Presentation Topic

Analog and Digital Electronics

Yogesh K Sharma
yogesh.sharma@viit.ac.in
Department of Computer Engineering

BRACT'S, Vishwakarma Institute of Information Technology, Pune-48

(An Autonomous Institute affiliated to Savitribai Phule Pune University)
(NBA and NAAC accredited, ISO 9001:2015 certified)

Combinational Logic
Unit-2

Course Objectives

1. To learn and understand basic digital circuit design techniques
2. To study the implementation of digital circuits using combinational logic.
3. To study the implementation of digital circuits using sequential logic.
4. To illustrate the concept of PLD's.
5. To study the implementation of digital circuits using VHDL.
6. To understand basics of Logic Families and IOT circuit boards in development of mini digital circuits.

Course Outcome

1. Simplify Boolean algebraic expressions for designing digital circuits using K-Maps.(Analyzing)
2. Apply digital concepts in designing combinational circuits.(Applying)
3. Apply digital concepts in designing sequential circuits. (Applying)
4. Implement digital circuits using PLA and PAL. (Applying)
5. Design digital circuits using VHDL. (Applying)
6. Design and implementation of Mini digital circuit applications. (Applying)

.

Contents

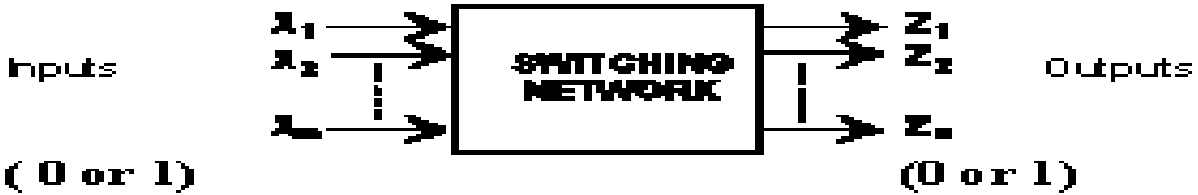
Unit-2: Combinational Logic.

- Design of Combinational Logic:
- Code converter - BCD, Excess-3, Gray code, Binary Code.
- Half- Adder, Full Adder, Half Subtractor, Full Subtractor, Binary Adder (IC 7483), BCD adder, Look ahead carry generator.
- Multiplexers (MUX): MUX (IC 74153, 74151), MUX tree,
- Demultiplexers (DEMUX)- Decoder. (IC 74138, IC 74154), DMUX Tree, Implementation of SOP and POS using MUX,
- DMUX, Comparators, Parity generators and Checker, One bit, Two bit , 4-bit Magnitude Comparator

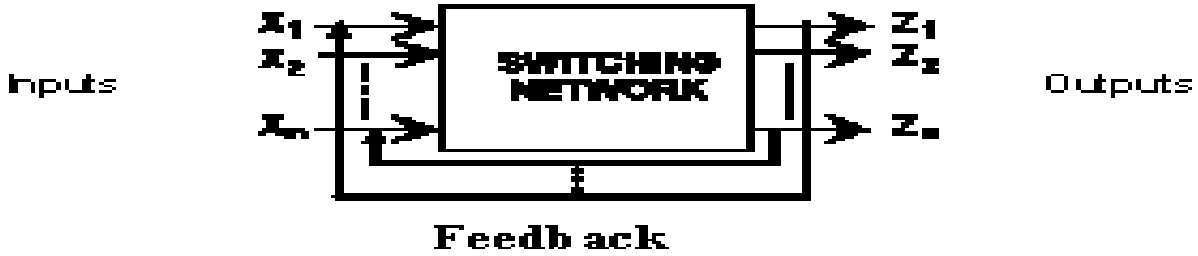
Combinational Logic

Two Types of Networks

Combinational: Output values depend only on *present* input values.



Sequential: Output values depends on present *and past* input values. i.e. A *sequence* of I/P values must be specified to define the O/P.



Why Digital ??

Why digital? - greater accuracy & reliability
- more versatile & cheaper
- more comprehensive theory and algorithms
- availability of CAD tools
- optimized device processes

Digital circuits used in:

Digital Computers	Data Processing	
Electronic Calculators	Instrumentation	
Control Devices	etc.	Telephone Networks, Cell Phones, CD Players, Medical Equipment, Modern TV sets, Modern Radios, etc.
Communication Equipment		

Analog Systems

Advantages

- most physical phenomena of interest are analog
- transducers are simple
- potentially high precision

Disadvantages

- behaviour of analog components is subject to drift distortion, noise, offsets, etc.
- errors in analog signals accumulate during processing, transmission, and storage
- only relatively simple signal processing is practical for most applications

Digital Circuits

Advantages

- the strength of digital signals is easily restored
- signal accuracy degrades very little during processing, transmission and storage
- digital components are cheap, reliable and low-power
- digital signal processing can be highly sophisticated using special-purpose hardware or programmable digital computers

Disadvantages

- signal precision is limited by the number of bits used to encode each sample
- analog-to-digital converters and digital-to-analog converters are required to interface a digital system with real-world analog signals

Combinational Logic

Codes Converter:

- **BCD**
- **Excess-3**
- **Gray Code**
- **Binary Code Conversion**

Combinational Logic

Codes Converter:

- **BCD**
- **Excess-3**
- **Gray Code**
- **Binary Code Conversion**

Combinational Logic

Natural BCD Code: Decimal digits 0 through 9 are represented by their natural binary equivalents using 4 bits.

Excess 3 code: Another BCD code in 4 bit binary code. The decimal digit is obtained by adding 3 to the natural BCD code.

Gray code: It is a code in which each gray code number differs from the preceding & succeeding number by a single bit.

Construction of Gray Code:

1. **A 1 bit gray code has two code words 0 and 1 representing decimal numbers 0 & 1 respectively.**
2. **An n-bit (n>=2) gray code will have first 2^{n-1} gray codes of (n-1) bits written in order with a leading 0 appended.**
3. **The last 2^{n-1} gray codes will be equal to the gray code words of an (n-1) bit gray code written in reverse order with a leading 1 appended.**

Code Converter

E.g. :

1. 1 bit gray code

Decimal number	Gray Code
0	0
1	1

2. 2 bit gray code

Decimal number	Gray Code	
0	0	0
1	0	1
2	1	1
3	1	0

3. 3 bit Gray Code

Decimal number	Gray Code		
0	0	0	0
1	0	0	1
2	0	1	1
3	0	1	0
4	1	1	0
5	1	1	1
6	1	0	1
7	1	0	0

Table 2.8 Various binary codes

Decimal Number	Binary				BCD				Excess-3				Gray			
	B_3	B_2	B_1	B_0	D	C	B	A	E_3	E_2	E_1	E_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
1	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	1
2	0	0	1	0	0	0	1	0	0	1	0	1	0	0	1	1
3	0	0	1	1	0	0	1	1	0	1	1	0	0	0	1	0
4	0	1	0	0	0	1	0	0	0	1	1	1	0	1	1	0
5	0	1	0	1	0	1	0	1	1	0	0	0	0	1	1	1
6	0	1	1	0	0	1	1	0	1	0	0	1	0	1	0	1
7	0	1	1	1	0	1	1	1	1	0	1	0	0	1	0	0
8	1	0	0	0	1	0	0	0	1	0	1	1	1	1	0	0
9	1	0	0	1	1	0	0	1	1	1	0	0	1	1	0	1
10	1	0	1	0									1	1	1	1
11	1	0	1	1									1	1	1	0
12	1	1	0	0									1	0	1	0
13	1	1	0	1									1	0	1	1
14	1	1	1	0									1	0	0	1
15	1	1	1	1									1	0	0	0

Courtesy: Modern Digital Electronics by R. P. Jain

Natural BCD Code or 8-4-2- 1 Code

Decimal digits 0 to 9 are represented in binary codes, ie, 0000 to 1001

10 is represented as 0001 0000 (eight bits)

47 0100 0111

Excess 3 Code

Obtained by adding 3 to BCD code

Decimal 2 is coded as 0010 + 0011 = 0101

Used in subtraction

Gray Code

Successive codes differ by only bit

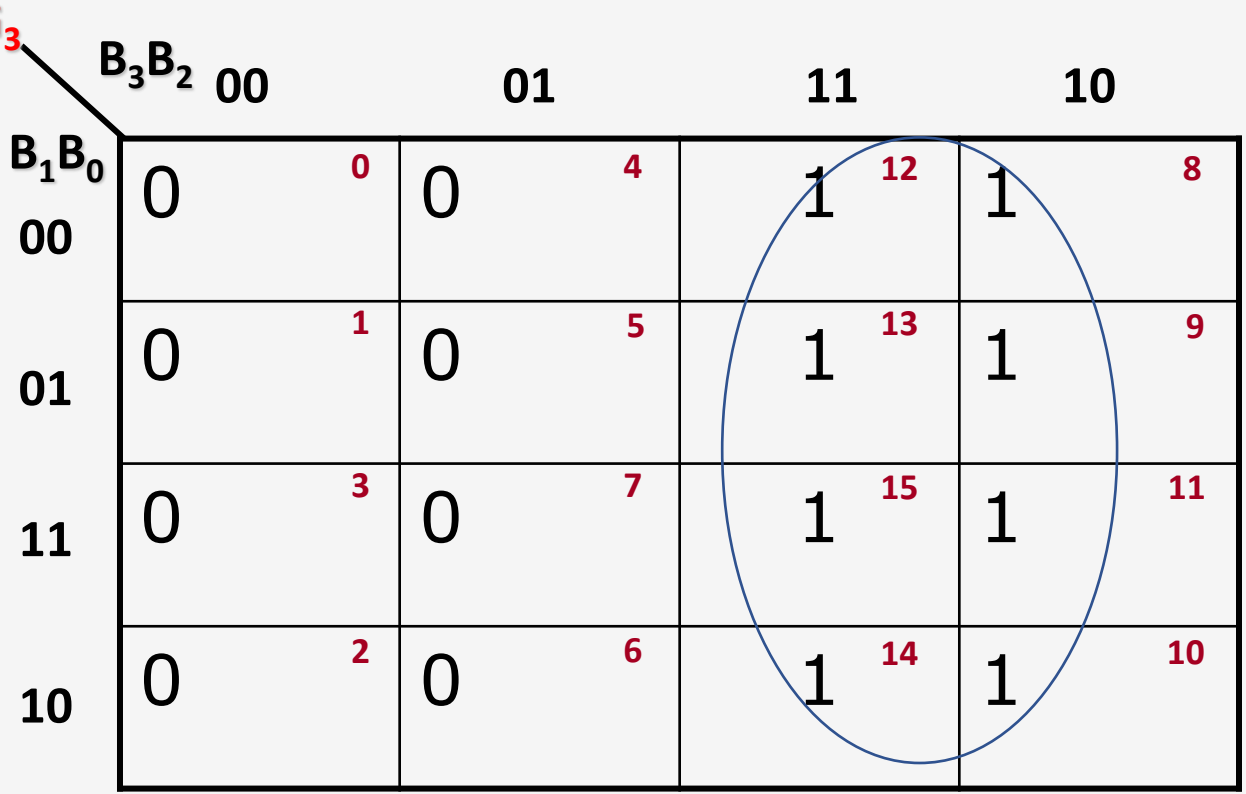
Used in shaft **encoders** and k maps

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

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

Binary to Gray Code Converter

Binary				Gray			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

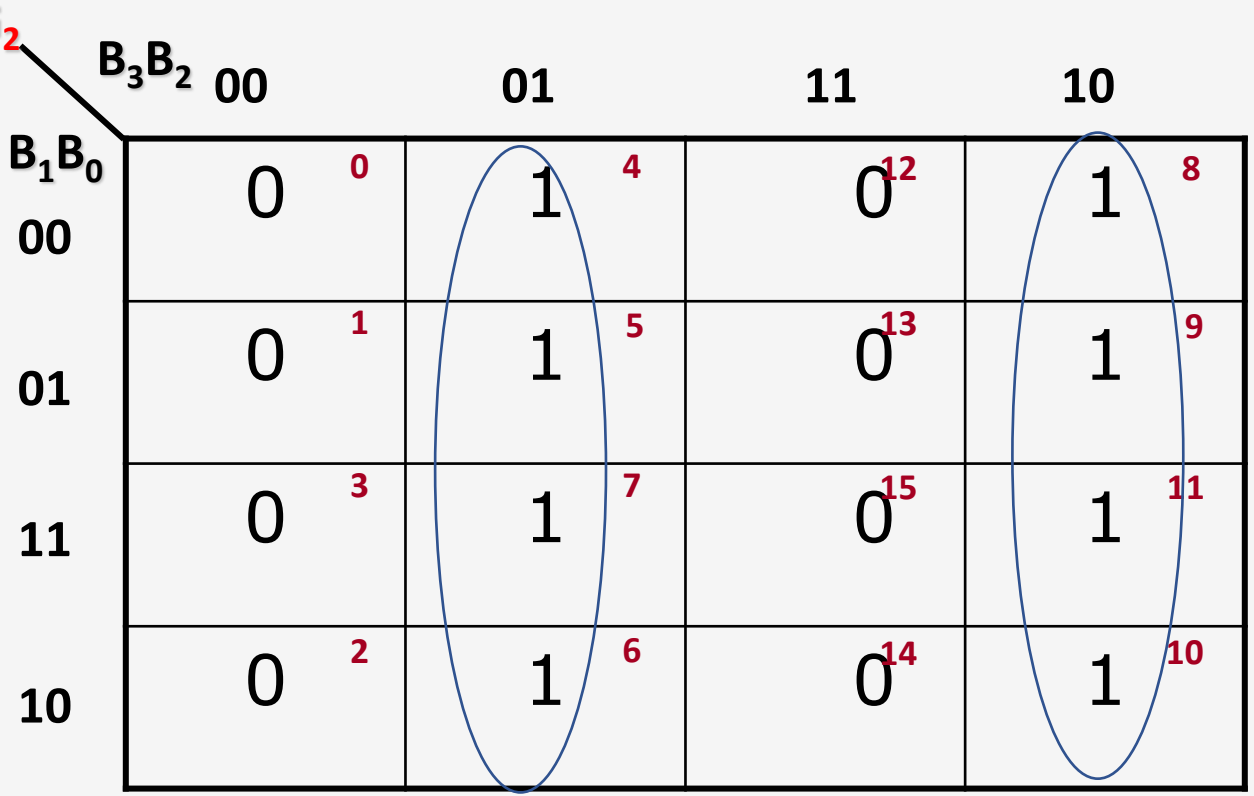


$$G_3 = B_3$$

Courtesy: Modern Digital Electronics by R. P. Jain

Binary to Gray Code Converter

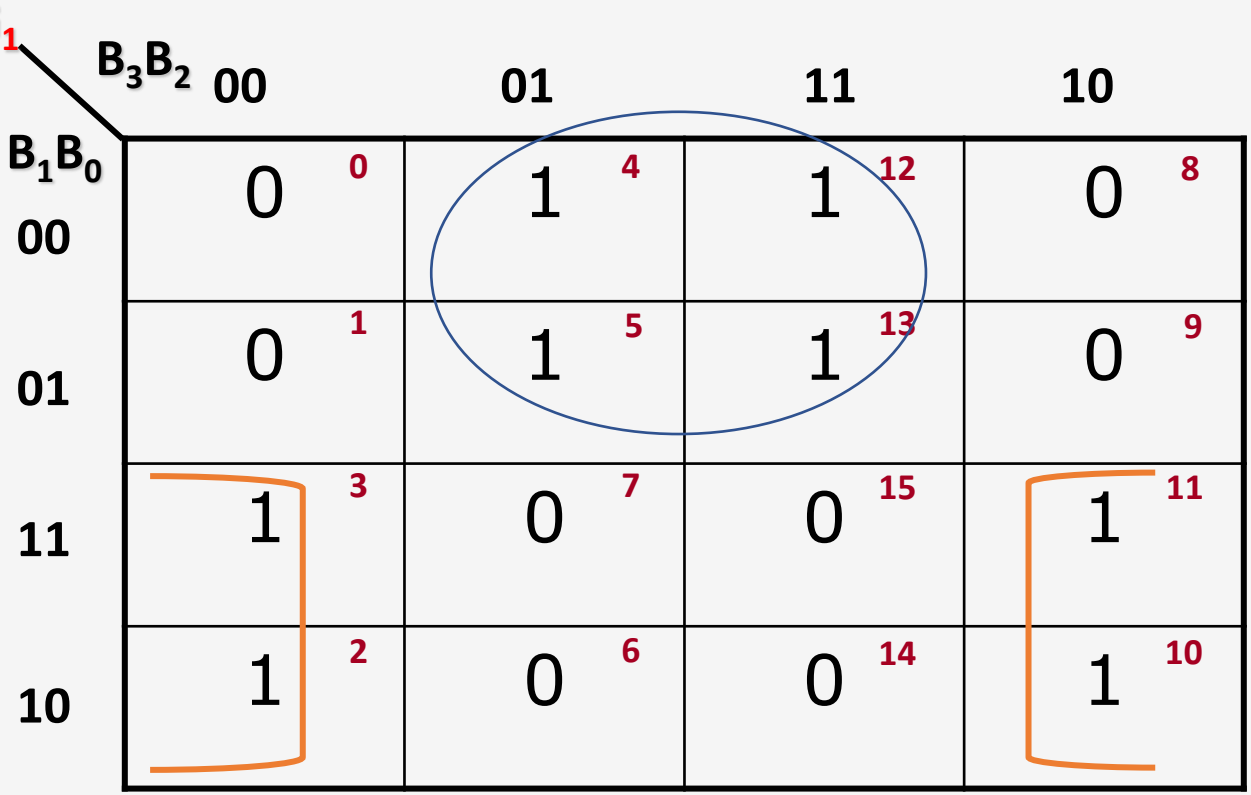
Binary				Gray			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0



$$G_2 = \overline{B_3}B_2 + B_3\overline{B_2} = B_3 \oplus B_2$$

Binary to Gray Code Converter

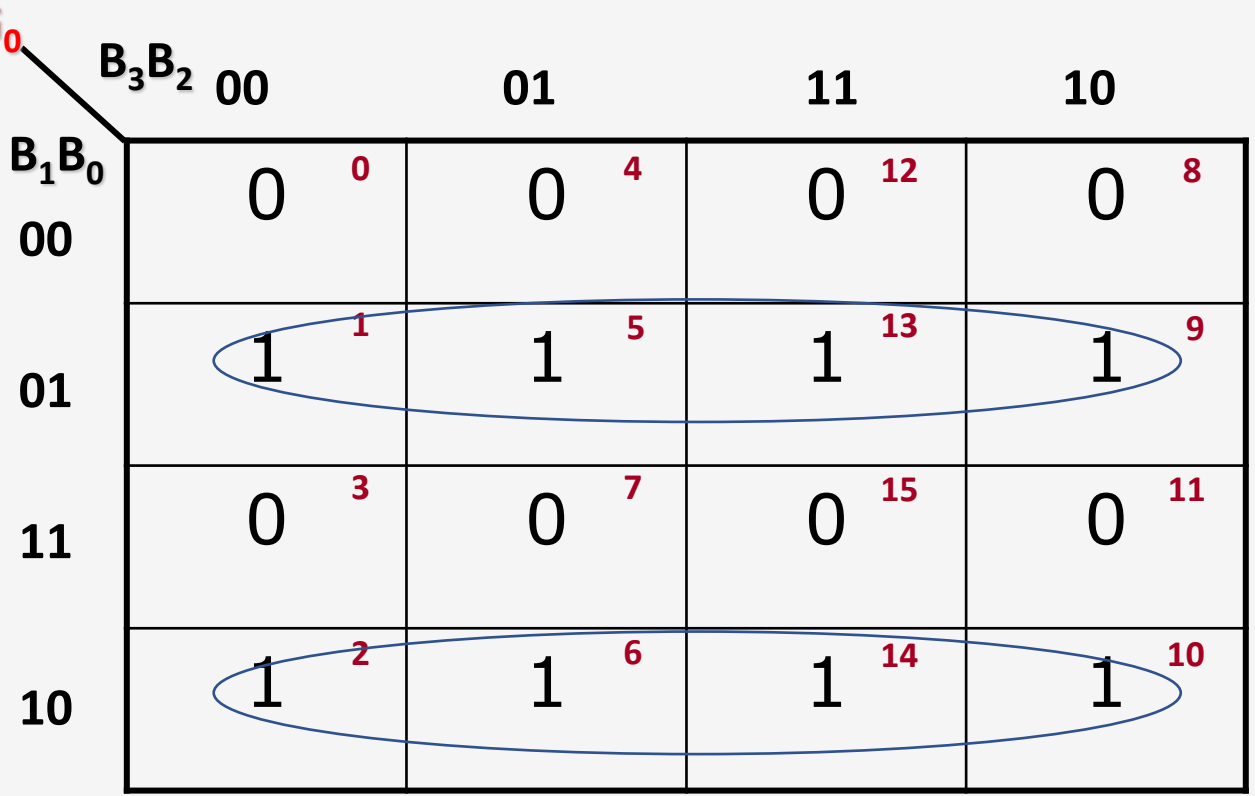
Binary				Gray			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0



$$G_1 = \overline{B_1}B_2 + B_1\overline{B_2} = B_1 \oplus B_2$$

Binary to Gray Code Converter

Binary				Gray			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0



$$G_0 = \overline{B_1}B_0 + B_1\overline{B_0} = B_1 \oplus B_0$$

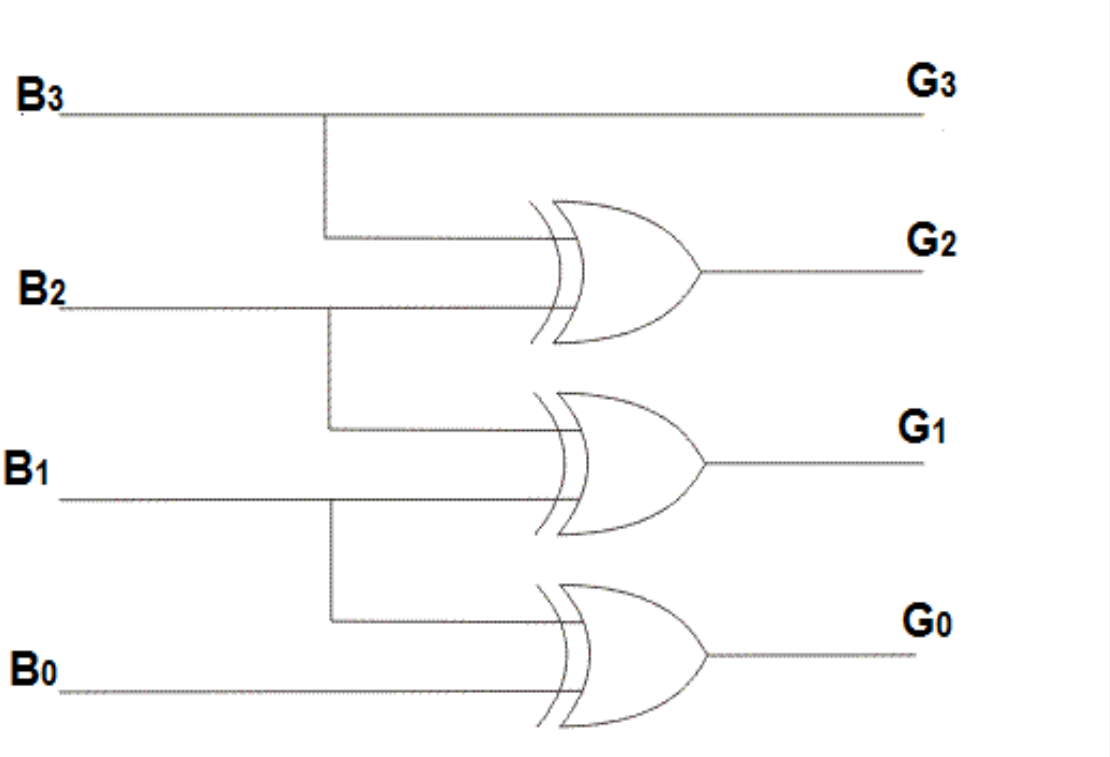
Binary to Gray Code Converter

$$G_3 = B_3$$

$$G_2 = B_3 \oplus B_2$$

$$G_1 = B_1 \oplus B_2$$

$$G_0 = B_1 \oplus B_0$$



Gray to Binary Code Converter

Example 5.20

Design a Gray-to-Binary code converter.

Solution

The truth table for this is given in Table 2.8. The K-maps are given in Fig. 5.36 and the simplified expressions are given by Eqs (5.51).

The converter circuit is given in Fig. 5.37.

$$B_3 = G_3 \qquad (5.51a)$$

Gray to Binary Code Converter

$$B_2 = G_2 \oplus G_3 \quad (5.51b)$$

$$B_1 = G_1 \oplus G_2 \oplus G_3 \quad (5.51c)$$

$$B_0 = G_0 \oplus G_1 \oplus G_2 \oplus G_3 \quad (5.51d)$$

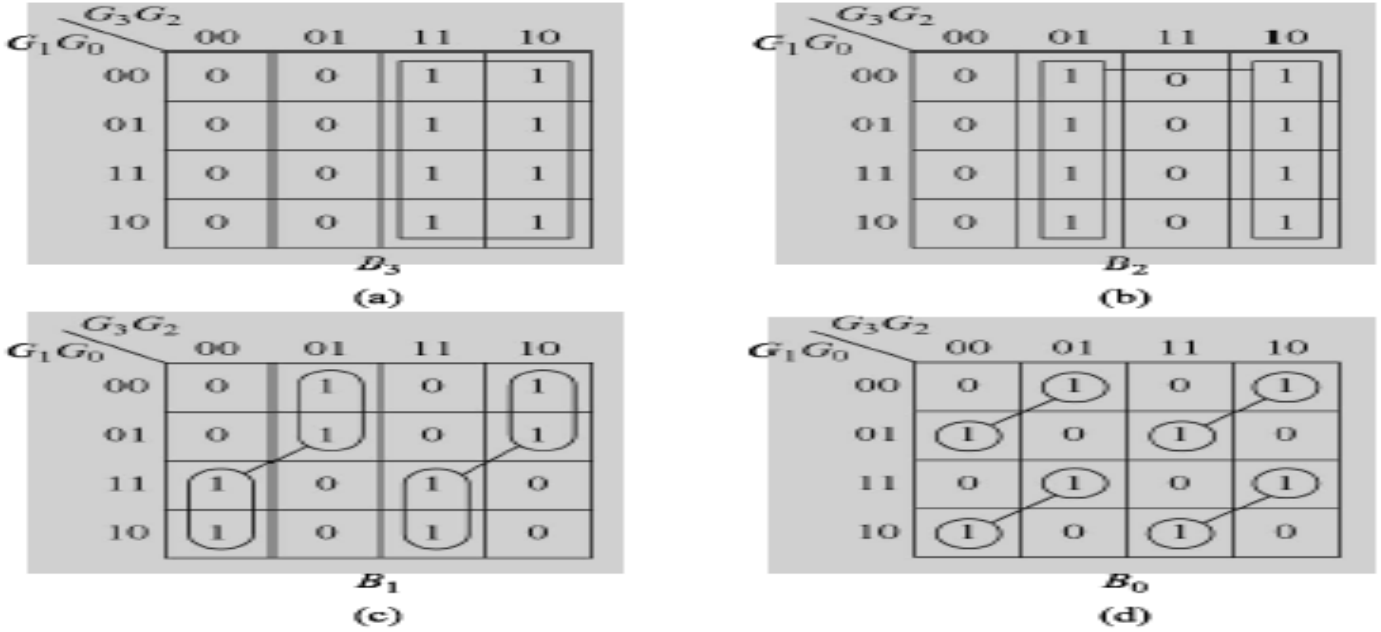


Fig. 5.36 *K-maps of Ex. 5.20*

Gray to Binary Code Converter

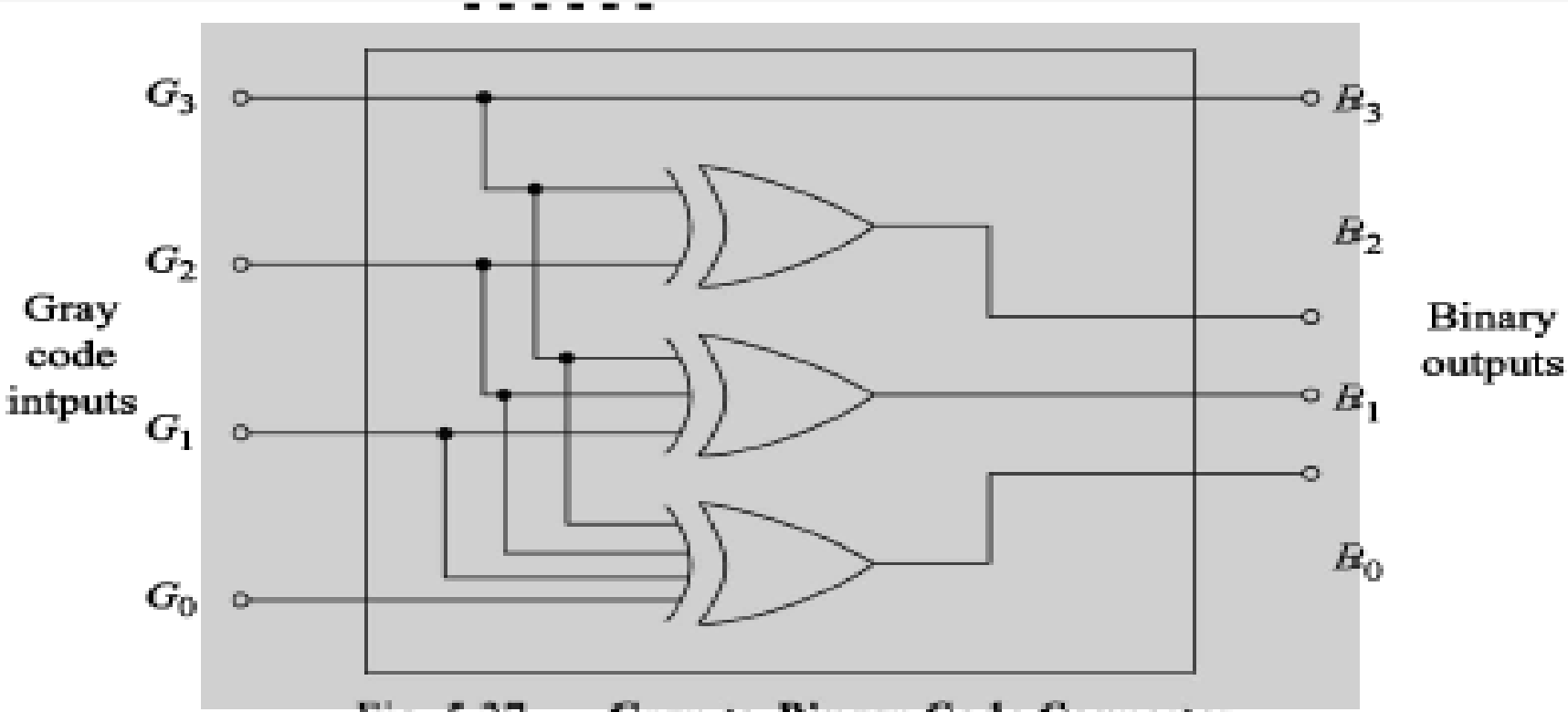


Fig. 5.37 *Gray-to-Binary Code Converter*

Arithmetic Operations

- **Binary Addition**
- **Subtraction**
- **Multiplication**
- **Division**
- **BCD Addition**

Binary Adders

- *Half-adder*: the circuit accepts two inputs, A and B, and generates two outputs, SUM and CARRY according to the table
- Half adder truth table

Inputs		*Outputs*	
A	*B*	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

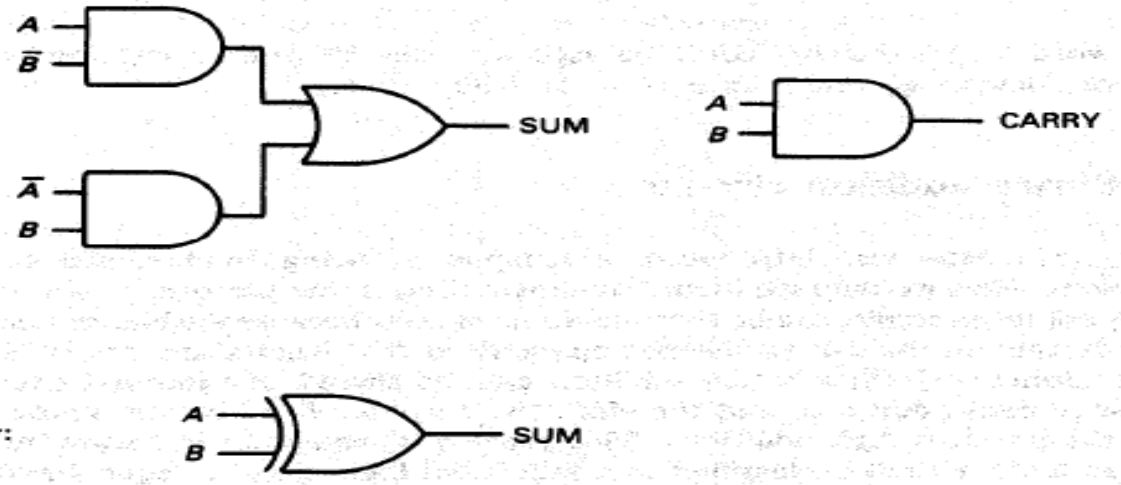
Binary Adders

- The Boolean expression for SUM and CARRY:

$$SUM = A'B + AB'$$

$$CARRY = AB$$

- Half adder circuit



Binary Adders

- *Full adder*: adds two binary digits A and B together with a 'carry-in' from a previous addition. A full adder has three inputs, A, B and 'CARRY-IN' (abbreviated here to C_{in}), and two outputs, SUM and 'CARRY OUT' (abbreviated to C_{out})
- Full adder circuit

Inputs			*Outputs*	
A	**B**	**C_{in}**	**SUM**	**C_{out}**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

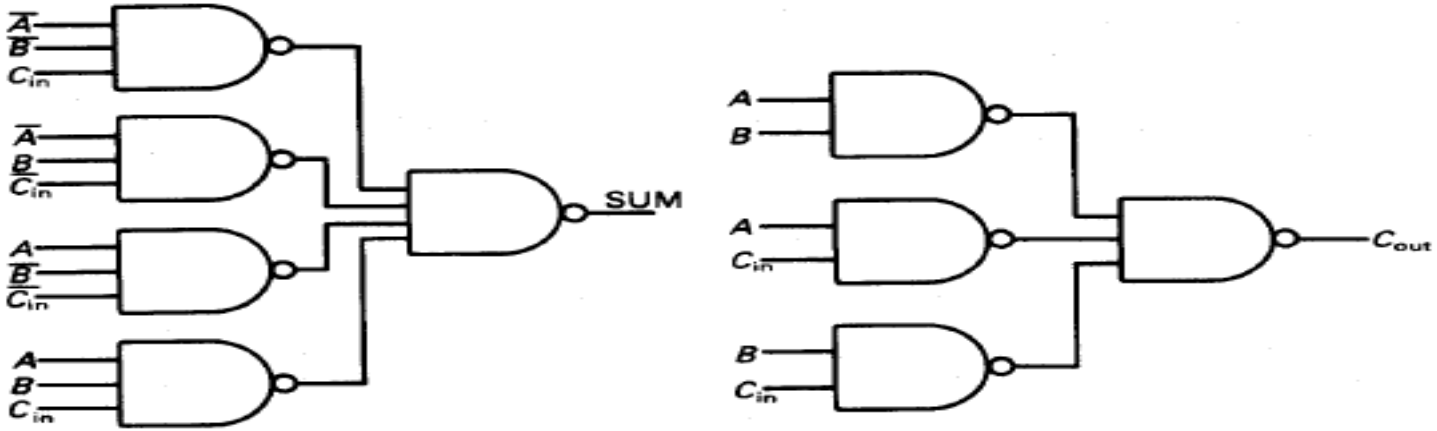
Binary Adders

- The Boolean expression for SUM and C_{out}:

$$SUM = A'B'C_{in} + A'BC'_{in} + AB'C'_{in} + ABC_{in}$$

$$C_{out} = \underline{A'BC_{in}} + AB'C_{in} + ABC'_{in} + \underline{ABC_{in}}$$

- Full adder circuit



Binary Adders

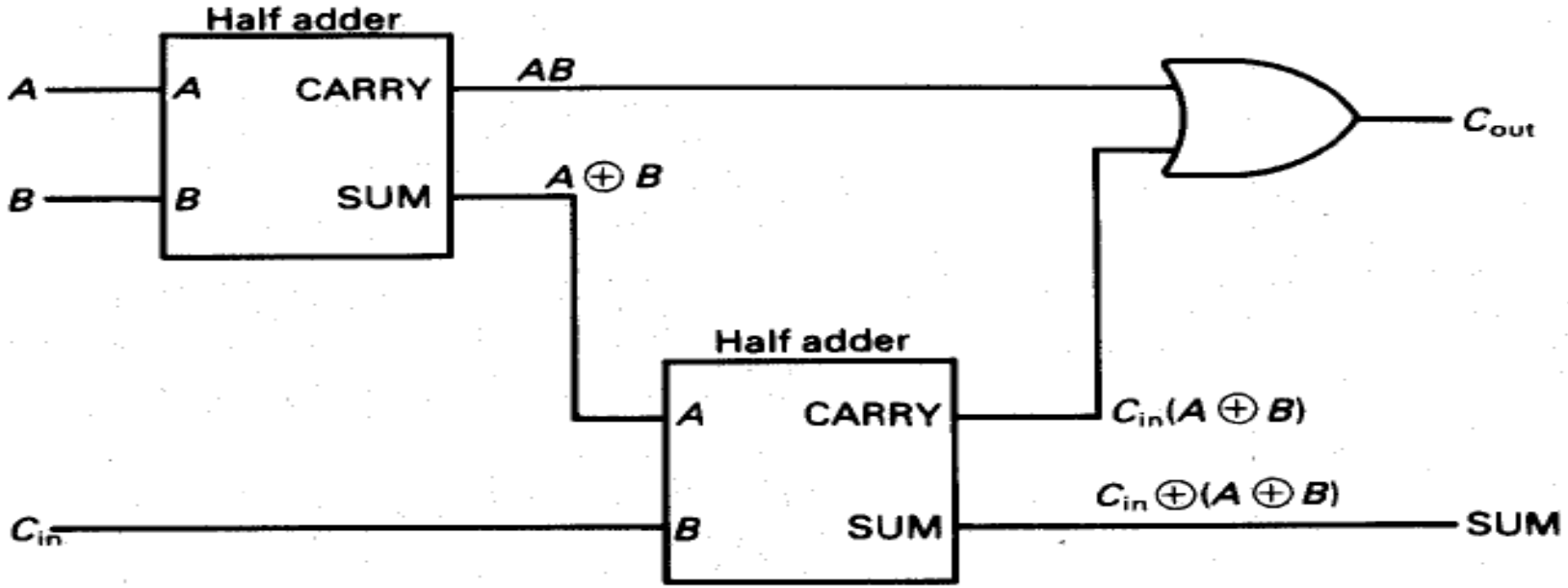
- A full adder can also be formed from two half adders

$$
\begin{aligned}
\text{SUM} &= \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in} \\
&= C_{in}(\bar{A}\bar{B} + AB) + \bar{C}_{in}(\bar{A}B + A\bar{B}) \\
&= C_{in}\overline{(\bar{A}B + A\bar{B})} + \bar{C}_{in}(\bar{A}B + A\bar{B}) \\
&= C_{in}\overline{(A \oplus B)} + \bar{C}_{in}(A \oplus B) \\
&= C_{in} \oplus (A \oplus B)
\end{aligned}
$$

$$
\begin{aligned}
C_{out} &= \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in} \\
&= C_{in}(\bar{A}B + A\bar{B}) + AB(\bar{C}_{in} + C_{in}) \\
&= C_{in}(\bar{A}B + A\bar{B}) + AB \\
&= C_{in}(A \oplus B) + AB
\end{aligned}
$$

Binary Adders

- Implementation of a full adder

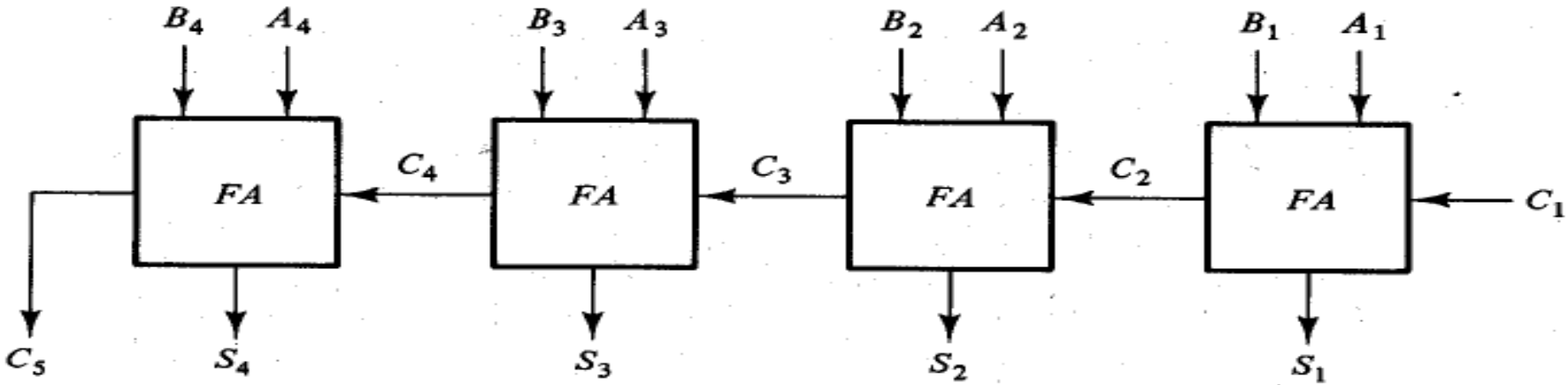


Binary Parallel Adders

- ***Parallel adder*** **: a digital circuit that produces the arithmetic sum of two binary numbers in parallel. It consists of full-adders connected in a chain, with the output carry from each full-adder connected to the input carry of the next full adder in the chain.**

- **The figure shows the interconnection of four full-adder (FA) circuits to provide a 4-bit binary parallel adder.**

Binary Parallel Adders

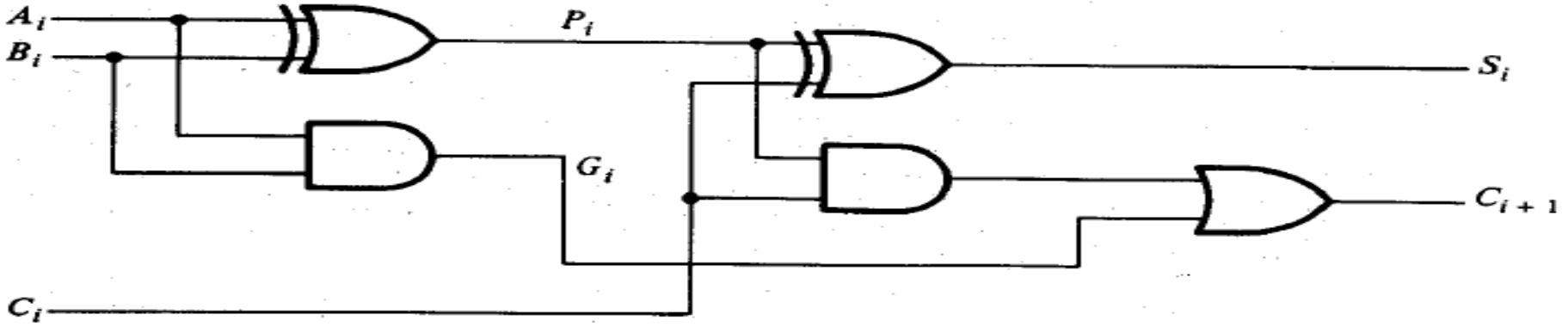


(a) 4-bit parallel adder

- Carry propagation: The longest propagation delay time in a parallel adder is the time it takes the carry to propagate through the full adders

Binary Parallel Adders

- The carry propagation time is a limiting factor on the speed with which two numbers are added in parallel
  - Employ faster gates with reduced delays
  - Increase the equipment complexity to reduce the delay time, for example, *look-ahead* carry
- Consider the circuit of the full-adder:



Binary Parallel Adders

- If we define two new binary variables:

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

- G_i is called a *carry generate* and it produces an output carry when both A_i and B_i are one, regardless of the input carry
- P_i is called a *carry propagate* because it is the term associated with the propagation of the carry from C_i to C_{i+1}

Binary Parallel Adders

- The output sum and carry can be expressed as

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

- Carry output of each stage:

$$C_2 = G_1 + P_1 C_1$$
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2(G_1 + P_1 C_1) = G_2 + P_2 G_1 + P_2 P_1 C_1$$
$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1$$

Binary Adders

- **Half Subtractor: A logic circuit for the subtraction of B (Subtrahend) from A (minuend) where A and B are 1-bit numbers is referred to as a half-subtractor. A and B are two inputs and D (difference) and C (borrow) are the two outputs.**

Truth Table:

Inputs		Outputs	
A	**B**	**D**	**C**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

From the truth table the logical expression for D & C is:

$D = A'B + AB' = A \oplus B$ & $C = A'B$

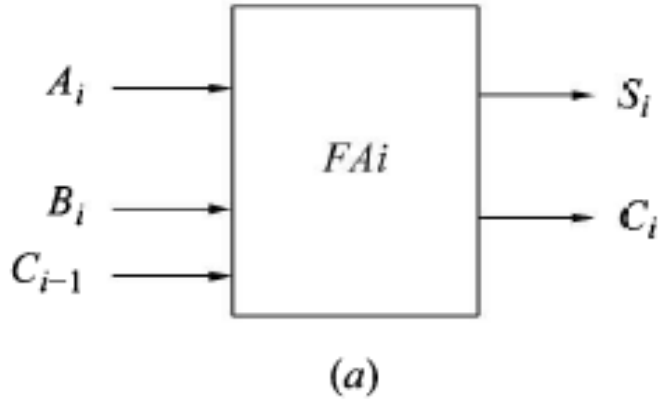
Binary Adders

- **Full Subtractor: Like a full-adder we require a full-subtractor circuit for performing multibit subtraction wherein a borrow from the previous bit position may be present as input.**
- **It will have three inputs, A (minuend), B (subtrahend), C (borrow from the previous stage) and two outputs, D (difference) and C (borrow).**
- **The truth table is:**

Inputs			Outputs	
A_n	B_n	C_{n-1}	D_n	C_n
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

Adder with Look-Ahead Carry

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$
$$S_i = P_i \oplus C_{i-1} = A_i \oplus B_i \oplus C_{i-1}$$
$$C_i = G_i + P_i C_{i-1}$$



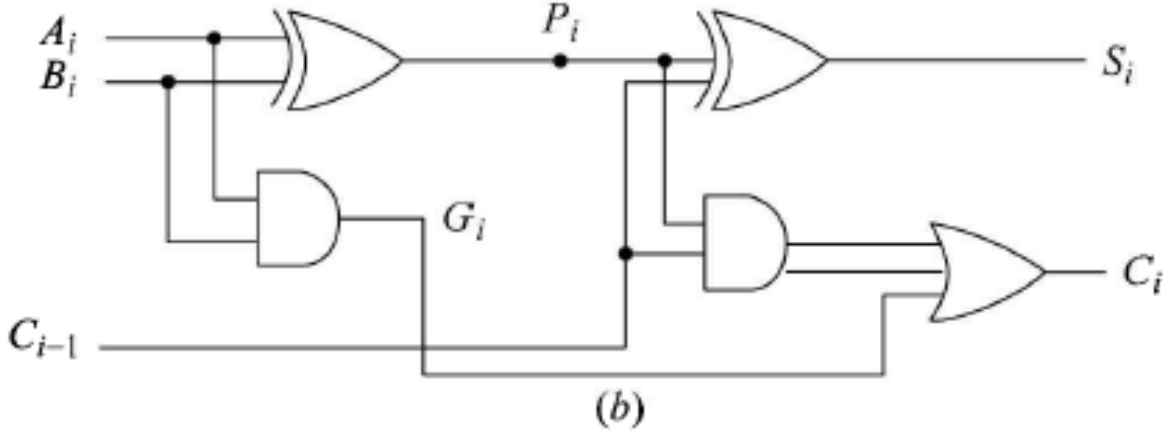


Fig. 6.13 **(a) *Block Diagram of ith Stage of a Full-Adder***
(b) EX-OR *Implementation of Full-Adder*

BCD ARITHMETIC

BCD Adder:

- **4-bit binary adder IC(7483) can be used to perform addition of BCD numbers.**
- **If 4-bit sum output is not a valid BCD digit, or if a carry C is generated, then decimal 6 (0110) is to be added to the sum to get the correct result.**
- **It can be cascaded to add numbers several digits long by connecting the carry-out of a stage to the carry-in of the next stage.**

BCD Subtractor:

- **9's complement of the subtrahend is added to the minuend.**
- **9's complement of a BCD number is given by nine minus that number.**
- **E.g. 9's complement of 7 = 9 – 7 = 2.**
- **i.e. in BCD code 9's complement of 0 1 1 1 is 0 0 1 0.**

BCD ARITHMETIC

BCD Addition:

- **In BCD addition we have to deal with 3 different situations. Assume two 4 bit BCD numbers A & B are added. Then the 3 cases to be considered are:**

Case 1: Sum is equal to or less than 9 and carry is 0.

E.g. **Addition of decimal nos. 2 and 6 in BCD gives result as 8.**
Thus we can say sum is in proper BCD form and requires no correction.

Case 2: Sum is greater than 9 and carry is 0.

E.g. **Addition of decimal nos. 7 & 6 in BCD gives result as 13. Since sum is greater than 9, it is a invalid BCD number, with carry 0.**
so to correct the sum, add decimal 6 or BCD 0110 to sum to get the correct BCD sum.

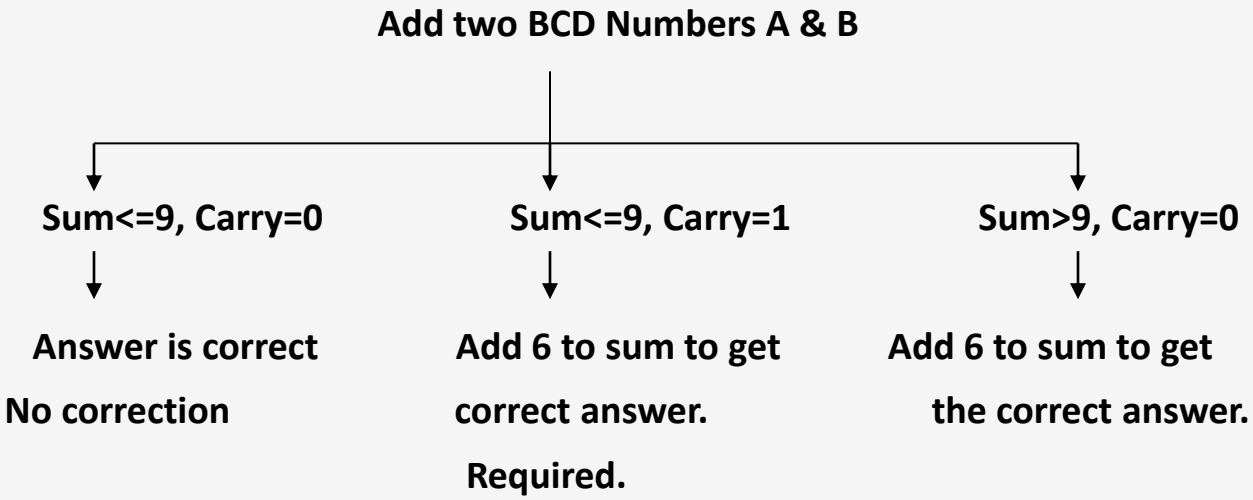
Case 3: Sum is less than or equal to 9 but carry is 1.

E.g. **Addition of decimal nos. 9 & 8 gives result as 17. We get sum as a valid BCD and carry =1, but the result is a incorrect BCD result. So add 6 to the sum obtained to get the correct BCD result.**

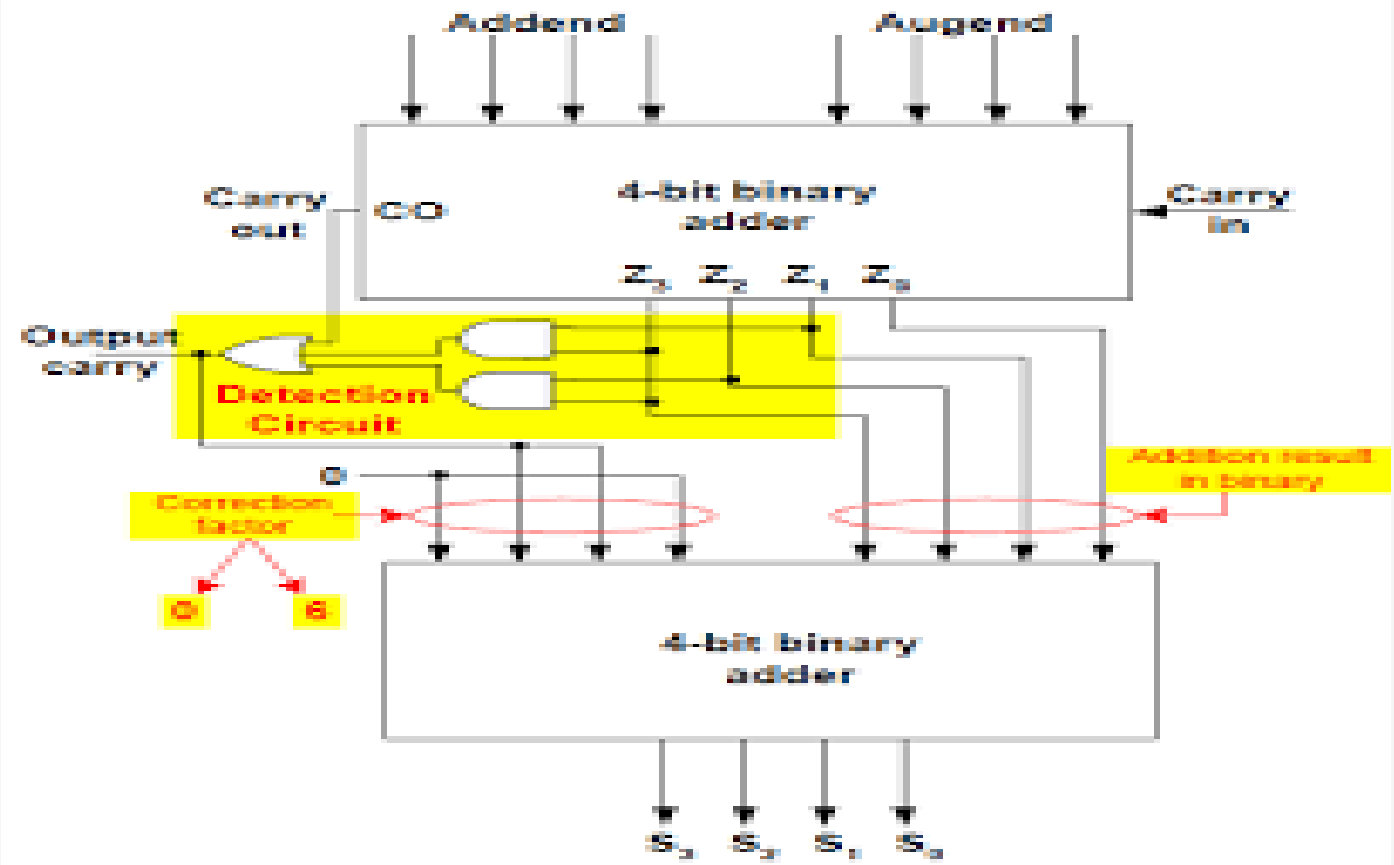
Note: The addition is to be carried out as normal binary addition.

BCD ARITHMETIC

Summary of BCD Addition:



BCD ADDER


Addend
Augend
Carry out
CO
4-bit binary adder
Carry in
Z_3 Z_2 Z_1 Z_0
Output carry
Detection Circuit
0
Correction factor
0
6
Addition result in binary
4-bit binary adder
S_3 S_2 S_1 S_0

BCD Adder

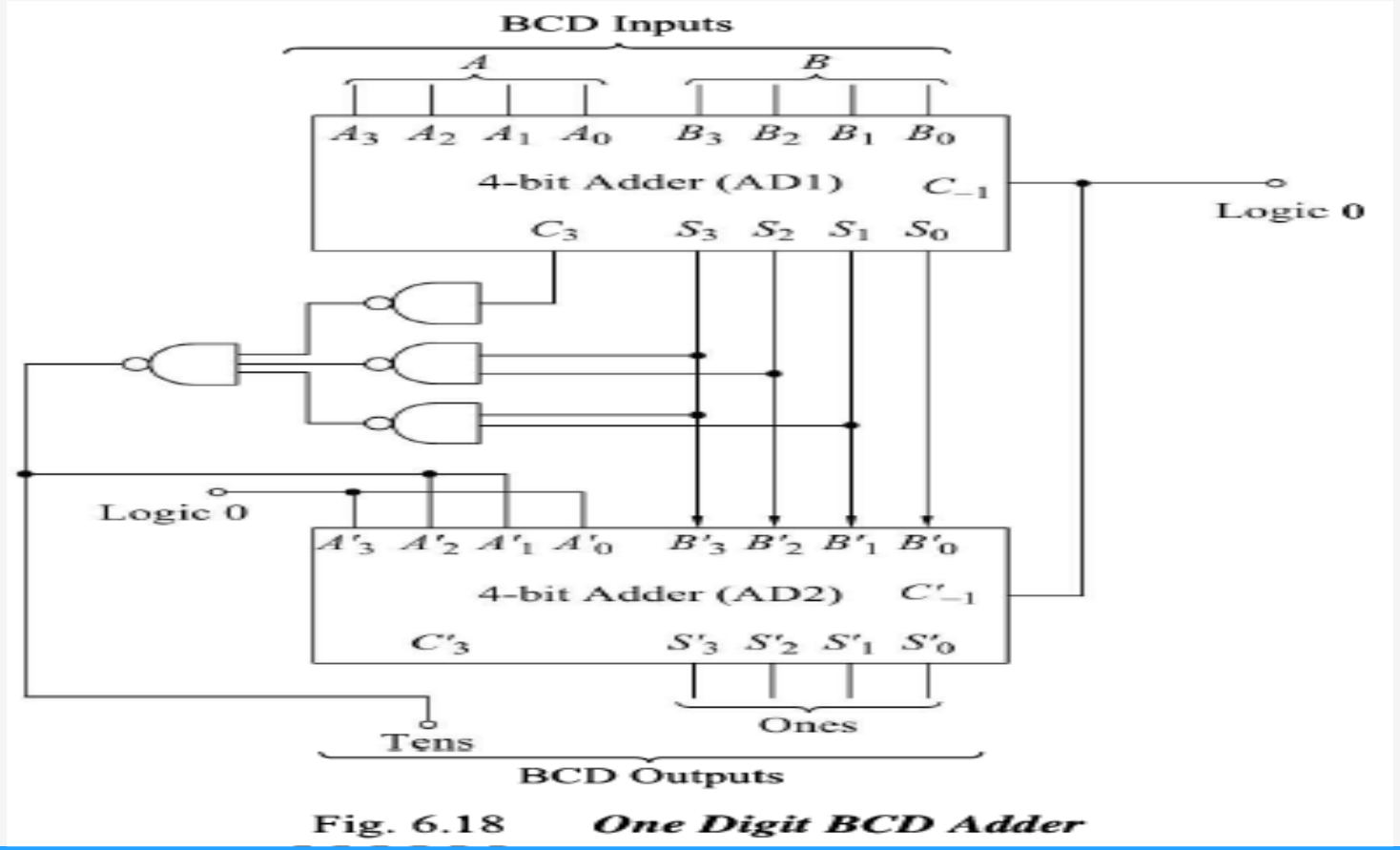
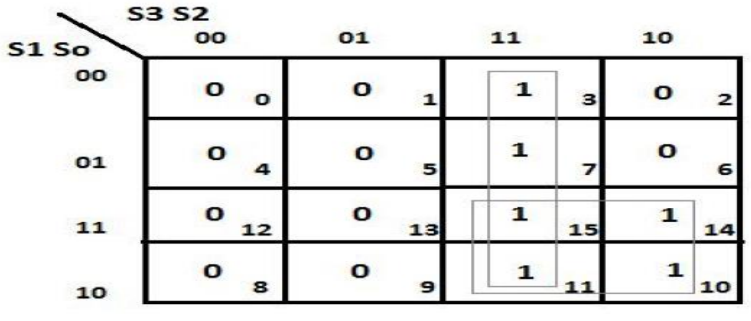


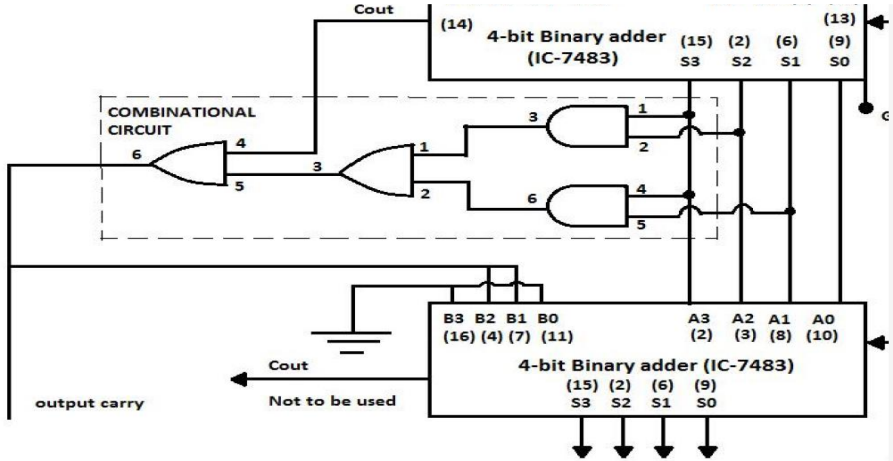
Fig. 6.18 ***One Digit BCD Adder***

BCD ADDER

Inputs				Output
S_3	S_2	S_1	S_0	Y
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
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

K-MAP:





Combinational Logic

- It involves simplification and realization using gates.
- It consists of an array of devices such as multiplexers, demultiplexers, adders, parity generators/checkers, priority encoders, decoders, comparators etc.
- Combinational logic is the general term for blocks of digital logic which contain no kind of memory
- The output of a block of combinational logic is determined solely by its inputs
- The relationship between the inputs and outputs is specified by the *truth table*
- For a logic block with *n* inputs, there are 2^n entries in the truth table
- Simple adders are an example of combinational logic
- So is the ALU, but this is much more complex.
- Registers are not combinational logic, since they contain memory.

Combinational logic example

- Consider a block of combinational logic with two inputs and four outputs
- Each of the four possible input combinations uniquely selects only one of the four outputs

I1	I2	Q1	Q2	Q3	Q4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

- This is known as a *decoder*
- Used commonly in memories to select a unique memory location based on an address
- No storage capability: when the input changes, the output changes after a short propagation delay

Multiplexer

- Passes one of several data inputs to output.
- Generally 2^n data inputs and always a single data output.
- n control lines determine which input is “steered” to the output.
- Allows logical (not “tri-state” or electrical) implementation of buses.
- Buses and register transfer operations fundamental to digital system design.
- Also possible to implement arbitrary combinational logic with multiplexers.
- Universal, combinational logic element.
- Also known as “Data Selector” and “Mux”.
- In sequential operation, provides parallel to serial Conversion.

Multiplexer Advantages

- Simplification of logic expression is not required.
- Minimizes the IC package count.
- Logic design is simplified.
- It can be used as a logic element in the design of combinational circuits.
- The standard ICs available are 2:1, 4:1, 8:1 and 16:1.

Design Procedure:

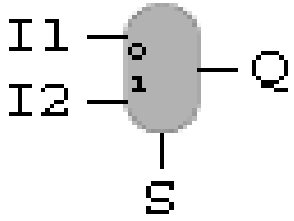
1. Identify the decimal number corresponding to each minterm in the expression, & connect the input lines corresponding to these numbers to logic 1 level.
2. Connect all other input lines to logic 0 level.
3. Apply inputs to select inputs.

Multiplexer Tree:

- To achieve larger input needs there is a provision for expansion, which is designed with the help of enable/strobe inputs and multiplexers stacks or trees.

Multiplexers

- Multiplexors are clearly something that we need to know how to build



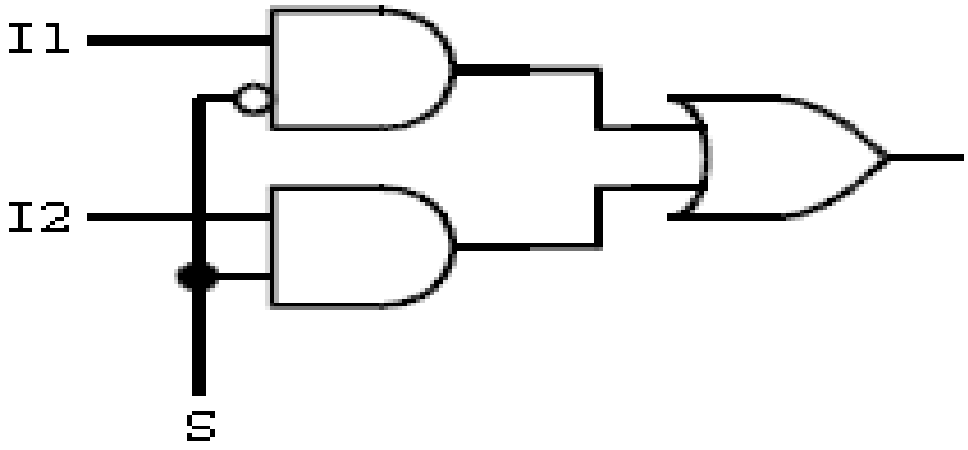
I1	I2	S	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

- It is possible to write a logic equation for Q in terms of I1, I2 and S:

$$Q = (I1 \cdot \overline{S}) + (I2 \cdot S)$$

- The logic circuit follows trivially

Multiplexers from logic



- Note the shorthand for NOT, this is very commonly used
- Obviously this is only a single-bit multiplexer, it should be obvious that to multiplex two 32-bit numbers, you need 32 of these controlled by the same "select" signal
- You should verify that this does what is intended

Two-to-One Multiplexer

F = Select' · x0 + Select · x1
x1 = w0
X2 = w1

$s\ x_1 x_2$	$f(s, x_1, x_2)$
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	1

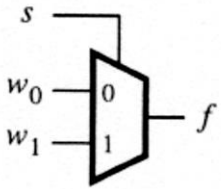
(a) Truth table

s	f
0	w_0
1	w_1

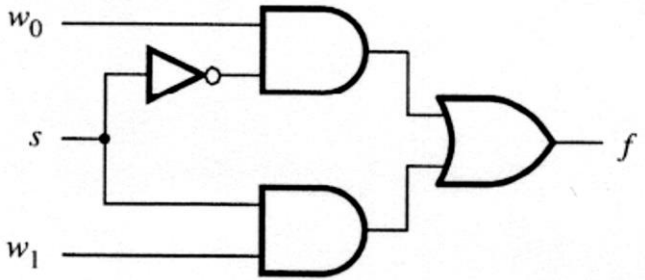
(b) Truth table

s	$f(s, x_1, x_2)$
0	x_1
1	x_2

(d) More compact truth-table representation



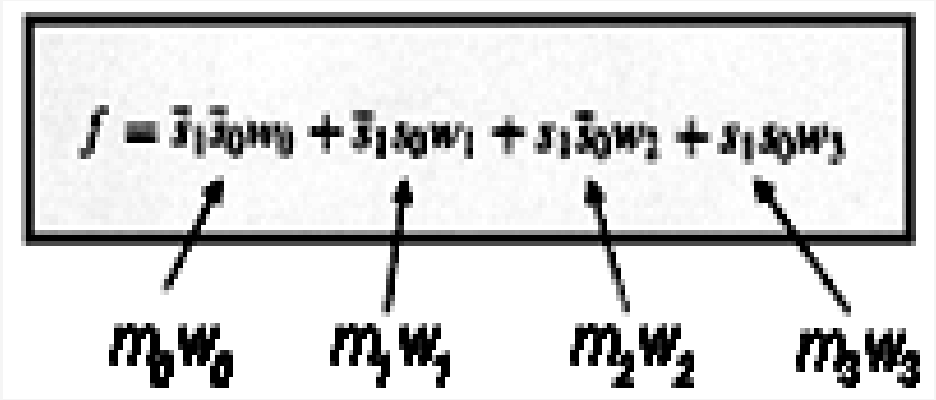
(a) Graphical symbol

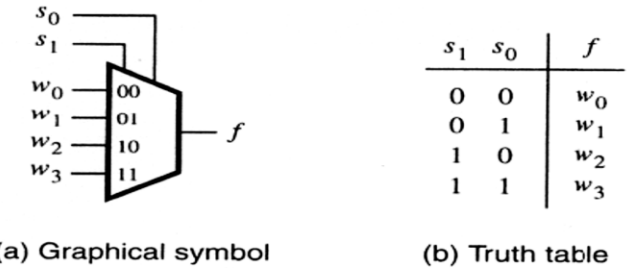


(c) Sum-of-products circuit

Four to One Multiplexer

- i^{th} data input ANDed with minterm m_i
- Embedded circuit generating minterms will become known as a decoder

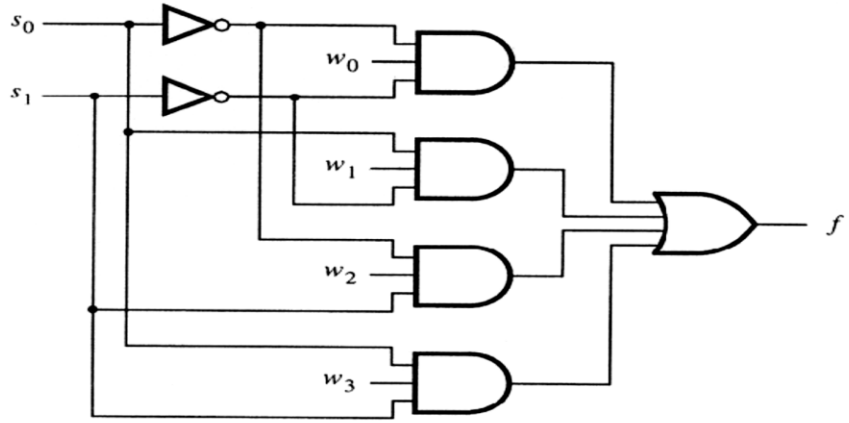




(a) Graphical symbol

s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

(b) Truth table

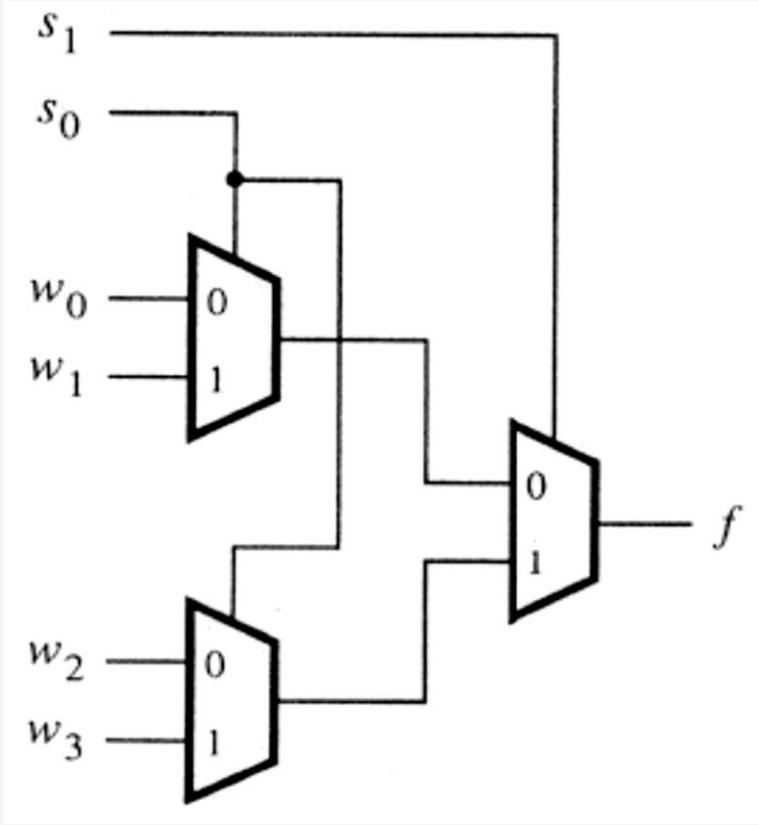


(c) Circuit

Figure 6.2 A 4-to-1 multiplexer.

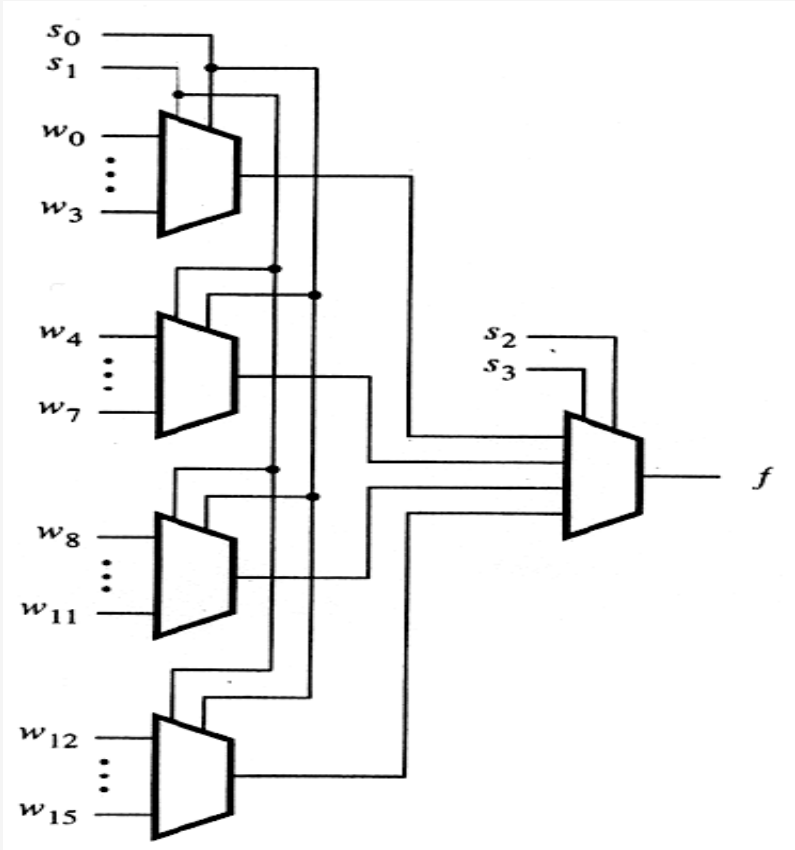
Building Larger Multiplexers

- 4-to-1 (4:1) Mux using 2-to-1 (2:1) Muxes
- Simple and modular
- Adds 2 levels of gate (propagation) delay



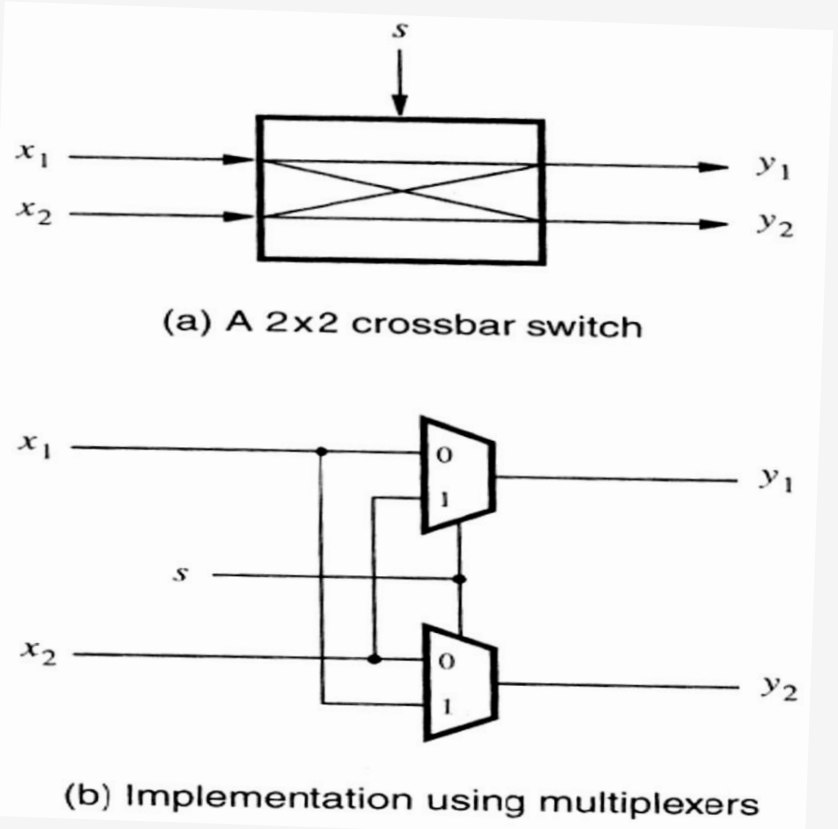
Building Larger Multiplexers

- **16:1 Mux constructed from 4:1 Muxes.**
- **Expandable to 32:1 and 64:1 with additional 2:1 and/or 4:1 Muxes.**
- **With additional levels of propagation delay.**



Multiplexer Application

- Crossbar Switch
- In general, n-inputs by n outputs
- Connectivity is any input to any output
- Important component of networking hardware
- The bigger, the faster, the better…

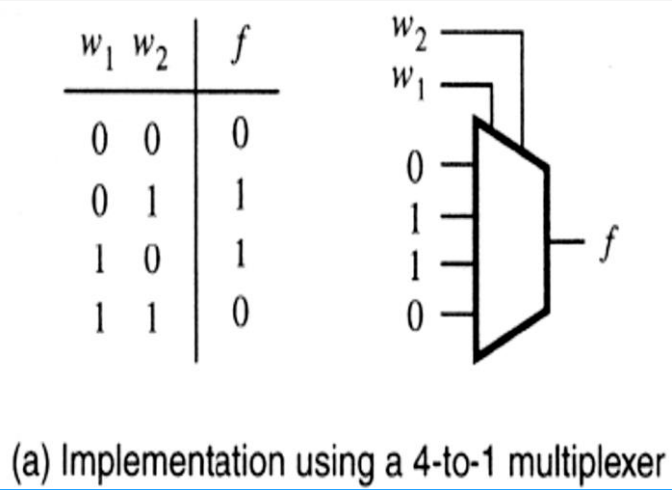


Combinational Design Using Multiplexers

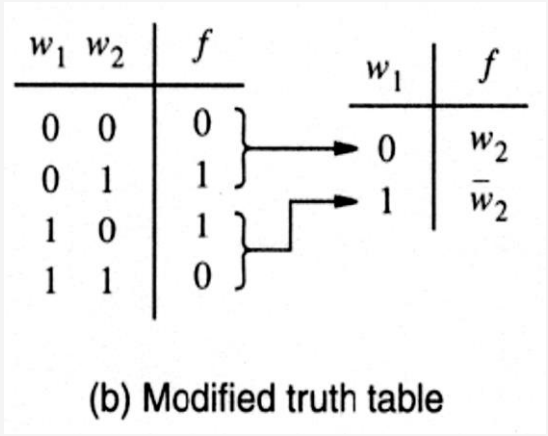
- Input variables applied to Mux select lines “Steer” (constant) value of function to output
- Allows implementation of n-variable function with 2^n-to-1 multiplexer
- “Steer” derived function (a variable, its complement, the constant 1 or the constant 0) to the output
- Allows implementation of n-variable function with 2^{n-1}-to-1 multiplexer

Combinational Design Using Multiplexers

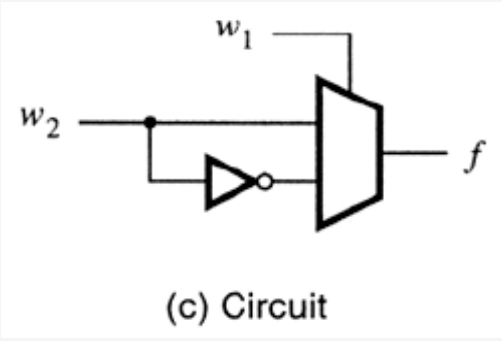
- **Example 1: XOR Function Using a 4:1 Mux**
- **The modified Truth Table**
  - **Possibilities are x, x', 0, 1**
- **The 2-input XOR using a 2:1 Mux**



(a) Implementation using a 4-to-1 multiplexer



(b) Modified truth table



(c) Circuit

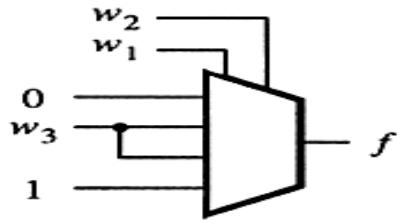
Combinational Design Using Multiplexers

- **Example 2 : Three input majority function**
- **Three input function with (2^{n-1}-to-1)**
- **4:1 Mux**

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	w_2	f
0	0	0
0	1	w_3
1	0	w_3
1	1	1

(a) Modified truth table



(b) Circuit

Combinational Design Using Multiplexers

- **Example 3 : Three input majority function with 2:1 Mux**
  - Algebraic expansion

$$f(w_1, w_2, w_3) = (w_1w_2 + w_1w_3 + w_2w_3)$$

$$f(w_1, w_2, w_3) = (w_1w_2 + w_1w_3 + w_2w_3)(w_1' + w_1)$$

$$f = w_1'(w_1w_2 + w_1w_3 + w_2w_3) + w_1(w_1w_2 + w_1w_3 + w_2w_3)$$

... and from Shannon

$$f = w_1'(0w_2 + 0w_3 + w_2w_3) + w_1(1w_2 + 1w_3 + w_2w_3)$$

$$f = w_1'(w_2w_3) + w_1(w_2 + w_3)$$

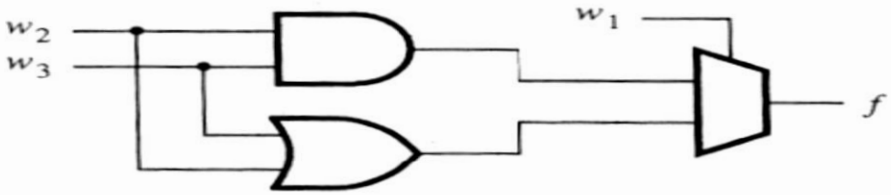
Combinational Design Using Multiplexers

- **Example 3: Three input majority function with 2:1 Mux**
  - Truth Table and circuit implementation

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	f
0	w_2w_3
1	$w_2 + w_3$

(a) Truth table



(b) Circuit

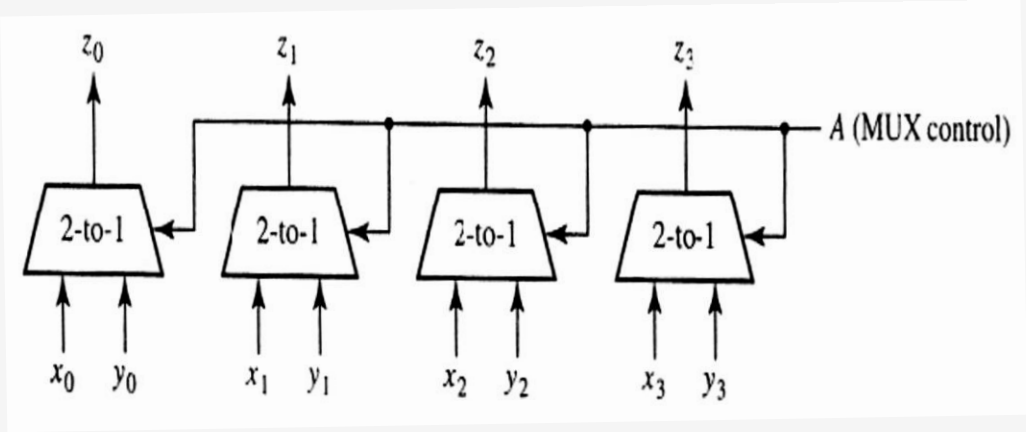
Multiplexers and Buses

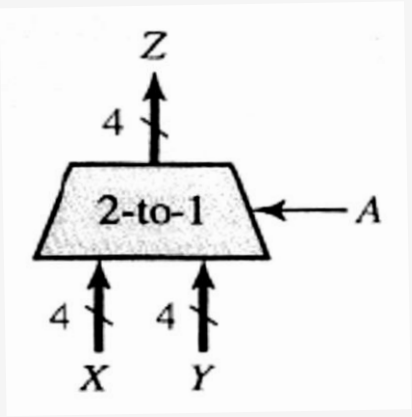
- Bus allows data transfers between multiple sources and single or multiple destinations over a shared path (wires).
- Bus includes multiple bits.
- Parallel data bus.
- Only one source on the bus at any time.
- Bus contention.

Multiplexers and Buses

- **Example below illustrates two, four-bit words (X and Y) multiplexed onto the Z bus.**
- **Register transfer operations.**

  A' : Z ← X , A : Z ← Y





Multiplexer

Example 6.1

Implement the expression using a multiplexer.

$$f(A, B, C, D) = \Sigma m(0, 2, 3, 6, 8, 9, 12, 14)$$

Solution

Since there are four variables, therefore, a multiplexer with four select inputs is required. The circuit of 16:1 multiplexer connected to implement the above expression is shown in Fig. 6.3. This implementation requires only one IC package. In case the output of the multiplexer is active-low, the logic 0 and logic 1 inputs of Fig. 6.3 are to be interchanged. The reader should verify the validity of this statement.

Multiplexer

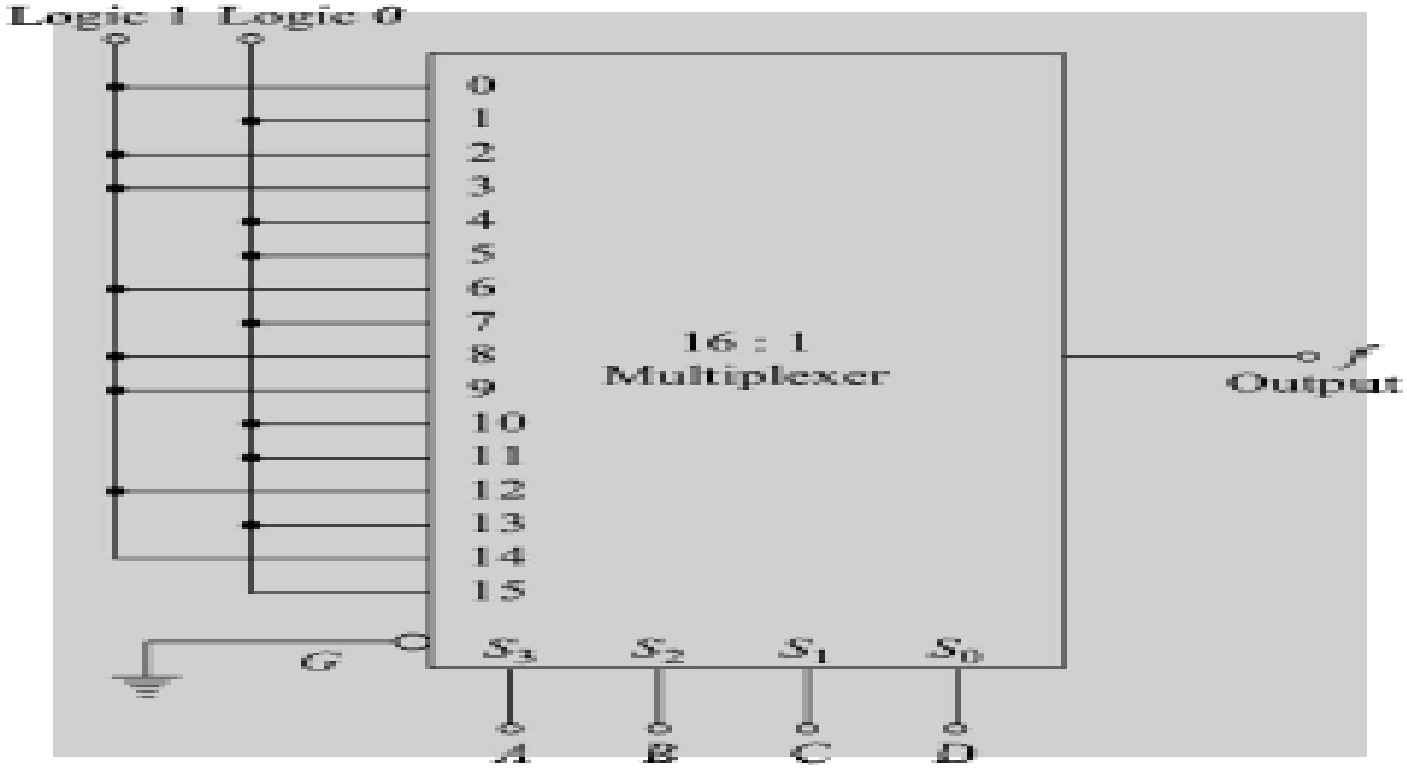


Fig.6.3 *Implementation of Logic Expression of Ex. 6.1*

Multiplexer

Example 6.2

Realise the logic function of the truth table given in Table 6.3.

Table 6.3 *Truth table of Example 6.2*

Inputs				Output
A	*B*	*C*	*D*	*Y*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Multiplexer

Solution

(i) *First Method*: This can be realised using the method used in Example 6.1. Here, the input lines 2, 4, 6, 7, 9, 10, 11, 12, and 15 are to be connected to logic 1 and the input lines 0, 1, 3, 5, 8, 13, and 14 are to be connected to logic 0.

(ii) *Second Method*: A four variable truth table or logic expression can be realised by using an 8:1 multiplexer instead of a 16:1 multiplexer. For this, partition the truth table as shown by dotted lines. Here the inputs A, B, and C are to be connected to S_2, S_1 and S_0 select inputs respectively. Now, we observe the relationship between input D and output Y for each group of two rows. There are four possible values of Y and these are 0, 1, D, and $\bar{D}$. These are given in Table 6.4. From this table, we note the output Y for each of the combinations of A, B, and C, and then make the connections accordingly. The implementation of this function using an 8:1 multiplexer is shown in Fig. 6.4.

Multiplexer

Inputs			Output
A	B	C	Y
0	0	0	0
0	0	1	$\bar{D}$
0	1	0	$\bar{D}$
0	1	1	1
1	0	0	D
1	0	1	1
1	1	0	$\bar{D}$
1	1	1	D

Multiplexer

The second method can also be used if the logic expression is specified.

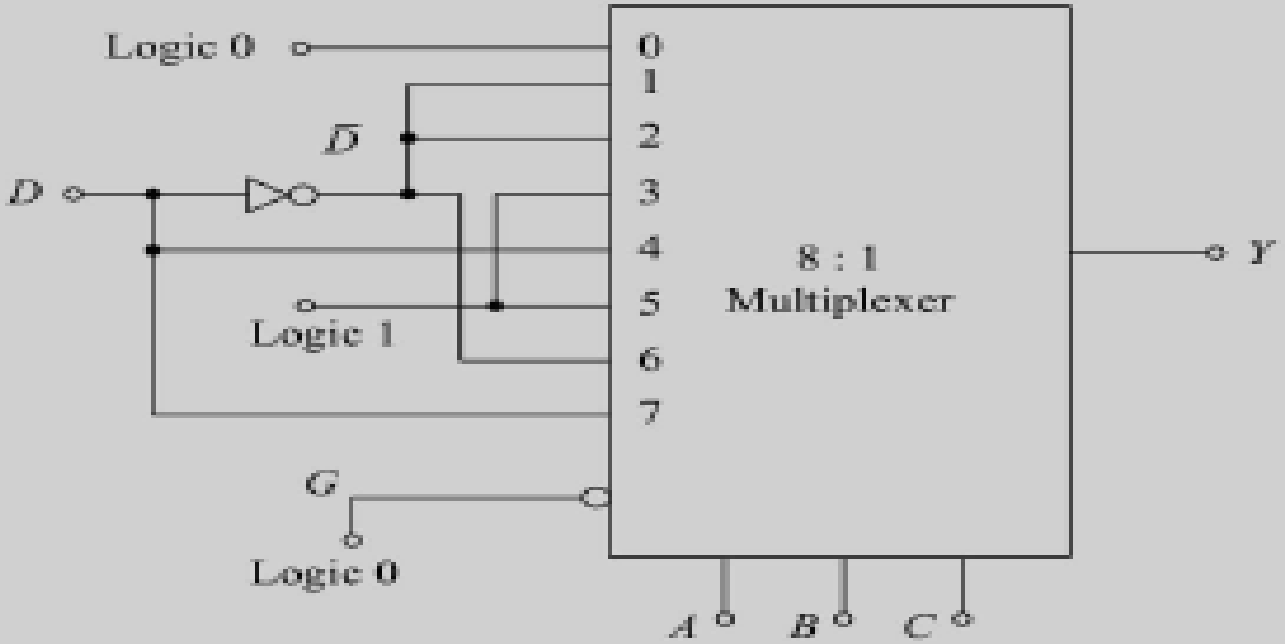


Fig. 6.4 *Realisation of 4-variable Truth Table Using 8:1 Multiplexer*

Multiplexer Tree

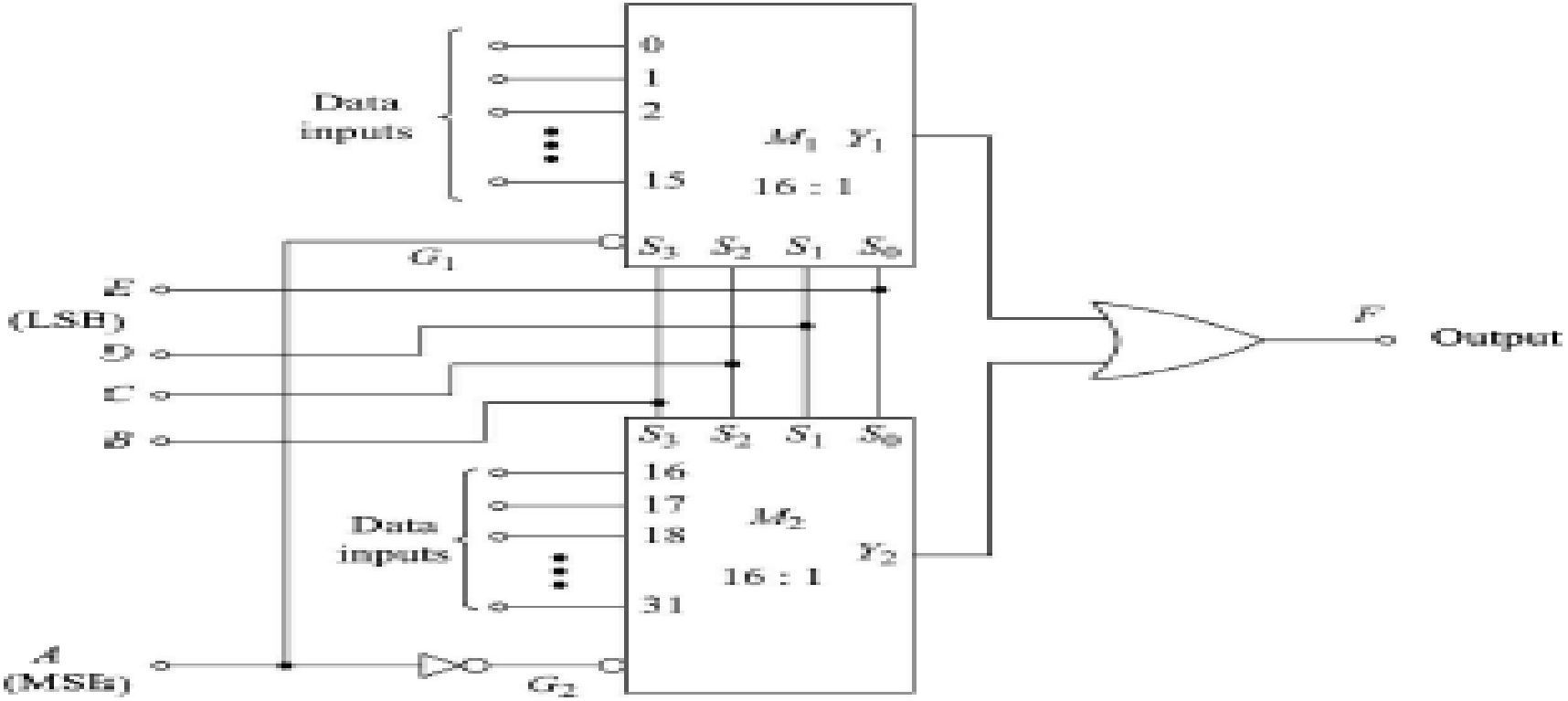


Fig. 6.5 *32:1 Multiplexer Using Two 16:1 Multiplexers and One* OR *Gate*

Multiplexer Tree

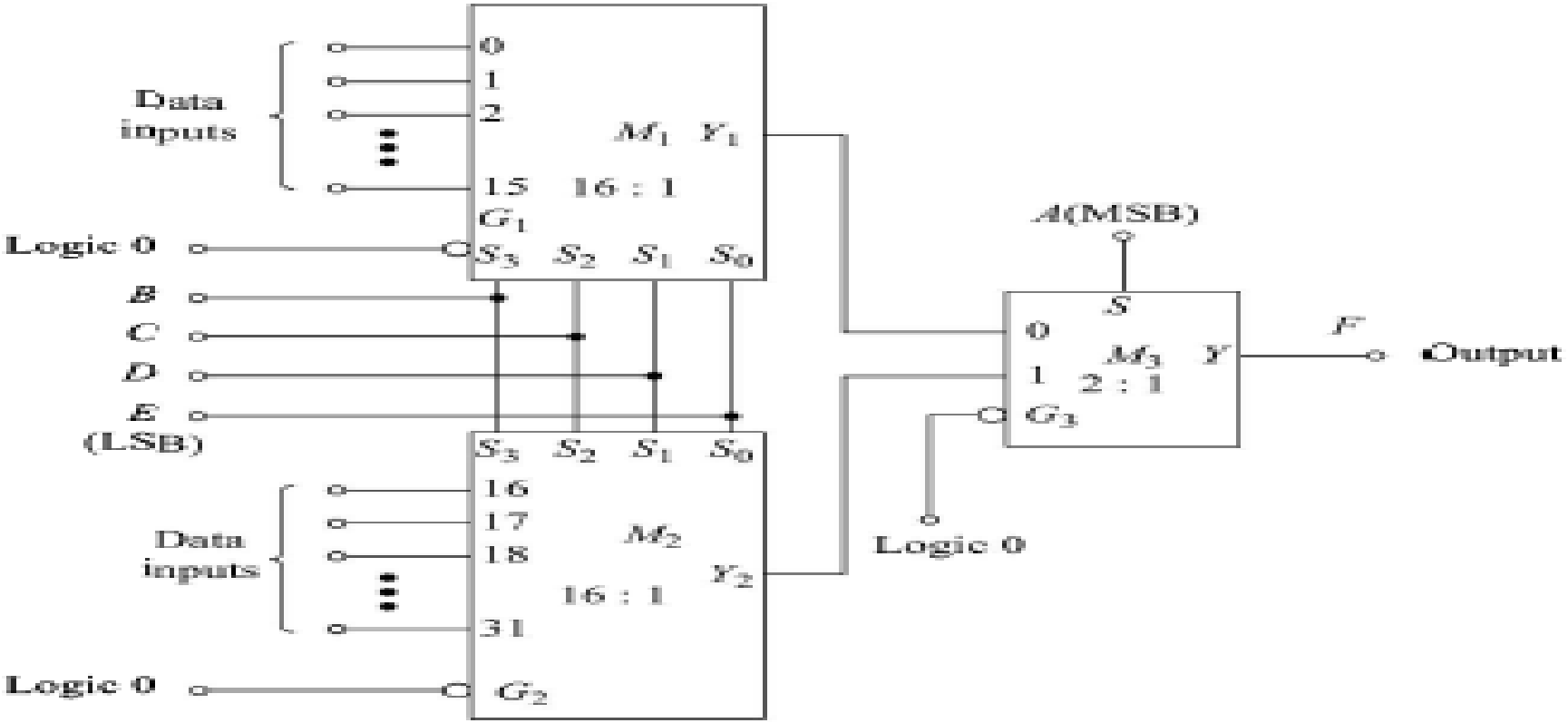


Fig. 6.6 *32:1 Multiplexer Using Two 16:1 Multiplexers and One 2:1 Multiplexer*

Demultiplexer

- Performs the reverse of a multiplexer.
- Accepts a single input and distributes it over several outputs.
- Select input code determines to which output the data input will be transmitted.
- It can be used as binary to decimal decoder with binary inputs applied at the select input lines and the output will be obtained on the corresponding line.
- It is useful in the design of multiple-output combinational circuit, because it needs minimum package count.
- It is available as 2-line-to-4-line, 3-line-to-8-line, and 4-line-to-16-line decoders.
- The output of most of these devices are active low, and it consists of a active-low enable/data input terminal.
- Decoder requires some gates in order to realize a boolean expression in the standard SOP form.

Demultiplexers

- Serial to parallel conversion
  - Send a single data bit to a specific address
- A 1-to-2^n demultiplexer is implemented using an n-to-2^n decoder
  - The (value of) the data is applied via the enable input
- Valuable circuit in sequential circuits
  - Not so much in combinational circuits
- Also referred to as Dmux's

Demultiplexer Tree

- Largest available decoders are 4-line-to-16-line decoders
- So to design a circuit for larger inputs, we make use of enable input terminals.
- E.g. 5-line-to-32-line decoder using two 4-line-to-16-line decoders

Demultiplexer Tree

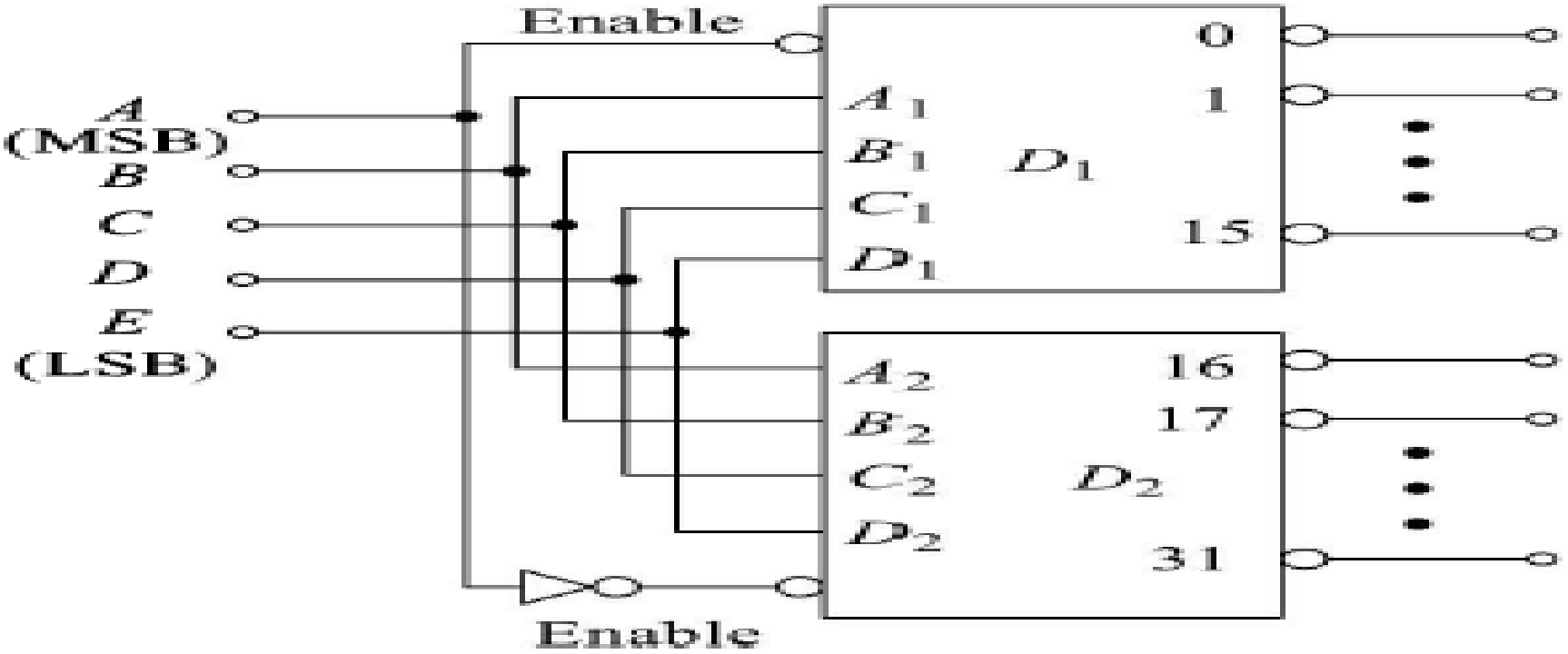


Fig. 6.9 *5-Line-to-32-Line Decoder Using Two 4-line-to-16-Line Decoders*

Demultiplexer

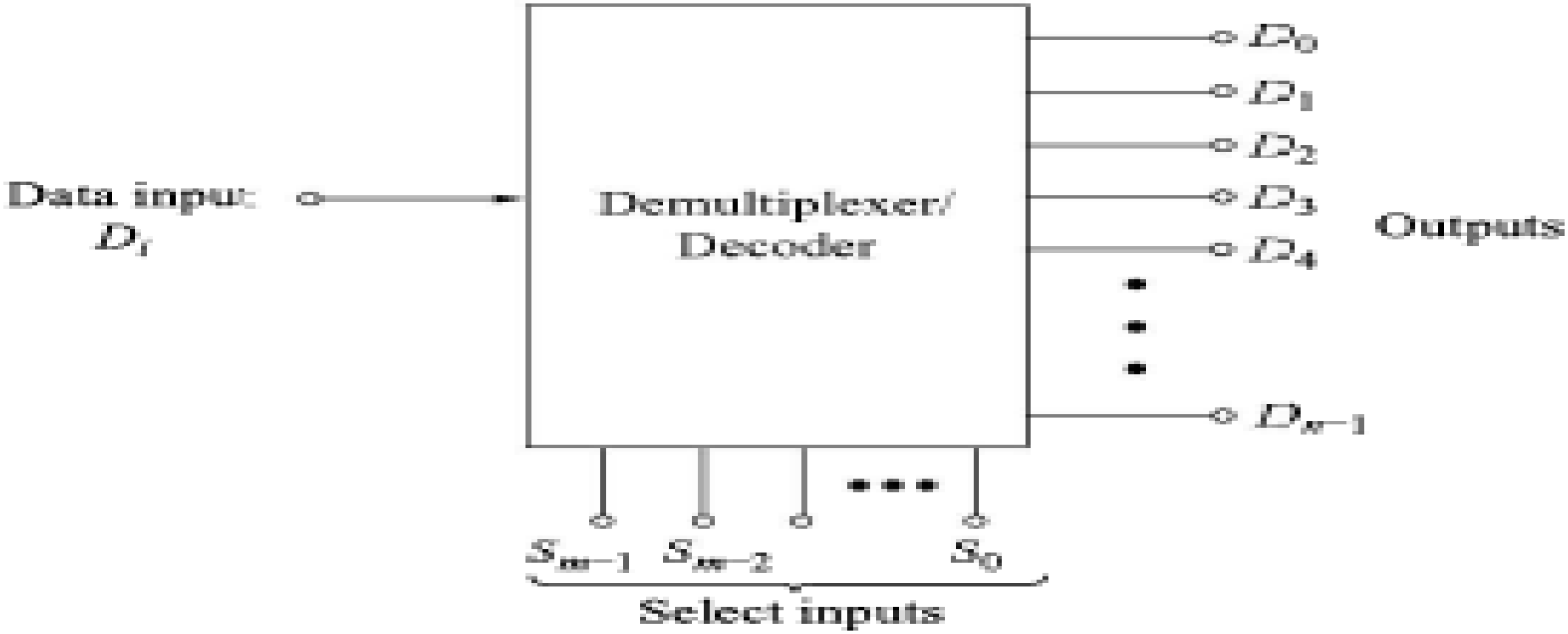
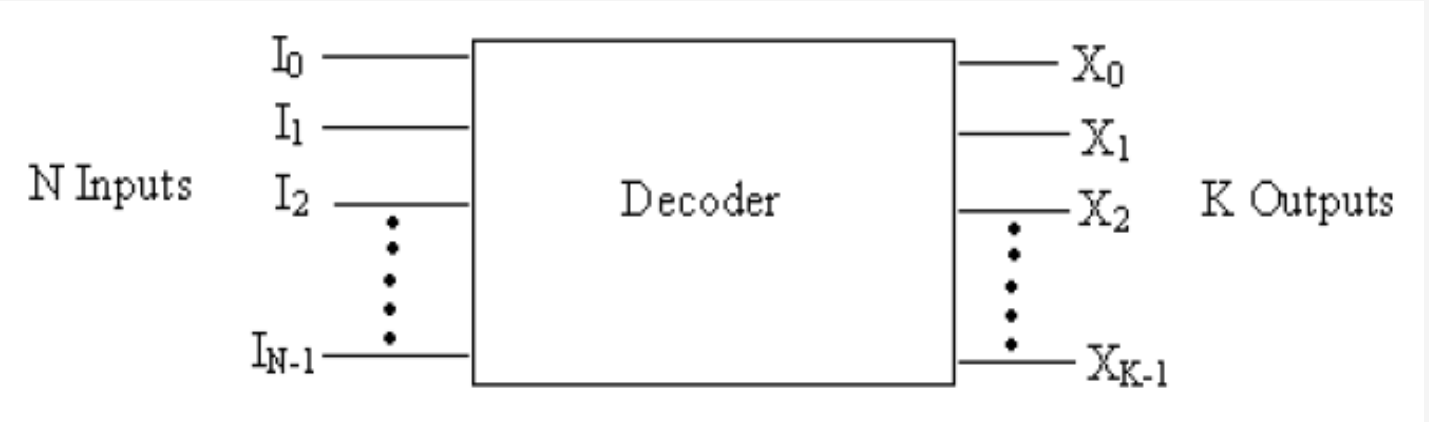


Fig. 6.7 ***Block Diagram of a Demultiplexer.***

Decoder

- **It** has a set of inputs representing a binary number and gives only one output corresponding to the input number.
- It activates one output at a time depending upon the input binary number, all other outputs will be inactive.
- Figure shows the functional block diagram of a decoder having *N* inputs and K outputs. The possible combinations of *N* inputs will be $2^N = K$, so there will be K outputs.



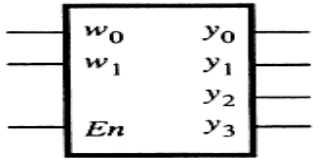
Decoder

Decoders

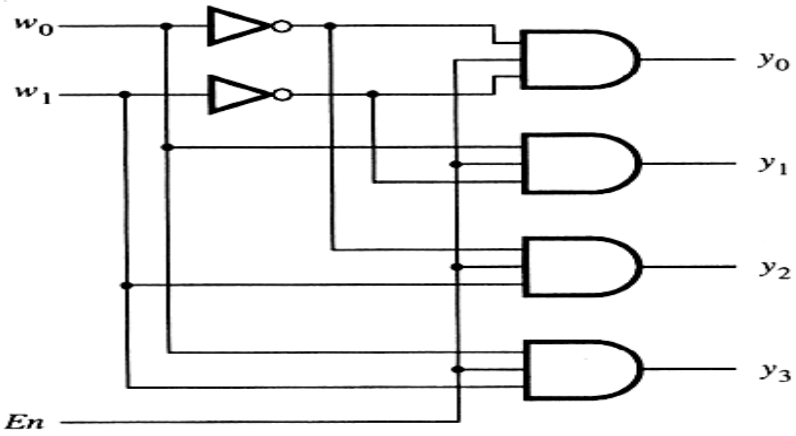
- 2-to-4 Decoder shown
  - 2-to-2^n in general
  - Enable input allows construction of decoder tree and demultiplexer
  - Generates all minterms when enabled
    - Multiple output circuits
  - One hot decoding

En	w_1	w_0	y_0	y_1	y_2	y_3
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
0	x	x	0	0	0	0

(a) Truth table



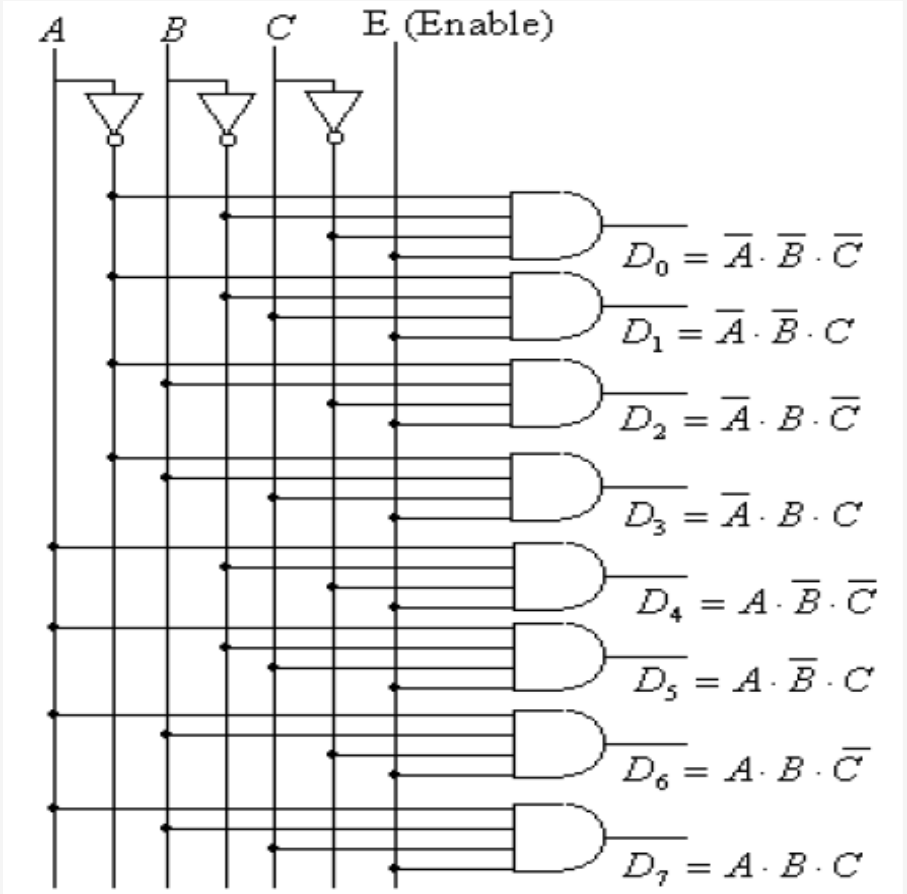
(b) Graphical symbol



(c) Logic circuit

3 to 8 line decoder

- It will have three input lines and $2^3 = 8$ output lines.
- When 3 bit binary number is fed to the input of the decoder, one output line corresponding to input binary is activated & all other output lines will be inactive.
- It is also called binary to octal decoder or converter.



Decoder

Implement the following multi-output combinational logic circuit using a 4-to-16-line decoder.

$$F_1 = \Sigma m\,(1, 2, 4, 7, 8, 11, 12, 13)$$
$$F_2 = \Sigma m\,(2, 3, 9, 11)$$
$$F_3 = \Sigma m\,(10, 12, 13, 14)$$
$$F_4 = \Sigma m\,(2, 4, 8)$$

Decoder

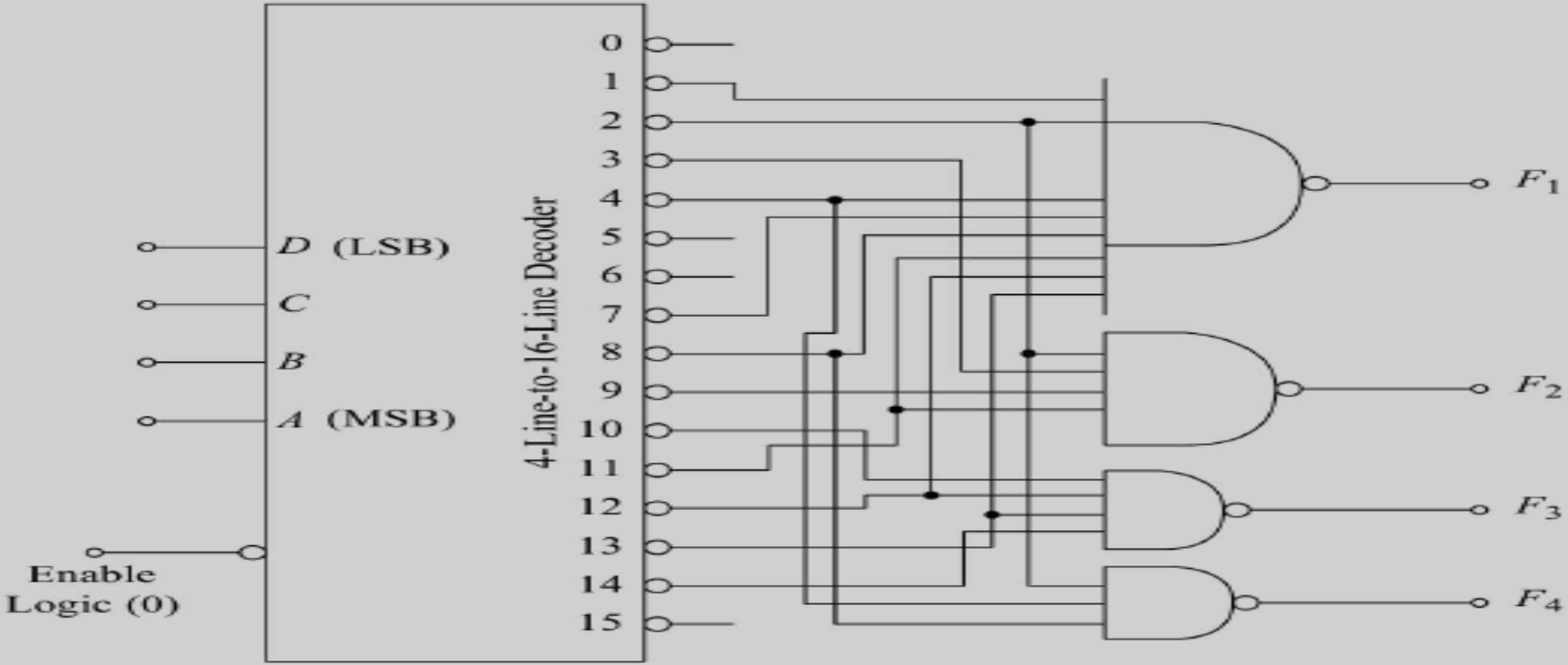
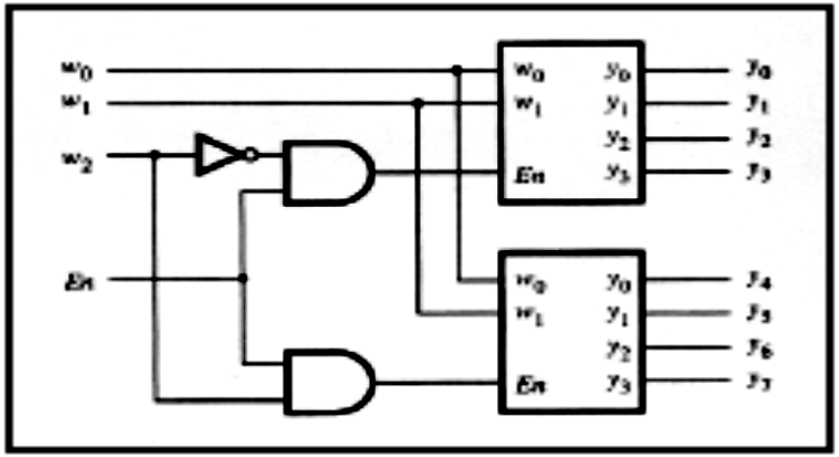


Fig. 6.8 *Implementation of Combinational Logic Circuit of Ex. 6.3*

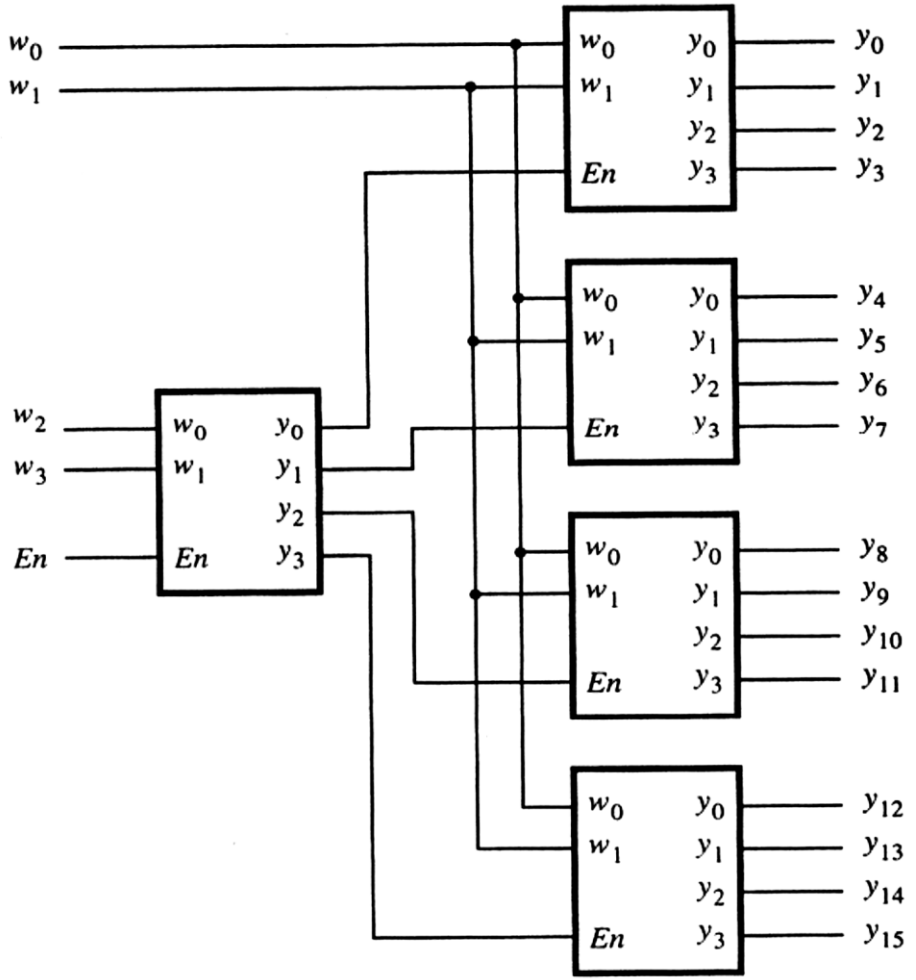
Decoder Tree

- One-bit expansion (3-to-8) by adding external decoding circuitry



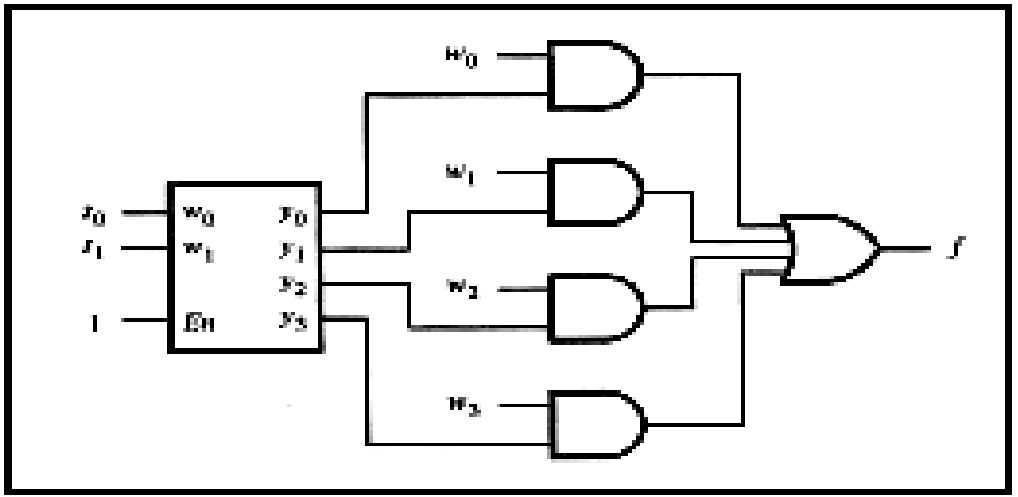
Decoder Tree

- Two-bit expansion (4-to-16) by adding another 2-to-4 decoder



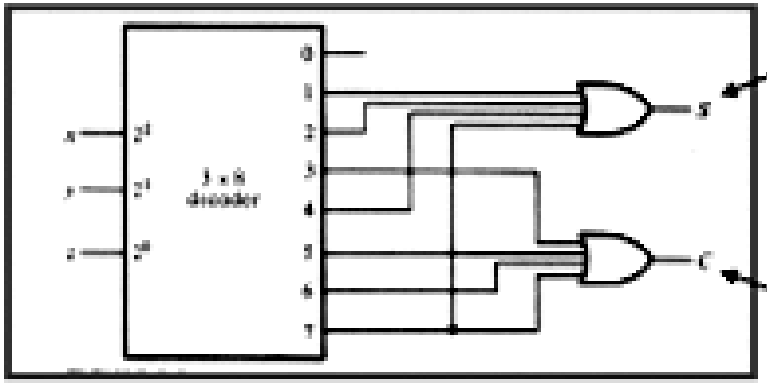
Decoder Applications

- **Multiplexer from decoder**
  - Recall "embedded decoder"



Decoder Applications

- **Multiple Output Circuits**
  - Full Adder using 3 X 8 Decoder

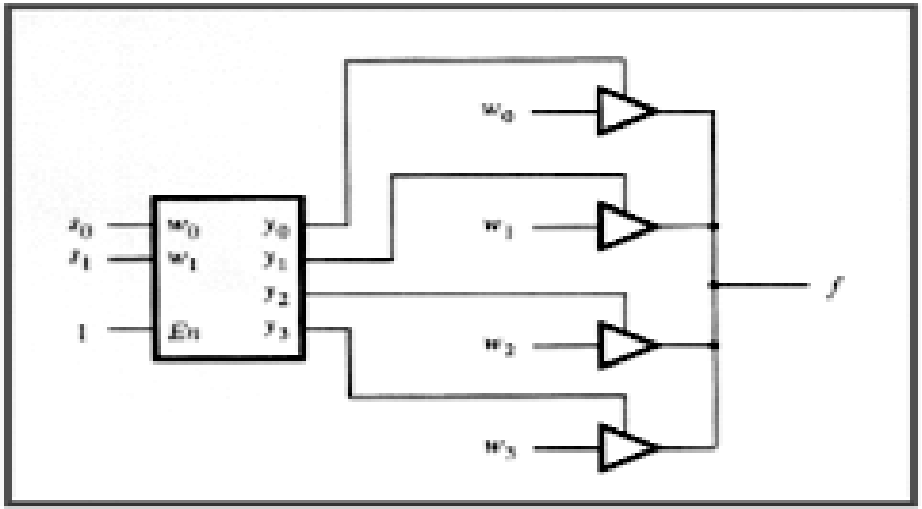


$Sum = m1 + m2 + m4 + m7$
$= xy'z + xyz' + xy'z' + xyz$

$Carry = m3 + m5 + m6 + m7$
$= xyz + xy'z + xyz' + xyz$

Decoder Applications

- Decoder Bus Control/Multiplexer



Combinational Logic Design

Decoder:

- It converts n-bit binary information at its input into a maximum of 2^n output lines.
- A decoder with enable input can function as a demux.

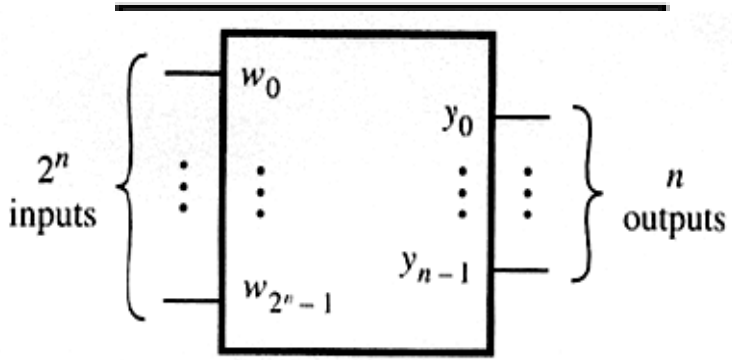
Inputs		Outputs			
A	B	D0	D1	D2	D3
0	0	1			
0	1		1		
1	0			1	
1	1				1

Encoder:

- It is just reverse of decoders.
- It has 2^n input lines and n output lines. The output lines generate the binary code corresponding to the input value.
- E.g. Decimal to BCD Encoder. IC available is (74147) decimal to BCD Priority Encoder.

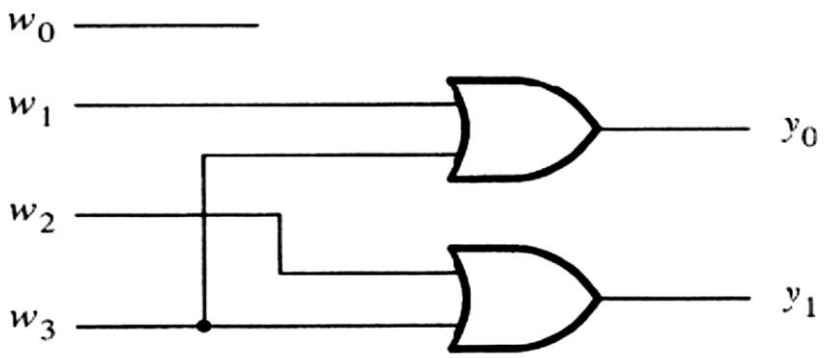
Encoders

- **Binary Encoders**
  - "One hot" input, binary (or other code) representation output
    - Reverse of decoder



w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

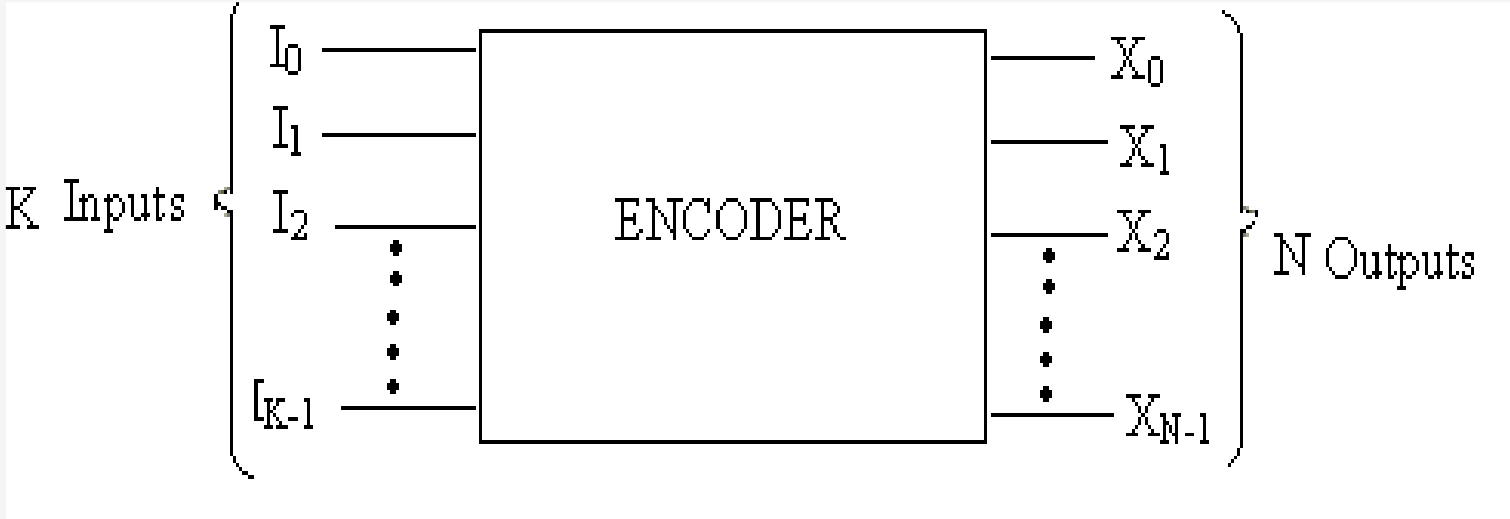
(a) Truth table



(b) Circuit

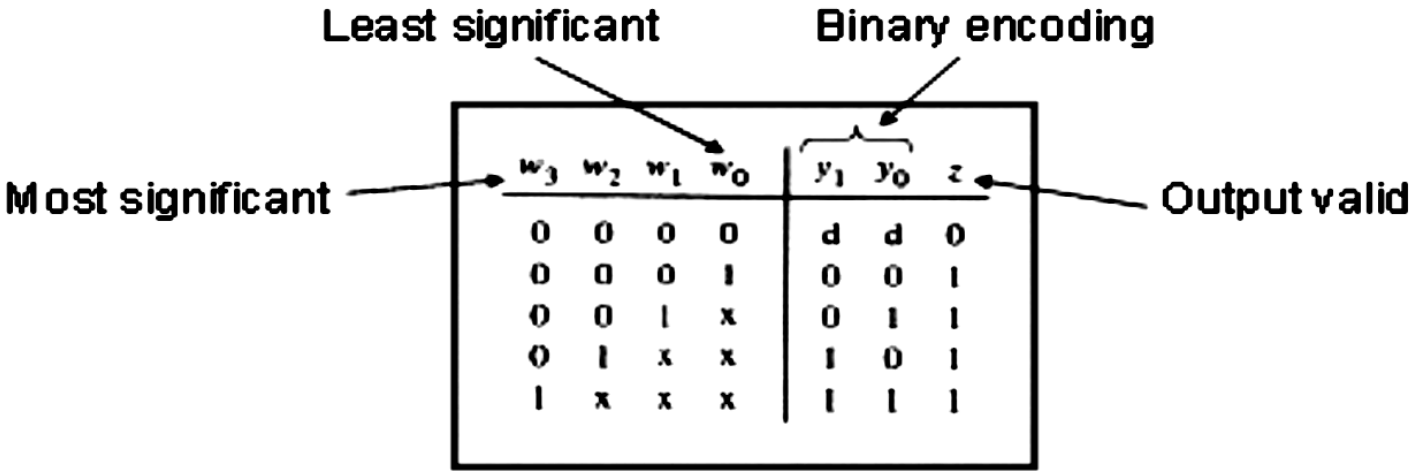
Encoders

- It is a combinational circuit which performs the reverse operation of decoder.
- It has a number of input lines, only one of which is activated at a time. It provides the N bit code at the output corresponding to activated input line.
- Figure shows the functional block diagram of an encoder having K inputs and N outputs.
- In the K input lines only one line will be high at a time.



Encoders

- **Priority Encoders**
  - Used in prioritizing interrupts (or other events)

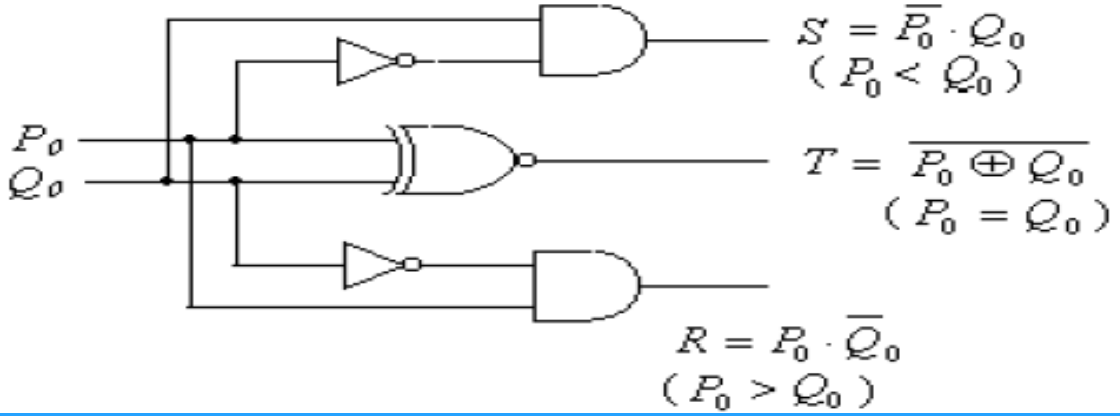


Priority Encoder

- In logic circuit for encoders, only one of the inputs is kept high at a time and output is obtained corresponding to the high input.
- But if two or more inputs are inadvertently activated at a time then undesirable results will be obtained.
- Priority encoder performs the same logic function as that of encoder with the additional facility of priority function.
- When two or more input lines are activated simultaneously. The priority function means that the encoder will provide the output corresponding to the highest order activated input line.

Magnitude Comparator

- It is also called as the magnitude digital (or binary) comparator.
- It compares and indicates if the binary number P is equal to or greater than or less than the other binary number Q.
- Let P0 and Q0 are the two bits to be compared. The result for the equality of these two bits may be given by XNOR gate.
- The XNOR gate gives an output as logic 1 if two bits are equal otherwise logic 0.
- The logic diagram for one bit comparator is shown in figure :



1-Bit Comparator

- It is used to compare two single bits and hence called a single bit comparator.
- It consists of two inputs each for two single bit numbers and three outputs to generate less than, equal to and greater than between two binary numbers.
- The truth table for a 1-bit comparator is given below:

A	B	A<B	A=B	A>B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

- From the above truth table logical expressions for each output can be expressed as follows:
- A>B: AB'
- A<B: A'B
- A=B: A'B' + AB

1-Bit Comparator

- From the above expressions we can derive the following formula:

(A<B)+(A>B) = A'B+AB'

Taking complement both sides

((A<B) + (A>B))' = (A'B + AB')'

((A<B) + (A>B))' = (A'B)' (AB')'

((A<B) + (A>B))' = (A + B') (A' +B)

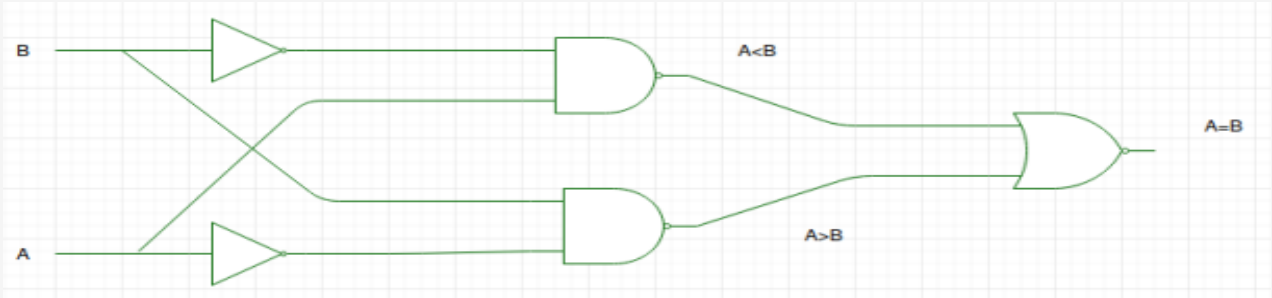
((A<B) + (A>B))' =(AA' + AB + A'B' +BB')

" " = (AB + A'B')

Thus,

((A<B) + (A > B))' = (A = B)

By using these Boolean expressions, we can implement a logic circuit for this comparator as given below:



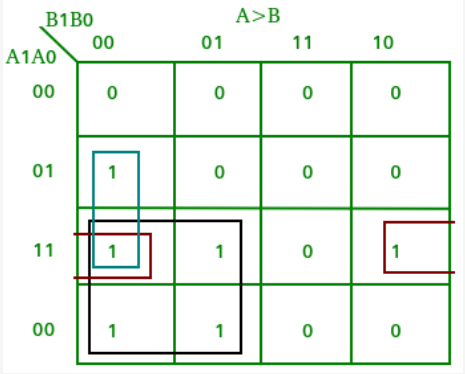
2-Bit Comparator

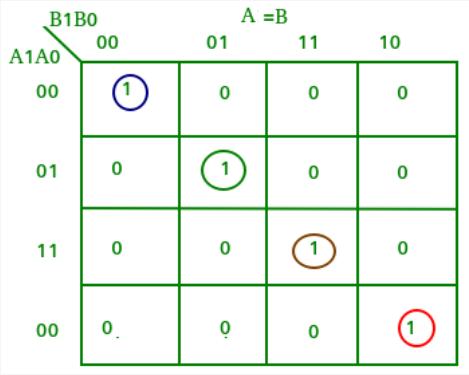
- It is used to compare two binary numbers each of two bits and hence called a 2-bit Magnitude comparator.
- It consists of four inputs and three outputs to generate less than, equal to and greater than between two binary numbers.
- The truth table for a 2-bit comparator is given below:

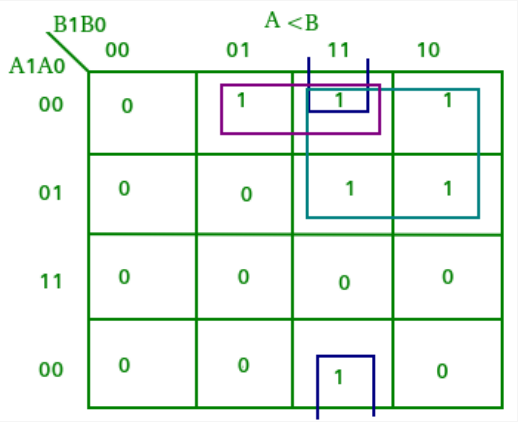
INPUT				OUTPUT		
A1	A0	B1	B0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

2-Bit Comparator

➢ From the above truth table K-map for each output can be drawn as follows:







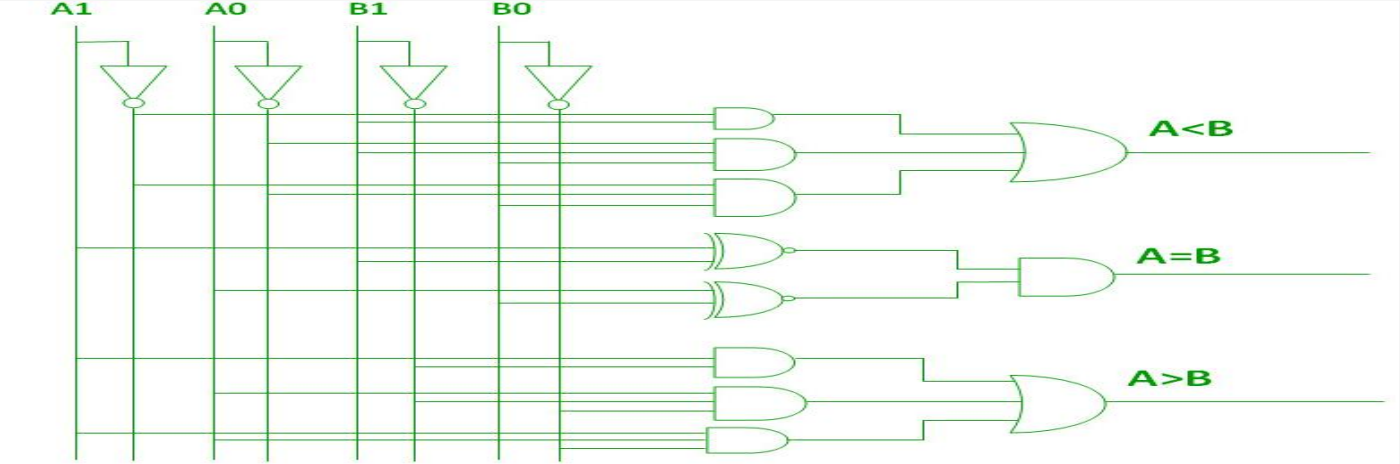
2-Bit Comparator

- From the above K-maps logical expressions for each output can be expressed as follows:
- **A>B : A1B1' + A0B1'B0' + A1A0B0'**
- **A=B : A1'A0'B1'B0' + A1'A0B1'B0 + A1A0B1B0 + A1A0'B1B0'**

  : A1'B1' (A0'B0' + A0B0) + A1B1 (A0B0 + A0'B0')

  : (A0B0 + A0'B0') (A1B1 + A1'B1')

  : (A0 Ex-Nor B0) (A1 Ex-Nor B1)
- **A<B : A1'B1 + A0'B1B0 + A1'A0'B0**
- By using these Boolean expressions, we can implement a logic circuit for this comparator as given below:



4-Bit Comparator

- It is used to compare two binary numbers each of four bits & hence called a 4-bit magnitude comparator.
- It consists of eight inputs each for two four bit numbers and three outputs to generate less than, equal to and greater than between two binary numbers.

In a 4-bit comparator the condition of A>B can be possible in the following four cases:

- If A3 = 1 and B3 = 0
- If A3 = B3 and A2 = 1 and B2 = 0
- If A3 = B3, A2 = B2 and A1 = 1 and B1 = 0
- If A3 = B3, A2 = B2, A1 = B1 and A0 = 1 and B0 = 0

Similarly the condition for A<B can be possible in the following four cases:

- If A3 = 0 and B3 = 1
- If A3 = B3 and A2 = 0 and B2 = 1
- If A3 = B3, A2 = B2 and A1 = 0 and B1 = 1
- If A3 = B3, A2 = B2, A1 = B1 and A0 = 0 and B0 = 1

The condition of A=B is possible only when all the individual bits of one number exactly coincide with corresponding bits of another number.

4-Bit Comparator

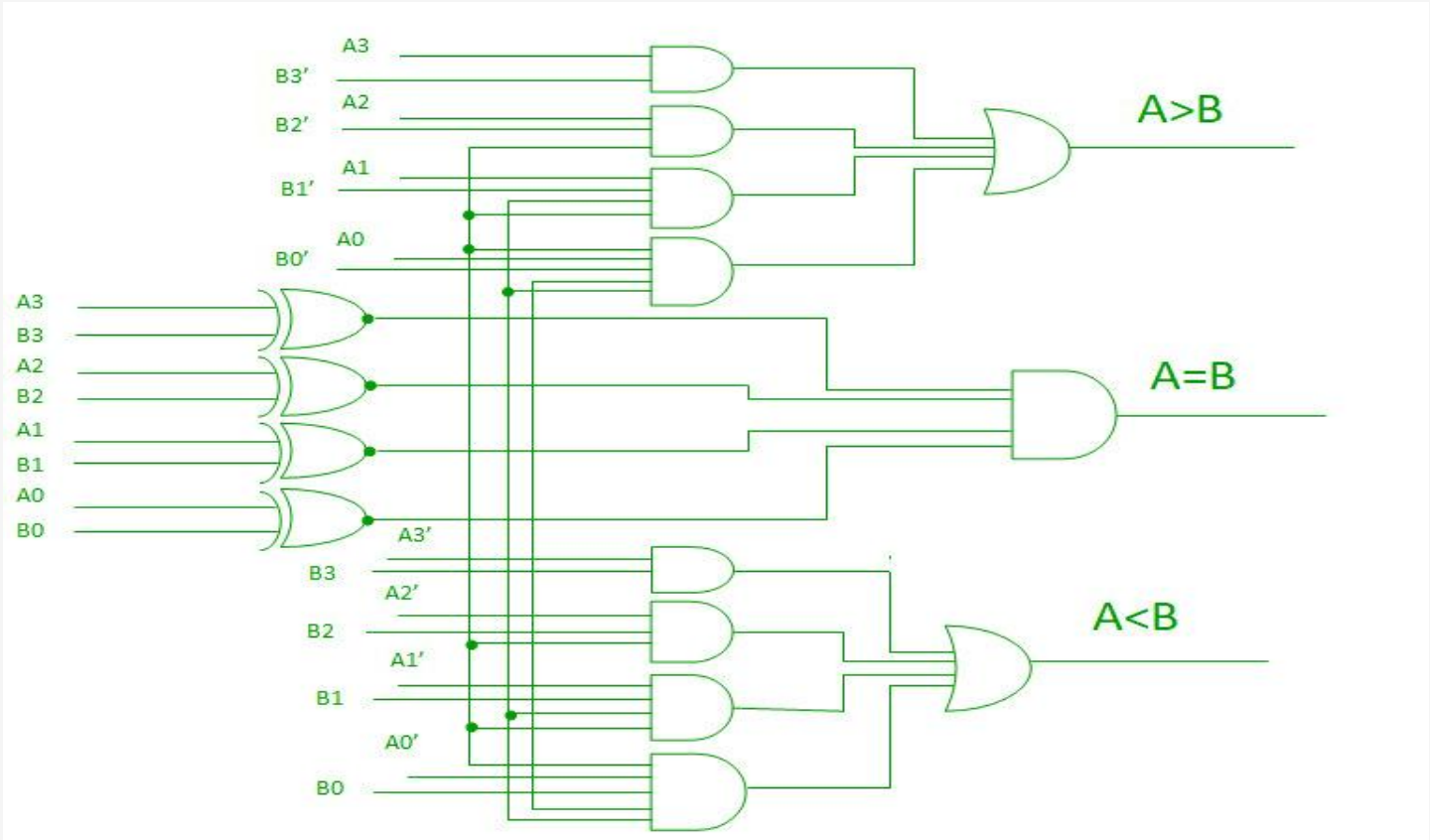
➢ Truth table of 4-Bit Comparator :

COMPARING INPUTS				OUTPUT		
A3, B3	**A2, B2**	**A1, B1**	**A0, B0**	**A > B**	**A < B**	**A = B**
A3 > B3	X	X	X	H	L	L
A3 < B3	X	X	X	L	H	L
A3 = B3	A2 >B2	X	X	H	L	L
A3 = B3	A2 < B2	X	X	L	H	L
A3 = B3	A2 = B2	A1 > B1	X	H	L	L
A3 = B3	A2 = B2	A1 < B1	X	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 > B0	H	L	L
A3 = B3	A2 = B2	A1 = B1	A0 < B0	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	H	L	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	L	H
H = High Voltage Level, L = Low Voltage, Level, X = Don't Care						

A=B: (A3 Ex-Nor B3) (A2 Ex-Nor 82) (A1 Ex-Nor B1) (A0 Ex-Nor B0)

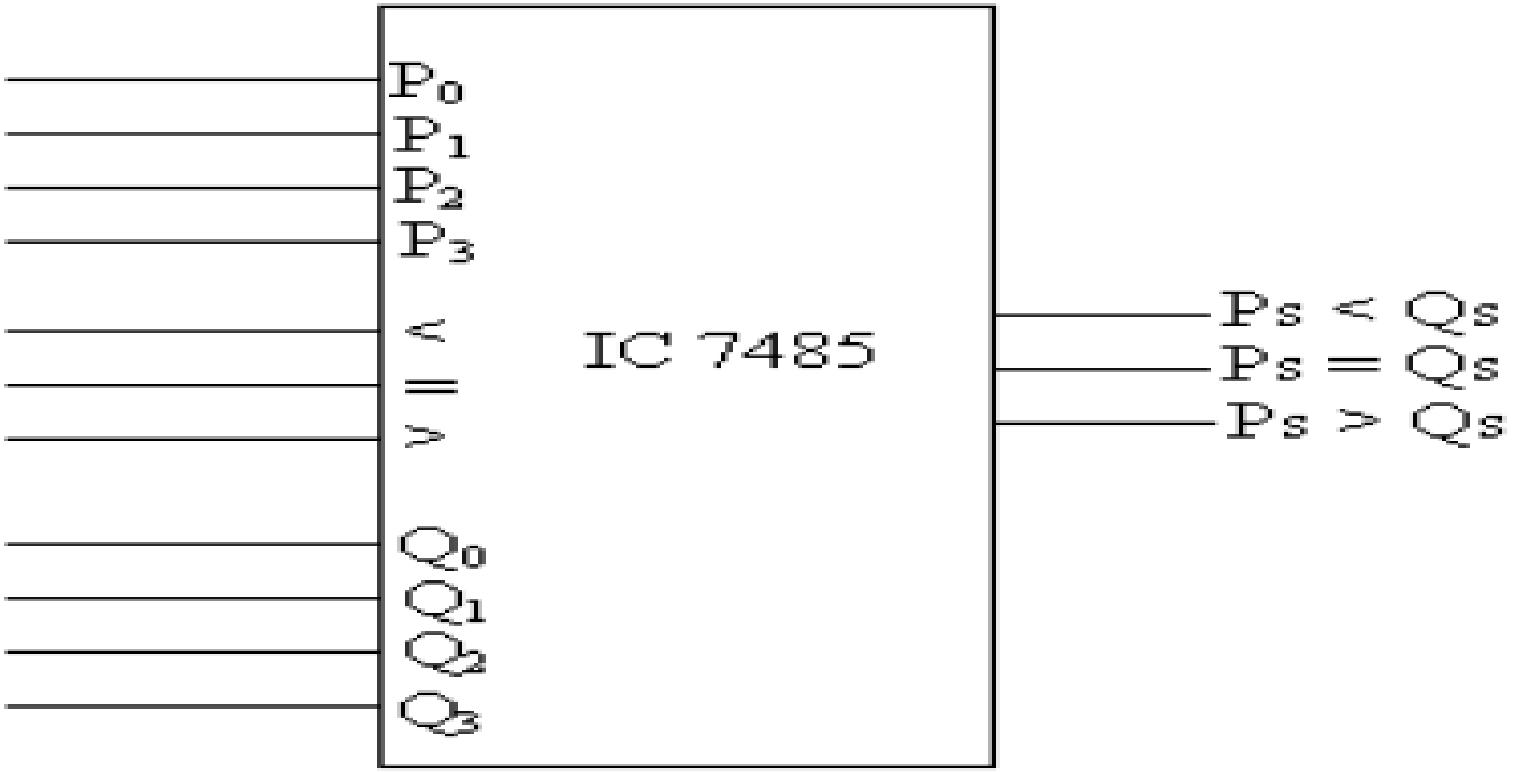
4-Bit Comparator

➢ By using these Boolean expressions, we can implement a logic circuit for this comparator as given below:



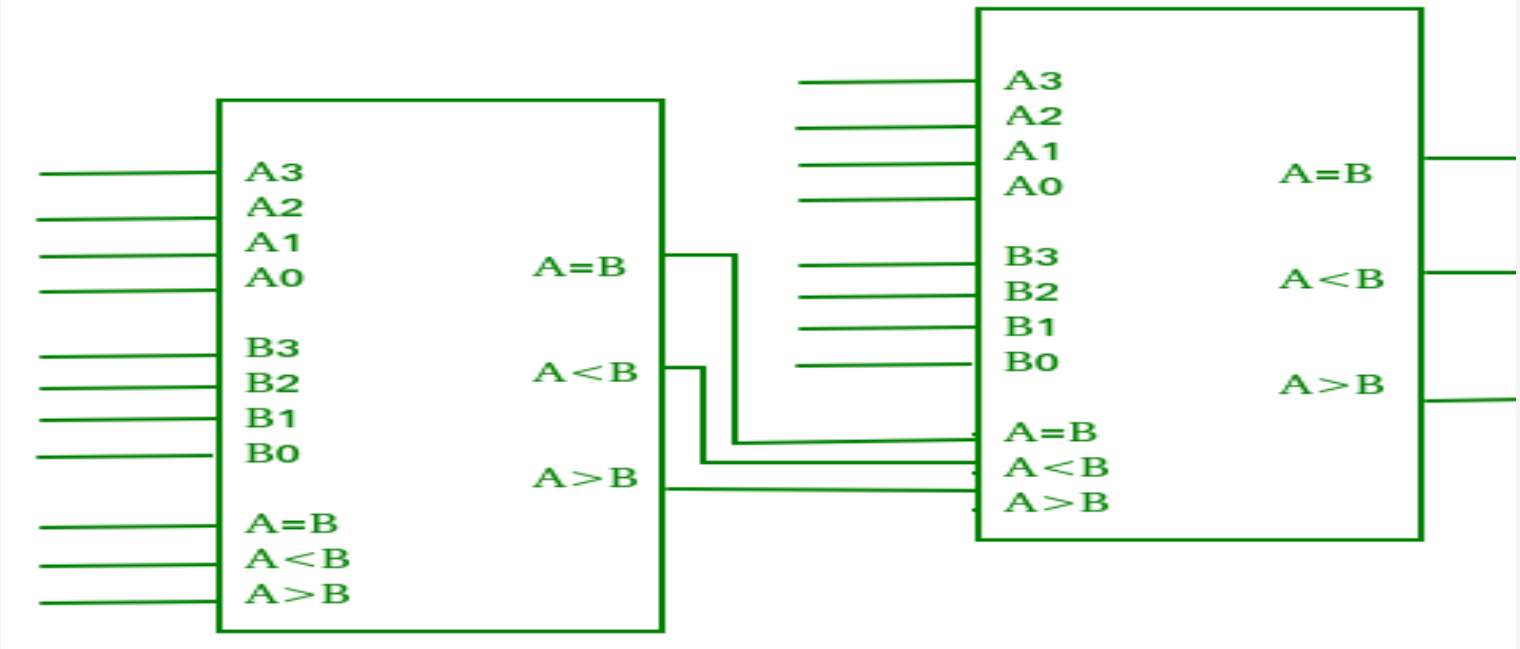
4-Bit Comparator(IC-7485)

- Figure shows a 4-bit comparator.



Cascading Comparator

- A comparator performing the comparison operation to more than four bits by cascading two or more 4-bit comparators is called cascading comparator.
- When two comparators are to be cascaded, the outputs of the lower-order comparator are connected to corresponding inputs of the higher-order comparator.



Applications of Comparators

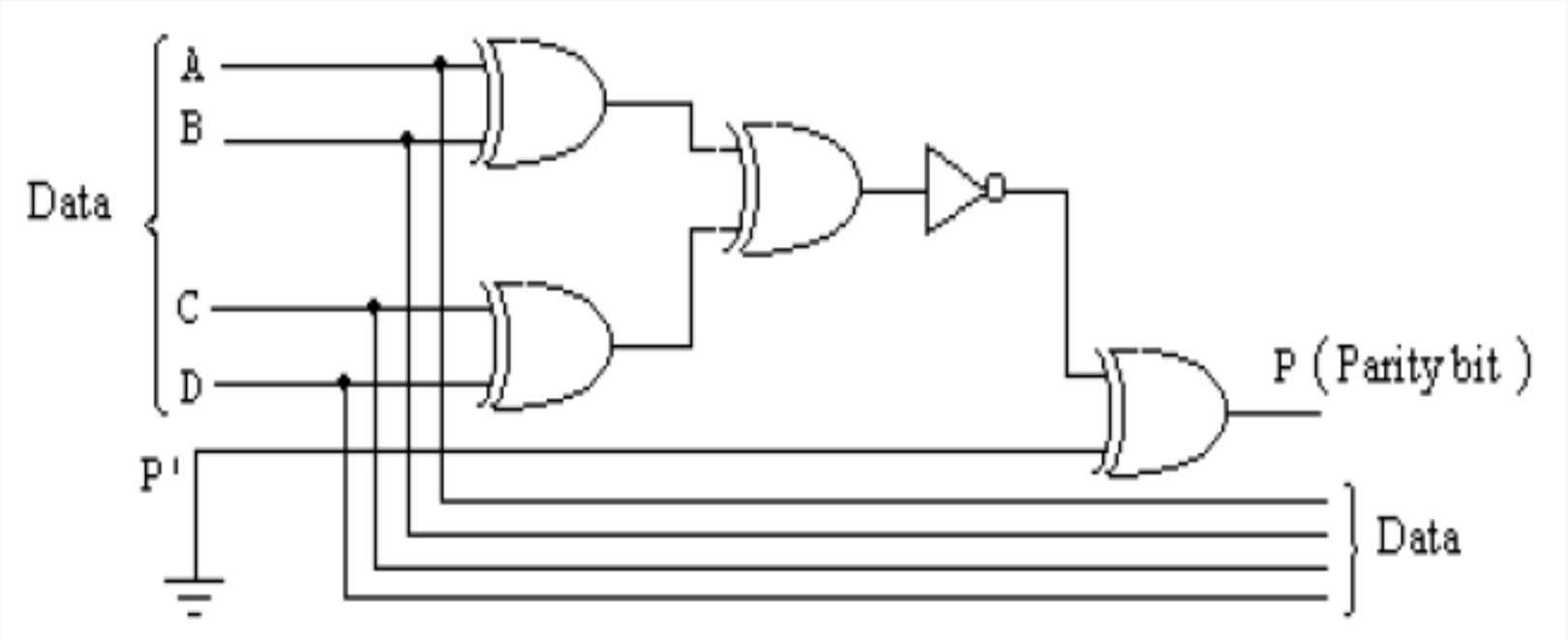
- Comparators are used in central processing units (CPUs) and microcontrollers (MCUs).
- These are used in control applications in which the binary numbers representing physical variables such as temperature, position, etc. are compared with a reference value.
- Comparators are also used as process controllers and for Servo motor control.
- Used in password verification and biometric applications.

Parity Generator/Checker

- In the transmission of data in digital systems, error may occur due to change of data bit (0 by 1 or vice versa).
- This change may be due to component malfunctions or the electrical noise.
- This problem is removed by adding one additional bit in the data to be transmitted. This extra bit is known as parity bit.
- The parity bit detects the single error in the transmission.
- Parity is the number of 1's in the given data or word. If the number of 1's in the given data is even then parity is called as even parity; if on the other hand the number of 1's is odd then the parity is called as odd parity.
- The parity bit of the data or the word is generated by the parity generator.
- This parity generator gives output P (parity bit) as logic 1 if the number of 1's in the four bit input data is even; and P is logic 0 if the number of 1's in the four bit input data is odd.
- That is for even parity of the input data, output is 1 and for odd parity of the input data, output is 0.

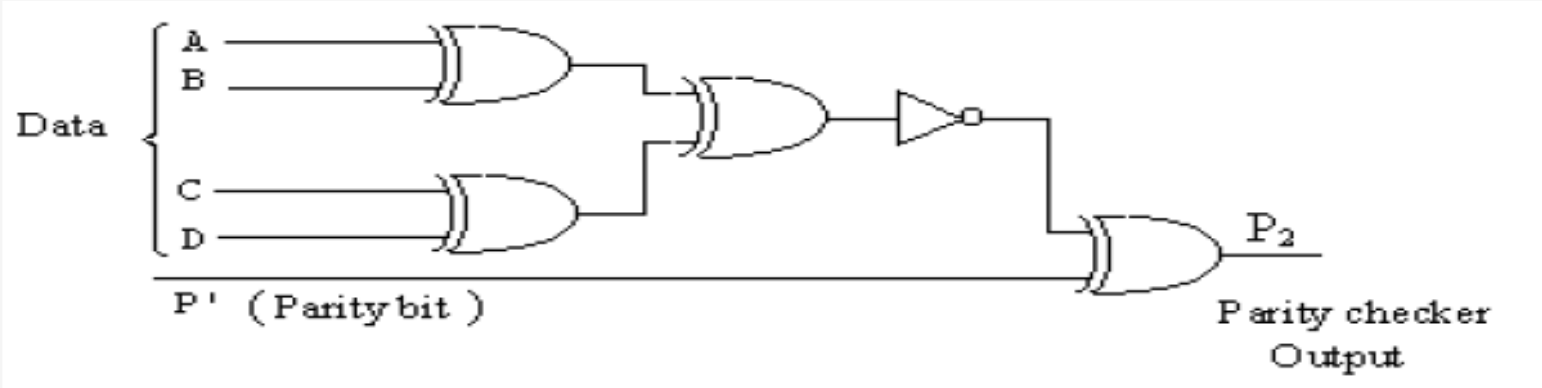
Parity Generator/Checker

- The logic diagram of the parity bit generator of four bit is shown in figure.



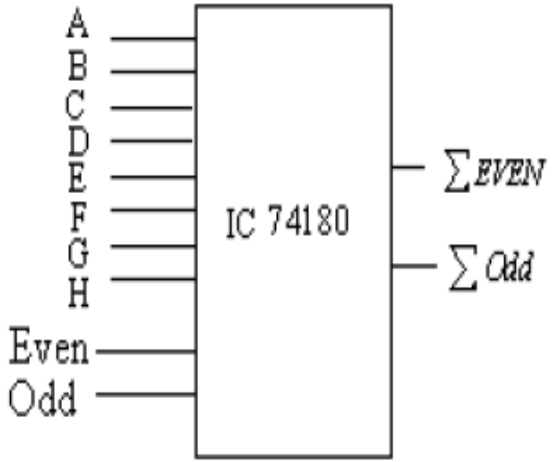
Parity Generator/Checker

- The parity bit P generated by the parity generator is sent along with the data and at the receiver end data as well as the parity bit is checked by the parity checker.
- The logic circuit for the parity checker is the same as that of the parity generator with the only difference that in the parity checker the terminal P’ in not grounded, but the parity bit received at the receiver end is connected to the point P’.
- So at the receiver the received data and the parity bit form the five bit data which is always having the odd parity.
- It is clear from the fact that if the data A, B, C and D is odd (even) then parity bit is 0 (1), and therefore the received data and the parity bit is always is odd.



Parity Generator/Checker(IC-74180)

- Parity generator/checker is available in the form of IC.
- Figure shows the block diagram of 8 bit parity generator/checker IC 74180 and its truth table.
- It can be used to check for even or odd parity on a 9 – bit code (8 – bit data and one parity bit).
- It can also be used to generate a 9 – bit even or odd parity code.



Parity of 8 bit data	Inputs		Outputs	
	Even	Odd	$\sum$ even	$\sum$ Odd
Even	1	0	1	0
Odd	1	0	0	1
Even	0	1	0	1
Odd	0	1	1	0
ϕ	1	1	0	0
ϕ	0	0	1	1

Definations

Natural BCD Code: Decimal digits 0 through 9 are represented by their natural binary equivalents using 4 bits.

Excess 3 code: Another BCD code in 4 bit binary code. The decimal digit is obtained by adding 3 to the natural BCD code.

Gray code: It is a code in which each gray code number differs from the preceding & succeeding number by a single bit.

Construction of Gray Code:

1. **A 1 bit gray code has two code words 0 and 1 representing decimal numbers 0 & 1 respectively.**
2. **An n-bit (n>=2) gray code will have first 2^{n-1} gray codes of (n-1) bits written in order with a leading 0 appended.**
3. **The last 2^{n-1} gray codes will be equal to the gray code words of an (n-1) bit gray code written in reverse order with a leading 1 appended.**

Continue….

E.g. :

1. 1 bit gray code

Decimal number	Gray Code
0	0
1	1

2. 2 bit gray code

Decimal number	Gray Code	
0	0	0
1	0	1
2	1	1
3	1	0

3. 3 bit Gray Code

Decimal number	Gray Code		
0	0	0	0
1	0	0	1
2	0	1	1
3	0	1	0
4	1	1	0
5	1	1	1
6	1	0	1

Instructions/Guidelines/References

Text Books :

1 R. P. Jain, “Modern Digital Electronics”, 3rd Edition, Tata McGraw-Hill, 2003, ISBN0 – 07 – 049492 – 4.

2 J. Bhaskar, “VHDL Primer” 3rd Edition, Pearson Edition.

3 Stephen Brown, ZvonkoVranesic, “Fundamentals of Digital Logic with VHDL Design”, McGraw Hill, ISBN– 13:978-1-25-902597-6.

4 G.K.Kharate, ”Digital Electronics”, Oxford University Press.

Reference Books :

1. John Yarbrough, “Digital Logic applications and Design” Thomson.

2 Flyod “Digital Principles”, Pearson Education.

3 Malvino, D.Leach “Digital Principles and Applications”, 5th edition, Tata Mc- Graw Hill.

4 Douglas L. Perry, “VHDL Programming by Example”, Tata McGraw Hill.

5 G. K. Kharate, “Digital Electronics”, Oxford University Press.

Thank You

End of Unit-2