Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

Digital Electronics and Computer Architecture

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

UNIT –II

Combinational Logic Design

Digital Electronics -Syllabus

Combinational Logic Design:

Design examples: Arithmetic circuits, Comparators, Code converters, Parity generators and checkers, Arithmatic and Logic Unit, design of adder and fast adder, look ahead carry generator, Implementation of SOP and POS using MUX,DEMUX,DECODER.

Design examples: Arithmetic circuits

Half Adder & Full Adder

Half Subtractor & Full Subtractor

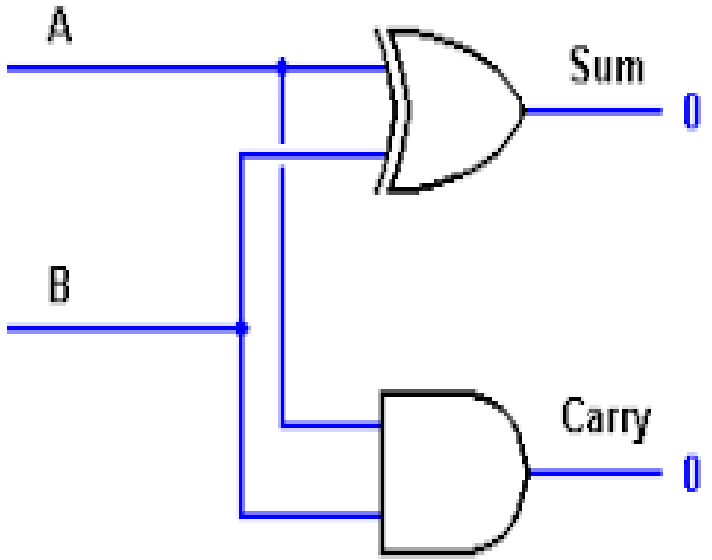
Half Adder

- It is used to add single bit
- A logic circuit for the addition of two 1-bit numbers is referred to as a **Half Adder**
- The addition process is similar to Binary addition
- E.g.

$$\begin{array}{rl} 1011 & (11)_{10} \\ +\ 1100 & (12)_{10} \\ \hline 1\,0111 & \end{array}$$

Half Adder

Inputs		Outputs	
A	B	Sum (S)	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$$S = \bar{A}B + A\bar{B} = A \oplus B$$

$$C = AB$$

Half Subtractor

- Subtracting a single-bit binary value Y from anther X (I.e. X -Y) produces a difference bit D and a borrow out bit B-out.
- This operation is called half subtraction and the circuit to realize it is called a half subtractor.

Half Subtractor Truth Table

Inputs		Outputs	
X	Y	D	B-out
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

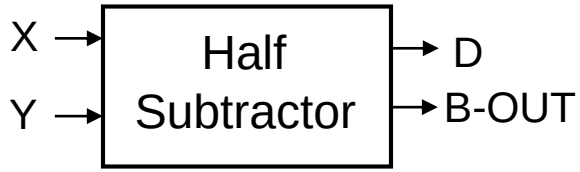
$D(X,Y) = \Sigma\,(1,2)$

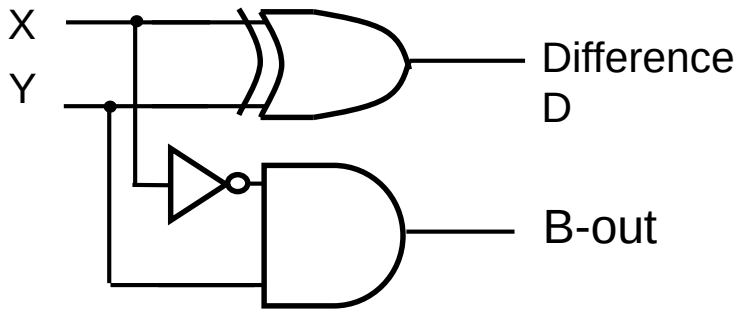
$D = X'Y + XY'$

$D = X \oplus Y$

$B\text{-out}(x, y, C\text{-in}) = \Sigma\,(1)$

$B\text{-out} = X'Y$



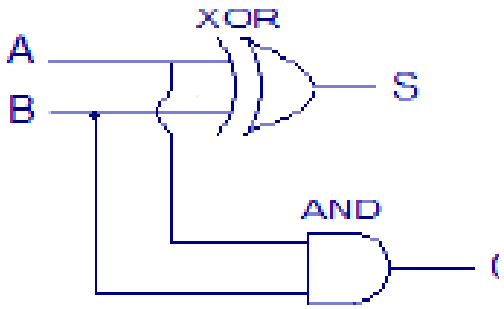


Half Adder USING 4:1 MUX :

Inputs		Outputs	
A	B	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Truth table

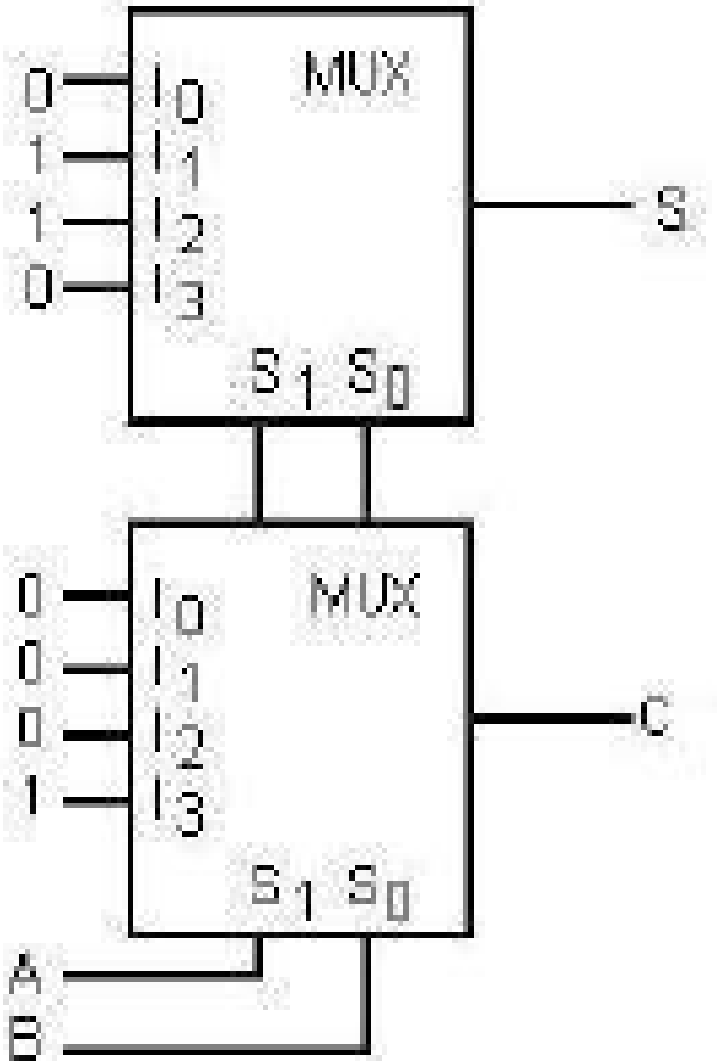
Schematic



Realization

MULTIPLEXER $Y = \bar{S}o\bar{S}1A + \overline{So}S1B + So\overline{S1}C + SoS1D$

SUM=A`B+AB`
CARRY=AB

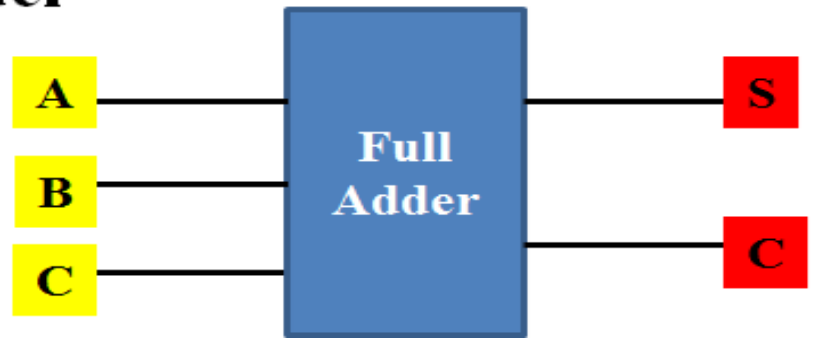


Full Adder

- The half adder has only 2 I/Ps and there is no provision to add carry coming from the lower order bits, when multi-bit addition is performed

- For this purpose 3rd I/P terminal is added & this circuit is used to add An, Bn & Cn-1

- This circuit is referred to as **Full Adder**

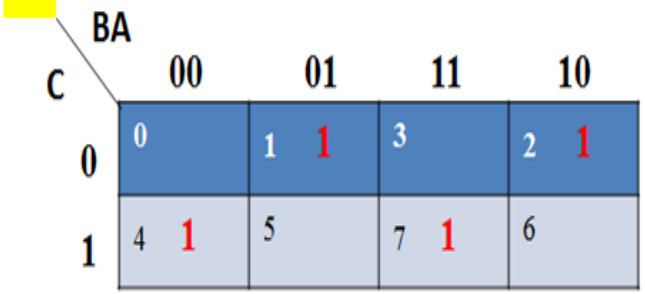


Full Adder Truth Table

Inputs			Outputs	
An	Bn	Cn-1	Sn	Cn
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

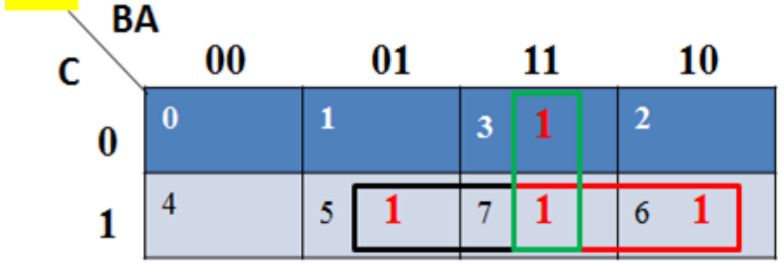
$S = \sum m(1,2,4,7)$

S



$Carry\ (C) = \sum m(3,5,6,7)$

C



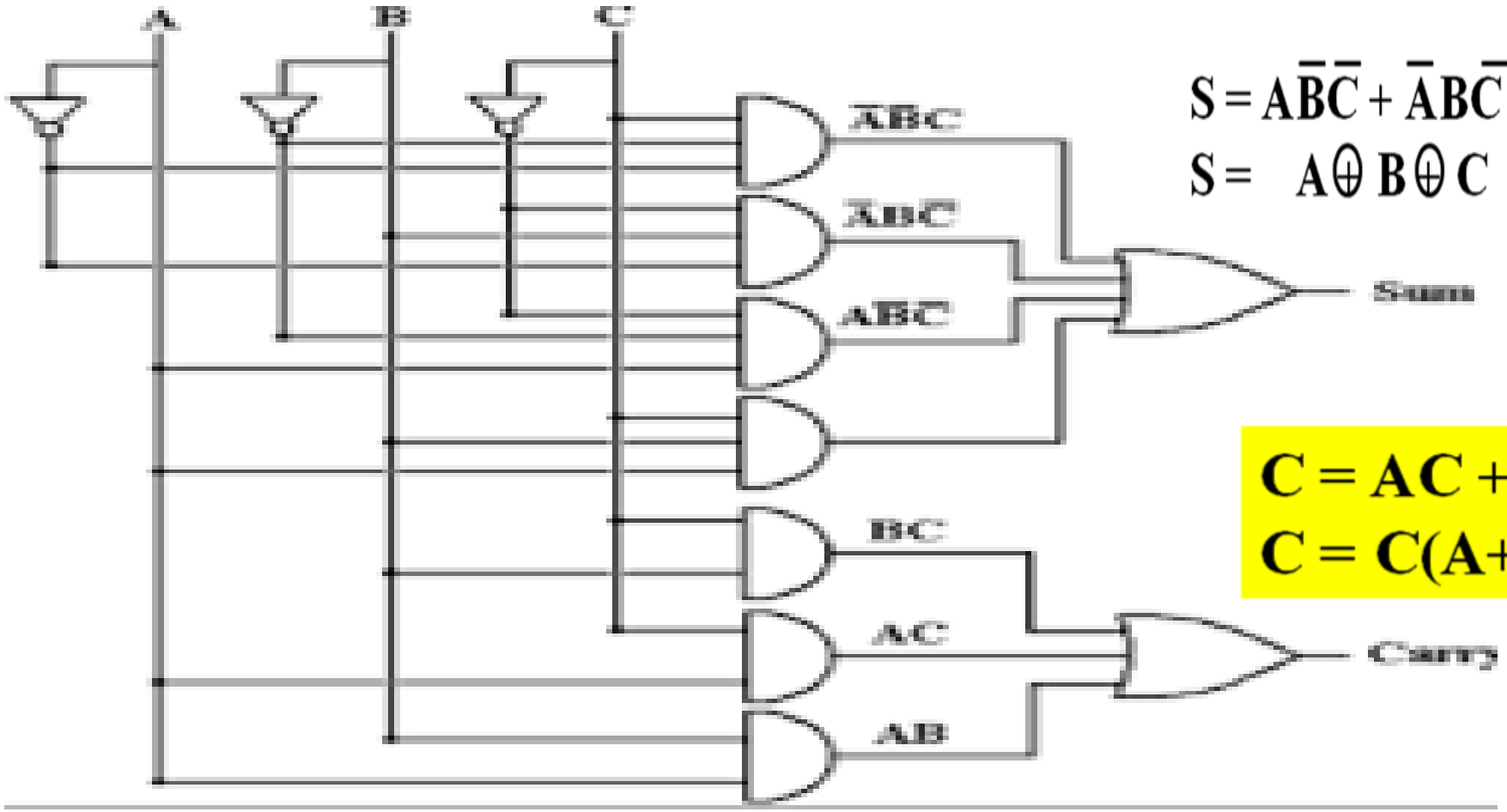
$$S = A\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C + ABC$$

$$S = A \oplus B \oplus C$$

$$C = AC + BC + AB$$

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

Full Adder Circuit (Sum & Carry) using Gates



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

$$S = A \oplus B \oplus C$$

$$C = AC + BC + AB$$

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

Full Subtractor

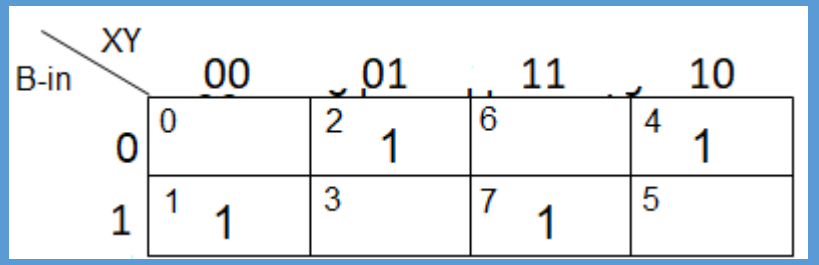
Subtracting two single-bit binary values, Y, B-in from a single-bit value X produces a difference bit D and a borrow out B-out bit. This is called full subtraction

Full Subtractor Truth Table

X	Y	B-in	D	B-out
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

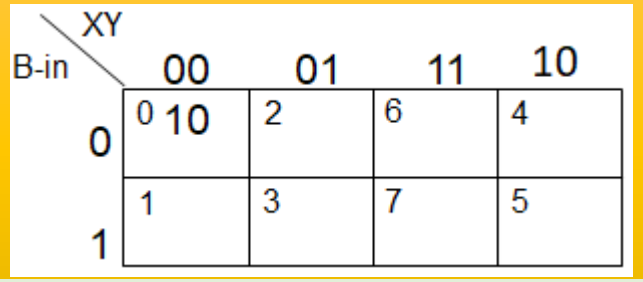
D(X,Y, B-in) = Σ (1,2,4,7)
B-out(x, y, B-in) = Σ (1,2,3,7)

Difference D



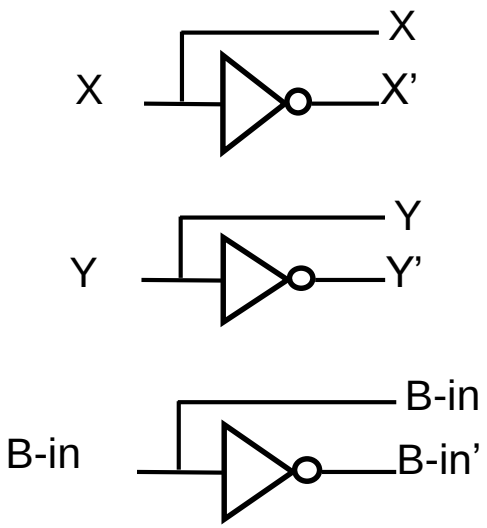
D = X'Y'(B-in) + XY'(B-in)' + XY'(B-in)' + XY(B-in)
D = X ⊕ Y ⊕ (B-in)

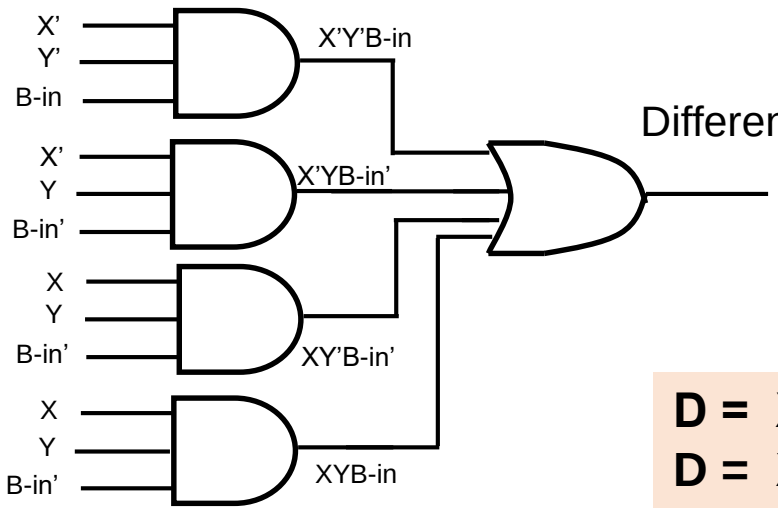
Borrow out B-out



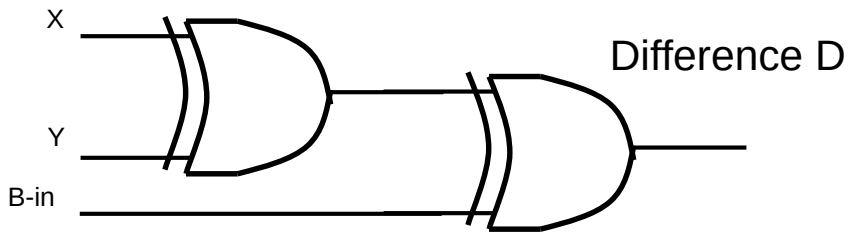
B-out = X'Y + X'(B-in) + Y(B-in)

Full Subtractor Circuit Using AND-OR

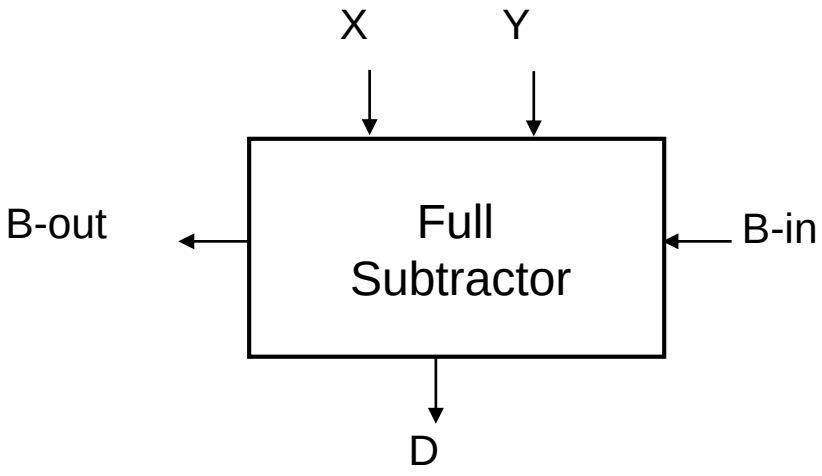


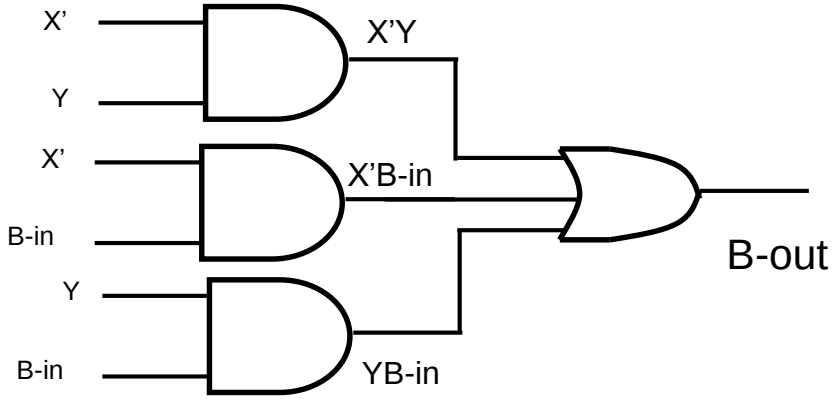


OR



D = X'Y'(B-in) + XY'(B-in)' + XY'(B-in)' + XY(B-in)
D = X ⊕ Y ⊕ (B-in)

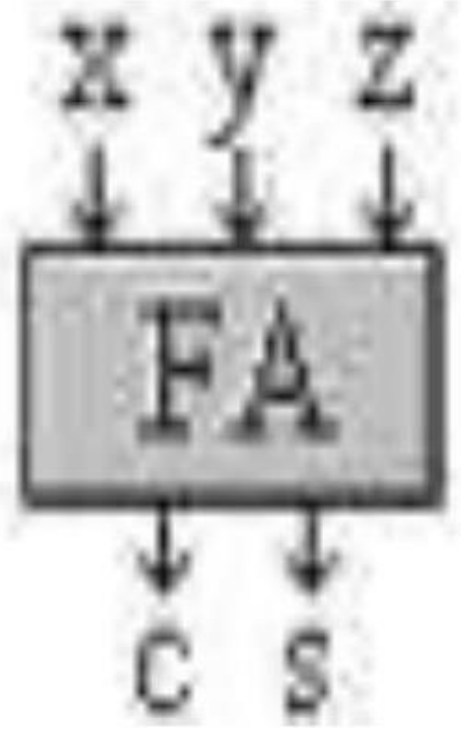




B-out = X'Y + X'(B-in) + Y(B-in)

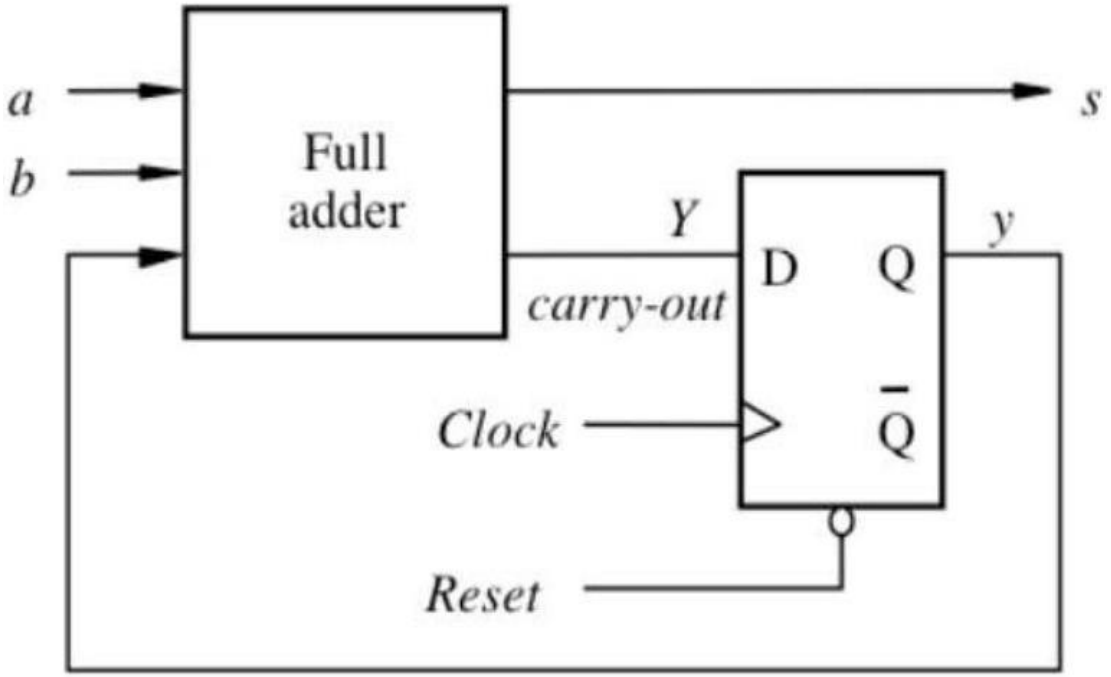
Parallel Adder

- A circuit , consisting of n full adders , that will add n-bit binary numbers.
- The output consists of n sum bits and a carry bit.
- CO of one full adder is connected to CI of the next full adder.



Serial Adder

- The serial adder is a binary adder that is capable of forming sum and carry outputs for addend and augend words of greater than one bit in length.



Comparison of Parallel and Serial Adder

Parallel	Serial
• Faster.	• Slower
• It uses registers with parallel load capacity.	• It uses shift registers.
• Number of full adder circuit is equal to number of bits in binary adder.	• It requires one full adder circuit.
• It is a combinational circuit.	• It is sequential circuit.
• Time required does not depend on the number of bits	• Time required for addition depends on number of bits.

Structure of Parallel Adder

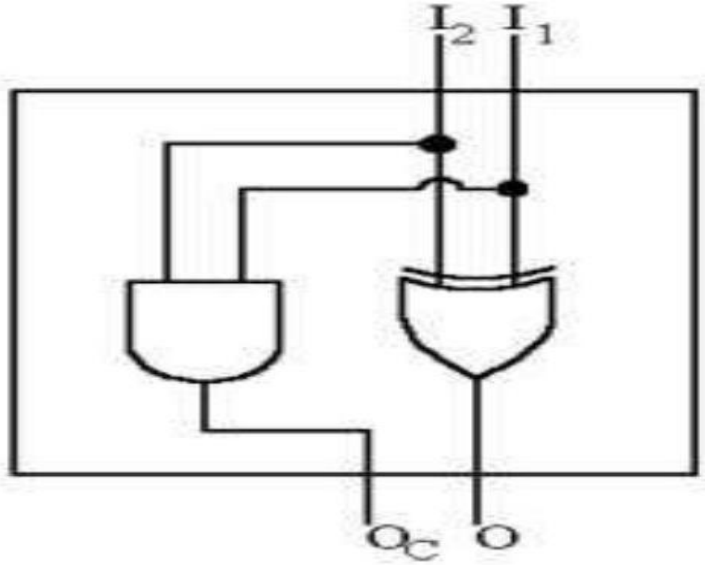
- Parallel adder nothing but a cascade of several full adders.
- The number of full adders used will depend on the number of bits in the binary digits which require to be added.

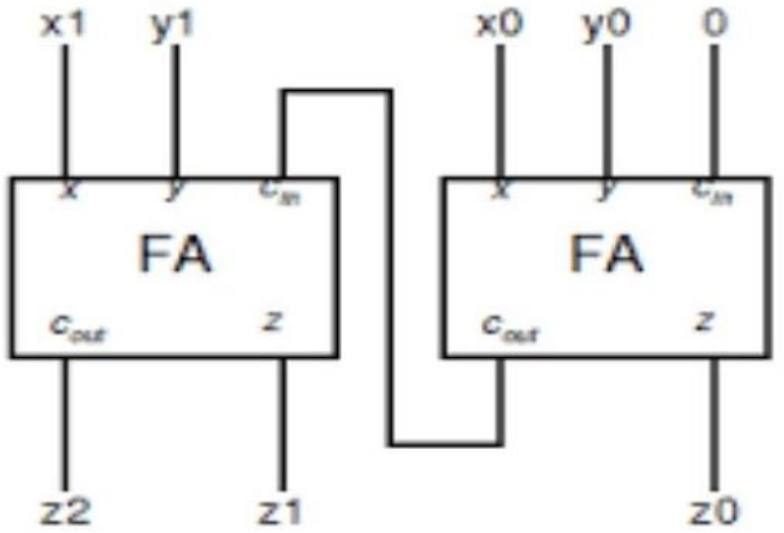
Types of Parallel Adders

- ONE BIT PARALLEL ADDER
- TWO BIT PARALLER ADDER
- FOUR BIT PARALLER ADDER

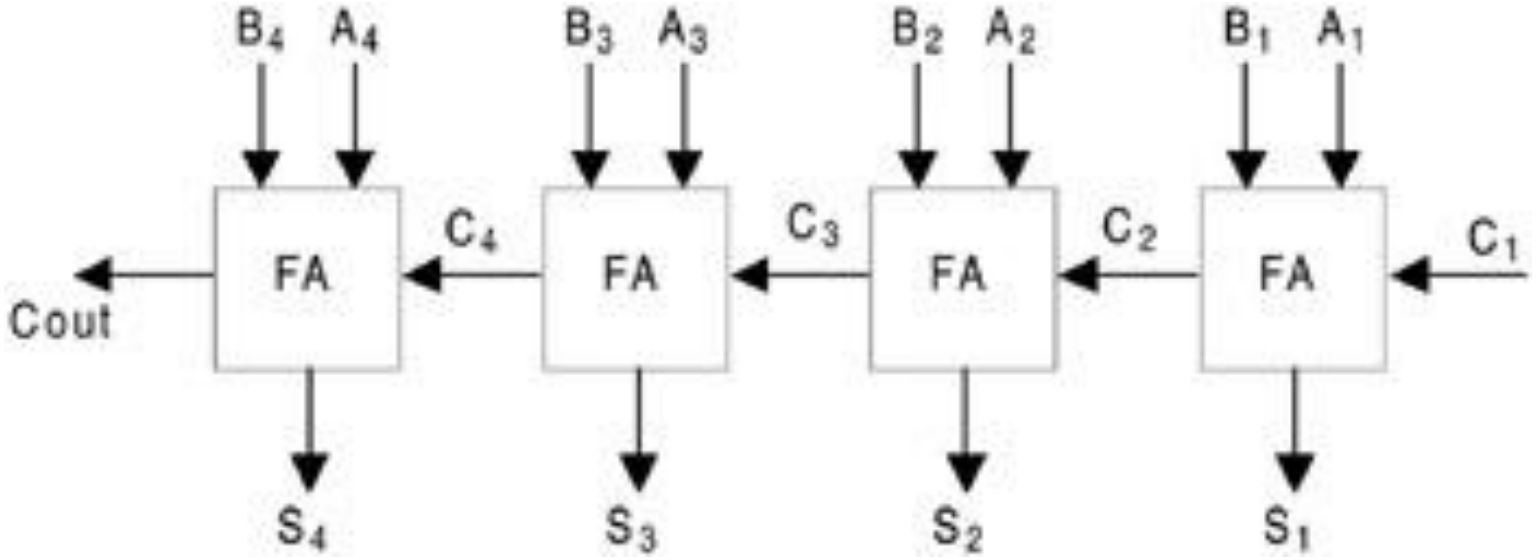
Types of Parallel Adders

One Bit Parallel Adder



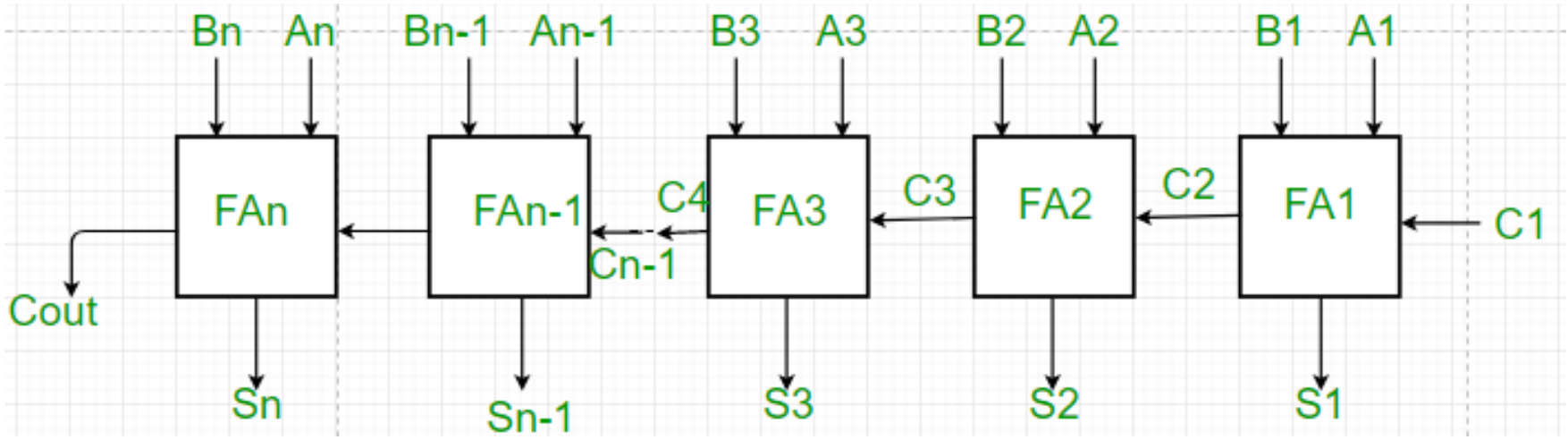


Two Bit Parallel Adder



Four Bit Parallel Adder

Working of Parallel Adder



1.As shown in the figure, firstly the full adder FA1 adds A1 and B1 along with the carry C1 to generate the sum S1 (the first bit of the output sum) and the carry C2 which is connected to the next adder in chain.

2. Next, the full adder FA2 uses this carry bit C2 to add with the input bits A2 and B2 to generate the sum S2(the second bit of the output sum) and the carry C3 which is again further connected to the next adder in chain and so on.

3. The process continues till the last full adder FAn uses the carry bit Cn to add with its input An and Bn to generate the last bit of the output along last carry bit Cout.

Advantages and Disadvantages of Parallel Adder

Advantages of parallel Adder –

1. The parallel adder performs the addition operation faster as compared to serial adder.
2. Time required for addition does not depend on the number of bits.
3. The output is in parallel form i.e all the bits are added/subtracted at the same time.
4. It is less costly.

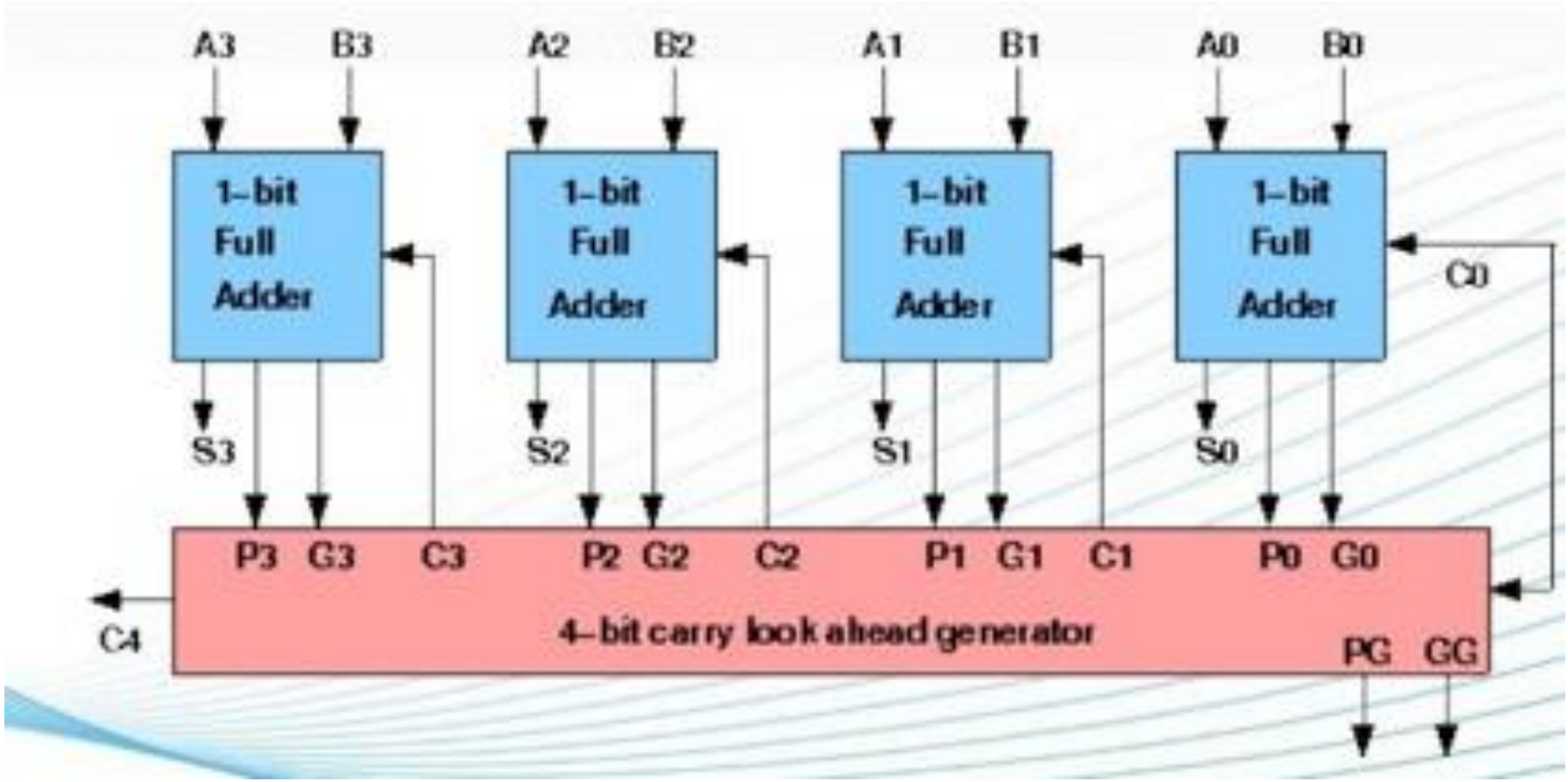
Disadvantages of parallel Adder–

1.Each adder has to wait for the carry which is to be generated from the previous adder in chain.

2.The propagation delay(delay associated with the travelling of carry bit) is found to increase with the increase in the number of bits to be added.

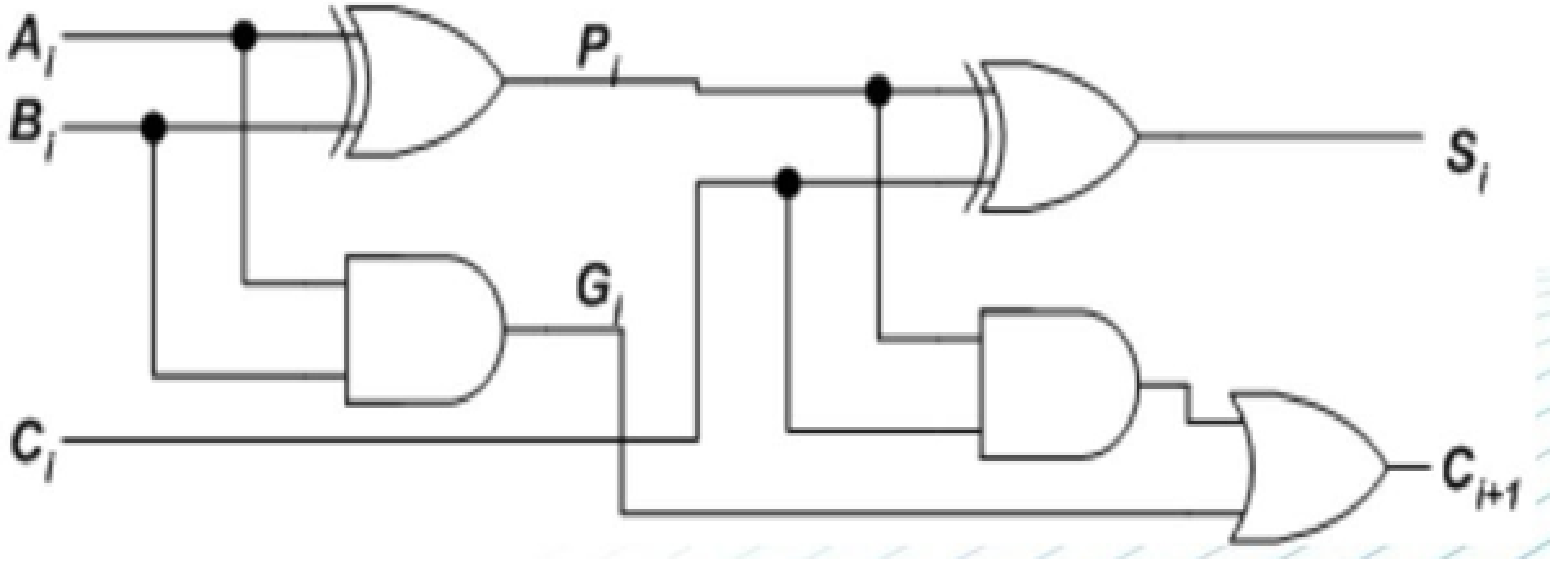
Carry Look-ahead Adder

A carry look-ahead adder reduces the propagation delay by introducing more complex hardware.



Carry Look-ahead Adder

A full adder in look ahead carry adder circuit is designed using two half adder circuits connected as shown. The circuit is design to reduce the dependency of next stage carry on previous stage carry



$G_i = A_iB_i$ where G is called carry generator

$P_i = A_i \oplus B_i$ where P is called carry propagator

Carry Look-ahead Adder

$G_i = A_iB_i$ where G is called carry generator

$P_i = A_i \oplus B_i$ where P is called carry propagator

$Ci+1 = PiCi + Gi$

For four bit parallel adder, Carry bits of all stages can be written as

$C_1 = C_0P_0 + G_0$ …….. (1)

$C_2 = C_1P_1 + G_1$ …….. (2)

$C_3 = C_2P_2 + G_2$ …….. (3)

$C_4 = C_3P_3 + G_3$ …….. (4)

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.

$C_1 = C_0P_0 + G_0$

$C_2 = C_0P_0P_1 + G_0P_1 + G_1$

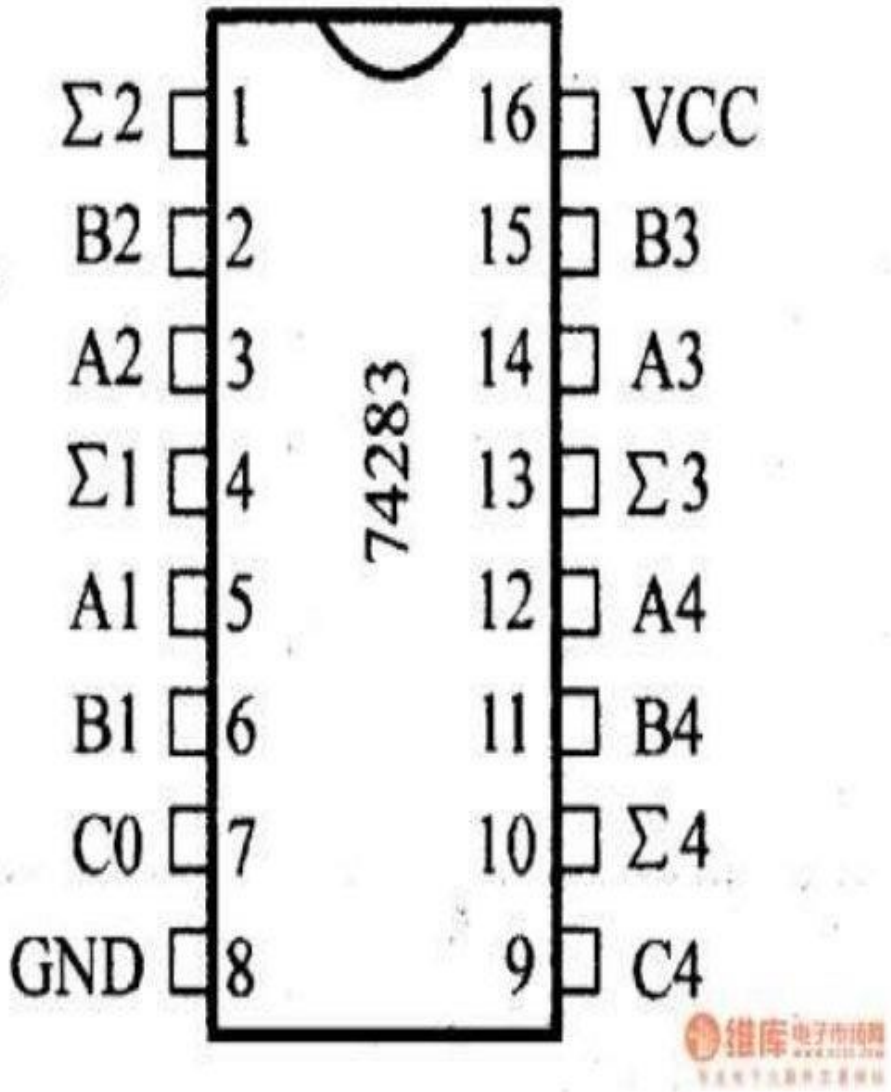
$C_3 = C_0P_0P_1P_2 + G_0P_1P_2 + G_1P_2 + G_2$

$C_4 = C_0P_0P_1P_2P3 + G_0P_1P_2P_3 + G_1P_2P_3 + G_2P_3 + G_3$

As Gi and Pi depend only on Ai and Bi, dependency of next carry being dependent on previous carry is reduced.

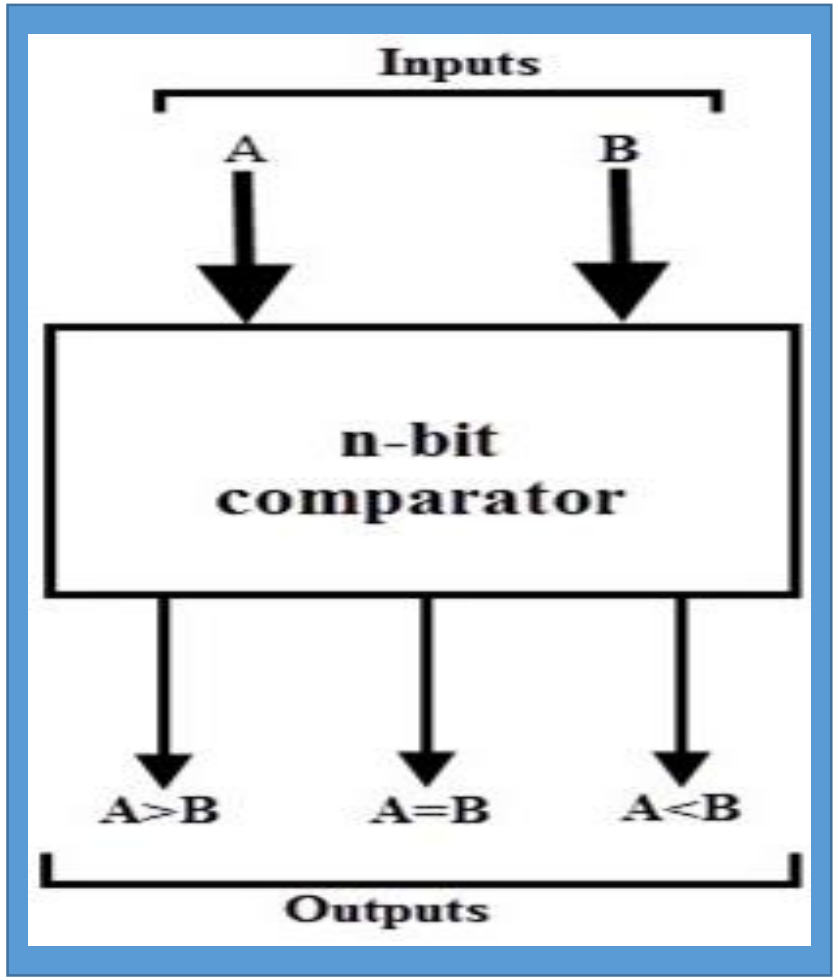
IC

- Several configuration of binary parallel adders are available in IC form.
- Popular example:74283(4-bit binary parallel adder)



Digital Comparator

- A digital comparator is a combinational circuit that compares two digital or binary numbers (consider A and B) and determines their relative magnitudes in order to find out whether one number is equal, less than or greater than the other digital number.
- Three binary variables are used to indicate the outcome of the comparison as A>B, A<B, or A=B.
- The figure shows the block diagram of a n-bit comparator which compares the two numbers of n-bit length and generates their relation between themselves.



Digital Comparator

Input		Output		
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

Y(A<B)

A \ B	0	1
0	0	1
1	0	0

Y(A<B) = A'B

Y(A=B)

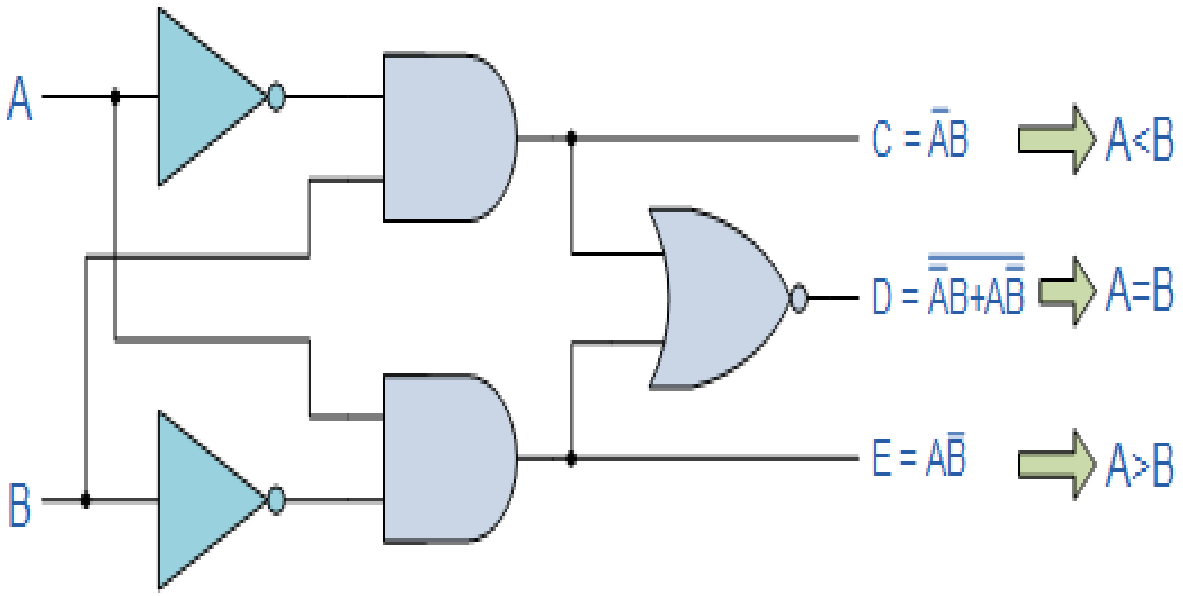
A \ B	0	1
0	1	0
1	0	1

Y(A=B) = A'B' + AB

Y(A>B)

A \ B	0	1
0	0	0
1	1	0

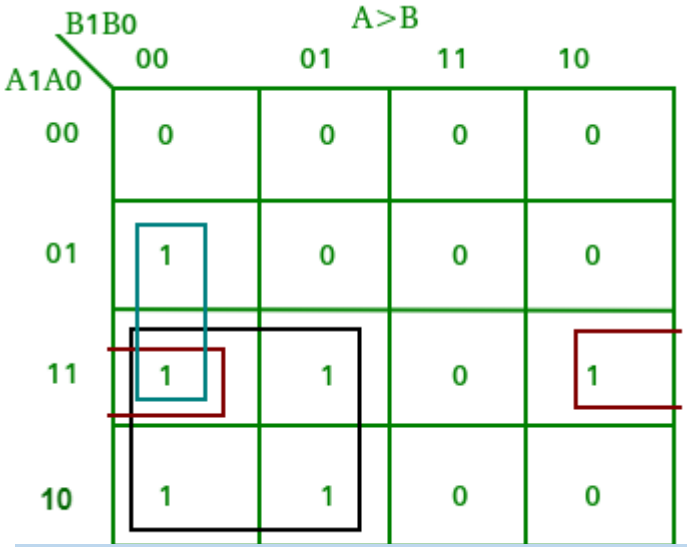
Y(A>B) = AB'



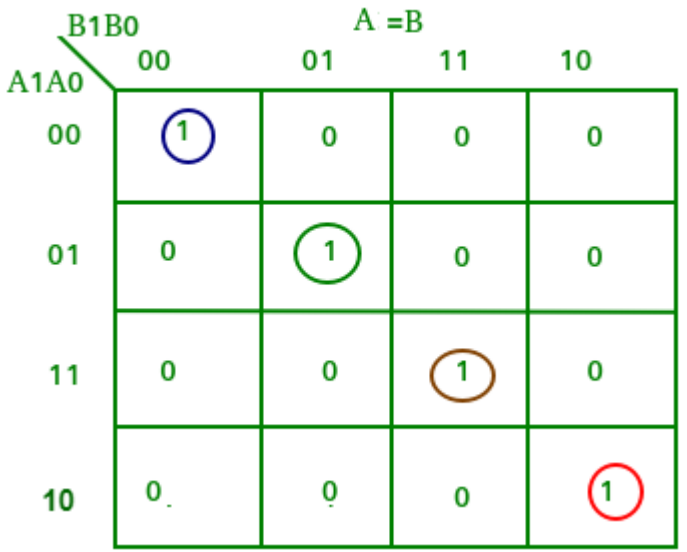
2-Bit Digital Comparator

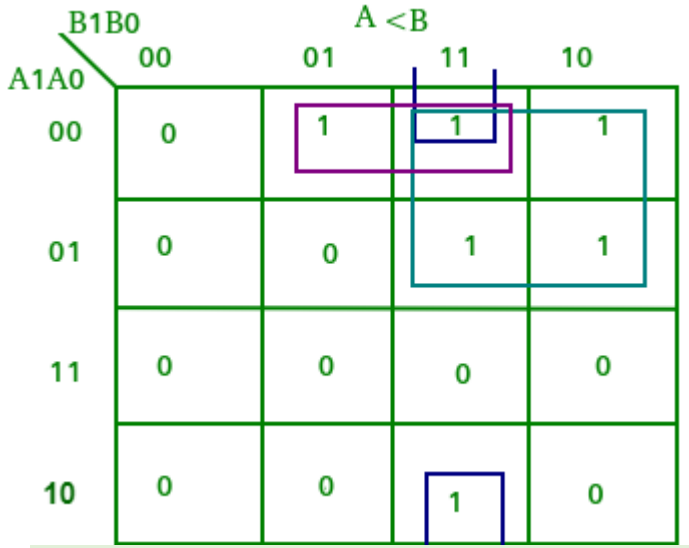
INPUT				OUTPUT		
A1	A0	B1	B0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

2-Bit Digital Comparator



A>B= A1B1'+ A0B1'B0'+ A1A0B0'





A<B: A1'B1 + A0'B1B0 + A1'A0'B0

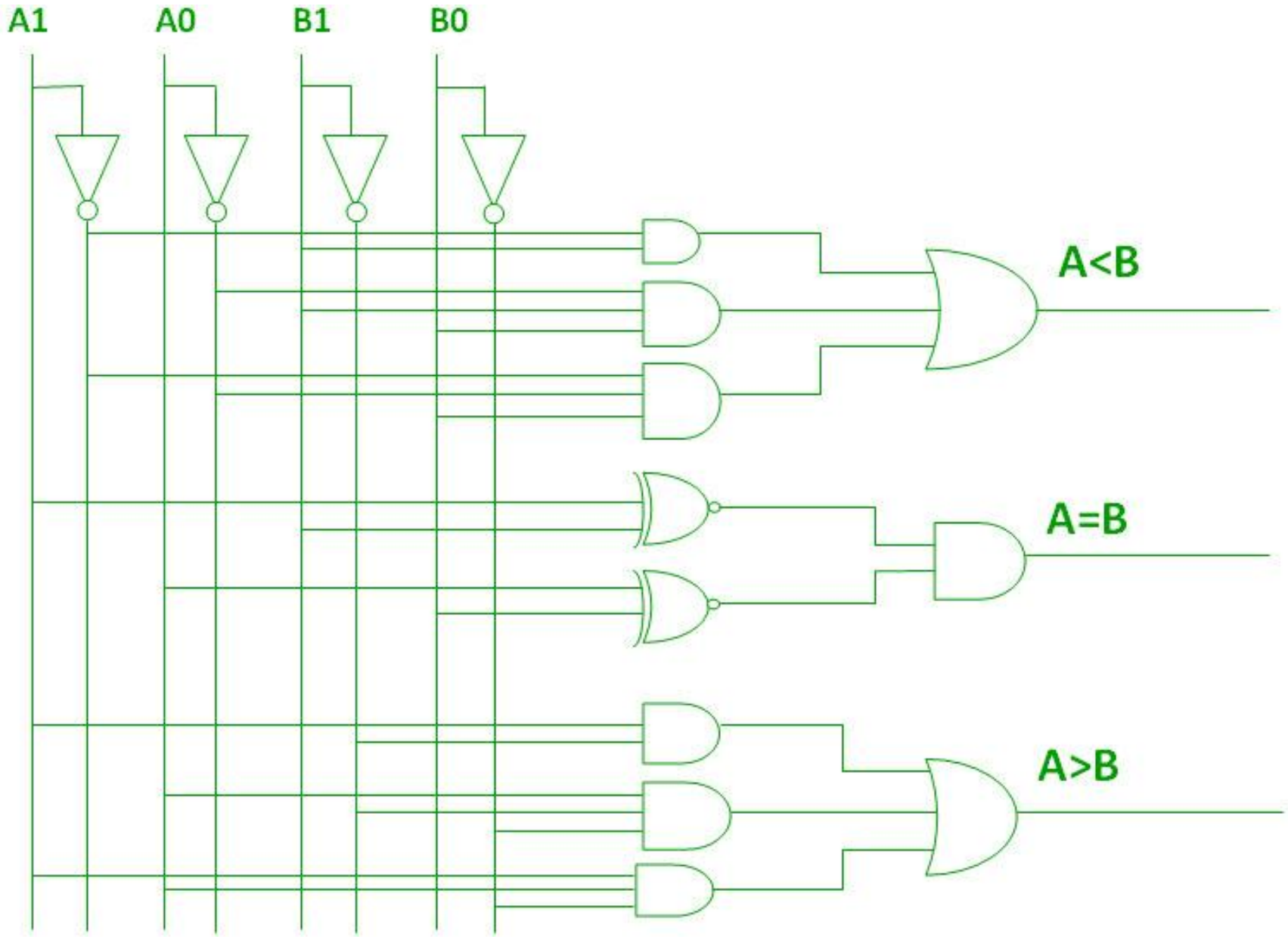
A=B: A1'A0'B1'B0' + A1'A0B1'B0 + A1A0B1B0 + A1A0'B1B0'

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

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

: (A0 Ex-Nor B0) (A1 Ex-Nor B1)

2-Bit Digital Comparator



4-Bit Digital Comparator

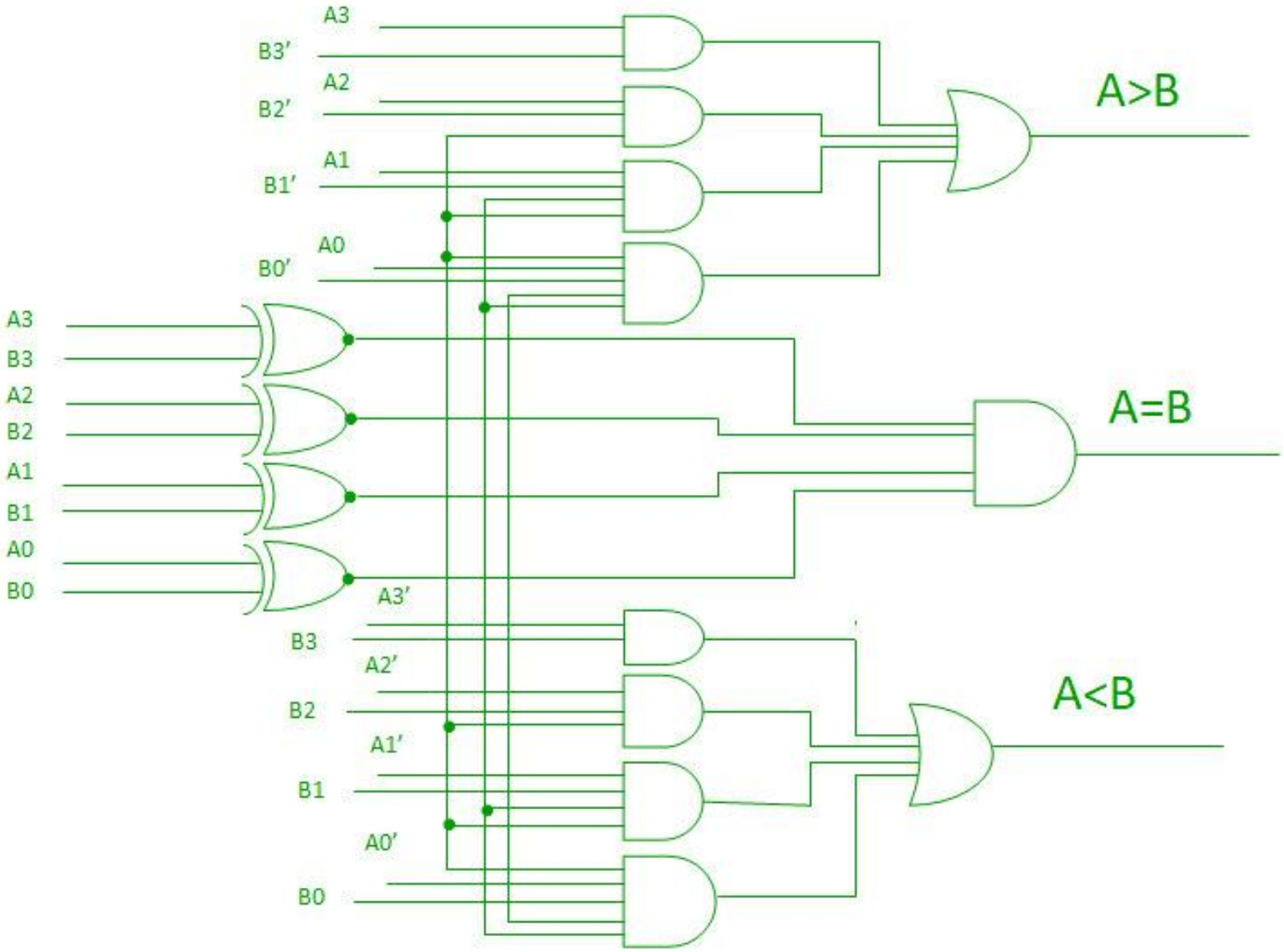
A 4 bit comparator compares two 4 bit binary numbers
The inputs are A(A3,A2,A1,A0) and B (B3,B2,B1,B0)

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

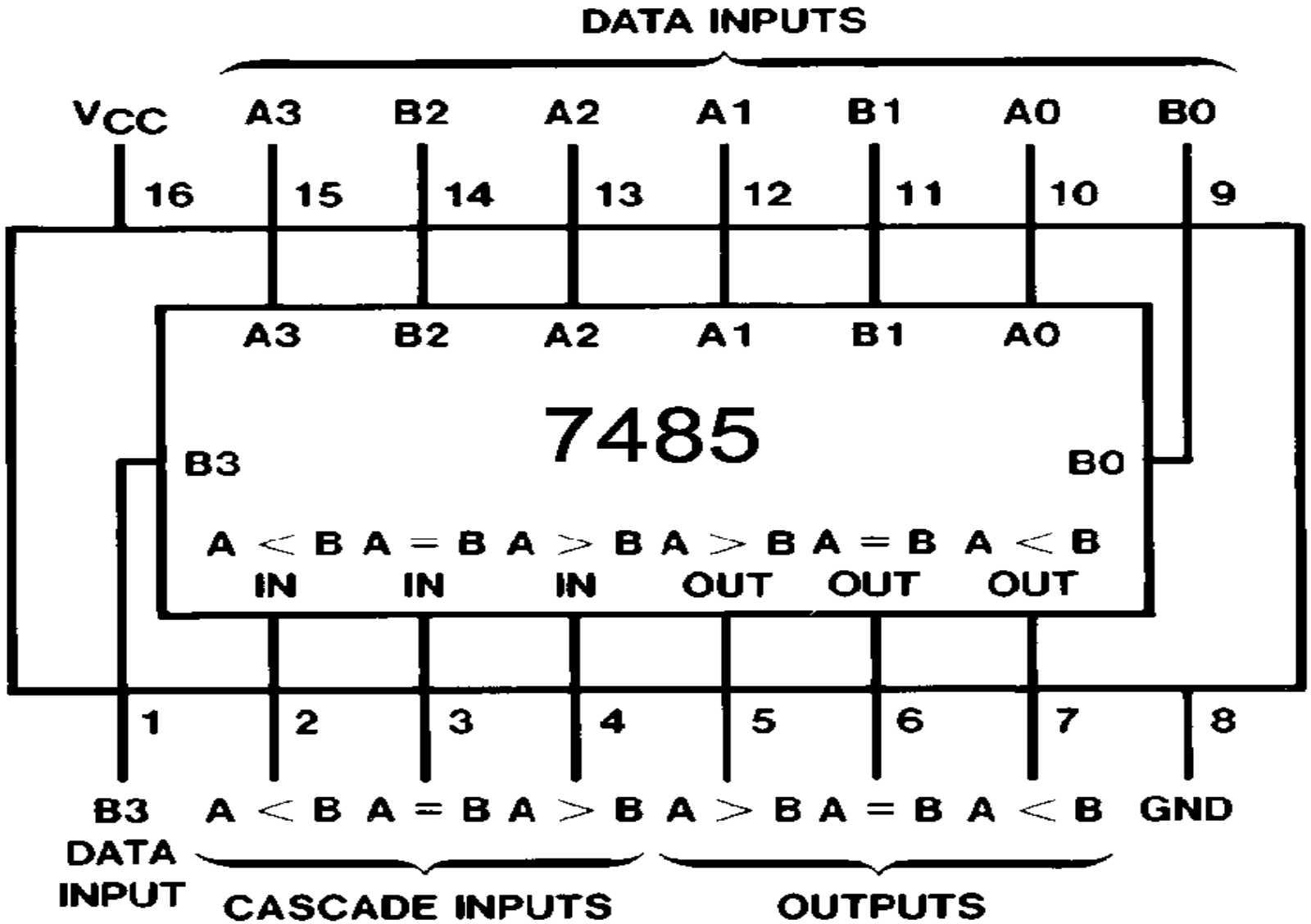
The condition for A<B can be possible in the following four cases:
If A3 = 0 and B3 = 1
If A3 = B3 and A2 = 0 and B2 = 1
If A3 = B3, A2 = B2 and A1 = 0 and B1 = 1
If A3 = B3, A2 = B2, A1 = B1 and A0 = 0 and B0 = 1

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

Schematic of 4-Bit Digital Comparator

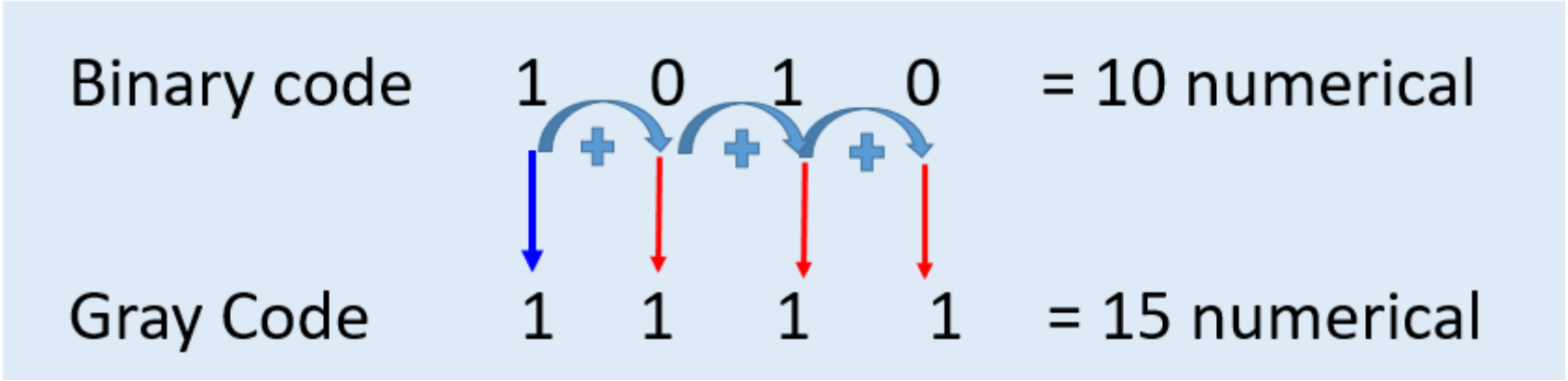


IC 7485: 4-Bit Digital Comparator



4-Bit Binary to gray and vice versa conversion

The most significant bit (MSB) of the Gray code is always equal to the MSB of the given binary code other bits of the output Gray code can be obtained by Xoring binary code bit at that index and previous index.



Ex-OR Truth Table

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

4-Bit Binary to gray conversion Truth Table

Binary				Gray Code			
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

4-Bit Binary to gray and vice versa conversion

g0

b3,b2 \ b1,b0	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$$g_0 = b_0b_1' + b_1b_0' = b_0 \oplus b_1$$

g1

b3,b2 \ b1,b0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

$$g_1 = b_2b_1' + b_1b_2' = b_1 \oplus b_2$$

g2

b3,b2 \ b1,b0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

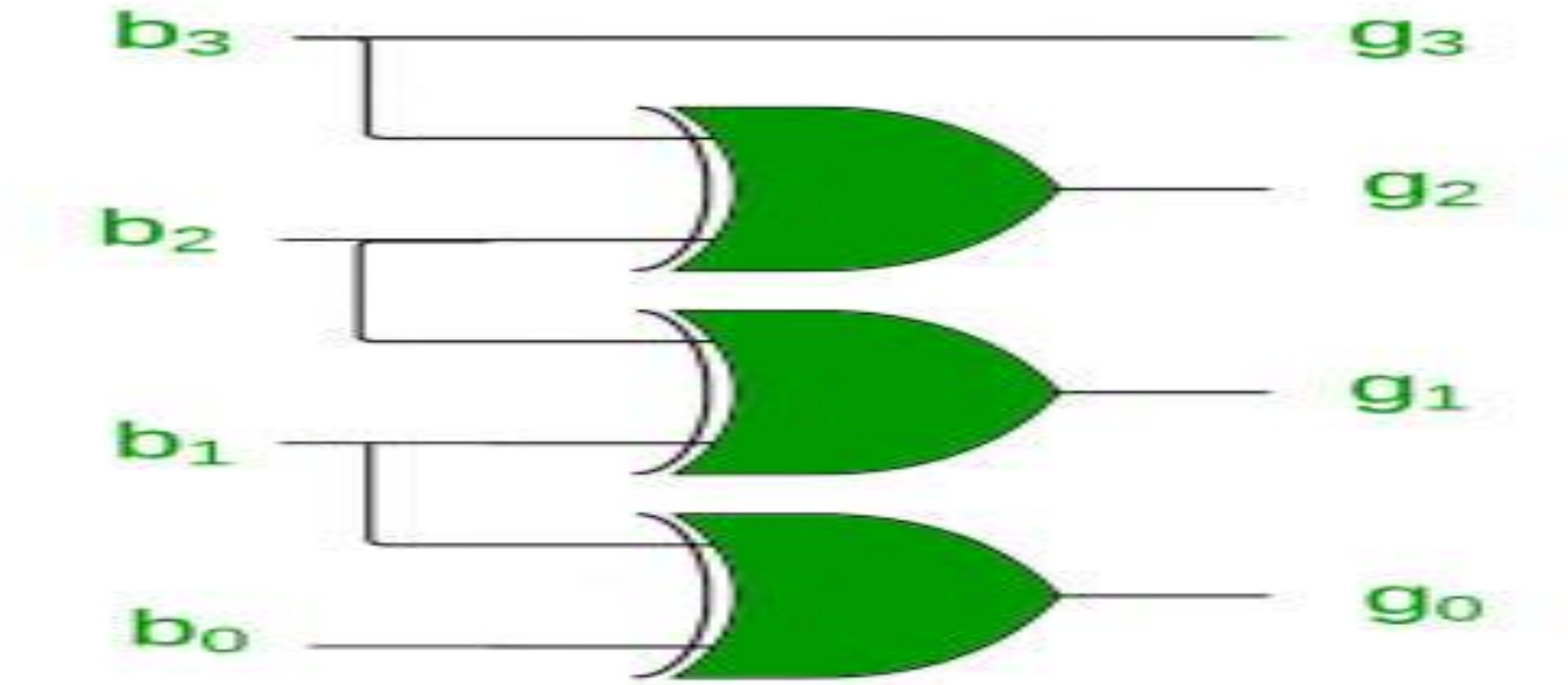
$$g_2 = b_2b_3' + b_3b_2' = b_2 \oplus b_3$$

g3

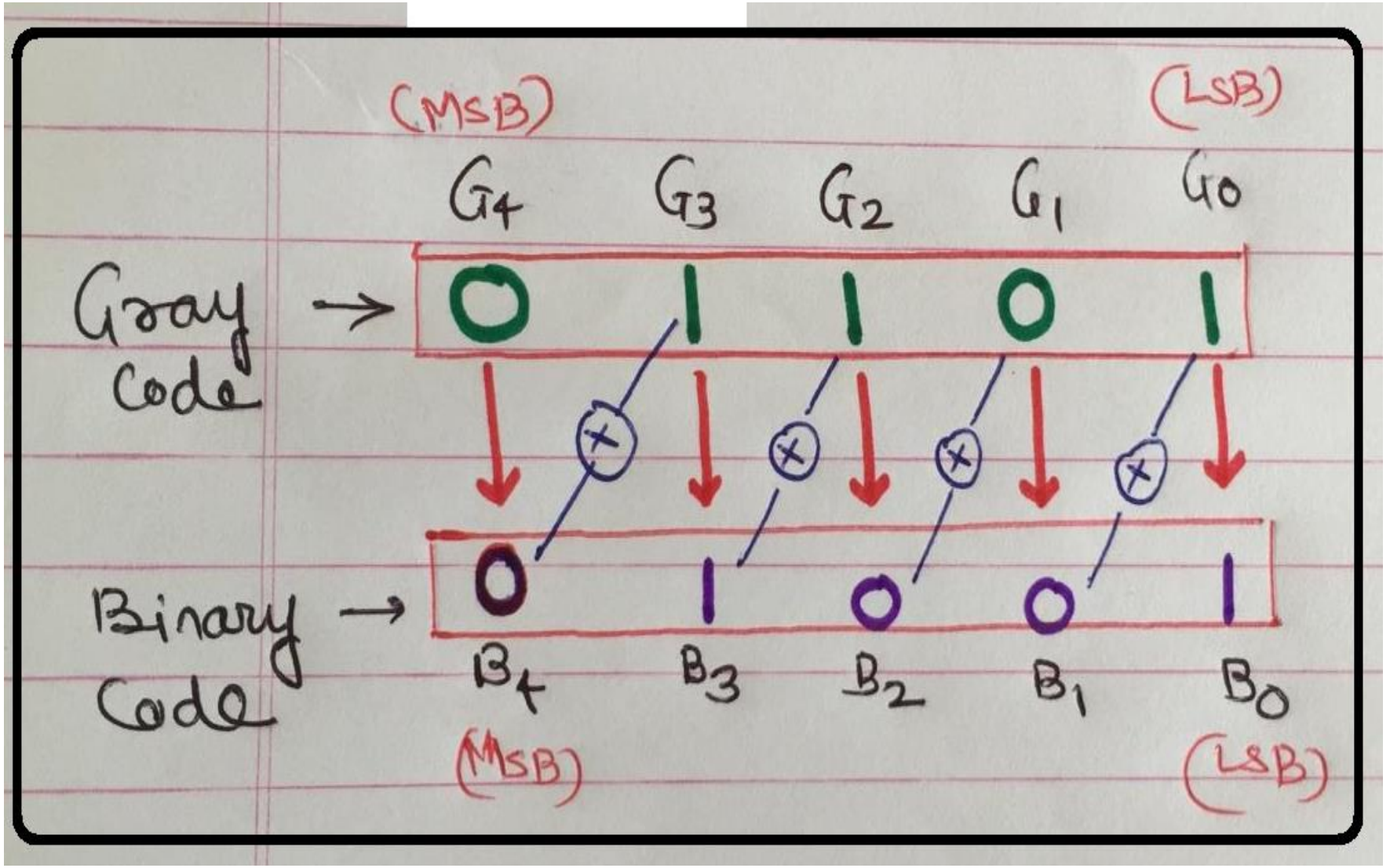
b3,b2 \ b1,b0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$$g_3 = b_3$$

4-Bit Binary to Gray code Circuit diagram



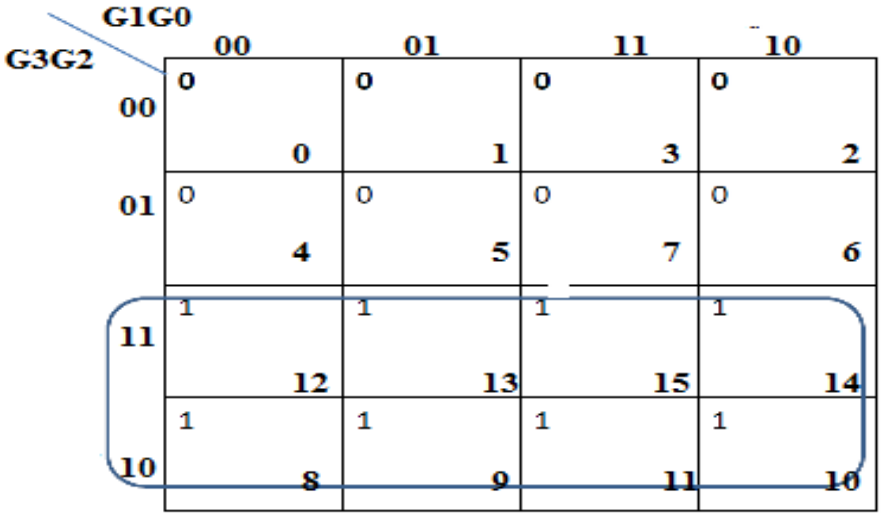
4-Bit Gray to Binary conversion process



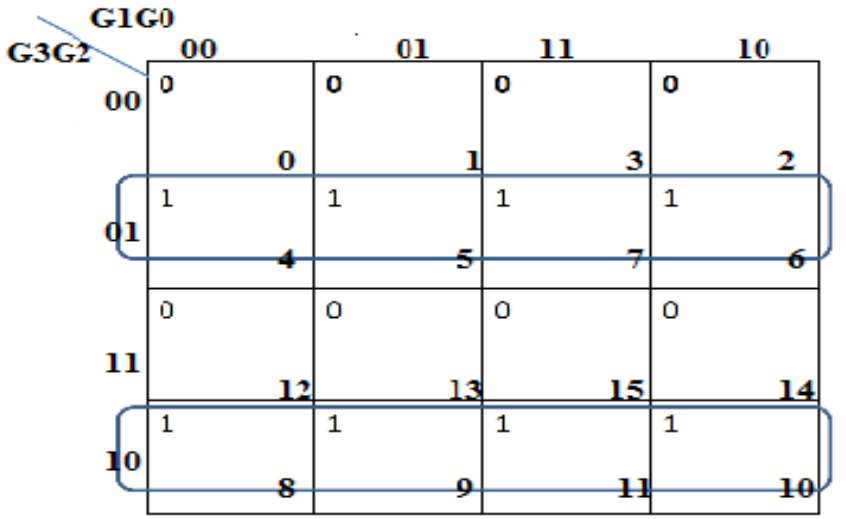
4-Bit Gray to Binary conversion truth table-Method1

Gray Code				Binary			
g_3	g_2	g_1	g_0	b_3	b_2	b_1	b_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	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

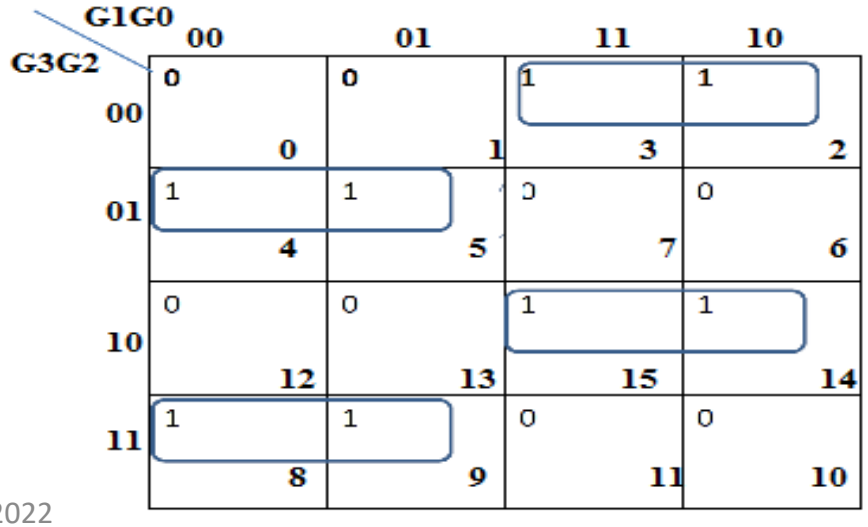
4-Bit Gray to Binary conversion Equation



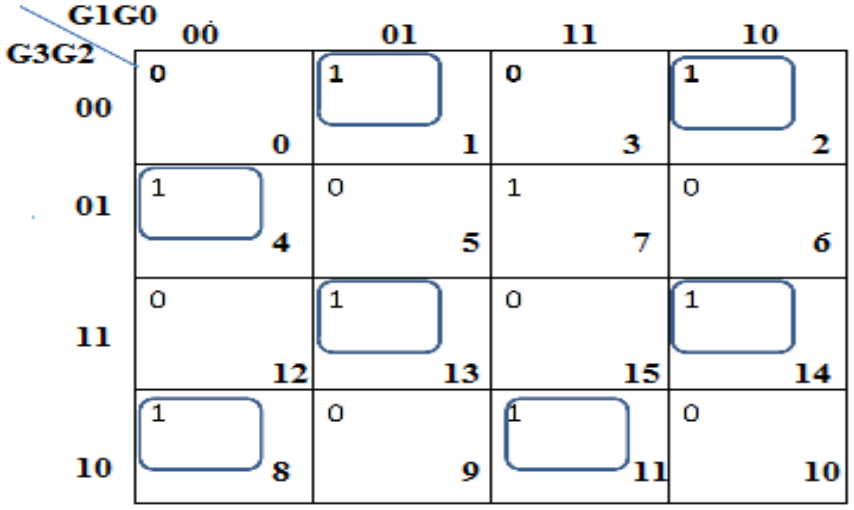
B3=G3



B2=G3⊕G2

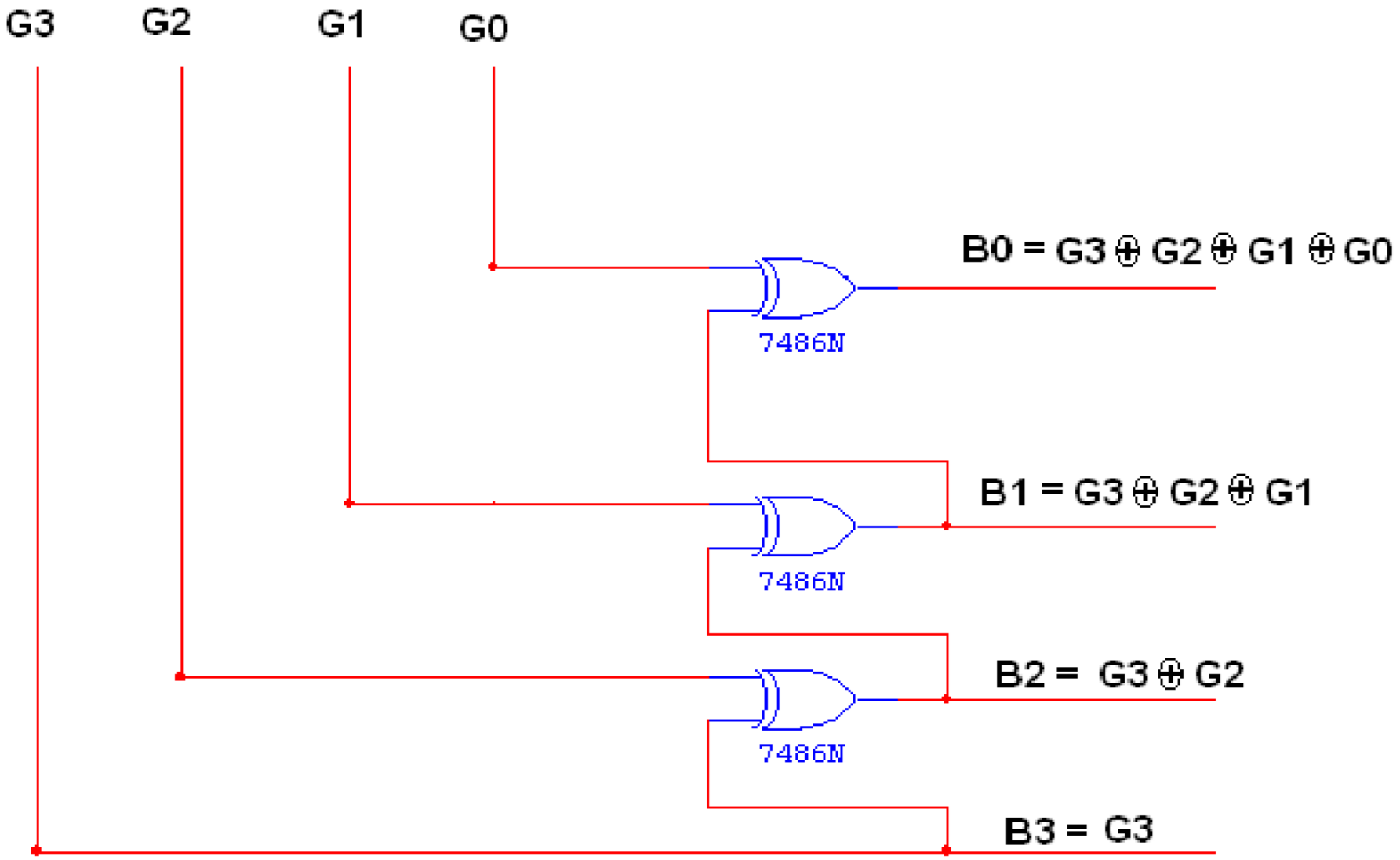


B1=G3⊕G2⊕G1



B0= G3⊕G2⊕G1⊕G0

4-Bit Gray to Binary conversion



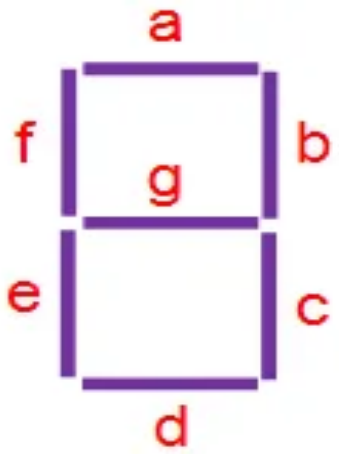
MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

BCD to 7 Segment Decoder

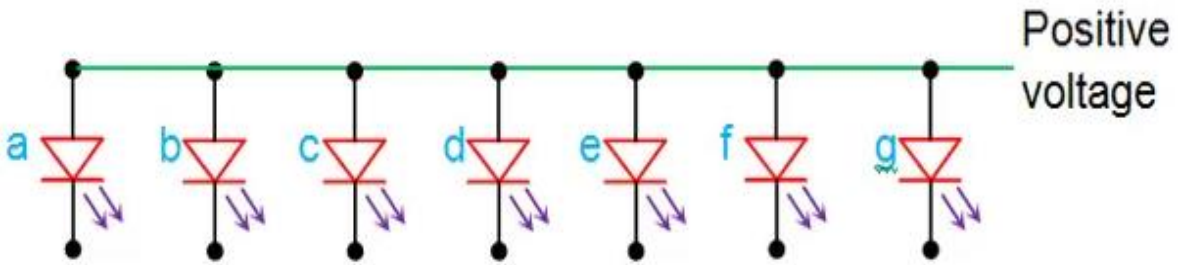
BCD to 7 Segment Decoder

- BCD (Binary Coded Decimal) is an encoding scheme which represents each of the decimal numbers by its equivalent 4-bit binary pattern.

- Seven segment displays comprise of seven individual segments formed by either Light Emitting Diodes (LEDs) or Liquid Crystal Displays (LCDs) arranged in a definite pattern



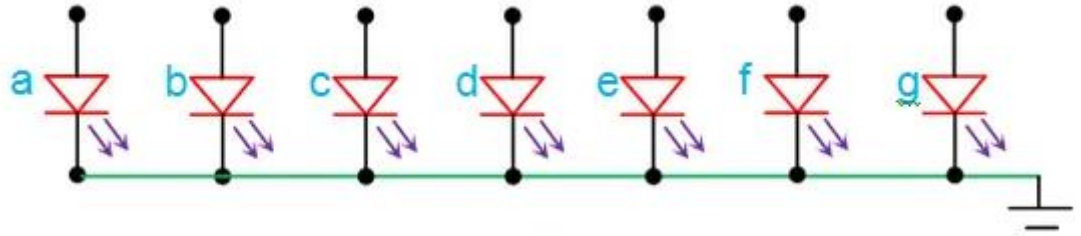
BCD to 7 Segment Decoder



Common Anode

The type of 7-Segment display in which all the anode terminals of 7 LEDs are connected together to form common anode terminal. This terminal should be connected with Vcc or logic ‘1’ during its operation.

To illuminate any of the LED segments we need to provide logic ‘0’ to it.



Common Cathode

In such type of 7-segment display, all the cathodes of the 7 LEDs are connected together to form a common terminal. It should be connected to GND or logic ‘0’ during its operation.

To illuminate any LED of the display, you need to supply logic ‘1’ to its corresponding input pin.

BCD to 7 Segment Decoder

- **BCD to seven segment decoder** is a circuit used to convert the input BCD into a form suitable for the display.
- It has four input lines (A, B, C and D) and 7 output lines (a, b, c, d, e, f and g) as shown.
- Considering common cathode type of arrangement, the truth table for the decoder .

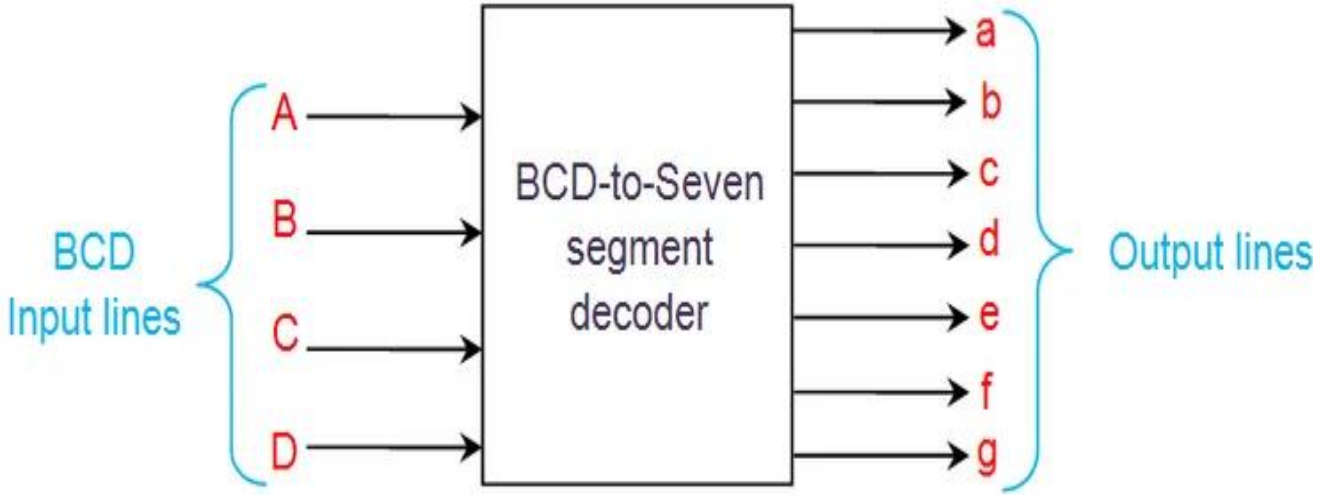
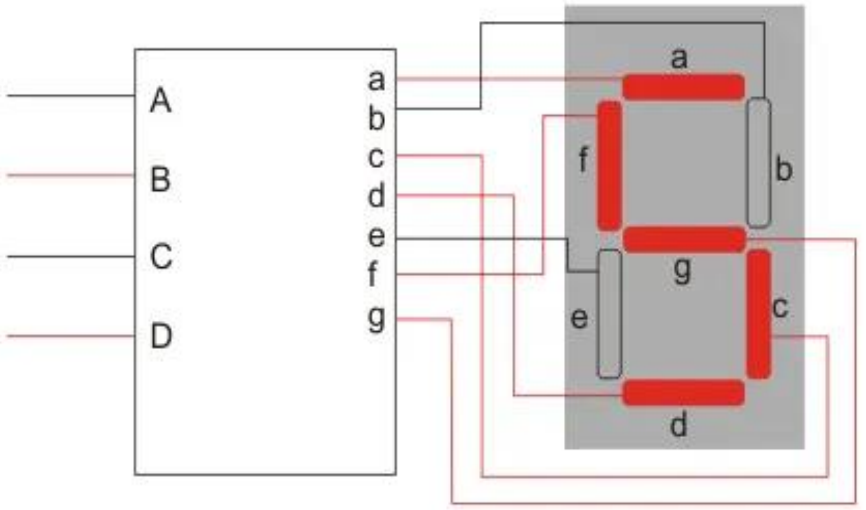


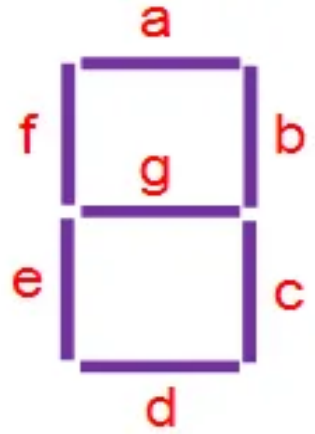
Figure 3 BCD-to-Seven segment decoder



BCD to 7 Segment Decoder

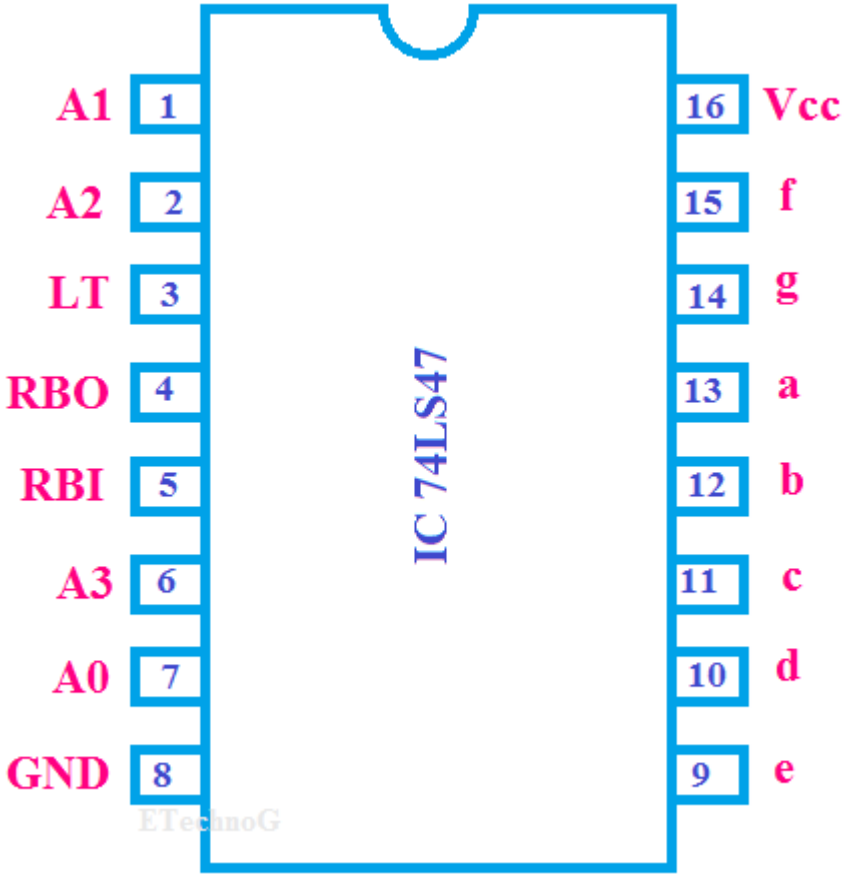
Truth table for common cathode type **BCD to seven segment decoder**

Decimal Digit	Input lines				Output lines							Display pattern
	A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	0	1	1	1	1	1	1	0	
1	0	0	0	1	0	1	1	0	0	0	0	
2	0	0	1	0	1	1	0	1	1	0	1	
3	0	0	1	1	1	1	1	1	0	0	1	
4	0	1	0	0	0	1	1	0	0	1	1	
5	0	1	0	1	1	0	1	1	0	1	1	
6	0	1	1	0	1	0	1	1	1	1	1	
7	0	1	1	1	1	1	1	0	0	0	0	
8	1	0	0	0	1	1	1	1	1	1	1	
9	1	0	0	1	1	1	1	1	0	1	1	



However, it is to be noted that in the case of common anode type, the only change will be to interchange ones and zeros on the table.

BCD to 7 Segment Decoder



- A0, A1, A2, A3 are the BCD input pins.
- LT - for display test
- RBO, RBI is the Ripple blanking output and Ripple blanking Input pins respectively.
- a, b, c, d, e, f, g are the outputs for seven segment display.

Parity Generator / Checker

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**.

- **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**

Table-1 shows the 3-bit message with even parity & odd parity

3-bit message			Odd parity bit	Even parity bit
A	B	C		
0	0	0	1	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Even Parity

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

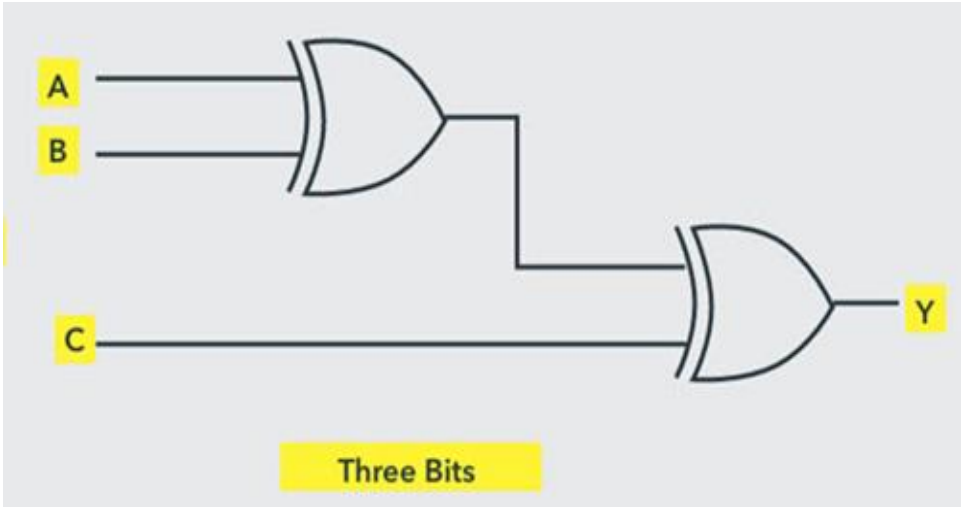
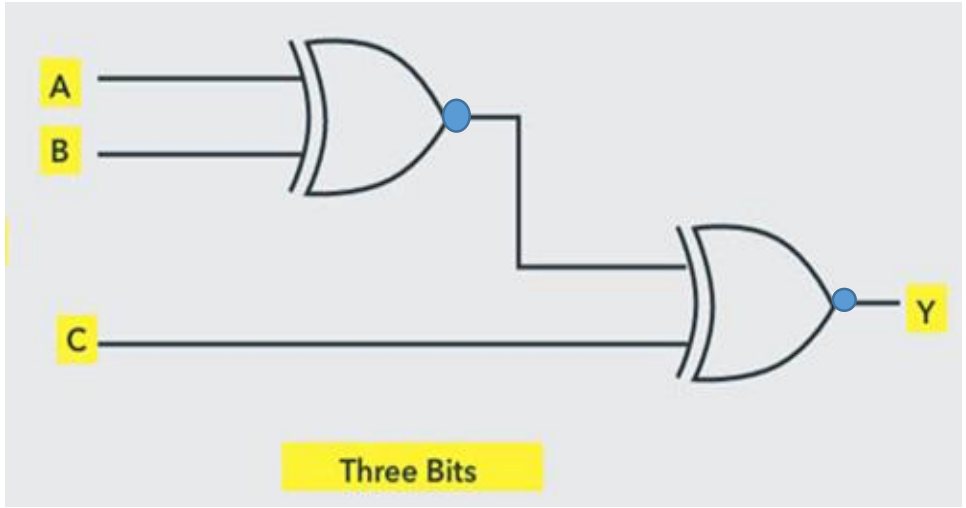


Table-1 shows the 3-bit message with even parity & odd parity

3-bit message			Odd parity bit	Even parity bit
A	B	C		
0	0	0	1	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Odd Parity

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



Parity Detector/ Checker:

- The three bits in the message together with the parity bit are transmitted to the destination, where they are applied to the parity checker circuit.
- The parity checker circuit checks for possible errors in the transmission. Since the information was transmitted with even parity , the four bits received must have an even number of 1's.
- An error occurs during the transmission if the four bits received have an odd number of 1's, indicating that one bit has changed in value during transmission.
- The output of the parity checker is denoted by PEC (Parity Error Check).
- It will be equal to 1 , if an error occurs , that is if the four bits received have an odd number of 1's.

Parity Detector/ Checker:

Truth Table for Even Parity Checker

Four Bits Received				Parity Error Check
A	B	C	D	PEC
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Parity Detector/ Checker:

Even parity checker.

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

$$PEC = \overline{A}\,\overline{B}(\overline{C}\,D + C\,\overline{D}) + \overline{A}\,B\,(\overline{C}\,\overline{D} + C\,D)$$
$$+ A\,\overline{B}(\overline{C}\,D + C\,\overline{D}) + A\,B\,(\overline{C}\,\overline{D} + C\,D)$$

$$= \overline{A}\,\overline{B}(C \oplus D) + \overline{A}\,B\,\overline{(C \oplus D)}$$
$$+ A\,\overline{B}(C \oplus D) + A\,B\,\overline{(C \oplus D)}$$

$$= (\overline{A}\,\overline{B} + A\,B)(C \oplus D) + (\overline{A}\,B + A\,\overline{B})\,\overline{(C \oplus D)}$$

$$= \overline{(A \oplus B)}(C \oplus D) + (A \oplus B)\,\overline{(C \oplus D)}$$

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

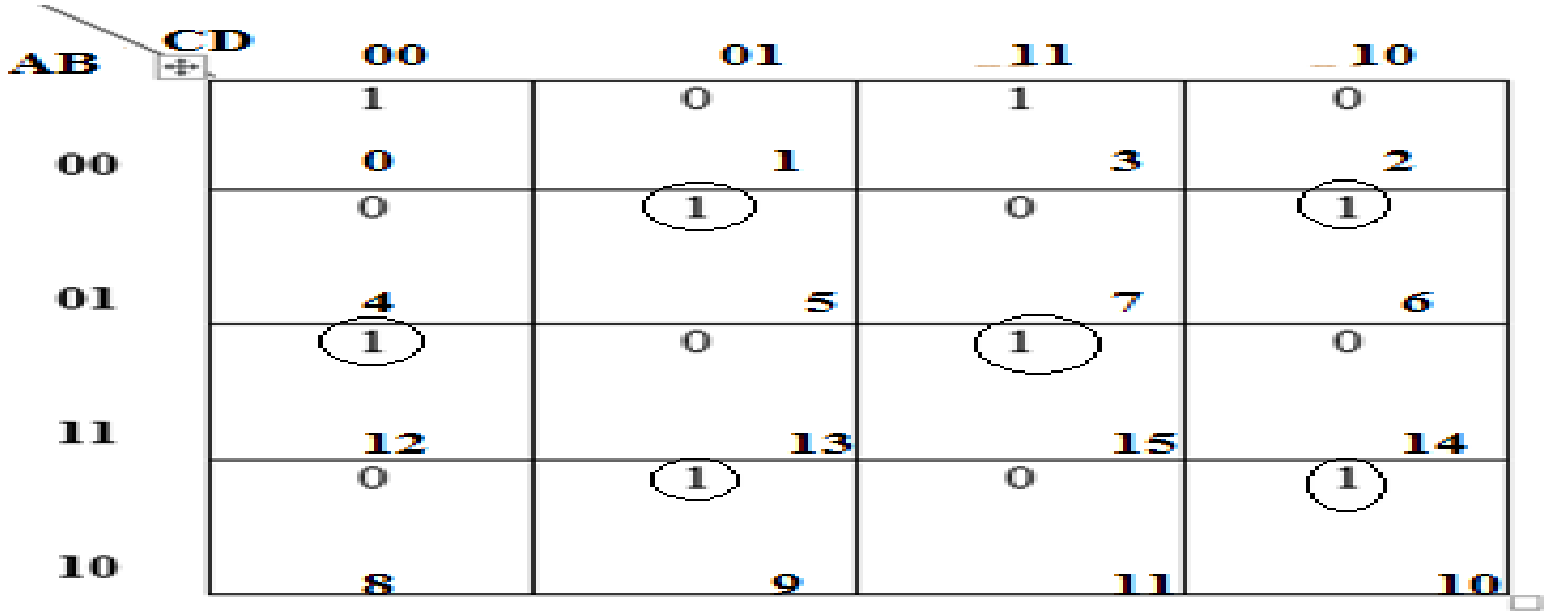
Parity Detector/ Checker:

Truth table for the Odd parity checker.

Four bits received				Parity Error Check
A	B	C	D	PEC
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

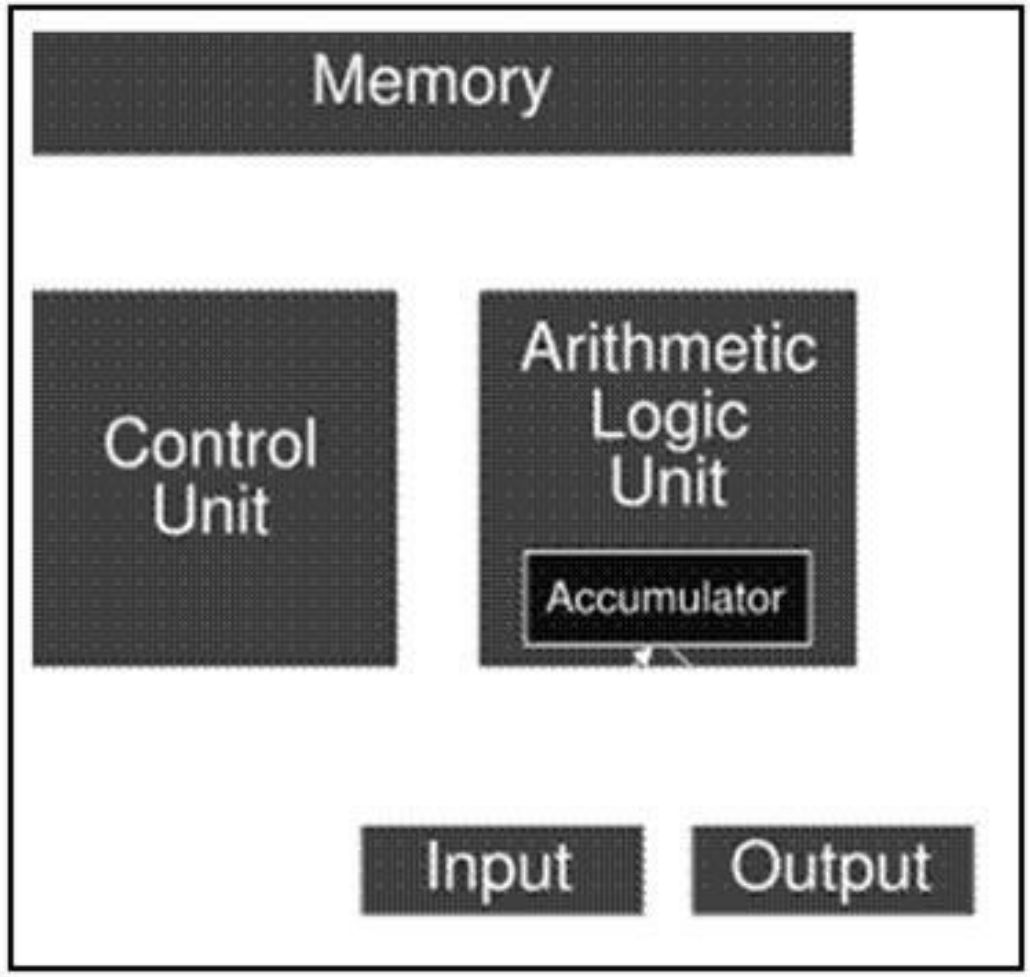
Parity Detector/ Checker:

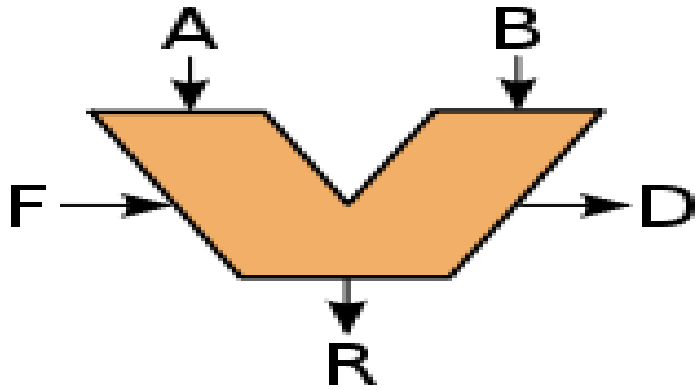
K-Map for the Odd parity checker.



$$PEC = \overline{A}\,\overline{B}\,(\overline{C}\,\overline{D} + C\,D) + \overline{A}\,B\,(\overline{C}\,D + C\,\overline{D})$$
$$+A\,\overline{B}\,(\overline{C}\,D + C\,\overline{D})\;A\,B\,(\overline{C}\,\overline{D} + C\,D)$$
$$= \overline{A}\,\overline{B}\,(\overline{C \oplus D}) + \overline{A}\,B\,(C \oplus D) + A\,\overline{B}\,(C \oplus D) + A\,B\,(\overline{C \oplus D})$$
$$= (A \oplus B)(C \oplus D) + (\overline{A \oplus B})(\overline{C \oplus D})$$
$$= \overline{(A \oplus B) \oplus (C \oplus D)}$$

Arithmetic Logic Unit





The ALU is an extremely versatile and useful device since, it makes available, in single package, facility for performing many different logical and arithmetic operations.

Arithmetic Logic Unit (ALU) is a critical component of a microprocessor and is the core component of central processing unit.

Arithmetic Logic Unit

ALU's comprise the combinational logic that implements logic operations such as AND, OR and arithmetic operations, such as ADD, SUBTRACT.

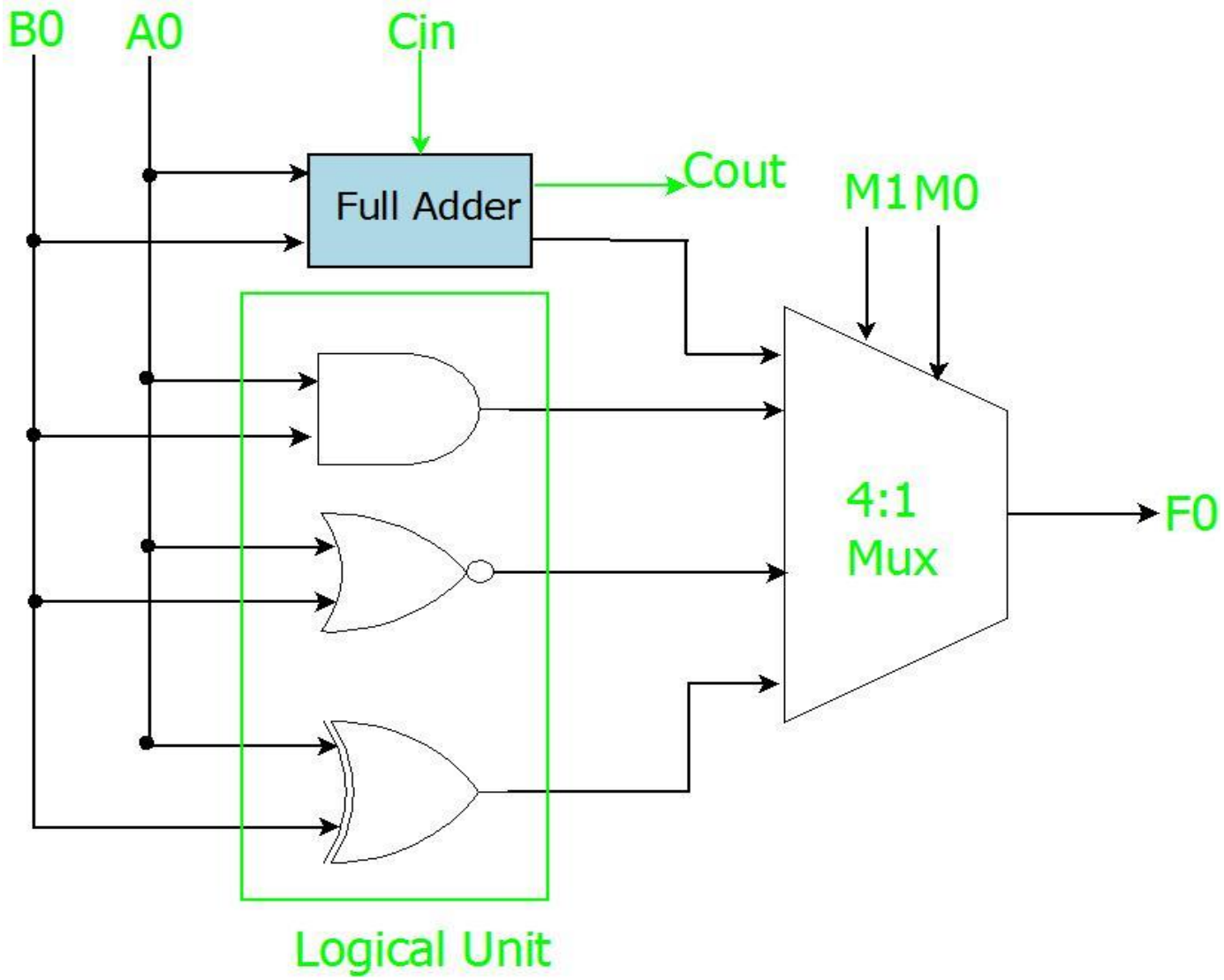
lets construct a simple ALU that performs a arithmetic operation (1 bit addition)and does 3 logical operations namely AND, NOR and XOR

The multiplexer selects only one operation at a time.

The operation selected depends on the selection lines of the multiplexer as shown in the truth table.

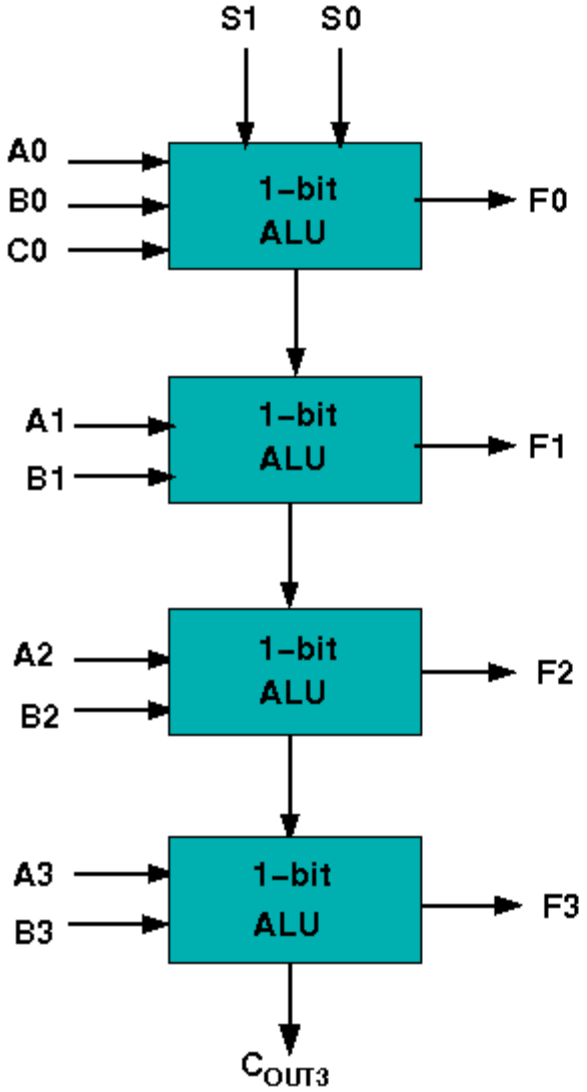
Arithmetic Logic Unit

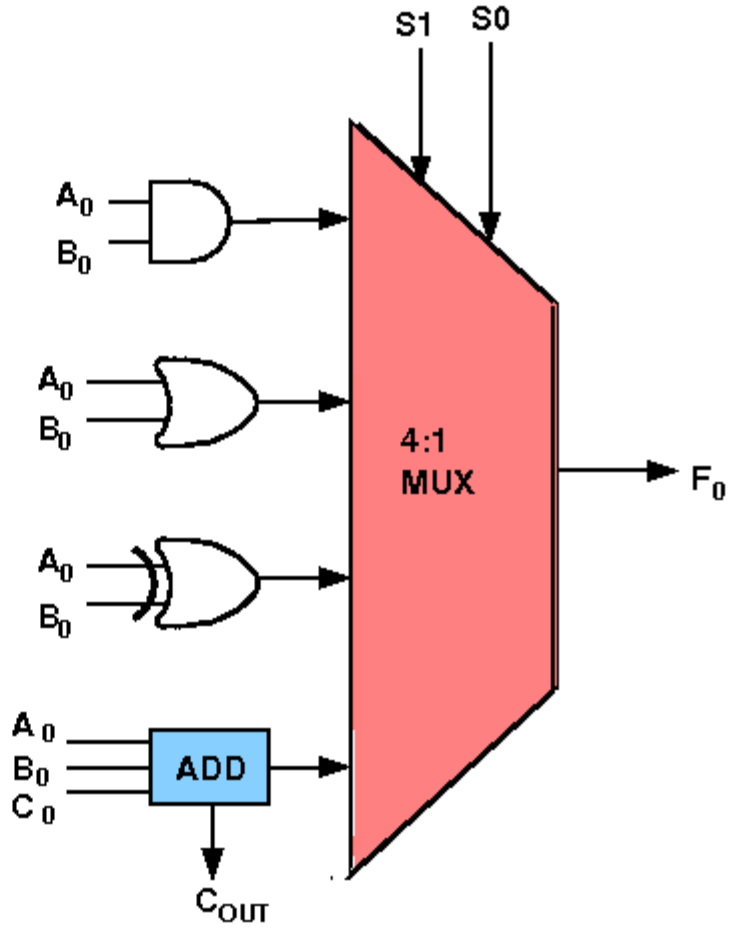
M1	M0	Operations
0	0	SUM
0	1	OR
1	0	AND
1	1	EXOR



Arithmetic Logic Unit

Four 1 bit ALU's can be concatenated to design 4 bit ALU



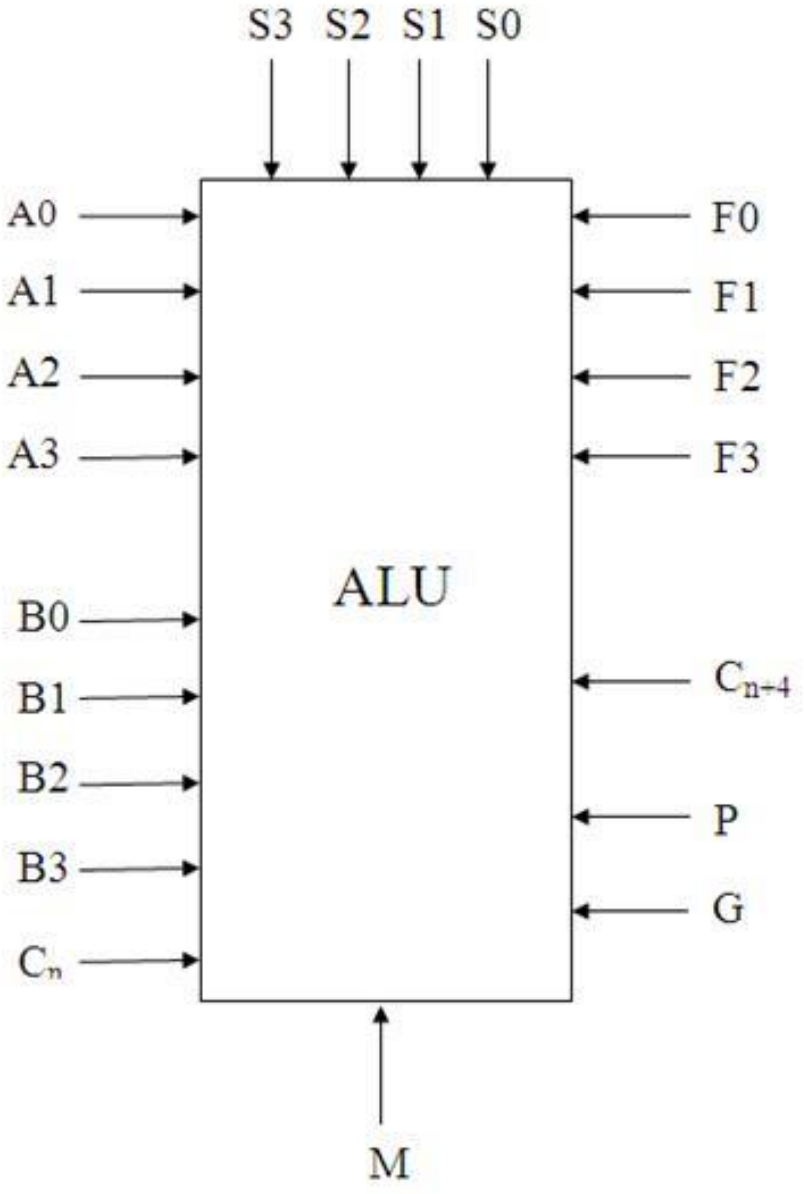


Arithmetic Logic Unit

The ALU IC has more functions as compared to 1 bit ALU
The diagram shows 4 bit ALU having capability to performing logical as well as arithmetic operations.

Controlled by the four function select inputs (S0 to S3) and the mode control input (M), ALU can perform all the 16 possible logic operations or 16 different arithmetic operations on active HIGH or active LOW operands.

It is designed using look ahead carry capability



Arithmetic Logic Unit

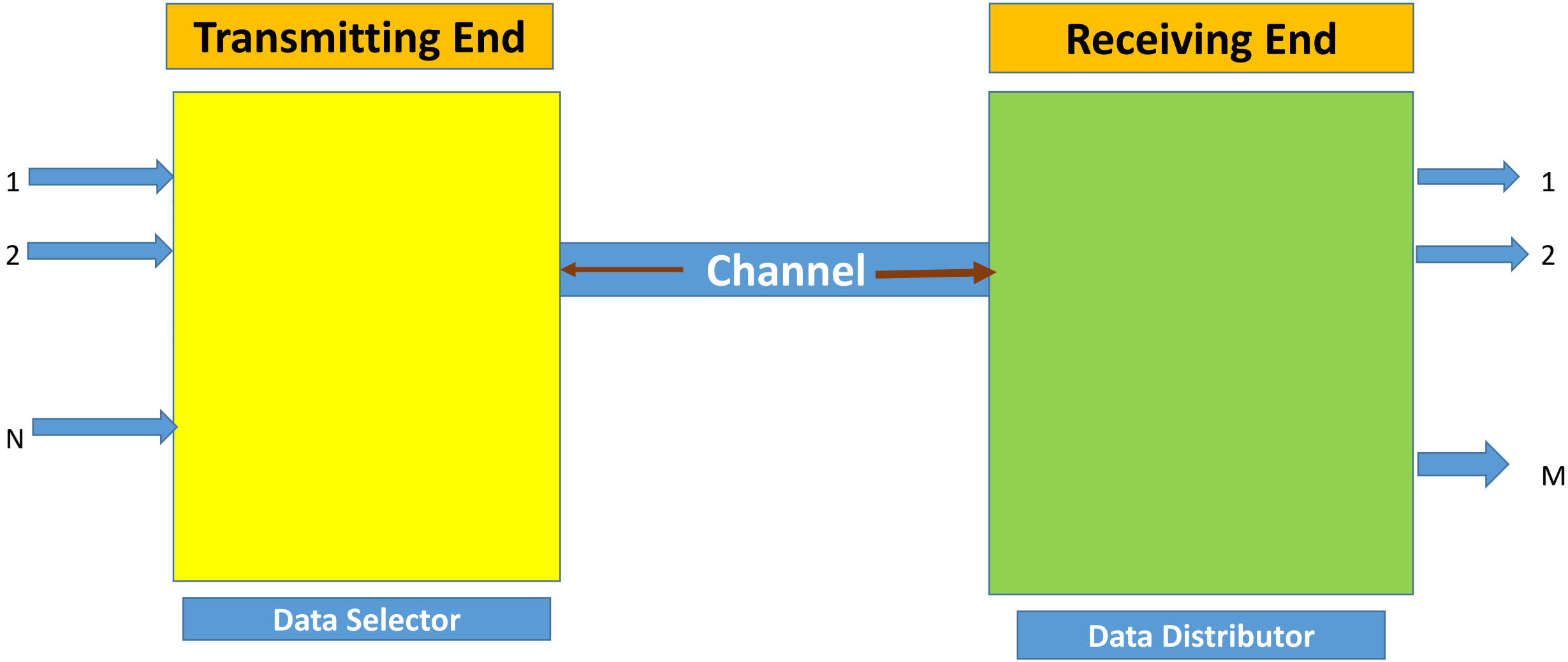
When the mode control input (M) is HIGH, all internal carries are inhibited and the device performs logic operations on the individual bits.

When M is LOW, the carries are enabled and the ALU performs arithmetic operations on the two 4-bit words.

MODE SELECT INPUTS				ACTIVE HIGH INPUTS AND OUTPUTS	
S_3	S_2	S_1	S_0	LOGIC (M=H)	ARITHMETIC$^{(2)}$ (M=L; C_n=H)
L	L	L	L	$\overline{A}$	A
L	L	L	H	$\overline{A + B}$	A + B
L	L	H	L	$\overline{A}B$	A + $\overline{B}$
L	L	H	H	logical 0	minus 1
L	H	L	L	$\overline{AB}$	A plus A$\overline{B}$
L	H	L	H	$\overline{B}$	(A + B) plus A$\overline{B}$
L	H	H	L	A ⊕ B	A minus B minus 1
L	H	H	H	A$\overline{B}$	A$\overline{B}$ minus 1
H	L	L	L	$\overline{A}$ + B	A plus AB
H	L	L	H	$\overline{A \oplus B}$	A plus B
H	L	H	L	B	(A + $\overline{B}$) plus AB
H	L	H	H	AB	AB minus 1
H	H	L	L	logical 1	A plus A$^{(1)}$
H	H	L	H	A + $\overline{B}$	(A + B) plus A
H	H	H	L	A + B	(A + $\overline{B}$) plus A
H	H	H	H	A	A minus 1

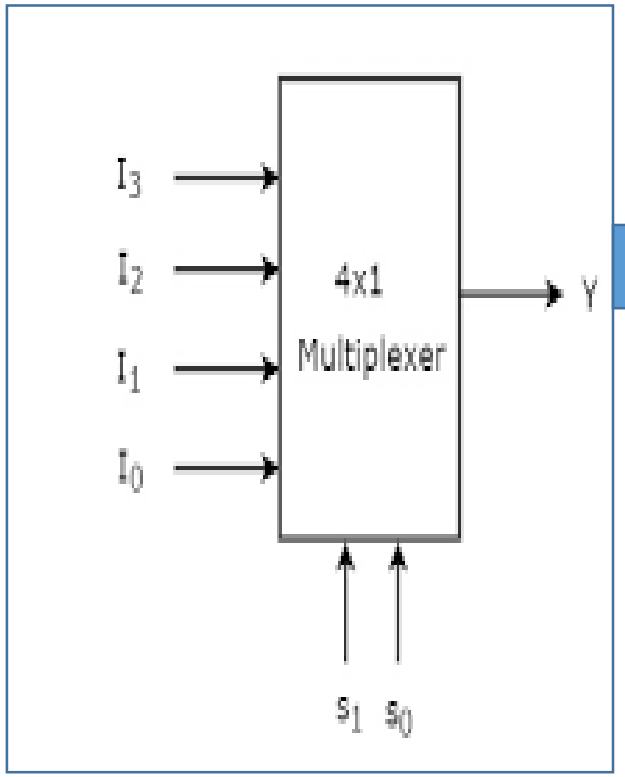
Multiplexer and Demultiplexer

Communication System



Communication System

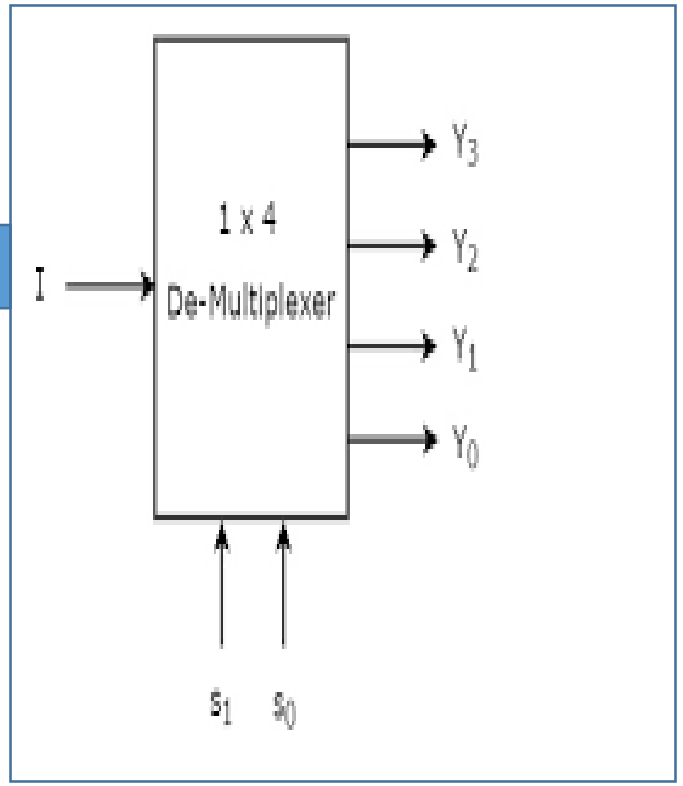
Transmitting End



Data Selector

Channel

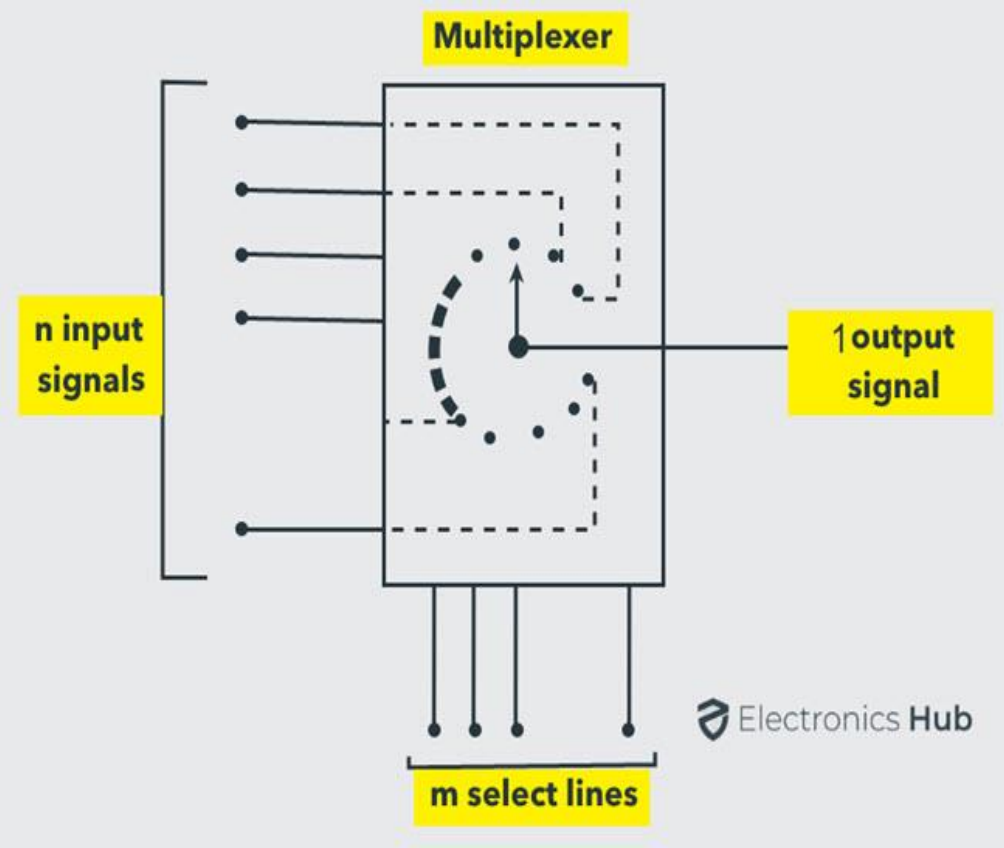
Receiving End

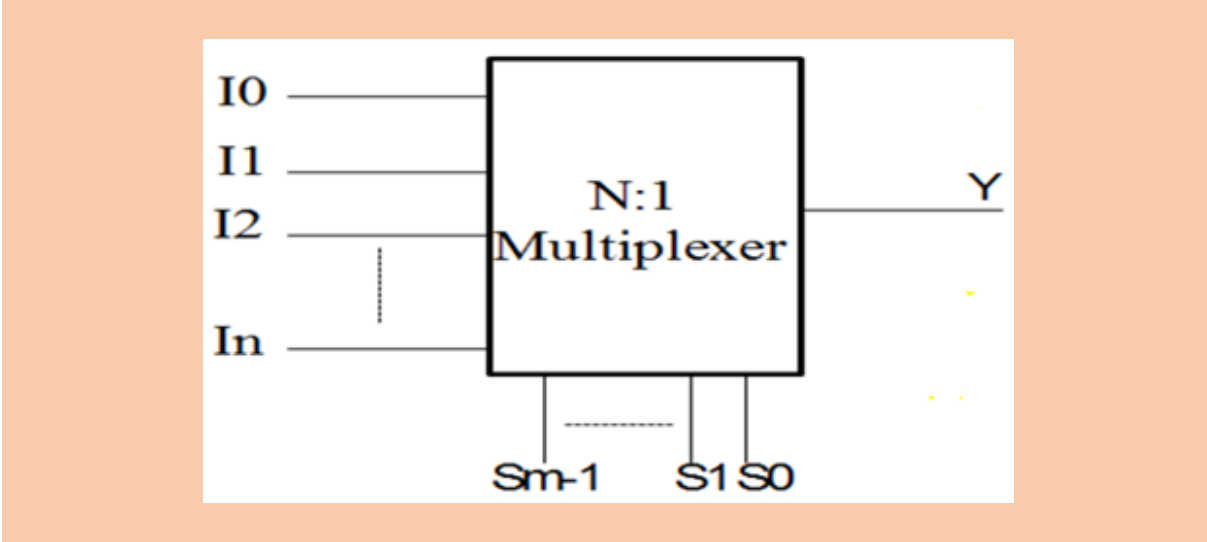


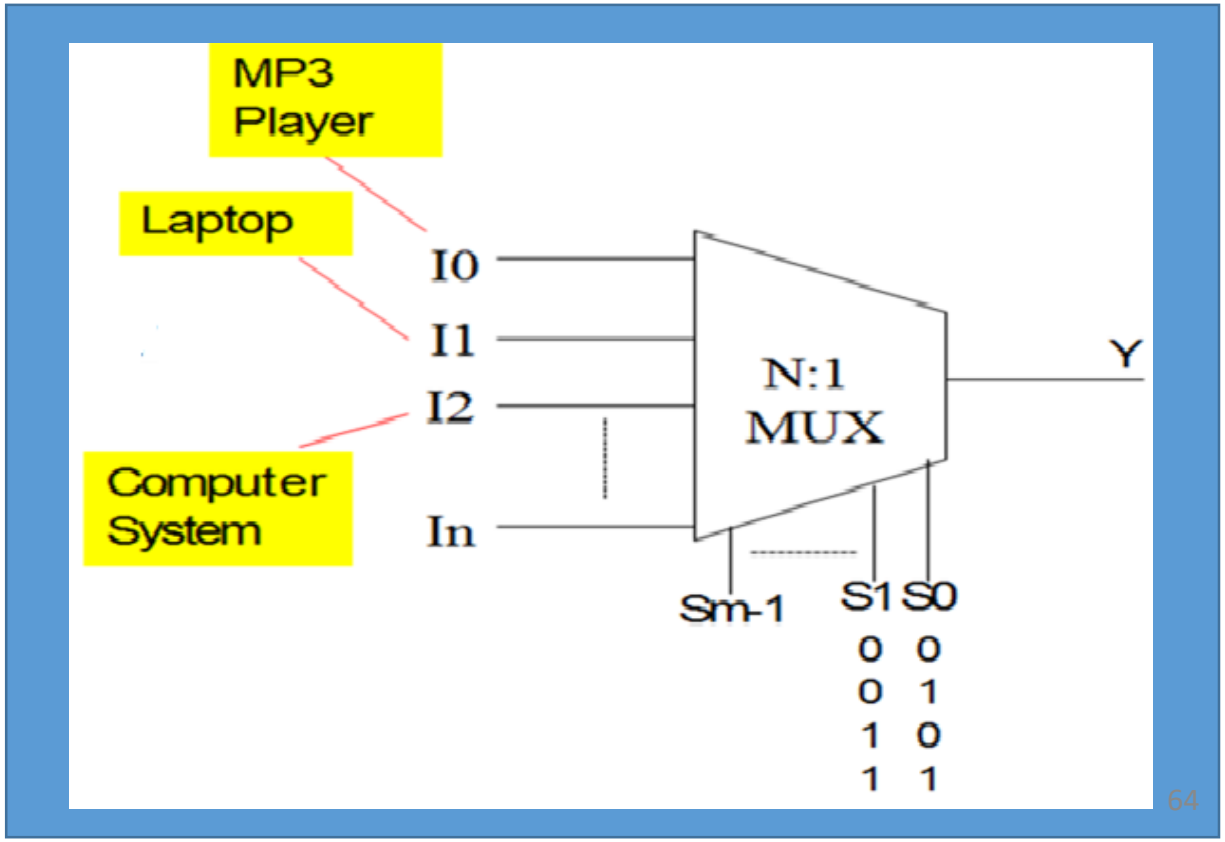
Data Distributor

Multiplexer

Data Selector or Many to one circuit







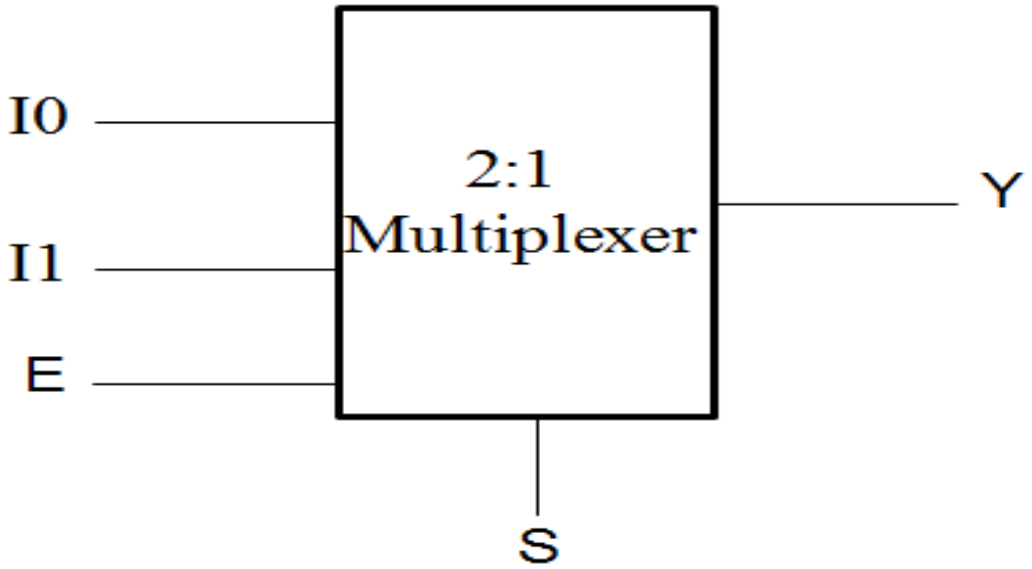
Multiplexer

- **Advantages of Mux**
  - Reduces the number of wires
  - Reduces circuit complexity & cost
  - Minimize the IC package count

Types

- 2:1 MUX
- 4:1 MUX
- 8:1 MUX
- 16:1 MUX

2:1 Multiplexer



Inputs		Output
E	S	Y
0	X	0
1	0	I0
1	1	I1

Relation between selector and Inputs

n=2^m

$Y = E.\overline{S}.I0 + E.S.I1$

$Y = E\,(\overline{S}.I0 + S.I1)$

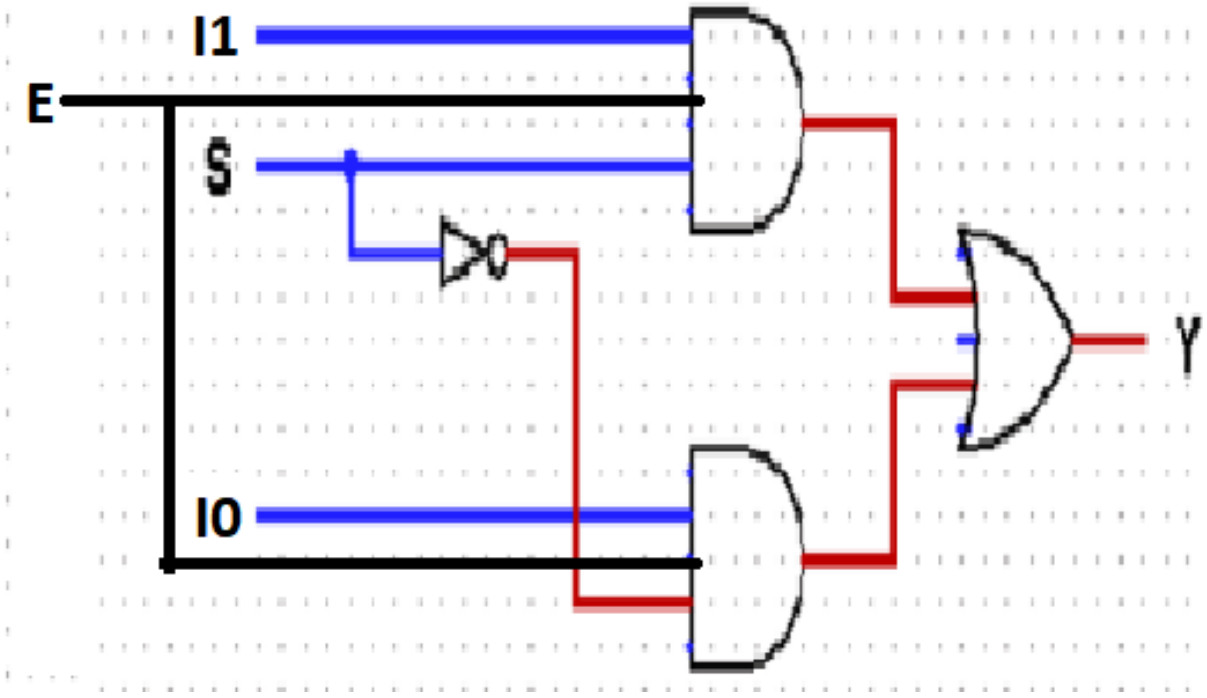
Where, n = no. of inputs

m = no. of select lines => $m=\log_2 n$

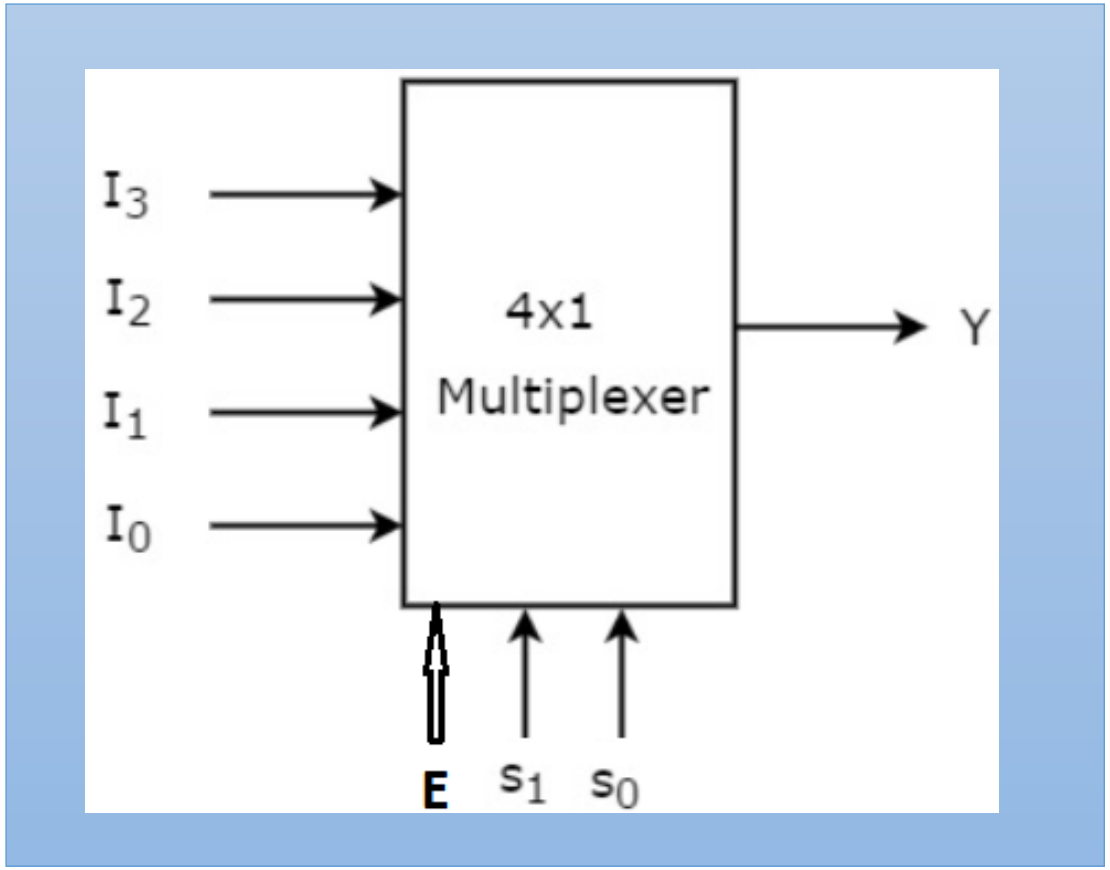
2:1 Multiplexer Circuit

Gate level circuit diagram of 2:1 Multiplexer.

$$Y = E.\overline{S}.I0 + E.S.I1 \qquad Y = E\,(\overline{S.I0} + S.I1)$$



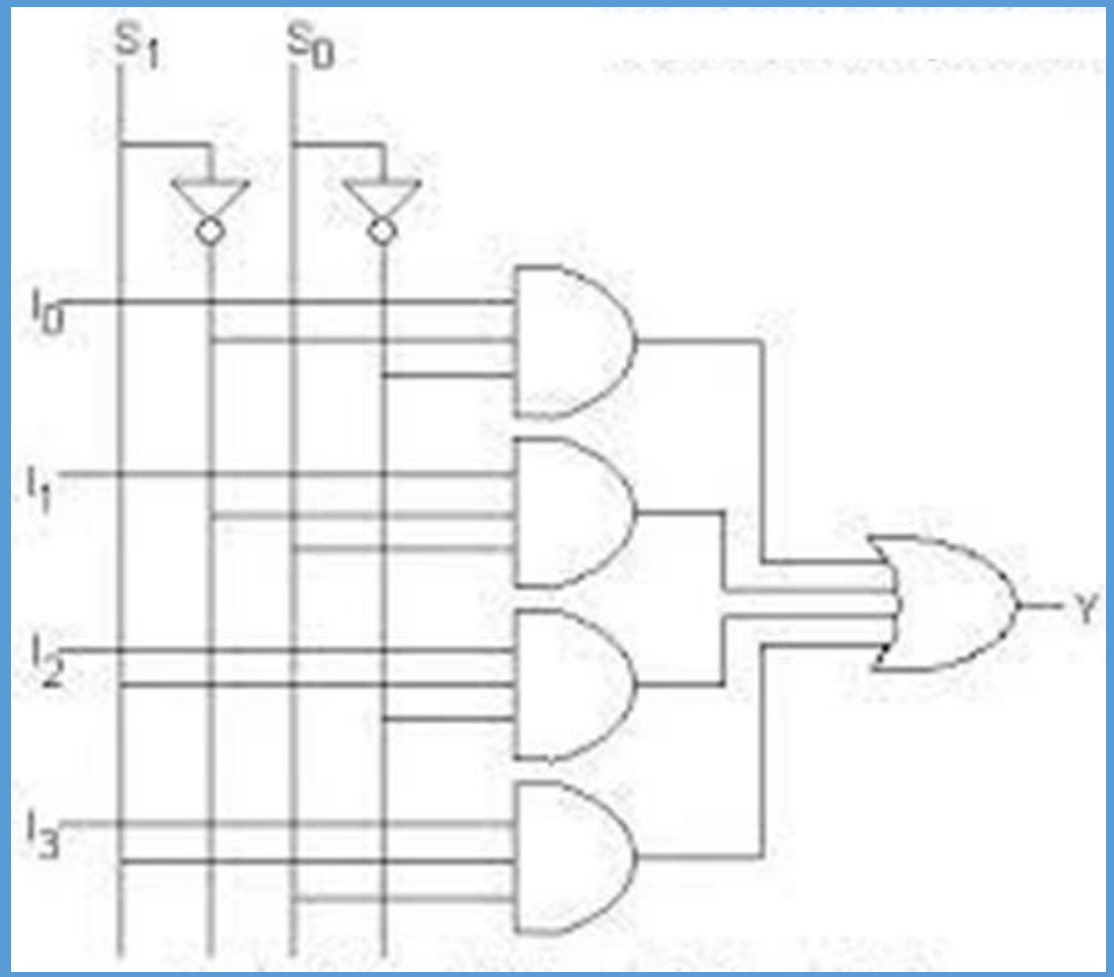
4:1 Multiplexer



Inputs			Output
E	**S1**	**S0**	**Y**
0	X	X	0
1	0	0	I0
1	0	1	I1
1	1	0	I2
1	1	1	I3

$$Y = \overline{S1}.\overline{S0}.I0 + \overline{S1}.S0.I1 + S1.\overline{S0}.I2 + S1.S0.I3$$

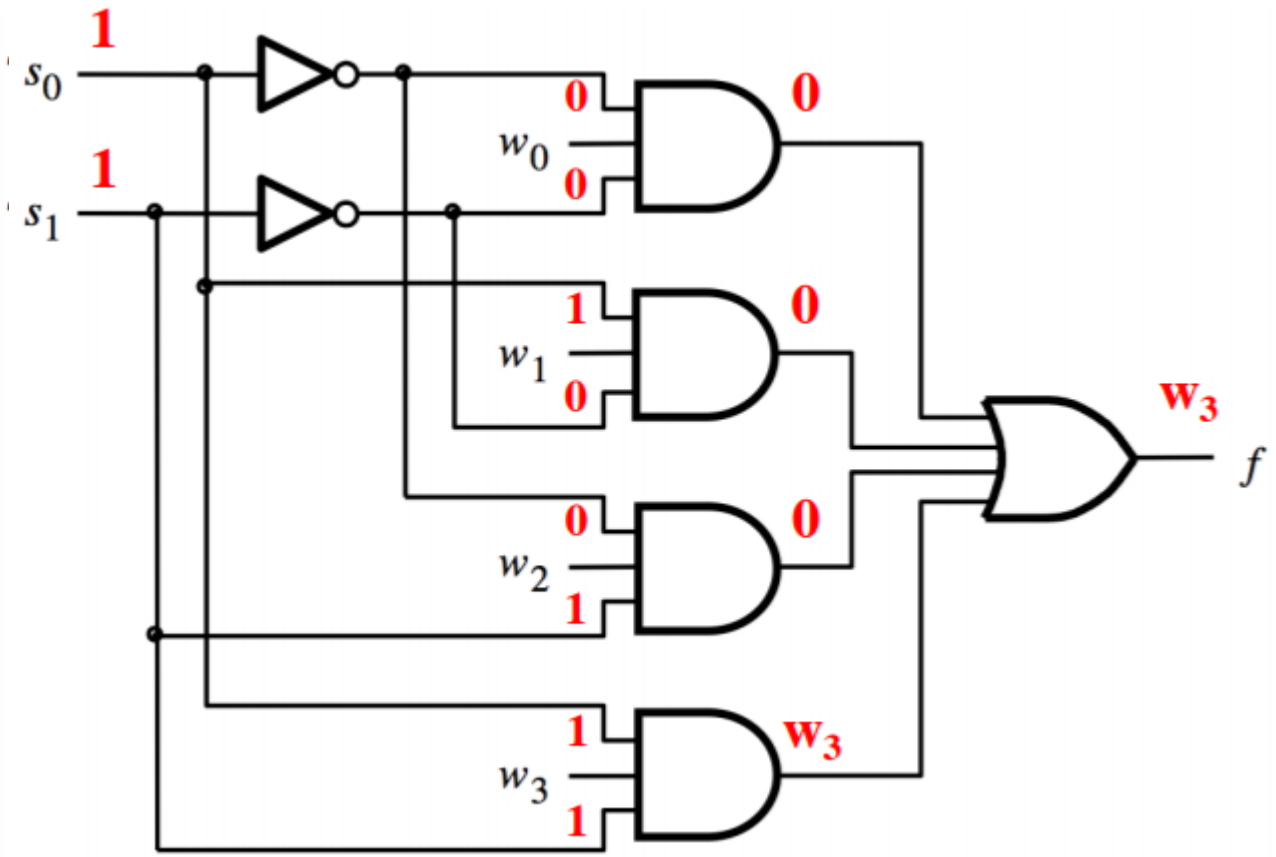
4:1 Multiplexer



$$Y = \overline{S1}.\overline{S0}.I0 + \overline{S1}.S0.I1 + S1.\overline{S0}.I2 + S1.S0.I3$$

Inputs		Output
S1	**S0**	**Y**
0	0	I0
0	1	I1
1	0	I2
1	1	I3

4:1 Multiplexer



Implementation of Logic expressions using Multiplexers

Like Logic Gates, the multiplexers can also be used to implement any Boolean expression.

In method 1, **type of mux can be decided by the given number of variables.**

1.The first step is to select the multiplexer. If the given expression has n variables, then determine the multiplexer using the formula $2^n : 1$.

2.Connect the inputs, that correspond to the given minterms to logic 1.

3.Connect all the other inputs to the ground(logic 0).

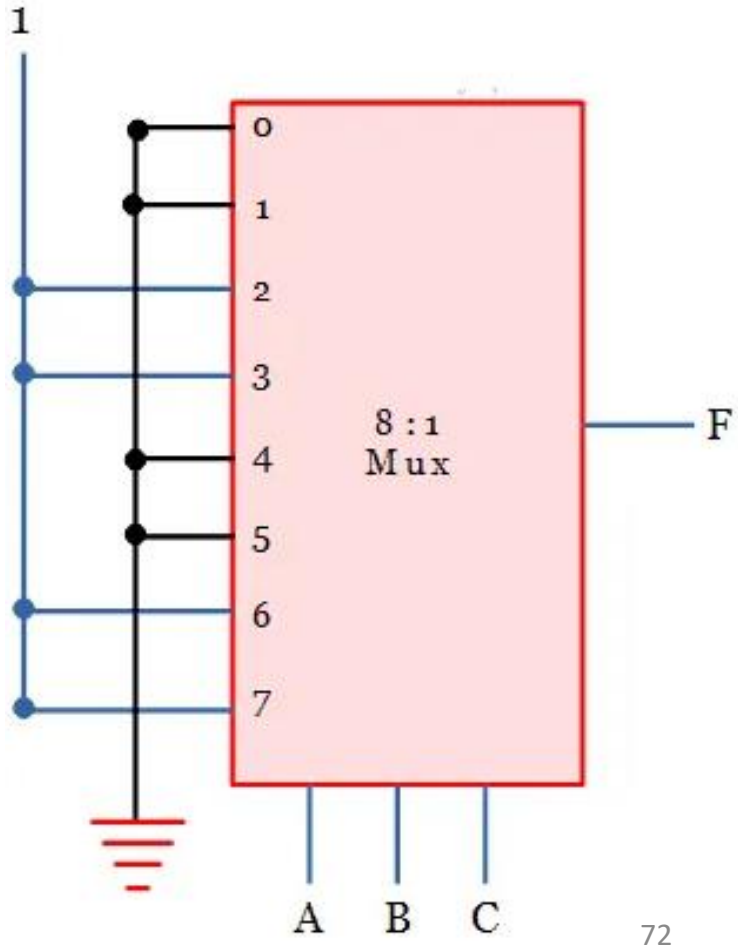
4.Connect the input variables(A, B, C….) as the selection lines.

Implementation of Logic expressions using Multiplexers

Implement the boolean expression F(A, B, C) = ∑ m(2, 3, 6, 7) using a multiplexer.

Solution:

- There are 3 variables in the given expression, hence $2^n = 2^3 = 8 : 1$ multiplexer.
- So, the mux has 8 input lines, 3 selection lines, and one output.
- The inputs, corresponding to the minterms (2, 3, 6, 7) are connected to the logic 1 and remaining terms to the logic 0(grounded).
- The given input variables are connected as three selection lines.

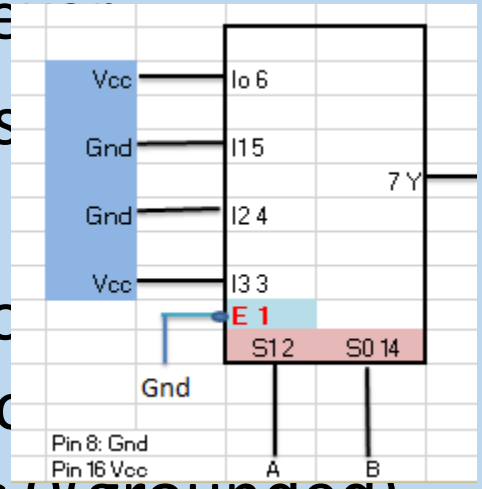


Implementation of Logic expressions using Multiplexers

Implement the Boolean expression F(A, B) = ∑ m(0, 3) using a multiplexer.

Solution:

- There are 2 variables in the given expression, hence $2^n = 2^2 = 4 : 1$ multiple
- So, the mux has 4 input lines lines, and one output.
- The inputs, corresponding to (0, 3) are connected to the lo remaining terms to the logic 0(grounded).
- The given input variables are connected as two selection lines.

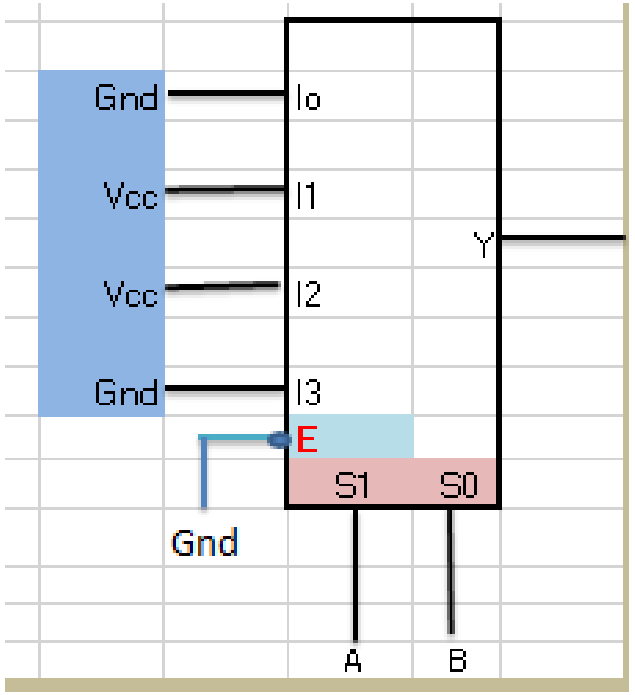


Implementation of Logic expressions using Multiplexers

Implement the boolean expression F(A, B) = π M(0, 3) using a multiplexer.

Solution:

- There are 2 variables in the given expression, hence $2^n = 2^2 = 4 : 1$ multiplexer.
- So, the mux has 4 input lines, 2 selection lines, and one output.
- The inputs, corresponding to the maxterms (0, 3) are connected to the logic 0 and remaining terms to the logic 1.
- The given input variables are connected as two selection lines.



Implement Circuit using MUX

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

Implement Boolean Equation (SOP) using MUX

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

As there are 4 variables, multiplexer having four select lines is required.

Mux selected = 16:1 Mux

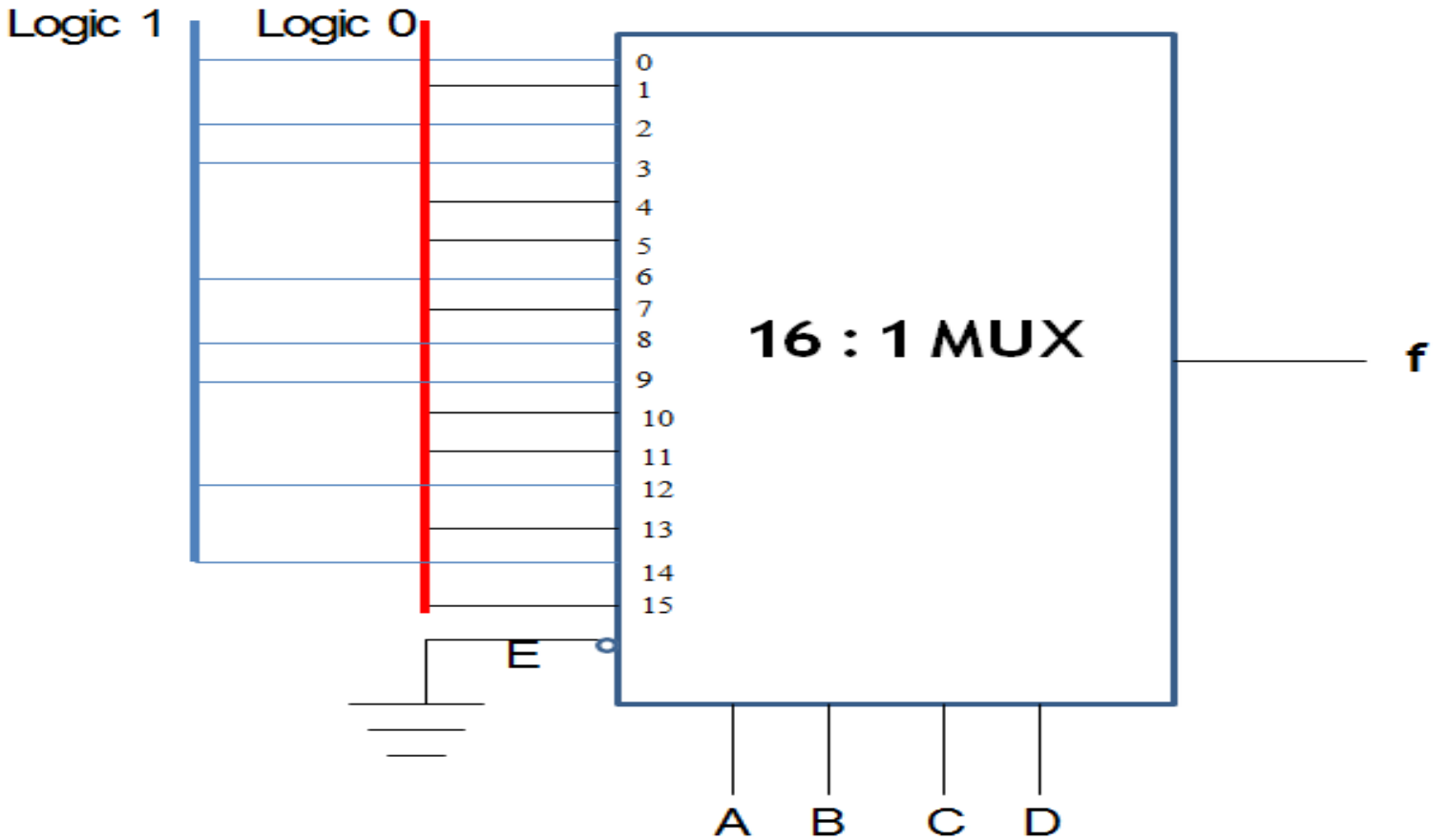
For the Select lines S0 is LSB and S3 is MSB.' For the inputs, D is LSB and A is MSB as per the table.

S0= D
S1= C
S2= B
S3 =A

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

Implement Circuit using MUX

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



Implement Boolean Equation (POS) using MUX

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

As there are 4 variables, multiplexer having four select lines is required.

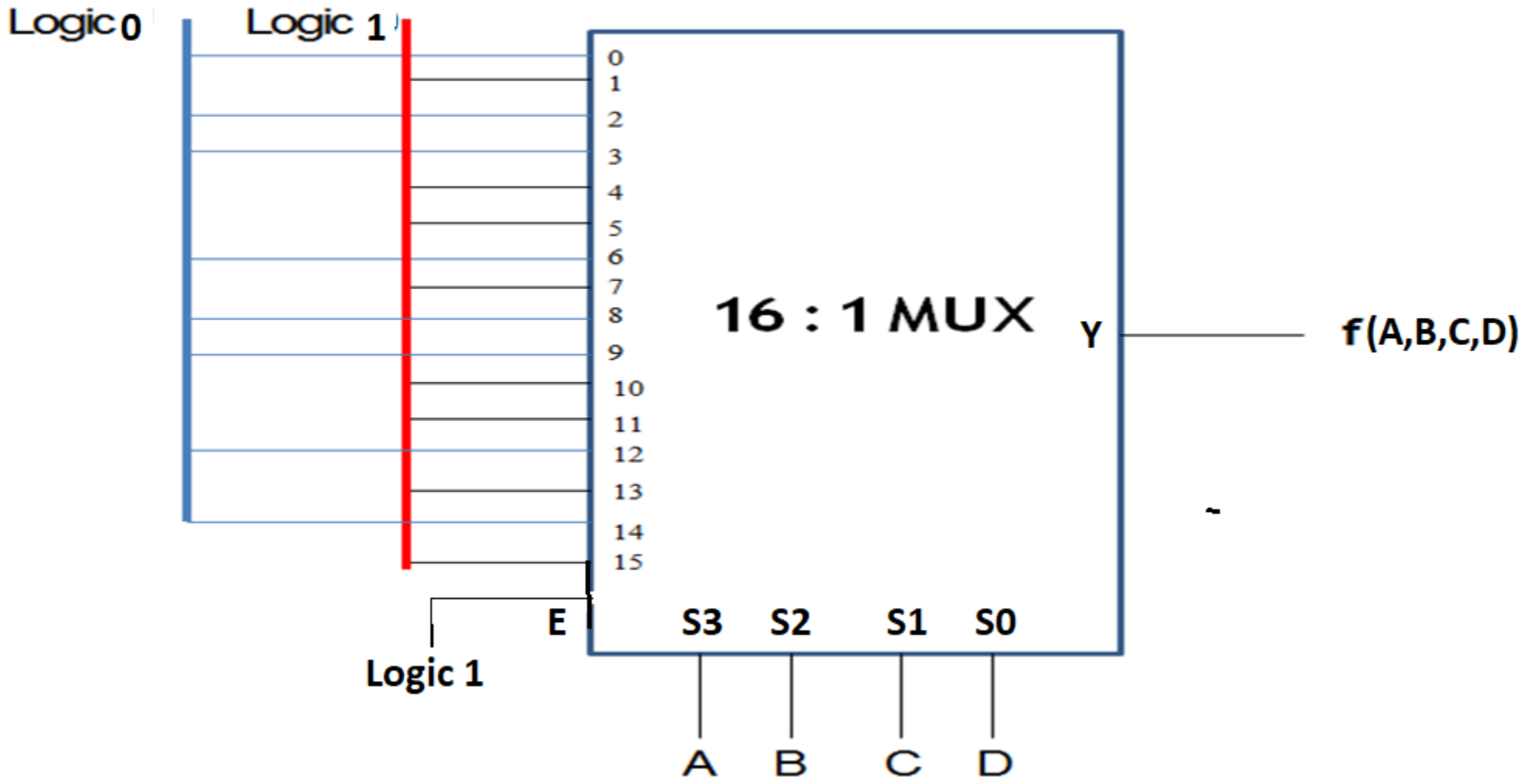
Mux selected = 16:1 Mux

For the Select lines S0 is LSB and S3 is MSB.'
For the inputs, D is LSB and A is MSB as per the table.

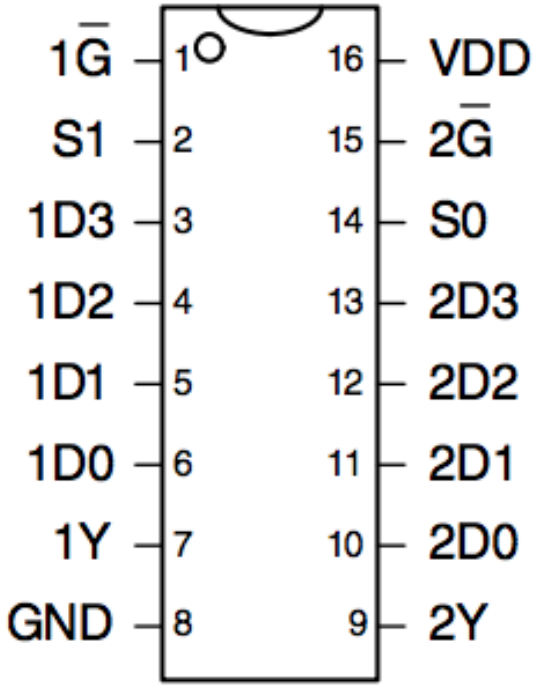
S0= D
S1= C
S2= B
S3 =A

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

Implement Circuit using MUX



IC 74153 Dual 4:1 Multiplexer



74153 4 :1 Mux

(A)

Strobe (G') Active Low	Select Inputs		Data Inputs				Y
	B	A	D0	D1	D2	D3	
H	X	X	X	X	X	X	L
L	L	L	L	X	X	X	L
L	L	L	H	X	X	X	H
L	L	H	X	L	X	X	L
L	L	H	X	H	X	X	H
L	H	L	X	X	L	X	L
L	H	L	X	X	H	X	H
L	H	H	X	X	X	L	L
L	H	H	X	X	X	H	H

Multiplexer Tree

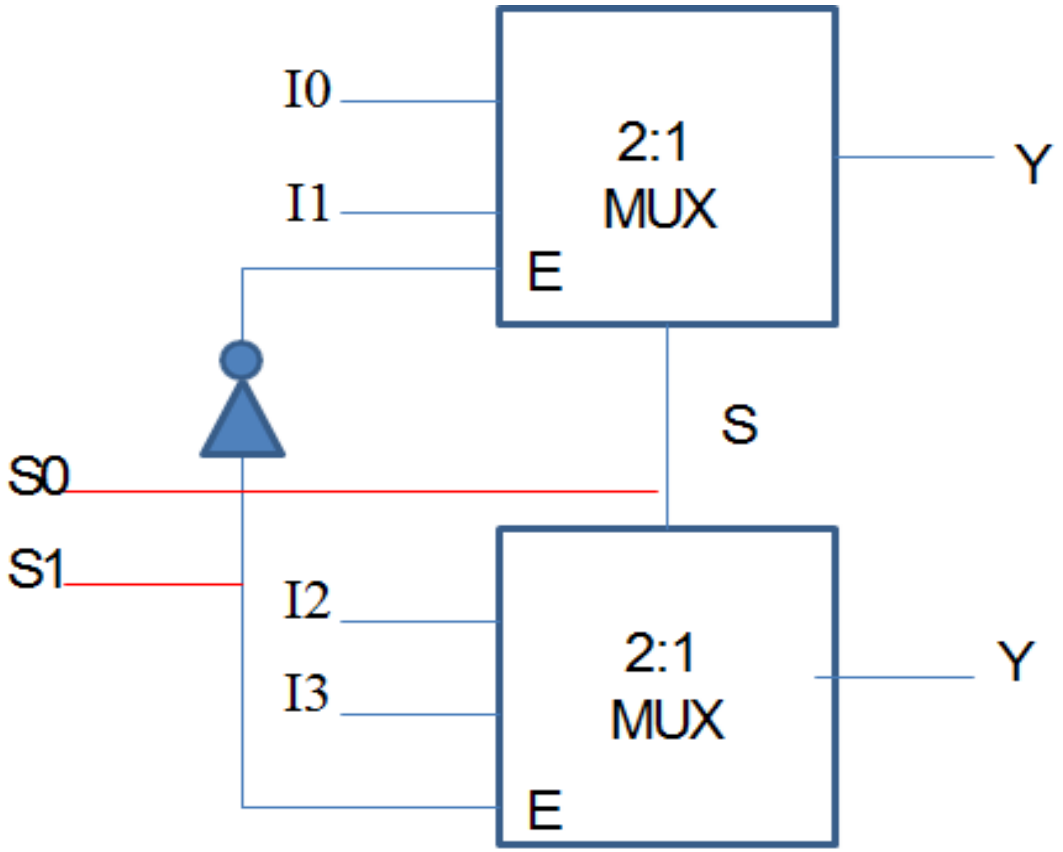
- 16-to-l multiplexers are the largest available lCs

- To meet the larger input needs, multiplexers stack or trees are designed with the help of enable/strobe inputs.

MUX Tree

Implement 4:1 Using 2:1

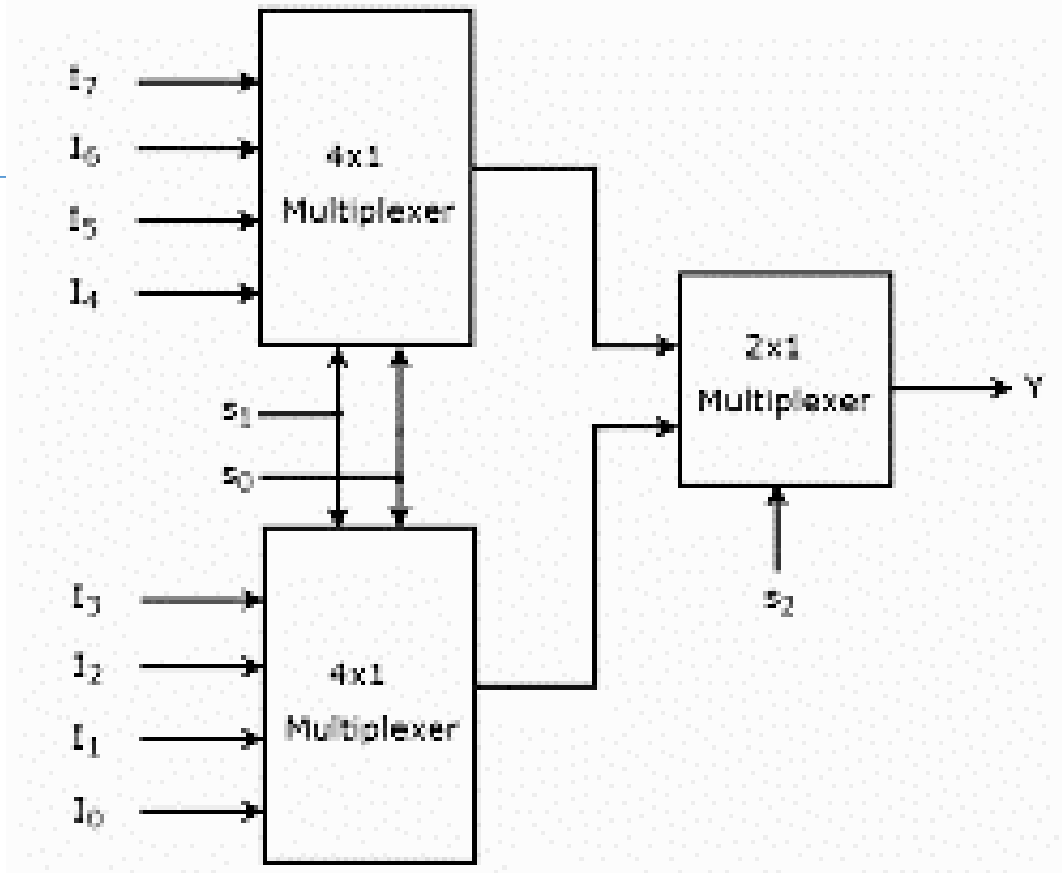
	S1	S0	
0	0	0	→ I0
	0	1	→ I1
1	1	0	→ I2
	1	1	→ I3



MUX Tree (Contd..)

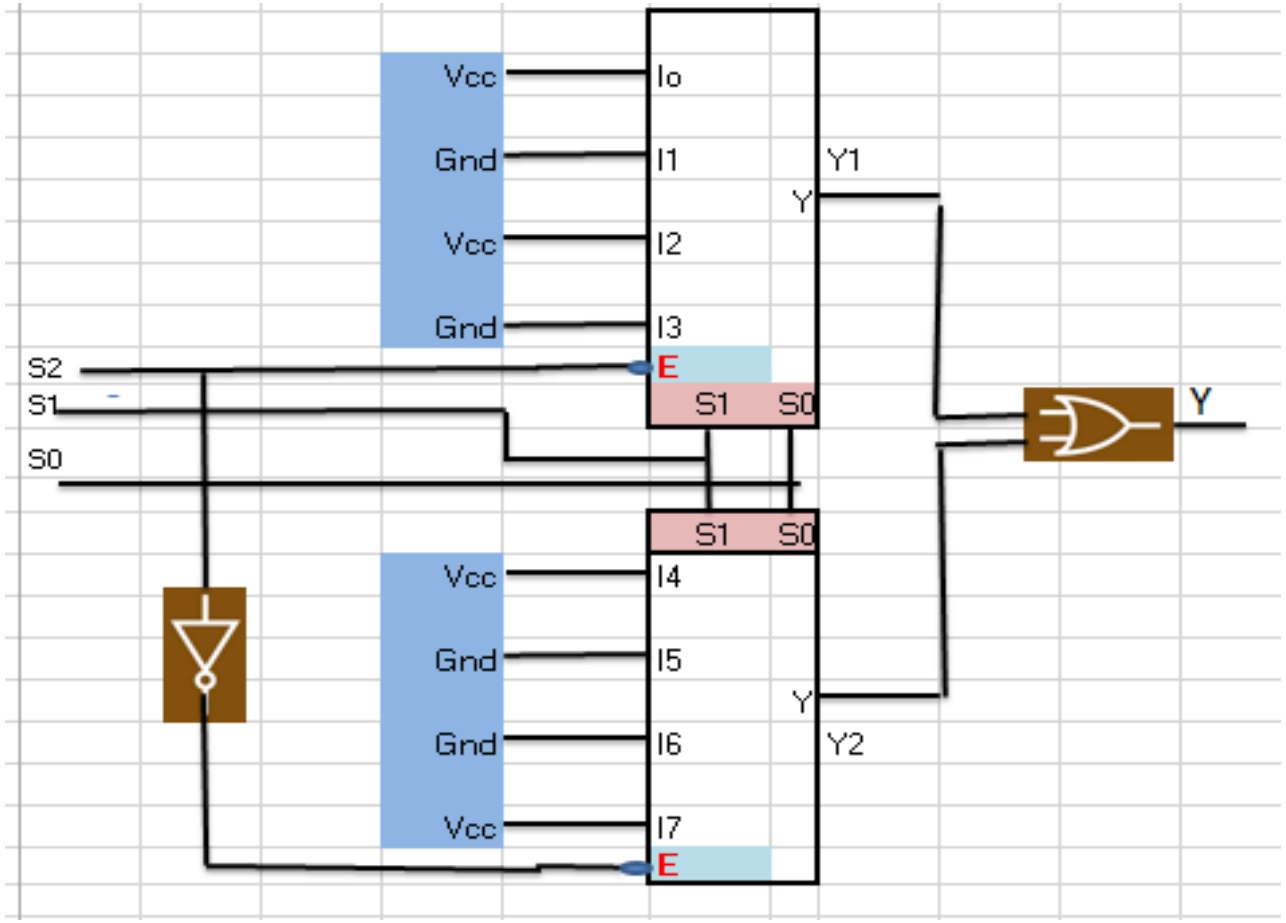
Implement 8:1 Multiplexer using 4:1 and 2:1 Multiplexer.

S2	S1	S0	Y
0	0	0	I0
0	0	1	I1
0	1	0	I2
0	1	1	I3
1	0	0	I4
1	0	1	I5
1	1	0	I6
1	1	1	I7



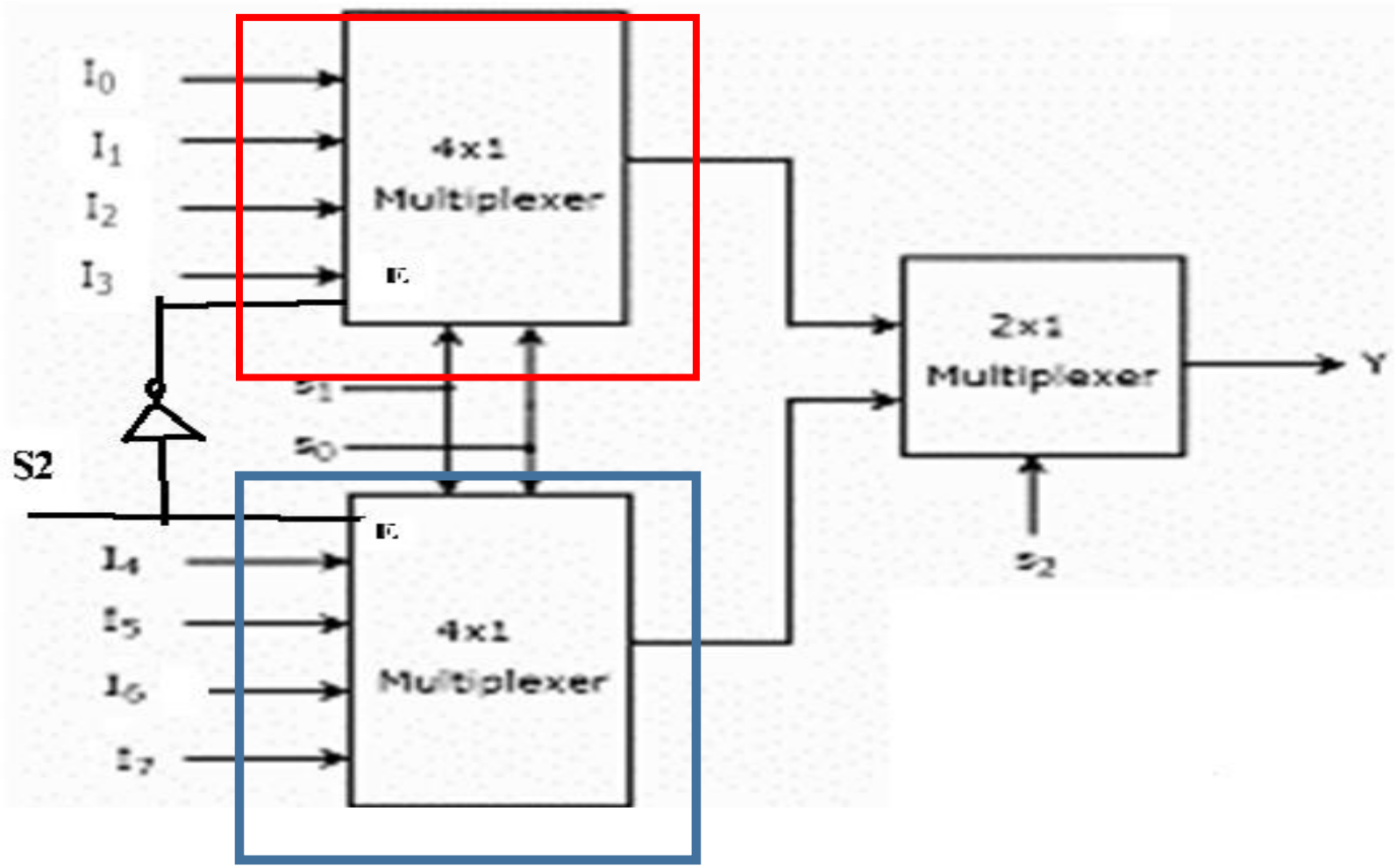
Implement 8:1 Multiplexer using 4:1 Multiplexers and OR gate

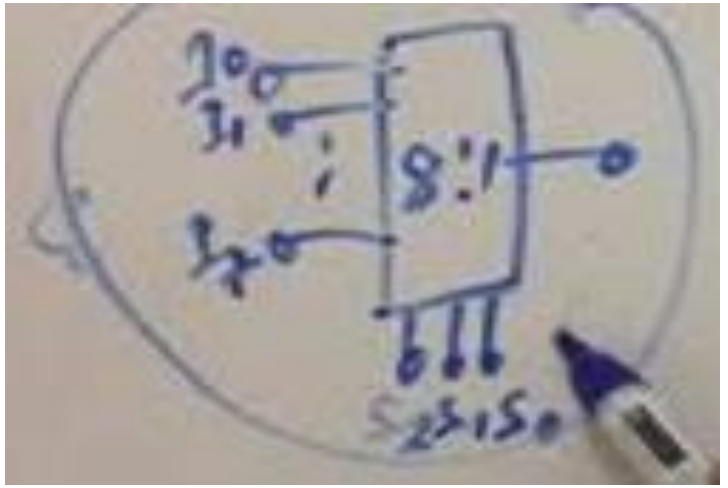
S2	S1	S0	Y
0	0	0	I0
0	0	1	I1
0	1	0	I2
0	1	1	I3
1	0	0	I4
1	0	1	I5
1	1	0	I6
1	1	1	I7



MUX Tree (Active High Enable)

- Implement 8:1 Multiplexer using 4:1 and 2:1 Multiplexer.

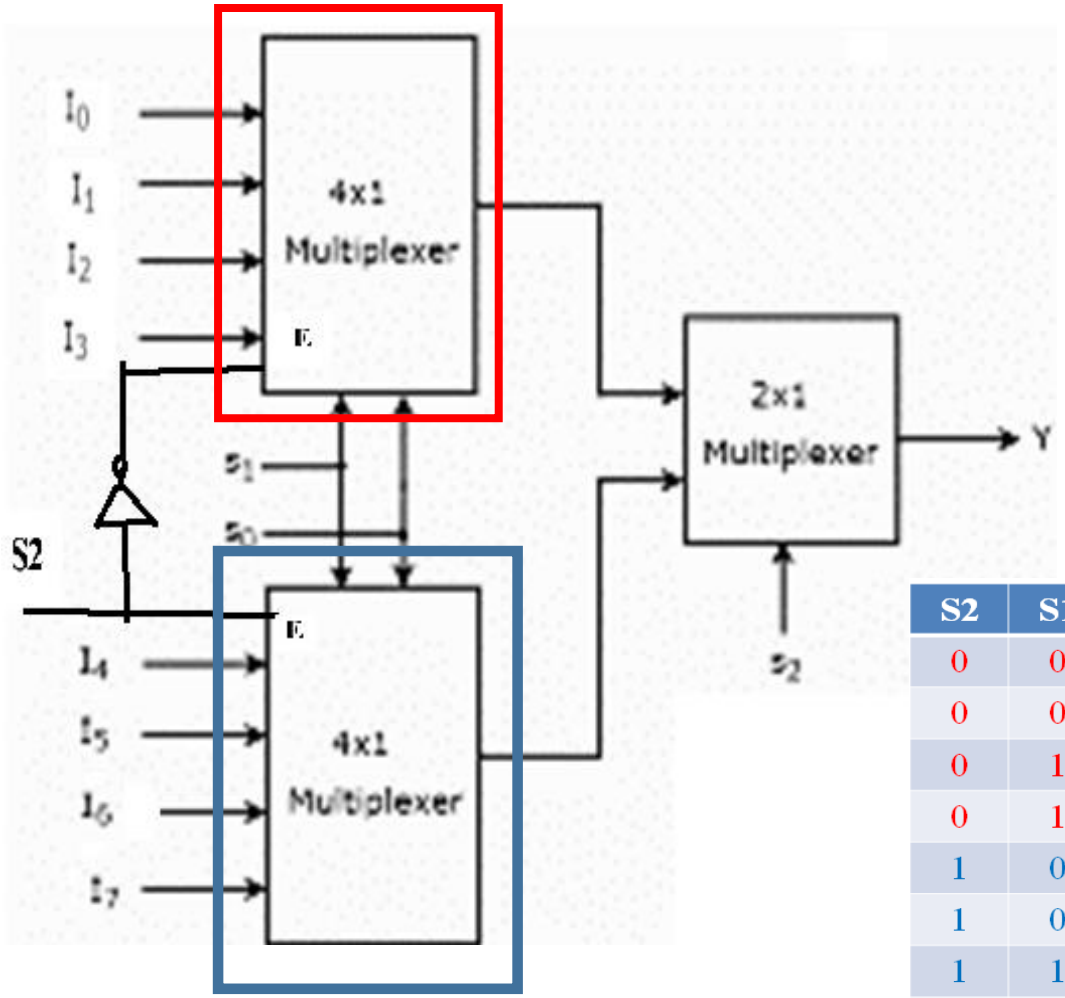




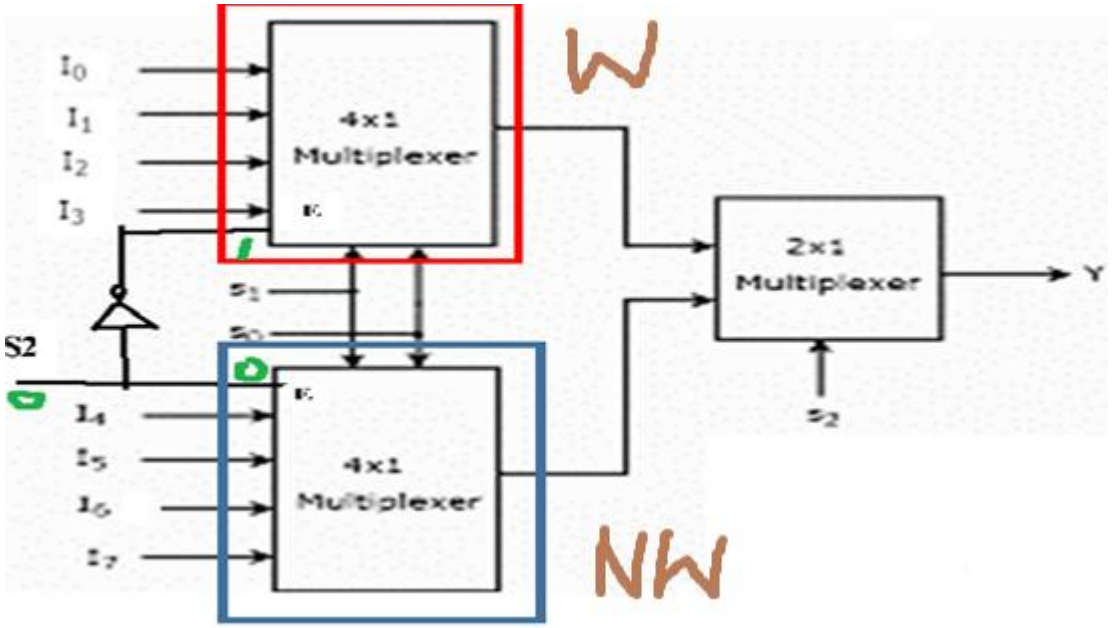
S2	S1	S0	Y
0	0	0	I0
0	0	1	I1
0	1	0	I2
0	1	1	I3
1	0	0	I4
1	0	1	I5
1	1	0	I6
1	1	1	I7

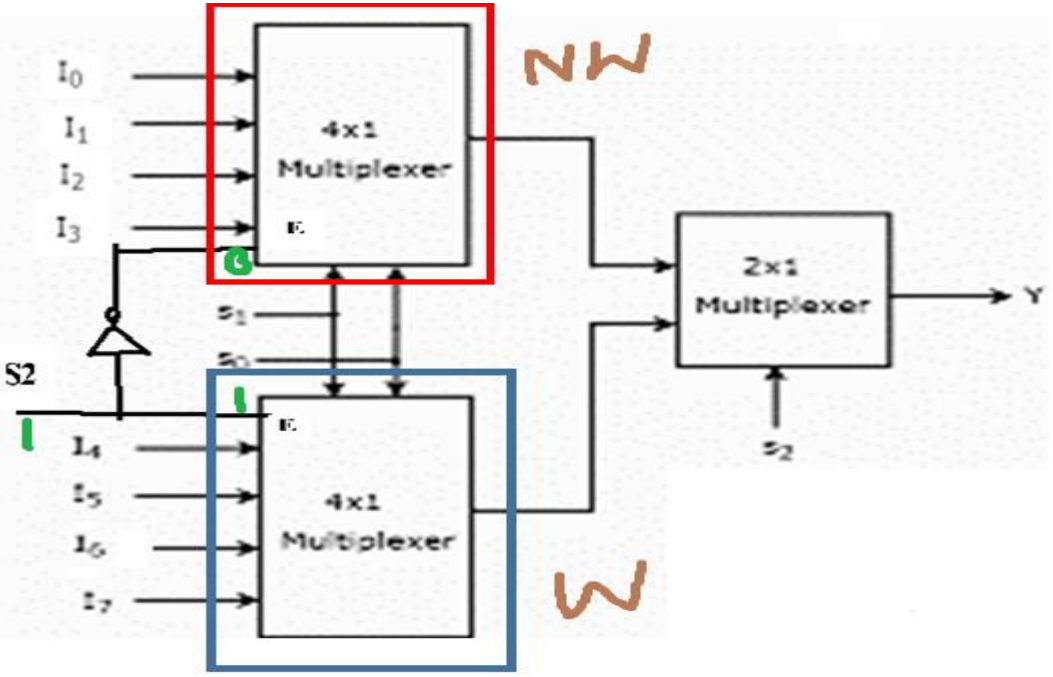
MUX Tree (Active High Enable)

- Implement 8:1 Multiplexer using 4:1 and 2:1 Multiplexer.

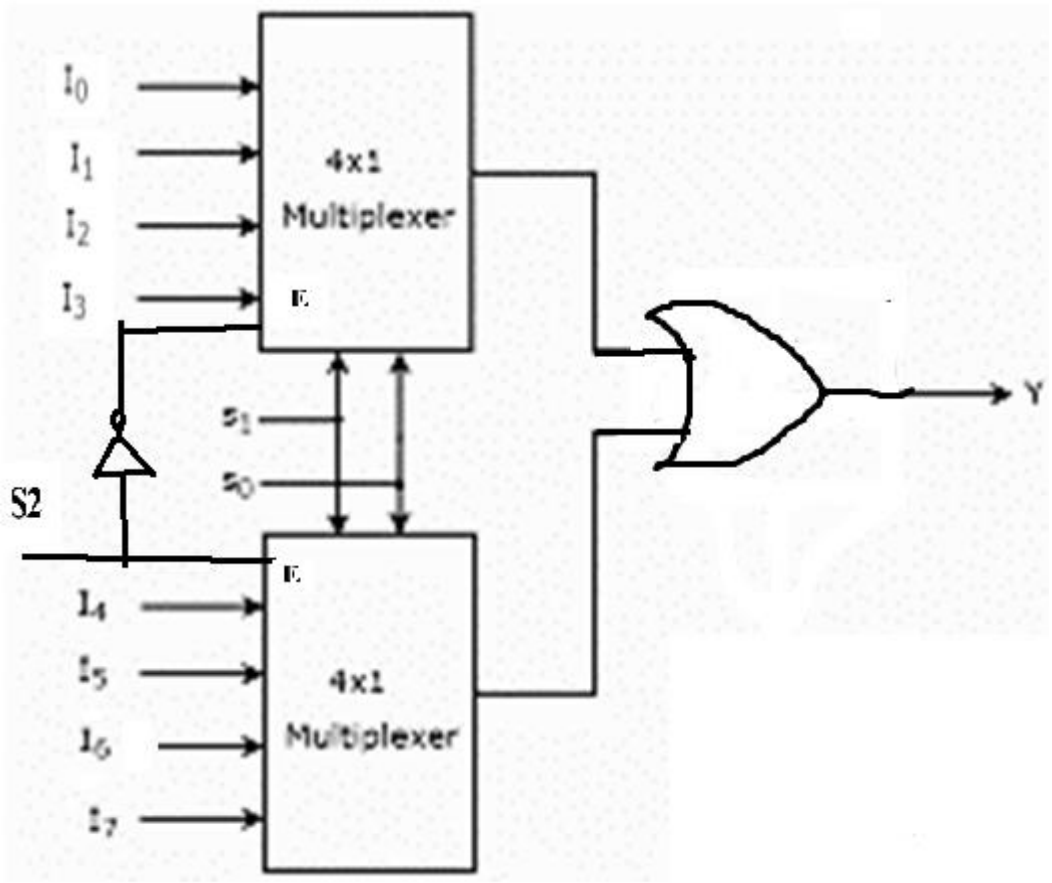


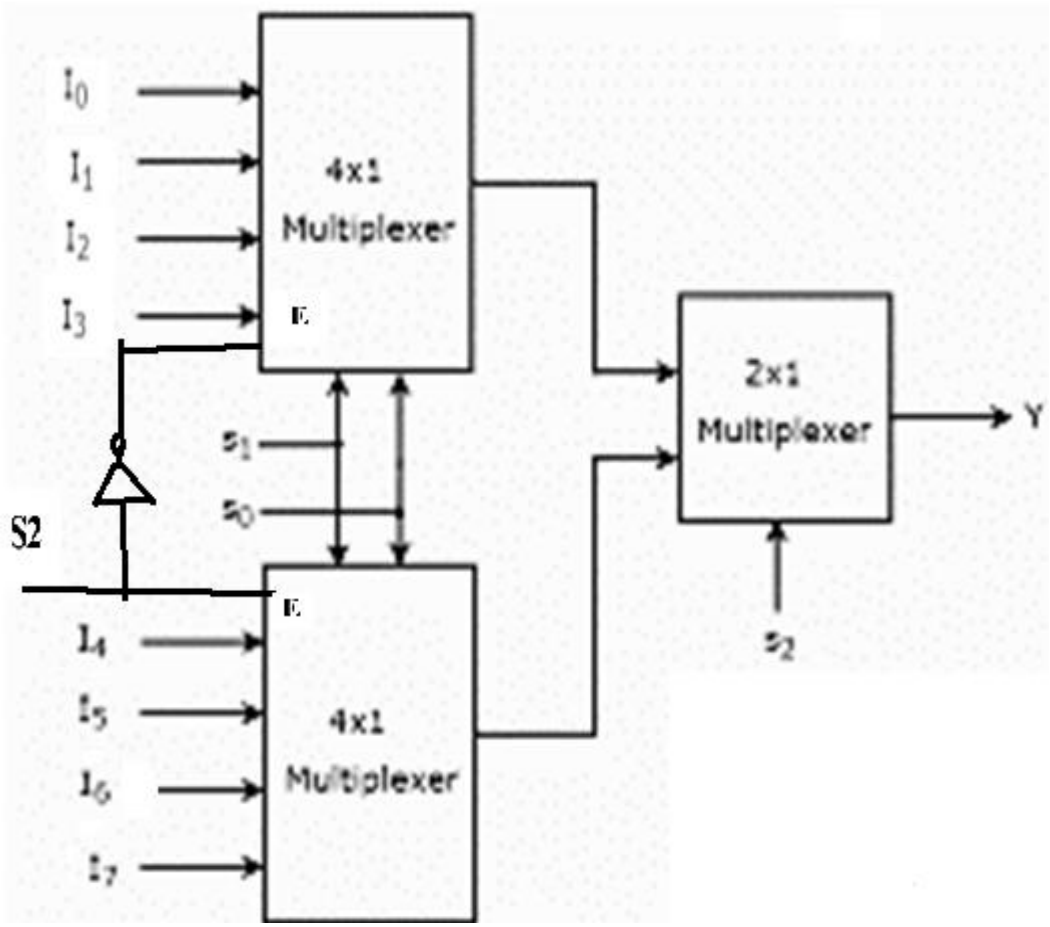
S2	S1	S0	Y
0	0	0	I0
0	0	1	I1
0	1	0	I2
0	1	1	I3
1	0	0	I4
1	0	1	I5
1	1	0	I6
1	1	1	I7





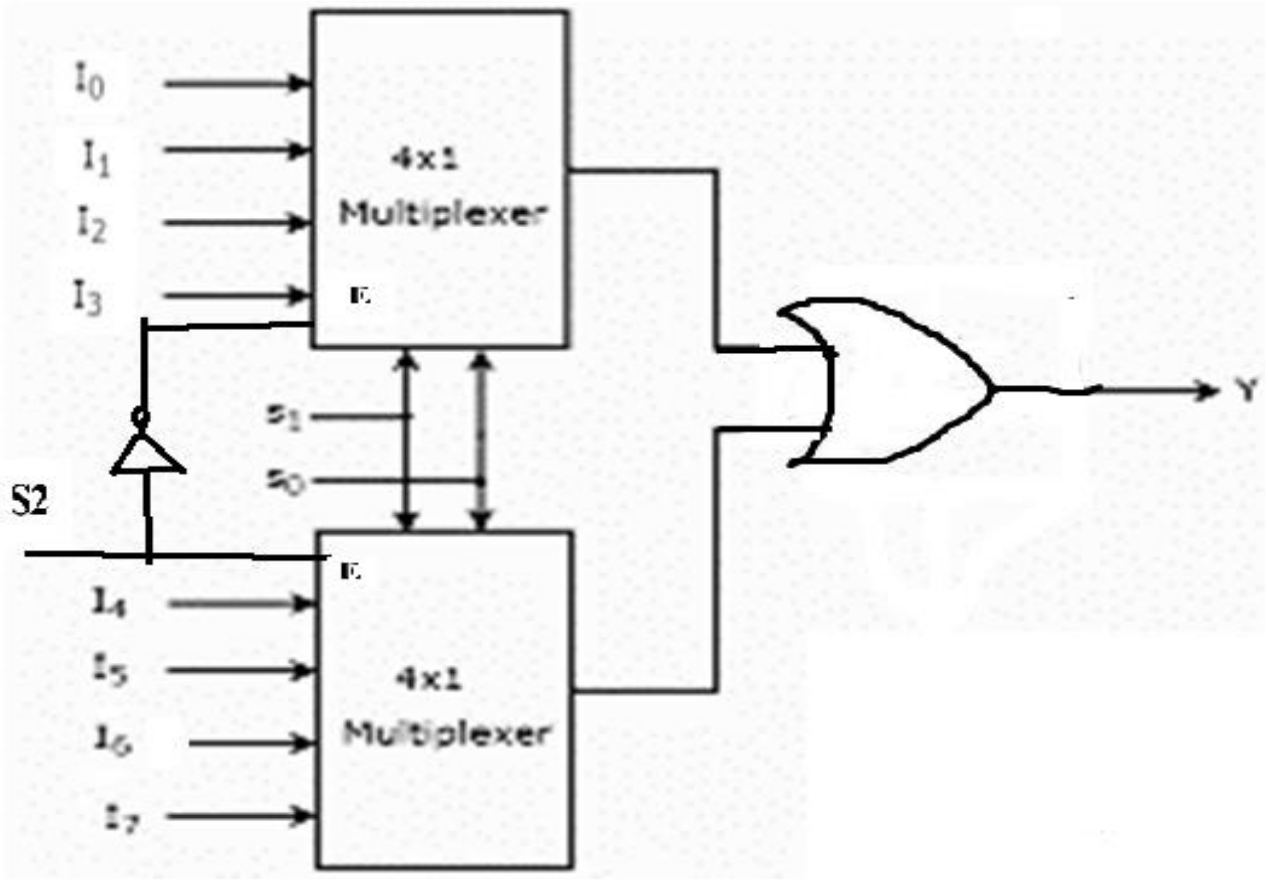
At Output side we can either use OR gate else Mux





MUX Tree (Contd..)

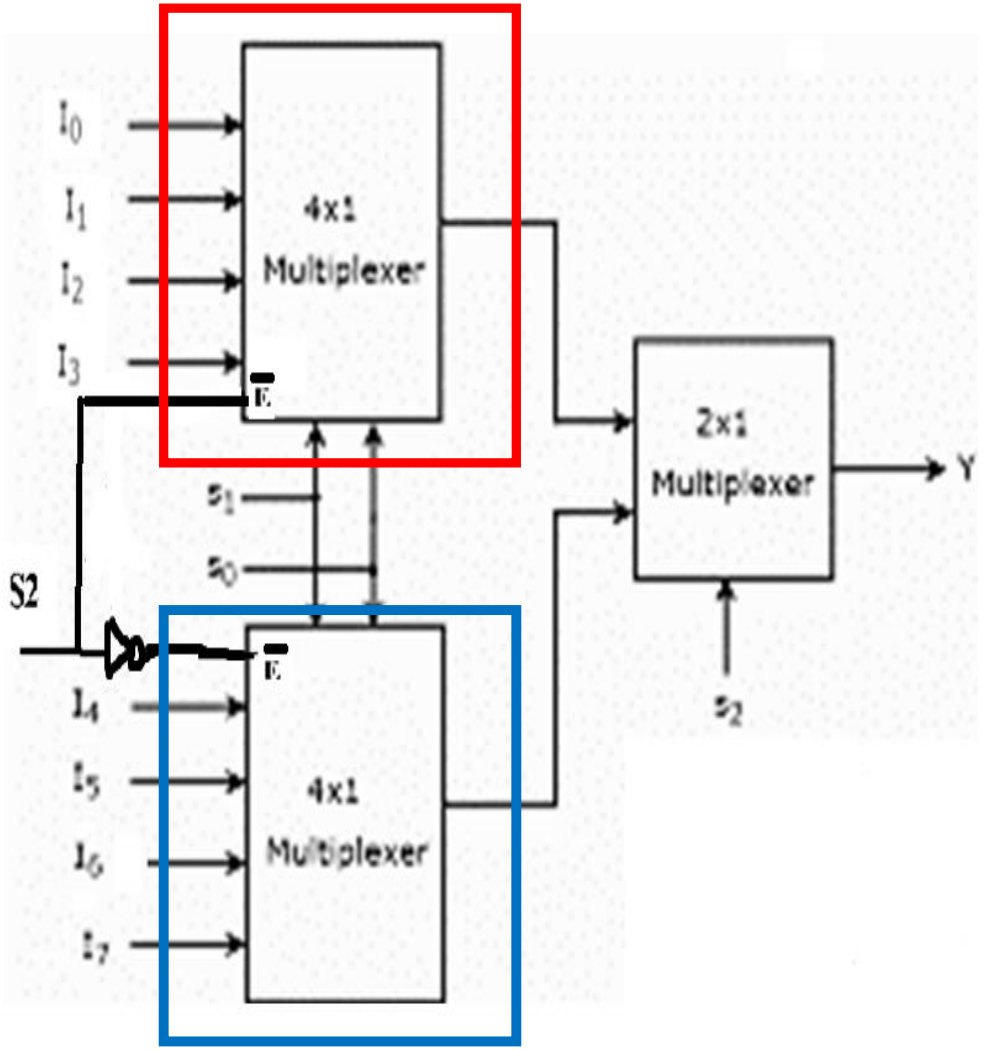
- Implement 8:1 Multiplexer using 4:1 and **OR Gate**



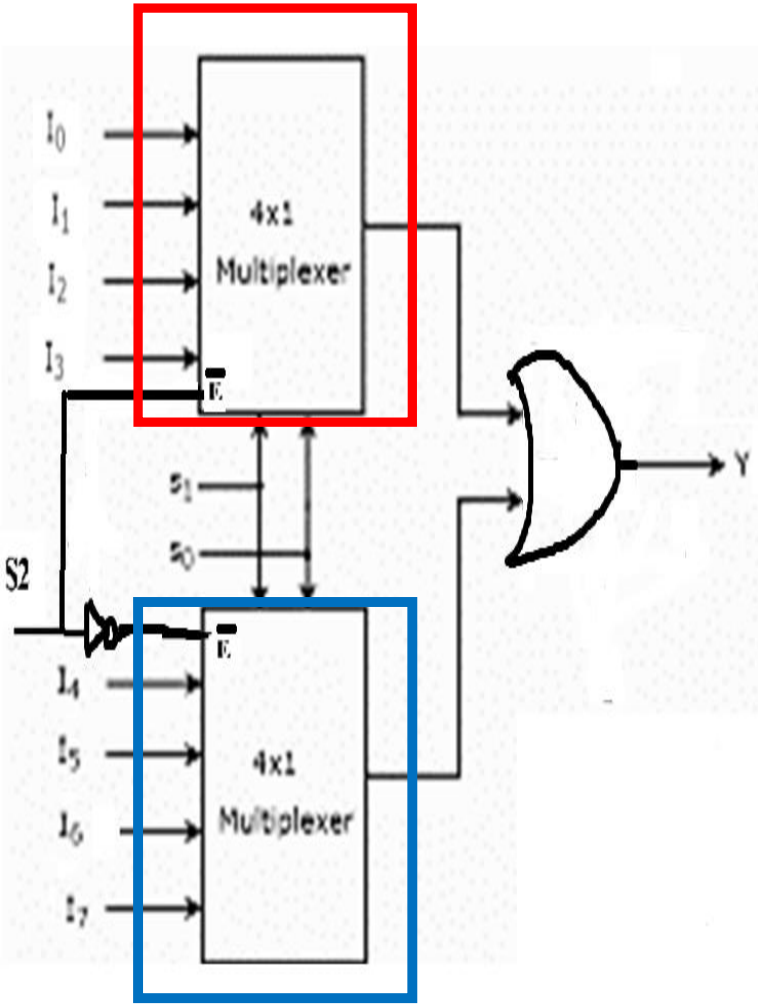
S2	S1	S0	Y
0	0	0	I0
0	0	1	I1
0	1	0	I2
0	1	1	I3
1	0	0	I4
1	0	1	I5
1	1	0	I6
1	1	1	I7

MUX Tree (Active Low Enable)

- Implement 8:1 Multiplexer using 4:1 and 2:1 Multiplexer.



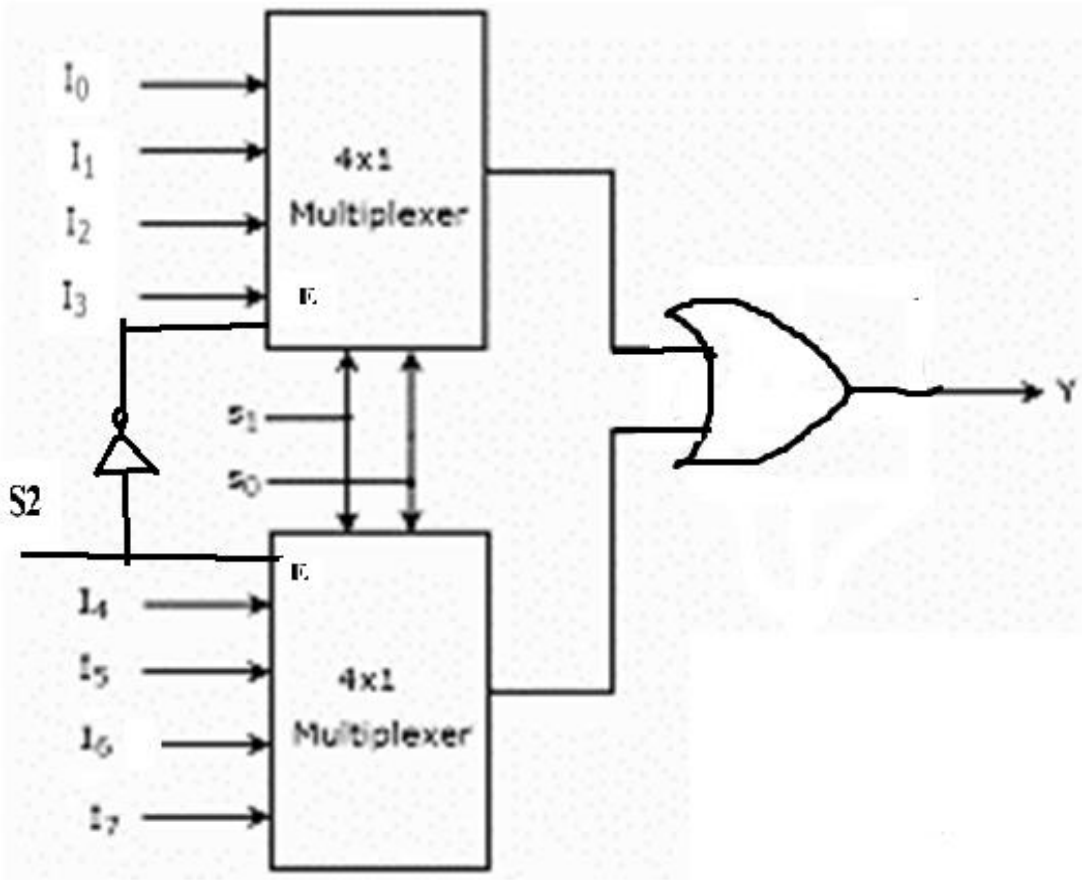
S2	S1	S0	Y
0	0	0	I0
0	0	1	I1
0	1	0	I2
0	1	1	I3
1	0	0	I4
1	0	1	I5
1	1	0	I6
1	1	1	I7



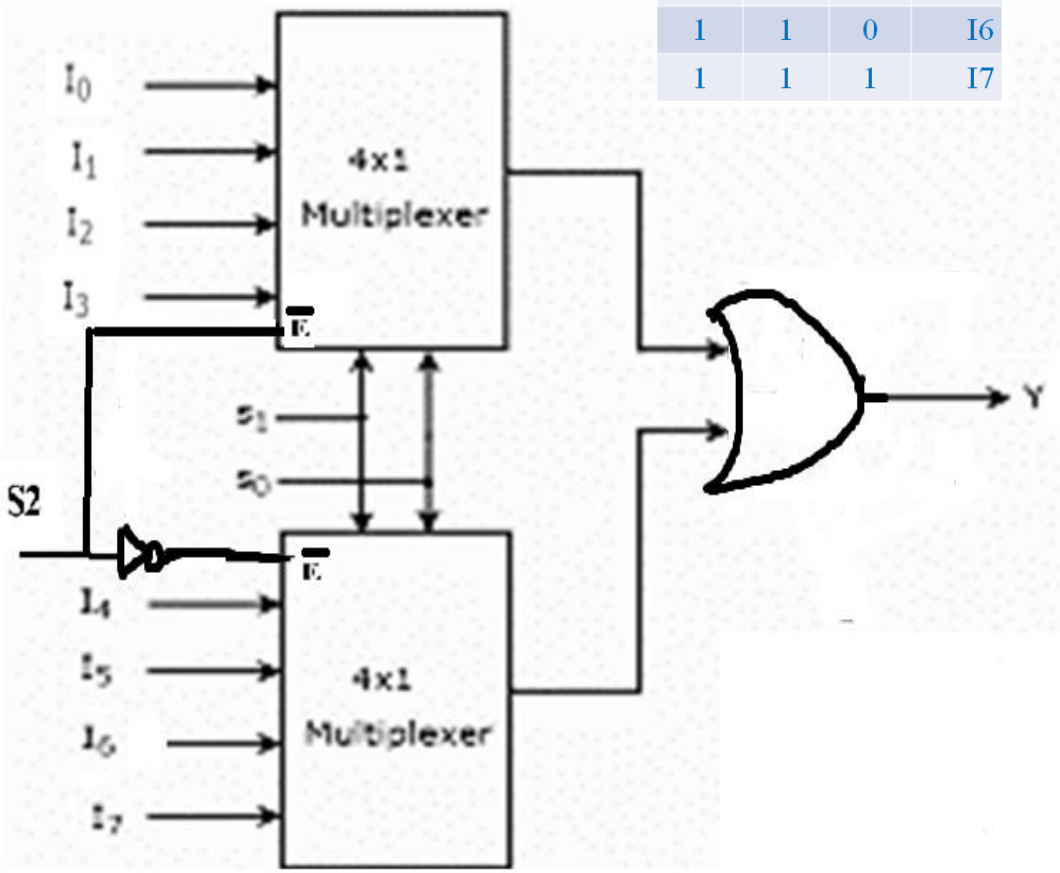
Active low and Active high enable – ckt comparison

S2	S1	S0	Y
0	0	0	I0
0	0	1	I1
0	1	0	I2
0	1	1	I3
1	0	0	I4
1	0	1	I5
1	1	0	I6
1	1	1	I7

ACTIVE High Enable



ACTIVE Low Enable



Implementation of 3 variable SOP expression using 4x1 MUX (Reduction Method)

- Any n–variable Boolean Expression can be implemented using 2^{n-1} to 1 MUX and NOT Gates.
- If we have a 3–variable boolean function, it can be implemented using $2^{3-1} = 4$ to 1 MUX.
- If we have a 4–variable boolean function, it can be implemented using $2^{4-1} = 8$ to 1 MUX.
- If you have a function of n variables, we convert it into SOP form.
- We leave out one variable and connect the remaining (n–1) variables to the select lines of Multiplexer.
- The remaining single variable of the function is used for the inputs of the multiplexer.
- Draw a implementation table.

Implementation of 3 variable SOP expression using 4x1 MUX (Reduction Method)

A	B	C	X
0	0	0	1
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	0

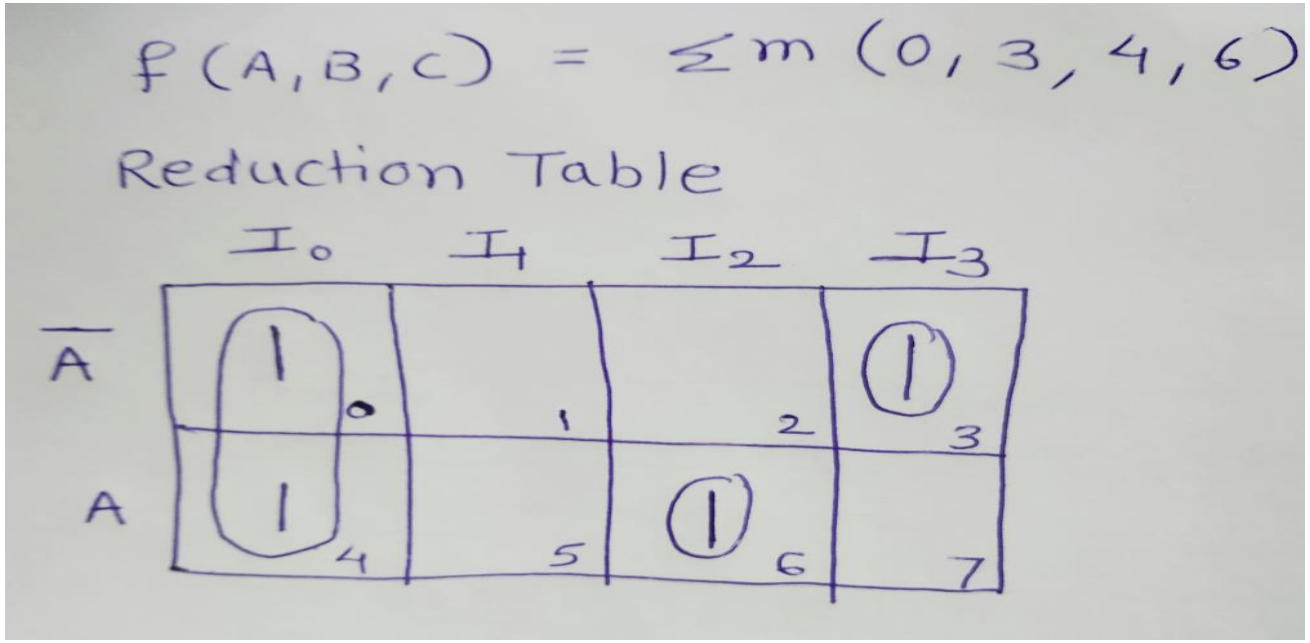
$$F(A,B,C) = X = \Sigma\, m(0, 3, 4, 6)$$

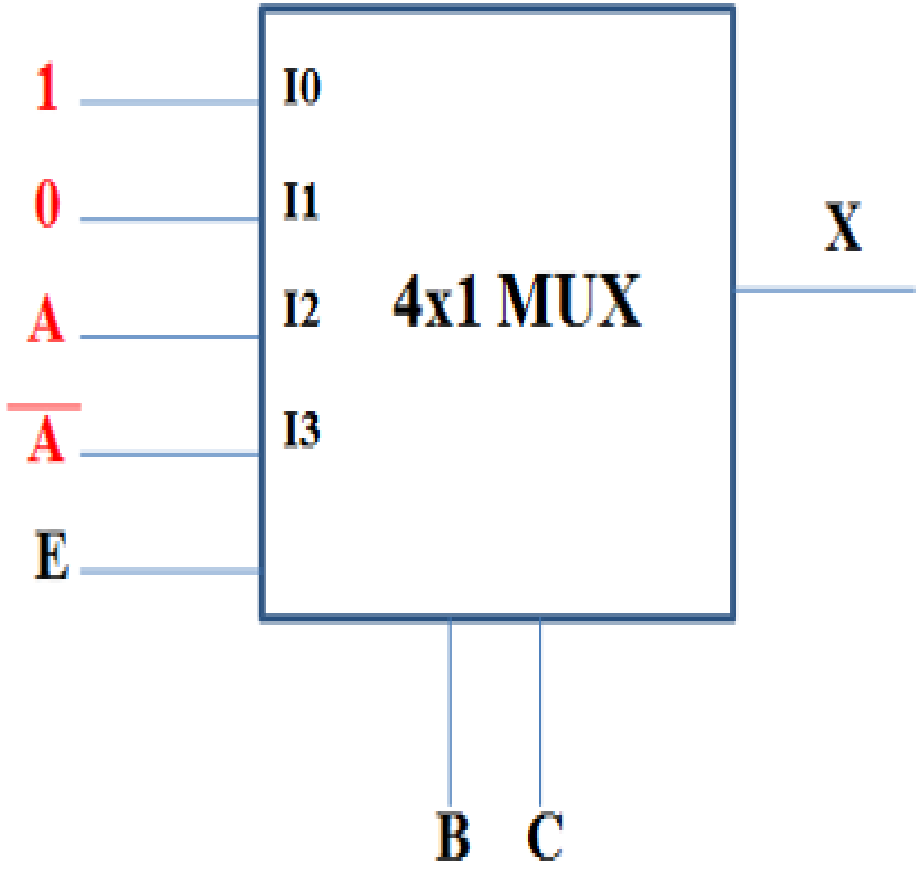
As only 2 select lines are available for 4:1 mux, there is a question of connection of one variable.

Lets connect B and C variables to S1 and S0 select lines of multiplexer.

Use reduction method to find where should third variable A should be connected.

Implementation of 3 variable SOP expression using 4x1 MUX (Reduction Method)





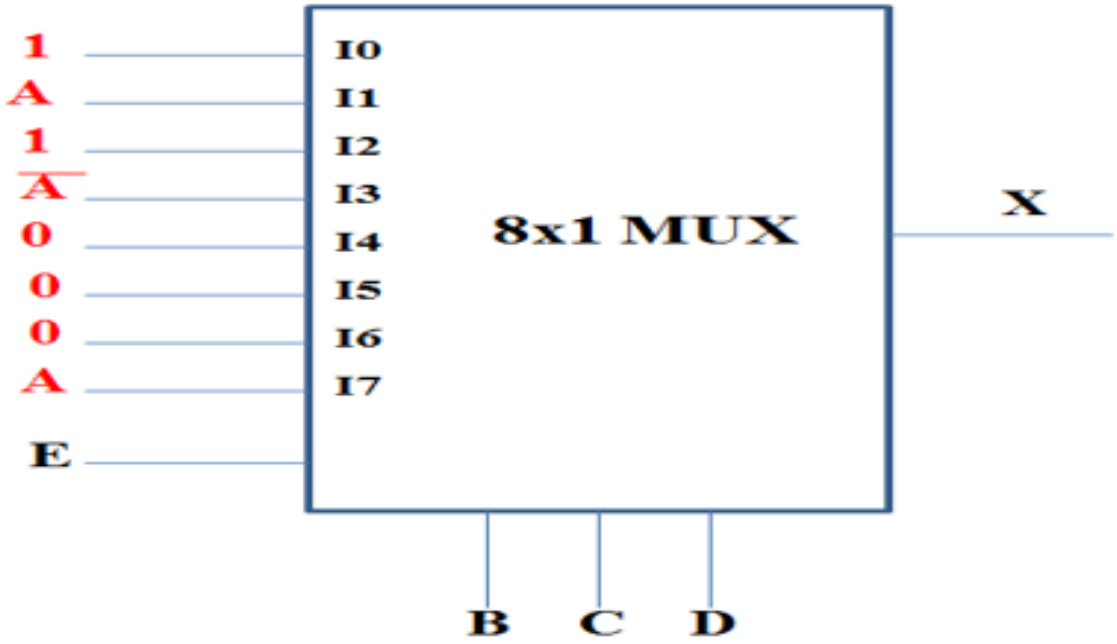
I0 = Vcc, I1 = Gnd

I2 = A, I3 = A’

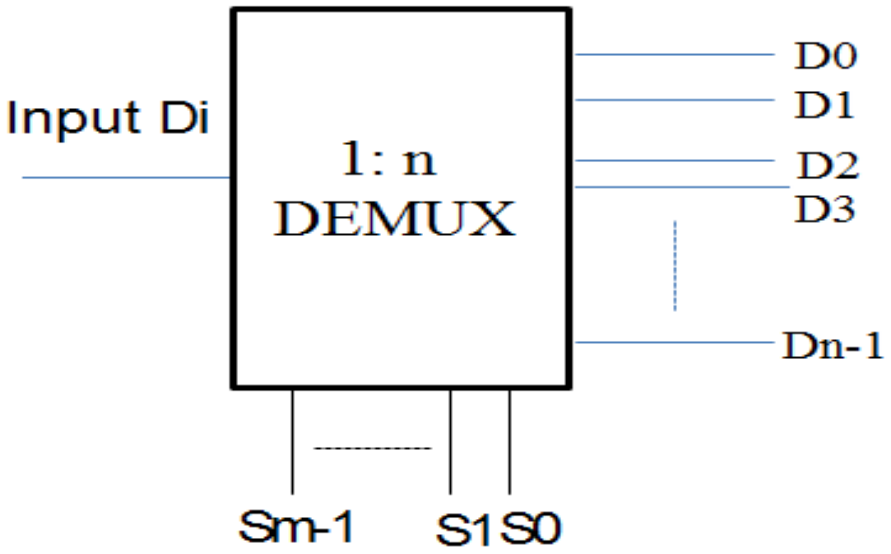
Implementation of 4 variable SOP expression using 8x1 MUX (Reduction Method)

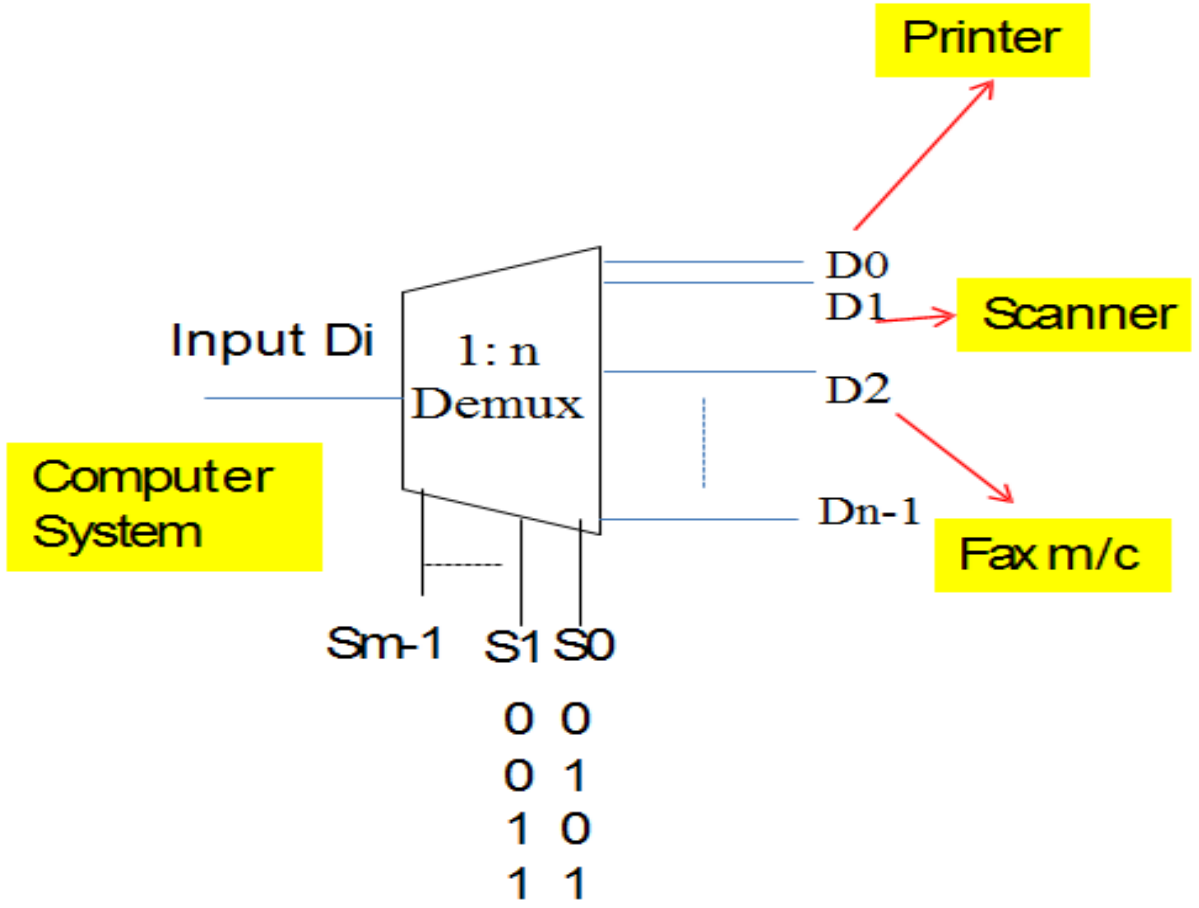
A	B	C	D	X
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

F(A,B,C,D) = X = Σ m(0, 2, 3, 8, 9, 10, 15)



Demultiplexer



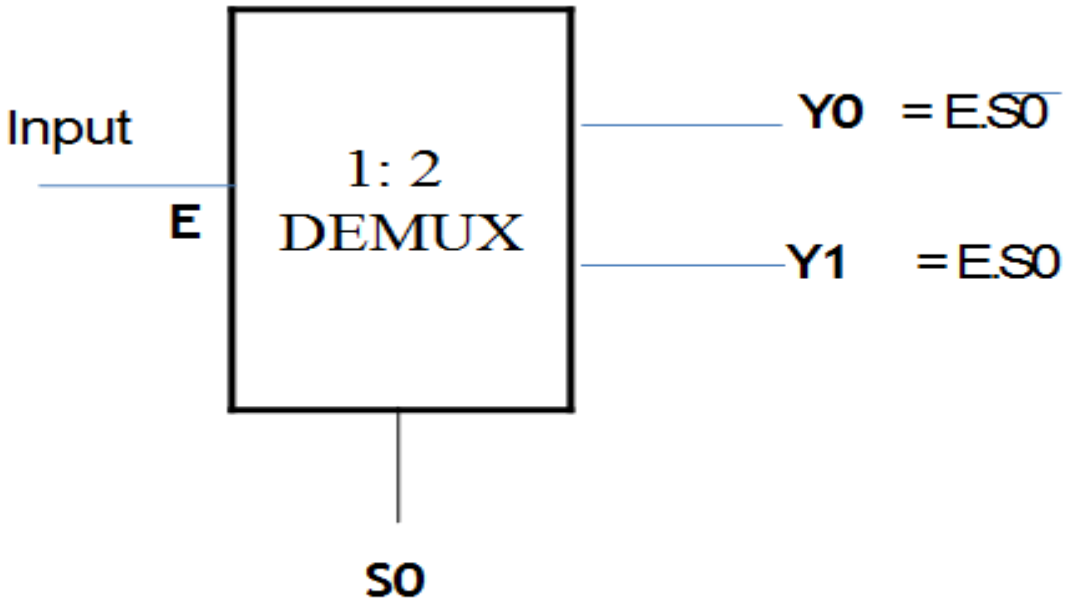


Demultiplexer

- This device is available as an MSI IC and can conveniently be used for the design of combinational circuits.
- These devices are available as 2line-to-4-line, 3-line-to-8-line, and 4-line-to-I6-line decoders.
- The outputs of most of these devices are active-low also there is an active-low enable/data input terminal available.

IC No.	Description	Output
74139	Dual 1:4 Demultiplexer (2-line-to-4-line decoder)	Inverted input
74155	Dual 1:4 Demultiplexer (2-line-to-4-line decoder)	1 *Y*–Inverted input 2*Y*–Same as input
74156	-do-	Open-collector 1 *Y*–Inverted input 2 *Y*–Same as input
74138	1:8 Demultiplexer (3-line-to-8-line decoder)	Inverted input
74154	1:16 Demultiplexer (4-line-to-16-line decoder)	Same as input
74159	-do-	Same as input Open-collector

1:2 Demultiplexer



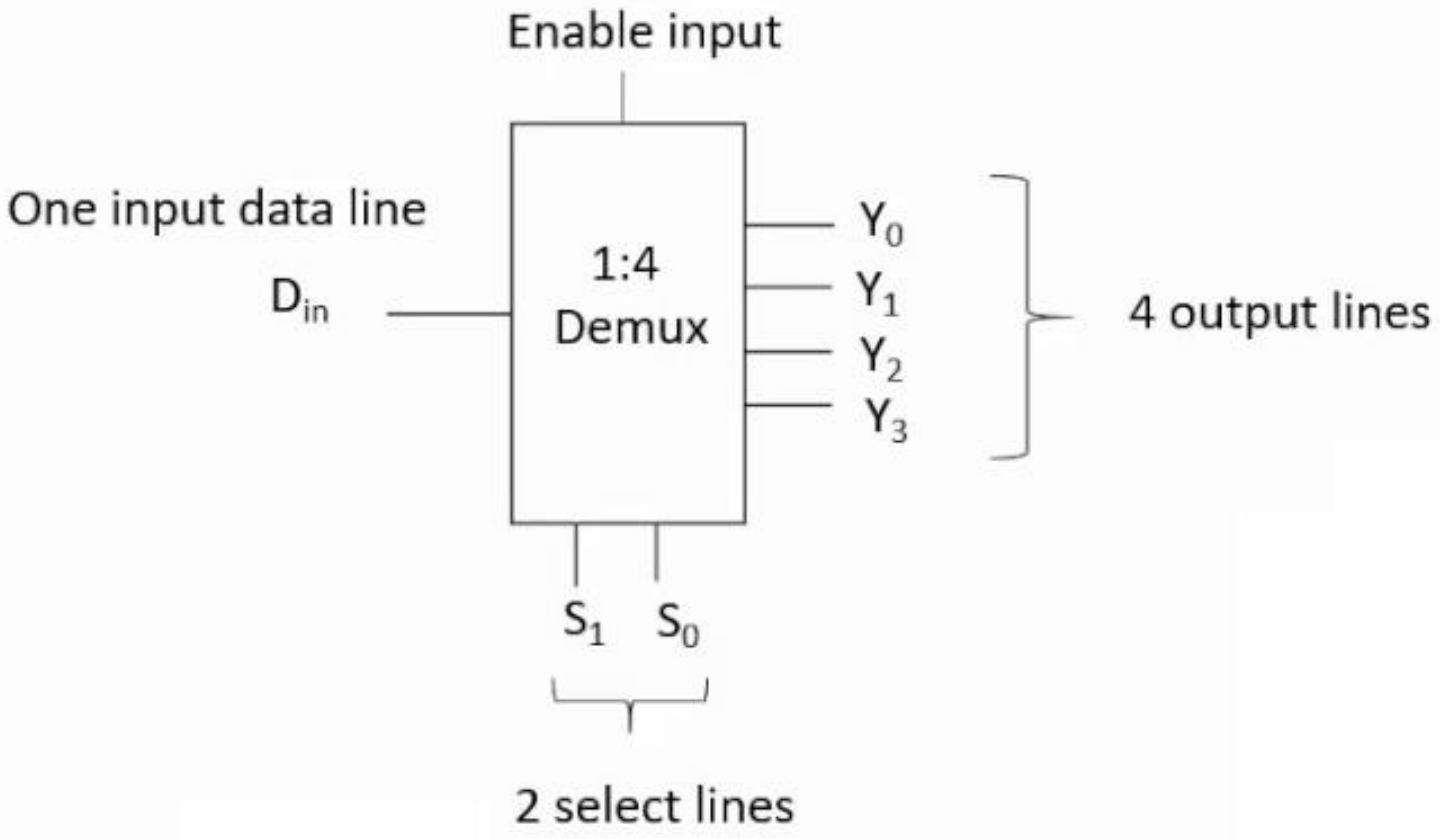
E	S0	Y0	Y1
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

$Y0 = E.\overline{S0}$

$Y1 = E.S0$

Draw 1:2 Demultiplexer using gates.

1:4 Demultiplexer



Truth Table

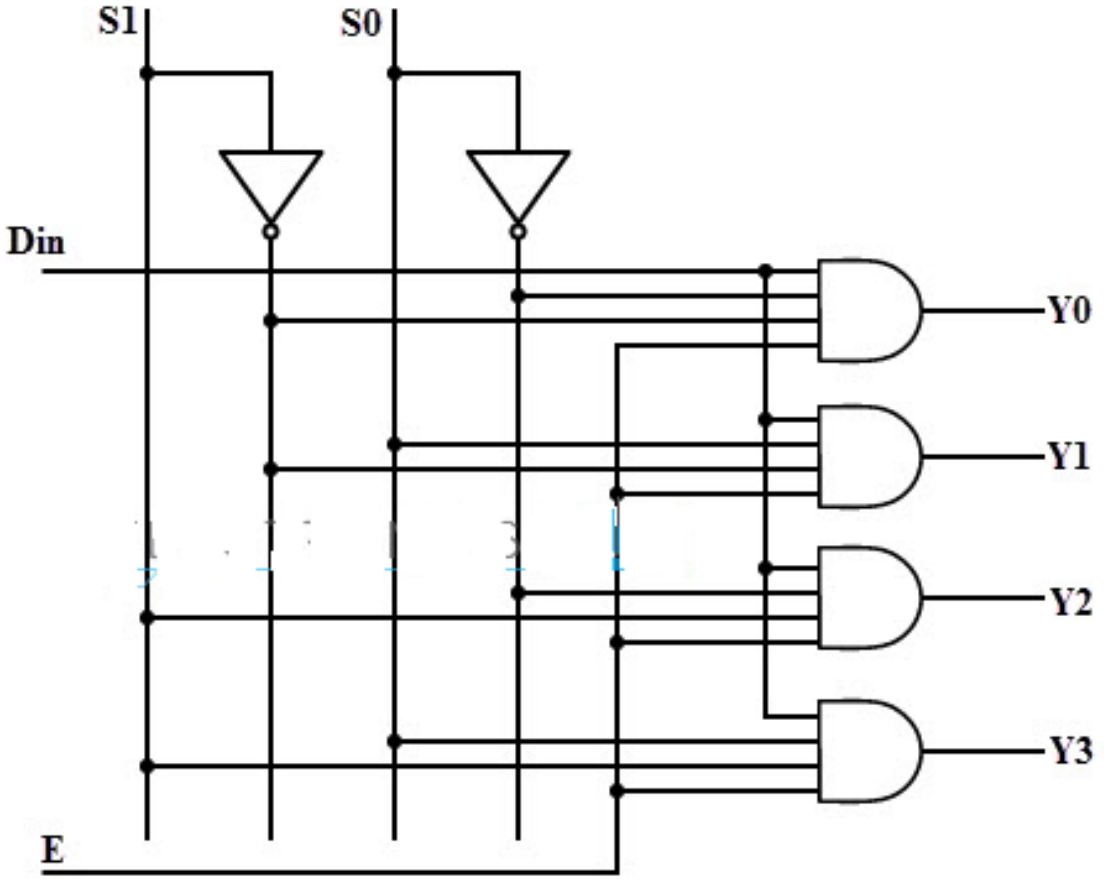
Inputs			Outputs			
E	S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0	0	0	0	D_{in}
1	0	1	0	0	D_{in}	0
1	1	0	0	D_{in}	0	0
1	1	1	D_{in}	0	0	0

$$Y0 = E.\overline{S1}.\overline{S0}.Din$$
$$Y1 = E.\overline{S1}.S0.Din$$
$$Y2 = E.S1.\overline{S0}.Din$$
$$Y3 = E.S1.S0.Din$$

Implement the Circuit

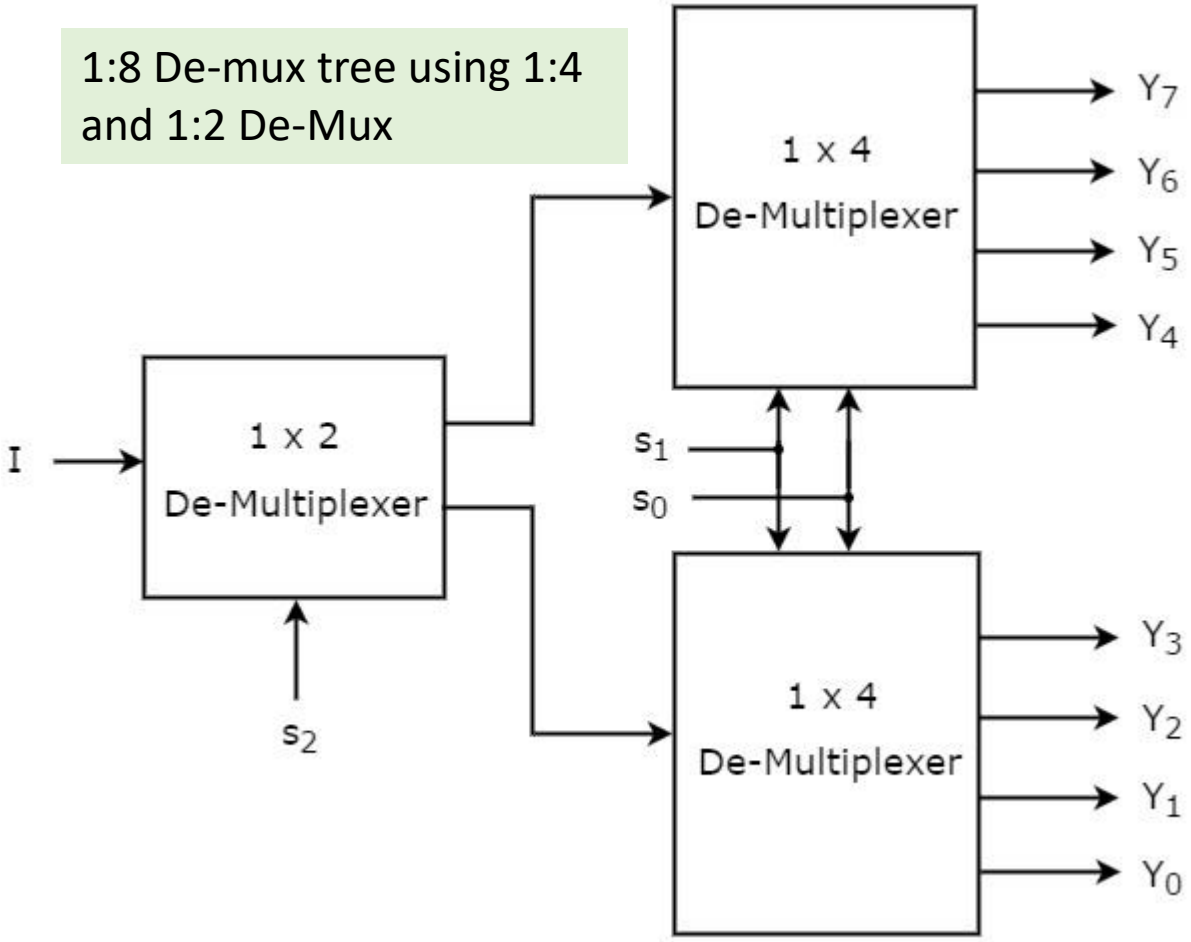
Draw 1:4 Demultiplexer using gates.

$$Y0 = E.\overline{S1}.\overline{S0}.Din$$
$$Y1 = E.\overline{S1}.S0.Din$$
$$Y2 = E.S1.\overline{S0}.Din$$
$$Y3 = E.S1.S0.Din$$

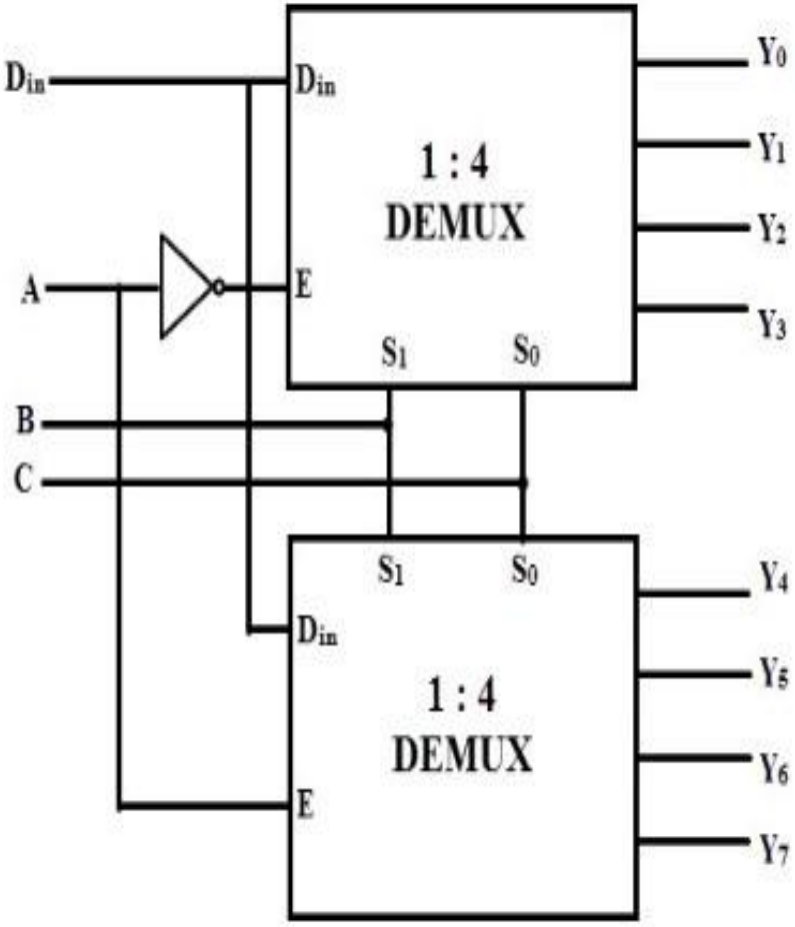


De-multiplexer Tree

1:8 De-mux tree using 1:4 and 1:2 De-Mux



1:8 De-mux tree using 1:4 De-Mux and Not Gate



Implement the function f(A,B,C)=m(0,1,3,5) using 1:8 demultiplexer

General Form of a Demultiplexer is i/p : o/p

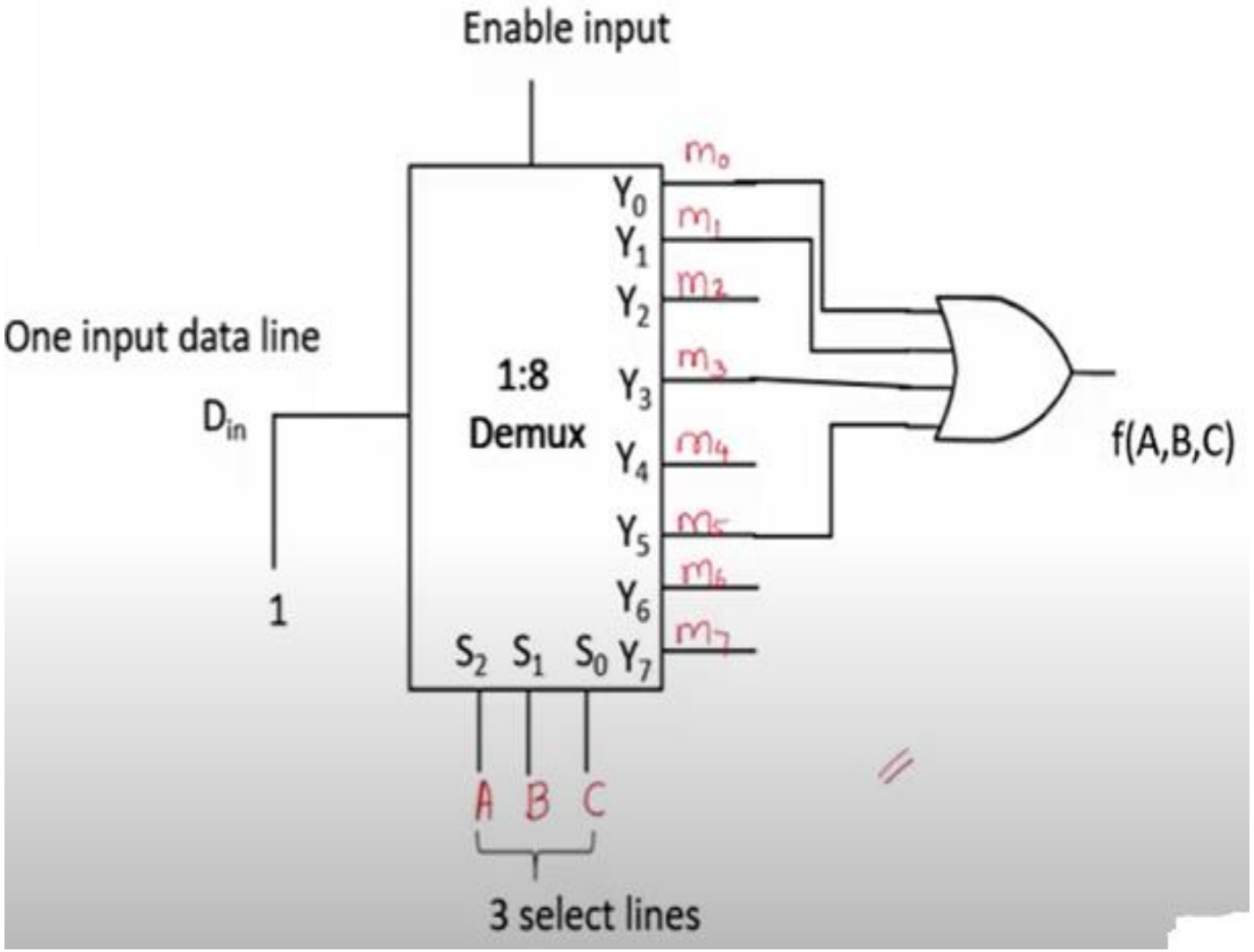
$1 : 2^n$

$1 : 2^3$

No of select lines (n) = 3

No of input = 1

No of outputs = 8



Decoder

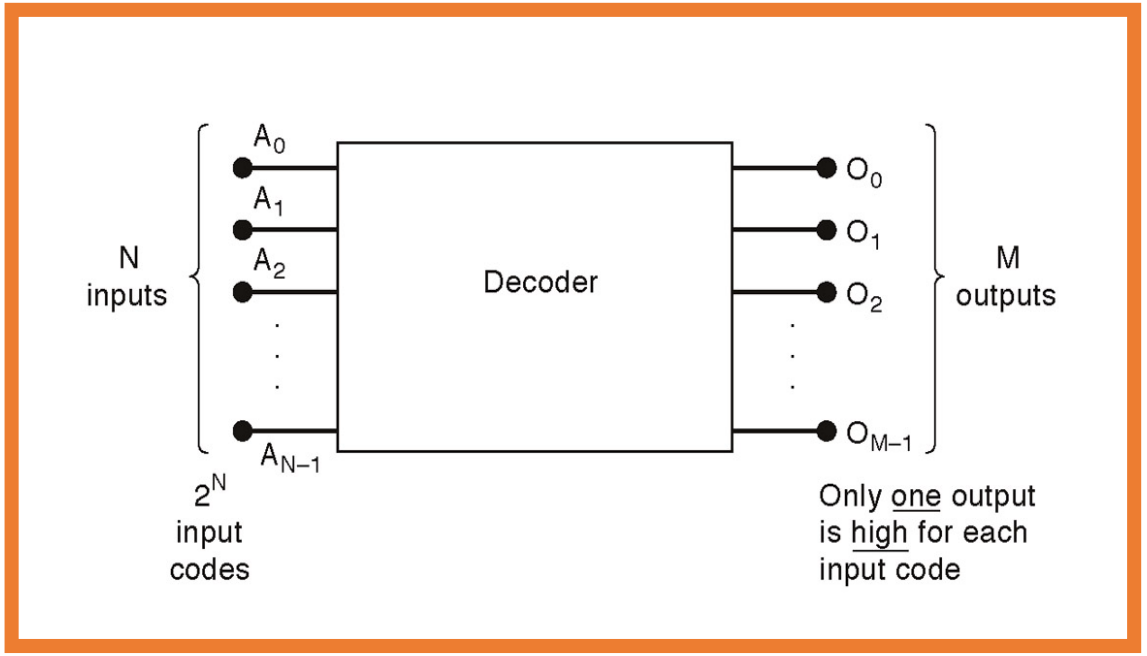
- A decoder is a logic circuit that accepts a set of inputs that represents a binary number and activates only the output that corresponds to the input number
- In other words, **a decoder circuit looks at its inputs, determines which binary number is present there, and activates the one output that corresponds to that number** ; all other outputs remain inactive
- In its general form, a decoder has N input lines to handle N bits and form one to 2^N output lines to indicate the presence of one or more N-bit combinations.

➢ The basic binary function

- An AND gate can be used as the basic decoding element because it produces a HIGH output only when all inputs are HIGH

General Decoder Diagram

N = 1 2 Outputs
N = 2 4 Outputs
N = 3 8 Outputs
N = 4 16 Outputs



1:2 Decoder
2:4 Decoder
3:8 Decoder
4:16 Decoder

There are 2^N possible input combinations, from A_0 to A_{N-1}.

For each of these input combinations only one of the *M* outputs will be active *HIGH* (1), all the other outputs are *LOW* (0).

General Decoder Diagram (Contd..)

- If an active-LOW output (**74138**, one of the output will low and the rest will be high) is required for each decoded number, the entire decoder can be implemented with

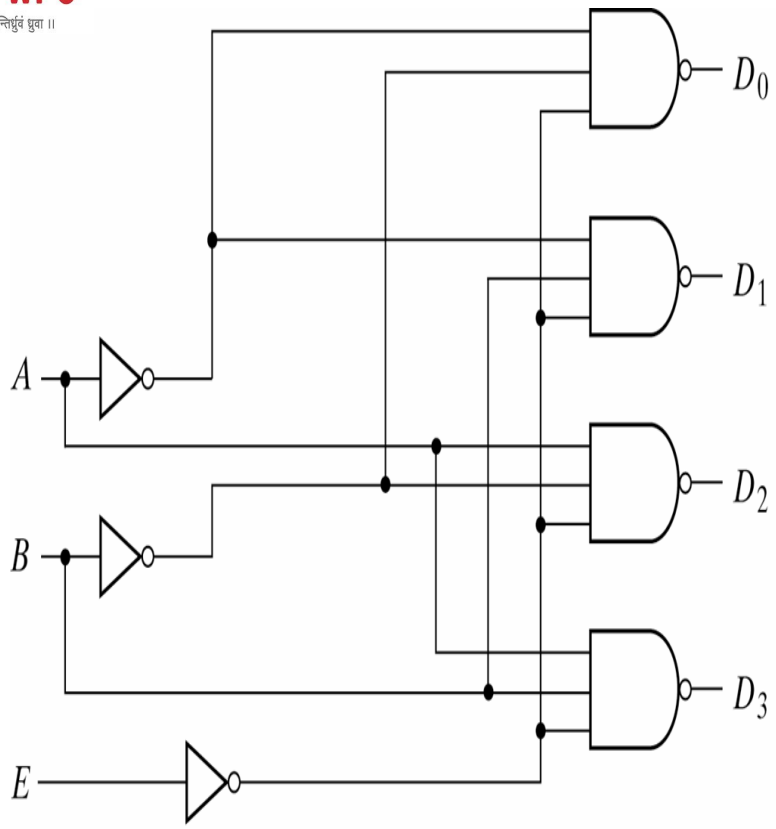
  1.NAND gates

  2.Inverters

- If an active-HIGH output (**74139,** one of the output will high and the rest will be low) is required for each decoded number, the entire decoder can be implemented with

  - **AND gates**
  - **Inverters**

2-to-4-Line Decoder
(with Active Low Enable input)-Active LOW output



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

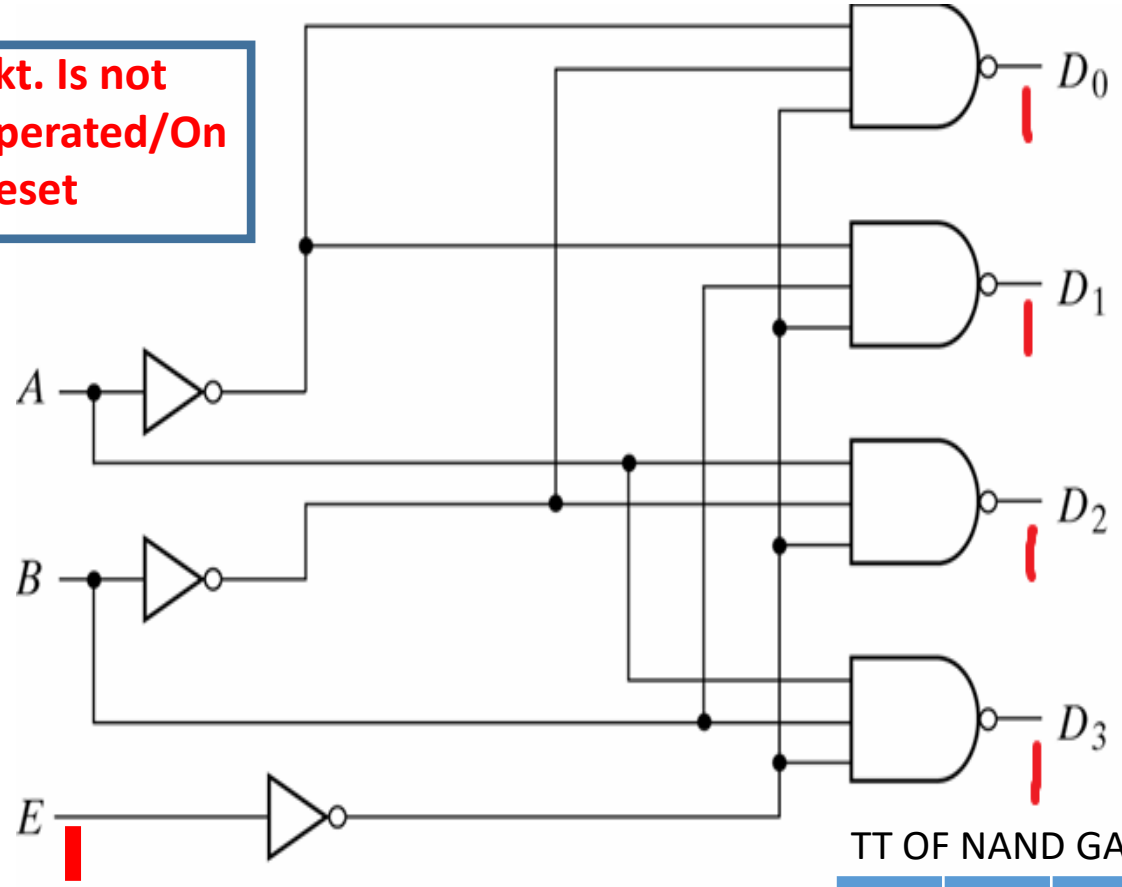
TT OF NAND GATE

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

- The circuit operates with complemented outputs and a complement enable input. The decoder is enabled when E is equal to 0.
- Only one output can be equal to 0 at any given time, all other outputs are equal to 1.
- The output whose value is equal to 0 represents the minterm selected by inputs A and B
- The circuit is disabled when E is equal to 1.

2-to-4-Line Decoder
(with Active Low Enable input)-Active LOW output

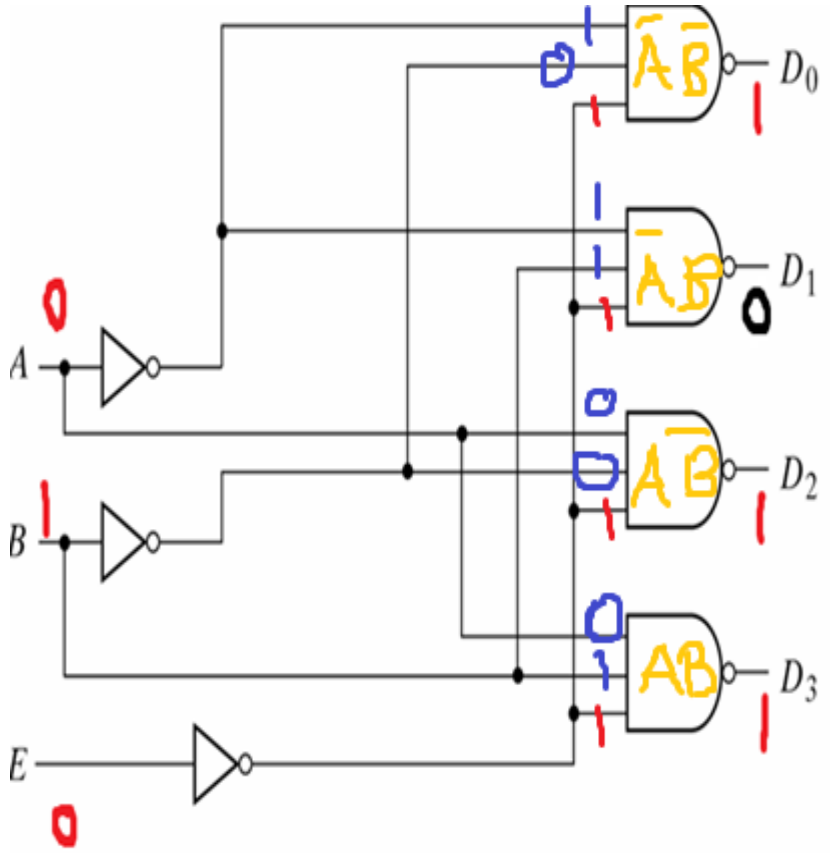
Ckt. Is not operated/On Reset



E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

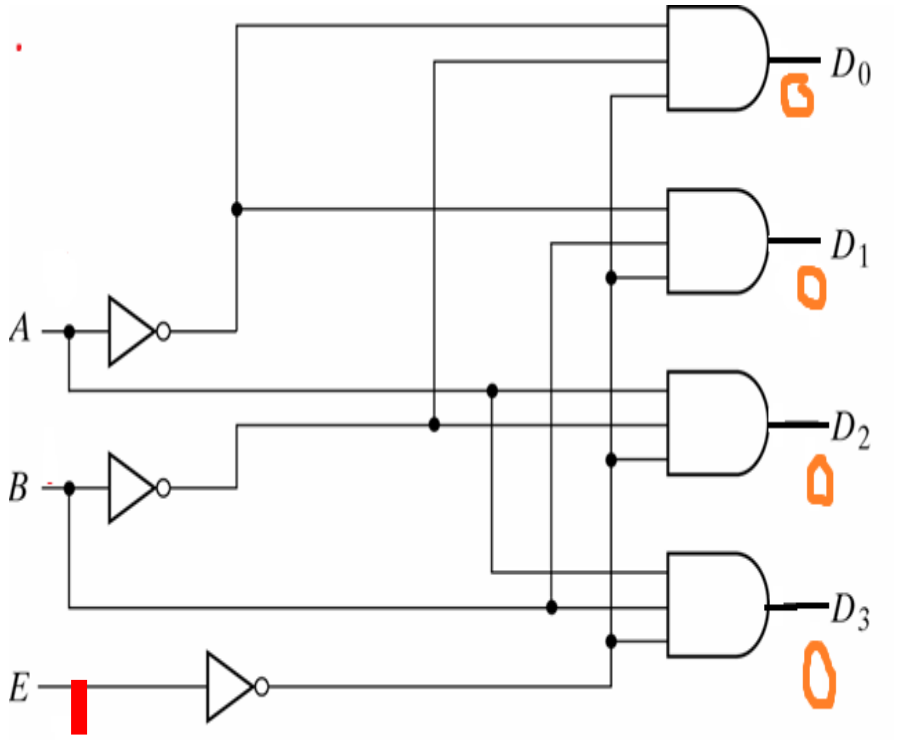
TT OF NAND GATE

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



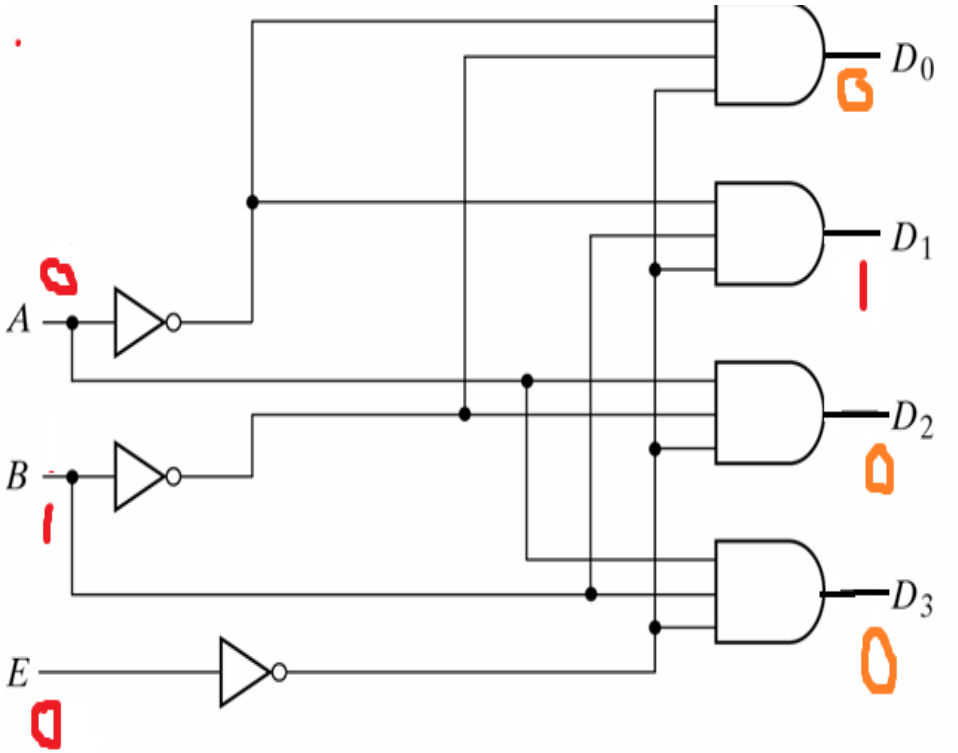
2-to-4-Line Decoder (with Active Low Enable input)-Active High output

Ckt. Is not operated/On Reset

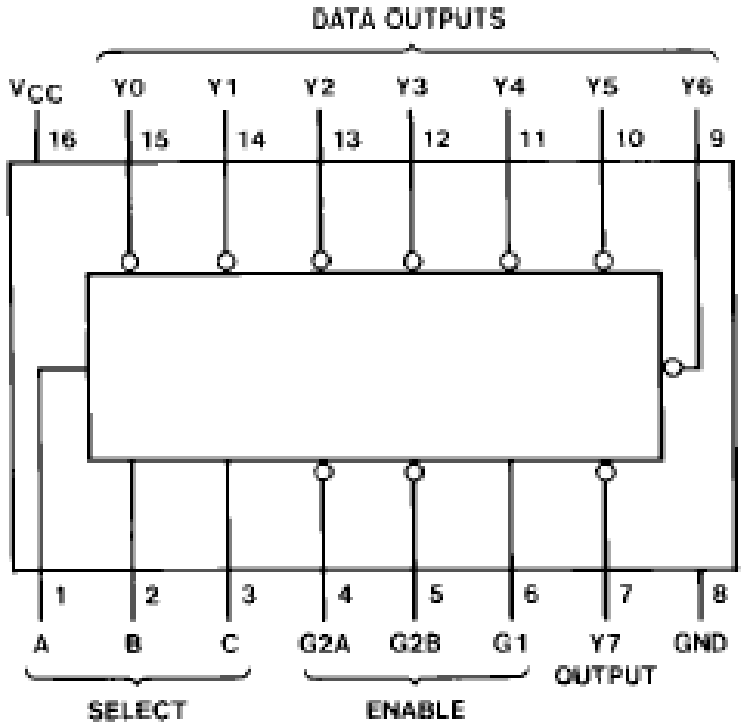


TT OF AND GATE

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



74138 Decoder



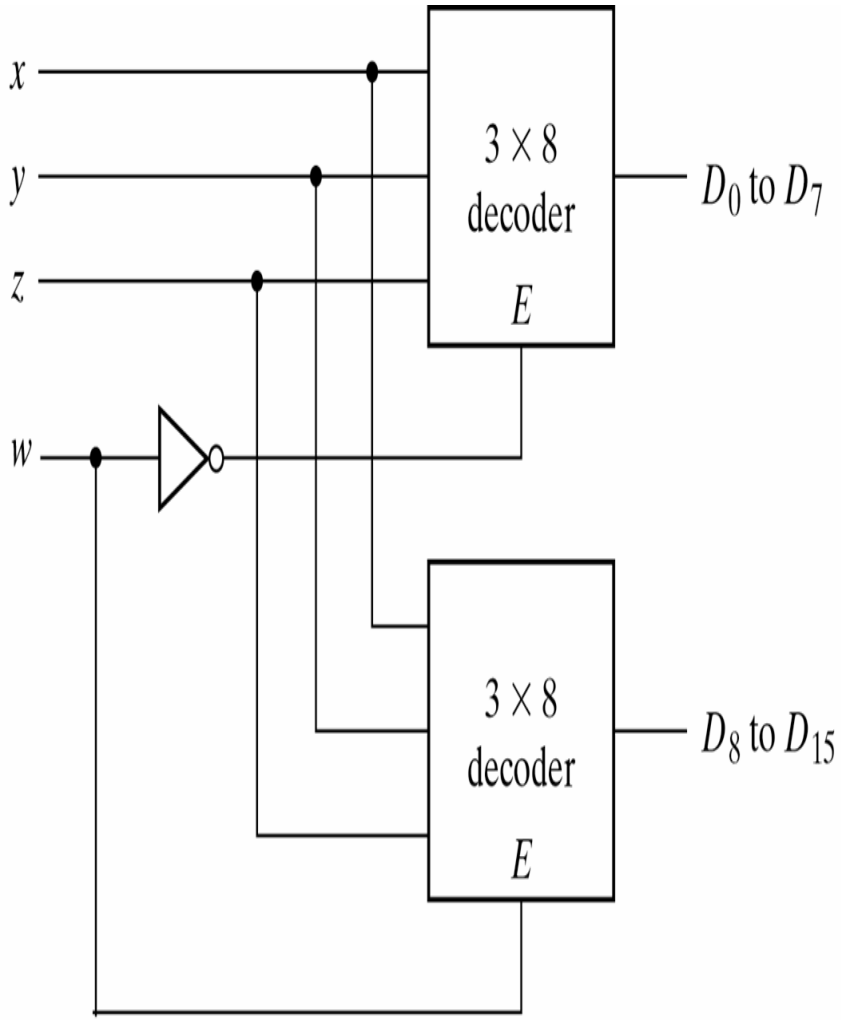
H = HIGH Level
L = LOW Level
X = Don't Care

Note 1: G2 = G2A + G2B

Function Table

Inputs					Outputs							
Enable		Select										
G1	G2 (Note 1)	C	B	A	YO	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

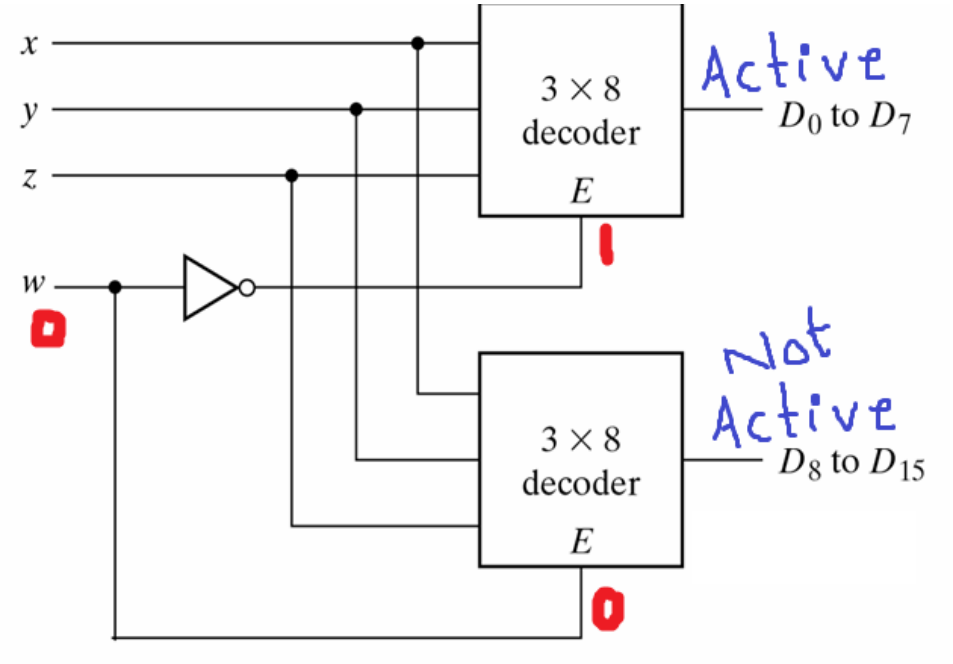
4-line-to-16 line Decoder constructed with two 3-line-to-8 line decoders



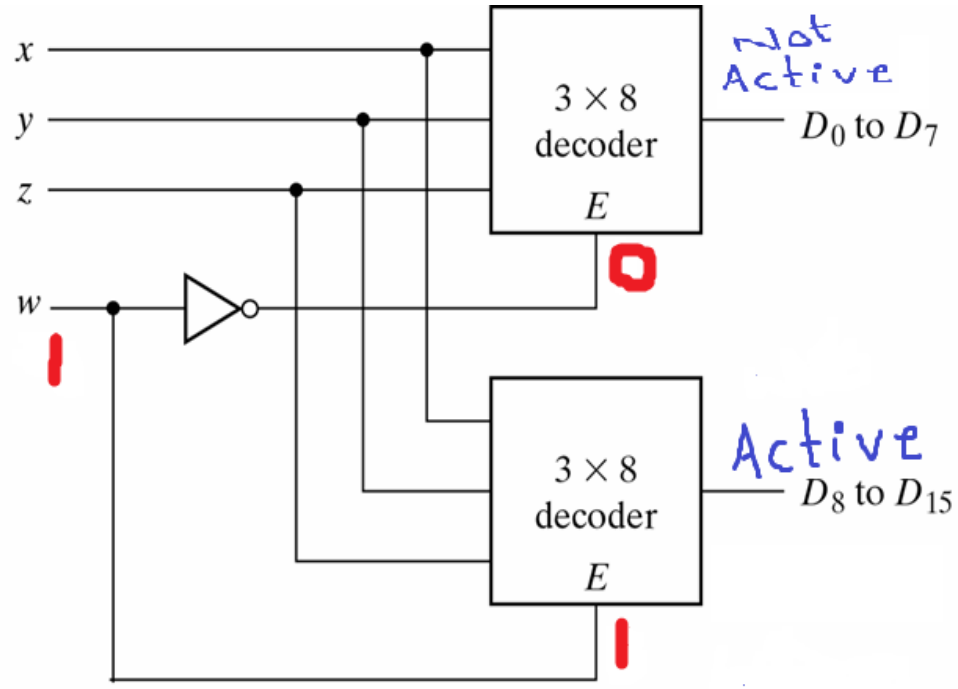
W	X	Y	Z	O/P
0	0	0	0	D0
0	0	0	1	D1
0	0	1	0	D2
0	0	1	1	D3
0	1	0	0	D4
0	1	0	1	D5
0	1	1	0	D6
0	1	1	1	D7
1	0	0	0	D8
1	0	0	1	D9
1	0	1	0	D10
1	0	1	1	D11
1	1	0	0	D12
1	1	0	1	D13
1	1	1	0	D14
1	1	1	1	D15

- When **w=0**, the **top decoder is enabled** and the other is disabled. The bottom decoder outputs are all 0's , and the top eight outputs generate min-terms **0000 to 0111**.
- When **w=1,** the enable conditions are reversed. The **bottom decoder** outputs generate min-terms **1000 to 1111**, while the outputs of the top decoder are all 0's

4-line-to-16 line Decoder constructed with two 3-line-to-8 line decoders



- When **w=0**, the **top decoder is enabled** and the other is disabled. The bottom decoder outputs are all 0's , and the top eight outputs generate min-terms **0000 to 0111**.



- When **w=1,** the enable conditions are reversed. The **bottom decoder** outputs generate min-terms **1000 to 1111**, while the outputs of the top decoder are all 0's

Full Adder Implementation using Decoder

- Let x, y and z represent these three bits. Sum and Carry outputs of a full adder have the following truth tables-

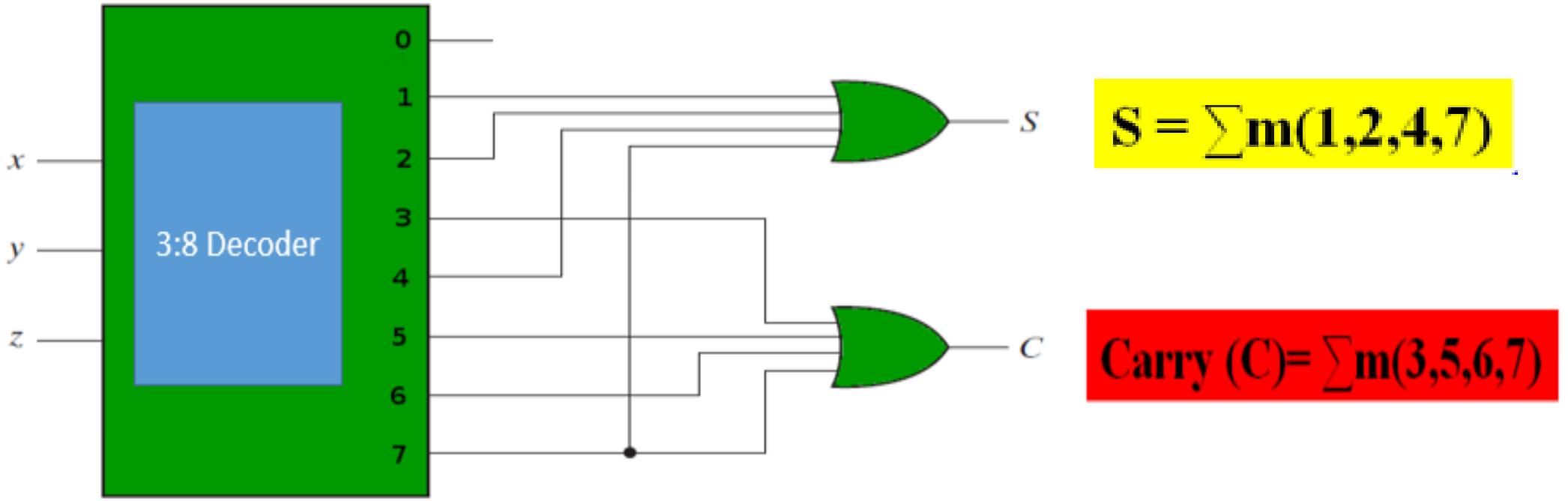
X	Y	Z	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = \sum m(1,2,4,7)$$

$$\text{Carry } (C) = \sum m(3,5,6,7)$$

Full Adder Implementation using Decoder

- Let x, y and z represent these three bits. Sum and Carry outputs of a full adder can be given as:
- Assumption: **Active high output decoder**



S = ∑m(1,2,4,7)

Carry (C)= ∑m(3,5,6,7)