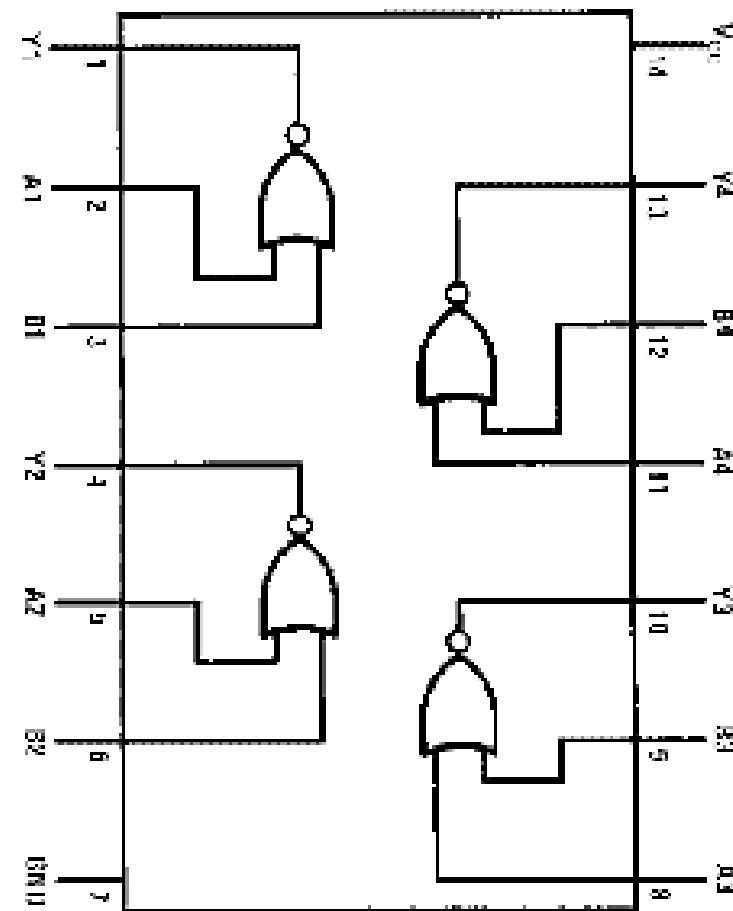


3 – Input NOR Gate



INPUT			OUTPUT
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

NOR Gate IC – IC 7402 (Quad 2 I/P NOR Gate)



Pin Configuration of IC 7402

✓ This device contains four independent gates each of which performs the logic NOR function.

$$Y = \overline{A + B}$$

Inputs		Output
A	B	Y
L	L	H
L	H	L
H	L	L
H	H	L

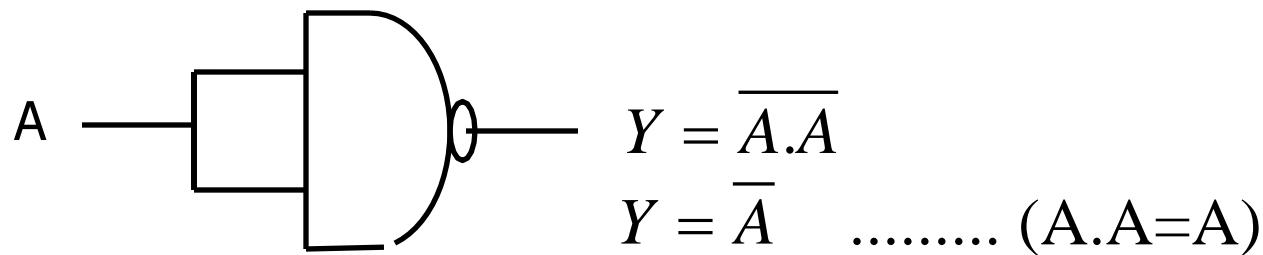
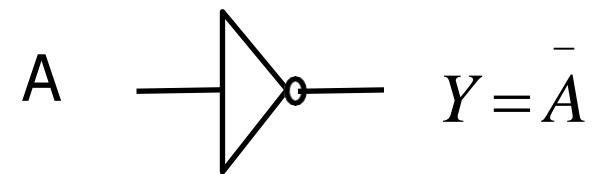
H = HIGH Logic Level

L = LOW Logic Level

Function Table of IC 7402

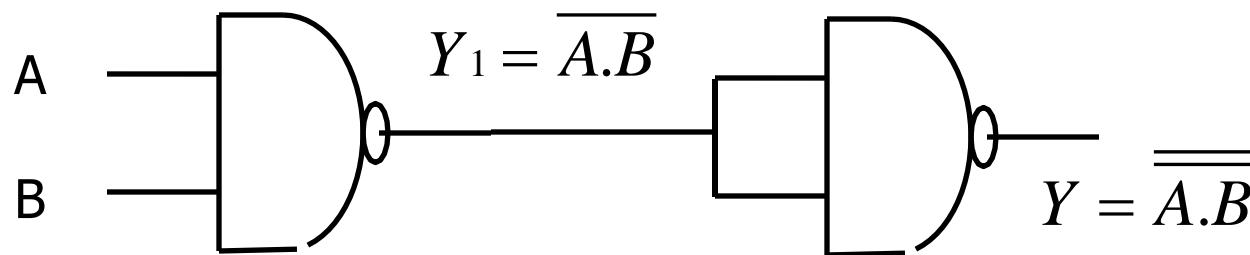
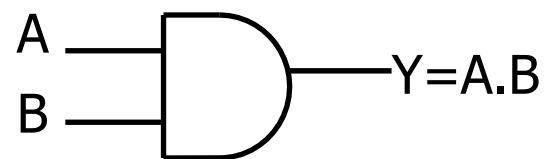
Universal Gate

NOT Gate using NAND Gate



Universal Gate

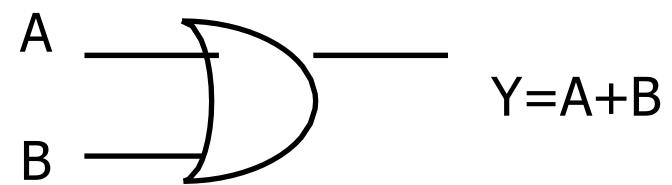
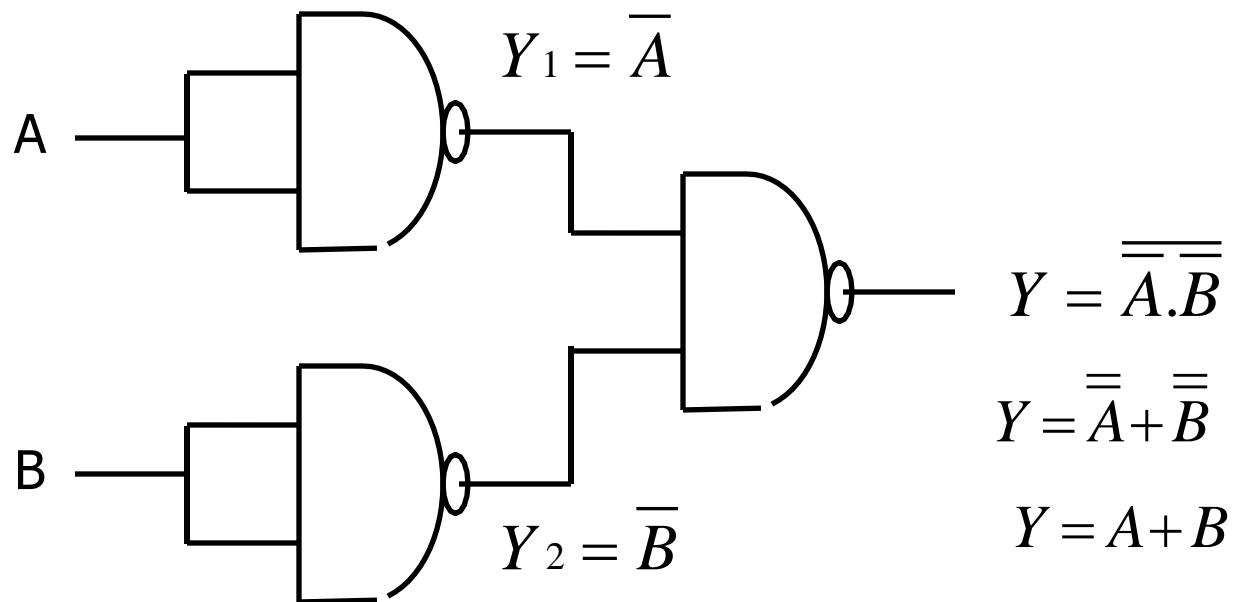
AND Gate using NAND Gate



$$Y = A \cdot B \quad (\because \overline{\overline{A}} = A)$$

Universal Gate

OR Gate using NAND Gate



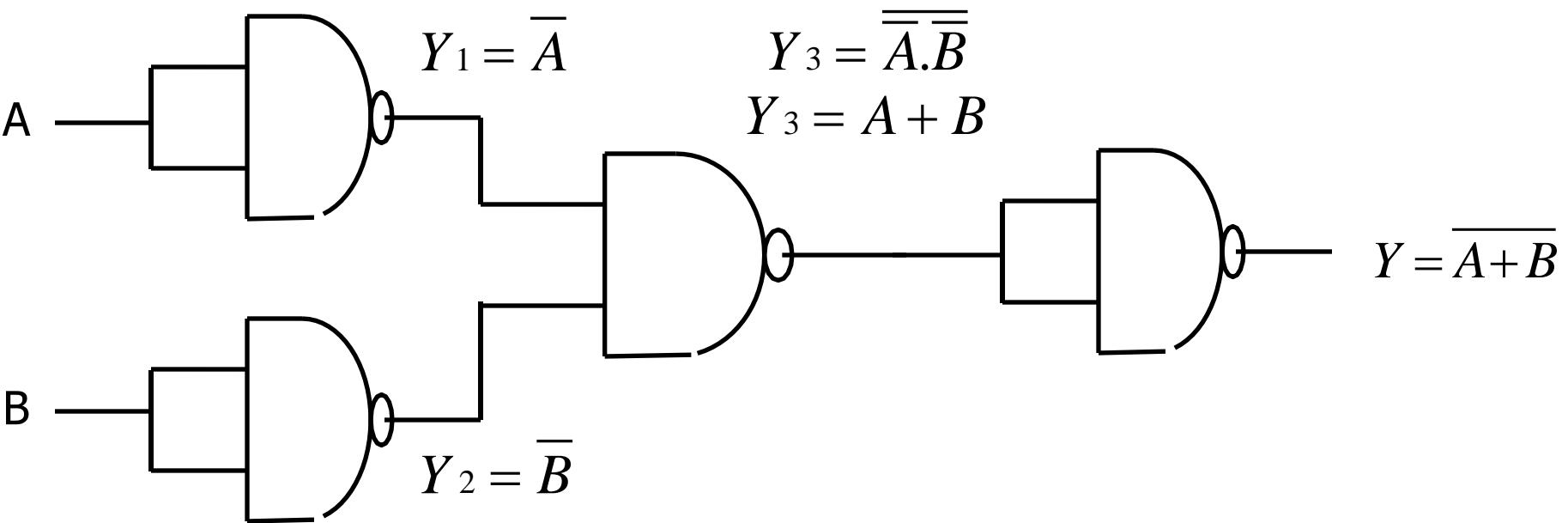
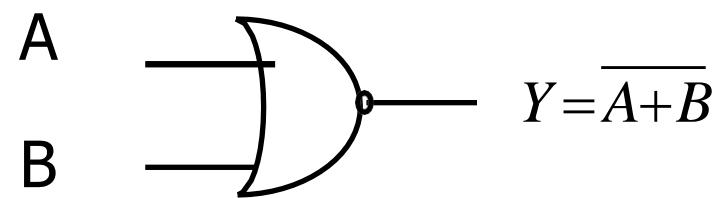
$$Y = \overline{\overline{A} \cdot \overline{B}}$$
$$Y = \overline{\overline{A}} + \overline{\overline{B}}$$

$$Y = A + B$$

(\because Demorgan's Theorem)

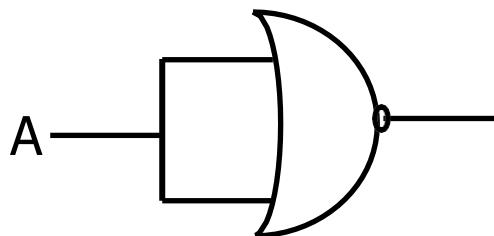
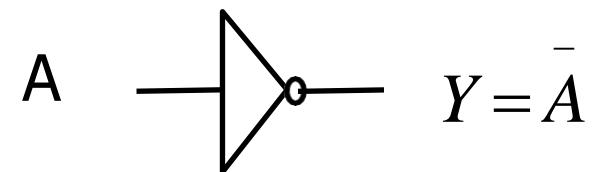
Universal Gate

NOR Gate using NAND Gate



Universal Gate

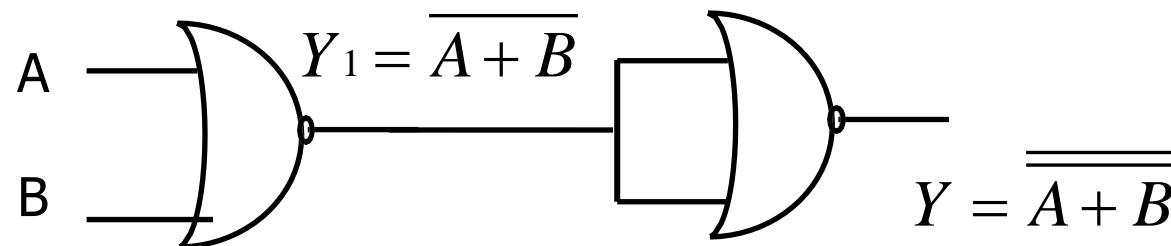
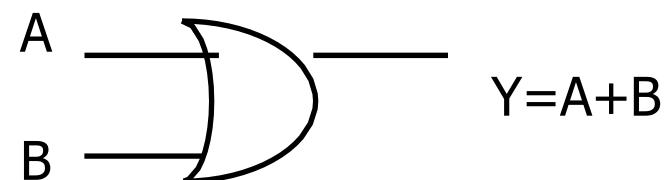
NOT Gate using NOR Gate



$$Y = \overline{\bar{A} + A}$$
$$Y = \overline{\bar{A}} \quad \dots\dots\dots (A + A = A)$$

Universal Gate

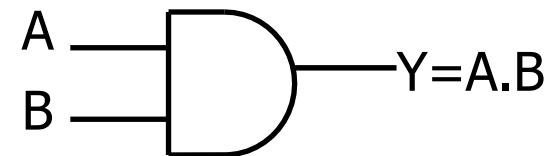
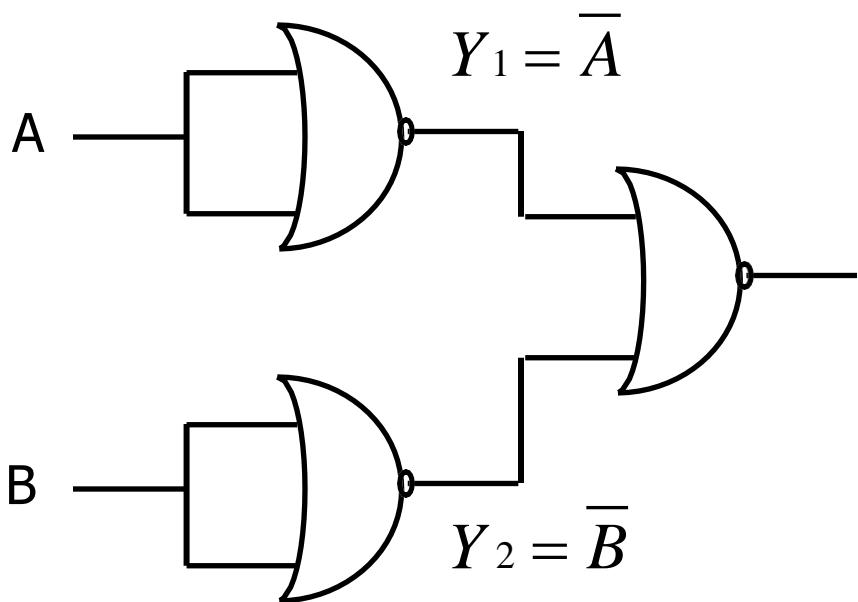
OR Gate using NOR Gate



$$Y = A + B \quad (\because \overline{\overline{A}} = A)$$

Universal Gate

AND Gate using NOR Gate



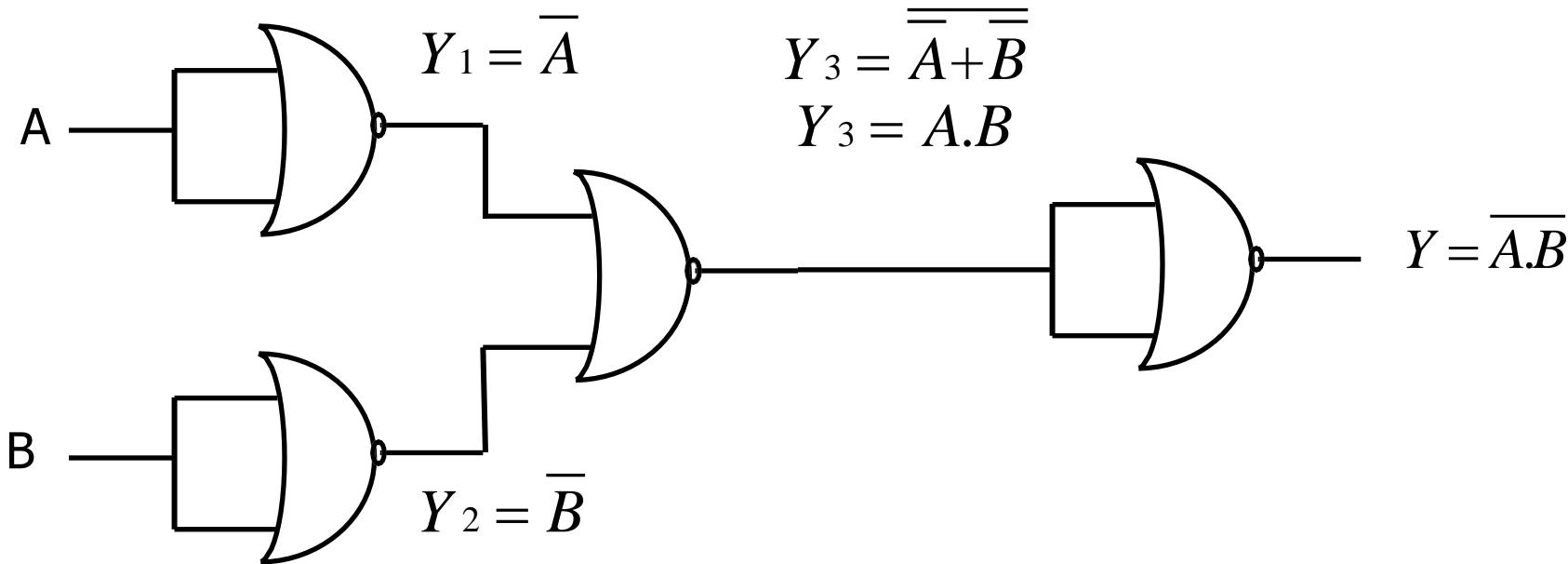
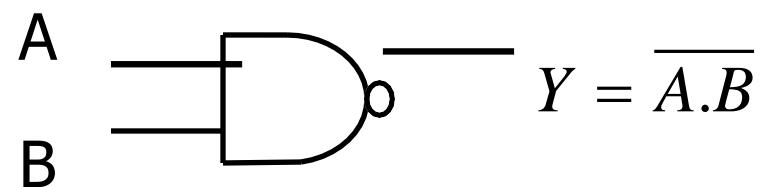
$$Y = \overline{\overline{A} + \overline{B}}$$

$$Y = \overline{\overline{\overline{A}} \cdot \overline{\overline{B}}} \quad (\because \text{Demorgan's Theorem})$$

$$Y = A \cdot B$$

Universal Gate

NAND Gate using NOR Gate

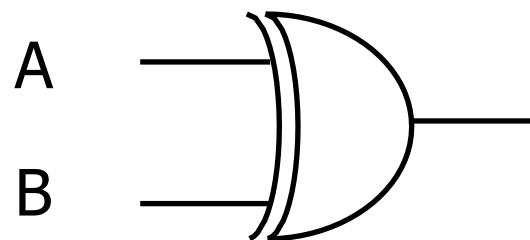


Special Purpose Gate – Ex-OR Gate

- ✓ An Ex-OR gate is two input, one output logic circuit.
- ✓ The output assumes the logic 1 state, when one and only one of its two inputs assumes a logic 1 state.
- ✓ Under the conditions when both the inputs assume the logic 0 state or logic 1 state, the output assumes logic 0.

Ex-OR Gate

✓ If input variables are represented by A and B and the output variable by Y the representation for the output of this gate is as



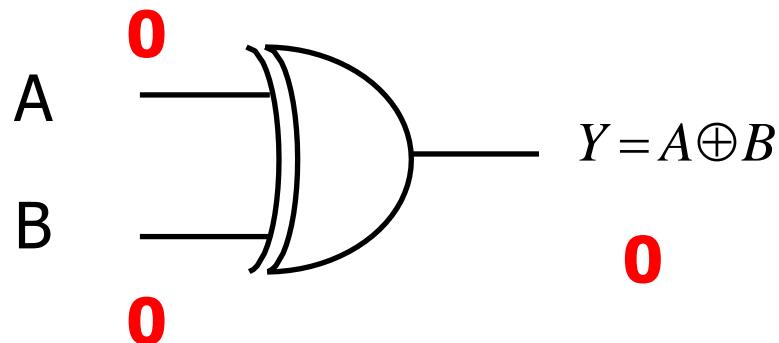
$$Y = A \oplus B$$

$$Y = \overline{A} \overline{B} + A \overline{B}$$

Logic Symbol

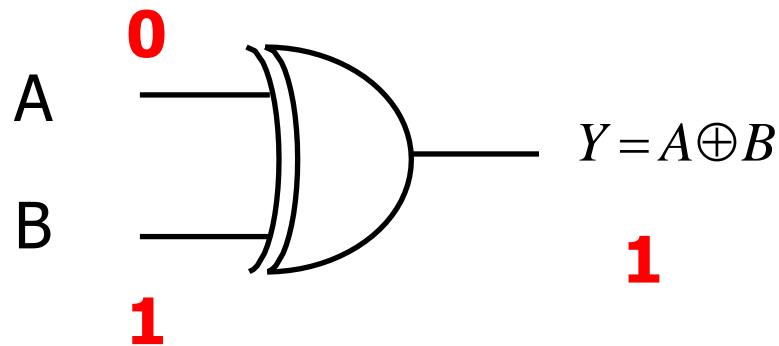
Logic Expression

Ex-OR Gate



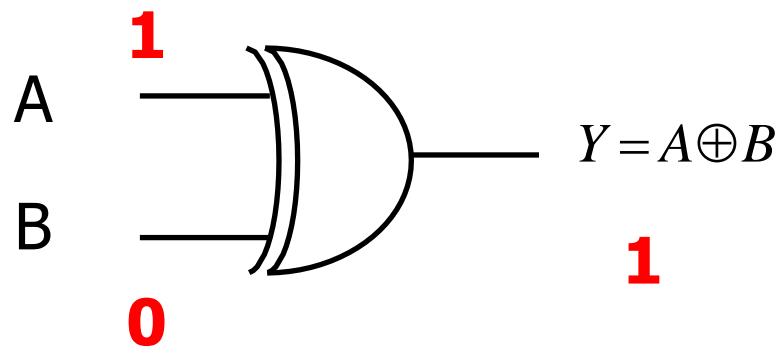
Inputs		Output
A	B	$Y = A \oplus B$
0	0	0

Ex-OR Gate



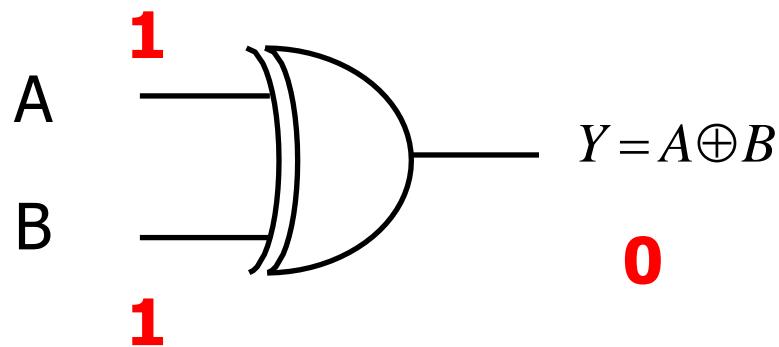
Inputs		Output
A	B	$Y = A \oplus B$
0	0	0
0	1	1

Ex-OR Gate



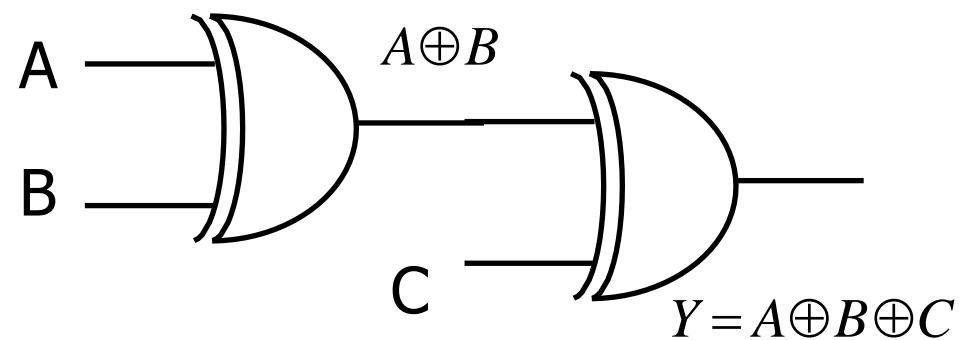
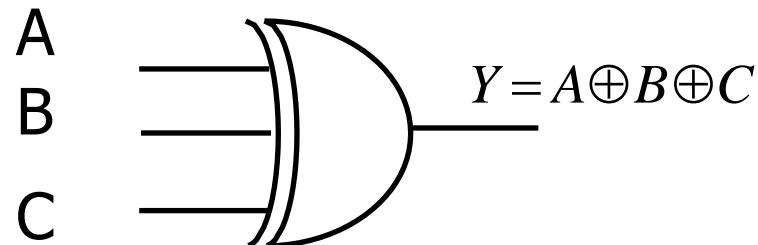
Inputs		Output
A	B	$Y = A \oplus B$
0	0	0
0	1	1
1	0	1

Ex-OR Gate



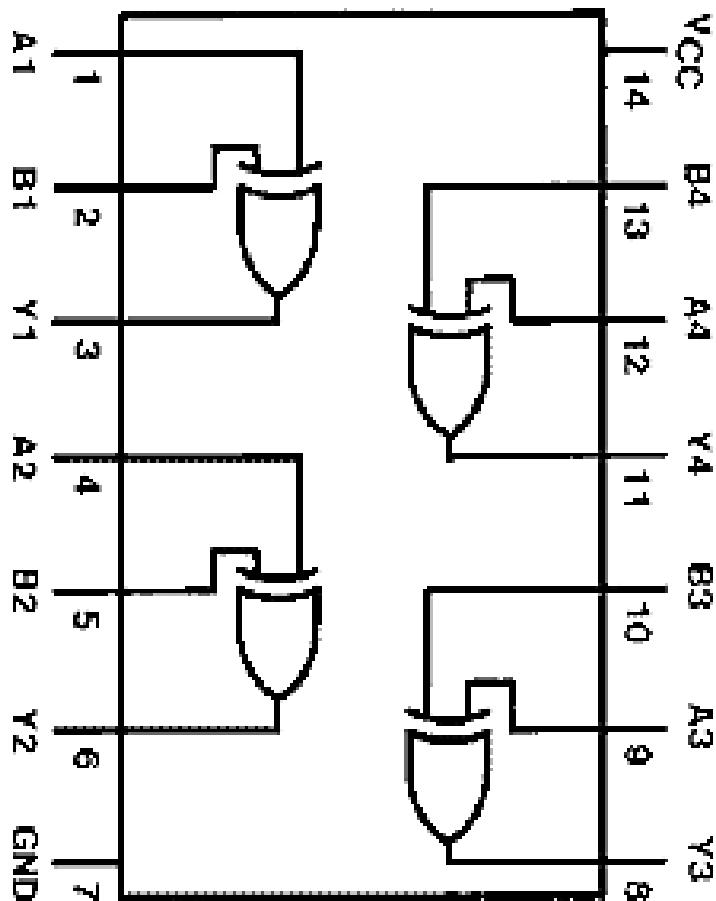
Inputs		Output
A	B	$Y = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

3 – Input Ex-OR Gate



INPUT			OUTPUT
A	B	C	$Y = A \oplus B \oplus C$
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

Ex-OR Gate IC – IC 7486 (Quad 2 I/P Ex-OR Gate)



Pin Configuration of IC 7486

- ✓ This device contains four independent gates each of which performs the logic XOR function.

INPUTS		OUTPUT
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	L

H = high level, L = low level

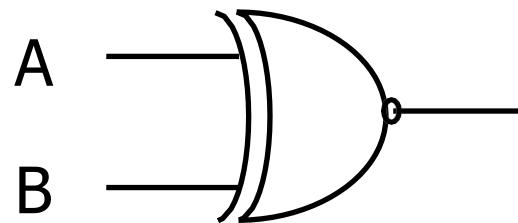
Function Table of IC 7486

Special Purpose Gate – Ex-NOR Gate

- ✓ An Ex-NOR gate is two input, one output logic circuit.
- ✓ The output assumes a logic 0 state, when one of the input assumes a logic 0 state and other a logic 1 state.
- ✓ The output assumes a logic 1 state only when both the inputs assume a logic 0 state or when both the inputs assume a logic state.

Ex-NOR Gate

✓ If input variables are represented by A and B and the output variable by Y the representation for the output of this gate is as



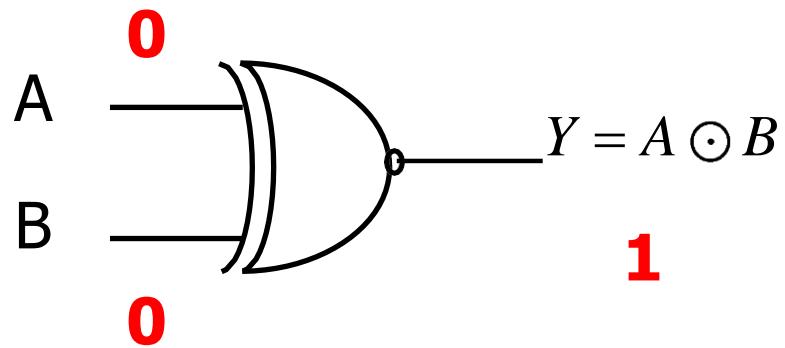
$$Y = A \odot B$$

$$Y = AB + \bar{A} \bar{B}$$

Logic Symbol

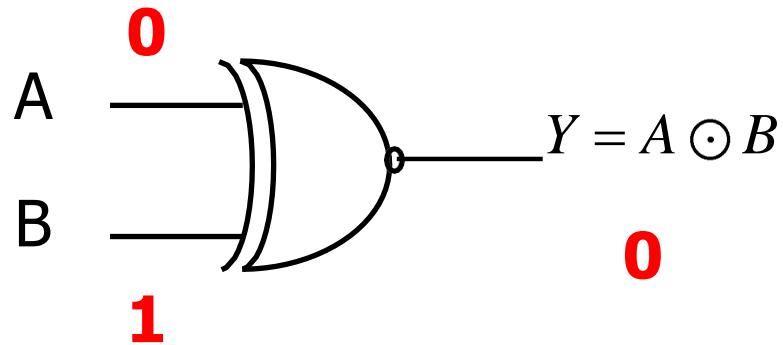
Logic Expression

Ex-NOR Gate



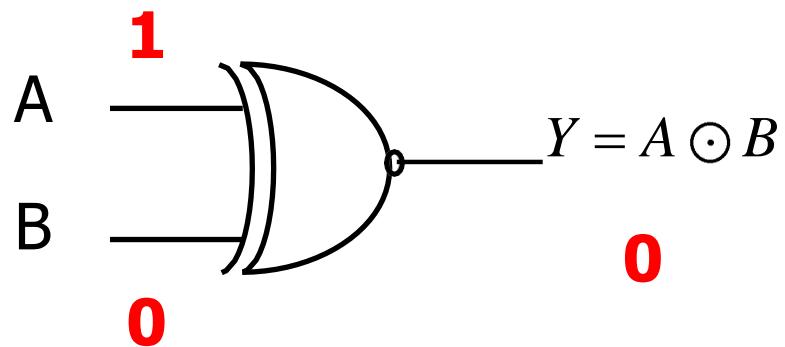
Inputs		Output
A	B	$Y = A \odot B$
0	0	1

Ex-NOR Gate



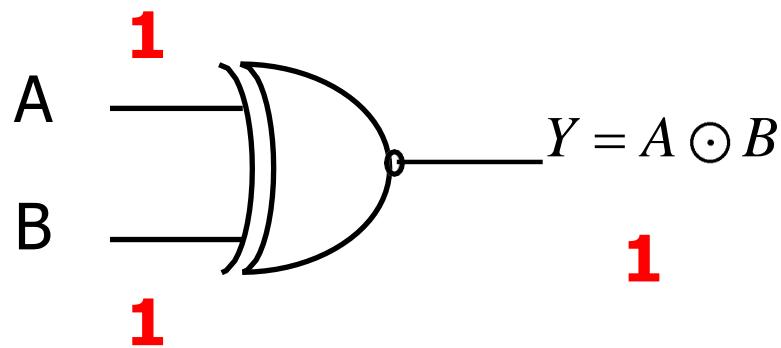
Inputs		Output
A	B	$Y = A \odot B$
0	0	1
0	1	0

Ex-NOR Gate



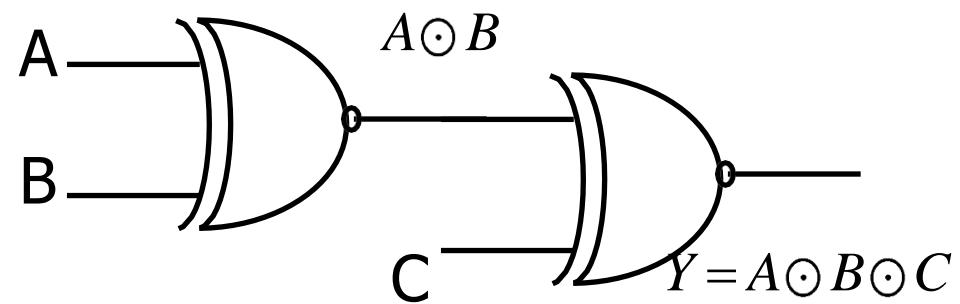
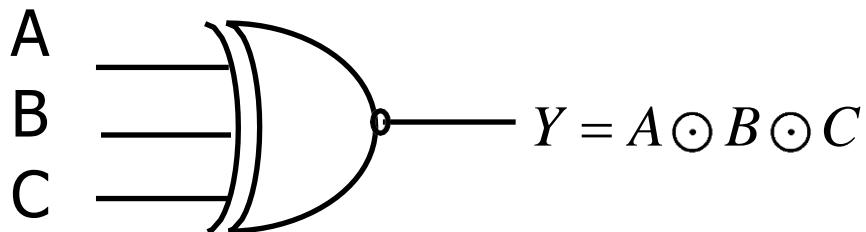
Inputs		Output
A	B	$Y = A \odot B$
0	0	1
0	1	0
1	0	0

Ex-NOR Gate



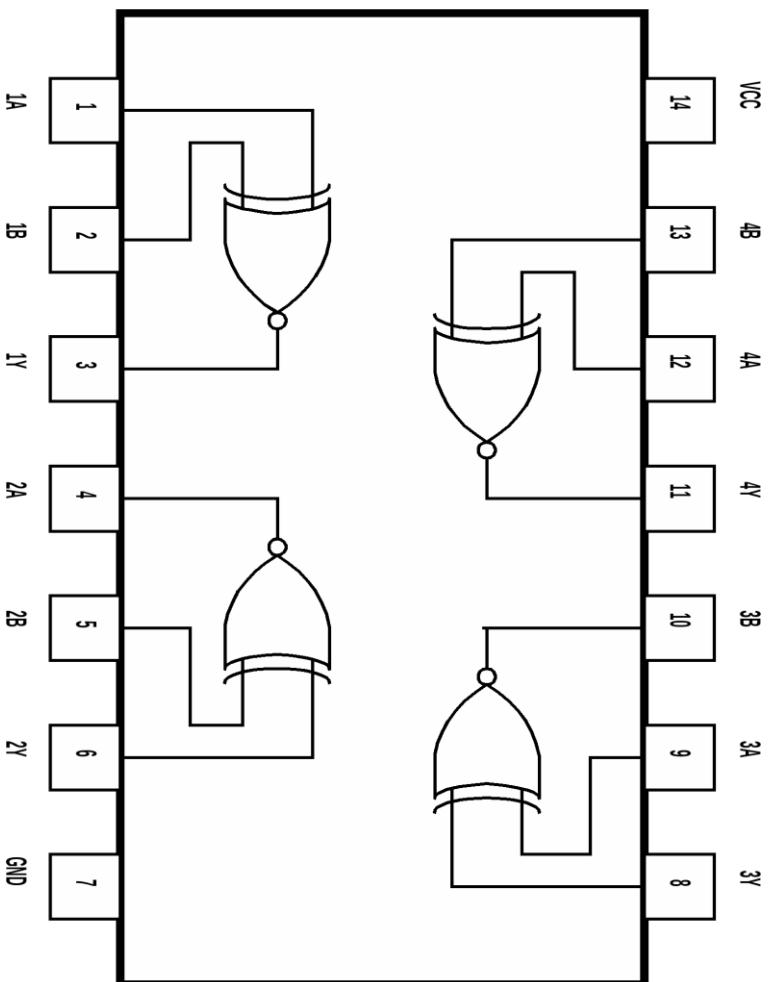
Inputs		Output
A	B	$Y = A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

3 – Input Ex-NOR Gate



INPUT			OUTPUT
A	B	C	$Y = A \odot B \odot C$
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

Ex-NOR Gate IC – IC 74266



Pin Configuration of IC 74266

✓ This device contains four independent gates each of which performs the logic XNOR function.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



Switching Theory and Logic Design

ARI-209

By: Dr. Divya Agarwal



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT- II** 10 HOURS
- **Sequential Logic Circuits:** Latches and Flip Flops- SR, D, T and MS-JK Flip Flops, Asynchronous Inputs.
- **Counters and Shift Registers:** Design of Synchronous and Asynchronous Counters- Binary, BCD, Decade and Up/Down Counters, Shift Registers, Types of Shift Registers, Counters using Shift Registers- Ring Counter and Johnson Counter.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-III** **08 HOURS**
- **Integrated circuits:** TTL and CMOS logic families and their characteristics. Brief introduction to RAM and ROM
- **Synchronous Sequential Circuits:** State Tables State Equations and State Diagrams, State Reduction and State Assignment, Design of Clocked Sequential Circuits using State Equations.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-IV** 06 HOURS
- **Finite state machine:** capabilities and limitations, Mealy and Moore models-minimization of completely specified and incompletely specified sequential machines, Partition techniques and merger chart methods, concept of minimal cover table.
- **Algorithmic State Machine:** Representation of sequential circuits using ASM charts synthesis of output and next state functions, Data path control path partition-based design.



Text Books:

1. Leach and Malvino (2011). *Digital principles and Applications*. Tata McGraw-Hill Education
2. Mano, M. M. (2017). *Digital logic and computer design*. Pearson Education India.
3. Jain, R. P. (2003). *Modern digital electronics*. Tata McGraw-Hill Education.

Reference Books:

1. A Anand Kumar. (2016) *Fundamentals of Digital Logic Circuits*, PHI
2. Taub, H., & Schilling, D. L. (1977). *Digital integrated electronics*. McGraw-Hill College.



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.

Boolean Algebra

- ✓ Boolean Algebra is used to analyze and simplify the digital (Logic) circuit.
- ✓ Since it uses only the binary numbers i.e. 0 and 1 it is also called as “Binary Algebra” or “Logical Algebra”.

Boolean Algebra

- ✓ The rules of Boolean Algebra are different from those of the conventional algebra.
- ✓ It is invented by George Boole in the year 1854.

Boolean Algebra

➤ Axioms

- ✓ Axioms or postulates of Boolean algebra are set of logical expressions that we accept without proof and upon which we can build a set of useful theorems.
- ✓ Actually, axioms are nothing more than the definitions of the three basic logic operations that we have already discussed AND, OR and INVERT.

Boolean Algebra

➤ Axioms

AND Operation

Axiom 1: $0 \cdot 0 = 0$

Axiom 2: $0 \cdot 1 = 0$

Axiom 3: $1 \cdot 0 = 0$

Axiom 4: $1 \cdot 1 = 1$

Boolean Algebra

➤ Axioms

OR Operation

Axiom 5: $0 + 0 = 0$

Axiom 6: $0 + 1 = 1$

Axiom 7: $1 + 0 = 1$

Axiom 8: $1 + 1 = 1$

Boolean Algebra

➤ Axioms

NOT Operation

Axiom 9: $\bar{1} = 0$

Axiom 10: $\bar{0} = 1$

Boolean Algebra

➤ Inversion Law (or Complementation Law)

- ✓ The term complement means to invert i.e. to change 0's to 1's and 1's to 0's.

Law 1: $\bar{1} = 0$

Law 2: $\bar{0} = 1$

Law 3: If $A=0$, then $\bar{A} = 1$

Law 4: If $A=1$, then $\bar{A} = 0$

Law 5: $\bar{\bar{A}} = A$ (Double Inversion Law)

Boolean Algebra

➤ AND Laws

- | | | |
|--------|-----------------------|--------------|
| Law 1: | $A \cdot 0 = 0$ | Null Law |
| Law 2: | $A \cdot 1 = A$ | Identity Law |
| Law 3: | $A \cdot A = A$ | |
| Law 4: | $A \cdot \bar{A} = 0$ | |

Boolean Algebra

➤ OR Laws

Law 1: $A + 0 = A$ Buffer Law

Law 2: $A + 1 = 1$ Identity Law

Law 3: $A + A = A$

Law 4: $A + \bar{A} = 1$

Boolean Algebra

➤ Commutative Laws

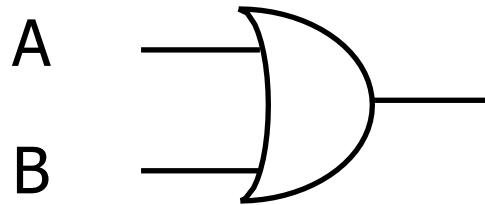
Law 1: $A+B = B+A$

- ✓ This Law states that, A OR B is the same as B OR A i.e. the order in which the variables are ORed is immaterial.
- ✓ This means that it makes no difference which input of an OR gate is connected to A and which to B.

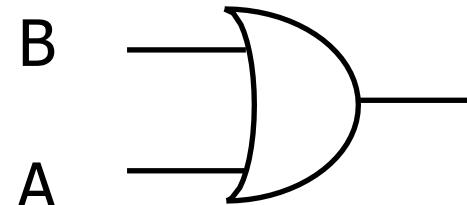
Boolean Algebra

Proof:

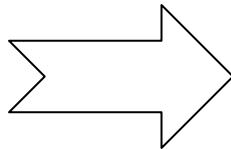
$$A+B$$



$$B+A$$



Inputs		Output
A	B	$Y=A+B$
0	0	0
0	1	1
1	0	1
1	1	1



Inputs		Output
B	A	$Y=B+A$
0	0	0
0	1	1
1	0	1
1	1	1

Boolean Algebra

➤ Commutative Laws

✓ This law can be extended to any number of variables. For example,

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

Boolean Algebra

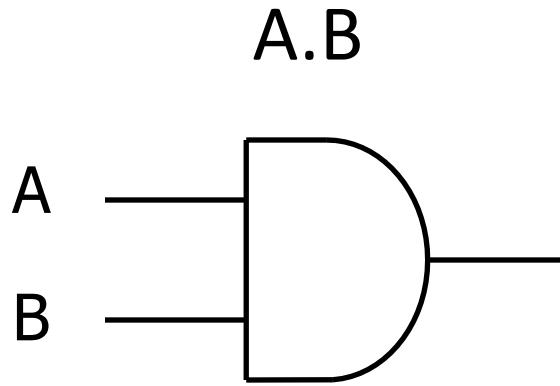
➤ Commutative Laws

Law 2: $A \cdot B = B \cdot A$

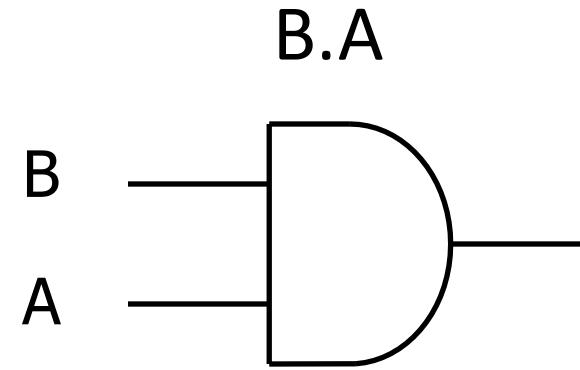
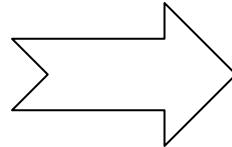
- ✓ This Law states that, A AND B is the same as B AND A i.e. the order in which the variables are ANDed is immaterial.
- ✓ This means that it makes no difference which input of an AND gate is connected to A and which to B.

Boolean Algebra

Proof:



Inputs		Output
A	B	$Y = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1



Inputs		Output
B	A	$Y = B \cdot A$
0	0	0
0	1	0
1	0	0
1	1	1

Boolean Algebra

➤ Commutative Laws

✓ This law can be extended to any number of variables. For example,

$$A \cdot B \cdot C = B \cdot C \cdot A = C \cdot A \cdot B = B \cdot A \cdot C$$

Boolean Algebra

➤ Associative Laws

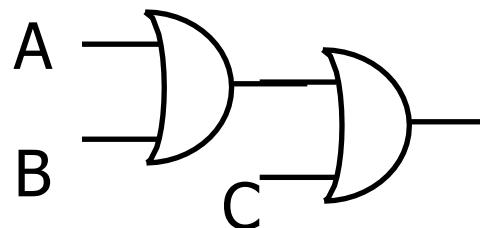
Law 1: $(A+B)+C = A+(B+C)$

- ✓ A OR B ORed with C is the same as A ORed with B OR C.
- ✓ This law states that the way the variables are grouped and ORed is immaterial.

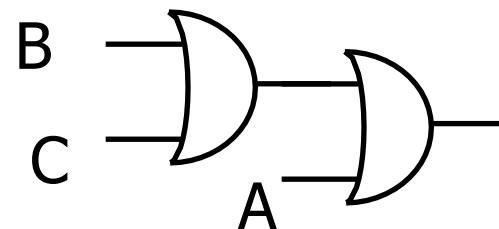
Boolean Algebra

Proof:

$$(A+B)+C$$



$$A+(B+C)$$



A	B	C	$A+B$	$(A+B)+C$
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

A	B	C	$B+C$	$A+(B+C)$
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Boolean Algebra

➤ Associative Laws

✓ This law can be extended to any number of variables. For example,

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

➤ Associative Laws

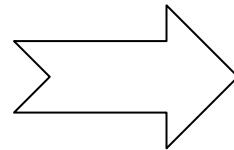
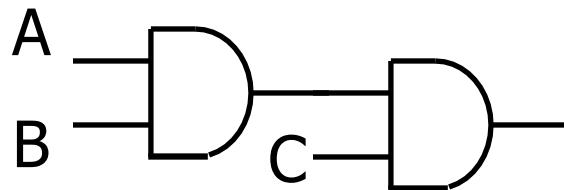
Law 2: $(A \cdot B) \cdot C = A \cdot (B \cdot C)$

- ✓ A AND B ANDed with C is the same as A ANDed with B AND C.
- ✓ This law states that the way the variables are grouped and ANDed is immaterial.

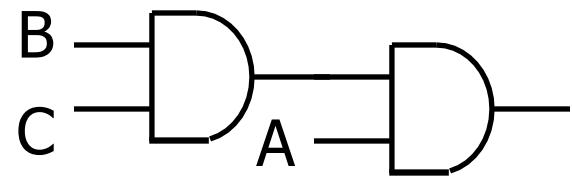
Boolean Algebra

Proof:

$$(A \cdot B) \cdot C$$



$$A \cdot (B \cdot C)$$



A	B	C	A.B	(A.B).C
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

A	B	C	B.C	A.(B.C)
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

➤ Associative Laws

✓ This law can be extended to any number of variables. For example,

$$A.(B.C.D) = (A.B.C).D = (A.B).(C.D)$$

➤ Distributive Laws

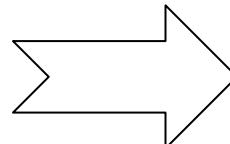
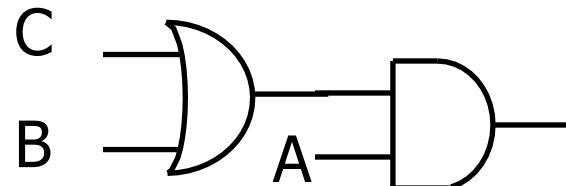
$$\text{Law 1: } A(B+C) = AB+AC$$

✓ This law states that ORing of several variables and ANDing the result with a single variable is equivalent to ANDing that single variable with each of the several variables and then ORing the products.

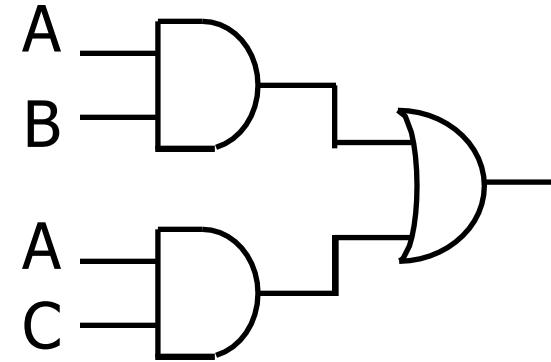
Boolean Algebra

Proof:

$$A.(B+C)$$



$$AB+AC$$



A	B	C	$B+C$	$A(B+C)$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

A	B	C	AB	AC	$AB+AC$
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

Boolean Algebra

➤ Distributive Laws

✓ This law can be extended to any number of variables. For example,

$$ABC(D+E) = ABCD + ABCE$$

$$AB(CD+EF) = ABCD + ABEF$$

Boolean Algebra

➤ Distributive Laws

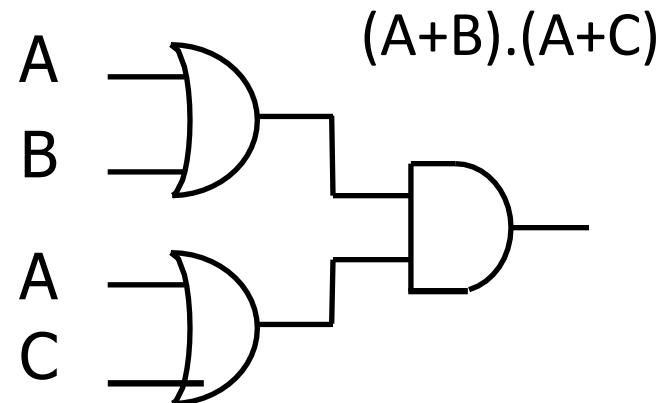
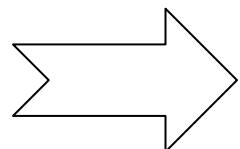
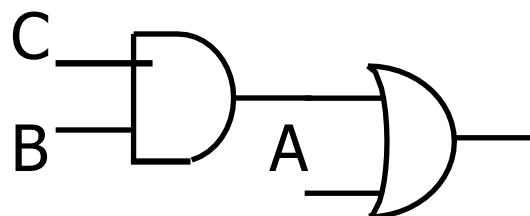
$$\text{Law 2: } A+BC = (A+B).(A+C)$$

✓ This law states that ANDing of several variables and ORing the result with a single variable is equivalent to ORing that single variable with each of the several variables and then ANDing the products.

Boolean Algebra

Proof:

$$A + (B \cdot C)$$



A	B	C	BC	$A+BC$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A	B	C	$A+B$	$A+C$	$(A+B) \cdot (A+C)$
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

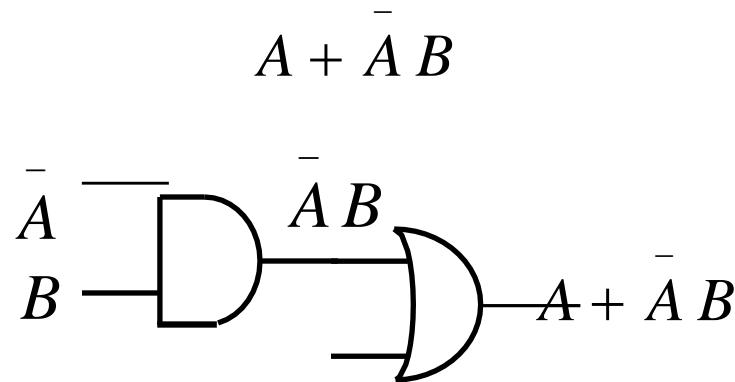
➤ Redundant Literal Rule

Law 1: $A + \bar{A}B = A + B$

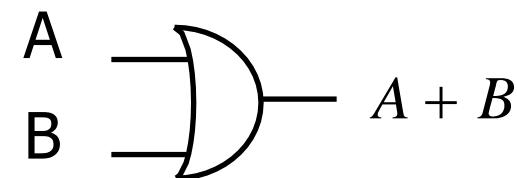
✓ This law states that ORing of variable with the AND of the complement of that variable with another variable, is equal to the ORing of the two variables

Boolean Algebra

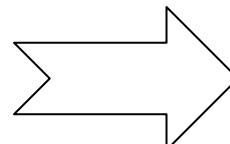
Proof:



$$A + B$$



A	B	$\bar{A}B$	$A + \bar{A}B$
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	1



A	B	$A+B$
0	0	0
0	1	1
1	0	1
1	1	1

Boolean Algebra

➤ Redundant Literal Rule

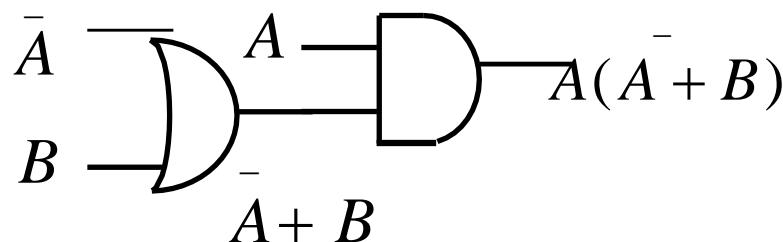
Law 2: $A(\bar{A} + B) = A \cdot B$

- ✓ This law states that ANDing of variable with the OR of the complement of that variable with another variable, is equal to the ANDing of the two variables

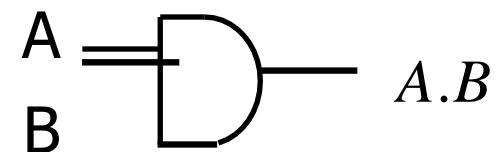
Boolean Algebra

Proof:

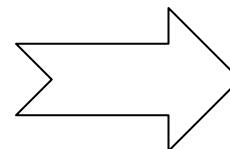
$$A(\bar{A} + B)$$



$$A \cdot B$$



A	B	$\bar{A} + B$	$A(\bar{A} + B)$
0	0	1	0
0	1	1	0
1	0	0	0
1	1	1	1

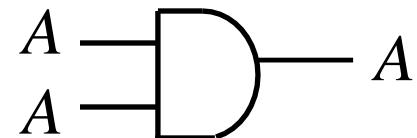


A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

Boolean Algebra

➤ Idempotence Laws

Law 1: $A \cdot A = A$

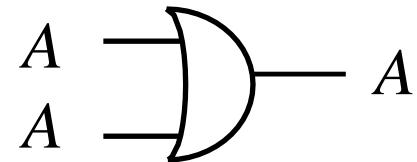


- ✓ Idempotence means the same value
- ✓ If $A=0$, then $A \cdot A = 0 \cdot 0 = 0 = A$
- ✓ If $A=1$, then $A \cdot A = 1 \cdot 1 = 1 = A$
- ✓ This law states that ANDing of a variable with itself is equal to that variable only.

Boolean Algebra

➤ Idempotence Laws

Law 2: $A + A = A$



- ✓ Idempotence means the same value
- ✓ If $A=0$, then $A+A = 0+0 = 0 = A$
- ✓ If $A=1$, then $A+A = 1+1 = 1 = A$
- ✓ This law states that ORing of a variable with itself is equal to that variable only.

Boolean Algebra

➤ Absorption Laws

Law 1: $A + A \cdot B = A$

- ✓ This law states that ORing of a variable with AND of that variable and another variable is equal to that variable itself.
- ✓ Therefore,

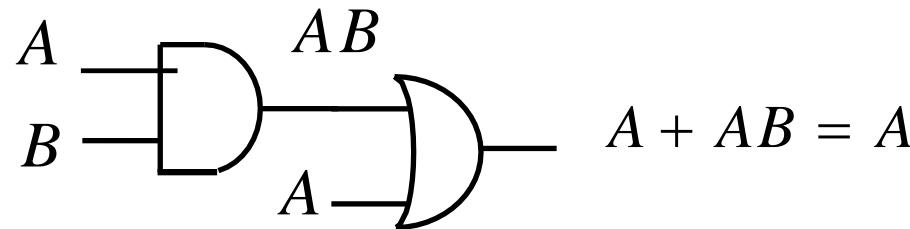
$$A + A \cdot \text{Any Term} = A$$

Boolean Algebra

Proof:

$$A + A \cdot B$$

$$A$$



A	B	$A \cdot B$	$A + A \cdot B$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Boolean Algebra

➤ Absorption Laws

Law 2: $A(A + B) = A$

- ✓ This law states that ANDing of a variable with OR of that variable and another variable is equal to that variable itself.
- ✓ Therefore,

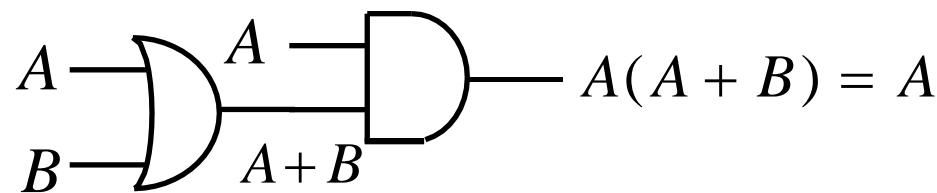
$$A \cdot (A + \text{Any Term}) = A$$

Boolean Algebra

Proof:

$$A(A + B)$$

$$A$$



A	B	$A + B$	$A(A + B)$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

Boolean Algebra

➤ De-Morgan's Theorem

First Theorem:

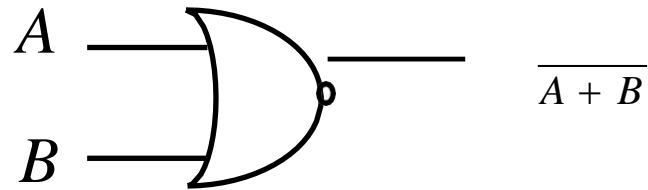
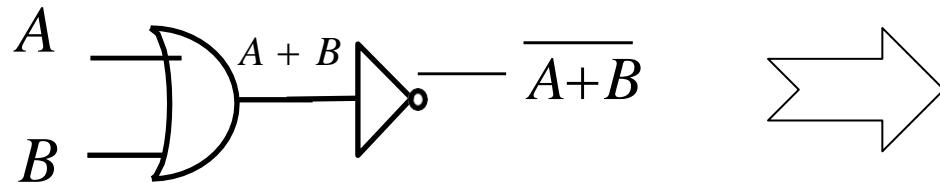
$$\overline{A + B} = \bar{A} \cdot \bar{B}$$

- ✓ This theorem states that the complement of a sum of variables is equal to the product of their individual complements.
- ✓ What it means is that the complement of two or more variables ORed together, is the same as the AND of the complements of each of the individual variables

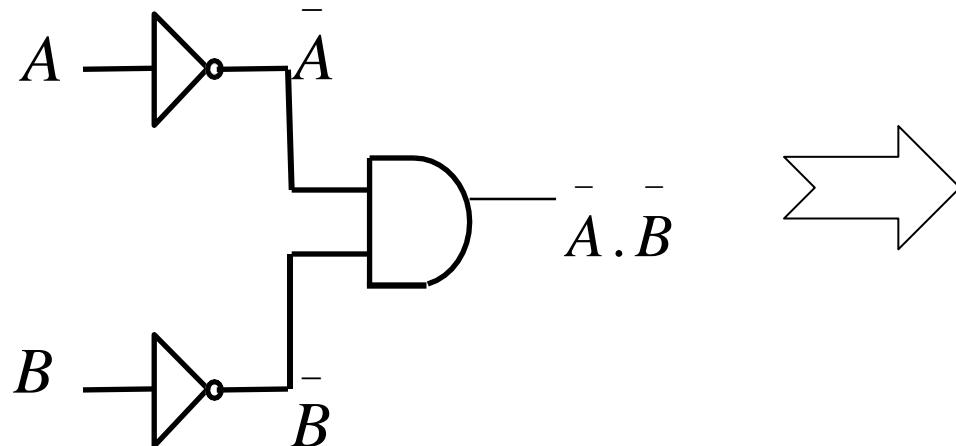
Boolean Algebra

Proof: Logic Diagram

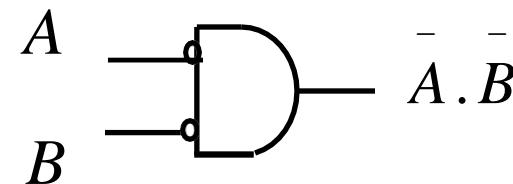
L.H.S.



R.H.S.



NOR Gate



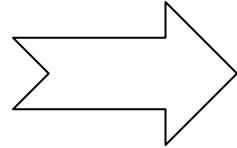
Bubbled AND Gate

Boolean Algebra

Proof: Logic Table

$$\overline{A + B}$$

A	B	$A + B$	$\overline{A + B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0



$$\overline{\overline{A} \cdot \overline{B}}$$

A	B	\overline{A}	\overline{B}	$\overline{\overline{A} \cdot \overline{B}}$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

Boolean Algebra

➤ De-Morgan's Theorem

✓ This law can be extended to any number of variables. For example,

$$\overline{A+B+C+D+\dots} = \overline{A}\overline{B}\overline{C}\overline{D}\dots$$

$$\overline{AB+CD+EFG+\dots} = (\overline{A}\overline{B})(\overline{C}\overline{D})(\overline{E}\overline{F}\overline{G})\dots$$

Boolean Algebra

➤ De-Morgan's Theorem

Second Theorem:

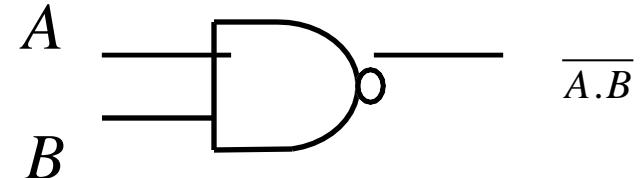
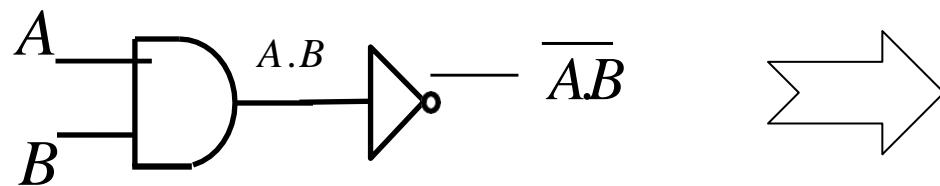
$$\overline{A \cdot B} = \overline{\overline{A}} + \overline{\overline{B}}$$

- ✓ This theorem states that the complement of a product of variables is equal to the sum of their individual complements.
- ✓ What it means is that the complement of two or more variables ANDed together, is the same as the OR of the complements of each of the individual variables

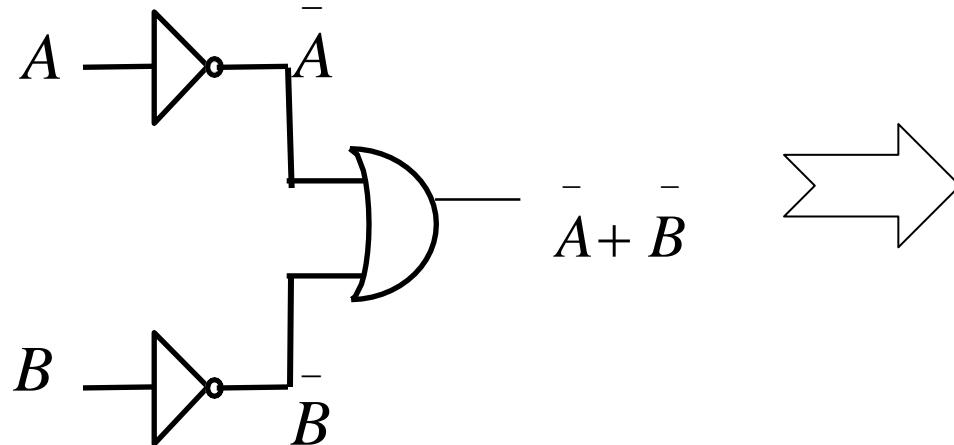
Boolean Algebra

Proof: Logic Diagram

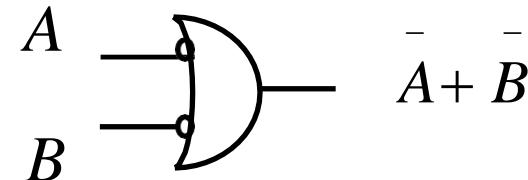
L.H.S.



R.H.S.



NAND Gate

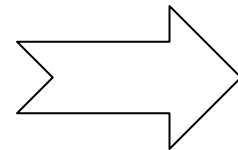


Bubbled OR Gate

Boolean Algebra

Proof:

$$\overline{A \cdot B}$$



$$\overline{\overline{A} + \overline{B}}$$

A	B	$A \cdot B$	$\overline{A \cdot B}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

A	B	\overline{A}	\overline{B}	$\overline{\overline{A} + \overline{B}}$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

Boolean Algebra

➤ De-Morgan's Theorem

- ✓ This law can be extended to any number of variables. For example,

$$\overline{A \cdot B \cdot C \cdot D \dots} = \overline{A} + \overline{B} + \overline{C} + \overline{D} \dots$$

$$\overline{(A \cdot B) \cdot (C \cdot D) \cdot (E \cdot F \cdot G) \dots} = \overline{\overline{A} \cdot \overline{B}} + \overline{\overline{C} \cdot \overline{D}} + \overline{\overline{E} \cdot \overline{F} \cdot \overline{G}} + \dots$$

Duality

✓ Duality represents relation between expressions in positive logic system and expression in negative logic system.

Duality

- ✓ The distinction between positive and negative logic system is important.
- ✓ An OR gate in positive logic system becomes an AND gate in negative logic system and vice versa.
- ✓ Positive & negative logics thus give rise to a basic duality in all Boolean identities.

Duality

- ✓ When changing from one logic system to another 0 becomes 1 and 1 becomes 0.
- ✓ Furthermore, an AND gate becomes an OR gate and an OR gate becomes AND gate.

Duality

- ✓ Given Boolean identity, we can produce a dual identity by changing all ‘+’ signs to “ signs, all “ signs to ‘+’ signs and complementing all 0's and 1's.
- ✓ The variables are not complemented in this process.

Examples of Dual Identities

Sr. No.	Given Expression	Dual
1	$\overline{0} = 1$	$\overline{1} = 0$
2	$0.\overline{1} = 0$	$\overline{1} + 0 = 1$
3	$0.\overline{0} = 0$	$\overline{1} + \overline{1} = 1$
4	$1.\overline{1} = 1$	$0 + 0 = 0$
5	$A.\overline{0} = 0$	$\overline{A} + 1 = 1$
6	$A.\overline{1} = A$	$\overline{A} + 0 = A$

Examples of Dual Identities

Sr. No.	Given Expression	Dual
7	$A \cdot B = B \cdot A$	$A + B = B + A$
8	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$
9	$A \cdot (B + C) = A \cdot B + A \cdot C$	$A + BC = (A + B)(A + C)$
10	$A \cdot (A + B) = A$	$A + AB = A$
11	$A \cdot (A \cdot B) = A \cdot B$	$A + A + B = A + B$
12	$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$
13	$\overline{(A + B)(A + C)} = \overline{(A + B)} \cdot \overline{(A + C)}$	$\overline{AB} + \overline{AC} = \overline{AB} \cdot \overline{AC}$

Example 1

Reduce the following Boolean Expression using Boolean Laws:

$$A\bar{B} + \bar{A}B + A.B + \bar{A}\bar{B}$$

Example 1

Reduce the following Boolean Expression using Boolean Laws:

$$A.\overline{B} + \overline{A}.B + A.B + \overline{A}.\overline{B}$$

$$= A.\overline{B} + \overline{A}.B + A.B + \overline{A}.\overline{B}$$

$$= A.\overline{B} + A.B + \overline{A}.B + \overline{A}.\overline{B}$$

$$= A.(\overline{B} + B) + \overline{A}(B + \overline{B}) \quad (\because B + \overline{B} = 1)$$

$$= A + \overline{A} \quad (\because A + \overline{A} = 1)$$

$$= 1$$

$$A.\overline{B} + \overline{A}.B + A.B + \overline{A}.\overline{B} = 1$$

Example 2

Reduce the following Boolean Expression using Boolean Laws:

$$A\bar{B}C + \bar{A}BC + ABC$$

Example 2

Reduce the following Boolean Expression using Boolean Laws:

$$A\bar{B}C + \bar{A}\bar{B}C + ABC$$

$$= A\bar{B}C + \bar{A}\bar{B}C + ABC$$

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

$$= A\bar{B}C + BC \quad (\because A + \bar{A} = 1)$$

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

$$= C(B + A)(\bar{B} + B) \quad (\because \text{Distributive Law})$$

$$= C(B + A) \quad (\because \bar{B} + B = 1)$$

$$= AC + BC$$

Example 3

Realize $Y=AB+AC$ using one OR gate and one AND gate

Example 3

Realize $Y=AB+AC$ using one OR gate and one AND gate

$$Y = AB + AC$$

A.B is one product term
Hence requires 1 AND gate

A.C is one product term
Hence requires 1 AND gate

A.C & A.B is one sum term
Hence requires 1 OR gate

Hence to implement $Y=AB+AC$ equation we require 2 AND gates and 1 OR gate

But we have to use only 1 AND gate and 1 OR gate

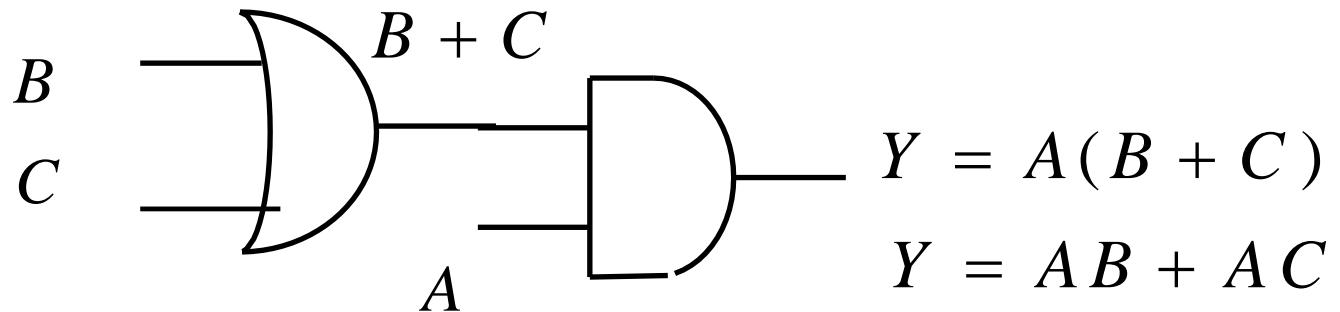
Hence simplification is necessary

Example 3

Continue.....

$$Y = AB + AC$$

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



Example 4

Prove that:

$$A + \overline{A}B = A + B$$

Example 4

Prove that:

$$A + \overline{A}B = A + B$$

$$\begin{aligned}\text{L.H.S} &= A + \overline{A}B \\&= A(1) + \overline{A}B \\&= A(1 + B) + \overline{A}B \\&= A + AB + \overline{A}B \\&= A + B(A + \overline{A}) \\&= A + B \quad (\because A + \overline{A} = 1)\end{aligned}$$

$$\text{L.H.S} = \text{R.H.S}$$

Example 5

Prove that:

$$(A + B)(A + \overline{B}) = A$$

Example 5

Prove that:

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

$$\begin{aligned} \text{L.H.S} &= (A + B)(A + \bar{B}) \\ &= AA + A\bar{B} + AB + B\bar{B} \\ &= A + A\bar{B} + AB + 0 \quad (\because AA=A, B\bar{B}=0) \\ &= A + A(\bar{B} + B) \\ &= A + A \quad (\because \bar{B} + B = 1) \\ &= A \quad (\because A + A = A) \end{aligned}$$

$$\text{L.H.S} = \text{R.H.S}$$

Example 6

With the help of Boolean Laws, Prove that:

$$(A + \overline{B} + AB)(A + B).\overline{A}\overline{B} = 0$$

Example 6

With the help of Boolean Laws, Prove that:

$$(A + \overline{B} + AB)(A + B).\overline{AB} = 0$$

$$\begin{aligned}\text{L.H.S.} &= (A + \overline{B} + AB)(A + B).\overline{AB} \\ &= (A + \overline{B} + AB)(\overline{A}\overline{B} + \overline{A}B) \\ &= (A + \overline{B} + AB).(0) \quad (\because A.\overline{A}=0, B.\overline{B}=0) \\ &= 0\end{aligned}$$

$$\text{L.H.S} = \text{R.H.S}$$

Example 7

With the help of Boolean Laws, Prove that:

$$AB + \overline{A}B + \overline{A}\overline{B} = \overline{A} + B$$

Example 7

With the help of Boolean Laws, Prove that:

$$AB + \overline{A}B + \overline{A}\overline{B} = \overline{A} + B$$

$$\text{L.H.S.} = AB + \overline{A}B + \overline{A}\overline{B}$$

$$= \overline{A}B + \overline{A}\overline{B} + AB$$

$$= \overline{A}(B + \overline{B}) + AB$$

$$= \overline{A} + AB \quad (\because B + \overline{B} = 0)$$

$$= (A + \overline{A})(\overline{A} + B) \quad (\because \overline{A} + AB = (A + \overline{A})(\overline{A} + B))$$

$$= 1 \cdot (\overline{A} + B) \quad (\because A + \overline{A} = 1)$$

$$= \overline{A} + B$$

$$\text{L.H.S} = \text{R.H.S}$$

Example 8

Simplify; $F = XY + XYZ + XYZ + X\bar{Z}Y$

Example 8

Simplify; $F = XY + XYZ + XYZ + X\bar{Z}Y$

$$\begin{aligned} F &= XY + XYZ + XYZ + X\bar{Z}Y \\ &= XY + XYZ + X\bar{Z}Y \quad (\because XYZ+XYZ=XYZ) \\ &= XY(1 + Z + \bar{Z}) \\ &= XY \quad (\because 1 + Z + \bar{Z} = 1) \\ &= XY \end{aligned}$$

Example 9

Prove that;

$$AB + ABC + A\bar{B} = A$$

Example 9

Prove that;

$$AB + ABC + A\bar{B} = A$$

$$\begin{aligned} L.H.S. &= AB + ABC + A\bar{B} \\ &= AB(1 + C) + A\bar{B} \\ &= AB + A\bar{B} \quad (\because 1 + C = 1) \\ &= A(B + \bar{B}) \\ &= A \quad (\because B + \bar{B} = 1) \end{aligned}$$

L.H.S = R.H.S

1.3 Laws of Boolean Algebra

The Boolean algebra is governed by certain well developed rules and laws.

1.3.1 Commutative Laws

- The commutative law allows change in position of AND or OR variables. There are two commutative laws.
 - (i) $A + B = B + A$
Thus, the order in which the variables are ORed is immaterial.
 - (ii) $A \cdot B = B \cdot A$
Thus, the order in which the variables are ANDed is immaterial.
- This law can be extended to any number of variables.

1.3.2 Associative Laws

- The associative law allows grouping of variables. There are two associative laws
 - (i) $(A + B) + C = A + (B + C)$
Thus, the way the variables are grouped and ORed is immaterial.
 - (ii) $(A \cdot B) \cdot C = A \cdot (B \cdot C)$
Thus, the way the variables are grouped and ANDed is immaterial.
- This law can be extended to any number of variables.

1.3.3 Distributive Laws

- The distributive law allows factoring or multiplying out of expressions. There are two distributive laws.
 - (i) $A(B + C) = AB + AC$
 - (ii) $A + BC = (A + B)(A + C)$
- This law is applicable for single variable as well as a combination of variables.

1.3.4 Idempotence Laws

Idempotence means the same value. There are two Idempotence laws

(i) $A \cdot A = A$

i.e. ANDing of a variable with itself is equal to that variable only.

(ii) $A + A = A$

i.e. ORing of a variable with itself is equal to that variable only.

1.3.5 Absorption Laws

There are two absorption laws

(i) $A + AB = A(1 + B) = A$ (ii) $A(A + B) = A$

1.3.6 Involutionary Law

This law states that, for any variable 'A'

$$\bar{\bar{A}} = (A')' = A$$

1.4 Boolean Algebraic Theorems

1.4.1 De Morgan's Theorem

- These are very useful in simplifying expressions in which a product or sum of variables is inverted.
- De Morgan's theorem represents two of the most important rules of Boolean algebra.

$$(i) \quad \overline{A \cdot B} = \bar{A} + \bar{B}$$

Thus, the complement of the product of variables is equal to the sum of their individual complements.

$$(ii) \quad \overline{A + B} = \bar{A} \cdot \bar{B}$$

Thus, the complement of a sum of variables is equal to the product of their individual complements.

- The above two laws can be extended for 'n' variables as

$$\overline{A_1 \cdot A_2 \cdot A_3 \cdots A_n} = \bar{A}_1 + \bar{A}_2 + \cdots + \bar{A}_n$$

$$\text{and } \overline{A_1 + A_2 + \cdots + A_n} = \bar{A}_1 \cdot \bar{A}_2 \cdot \bar{A}_3 \cdot \bar{A}_4 \cdots \bar{A}_n$$

1.4.2 Transposition Theorem

The transposition theorem states that $(AB + \bar{A}C) = (A + C)(\bar{A} + B)$

1.4.3 Consensus Theorem/Redundancy Theorem

- This theorem is used to eliminate redundant term.
- A variable is associated with some variable and its compliment is associated with some other variable and the next term is formed by the left over variables, then the term becomes redundant.
- It is applicable only if a Boolean function,
 - (i) Contains 3-variables.
 - (ii) Each variable used two times.
 - (iii) Only one variable is in complemented or uncomplemented form.

Then, the related terms to that complemented and uncomplemented variable is the answer.

- Consensus theorem can be extended to any number of variables.
e.g. $AB + \bar{A}C + BC = AB + \bar{A}C$

1.4.4 Duality Theorem

It is one of the elegant theorems proved in advance mathematics.

"Dual expression" is equivalent to write a negative logic of the given Boolean relation. For this we have to

- (i) change each **OR** sign by an **AND** sign and vice-versa.
- (ii) complement any '**0**' or '**1**' appearing in expression.
- (iii) keep literals/variables as it is.

1.4.5 Complementary Theorem

For obtaining complement expression we have to

- (i) change each **OR** sign by **AND** sign and vice-versa.
- (ii) complement any '**0**' or '**1**' appearing in expression.
- (iii) complement the individual literals/variables.



Example - 1.1 The Boolean expression $(A + B)(\bar{B} + C)(C + A) = (A + B)(\bar{B} + C)$ can be simplified as

- (a) $(A + B)(\bar{B} + C)$
- (b) $(\bar{A} + B)(\bar{B} + C)$
- (c) $(A + \bar{B})(\bar{B} + C)$
- (d) $(A + \bar{B})(B + \bar{C})$



Example - 1.2 $\overline{\bar{A}\bar{B}\bar{C}}$ is equal to

- (a) $\bar{A} + \bar{B} + \bar{C}$
- (b) \overline{ABC}
- (c) $A + B + C$
- (d) $A \cdot B \cdot C$



Example - 1.3 The self dual expression of boolean relation, $\bar{A}BC + AB\bar{C} + A\bar{B}\bar{C}$ is

- (a) $(A + B + \bar{C})(A + B + \bar{C})(A + \bar{B} + \bar{C})$
- (b) $(A + B + C)(A + B + C)(A + \bar{B} + \bar{C})$
- (c) $(\bar{A} + B + C)(A + B + \bar{C})(A + \bar{B} + \bar{C})$
- (d) $(\bar{A} + B + \bar{C})(A + \bar{B} + \bar{C})(A + \bar{B} + \bar{C})$



Example - 1.4 The compliment of the function $f = A\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C}$ is equal to

- (a) $(\bar{A} + B + \bar{C})(A + \bar{B} + C)(\bar{A} + B + C)$
- (b) $(\bar{A} + \bar{B} + C)(A + \bar{B} + \bar{C})(\bar{A} + B + C)$
- (c) $(\bar{A} + \bar{B} + C)(A + \bar{B} + C)(\bar{A} + \bar{B} + C)$
- (d) $(\bar{A} + B + \bar{C})(A + \bar{B} + C)(\bar{A} + \bar{B} + \bar{C})$



Example - 1.5 The Boolean expression $A(A + B)$ is equal to

- (a) 1
- (b) B
- (c) A
- (d) $A + B$



Example - 1.6 The Boolean function $(x + y)(\bar{x} + z)(y + z)$ is equal to which one of the following expressions?

- (a) $(x + y)(y + z)$
- (c) $(x + y)(\bar{x} + z)$

- (b) $(\bar{x} + z)(y + z)$
- (d) $(x + y)(x + \bar{z})$



Example - 1.7 The minimized form of the logical expression $(\bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C})$ is

- (a) $\bar{A}\bar{C} + B\bar{C} + \bar{A}B$
- (c) $\bar{A}C + \bar{B}C + \bar{A}B$

- (b) $A\bar{C} + \bar{B}C + \bar{A}B$
- (d) $A\bar{C} + \bar{B}C + A\bar{B}$



Example - 1.8 If X and Y are Boolean variables, which one of the following is the equivalent of $X \oplus Y \oplus XY$?

- (a) $X + \bar{Y}$
- (c) 0

- (b) $X + Y$
- (d) 1



Example - 1.1 The Boolean expression $(A + B)(\bar{B} + C)(C + A) = (A + B)(\bar{B} + C)$ can be simplified as

(a) $(A + B)(\bar{B} + C)$

(b) $(\bar{A} + B)(\bar{B} + C)$

(c) $(A + \bar{B})(\bar{B} + C)$

(d) $(A + \bar{B})(B + \bar{C})$

Solution : (a)

Proof:

$$\begin{aligned}\text{LHS} &= (A + B)(\bar{B} + C)(C + A) \\&= (A\bar{B} + AC + BC)(C + A) \\&= ABC + AC + BC + A\bar{B} + AC + ABC \\&= AC + BC + A\bar{B} \\ \text{RHS} &= (A + B)(\bar{B} + C) \\&= A\bar{B} + AC + BC = \text{LHS}\end{aligned}$$



Example - 1.2 \overline{ABC} is equal to

(a) $\bar{A} + \bar{B} + \bar{C}$

(b) \overline{ABC}

(c) $A + B + C$

(d) $A \cdot B \cdot C$

Solution: (c)

$$\overline{ABC} = \overline{\bar{A}} + \overline{\bar{B}} + \overline{\bar{C}} = A + B + C$$



Example - 1.3 The self dual expression of boolean relation, $\bar{A}BC + AB\bar{C} + A\bar{B}\bar{C}$ is

- (a) $(A + B + \bar{C})(A + B + \bar{C})(A + \bar{B} + \bar{C})$
- (b) $(A + B + C)(A + B + C)(A + \bar{B} + \bar{C})$
- (c) $(\bar{A} + B + C)(A + B + \bar{C})(A + \bar{B} + \bar{C})$
- (d) $(\bar{A} + B + \bar{C})(A + \bar{B} + \bar{C})(A + \bar{B} + \bar{C})$

Solution: (c)

$$\bar{A}BC + AB\bar{C} + A\bar{B}\bar{C}$$

Its DUAL



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



Example - 1.4 The complement of the function $f = A\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C}$ is equal to

- (a) $(\bar{A} + B + \bar{C})(A + \bar{B} + C)(\bar{A} + B + C)$
- (b) $(\bar{A} + \bar{B} + C)(A + \bar{B} + \bar{C})(\bar{A} + B + C)$
- (c) $(\bar{A} + \bar{B} + C)(A + \bar{B} + C)(\bar{A} + \bar{B} + C)$
- (d) $(\bar{A} + B + \bar{C})(A + \bar{B} + C)(\bar{A} + \bar{B} + \bar{C})$

Solution : (a)

\bar{f} = Complement of f

$$\bar{f} = (\bar{A} + B + \bar{C})(A + \bar{B} + C)(\bar{A} + B + C)$$



Example - 1.5 The Boolean expression $A(A + B)$ is equal to

- | | |
|-------|-------------|
| (a) 1 | (b) B |
| (c) A | (d) $A + B$ |

Solution: (c)

$$\begin{aligned}Y &= A(A + B) \\&= A \cdot A + A \cdot B \\&= A + AB \\&= A(1 + B) \\&= A\end{aligned}$$



Example - 1.6 The Boolean function $(x + y)(\bar{x} + z)(y + z)$ is equal to which one of the following expressions?

- | | |
|----------------------------|----------------------------|
| (a) $(x + y)(y + z)$ | (b) $(\bar{x} + z)(y + z)$ |
| (c) $(x + y)(\bar{x} + z)$ | (d) $(x + y)(x + \bar{z})$ |

Solution: (c)

By using consensus theorem,

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



Example - 1.7 The minimized form of the logical expression $(\bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C})$ is

- (a) $\bar{A}\bar{C} + B\bar{C} + \bar{A}B$
- (b) $A\bar{C} + \bar{B}C + \bar{A}B$
- (c) $\bar{A}C + \bar{B}C + \bar{A}B$
- (d) $A\bar{C} + \bar{B}C + A\bar{B}$

Solution: (a)

Given,

$$\begin{aligned} Y &= \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + \bar{A}BC + \bar{A}B\bar{C} \\ &= \bar{A}\bar{C}(B + \bar{B}) + \bar{A}B(C + \bar{C}) + B\bar{C}(A + \bar{A}) \\ &= \bar{A}\bar{C} + \bar{A}B + B\bar{C} \end{aligned}$$



Example - 1.8 If X and Y are Boolean variables, which one of the following is the equivalent of $X \oplus Y \oplus XY$?

- (a) $X + \bar{Y}$
- (b) $X + Y$
- (c) 0
- (d) 1

Solution: (b)

Let,

$$\begin{aligned} Z &= X \oplus Y \oplus XY \\ Z &= X \oplus [Y(XY) + Y(\bar{X}Y)] \\ &= X \oplus [Y(\bar{X} + \bar{Y})] = X \oplus [(\bar{X}Y)] \\ &= \bar{X}(\bar{X}Y) + X(\bar{X}Y) = \bar{X}Y + X(X + \bar{Y}) \\ &= \bar{X}Y + X + X\bar{Y} = \bar{X}Y + X(1 + \bar{Y}) \\ &= \bar{X}Y + X = (X + \bar{X})(X + Y) = (X + Y) \end{aligned}$$



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



Switching Theory and Logic Design

ARI-209

By: Dr. Divya Agarwal



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT- II** 10 HOURS
- **Sequential Logic Circuits:** Latches and Flip Flops- SR, D, T and MS-JK Flip Flops, Asynchronous Inputs.
- **Counters and Shift Registers:** Design of Synchronous and Asynchronous Counters- Binary, BCD, Decade and Up/Down Counters, Shift Registers, Types of Shift Registers, Counters using Shift Registers- Ring Counter and Johnson Counter.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-III** **08 HOURS**
- **Integrated circuits:** TTL and CMOS logic families and their characteristics. Brief introduction to RAM and ROM
- **Synchronous Sequential Circuits:** State Tables State Equations and State Diagrams, State Reduction and State Assignment, Design of Clocked Sequential Circuits using State Equations.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-IV** 06 HOURS
- **Finite state machine:** capabilities and limitations, Mealy and Moore models-minimization of completely specified and incompletely specified sequential machines, Partition techniques and merger chart methods, concept of minimal cover table.
- **Algorithmic State Machine:** Representation of sequential circuits using ASM charts synthesis of output and next state functions, Data path control path partition-based design.



Text Books:

1. Leach and Malvino (2011). *Digital principles and Applications*. Tata McGraw-Hill Education
2. Mano, M. M. (2017). *Digital logic and computer design*. Pearson Education India.
3. Jain, R. P. (2003). *Modern digital electronics*. Tata McGraw-Hill Education.

Reference Books:

1. A Anand Kumar. (2016) *Fundamentals of Digital Logic Circuits*, PHI
2. Taub, H., & Schilling, D. L. (1977). *Digital integrated electronics*. McGraw-Hill College.



■ UNIT- I

14 HOURS

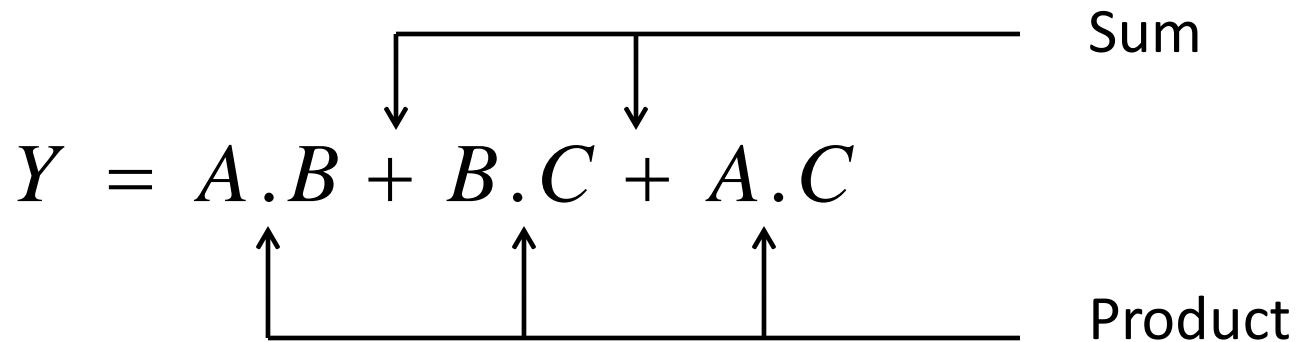
- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.

Standard Representation

- Any logical expression can be expressed in the following two forms:
 - ✓ Sum of Product (SOP) Form
 - ✓ Product of Sum (POS) Form

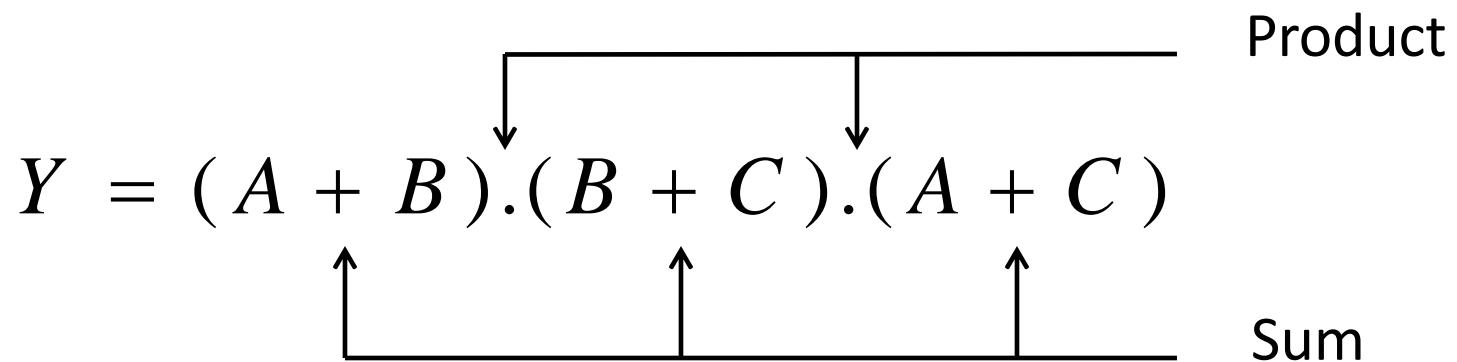
SOP Form

For Example, logical expression given is;



POS Form

For Example, logical expression given is;

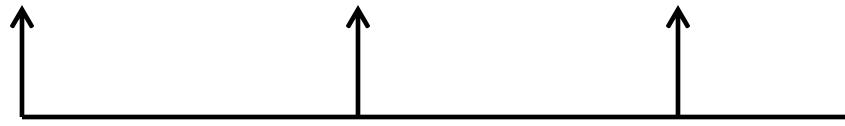


Standard or Canonical SOP & POS Forms

✓ We can say that a logic expression is said to be in the standard (or canonical) SOP or POS form if each product term (for SOP) and sum term (for POS) consists of all the literals in their complemented or uncomplemented form.

Standard SOP

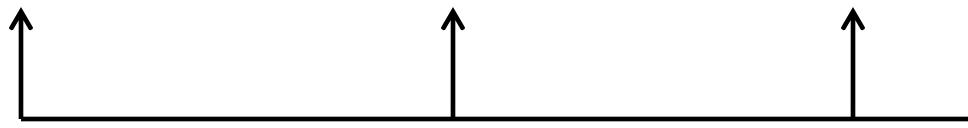
$$Y = A B C + A \overline{B} \overline{C} + \overline{A} B C$$



Each product term
consists all the
literals

Standard POS

$$Y = (A + B + C).(A + \overline{B} + \overline{C}).(\overline{A} + B + C)$$



Each sum term
consists all the
 literals

Examples

Sr. No.	Expression	Type
1	$Y = AB + ABC + \overline{A}BC$	Non Standard SOP
2	$Y = AB + A\overline{B} + \overline{A}\overline{B}$	Standard SOP
3	$Y = (\overline{A} + B).(A + \overline{B}).(\overline{A} + \overline{B})$	Standard POS
4	$Y = (\overline{A} + B).(A + \overline{B} + C)$	Non Standard POS

Conversion of SOP form to Standard SOP

Procedure:

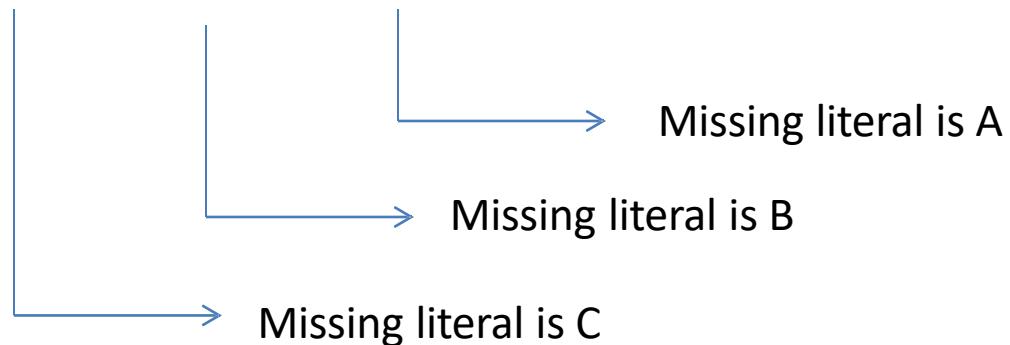
1. Write down all the terms.
2. If one or more variables are missing in any product term, expand the term by multiplying it with the sum of each one of the missing variable and its complement .
3. Drop out the redundant terms

Example 1

Convert given expression into its standard SOP form

$$Y = AB + A\bar{C} + BC$$

$$Y = AB + A\bar{C} + BC$$



$$Y = AB \cdot (C + \bar{C}) + A\bar{C} \cdot (B + \bar{B}) + BC \cdot (A + \bar{A})$$

Term formed by ORing of missing literal & its complement

Example 1

Continue....

$$Y = AB \cdot (C + \overline{C}) + A\overline{C} \cdot (B + \overline{B}) + BC \cdot (A + \overline{A})$$

$$Y = ABC + AB\overline{C} + A\overline{B}\overline{C} + A\overline{B}C + \overline{A}BC$$

$$Y = \underline{\underline{ABC}} + \underline{\underline{AB\overline{C}}} + \underline{\underline{A\overline{B}\overline{C}}} + \underline{\underline{A\overline{B}C}} + \underline{\underline{\overline{A}BC}}$$

$$Y = ABC + AB\overline{C} + A\overline{B}\overline{C} + \overline{A}BC$$

Standard SOP form

Each product term consists all the literals

Conversion of POS form to Standard POS

Procedure:

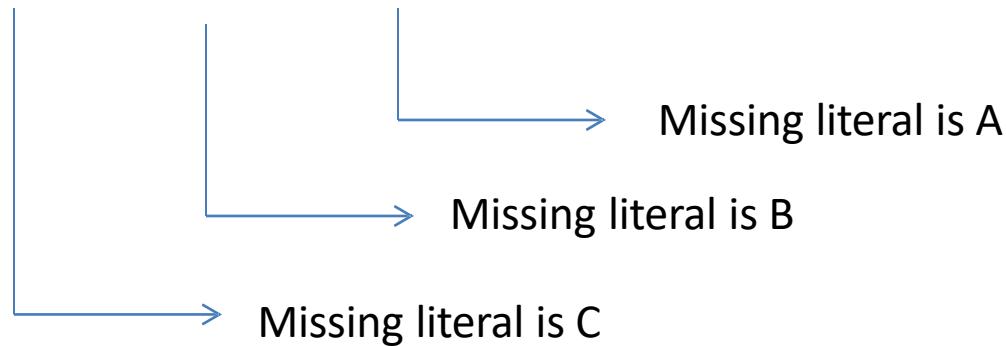
1. Write down all the terms.
2. If one or more variables are missing in any sum term, expand the term by adding the products of each one of the missing variable and its complement .
3. Drop out the redundant terms

Example 2

Convert given expression into its standard SOP form

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

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



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

Term formed by ANDing of missing
literal & its complement

Example 2

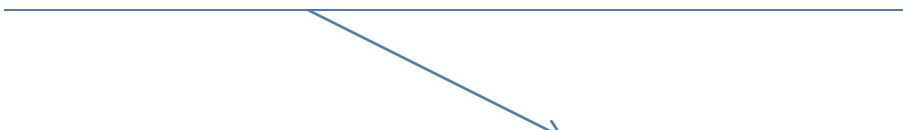
Continue....

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

$$Y = \underline{(A+B+C)} \underline{(A+B+\bar{C})} \underline{(A+B+C)} \underline{(A+\bar{B}+C)} \underline{(A+B+\bar{C})} \underline{(A+\bar{B}+\bar{C})}$$

$$Y = (A+B+C)(A+B+\bar{C})(A+\bar{B}+C)(\bar{A}+B+\bar{C})$$

$$Y = (A+B+C)(A+B+\bar{C})(A+\bar{B}+C)(\bar{A}+B+\bar{C})$$



Standard POS form
Each sum term consists all the literals

Examples

1. Convert the given expression into standard form

$$Y = A + BC + ABC$$

2. Convert the given expression into standard form

$$Y = (A + \overline{B}).(A + C)$$

Concept of Minterm and Maxterm

- ✓ **Minterm:** Each individual term in the standard SOP form is called as “Minterm”.
- ✓ **Maxterm:** Each individual term in the standard POS form is called as “Maxterm”.

✓ The concept of minterm and max term allows us to introduce a very convenient shorthand notation to express logic functions

Minterms & Maxterms for 3 variable/literal logic function

Variables			Minterms	Maxterms
A	B	C	mi	Mi
0	0	0	$\overline{A}\overline{B}\overline{C} = m_0$	$A + B + C = M_0$
0	0	1	$\overline{A}\overline{B}C = m_1$	$A + B + \overline{C} = M_1$
0	1	0	$\overline{A}B\overline{C} = m_2$	$A + \overline{B} + C = M_2$
0	1	1	$\overline{A}BC = m_3$	$A + \overline{B} + \overline{C} = M_3$
1	0	0	$A\overline{B}\overline{C} = m_4$	$\overline{A} + B + C = M_4$
1	0	1	$A\overline{B}C = m_5$	$\overline{A} + B + \overline{C} = M_5$
1	1	0	$AB\overline{C} = m_6$	$\overline{A} + \overline{B} + C = M_6$
1	1	1	$ABC = m_7$	$\overline{A} + \overline{B} + \overline{C} = M_7$

Minterms and maxterms

- ✓ Each minterm is represented by m_i where $i=0,1,2,3,\dots,2^{n-1}$
- ✓ Each maxterm is represented by M_i where $i=0,1,2,3,\dots,2^{n-1}$
- ✓ If 'n' number of variables forms the function, then number of minterms or maxterms will be 2^n
 - i.e. for 3 variables function $f(A,B,C)$, the number of minterms or maxterms are $2^3=8$

Minterms & Maxterms for 2 variable/literal logic function

Variables		Minterms	Maxterms
A	B	m_i	M_i
0	0	$\overline{A} \overline{B} = m_0$	$A + B = M_0$
0	1	$\overline{A} B = m_1$	$A + \overline{B} = M_1$
1	0	$A \overline{B} = m_2$	$\overline{A} + B = M_2$
1	1	$A B = m_3$	$\overline{A} + \overline{B} = M_3$

Representation of Logical expression using minterm

$$Y = \underline{ABC} + \underline{\overline{A}BC} + \underline{A\overline{B}\overline{C}} + \underline{A\overline{B}C}$$

m_7 m_3 m_4 m_5

Logical Expression ←
Corresponding minterms ←

$$Y = m_7 + m_3 + m_4 + m_5$$

$$Y = \Sigma m(3, 4, 5, 7)$$

OR

$$Y = f(A, B, C) = \Sigma m(3, 4, 5, 7)$$

where Σ denotes sum of products

Representation of Logical expression using maxterm

$$Y = (\underline{A + \overline{B} + C}).(\underline{A + B + C}).(\underline{\overline{A} + \overline{B} + C}) \leftarrow \begin{array}{l} \text{Logical Expression} \\ \text{Corresponding} \\ \text{maxterms} \end{array}$$

M_2 M_0 M_6

$$Y = M_2 \cdot M_0 \cdot M_6$$

$$Y = \prod M (0, 2, 6) \quad \text{OR}$$

$$Y = f(A, B, C) = \prod M (0, 2, 6)$$

where \prod denotes product of sum

Conversion from SOP to POS & Vice versa

- ✓ The relationship between the expressions using minterms and maxterms is complementary.
- ✓ We can exploit this complementary relationship to write the expressions in terms of maxterms if the expression in terms of minterms is known and vice versa

Conversion from SOP to POS & Vice versa

- ✓ For example, if a SOP expression for 4 variable is given by,

$$Y = \Sigma m(0,1,3,5,6,7,11,12,15)$$

- ✓ Then we can get the equivalent POS expression using the complementary relationship as follows,

$$Y = \Pi M(2,4,8,9,10,13,14)$$

Examples

Example 7: Find the sum-of-products and product of sums equations from the given truth Table - 16.

Table 16

A	B	C	Output Functional Values
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	1
1	1	1	0

Examples

Example 7: Find the sum-of-products and product of sums equations from the given truth Table - 16.

Table 16

A	B	C	Output Functional Values
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	1
1	1	1	0

Sum-of-Product Equation

$$X = \overline{A} \overline{B} C + \overline{A} B \overline{C} + A \overline{B} \overline{C} + AB \overline{C}$$

Product-of-Sums Equation:

$$Y = (A + B + C)(A + \overline{B} + \overline{C})(\overline{A} + B + \overline{C})(\overline{A} + \overline{B} + \overline{C})$$

Examples

1. Express the Boolean function $F = A + B'C$ as a sum of minterms.
2. Express the Boolean function $F = xy + x'z$ as a product of maxterms.
3. Convert the following expression $\overline{}$ to sum-of-product form.
 - (i) $(A + B)(\overline{B} + C)(\overline{A} + C)$
 - (ii) $(A + C)(A\overline{B} + AC)(\overline{A}\overline{B} + \overline{C})$
 - (iii) $Y = (B + \overline{C}) \cdot (A + \overline{B})$
4. Convert the following expression to product-of-sum form:
 - (i) $A + \overline{A}B + \overline{A}C$
 - (ii) $(\overline{AB} + \overline{C}) + A\overline{C}B + B$
 - (iii) $AB + ABC + A\overline{B}\overline{C} + A\overline{C}$
 - (iv) $A\overline{B} + \overline{A}\overline{B}$

Examples 1

Express the Boolean function $F = A + B'C$ as a sum of minterms.

Solution: The function has three variables: A, B, and C. The first term A is missing two variables; therefore,

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

This function is still missing one variable, so

$$\begin{aligned}A &= AB(C + C') + AB'(C + C') \\&= ABC + ABC' + AB'C + AB'C'\end{aligned}$$

The second term $B'C$ is missing one variable; hence,

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

Combining all terms, we have

$$\begin{aligned}F &= A + B'C \\&= ABC + ABC' + AB'C + AB'C' + A'B'C\end{aligned}$$

But $AB'C$ appears twice, and according to theorem $(x + x = x)$, it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

$$\begin{aligned}F &= A'B'C + AB'C + AB'C + ABC' + ABC \\&= m1 + m4 + m5 + m6 + m7\end{aligned}$$

When a Boolean function is in its sum-of-minterms form, it is sometimes convenient to express the function in the following brief notation:

$$F(A, B, C) = \sum m(1, 4, 5, 6, 7)$$

Examples 2

Express the Boolean function $F = xy + x'z$ as a product of maxterms.

Solution: First, convert the function into OR terms by using the distributive law:

$$\begin{aligned}F &= xy + x'z = (xy + x')(xy + z) \\&= (x + x')(y + x')(x + z)(y + z) \\&= (x' + y)(x + z)(y + z)\end{aligned}$$

The function has three variables: x , y , and z . Each OR term is missing one variable; therefore,

$$\begin{aligned}x' + y &= x' + y + zz' = (x' + y + z)(x' + y + z') \\x + z &= x + z + yy' = (x + y + z)(x + y' + z) \\y + z &= y + z + xx' = (x + y + z)(x' + y + z)\end{aligned}$$

Combining all the terms and removing those which appear more than once, we finally obtain

$$\begin{aligned}F &= (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z') \\F &= M0M2M4M5\end{aligned}$$

A convenient way to express this function is as

$$\text{follows: } F(x, y, z) = \pi M(0, 2, 4, 5)$$

The product symbol, π , denotes the ANDing of maxterms; the numbers are the indices of the maxterms of the function.

Unit I.2 – Switching Theory and Combinational Logic Circuits

- ✓ **Switching Theory:** - Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions- Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- ✓ **Combinational Logic Circuits:-** Review of basic gates- Universal gates, Adder, Subtractor ,Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL

Karnaugh Map (K-map)

- ✓ In the algebraic method of simplification, we need to write lengthy equations, find the common terms, manipulate the expressions etc., so it is time consuming work.
- ✓ Thus “K-map” is another simplification technique to reduce the Boolean equation.

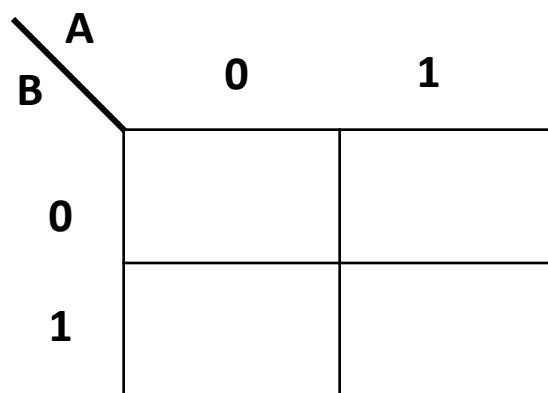
Karnaugh Map (K-map)

- ✓ It overcomes all the disadvantages of algebraic simplification techniques.
- ✓ The information contained in a truth table or available in the SOP or POS form is represented on K-map.

Karnaugh Map (K-map)

➤ K-map Structure - 2 Variable

- ✓ A & B are variables or inputs
- ✓ 0 & 1 are values of A & B
- ✓ 2 variable k-map consists of 4 boxes i.e. $2^2=4$



Karnaugh Map (K-map)

➤ K-map Structure - 2 Variable

- ✓ Inside 4 boxes we have enter values of Y i.e. output

		\bar{A}	A
	\bar{B}	0	1
\bar{B}	0	$\bar{A}\bar{B}$	$\bar{A}B$
B	1	$A\bar{B}$	AB

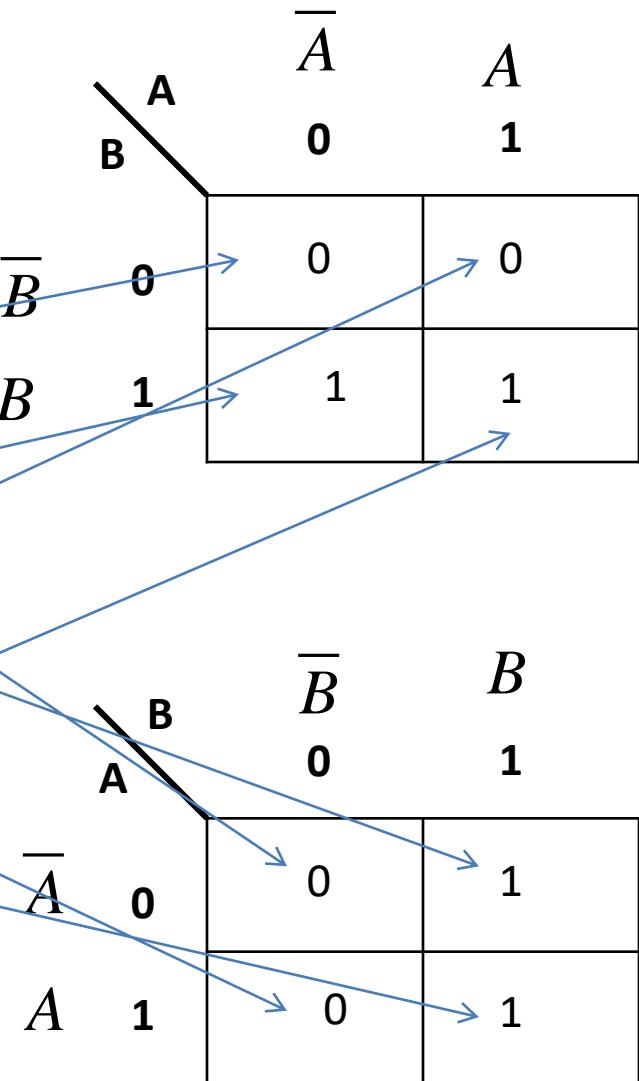
		\bar{A}	A
	\bar{B}	0	1
\bar{B}	0	m_0	m_1
B	1	m_2	m_3

K-map & its associated minterms

Karnaugh Map (K-map)

✓ Relationship between Truth Table & K-map

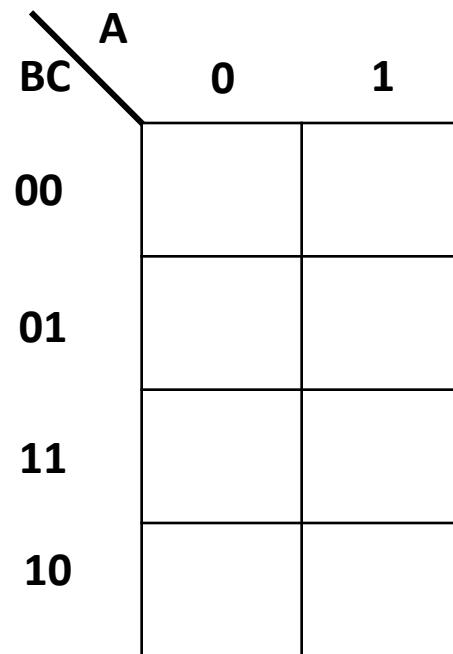
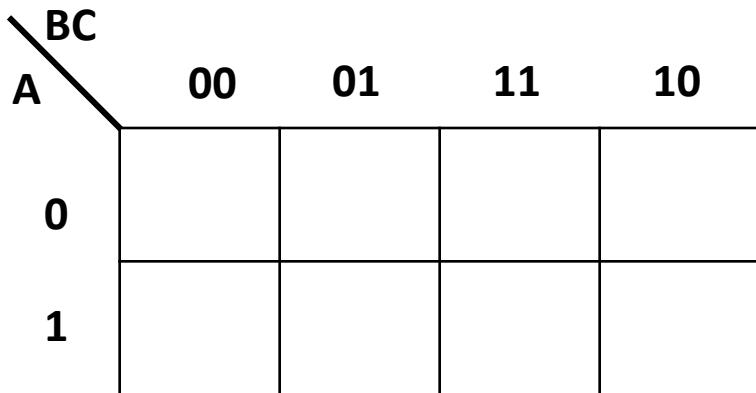
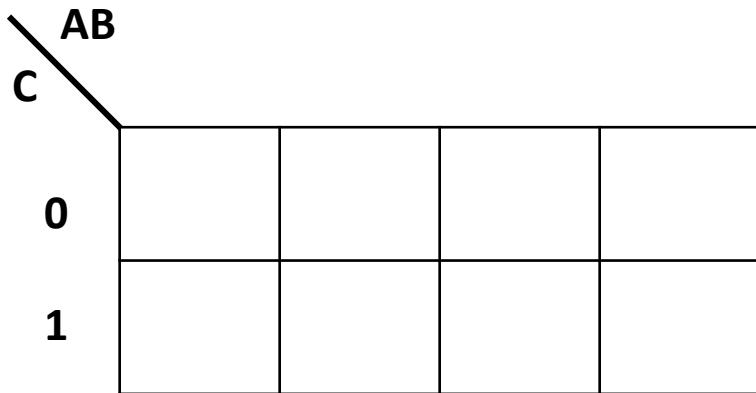
A	B	Y
0	0	0
0	1	1
1	0	0
1	1	1



Karnaugh Map (K-map)

➤ K-map Structure - 3 Variable

- ✓ A, B & C are variables or inputs
- ✓ 3 variable k-map consists of 8 boxes i.e. $2^3=8$



Karnaugh Map (K-map)

✓ 3 Variable K-map & its associated product terms

		AB	00	01	11	10	
		C	0	\bar{ABC}	\bar{ABC}	ABC	$A\bar{B}\bar{C}$
		1	1	\bar{ABC}	\bar{ABC}	ABC	$A\bar{B}C$
0	0	\bar{ABC}	\bar{ABC}	ABC	$A\bar{B}\bar{C}$	\bar{ABC}	\bar{ABC}
1	1	\bar{ABC}	\bar{ABC}	ABC	$A\bar{B}C$	\bar{ABC}	\bar{ABC}

		BC	00	01	11	10	
		A	0	\bar{ABC}	\bar{ABC}	\bar{ABC}	\bar{ABC}
		1	1	\bar{ABC}	\bar{ABC}	ABC	$A\bar{B}\bar{C}$
0	0	\bar{ABC}	\bar{ABC}	\bar{ABC}	\bar{ABC}	\bar{ABC}	\bar{ABC}
1	1	\bar{ABC}	\bar{ABC}	ABC	$A\bar{B}\bar{C}$	\bar{ABC}	\bar{ABC}

		A	0	1	
		BC	00	\bar{ABC}	$A\bar{B}\bar{C}$
		01	\bar{ABC}	$A\bar{B}C$	
		11	\bar{ABC}	ABC	
		10	\bar{ABC}	$A\bar{B}\bar{C}$	

Karnaugh Map (K-map)

✓ 3 Variable K-map & its associated minterms

AB		00	01	11	10	
C		0	m_0	m_2	m_6	m_4
		1	m_1	m_3	m_7	m_5

BC		00	01	11	10	
A		0	m_0	m_1	m_3	m_2
		1	m_4	m_5	m_7	m_6

BC		0	1	
A		00	m_0	m_4
		01	m_1	m_5
		11	m_3	m_7
		10	m_2	m_6

Karnaugh Map (K-map)

➤ K-map Structure - 4 Variable

- ✓ A, B, C & D are variables or inputs
- ✓ 4 variable k-map consists of 16 boxes i.e. $2^4=16$

		AB	CD		
		00	01	11	10
CD	00				
	01				
	11				
	10				

		AB	CD		
		00	01	11	10
CD	00				
	01				
	11				
	10				

Karnaugh Map (K-map)

✓ 4 Variable K-map and its associated product terms

		AB	CD			
		00	01	11	10	
CD		00	$\overline{ABC}\overline{D}$	$\overline{A}\overline{B}\overline{C}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$	$\overline{A}\overline{B}\overline{C}\overline{D}$
01		01	$\overline{ABC}\overline{D}$	$\overline{A}\overline{B}\overline{C}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$	$\overline{A}\overline{B}\overline{C}\overline{D}$
11		11	$\overline{ABC}\overline{D}$	$\overline{A}\overline{B}\overline{C}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$	$\overline{A}\overline{B}\overline{C}\overline{D}$
10		10	$\overline{ABC}\overline{D}$	$\overline{A}\overline{B}\overline{C}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$	$\overline{A}\overline{B}\overline{C}\overline{D}$

		CD	AB			
		00	01	11	10	
CD		00	$\overline{ABC}\overline{D}$	$\overline{ABC}\overline{D}$	$\overline{ABC}\overline{D}$	$\overline{ABC}\overline{D}$
01		01	$\overline{ABC}\overline{D}$	$\overline{ABC}\overline{D}$	$\overline{ABC}\overline{D}$	$\overline{ABC}\overline{D}$
11		11	\overline{ABC} <i>D</i>	$\overline{ABC}\overline{D}$	$ABC\overline{D}$	$ABC\overline{D}$
10		10	$ABC\overline{D}$	$ABC\overline{D}$	$ABC\overline{D}$	$ABC\overline{D}$

Karnaugh Map (K-map)

✓ 4 Variable K-map and its associated minterms

		AB	00	01	11	10
		CD	00	01	11	10
00	00	m_0	m_4	m_{12}	m_8	
	01	m_1	m_5	m_{13}	m_9	
	11	m_3	m_7	m_{15}	m_{11}	
	10	m_2	m_6	m_{14}	m_{10}	

		AB	00	01	11	10
		CD	00	01	11	10
00	00	m_0	m_1	m_3	m_2	
	01	m_4	m_5	m_7	m_6	
	11	m_{12}	m_{13}	m_{15}	m_{14}	
	10	m_8	m_9	m_{11}	m_{10}	



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



Switching Theory and Logic Design

ARI-209

By: Dr. Divya Agarwal



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT- II** 10 HOURS
- **Sequential Logic Circuits:** Latches and Flip Flops- SR, D, T and MS-JK Flip Flops, Asynchronous Inputs.
- **Counters and Shift Registers:** Design of Synchronous and Asynchronous Counters- Binary, BCD, Decade and Up/Down Counters, Shift Registers, Types of Shift Registers, Counters using Shift Registers- Ring Counter and Johnson Counter.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-III** **08 HOURS**
- **Integrated circuits:** TTL and CMOS logic families and their characteristics. Brief introduction to RAM and ROM
- **Synchronous Sequential Circuits:** State Tables State Equations and State Diagrams, State Reduction and State Assignment, Design of Clocked Sequential Circuits using State Equations.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-IV** 06 HOURS
- **Finite state machine:** capabilities and limitations, Mealy and Moore models-minimization of completely specified and incompletely specified sequential machines, Partition techniques and merger chart methods, concept of minimal cover table.
- **Algorithmic State Machine:** Representation of sequential circuits using ASM charts synthesis of output and next state functions, Data path control path partition-based design.



Text Books:

1. Leach and Malvino (2011). *Digital principles and Applications*. Tata McGraw-Hill Education
2. Mano, M. M. (2017). *Digital logic and computer design*. Pearson Education India.
3. Jain, R. P. (2003). *Modern digital electronics*. Tata McGraw-Hill Education.

Reference Books:

1. A Anand Kumar. (2016) *Fundamentals of Digital Logic Circuits*, PHI
2. Taub, H., & Schilling, D. L. (1977). *Digital integrated electronics*. McGraw-Hill College.



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.

Karnaugh Map (K-map)

- ✓ In the algebraic method of simplification, we need to write lengthy equations, find the common terms, manipulate the expressions etc., so it is time consuming work.
- ✓ Thus “K-map” is another simplification technique to reduce the Boolean equation.

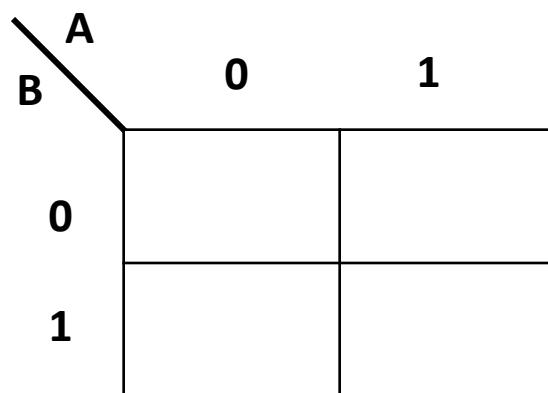
Karnaugh Map (K-map)

- ✓ It overcomes all the disadvantages of algebraic simplification techniques.
- ✓ The information contained in a truth table or available in the SOP or POS form is represented on K-map.

Karnaugh Map (K-map)

➤ K-map Structure - 2 Variable

- ✓ A & B are variables or inputs
- ✓ 0 & 1 are values of A & B
- ✓ 2 variable k-map consists of 4 boxes i.e. $2^2=4$



Karnaugh Map (K-map)

➤ K-map Structure - 2 Variable

- ✓ Inside 4 boxes we have enter values of Y i.e. output

		\bar{A}	A
	\bar{B}	0	1
\bar{B}	0	$\bar{A}\bar{B}$	$\bar{A}B$
B	1	$A\bar{B}$	AB

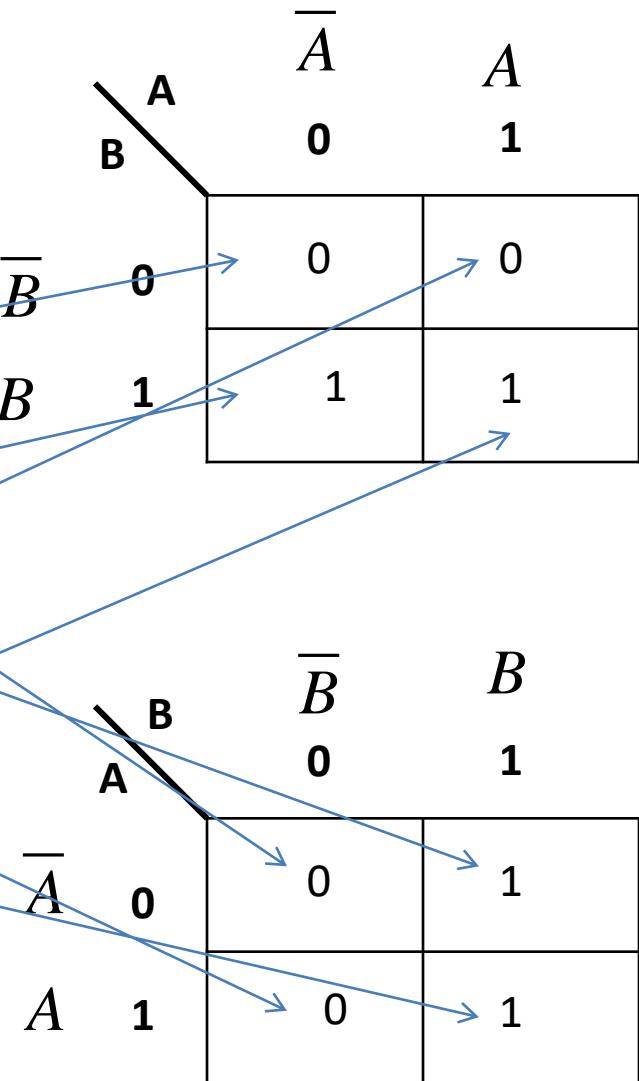
		\bar{A}	A
	\bar{B}	0	1
\bar{B}	0	m_0	m_1
B	1	m_2	m_3

K-map & its associated minterms

Karnaugh Map (K-map)

✓ Relationship between Truth Table & K-map

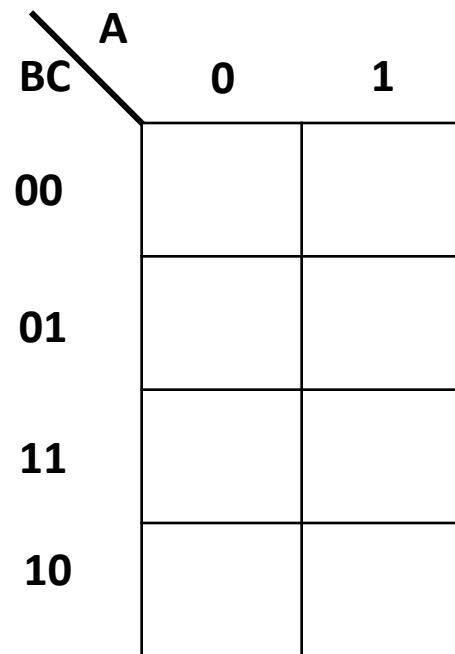
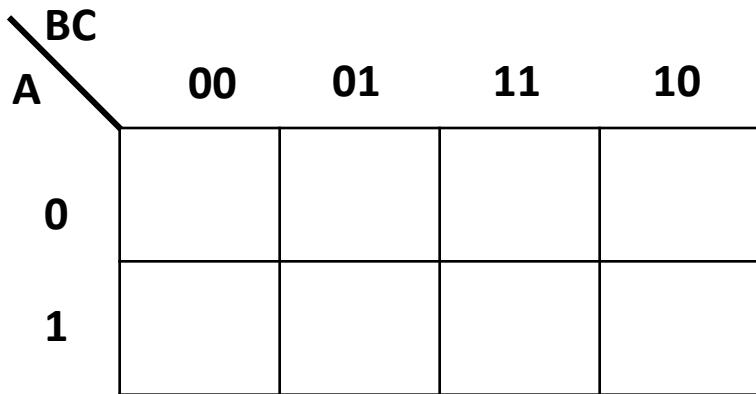
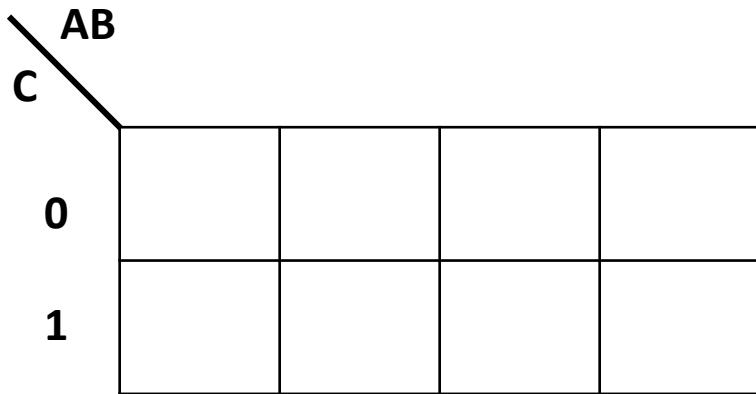
A	B	Y
0	0	0
0	1	1
1	0	0
1	1	1



Karnaugh Map (K-map)

➤ K-map Structure - 3 Variable

- ✓ A, B & C are variables or inputs
- ✓ 3 variable k-map consists of 8 boxes i.e. $2^3=8$



Karnaugh Map (K-map)

✓ 3 Variable K-map & its associated product terms

		AB	00	01	11	10	
		C	0	\bar{ABC}	\bar{ABC}	ABC	$A\bar{B}\bar{C}$
		1	1	\bar{ABC}	\bar{ABC}	ABC	$A\bar{B}C$
0	0	\bar{ABC}	\bar{ABC}	ABC	$A\bar{B}\bar{C}$	\bar{ABC}	\bar{ABC}
1	1	\bar{ABC}	\bar{ABC}	ABC	$A\bar{B}C$	\bar{ABC}	\bar{ABC}

		BC	00	01	11	10	
		A	0	\bar{ABC}	\bar{ABC}	\bar{ABC}	\bar{ABC}
		1	1	\bar{ABC}	\bar{ABC}	ABC	$A\bar{B}\bar{C}$
0	0	\bar{ABC}	\bar{ABC}	\bar{ABC}	\bar{ABC}	\bar{ABC}	\bar{ABC}
1	1	\bar{ABC}	\bar{ABC}	ABC	$A\bar{B}\bar{C}$	\bar{ABC}	\bar{ABC}

		A	0	1	
		BC	00	\bar{ABC}	$A\bar{B}\bar{C}$
		01	\bar{ABC}	$A\bar{B}C$	
		11	\bar{ABC}	ABC	
		10	\bar{ABC}	$A\bar{B}\bar{C}$	

Karnaugh Map (K-map)

✓ 3 Variable K-map & its associated minterms

		AB	00	01	11	10	
		C	0	m_0	m_2	m_6	m_4
		1	1	m_1	m_3	m_7	m_5

		BC	00	01	11	10	
		A	0	m_0	m_1	m_3	m_2
		1	1	m_4	m_5	m_7	m_6

		A	0	1	
		BC	00	m_0	m_4
		01	m_1	m_5	
0	11	m_3	m_7		
1	10	m_2	m_6		

Karnaugh Map (K-map)

➤ K-map Structure - 4 Variable

- ✓ A, B, C & D are variables or inputs
- ✓ 4 variable k-map consists of 16 boxes i.e. $2^4=16$

		AB	CD		
		00	01	11	10
CD	00				
	01				
	11				
	10				

		AB	CD		
		00	01	11	10
CD	00				
	01				
	11				
	10				

Karnaugh Map (K-map)

✓ 4 Variable K-map and its associated product terms

		AB	CD			
		00	01	11	10	
		00	$\overline{AB}\overline{CD}$	$\overline{ABC}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$
		01	$\overline{ABC}\overline{D}$	$\overline{AB}\overline{CD}$	$AB\overline{C}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$
		11	$\overline{ABC}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$	$ABC\overline{D}$	$A\overline{B}CD$
		10	$\overline{ABC}\overline{D}$	$\overline{AB}\overline{CD}$	$ABC\overline{D}$	$A\overline{B}\overline{C}\overline{D}$

		CD	AB			
		00	01	11	10	
		00	$\overline{AB}\overline{CD}$	$\overline{ABC}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$
		01	$A\overline{B}\overline{C}\overline{D}$	$\overline{ABC}\overline{D}$	$\overline{ABC}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$
		11	$A\overline{B}\overline{C}$	$AB\overline{C}\overline{D}$	$ABC\overline{D}$	$A\overline{B}\overline{C}\overline{D}$
		10	$ABC\overline{D}$	$A\overline{B}\overline{C}\overline{D}$	$ABC\overline{D}$	$A\overline{B}\overline{C}\overline{D}$

Karnaugh Map (K-map)

✓ 4 Variable K-map and its associated minterms

		AB	00	01	11	10
		CD	00	01	11	10
00	00	m_0	m_4	m_{12}	m_8	
	01	m_1	m_5	m_{13}	m_9	
	11	m_3	m_7	m_{15}	m_{11}	
	10	m_2	m_6	m_{14}	m_{10}	

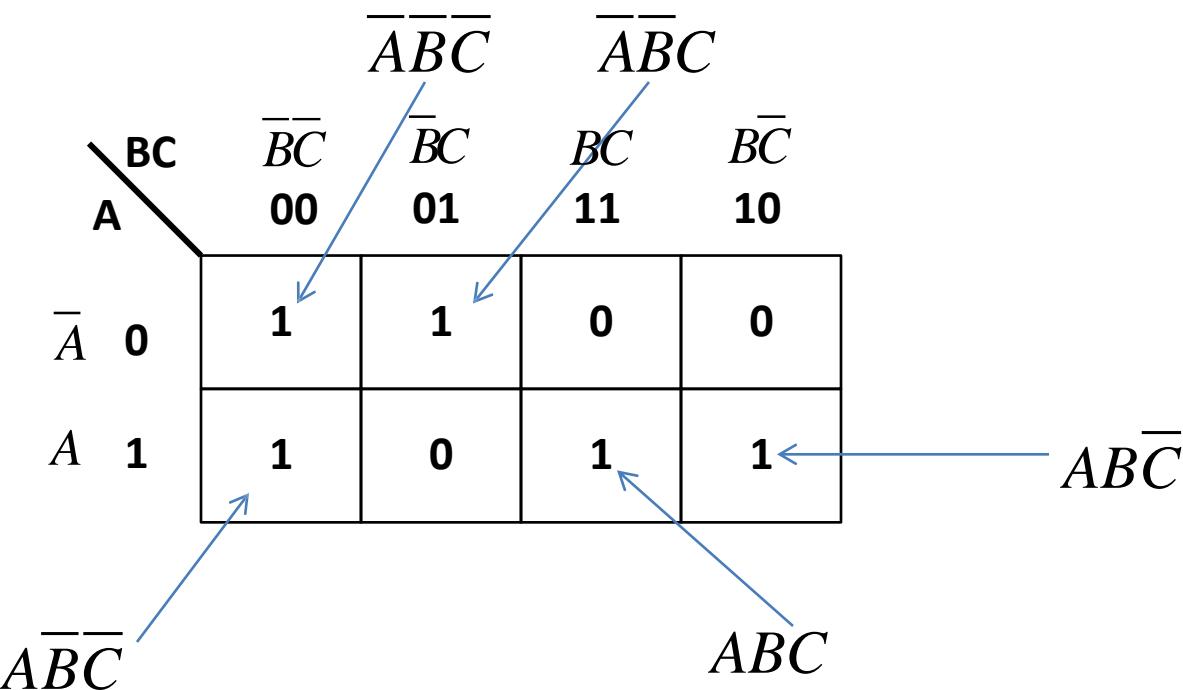
		AB	00	01	11	10
		CD	00	01	11	10
00	00	m_0	m_1	m_3	m_2	
	01	m_4	m_5	m_7	m_6	
	11	m_{12}	m_{13}	m_{15}	m_{14}	
	10	m_8	m_9	m_{11}	m_{10}	

Representation of Standard SOP form expression on K-map

For example, SOP equation is given as

$$Y = \overline{ABC} + \overline{A}\overline{B}C + A\overline{B}\overline{C} + A\overline{B}C + ABC$$

- ✓ The given expression is in the standard SOP form.
- ✓ Each term represents a minterm.
- ✓ We have to enter '1' in the boxes corresponding to each minterm as below



Simplification of K-map

- ✓ Once we plot the logic function or truth table on K-map, we have to use the grouping technique for simplifying the logic function.
- ✓ Grouping means the combining the terms in adjacent cells.
- ✓ The grouping of either 1's or 0's results in the simplification of Boolean expression.

Simplification of K-map

- ✓ If we group the adjacent 1's then the result of simplification is SOP form
- ✓ If we group the adjacent 0's then the result of simplification is POS form

Grouping

- ✓ While grouping, we should group most number of 1's.
- ✓ The grouping follows the binary rule i.e we can group 1,2,4,8,16,32,.....number of 1's.
- ✓ We cannot group 3,5,7,.....number of 1's
- ✓ **Pair:** A group of two adjacent 1's is called as Pair
- ✓ **Quad:** A group of four adjacent 1's is called as Quad
- ✓ **Octet:** A group of eight adjacent 1's is called as Octet

Grouping of Two Adjacent 1's : Pair

✓ A pair eliminates 1 variable

		$\bar{A}BC$		$\bar{A}\bar{B}\bar{C}$	
		$\bar{B}C$	$\bar{B}C$	BC	$B\bar{C}$
		00	01	11	10
A	0	0	0	1	1
	1	0	0	0	0

$$Y = \bar{A}BC + \bar{A}\bar{B}\bar{C}$$

$$Y = \bar{A}B(C + \bar{C})$$

$$Y = \bar{A}B$$

$$(\because C + \bar{C} = 1)$$

Grouping of Two Adjacent 1's : Pair

		\overline{BC}	\overline{BC}	\overline{BC}	BC	BC
		00	01	11	11	10
A	\overline{A}	0	0	0	0	0
	A	1	1	0	0	1

		\overline{BC}	BC	BC	BC	BC
		00	01	11	10	10
\overline{A}	0	0	1	1	1	1
	1	0	0	1	0	0

		\overline{BC}	\overline{BC}	BC	BC
		00	01	11	10
A	\overline{A}	0	1	0	0
	1	0	1	0	0

		\overline{B}	B
		0	1
\overline{A}	0	1	1
	1	1	0

Grouping of Two Adjacent 1's : Pair

\overline{AB}	\overline{CD} 00	\overline{CD} 01	CD 11	\overline{CD} 10
\overline{AB}	0	1	0	0
\overline{AB}	0	0	0	0
AB	0	0	0	0
$A\bar{B}$	0	1	0	0

Possible Grouping of Four Adjacent 1's : Quad

✓ A Quad eliminates 2 variable

\bar{AB}	CD	\bar{CD}	$\bar{C}D$	CD	$C\bar{D}$
\bar{AB}	00	01	11	10	
\bar{AB}	00	0	0	0	0
\bar{AB}	01	0	0	0	0
AB	11	0	0	0	0
$A\bar{B}$	10	1	1	1	1

\bar{AB}	CD	\bar{CD}	$\bar{C}D$	CD	$C\bar{D}$
\bar{AB}	00	01	11	10	
\bar{AB}	00	0	1	0	0
\bar{AB}	01	0	1	0	0
AB	11	0	1	0	0
$A\bar{B}$	10	0	1	0	0

Possible Grouping of Four Adjacent 1's : Quad

✓ A Quad eliminates 2 variable

\bar{AB}	CD	\bar{CD}	$\bar{C}D$	$C\bar{D}$	CD
\bar{AB}	00	00	01	11	10
\bar{AB}	00	0	0	0	0
\bar{AB}	01	1	1	0	0
\bar{AB}	11	1	1	0	0
\bar{AB}	10	0	0	0	0

\bar{AB}	CD	\bar{CD}	$\bar{C}D$	$C\bar{D}$	CD
\bar{AB}	00	0	1	1	0
\bar{AB}	01	0	0	0	0
\bar{AB}	11	0	0	0	0
\bar{AB}	10	0	1	1	0

Possible Grouping of Four Adjacent 1's : Quad

✓ A Quad eliminates 2 variable

\bar{AB}	CD	\bar{CD}	\bar{CD}	CD	CD
\bar{AB}	00	00	01	11	10
\bar{AB}	00	1	0	0	1
\bar{AB}	01	0	0	0	0
\bar{AB}	11	0	0	0	0
\bar{AB}	10	1	0	0	1

\bar{AB}	CD	\bar{CD}	\bar{CD}	CD	CD
\bar{AB}	00	00	01	11	10
\bar{AB}	00	0	0	0	0
\bar{AB}	01	1	0	0	1
\bar{AB}	11	1	0	0	1
\bar{AB}	10	0	0	0	0

Possible Grouping of Four Adjacent 1's : Quad

✓ A Quad eliminates 2 variable

\bar{AB}	CD	\bar{CD}	$\bar{C}D$	$C\bar{D}$	CD	$\bar{C}\bar{D}$
\bar{AB}	00	00	01	11	11	10
\bar{AB}	00	0	0	0	0	0
\bar{AB}	01	0	1	1	1	1
\bar{AB}	11	0	1	1	1	1
\bar{AB}	10	0	0	0	0	0

\bar{AB}	CD	\bar{CD}	$\bar{C}D$	$C\bar{D}$	CD	$\bar{C}\bar{D}$
\bar{AB}	00	00	01	11	11	10
\bar{AB}	00	0	0	0	0	0
\bar{AB}	01	0	1	1	1	0
\bar{AB}	11	0	1	1	1	0
\bar{AB}	10	0	1	1	1	0

Possible Grouping of Eight Adjacent 1's : Octet

✓ A Octet eliminates 3 variable

\bar{AB}	\bar{CD}	\bar{CD}	\bar{CD}	CD	CD
\bar{AB}	00	01	11	11	10
AB	0	0	0	0	0
\bar{AB}	0	0	0	0	0
AB	1	1	1	1	1
\bar{AB}	1	1	1	1	1

\bar{AB}	\bar{CD}	\bar{CD}	\bar{CD}	CD	CD
\bar{AB}	00	01	11	11	10
AB	0	1	1	0	0
\bar{AB}	0	1	1	0	0
AB	0	1	1	0	0
\bar{AB}	0	1	1	0	0

Possible Grouping of Eight Adjacent 1's : Octet

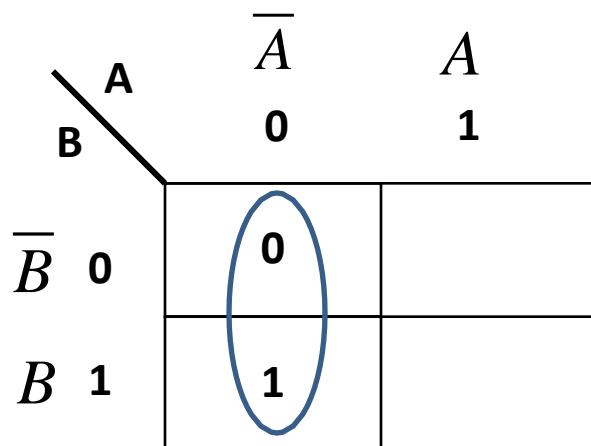
✓ A Octet eliminates 3 variable

\bar{AB}	CD	\bar{CD}	\bar{CD}	CD	CD	\bar{CD}
\bar{AB}	00	00	01	11	11	10
\bar{AB}	00	1	1	1	1	1
\bar{AB}	01	0	0	0	0	0
\bar{AB}	11	0	0	0	0	0
\bar{AB}	10	1	1	1	1	1

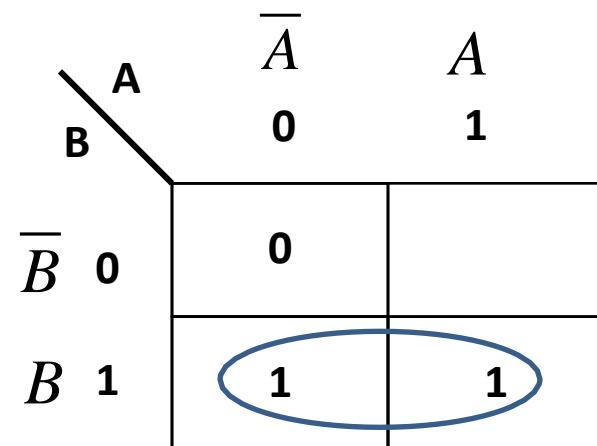
\bar{AB}	CD	\bar{CD}	\bar{CD}	CD	CD	\bar{CD}
\bar{AB}	00	00	01	11	11	10
\bar{AB}	00	1	0	0	1	
\bar{AB}	01	1	0	0	1	
\bar{AB}	11	1	0	0	1	
\bar{AB}	10	1	0	0	1	

Rules for K-map simplification

- Groups may not include any cell containing a zero.***



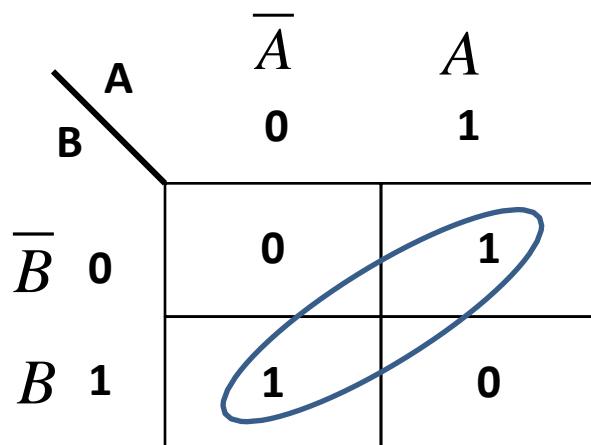
Not Accepted



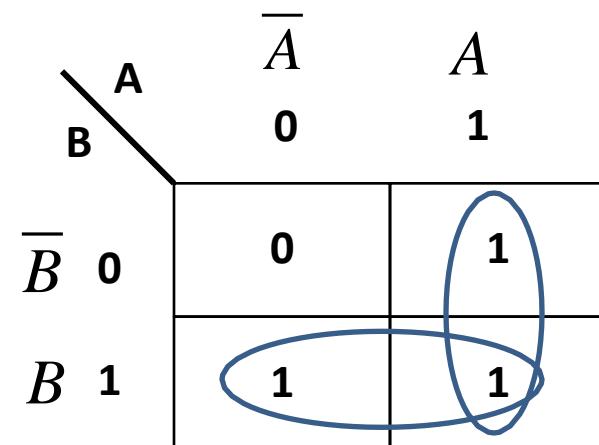
Accepted

Rules for K-map simplification

2. Groups may be horizontal or vertical, but may not be diagonal



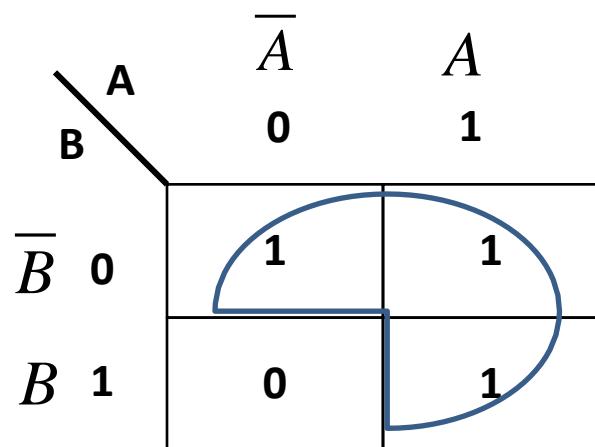
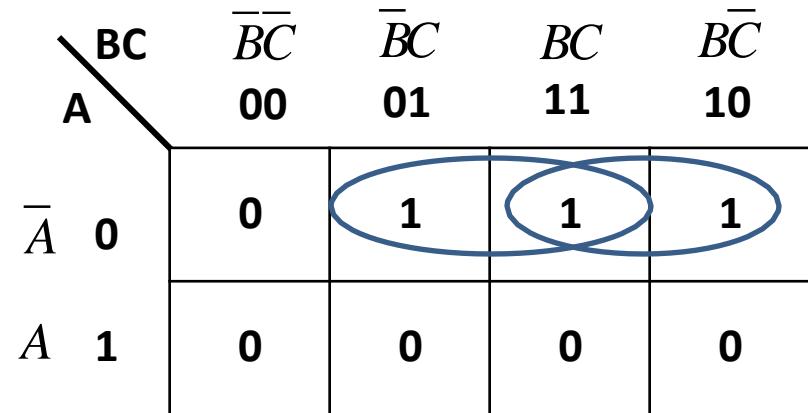
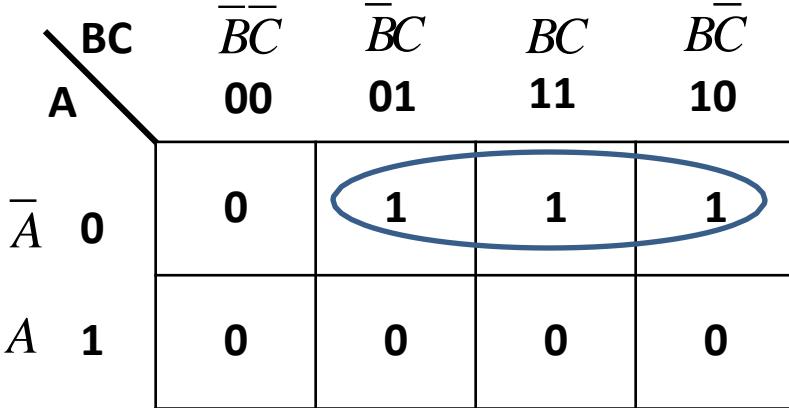
Not Accepted



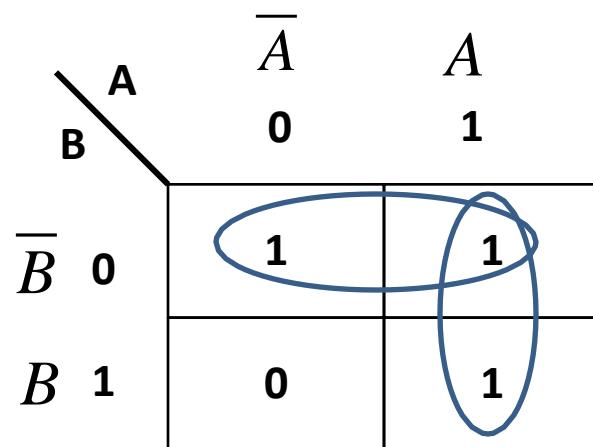
Accepted

Rules for K-map simplification

3. Groups must contain 1,2,4,8 or in general 2^n cells



Not Accepted



Accepted

Rules for K-map simplification

4. Each group should be as large as possible

		$\bar{B}\bar{C}$	$\bar{B}C$	$B\bar{C}$	BC
		00	01	11	10
A	\bar{A}	1	1	1	1
	A	0	0	1	1

Not Accepted

		$\bar{B}\bar{C}$	$\bar{B}C$	$B\bar{C}$	BC
		00	01	11	10
A	\bar{A}	1	1	1	1
	A	0	0	1	1

Accepted

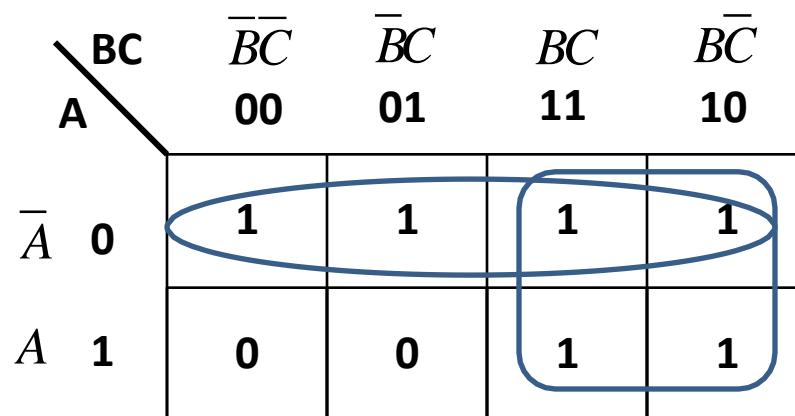
Rules for K-map simplification

5. *Each cell containing a one must be in at least one group*

		$\bar{B}\bar{C}$	$\bar{B}C$	$B\bar{C}$	BC
		00	01	11	10
A	\bar{A}	0	0	0	1
	A	0	0	1	0

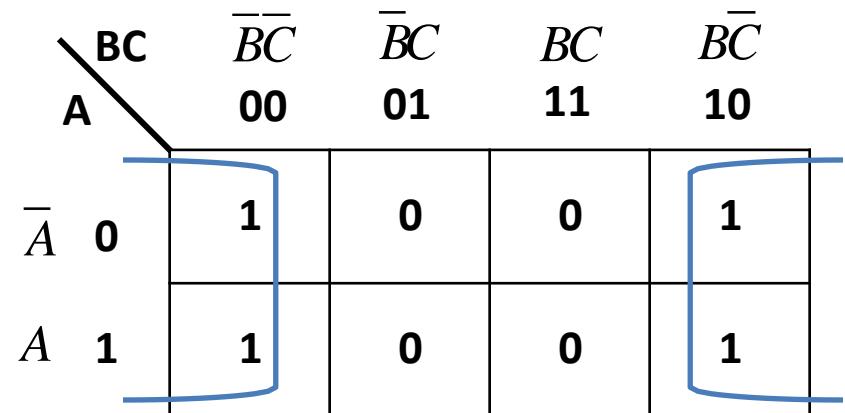
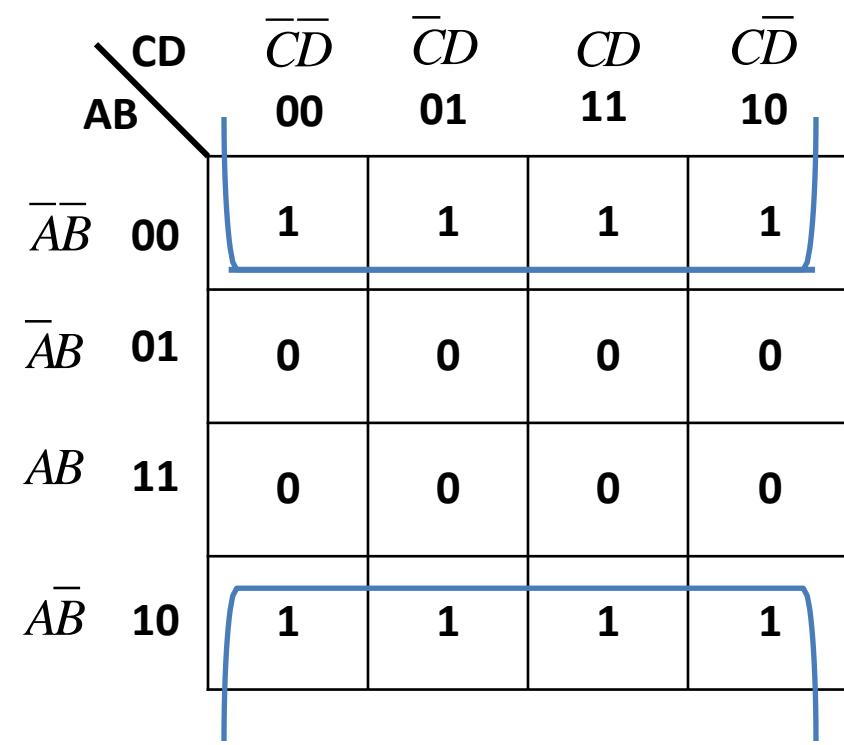
Rules for K-map simplification

6. Groups may be overlap



Rules for K-map simplification

7. Groups may wrap around the table. The leftmost cell in a row may be grouped with rightmost cell and the top cell in a column may be grouped with bottom cell



Rules for K-map simplification

8. There should be as few groups as possible, as long as this does not contradict any of the previous rules.

		$\bar{B}C$	$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
		00	01	11	10	
		\bar{A}	0	1	0	1
\bar{A}		1	1	1	1	
A		0	0	1	1	

Not Accepted

		$\bar{B}C$	$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
		00	01	11	10	
		\bar{A}	0	1	0	1
\bar{A}		1	1	1	1	
A		0	0	1	1	
		1	0	1	1	

Accepted

Rules for K-map simplification

9. A pair eliminates one variable.

10. A Quad eliminates two variables.

11. A octet eliminates three variables

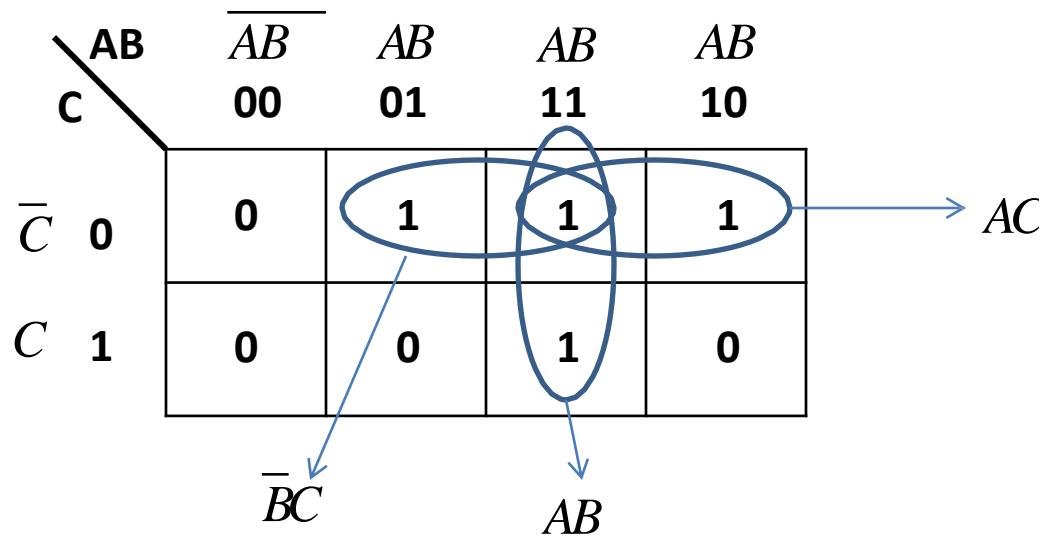
Example 1

For the given K-map write simplified Boolean expression

		$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
		00	01	11	10
C	0	0	1	1	1
	1	0	0	1	0

Example 1

continue.....



Simplified Boolean expression

$$Y = B\bar{C} + AB + \bar{A}C$$

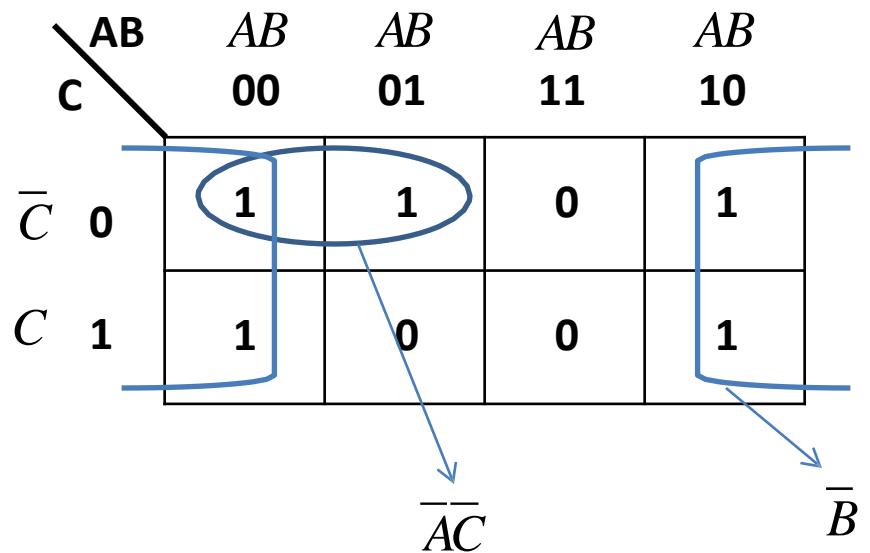
Example 2

For the given K-map write simplified Boolean expression

		$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
		00	01	11	10
C	0	1	1	0	1
	1	1	0	0	1

Example 2

continue.....



Simplified Boolean expression

$$Y = \overline{B} + A'C'$$

Example 3

A logical expression in the standard SOP form is as follows;

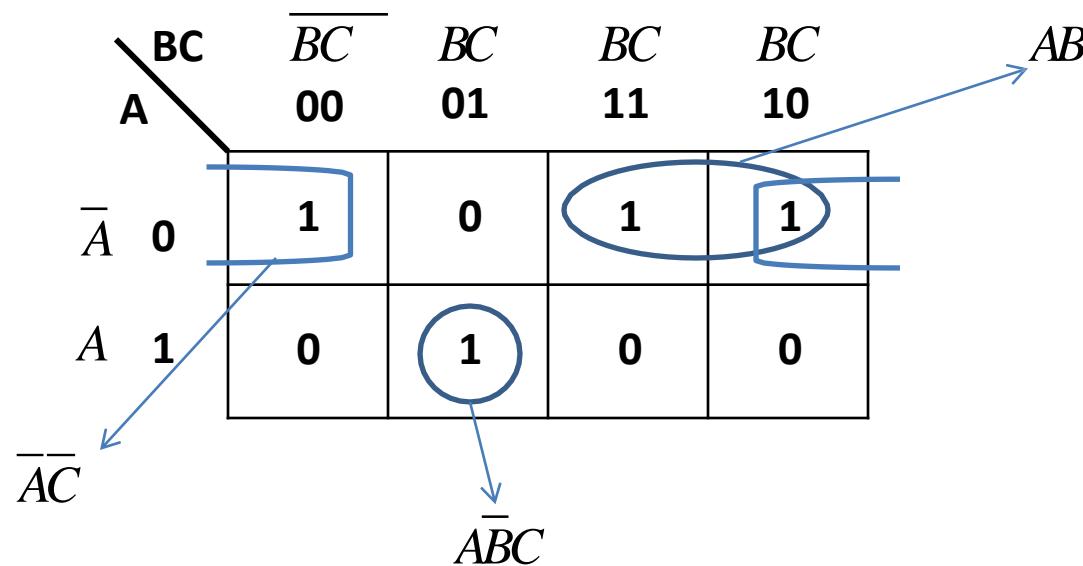
$$Y = \overline{A} \overline{B} \overline{C} + \overline{A} B \overline{C} + \overline{A} B C + A \overline{B} C$$

Minimize it with using the K-map technique

Example 3

continue.....

$$Y = \overline{A} \overline{B} \overline{C} + \overline{A} B \overline{C} + \overline{A} B C + A \overline{B} \overline{C}$$



Simplified Boolean expression

$$Y = \overline{A} \overline{C} + \overline{A} B + A \overline{B} C$$

Example 4

A logical expression representing a logic circuit is;

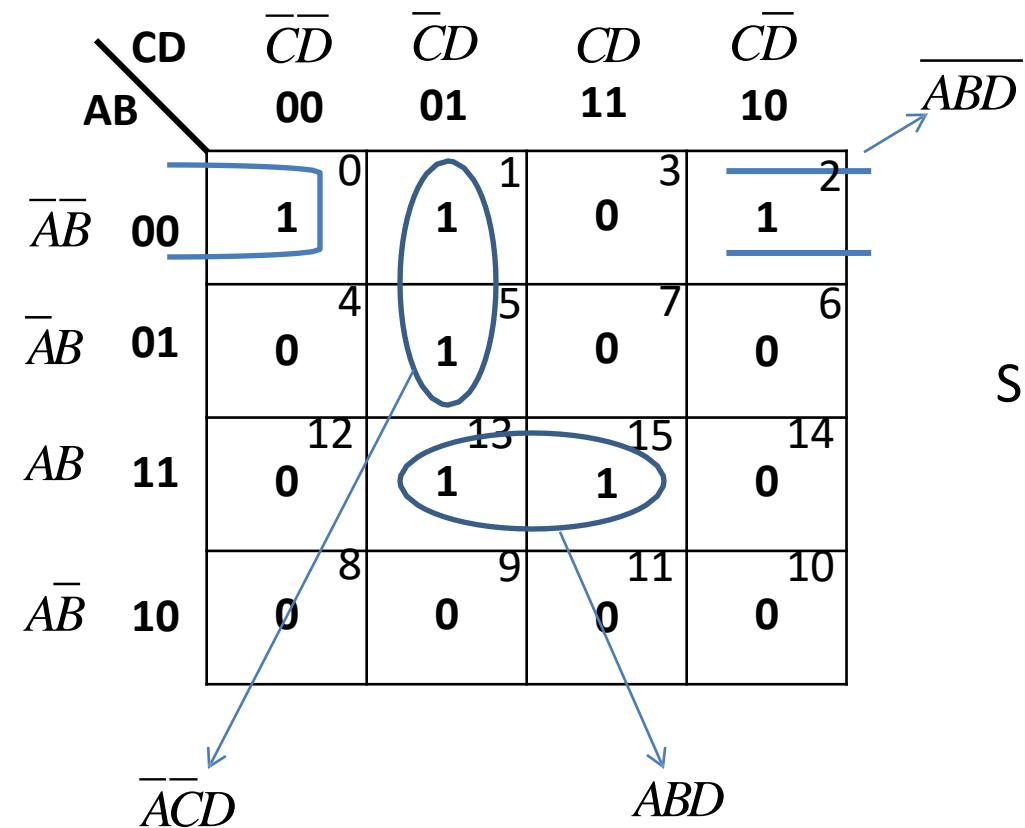
$$Y = \Sigma m (0, 1, 2, 5, 13, 15)$$

Draw the K-map and find the minimized logical expression

Example 4

continue.....

$$Y = \sum m (0, 1, 2, 5, 13, 15)$$



Simplified Boolean expression

$$Y = \overline{ABD} + \overline{ACD} + ABD$$

Example 5

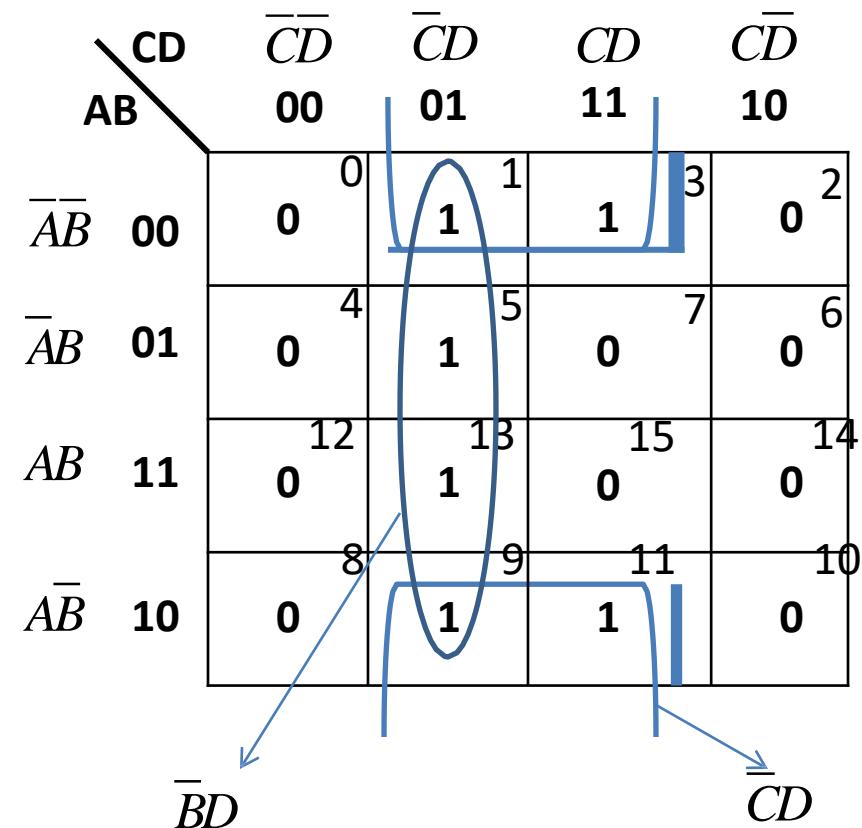
Minimize the following Boolean expression using K-map ;

$$f(A, B, C, D) = \Sigma m(1, 3, 5, 9, 11, 13)$$

Example 5

continue.....

$$f(A, B, C, D) = \sum m(1, 3, 5, 9, 11, 13)$$



Simplified Boolean expression

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

$$f = D(\bar{B} + \bar{C})$$

Example 6

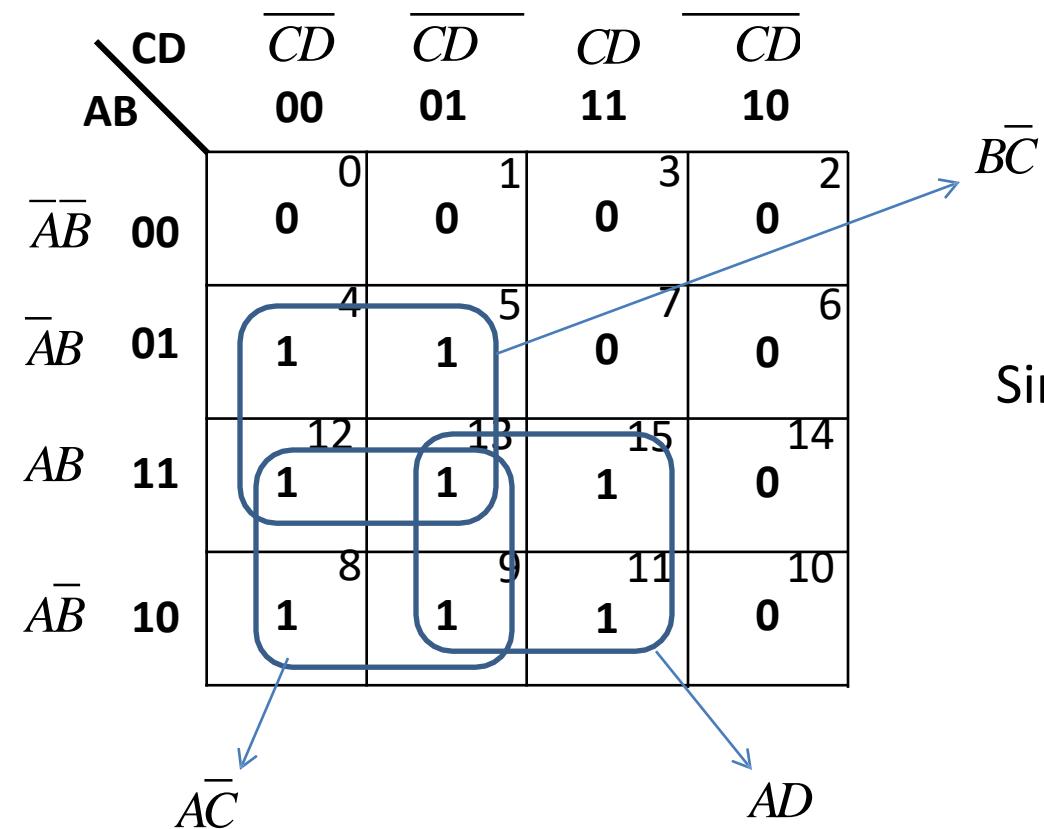
Minimize the following Boolean expression using K-map ;

$$f(A, B, C, D) = \Sigma m(4, 5, 8, 9, 11, 12, 13, 15)$$

Example 6

continue.....

$$f(A, B, C, D) = \sum m(4, 5, 8, 9, 11, 12, 13, 15)$$



Simplified Boolean expression

$$f = B\bar{C} + A\bar{C} + AD$$

Example 7

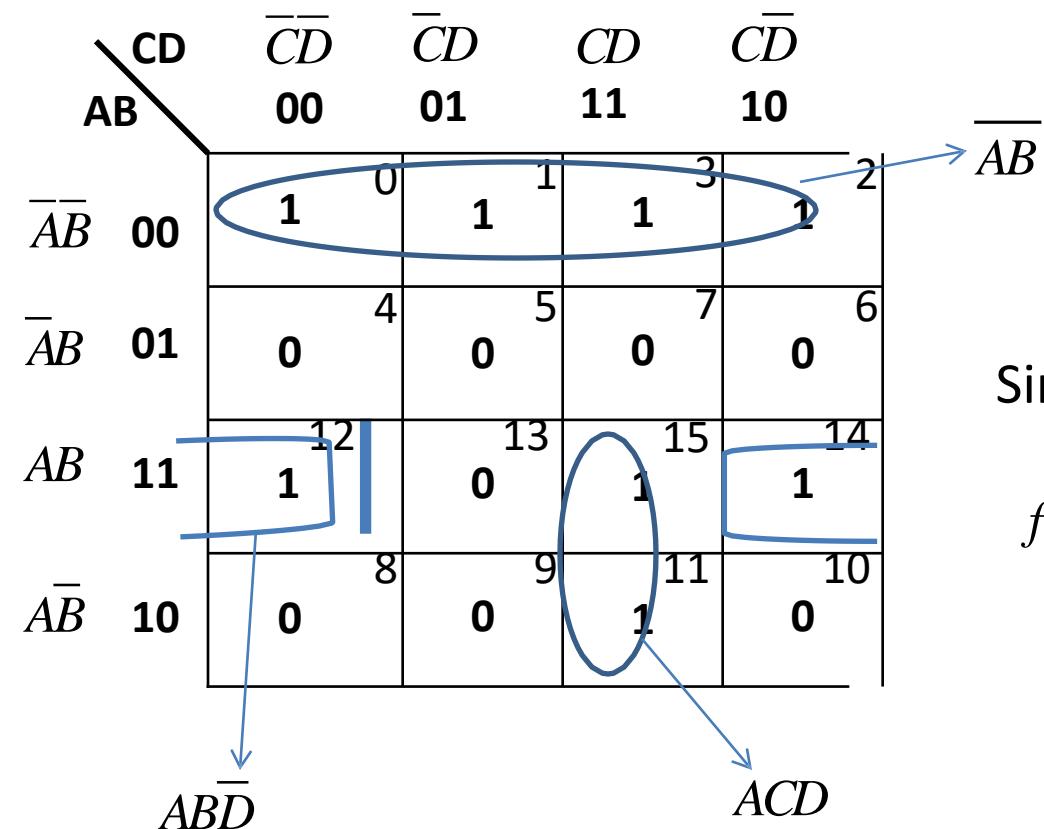
Minimize the following Boolean expression using K-map ;

$$f_2(A, B, C, D) = \Sigma m(0, 1, 2, 3, 11, 12, 14, 15)$$

Example 7

continue....

$$f_2(A, B, C, D) = \sum m(0, 1, 2, 3, 11, 12, 14, 15)$$



Simplified Boolean expression

$$f_2 = \overline{A}\overline{B} + A\overline{B}D' + ACD$$

Example 8

Solve the following expression with K-maps;

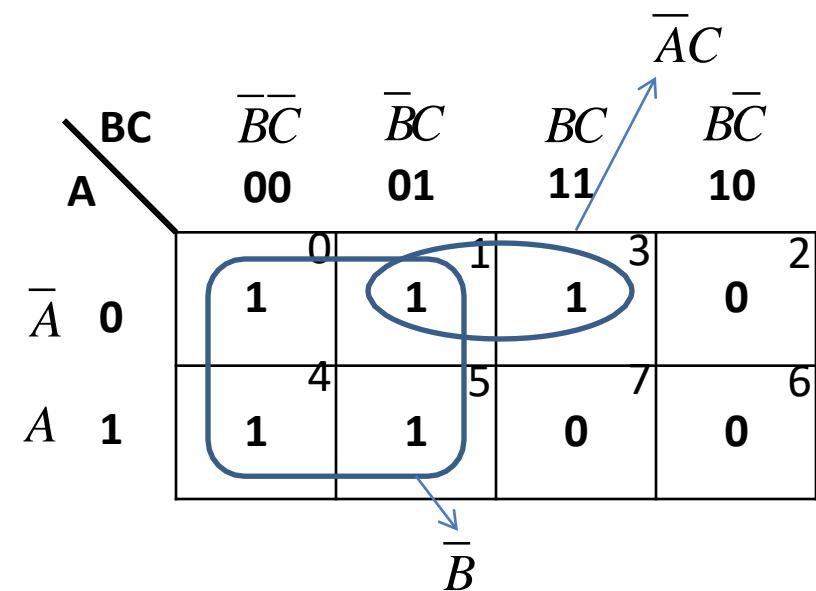
1. $f_1(A, B, C) = \Sigma m(0, 1, 3, 4, 5)$
2. $f_2(A, B, C) = \Sigma m(0, 1, 2, 3, 6, 7)$

Example 8

continue.....

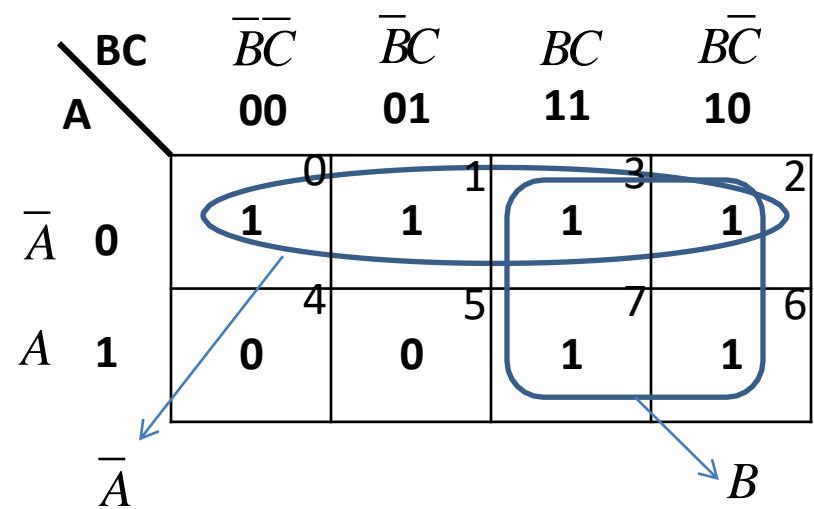
$$f_1(A, B, C) = \Sigma m(0, 1, 3, 4, 5)$$

$$f_2(A, B, C) = \Sigma m(0, 1, 2, 3, 6, 7)$$



Simplified Boolean expression

$$f_1 = \bar{A}C + \bar{B}$$



Simplified Boolean expression

$$f_2 = \bar{A} + B$$

Example 9

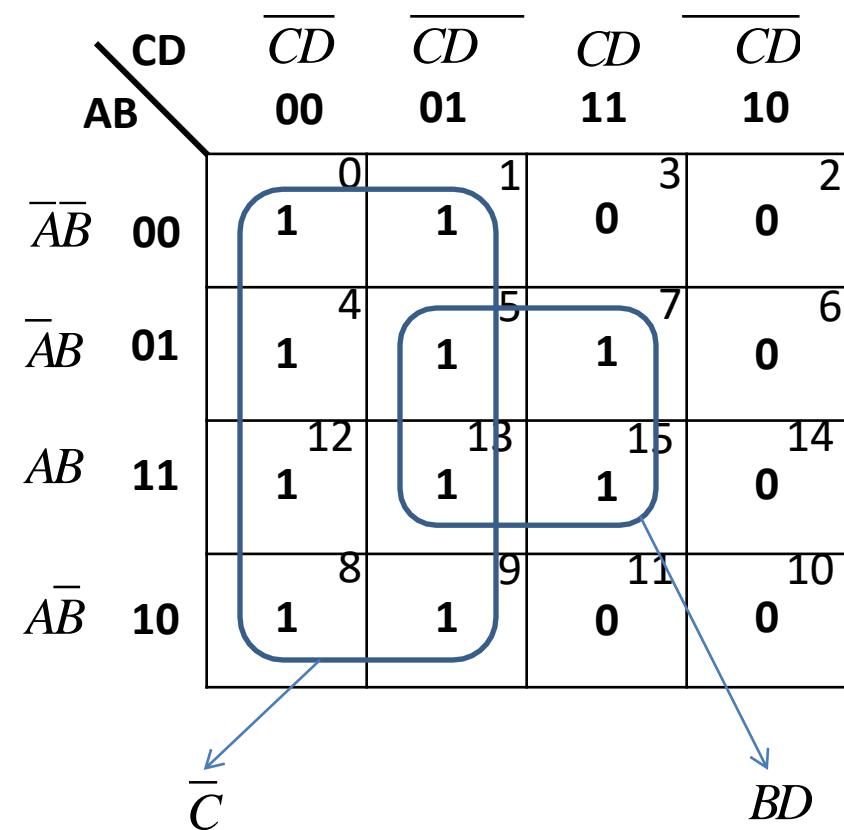
Simplify ;

$$f(A, B, C, D) = \Sigma m(0, 1, 4, 5, 7, 8, 9, 12, 13, 15)$$

Example 9

continue....

$$f(A, B, C, D) = \sum m(0, 1, 4, 5, 7, 8, 9, 12, 13, 15)$$



Simplified Boolean expression

$$f = \overline{C} + BD$$

Example 10

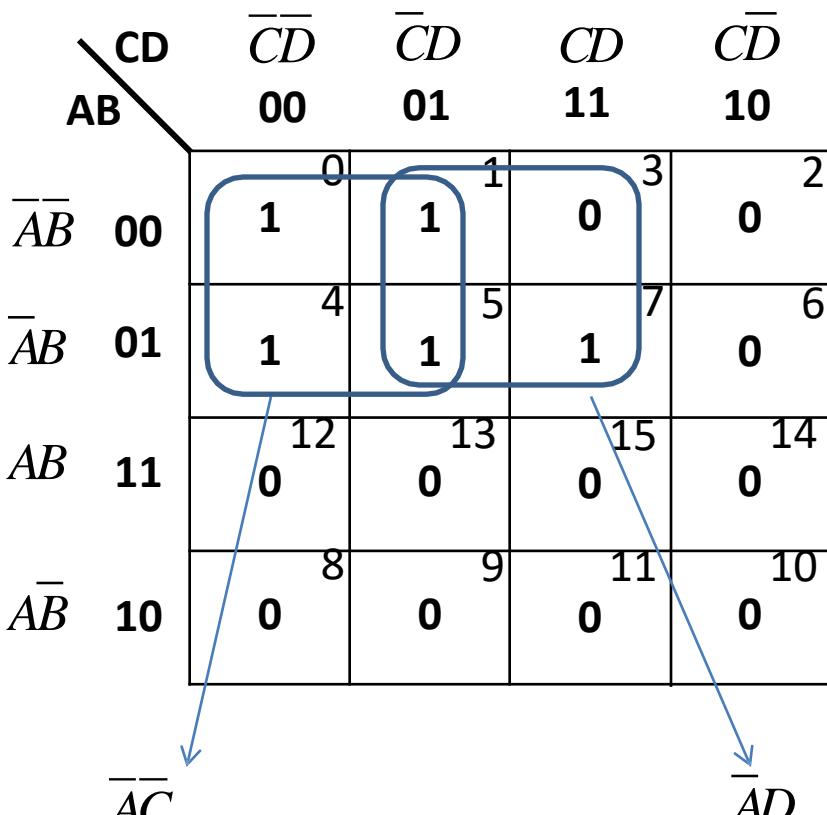
Solve the following expression with K-maps;

1. $f_1(A, B, C, D) = \Sigma m(0, 1, 3, 4, 5, 7)$
2. $f_2(A, B, C) = \Sigma m(0, 1, 3, 4, 5, 7)$

Example 10

continue.....

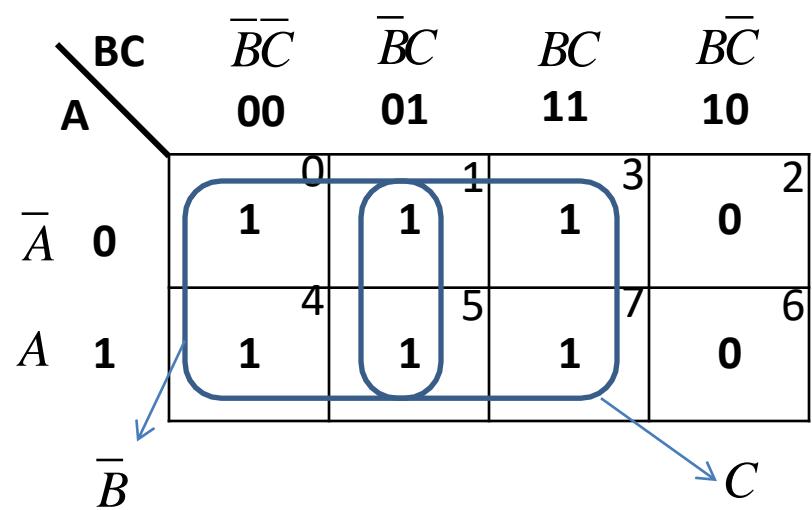
$$f_1(A, B, C, D) = \Sigma m(0, 1, 3, 4, 5, 7)$$



Simplified Boolean expression

$$f_1 = \overline{AC} + \overline{AD}$$

$$f_2(A, B, C) = \Sigma m(0, 1, 3, 4, 5, 7)$$



Simplified Boolean expression

$$f_2 = \overline{B} + C$$

K-map for Product of Sum Form (POS Expressions)

- ✓ Karnaugh map can also be used for Boolean expression in the Product of sum form (POS).
- ✓ The procedure for simplification of expression by grouping of cells is also similar

K-map for Product of Sum Form (POS Expressions)

- ✓ The letters with bars (NOT) represent 1 and unbarred letters represent 0 of Binary.
- ✓ A zero is put in the cell for which there is a term in the Boolean expression
- ✓ Grouping is done for adjacent cells containing zeros.

Example 11

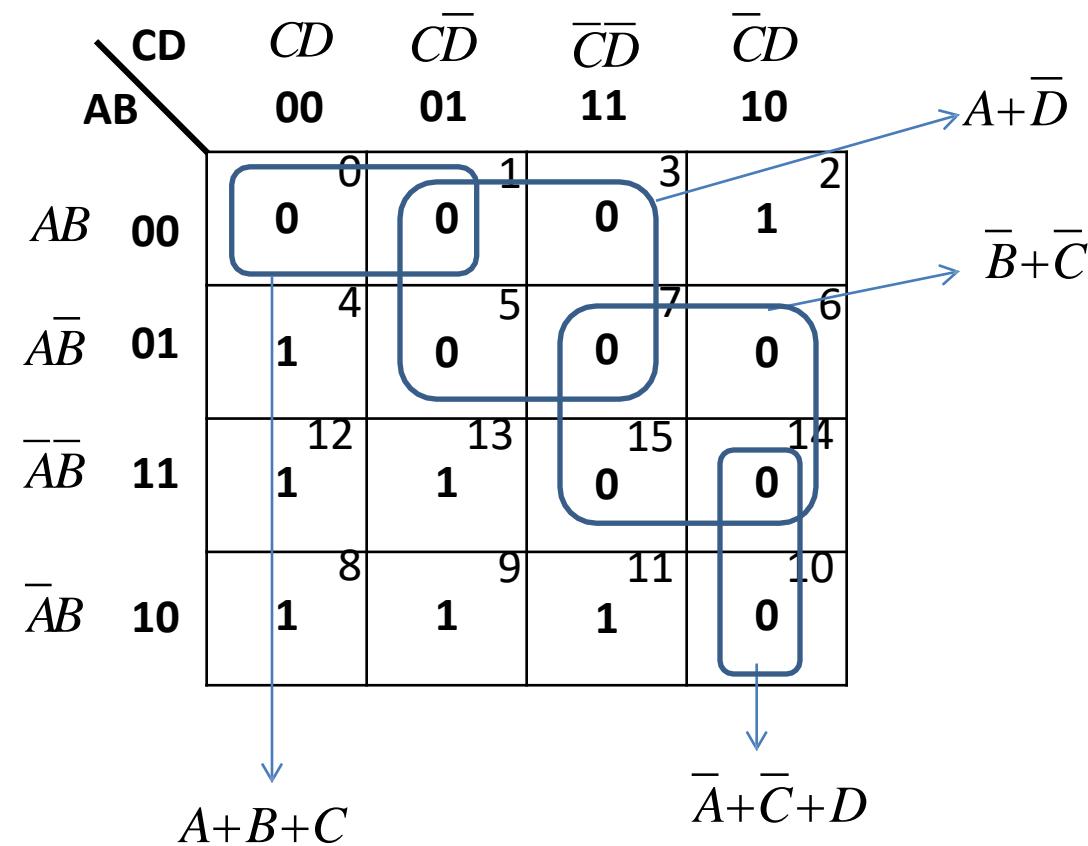
Simplify ;

$$f(A, B, C, D) = \prod M(0, 1, 3, 5, 6, 7, 10, 14, 15)$$

Example 11

continue.....

$$f(A, B, C, D) = \prod M(0, 1, 3, 5, 6, 7, 10, 14, 15)$$



Simplified Boolean expression

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

Example 12

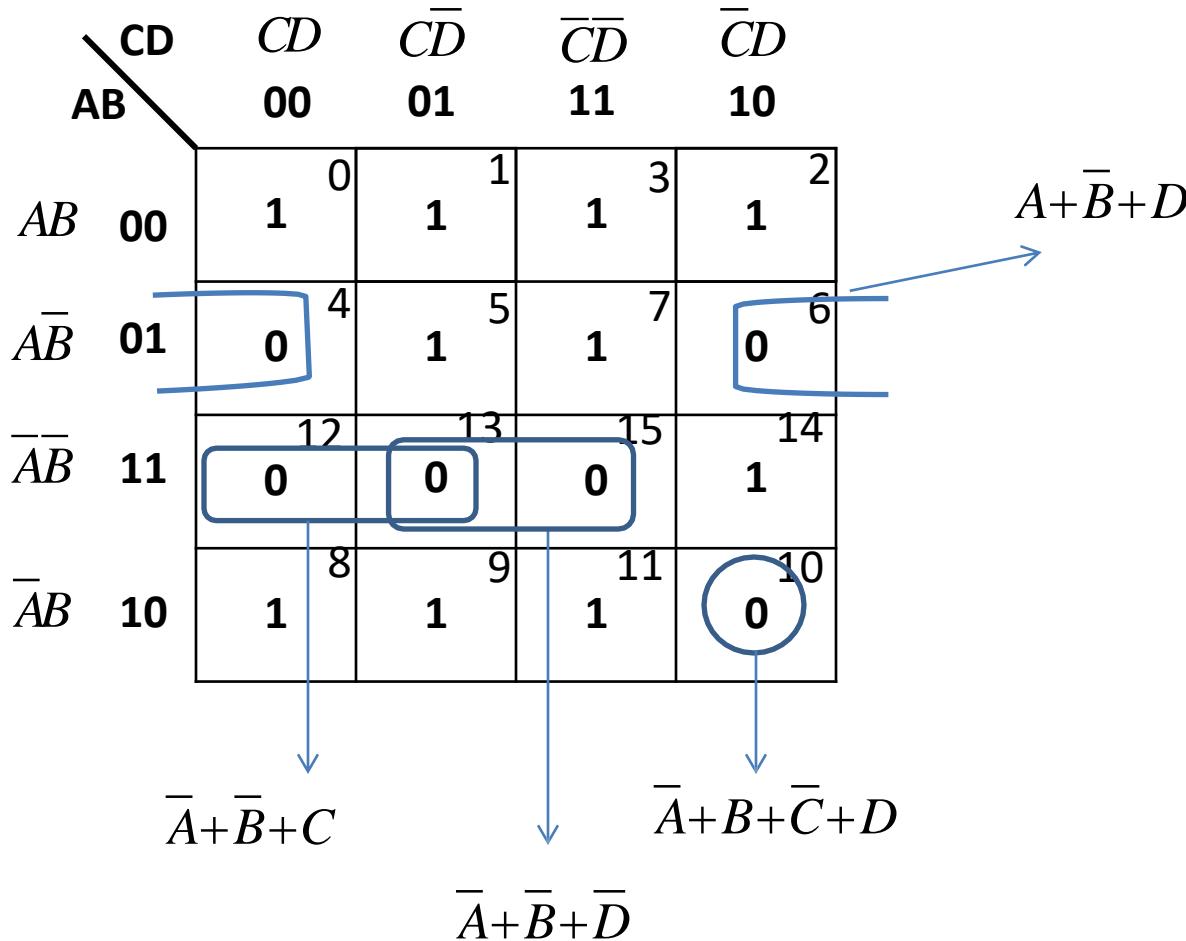
Simplify ;

$$f(A, B, C, D) = \prod M(4, 6, 10, 12, 13, 15)$$

Example 12

continue....

$$f(A, B, C, D) = \prod M(4, 6, 10, 12, 13, 15)$$



Simplified Boolean expression

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

K-map and don't care conditions

- ✓ For SOP form we enter 1's corresponding to the combinations of input variables which produce a high output and we enter 0's in the remaining cells of the K-map.
- ✓ For POS form we enter 0's corresponding to the combinations of input variables which produce a high output and we enter 1's in the remaining cells of the K-map.

K-map and don't care conditions

- ✓ But it is not always true that the cells not containing 1's (in SOP) will contain 0's, because some combinations of input variable do not occur.
- ✓ Also for some functions the outputs corresponding to certain combinations of input variables do not matter.

K-map and don't care conditions

- ✓ In such situations we have a freedom to assume a 0 or 1 as output for each of these combinations.
- ✓ These conditions are known as the “Don’t Care Conditions” and in the K-map it is represented as ‘X’, in the corresponding cell.
- ✓ The don’t care conditions may be assumed to be 0 or 1 as per the need for simplification

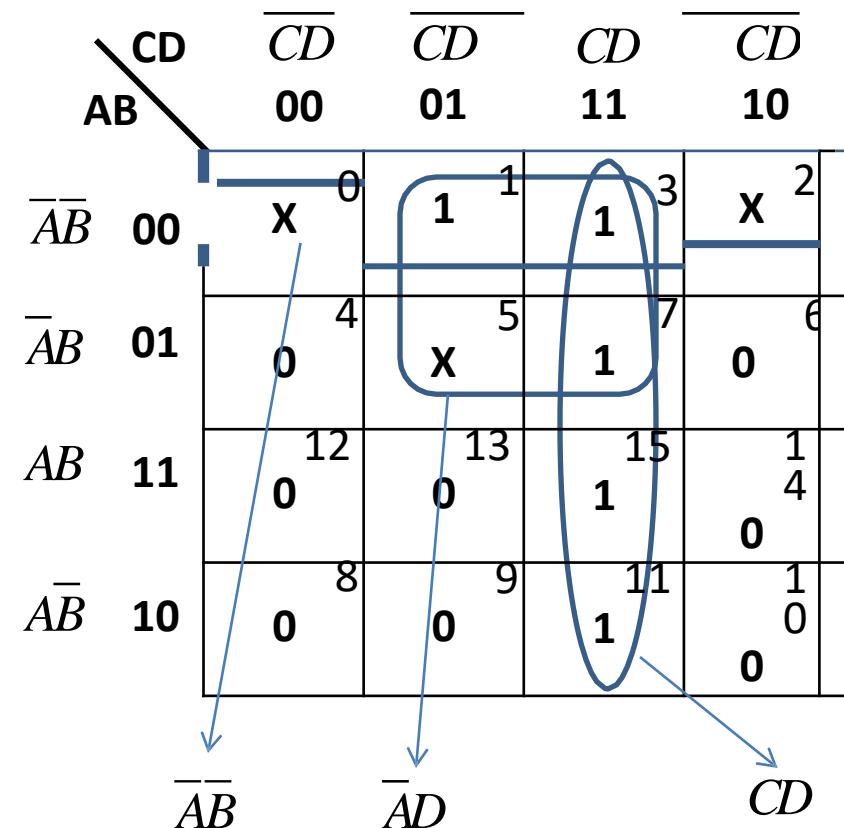
K-map and don't care conditions - Example

Simplify ;

$$f(A, B, C, D) = \sum m(1, 3, 7, 11, 15) + d(0, 2, 5)$$

K-map and don't care conditions - Example

$$f(A, B, C, D) = \sum m(1, 3, 7, 11, 15) + d(0, 2, 5)$$



Simplified Boolean expression

$$f = CD + \bar{A}\bar{B} + \bar{A}D$$



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



Switching Theory and Logic Design

ARI-209

By: Dr. Divya Agarwal



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT- II** 10 HOURS
- **Sequential Logic Circuits:** Latches and Flip Flops- SR, D, T and MS-JK Flip Flops, Asynchronous Inputs.
- **Counters and Shift Registers:** Design of Synchronous and Asynchronous Counters- Binary, BCD, Decade and Up/Down Counters, Shift Registers, Types of Shift Registers, Counters using Shift Registers- Ring Counter and Johnson Counter.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-III** **08 HOURS**
- **Integrated circuits:** TTL and CMOS logic families and their characteristics. Brief introduction to RAM and ROM
- **Synchronous Sequential Circuits:** State Tables State Equations and State Diagrams, State Reduction and State Assignment, Design of Clocked Sequential Circuits using State Equations.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-IV** 06 HOURS
- **Finite state machine:** capabilities and limitations, Mealy and Moore models-minimization of completely specified and incompletely specified sequential machines, Partition techniques and merger chart methods, concept of minimal cover table.
- **Algorithmic State Machine:** Representation of sequential circuits using ASM charts synthesis of output and next state functions, Data path control path partition-based design.



Text Books:

1. Leach and Malvino (2011). *Digital principles and Applications*. Tata McGraw-Hill Education
2. Mano, M. M. (2017). *Digital logic and computer design*. Pearson Education India.
3. Jain, R. P. (2003). *Modern digital electronics*. Tata McGraw-Hill Education.

Reference Books:

1. A Anand Kumar. (2016) *Fundamentals of Digital Logic Circuits*, PHI
2. Taub, H., & Schilling, D. L. (1977). *Digital integrated electronics*. McGraw-Hill College.



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.

K-Map Pros and Cons

- K-Map is systemic
- Require the ability to identify and visualize the prime implicants in order to cover all minterms
- But effective only up to 5-6 input variables!

Quine-McCluskey Algorithm

- Tabular Method
 - Compute all prime implicants
 - Find a minimum expression for Boolean functions
 - No visualization of prime implicants
 - Can be programmed and implemented in a computer
-

QM Method Example

$$F(W, X, Y, Z) = \sum m(0, 3, 5, 6, 7, 10, 12, 13) + \sum d(2, 9, 15)$$

- Step 1 : Divide all the minterms (and don't cares) of a function into groups

For
Minterms:

Minterm ID	W	X	Y	Z
0	0	0	0	0
3	0	0	1	1
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
10	1	0	1	0
12	1	1	0	0
13	1	1	0	1

For don't
cares:

Minterm ID	W	X	Y	Z
2	0	0	1	0
9	1	0	0	1
15	1	1	1	1

QM Method Example

- Step 1 : Divide all the minterms (and don't cares) of a function into groups

Groups	Minterm ID	W	X	Y	Z	Merge Mark
G0	0	0	0	0	0	
G1	2	0	0	1	0	
G2	3	0	0	1	1	
	5	0	1	0	1	
	6	0	1	1	0	
	9	1	0	0	1	
	10	1	0	1	0	
	12	1	1	0	0	
	7	0	1	1	1	
G3	13	1	1	0	1	
	15	1	1	1	1	

QM Method Example

- ❖ Step 2: Merge minterms from adjacent groups to form a new implicant table

Groups	Minterm ID	W	X	Y	Z	Merge Mark
G0	0	0	0	0	0	
G1	2	0	0	1	0	
G2	3	0	0	1	1	
	5	0	1	0	1	
	6	0	1	1	0	
	9	1	0	0	1	
	10	1	0	1	0	
	12	1	1	0	0	
	7	0	1	1	1	
G3	13	1	1	0	1	
G4	15	1	1	1	1	

Groups	Minterm ID	W	X	Y	Z
G0'	0, 2	0	0	d	0
G1'	2, 3	0	0	1	d
	2, 6	0	d	1	0
	2, 10	d	0	1	0
	3, 7	0	d	1	1
G2'	5, 7	0	1	d	1
	6, 7	0	1	1	d
	5, 13	d	1	0	1
	9, 13	1	d	0	1
	12, 13	1	1	0	d
	7, 15	d	1	1	1
G3'	13, 15	1	1	d	1

QM Method Example

- Step 3: Repeat step 2 until no more merging is possible

Groups	Minterm ID	W	X	Y	Z	Merge Mark
G0'	0, 2	0	0	d	0	
G1'	2, 3	0	0	1	d	
	2, 6	0	d	1	0	
	2, 10	d	0	1	0	
G2'	3, 7	0	d	1	1	
	5, 7	0	1	d	1	
	6, 7	0	1	1	d	
	5, 13	d	1	0	1	
	9, 13	1	d	0	1	
	12, 13	1	1	0	d	
G3'	7, 15	d	1	1	1	
	13, 15	1	1	d	1	

Groups	Minterm ID	W	X	Y	Z
G1''	2, 3, 6, 7	0	d	1	d
	2, 6, 3, 7	0	d	1	d
G2''	5, 7, 13, 15	d	1	d	1
	5, 7, 13, 15	d	1	d	1

QM Method Example

- Step 3: Repeat step 2 until no more merging is possible

Groups	Minterm ID	W	X	Y	Z	Merge Mark
G0"	0, 2	0	0	d	0	
G1"	2, 3, 6, 7	0	d	1	d	
	2, 10	d	0	1	0	
G2"	5, 7, 13, 15	d	1	d	1	
	9, 13	1	d	0	1	
	12, 13	1	1	0	d	

- No more merging possible!

QM Method Example

- Step 4: Put all prime implicants in a cover table (don't cares excluded)

Minterm ID	\overline{W}	\overline{X}	\overline{Z}	\overline{WY}	\overline{XYZ}	XZ	$WX\overline{Y}$	$W\overline{Y}Z$
0		1						
3				1				
5						1		
6				1				
7				1		1		
10					1			
12							1	
13						1	1	1

Need not include
don't cares

QM Method Example

- ❖ Step 5: Identify essential minterms, and hence essential prime implicants

Minterm ID	$\bar{W} \bar{X} \bar{Z}$	$\bar{W}Y$	$\bar{X}YZ$	XZ	$WX\bar{Y}$	$W\bar{Y}Z$
0	1					
3		1				
5				1		
6		1				
7		1				
10			1			
12					1	
13				1	1	1



E.M.T



E.PI

QM Method Example

- ❖ Step 6: Add prime implicants to the minimum expression of F until all minterms of F are covered

Minterm ID	$\bar{W} \bar{X} \bar{Z}$	$\bar{W}Y$	$\bar{X}YZ$	XZ	$W\bar{X}\bar{Y}$	$W\bar{Y}Z$
0	1					
3		1				
5				1		
6		1				
7						
10			1			
12					1	
13				1	1	1

Already cover
all minterms!



E.M.T



E.PI

QM Method Example

$$F(W, X, Y, Z) = \sum m(0, 3, 5, 6, 7, 10, 12, 13) + \sum d(2, 9, 15)$$

- So after simplification through QM method, a minimum expression for $F(W, X, Y, Z)$ is:

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

Finding Prime Implicants (PIs)

$$F(W,X,Y,Z) = \Sigma(5,7,9,11,13,15)$$

Step 1			Step 2			Step 3		
2	5							
	9							
	7							
3	11							
	13							
4	15							

List minterms by the number of **1s** it contains.

Finding Prime Implicants (PIs)

$$F(W,X,Y,Z) = \Sigma(5,7,9,11,13,15)$$

Step 1			Step 2			Step 3		
	5	0101						
	9	1001						
	7	0111						
	11	1011						
	13	1101						
	15	1111						

Finding Prime Implicants (PIs)

$$F(W,X,Y,Z) = \Sigma(5,7,9,11,13,15)$$

Step 1			Step 2			Step 3		
	5	0101		5,7				
	9	1001		5,13				
				9,11				
	7	0111		9,13				
	11	1011						
	13	1101		7,15				
				11,15				
	15	1111		13,15				

Enter combinations of minterms by the number of **1s** it contains.

Finding Prime Implicants (PIs)

$$F(W,X,Y,Z) = \Sigma(5,7,9,11,13,15)$$

Step 1			Step 2			Step 3		
☒	5	0101		5,7	01-1			
☒	9	1001		5,13	-101			
				9,11	10-1			
☒	7	0111		9,13	1-01			
☒	11	1011						
☒	13	1101		7,15	-111			
				11,15	1-11			
☒	15	1111		13,15	11-1			

Check off elements used from Step 1.

Finding Prime Implicants (PIs)

$$F(W,X,Y,Z) = \Sigma(5,7,9,11,13,15)$$

Step 1			Step 2			Step 3		
☒	5	0101		5,7	01-1		5,7,13,15	-1-1
☒	9	1001		5,13	-101		5,13,7,15	-1-1
				9,11	10-1		9,11,13,15	1--1
☒	7	0111		9,13	1-01		9,13,11,15	1--1
☒	11	1011						
☒	13	1101		7,15	-111			
				11,15	1-11			
☒	15	1111		13,15	11-1			

Enter combinations of minterms by the number of **1s** it contains.

Finding Prime Implicants (PIs)

$$F(W,X,Y,Z) = \Sigma(5,7,9,11,13,15)$$

Step 1			Step 2			Step 3		
☒	5	0101	☒	5,7	01-1		5,7,13,15	-1-1
☒	9	1001	☒	5,13	-101		5,13,7,15	-1-1
			☒	9,11	10-1		9,11,13,15	1--1
☒	7	0111	☒	9,13	1-01		9,13,11,15	1--1
☒	11	1011						
☒	13	1101	☒	7,15	-111			
			☒	11,15	1-11			
☒	15	1111	☒	13,15	11-1			

The entries left unchecked are Prime Implicants.

Finding Essential Prime Implicants (EPIs)

	Prime Implicants	Covered Minterms	<u>Minterms</u>					
			5	7	9	11	13	15
	- 1 - 1	5,7,13,15						
	1 -- 1	9,13,11,15						

Enter the Prime Implicants and their minterms.

Finding Essential Prime Implicants (EPIs)

	Prime Implicants	Covered Minterms	<u>Minterms</u>					
			5	7	9	11	13	15
	- 1 - 1	5,7,13,15	X	X			X	X
	1 -- 1	9,13,11,15			X	X	X	X

Enter Xs for the minterms covered.

Finding Essential Prime Implicants (EPIs)

	Prime Implicants	Covered Minterms	<u>Minterms</u>					
			5	7	9	11	13	15
	- 1 - 1	5,7,13,15	X	X			X	X
	1 -- 1	9,13,11,15			X	X	X	X

Circle Xs that are in a column singularly.

Finding Essential Prime Implicants (EPIs)

	Prime Implicants	Covered Minterms	<u>Minterms</u>					
			5	7	9	11	13	15
☒	- 1 - 1	5,7,13,15	X	X			X	X
☒	1 -- 1	9,13,11,15			X	X	X	X

The circled Xs are the Essential Prime Implicants,
so we check them off.

Finding Essential Prime Implicants (EPIs)

	Prime Implicants	Covered Minterms	<u>Minterms</u>					
			5	7	9	11	13	15
☒	- 1 - 1	5,7,13,15	☒	☒			☒	☒
☒	1 -- 1	9,13,11,15			☒	☒	☒	☒
			☒	☒	☒	☒	☒	☒

We check off the minterms covered by each of the EPIs.

Finding Essential Prime Implicants (EPIs)

	Prime Implicants	Covered Minterms	<u>Minterms</u>					
			5	7	9	11	13	15
☒	- 1 - 1	5,7,13,15	☒	☒			☒	☒
☒	1 -- 1	9,13,11,15			☒	☒	☒	☒
			☒	☒	☒	☒	☒	☒

EPIs:

W	X	Y	Z
-	1	-	1
1	-	-	1

$$\begin{aligned}
 F &= (X \cdot Z) + (W \cdot Z) \\
 &= (X + W) \cdot Z
 \end{aligned}$$

Finding Prime Implicants (PIs)

$$F(W,X,Y,Z) = \Sigma(2,3,6,7,8,10,11,12,14,15)$$

Finding Prime Implicants (PIs)

$$F(W,X,Y,Z) = \Sigma(2,3,6,7,8,10,11,12,14,15)$$

Step 1			Step 2			Step 3			Step 4		
☒	2	0010		2,3	001-						
☒	8	1000		2,6	0-10						
				2,10	-010						
☒	3	0011		8,10	10-0						
☒	6	0110		8,12	1-00						
☒	10	1010									
☒	12	1100		3,7	0-11						
				3,11	-011						
☒	7	0111		6,7	011-						
☒	11	1011		6,14	-110						
☒	14	1110		10,14	1-10						
				10,11	101-						
☒	15	1111		12,14	11-0						
				7,15	-111						
				11,15	1-11						
				14,15	111-						

Finding Prime Implicants (PIs)

$$F(W,X,Y,Z) = \Sigma(2,3,6,7,8,10,11,12,14,15)$$

Step 1			Step 2			Step 3			Step 4		
☒	2	0010	☒	2,3	001-		2,3,6,7	0-1-			
☒	8	1000	☒	2,6	0-10		2,6,3,7	0-1-			
			☒	2,10	-010		2,3,10,11	-01-			
☒	3	0011	☒	8,10	10-0		2,6,10,14	-- 10			
☒	6	0110	☒	8,12	1-00		2,10,3,11	-01-			
☒	10	1010					2,10,6,14	-- 10			
☒	12	1100	☒	3,7	0-11		8,10,12,14	1-- 0			
			☒	3,11	-011		8,12,10,14	1-- 0			
☒	7	0111	☒	6,7	011-						
☒	11	1011	☒	6,14	-110		3,7,11,15	-- 11			
☒	14	1110	☒	10,14	1-10		3,11,7,15	-- 11			
			☒	10,11	101-		6,7,14,15	-11-			
☒	15	1111	☒	12,14	11-0		6,14,7,15	-11-			
							10,14,11,15	1- 1 -			
			☒	7,15	-111		10,11,14,15	1- 1 -			
			☒	11,15	1-11						
			☒	14,15	111-						

Finding Prime Implicants (PIs)

$$F(W,X,Y,Z) = \Sigma(2,3,6,7,8,10,11,12,14,15)$$

Step 1			Step 2			Step 3			Step 4		
☒	2	0010	☒	2,3	001-	☒	2,3,6,7	0-1-		2,3,6,7,10,14,11,15	-- 1 -
☒	8	1000	☒	2,6	0-10	☒	2,6,3,7	0-1-		2,3,10,11,6,14,7,15	-- 1 -
			☒	2,10	-010	☒	2,3,10,11	-01-		2,6,3,7,10,11,14,15	-- 1 -
☒	3	0011	☒	8,10	10-0	☒	2,6,10,14	-- 10		2,6,10,14,3,7,11,15	-- 1 -
☒	6	0110	☒	8,12	1-00	☒	2,10,3,11	-01-		2,10,3,11,6,7,14,15	-- 1 -
☒	10	1010				☒	2,10,6,14	-- 10		2,10,6,14,3,11,7,15	-- 1 -
☒	12	1100	☒	3,7	0-11		8,10,12,14	1-- 0			
			☒	3,11	-011		8,12,10,14	1-- 0			
☒	7	0111	☒	6,7	011-						
☒	11	1011	☒	6,14	-110	☒	3,7,11,15	-- 11			
☒	14	1110	☒	10,14	1-10	☒	3,11,7,15	-- 11			
			☒	10,11	101-	☒	6,7,14,15	-11-			
☒	15	1111	☒	12,14	11-0	☒	6,14,7,15	-11-			
						☒	10,14,11,15	1- 1 -			
			☒	7,15	-111	☒	10,11,14,15	1- 1 -			
			☒	11,15	1-11						
			☒	14,15	111-						

Finding Essential Prime Implicants (EPIs)

	Prime Implicants	Covered Minterms	Minterms									
			2	3	6	7	8	10	11	12	14	15
	☒ 1--0	8,12,10,14					X	X		X	X	
	☒ --1-	2,3,6,7,10,11,14,15	X	X	X	X		X	X		X	X
			☒		☒		☒		☒		☒	
				☒		☒		☒		☒		

Finding Essential Prime Implicants (EPIs)

	Prime Implicants	Covered Minterms	Minterms									
			2	3	6	7	8	10	11	12	14	15
	\boxtimes 1--0	8,12,10,14					X	X		X	X	
	\boxtimes --1-	2,3,6,7,10,11,14,15	X	X	X	X		X	X		X	X
			\boxtimes		\boxtimes		\boxtimes		\boxtimes		\boxtimes	
				\boxtimes		\boxtimes		\boxtimes		\boxtimes		

EPIs:

W	X	Y	Z
1	-	-	0
-	-	1	-

$$F = (W \cdot Z') + Y$$

Example

$$F(A, B, C, D) = \sum m(4,5,6,8,10,13) + \sum d(0,7,15)$$

Implication Table (1,0,-)

- Quine-McCluskey Method

- Tabular method to systematically find all prime implicants
- $f(A,B,C,D) = \sum m(4,5,6,8,9,10,13) + \sum d(0,7,15)$
- Part 1: Find all prime implicants
- Step 1: Fill Column 1 with active-set and DC-set minterm indices. Group by number of true variables (# of 1's).

NOTE: DCs are included in this step!

Implication Table		
Column I		
0000		
0100		
1000		
0101		
0110		
1001		
1010		
0111		
1101		
1111		

Minimization - First Pass (1,0,-)

- Quine-McCluskey Method

- Tabular method to systematically find all prime implicants
- $f(A,B,C,D) = \sum m(4,5,6,8,9,10,13) + \sum d(0,7,15)$
- Part 1: Find all prime implicants
- Step 2: Apply Adjacency - Compare elements of group with N 1's against those with N+1 1's. One bit difference implies adjacent. Eliminate variable and place in next column.

E.g., 0000 vs. 0100 yields 0-00

0000 vs. 1000 yields -000

When used in a combination, mark with a check. If cannot be combined, mark with a star. These are the prime implicants.

Repeat until nothing left.

Implication Table		
Column I	Column II	
0000 ✓	0-00	
	-000	
0100 ✓		
1000 ✓	010-	
	01-0	
0101 ✓	100-	
0110 ✓	10-0	
1001 ✓		
1010 ✓	01-1	
	-101	
0111 ✓	011-	
1101 ✓	1-01	
1111 ✓	-111	
	11-1	

Minimization - Second Pass (1,0,-)

- Quine-McCluskey Method

- Step 2 cont.: Apply Adjacency - Compare elements of group with N 1's against those with N+1 1's. One bit difference implies adjacent. Eliminate variable and place in next column.

E.g., 0000 vs. 0100 yields 0-00

0000 vs. 1000 yields -000

When used in a combination, mark with a check. If cannot be combined, mark with a star. These are the prime implicants.

Repeat until nothing left.

Implication Table		
Column I	Column II	Column III
0000 ✓	0-00 *	01-- *
	-000 *	
0100 ✓		-1-1 *
1000 ✓	010- ✓	
	01-0 ✓	
0101 ✓	100- *	
0110 ✓	10-0 *	
1001 ✓		
1010 ✓	01-1 ✓	
	-101 ✓	
0111 ✓	011- ✓	
1101 ✓	1-01 *	
1111 ✓	-111 ✓	
	11-1 ✓	

Prime Implicants

Prime Implicants:

$$0\text{-}00 = \overline{A}\ \overline{C}\ \overline{D} \quad -000 = \overline{B}\ \overline{C}\ \overline{D}$$

$$100\text{-} = A\ \overline{B}\ \overline{C} \quad 10\text{-}0 = A\ \overline{B}\ \overline{D}$$

$$1\text{-}01 = A\ \overline{C}\ D \quad -1\text{-}1 = B\ D$$

$$01\text{--} = \overline{A}\ B$$

Stage 2: find smallest set of prime implicants that cover the active-set

recall that essential prime implicants must be in final expression

Coverage Table

Coverage Chart

	4	5	6	8	9	10	13
0,4(0-00)	X						
0,8(-000)				X			
8,9(100-)				X	X		
8,10(10-0)				X		X	
9,13(1-01)					X		X
4,5,6,7(01--)	X	X	X				
5,7,13,15(-1-1)			X			X	

Note: Don't include DCs in coverage table; they don't have covered by the final logic expression!

rows = prime implicants
columns = ON-set elements
place an "X" if ON-set element is covered by the prime implicant

Coverage Table (cont.)

Coverage Chart

	4	5	6	8	9	10	13
0,4(0-00)	X						
0,8(-000)				X			
8,9(100-)				X	X		
8,10(10-0)				X	X		
9,13(1-01)				X	X		
4,5,6,7(01--)	X	X	X				
5,7,13,15(-1-1)		X			X		

	4	5	6	8	9	10	13
0,4(0-00)	X						
0,8(-000)				X			
8,9(100-)				X	X		
8,10(10-0)				X	X	X	X
9,13(1-01)						X	X
4,5,6,7(01--)	X	X	X			X	X
5,7,13,15(-1-1)				X			X

rows = prime implicants
columns = ON-set elements
place an "X" if ON-set element is covered by the prime implicant

If column has a single X, than the implicant associated with the row is essential. It must appear in minimum cover

Coverage Table (cont.)

	4	5	6	8	9	10	13
0,4(0-00)	X						
0,8(-000)				X			
8,9(100-)				X	X		
8,10(10-0)				*		X	
9,13(1-01)					X		X
4,5,6,7(01--)	*	*		X			
5,7,13,15(-1-1)		X				X	

**Eliminate all columns covered by
essential primes**

Coverage Table (cont.)

	4	5	6	8	9	10	13
0,4(0-00)	X						
0,8(\000)				X			
8,9(100-)				X	X		
8,10(10-0)				*		X	
9,13(1-01)					X	X	
4,5,6,7(01--)	*	*	X				X
5,7,13,15(-1-1)			X				

	4	5	6	8	9	10	13
0,4(0-00)	X						
0,8(\000)				X			
8,9(100-)				X	X		
8,10(10-0)				*		X	
9,13(1-01)					*		X
4,5,6,7(01--)	*	*	X			X	
5,7,13,15(-1-1)			X				X

Eliminate all columns covered by essential primes

Find minimum set of rows that cover the remaining columns

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

Example

$$\text{Ex) } f(x_1, x_2, x_3, x_4) = \Sigma(0, 1, 2, 5, 6, 7, 8, 9, 10, 13, 15)$$

$$\text{Ex)} f(x_1, x_2, x_3, x_4) = \Sigma(0, 1, 2, 5, 6, 7, 8, 9, 10, 13, 15)$$

#	X ₁ , X ₂ , X ₃ , X ₄		X ₁ , X ₂ , X ₃ , X ₄		X ₁ , X ₂ , X ₃ , X ₄		X ₁ , X ₂ , X ₃ , X ₄		
0	0 0 0 0	✓	0 0 0 -	✓	(0,1)	- 0 0 -	(0,1,8,9)		
1	0 0 0 1	✓	0 0 - 0	✓	(0,2)	- 0 - 0	(0,2,8,10)		
2	0 0 1 0	✓	- 0 0 0	✓	(0,8)	- - 0 1	(1,5,9,13)		
8	1 0 0 0	✓	0 - 0 1	✓	(1,5)	- 1 - 1	(5,7,13,15)		
5	0 1 0 1	✓	- 0 0 1	✓	(1,9)				
6	0 1 1 0	✓	0 - 1 0		(2,6)				
9	1 0 0 1	✓	- 0 1 0	✓	(2,10)				
10	1 0 1 0	✓	1 0 0 -	✓	(8,9)				
7	0 1 1 1	✓	1 0 - 0	✓	(8,10)				
13	1 1 0 1	✓	0 1 - 1	✓	(5,7)				
15	1 1 1 1	✓	- 1 0 1	✓	(5,13)				
			0 1 1 -		(6,7)				
			1 - 0 1	✓	(9,13)				
			- 1 1 1	✓	(7,15)				
			1 1 - 1	✓	(13,15)				

Using prime implicant chart, we can find essential PI

	0	1	2	5	6	7	8	9	10	13	15
(2,6)			✓		✓	✓					
(6,7)				✓	✓						
(0,1,8,9)	✓	✓									
(0,2,8,10)	✓		✓				✓	✓		✓	
(1,5,9,13)		✓		✓				✓			✓
(5,7,13,15)			✓			✓			✓	✓	✓

The reduced PI chart

	1	6	9
(2,6)	✓		
(6,7)		✓	
(0,1,8,9)	✓		✓
(1,5,9,13)	✓		✓

The essential PI's are $(0,2,8,10)$ and $(5,7,13,15)$. So,
 $f(x_1, x_2, x_3, x_4) = (0,2,7,8) + (5,7,13,15) + \text{PI's}$

Here are 4 different choices $(2,6) + (0,1,8,9)$, $(2,6) + (1,5,9,13)$
 $(6,7) + (0,1,8,9)$, or $(6,7) + (1,5,9,13)$

A PI p_j dominates PI p_k iff every minterm covered by p_k is also covered by p_j .

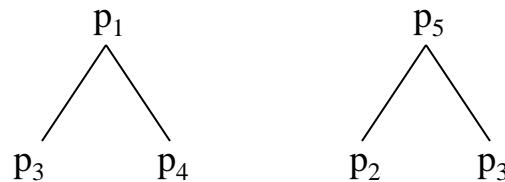
	m_1	m_2	m_3	m_4
p_j	✓	✓	✓	
p_k	✓	✓		

(can remove)

Branching method

	m_1	m_2	m_3	m_4	m_5
p_1	✓	✓			
p_2		✓	✓		
p_3			✓	✓	
p_4				✓	✓
p_5	✓			✓	

If we choose p_1 first, then p_3, p_5 are next.



Quine – McCluskey method (no limitation of the # of variables)

Quine-McCluskey example

- $F(A,B,C,D) = \Sigma (3,9,11,12,13,14,15) + \Sigma_d (1,4,6)$

$$\text{Ex) } f(A,B,C,D) = \Sigma(3,9,11,12,13,14,15) + \Sigma_d(1,4,6)$$

#	A,B,C,D		A,B,C,D		A,B,C,D		A,B,C,D
1	0 0 0 1	✓	0 0 - 1	(1,3)	✓	- 0 - 1	(1,3,9,11)
4	0 1 0 0	✓	- 0 0 1	(1,9)	✓	- 1 - 0	(4,6,12,14)
3	0 0 1 1	✓	0 1 - 0	(4,6)	✓	1 -- 1	(9,13,11,15)
6	0 1 1 0	✓	- 1 0 0	(4,12)	✓	1 1 --	(12,13,14,15)
9	1 0 0 1	✓	- 0 1 1	(3,11)	✓		
12	1 1 0 0	✓	- 1 1 0	(6,14)	✓		
11	1 0 1 1	✓	1 0 - 1	(9,11)	✓		
13	1 1 0 1	✓	1 - 0 1	(9,13)	✓		
14	1 1 1 0	✓	1 1 0 -	(12,13)	✓		
15	1 1 1 1	✓	1 1 - 0	(12,14)	✓		
			1 - 1 1	(11,15)	✓		
			1 1 - 1	(13,15)	✓		
			1 1 1 -	(14,15)	✓		

PI chart:

	3	9	11	12	13	14	15
(1,3,9,11)	✓	✓	✓				
(4,6,12,14)				✓	✓		
(9,13,11,15)		✓	✓		✓	✓	✓
(12,13,14,15)				✓	✓	✓	✓

Reduced PI chart:

	12	13	14	15
(4,6,12,14)	✓	✓		
(9,13,11,15)		✓		✓
(12,13,14,15)	✓	✓	✓	✓

Result: (1,3,9,11) + (12,13,14,15)

Example Problem

Evaluate :

$$\Sigma (0,4,5,7,10,12,13,14,15)$$

Example Problem

Evaluate :

$$\Sigma (0,4,5,7,10,12,13,14,15)$$

Step-1: Listing the Binary codes of each number.

No	Binary
0	0000
4	0100
5	0101
7	0111
10	1010
12	1100
13	1101
14	1110
15	1111

Example Problem

Step-2: Listing Binary Numbers according to their number of 1's as in Table.2

Step-3: Making Table of Duals from Table:2

Step:2

	ABCD	
0	0 0 0 0	X
4	0 1 0 0	X
5	0 1 0 1	X
10	1 0 1 0	X
12	1 1 0 0	X
7	0 1 1 1	X
13	1 1 0 1	X
14	1 1 1 0	X
15	1 1 1 1	X

Step:3

	ABCD	
0.4	0 - 0 0	
4.5	0 1 0 -	
4.12	- 1 0 0	
5.7	0 1 - 0	
5.13	- 1 0 1	
10.14	1 - 1 0	
12.13	1 1 0 -	
12.14	1 1 - 0	
7.15	- 1 1 1	
13.15	1 1 - 1	
14.15	1 1 1 -	

Example Problem

Step-4: Generating table of Quads from Table.3

	ABCD			
0.4	0 - 0 0		P I	
4.5	0 1 0 -		X	
4.12	- 1 0 0		X	
5.7	0 1 - 0		X	
5.13	- 1 0 1		X	
10.14	1 - 1 0		P	
12.13	1 1 0 -		I	
12.14	1 1 - 0		X	
7.15	- 1 1 1		X	
13.15	1 1 - 1		X	
14.15	1 1 1 -		X	

	ABCD			
4.5-12.13	- 1 0 -	{		P I
4.12-5.13	- 1 0 -			
5.13-7.15				
5.7-13.15	-1- 1 -	{		P I
	-1- 1 -			
12.13-14.15	1 1 - -	{		P I
12.14-13.15	1 1 - -			

Step-5: Making table of Prime Implicants

?

Prime Implicants	0	4	5	7	10	12	13	14	15
0.4	✓	✓							
10.14					✓			✓	
4.5-12.13	✓	✓				✓	✓		
5.13-7.15		✓		✓			✓		✓
12.13-14.15						✓	✓	✓	✓

$0.4 + 10.14 + 5.13-7.15 + 4.5-12.13$

¶ Step-6: Generating SOP from Prime Implicants

$$0.4 + 10.14 + 5.13-7.15 + 4.5-12.13$$

Now Since

PI	ABCD
0.4	0 - 0 0
10.14	1 - 1 0
5.13-7.15	- 1 - 1
4.5-12.13	- 1 0 -

So the minimized SOP is: $A'C'D' + ACD' + BD + BC'$

Example Problem

Evaluate :

$$f(A,B,C,D) = \sum_m(1,2,5,6,7,9,10) + \sum_d(0,13,15)$$

Implication Table (1,0,-)

✓ Quine-McCluskey Method

- Tabular method to systematically find all prime implicants

$$f(A,B,C,D) = \sum_m(1,2,5,6,7,9,10) + \sum_d(0,13,15)$$

- Part 1: Find all prime implicants
- Step 1: Fill Column 1 with **onset** and **DC-set minterm indices**. Group by number of true variables (# of 1's).

NOTE THAT DCs ARE INCLUDED IN THIS STEP!

Implication Table	
Column I	
0000	
0001	
0010	
0101	
0110	
1001	
1010	
0111	
1101	
1111	

Minimization - First Pass (1,0,-)

✓ Quine-McCluskey Method

- Tabular method to systematically find all prime implicants
- $f(A,B,C,D) = \sum_m(1,2,5,6,7,9,10) + \sum_d(0,13,15)$
- Part 1: Find all prime implicants
- Step 2: Apply Adjacency - Compare elements of group with N 1's against those with N+1 1's. One bit difference implies adjacent. Eliminate variable and place in next column.

E.g., 0000 vs. 0100 yields 0-00

0000 vs. 1000 yields -000

When used in a combination, mark with a check. If cannot be combined, mark with a star. These are the prime implicants.

Repeat until nothing left.

Implication Table			
Column I		Column II	
0000	0	000-	0,1
		00-0	0,2
0001	1		
0010	2	0-01	1,5
		-001	1,9
0101	5	0-10	2,6
0110	6	-010	2,10
1001	9		
1010	10	01-1	5,7
		-101	5,13
0111	7	011-	6,7
1101	13	1-01	9,13
1111	15	-111	7,15
		11-1	13,15

Minimization - Second Pass (1,0,-)

✓ Quine-McCluskey Method

- Step 2 cont.: Apply Adjacency - Compare elements of group with N 1's against those with N+1 1's. One bit difference implies adjacent. Eliminate variable and place in next column.

E.g., 0000 vs. 0100 yields 0-00

00-0 vs. 10-0 yields -0-0

- When used in a combination, mark with a check ✓.

- If cannot be combined, mark with a star ✕. **THESE ARE THE PRIME IMPLICANTS.**

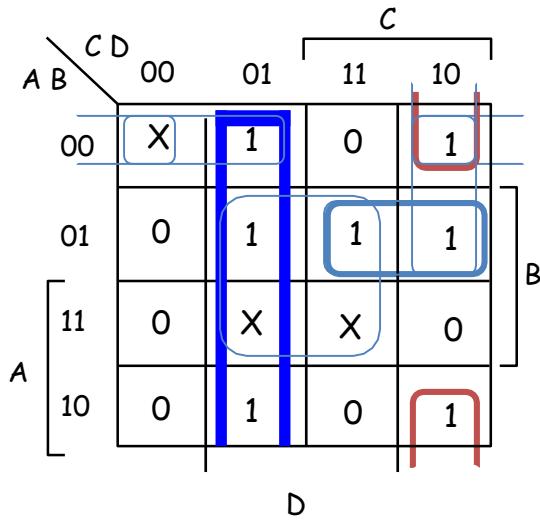
Repeat until nothing left.

- ✓ The set of ✕ constitutes the Complete Sum Σ_c

Implication Table		
Column I	Column II	Column III
0000 ✓ 0	000- 0,1	--01 1,5,9,13
	00-0 0,2	- 1-1 5,7,13,15
0001 ✓ 1		
0010 ✓ 2	0-01 1,5	
	-001 1,9	
0101 ✓ 5	0-10 2,6	
0110 ✓ 6	-010 2,10	
1001 ✓ 9		
1010 ✓ 10	01-1 5,7	
	-101 5,13	
0111 ✓ 7	011- 6,7	
1101 ✓ 13	1-01 9,13	
1111 ✓ 15	-111 7,15	
	11-1 13,15	

Prime Implicants

$$f(A,B,C,D) = \sum_m (1,2,5,6,7,9,10) + \sum_d (0,13,15)$$



Prime Implicants:

$$000 - = \overline{A} \overline{B} \overline{C}$$

$$0 - 10 = \overline{A} C \overline{D}$$

$$011 - = \overline{A} B C$$

$$-- 01 = \overline{C} D$$

$$00 - 0 = \overline{A} \overline{B} \overline{D}$$

$$-010 = \overline{B} C \overline{D}$$

$$-1 - 1 = B D$$

Stage 2: find smallest set of prime implicants that cover the active-set
Note that essential prime implicants must be in the final expression

Coverage Table

rows = prime implicants

columns = ON-set elements (minterms)

place an "X" if ON-set element is covered by the prime implicant

NOTE: DON'T INCLUDE DCs IN COVERAGE TABLE; THEY DON'T HAVE TO BE MANDATORY COVERED

Coverage Chart
minterms

		1	2	5	6	7	9	10
primes	0,1	000-	X					
	0,2	00-0		X				
	2,6	0-10		X		X		
	2,10	-010		X				(X)
	6,7	011-			X	X		
	1,5,9,13	--01	X		X		(X)	
	5,7,13,15	-1-1			X	X		

Row and Column Dominance

- ✓ **Definition:** Given two rows i_1 and i_2 , a row i_1 is said to **dominate** i_2 if it has checks in all columns in which i_2 has checks, i.e. it is a superset of i_2

Example:

i_1	$\times \times$	\times	$\times \times$	\times
i_2	$\times \times$		$\times \times$	

i_1 dominates i_2

- ✓ We can remove row i_2 , because we would never choose i_2 in a minimum cover since it can always be replaced by i_1 (i_2 is anymore a prime implicant).

DOMINATED ROWS (IMPLICANTS) CAN BE ELIMINATED

Row and Column Dominance

- ✓ Definition: Given two columns j_1 and j_2 , if the set of primes of column j_2 is contained in the set of primes of column j_1

Example:

	j_1	j_2
	✗	
		✗
	✗	
		✗
	✗	
		✗

j_2 dominates j_1

- ✓ We can remove column j_1 since we have to choose a prime to cover j_2 , any such prime also covers j_1 , that would result covered as well.

DOMINATED COLUMNS (MINTERMS) CAN BE ELIMINATED

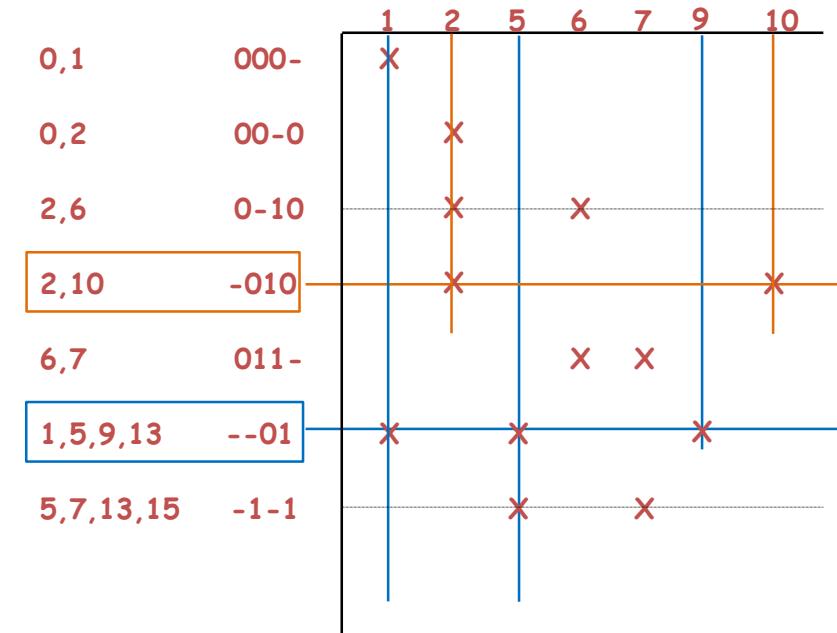
Pruning the Covering Table

1. Remove all rows covered by **essential primes** (columns in row **singletons**). Put these primes in the cover G .
 2. Group identical rows together and remove dominated rows.
 3. Remove dominating columns. For equal columns, keep just one to represent them.
 4. Newly formed row singletons define **n-ary essential primes**.
 5. Go to 1 if covering table decreased.
-
- ✓ The algorithm may terminate successfully with a set of primes and an empty table.
 - ✓ In case it terminates with a non empty table, the resulting reduced covering table is called the **cyclic core**. This has to be solved. A minimum solution for the cyclic core must be added to the resulting G .

Coverage Table (cont.)

Coverage Chart

	1	2	5	6	7	9	10
0,1	000-	x					
0,2	00-0		x				
2,6	0-10	x		x			
2,10	-010	x				x	
6,7	011-		x	x			
1,5,9,13	--01	x	x		x		
5,7,13,15	-1-1		x	x			



If column has a single x, than the implicant associated with the row is **essential**. It must appear in the minimum cover

Coverage Table (cont.)

		1	2	5	6	7	9	10
0,1	000-	X						
0,2	00-0		X					
2,6	0-10			X				
2,10	-010		X					
6,7	011-			X	X			
1,5,9,13	--01	X	X					
5,7,13,15	-1-1		X		X			

Eliminate all columns covered by essential primes

		1	2	5	6	7	9	10
0,1	000-	X						
0,2	00-0		X					
2,6	0-10			X				
2,10	-010				X			
6,7	011-					X	X	
1,5,9,13	--01					X		
5,7,13,15	-1-1						X	

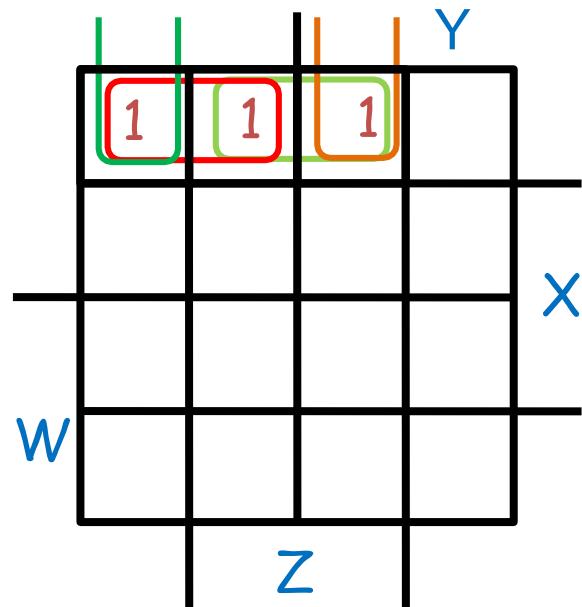
Find minimum set of rows that cover the remaining columns

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

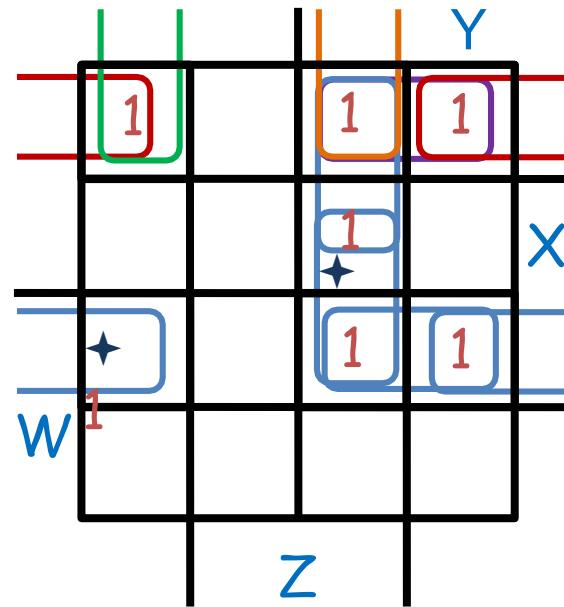
Quine Mc Cluskey: Cyclic Core example

$$F = \sum_m (0, 1, 3, 16, 18, 19, 23, 28, 30, 31)$$

$V = 0$



$V = 1$



$$F = v'w'y'x' + v'w'x'z + w'x'y'z' + w'x'yz + vw'x'z' + vw'x'y + vwxz' + vwxy + vw'yz + vxxyz$$

A B C D E F G H I J

Implication Table (1,0,-)

✓ Quine-McCluskey Method

- Tabular method to systematically find all prime implicants
- $f(v,w,x,y,z) = \sum m(0,1,3,16,18,19,23,28,30,31)$
- Part 1: Find all prime implicants
- Step 1: Fill Column 1 with active-set and DC-set minterm indices. Group by number of true variables (# of 1's).

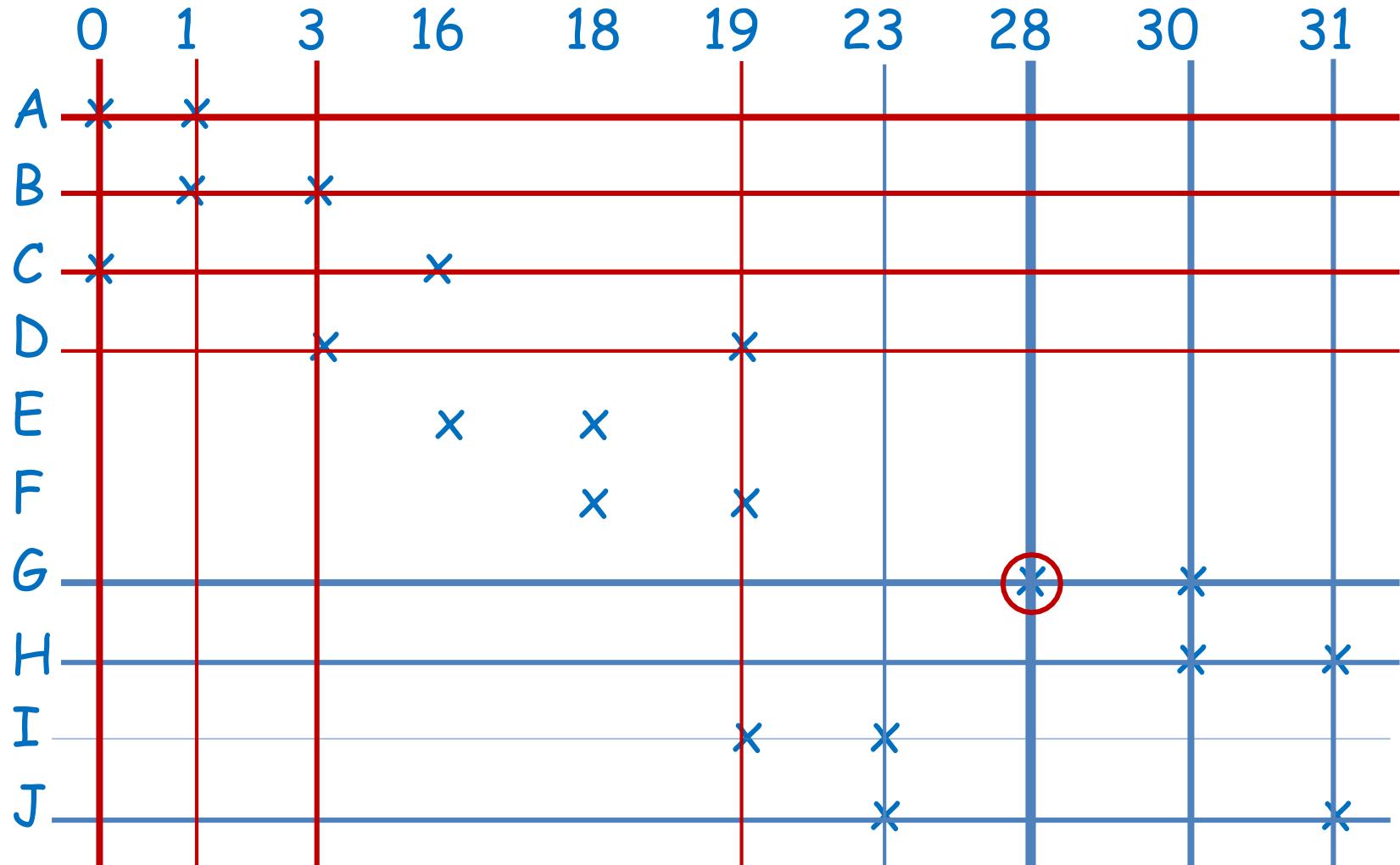
Implication Table			
	Column I	Column II	
0	00000	0000-	A: 0 1
1	00001	-0000	C: 0 16
16	1 0000	000-1	B: 1 3
3	00 011	100-0	E: 16 18
18	1001 0	-0011	D: 3 19
19	100 11	1001-	F: 18 19
28	1 1 100	10-11	I: 19 23
23	10 111	111-0	G: 28 30
30	111 10	1-111	J: 23 31
31	1111 1	1111-	H: 30 31

$$F = v'w'y'x' + v'w'x'z + w'x'y'z' + w'x'yz + vw'x'z' + vw'x'y + vwxz' + vwxy + vw'yz + vxzy$$

A B C D E F G H I J

Quine Mc Cluskey

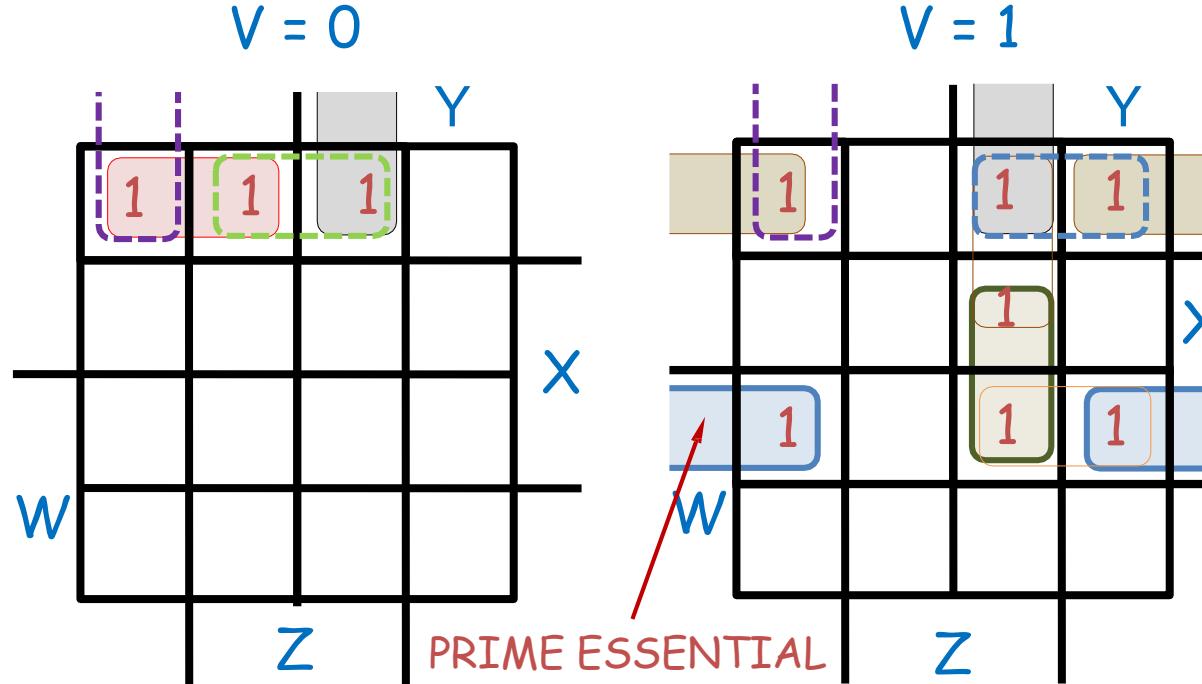
$$F = \sum_5(0, 1, 3, 16, 18, 19, 23, 28, 30, 31)$$



$G + J + A + D + E;$

Quine Mc Cluskey: Cyclic Core example

$$F = \sum_m(0, 1, 3, 16, 18, 19, 23, 28, 30, 31)$$



$$F = v'w'y'x' + \underline{v'w'x'z} + \underline{w'x'y'z'} + w'x'yz + vw'x'z' + \underline{vw'x'y} + \underline{vwxz'} + \underline{vwxy} + \underline{vw'yz} + \underline{vxzy}$$

A B C D E F G H I J

G+J+A+D+E;

Generating Primes - multiple outputs

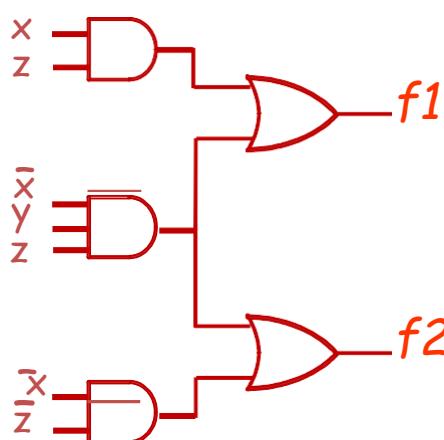
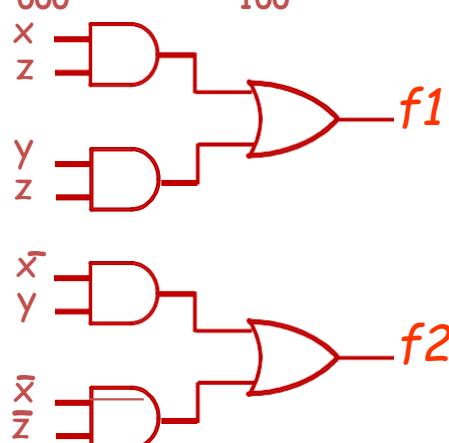
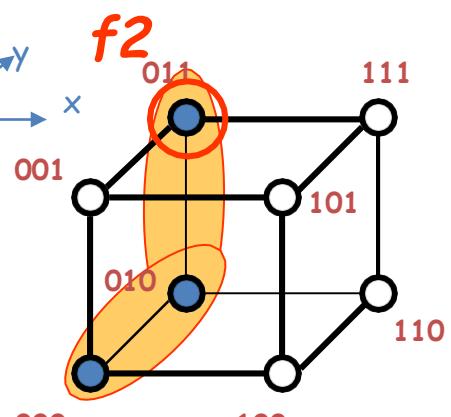
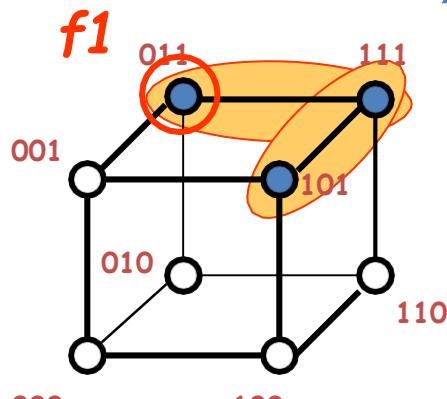
Example: $f_1(x, y, z) = \sum m(3, 5, 7)$, $f_2(x, y, z) = \sum m(0, 2, 3)$

x	y	z	
0	1	1	-11
1	0	1	1-1
1	1	1	111

$$f_1 = yz + zx$$

x	y	z	
0	0	0	-0
0	1	0	1-
0	1	1	11

$$f_2 = x\bar{z} + x\bar{y}$$



The idea is that we can share terms:

- using separate optimizations 6 gates and $G=12$
- sharing a term 5 gates and $G=11$.

$$f_1 f_2 = (yz + zx)(x\bar{z} + x\bar{y})$$

$$f_1 f_2 = x\bar{y}z$$

Generating Primes - multiple outputs

- ✓ Theorem: if p_1 is a prime implicant for f_1 , and p_2 is a prime implicant for f_2 , then if $p_1 \cdot p_2 \neq 0$, $p_1 \cdot p_2$ is a prime implicant of $f_1 \cdot f_2$
- ✓ Theorem: if p_3 is a prime implicant for $f_1 \cdot f_2$, then there exist p_1 for f_1 , and p_2 for f_2 , such that $p_3 = p_1 \cdot p_2$
- ✓ We can conclude that all prime implicants of $f_1 \cdot f_2$ are minimal sharable products for f_1 and f_2 : and that all prime implicants for $f_1 \cdot f_2$ are created by products of prime implicants for f_1 and f_2
- ✓ The way to use this is to make the prime implicants of $f_1 \cdot f_2$ available to the minimizations of f_1 and f_2 by extending the table concept

Generating Primes - multiple outputs

- ✓ Procedure similar to single-output function,
except: include also the primes of the products
of individual functions

	f_1 minterms	f_2 minterms
Rows for f_1 prime implicants: mark only f_1 columns		
Rows for f_2 prime implicants: mark only f_2 columns		
Rows for f_1f_2 prime implicants: mark both f_1 and f_2 columns		

Minimize multiple-output cover

✓ Example, cont.

$$f_1 \quad m_3 = 011 \quad p_1 = yz \\ m_5 = 101 \quad p_2 = xz \\ m_7 = 111$$

$$f_2 \quad m_0 = 000 \quad p_3 = \bar{x}y \\ m_2 = 010 \quad p_4 = \bar{x}\bar{z} \\ m_3 = 011$$

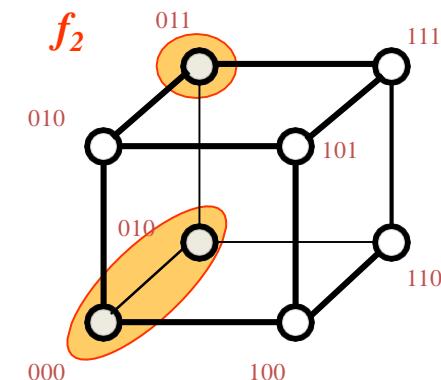
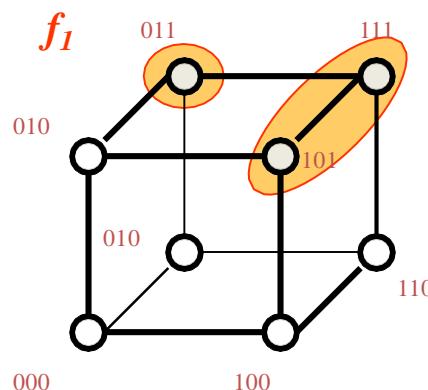
$$f_1 f_2 \quad m_3 = 011 \quad p_5 = \bar{x}yz$$

Min cover has 3 primes:
 $F = \{ p_2, p_4, p_5 \}$

	m_3	m_5	m_7	m_0	m_2	m_3
p_1	✓		✓			
p_2		✓	✓			
p_3					✓	✓
p_4				✓	✓	
p_5	✓					✓

	m_3	m_3
p_1	✓	
p_3		✓
p_5	✓	✓

Note that row p_5 dominates rows p_1 and p_3 , removing these rows, the coverage is complete





■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.

Thank You



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



Switching Theory and Logic Design

ARI-209

By: Dr. Divya Agarwal



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT- II** 10 HOURS
- **Sequential Logic Circuits:** Latches and Flip Flops- SR, D, T and MS-JK Flip Flops, Asynchronous Inputs.
- **Counters and Shift Registers:** Design of Synchronous and Asynchronous Counters- Binary, BCD, Decade and Up/Down Counters, Shift Registers, Types of Shift Registers, Counters using Shift Registers- Ring Counter and Johnson Counter.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-III** **08 HOURS**
- **Integrated circuits:** TTL and CMOS logic families and their characteristics. Brief introduction to RAM and ROM
- **Synchronous Sequential Circuits:** State Tables State Equations and State Diagrams, State Reduction and State Assignment, Design of Clocked Sequential Circuits using State Equations.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-IV** 06 HOURS
- **Finite state machine:** capabilities and limitations, Mealy and Moore models-minimization of completely specified and incompletely specified sequential machines, Partition techniques and merger chart methods, concept of minimal cover table.
- **Algorithmic State Machine:** Representation of sequential circuits using ASM charts synthesis of output and next state functions, Data path control path partition-based design.



Text Books:

1. Leach and Malvino (2011). *Digital principles and Applications*. Tata McGraw-Hill Education
2. Mano, M. M. (2017). *Digital logic and computer design*. Pearson Education India.
3. Jain, R. P. (2003). *Modern digital electronics*. Tata McGraw-Hill Education.

Reference Books:

1. A Anand Kumar. (2016) *Fundamentals of Digital Logic Circuits*, PHI
2. Taub, H., & Schilling, D. L. (1977). *Digital integrated electronics*. McGraw-Hill College.



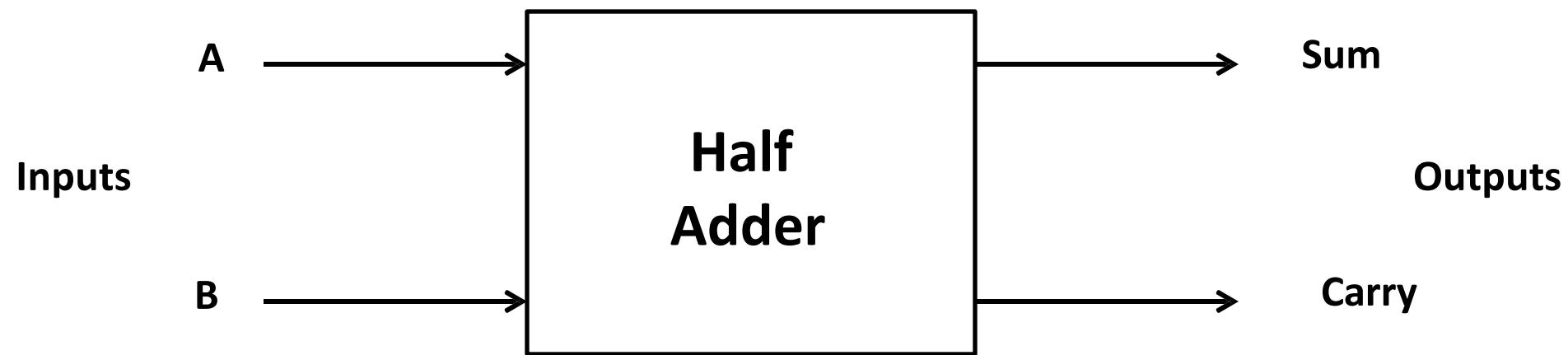
■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.

Half Adder

- ✓ Half adder is a combinational logic circuit with two inputs and two outputs.
- ✓ It is a basic building block for addition of two single bit numbers.



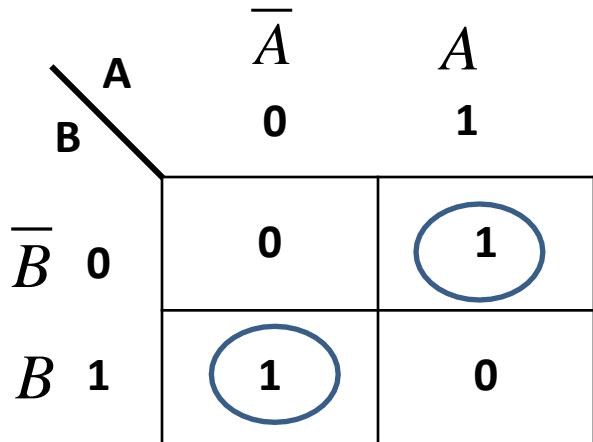
Half Adder

Truth Table for Half Adder

Input		Output	
A	B	Sum (S)	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half Adder

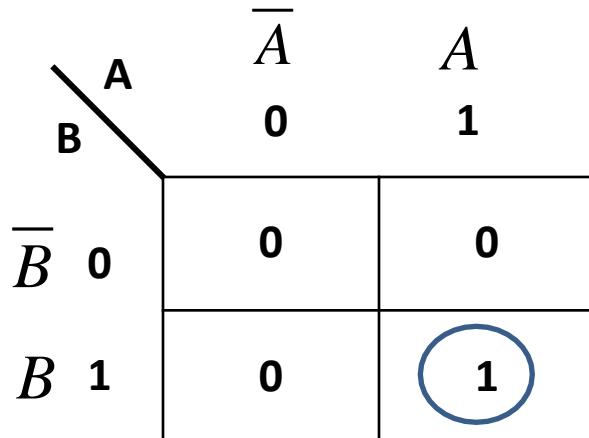
K-map for Sum Output:



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

$$S = A \oplus B$$

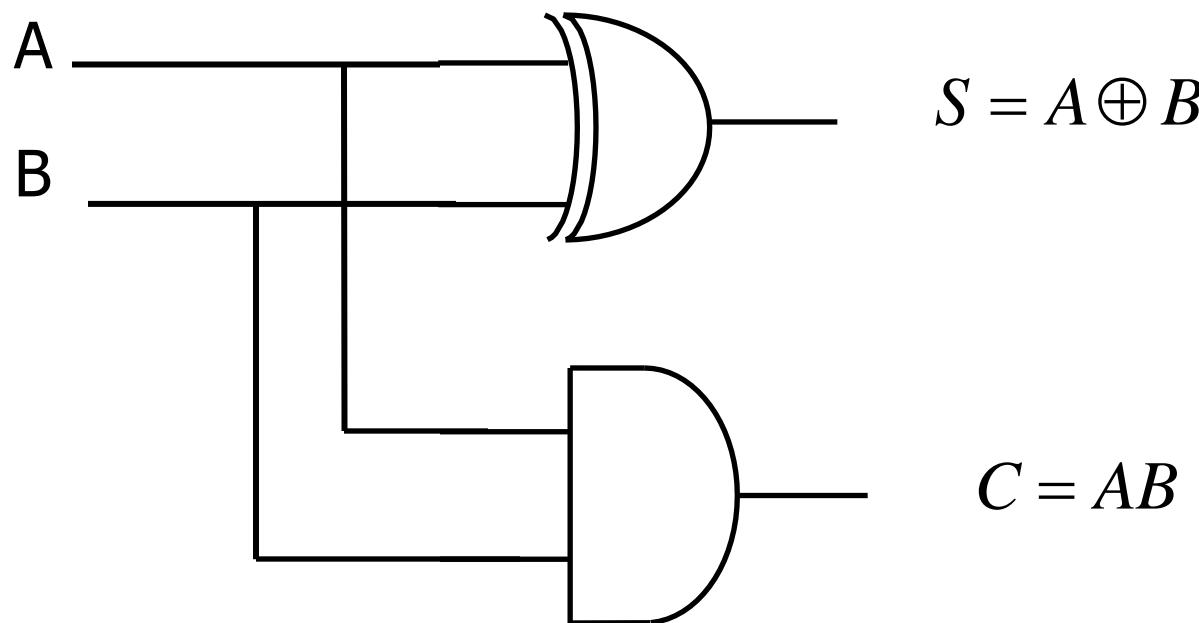
K-map for Carry Output:



$$C = AB$$

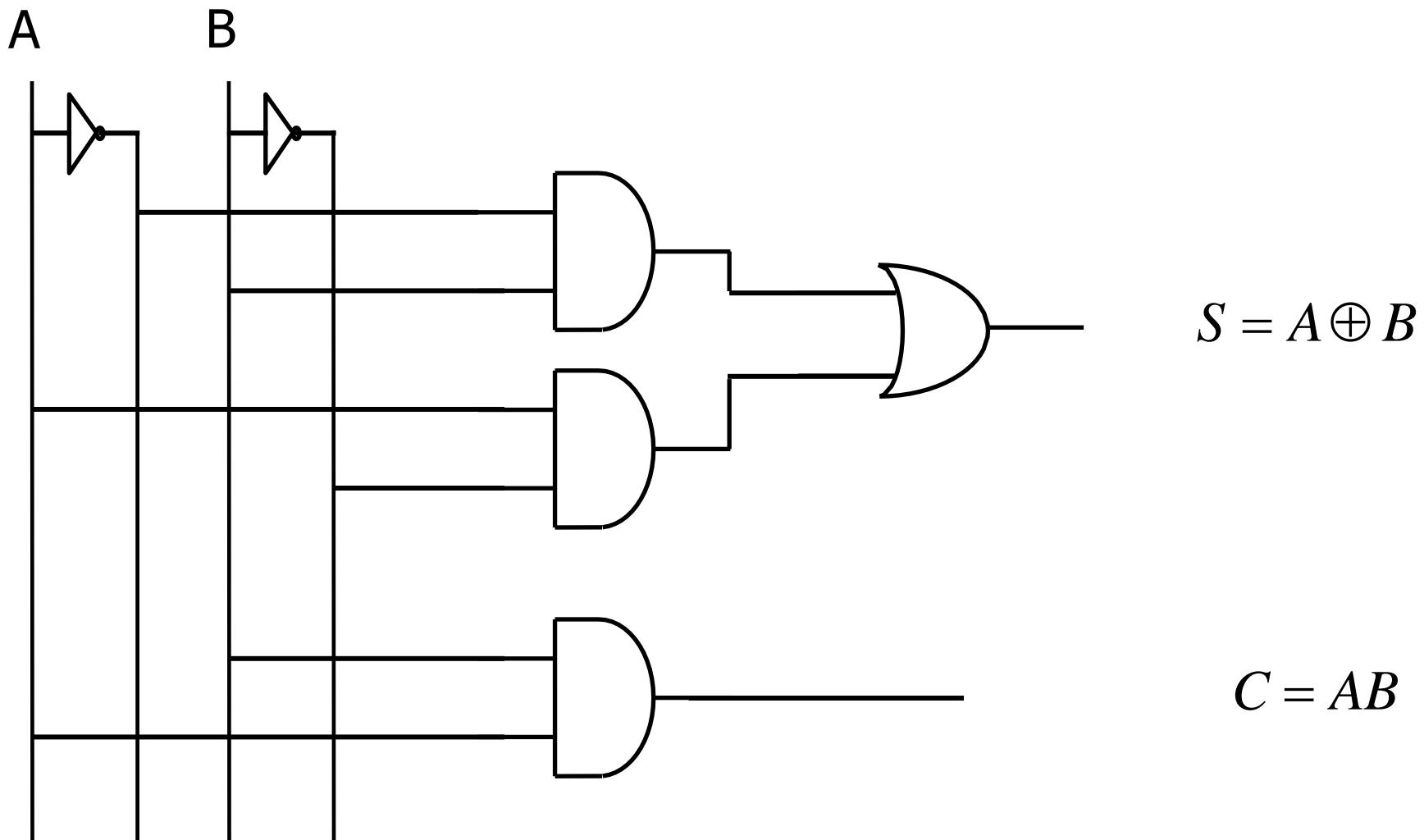
Half Adder

Logic Diagram:



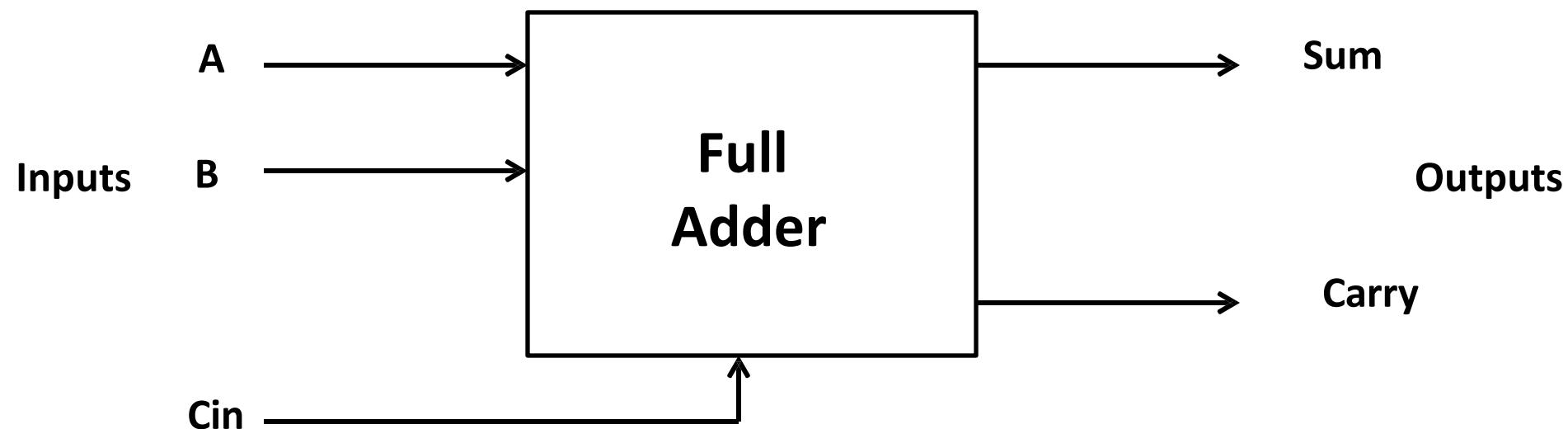
Half Adder

Logic Diagram using Basic Gates:



Full Adder

- ✓ Full adder is a combinational logic circuit with three inputs and two outputs.



Full Adder

Truth Table

Inputs			Outputs	
A	B	Cin	Sum (S)	Carry (C)
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

Full Adder

K-map for Sum Output:

		$\bar{B}\bar{C}$	$\bar{B}C$	$B\bar{C}$	BC
		00	01	11	10
		0	1	0	1
\bar{A}	0	0	1	0	1
	1	1	0	1	0

Arrows point from the circled '1's in the K-map to the output terms $A\bar{B}\bar{C}$, $\bar{A}\bar{B}C$, ABC , and $\bar{A}B\bar{C}$.

$$S = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + ABC + A\bar{B}\bar{C}$$

$$S = \bar{A}\bar{B}\bar{C} + ABC + \bar{A}B\bar{C} + A\bar{B}\bar{C}$$

$$S = C(\bar{A}\bar{B}) + AB + \bar{C}(\bar{A}B + A\bar{B})$$

$$\text{Let } \bar{A}B + A\bar{B} = X$$

$$\therefore S = C(\bar{X}) + \bar{C}(X)$$

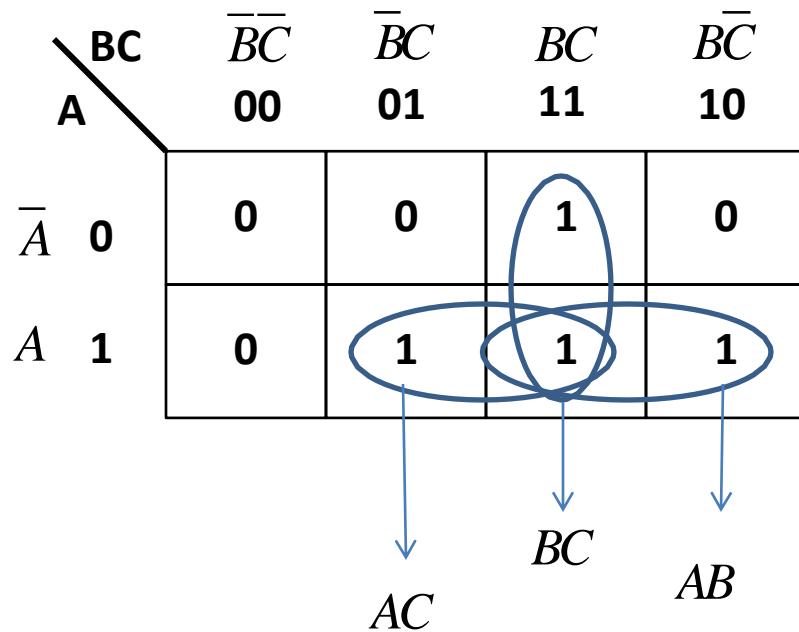
$$S = C \oplus X$$

$$\text{Let } X = A \oplus B$$

$$\therefore S = C \oplus A \oplus B$$

Full Adder

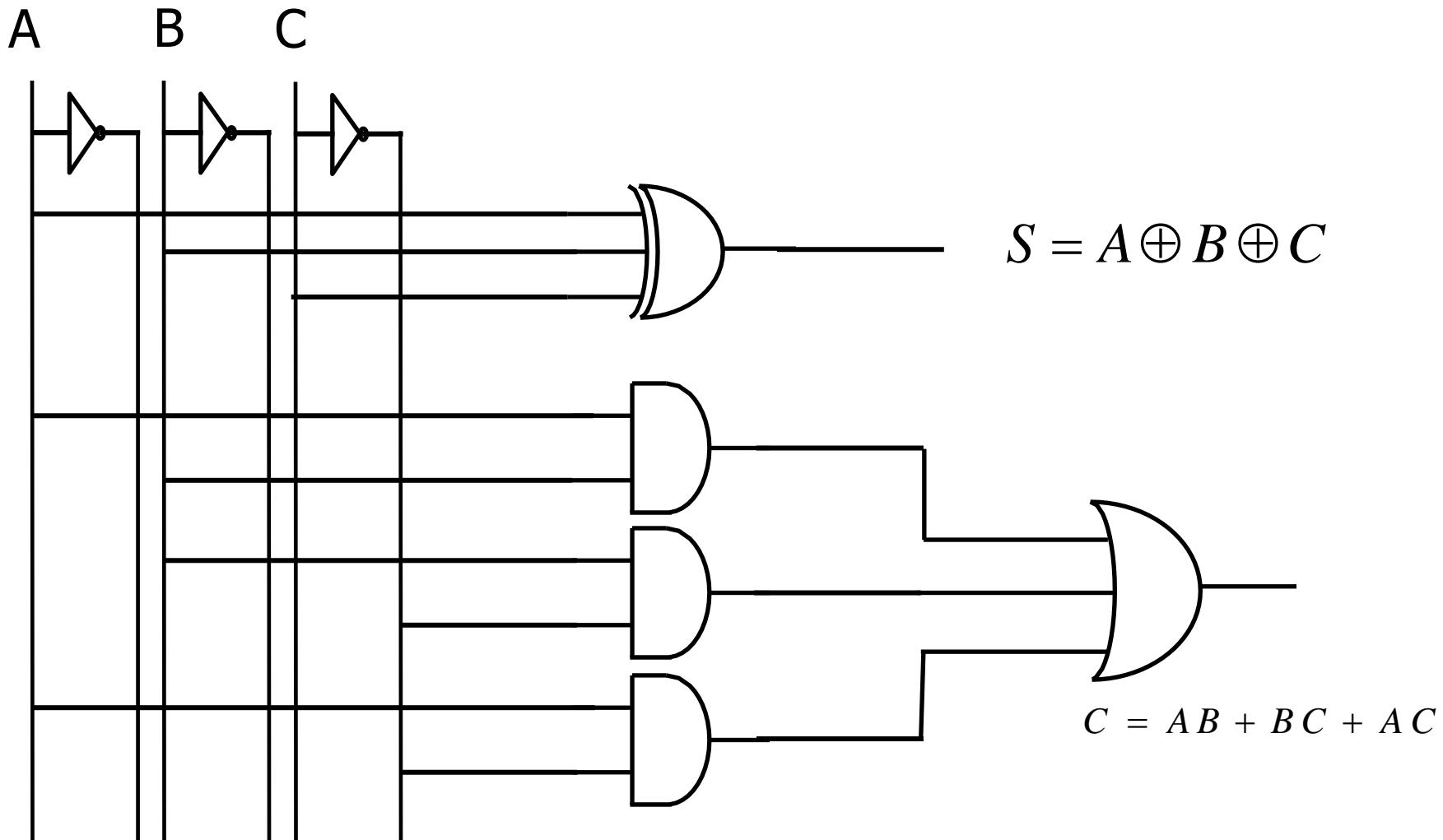
K-map for Carry Output:



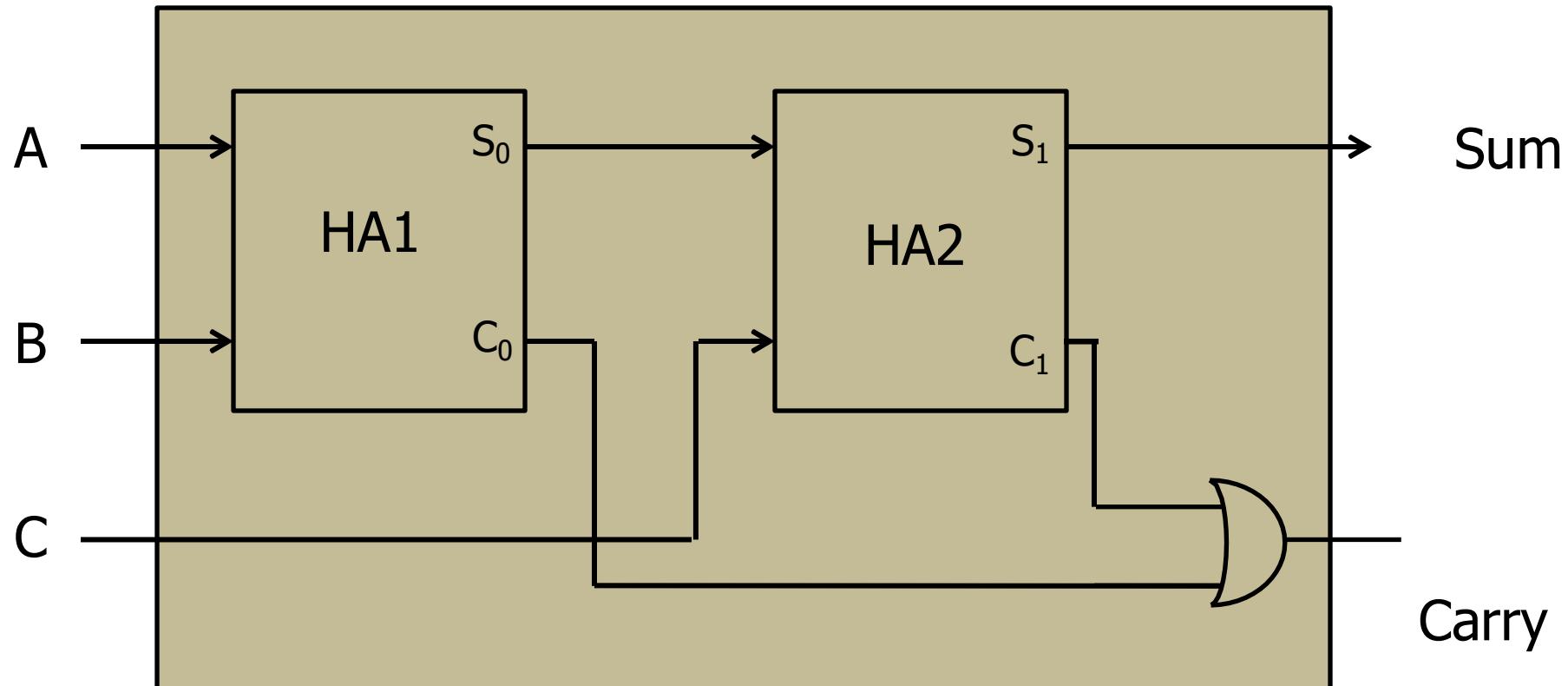
$$C = AB + BC + AC$$

Full Adder

Logic Diagram:

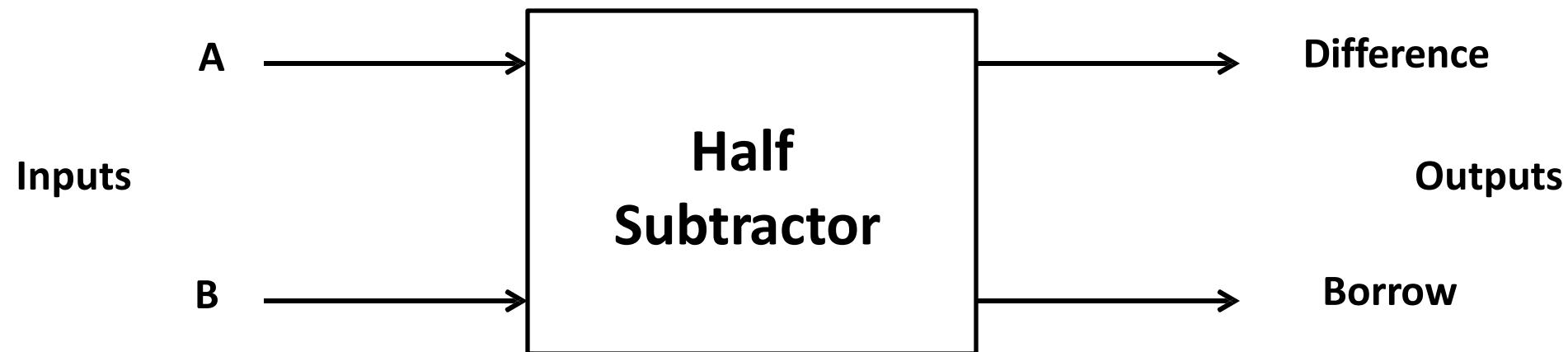


Full Adder using Half Adders



Half Subtractor

- ✓ Half subtractor is a combinational logic circuit with two inputs and two outputs.
- ✓ It is a basic building block for subtraction of two single bit numbers.



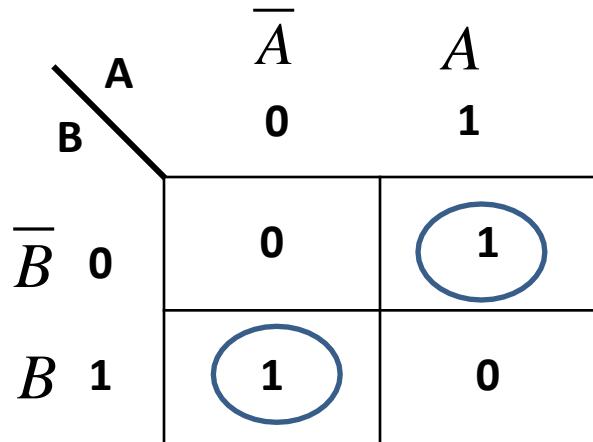
Half Subtractor

Truth Table

Input		Output	
A	B	Difference (D)	Borrow (B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Half Subtractor

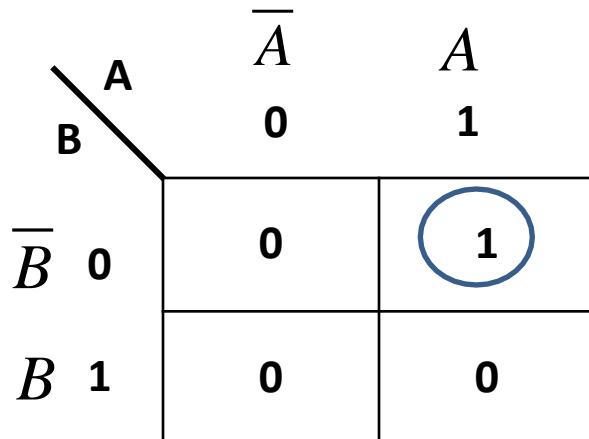
K-map for Difference Output:



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

$$D = A \oplus B$$

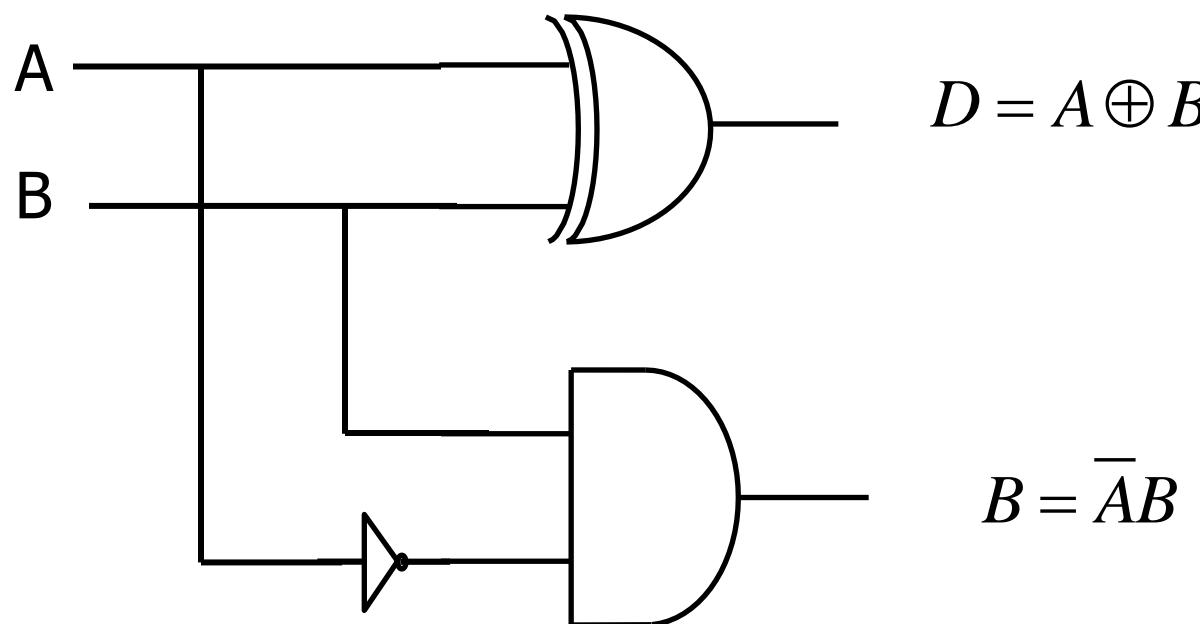
K-map for Borrow Output:



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

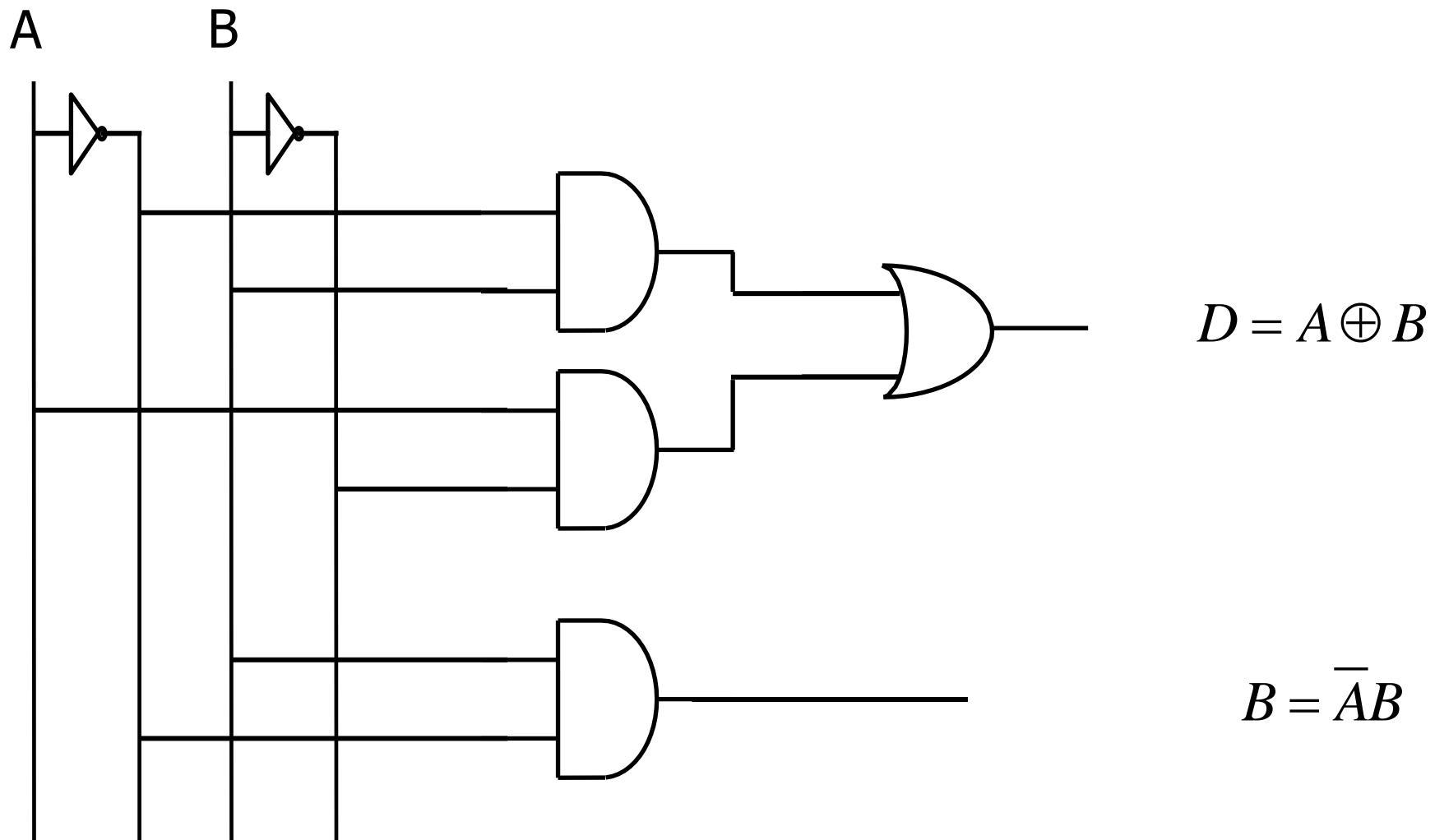
Half Subtractor

Logic Diagram:



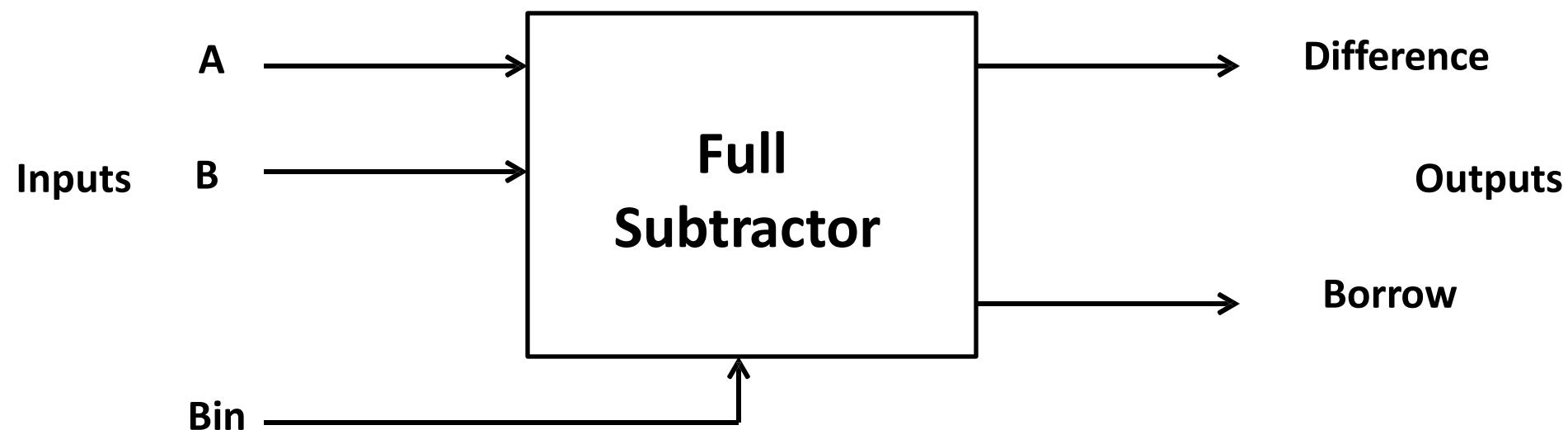
Half Subtractor

Logic Diagram using Basic Gates:



Full Subtractor

- ✓ Full subtractor is a combinational logic circuit with three inputs and two outputs.



Full Subtractor

Truth Table

Inputs			Outputs	
A	B	Bin (C)	Difference (D)	Borrow (B0)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Full Subtractor

K-map for Difference Output:

		$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
		00	01	11	10
A		0	1	0	1
\bar{A}	0	0	1	0	1
	1	1	0	1	0

Labels below the K-map:

- \bar{ABC} (top-left)
- \bar{ABC} (top-right)
- ABC (bottom-middle)
- \bar{ABC} (bottom-right)

$$D = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}C + AB\bar{C}$$

$$D = \bar{A}\bar{B}\bar{C} + ABC + \bar{A}B\bar{C} + A\bar{B}\bar{C}$$

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

Let $\bar{A}B + A\bar{B} = X$

$$\therefore D = C(\bar{X}) + \bar{C}(X)$$

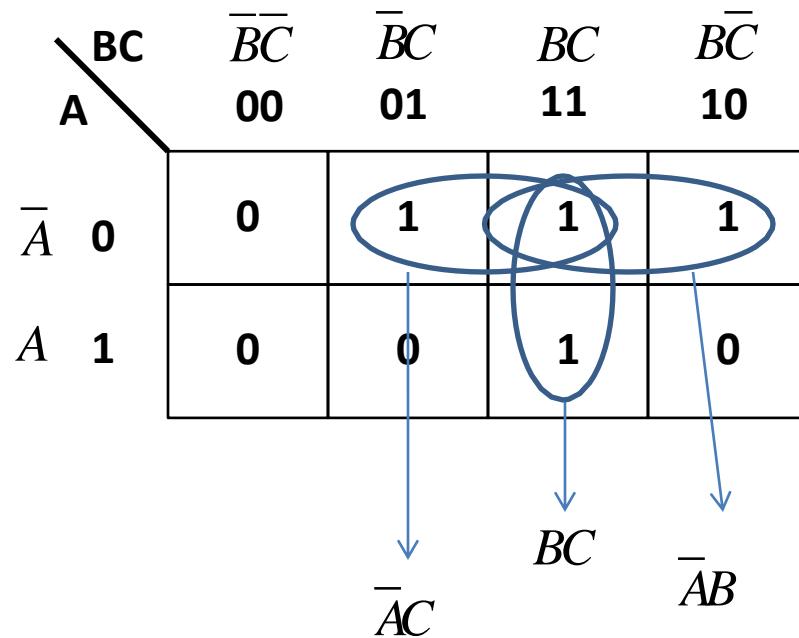
$$D = C \oplus X$$

Let $X = A \oplus B$

$$\therefore D = C \oplus A \oplus B$$

Full Subtractor

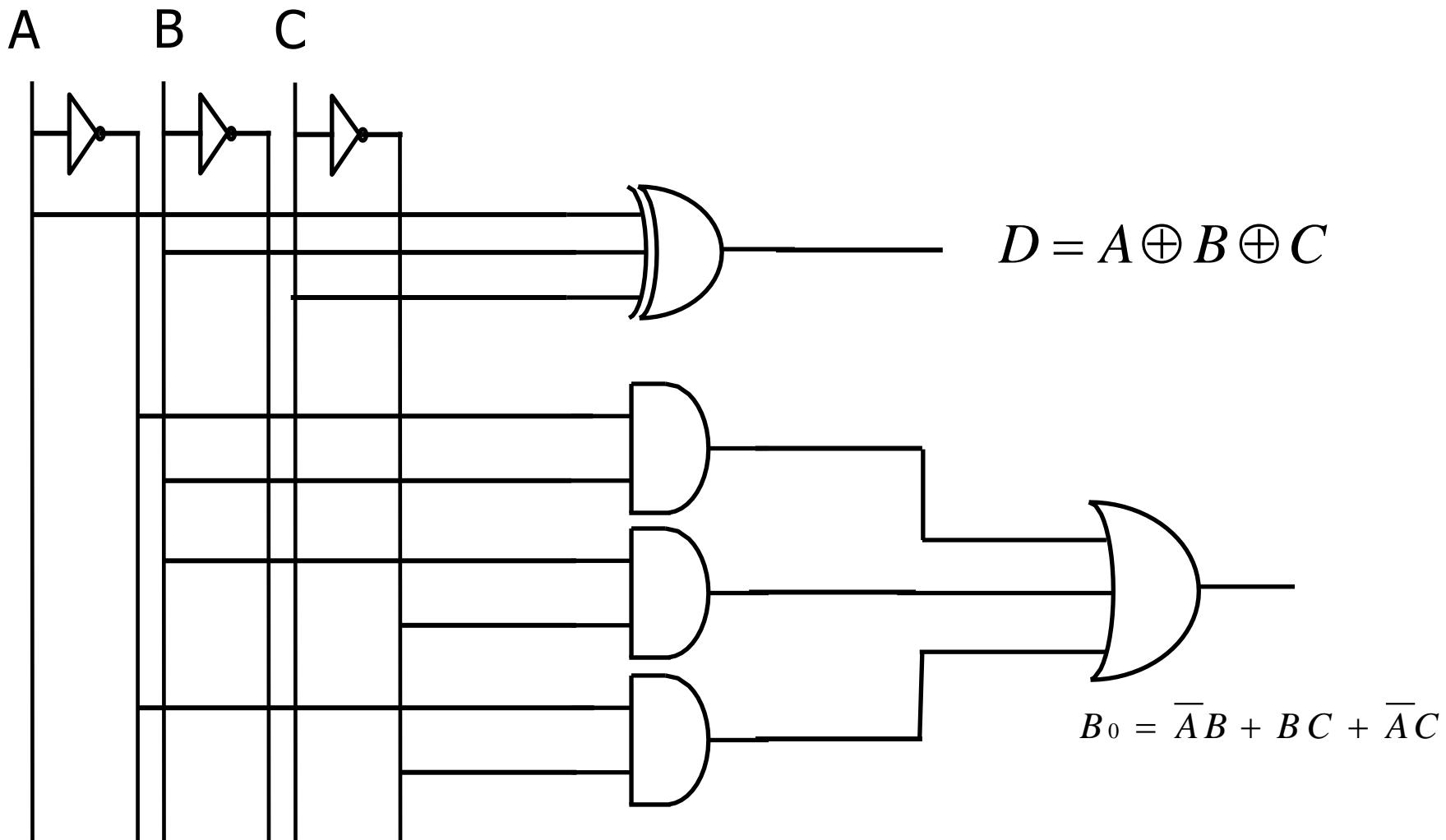
K-map for Borrow Output:



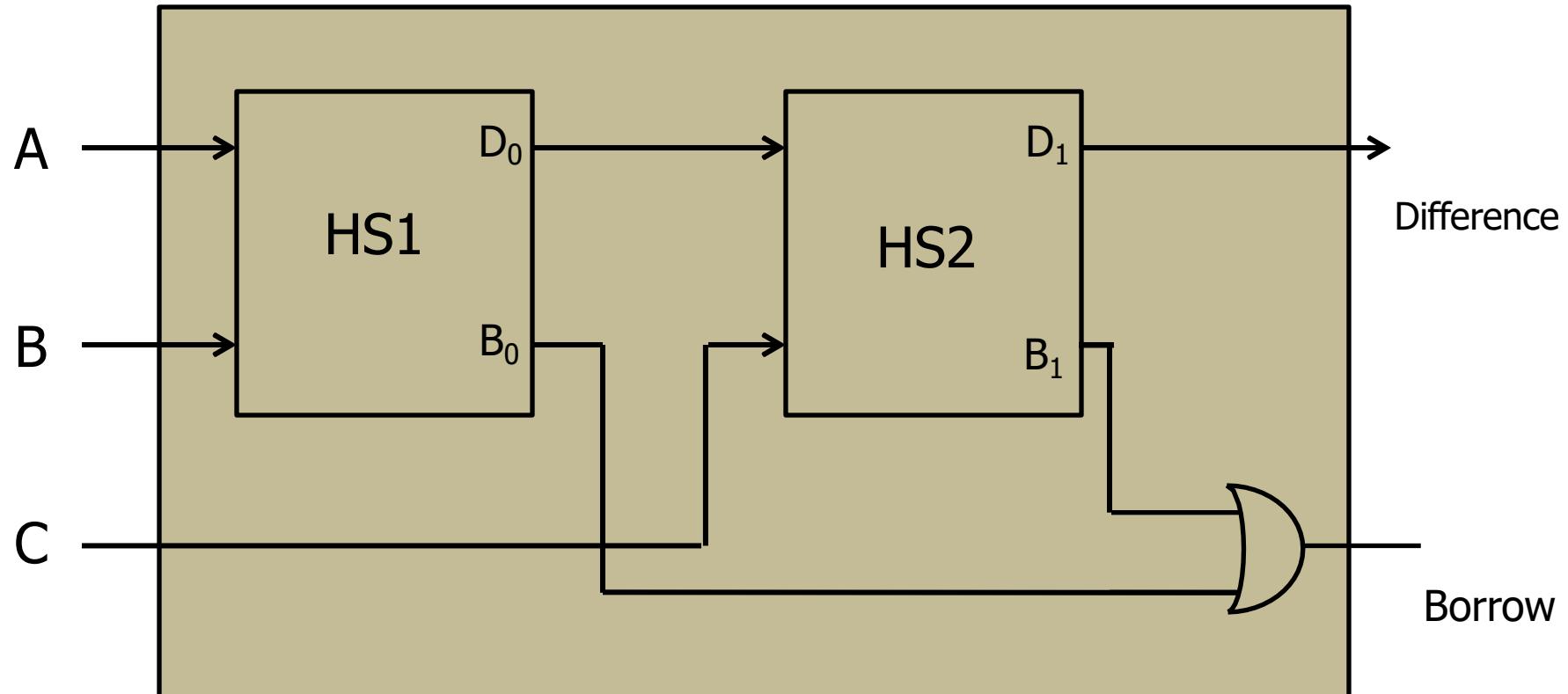
$$B_0 = \overline{AB} + BC + \overline{AC}$$

Full Subtractor

Logic Diagram:



Full Subtractor using Half Subtractor



Disadvantages of Ripple Carry Adder/ Parallel Adder

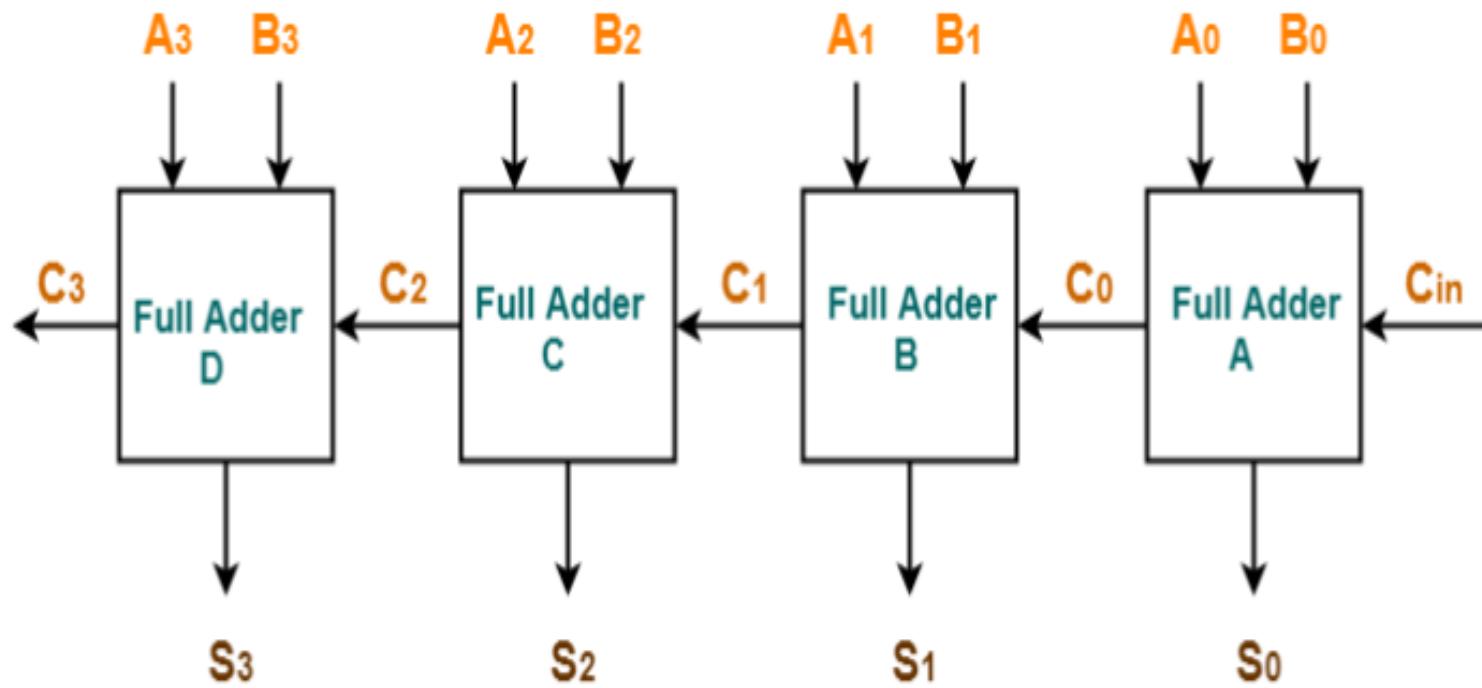
- Parallel Adder does not allow to use all the full adders simultaneously.
- Each full adder has to necessarily wait until the carry bit becomes available from its adjacent full adder.
- This increases the propagation time.
- Due to this reason, Parallel Adder becomes extremely slow.
- This is considered to be the biggest disadvantage of using Parallel adder.

Summary of Ripple Carry Adder/ Parallel Adder

In Parallel Adder,

- Each full adder has to wait for its carry-in from its previous stage full adder.
- Thus, n^{th} full adder has to wait until all $(n-1)$ full adders have completed their operations.
- This causes a delay and makes ripple carry adder extremely slow.
- The situation becomes worst when the value of n becomes very large.
- To overcome this disadvantage, Carry Look Ahead Adder comes into play.

Ripple Carry Adder/ Parallel Adder

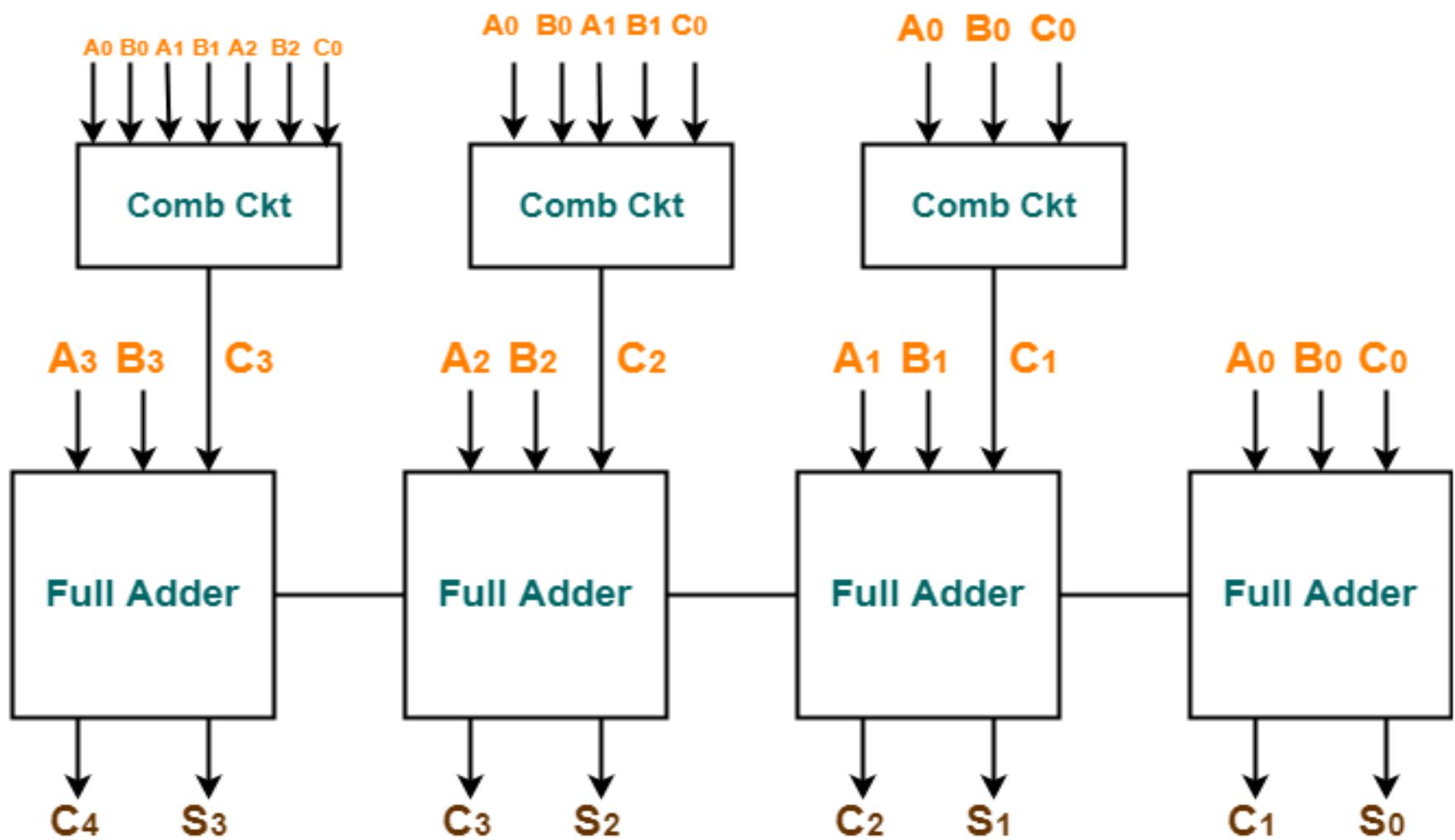


4-bit Ripple Carry Adder

Carry Look Ahead Adder-

- Carry Look Ahead Adder is an improved version of the ripple carry adder.
- It generates the carry-in of each full adder simultaneously without causing any delay.
- The time complexity of carry look ahead adder = $\Theta(\log n)$.

Carry Look Ahead Adder-



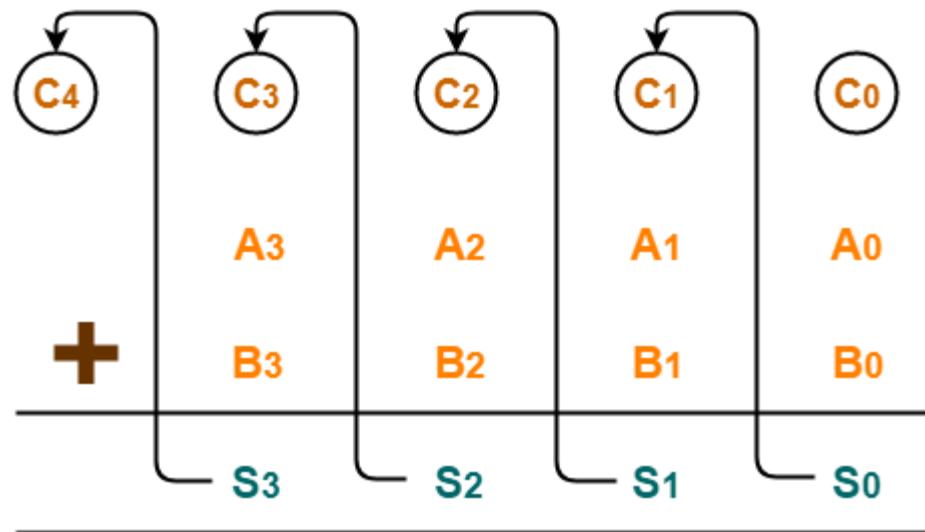
Carry Look Ahead Adder Logic Diagram

Carry Look Ahead Adder Working-

- The working of carry look ahead adder is based on the principle-
“The carry-in of any stage full adder is independent of the carry bits generated during intermediate stages”
- The carry-in of any stage full adder depends only on the following two parameters-
 - Bits being added in the previous stages
 - Carry-in provided in the beginning
- Now, The above two parameters are always known from the beginning.
- So, the carry-in of any stage full adder can be evaluated at any instant of time.
- Thus, any full adder need not wait until its carry-in is generated by its previous stage full adder.

4-Bit Carry Look Ahead Adder-

- Consider two 4-bit binary numbers $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are to be added.
- Mathematically, the two numbers will be added as-



Adding two 4-bit Numbers

4-Bit Carry Look Ahead Adder-

From here, we have-

$$C_1 = C_0 (A_0 \oplus B_0) + A_0 B_0$$

$$C_2 = C_1 (A_1 \oplus B_1) + A_1 B_1$$

$$C_3 = C_2 (A_2 \oplus B_2) + A_2 B_2$$

$$C_4 = C_3 (A_3 \oplus B_3) + A_3 B_3$$

For simplicity,

Let-

$G_i = A_i B_i$ where G is called carry generator

$P_i = A_i \oplus B_i$ where P is called carry propagator

4-Bit Carry Look Ahead Adder-

Then, re-writing the above equations, we have-

$$C_1 = C_0 P_0 + G_0 \dots \dots \dots \quad (1)$$

$$C_2 = C_1 P_1 + G_1 \dots \dots \dots \quad (2)$$

$$C_3 = C_2 P_2 + G_2 \dots \dots \dots \quad (3)$$

$$C_4 = C_3 P_3 + G_3 \dots \dots \dots \quad (4)$$

Now,

Clearly, C1, C2 and C3 are intermediate carry bits.

So, let's remove C_1 , C_2 and C_3 from RHS of every equation.

Substituting (1) in (2), we get C_2 in terms of C_0 .

Then, substituting (2) in (3), we get C_3 in terms of C_0 and so on.

4-Bit Carry Look Ahead Adder-

Finally, we have the following equations-

$$C_1 = C_0 P_0 + G_0$$

$$C_2 = C_0 P_0 P_1 + G_0 P_1 + G_1$$

$$C_3 = C_0 P_0 P_1 P_2 + G_0 P_1 P_2 + G_1 P_2 + G_2$$

$$C_4 = C_0 P_0 P_1 P_2 P_3 + G_0 P_1 P_2 P_3 + G_1 P_2 P_3 + G_2 P_3 + G_3$$

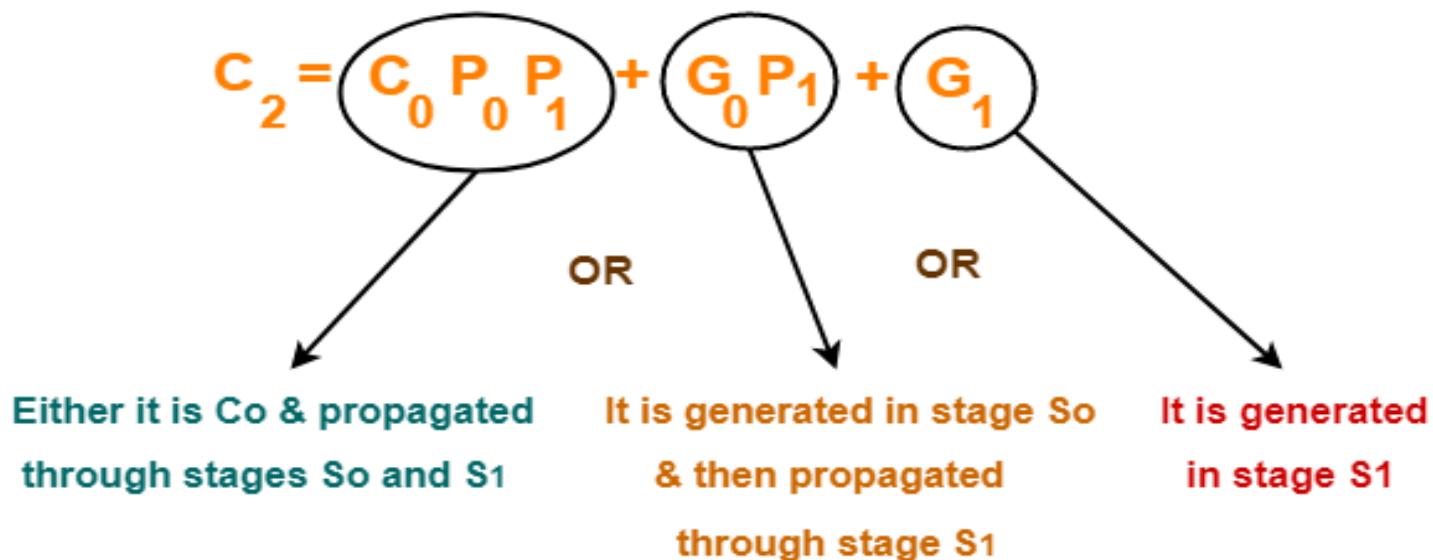
These equations are important to remember.

These equations show that the carry-in of any stage full adder depends only on-

- Bits being added in the previous stages
- Carry bit which was provided in the beginning

Trick To Memorize Above Equations-

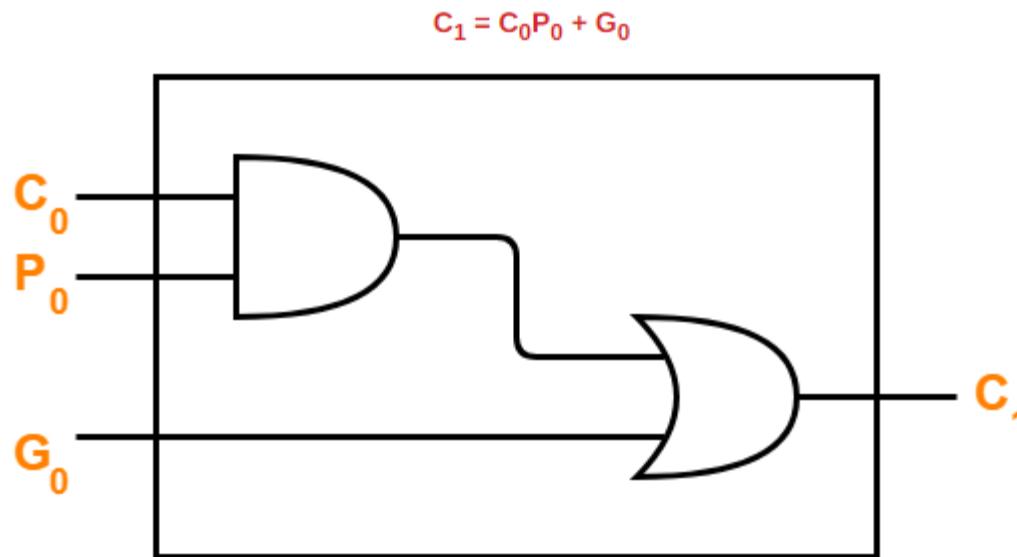
- As an example, let us consider the equation for generating carry bit C2.
- There are three possible reasons for generation of C2 as depicted in the following picture-



- In the similar manner, we can write other equations as well very easily.

Implementation Of Carry Generator Circuits-

- The above carry generator circuits are usually implemented as-
 - Two level combinational circuits.
 - Using AND and OR gates where gates are assumed to have any number of inputs.
- **Implementation Of C1–**
 - The carry generator circuit for C1 is implemented as shown below.
 - It requires 1 AND gate and 1 OR gate.



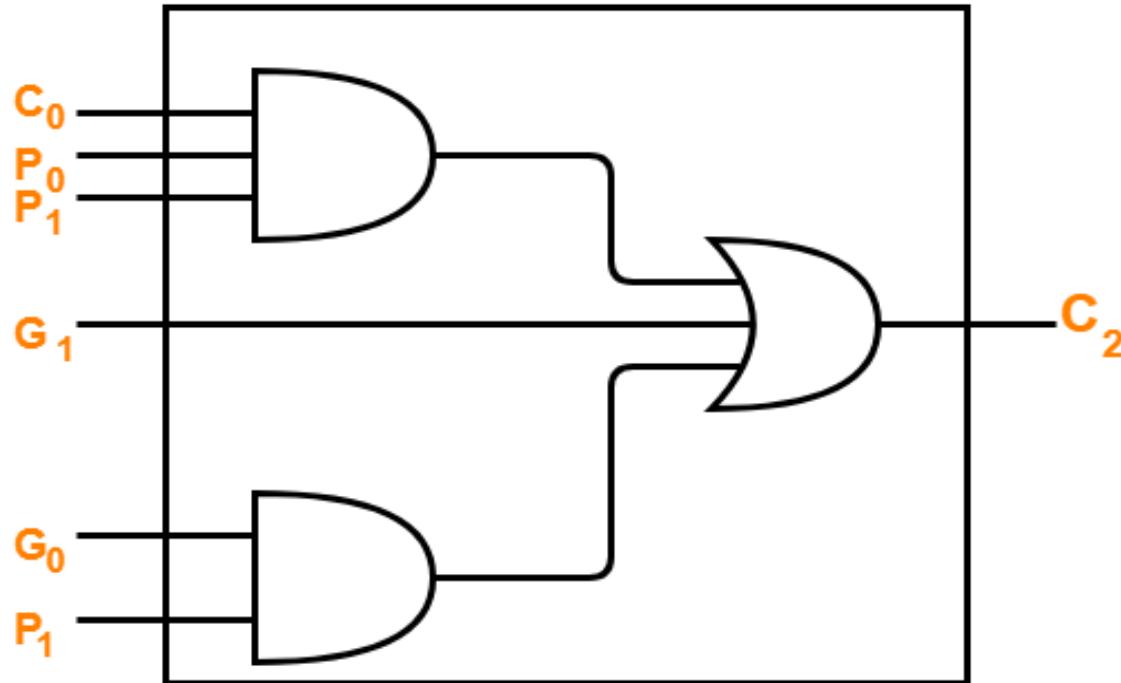
Implementation of C1

Implementation Of Carry Generator Circuits-

Implementation Of C_2 -

- The carry generator circuit for C_2 is implemented as shown below.
- It requires 2 AND gates and 1 OR gate.

$$C_2 = C_0P_0P_1 + G_0P_1 + G_1$$



Implementation of C_2

Implementation Of Carry Generator Circuits-

Implementation Of C_3 & C_4 -

- Similarly, we implement C_3 and C_4 .
 - Implementation of C_3 uses 3 AND gates and 1 OR gate.
 - Implementation of C_4 uses 4 AND gates and 1 OR gate.
- Total number of gates required to implement carry generators (provided carry propagators P_i and carry generators G_i) are-
- Total number of AND gates required for addition of 4-bit numbers = $1 + 2 + 3 + 4 = 10$.
- Total number of OR gates required for addition of 4-bit numbers = $1 + 1 + 1 + 1 = 4$.
- For a n-bit carry look ahead adder to evaluate all the carry bits, it requires-
 - Number of AND gates = $n(n+1) / 2$
 - Number of OR gates = n

Implementation Of Carry Generator Circuits-

Advantages of Carry Look Ahead Adder-

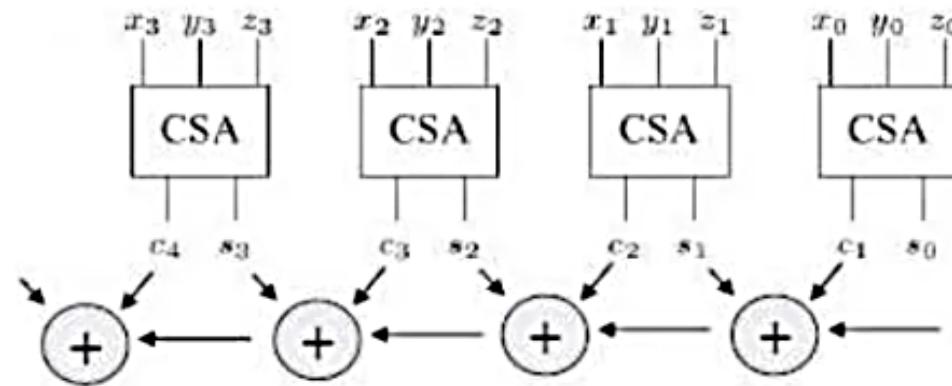
- The advantages of carry look ahead adder are-
- It generates the carry-in for each full adder simultaneously.
- It reduces the propagation delay.

Disadvantages of Carry Look Ahead Adder-

- The disadvantages of carry look ahead adder are-
- It involves complex hardware.
- It is costlier since it involves complex hardware.
- It gets more complicated as the number of bits increases.

Carry Save Adder (CSA)

- In Carry Save Adder (CSA), three bits are added parallelly at a time.
- In this scheme, the carry is not propagated through the stages. Instead, carry is stored in present stage, and updated as addend value in the next stage [2]. Hence, the delay due to the
- carry is reduced in this scheme.



Block Diagram of CSA

Carry Save Adder (CSA)

The carry save adder seems to be the most useful adder for our application. It is simply a parallel ensemble of k full-adders without any horizontal connection. Its main function is to add three k -bit integers A , B , and C to produce two integers C' and S such that

$$C' + S = A + B + C .$$

As an example, let $A = 40$, $B = 25$, and $C = 20$, we compute S and C' as shown below:

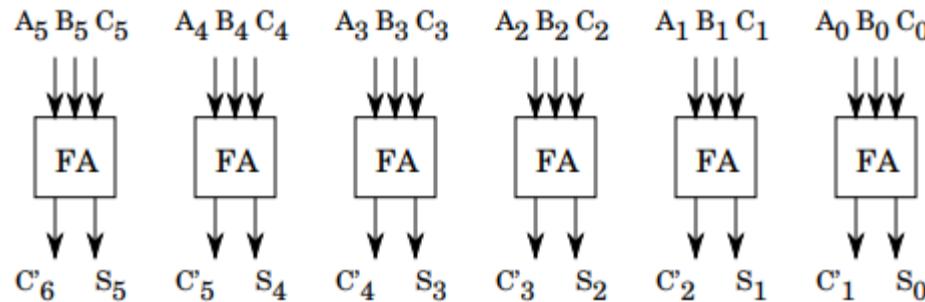
$$\begin{array}{rcl} A = 40 & = & 1 \ 0 \ 1 \ 0 \ 0 \ 0 \\ B = 25 & = & 0 \ 1 \ 1 \ 0 \ 0 \ 1 \\ C = 20 & = & 0 \ 1 \ 0 \ 1 \ 0 \ 0 \\ \hline S = 37 & = & 1 \ 0 \ 0 \ 1 \ 0 \ 1 \\ \hline C' = 48 & = & 0 \ 1 \ 1 \ 0 \ 0 \ 0 \end{array}$$

The i th bit of the sum S_i and the $(i+1)$ st bit of the carry C'_{i+1} is calculated using the equations

$$\begin{aligned} S_i &= A_i \oplus B_i \oplus C_i . \\ C'_{i+1} &= A_i B_i + A_i C_i + B_i C_i , \end{aligned}$$

in other words, a carry save adder cell is just a full-adder cell. A carry save adder, sometimes named a one-level CSA, is illustrated below for $k = 6$.

Carry Save Adder (CSA)



Since the input vectors A , B , and C are applied in parallel, the total delay of a carry save adder is equal to the total delay of a single FA cell. Thus, the addition of three integers to compute two integers requires a single FA delay. Furthermore, the CSA requires only k times the areas of FA cell, and scales up very easily by adding more parallel cells. The subtraction operation can also be performed by using 2's complement encoding. There are basically two disadvantages of the carry save adders:

- It does not really solve our problem of adding two integers and producing a single output. Instead, it adds three integers and produces two such that sum of these two is equal to the sum of three inputs. This method may not be suitable for application which only needs the regular addition.
- The sign detection is hard: When a number is represented as a carry-save pair (C, S) such that its actual value is $C + S$, we may not know the exact sign of total sum $C + S$. Unless the addition is performed in full length, the correct sign may never be determined.

Magnitude Comparator

- It is a combinational logic circuit.
- Digital Comparator is used to compare the value of two binary digits.
- There are two types of digital comparator (i) Identity Comparator (ii) Magnitude Comparator.
- **IDENTITY COMPARATOR:** This comparator has only one output terminal for when $A=B$, either $A=B=1$ (High) or $A=B=0$ (Low)
- **MAGNITUDE COMPARATOR:** This Comparator has three output terminals namely $A>B$, $A=B$, $A<B$. Depending on the result of comparison, one of these output will be high (1)
- Block Diagram of Magnitude Comparator is shown in Fig. 1

Magnitude Comparator

BLOCK DIAGRAM OF MAGNITUDE COMPARATOR

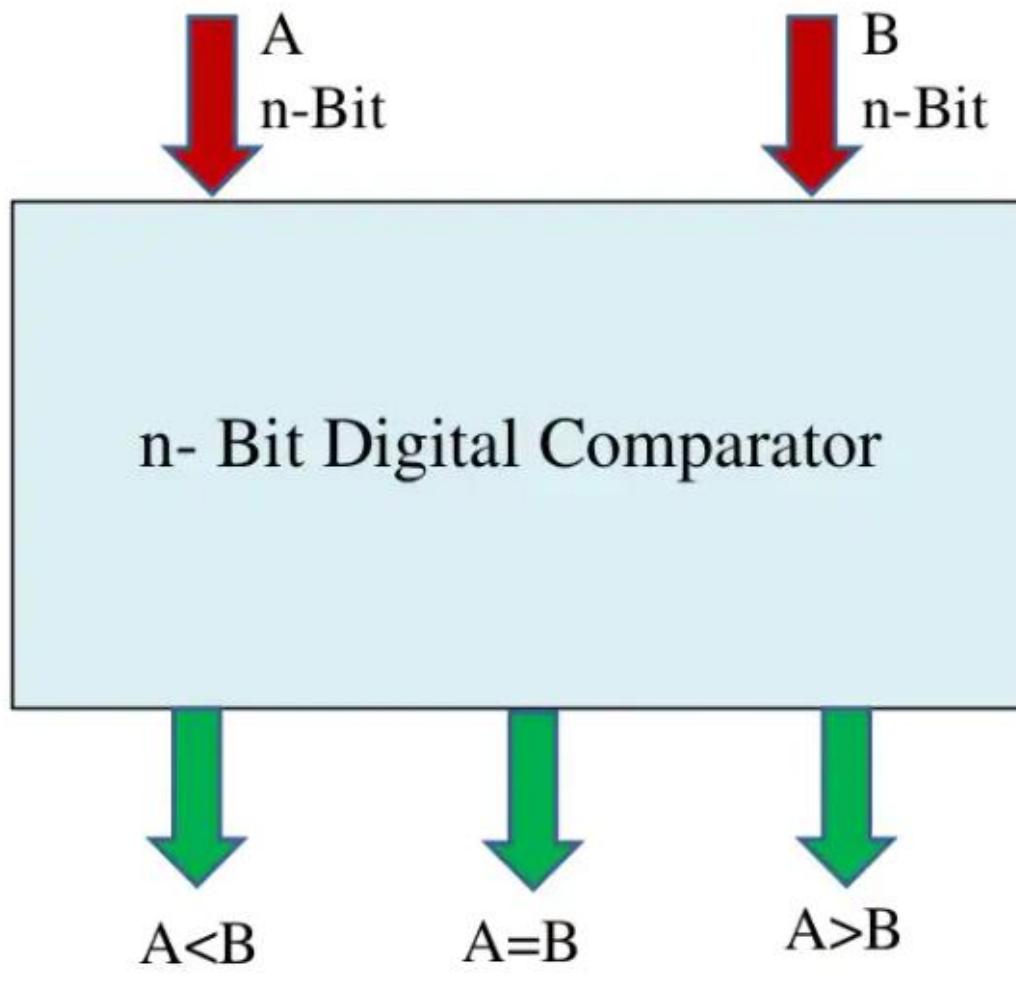


Fig. 1

Magnitude Comparator

1- Bit Magnitude Comparator:

- This magnitude comparator has two inputs A and B and three outputs $A < B$, $A = B$ and $A > B$.
- This magnitude comparator compares the two numbers of single bits.
- Truth Table of 1-Bit Comparator

INPUTS		OUTPUTS		
A	B	Y_1 ($A < B$)	Y_2 ($A = B$)	Y_3 ($A > B$)
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Magnitude Comparator

K-Maps For All Three Outputs :

		B	\bar{B}	B
		A	0	1
A	\bar{A}	0	0	1
	1	0	0	0

K-Map for Y_1 : $A < B$

$$Y_1 = \bar{A}B$$

		B	\bar{B}	B
		A	0	1
\bar{A}	0	1	0	0
	1	0	0	1

K-Map for Y_2 : $A=B$

$$Y_2 = \bar{A}\bar{B} + AB$$

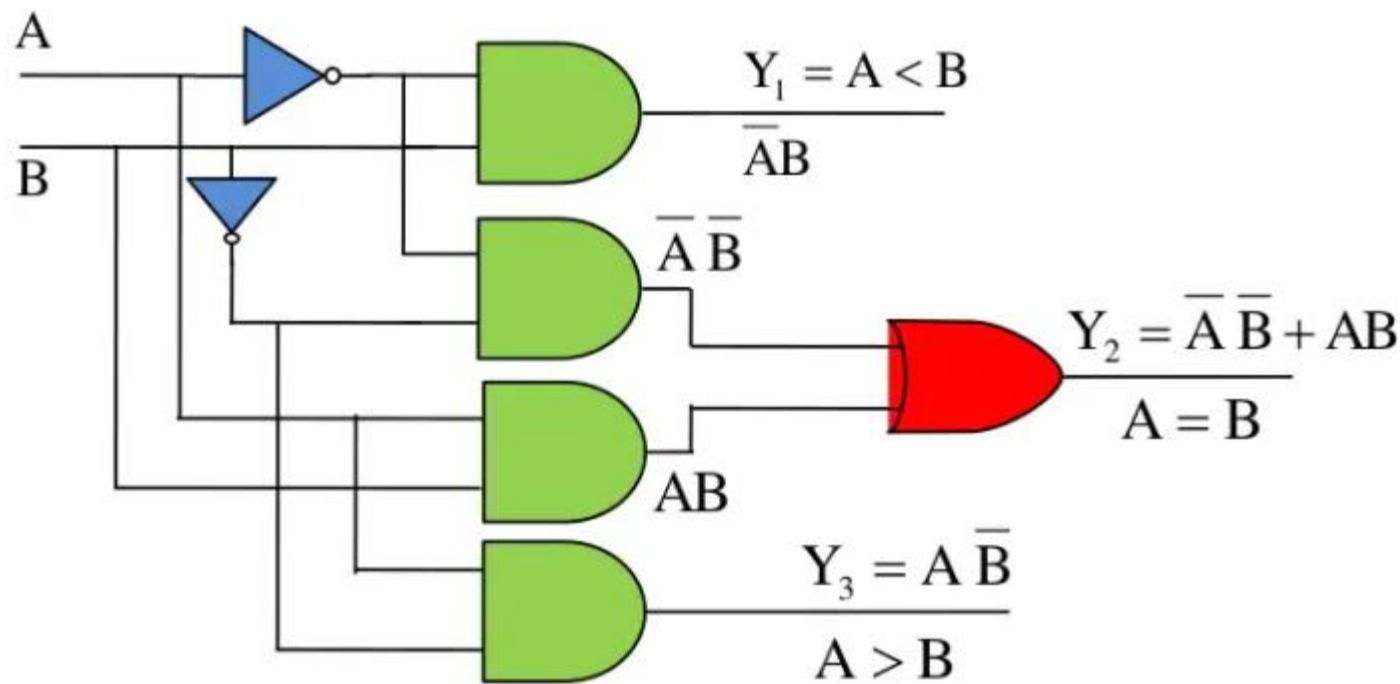
		B	\bar{B}	B
		A	0	1
\bar{A}	0	0	0	0
	1	1	0	0

K-Map for Y_2 : $A > B$

$$Y_2 = A\bar{B}$$

Magnitude Comparator

Realization of One Bit Comparator



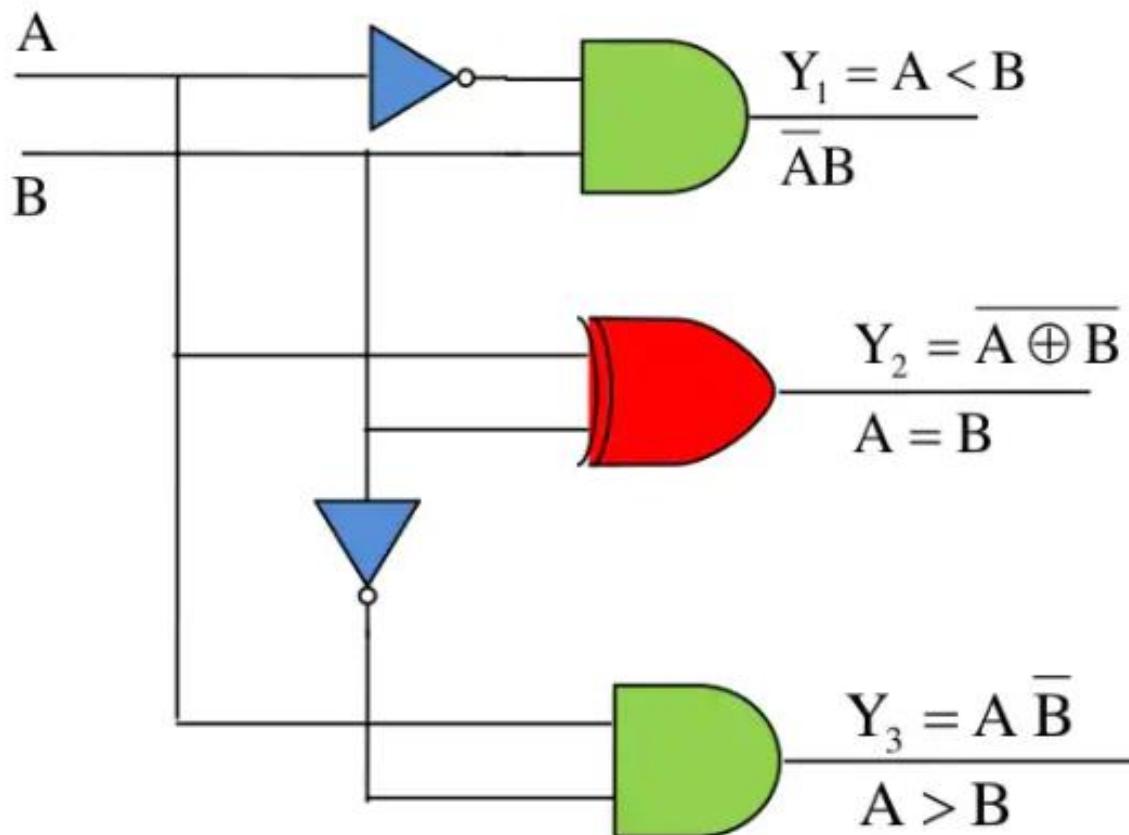
$$Y_1 = \overline{A}B$$

$$Y_2 = \overline{A}\overline{B} + AB$$

$$Y_3 = A\overline{B}$$

Magnitude Comparator

Realization of by Using AND , EX-NOR gates



Magnitude Comparator

2-Bit Comparator:

- A comparator which is used to compare two binary numbers each of two bits is called a 2-bit magnitude comparator.
- Fig. 2 shows the block diagram of 2-Bit magnitude comparator.
- It has four inputs and three outputs.
- Inputs are A_0, A_1, B_0 and B_1 and Outputs are Y_1, Y_2 and Y_3

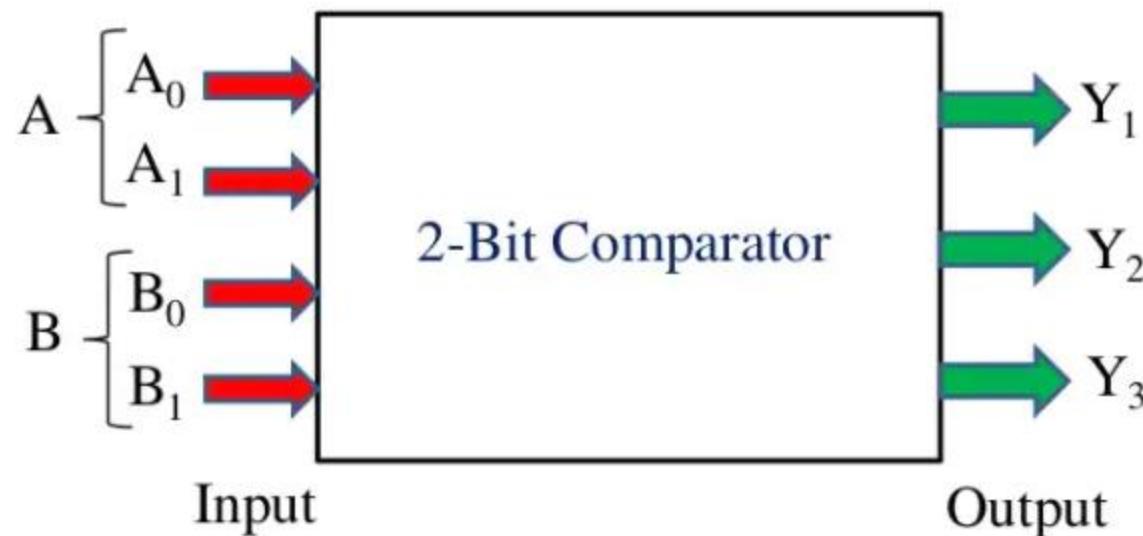


Fig. 2

Magnitude Comparator

GREATER THAN (A>B)

A_1	A_0	B_1	B_0
1	0	0	1
1	1	1	0
0	1	0	0

1. If $A_1 = 1$ and $B_1 = 0$ then $A > B$
2. If A_1 and B_1 are same, i.e $A_1 = B_1 = 1$ or $A_1 = B_1 = 0$ and $A_0 = 1$, $B_0 = 0$ then $A > B$

LESS THAN (A<B)

Similarly,

1. If $A_1 = B_1 = 1$ and $A_0 = 0$, $B_0 = 1$, then $A < B$
2. If $A_1 = B_1 = 0$ and $A_0 = 0$, $B_0 = 1$ then $A < B$

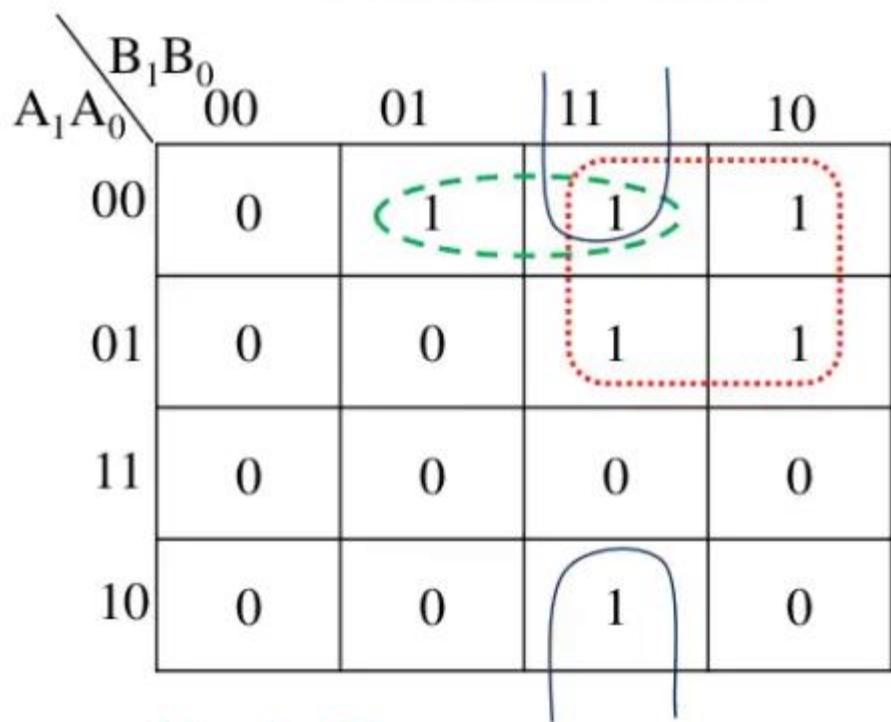
Magnitude Comparator

TRUTH TABLE

INPUT				OUTPUT		
A_1	A_0	B_1	B_0	$Y_1 = A < B$	$Y_2 = (A = B)$	$Y_3 = A > B$
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

Magnitude Comparator

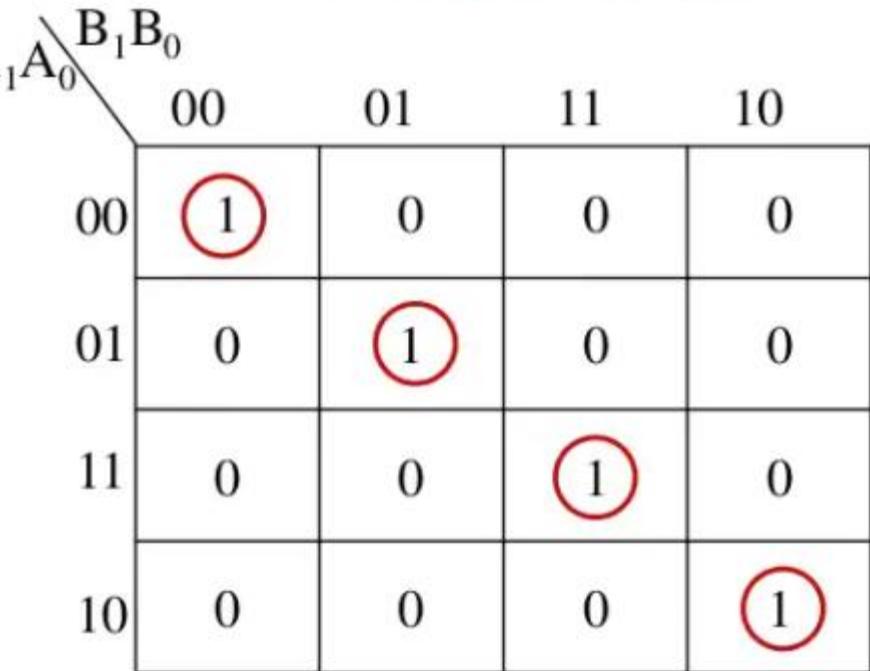
K-Map for A < B:



For A < B

$$Y_1 = \overline{A}_1 \overline{A}_0 B_0 + \overline{A}_1 B_1 + \overline{A}_0 B_1 B_0$$

K-Map for A = B:

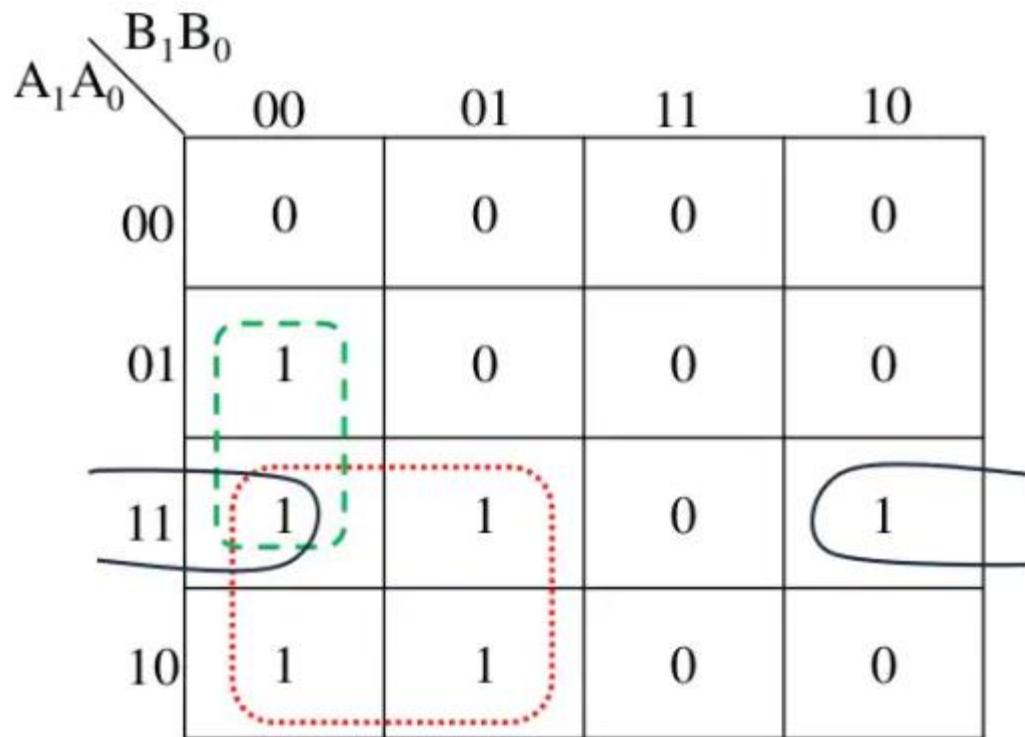


For A = B

$$Y_2 = \overline{A}_1 \overline{A}_0 \overline{B}_1 \overline{B}_0 + \overline{A}_1 A_0 \overline{B}_1 B_0 + A_1 A_0 B_1 B_0 + A_1 \overline{A}_0 B_1 \overline{B}_0$$

Magnitude Comparator

K-Map For A>B



$$Y_3 = A_0 \overline{B_1} \overline{B_0} + A_1 \overline{B_1} + A_1 A_0 \overline{B_0}$$

Magnitude Comparator

For A=B From K-Map

$$Y_2 = \overline{A_1} \overline{A_0} \overline{B_1} \overline{B_0} + \overline{A_1} A_0 \overline{B_1} B_0 + A_1 \overline{A_0} B_1 B_0 + A_1 \overline{A_0} B_1 \overline{B_0}$$

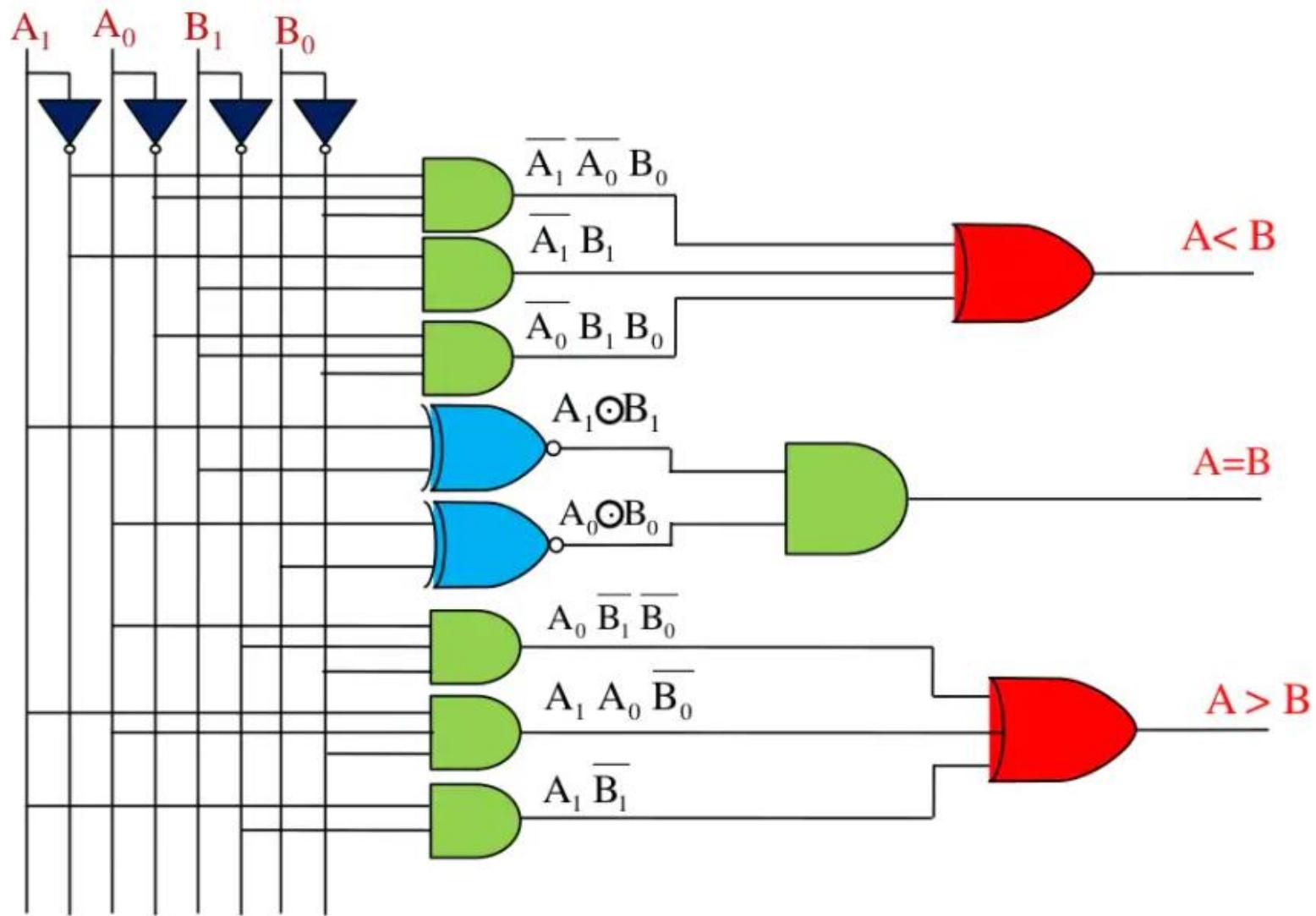
$$Y_2 = \overline{A_0} B_0 (A_1 \overline{B_1} + A_1 B_1) + A_0 \overline{B_0} (\overline{A_1} \overline{B_1} + A_1 B_1)$$

$$Y_2 = (\overline{A_1} \overline{B_1} + A_1 B_1) (\overline{A_0} \overline{B_0} + A_0 B_0)$$

$$Y_2 = (A_1 \odot B_1) (A_0 \odot B_0)$$

Magnitude Comparator

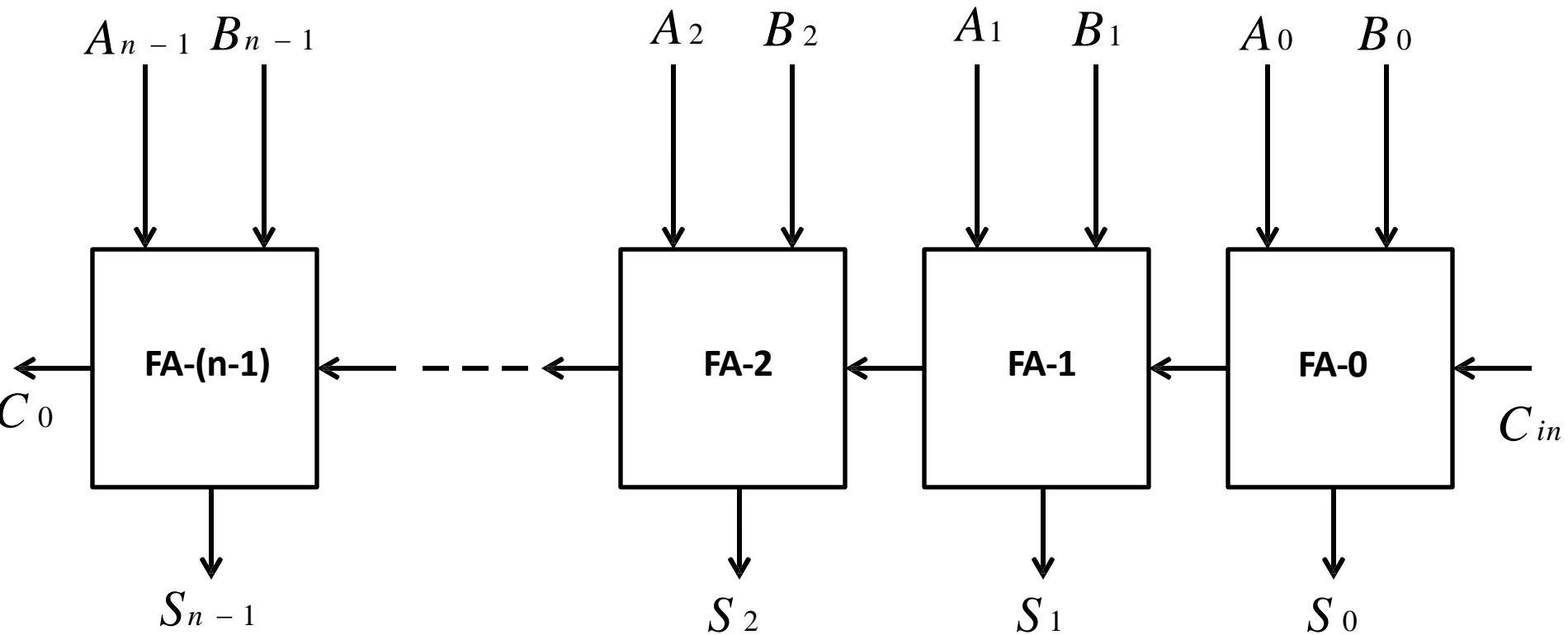
LOGIC DIAGRAM OF 2-BIT COMPARATOR:



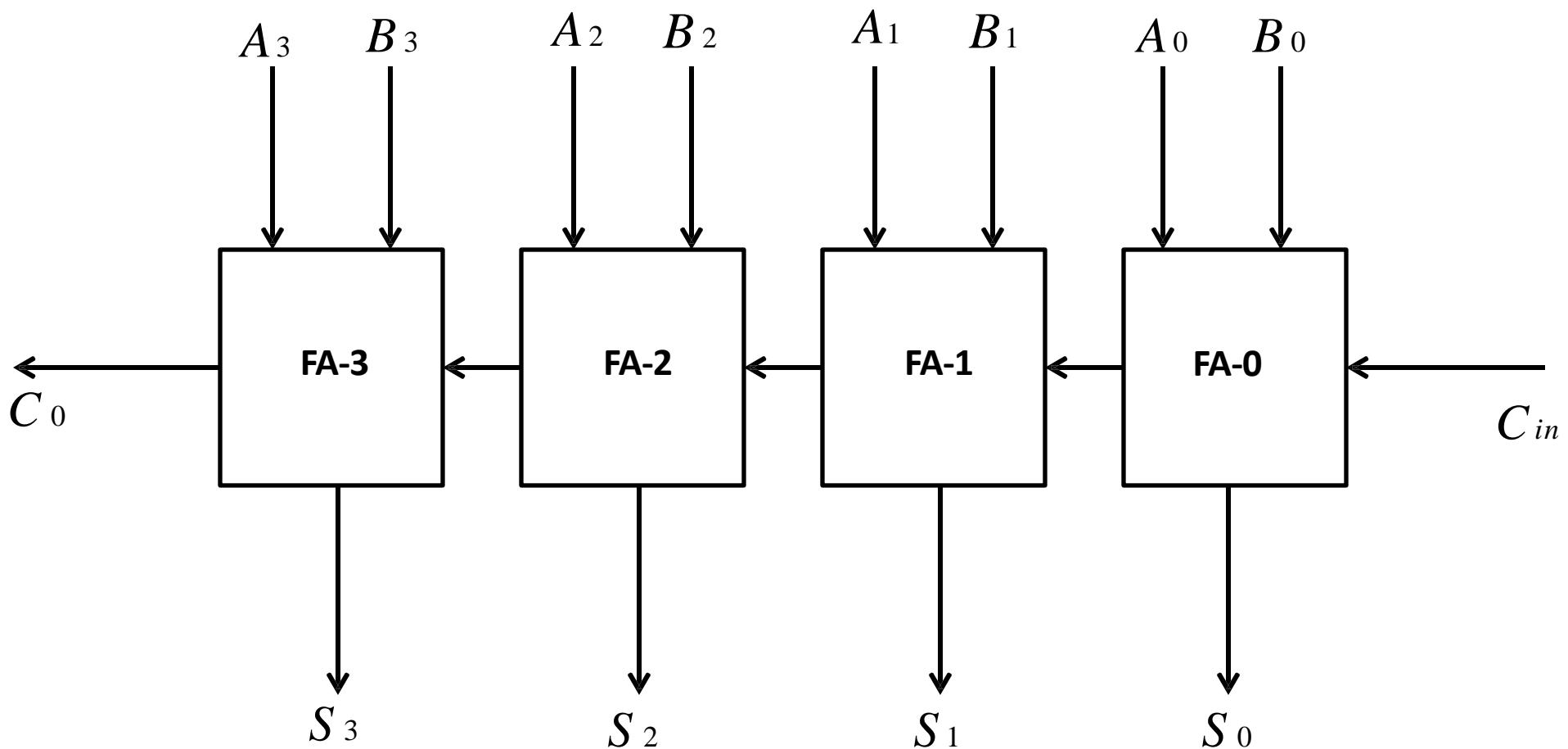
N – Bit Parallel Adder

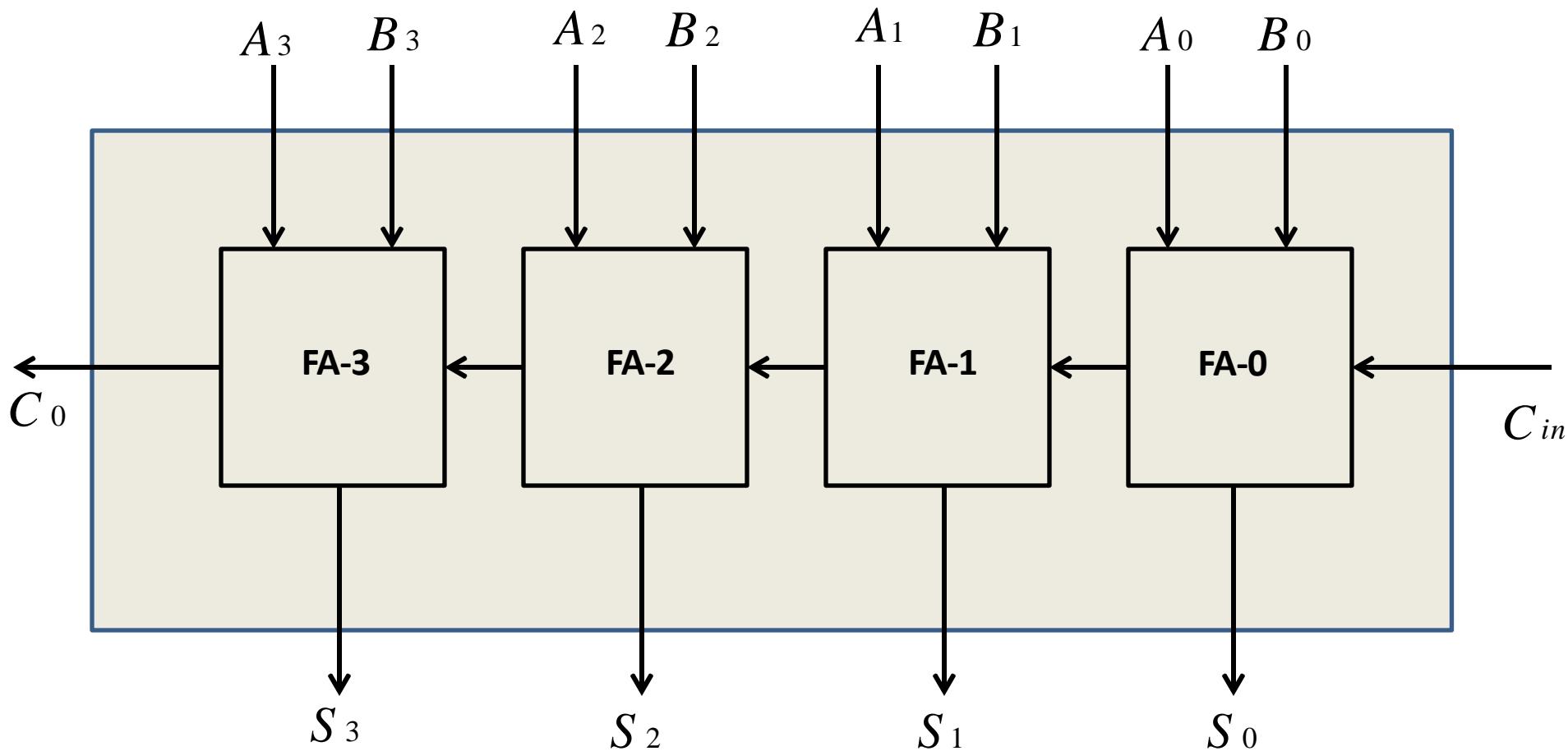
- ✓ The full adder is capable of adding two single digit binary numbers along with a carry input.
- ✓ But in practice we need to add binary numbers which are much longer than one bit.
- ✓ To add two n-bit binary numbers we need to use the n-bit parallel adder.
- ✓ It uses a number of full adders in cascade.
- ✓ The carry output of the previous full adder is connected to the carry input of the next full adder..

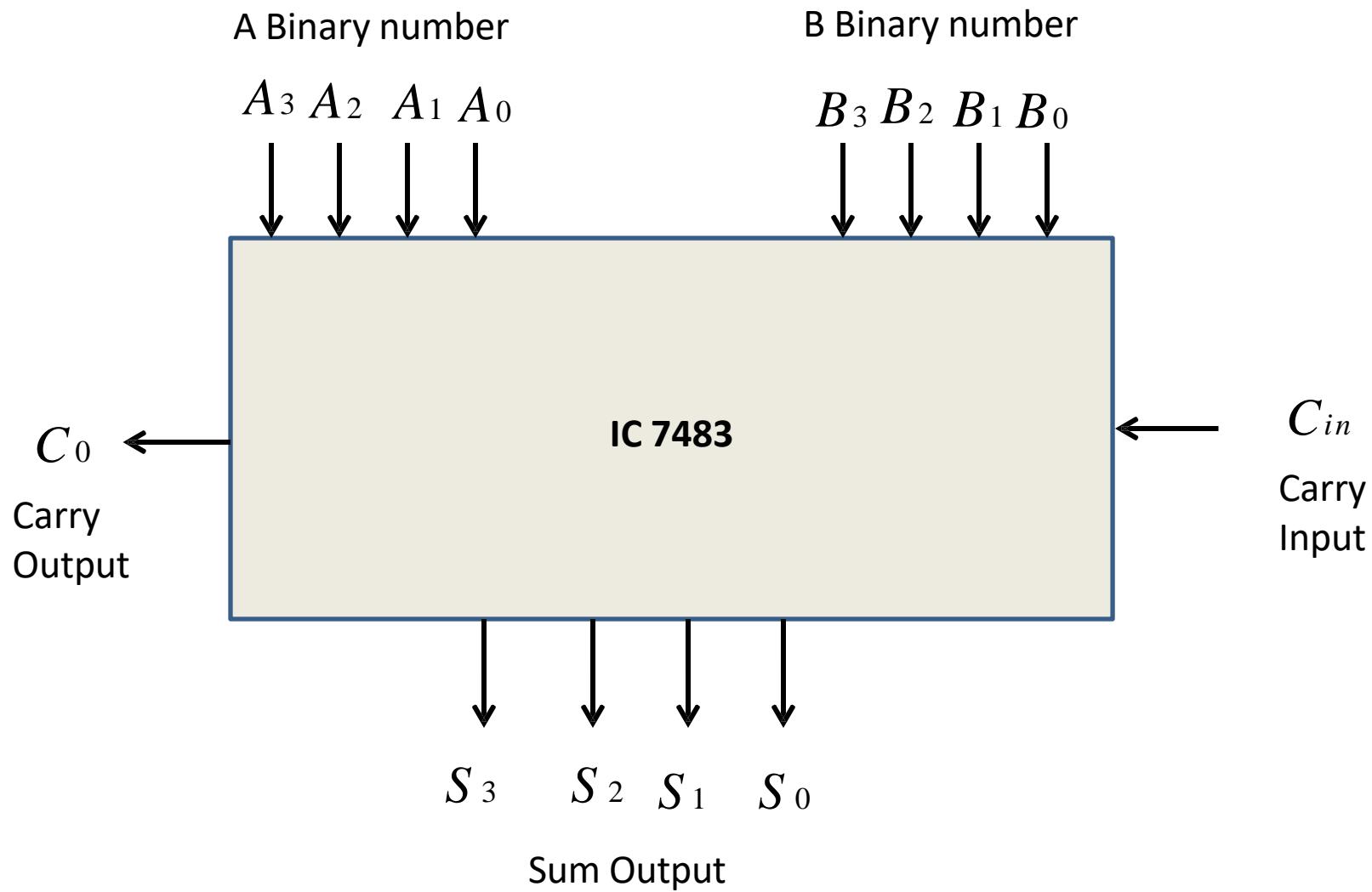
N – Bit Parallel Adder



4 – Bit Parallel Adder using full adder

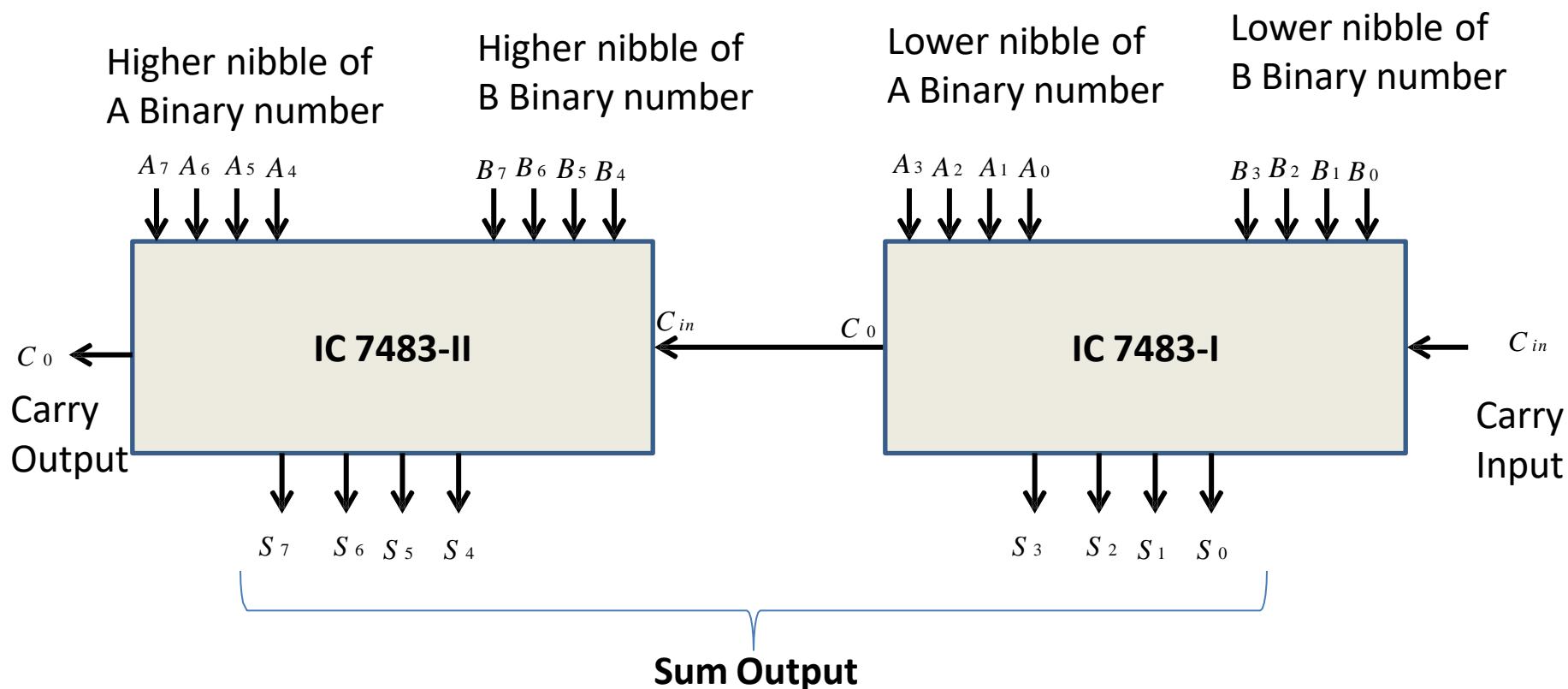






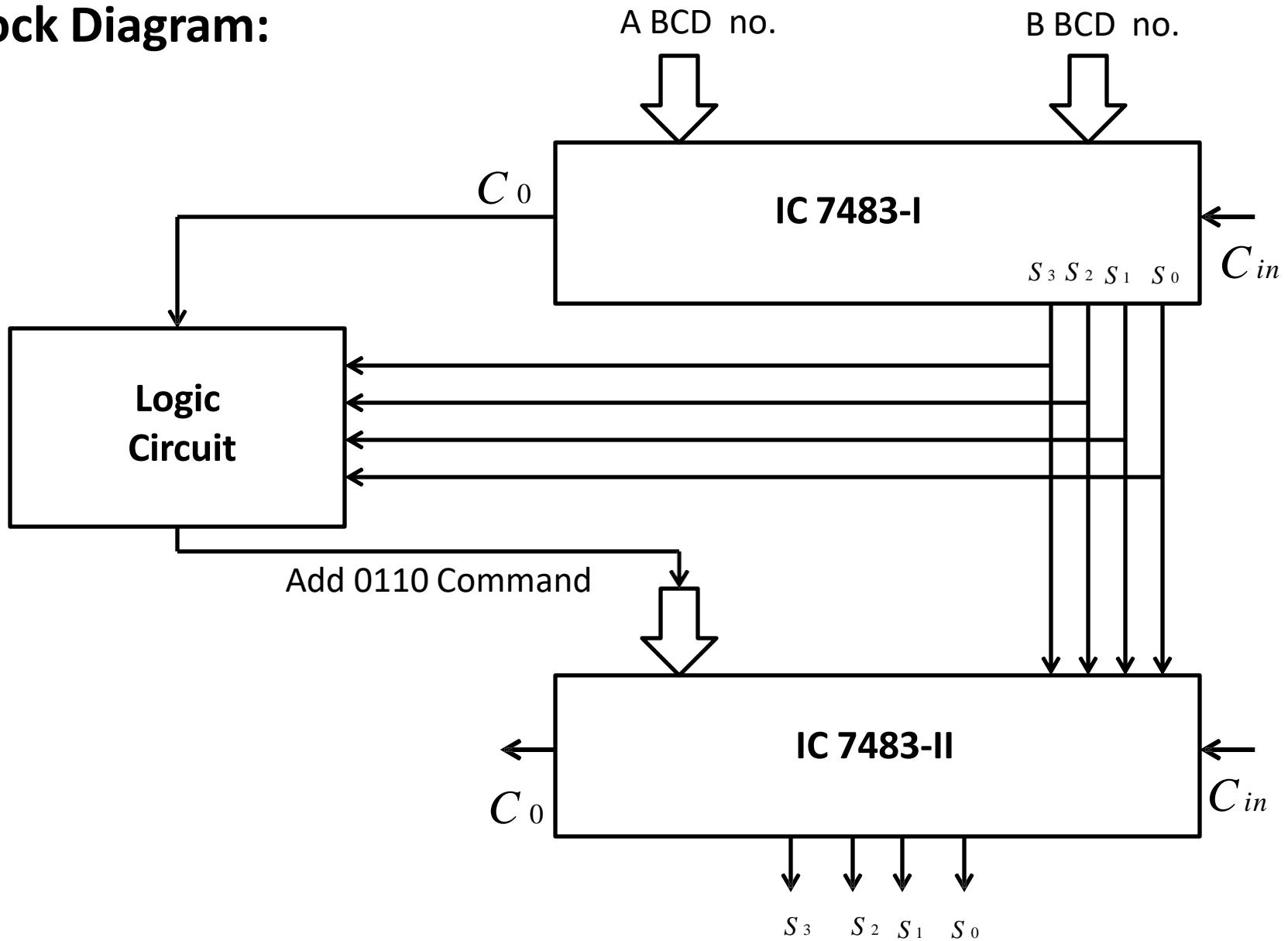
Cascading of IC 7483

- ✓ If we want to add two 8 bit binary numbers using 4 bit binary parallel adder IC 7483, then we have to cascade the two ICs in following way



Design of 1 Digit BCD Adder

Block Diagram:



Design of 1 Digit BCD Adder

As we know BCD addition rules, we understand that the 4 bit BCD adder should consists of following:

- ✓ A 4 bit binary adder to add the given two (4 bit numbers).
- ✓ A combinational logic circuit to check if sum is greater than 9 or carry 1.
- ✓ One more 4 bit binary adder to add 0110 to the invalid BCD sum or if carry is 1

Design of 1 Digit BCD Adder

Logic Table for design of Logic circuit:

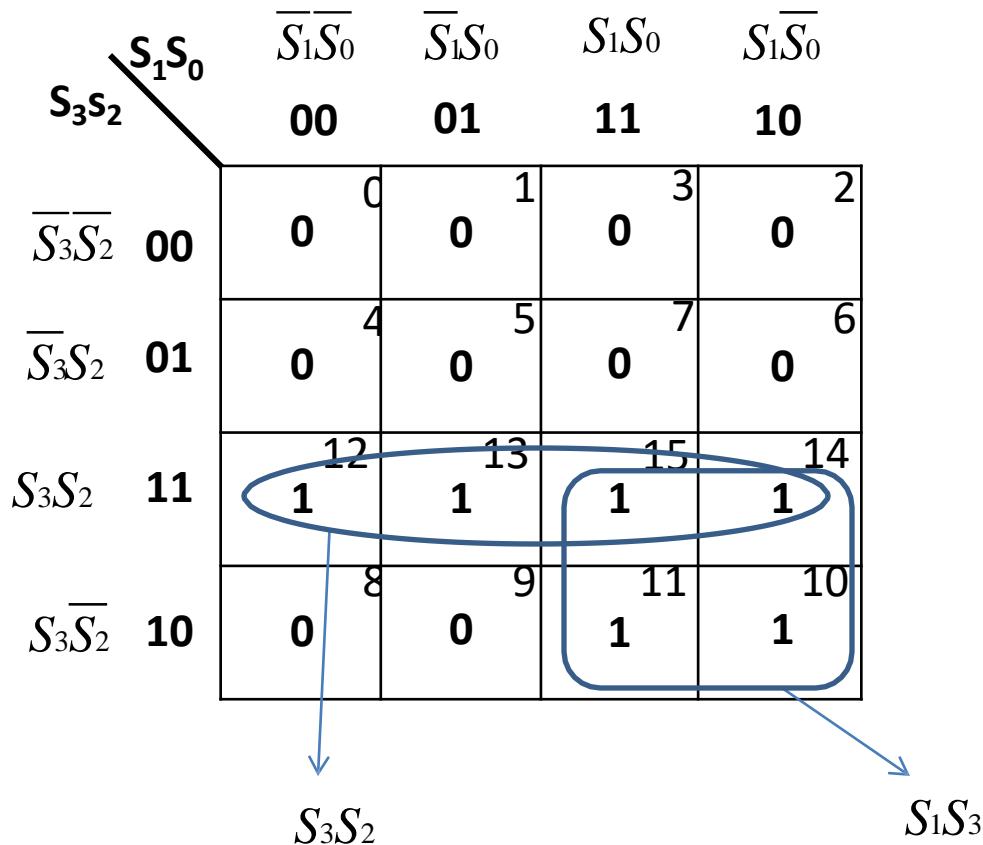
Inputs				Y
S_3	S_2	S_1	S_0	
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0

Inputs				Y
S_3	S_2	S_1	S_0	
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Sum is
invalid
BCD
Number
 $Y=1$

Design of 1 Digit BCD Adder

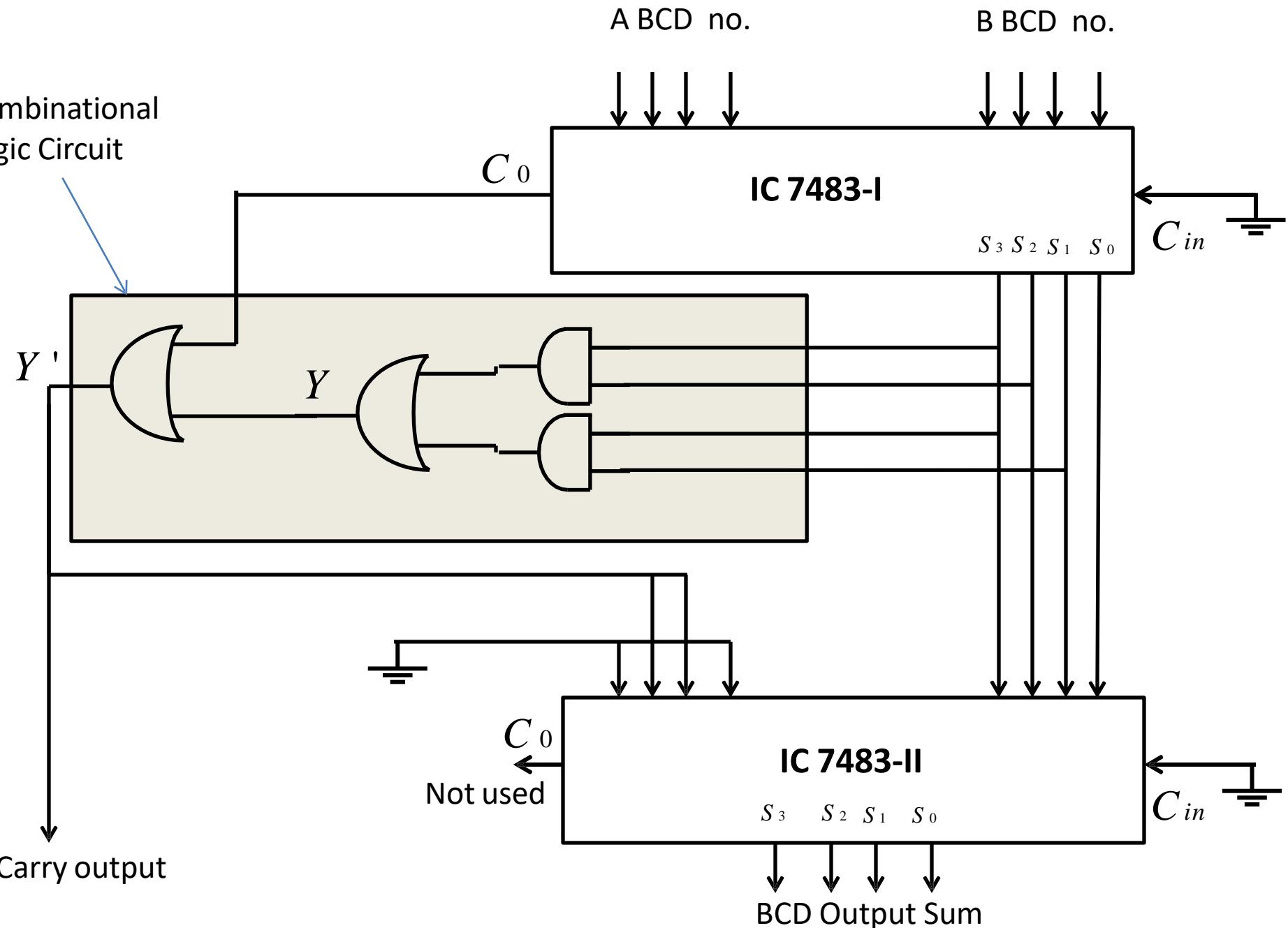
K-map for Logic circuit:



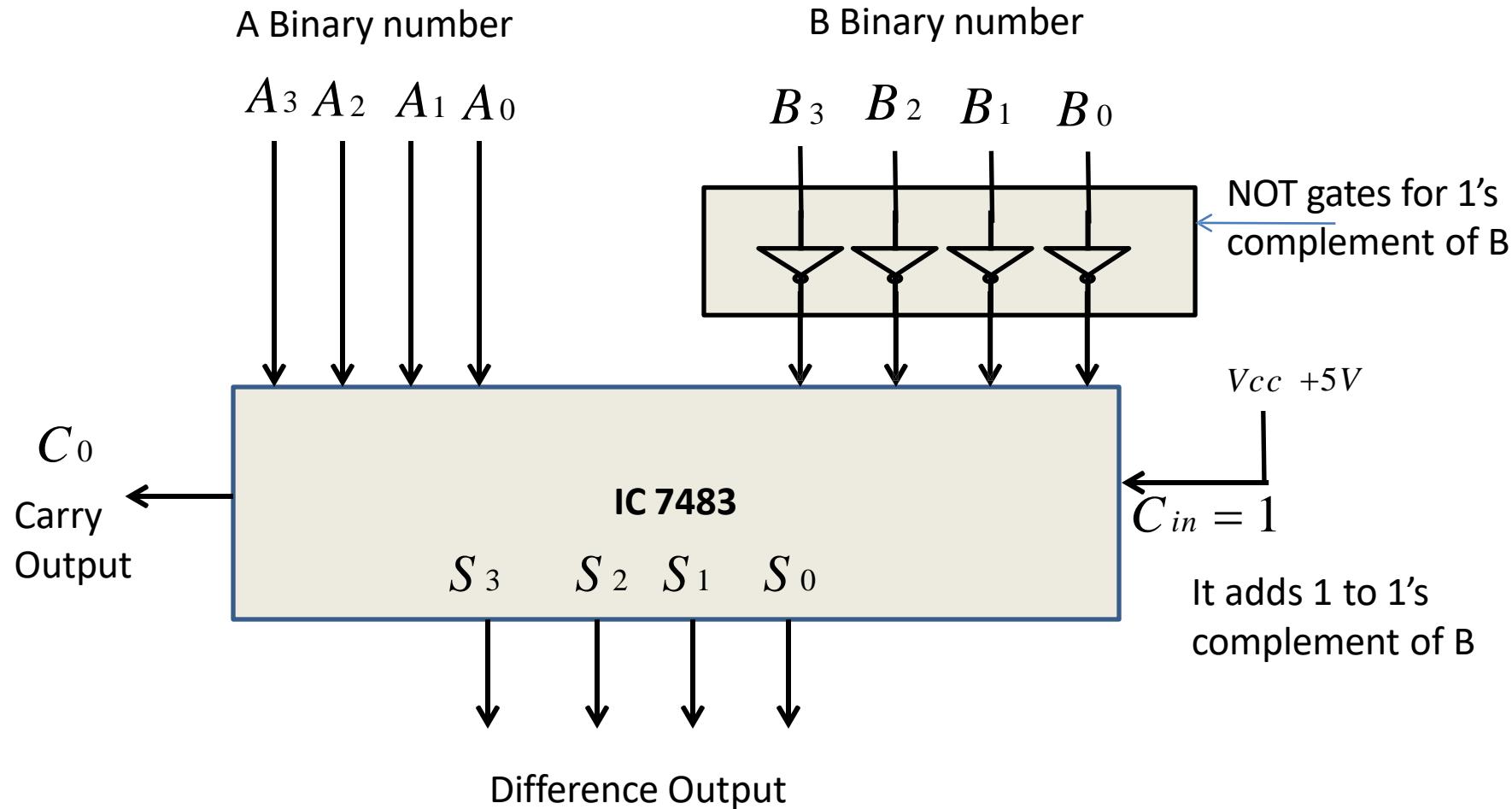
$$Y = S_3S_2 + S_3S_1$$

Design of 1 Digit BCD Adder

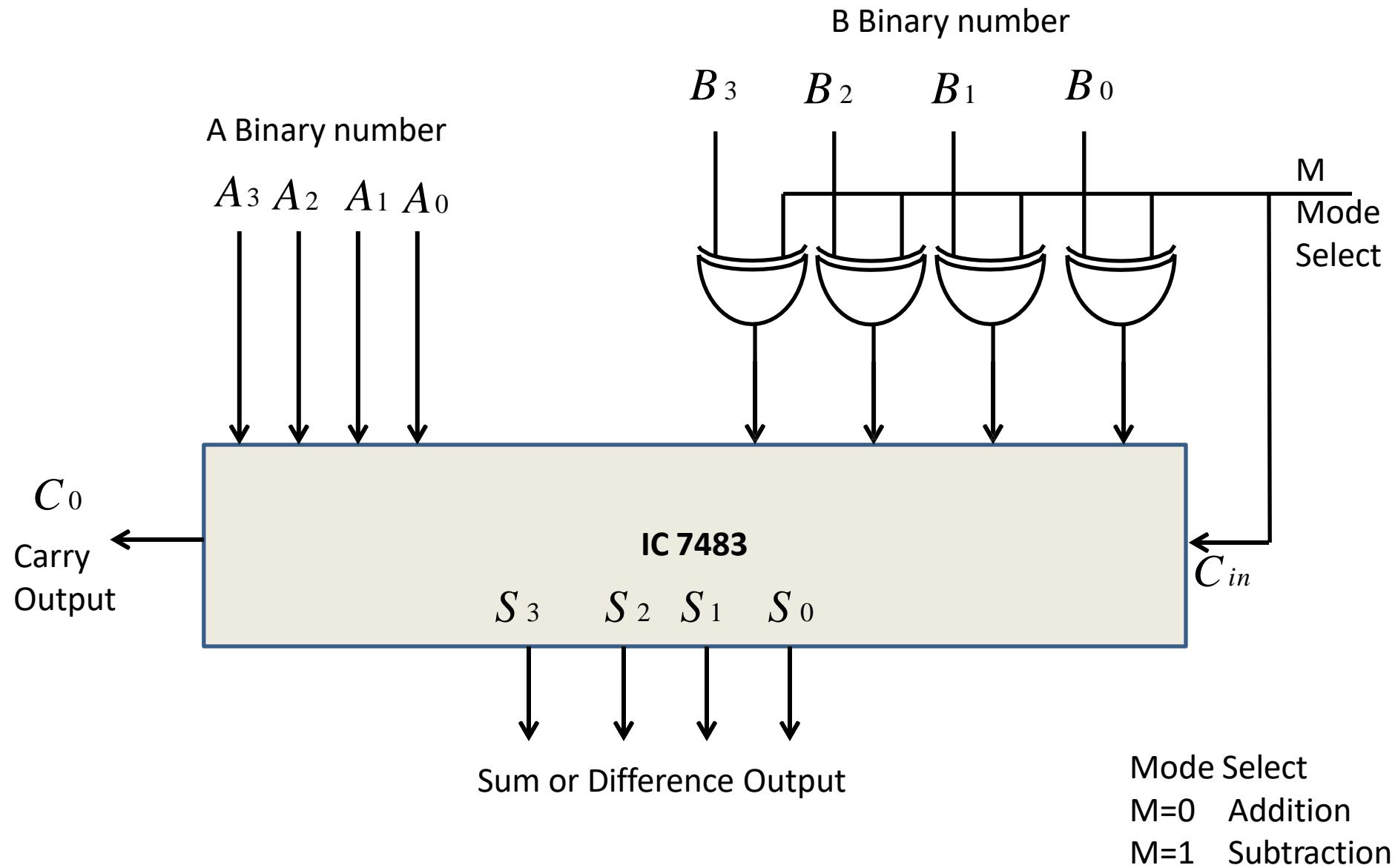
Combinational Logic Circuit



4 Bit Binary Parallel Subtractor using IC 7483



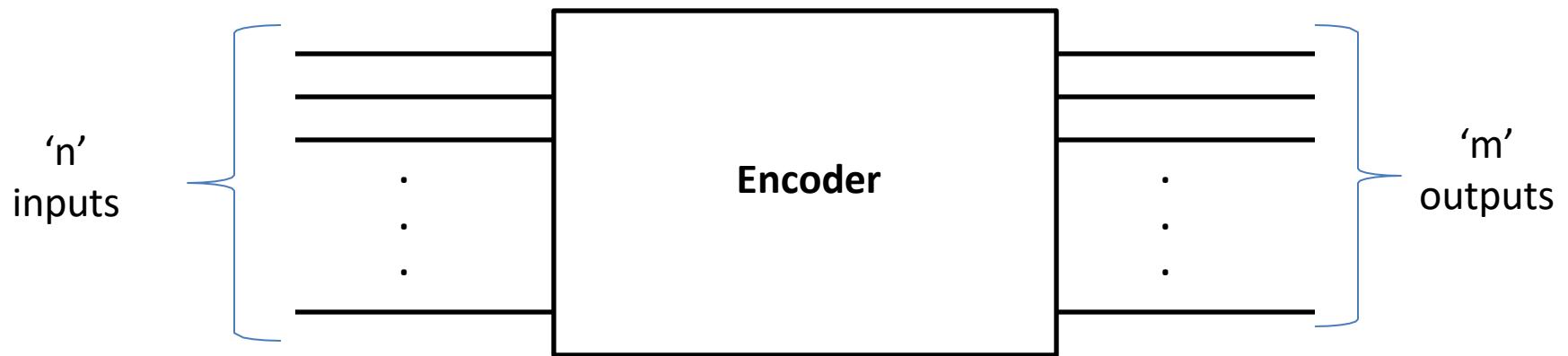
IC 7483 as Parallel Adder/Subtractor



Encoder

- ✓ Encoder is a combinational circuit which is designed to perform the inverse operation of decoder.
- ✓ An encoder has ‘n’ number of input lines and ‘m’ number of output lines.
- ✓ An encoder produces an m bit binary code corresponding to the digital input number.
- ✓ The encoder accepts an n input digital word and converts it into m bit another digital word

Encoder



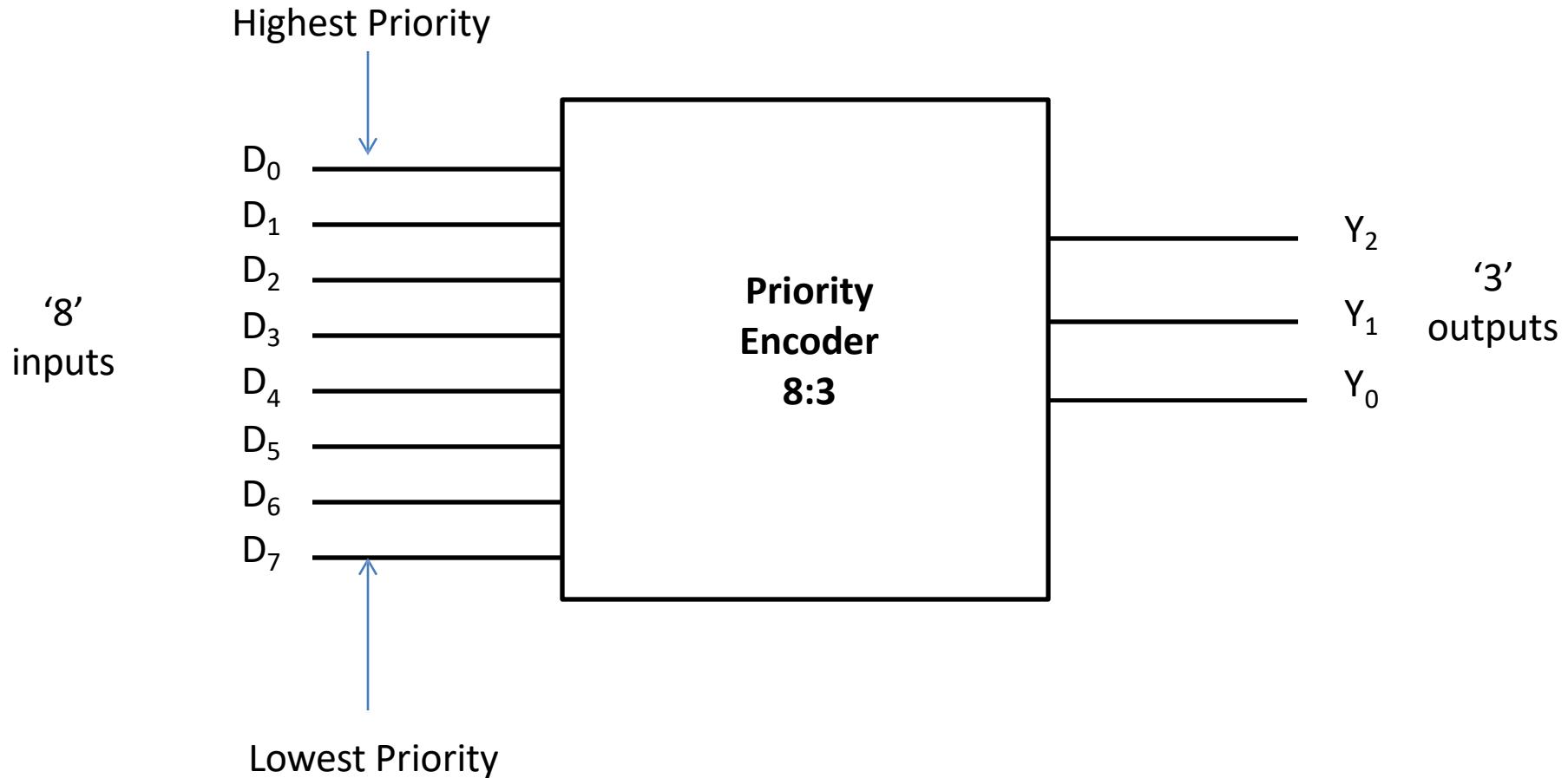
Types of Encoders

- ✓ Priority Encoder
- ✓ Decimal to BCD Encoder
- ✓ Octal to BCD Encoder
- ✓ Hexadecimal to Binary Encoder

Priority Encoder

- ✓ This is a special type of encoder.
- ✓ Priorities are given to the input lines.
- ✓ If two or more input lines are “1” at the same time, then the input line with highest priority will be considered.

Priority Encoder 8:3

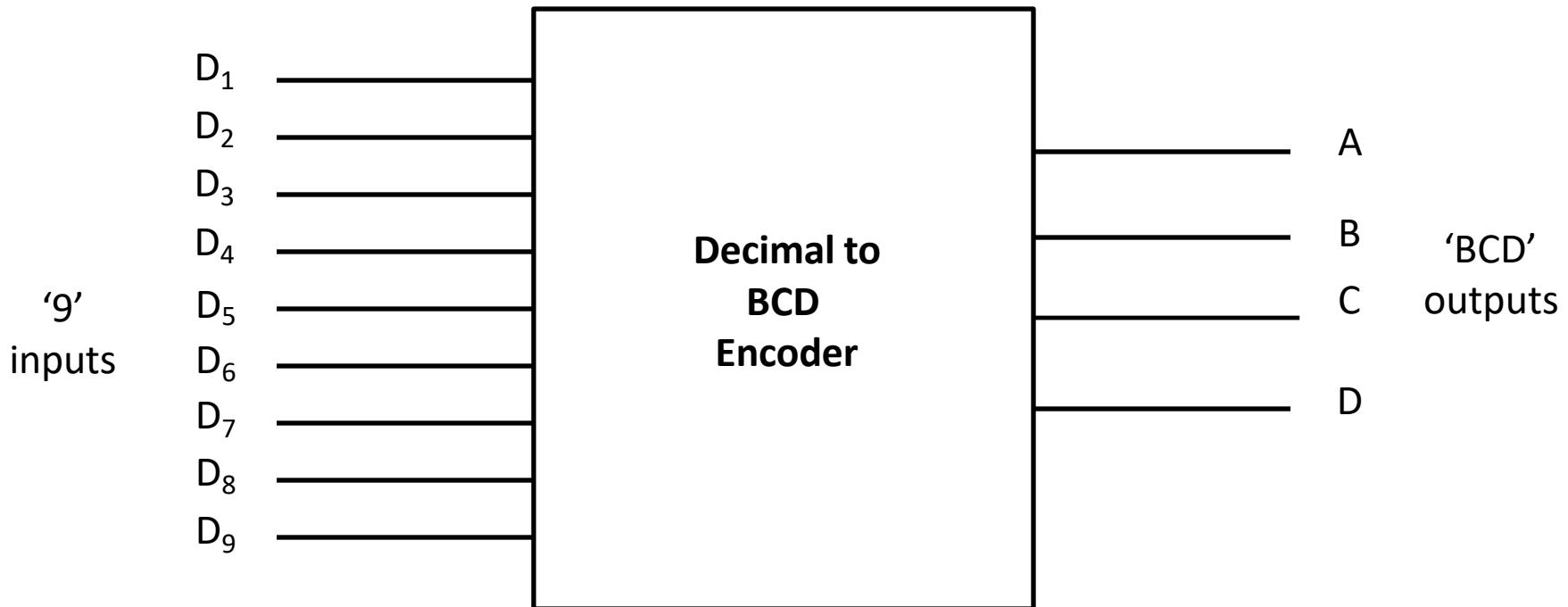


Priority Encoder 8:3

Truth Table:

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	X	X	X
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1

Decimal to BCD Encoder



Decimal to BCD Encoder

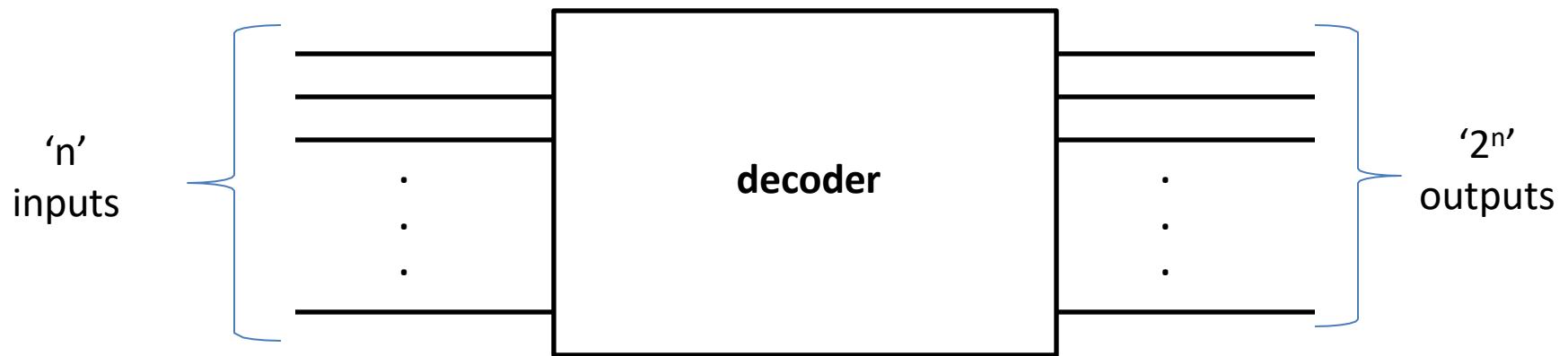
Truth Table:

Inputs										Outputs			
D ₉	D ₈	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁		D	C	B	A
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	X	0	0	1	0	0
0	0	0	0	0	0	1	X	X	0	0	1	1	1
0	0	0	0	0	1	X	X	X	0	1	0	0	0
0	0	0	0	1	X	X	X	X	0	1	0	0	1
0	0	0	1	X	X	X	X	X	0	1	1	0	0
0	0	1	X	X	X	X	X	X	0	1	1	1	1
0	1	X	X	X	X	X	X	X	1	0	0	0	0
1	X	X	X	X	X	X	X	X	1	0	0	0	1

Decoder

- ✓ Decoder is a combinational circuit which is designed to perform the inverse operation of encoder.
- ✓ An decoder has ‘n’ number of input lines and maximum ‘ 2^n ’ number of output lines.
- ✓ Decoder is identical to a demultiplexer without data input.

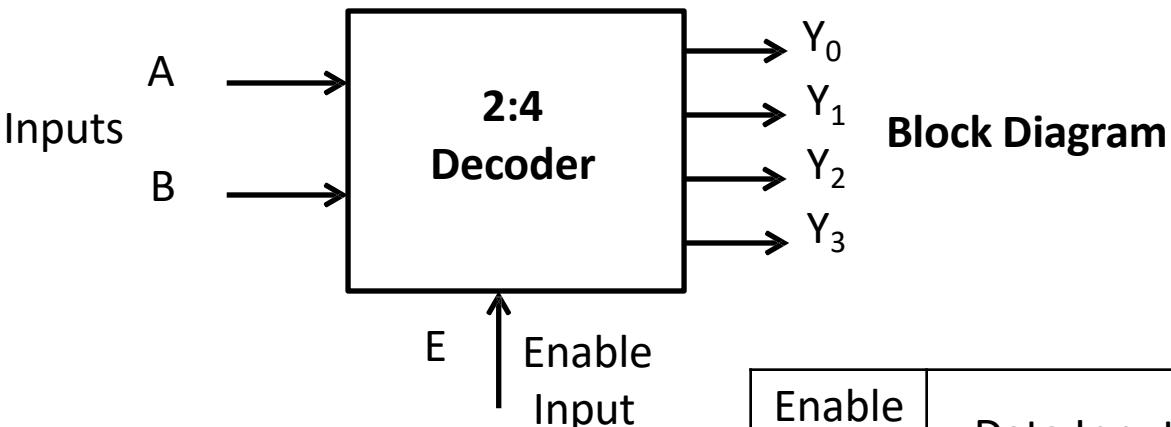
Decoder



Types of Decoders

- ✓ 2 to 4 line Decoder
- ✓ 3 to 8 line Decoder
- ✓ BCD to 7 Segment Decoder

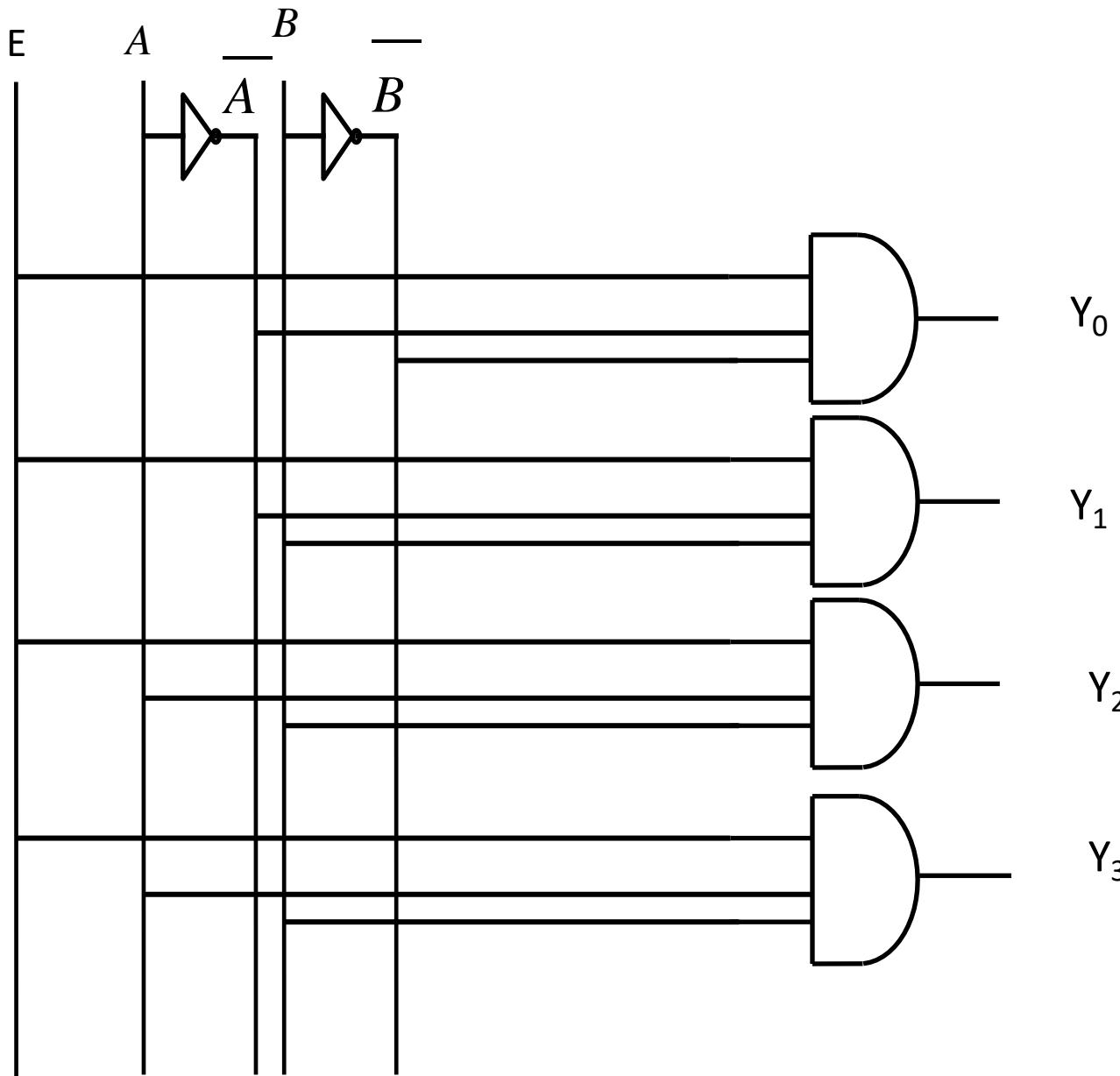
2 to 4 Line Decoder



Truth Table

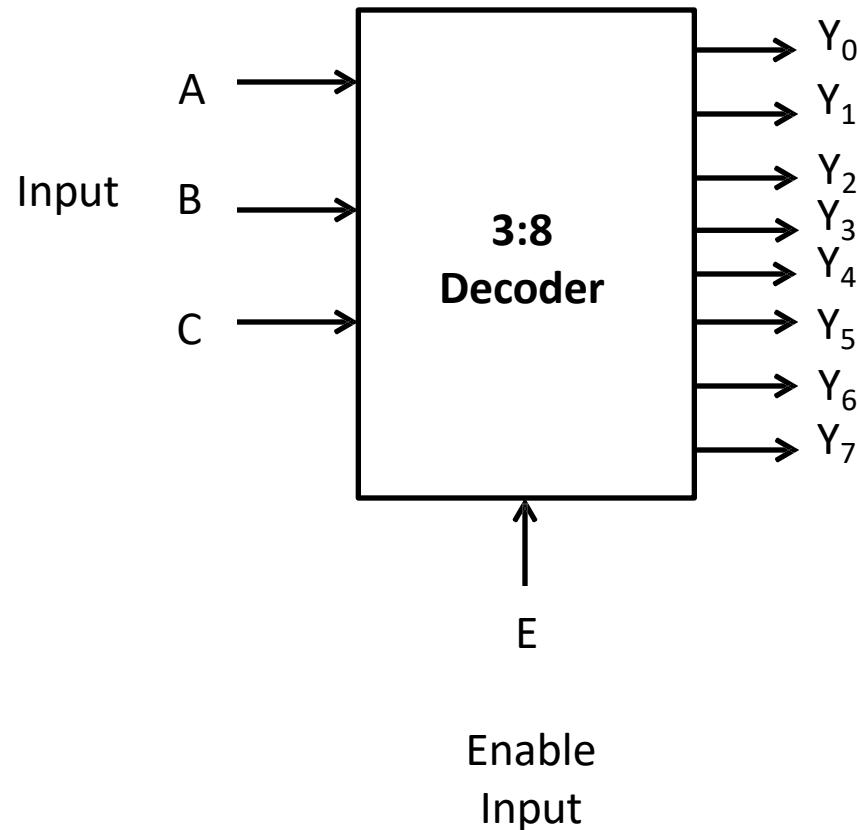
Enable i/p	Data Inputs		Outputs				
	E	A	B	Y_0	Y_1	Y_2	Y_3
0	X	X		0	0	0	0
1	0	0		1	0	0	0
1	0	1		0	1	0	0
1	1	0		0	0	1	0
1	1	1		0	0	0	1

2 to 4 Line Decoder



3 to 8 Line Decoder

Block Diagram



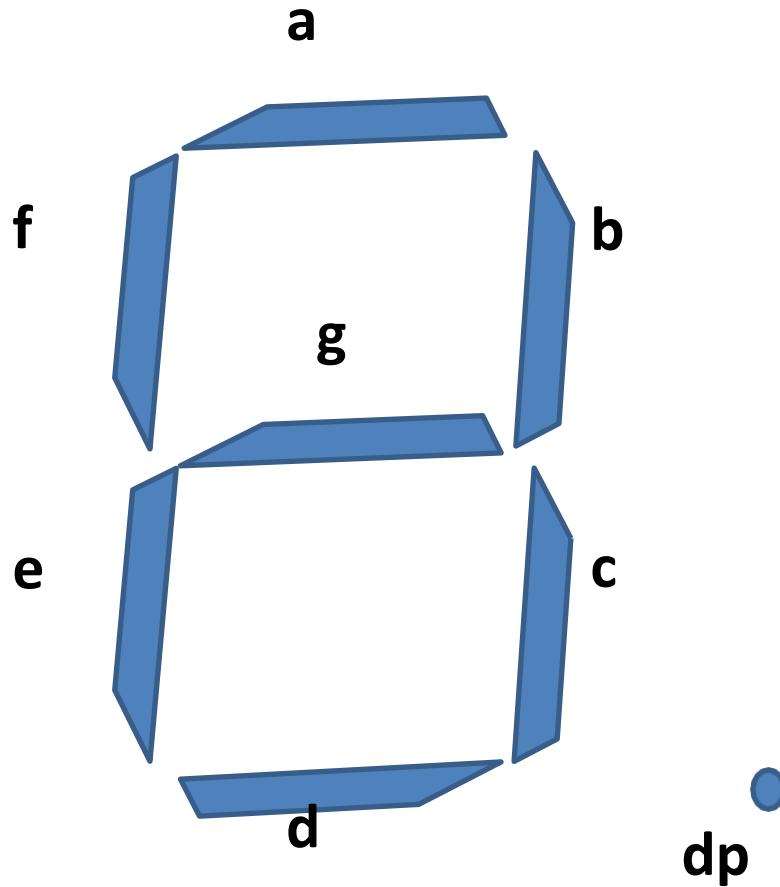
3 to 8 Line Decoder

Truth Table

Comparison between Encoder & Decoder

Sr. No.	Parameter	Encoder	Decoder
1	Input applied	Active input signal (original message signal)	Coded binary input
2	Output generated	Coded binary output	Active output signal (original message)
3	Input lines	2^n	n
4	Output lines	N	2^n
5	Operation	Simple	Complex
6	Applications	E-mail , video encoders etc.	Microprocessors, memory chips etc.

BCD to 7 Segment Decoder - Seven Segment Display



Seven Segment Display

Segments							Display Number	Seven Segment Display
a	b	c	d	e	f	g		
ON	ON	ON	ON	ON	ON	OFF	0	
OFF	ON	ON	OFF	OFF	OFF	OFF	1	
ON	ON	OFF	ON	ON	OFF	ON	2	
ON	ON	ON	ON	OFF	OFF	ON	3	
OFF	ON	ON	OFF	OFF	ON	ON	4	
ON	OFF	ON	ON	OFF	ON	ON	5	
ON	OFF	ON	ON	ON	ON	ON	6	
ON	ON	ON	OFF	OFF	OFF	OFF	7	
ON	ON	ON	ON	ON	ON	ON	8	
ON	ON	ON	ON	OFF	ON	ON	9	

Multiplexers

- ✓ Multiplexer is a circuit which has a number of inputs but only one output.
- ✓ Multiplexer is a circuit which transmits large number of information signals over a single line.
- ✓ Multiplexer is also known as “Data Selector” or MUX.

Necessity of Multiplexers

- ✓ In most of the electronic systems, the digital data is available on more than one lines. It is necessary to route this data over a single line.
- ✓ Under such circumstances we require a circuit which select one of the many inputs at a time.
- ✓ This circuit is nothing but a multiplexer. Which has many inputs, one output and some select lines.
- ✓ Multiplexer improves the reliability of the digital system because it reduces the number of external wired connections.

Advantages of Multiplexers

- ✓ It reduces the number of wires.
- ✓ So it reduces the circuit complexity and cost.
- ✓ We can implement many combinational circuits using Mux.
- ✓ It simplifies the logic design.
- ✓ It does not need the k-map and simplification.

Applications of Multiplexers

- ✓ It is used as a data selector to select one out of many data inputs.
- ✓ It is used for simplification of logic design.
- ✓ It is used in data acquisition system.
- ✓ In designing the combinational circuits.
- ✓ In D to A converters.
- ✓ To minimize the number of connections.

Block Diagram of Multiplexer

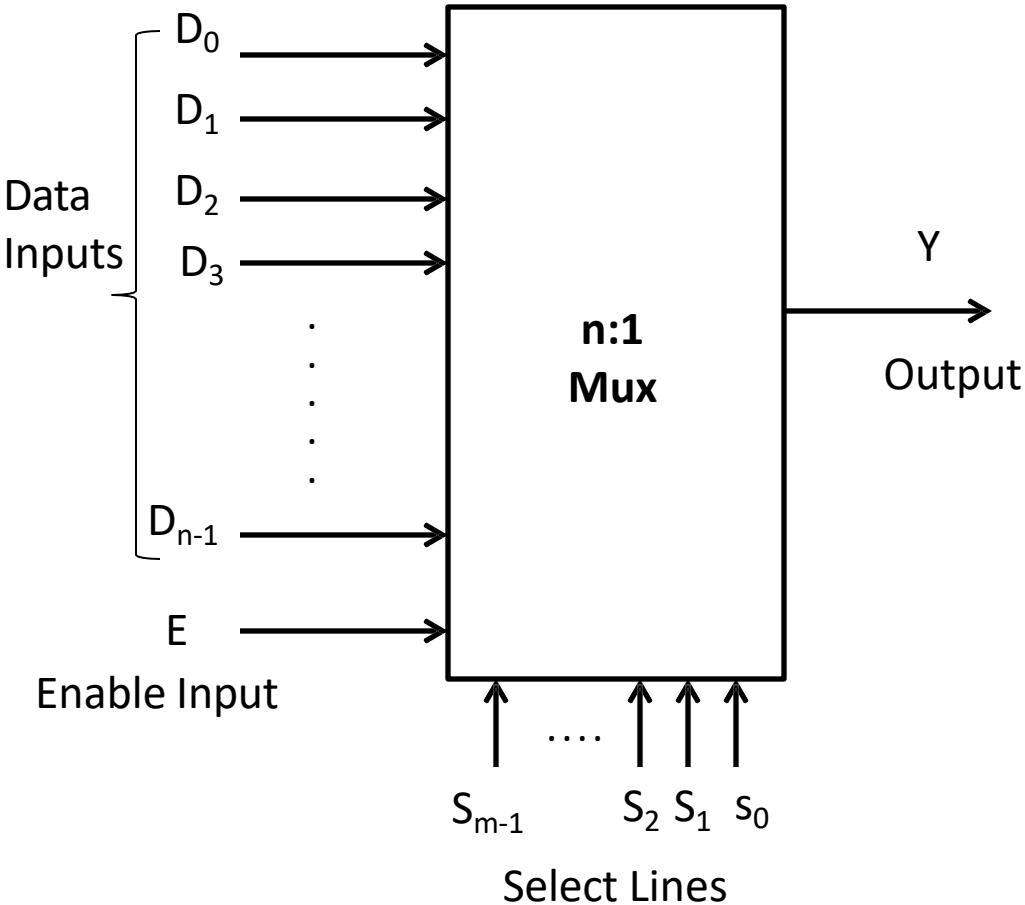


Fig. General Block Diagram

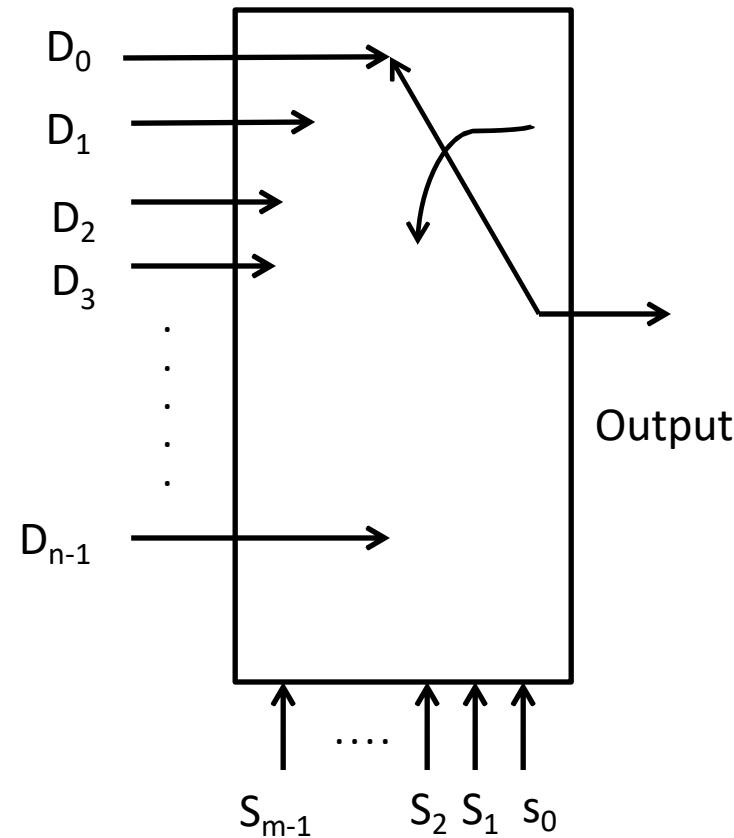


Fig. Equivalent Circuit

Relation between Data Input Lines & Select Lines

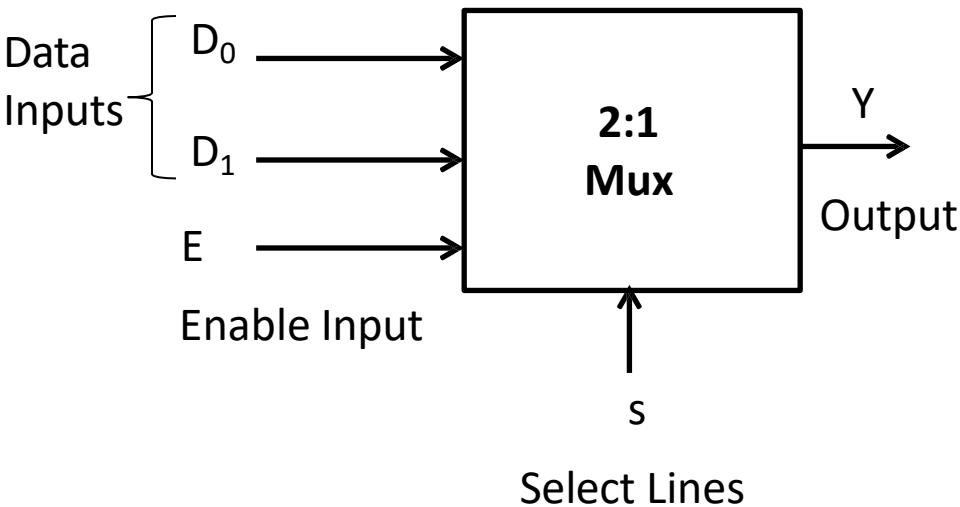
- ✓ In general multiplexer contains , n data lines, one output line and m select lines.
- ✓ To select n inputs we need m select lines such that $2^m=n$.

Types of Multiplexers

- ✓ 2:1 Multiplexer
- ✓ 4:1 Multiplexer
- ✓ 8:1 Multiplexer
- ✓ 16:1 Multiplexer
- ✓ 32:1 Multiplexer
- ✓ 64:1 Multiplexer

and so on.....

2:1 Multiplexer



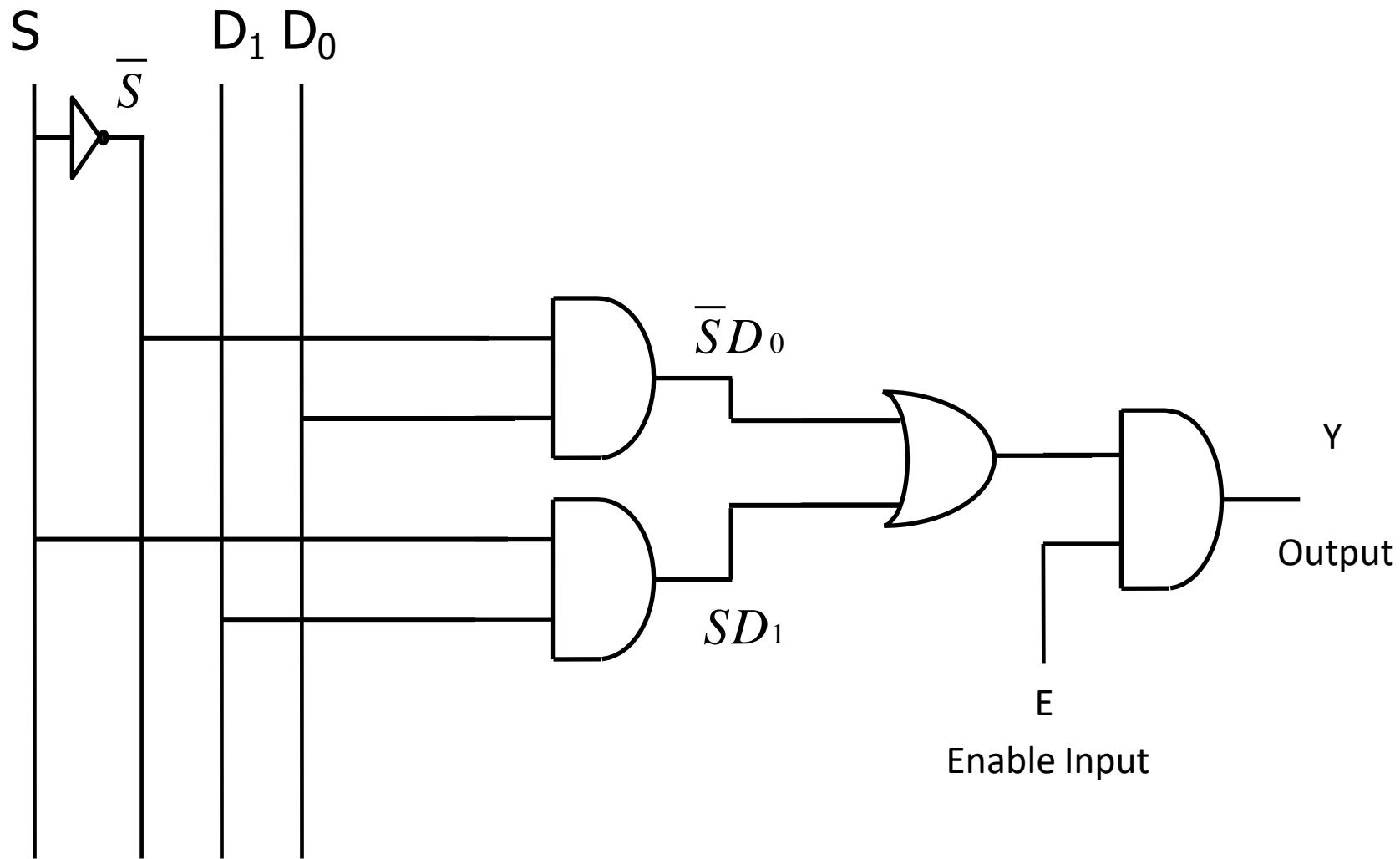
Block Diagram

Select Lines

Truth Table

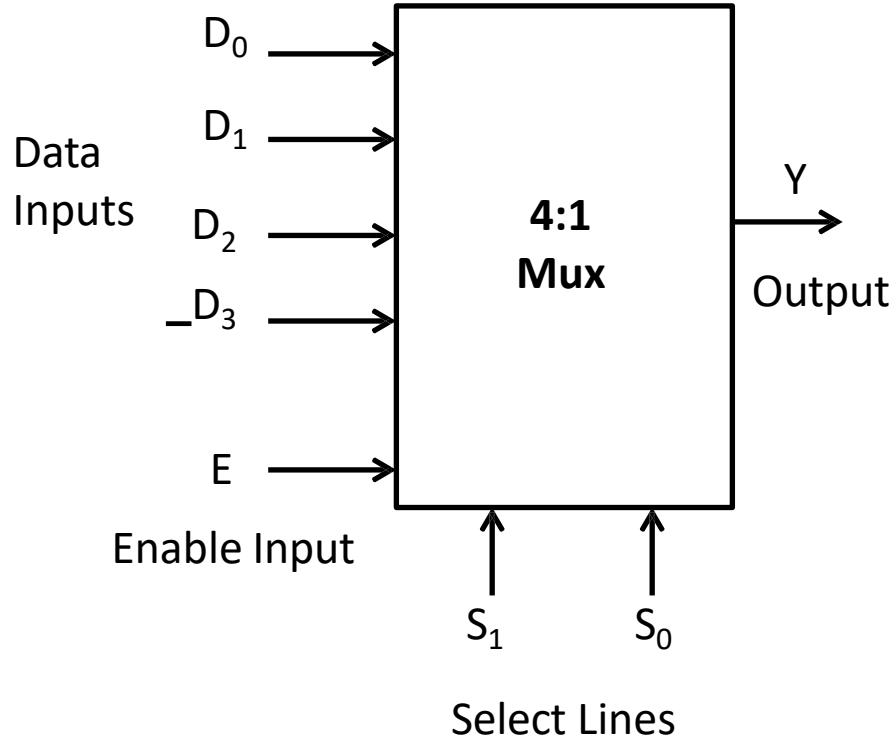
Enable i/p (E)	Select i/p (S)	Output (Y)
0	X	0
1	0	D_0
1	1	D_1

Realization of 2:1 Mux using gates



4:1 Multiplexer

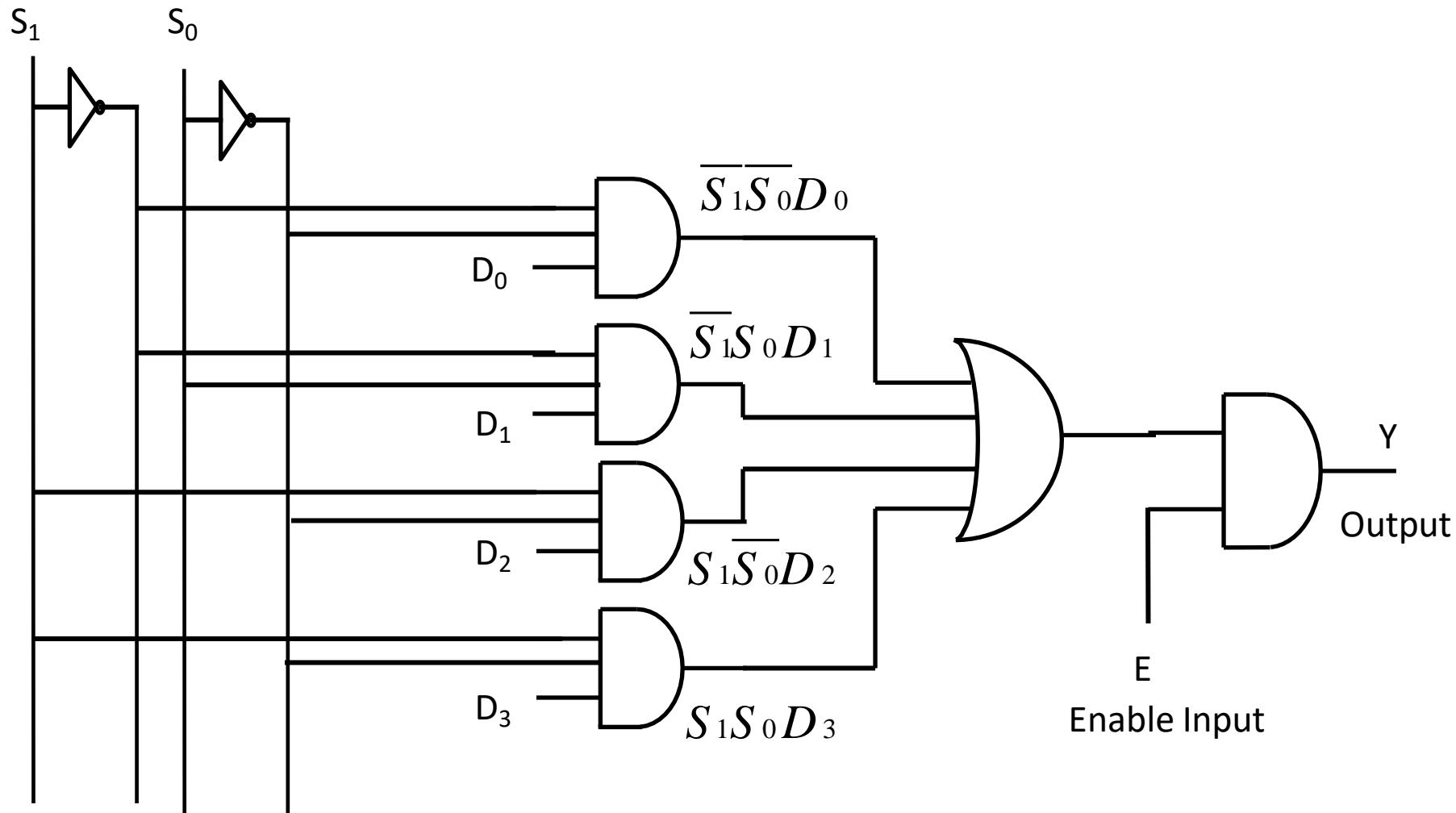
Block Diagram



Truth Table

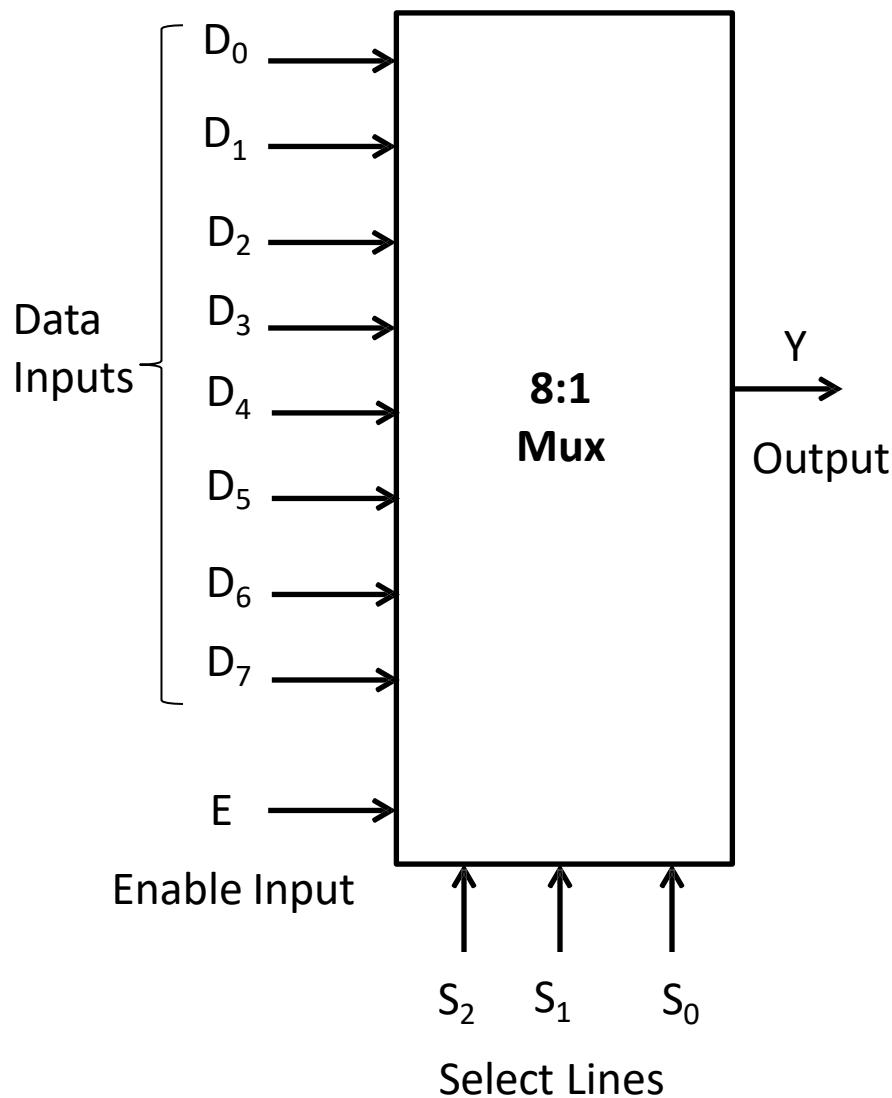
Enable i/p	Select i/p		Output
E	S_1	S_0	Y
0	X	X	0
1	0	0	D_0
1	0	1	D_1
1	1	0	D_2
1	1	1	D_3

Realization of 4:1 Mux using gates



8:1 Multiplexer

Block Diagram

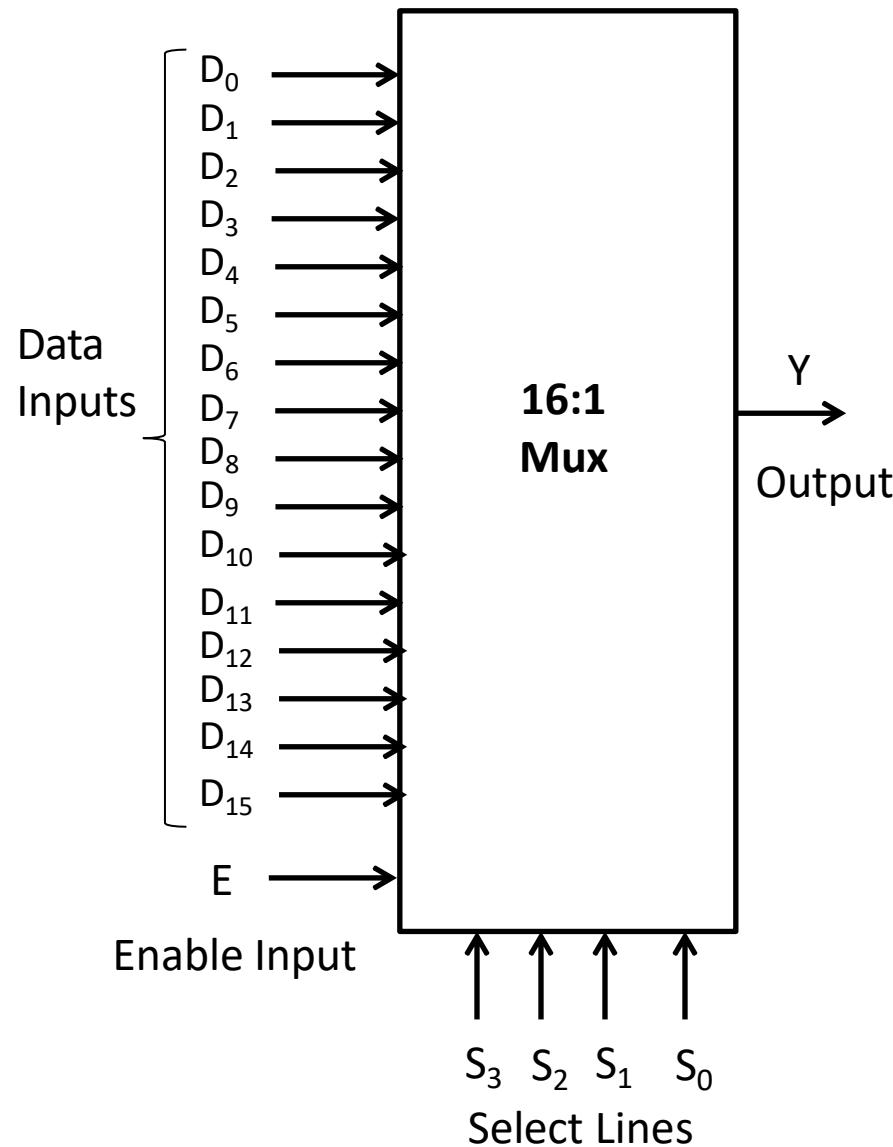


Truth Table

Enable i/p	Select i/p			Output
E	S_2	S_1	S_0	Y
0	X	X	X	0
1	0	0	0	D_0
1	0	0	1	D_1
1	0	1	0	D_2
1	0	1	1	D_3
1	1	0	0	D_4
1	1	0	1	D_5
1	1	1	0	D_6
1	1	1	1	D_7

16:1 Multiplexer

Block Diagram



16:1 Multiplexer

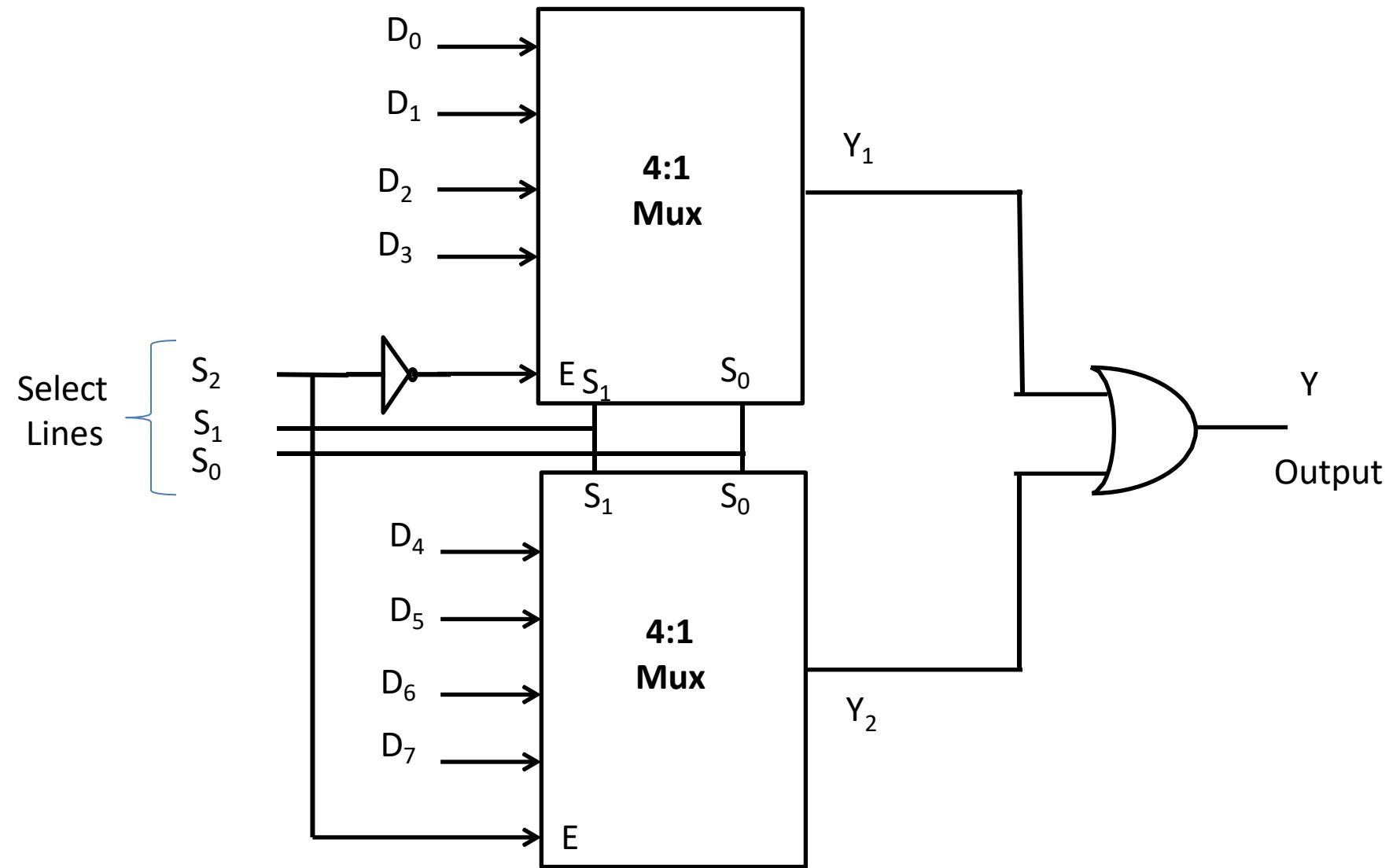
Truth Table

Enable	Select Lines				Output
	S_3	S_2	S_1	S_0	
E	X	X	X	X	Y
0	X	X	X	X	0
1	0	0	0	0	D_0
1	0	0	0	1	D_1
1	0	0	1	0	D_2
1	0	0	1	1	D_3
1	0	1	0	0	D_4
1	0	1	0	1	D_5
1	0	1	1	0	D_6
1	0	1	1	1	D_7
1	1	0	0	0	D_8
1	1	0	0	1	D_9
1	1	0	1	0	D_{10}
1	1	0	1	1	D_{11}
1	1	1	0	0	D_{12}
1	1	1	0	1	D_{13}
1	1	1	1	0	D_{14}
1	1	1	1	1	D_{15}

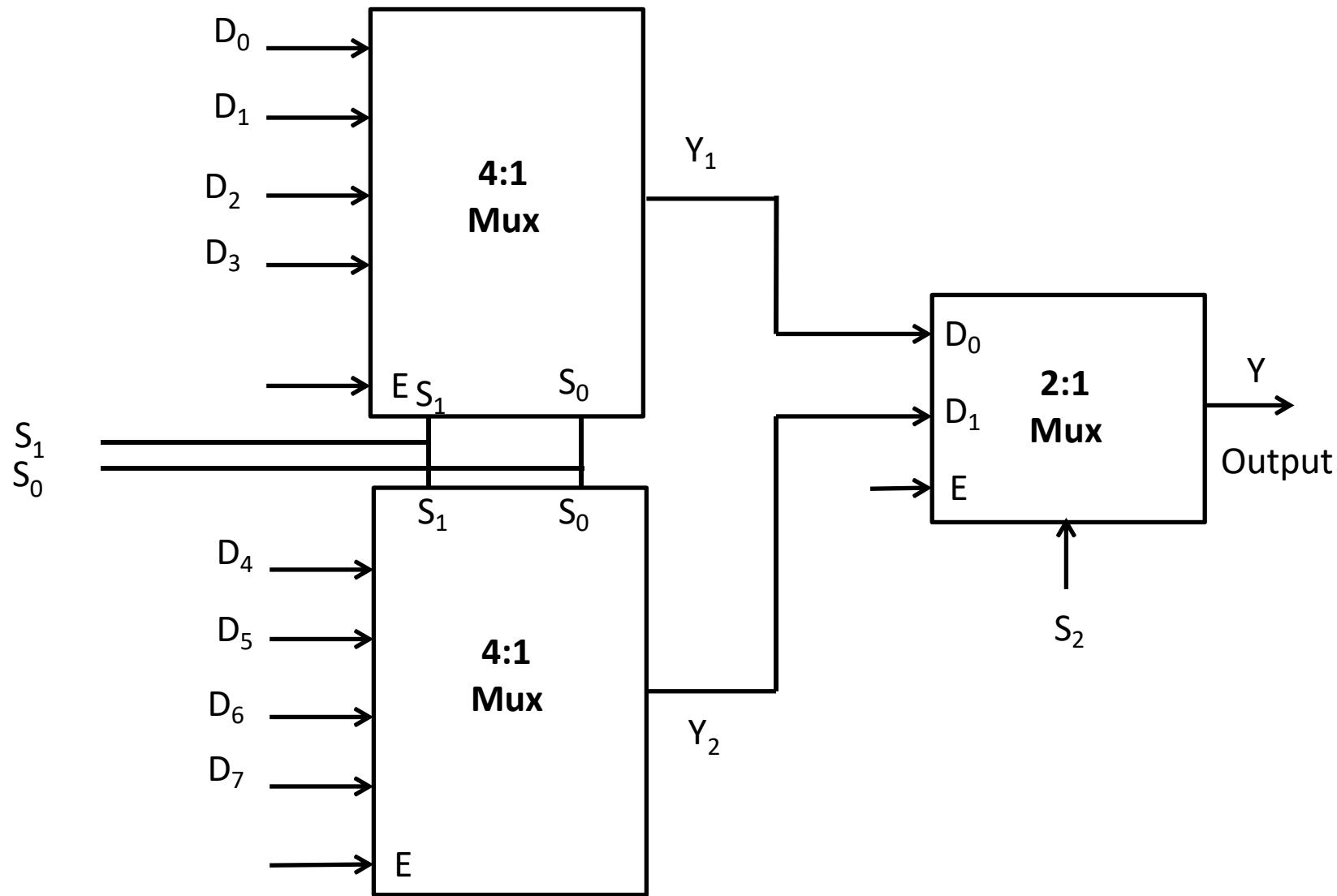
Mux Tree

- ✓ The multiplexers having more number of inputs can be obtained by cascading two or more multiplexers with less number of inputs. This is called as Multiplexer Tree.
- ✓ For example, 32:1 mux can be realized using two 16:1 mux and one 2:1 mux.

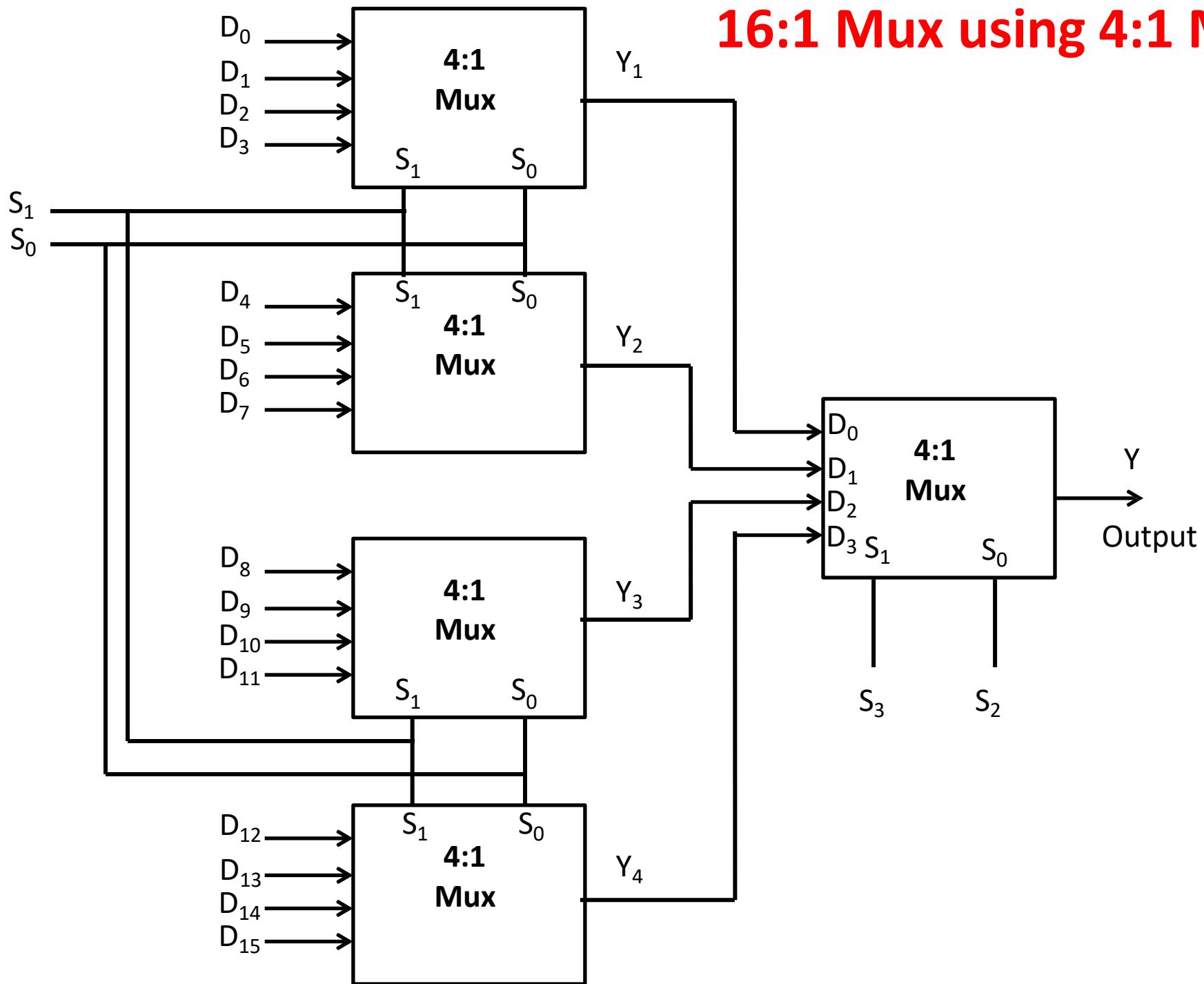
8:1 Multiplexer using 4:1 Multiplexer



8:1 Multiplexer using 4:1 Multiplexer



16:1 Mux using 4:1 Mux



Realization of Boolean expression using Mux

- ✓ We can implement any Boolean expression using Multiplexers.
- ✓ It reduces circuit complexity.
- ✓ It does not require any simplification

Example 1

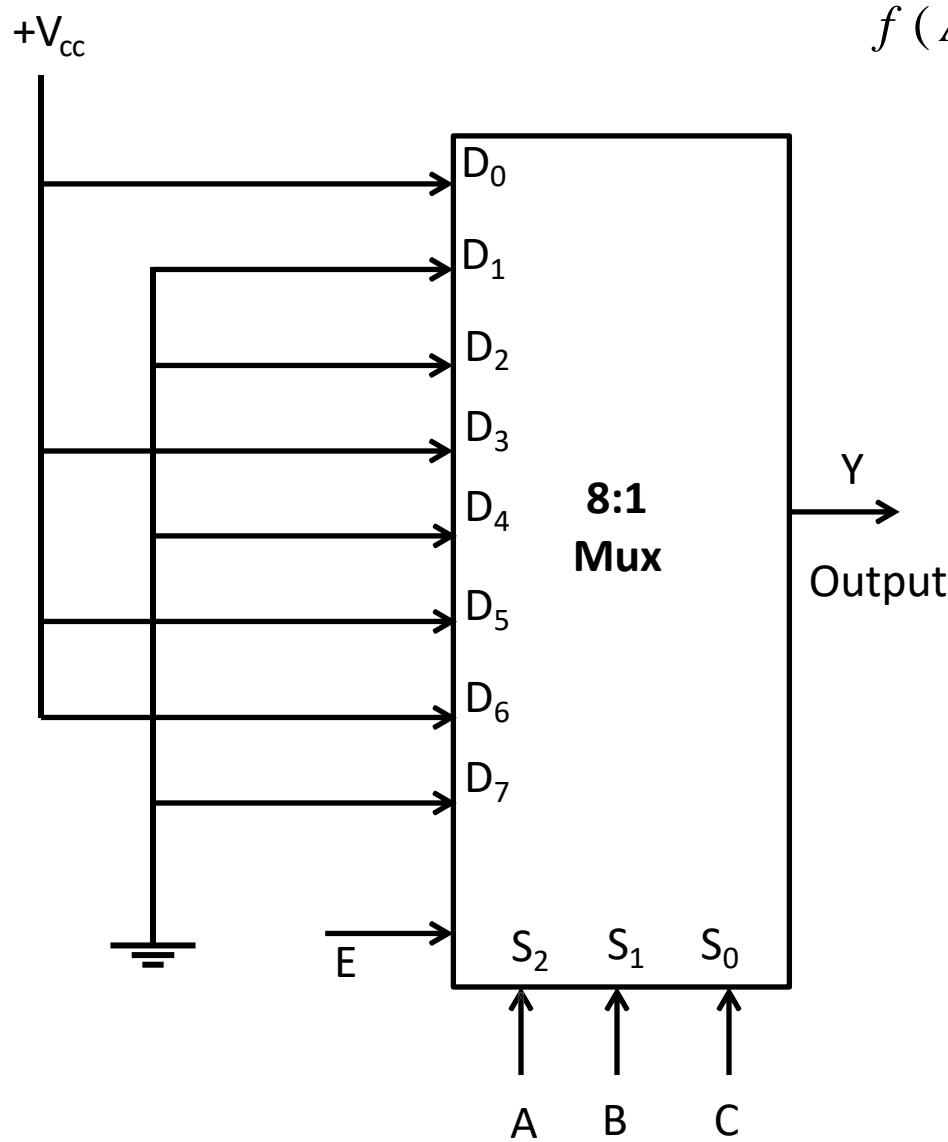
Implement following Boolean expression using multiplexer

$$f(A, B, C) = \sum m(0, 3, 5, 6)$$

- ✓ Since there are three variables, therefore a multiplexer with three select input is required i.e. 8:1 multiplexer is required
- ✓ The 8:1 multiplexer is configured as below to implement given Boolean expression

Example 1

continue.....



$$f(A, B, C) = \sum m(0, 3, 5, 6)$$

Example 2

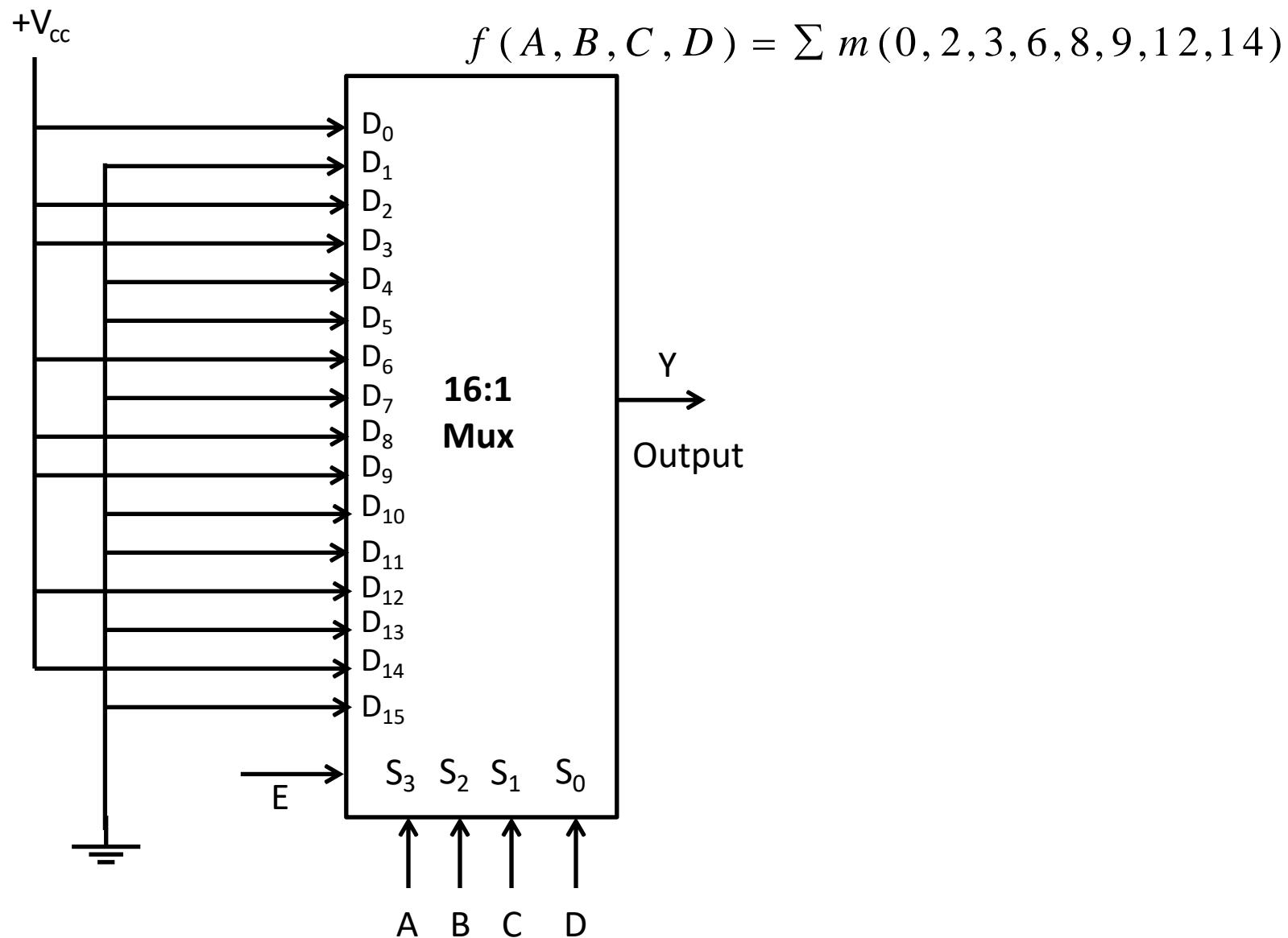
Implement following Boolean expression using multiplexer

$$f(A, B, C, D) = \sum m(0, 2, 3, 6, 8, 9, 12, 14)$$

- ✓ Since there are four variables, therefore a multiplexer with four select input is required i.e. 16:1 multiplexer is required
- ✓ The 16:1 multiplexer is configured as below to implement given Boolean expression

Example 2

continue.....



De-multiplexer

- ✓ A de-multiplexer performs the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs.
- ✓ At a time only one output line is selected by the select lines and the input is transmitted to the selected output line.
- ✓ It has only one input line, n number of output lines and m number of select lines.

Block Diagram of De-multiplexer

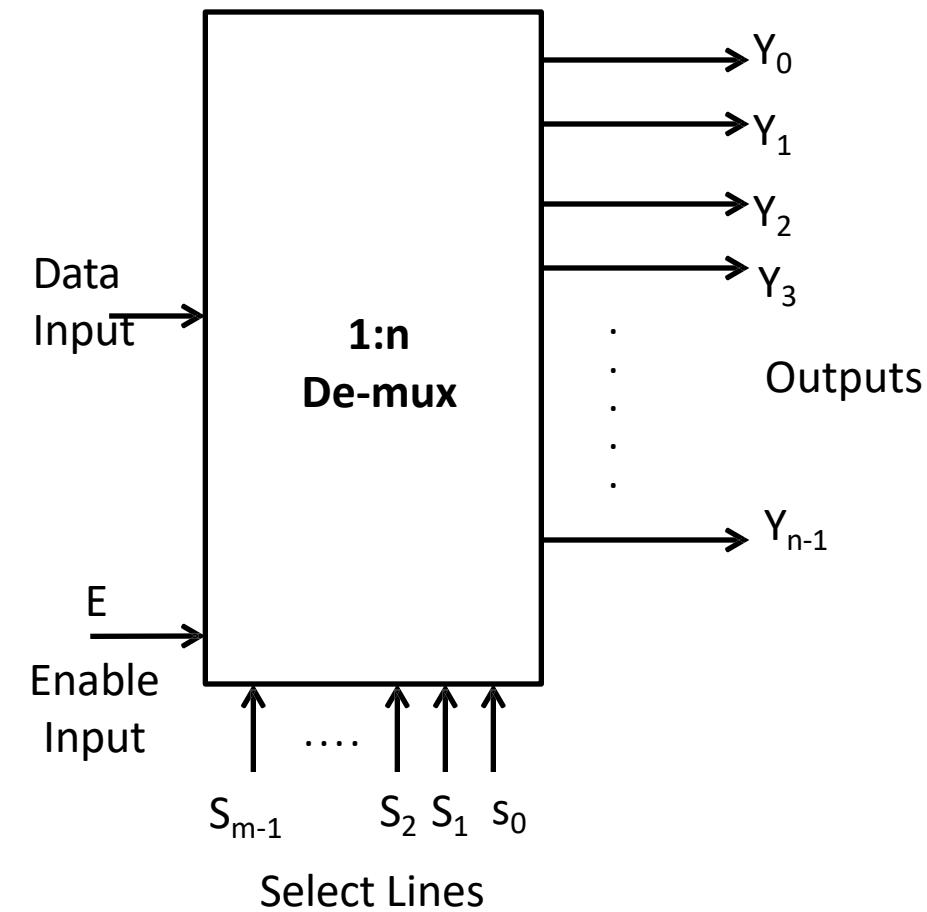


Fig. General Block Diagram

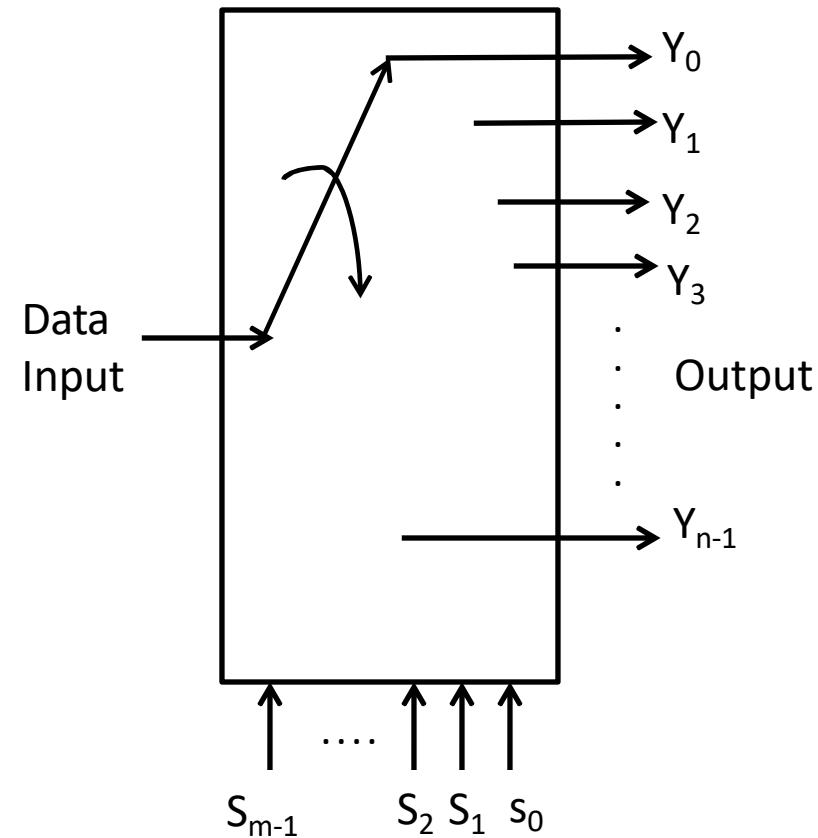


Fig. Equivalent Circuit

Relation between Data Output Lines & Select Lines

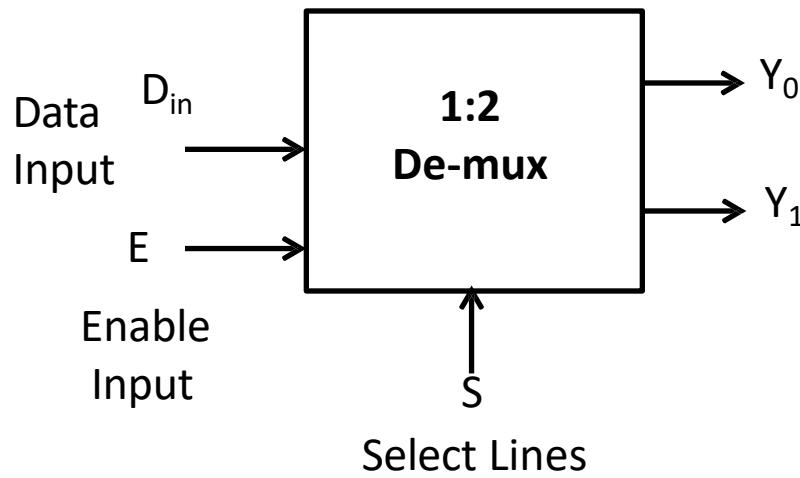
- ✓ In general de-multiplexer contains , n output lines, one input line and m select lines.
- ✓ To select n outputs we need m select lines such that $n=2^m$.

Types of De-multiplexers

- ✓ 1:2 De-multiplexer
- ✓ 1:4 De-multiplexer
- ✓ 1:8 De-multiplexer
- ✓ 1:16 De-multiplexer
- ✓ 1:32 De-multiplexer
- ✓ 1:64 De-multiplexer

and so on.....

1: 2 De-multiplexer

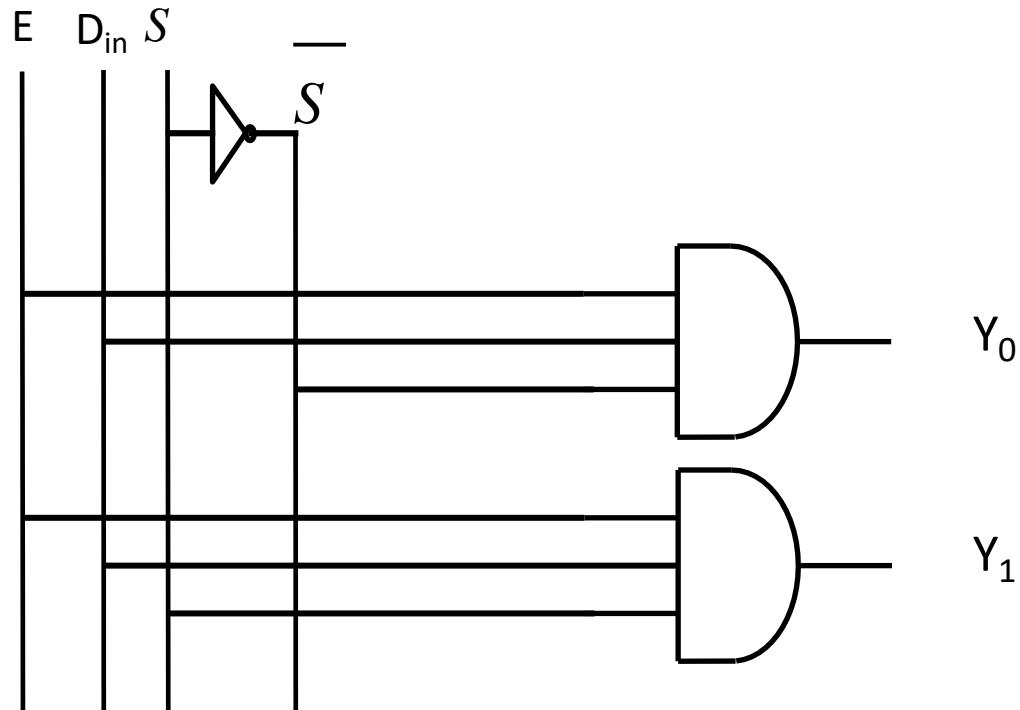


Block Diagram

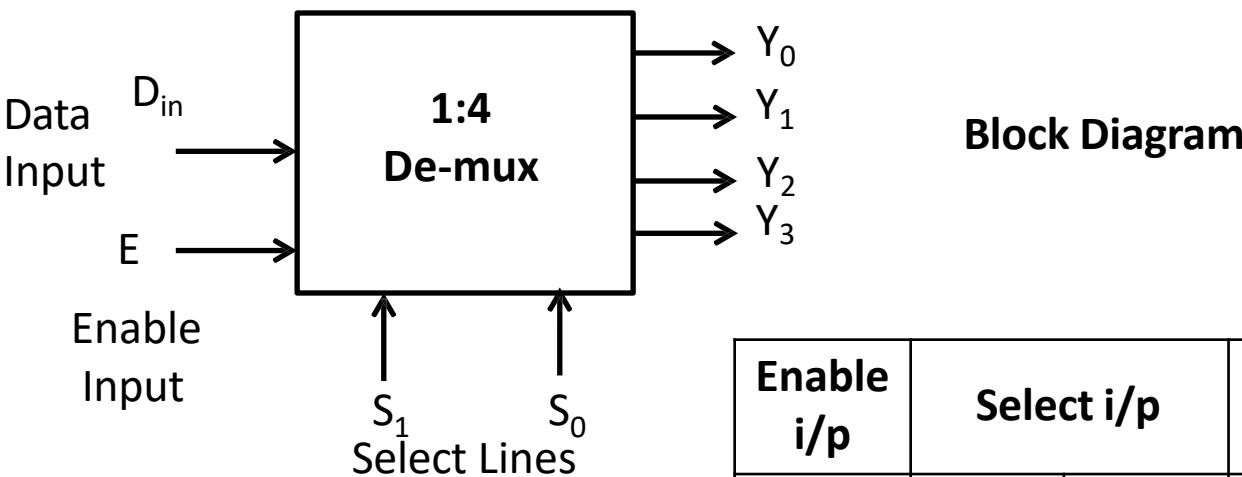
Truth Table

Enable i/p	Select i/p	Outputs	
E	S	Y_0	Y_1
0	X	0	0
1	0	D_{in}	0
1	1	0	D_{in}

1:2 De-mux using basic gates



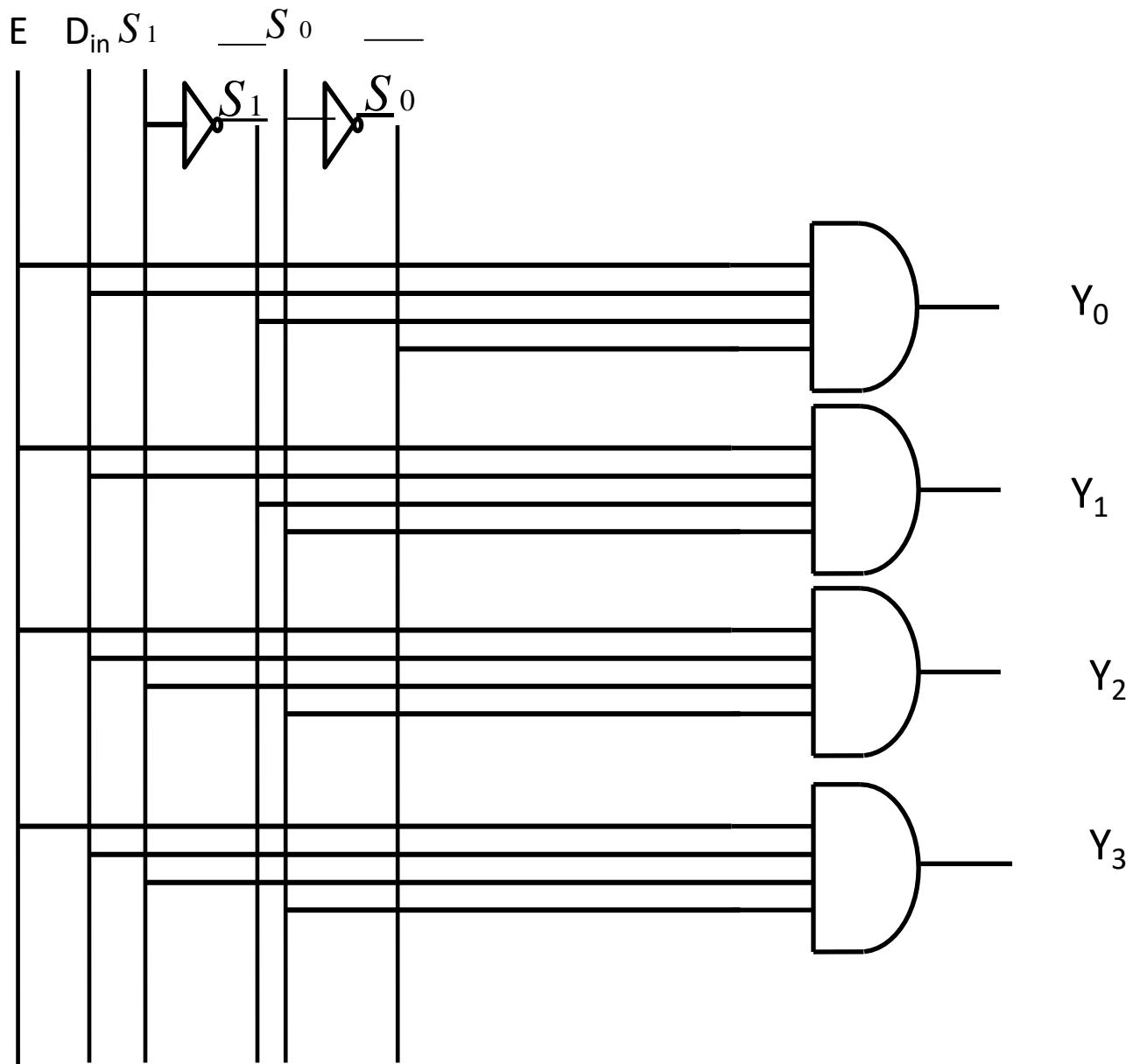
1: 4 De-multiplexer



Truth Table

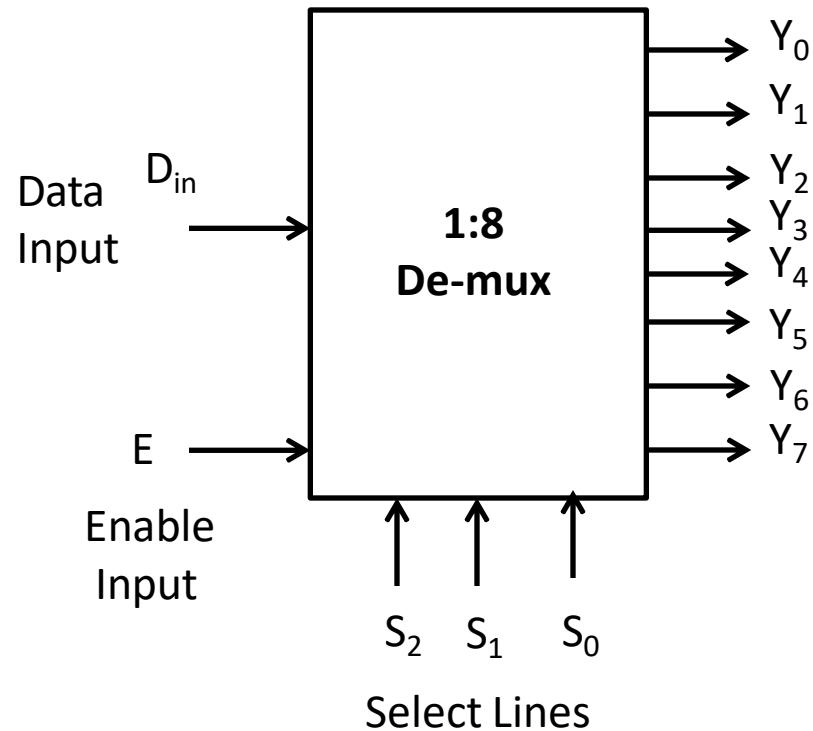
Enable i/p	Select i/p		Outputs			
	S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	X	X	0	0	0	0
1	0	0	D_{in}	0	0	0
1	0	1	0	D_{in}	0	0
1	1	0	0	0	D_{in}	0
1	1	1	0	0	0	D_{in}

1:4 De-mux using basic gates



1: 8 De-multiplexer

Block Diagram



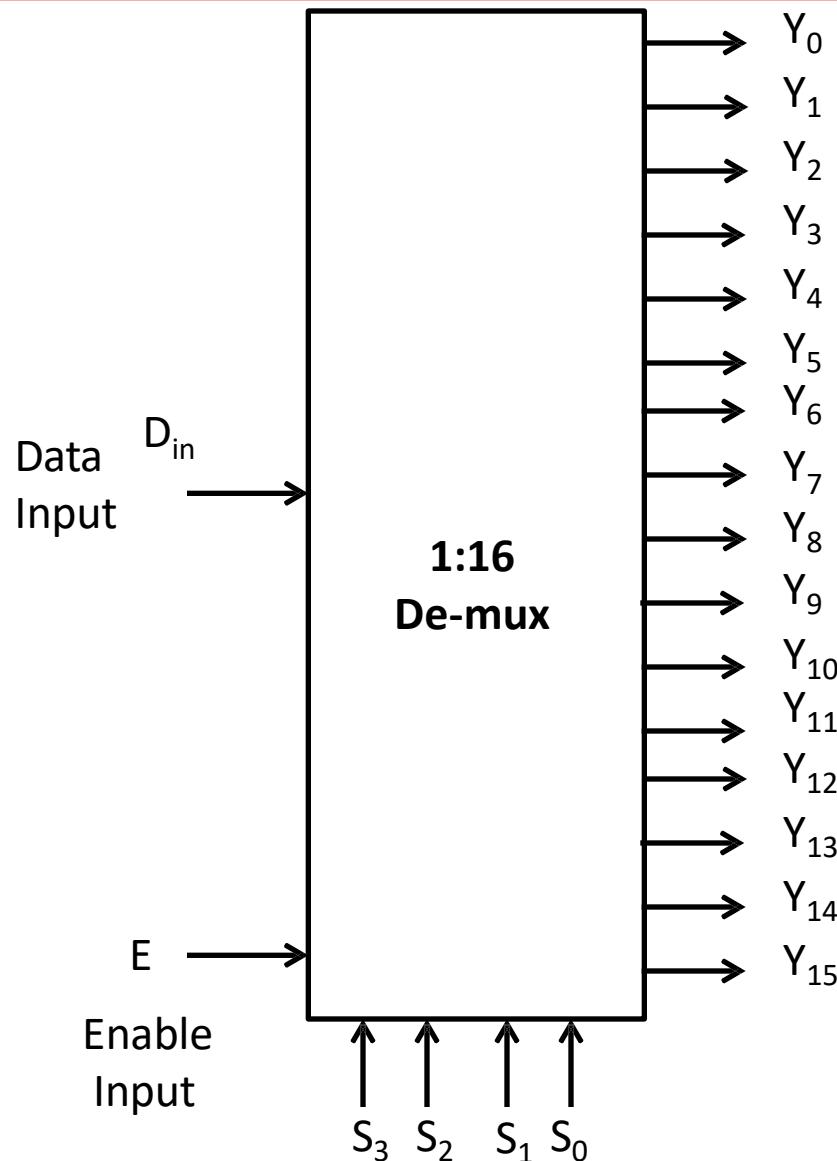
1: 8 De-multiplexer

Truth Table

Enable i/p	Select i/p			Outputs								
	E	S ₂	S ₁	S ₀	Y ₇	Y ₆	Y ₅	Y ₄	Y ₃	Y ₂	Y ₁	Y ₀
0	X	X	X	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	D _{in}
1	0	0	1	0	0	0	0	0	0	0	D _{in}	0
1	0	1	0	0	0	0	0	0	0	D _{in}	0	0
1	0	1	1	0	0	0	0	0	D _{in}	0	0	0
1	1	0	0	0	0	0	0	D _{in}	0	0	0	0
1	1	0	1	0	0	D _{in}	0	0	0	0	0	0
1	1	1	0	0	D _{in}	0	0	0	0	0	0	0
1	1	1	1	1	D _{in}	0	0	0	0	0	0	0

1: 16 De-multiplexer

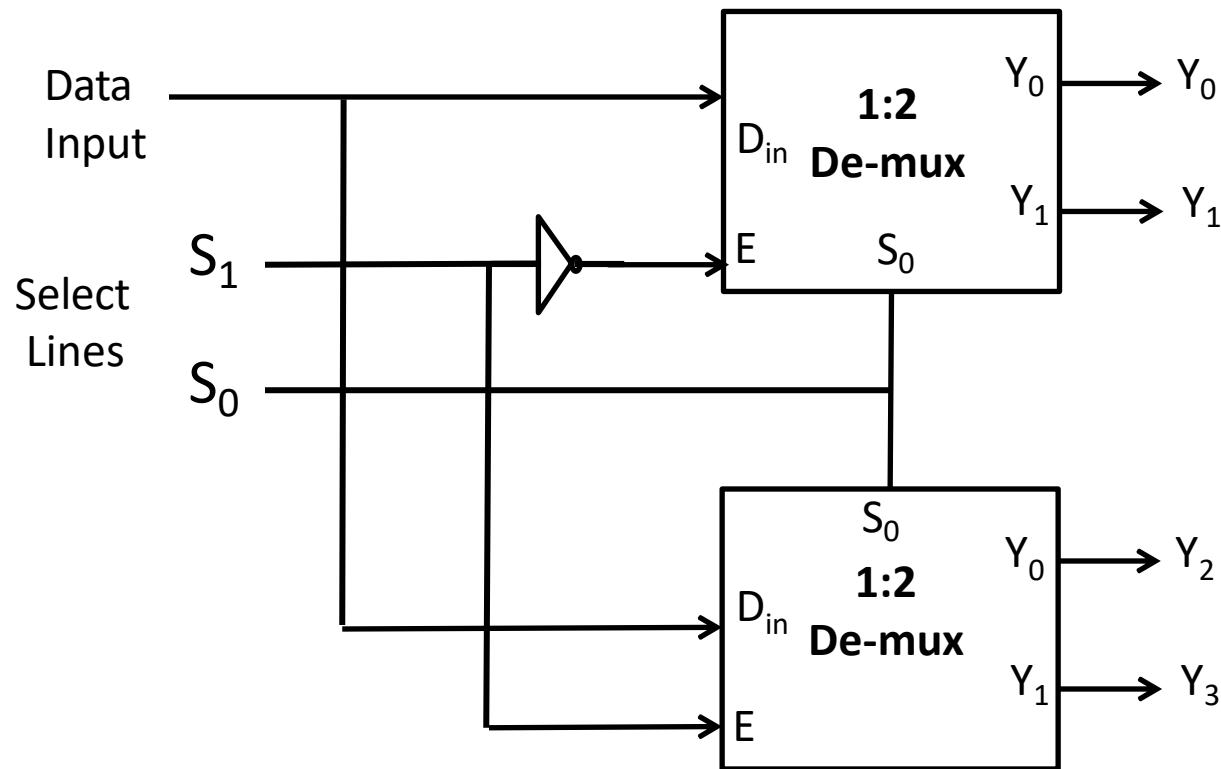
Block Diagram



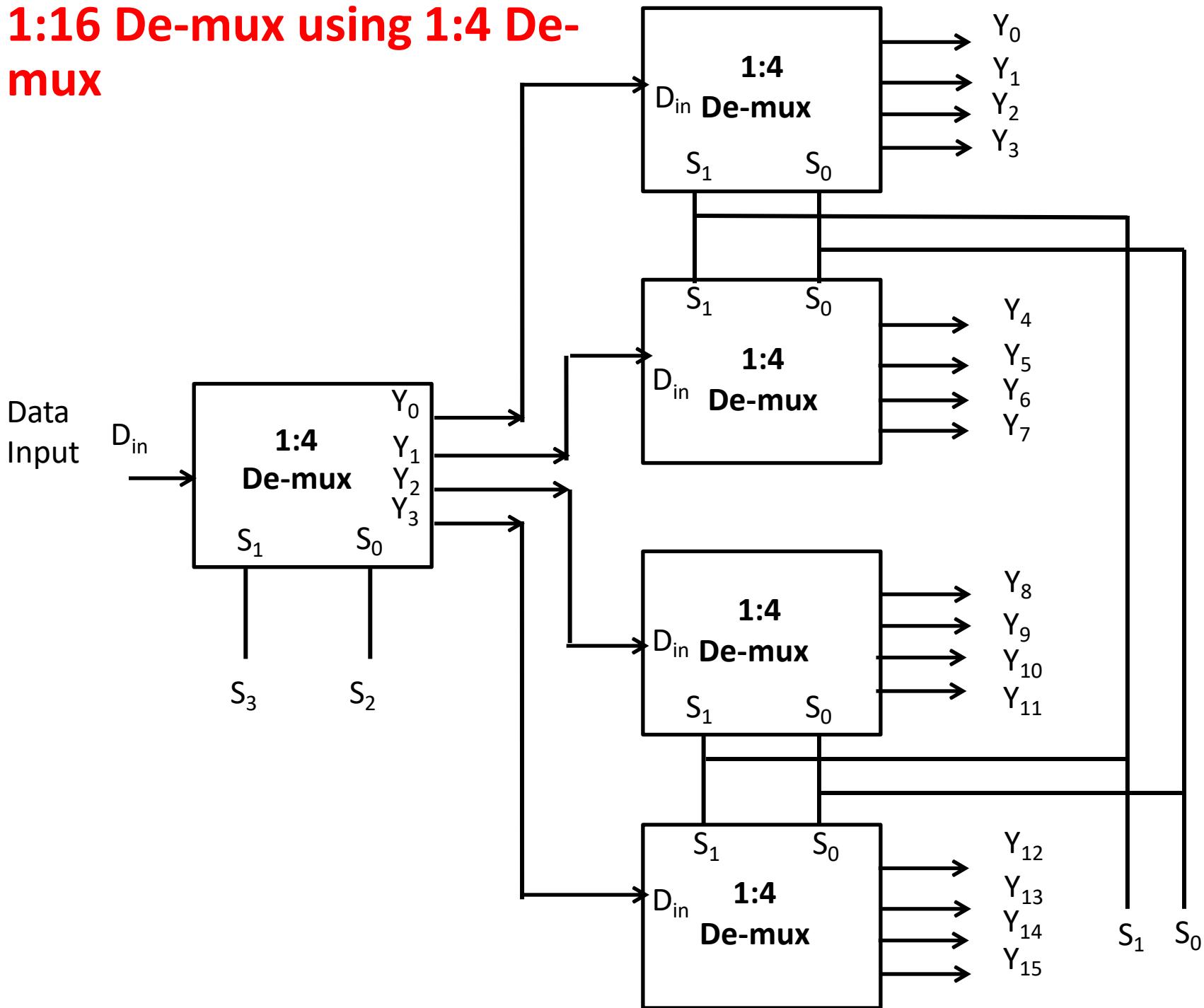
De-mux Tree

✓ Similar to multiplexer we can construct the de-multiplexer with more number of lines using de-multiplexer having less number of lines. This is called as “De-mux Tree”.

1:4 De-mux using 1:2 De-mux



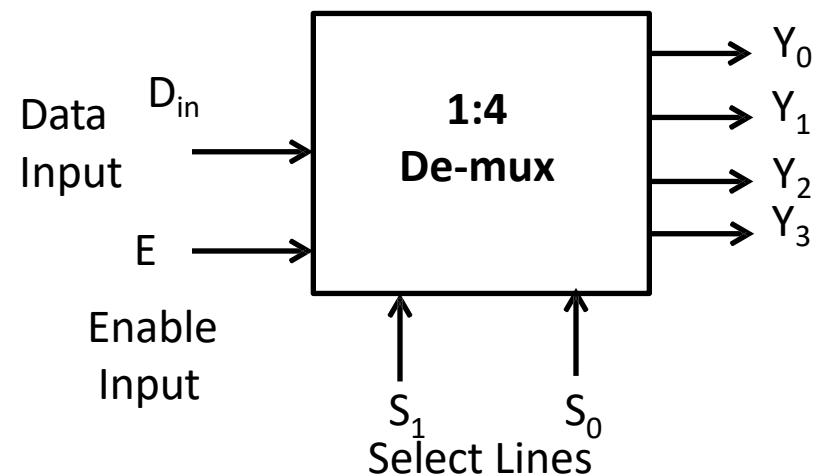
1:16 De-mux using 1:4 De-mux



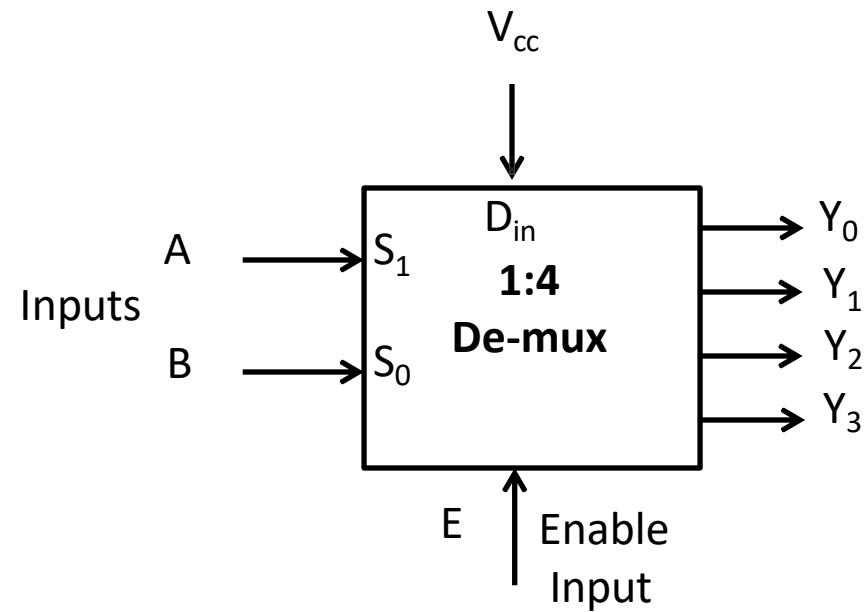
De-multiplexer as Decoder

- ✓ It is possible to operate a de-multiplexer as a decoder.
- ✓ Let us consider an example of 1:4 de-mux can be used as 2:4 decoder

1:4 De-multiplexer as 2:4 Decoder



1: 4 De-multiplexer



1: 4 De-multiplexer as 2:4 Decoder

Realization of Boolean expression using De-mux

- ✓ We can implement any Boolean expression using de-multiplexers.
- ✓ It reduces circuit complexity.
- ✓ It does not require any simplification

Example 1

Implement following Boolean expression using de-multiplexer

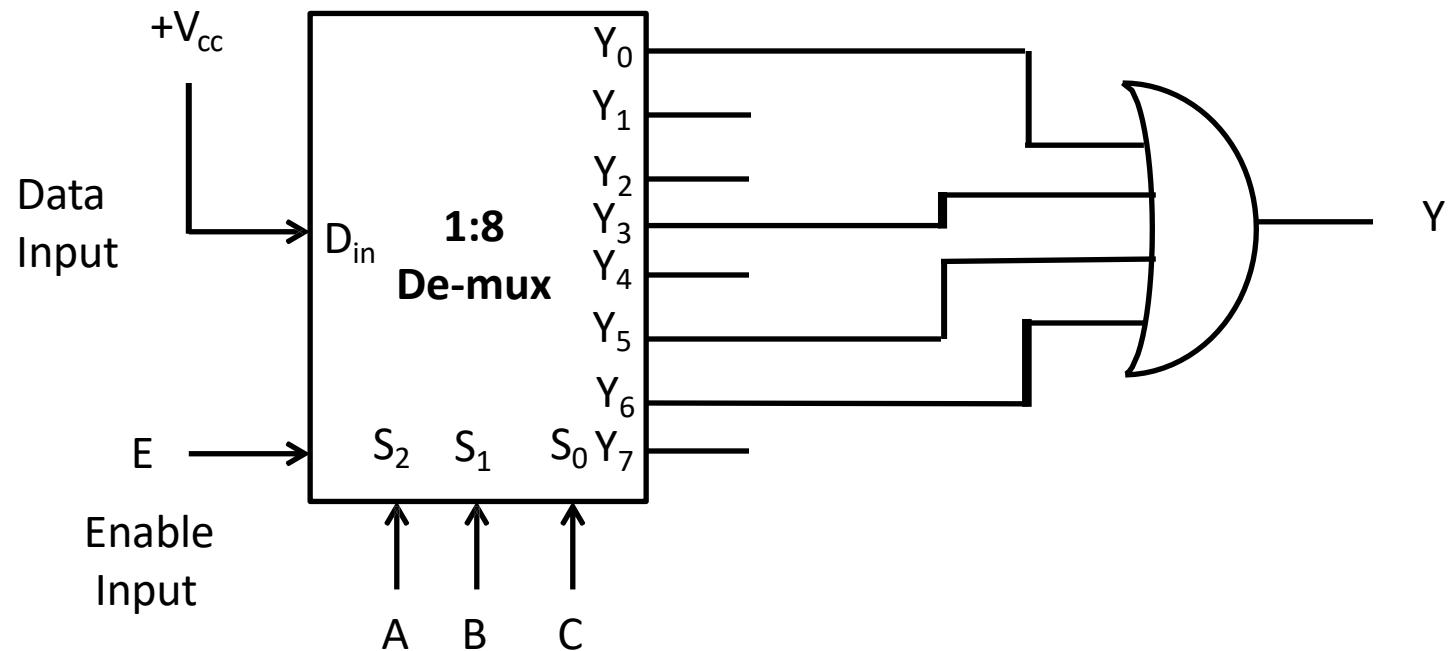
$$f(A, B, C) = \sum m(0, 3, 5, 6)$$

- ✓ Since there are three variables, therefore a de-multiplexer with three select input is required i.e. 1:8 de-multiplexer is required
- ✓ The 1:8 de-multiplexer is configured as below to implement given Boolean expression

Example 1

continue.....

$$f(A, B, C) = \sum m(0, 3, 5, 6)$$



Example 2

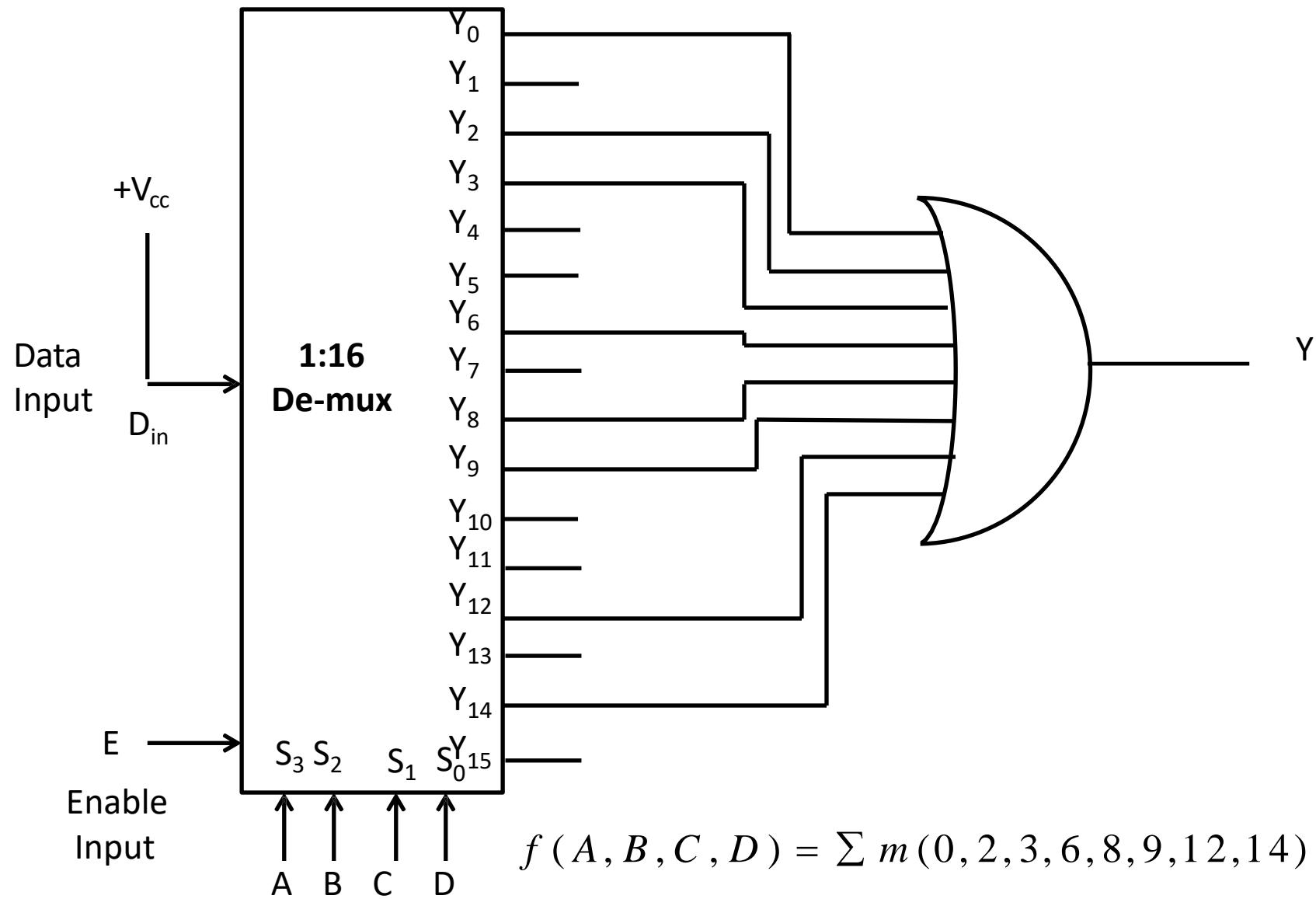
Implement following Boolean expression using de-multiplexer

$$f(A, B, C, D) = \sum m(0, 2, 3, 6, 8, 9, 12, 14)$$

- ✓ Since there are four variables, therefore a de-multiplexer with four select input is required i.e. 1:16 de-multiplexer is required
- ✓ The 1:16 de-multiplexer is configured as below to implement given Boolean expression

Example 2

continue.....



Multiplexer ICs

IC Number	Description	Output
IC 74157	Quad 2:1 Mux	Same as input
IC 74158	Quad 2:1 Mux	Inverted Output
IC 74153	Dual 4:1 Mux	Same as input
IC 74352	Dual 4:1 Mux	Inverted Output
IC 74151	8:1 Mux	Inverted Output
IC 74152	8:1 Mux	Inverted Output
IC 74150	16:1 Mux	Inverted Output

De-multiplexer ICs

IC Number	Description
IC 74138	1:8 De-multiplexer
IC 74139	Dual 1:4 De-multiplexer
IC 74154	1:16 De-multiplexer
IC 74155	Dual 1:4 De-multiplexer



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



Switching Theory and Logic Design

ARI-209

By: Dr. Divya Agarwal



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT- II** 10 HOURS
- **Sequential Logic Circuits:** Latches and Flip Flops- SR, D, T and MS-JK Flip Flops, Asynchronous Inputs.
- **Counters and Shift Registers:** Design of Synchronous and Asynchronous Counters- Binary, BCD, Decade and Up/Down Counters, Shift Registers, Types of Shift Registers, Counters using Shift Registers- Ring Counter and Johnson Counter.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-III** **08 HOURS**
- **Integrated circuits:** TTL and CMOS logic families and their characteristics. Brief introduction to RAM and ROM
- **Synchronous Sequential Circuits:** State Tables State Equations and State Diagrams, State Reduction and State Assignment, Design of Clocked Sequential Circuits using State Equations.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-IV** 06 HOURS
- **Finite state machine:** capabilities and limitations, Mealy and Moore models-minimization of completely specified and incompletely specified sequential machines, Partition techniques and merger chart methods, concept of minimal cover table.
- **Algorithmic State Machine:** Representation of sequential circuits using ASM charts synthesis of output and next state functions, Data path control path partition-based design.



Text Books:

1. Leach and Malvino (2011). *Digital principles and Applications*. Tata McGraw-Hill Education
2. Mano, M. M. (2017). *Digital logic and computer design*. Pearson Education India.
3. Jain, R. P. (2003). *Modern digital electronics*. Tata McGraw-Hill Education.

Reference Books:

1. A Anand Kumar. (2016) *Fundamentals of Digital Logic Circuits*, PHI
2. Taub, H., & Schilling, D. L. (1977). *Digital integrated electronics*. McGraw-Hill College.



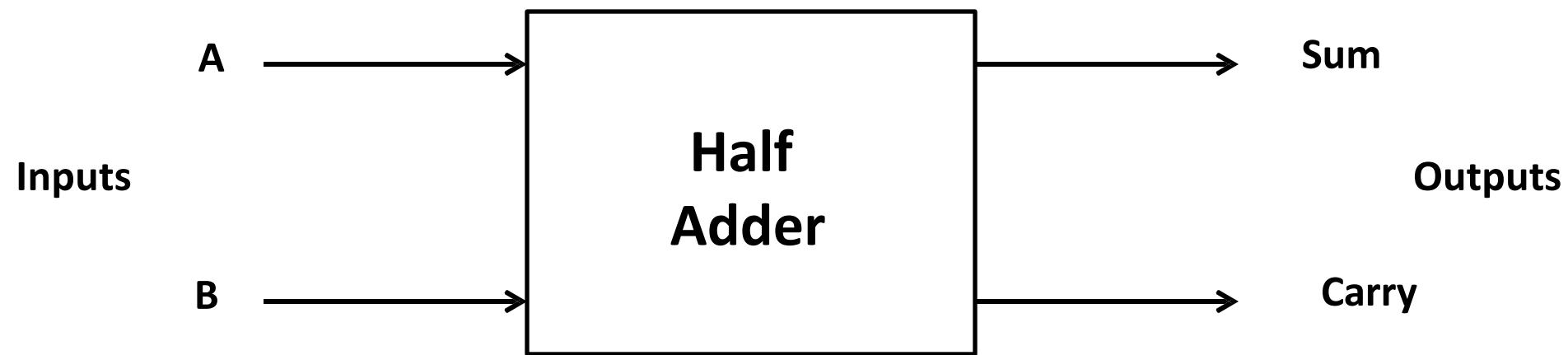
■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.

Half Adder

- ✓ Half adder is a combinational logic circuit with two inputs and two outputs.
- ✓ It is a basic building block for addition of two single bit numbers.



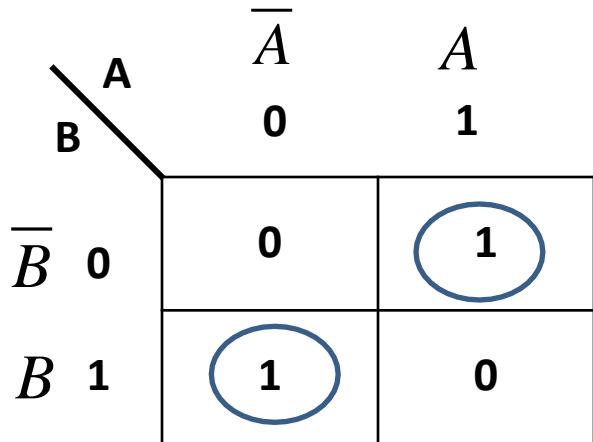
Half Adder

Truth Table for Half Adder

Input		Output	
A	B	Sum (S)	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half Adder

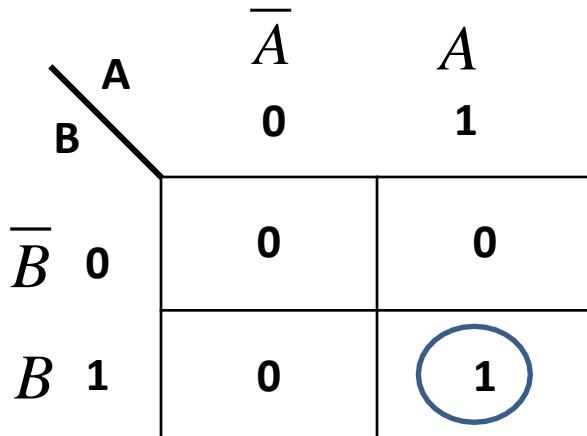
K-map for Sum Output:



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

$$S = A \oplus B$$

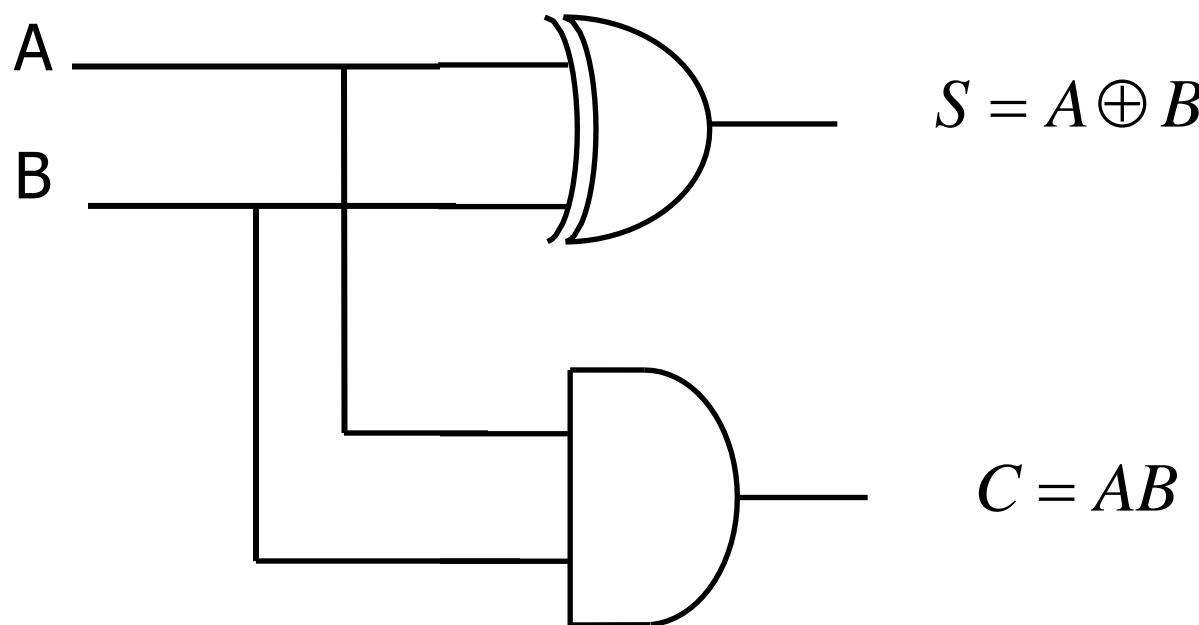
K-map for Carry Output:



$$C = AB$$

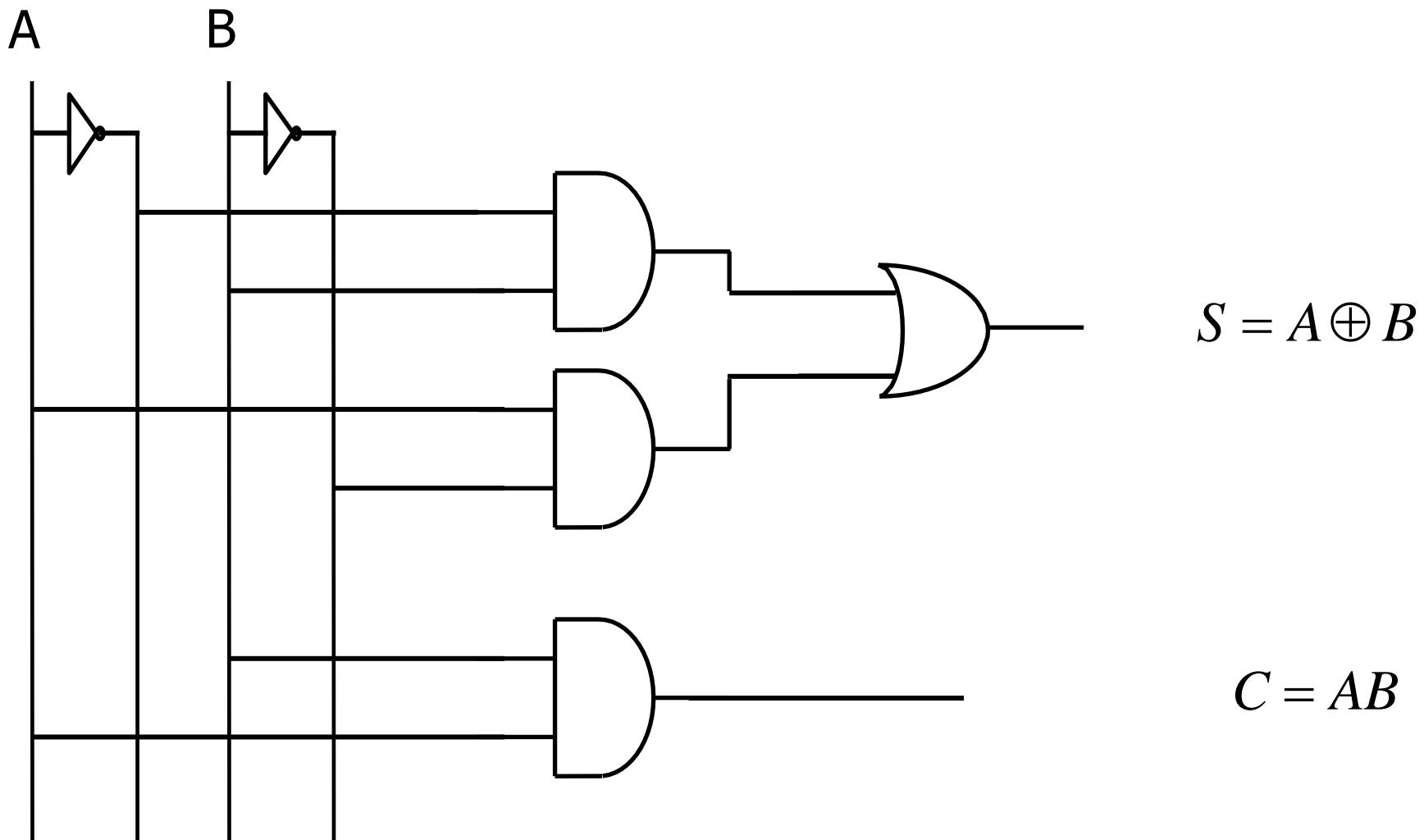
Half Adder

Logic Diagram:



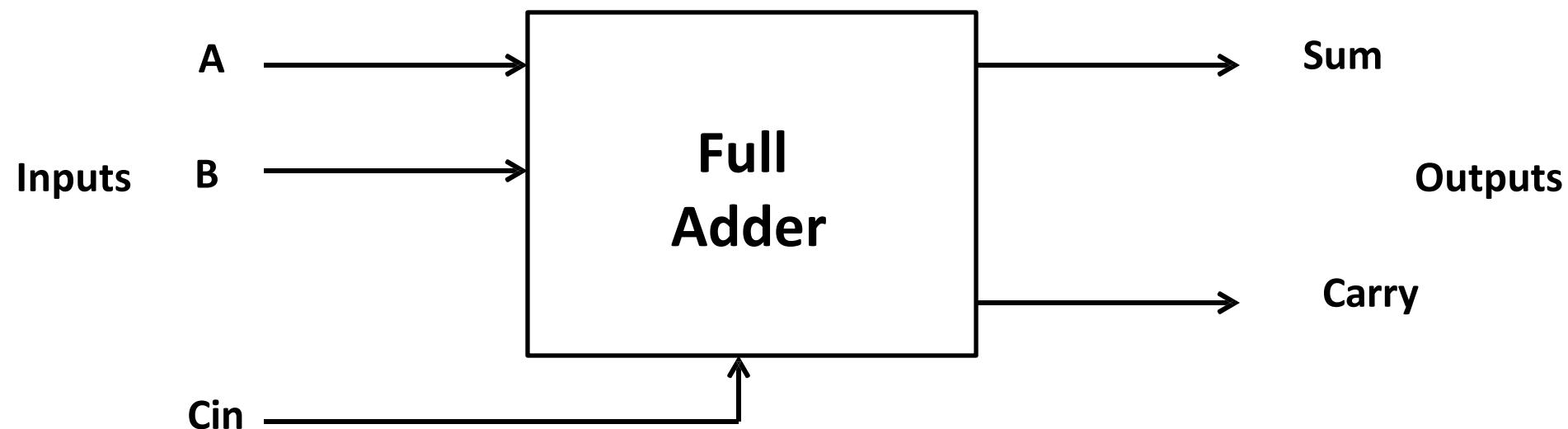
Half Adder

Logic Diagram using Basic Gates:



Full Adder

- ✓ Full adder is a combinational logic circuit with three inputs and two outputs.



Full Adder

Truth Table

Inputs			Outputs	
A	B	Cin	Sum (S)	Carry (C)
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

Full Adder

K-map for Sum Output:

		$\bar{B}\bar{C}$	$\bar{B}C$	$B\bar{C}$	BC
		00	01	11	10
		0	1	0	1
\bar{A}	0	0	1	0	1
	1	1	0	1	0

Arrows point from the circled '1's to the output terms $A\bar{B}\bar{C}$, $\bar{A}\bar{B}C$, ABC , and $\bar{A}B\bar{C}$.

$$S = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + ABC + A\bar{B}\bar{C}$$

$$S = \bar{A}\bar{B}\bar{C} + ABC + \bar{A}B\bar{C} + A\bar{B}\bar{C}$$

$$S = C(\bar{A}\bar{B}) + AB + \bar{C}(\bar{A}B + A\bar{B})$$

$$\text{Let } \bar{A}B + A\bar{B} = X$$

$$\therefore S = C(\bar{X}) + \bar{C}(X)$$

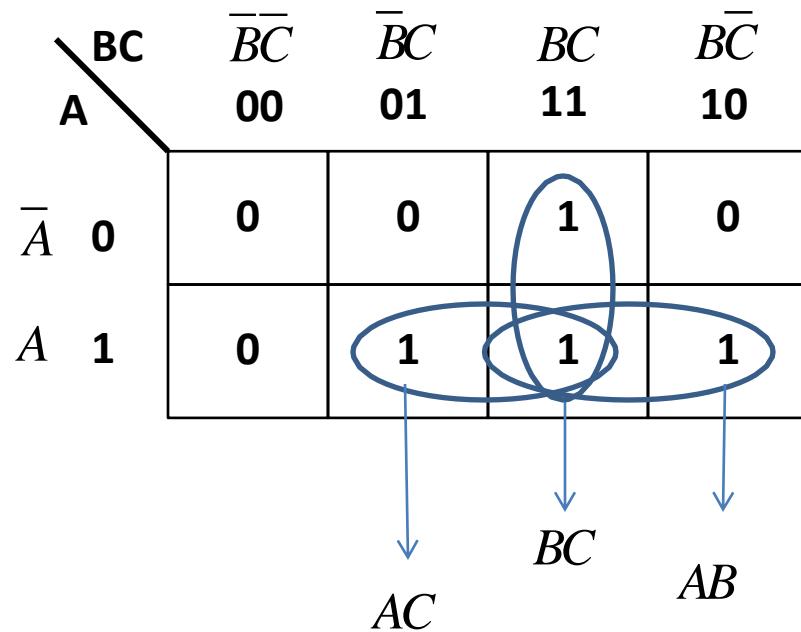
$$S = C \oplus X$$

$$\text{Let } X = A \oplus B$$

$$\therefore S = C \oplus A \oplus B$$

Full Adder

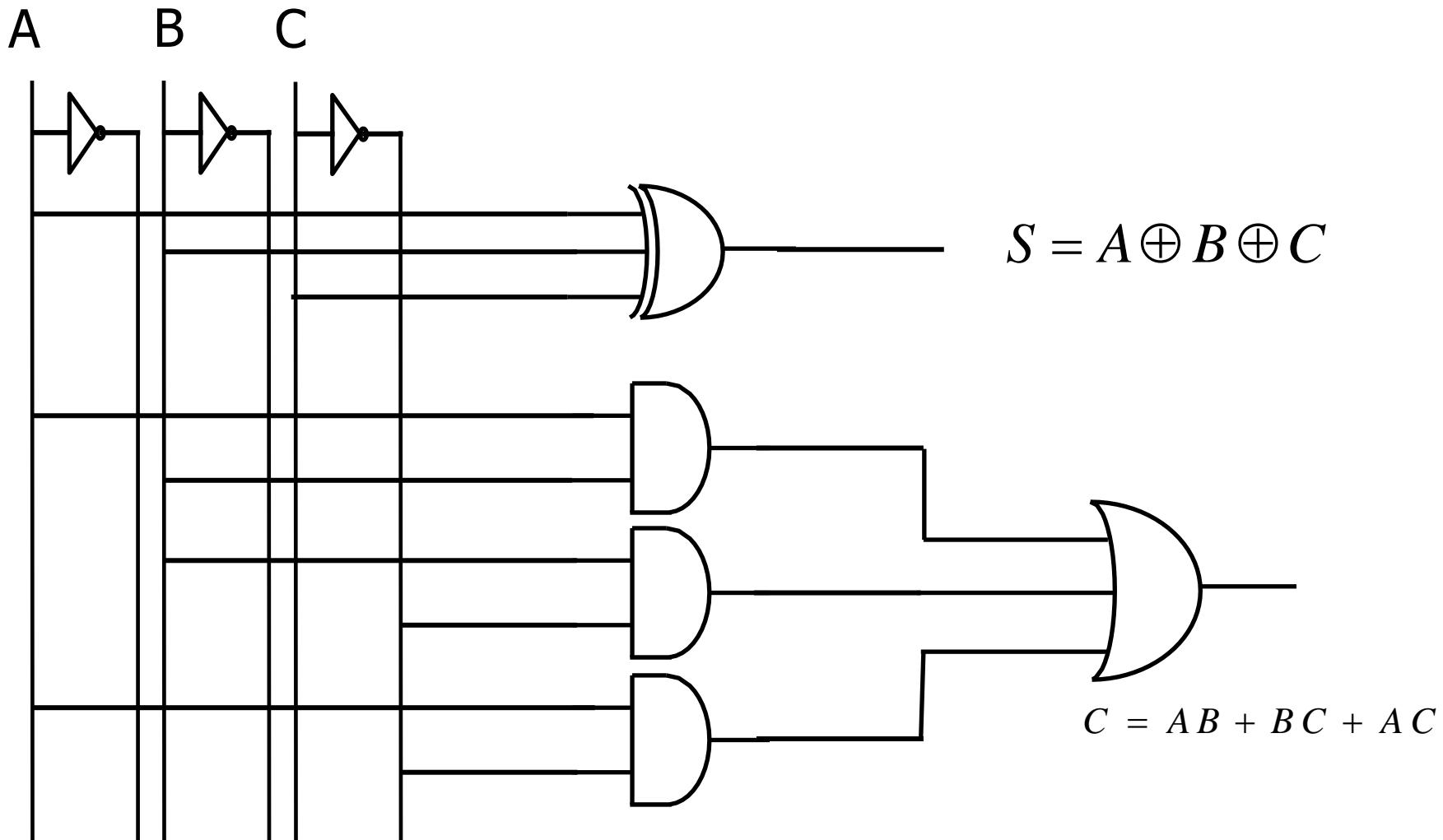
K-map for Carry Output:



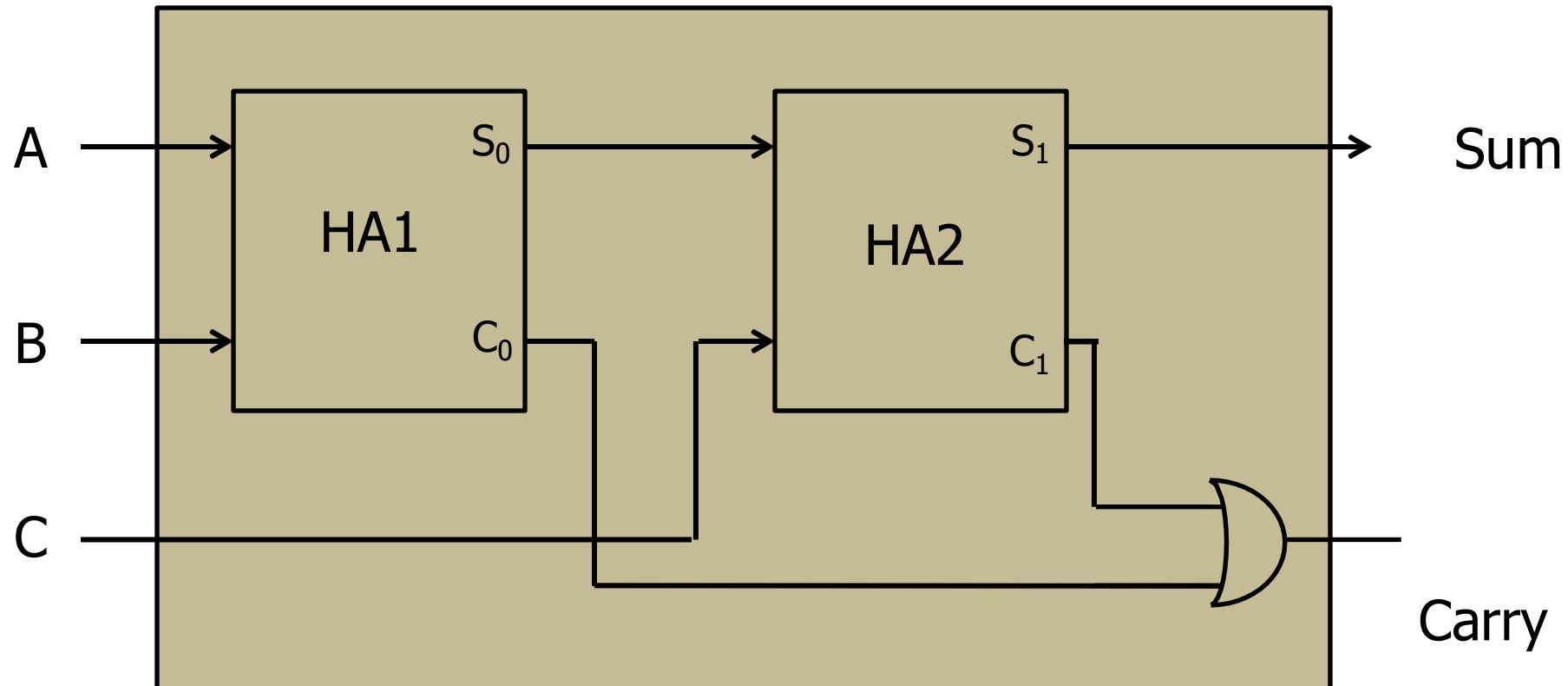
$$C = AB + BC + AC$$

Full Adder

Logic Diagram:

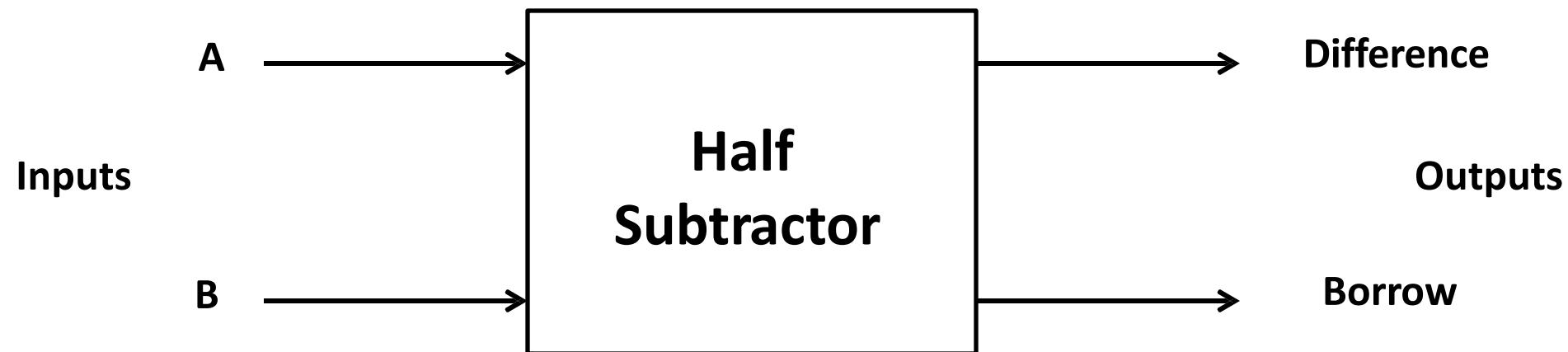


Full Adder using Half Adders



Half Subtractor

- ✓ Half subtractor is a combinational logic circuit with two inputs and two outputs.
- ✓ It is a basic building block for subtraction of two single bit numbers.



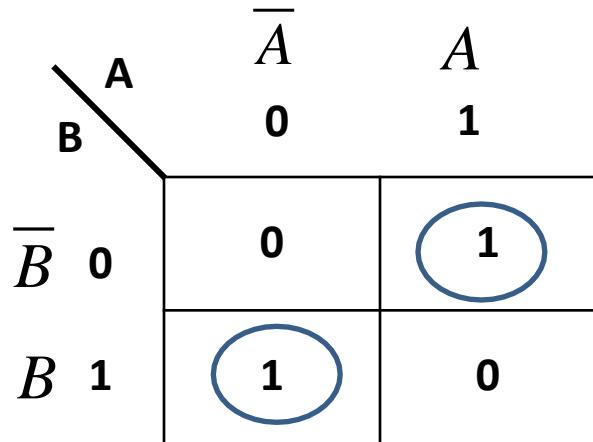
Half Subtractor

Truth Table

Input		Output	
A	B	Difference (D)	Borrow (B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Half Subtractor

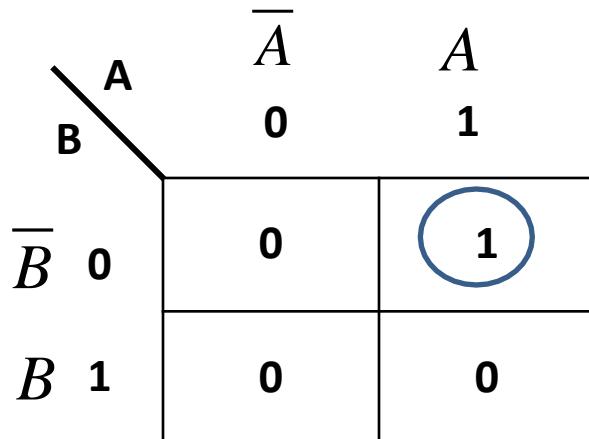
K-map for Difference Output:



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

$$D = A \oplus B$$

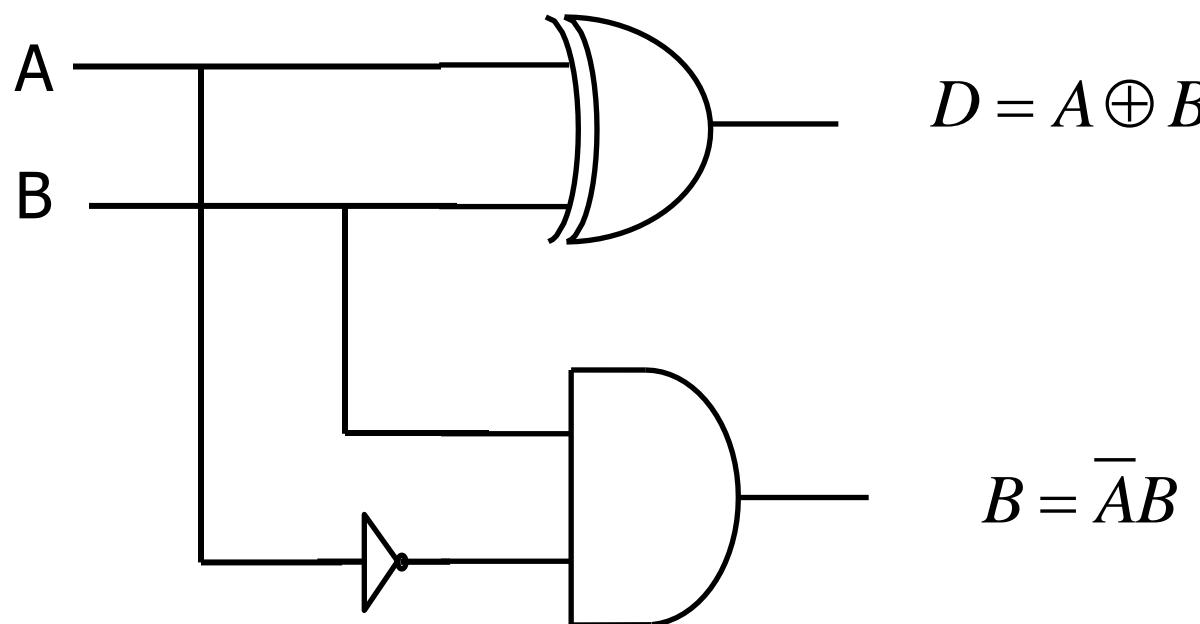
K-map for Borrow Output:



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

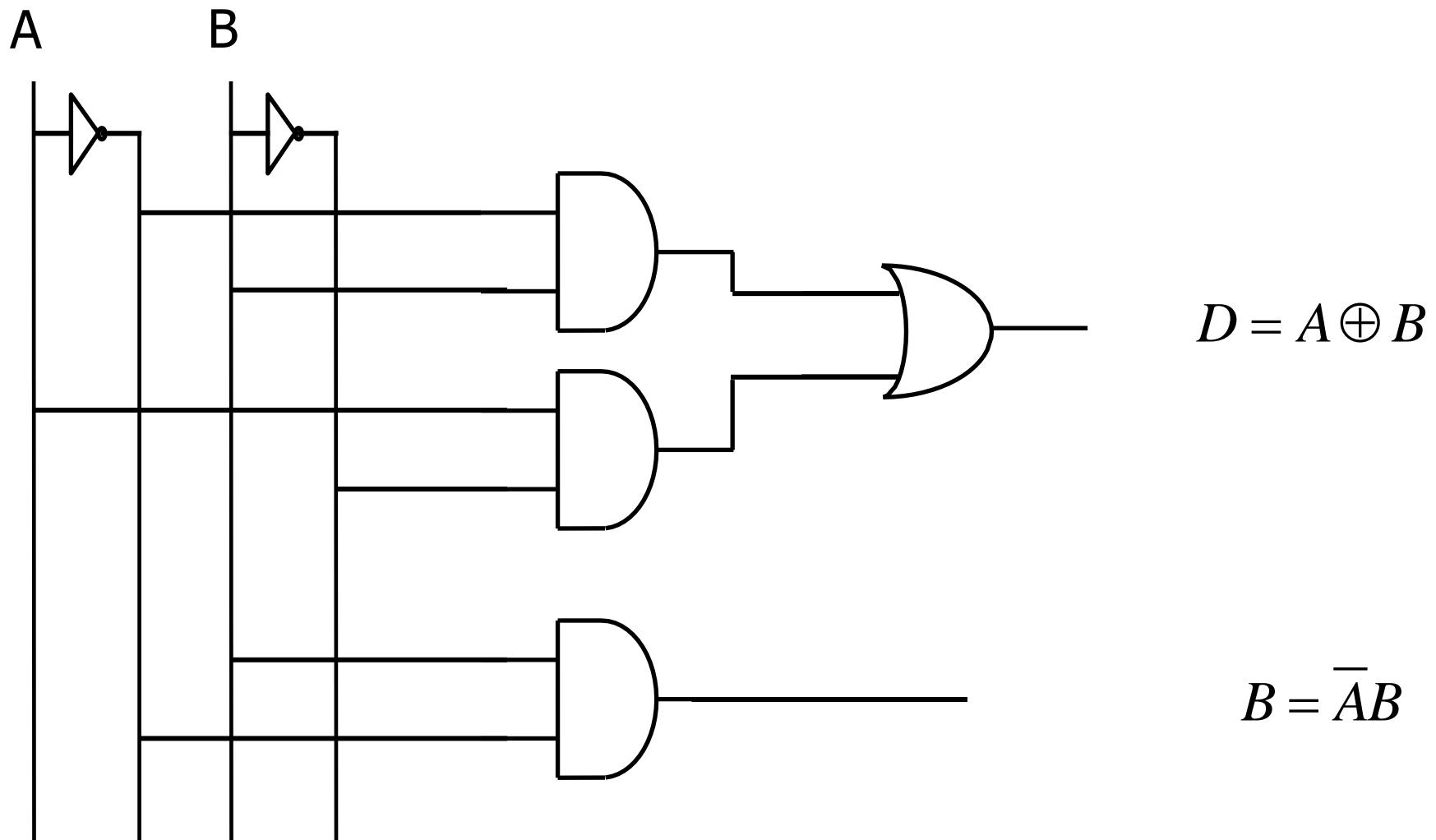
Half Subtractor

Logic Diagram:



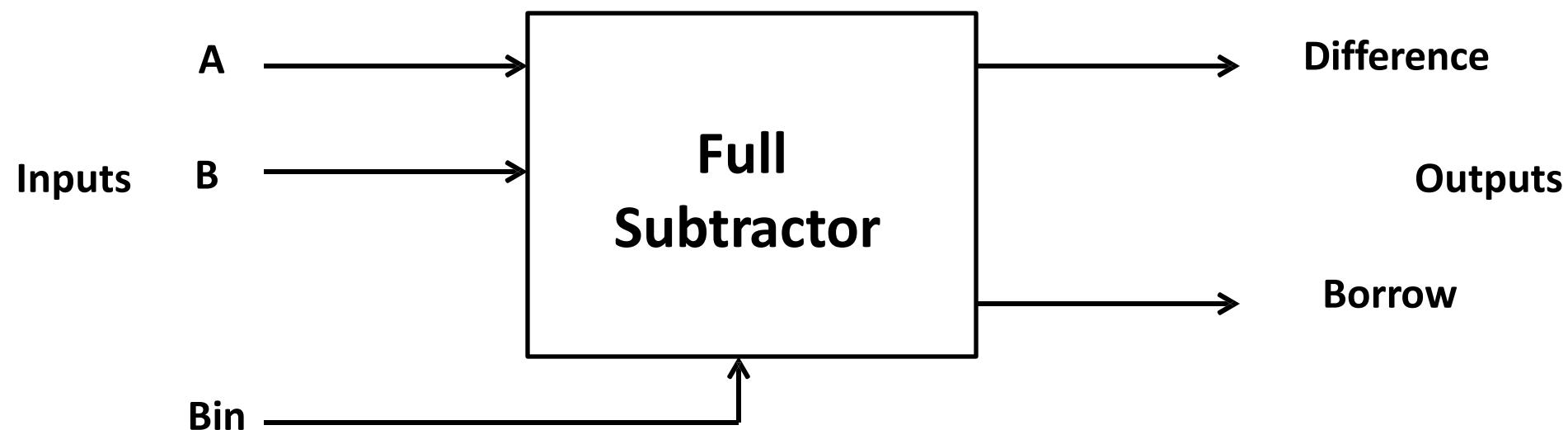
Half Subtractor

Logic Diagram using Basic Gates:



Full Subtractor

- ✓ Full subtractor is a combinational logic circuit with three inputs and two outputs.



Full Subtractor

Truth Table

Inputs			Outputs	
A	B	Bin (C)	Difference (D)	Borrow (B0)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Full Subtractor

K-map for Difference Output:

		$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
		00	01	11	10
A		0	1	0	1
\bar{A}	0	0	1	0	1
	1	1	0	1	0

Labels below the K-map:

- \bar{ABC} (top-left)
- \bar{ABC} (top-right)
- ABC (bottom-middle)
- \bar{ABC} (bottom-right)

$$D = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}C + AB\bar{C}$$

$$D = \bar{A}\bar{B}\bar{C} + ABC + \bar{A}B\bar{C} + A\bar{B}\bar{C}$$

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

Let $\bar{A}B + A\bar{B} = X$

$$\therefore D = C(\bar{X}) + \bar{C}(X)$$

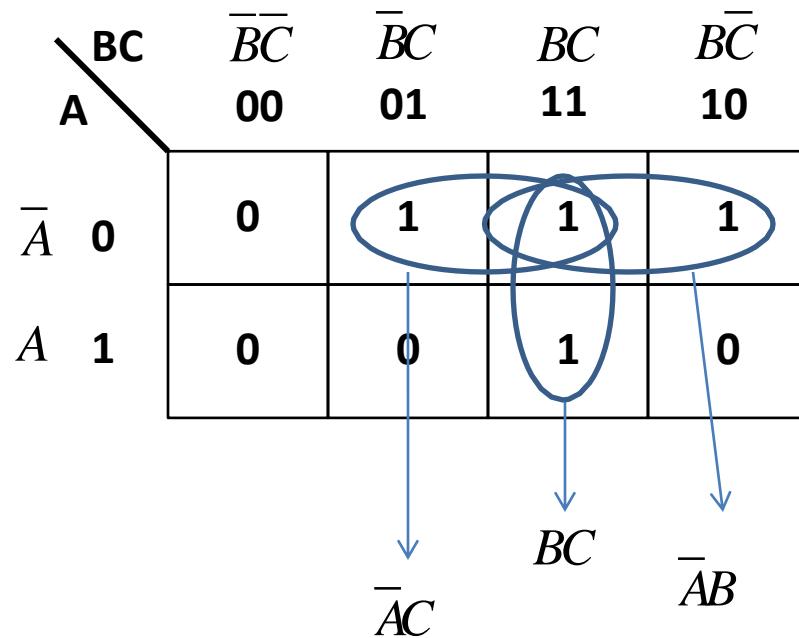
$$D = C \oplus X$$

Let $X = A \oplus B$

$$\therefore D = C \oplus A \oplus B$$

Full Subtractor

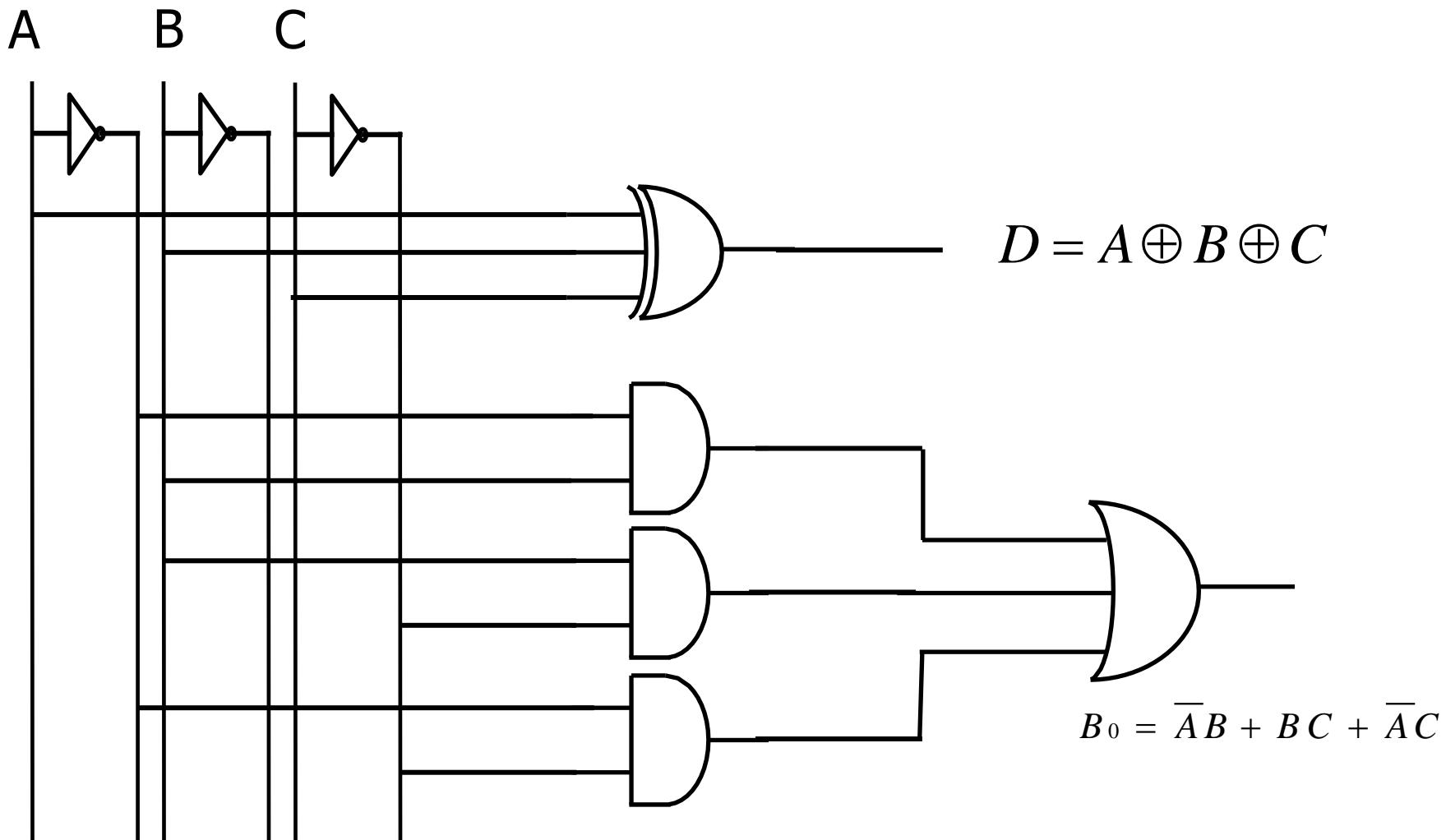
K-map for Borrow Output:



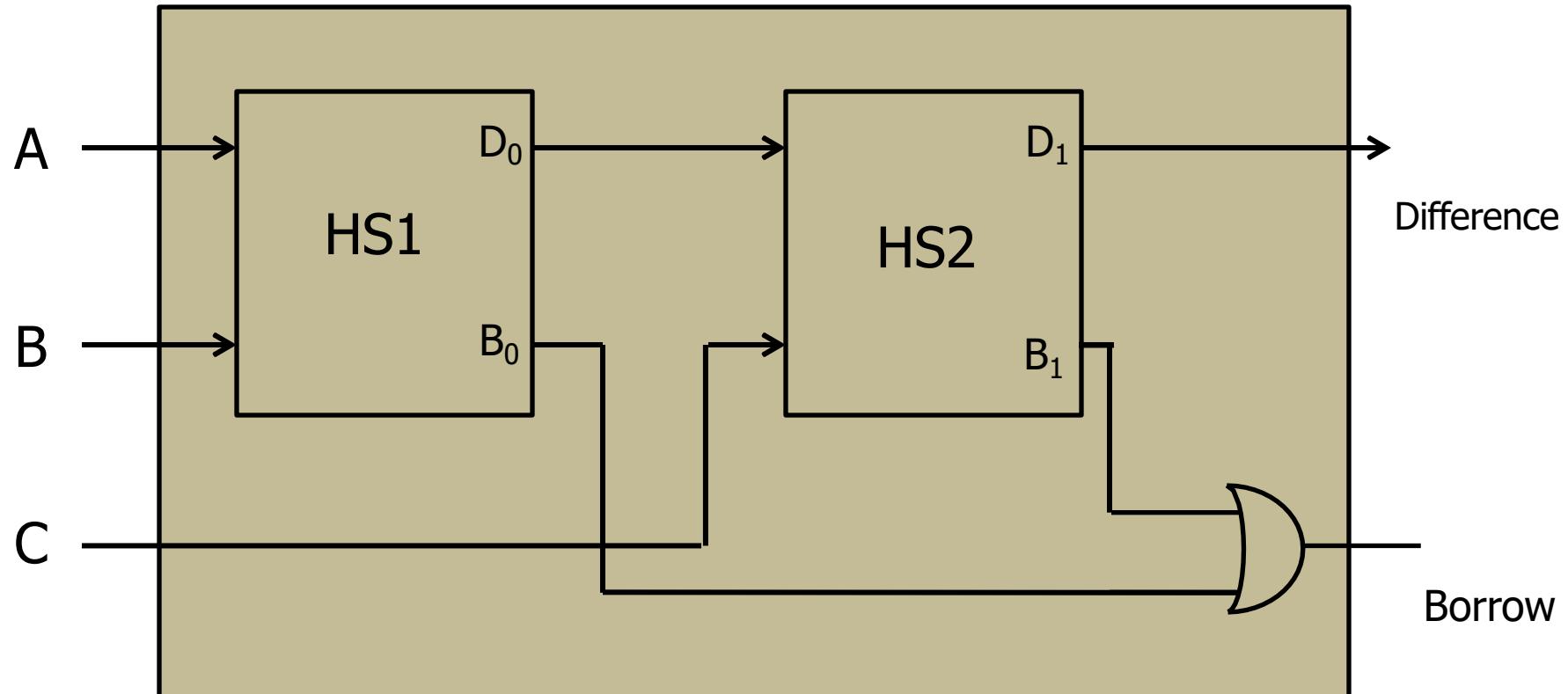
$$B_0 = \overline{AB} + BC + \overline{AC}$$

Full Subtractor

Logic Diagram:



Full Subtractor using Half Subtractor



Disadvantages of Ripple Carry Adder/ Parallel Adder

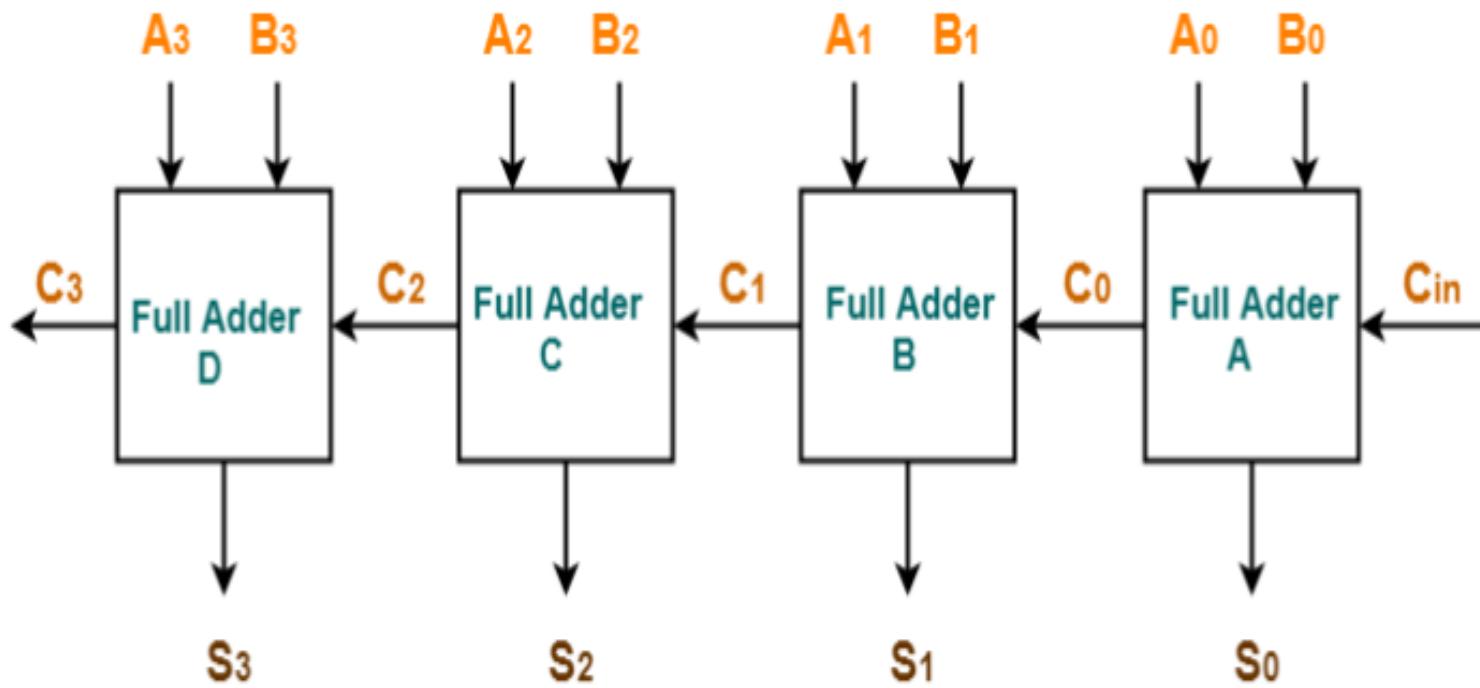
- Parallel Adder does not allow to use all the full adders simultaneously.
- Each full adder has to necessarily wait until the carry bit becomes available from its adjacent full adder.
- This increases the propagation time.
- Due to this reason, Parallel Adder becomes extremely slow.
- This is considered to be the biggest disadvantage of using Parallel adder.

Summary of Ripple Carry Adder/ Parallel Adder

In Parallel Adder,

- Each full adder has to wait for its carry-in from its previous stage full adder.
- Thus, n^{th} full adder has to wait until all $(n-1)$ full adders have completed their operations.
- This causes a delay and makes ripple carry adder extremely slow.
- The situation becomes worst when the value of n becomes very large.
- To overcome this disadvantage, Carry Look Ahead Adder comes into play.

Ripple Carry Adder/ Parallel Adder

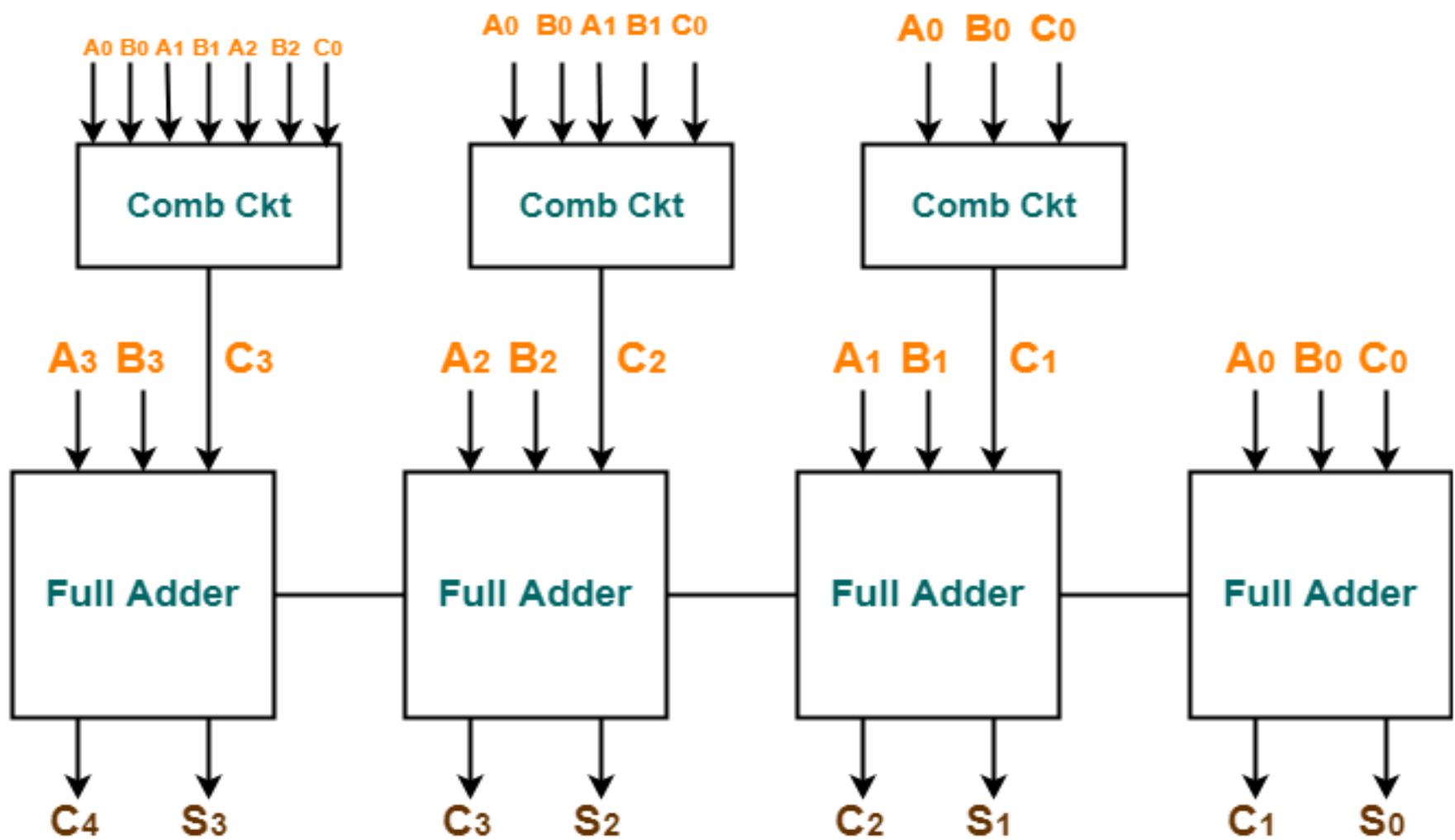


4-bit Ripple Carry Adder

Carry Look Ahead Adder-

- Carry Look Ahead Adder is an improved version of the ripple carry adder.
- It generates the carry-in of each full adder simultaneously without causing any delay.
- The time complexity of carry look ahead adder = $\Theta(\log n)$.

Carry Look Ahead Adder-



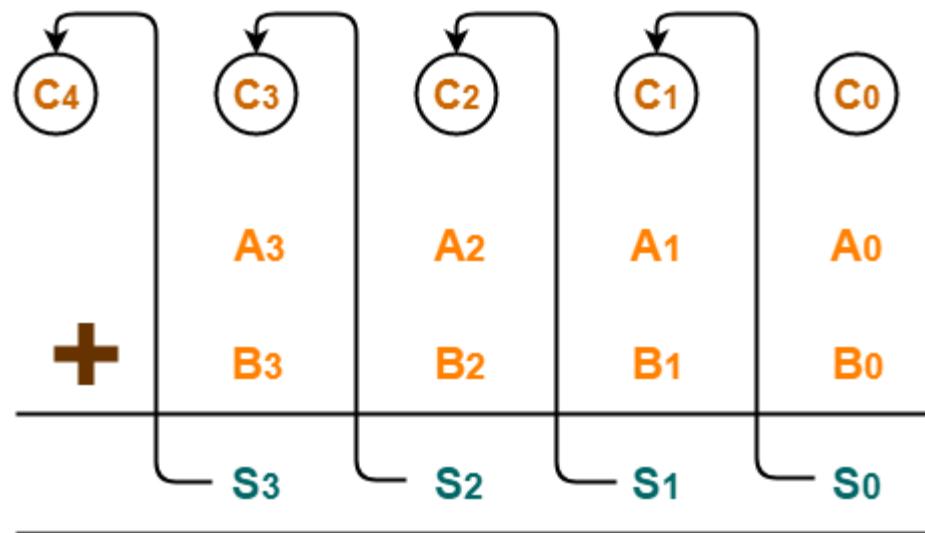
Carry Look Ahead Adder Logic Diagram

Carry Look Ahead Adder Working-

- The working of carry look ahead adder is based on the principle-
“The carry-in of any stage full adder is independent of the carry bits generated during intermediate stages”
- The carry-in of any stage full adder depends only on the following two parameters-
 - Bits being added in the previous stages
 - Carry-in provided in the beginning
- Now, The above two parameters are always known from the beginning.
- So, the carry-in of any stage full adder can be evaluated at any instant of time.
- Thus, any full adder need not wait until its carry-in is generated by its previous stage full adder.

4-Bit Carry Look Ahead Adder-

- Consider two 4-bit binary numbers $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are to be added.
- Mathematically, the two numbers will be added as-



Adding two 4-bit Numbers

4-Bit Carry Look Ahead Adder-

From here, we have-

$$C_1 = C_0 (A_0 \oplus B_0) + A_0 B_0$$

$$C_2 = C_1 (A_1 \oplus B_1) + A_1 B_1$$

$$C_3 = C_2 (A_2 \oplus B_2) + A_2 B_2$$

$$C_4 = C_3 (A_3 \oplus B_3) + A_3 B_3$$

For simplicity,

Let-

$G_i = A_i B_i$ where G is called carry generator

$P_i = A_i \oplus B_i$ where P is called carry propagator

4-Bit Carry Look Ahead Adder-

Then, re-writing the above equations, we have-

$$C_3 = C_2 P_2 + G_2 \dots \dots \dots \quad (3)$$

$$C_4 = C_3 P_3 + G_3 \dots \dots \dots \quad (4)$$

Now,

Clearly, C1, C2 and C3 are intermediate carry bits.

So, let's remove C_1 , C_2 and C_3 from RHS of every equation.

Substituting (1) in (2), we get C_2 in terms of C_0 .

Then, substituting (2) in (3), we get C_3 in terms of C_0 and so on.

4-Bit Carry Look Ahead Adder-

Finally, we have the following equations-

$$C_1 = C_0 P_0 + G_0$$

$$C_2 = C_0 P_0 P_1 + G_0 P_1 + G_1$$

$$C_3 = C_0 P_0 P_1 P_2 + G_0 P_1 P_2 + G_1 P_2 + G_2$$

$$C_4 = C_0 P_0 P_1 P_2 P_3 + G_0 P_1 P_2 P_3 + G_1 P_2 P_3 + G_2 P_3 + G_3$$

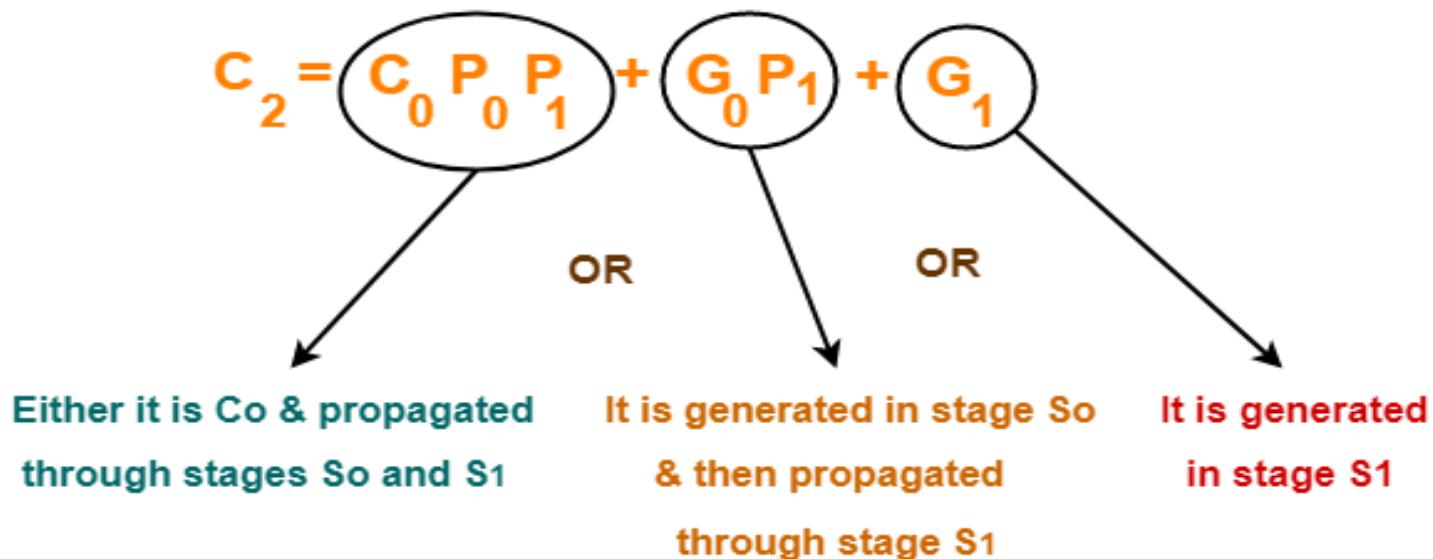
These equations are important to remember.

These equations show that the carry-in of any stage full adder depends only on-

- Bits being added in the previous stages
- Carry bit which was provided in the beginning

Trick To Memorize Above Equations-

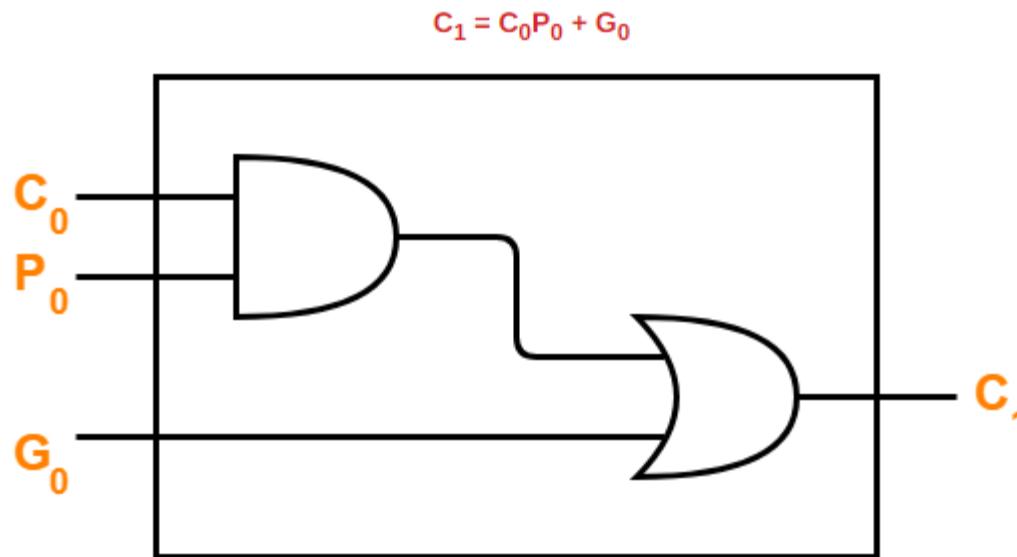
- As an example, let us consider the equation for generating carry bit C2.
- There are three possible reasons for generation of C2 as depicted in the following picture-



- In the similar manner, we can write other equations as well very easily.

Implementation Of Carry Generator Circuits-

- The above carry generator circuits are usually implemented as-
 - Two level combinational circuits.
 - Using AND and OR gates where gates are assumed to have any number of inputs.
- **Implementation Of C1–**
 - The carry generator circuit for C1 is implemented as shown below.
 - It requires 1 AND gate and 1 OR gate.



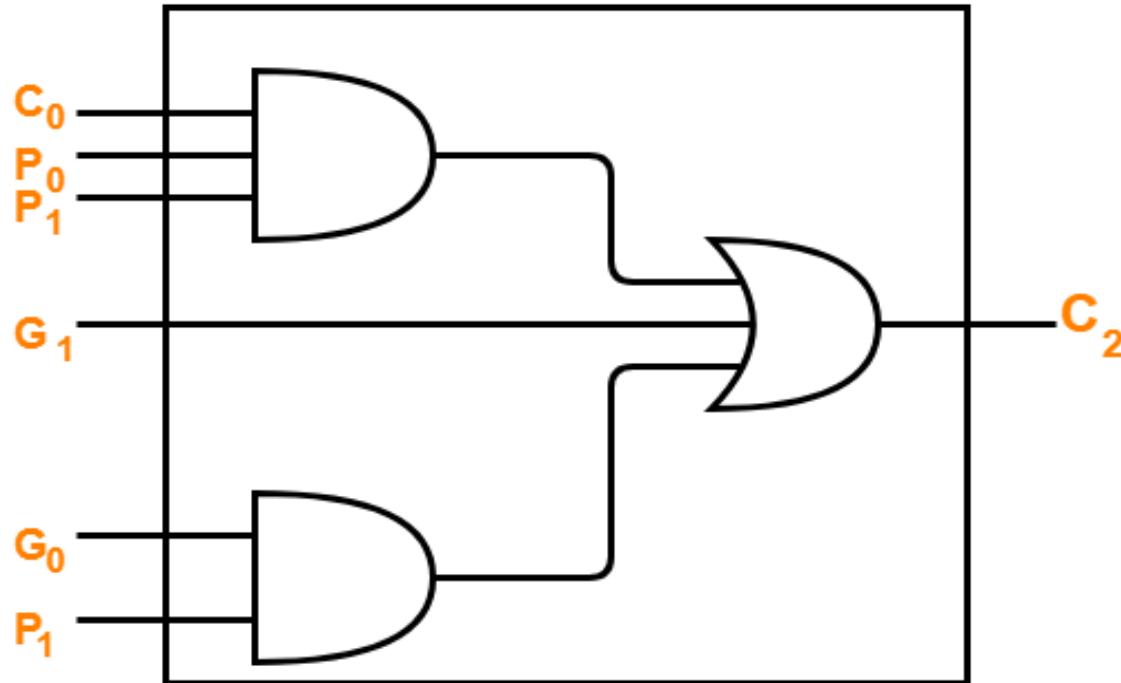
Implementation of C1

Implementation Of Carry Generator Circuits-

Implementation Of C_2 -

- The carry generator circuit for C_2 is implemented as shown below.
- It requires 2 AND gates and 1 OR gate.

$$C_2 = C_0P_0P_1 + G_0P_1 + G_1$$



Implementation of C_2

Implementation Of Carry Generator Circuits-

Implementation Of C_3 & C_4 -

- Similarly, we implement C_3 and C_4 .
 - Implementation of C_3 uses 3 AND gates and 1 OR gate.
 - Implementation of C_4 uses 4 AND gates and 1 OR gate.
- Total number of gates required to implement carry generators (provided carry propagators P_i and carry generators G_i) are-
- Total number of AND gates required for addition of 4-bit numbers $= 1 + 2 + 3 + 4 = 10$.
- Total number of OR gates required for addition of 4-bit numbers $= 1 + 1 + 1 + 1 = 4$.
- For a n-bit carry look ahead adder to evaluate all the carry bits, it requires-
 - Number of AND gates $= n(n+1) / 2$
 - Number of OR gates $= n$

Implementation Of Carry Generator Circuits-

Advantages of Carry Look Ahead Adder-

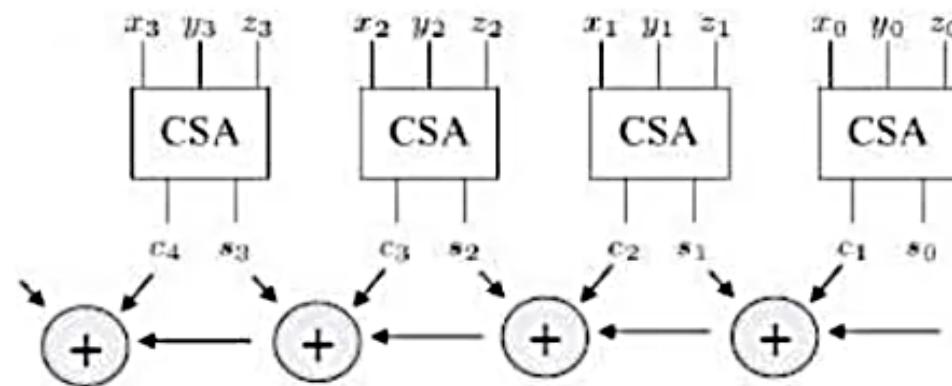
- The advantages of carry look ahead adder are-
- It generates the carry-in for each full adder simultaneously.
- It reduces the propagation delay.

Disadvantages of Carry Look Ahead Adder-

- The disadvantages of carry look ahead adder are-
- It involves complex hardware.
- It is costlier since it involves complex hardware.
- It gets more complicated as the number of bits increases.

Carry Save Adder (CSA)

- In Carry Save Adder (CSA), three bits are added parallelly at a time.
- In this scheme, the carry is not propagated through the stages. Instead, carry is stored in present stage, and updated as addend value in the next stage [2]. Hence, the delay due to the
- carry is reduced in this scheme.



Block Diagram of CSA

Carry Save Adder (CSA)

The carry save adder seems to be the most useful adder for our application. It is simply a parallel ensemble of k full-adders without any horizontal connection. Its main function is to add three k -bit integers A , B , and C to produce two integers C' and S such that

$$C' + S = A + B + C .$$

As an example, let $A = 40$, $B = 25$, and $C = 20$, we compute S and C' as shown below:

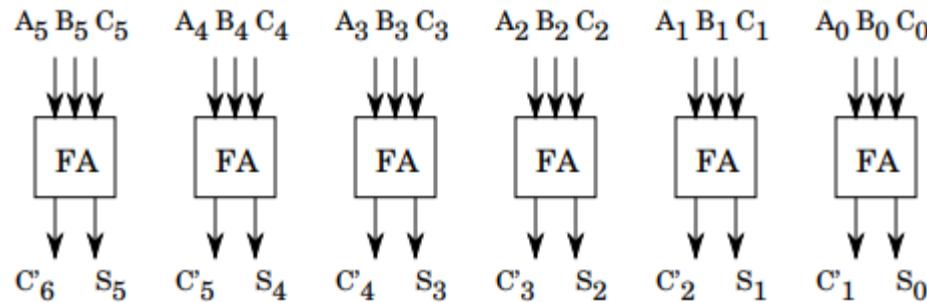
$$\begin{array}{rcl} A = 40 & = & 1 \ 0 \ 1 \ 0 \ 0 \ 0 \\ B = 25 & = & 0 \ 1 \ 1 \ 0 \ 0 \ 1 \\ C = 20 & = & 0 \ 1 \ 0 \ 1 \ 0 \ 0 \\ \hline S = 37 & = & 1 \ 0 \ 0 \ 1 \ 0 \ 1 \\ \hline C' = 48 & = & 0 \ 1 \ 1 \ 0 \ 0 \ 0 \end{array}$$

The i th bit of the sum S_i and the $(i+1)$ st bit of the carry C'_{i+1} is calculated using the equations

$$\begin{aligned} S_i &= A_i \oplus B_i \oplus C_i . \\ C'_{i+1} &= A_i B_i + A_i C_i + B_i C_i , \end{aligned}$$

in other words, a carry save adder cell is just a full-adder cell. A carry save adder, sometimes named a one-level CSA, is illustrated below for $k = 6$.

Carry Save Adder (CSA)



Since the input vectors A , B , and C are applied in parallel, the total delay of a carry save adder is equal to the total delay of a single FA cell. Thus, the addition of three integers to compute two integers requires a single FA delay. Furthermore, the CSA requires only k times the areas of FA cell, and scales up very easily by adding more parallel cells. The subtraction operation can also be performed by using 2's complement encoding. There are basically two disadvantages of the carry save adders:

- It does not really solve our problem of adding two integers and producing a single output. Instead, it adds three integers and produces two such that sum of these two is equal to the sum of three inputs. This method may not be suitable for application which only needs the regular addition.
- The sign detection is hard: When a number is represented as a carry-save pair (C, S) such that its actual value is $C + S$, we may not know the exact sign of total sum $C + S$. Unless the addition is performed in full length, the correct sign may never be determined.

Magnitude Comparator

- It is a combinational logic circuit.
- Digital Comparator is used to compare the value of two binary digits.
- There are two types of digital comparator (i) Identity Comparator (ii) Magnitude Comparator.
- **IDENTITY COMPARATOR:** This comparator has only one output terminal for when $A=B$, either $A=B=1$ (High) or $A=B=0$ (Low)
- **MAGNITUDE COMPARATOR:** This Comparator has three output terminals namely $A>B$, $A=B$, $A<B$. Depending on the result of comparison, one of these output will be high (1)
- Block Diagram of Magnitude Comparator is shown in Fig. 1

Magnitude Comparator

BLOCK DIAGRAM OF MAGNITUDE COMPARATOR

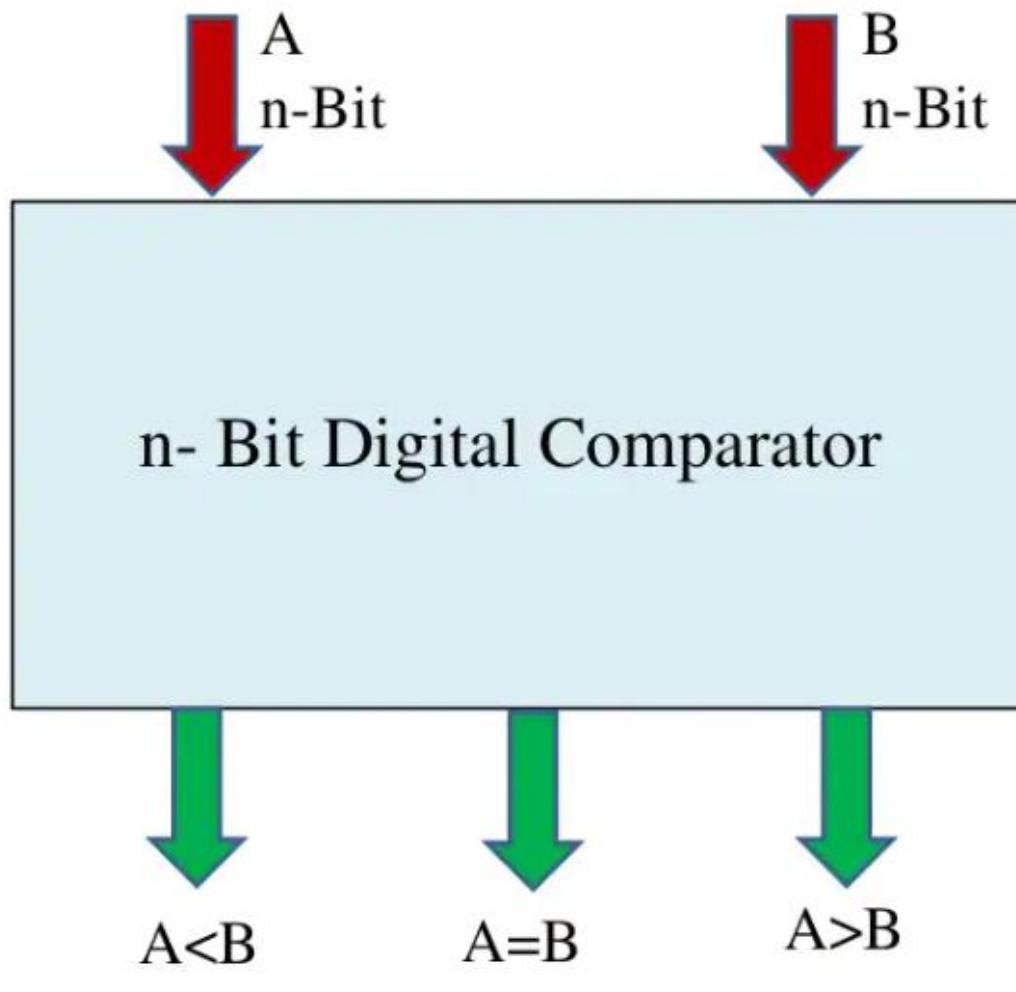


Fig. 1

Magnitude Comparator

1- Bit Magnitude Comparator:

- This magnitude comparator has two inputs A and B and three outputs $A < B$, $A = B$ and $A > B$.
- This magnitude comparator compares the two numbers of single bits.
- Truth Table of 1-Bit Comparator

INPUTS		OUTPUTS		
A	B	Y_1 ($A < B$)	Y_2 ($A = B$)	Y_3 ($A > B$)
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Magnitude Comparator

K-Maps For All Three Outputs :

		B	\bar{B}	B
		A	0	1
A	\bar{A}	0	0	1
	1	0	0	0

K-Map for Y_1 : $A < B$

$$Y_1 = \bar{A}B$$

		B	\bar{B}	B
		A	0	1
\bar{A}	0	1	0	0
	1	0	0	1

K-Map for Y_2 : $A=B$

$$Y_2 = \bar{A}\bar{B} + AB$$

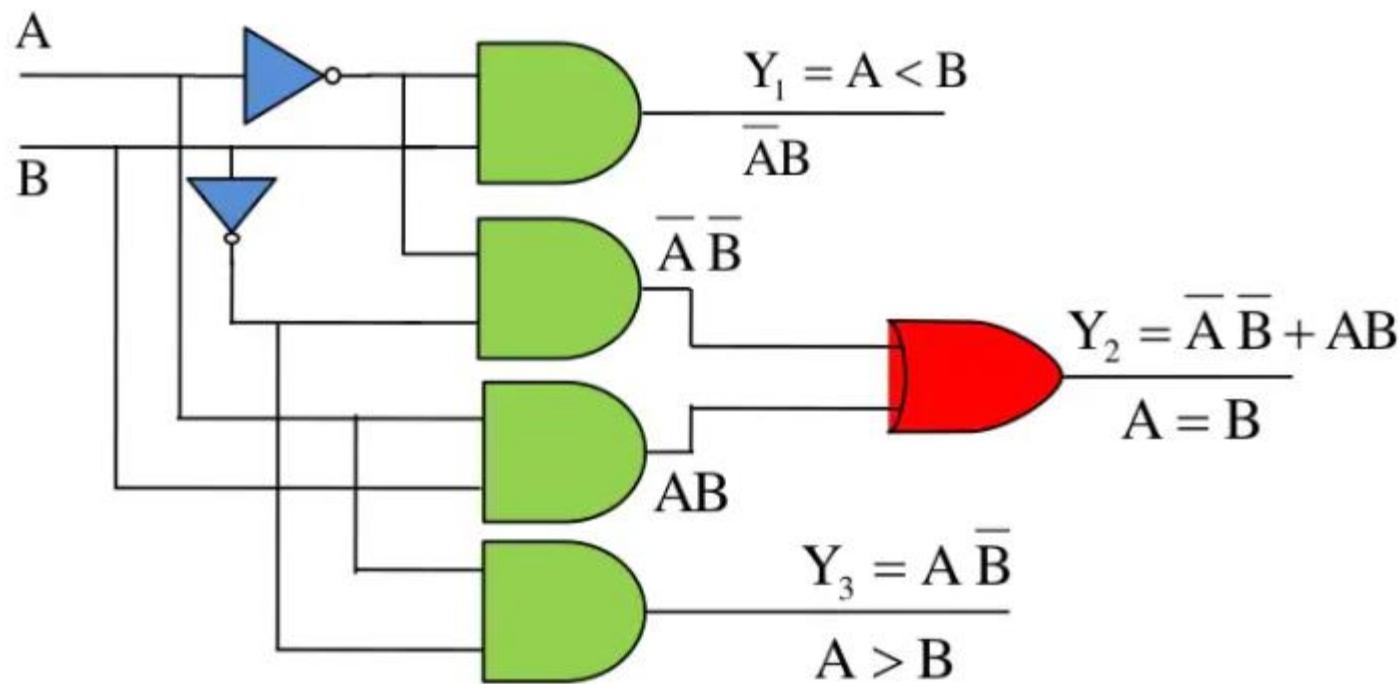
		B	\bar{B}	B
		A	0	1
\bar{A}	0	0	0	0
	1	1	0	0

K-Map for Y_2 : $A > B$

$$Y_2 = A\bar{B}$$

Magnitude Comparator

Realization of One Bit Comparator



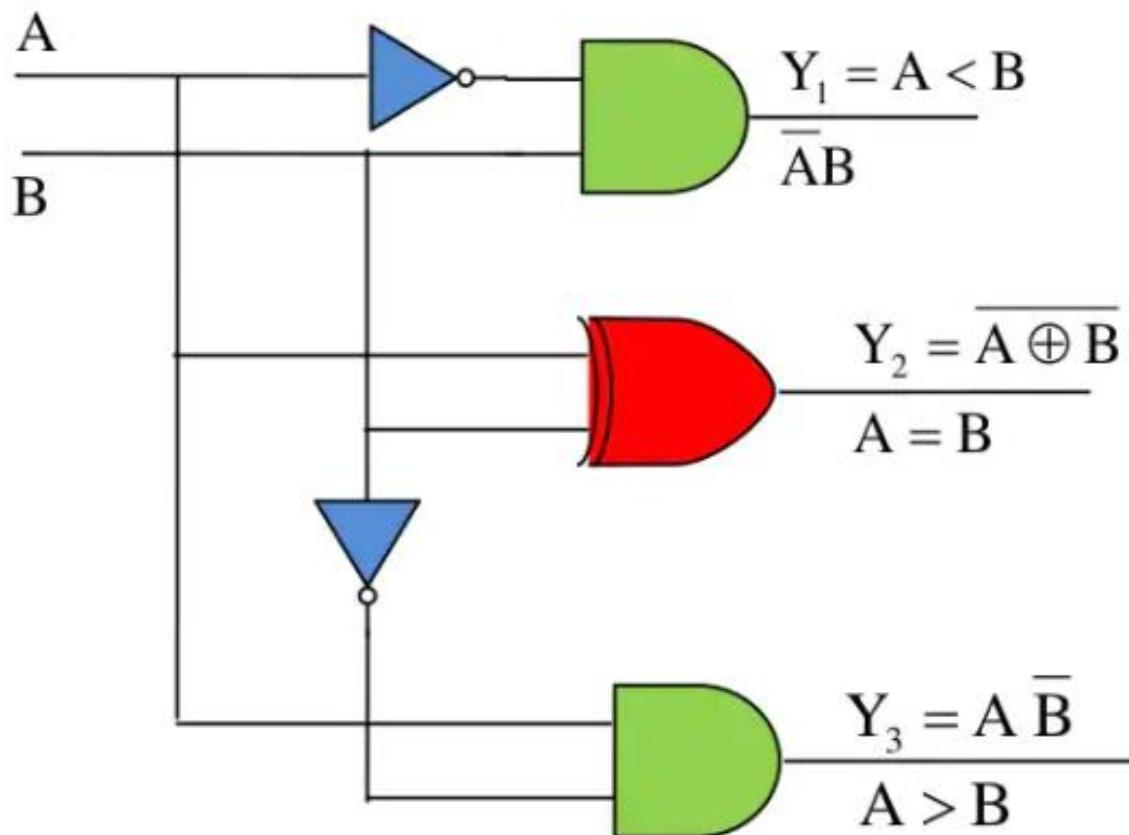
$$Y_1 = \overline{A}B$$

$$Y_2 = \overline{A}\overline{B} + AB$$

$$Y_3 = A\overline{B}$$

Magnitude Comparator

Realization of by Using AND , EX-NOR gates



Magnitude Comparator

2-Bit Comparator:

- A comparator which is used to compare two binary numbers each of two bits is called a 2-bit magnitude comparator.
- Fig. 2 shows the block diagram of 2-Bit magnitude comparator.
- It has four inputs and three outputs.
- Inputs are A_0, A_1, B_0 and B_1 and Outputs are Y_1, Y_2 and Y_3

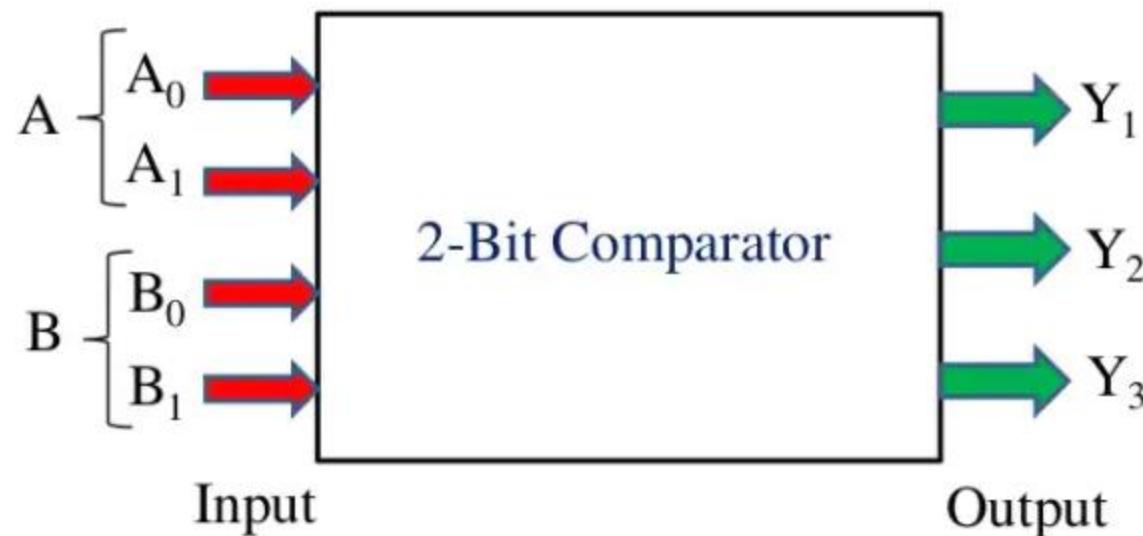


Fig. 2

Magnitude Comparator

GREATER THAN (A>B)

A_1	A_0	B_1	B_0
1	0	0	1
1	1	1	0
0	1	0	0

1. If $A_1 = 1$ and $B_1 = 0$ then $A > B$
2. If A_1 and B_1 are same, i.e $A_1 = B_1 = 1$ or $A_1 = B_1 = 0$ and $A_0 = 1$, $B_0 = 0$ then $A > B$

LESS THAN (A<B)

Similarly,

1. If $A_1 = B_1 = 1$ and $A_0 = 0$, $B_0 = 1$, then $A < B$
2. If $A_1 = B_1 = 0$ and $A_0 = 0$, $B_0 = 1$ then $A < B$

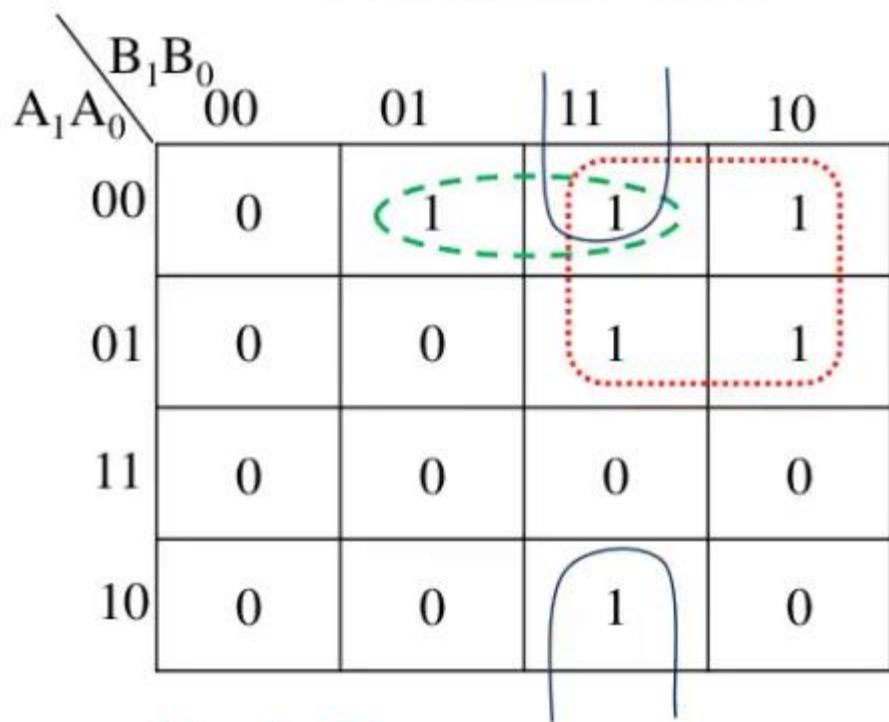
Magnitude Comparator

TRUTH TABLE

INPUT				OUTPUT		
A_1	A_0	B_1	B_0	$Y_1 = A < B$	$Y_2 = (A = B)$	$Y_3 = A > B$
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

Magnitude Comparator

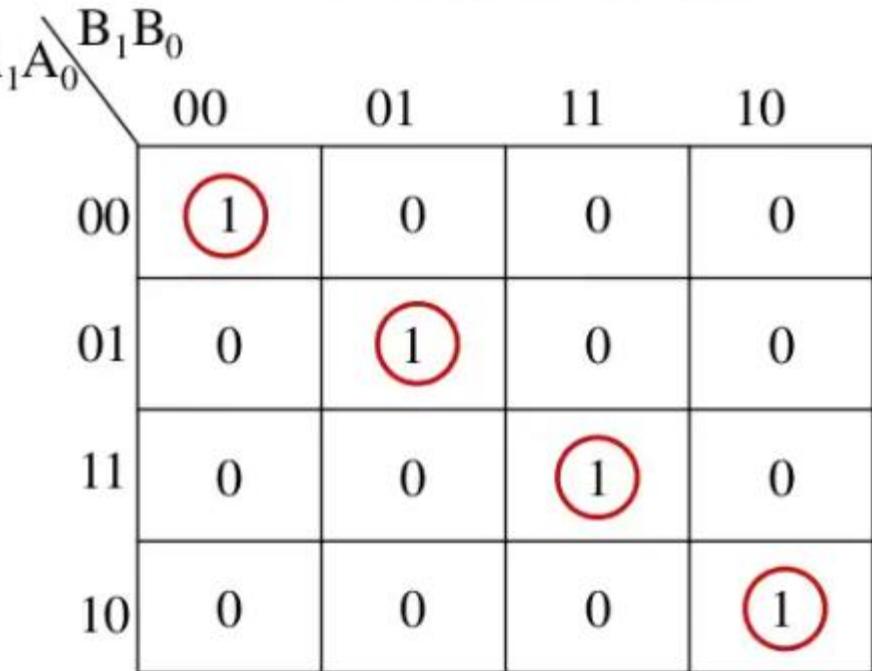
K-Map for A < B:



For A < B

$$Y_1 = \overline{A}_1 \overline{A}_0 B_0 + \overline{A}_1 B_1 + \overline{A}_0 B_1 B_0$$

K-Map for A = B:

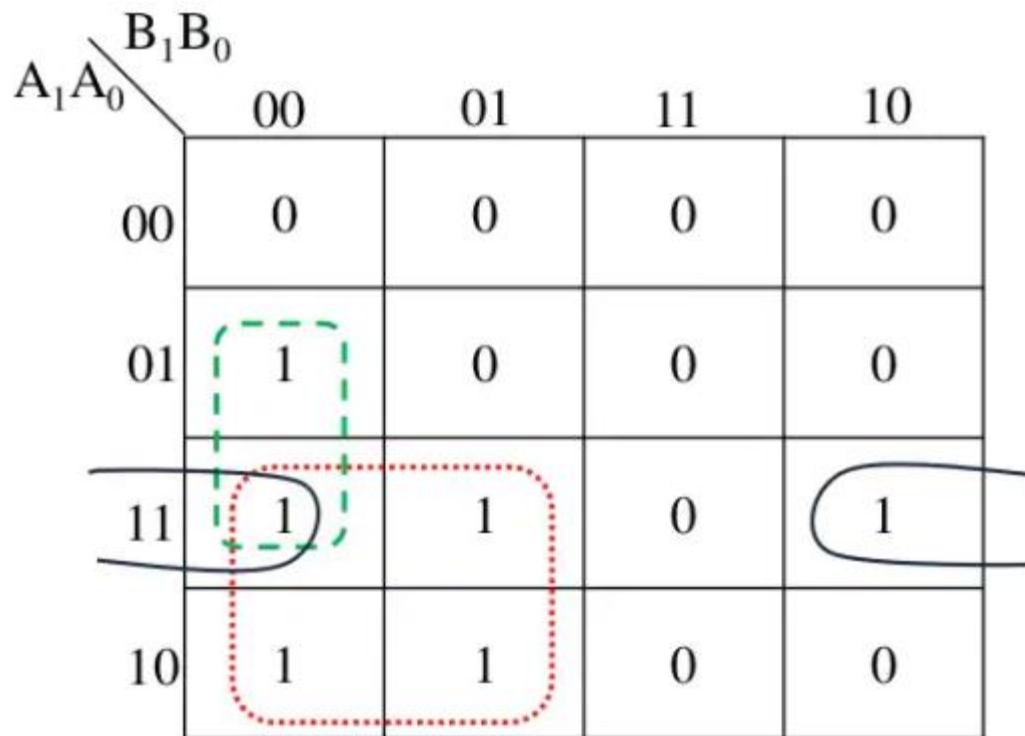


For A = B

$$Y_2 = \overline{A}_1 \overline{A}_0 \overline{B}_1 \overline{B}_0 + \overline{A}_1 A_0 \overline{B}_1 B_0 + A_1 A_0 B_1 B_0 + A_1 \overline{A}_0 B_1 \overline{B}_0$$

Magnitude Comparator

K-Map For A>B



$$Y_3 = A_0 \overline{B_1} \overline{B_0} + A_1 \overline{B_1} + A_1 A_0 \overline{B_0}$$

Magnitude Comparator

For A=B From K-Map

$$Y_2 = \overline{A_1} \overline{A_0} \overline{B_1} \overline{B_0} + \overline{A_1} A_0 \overline{B_1} B_0 + A_1 \overline{A_0} B_1 B_0 + A_1 \overline{A_0} B_1 \overline{B_0}$$

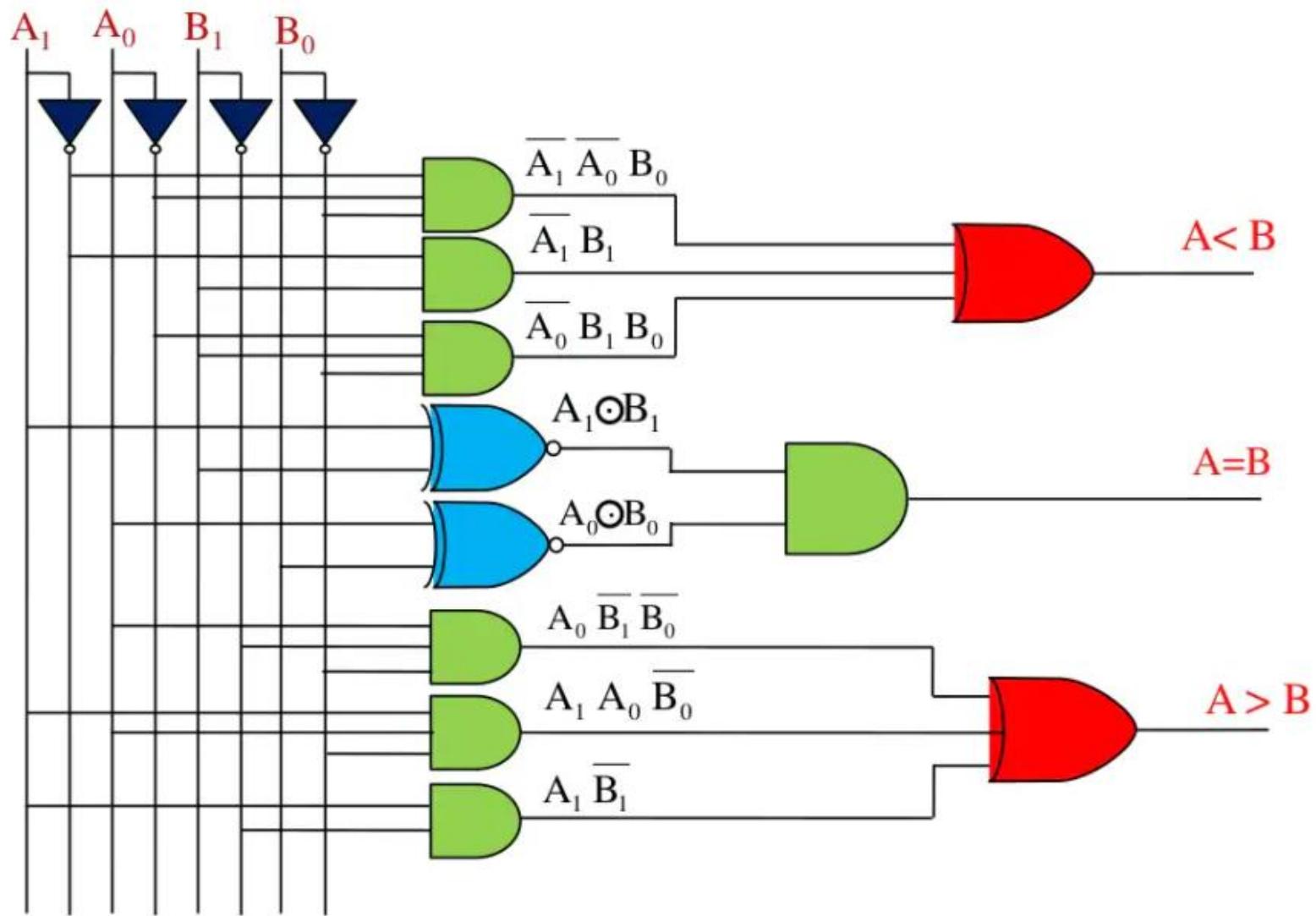
$$Y_2 = \overline{A_0} B_0 (A_1 \overline{B_1} + A_1 B_1) + A_0 \overline{B_0} (\overline{A_1} \overline{B_1} + A_1 B_1)$$

$$Y_2 = (\overline{A_1} \overline{B_1} + A_1 B_1) (\overline{A_0} \overline{B_0} + A_0 B_0)$$

$$Y_2 = (A_1 \odot B_1) (A_0 \odot B_0)$$

Magnitude Comparator

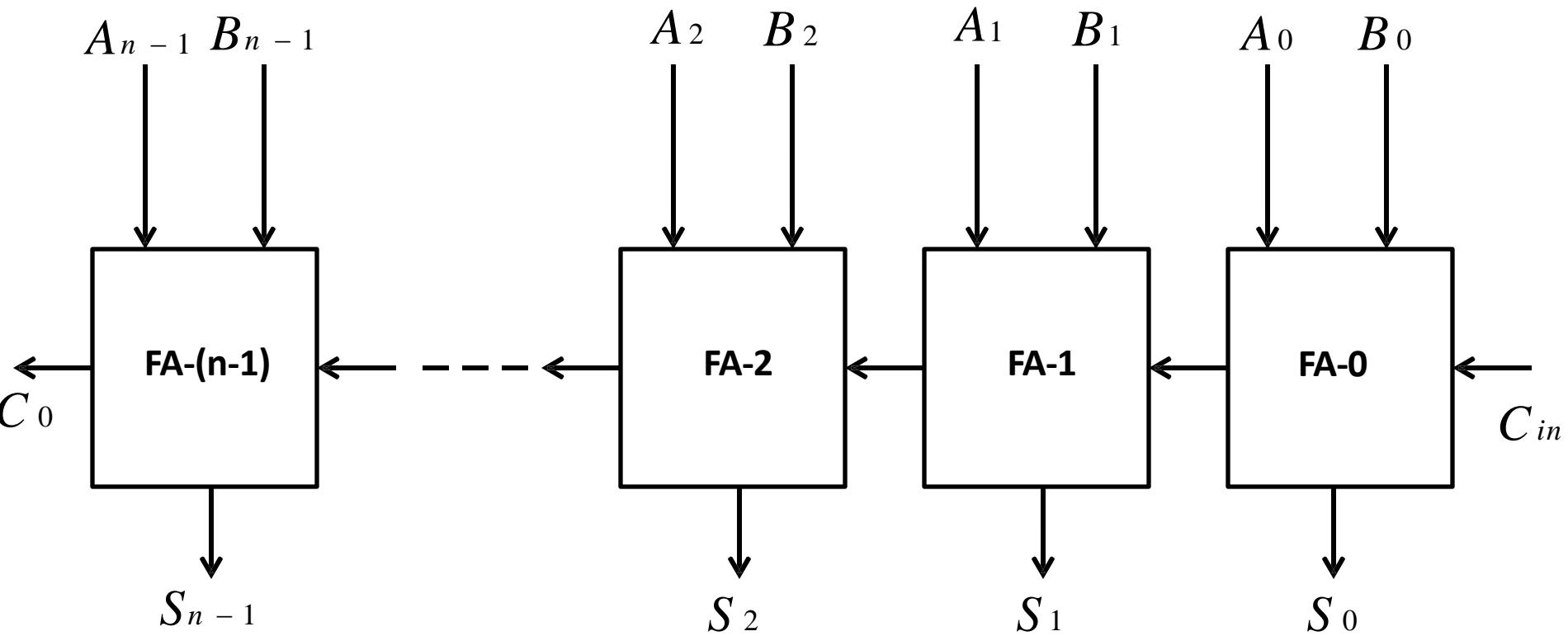
LOGIC DIAGRAM OF 2-BIT COMPARATOR:



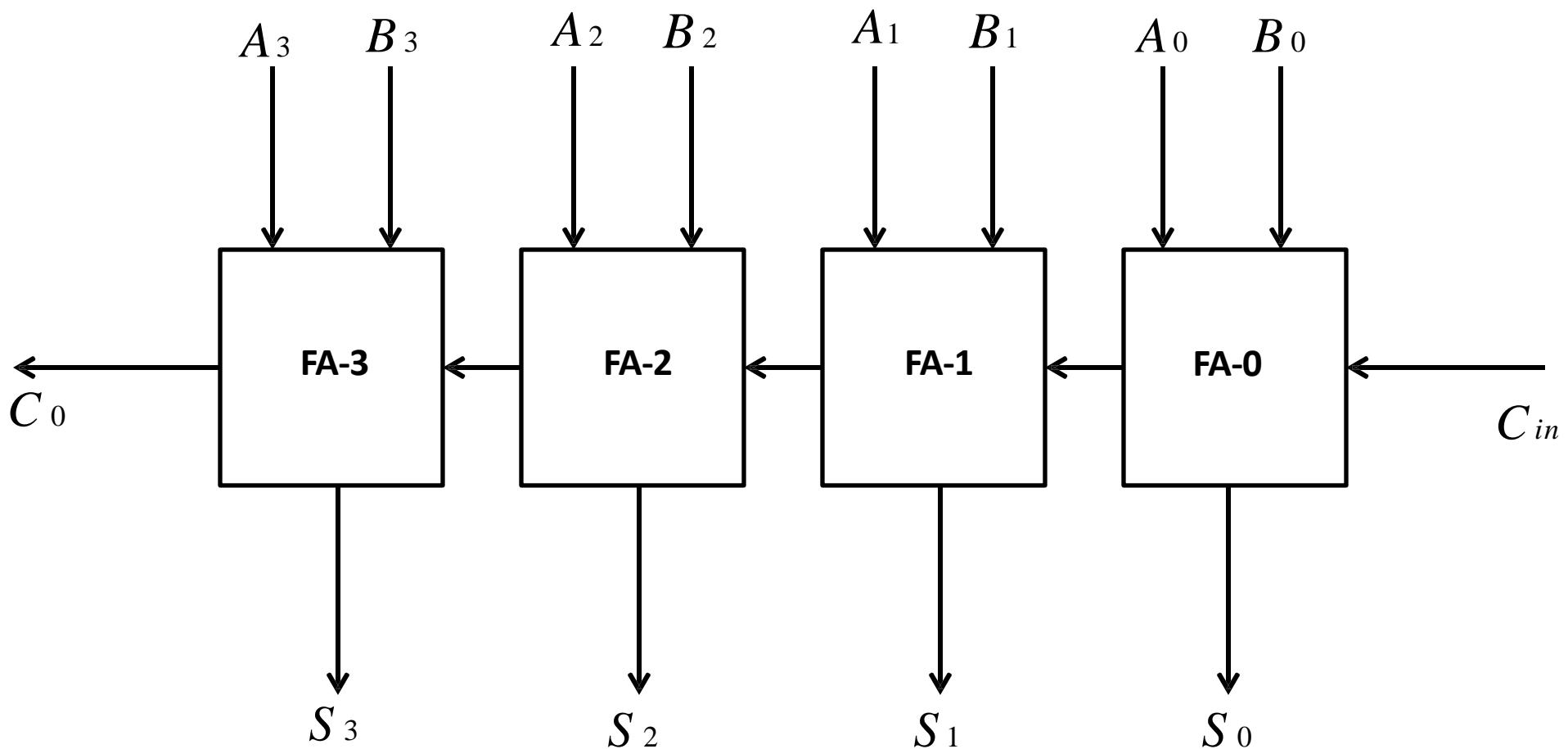
N – Bit Parallel Adder

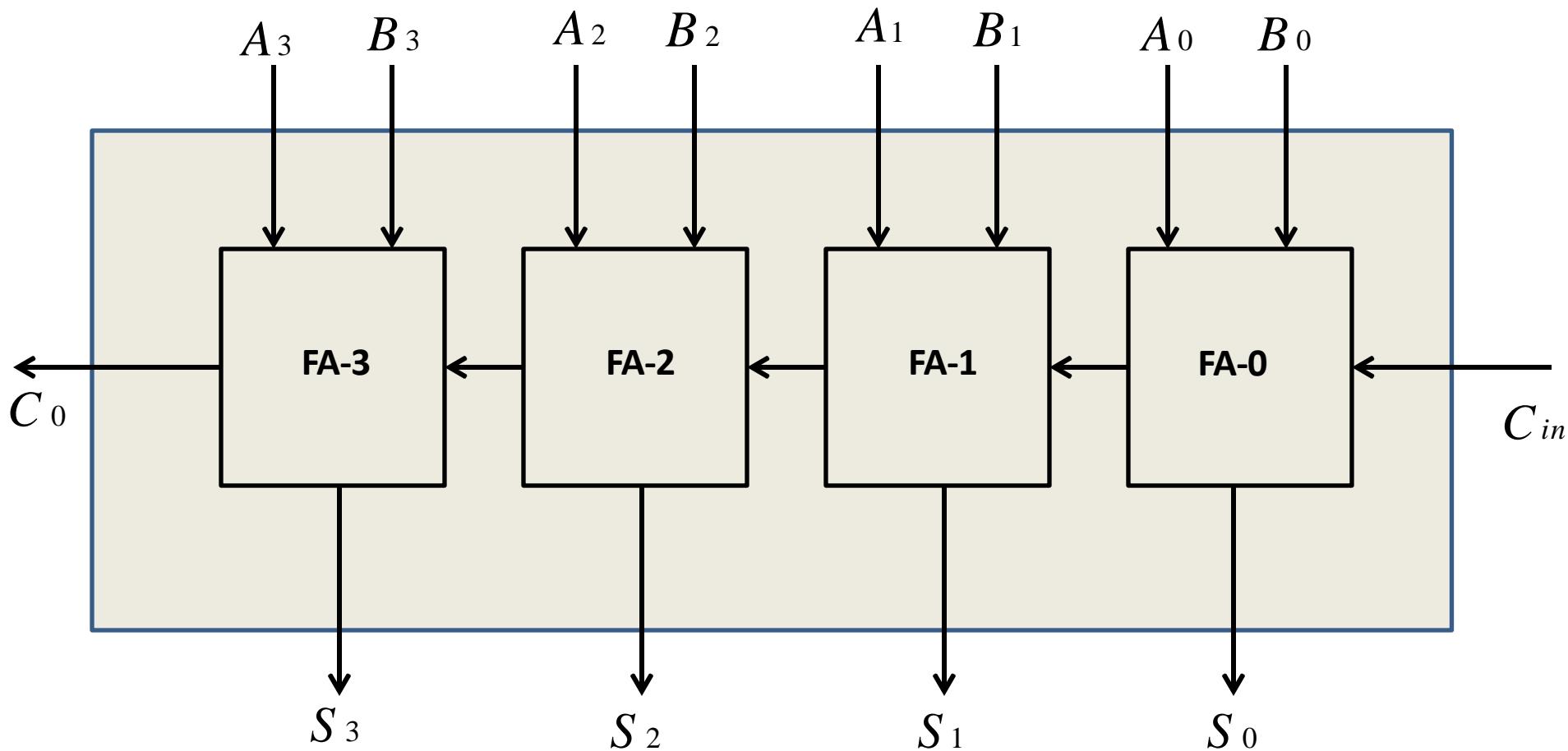
- ✓ The full adder is capable of adding two single digit binary numbers along with a carry input.
- ✓ But in practice we need to add binary numbers which are much longer than one bit.
- ✓ To add two n-bit binary numbers we need to use the n-bit parallel adder.
- ✓ It uses a number of full adders in cascade.
- ✓ The carry output of the previous full adder is connected to the carry input of the next full adder..

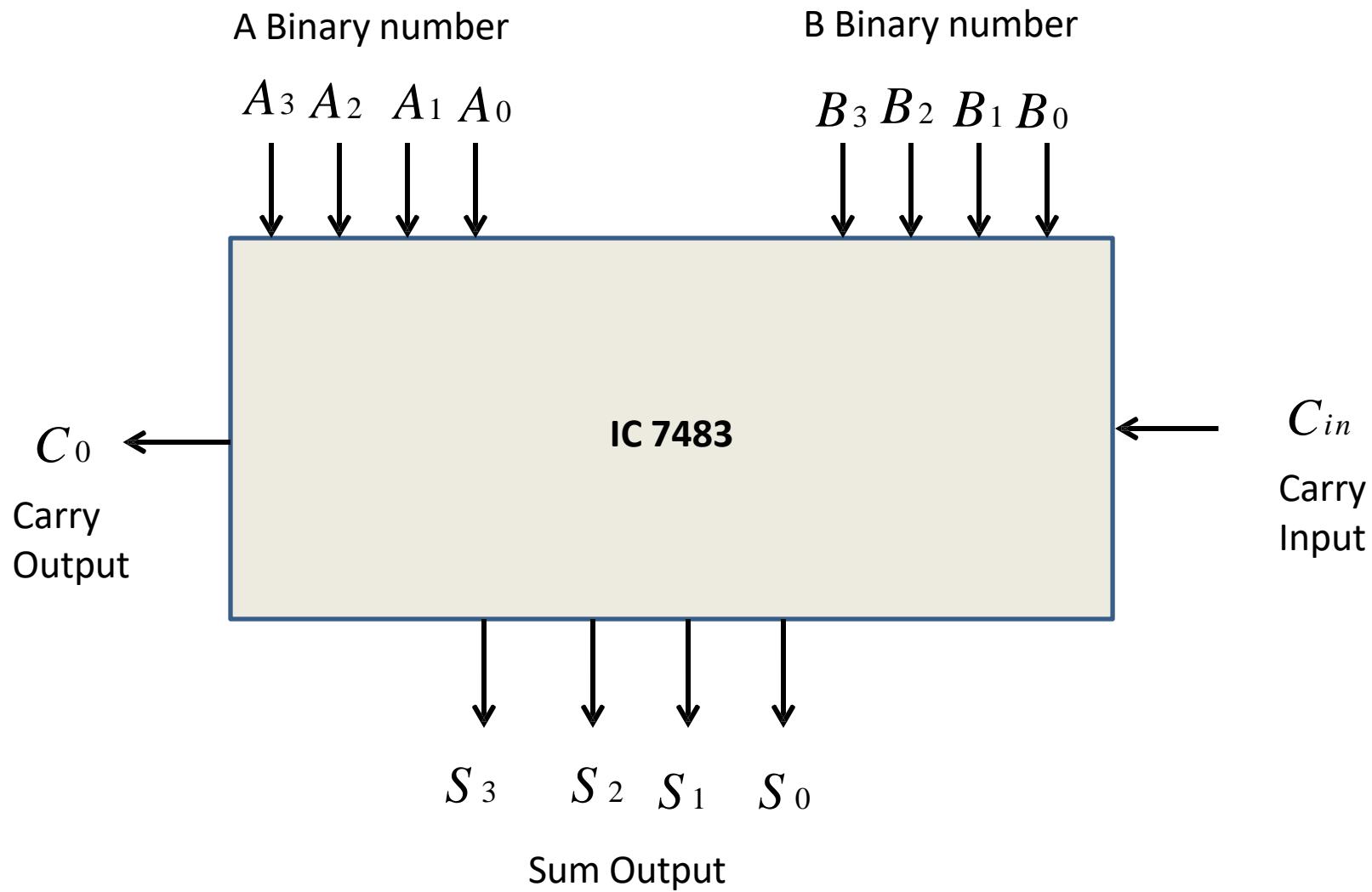
N – Bit Parallel Adder



4 – Bit Parallel Adder using full adder

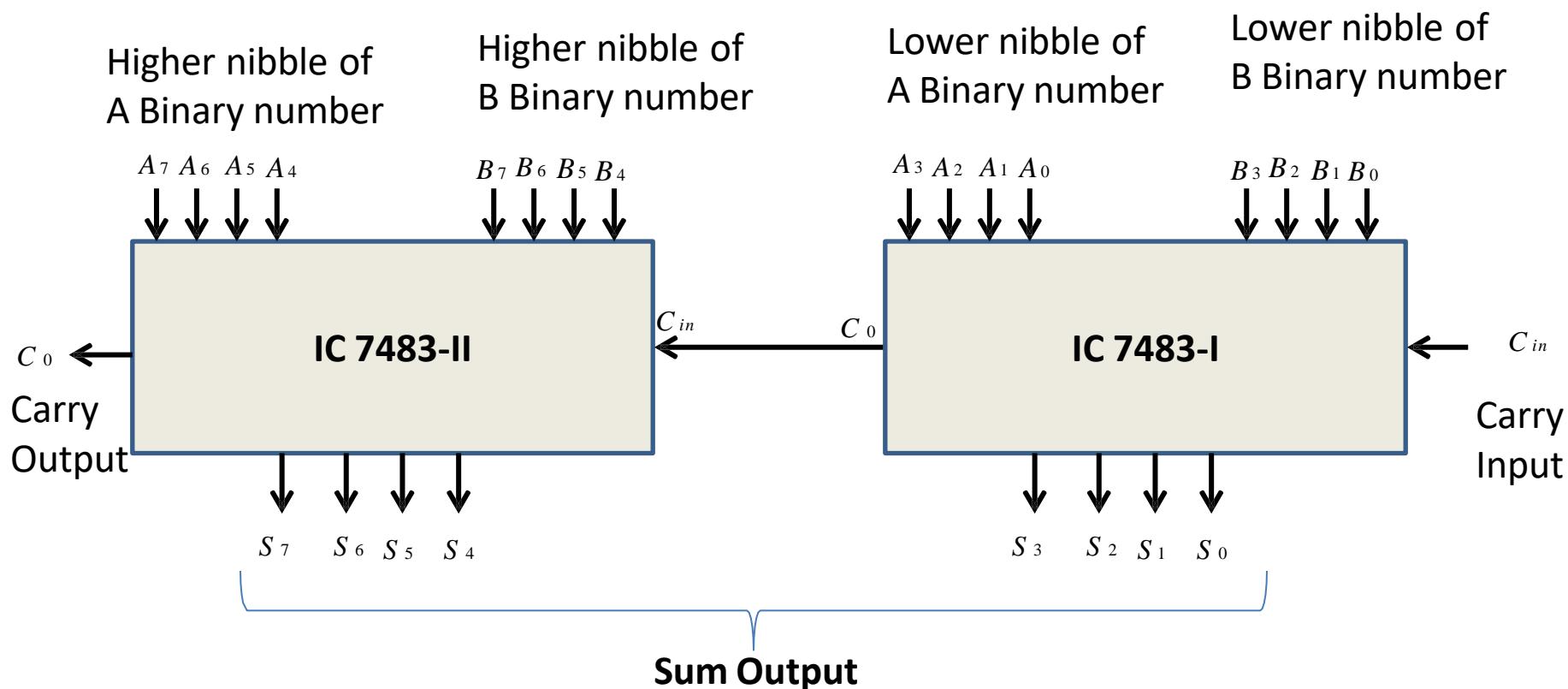






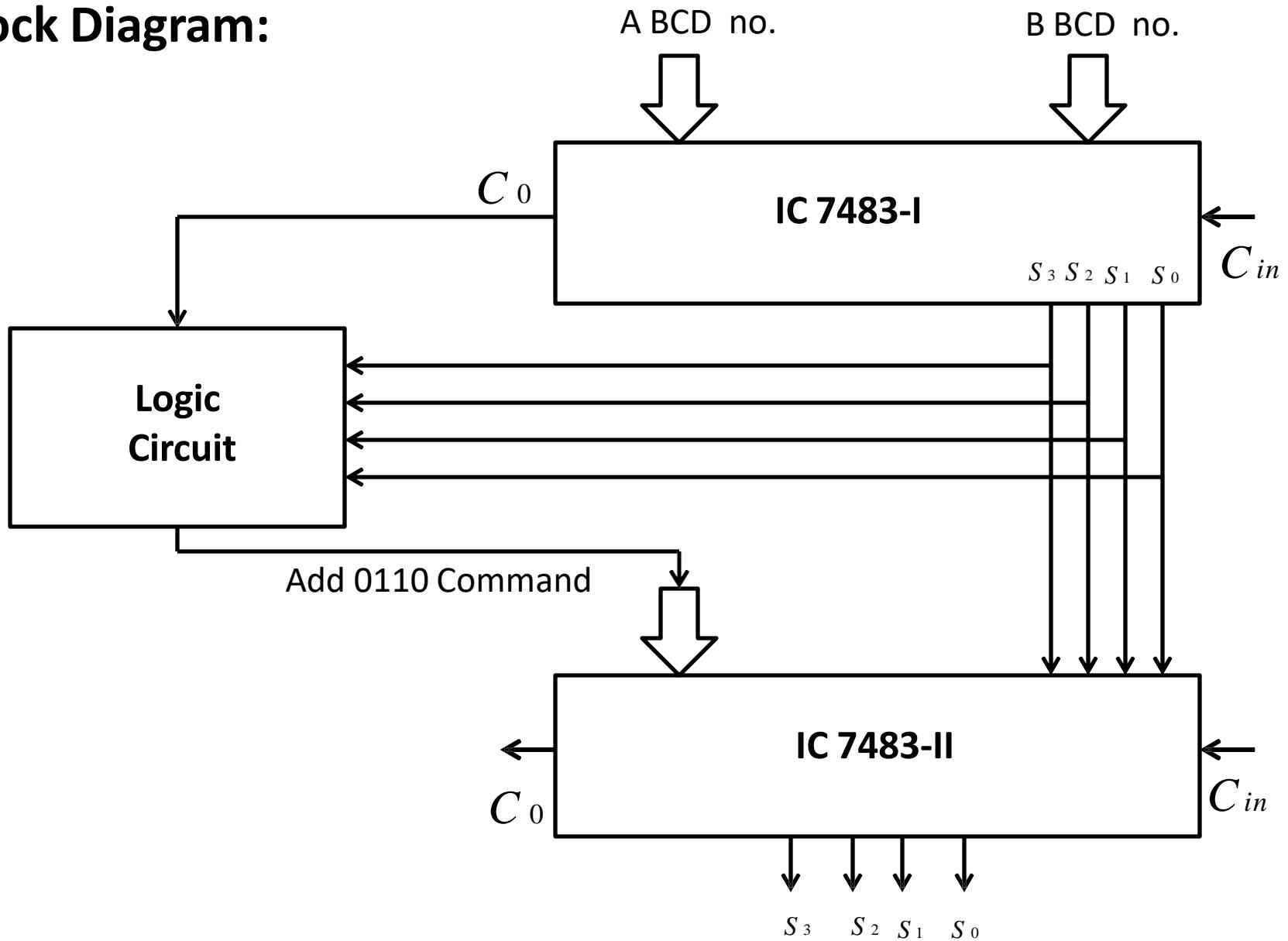
Cascading of IC 7483

- ✓ If we want to add two 8 bit binary numbers using 4 bit binary parallel adder IC 7483, then we have to cascade the two ICs in following way



Design of 1 Digit BCD Adder

Block Diagram:



Design of 1 Digit BCD Adder

As we know BCD addition rules, we understand that the 4 bit BCD adder should consists of following:

- ✓ A 4 bit binary adder to add the given two (4 bit numbers).
- ✓ A combinational logic circuit to check if sum is greater than 9 or carry 1.
- ✓ One more 4 bit binary adder to add 0110 to the invalid BCD sum or if carry is 1

Design of 1 Digit BCD Adder

Logic Table for design of Logic circuit:

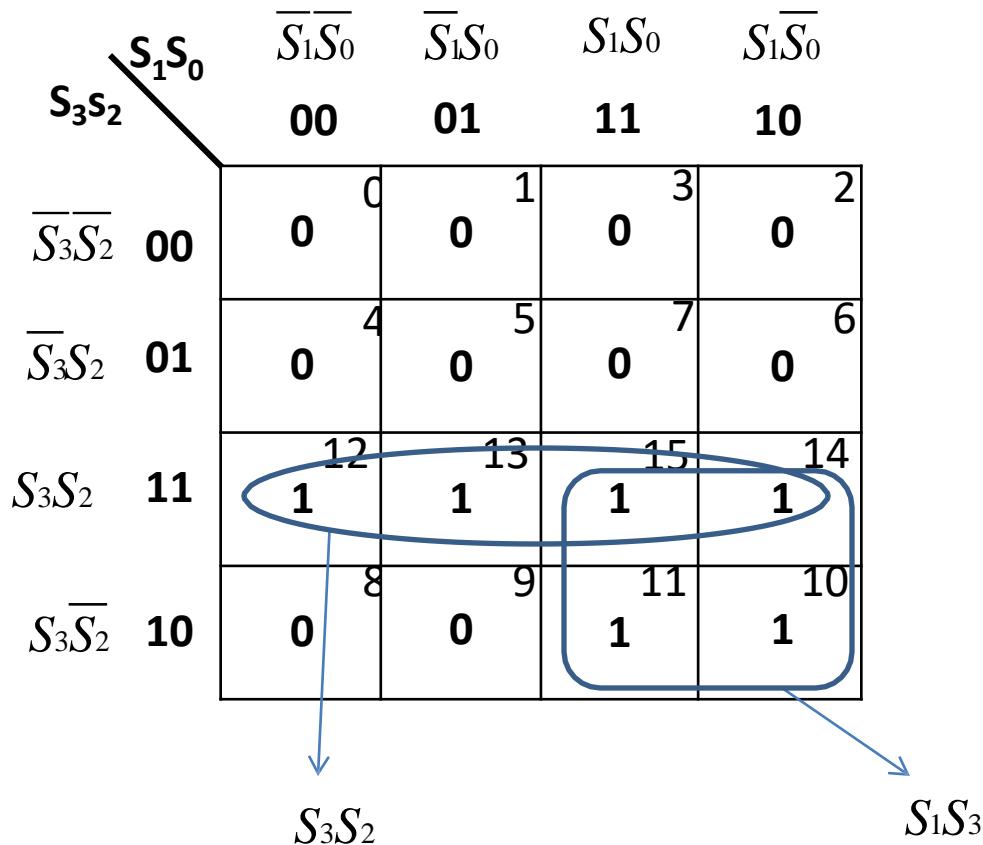
Inputs				Y
S_3	S_2	S_1	S_0	
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0

Inputs				Y
S_3	S_2	S_1	S_0	
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Sum is
invalid
BCD
Number
 $Y=1$

Design of 1 Digit BCD Adder

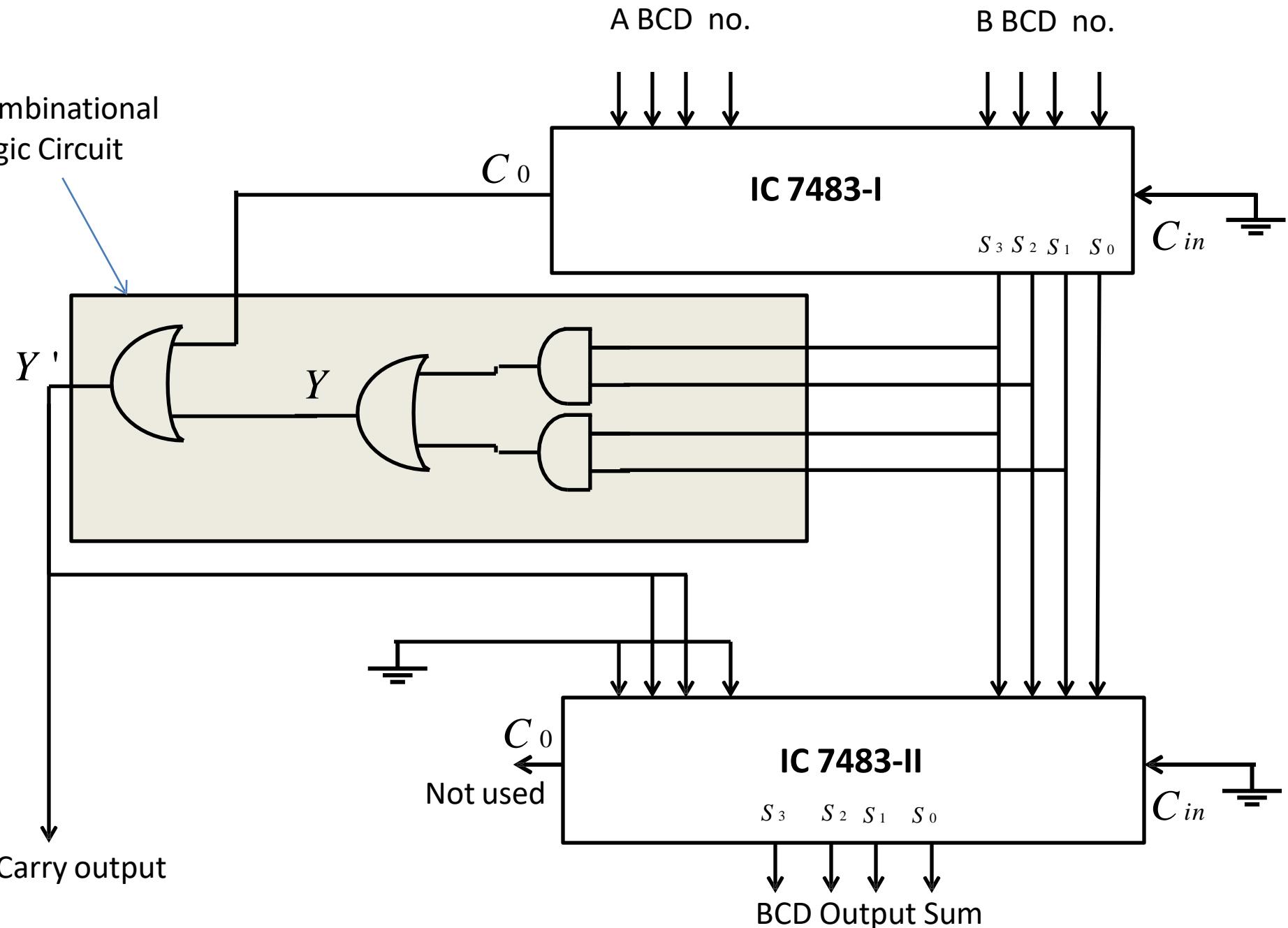
K-map for Logic circuit:



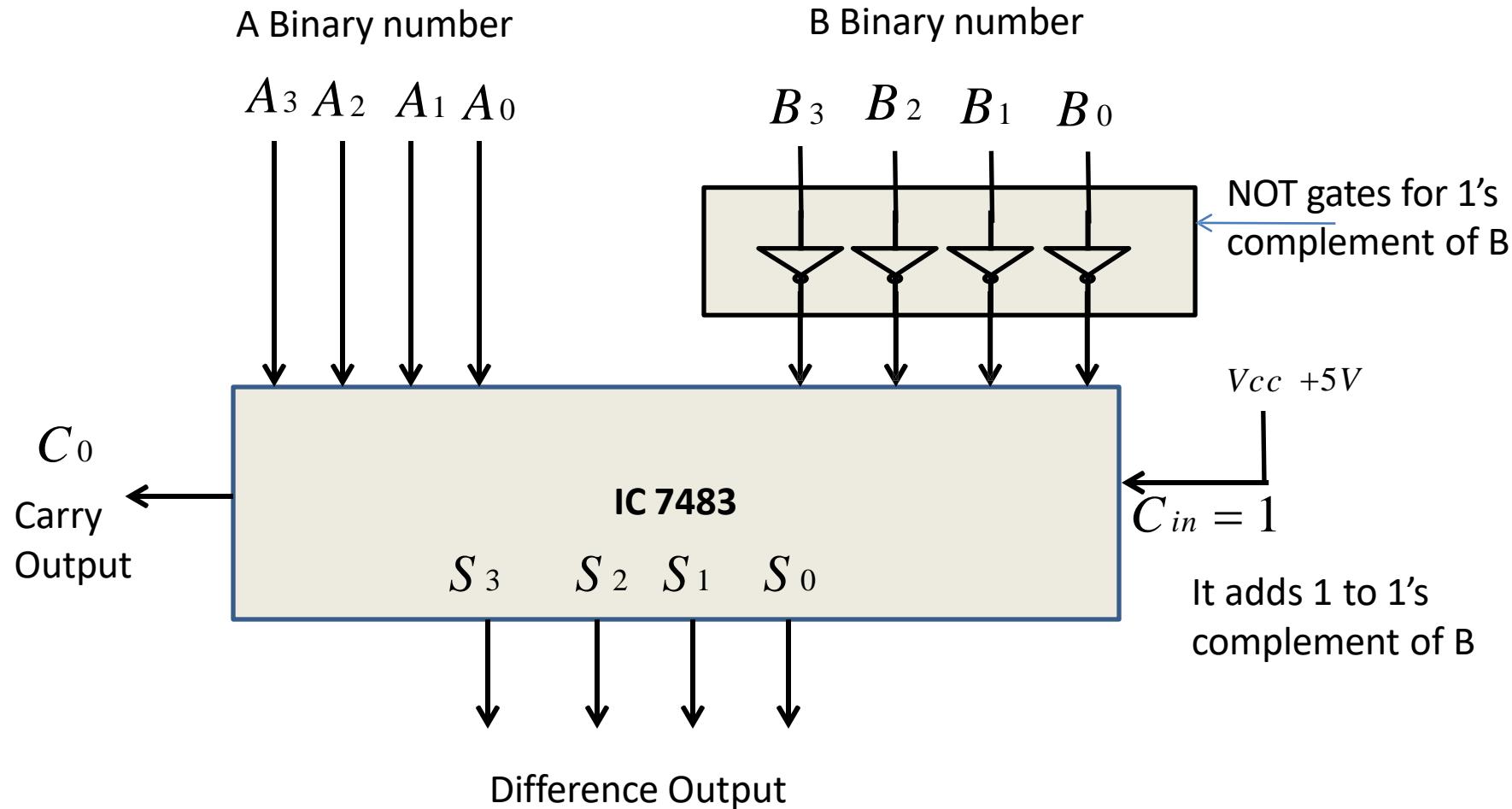
$$Y = S_3S_2 + S_3S_1$$

Design of 1 Digit BCD Adder

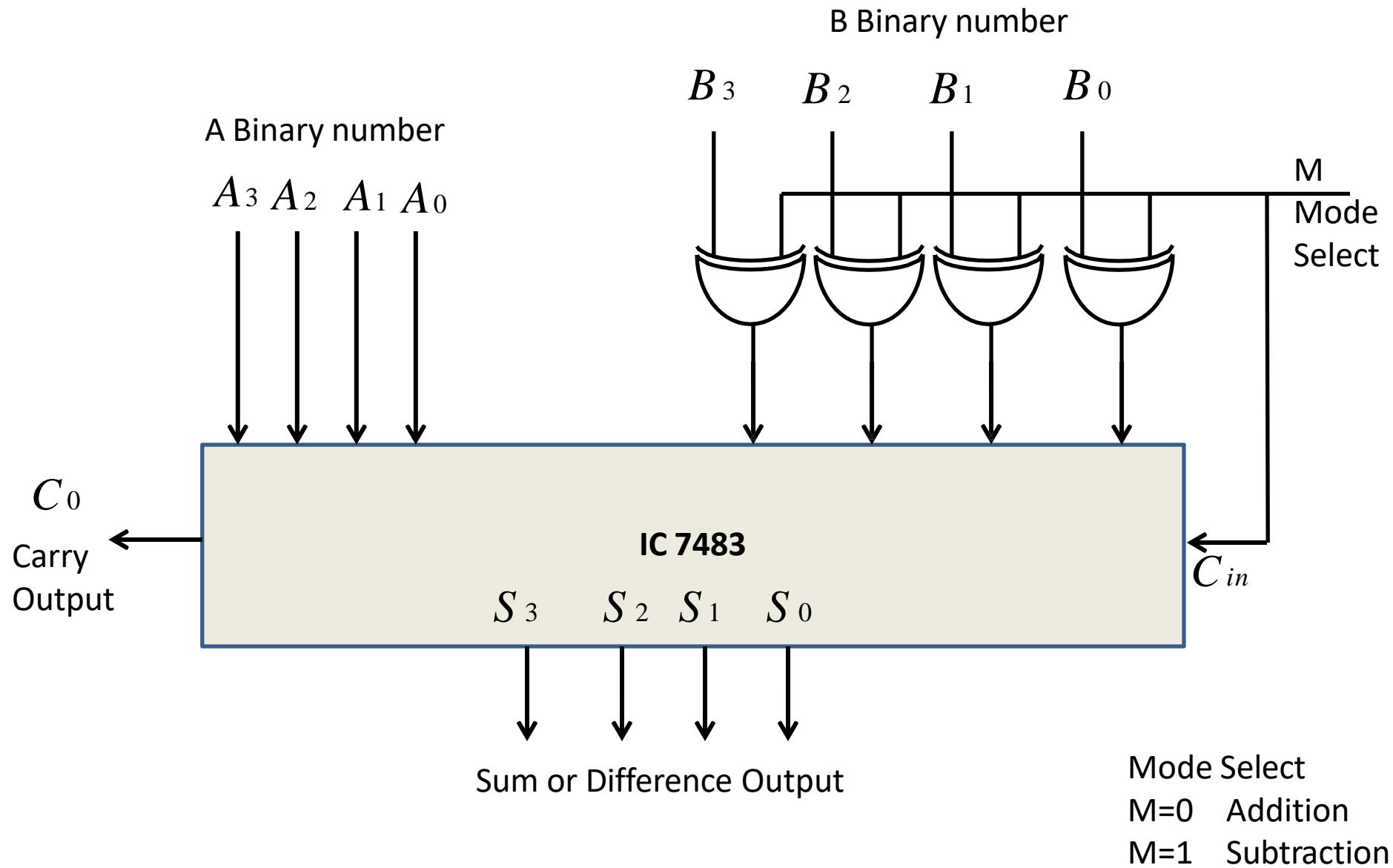
Combinational Logic Circuit



4 Bit Binary Parallel Subtractor using IC 7483



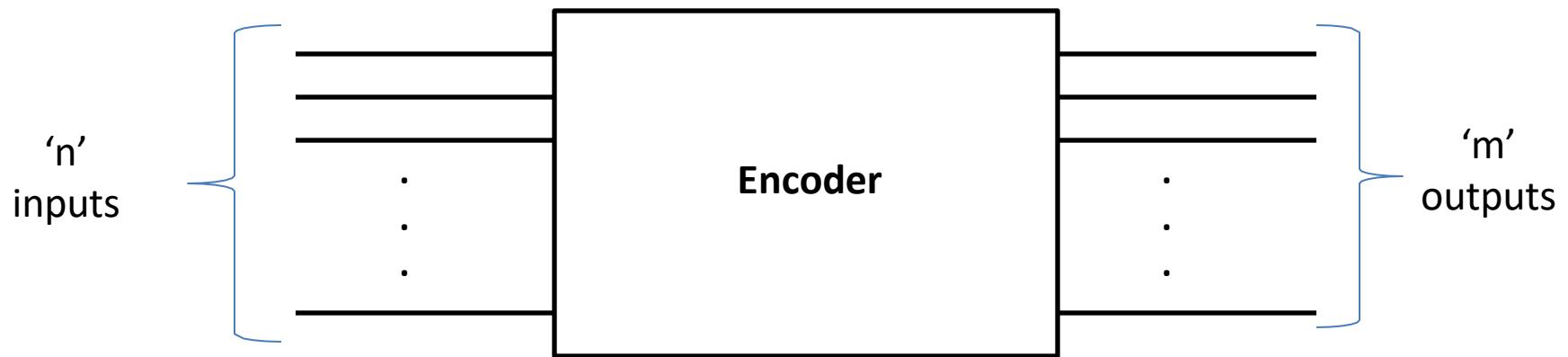
IC 7483 as Parallel Adder/Subtractor



Encoder

- ✓ Encoder is a combinational circuit which is designed to perform the inverse operation of decoder.
- ✓ An encoder has ‘n’ number of input lines and ‘m’ number of output lines.
- ✓ An encoder produces an m bit binary code corresponding to the digital input number.
- ✓ The encoder accepts an n input digital word and converts it into m bit another digital word

Encoder



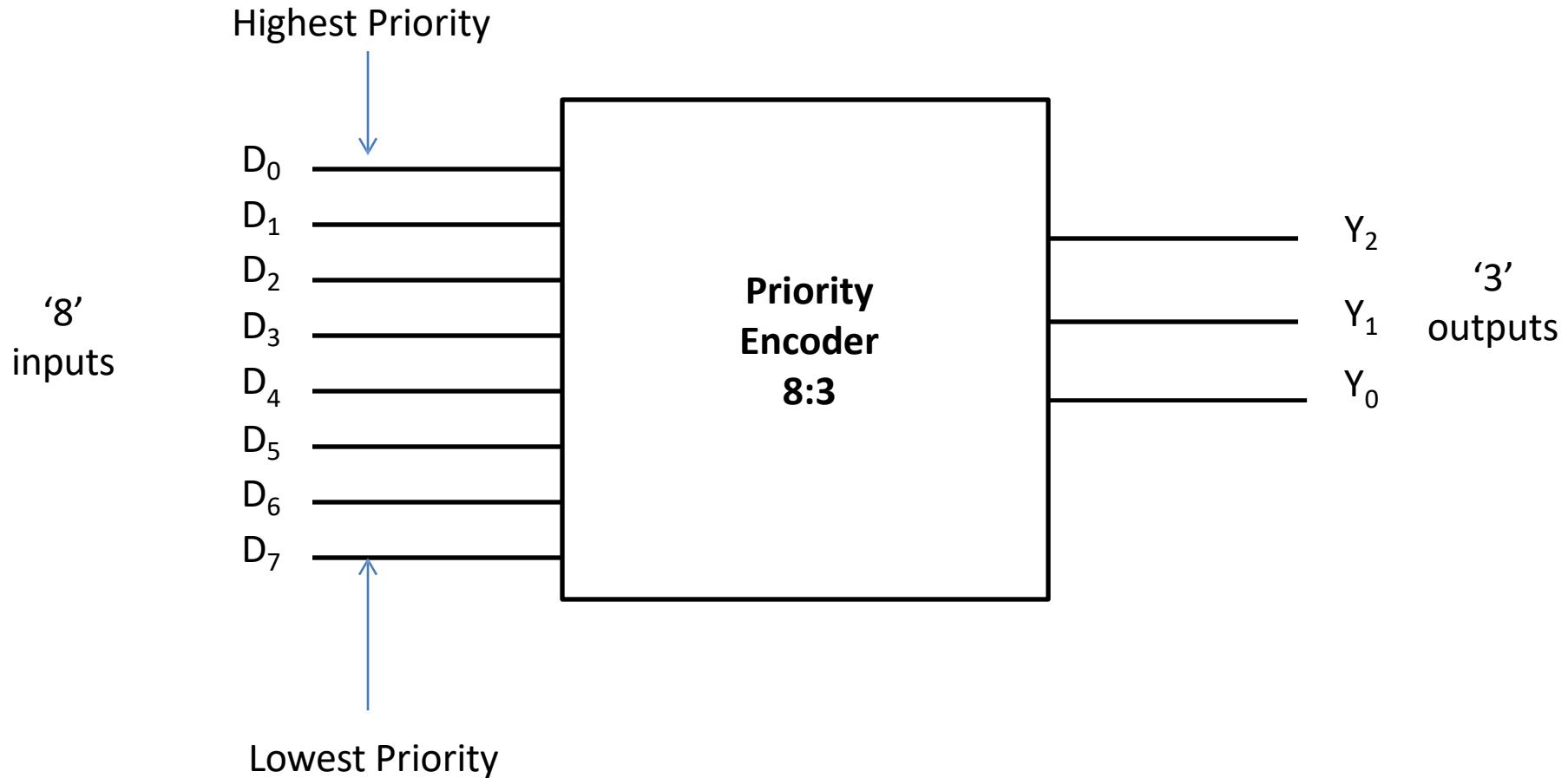
Types of Encoders

- ✓ Priority Encoder
- ✓ Decimal to BCD Encoder
- ✓ Octal to BCD Encoder
- ✓ Hexadecimal to Binary Encoder

Priority Encoder

- ✓ This is a special type of encoder.
- ✓ Priorities are given to the input lines.
- ✓ If two or more input lines are “1” at the same time, then the input line with highest priority will be considered.

Priority Encoder 8:3

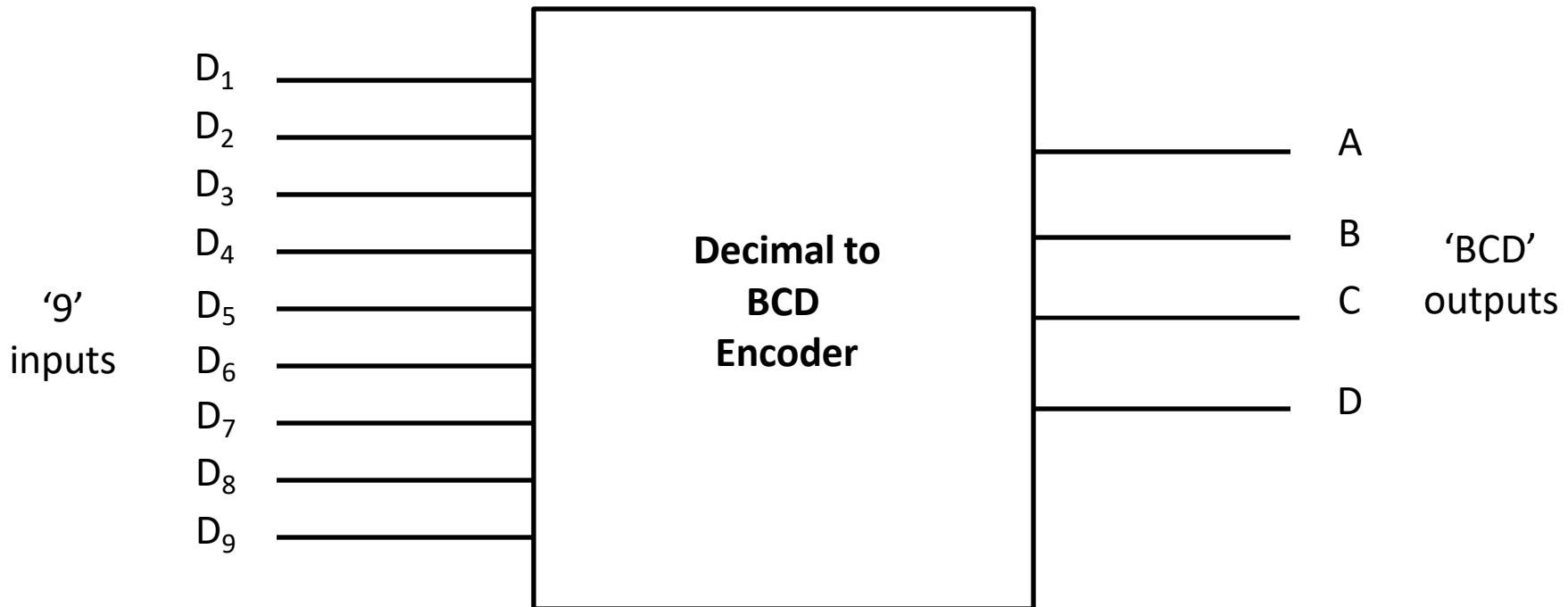


Priority Encoder 8:3

Truth Table:

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	X	X	X
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1

Decimal to BCD Encoder



Decimal to BCD Encoder

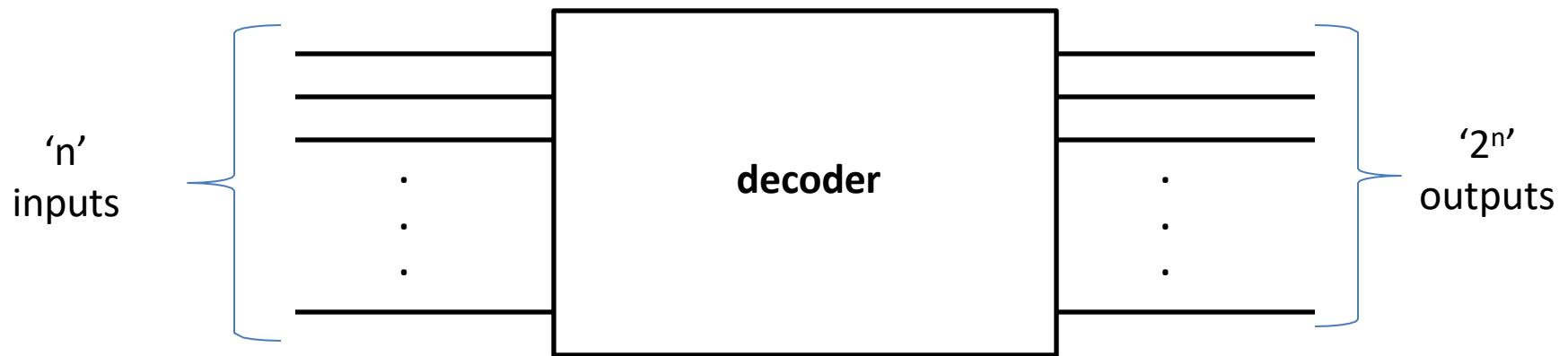
Truth Table:

Inputs										Outputs			
D ₉	D ₈	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁		D	C	B	A
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	X	0	0	1	0	0
0	0	0	0	0	0	1	X	X	0	0	1	1	1
0	0	0	0	0	1	X	X	X	0	1	0	0	0
0	0	0	0	1	X	X	X	X	0	1	0	0	1
0	0	0	1	X	X	X	X	X	0	1	1	0	0
0	0	1	X	X	X	X	X	X	0	1	1	1	1
0	1	X	X	X	X	X	X	X	1	0	0	0	0
1	X	X	X	X	X	X	X	X	1	0	0	0	1

Decoder

- ✓ Decoder is a combinational circuit which is designed to perform the inverse operation of encoder.
- ✓ An decoder has ‘n’ number of input lines and maximum ‘ 2^n ’ number of output lines.
- ✓ Decoder is identical to a demultiplexer without data input.

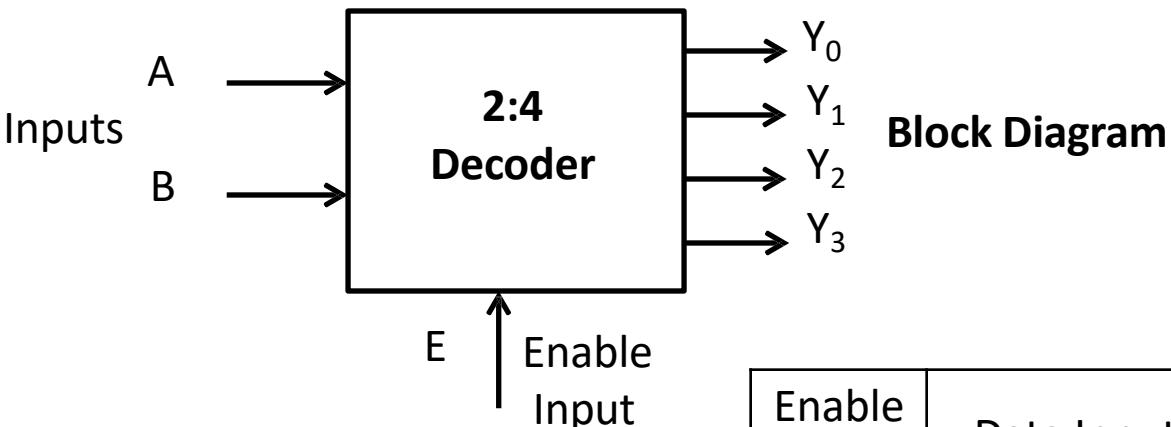
Decoder



Types of Decoders

- ✓ 2 to 4 line Decoder
- ✓ 3 to 8 line Decoder
- ✓ BCD to 7 Segment Decoder

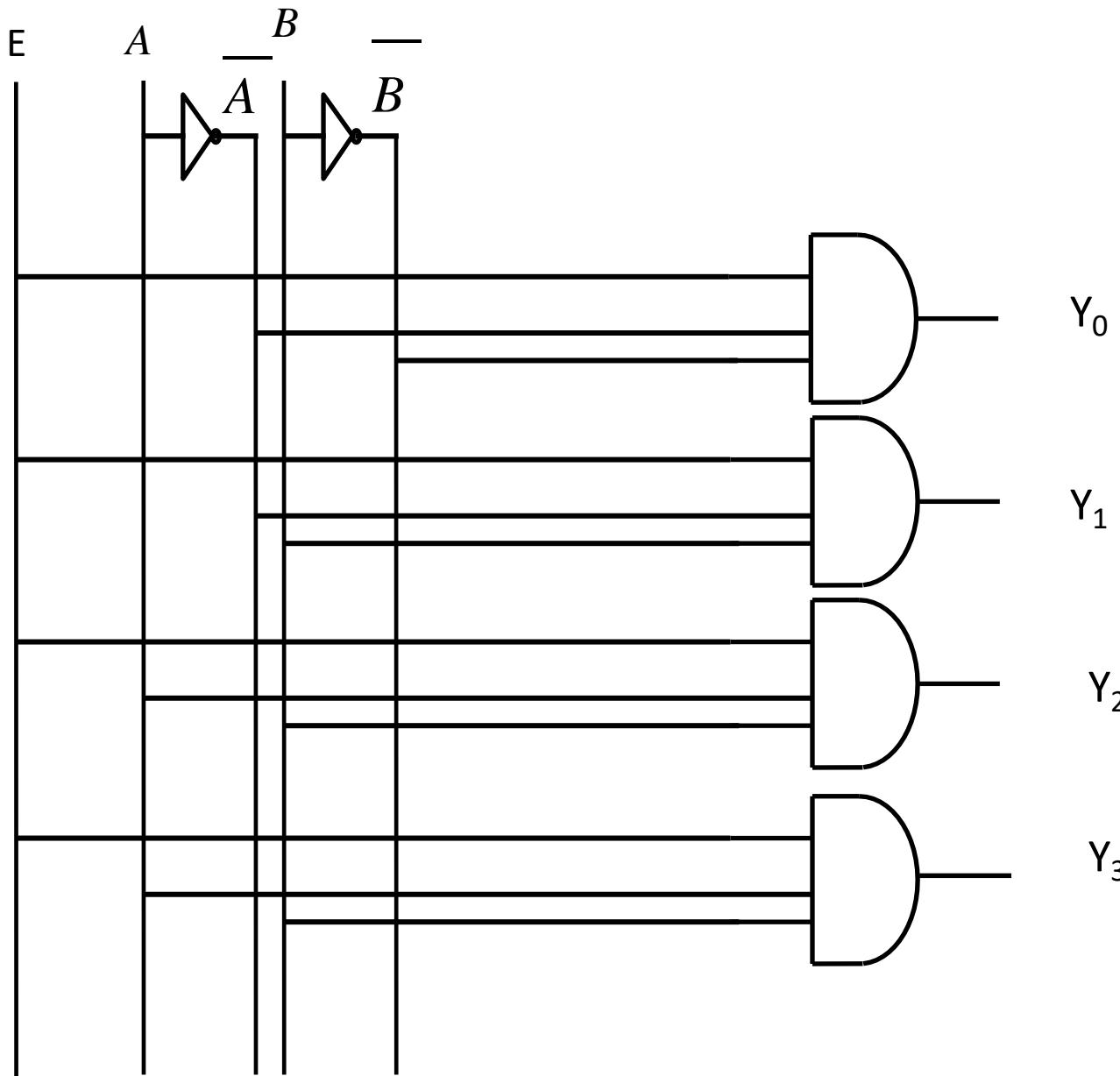
2 to 4 Line Decoder



Truth Table

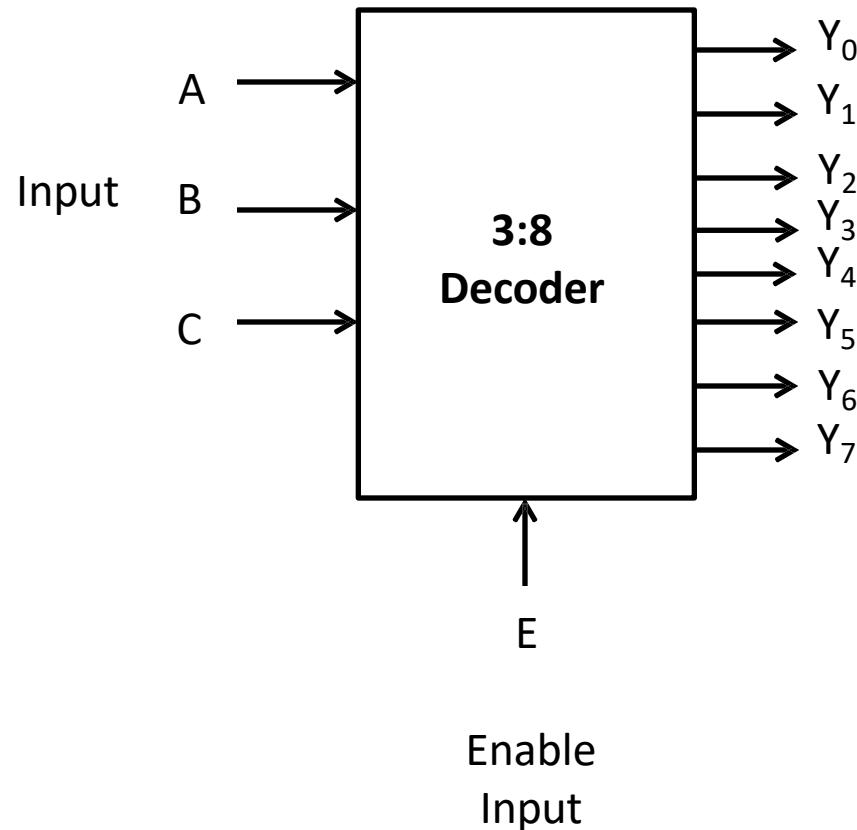
Enable i/p	Data Inputs		Outputs				
	E	A	B	Y_0	Y_1	Y_2	Y_3
0	X	X		0	0	0	0
1	0	0		1	0	0	0
1	0	1		0	1	0	0
1	1	0		0	0	1	0
1	1	1		0	0	0	1

2 to 4 Line Decoder



3 to 8 Line Decoder

Block Diagram



3 to 8 Line Decoder

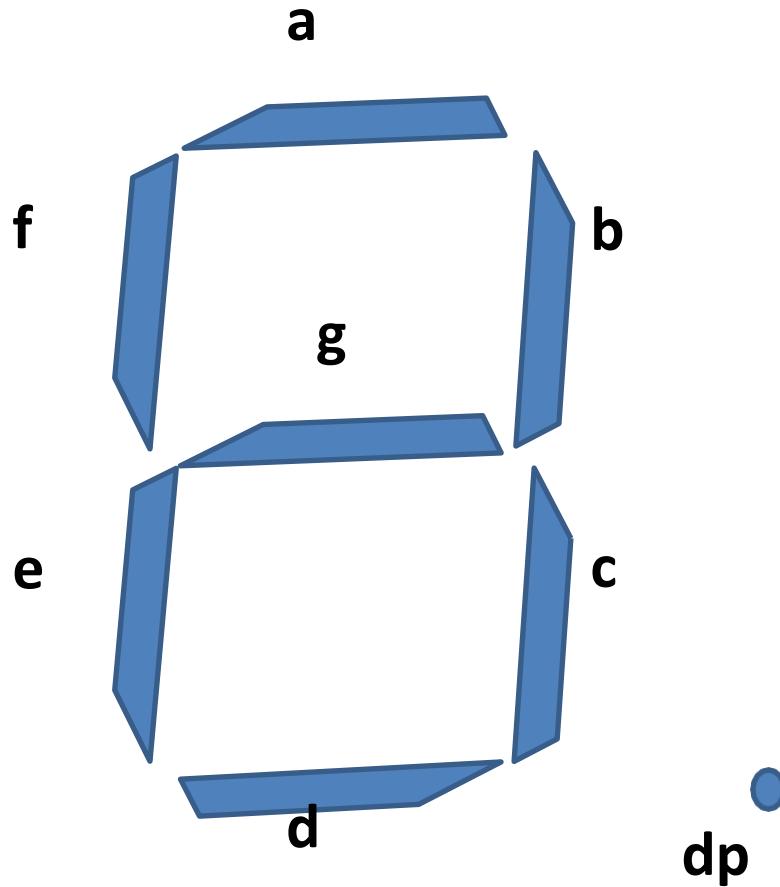
Truth Table

Enabl e i/p	Inputs				Outputs							
	E	A	B	C	Y ₇	Y ₆	Y ₅	Y ₄	Y ₃	Y ₂	Y ₁	Y ₀
0	X	X	X	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0
1	1	1	0	0	0	1	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0

Comparison between Encoder & Decoder

Sr. No.	Parameter	Encoder	Decoder
1	Input applied	Active input signal (original message signal)	Coded binary input
2	Output generated	Coded binary output	Active output signal (original message)
3	Input lines	2^n	n
4	Output lines	N	2^n
5	Operation	Simple	Complex
6	Applications	E-mail , video encoders etc.	Microprocessors, memory chips etc.

BCD to 7 Segment Decoder - Seven Segment Display



Seven Segment Display

Segments							Display Number	Seven Segment Display
a	b	c	d	e	f	g		
ON	ON	ON	ON	ON	ON	OFF	0	
OFF	ON	ON	OFF	OFF	OFF	OFF	1	
ON	ON	OFF	ON	ON	OFF	ON	2	
ON	ON	ON	ON	OFF	OFF	ON	3	
OFF	ON	ON	OFF	OFF	ON	ON	4	
ON	OFF	ON	ON	OFF	ON	ON	5	
ON	OFF	ON	ON	ON	ON	ON	6	
ON	ON	ON	OFF	OFF	OFF	OFF	7	
ON	ON	ON	ON	ON	ON	ON	8	
ON	ON	ON	ON	OFF	ON	ON	9	

Multiplexers

- ✓ Multiplexer is a circuit which has a number of inputs but only one output.
- ✓ Multiplexer is a circuit which transmits large number of information signals over a single line.
- ✓ Multiplexer is also known as “Data Selector” or MUX.

Necessity of Multiplexers

- ✓ In most of the electronic systems, the digital data is available on more than one lines. It is necessary to route this data over a single line.
- ✓ Under such circumstances we require a circuit which select one of the many inputs at a time.
- ✓ This circuit is nothing but a multiplexer. Which has many inputs, one output and some select lines.
- ✓ Multiplexer improves the reliability of the digital system because it reduces the number of external wired connections.

Advantages of Multiplexers

- ✓ It reduces the number of wires.
- ✓ So it reduces the circuit complexity and cost.
- ✓ We can implement many combinational circuits using Mux.
- ✓ It simplifies the logic design.
- ✓ It does not need the k-map and simplification.

Applications of Multiplexers

- ✓ It is used as a data selector to select one out of many data inputs.
- ✓ It is used for simplification of logic design.
- ✓ It is used in data acquisition system.
- ✓ In designing the combinational circuits.
- ✓ In D to A converters.
- ✓ To minimize the number of connections.

Block Diagram of Multiplexer

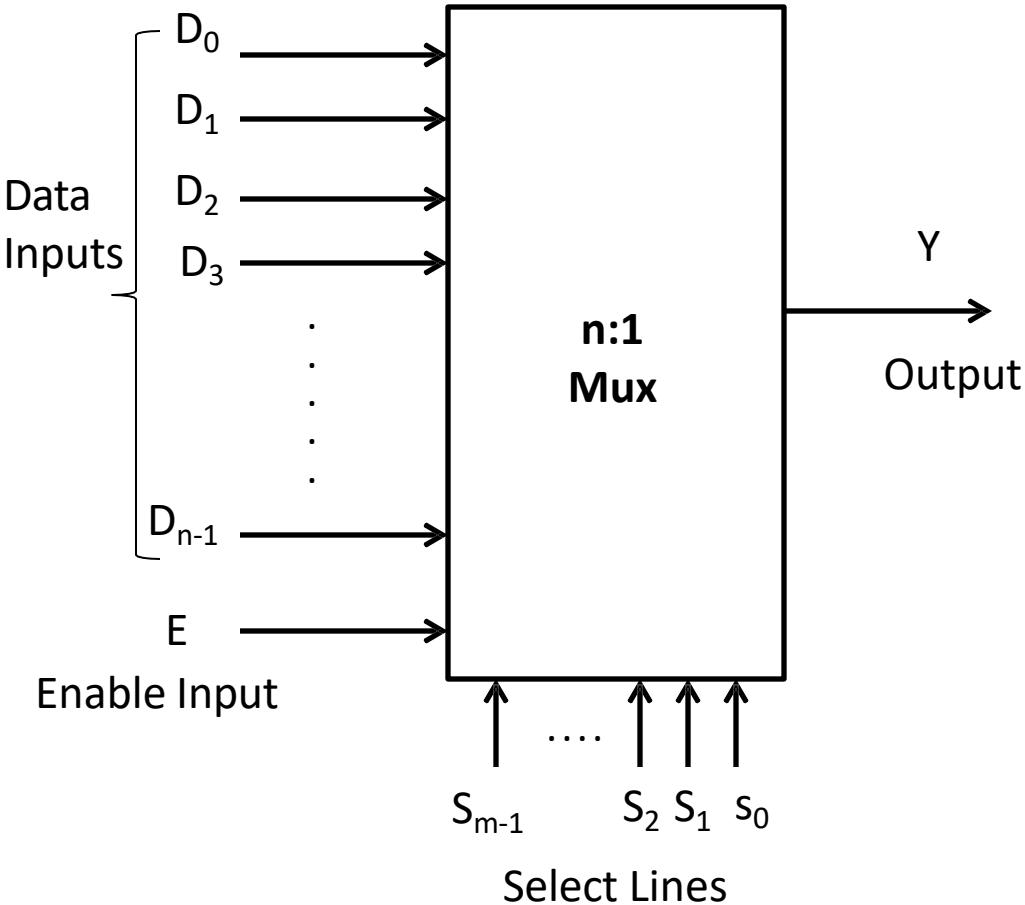


Fig. General Block Diagram

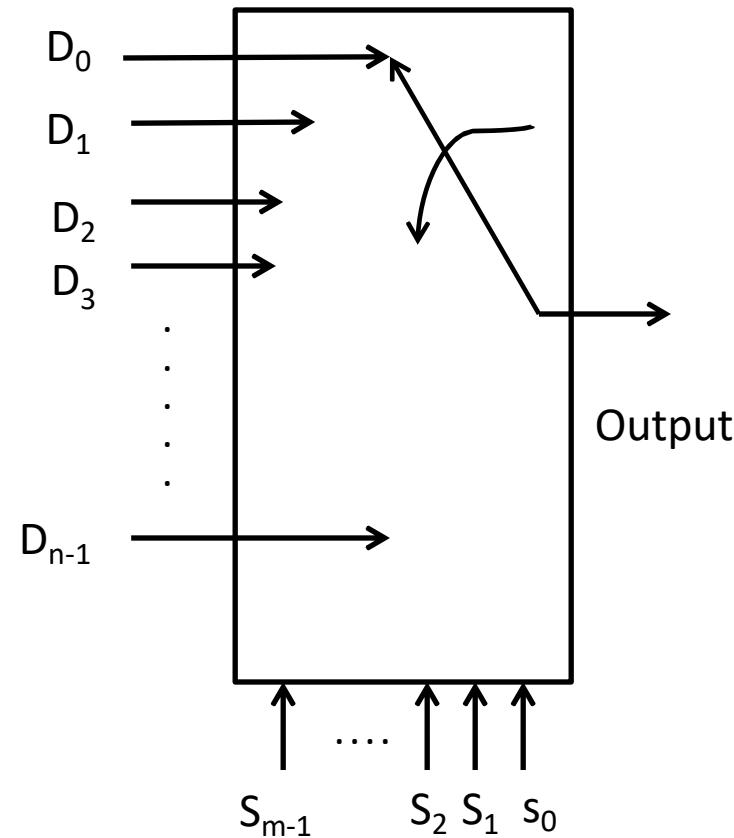


Fig. Equivalent Circuit

Relation between Data Input Lines & Select Lines

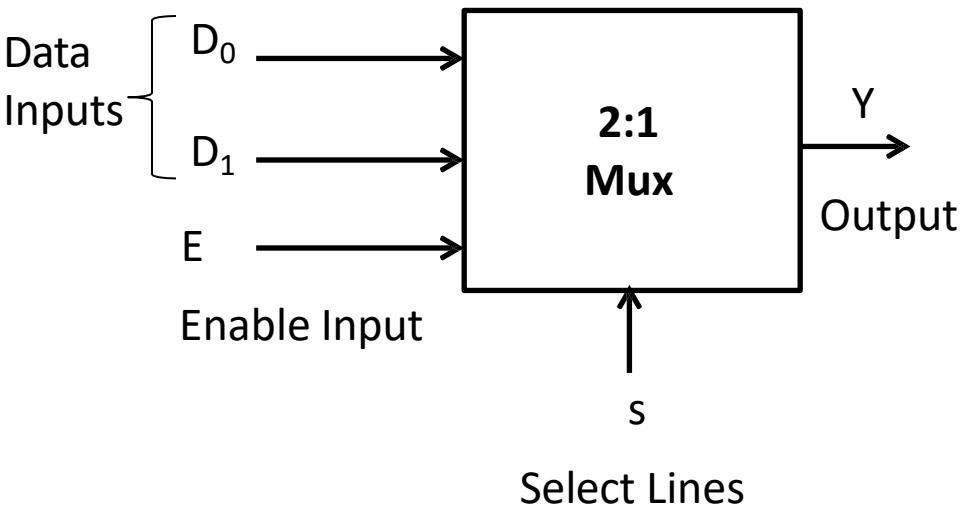
- ✓ In general multiplexer contains , n data lines, one output line and m select lines.
- ✓ To select n inputs we need m select lines such that $2^m=n$.

Types of Multiplexers

- ✓ 2:1 Multiplexer
- ✓ 4:1 Multiplexer
- ✓ 8:1 Multiplexer
- ✓ 16:1 Multiplexer
- ✓ 32:1 Multiplexer
- ✓ 64:1 Multiplexer

and so on.....

2:1 Multiplexer



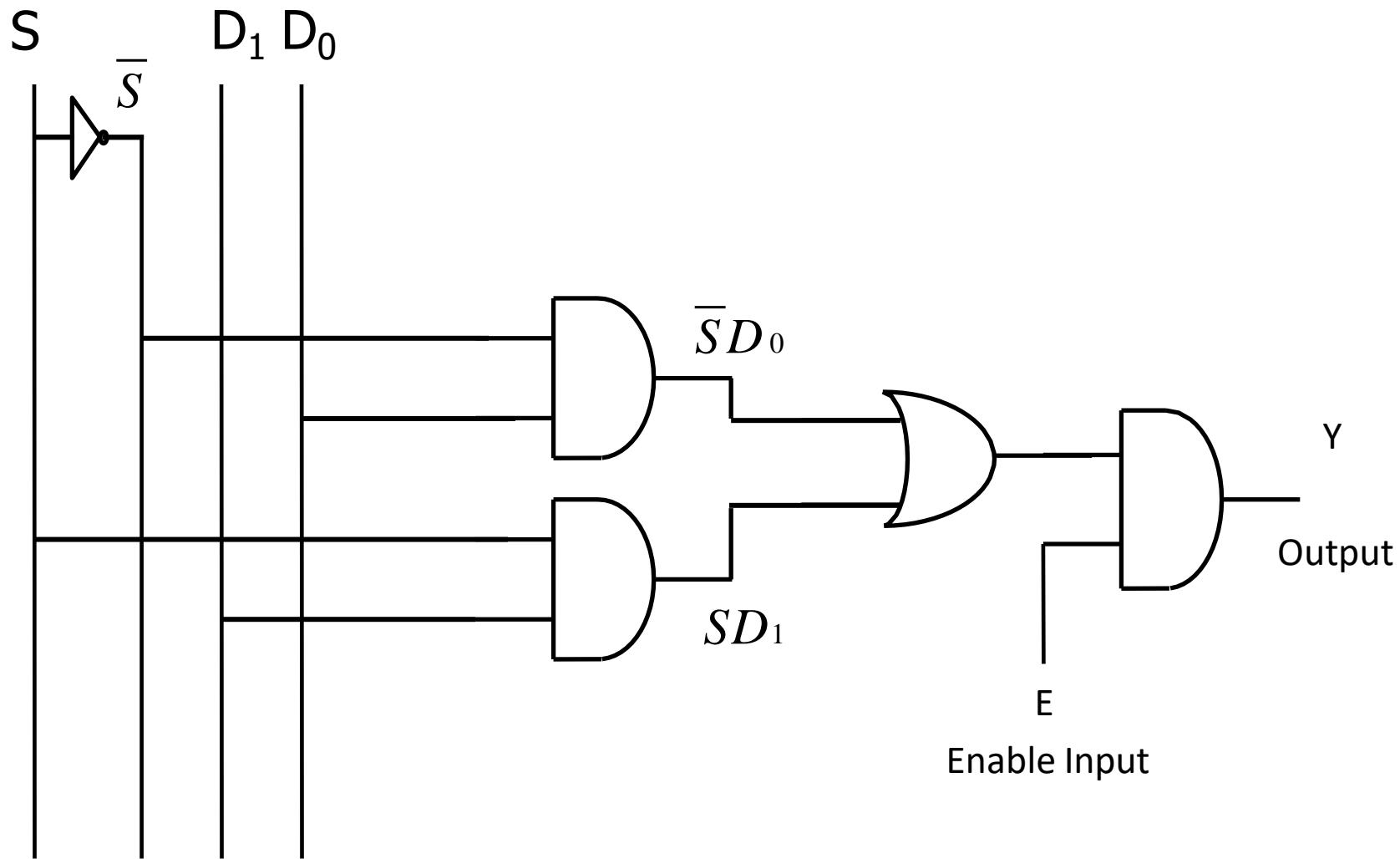
Block Diagram

Select Lines

Truth Table

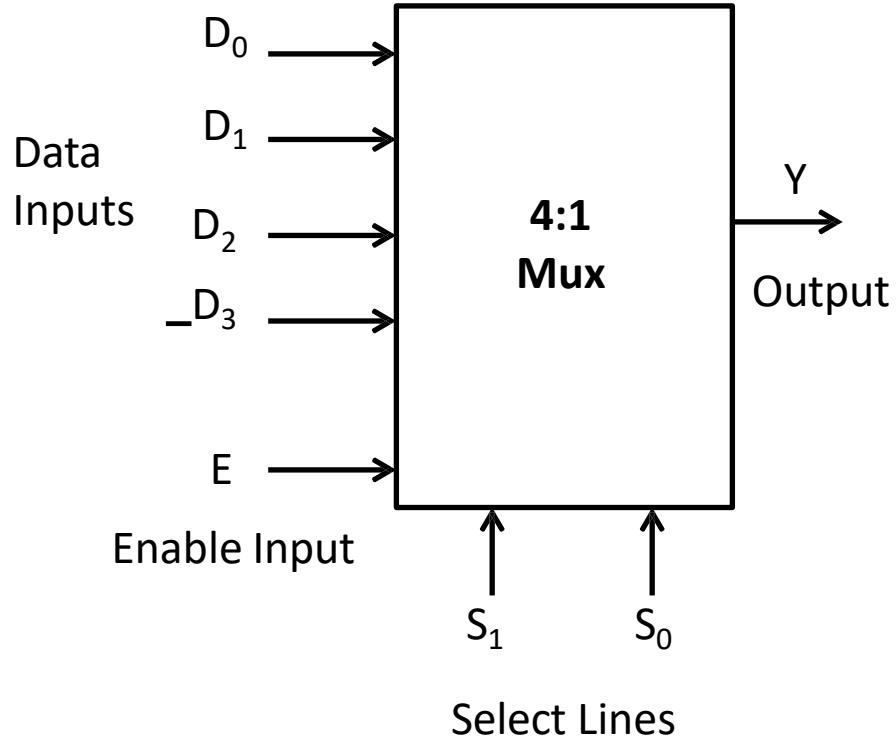
Enable i/p (E)	Select i/p (S)	Output (Y)
0	X	0
1	0	D_0
1	1	D_1

Realization of 2:1 Mux using gates



4:1 Multiplexer

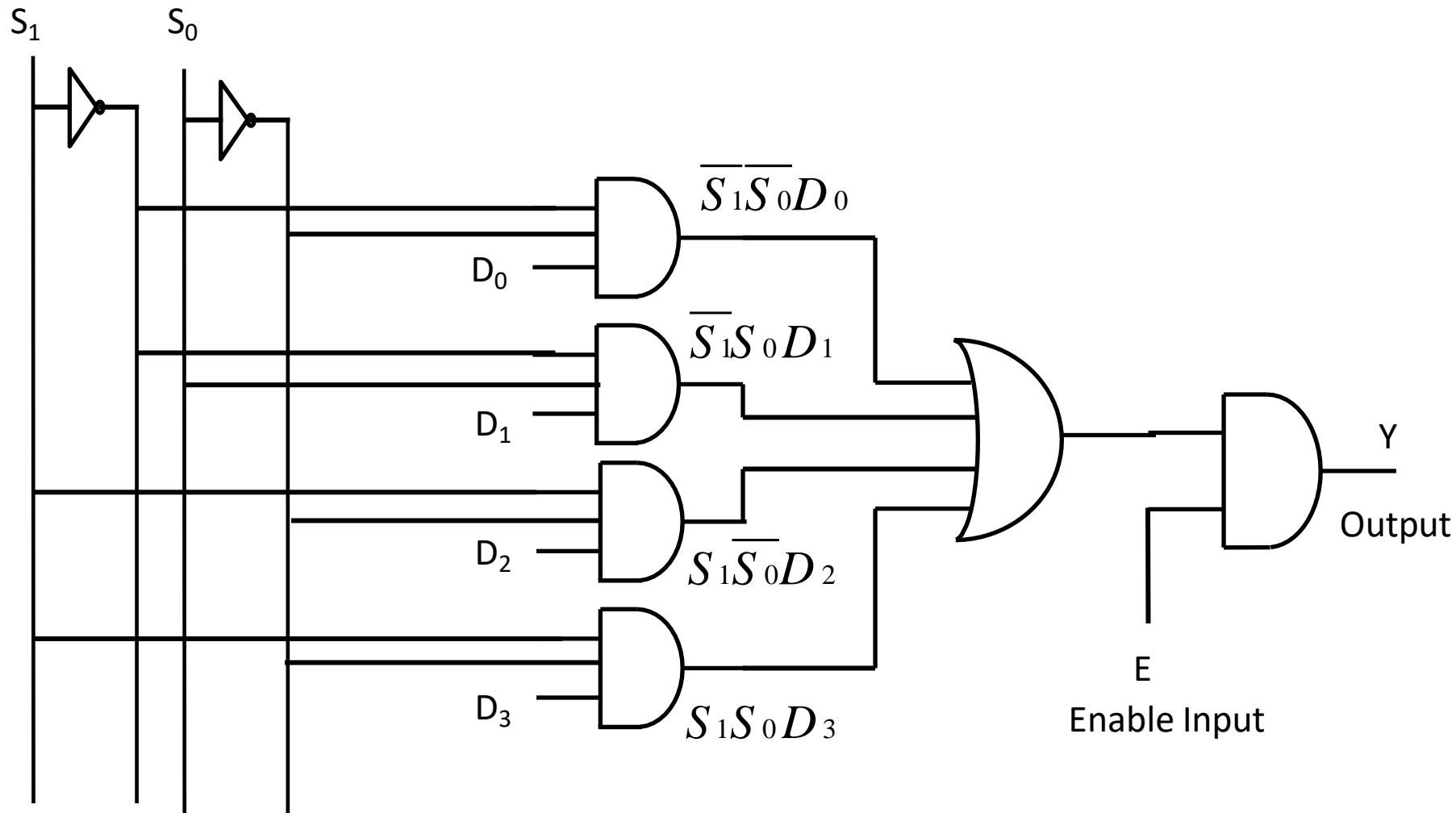
Block Diagram



Truth Table

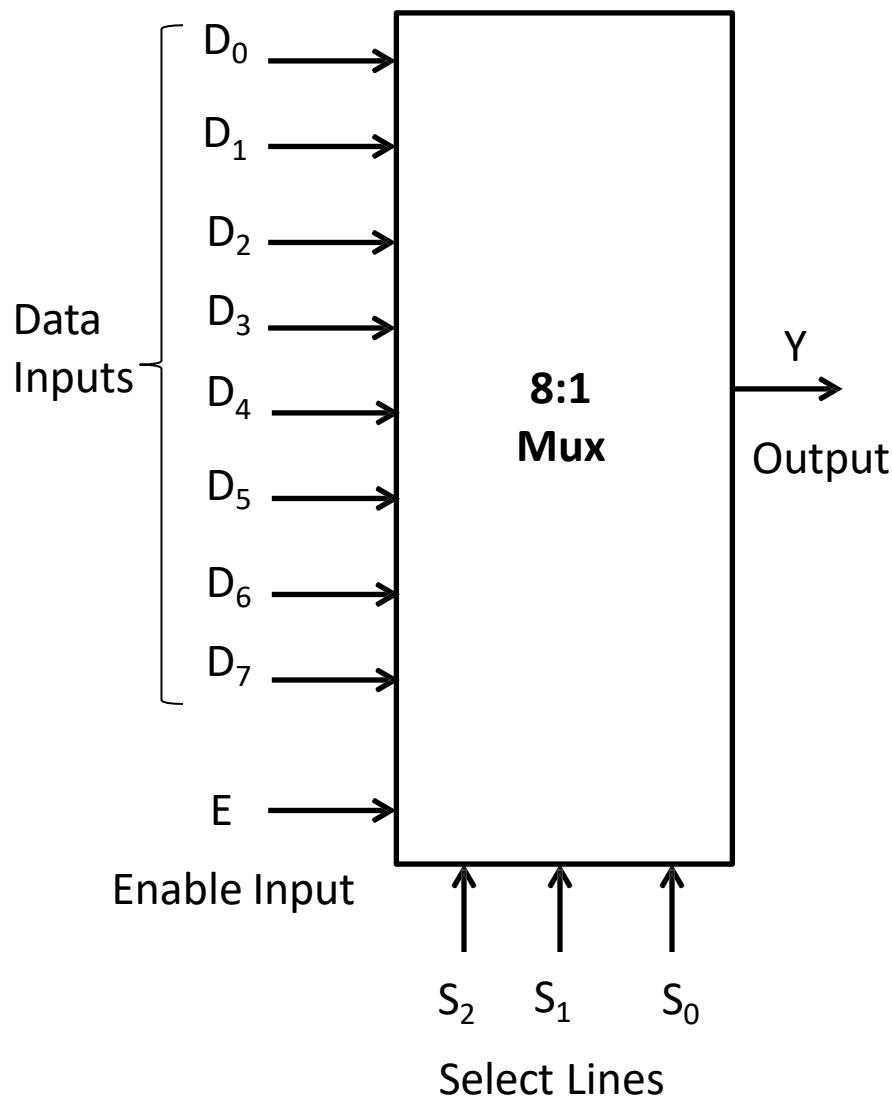
Enable i/p	Select i/p		Output
E	S_1	S_0	Y
0	X	X	0
1	0	0	D_0
1	0	1	D_1
1	1	0	D_2
1	1	1	D_3

Realization of 4:1 Mux using gates



8:1 Multiplexer

Block Diagram

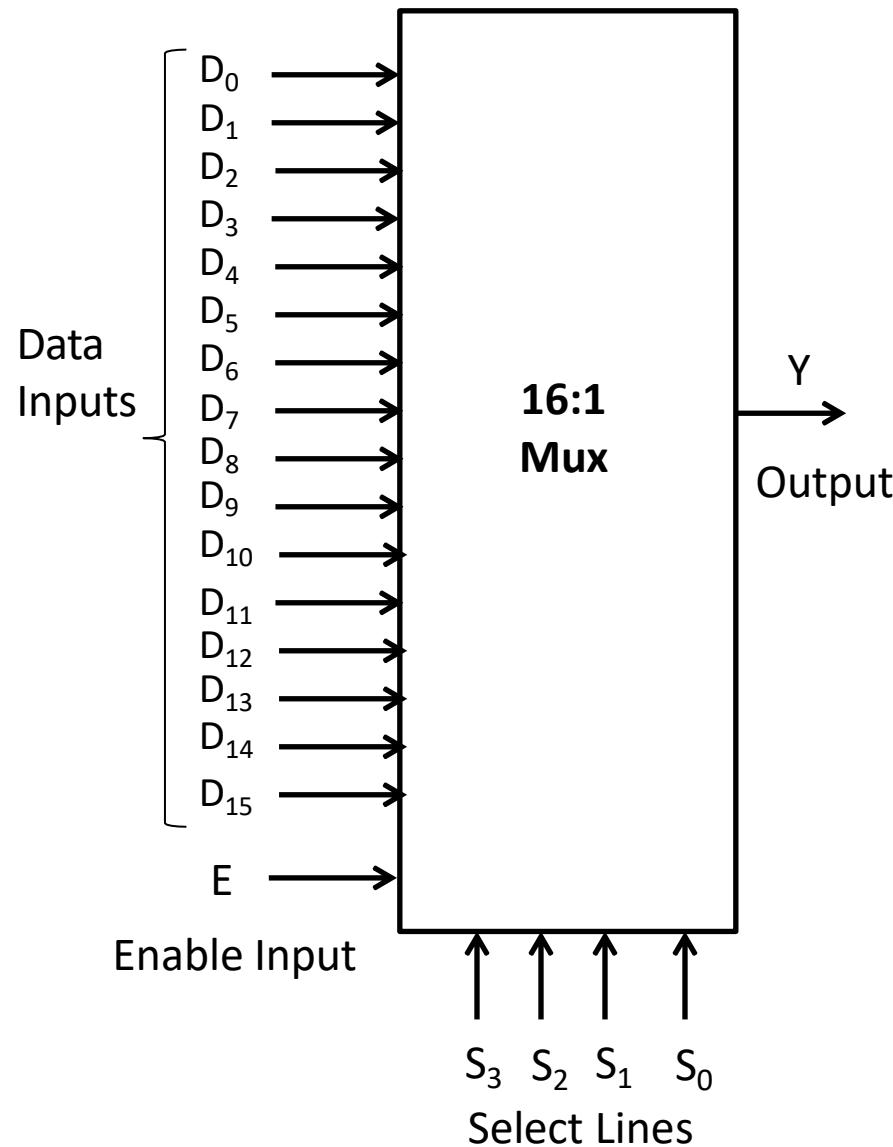


Truth Table

Enable i/p	Select i/p			Output
E	S_2	S_1	S_0	Y
0	X	X	X	0
1	0	0	0	D_0
1	0	0	1	D_1
1	0	1	0	D_2
1	0	1	1	D_3
1	1	0	0	D_4
1	1	0	1	D_5
1	1	1	0	D_6
1	1	1	1	D_7

16:1 Multiplexer

Block Diagram



16:1 Multiplexer

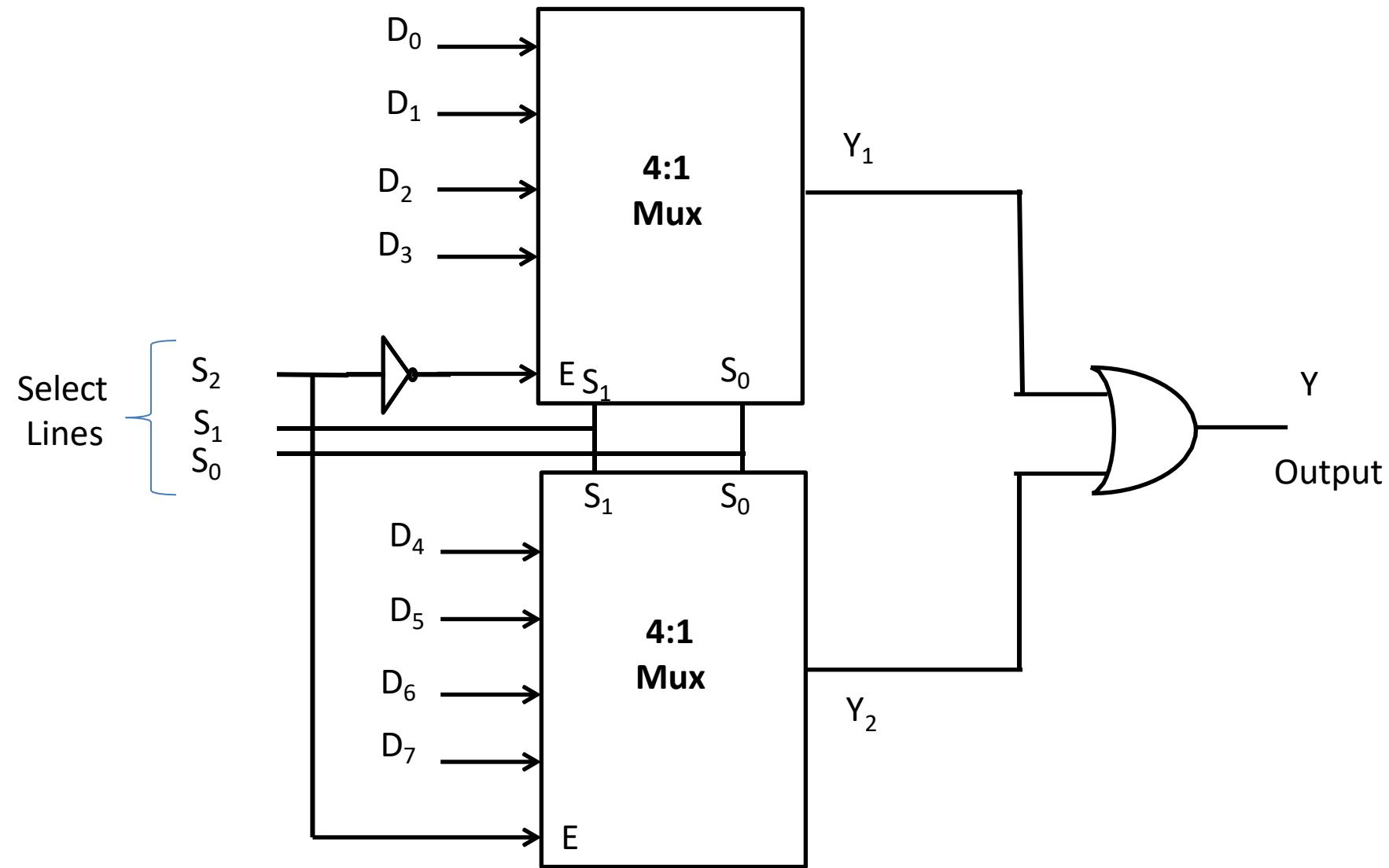
Truth Table

Enable	Select Lines				Output
	S_3	S_2	S_1	S_0	
E	X	X	X	X	Y
0	X	X	X	X	0
1	0	0	0	0	D_0
1	0	0	0	1	D_1
1	0	0	1	0	D_2
1	0	0	1	1	D_3
1	0	1	0	0	D_4
1	0	1	0	1	D_5
1	0	1	1	0	D_6
1	0	1	1	1	D_7
1	1	0	0	0	D_8
1	1	0	0	1	D_9
1	1	0	1	0	D_{10}
1	1	0	1	1	D_{11}
1	1	1	0	0	D_{12}
1	1	1	0	1	D_{13}
1	1	1	1	0	D_{14}
1	1	1	1	1	D_{15}

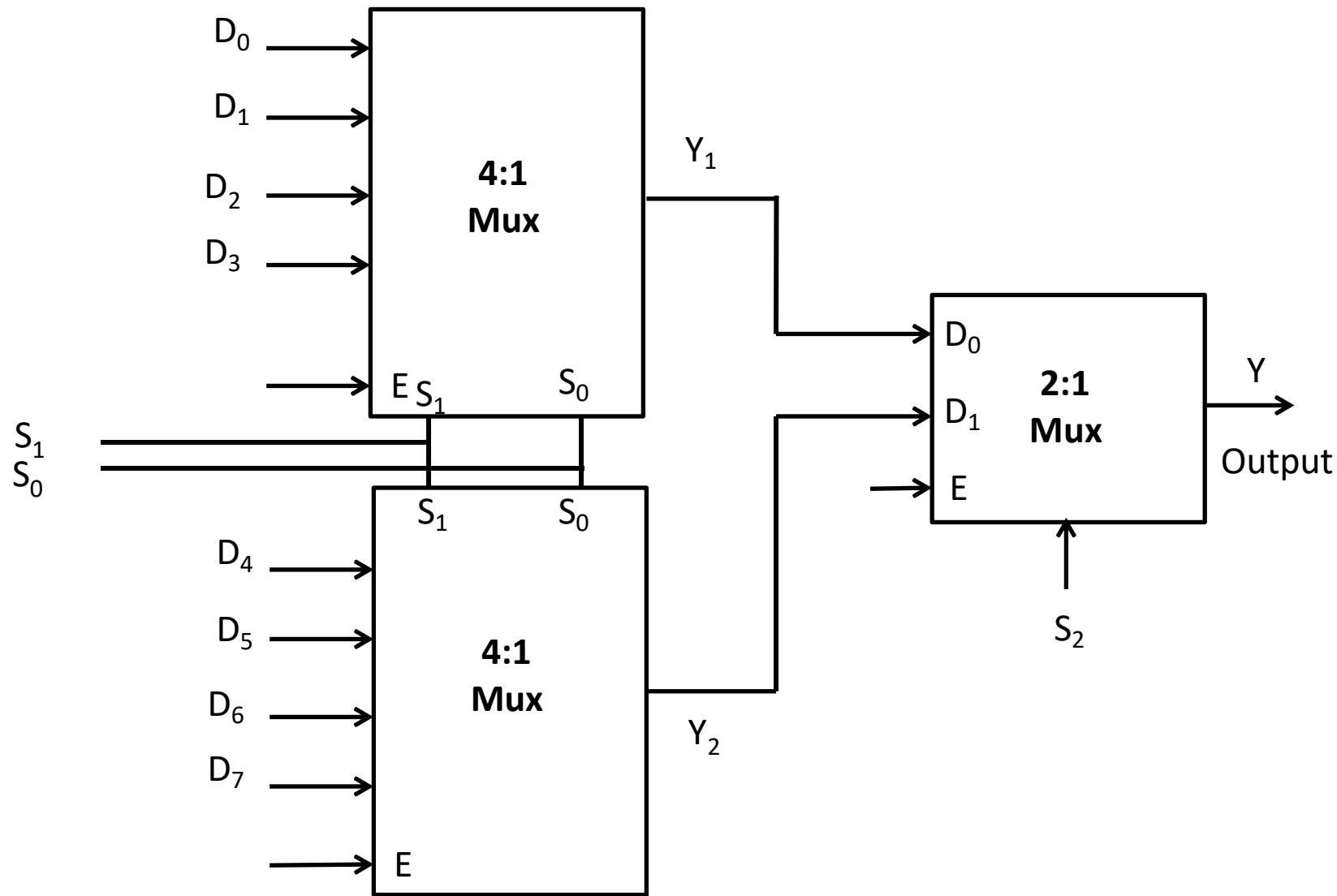
Mux Tree

- ✓ The multiplexers having more number of inputs can be obtained by cascading two or more multiplexers with less number of inputs. This is called as Multiplexer Tree.
- ✓ For example, 32:1 mux can be realized using two 16:1 mux and one 2:1 mux.

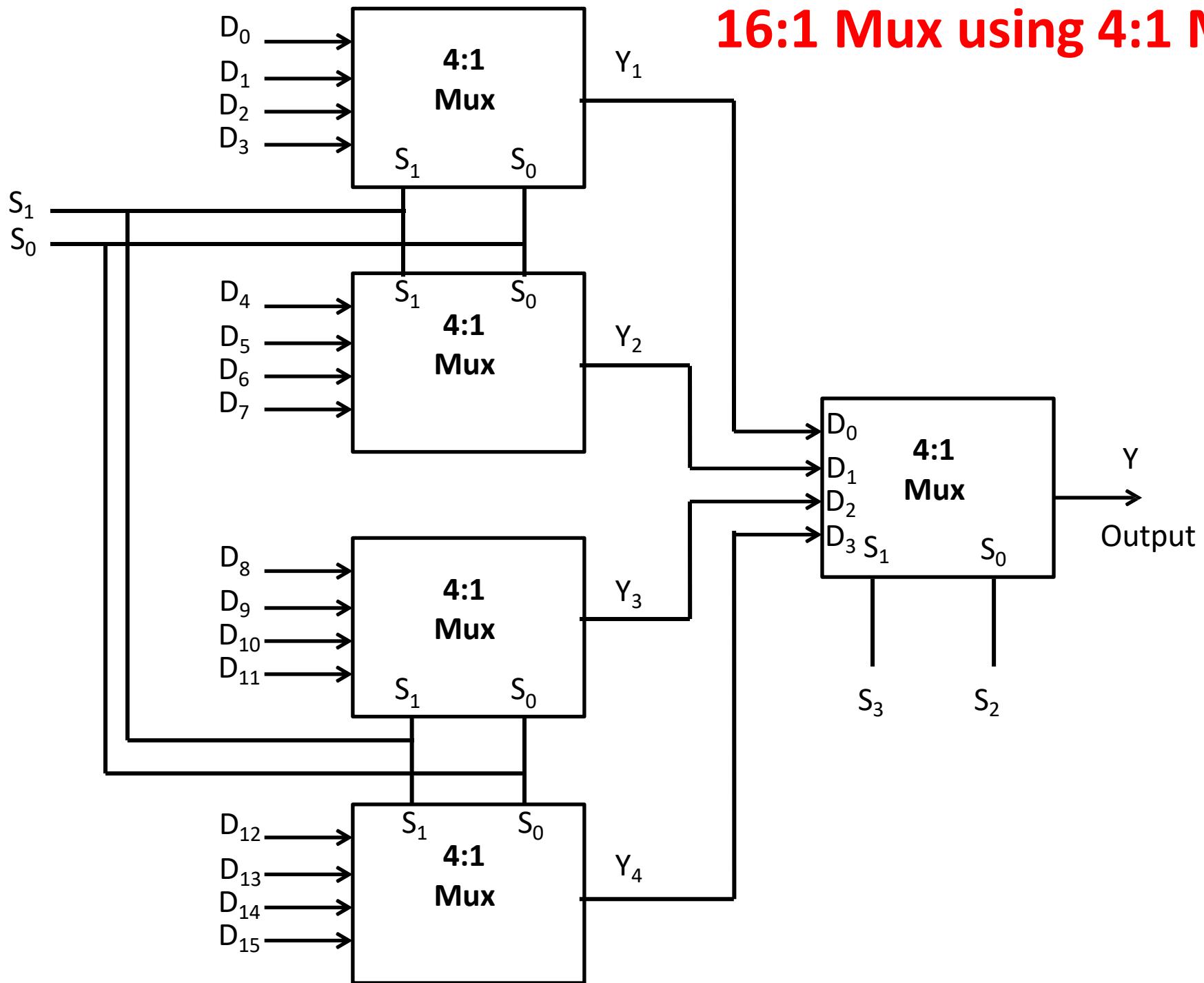
8:1 Multiplexer using 4:1 Multiplexer



8:1 Multiplexer using 4:1 Multiplexer



16:1 Mux using 4:1 Mux



Realization of Boolean expression using Mux

- ✓ We can implement any Boolean expression using Multiplexers.
- ✓ It reduces circuit complexity.
- ✓ It does not require any simplification

Example 1

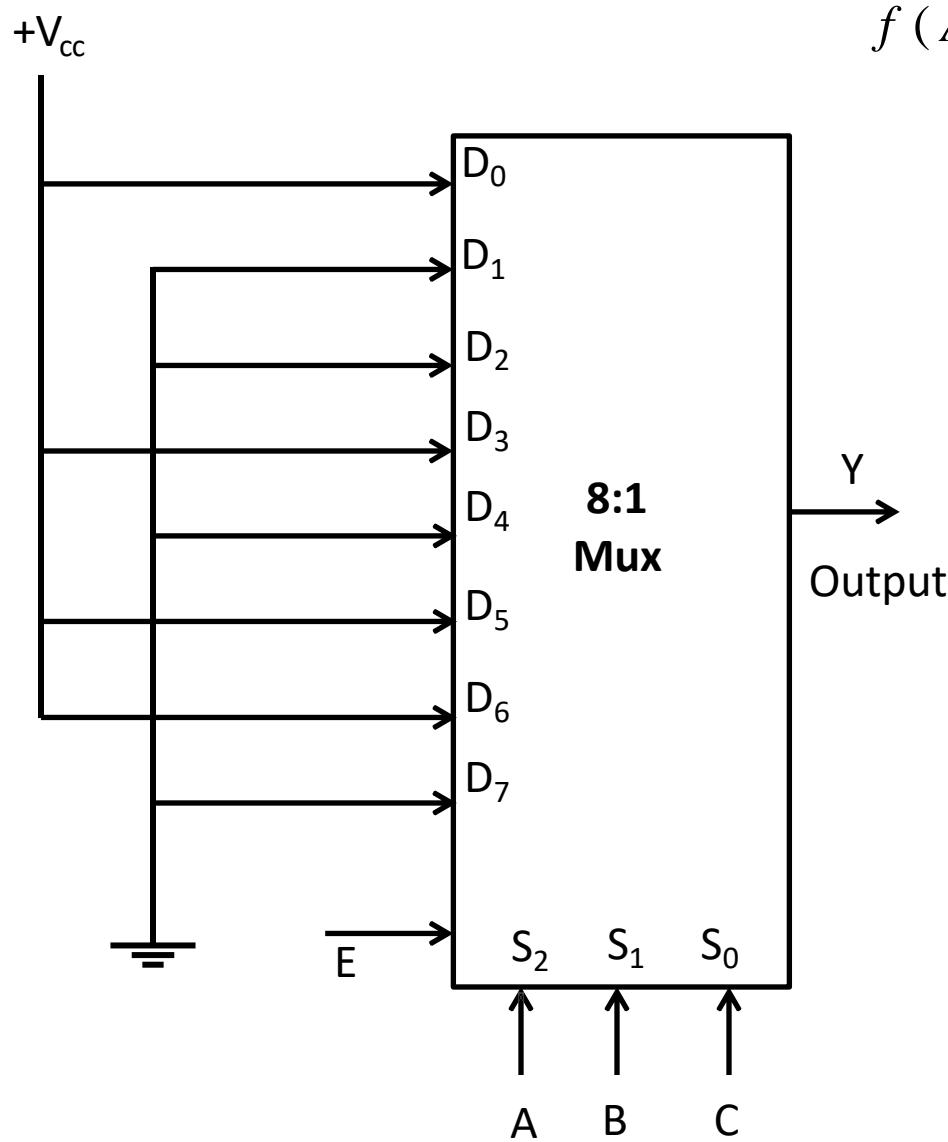
Implement following Boolean expression using multiplexer

$$f(A, B, C) = \sum m(0, 3, 5, 6)$$

- ✓ Since there are three variables, therefore a multiplexer with three select input is required i.e. 8:1 multiplexer is required
- ✓ The 8:1 multiplexer is configured as below to implement given Boolean expression

Example 1

continue.....



$$f(A, B, C) = \sum m(0, 3, 5, 6)$$

Example 2

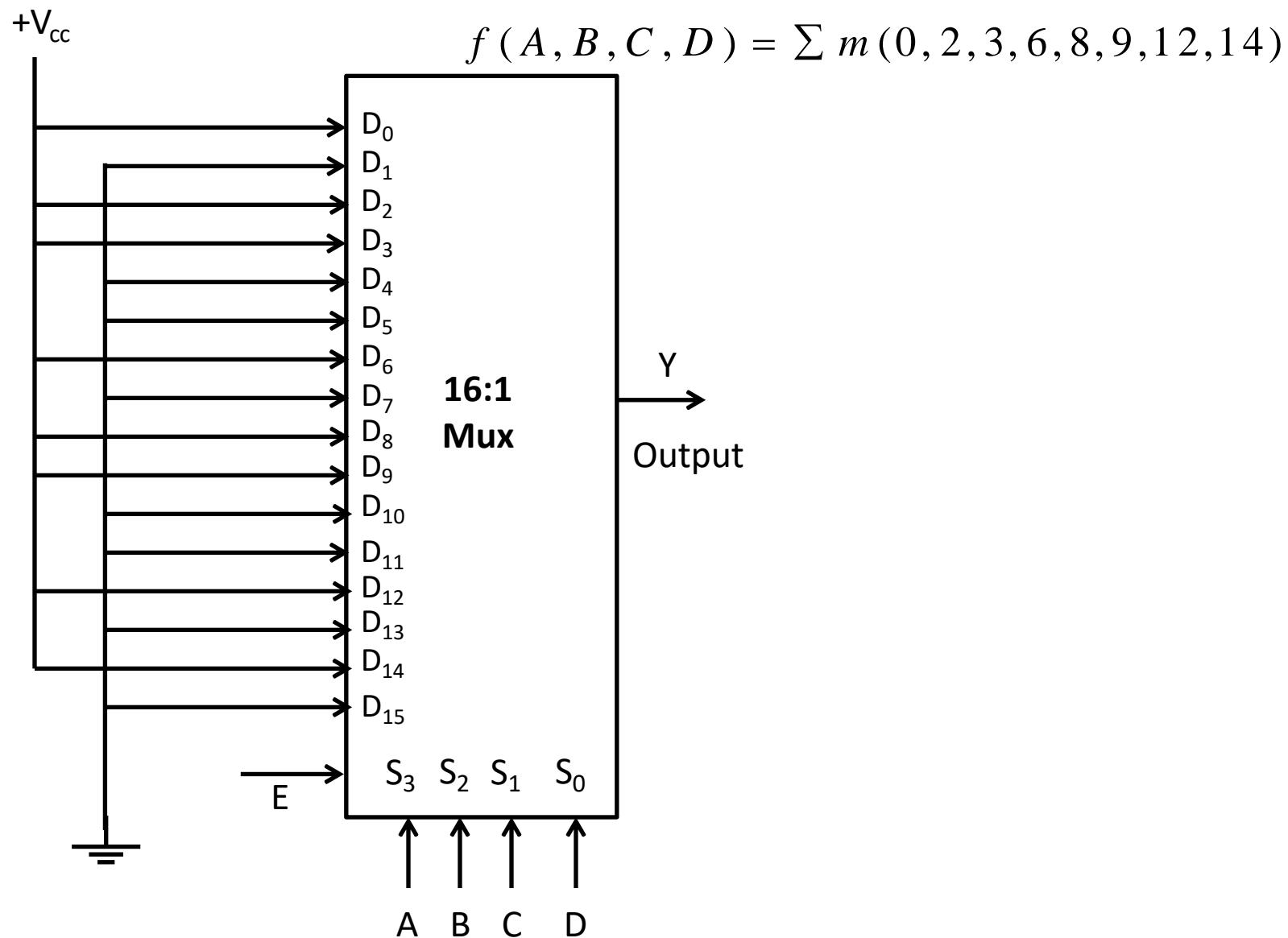
Implement following Boolean expression using multiplexer

$$f(A, B, C, D) = \sum m(0, 2, 3, 6, 8, 9, 12, 14)$$

- ✓ Since there are four variables, therefore a multiplexer with four select input is required i.e. 16:1 multiplexer is required
- ✓ The 16:1 multiplexer is configured as below to implement given Boolean expression

Example 2

continue.....



De-multiplexer

- ✓ A de-multiplexer performs the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs.
- ✓ At a time only one output line is selected by the select lines and the input is transmitted to the selected output line.
- ✓ It has only one input line, n number of output lines and m number of select lines.

Block Diagram of De-multiplexer

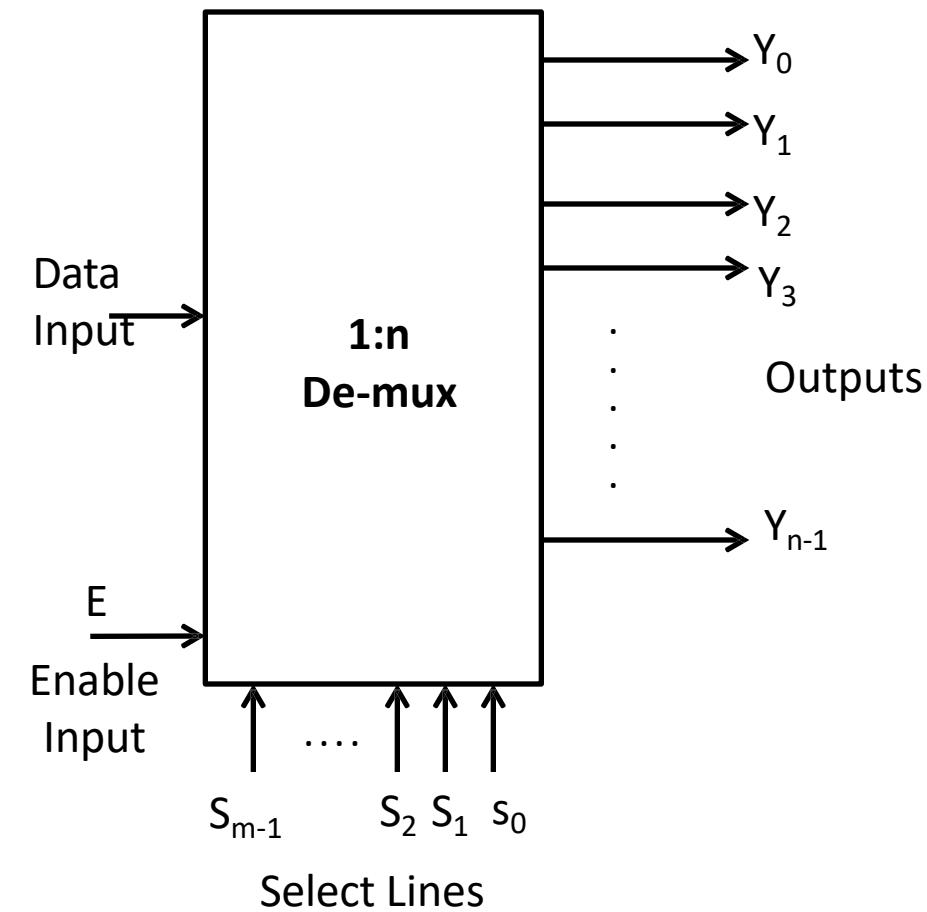


Fig. General Block Diagram

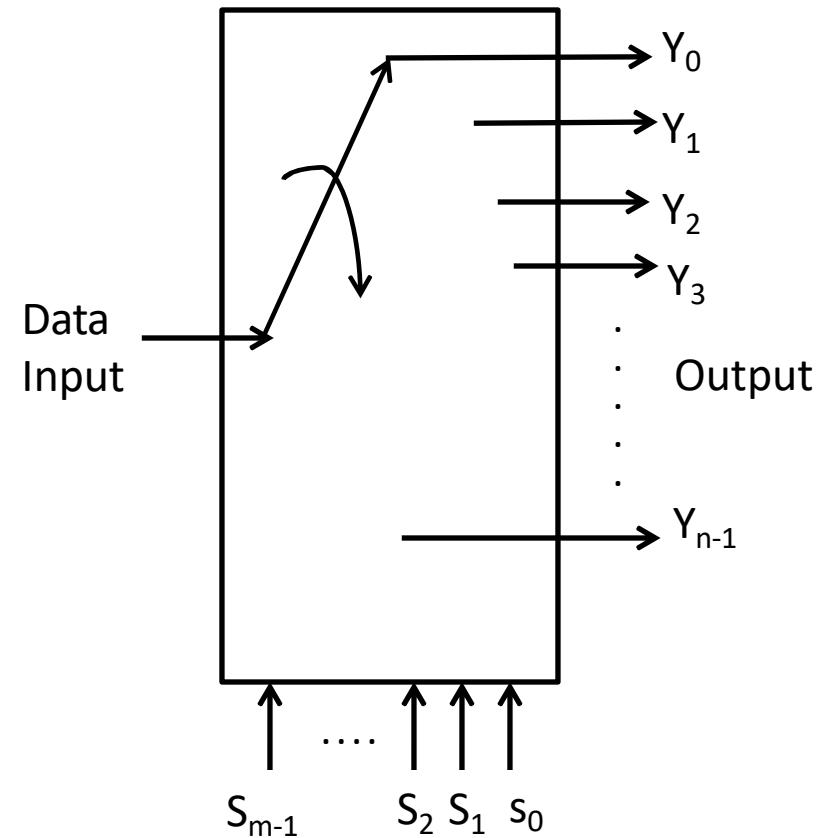


Fig. Equivalent Circuit

Relation between Data Output Lines & Select Lines

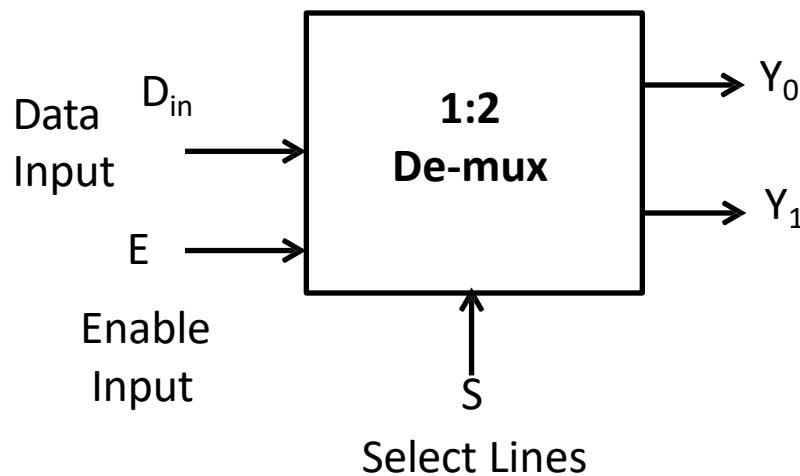
- ✓ In general de-multiplexer contains , n output lines, one input line and m select lines.
- ✓ To select n outputs we need m select lines such that $n=2^m$.

Types of De-multiplexers

- ✓ 1:2 De-multiplexer
- ✓ 1:4 De-multiplexer
- ✓ 1:8 De-multiplexer
- ✓ 1:16 De-multiplexer
- ✓ 1:32 De-multiplexer
- ✓ 1:64 De-multiplexer

and so on.....

1: 2 De-multiplexer



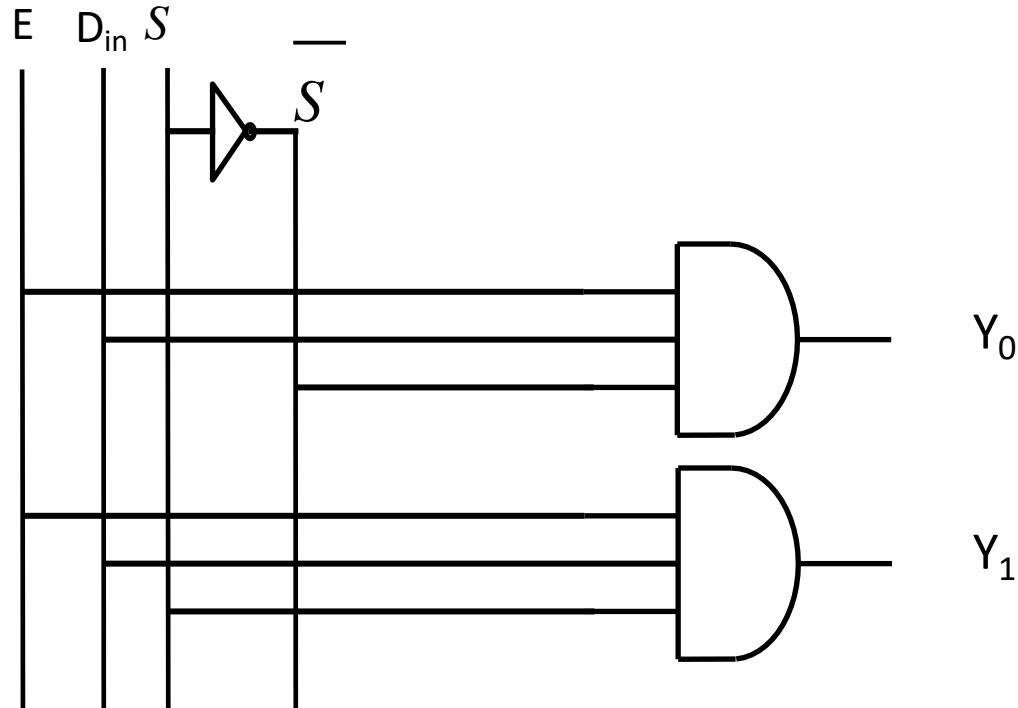
Block Diagram

Select Lines

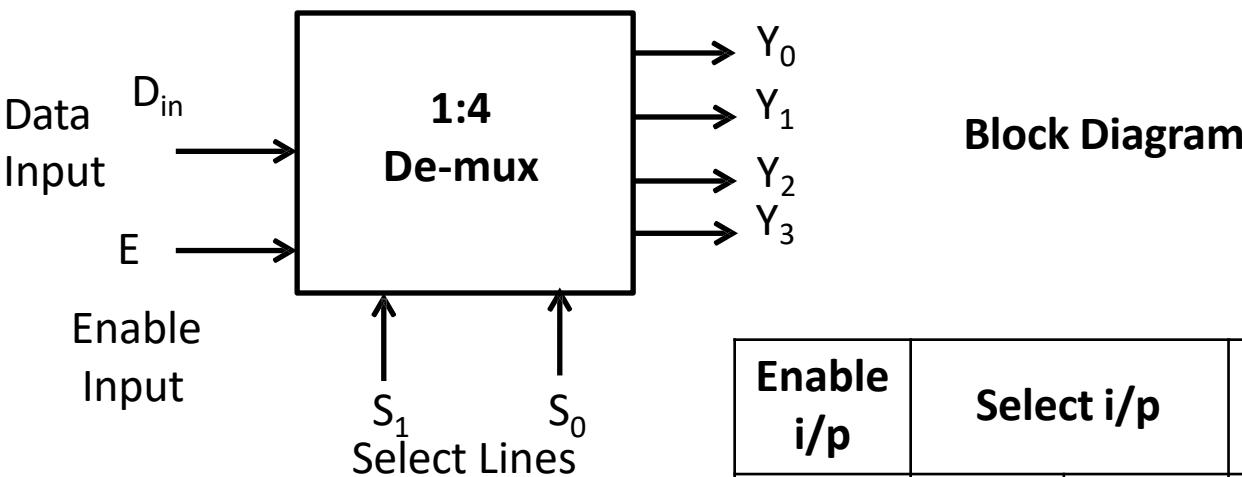
Truth Table

Enable i/p	Select i/p	Outputs	
E	S	Y_0	Y_1
0	X	0	0
1	0	D_{in}	0
1	1	0	D_{in}

1:2 De-mux using basic gates



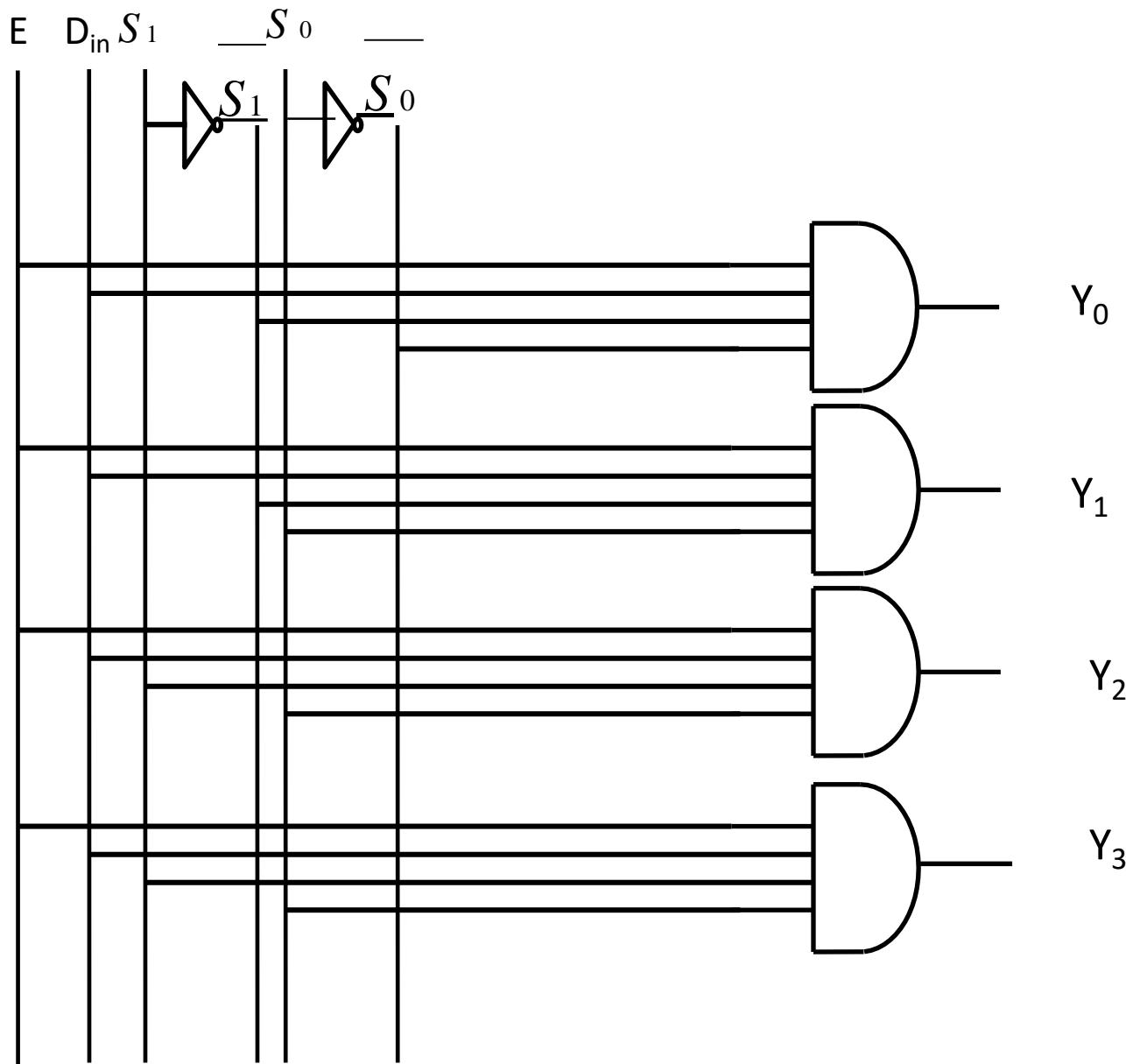
1: 4 De-multiplexer



Truth Table

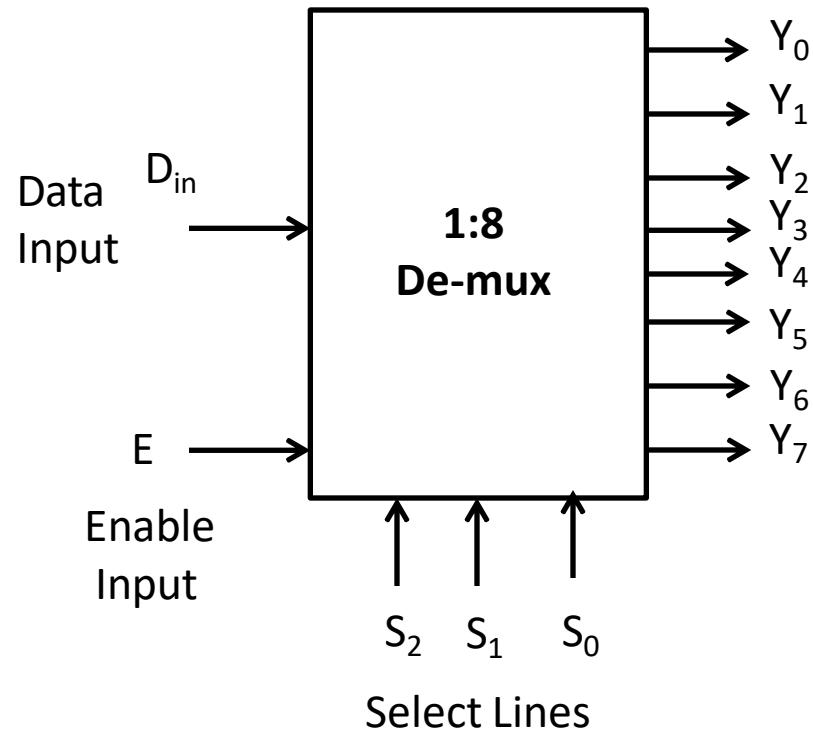
Enable i/p	Select i/p		Outputs			
	S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	X	X	0	0	0	0
1	0	0	D_{in}	0	0	0
1	0	1	0	D_{in}	0	0
1	1	0	0	0	D_{in}	0
1	1	1	0	0	0	D_{in}

1:4 De-mux using basic gates



1: 8 De-multiplexer

Block Diagram

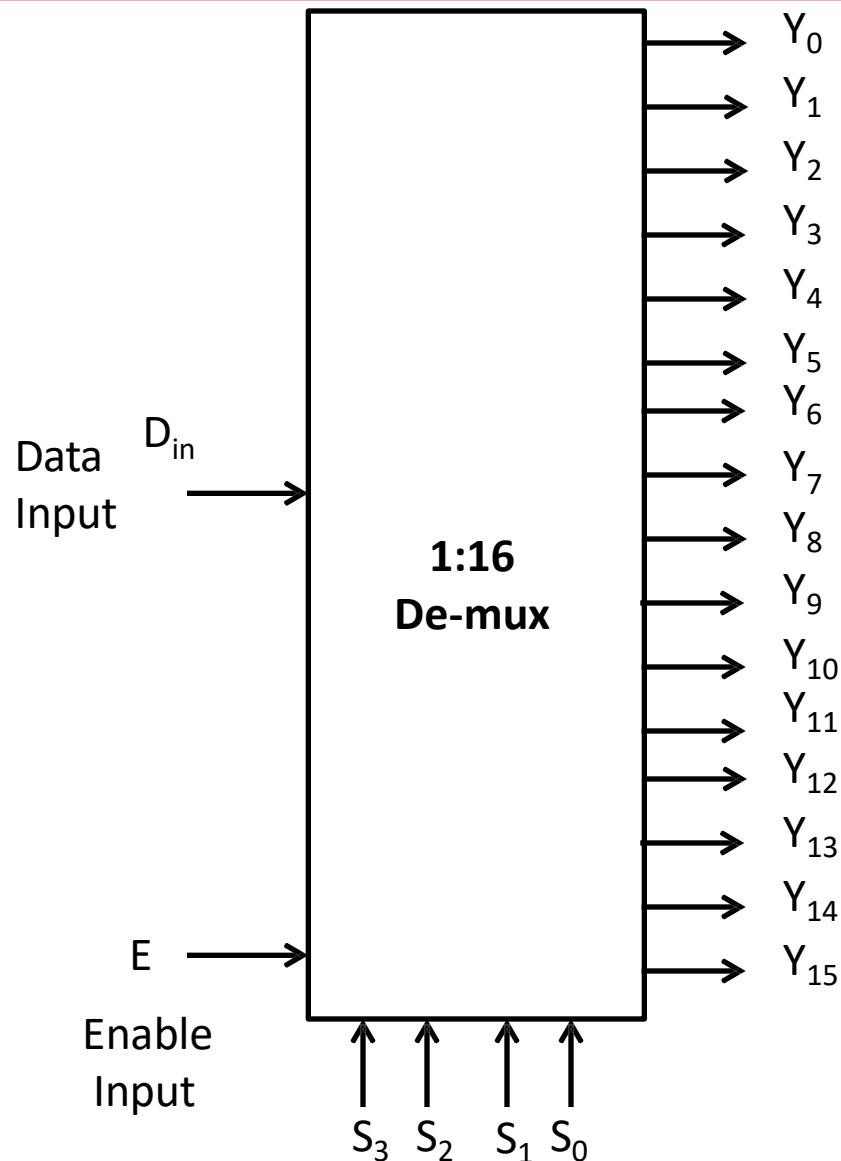


1: 8 De-multiplexer

Truth Table

1: 16 De-multiplexer

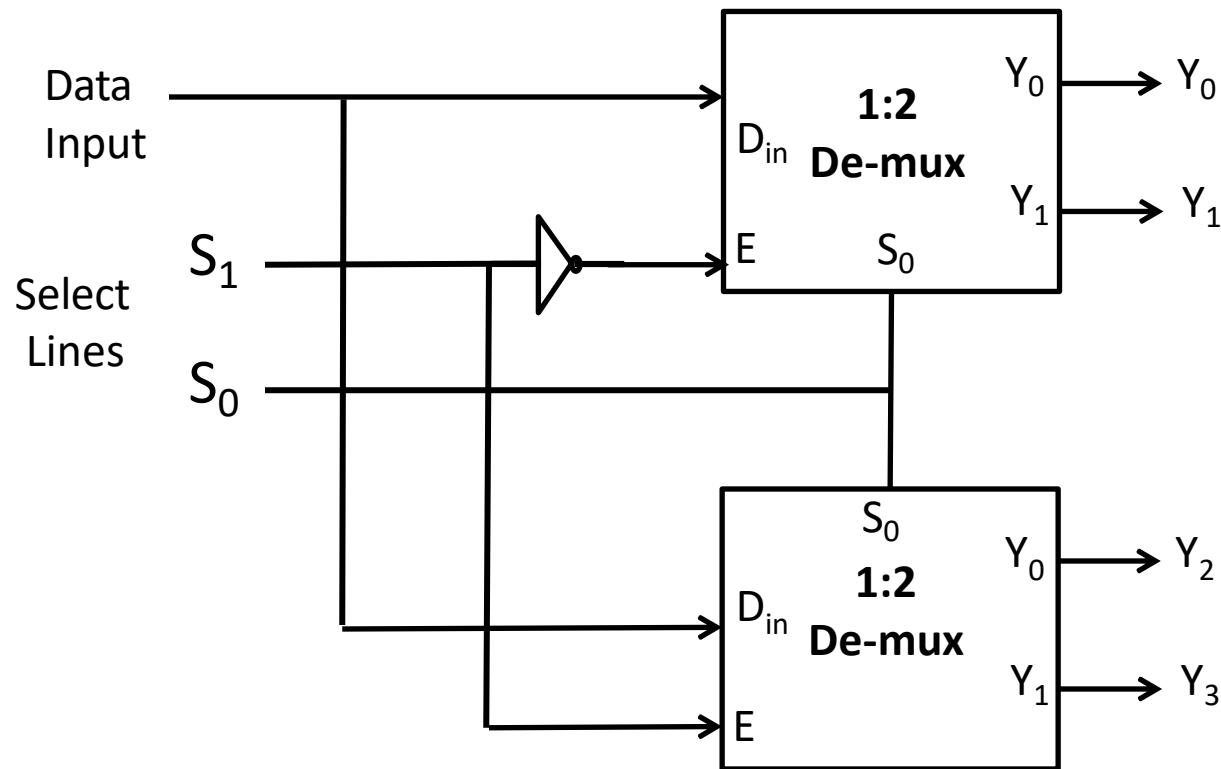
Block Diagram



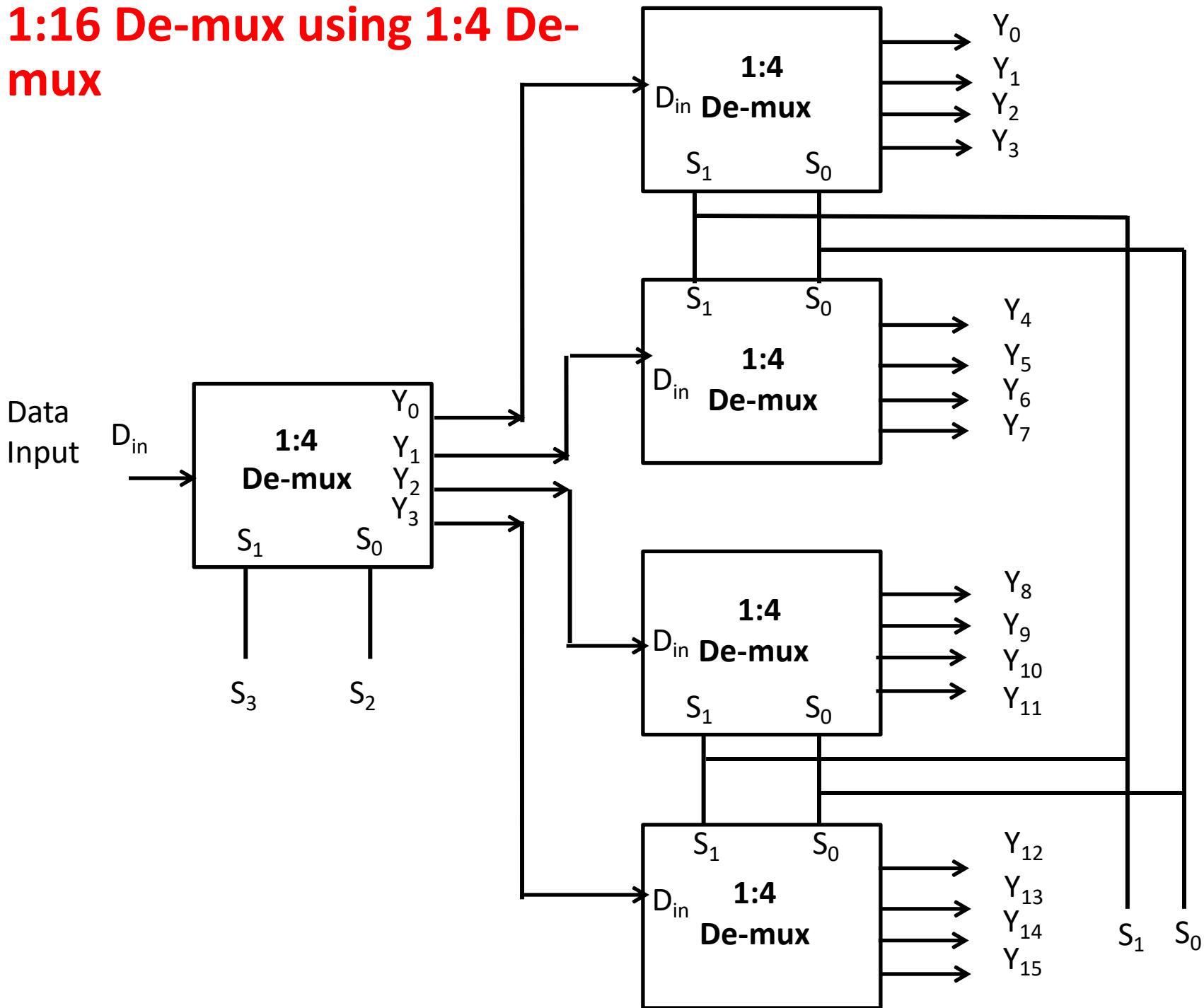
De-mux Tree

✓ Similar to multiplexer we can construct the de-multiplexer with more number of lines using de-multiplexer having less number of lines. This is called as “De-mux Tree”.

1:4 De-mux using 1:2 De-mux



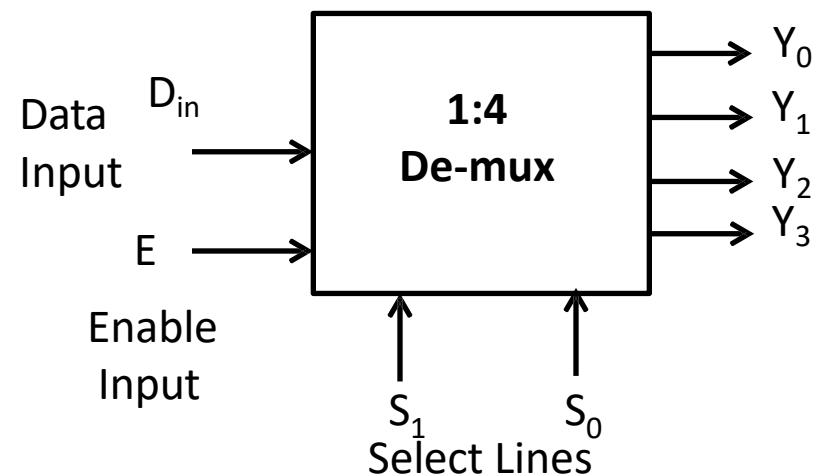
1:16 De-mux using 1:4 De-mux



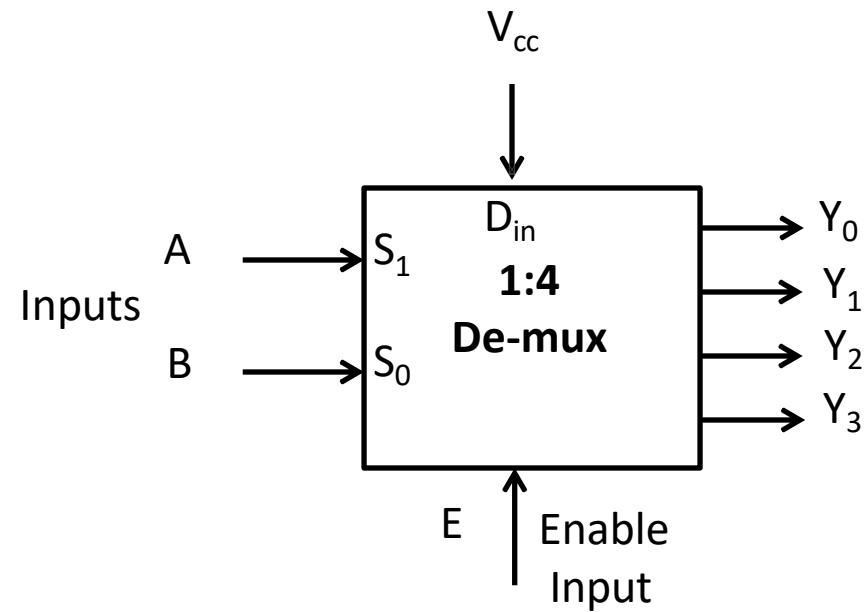
De-multiplexer as Decoder

- ✓ It is possible to operate a de-multiplexer as a decoder.
- ✓ Let us consider an example of 1:4 de-mux can be used as 2:4 decoder

1:4 De-multiplexer as 2:4 Decoder



1: 4 De-multiplexer



1: 4 De-multiplexer as 2:4 Decoder

Realization of Boolean expression using De-mux

- ✓ We can implement any Boolean expression using de-multiplexers.
- ✓ It reduces circuit complexity.
- ✓ It does not require any simplification

Example 1

Implement following Boolean expression using de-multiplexer

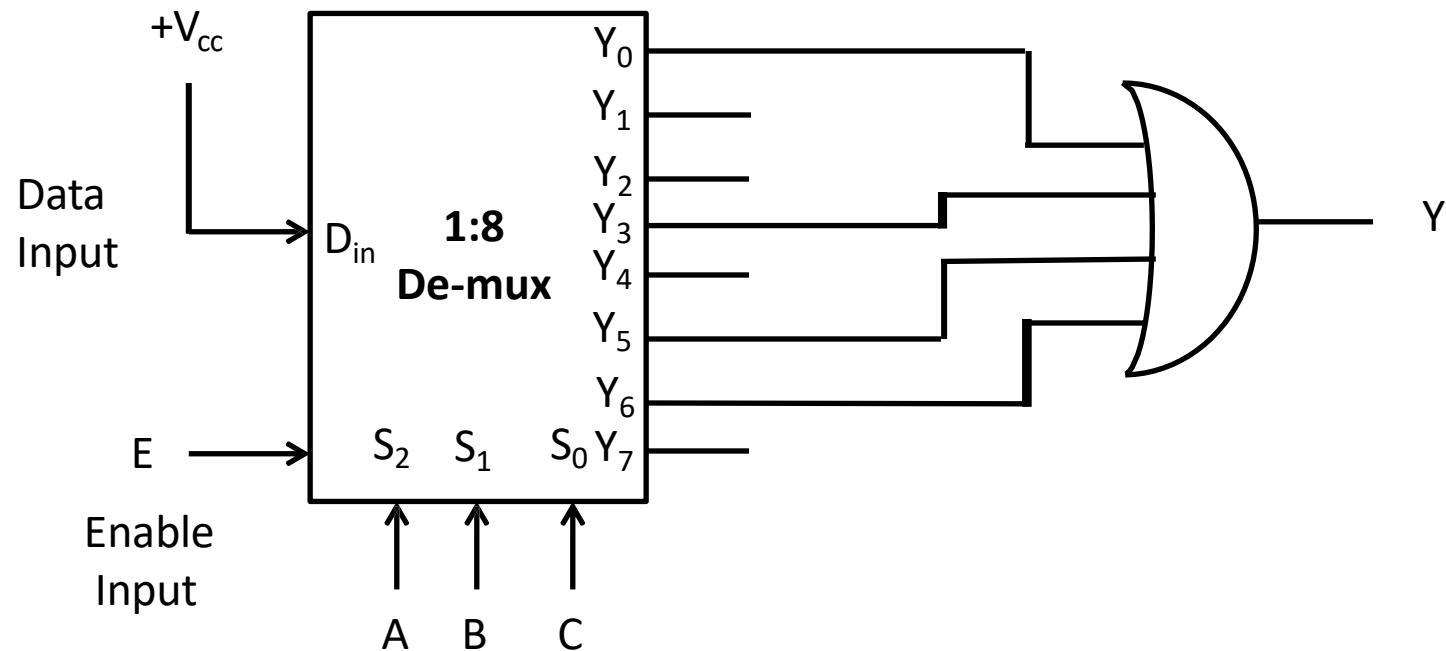
$$f(A, B, C) = \sum m(0, 3, 5, 6)$$

- ✓ Since there are three variables, therefore a de-multiplexer with three select input is required i.e. 1:8 de-multiplexer is required
- ✓ The 1:8 de-multiplexer is configured as below to implement given Boolean expression

Example 1

continue.....

$$f(A, B, C) = \sum m(0, 3, 5, 6)$$



Example 2

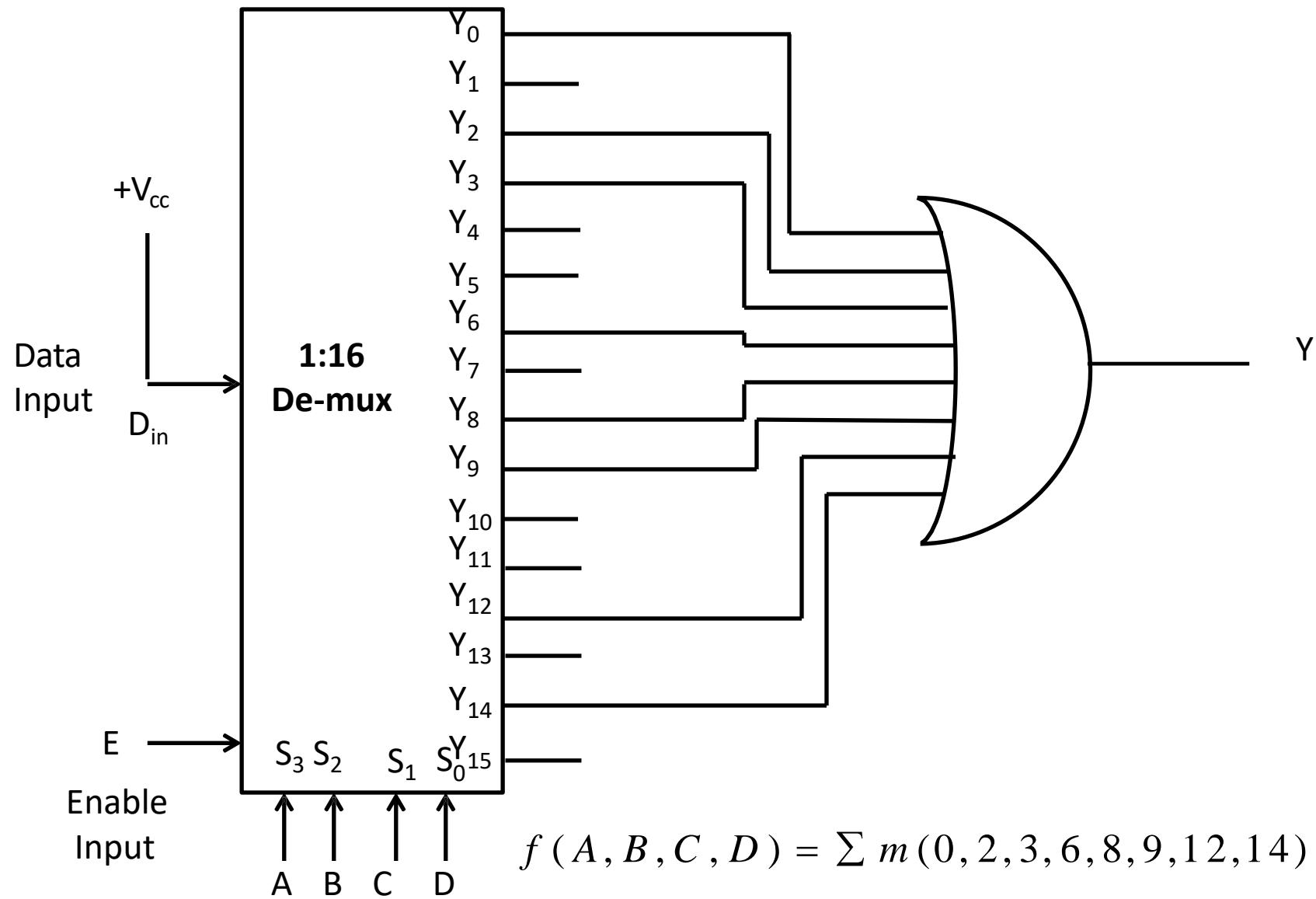
Implement following Boolean expression using de-multiplexer

$$f(A, B, C, D) = \sum m(0, 2, 3, 6, 8, 9, 12, 14)$$

- ✓ Since there are four variables, therefore a de-multiplexer with four select input is required i.e. 1:16 de-multiplexer is required
- ✓ The 1:16 de-multiplexer is configured as below to implement given Boolean expression

Example 2

continue.....



Multiplexer ICs

IC Number	Description	Output
IC 74157	Quad 2:1 Mux	Same as input
IC 74158	Quad 2:1 Mux	Inverted Output
IC 74153	Dual 4:1 Mux	Same as input
IC 74352	Dual 4:1 Mux	Inverted Output
IC 74151	8:1 Mux	Inverted Output
IC 74152	8:1 Mux	Inverted Output
IC 74150	16:1 Mux	Inverted Output

De-multiplexer ICs

IC Number	Description
IC 74138	1:8 De-multiplexer
IC 74139	Dual 1:4 De-multiplexer
IC 74154	1:16 De-multiplexer
IC 74155	Dual 1:4 De-multiplexer



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



Switching Theory and Logic Design

ARI-209

By: Dr. Divya Agarwal



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT- II** 10 HOURS
- **Sequential Logic Circuits:** Latches and Flip Flops- SR, D, T and MS-JK Flip Flops, Asynchronous Inputs.
- **Counters and Shift Registers:** Design of Synchronous and Asynchronous Counters- Binary, BCD, Decade and Up/Down Counters, Shift Registers, Types of Shift Registers, Counters using Shift Registers- Ring Counter and Johnson Counter.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAAPRASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-III** **08 HOURS**
- **Integrated circuits:** TTL and CMOS logic families and their characteristics. Brief introduction to RAM and ROM
- **Synchronous Sequential Circuits:** State Tables State Equations and State Diagrams, State Reduction and State Assignment, Design of Clocked Sequential Circuits using State Equations.



University School of Automation and Robotics
GURU GOBIND SINGH INDRAVASTHA UNIVERSITY
East Delhi Campus, Surajmal Vihar
Delhi - 110092



- **UNIT-IV** 06 HOURS
- **Finite state machine:** capabilities and limitations, Mealy and Moore models-minimization of completely specified and incompletely specified sequential machines, Partition techniques and merger chart methods, concept of minimal cover table.
- **Algorithmic State Machine:** Representation of sequential circuits using ASM charts synthesis of output and next state functions, Data path control path partition-based design.



Text Books:

1. Leach and Malvino (2011). *Digital principles and Applications*. Tata McGraw-Hill Education
2. Mano, M. M. (2017). *Digital logic and computer design*. Pearson Education India.
3. Jain, R. P. (2003). *Modern digital electronics*. Tata McGraw-Hill Education.

Reference Books:

1. A Anand Kumar. (2016) *Fundamentals of Digital Logic Circuits*, PHI
2. Taub, H., & Schilling, D. L. (1977). *Digital integrated electronics*. McGraw-Hill College.



■ UNIT- I

14 HOURS

- **Number Systems and Codes:** Decimal, Binary, Octal and Hexadecimal Number systems, Codes- BCD, Gray Code, Excess-3 Code, ASCII, EBCDIC, Conversion between various Codes.
- **Switching Theory:** Boolean Algebra- Postulates and Theorems, De' Morgan's Theorem, Switching Functions Canonical Forms- Simplification of Switching Functions- Karnaugh Map and Quine Mc-Clusky Methods.
- **Combinational Logic Circuits:** Review of basic gates- Universal gates, Adder, Subtractor, Serial Adder, Parallel Adder- Carry Propagate Adder, Carry Look-ahead Adder, Carry Save Adder, Comparators, Parity Generators, Decoder and Encoder, Multiplexer and De-multiplexer, ALU, PLA and PAL.

Error Detection and Correction Code

- Error detection and correction code plays an important role in the transmission of data from one source to another. The noise also gets added into the data when it transmits from one system to another, which causes errors in the received binary data at other systems. The bits of the data may change(either 0 to 1 or 1 to 0) during transmission.
- It is impossible to avoid the interference of noise, but it is possible to get back the original data. For this purpose, we first need to detect either an error z is present or not using **error detection codes**. If the error is present in the code, then we will correct it with the help of **error correction codes**.

Error Detection and Correction Code

- **Error detection codes** – are used to detect the errors present in the received data bitstream. These codes contain some bits, which are included appended to the original bit stream. These codes detect the error, if it is occurred during transmission of the original data bitstream. Example – Parity code, Hamming code.
- **Error correction codes** – are used to correct the errors present in the received data bitstream so that, we will get the original data. Error correction codes also use the similar strategy of error detection codes. Example – Hamming code.
- Therefore, to detect and correct the errors, additional bits are appended to the data bits at the time of transmission.

Parity Code

- It is easy to include append one parity bit either to the left of MSB or to the right of LSB of original bit stream. There are two types of parity codes, namely even parity code and odd parity code based on the type of parity being chosen.

Even Parity Code

- The value of even parity bit should be zero, if even number of ones present in the binary code. Otherwise, it should be one. So that, even number of ones present in even parity code. Even parity code contains the data bits and even parity bit.
- The following table shows the even parity codes corresponding to each 3-bit binary code. Here, the even parity bit is included to the right of LSB of binary code.

Parity Code

Binary Code	Even Parity bit	Even Parity Code
000	0	0000
001	1	0011
010	1	0101
011	0	0110
100	1	1001
101	0	1010
110	0	1100
111	1	1111

Parity Code

- Here, the number of bits present in the even parity codes is 4. So, the possible even number of ones in these even parity codes are 0, 2 & 4.
- If the other system receives one of these even parity codes, then there is no error in the received data. The bits other than even parity bit are same as that of binary code.
- If the other system receives other than even parity codes, then there will be errors in the received data. In this case, we can't predict the original binary code because we don't know the bit positions of error.
- Therefore, even parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

Parity Code- Odd Parity Code

- The value of odd parity bit should be zero, if odd number of ones present in the binary code. Otherwise, it should be one. So that, odd number of ones present in odd parity code. Odd parity code contains the data bits and odd parity bit.
- Here, the odd parity bit is included to the right of LSB of binary code.

Binary Code	Odd Parity bit	Odd Parity Code
000	1	0001
001	0	0010
010	0	0100
011	1	0111
100	0	1000
101	1	1011
110	1	1101
111	0	1110

Parity Code- Odd Parity Code

- Here, the number of bits present in the odd parity codes is 4. So, the possible odd number of ones in these odd parity codes are 1 & 3.
- If the other system receives one of these odd parity codes, then there is no error in the received data. The bits other than odd parity bit are same as that of binary code.
- If the other system receives other than odd parity codes, then there is an errors in the received data. In this case, we can't predict the original binary code because we don't know the bit positions of error.
- Therefore, odd parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

Features of Error detection codes

- These codes are used when we use message backward error correction techniques for reliable data transmission. A feedback message is sent by the receiver to inform the sender whether the message is received without any error or not at the receiver side. If the message contains errors, the sender retransmits the message.
- In error detection codes, in fixed-size blocks of bits, the message is contained. In this, the redundant bits are added for correcting and detecting errors.
- These codes involve checking of the error. No matter how many error bits are there and the type of error.
- Parity check, Checksum, and CRC are the error detection technique.

Hamming Code

- Hamming code is an example of a block code. The two simultaneous bit errors are detected, and single-bit errors are corrected by this code. In the hamming coding mechanism, the sender encodes the message by adding the unessential bits in the data. These bits are added to the specific position in the message because they are the extra bits for correction.
- Hamming code is useful for both detection and correction of error present in the received data. This code uses multiple parity bits and we have to place these parity bits in the positions of powers of 2.
- The minimum value of 'k' for which the following relation is correct valid is nothing but the required number of parity bits.
$$2^k \geq n + k + 1$$
- Where,
 - ‘n’ is the number of bits in the binary code information
 - ‘k’ is the number of parity bits
- Therefore, the number of bits in the Hamming code is equal to $n + k$.

Hamming Code

- Let the Hamming code is $b_{n+k}b_{n+k-1}\dots.b_3b_2b_1$ & parity bits p_k, p_{k-1}, \dots, p_1 . We can place the 'k' parity bits in powers of 2 positions only. In remaining bit positions, we can place the 'n' bits of binary code.
- Based on requirement, we can use either even parity or odd parity while forming a Hamming code. But, the same parity technique should be used in order to find whether any error present in the received data.
- Follow this procedure for finding parity bits.**
- Find the value of **p1**, based on the number of ones present in bit positions b_3, b_5, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^0 .
- Find the value of **p2**, based on the number of ones present in bit positions b_3, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^1 .
- Find the value of **p3**, based on the number of ones present in bit positions b_5, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^2 .
- Similarly, find other values of parity bits.

Hamming Code

- **Follow this procedure for finding check bits.**
- Find the value of c_1 , based on the number of ones present in bit positions b_1, b_3, b_5, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^0 .
- Find the value of c_2 , based on the number of ones present in bit positions b_2, b_3, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^1 .
- Find the value of c_3 , based on the number of ones present in bit positions b_4, b_5, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^2 .
- Similarly, find other values of check bits.
- The decimal equivalent of the check bits in the received data gives the value of bit position, where the error is present. Just complement the value present in that bit position. Therefore, we will get the original binary code after removing parity bits.

Example

Example 1

Let us find the Hamming code for binary code, $d_4d_3d_2d_1 = 1000$.

Consider even parity bits.

Example

- The number of bits in the given binary code is $n=4$.
- We can find the required number of parity bits by using the following mathematical relation.

$$2k \geq n+k+1$$

- Substitute, $n=4$ in the above mathematical relation.

$$\Rightarrow 2k \geq 4+k+1$$

$$\Rightarrow 2k \geq 5+k$$

- The minimum value of k that satisfied the above relation is 3. Hence, we require 3 parity bits p_1 , p_2 , and p_3 . Therefore, the number of bits in Hamming code will be 7, since there are 4 bits in binary code and 3 parity bits. We have to place the parity bits and bits of binary code in the Hamming code as shown below.

Example

- The 7-bit Hamming code is $b_7b_6b_5b_4b_3b_2b_1=d_4d_3d_2p_3d_1p_2p_1$
- By substituting the bits of binary code, the Hamming code will be $b_7b_6b_5b_4b_3b_2b_1=100p_3Op_2p_1$. Now, let us find the parity bits.

$$p_1 = b_7 \oplus b_5 \oplus b_3 = 1 \oplus 0 \oplus 0 = 1$$

$$p_2 = b_7 \oplus b_6 \oplus b_3 = 1 \oplus 0 \oplus 0 = 1$$

$$p_3 = b_7 \oplus b_6 \oplus b_5 = 1 \oplus 0 \oplus 0 = 1$$

- By substituting these parity bits, the Hamming code will be $b_7b_6b_5b_4b_3b_2b_1=1001011$.

Example

Example 2

In the above example, we got the Hamming code as $b_7b_6b_5b_4b_3b_2b_1=1001011$. Now, let us find the error position when the code received is $b_7b_6b_5b_4b_3b_2b_1=1001111$.

Example

- Now, let us find the check bits.
- $c_1 = b_7 \oplus b_5 \oplus b_3 \oplus b_1 = 1 \oplus 0 \oplus 1 \oplus 1 = 1$
- $c_2 = b_7 \oplus b_6 \oplus b_3 \oplus b_2 = 1 \oplus 0 \oplus 1 \oplus 1 = 1$
- $c_3 = b_7 \oplus b_6 \oplus b_5 \oplus b_4 = 1 \oplus 0 \oplus 0 \oplus 1 = 0$
- The decimal value of check bits gives the position of error in received Hamming code.
- $c_3 c_2 c_1 = (011)_2 = (3)_{10}$
- Therefore, the error present in third bit (b3) of Hamming code. Just complement the value present in that bit and remove parity bits in order to get the original binary code.

Introduction of ALU

- ✓ An arithmetic logic unit (ALU) is a digital electronic circuit that performs arithmetic and bitwise logical operations on integer binary numbers.
- ✓ This is in contrast to a floating-point unit (FPU), which operates on floating point numbers. It is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units.
- ✓ A single CPU, FPU or GPU may contain multiple ALUs.

History of ALU

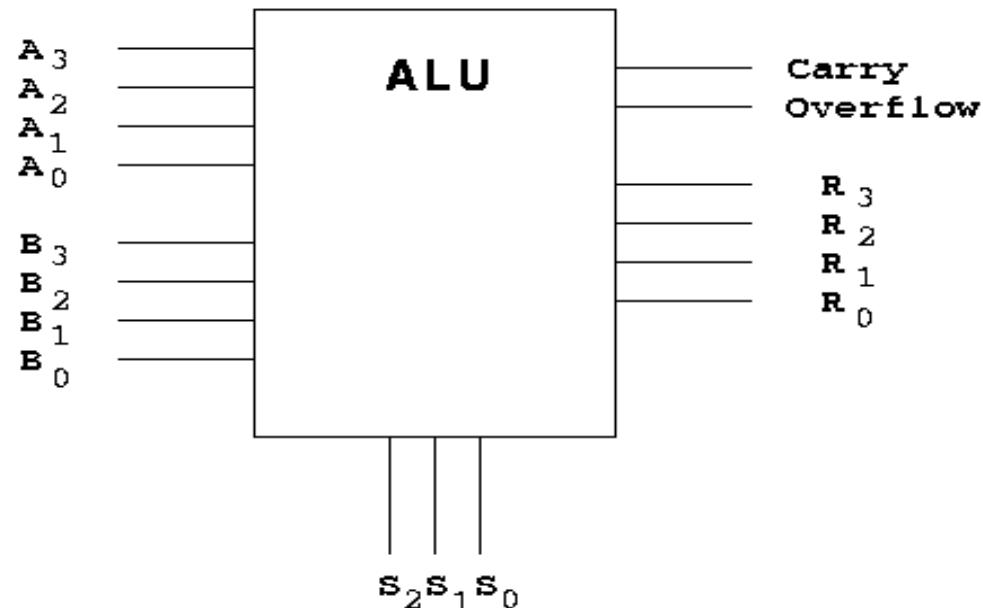
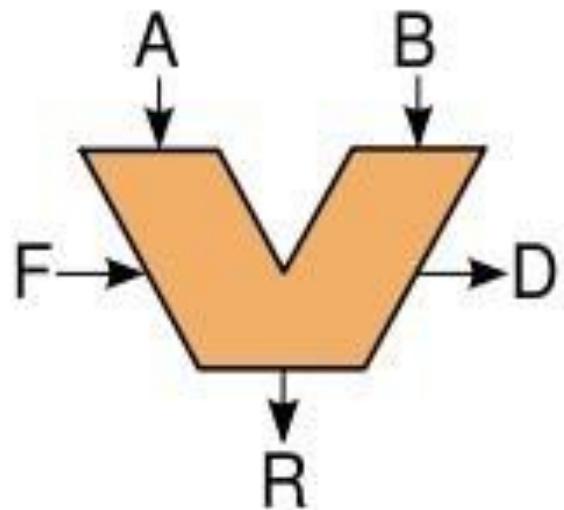
- ✓ Mathematician John von Neumann proposed the ALU concept in 1945 in a report on the foundations for a new computer called the EDVAC(Electronic Discrete Variable Automatic Computer).

What is ALU?

- ✓ ALU is the digital circuit that performs integer arithmetic and logical operation.
- ✓ An ALU is a combinational logic circuit that stands for Arithmetic logic unit
- ✓ Its outputs will change asynchronously in response to input changes
- ✓ The external circuitry connected to the ALU is responsible for ensuring the stability of ALU input signals throughout the operation.
- ✓ Most of a processor's operations are performed by one or more ALU. An ALU loads data from input registers, executes, and stores the result into an output register. A Control Unit tells the ALU what operation to perform on the data. Other mechanisms move data between these registers and memory.

Typical Schematic Symbol of an ALU

- ✓ A and B: the inputs to the ALU
- ✓ R: Output or Result
- ✓ F: Code or Instruction from the Control Unit
- ✓ D: Output status; it indicates cases



ALU Operations

- ✓ **1. Fixed point operations**
 - Add; Add with carry
 - Subtract; Subtract with Borrow
 - Multiply
 - Divide and Unsigned Divide
 - OR, XOR, NOT
 - L-Shift R-Shift
- ✓ **2. Floating point operations**
 - F add
 - F sub
 - F Mul
 - F Div
- ✓ **3. The computer uses binary code system to do arithmetic operations.**
- ✓ **4. Logical operation is performed on data it is called by data processing**

PLDs

✓ **Programmable Logic Devices (PLD)**

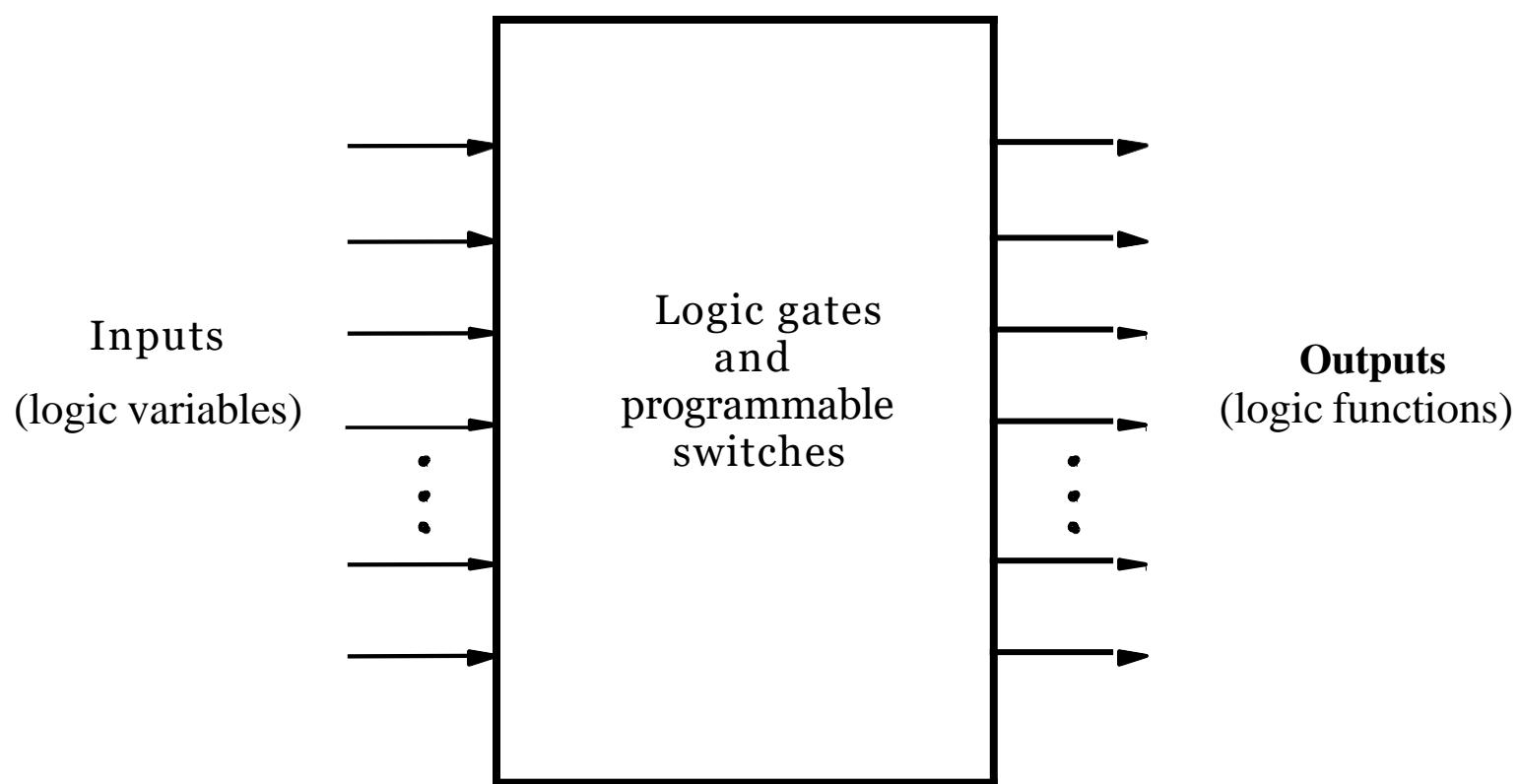
- General purpose chip for implementing circuits
- Can be customized using programmable switches

✓ **Main types of PLDs**

- PLA
- PAL
- ROM
- CPLD
- FPGA

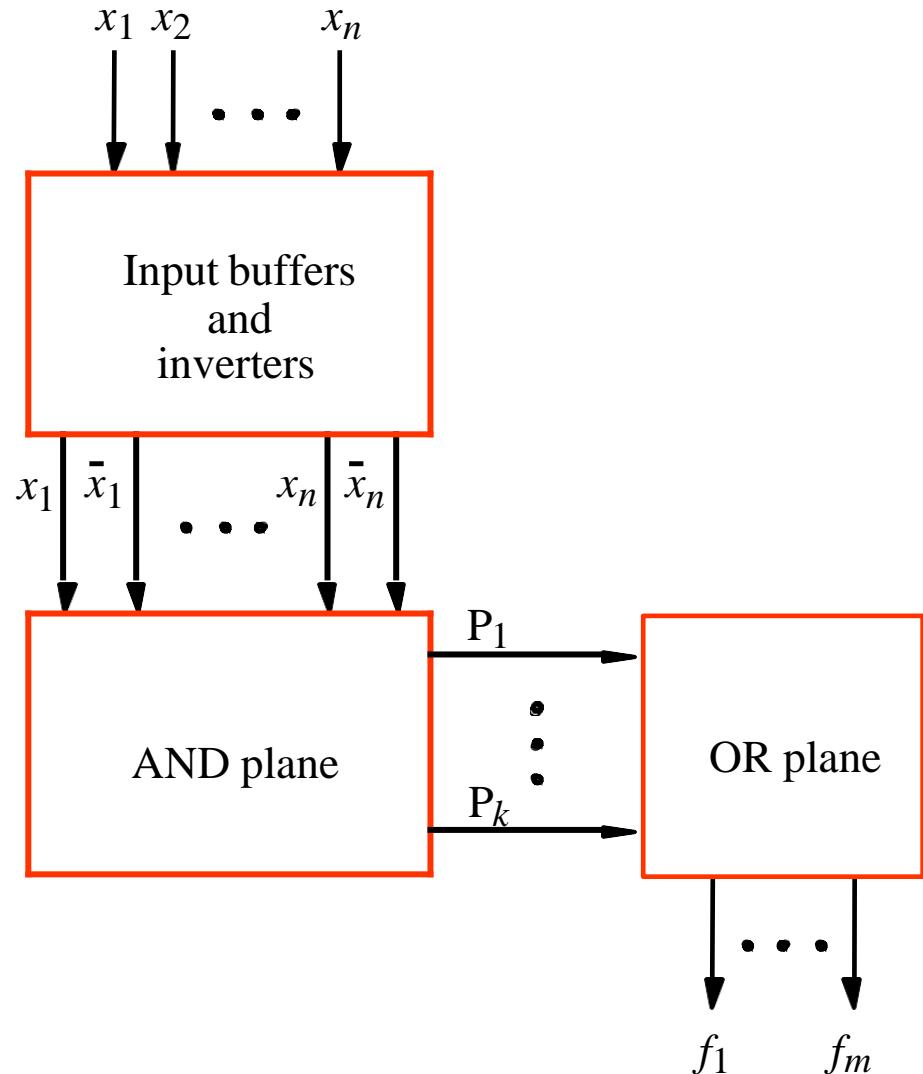
✓ **Custom chips: standard cells, sea of gates**

PLD as a Black Box



PLA(Programmable Logic Array)

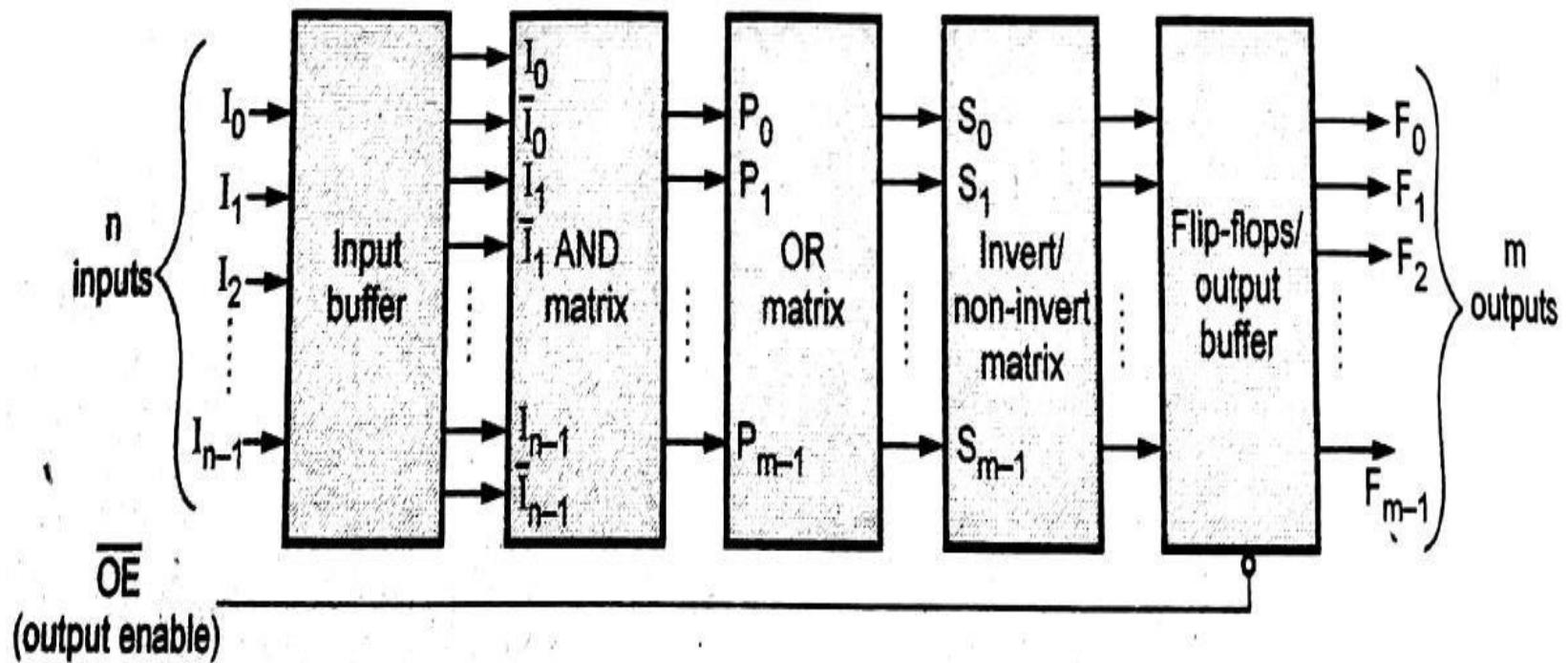
- ✓ Use to implement circuits in SOP form
- ✓ The connections in the AND plane are programmable
- ✓ The connections in the OR plane are programmable



PLA(Programmable Logic Array)

- ✓ PLA is similar to PROM but it does not provide full decoding of the variables and does not generate all the minterms.
- ✓ The PLA replaces decoder by a group of AND gates, each of which can be programmed to generate a product term of the input variables.
- ✓ There are three sets of fuses present:
 - between n-inputs & their complement values
 - between the output of the AND gates and input of the OR gates.
 - third set allows the output function to be generated either in AND-OR-INVERT form or in the AND-OR form.

PLA(Programmable Logic Array) Block Diagram



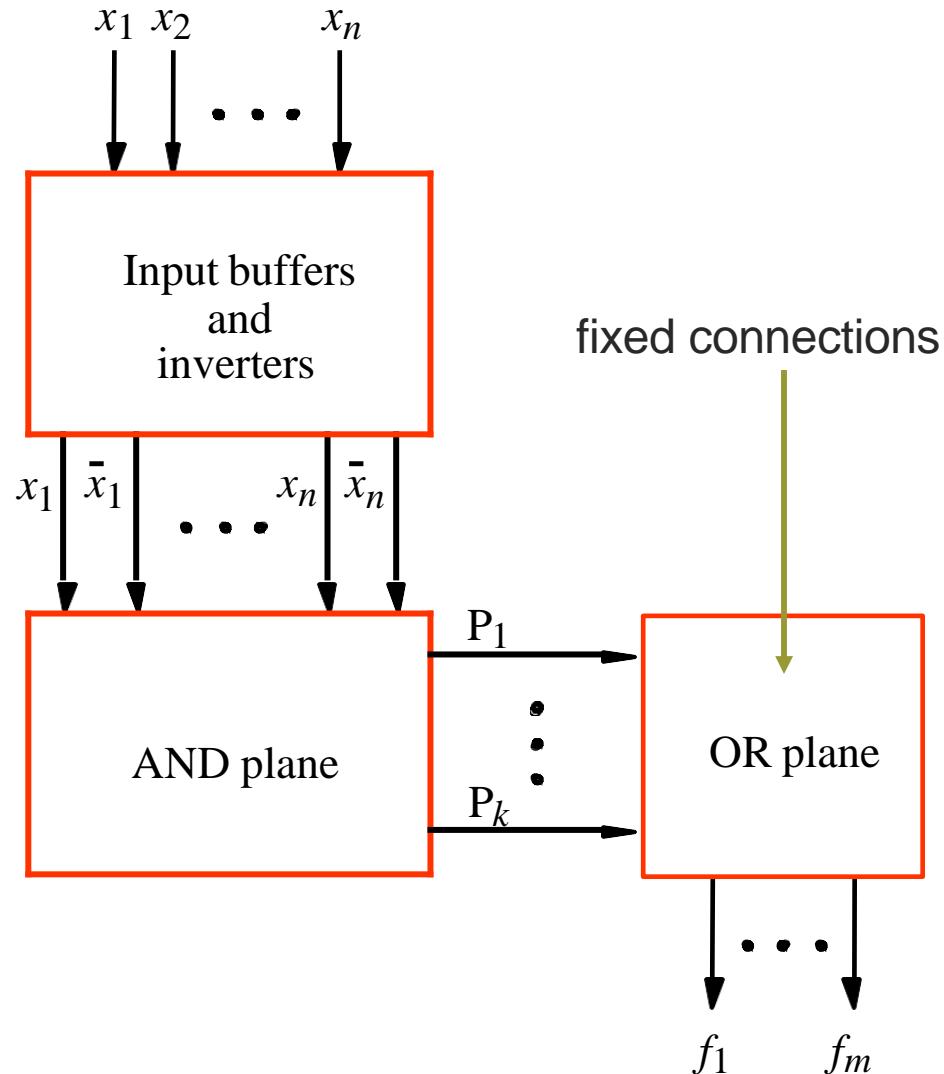
Limitations of PLAs (Programmable Logic Array)

PLAs come in various sizes

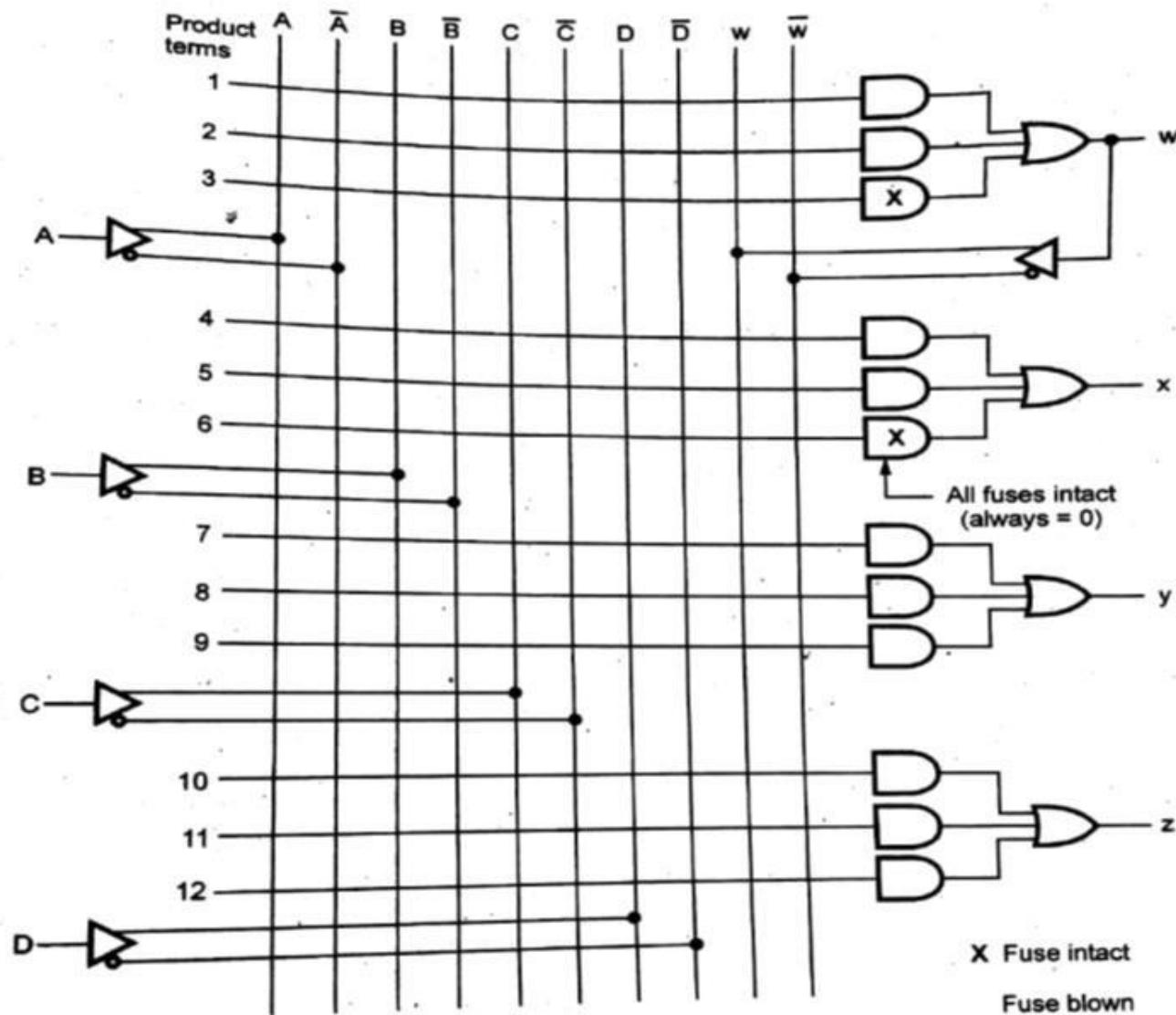
- ✓ Typical size is 16 inputs, 32 product terms, 8 outputs
- ✓ Each AND gate has large fan-in which limits the number of inputs that can be provided in a PLA
- ✓ 16 inputs = 316 possible input combinations; only 32 permitted (since 32 AND gates) in a typical PLA
- ✓ 32 AND terms permitted - large fan-in for OR gates as well which makes PLAs slower and slightly more expensive than some alternatives to be discussed shortly
- ✓ 8 outputs - could have shared minterms, but not required

Programmable Array Logic (PAL)

- ✓ Also used to implement circuits in SOP form
- ✓ It is a programmable logic device with a fixed OR array and a programmable AND array.
- ✓ Since only AND gates are programmable, the PAL is easier to program but it is not flexible as PLA.



PAL(Programmable Array Logic)



Difference between PAL & PLA

PAL	PLA
<ul style="list-style-type: none">• Both AND and OR arrays are programmable• Less flexibility than PLA	<ul style="list-style-type: none">• OR array is fixed and AND array is programmable.
<ul style="list-style-type: none">• Only desired minterms are programmed using AND array.	<ul style="list-style-type: none">• Only desired minterms are programmed using AND array.
<ul style="list-style-type: none">• PALs are simpler to manufacture, cheaper, and faster (better performance)	<ul style="list-style-type: none">• Costliest and complex than PAL and PROMs
<ul style="list-style-type: none">• PALs also often have extra circuitry connected to the output of each OR gate	

Example

Design a PLA structure using AND and OR logic for the following functions.

$$F_1 = \sum m(0, 1, 2, 3, 4, 7, 8, 11, 12, 15) ; F_2 = \sum m(2, 3, 6, 7, 8, 9, 12, 13)$$

$$F_3 = \sum m(1, 3, 7, 8, 11, 12, 15) ; F_4 = \sum m(0, 1, 4, 8, 11, 12, 15)$$

Example

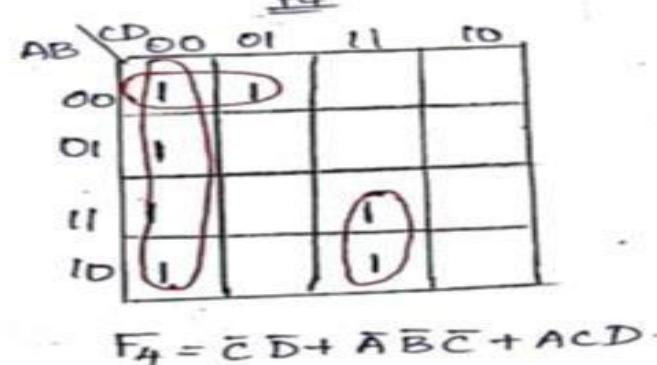
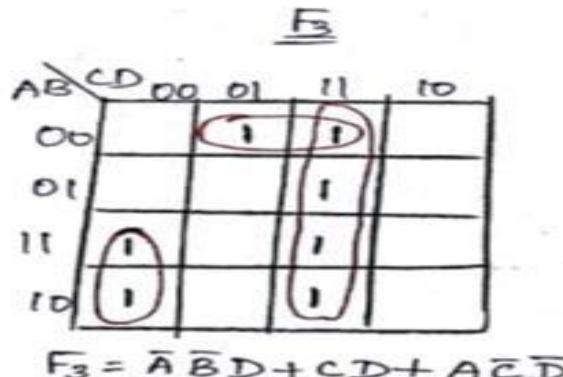
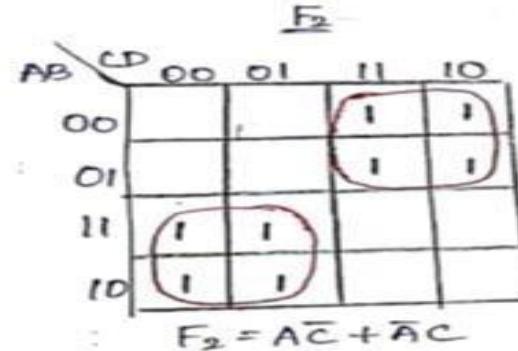
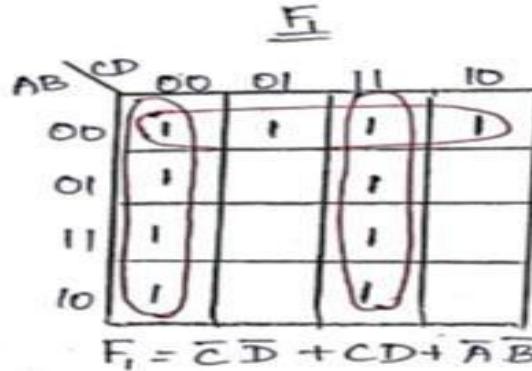
Design a PLA structure using AND and OR logic for the following functions.

$$F_1 = \sum m(0, 1, 2, 3, 4, 7, 8, 11, 12, 15) ; F_2 = \sum m(2, 3, 6, 7, 8, 9, 12, 13)$$

$$F_3 = \sum m(1, 3, 7, 8, 11, 12, 15) ; F_4 = \sum m(0, 1, 4, 8, 11, 12, 15)$$

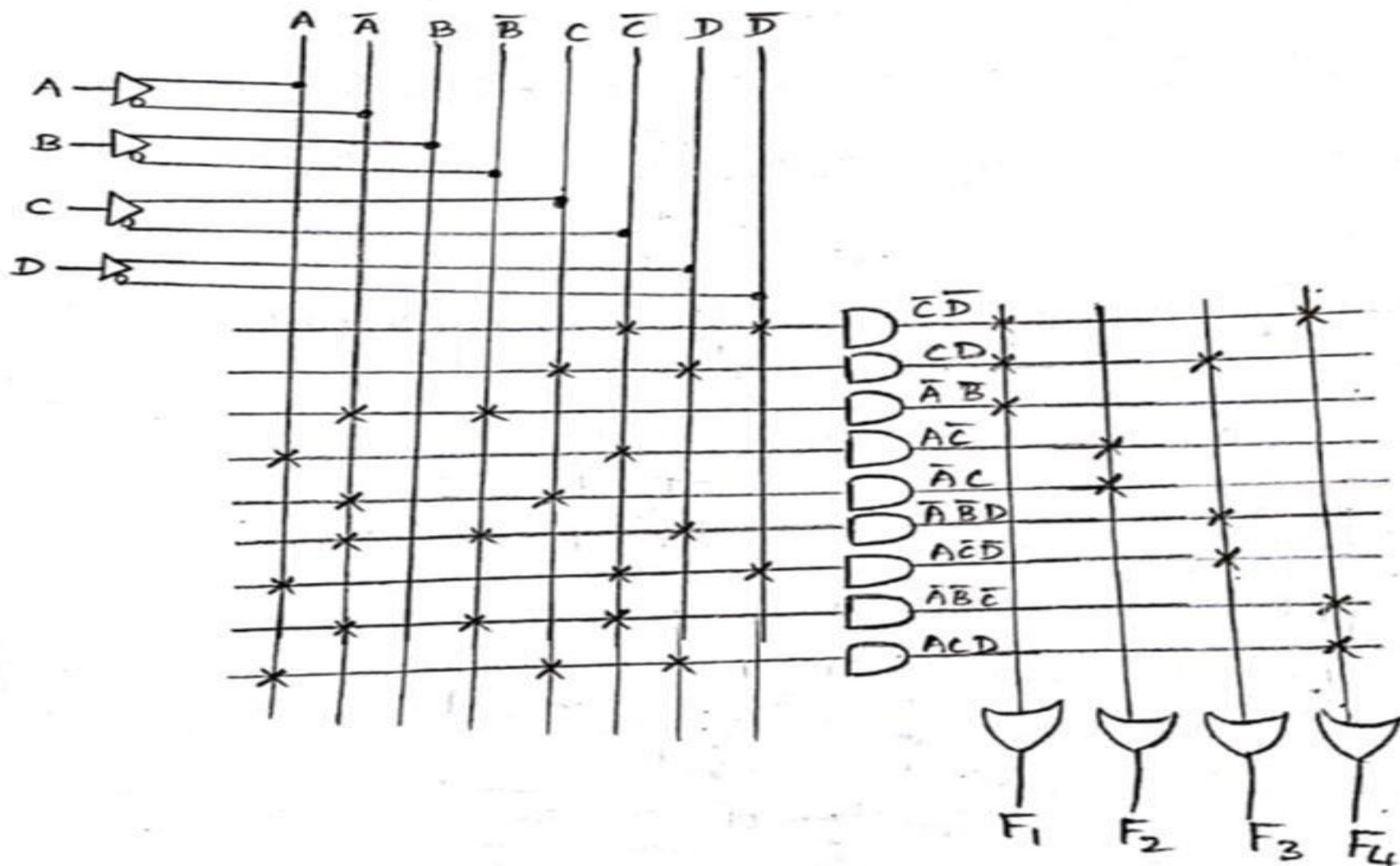
Solution:-

Step 1: Simplification of boolean function by K-map.



Example: Implementation

Step 2: Implementation.



Example

A combinational logic circuit is defined by the following function. $F_1(a,b,c) = \sum(0,1,b,7)$, $F_2(a,b,c) = \sum(2,3,5,7)$. Implement the circuit with a PAL having three inputs, three product terms and two outputs. (13)

Example

A combinational logic circuit is defined by the following function. $F_1(a,b,c) = \sum(0,1,6,7)$, $F_2(a,b,c) = \sum(2,3,5,7)$. Implement the circuit with a PAL having three inputs, three product terms and two outputs. (13)

Solution:-

$$F_1(a,b,c) = \sum(0,1,6,7)$$

$$F_2(a,b,c) = \sum(2,3,5,7)$$

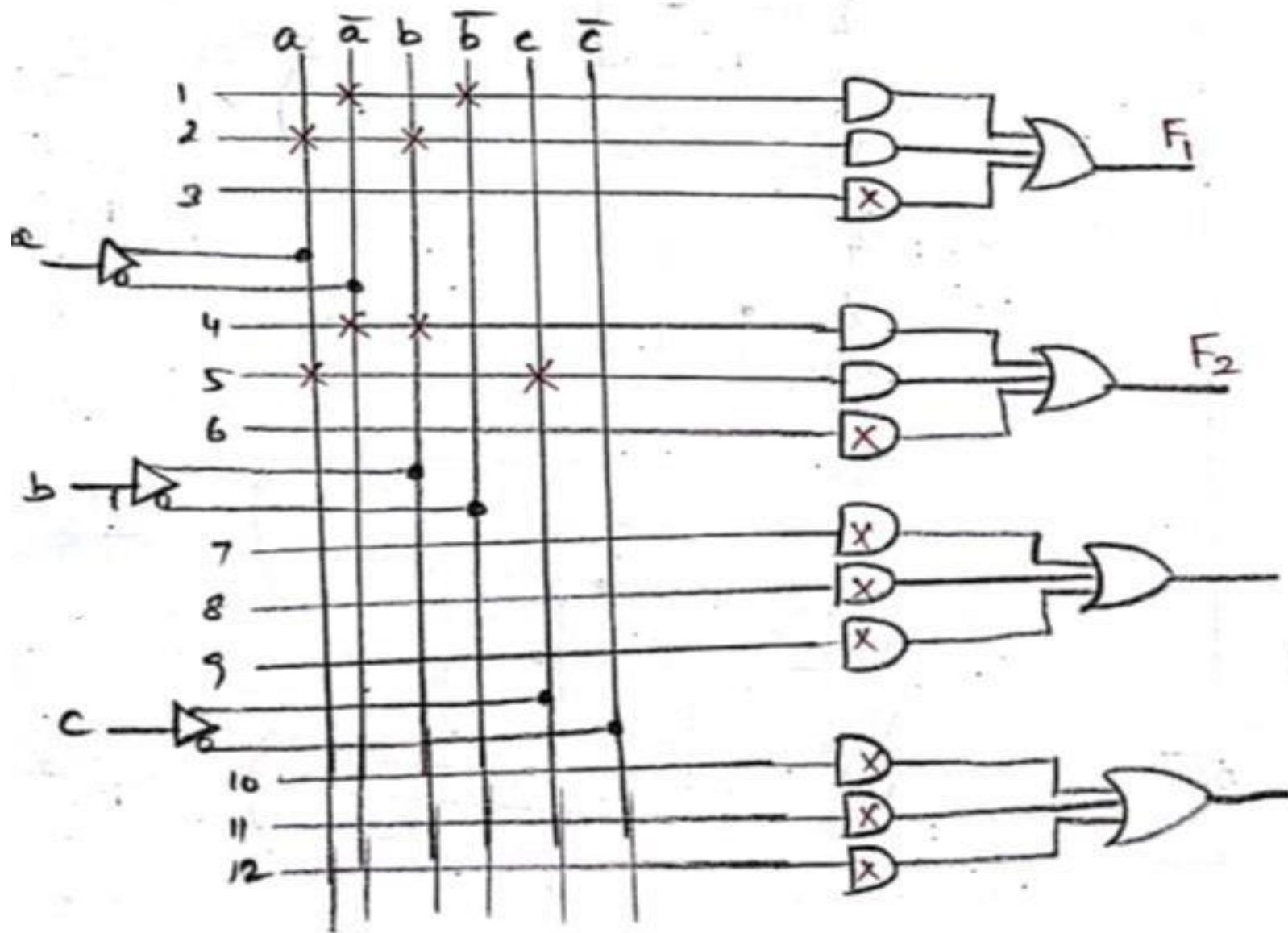
		<u>F₁</u>			
		00	01	11	10
a	0	1	1	0	0 ₂
	1	0 ₄	0 ₅	1	1 ₆

$$F_1(a,b,c) = \bar{a}\bar{b} + ab$$

		<u>F₂</u>			
		00	01	11	10
a	0	0	0 ₁	1	1 ₂
	1	0 ₄	1 ₅	1 ₇	0 ₆

$$F_2(a,b,c) = \bar{a}b + ac$$

Example: Implementation



Thank You