RV College of Engineering®

UNIT II
Combinational Circuit Design

Go, change the world®

Syllabus

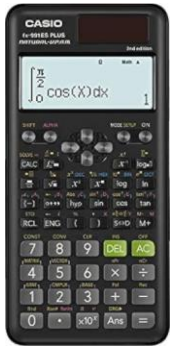
Combinational Circuits Design: Arithmetic circuits, code converters, and logic functions implementation using Decoders/ Demultiplexers and Multiplexers. Design of a Priority encoder, Magnitude comparator, Parallel Adder/Subtractor, Concepts of ripple carry and carry look-ahead adders and BCD adder.

Dataflow/ Behavioral/ Structural Modelling: Verilog Data flow/ Behavioral/ Structural Models, Module Ports, Top-Down Design and Nested Modules.

Digital Systems

- A digital system is an electronic system that processes information in digital form (binary values: 0 and 1).
- **0s and 1s are interpreted as:**
  - Logic 0 (LOW) → 0V (or close to 0)
  - Logic 1 (HIGH) → +5V(TTL*), +3.3V(CMOS*)

Examples of large-scale digital systems in the real world:



*TTL: Transistor to Transistor Logic
*CMOS: Complementary Metal–Oxide–Semiconductor

Digital Circuits

- Digital systems are built using digital circuits (logic gates, combinational and sequential blocks).
- **Combinational Circuits**

  Output depends only on the current input.

  Examples: Adders, Subtractors, Encoders, Decoders, Multiplexers, Comparators.
- **Sequential Circuits**

  Output depends on the current input and the past state of the circuit.

  Examples: Flip-flops, Registers, Counters, Shift Registers.
- Parameters to be considered while designing digital circuits: Power dissipation, Area, and Propagation delay.

Arithmetic Circuits

- Arithmetic circuits are a special type of combinational digital circuit that perform mathematical operations(addition, subtraction, multiplication, and division) on binary numbers.
- They form the core of arithmetic logic units (ALUs) inside microprocessors(Intel i3,i5,i7,etc) and microcontrollers(ATMEGA328P on Arduino board)
- Arithmetic circuits are built using logic gates (AND, OR, XOR, NOT, NAND, NOR).

Half Adder

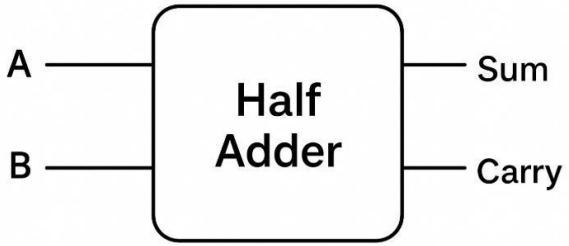
- It performs the addition of two single-bit binary numbers (A and B).
- **Inputs:** A, B (1-bit each)
- **Outputs:**

  Sum (S): Result (A XOR B).

  Carry (C): Carry output (A AND C).
- Logic Expressions:

  Sum (S) = $A \oplus B$ (XOR gate)

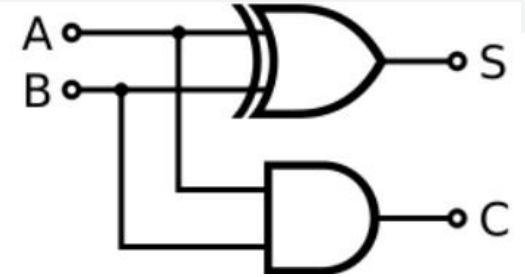
  Carry (C) = $A \cdot B$ (AND gate)



Truth table

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

Circuit diagram



Full Adder

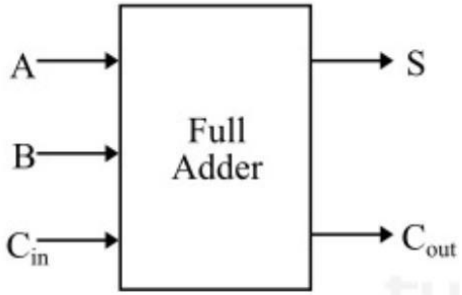
- A Full Adder is a combinational logic circuit that adds **three binary inputs**:

  A: First bit, **B:** Second bit, **Cin:** Carry input (from the previous stage)

- **Outputs:** Sum (S), Carry out (Cout)
- **Logic Expressions:**

$$\text{Sum (S)} = A \oplus B \oplus Cin$$

$$\text{Carry (Cout)} = (A \cdot B)+(B \cdot Cin)+(A \cdot Cin)$$



Truth table

A	B	Cin	Sum (S)	Carry (Cout)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full Adder

- **SOP expression from truth table:**

$$S = A'B'Cin + A'BCin' + AB'Cin' + ABCin$$

$$= A \oplus B \oplus Cin$$

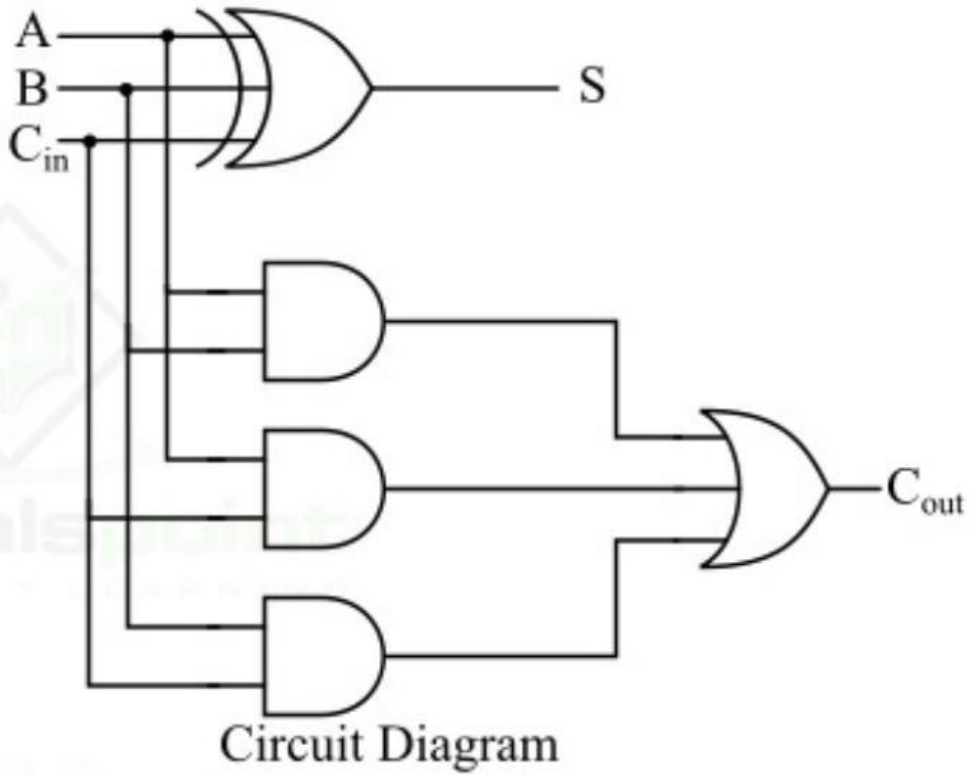
$$Cout = A'BCin + AB'Cin + ABCin' + ABCin$$

$$= Cin(A'B + AB') + AB(Cin' + Cin)$$

$$= Cin(A \oplus B) + AB$$

$$= AB + ACin + BCin$$

Circuit diagram



Circuit Diagram

NAND gate realization of Half Adder

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

= $A\overline{B} + \overline{A}B + A\overline{A} + B\overline{B}$ Because $A\overline{A} = 0$

=$A\overline{B} + A\overline{A} + \overline{A}B + B\overline{B}$ Because $B\overline{B} = 0$

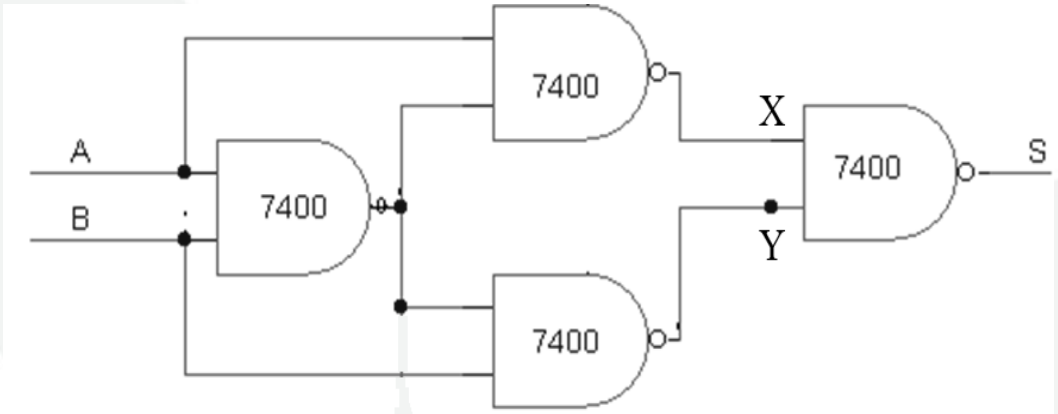
=$A(\overline{B} + \overline{A}) + B(\overline{A} + \overline{B})$

=$A(\overline{AB}) + B(\overline{AB})$ By De-Morgan's Theorem

=$\overline{\overline{A(\overline{AB}) + B(\overline{AB})}}$ Because $\overline{\overline{A}}$=A

Sum=$\overline{\overline{A(\overline{AB})}.\overline{B(\overline{AB})}}$ By De-Morgan's Theorem

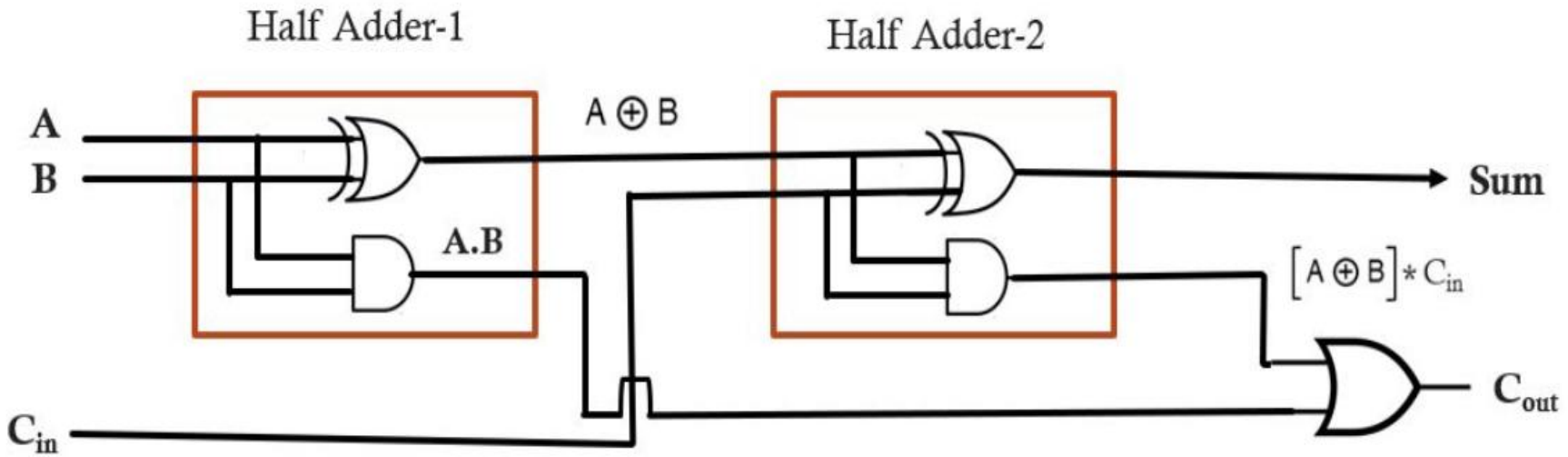
Circuit diagram



Question 1

- Realize a full adder using half adders
- Realize a full adder using NAND gates only.

Full adder using two Half Adders



$S = A' B' C_{in} + A' B C'_{in} + A B' C'_{in} + A B C_{in}$

$= C_{in} (A' B' + A B) + C'_{in} (A' B + A B')$

$= C_{in}$ (A Ex-NOR B) + C'_{in} (A Ex-OR B)

$= C_{in} (A \oplus B)' + C'_{in} (A \oplus B)$

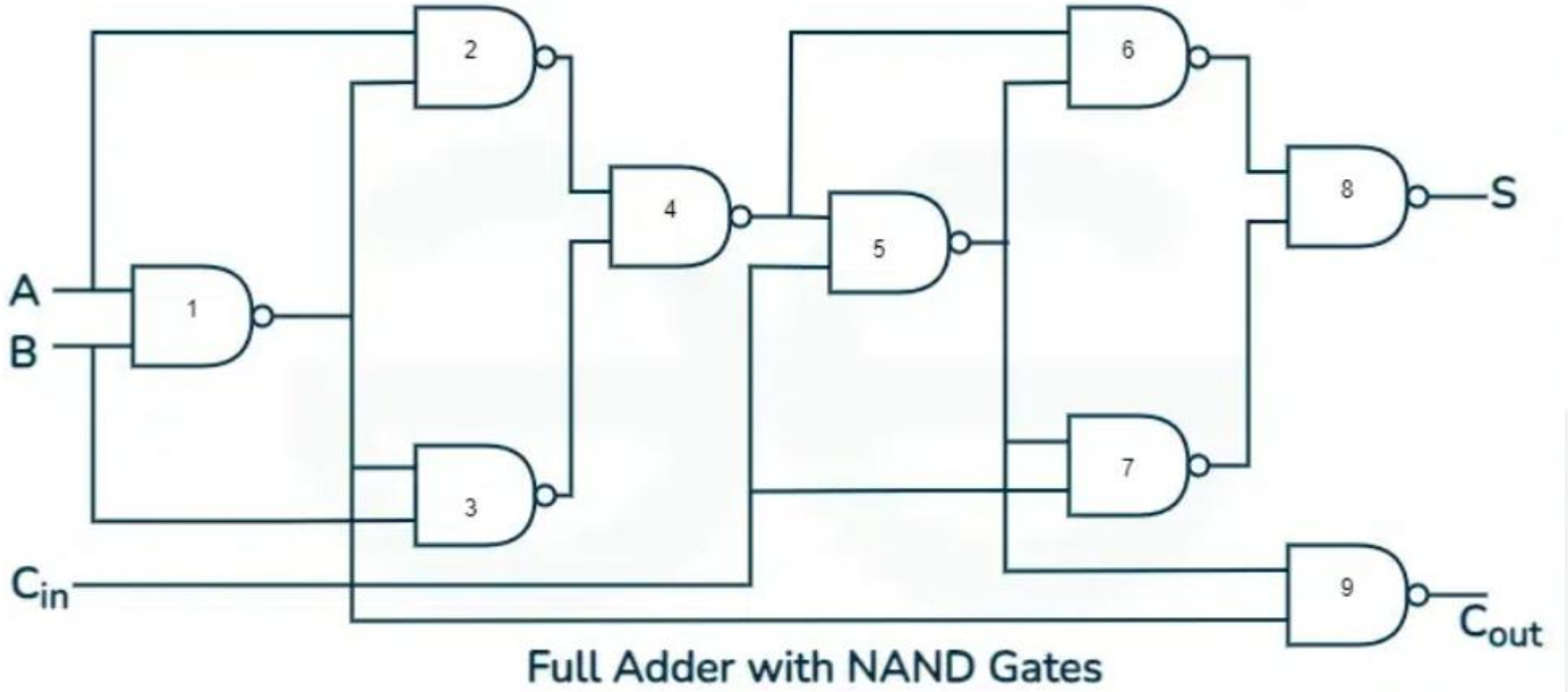
Therefore, $S = C_{in} \oplus (A \oplus B) = A \oplus B \oplus C_{in}$

$C_O = A B + A C_{in} + B C_{in}$

$= A B + C_{in} (A B' + A' B)$

Therefore, $C_O = A B + C_{in} (A \oplus B)$

Full Adder using only NAND gates



Full Adder with NAND Gates

Half Subtractor

- It subtracts two single-bit binary numbers:

  Inputs: A (minuend), B (subtrahend)

  Outputs: Difference (D) and Borrow (B)

- Logic Expressions:

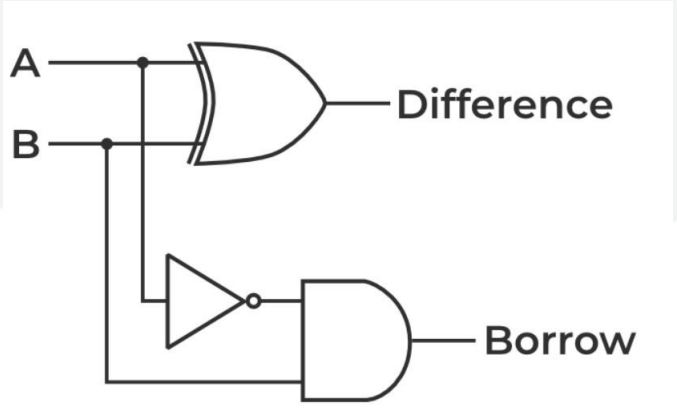
  Difference(D) = $A \oplus B$ (XOR gate)

  Borrow(B) = $A' \cdot B$

Truth table

A	B	Diff (D)	Borrow (B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Circuit diagram



Full Subtractor

- A Full Subtractor is a combinational circuit that subtracts three input bits:
- Inputs: A (minuend), B (subtrahend), Bin (borrow in)
- Outputs: Difference (D), Borrow out (Bout)
- Logical Expressions:

  Difference: $D = A \oplus B \oplus Bin$

  Borrow out : $Bout = \overline{B} \cdot A + Bin \cdot (\overline{A \oplus B})$

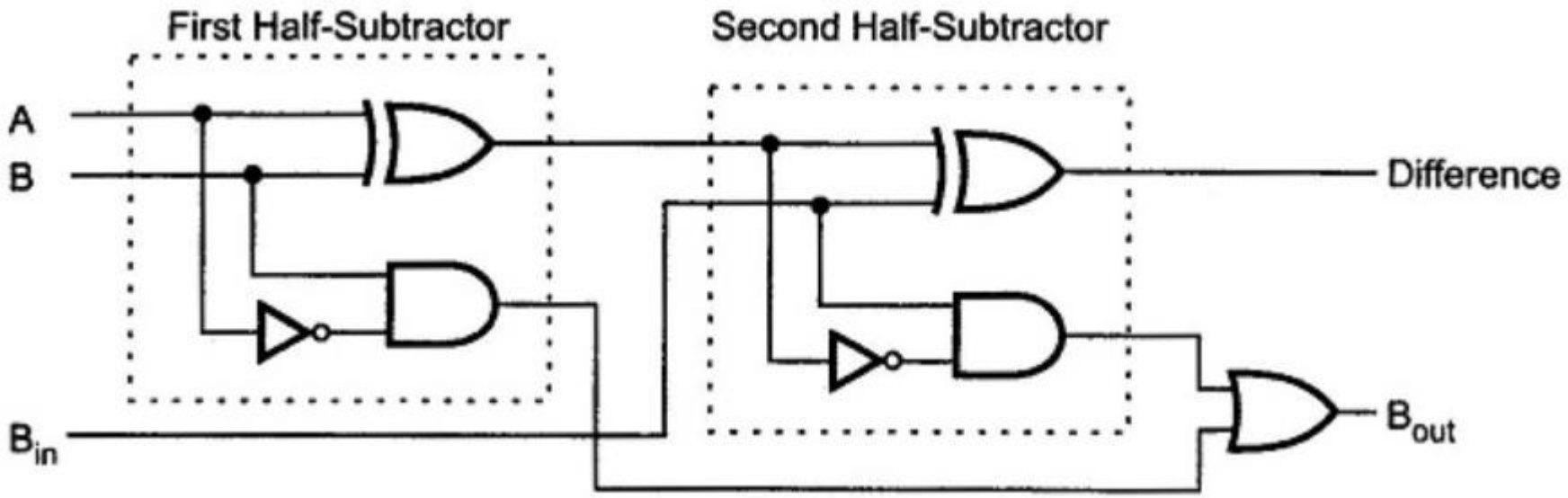
Truth table

A	B	Bin	Diff (D)	Borrow (Bout)
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

Question 2

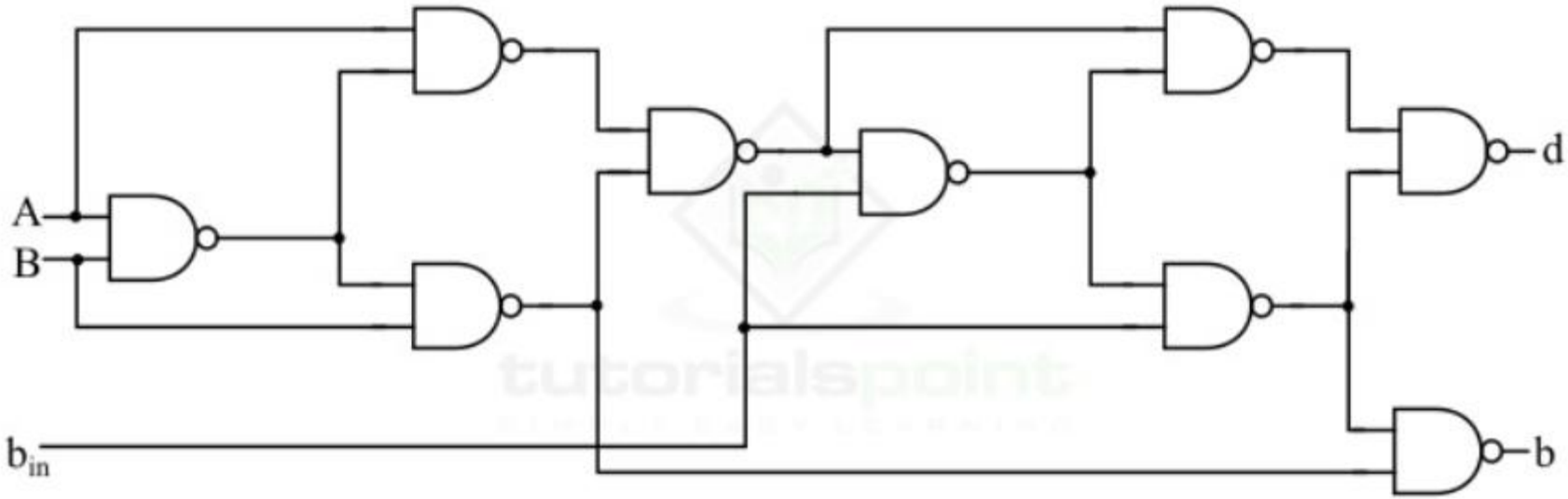
- Design a full subtractor circuit using only NAND gates.
- Design a 4-bit parallel subtractor using full subtractors.

Full Subtractor using two Half subtractors



$$
\begin{aligned}
B_{out} &= \bar{A}B+\overline{(A\bar{B}+\bar{A}B)}\,B_{in} = \bar{A}B+(AB+\bar{A}\bar{B})\,B_{in} \\
&= \bar{A}B+ABB_{in}+\bar{A}\bar{B}B_{in} \\
&= \bar{A}B(1+B_{in})+ABB_{in}+\bar{A}\bar{B}B_{in} && \because (1+B_{in}) = 1 \\
&= \bar{A}B+\bar{A}BB_{in}+ABB_{in}+\bar{A}\bar{B}B_{in} = \bar{A}B+BB_{in}(\bar{A}+A)+\bar{A}\bar{B}B_{in} \\
&= \bar{A}B+BB_{in}+\bar{A}\bar{B}B_{in} \\
&= \bar{A}B(1+B_{in})+BB_{in}+\bar{A}\bar{B}B_{in} && \because (1+B_{in}) = 1 \\
&= \bar{A}B+\bar{A}BB_{in}+BB_{in}+\bar{A}\bar{B}B_{in} = \bar{A}B+\bar{A}B_{in}(B+\bar{B})+BB_{in} \\
&= \bar{A}B+\bar{A}B_{in}+BB_{in}
\end{aligned}
$$

Full Subtractor using NAND gates



Borrow Bit (b)

$$\text{Borrow, } b = \overline{\overline{B \cdot \overline{AB}} \cdot \overline{b_{in}[b_{in} \cdot (A \oplus B)]}} = \overline{A}B + b_{in}(\overline{A \oplus B})$$

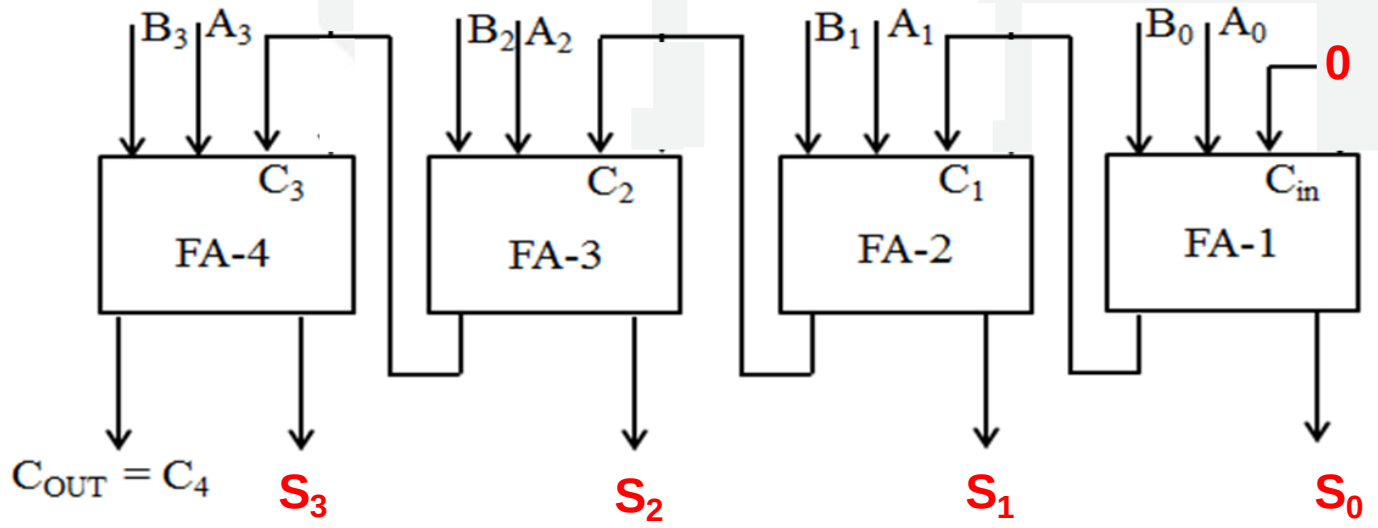
Parallel Adder

- It is used to add two multi-bit binary numbers simultaneously(in parallel). It is designed using multiple full adders connected together.
- **4-bit Parallel Adder:**

Inputs: Two 4-bit binary numbers A = $A_3A_2A_1A_0$ and B = $B_3B_2B_1B_0$

Output: 4-bit Sum = $S_3S_2S_1S_0$ and a final Carry (C_{OUT})

Block diagram



A_0, B_0 & S_0: Least significant bits

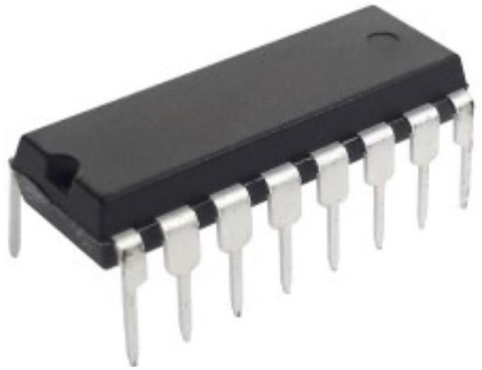
A_3, B_3 & S_3: Most significant bits

Parallel Adder

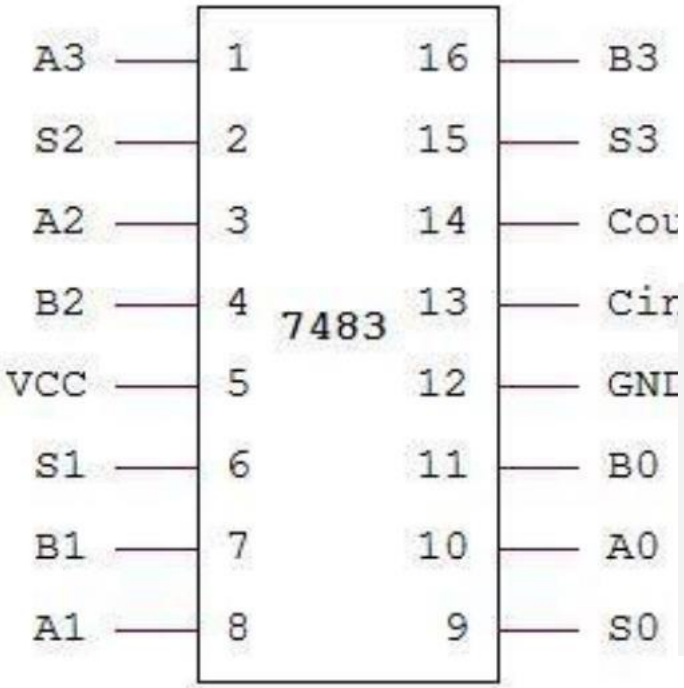
- Each full adder adds two input bits (A, B) and a carry-in (C_{in}).
- The carry-out (C_{OUT}) of each full adder is passed to the carry-in of the next higher-order full adder.
- The carry signal "**ripples**" through the adder chain from the least significant bit (LSB) to the most significant bit (MSB). This is called as **Ripple Carry Adder(RCA).**
- As each carry must ripple through all full adders, making it slow for large numbers (e.g., 32-bit or 64-bit).
- **Delay** ≈ (number of bits) × (delay of one full adder).

Question 3: Design a 4-bit binary adder/subtractor using TTL IC 7483

- The IC 7483 is a standard 4-bit binary parallel adder integrated circuit from the 74xx TTL logic family.
- **IC 7483 pin diagram:**



Dual in Line Package(DIP)



4-bit binary adder

Example 1: 7+2=9 (1001)

7 is realized at $A_3A_2A_1A_0 = 0111$

2 is realized at $B_3B_2B_1B_0 = 0010$

Sum $S_3S_2S_1S_0 = 1001$

Example 2: 9+9=18(10010)

9 is realized at $A_3A_2A_1A_0 = 1001$

2 is realized at $B_3B_2B_1B_0 = 1001$

Sum $S_3S_2S_1S_0 = 0010$

with $C_{OUT} = 1$

4-bit binary subtractor

Subtraction is carried out by adding the 2's complement of the subtrahend.

Example 1: 8 – 3 = 5 (0101)

1's complement of 3 = 1100

Add 1 to get 2's complement=1100+1=1101

8 is realized at $A_3A_2A_1A_0$ = 1000

2's complement of 3 = 1101

Difference($S_3S_2S_1S_0$) = 0101 **(with C_{OUT} = 1 ignored)**

4-bit binary subtractor

- Subtraction is carried out by adding the 2's complement of the subtrahend.
- Note that negative numbers are represented in 2's complement

Example 2: 5– 6 = -1 (2's complement:1111)

1's complement of 6 = 1001

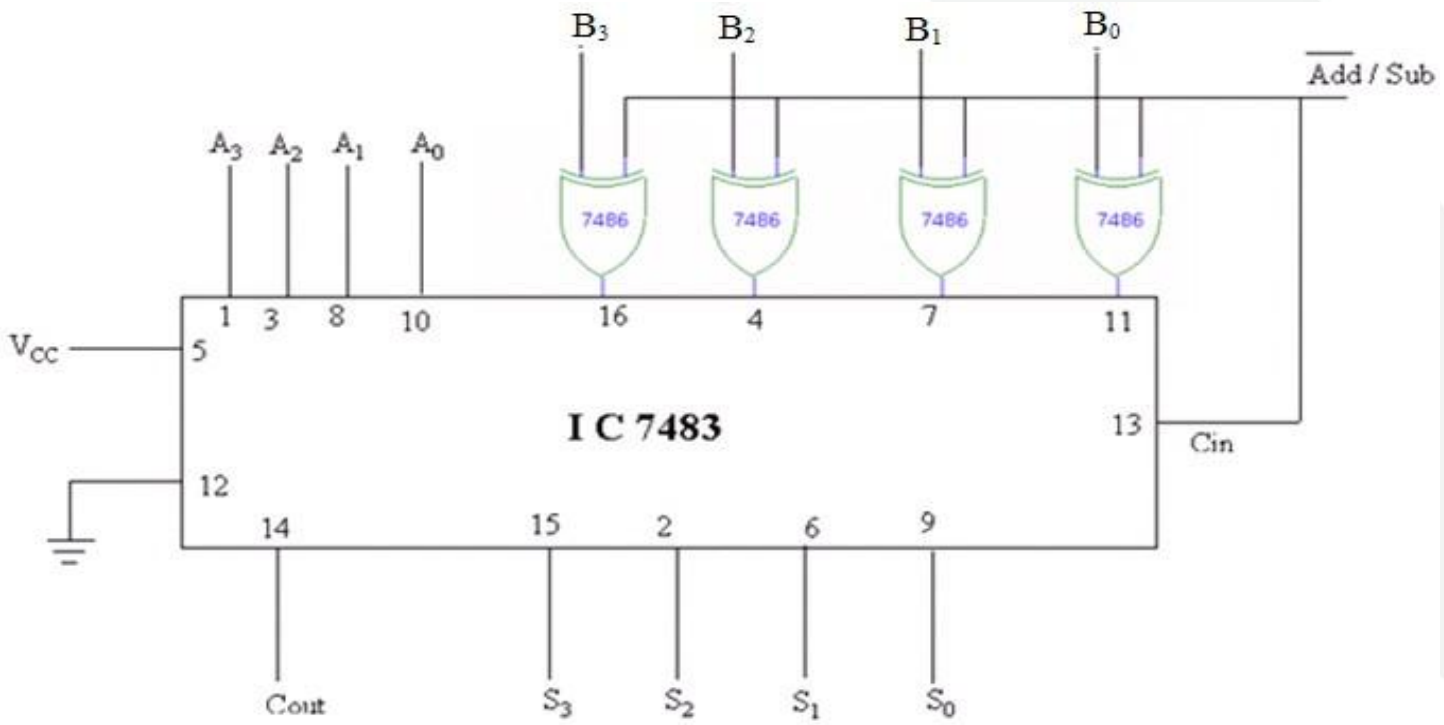
Add 1 to get 2's complement=1001+1=1010

5 is realized at $A_3A_2A_1A_0$ = 0101

2's complement of 6 = 1010

Difference($S_3S_2S_1S_0$) = 1111**(with C_{OUT} = 0 result is negative in 2's complement)**

4-bit binary adder/subtractor using 7483



- If the control input **ADD'/SUB =0,** circuit performs addition.

Question 4: Design a BCD adder using IC 7483

- A BCD adder adds two BCD digits and produces output as a BCD digit. A BCD or Binary Coded Decimal digit cannot be greater than 9.
- The two BCD digits are to be added using the rules of binary addition. If the sum is less than or equal to 9 and carry is 0, then no correction is needed. The sum is correct and in true BCD form.
- But if the sum is greater than 9 or carry =1, the result is wrong and correction must be done. The wrong result can be corrected adding six (0110) to it.

BCD adder

- For implementing a BCD adder using a binary adder circuit IC 7483, an additional combinational circuit will be required, where the Sum output S_3–S_0 is checked for invalid values from 10 to 15.
- Using a K-map, the Boolean expression is obtained, $Y=S_3S_2+S_3S_1$

I/P				O/p
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

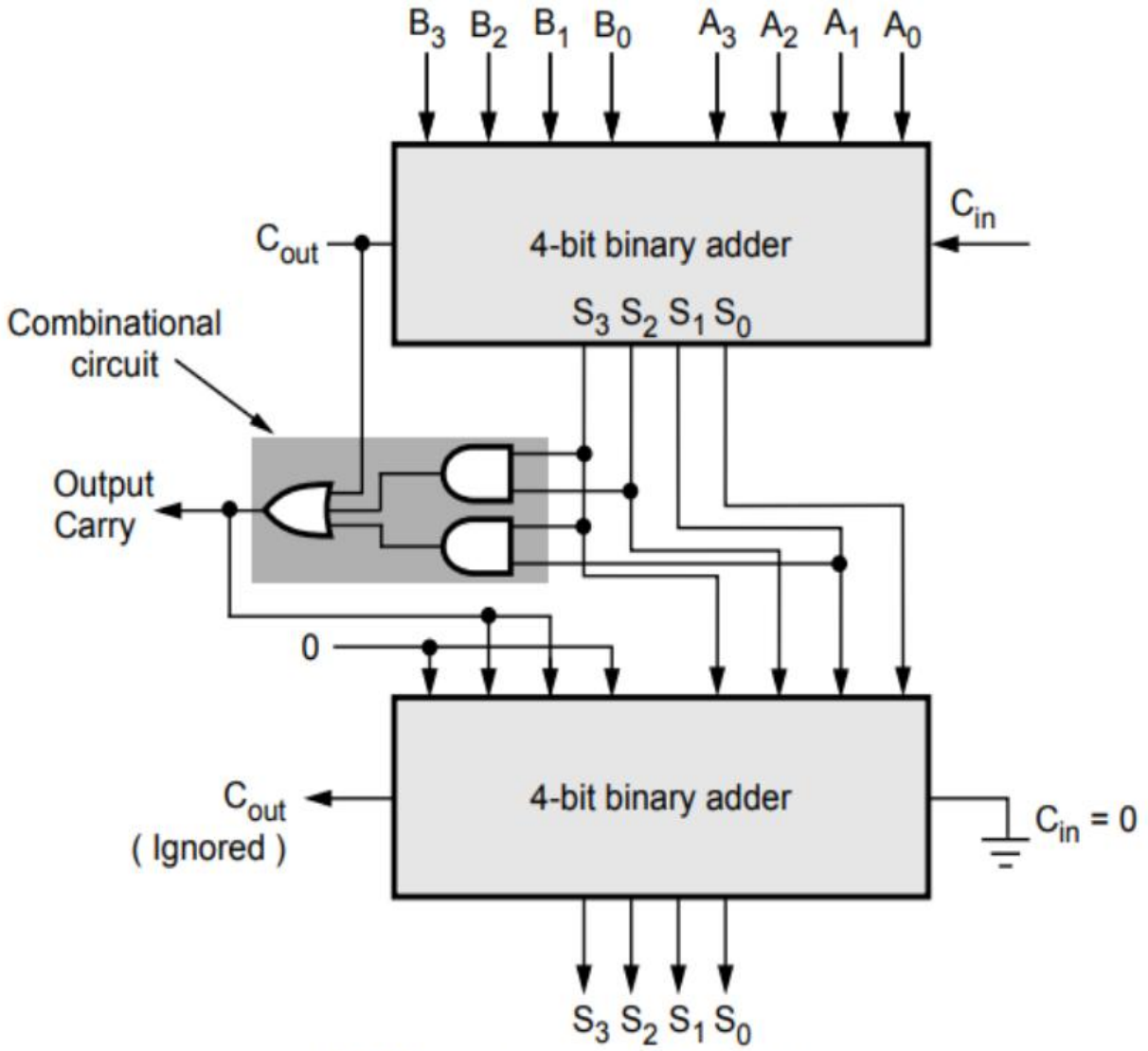
Invalid O/P: Y=1

BCD adder: Circuit Diagram

S_3S_2 \ S_1S_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

$Y = S_3S_2 + S_3S_1$

(a) K-map simplification



(b) Block diagram of BCD adder

=0

Carry Look Ahead Adder

- A **Carry Look-Ahead Adder (CLA)** is a fast adder circuit that improves upon the Ripple Carry Adder (RCA) by reducing the carry propagation delay.
- In a CLA, the carry signals are generated in parallel using **generate** and **propagate** functions, greatly speeding up the addition.
- **Carry Generate (G):** A carry is generated at bit i if both inputs are 1.

$$G_i = A_i \cdot B_i$$

Example: $Ai=1, Bi=1$: Sum will be 0, Carry = 1 (regardless of incoming carry).

- **Carry Propagate (P):** A carry is propagated through bit i if either A or B is 1.

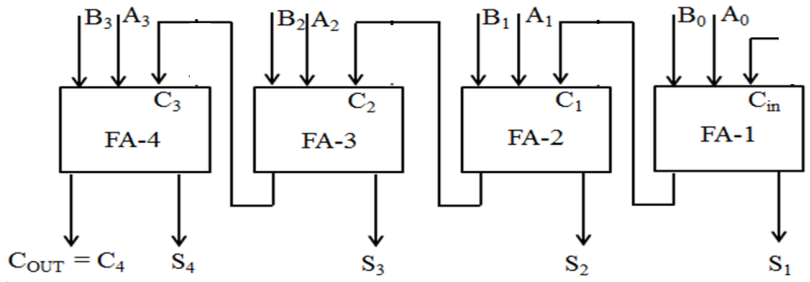
$$P_i = A_i \oplus B_i$$

If a carry comes in, it will pass to the next stage.

Carry Look Ahead Adder

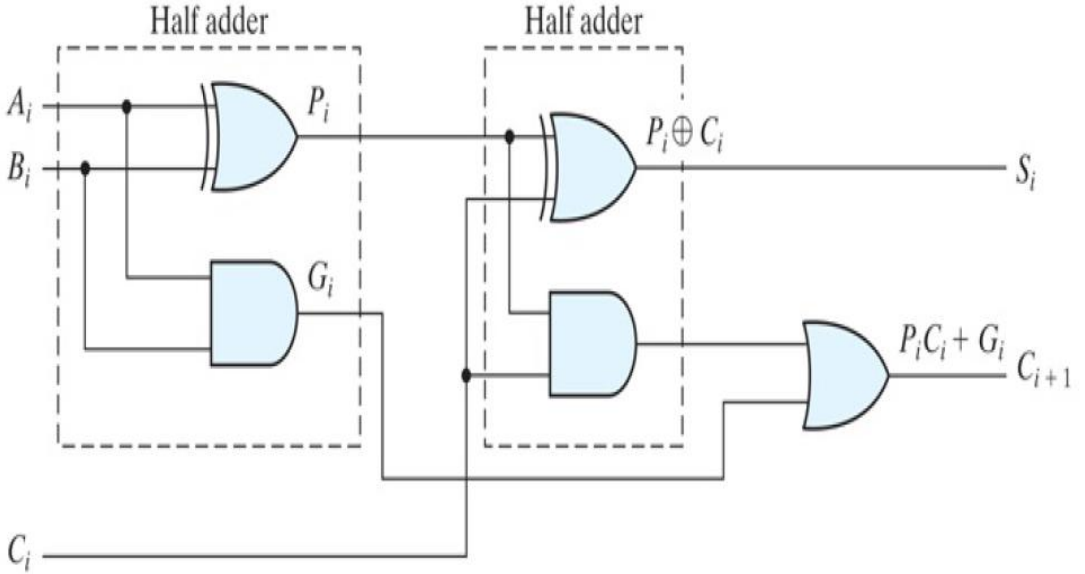
- Consider a 4-bit binary adder with two inputs:

  $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$



- Referring to the RCA shown in fig 1 intermediate carry equations are given as:

$$C_1 = G_0 + (P_0 \cdot C_0)$$
$$C_2 = G_1 + (P_1 \cdot C_1)$$
$$C_3 = G_2 + (P_2 \cdot C_2)$$



- Expanding in terms of G and P:

$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1G_0 + P_1P_0C_0$$
$$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$
$$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

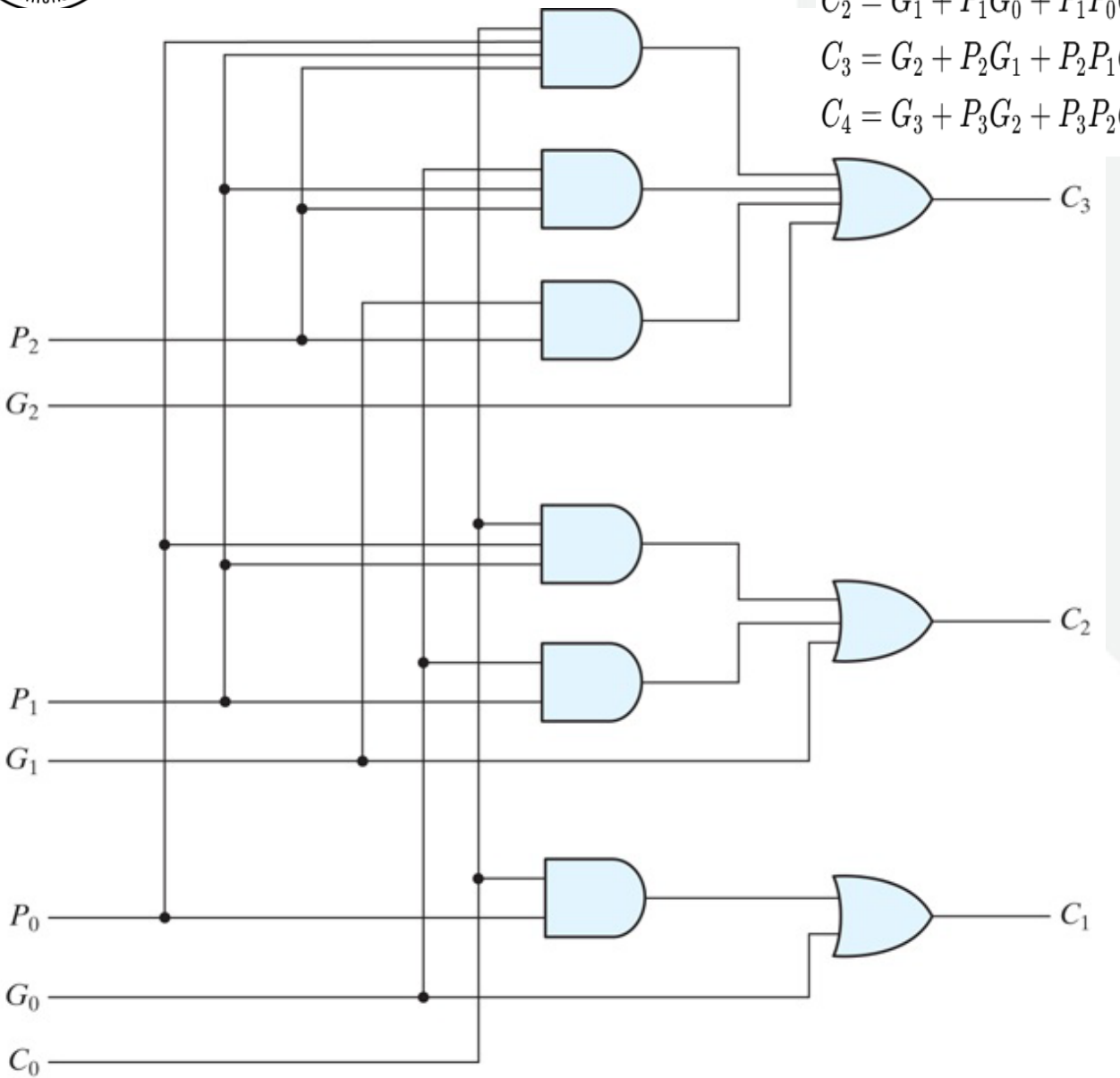
- These can all be computed in parallel instead of waiting for ripple effect.

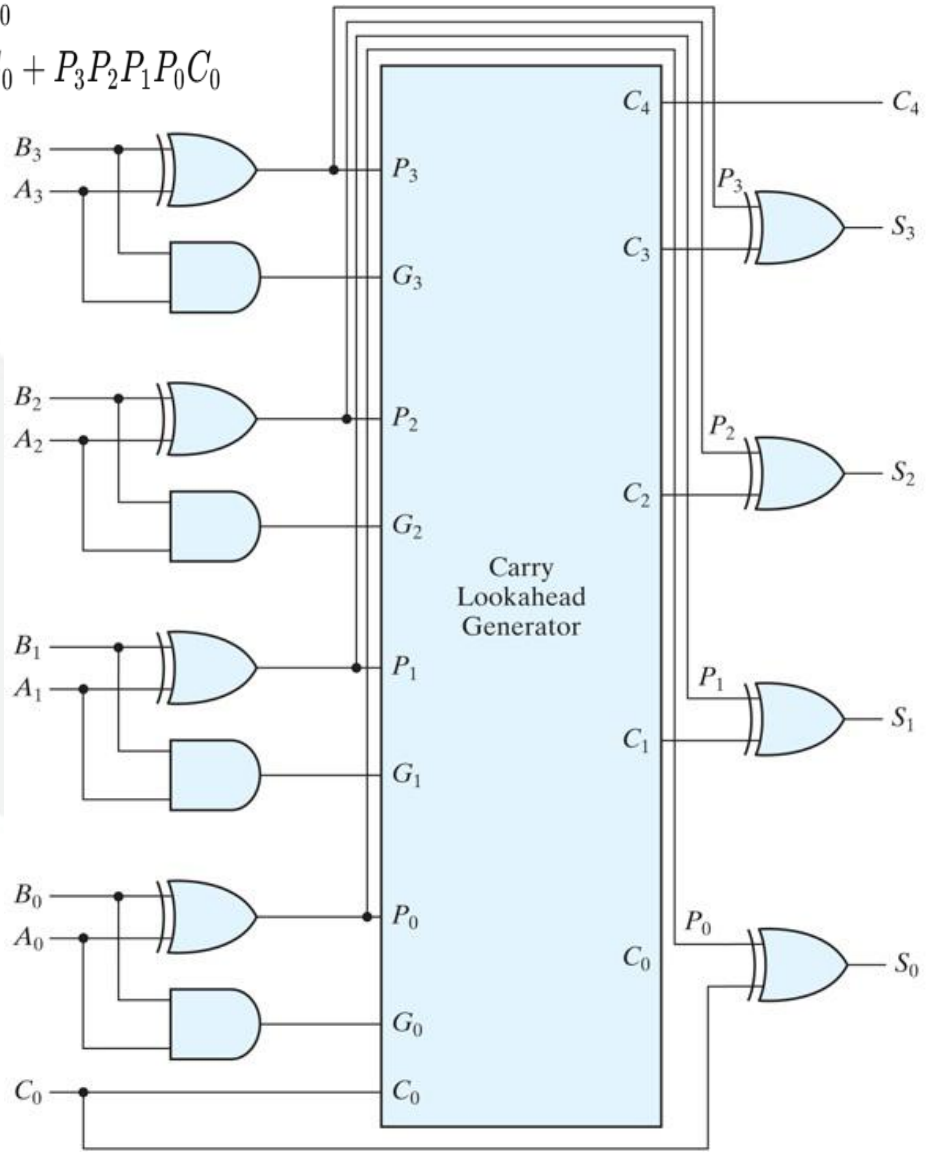
RV College of Engineering®

Logic circuit of Carry Look Ahead Adder

$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1G_0 + P_1P_0C_0$$
$$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$
$$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

Go, change the world®





Merits and Demerits of CLA Adder

- Since the Boolean function for each output carry is expressed in sum-of products form, each function can be implemented with one level of AND gates followed by an OR gate (or by a two-level NAND).
- The three Boolean functions for C 1 , C 2 , and C 3 are implemented in the carry look ahead generator shown in Fig. (previous slide).
- This circuit can add in less time because C 3 does not have to wait for C 2 and C 1 to propagate; in fact, C 3 is propagated at the same time as C 1 and C 2 .
- This gain in speed of operation is achieved at the expense of additional complexity (hardware).

Code Converter: Binary to Gray

- In normal binary, several bits may change at once (e.g., 3 → 4: 011 → 100 → three bits change).
- Gray code avoids this since only one-bit changes per step.
- This is useful in shaft encoders, Karnaugh maps, error detection, and digital communications.
- For binary bits $B_3B_2B_1B_0$→ Gray bits $G_3G_2G_1G_0$:

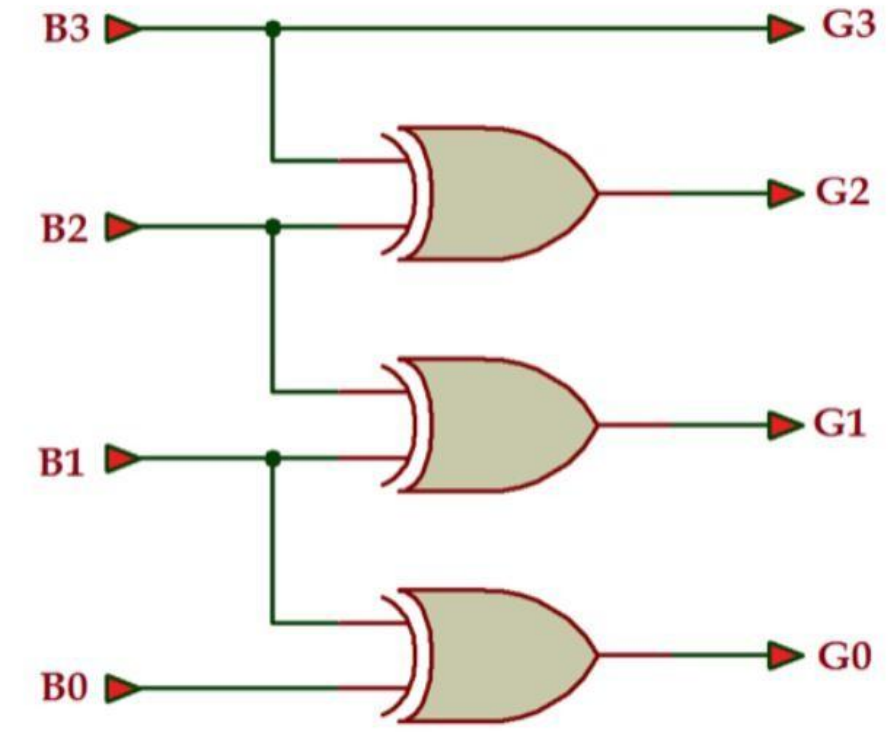
$$G_3 = B_3$$
$$G_2 = B_3 \oplus B_2$$
$$G_1 = B_2 \oplus B_1$$
$$G_0 = B_1 \oplus B_0$$

- Gray codes are widely used in shaft encoders, Karnaugh maps, error detection, and digital communications.
- Advantagesof gray code: https://www.youtube.com/watch?v=W730NOJYXAI&t=2s

Truth table

Binary	Gray
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Logic diagram



Code Converter: BCD to Excess-3

- XS-3 is just BCD + 3.
- Unlike BCD, XS-3 avoids all-zeros (0000)
- its self-complementary nature, which simplifies subtraction, and its simplification of arithmetic operations, especially addition, by eliminating the need for correction after a sum exceeds 9

Truth table

Decimal Digit	BCD (8421)	XS-3 Code (BCD + 3)
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

RV College of Engineering®

Logic Circuit of BCD to Excess-3 Code Converter-Truth Table

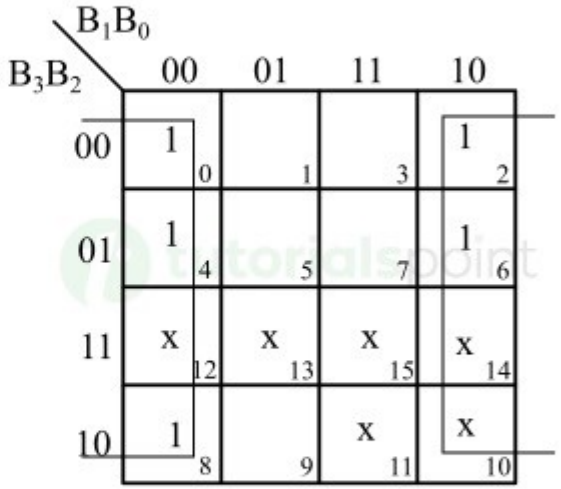
Go, change the world®

BCD Code				Excess-3 Code			
B_3	B_2	B_1	B_0	X_3	X_2	X_1	X_0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Kmap for X0 and X1 output

K-Map for XS-3 Bit X_0

The K-map simplification for the XS-3 bit X_0 is shown in the following figure –

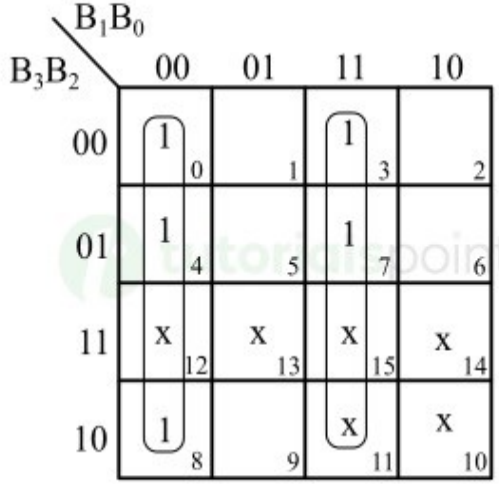


On simplifying this K-map, we obtain the following Boolean expression,

$$X_0 = \overline{B_0}$$

K-Map for XS-3 Bit X_1

The K-map simplification for the XS-3 bit X_1 is depicted below –



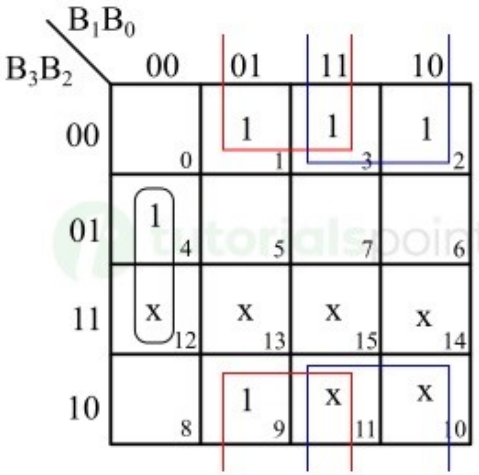
This K-map simplification gives the following Boolean expression,

$$X_1 = \overline{B_1}\,\overline{B_0} + B_1\,B_0$$

Kmap for X2 and X3 output

K-Map for XS-3 Bit X_2

The K-map simplification for the XS-3 bit X_2 is shown in the figure below.

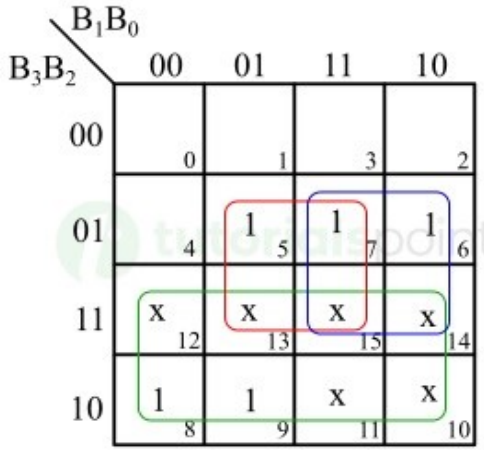


On simplifying this K-map, we obtain the following Boolean expression,

$$X_2 = \overline{B_2}\, B_1 + \overline{B_2}\, B_0 + B_2\, \overline{B_1}\, \overline{B_0}$$

K-Map for XS-3 Bit X_3

The K-map simplification for the XS-3 bit X_3 is depicted in the figure below –

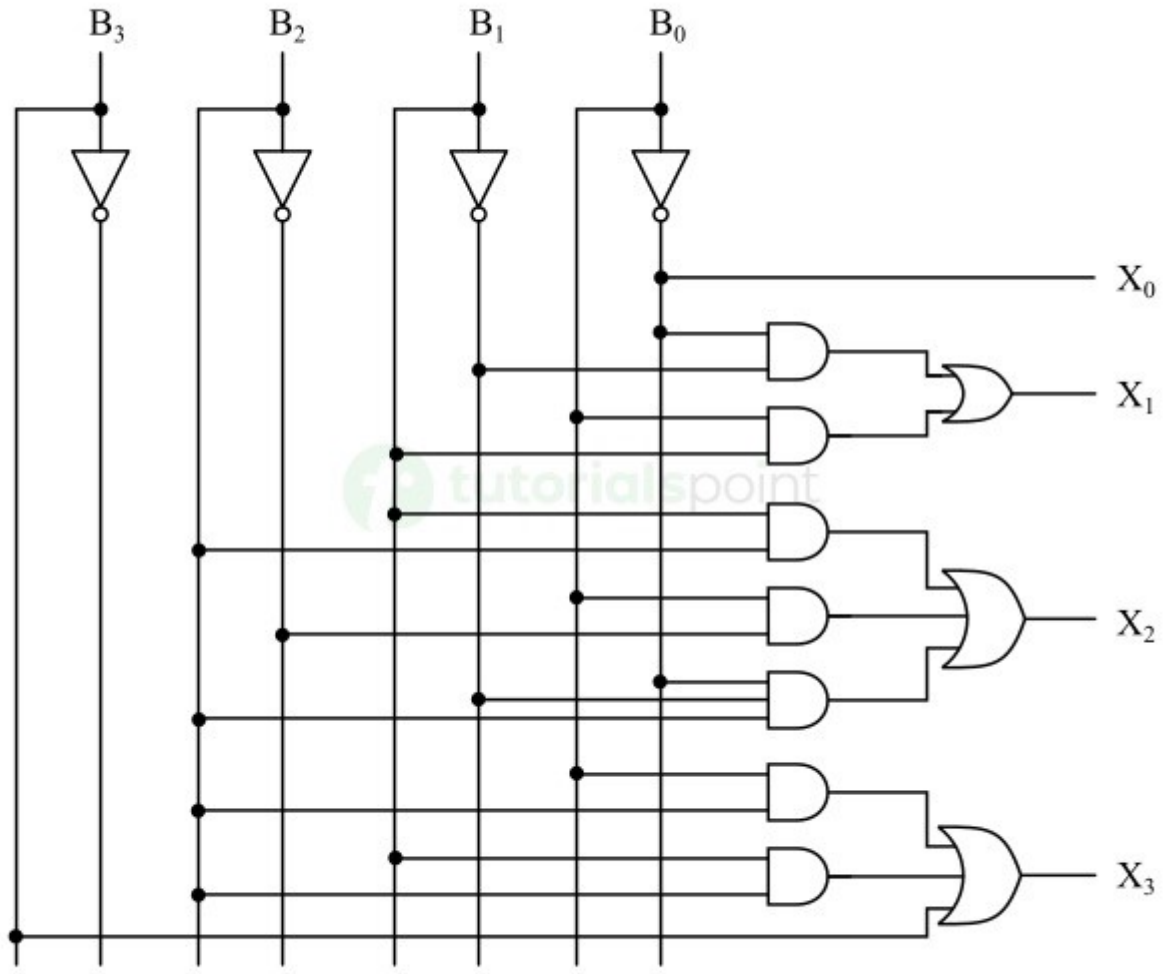


This K-map gives the following Boolean expression,

$$X_3 = B_3 + B_2\, B_1 + B_2\, B_0$$

Logic Circuit of BCD to XS-3 Code Converter

The logic circuit diagram of the BCD to XS-3 converter is shown in the following figure –



This circuit converters a 4-bit BCD code into an equivalent XS-3 code.

Logic functions implementation using Multiplexers

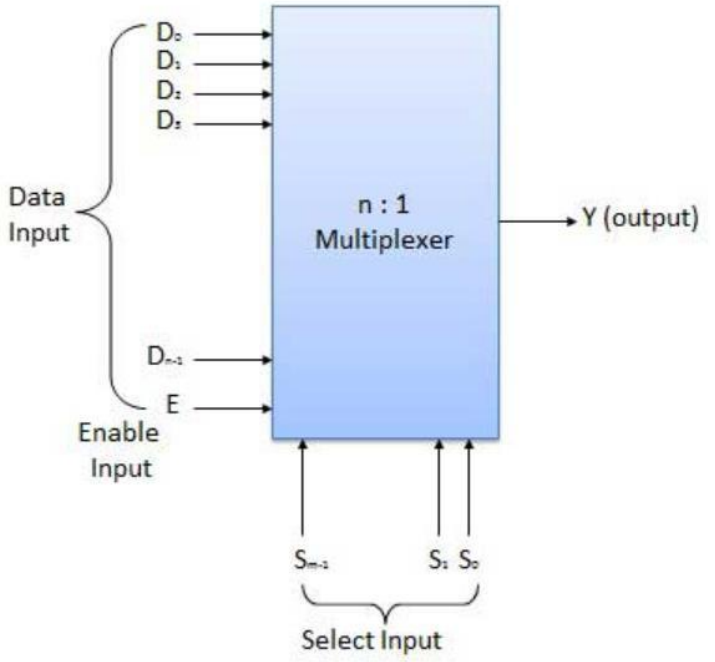
- A multiplexer (MUX) is a combinational circuit that selects one of many input signals and forwards the selected input to a single output line.
- **Basic terms:**

n:1 MUX: n data inputs, 1 output.

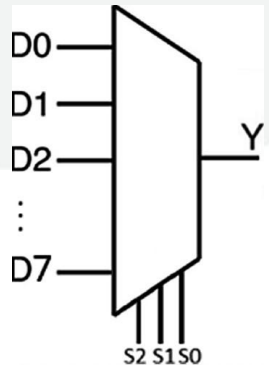
k select lines: where k = ⌈log2 n⌉. Select lines, and choose which input is connected to the output.

Enable (optional) — an active-high/low input that allows the mux to drive the output only when enabled.

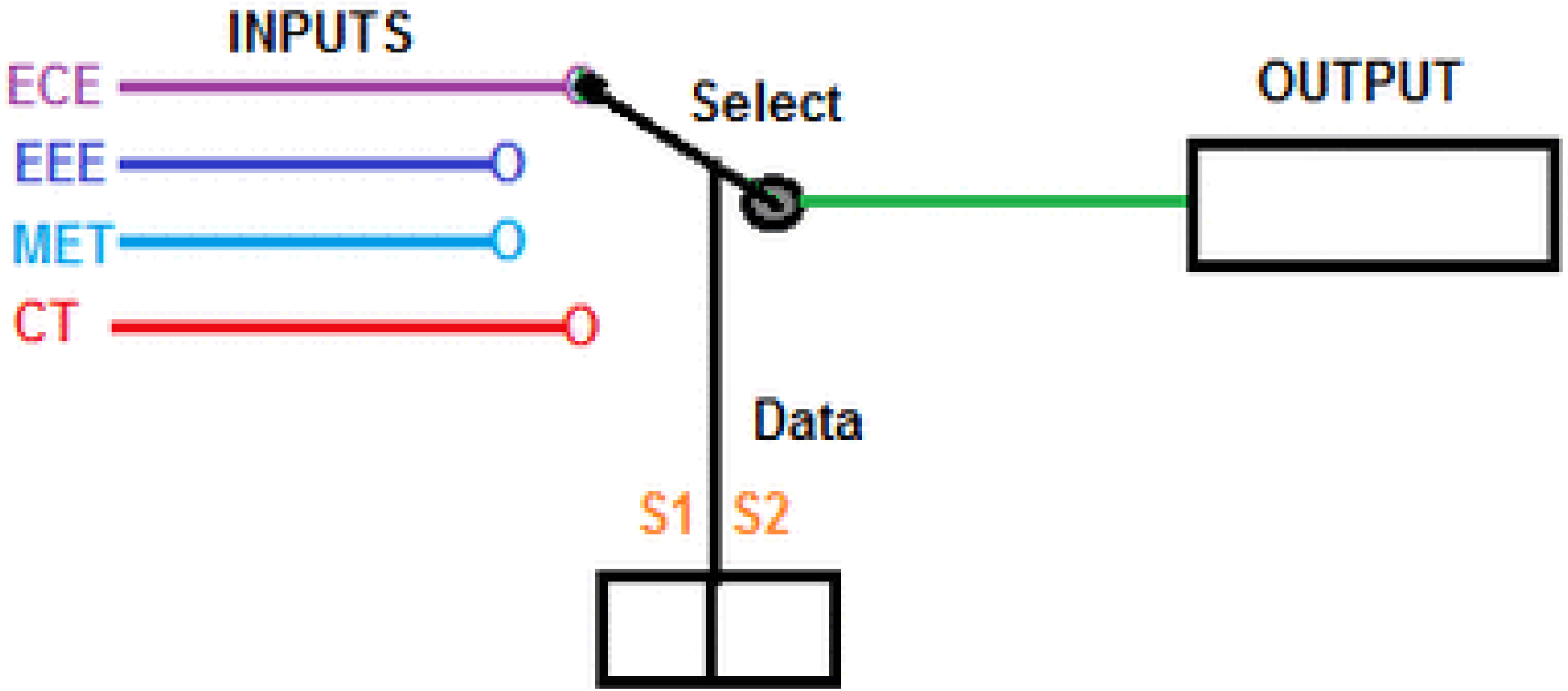
Block Diagram



Symbol (8:1)



Multiplexer operation

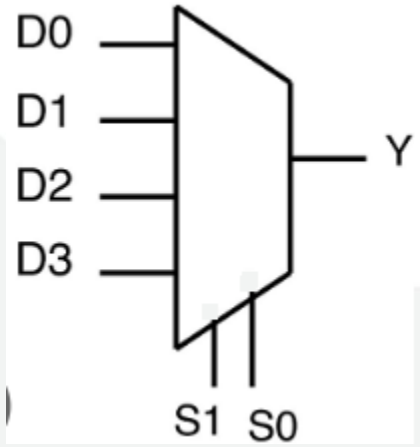


Logic functions implementation using Multiplexers

A Multiplexer can be used to implement any Boolean function by selecting the appropriate combination of inputs.

- If a Boolean expression has (n+1) variables, then 'n' of these variables can be connected to the select lines of the multiplexer. The remaining single variable, along with constants 1 and 0, is used as the input of the multiplexer.
- In general, a Boolean expression of (n+1) variables can be implemented using multiplexer with 2^n inputs.

4:1 Multiplexer



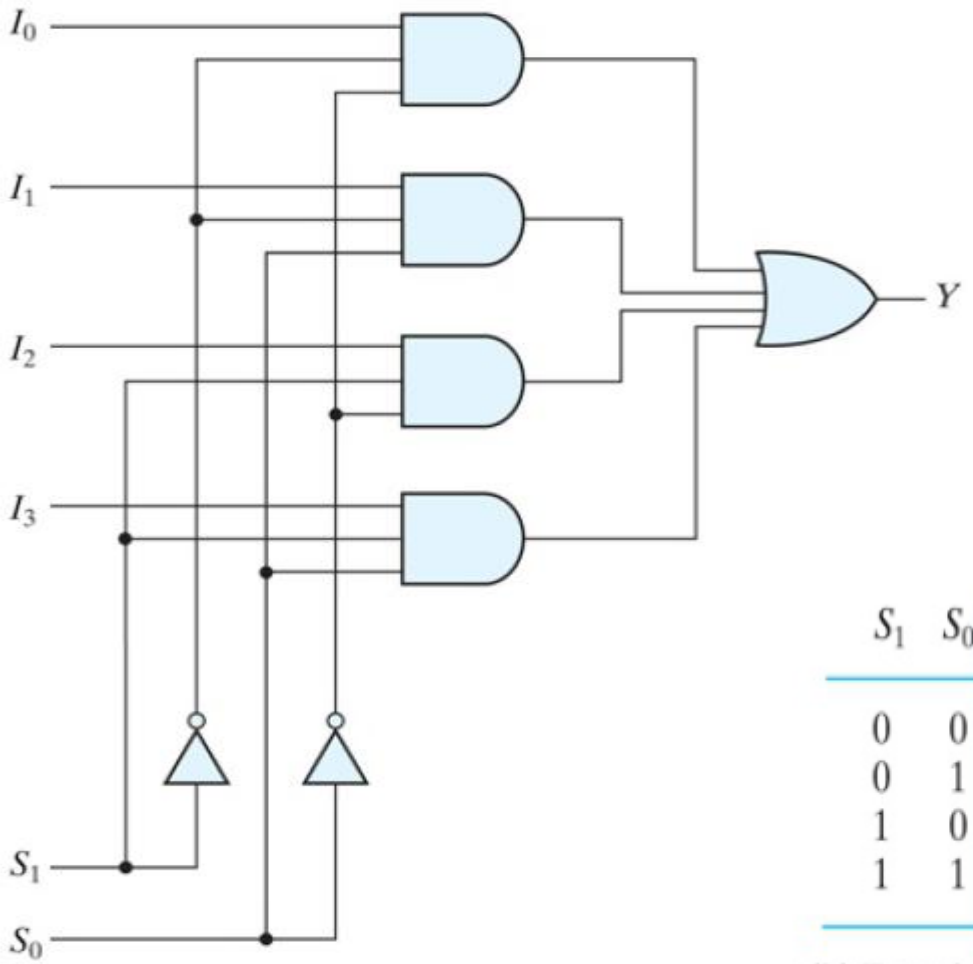
Truth table

S1	S0	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

Logic Equation

$$Y = \overline{S1}\,\overline{S0}\,D0 + \overline{S1}S0\,D1 + S1\overline{S0}\,D2 + S1S0\,D3$$

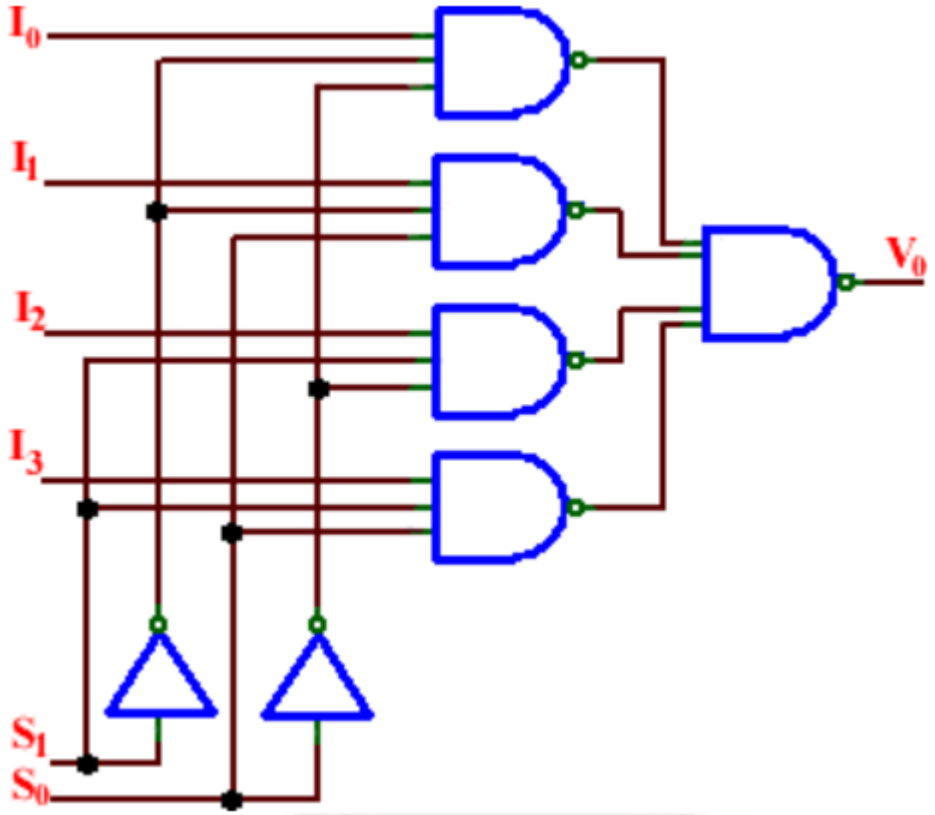
4:1 Mux Logic Circuit using basic gates, NAND gates



(a) Logic diagram

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

(b) Function table



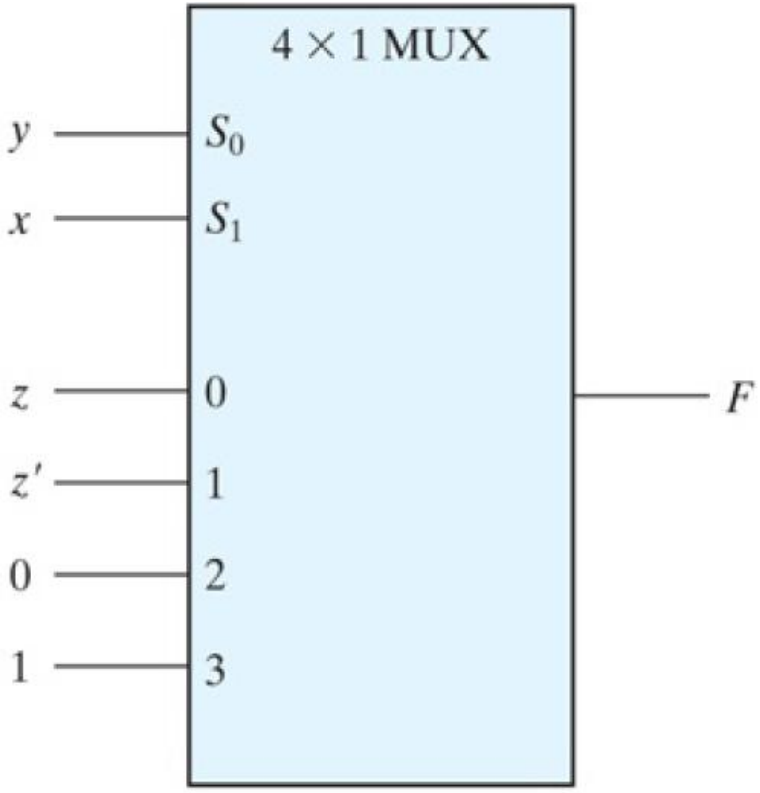
Logic functions implementation using Multiplexers

$F(x, y, z) = \Sigma(1, 2, 6, 7)$

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table

The values for the data input lines are determined from the truth table of the function. When x y = 00 , output *F* is equal to *z* because F = 0 when z = 0 and F = 1 when z = 1. This requires that variable *z* be applied to data input 0.



(b) Multiplexer implementation

Logic functions implementation using 4:1 Mux

- **Example: F (A, B, C) = ∑m (1, 3, 5, 6)**
- **Implementation table:**

Apply variables B and C to the select lines. The procedures for implementing the function are:

i. List the input of the multiplexer

ii. List under them all the minterms in two rows, with the first half of the minterms associated with A‘ and the second half with A.

	D_0	D_1	D_2	D_3
$\bar{A}$	0	1	2	3
A	4	5	6	7

C\AB	00	01	10	11
0	0	1	0	1
1	0	1	1	0

Logic functions implementation using 4:1 Mux

The given function is implemented by circling the minterms of the function and applying the following rules to find the values for the inputs of the multiplexer.

- If both the minterms in the column are not circled, apply 0 to the corresponding input.
- If both the minterms in the column are circled, apply 1 to the corresponding input.
- If the only bottom minterm is circled, apply A to the input.
- If the only top minterm is circled, apply $\bar{A}$ to the input.

	D_0	D_1	D_2	D_3
$\bar{A}$	0	(1)	2	(3)
A	4	(5)	(6)	7
	0	1	A	$\bar{A}$

From the table,

$D_0=0$, $D_1=1$, $D_2=A$, $D_3=\bar{A}$

C\AB	00	01	10	11
0	0	1	0	1
1	0	1	1	0

0	1	C	Cbar

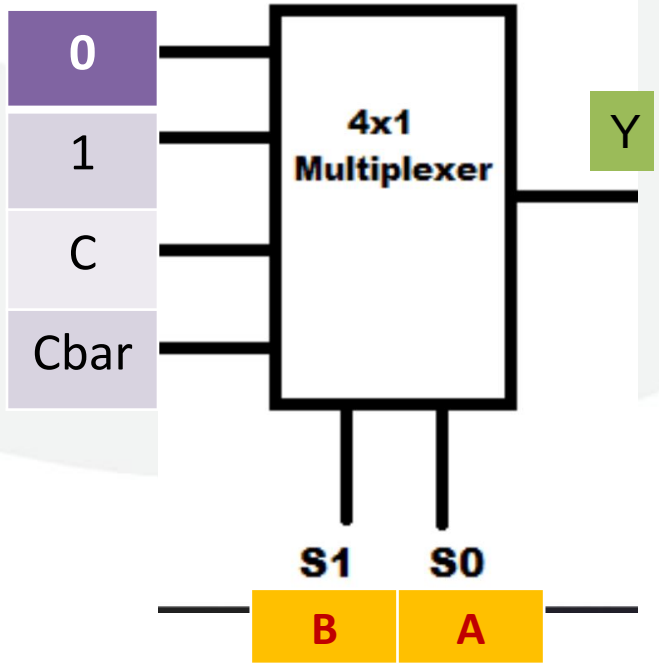
RV College of Engineering®

Design of logic circuit using MUX

Go, change the world®

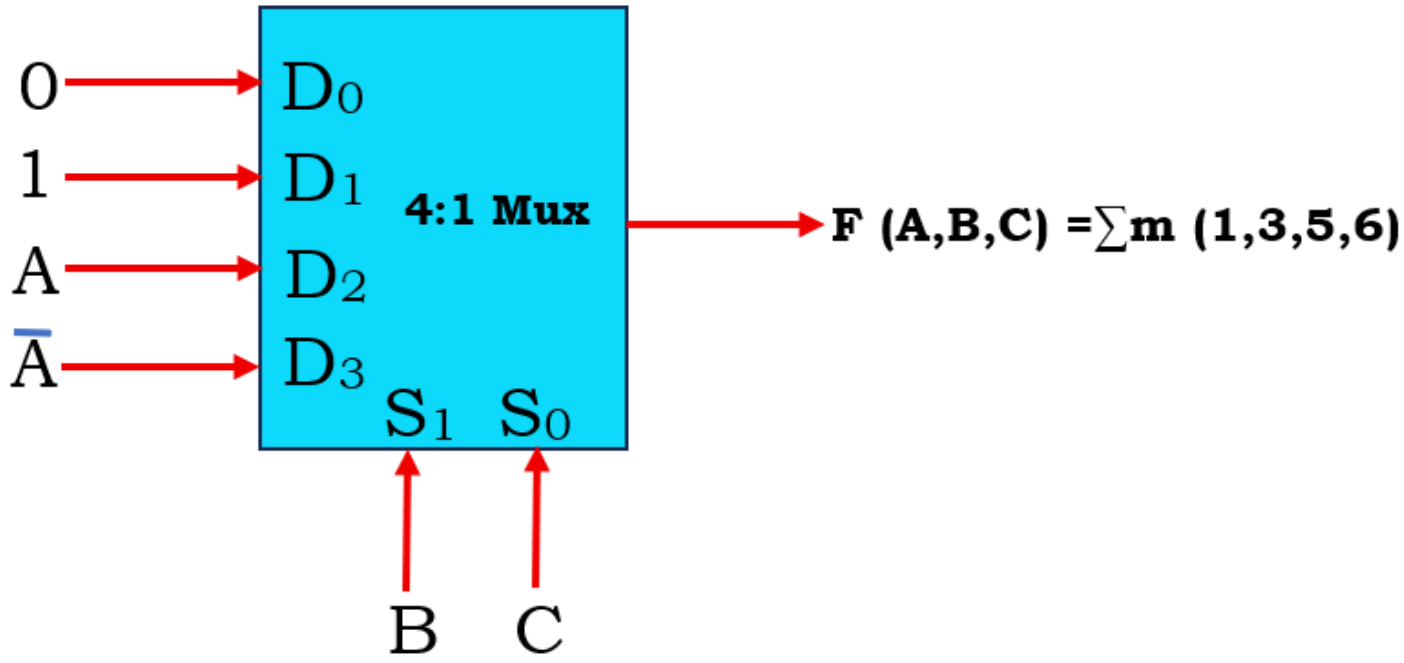
- Select any two inputs as select lines (AB as select line)
- Write K-Map for AB as row (00,01,10,11) with C as coloumn
- Enter the 1's in the expression into K-Map
- If both row entry is 0 then connect 0 to respective data line (e.g: 0 to D0)
- If both row entry is 1, then connect 1 to respective data line
- Compare the outputs with variable C value in K map if both rows are not same.
- If output is same as C, connect C else connect Cbar

C\AB	00	01	10	11
0	0	1	0	1
1	0	1	1	0
	0	1	C	Cbar



Implementation of F (A, B, C) = ∑m (1, 3, 5, 6) using 4:1 Mux

Logic Diagram



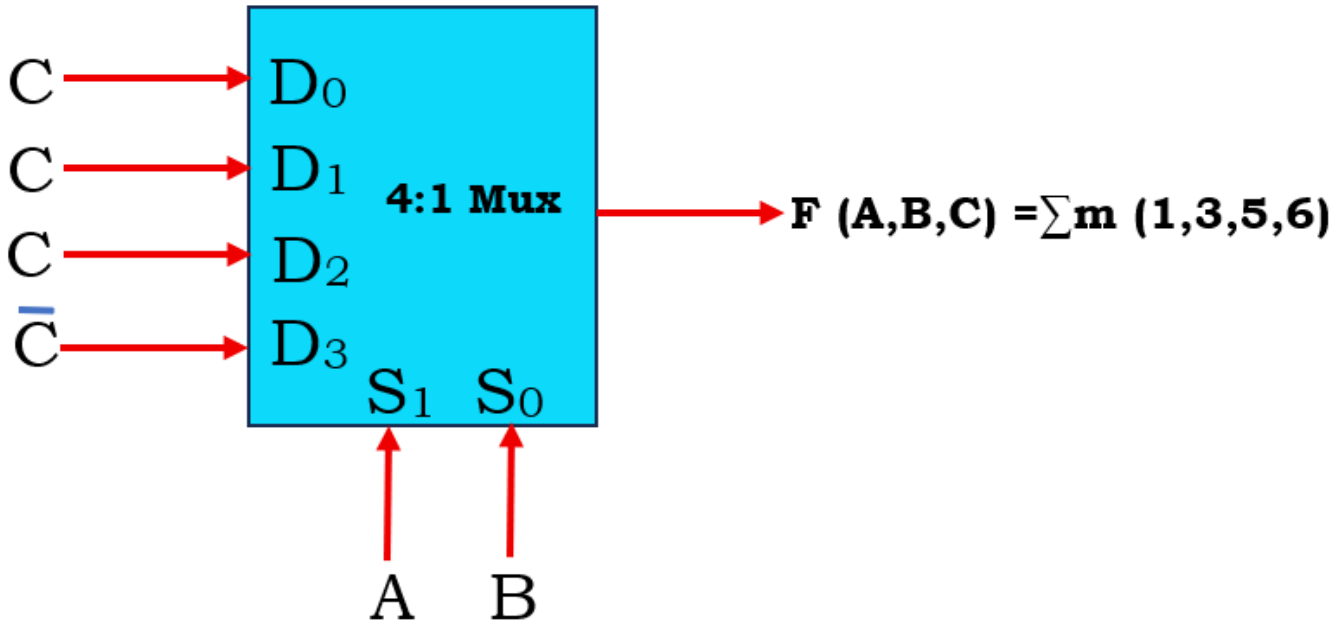
Implementation of F (A, B, C) = ∑m (1, 3, 5, 6) using 4:1 Mux

- **Apply variables A and B to the select lines.**

Implementation Table

	D_0	D_1	D_2	D_3
$\bar{C}$	0	2	4	(6)
C	(1)	(3)	(5)	7
	C	C	C	$\bar{C}$

Logic Diagram



Question 5: Implement the following Boolean function using MUX

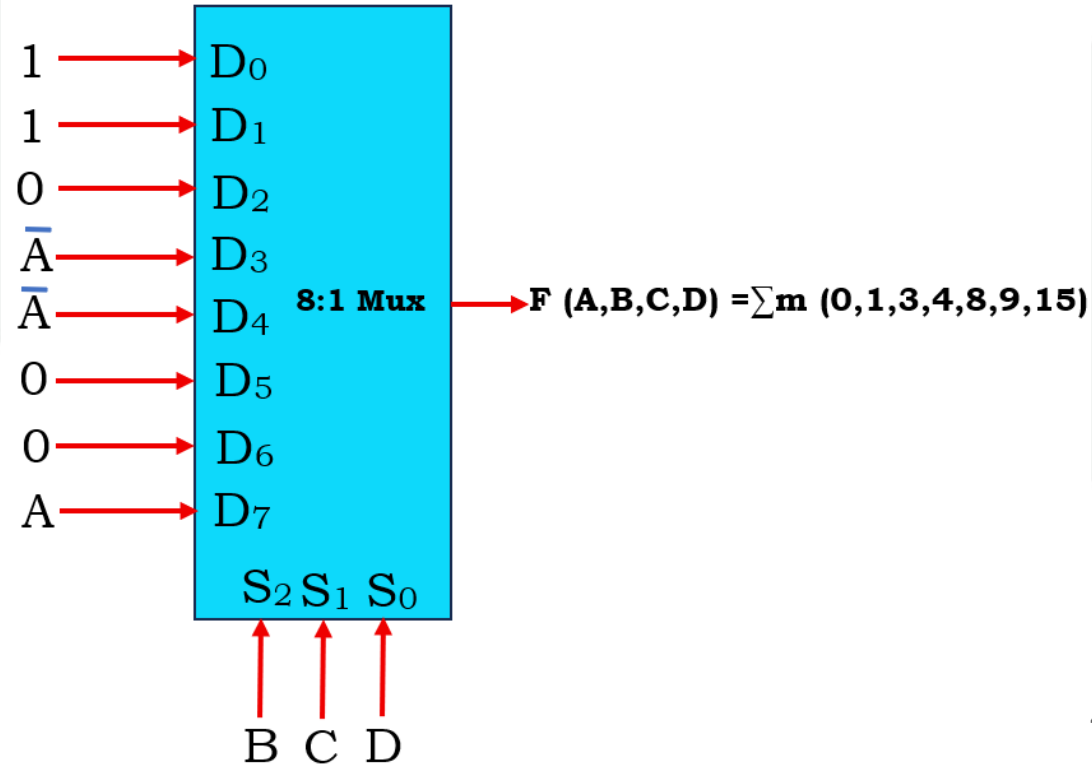
$$F\ (A,B,C,D) = \sum m\ (0,1,\ 3,\ 4,\ 8,9,15)$$

- Assuming A is connected to inputs and B,C,D are connected to select lines.

Implementation Table

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	(0)	(1)	2	(3)	(4)	5	6	7
A	(8)	(9)	10	11	12	13	14	(15)
	1	1	0	$\bar{A}$	$\bar{A}$	0	0	A

Logic Diagram



RV College of Engineering®

Question 6: Implement the following Boolean function using MUX

Go, change the world®

$F(A, B, C, D) = \Sigma(1, 3, 4, 11, 12, 13, 14, 15)$

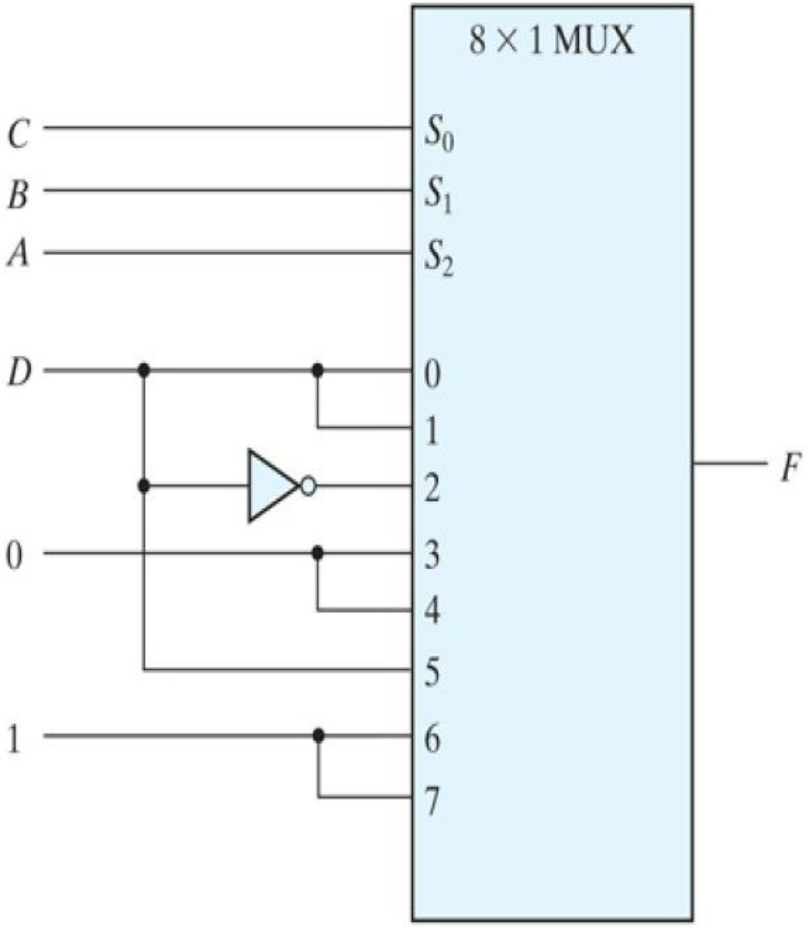
A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

Truth table method

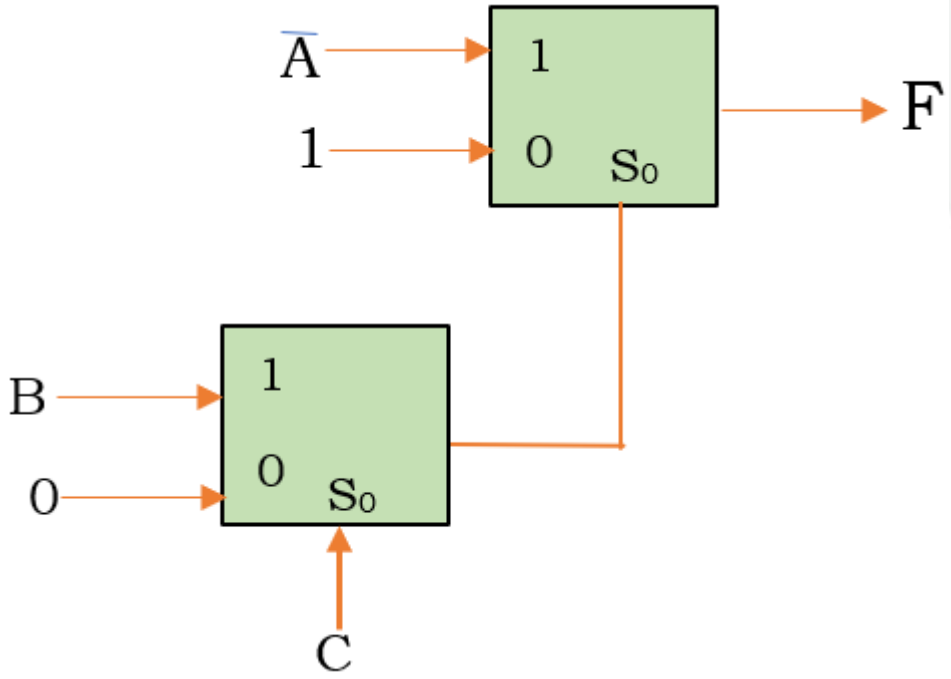
D

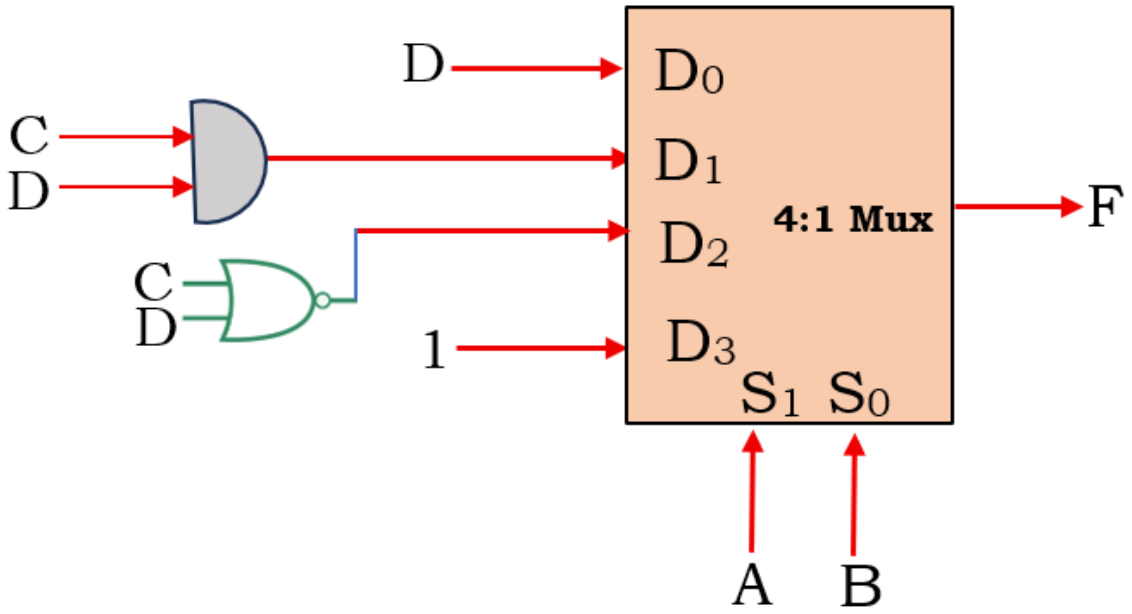
ABC	I0	I1	I2	I3	I4	I5	I5	I7
	000	001	010	011	100	101	110	111
0	0	0	1	0	0	0	1	1
1	1	1	0	0	0	1	1	1
F	D	D	D′	0	0	D	1	1

K-map method



Question 6: Determine the F in the following

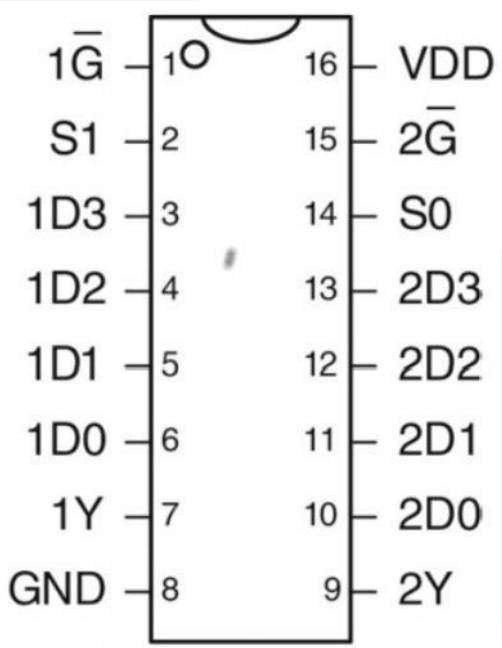




74153 dual 4:1 Multiplexer IC

- It contains two independent 4-input multiplexers on a single chip, each capable of selecting one of four data inputs to be passed to the output.
- Common select lines (S0, S1) for both multiplexers.
- Separate enable inputs (active LOW) for each MUX.
- TTL-compatible inputs/outputs.

Pin Diagram



Function Table

$D_{3:0}$:	data
$S_{1:0}$:	select
Y:	output
Gb:	enable

Question 7: Realize a full adder using 74153 MUX IC

- Truth table of Full adder

A	B	Cin	Sum (S)	Carry (Cout)
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= F1(A,B,C) = ∑m (1,2,4,7)

Cout= F2(A,B,C) =∑m (3,5,6,7)

- Assuming B & C are connected to select lines, Mux 1 and Mux 2 are used to realize sum and carry, respectively.

Implementation Tables

Sum

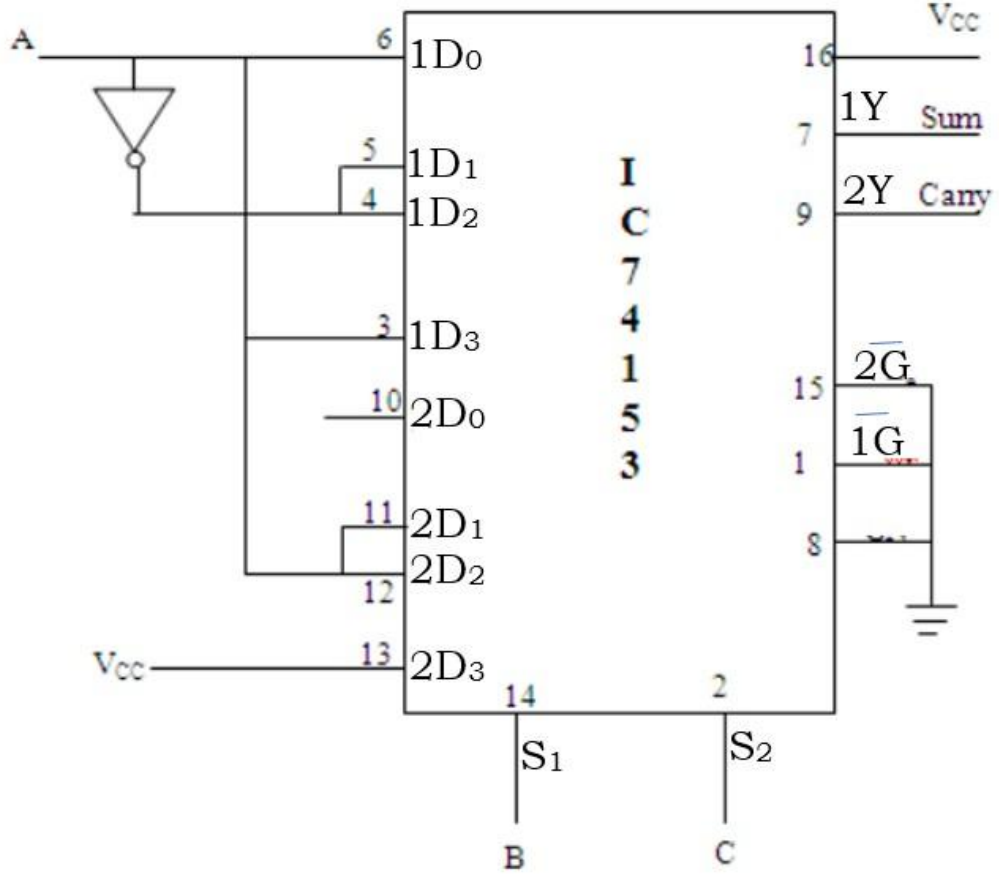
	D_0	D_1	D_2	D_3
$\bar{A}$	0	(1)	(2)	3
A	(4)	5	6	(7)
	A	$\bar{A}$	$\bar{A}$	A

Cout

	D_0	D_1	D_2	D_3
$\bar{A}$	0	1	2	(3)
A	4	(5)	(6)	(7)
	0	A	A	1

Question 7: Realize a full adder using 74153 MUX IC

Logic Diagram



Question 8

Realize a 8:1 multiplexer using the 74153 IC.

Realize a 8:1 multiplexer using the 74153 IC

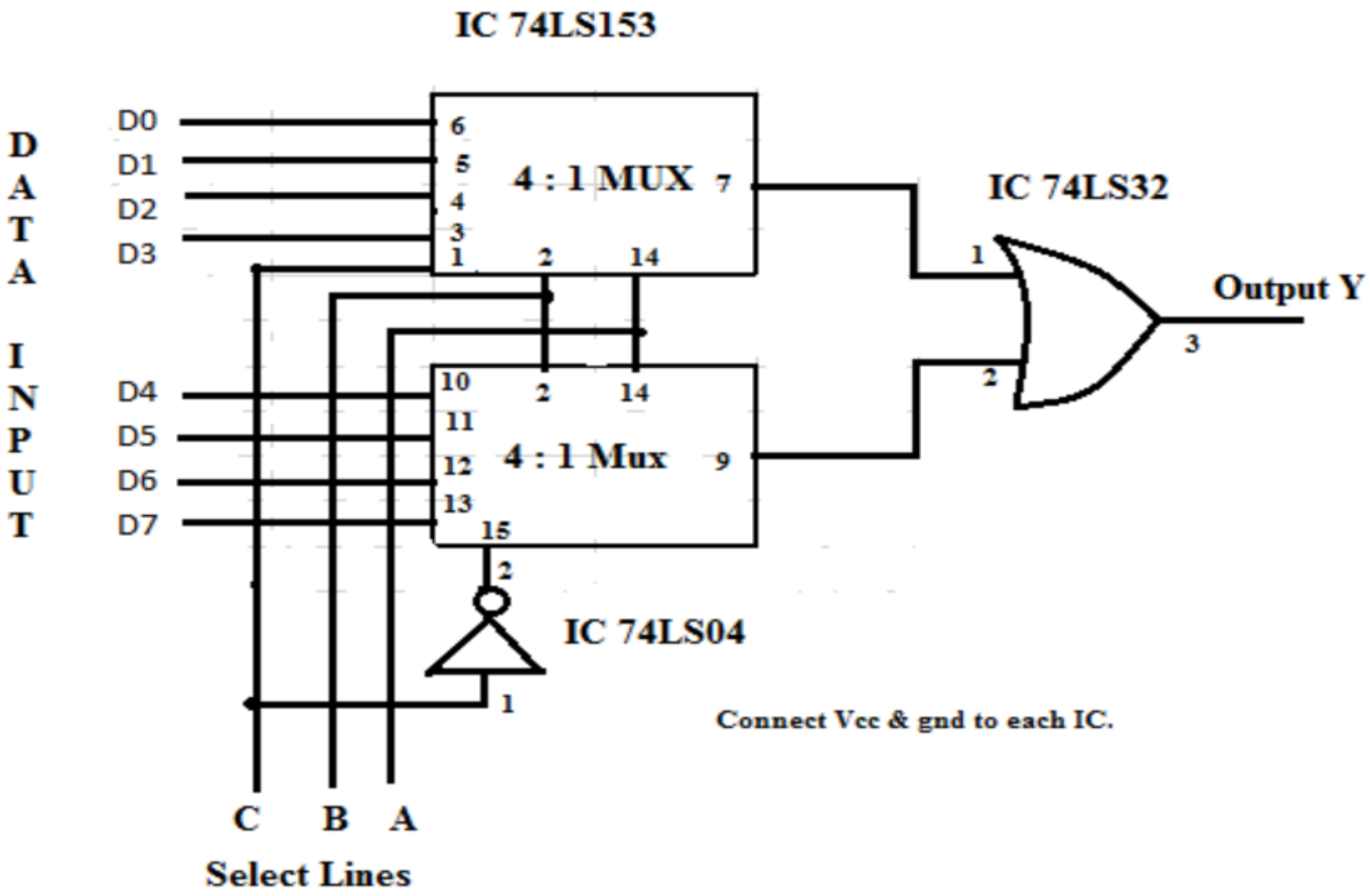
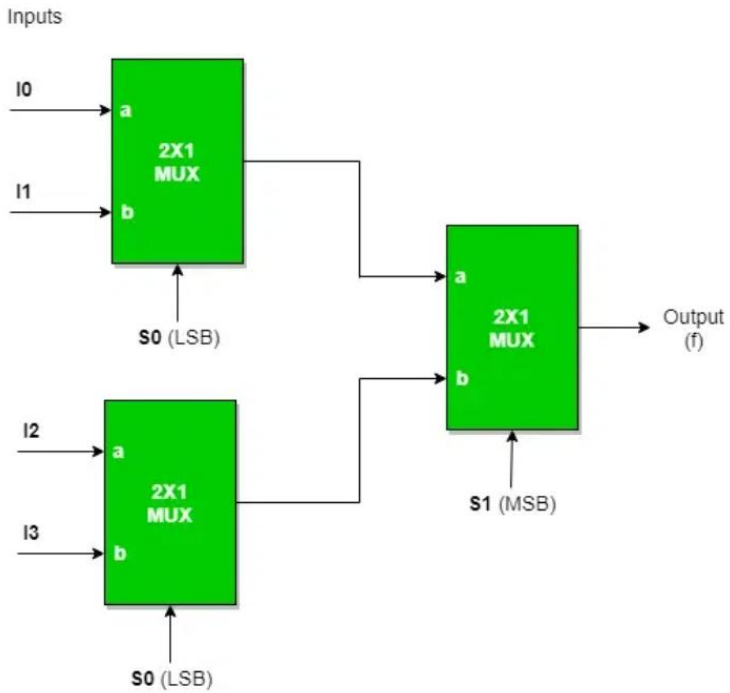


Fig. 3. Eight to One Multiplexer using IC 74LS153

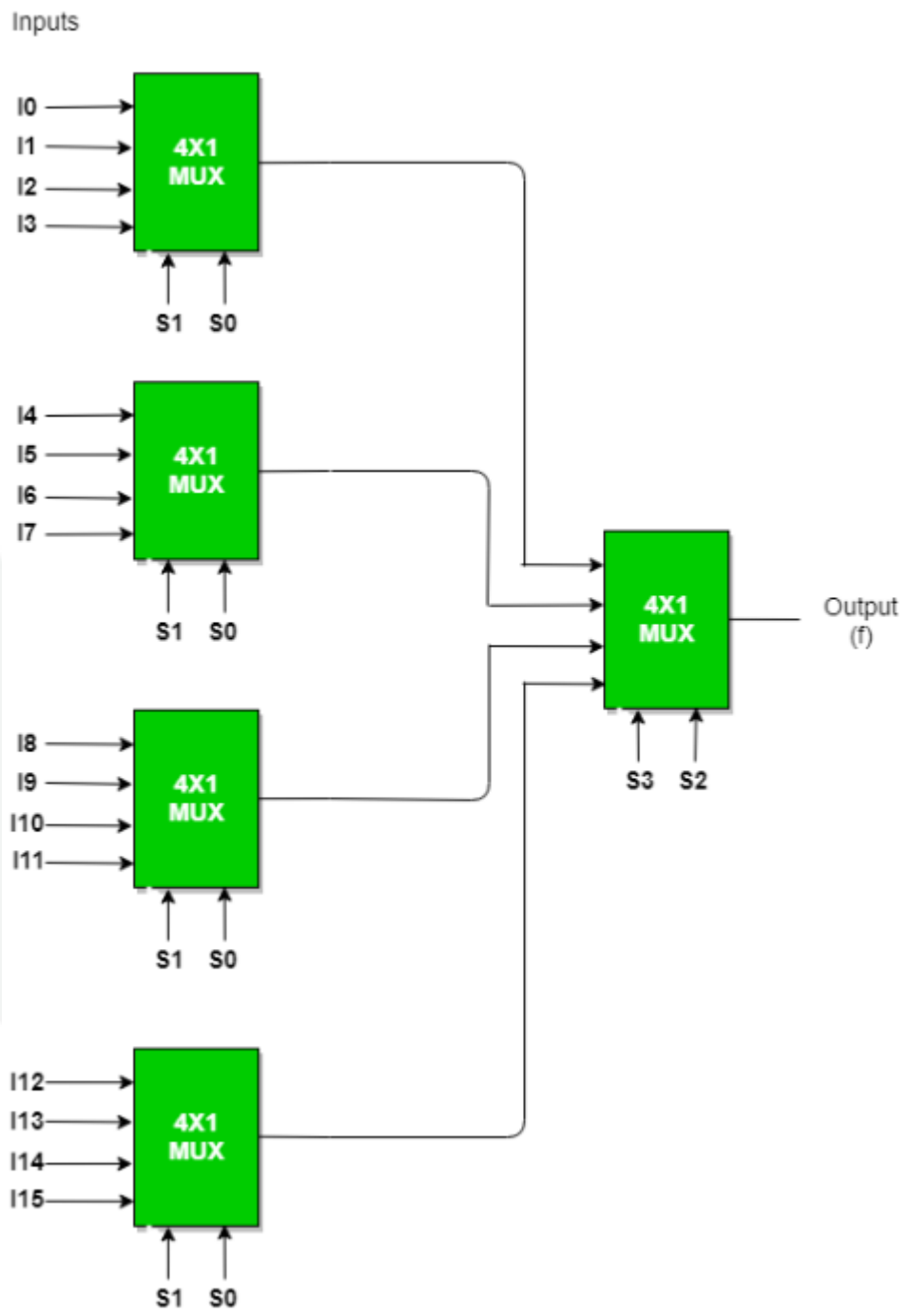
RV College of Engineering®

16:1 MUX using 4:1 MUX



Truth Table

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

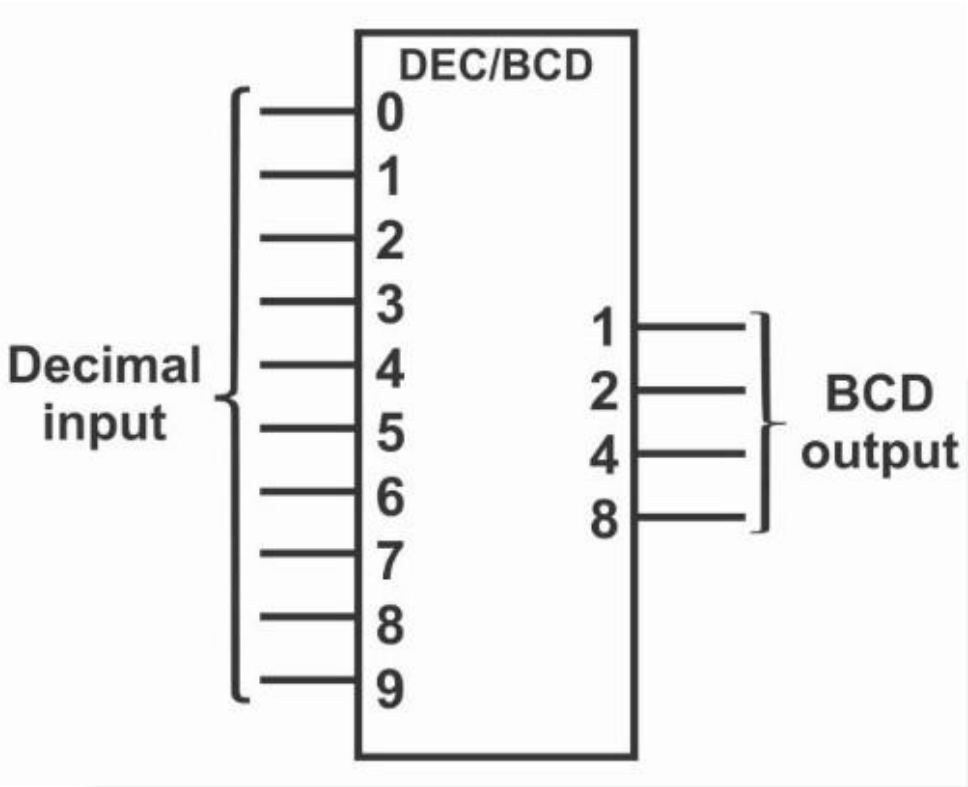


Encoders

- An Encoder is a combinational logic circuit that converts 2^n(or fewer) input lines into an n-bit binary code.

Example: Decimal to BCD encoder

- A Decimal-to-Binary Encoder converts a decimal input (0–9) into its equivalent 4-bit binary output (0000–1001).

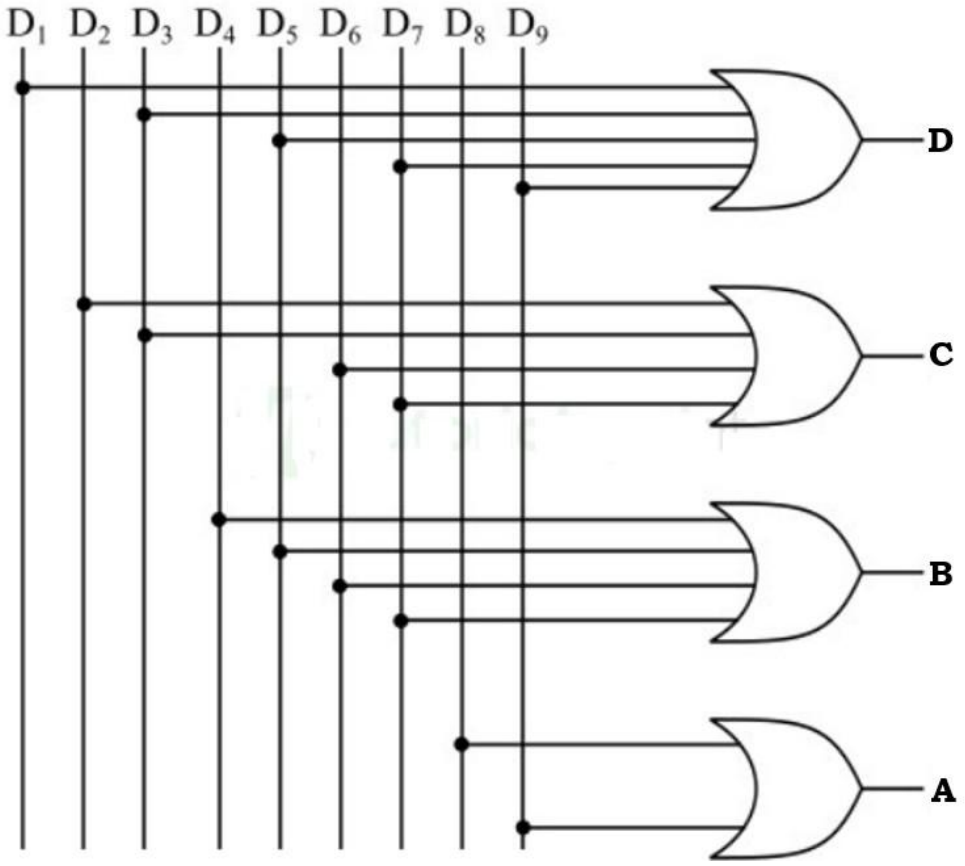


Decimal to BCD Encoder

Truth Table

Decimal Inputs										BCD Output			
D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A	B	C	D
1	0	0	0	0	0	0	0	0	0	1	0	0	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	0	0	1	0	0	0	0

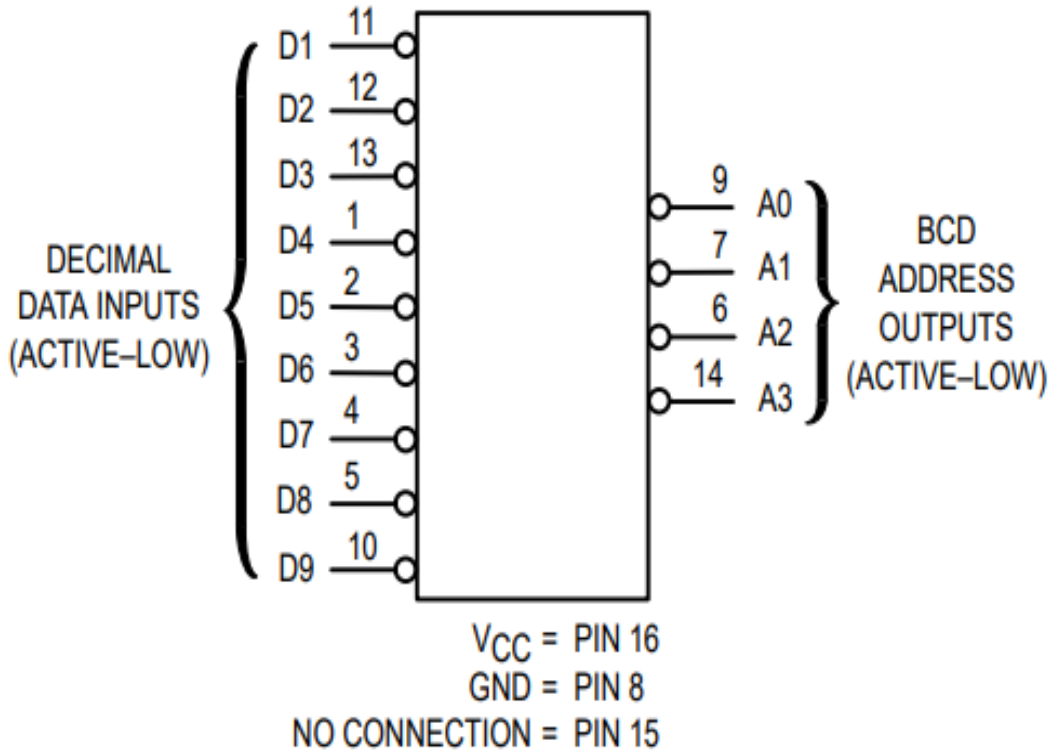
Circuit Diagram



Priority Encoders

- A priority encoder converts multiple input lines into a binary code on the output lines.
- Unlike a simple encoder, it assigns priority to the inputs — if multiple inputs are active at the same time, the highest-priority input determines the output.
- Example: 74HC147 Decimal to BCD active low priority encoder

Pin Diagram:74HC147

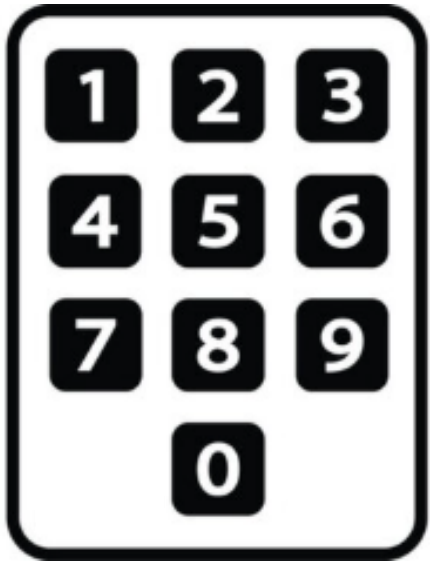


74HC147:Function Table

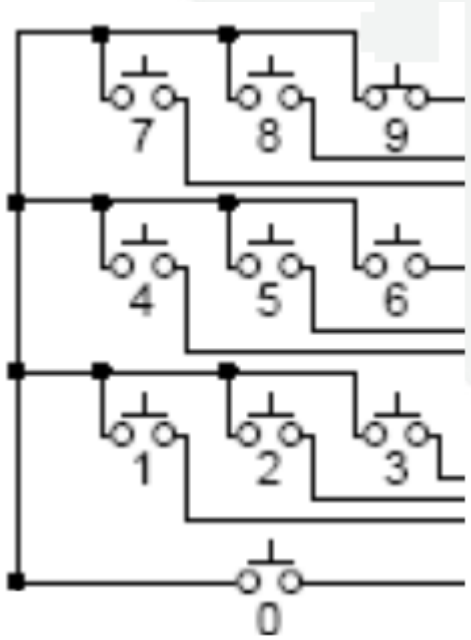
Inputs									Output			
D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	A	B	C	D
H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	H	H	H	H	H	H	L	H	H	H	L
H	H	H	H	H	H	H	L	x	H	H	L	H
H	H	H	H	H	H	L	x	x	H	H	L	L
H	H	H	H	H	L	x	x	x	H	L	H	H
H	H	H	H	L	x	x	x	x	H	L	H	L
H	H	H	L	x	x	x	x	x	H	L	L	H
H	H	L	x	x	x	x	x	x	H	L	L	L
H	L	x	x	x	x	x	x	x	L	H	H	H
L	x	x	x	x	x	x	x	x	L	H	H	L

Keypad encoder using 74147

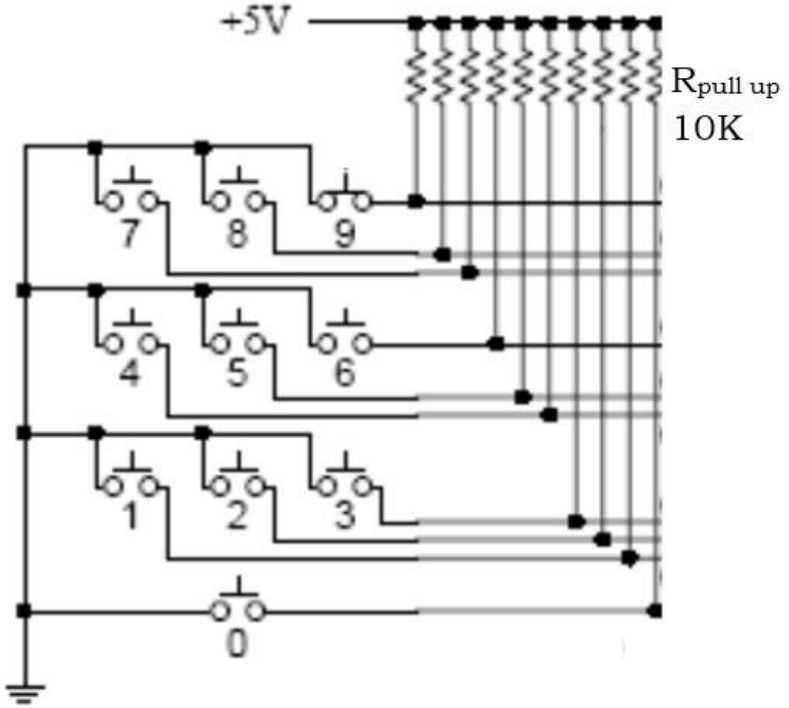
Decimal Keypad



Circuit diagram

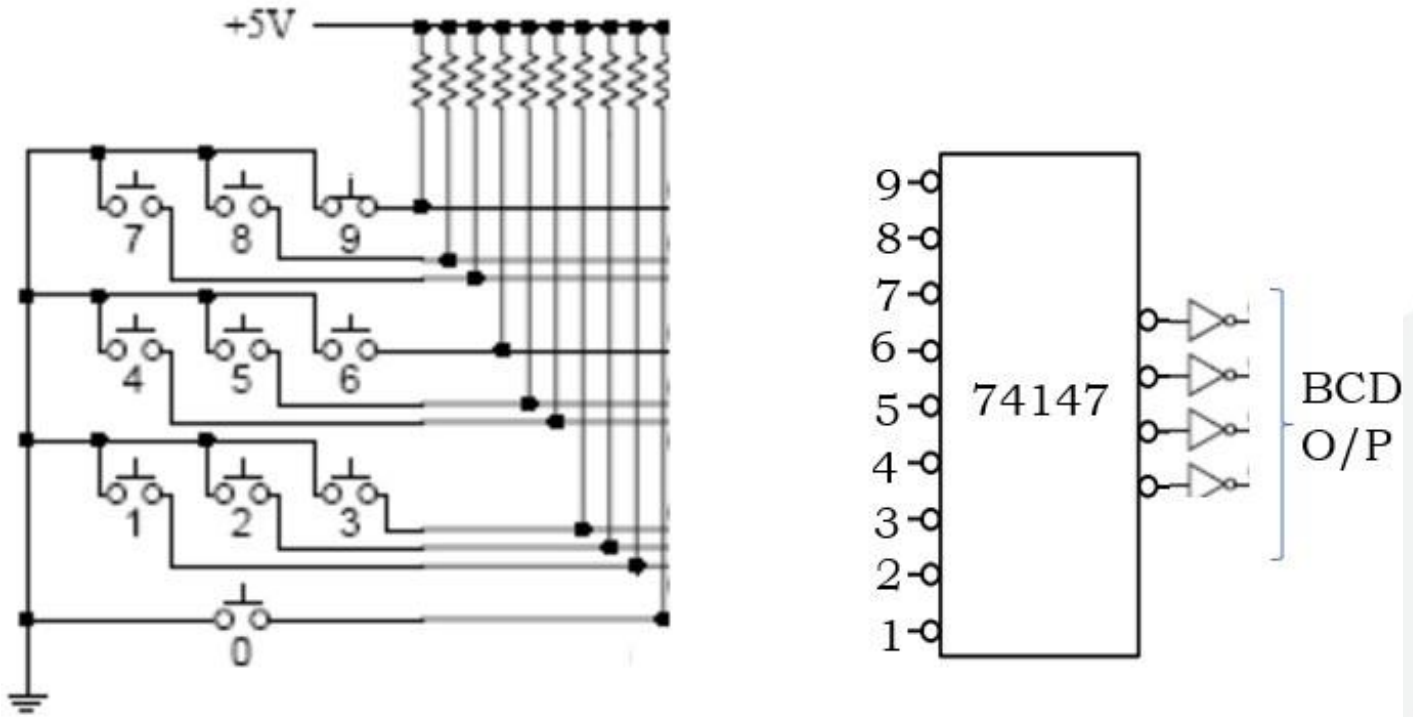


With Pull up



Keypad encoder using 74147

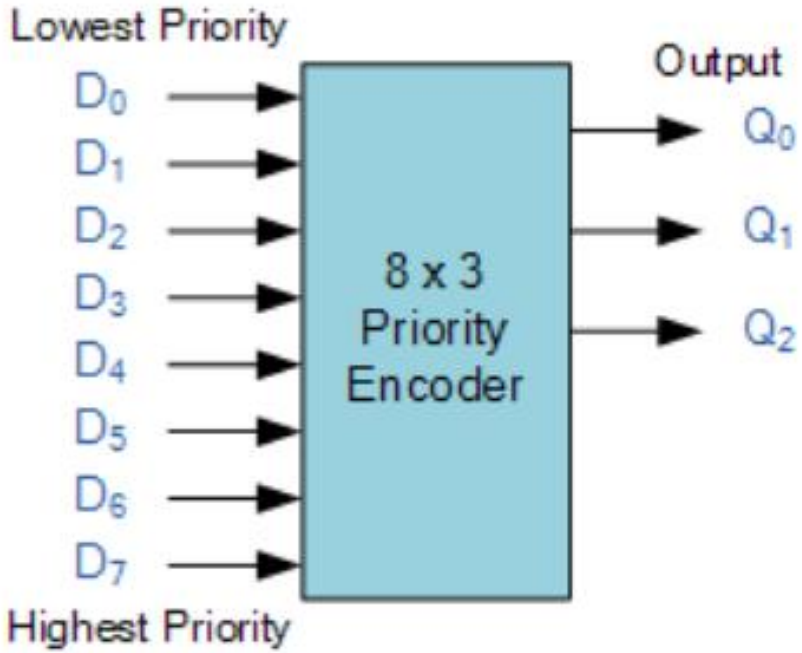
Circuit Diagram



- The O/P of the encoder is always low when no key is pressed.
- The zero key is not connected because BCD O/P is zero when no key is pressed.

Priority Encoder with active high I/O lines

8-to-3 Bit Priority Encoder



Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

X = dont care

Logic Expression for Priority Encoder

$$Q_0 = \sum(1, 3, 5, 7)$$

$$Q_0 = \sum(\bar{D}_7\bar{D}_6\bar{D}_5\bar{D}_4\bar{D}_3\bar{D}_2D_1 + \bar{D}_7\bar{D}_6\bar{D}_5\bar{D}_4D_3 + \bar{D}_7\bar{D}_6D_5 + D_7)$$

$$Q_0 = \sum(\bar{D}_6\bar{D}_4\bar{D}_2D_1 + \bar{D}_6\bar{D}_4D_3 + \bar{D}_6D_5 + D_7)$$

$$Q_0 = \sum(\bar{D}_6(\bar{D}_4\bar{D}_2D_1 + \bar{D}_4D_3 + D_5) + D_7)$$

Output Q_1

$$Q_1 = \sum(2, 3, 6, 7)$$

$$Q_1 = \sum(\bar{D}_7\bar{D}_6\bar{D}_5\bar{D}_4\bar{D}_3D_2 + \bar{D}_7\bar{D}_6\bar{D}_5\bar{D}_4D_3 + \bar{D}_7D_6 + D_7)$$

$$Q_1 = \sum(\bar{D}_5\bar{D}_4D_2 + \bar{D}_5\bar{D}_4D_3 + D_6 + D_7)$$

$$Q_1 = \sum(\bar{D}_5\bar{D}_4(D_2 + D_3) + D_6 + D_7)$$

Logic Expression for Priority Encoder-contd

Output Q_2

$$Q_2 = \sum(4, 5, 6, 7)$$

$$Q_2 = \sum(\bar{D}_7\bar{D}_6\bar{D}_5D_4 + \bar{D}_7\bar{D}_6D_5 + \bar{D}_7D_6 + D_7)$$

$$Q_2 = \sum(D_4 + D_5 + D_6 + D_7)$$

Priority Encoder Output Expression

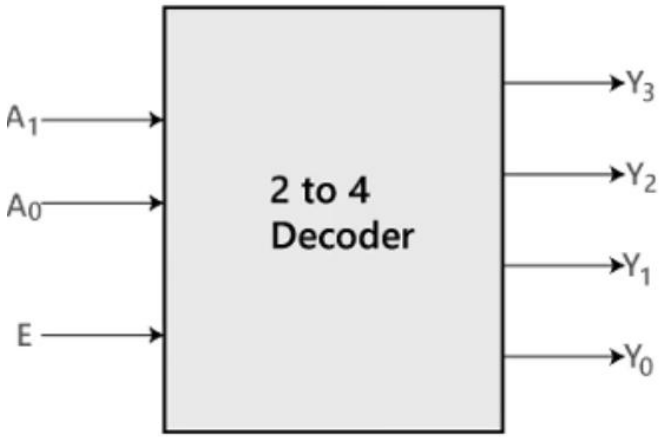
$$Q_0 = \sum(\bar{D}_6(\bar{D}_4\bar{D}_2D_1 + \bar{D}_4D_3 + D_5) + D_7)$$

$$Q_1 = \sum(\bar{D}_5\bar{D}_4(D_2 + D_3) + D_6 + D_7)$$

$$Q_2 = \sum(D_4 + D_5 + D_6 + D_7)$$

Decoders

- A decoder converts binary informationfrom n input lines into a maximum of 2^n unique output lines.
- **Example: 2 to 4 Decoder**

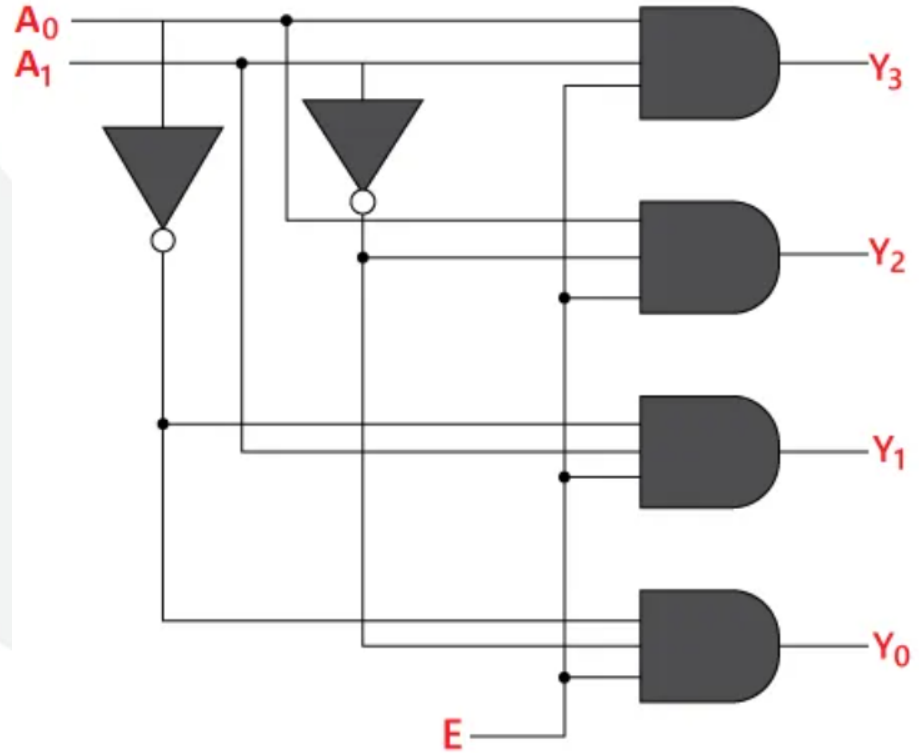


Block Diagram

Block Diagram

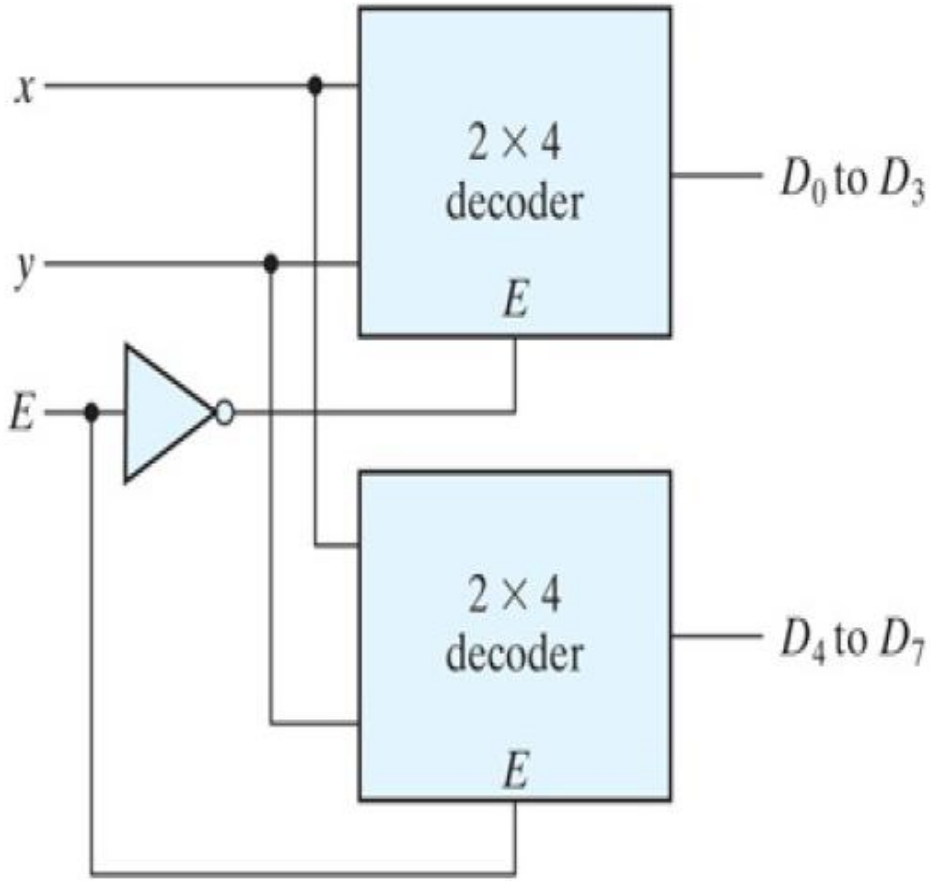
Enable	INPUTS		OUTPUTS			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Truth Table



Circuit Diagram

3:8 decoder using 2:4 decoder



E	x	y	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Logic functions implementation decoder

- A Booleanfunction with n variablescan be implementedusing a decoder with n inputs.

Example: Implementation of a full subtractor using 3 to 8 decoder

Truth table

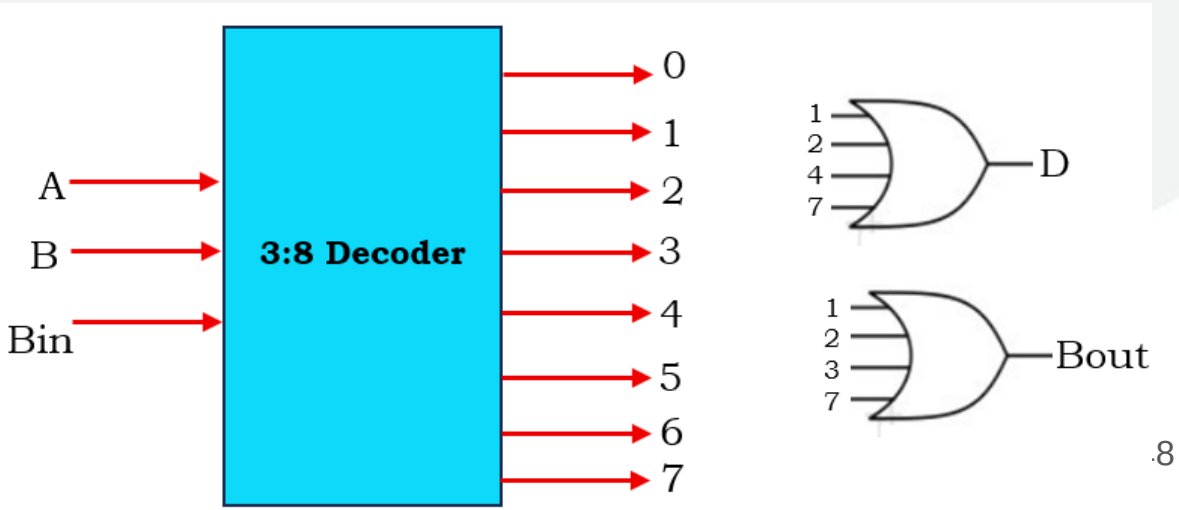
A	B	Bin	Diff (D)	Borrow (Bout)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Minterms

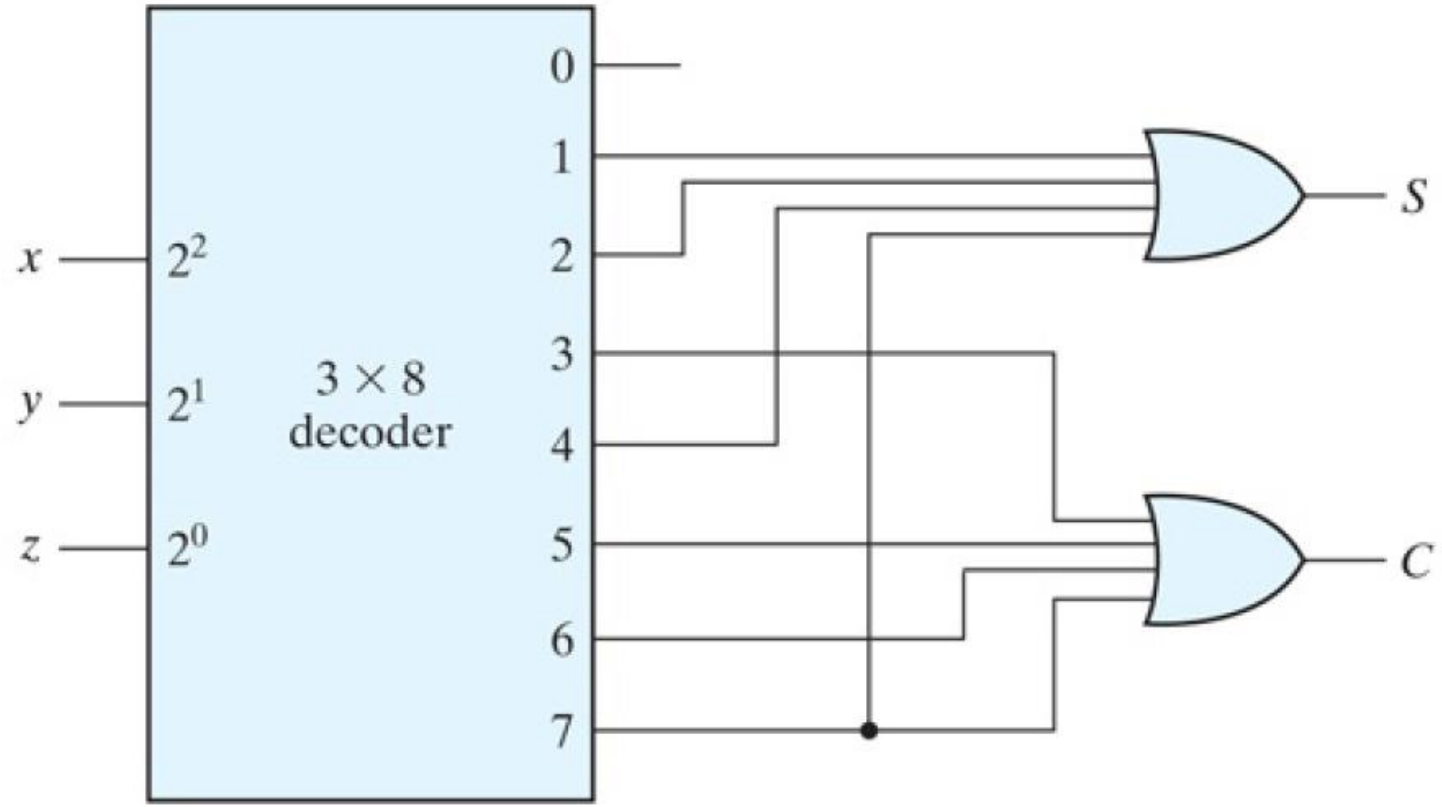
$D(A,B,Bin) = \sum m\ (1,2,4,7)$

$Bout(A,B,Bin) = \sum m\ (1,2,3,7)$

Logic Diagram



$S(x, y, z) = \Sigma(1, 2, 4, 7)$ and $C(x, y, z) = \Sigma(3, 5, 6, 7)$

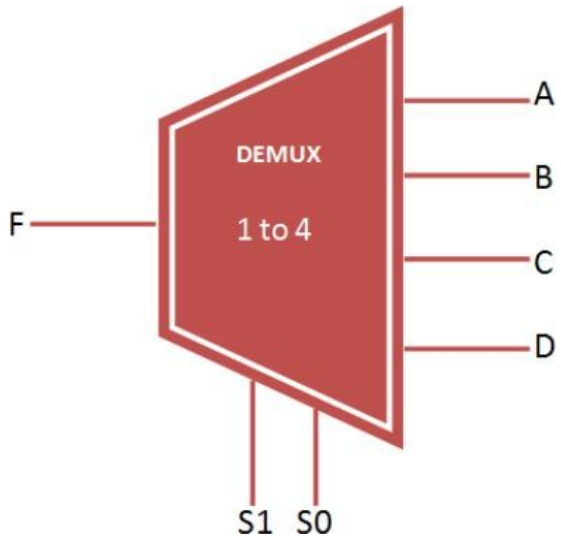


Question 9

Realize a 5 to 32 decoder using 3 to 8 decoders. Show the function table.

Logic function implementation using Demultiplexers

- A demultiplexer is a digital circuit that takes one input and routes it to one of many outputs, depending on select (control) inputs.
- **Example: 1 to 4 Demux**

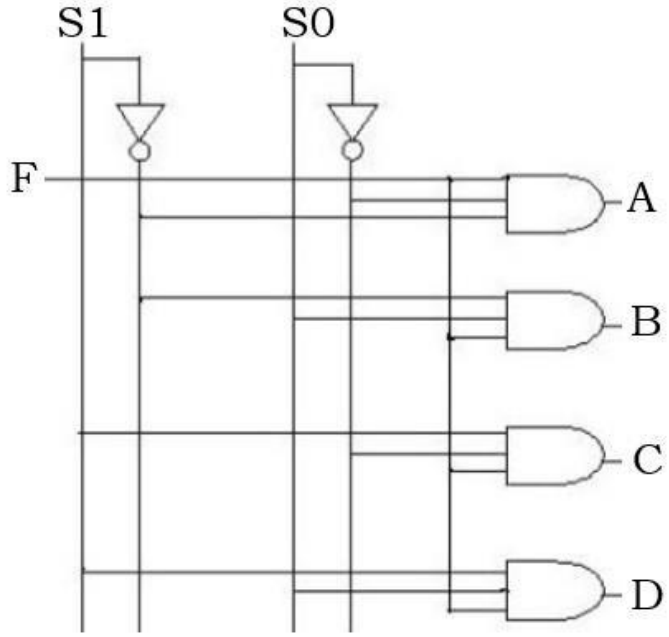


Truth table

Select Inputs		Outputs			
S1	S0	D	C	B	A
0	0	x	x	x	F
0	1	x	x	F	x
1	0	x	F	x	x
1	1	F	x	x	x

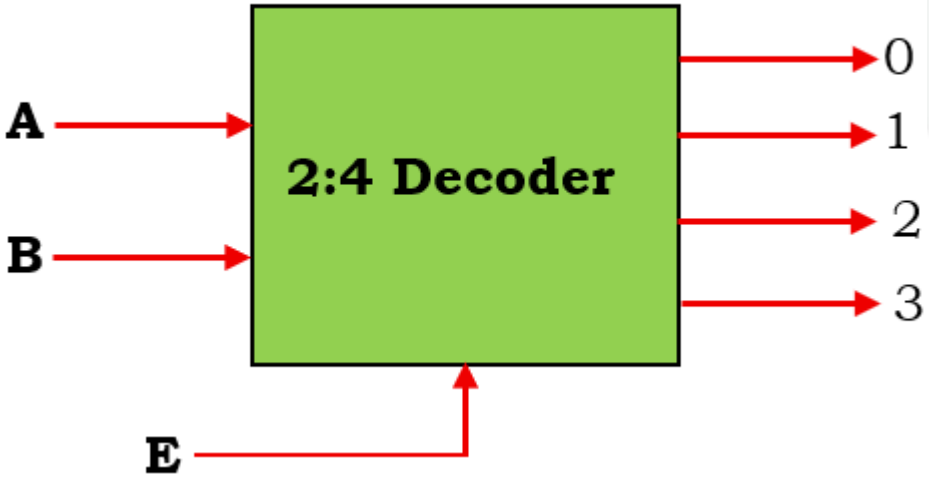
x-Don't care

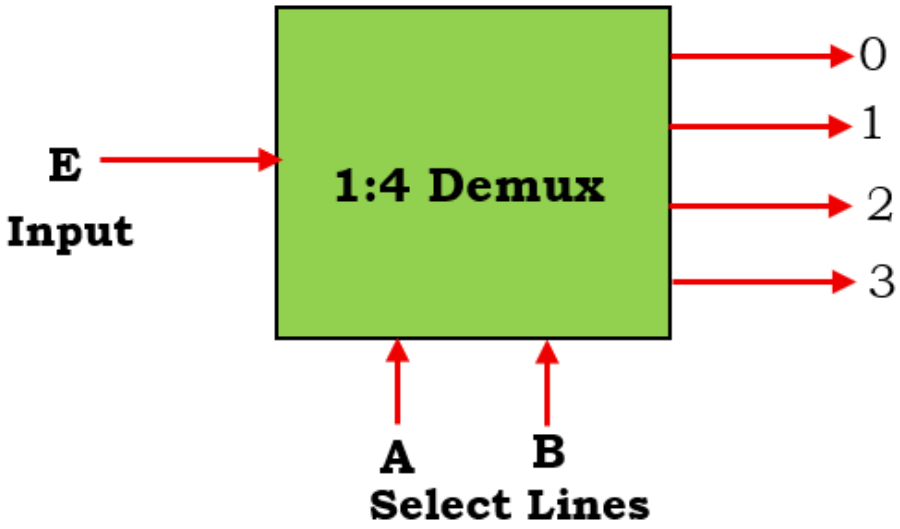
Circuit Diagram



Decoder & Demultiplexer

- A decoder with an enable line can function as a demultiplexer.





3-bit binary(B2,B1,B0) to Gray (G2,G1,G0) code converter using demultiplexer

Realize 3 bit binary(B2,B1,B0) to Gray (G2,G1,G0) code converter using demultiplexer.

Truth table

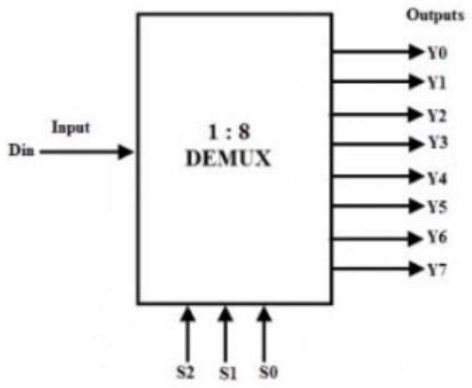
Binary Inputs			Gray Inputs		
B2	B1	B0	G2	G1	G0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

Minterms

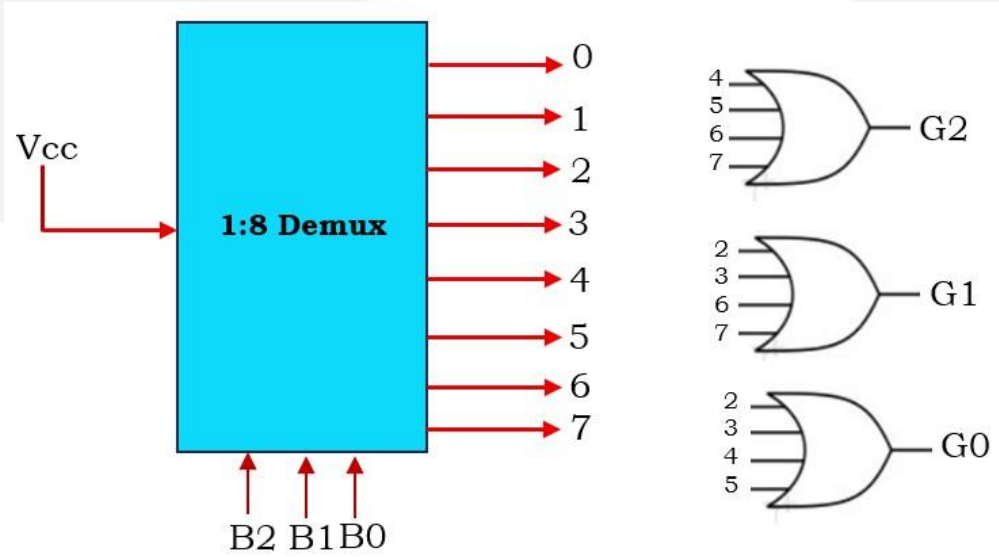
$G2 = \sum m\ (4,5,6,7)$

$G1 = \sum m\ (2,3,4,5)$

$G0 = \sum m\ (1,2,5,6)$

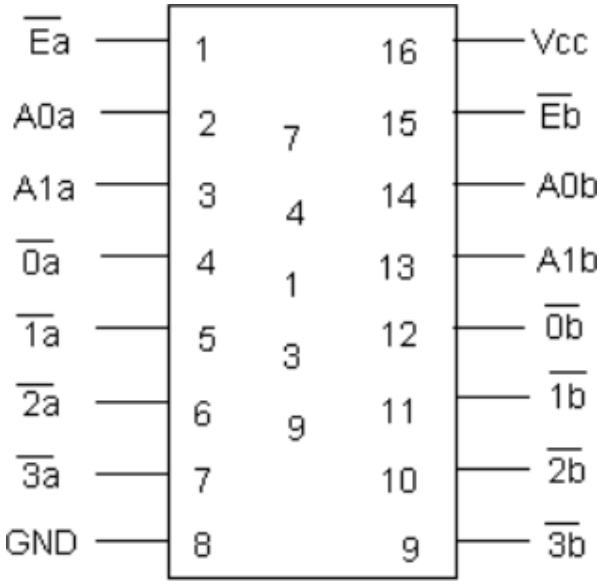


Circuit diagram



74139 Dual 2-to-4 Line Decoder/Demultiplexer

Pin diagram



Function Table

Decoder A

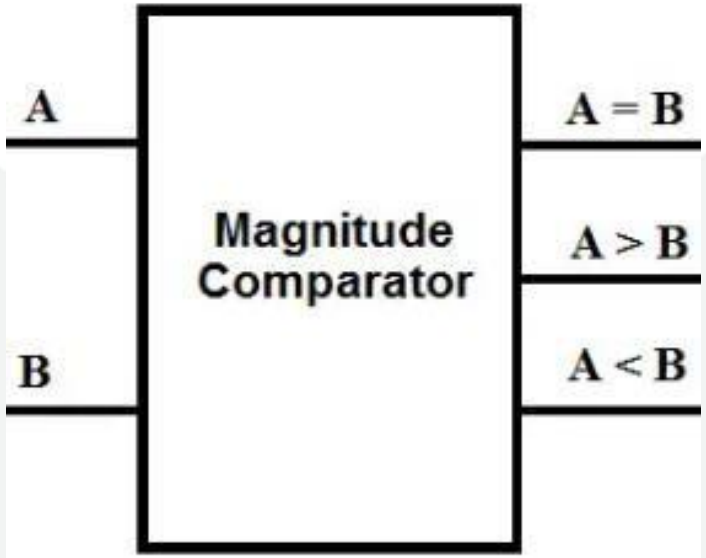
$\overline{Ea}$	A1a	A0a	$\overline{0a}$	$\overline{1a}$	$\overline{2a}$	$\overline{3a}$
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Decoder B

$\overline{Eb}$	A1b	A0b	$\overline{0b}$	$\overline{1b}$	$\overline{2b}$	$\overline{3b}$
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

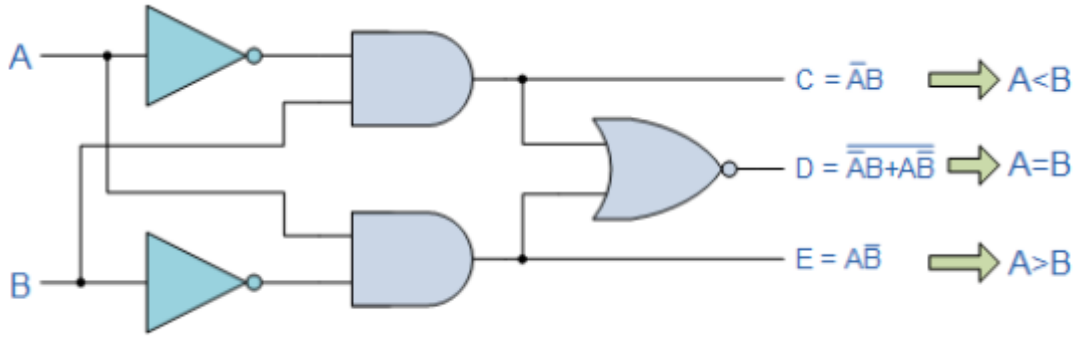
Magnitude Comparator

- A magnitude comparator is a combinational logic circuit that compares two binary numbers and determines their relative magnitude.
- Given two binary numbers A and B (each of n bits), the comparator outputs indicate whether: **A > B, A = B, A < B**



1-bit Magnitude Comparator

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



A > B : AB'

A < B : A'B

A = B : A'B' + AB

2-bit Magnitude Comparator

Comparing two 2-bit binary numbers: $A=A_1A_0$ $B=B_1B_0$

Truth Table

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

Conditions

1. A=B

$$A = B = (A_1 \text{ XNOR } B_1) \cdot (A_0 \text{ XNOR } B_0)$$

2. A>B

If $A_1 > B_1$, then A > B (regardless of LSB).

If $A_1 = B_1$ and $A_0 > B_0$, then A > B.

$$A > B = (A_1 \cdot B_1') + [(A_1 \odot B_1) \cdot (A_0 \cdot B_0')]$$

3.A<B

If $A_1 < B_1$, then A < B.

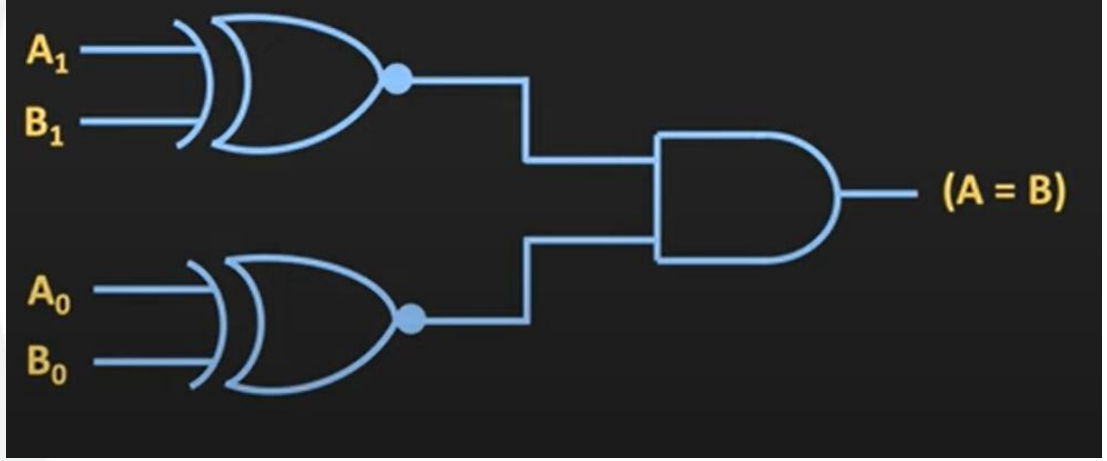
If $A_1 = B_1$ and $A_0 < B_0$, then A < B.

$$A < B = (A_1' \cdot B_1) + [(A_1 \odot B_1) \cdot (A_0' \cdot B_0')]$$

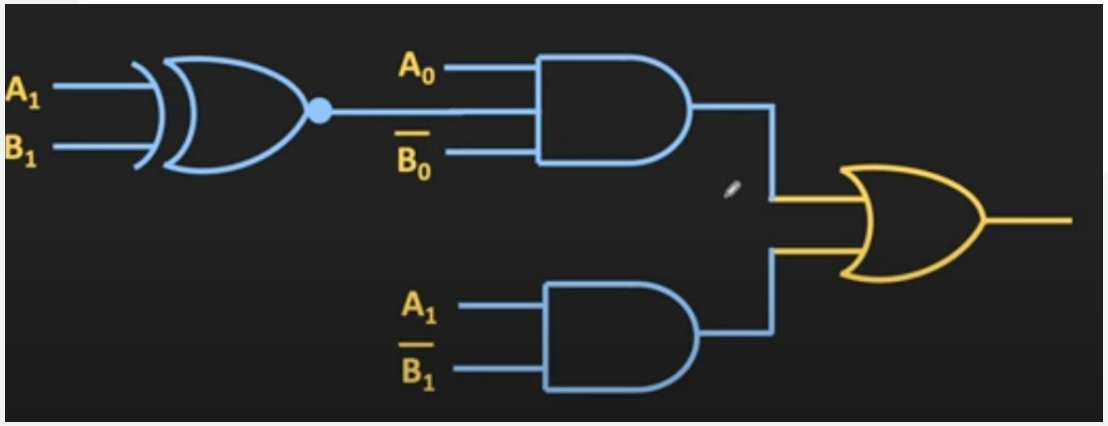
RV College of Engineering®

2 Bit Comparator Logic Circuit

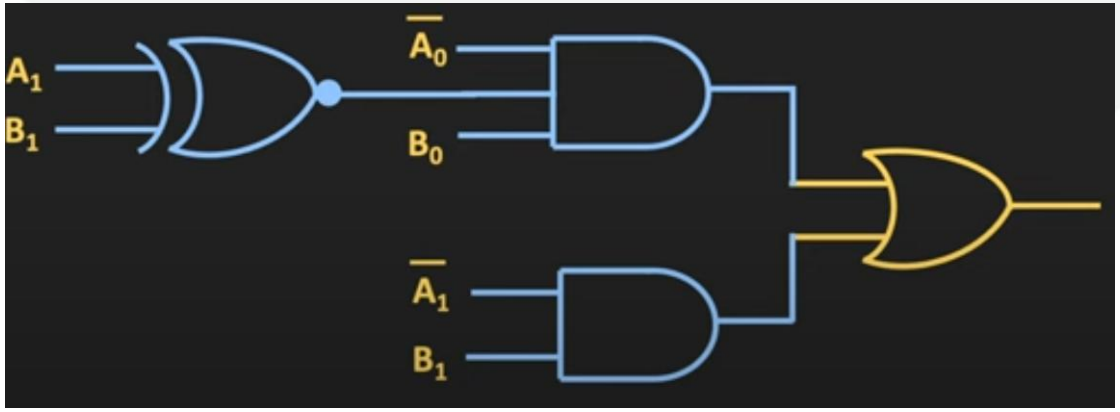
$$(A = B) = (A_1 \odot B_1)(A_0 \odot B_0)$$



$$(A > B) = (A_1 \overline{B_1}) + (A_1 \odot B_1)(A_0 \overline{B_0})$$



$$(A < B) = (\overline{A_1} B_1) + (A_1 \odot B_1)(\overline{A_0} B_0)$$

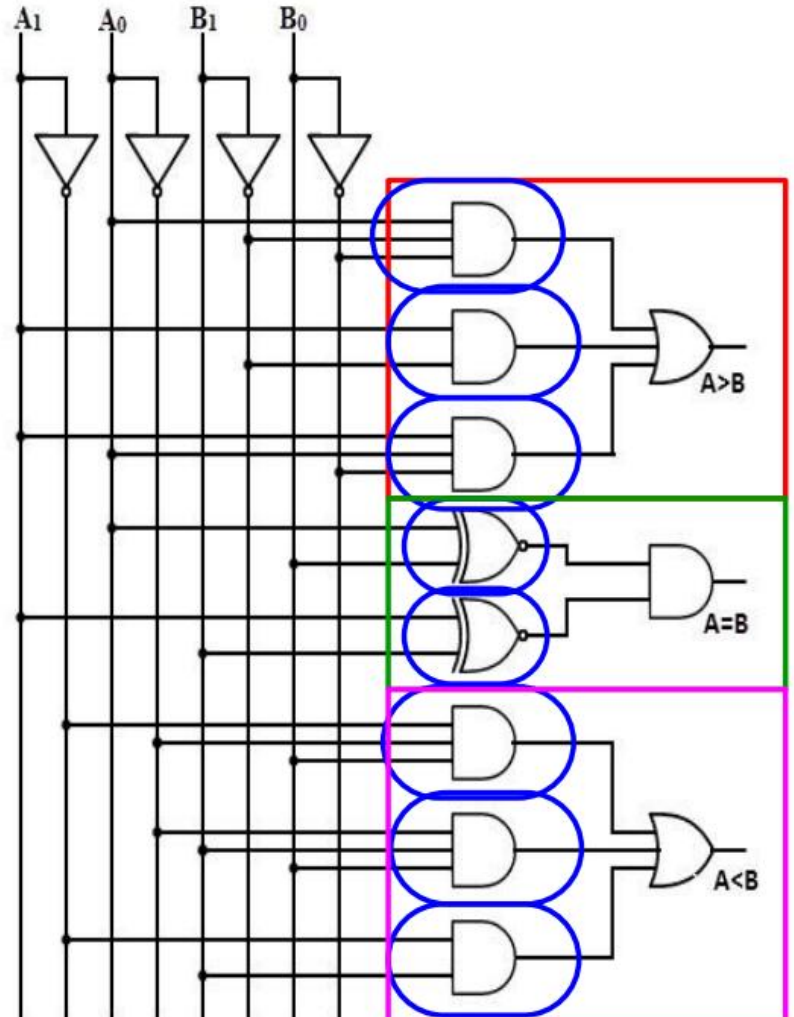


1- bit Comparator	2 variables	4 rows
2- bit Comparator	4 variables	16 rows
n- bit Comparator	2n variables	2^{2n} rows

$$A>B:\ G = A0\,\overline{B1}\ \overline{B0} + A1\,\overline{B1} + A1\ A0\,\overline{B0}$$

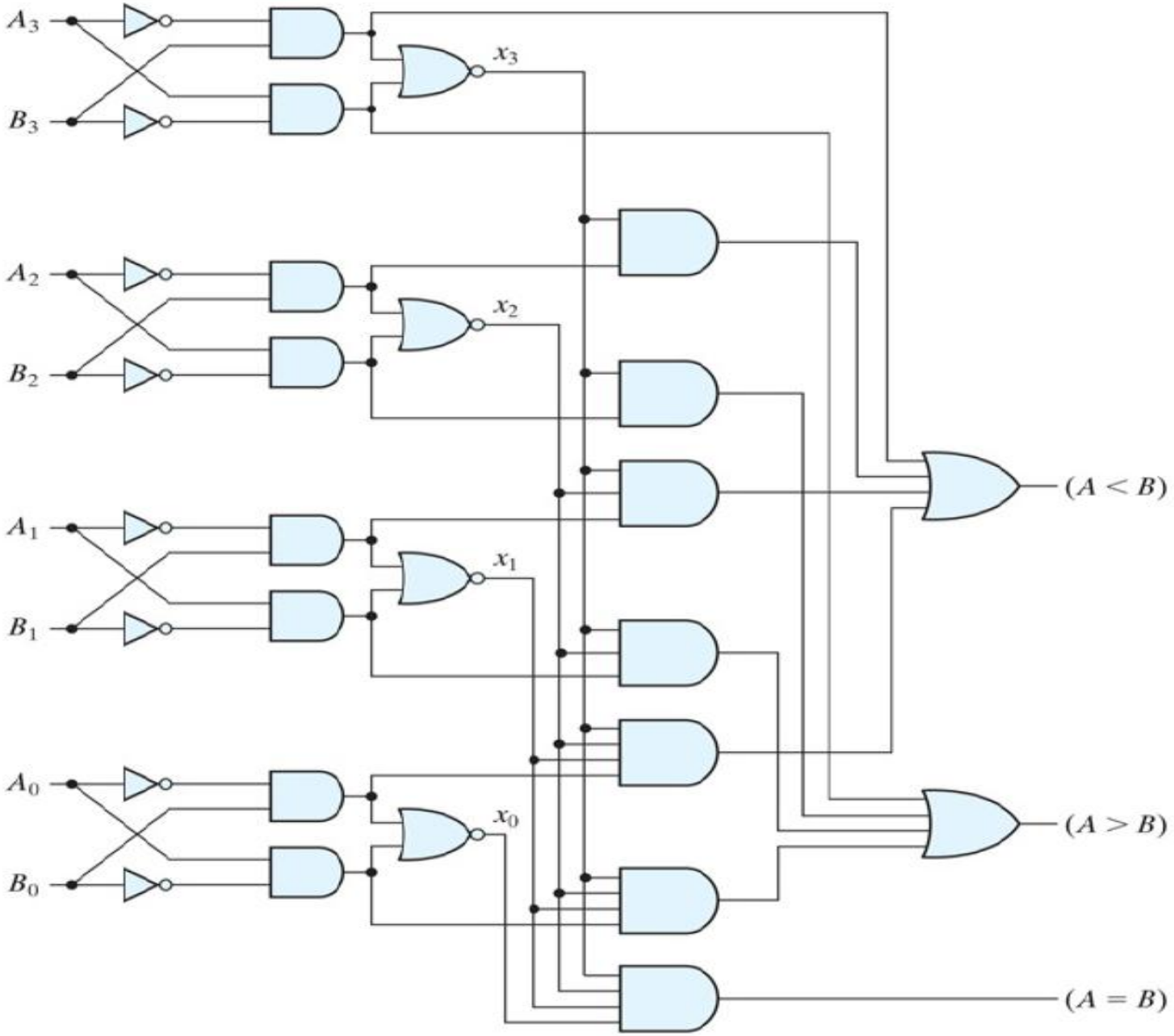
$$A=B:\ E = \overline{(A0 \oplus B0)}\ \overline{(A1 \oplus B1)}$$

$$A<B:\ L = \overline{A1}\ B1 + \overline{A0}\,B1\,B0 + \overline{A1}\ \overline{A0}\,B0$$



RV College of Engineering®

Logic Circuit of 4 bit Magnitude Comparator



Suggested Reading

- TTL,CMOS,ECL IC technologies
- IC Packages
- Datasheets of ICs