UNIT 4/5

COMBINATIONAL LOGIC

Combinational Logic

- **Combinational logic** is a *type of digital logic which is implemented by boolean circuits, where the output is a pure function of the present input only.*
- No memory to store data.

Combinational circuit:

- Consists of *n input variables*, *logic gates, and m output variables*.
- The logic gates accept signals from the inputs and generate signals to the outputs.
- For *n input variables, there are 2^*n possible combinations of binary input values.
- For *each possible input combination*, *there is one and only one possible output combination.*

- A combinational circuit can be described by *m Boolean functions, one for each output variable.*
- Each output function is expressed in terms of the *n input variables.*
- The *n input* binary variables come from an external source; the *m output* variables go to an external destination.

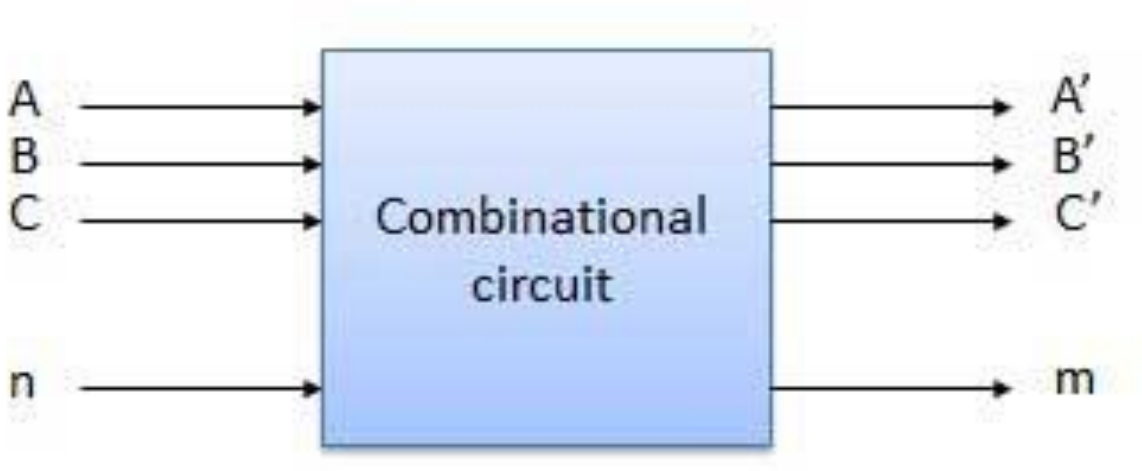


Fig: Block diagram of combinational circuit

Design procedure

- *Starts from the verbal outline of the problem and ends in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be easily obtained.*
- Involves the following steps:
  - **1. Specification**
    - *Write a specification for the circuit if one is not already available*
  - **2. Formulation**
    - *Derive a truth table or initial Boolean equations that define the required relationships between the inputs and outputs, if not in the specification.*
    - *Apply hierarchical design if appropriate*
  - **3. Optimization**
    - *Apply 2-level and multiple-level optimization*
    - *Draw a logic diagram for the resulting circuit using ANDs, ORs, and inverters*
  - **4. Technology mapping**
    - *Map the logic diagram to the implementation technology selected*
  - **5. Verification**
    - *Verify the correctness of the final design manually or using simulation*

- Simply, we can list out the design procedure of combinational circuits as:
  1. *The problem is stated.*
  2. *The number of available input variables and required output variables is determined.*
  3. *The input and output variables are assigned letter symbols.*
  4. *The truth table that defines the required relationships between inputs and outputs is derived.*
  5. *The simplified Boolean function for each output is obtained.*
  6. *The logic diagram is drawn.*

Adder

- Arithmetic operation.
- The most basic arithmetic operation-addition of two binary digits.
- Half-adder/Full adder

1. Half-Adder

- A combinational circuit that performs the addition of two bits.
- Circuit needs **two inputs and two outputs.**
- The input variables designate **the augend (x) and addend (y) bits**; the output variables produce **the sum (S) and carry (C).**
- Now we formulate a Truth table to exactly identify the function of half-adder.

Fig: Block diagram of Half Adder

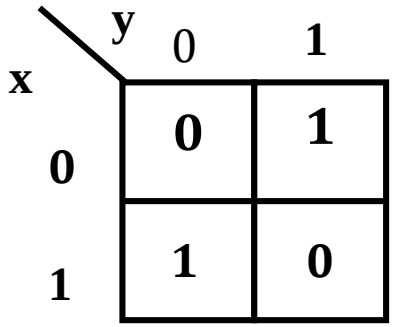
Inputs		Outputs	
x	**y**	**S**	**C**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Truth table

- The simplified Boolean functions for the two outputs can be obtained directly from the truth table.
- The simplified sum of products expressions are:

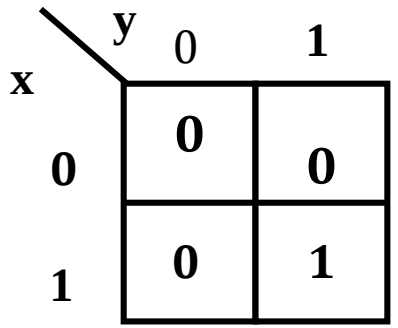
$$S = x'y + xy'$$

$$C = xy$$



Sum (S) = xy'+x'y

Fig: K-map for SUM



Carry (C) =xy

Fig: K-map for Carry

Implementation:

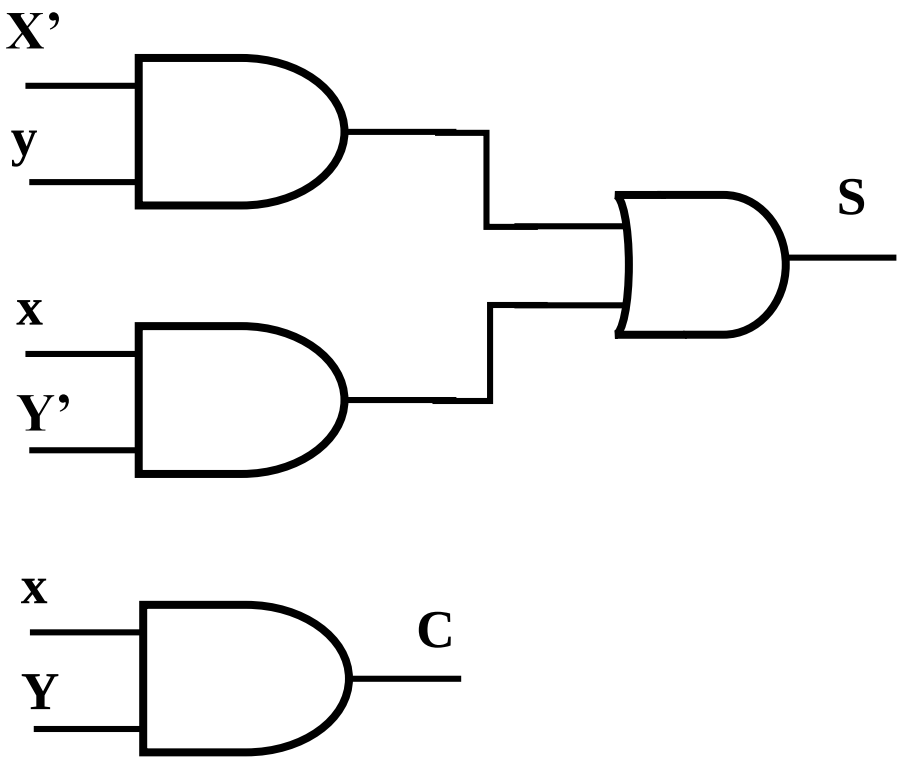


Fig: S=x’y+xy’
c=xy

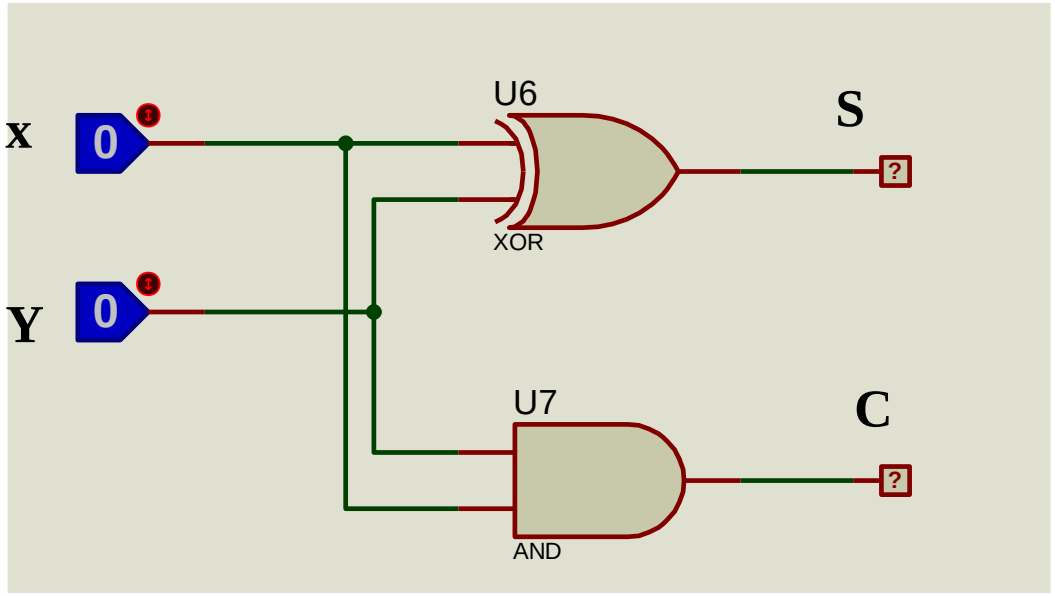


Fig: S=x⊕y
c=xy

2. Full Adder

- Add 3-bits (two inputs and one carry)
- Produces sum and carry

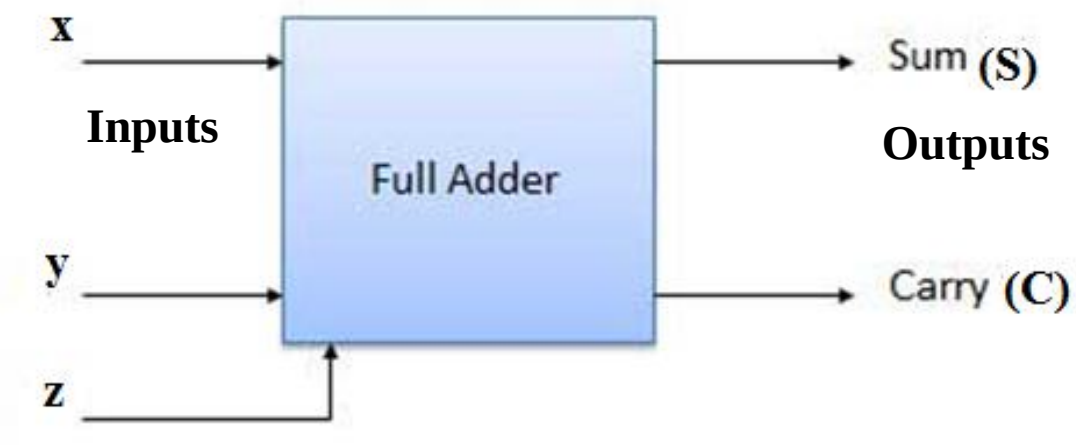


Fig: Block diagram of Full Adder

Inputs			Outputs	
x	**y**	**Cin**	**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

Truth table

Note: Z=Cin

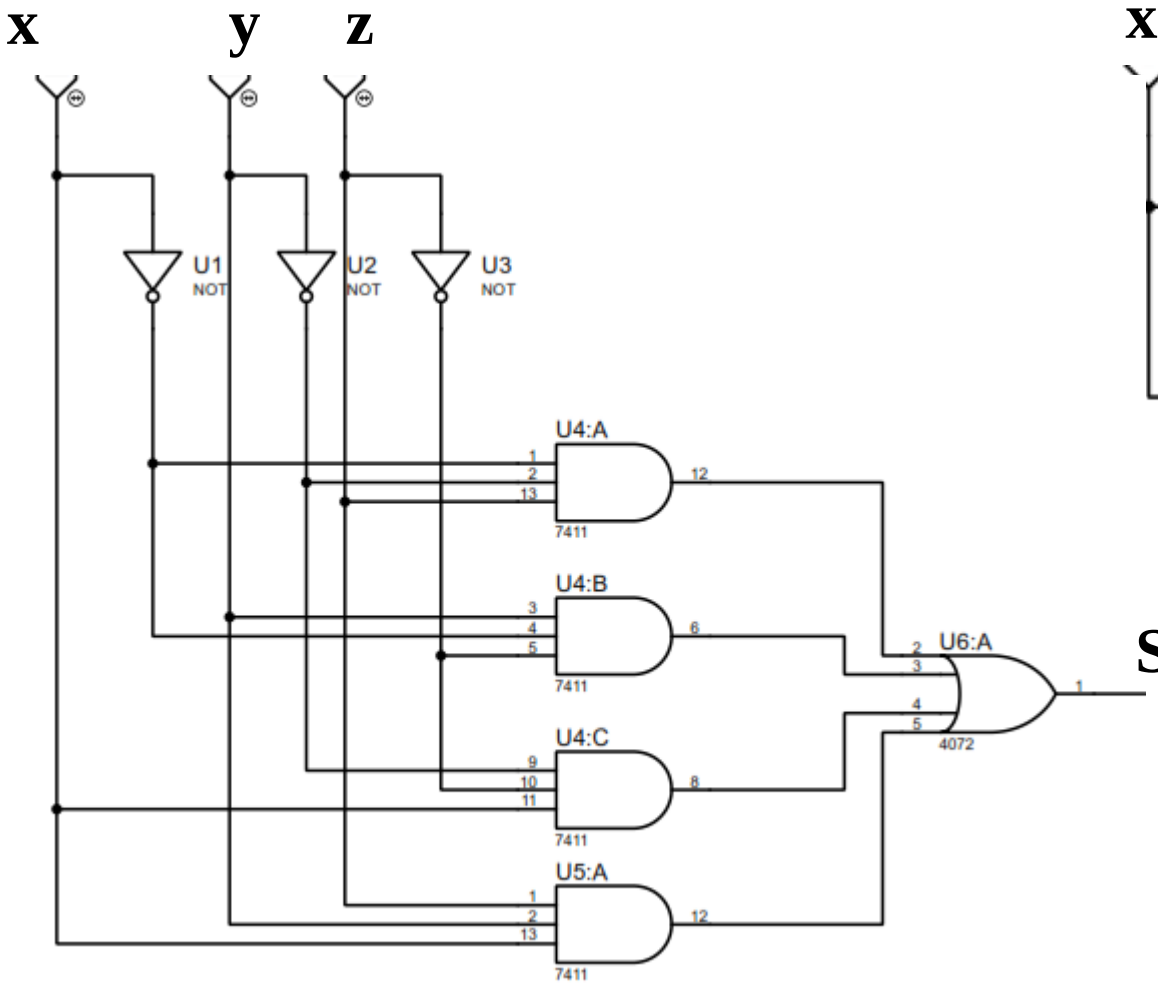
$$Sum\ (S) = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$$
$$= x \oplus y \oplus z$$
$$Carry\ (C) = xy + xz + yz$$

x \ yz	$\bar{y}\bar{z}$ (00)	$\bar{y}z$ (01)	yz (11)	$y\bar{z}$ (10)
$\bar{x}$ (0)	**0**	**1**	**0**	**1**
x (1)	**1**	**0**	**1**	**0**

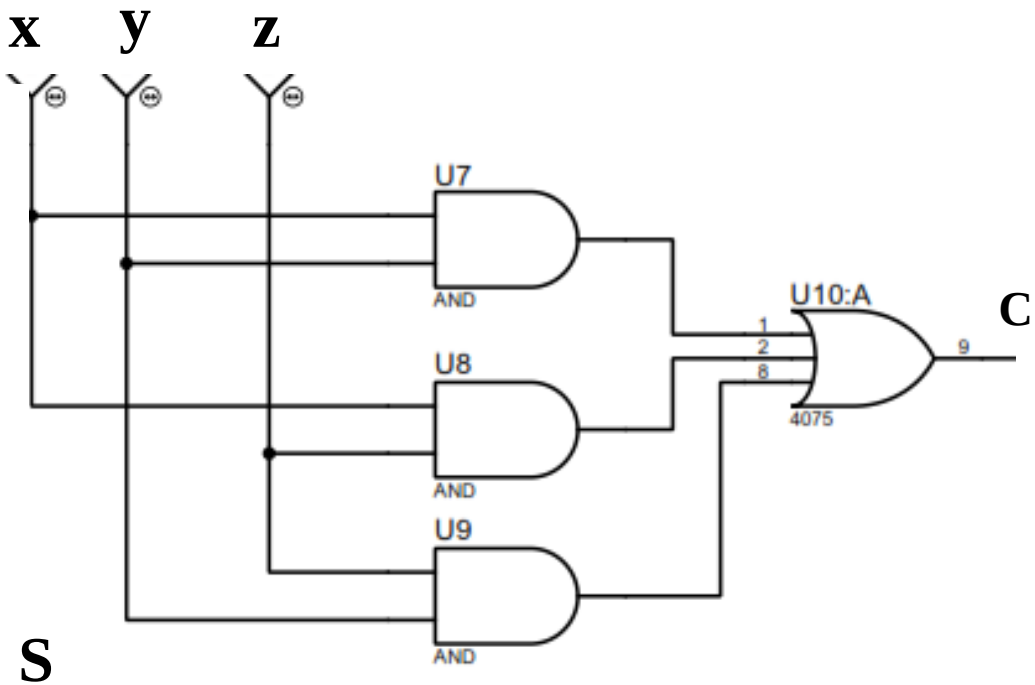
Fig: K-map for SUM

x \ yz	$\bar{y}\bar{z}$ (00)	$\bar{y}z$ (01)	yz (11)	$y\bar{z}$ (10)
$\bar{x}$ (0)	**0**	**0**	**1**	**0**
x (1)	**0**	**1**	**1**	**1**

Fig: K-map for Carry

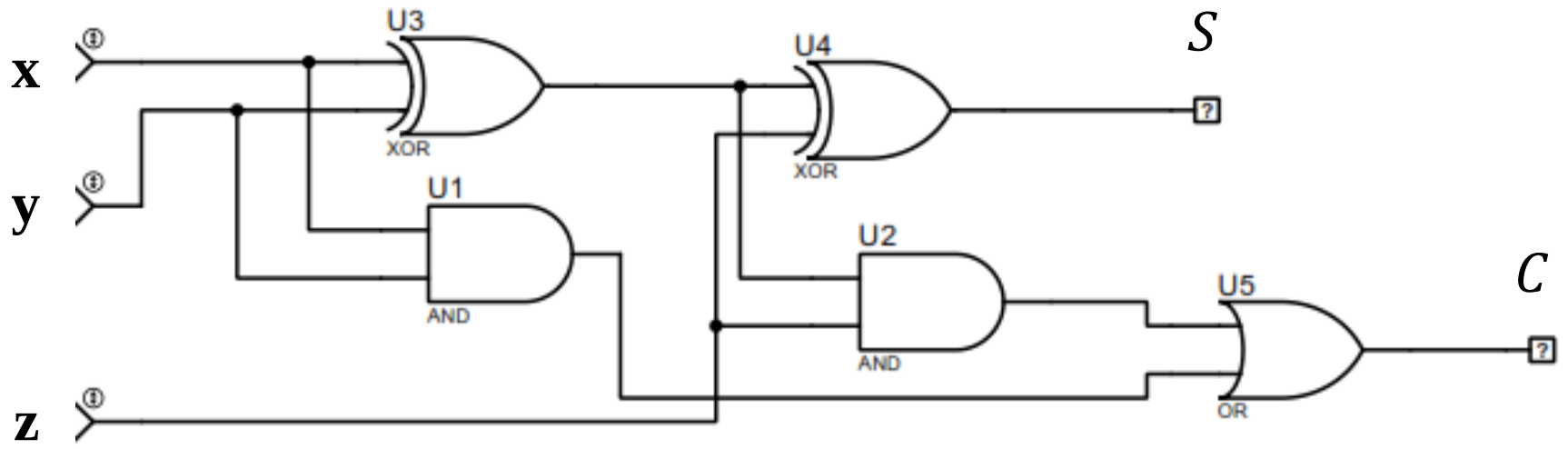


$$Sum\ (S) = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$$



$$Carry\ (C) = xy + xz + yz$$

- ***This approach uses two half-adders and a OR gate.***



$$Sum\ (S) = \mathrm{x} \oplus y \oplus z$$

$$Carry\ (C) = xy + (\mathrm{x} \oplus y)z$$

Subtractor

- Subtraction of two binary numbers is accomplished by taking the complement of the subtrahend and adding it to the minuend.

0-0=0

0-1=1

1-0=1

1-1=0

1. Half Subtractor

- Performs subtraction of two numbers of one bit and produces difference.
- Two input as X and Y and two output as difference (D) and borrow (B).

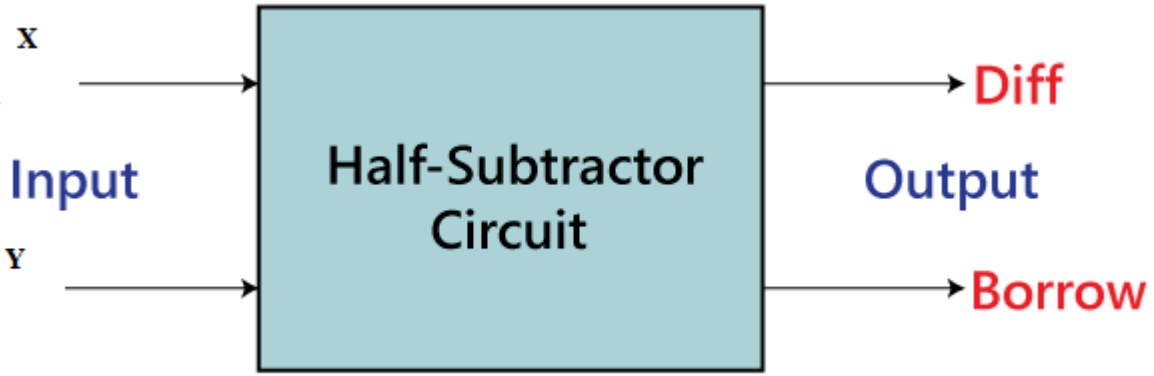


Fig: Block diagram of Half Subtractor

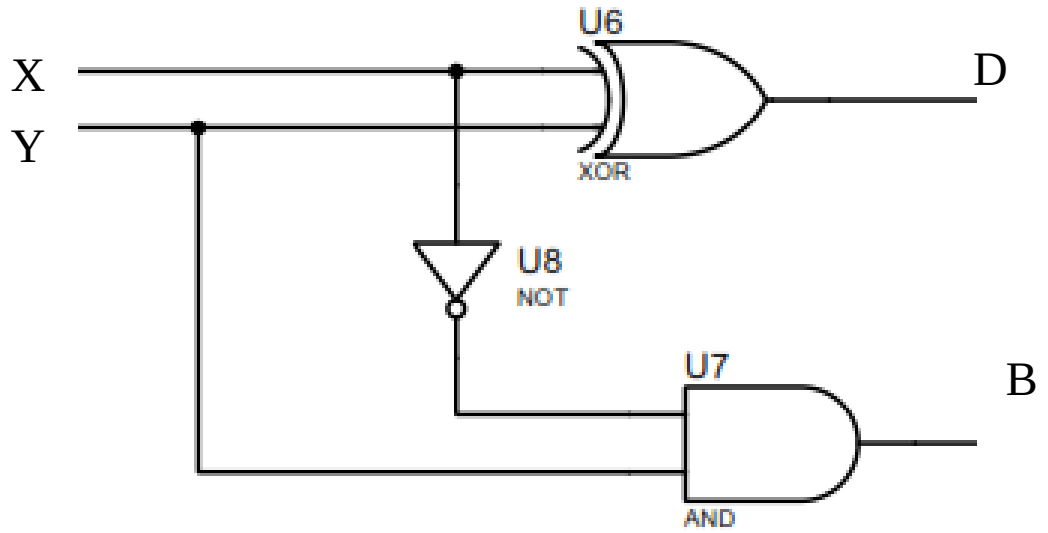
Inputs		Outputs	
X	**Y**	**B**	**D**
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Truth table for (X-Y)

- The Boolean function of the above truth table is:

  Difference (D) $= \bar{X}Y + X\bar{Y} = X \oplus Y$

  Borrow (B) $= \bar{X}Y$



2. Full Subtractor

- Subtraction of three single bit numbers.

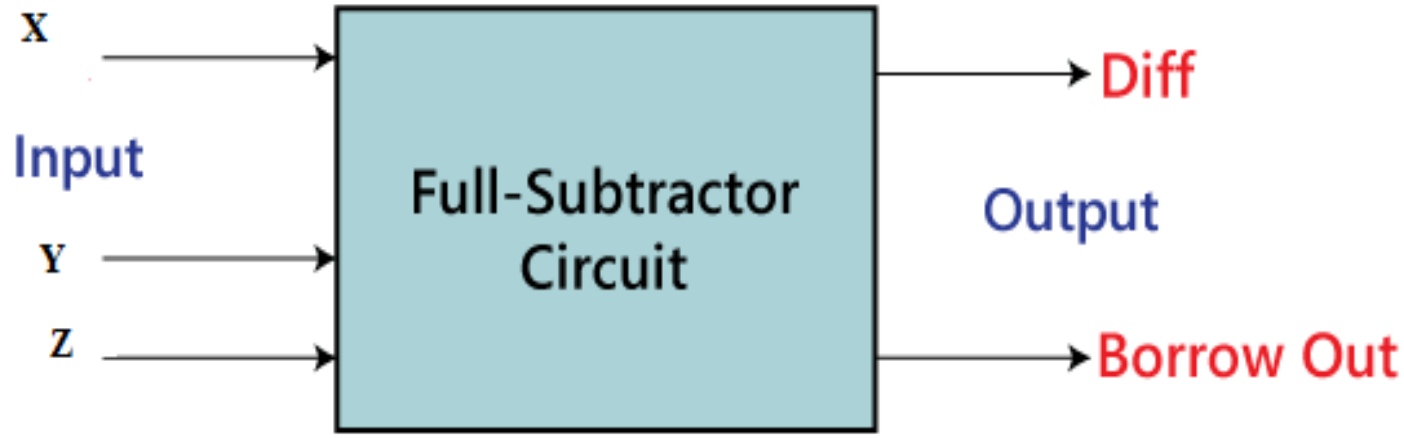


Fig: Block diagram of Full Subtractor

Inputs			Outputs	
X	**Y**	**Z**	**B**	**D**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Truth table

- The Boolean function from the truth table is:

$$\textit{Difference (D)} = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ$$
$$= X \oplus Y \oplus Z$$
$$\textit{Borrow (B)} = \bar{X}Y + \bar{X}Z + YZ$$
$$= \bar{X}Y + (\overline{X \oplus Y}).Z$$

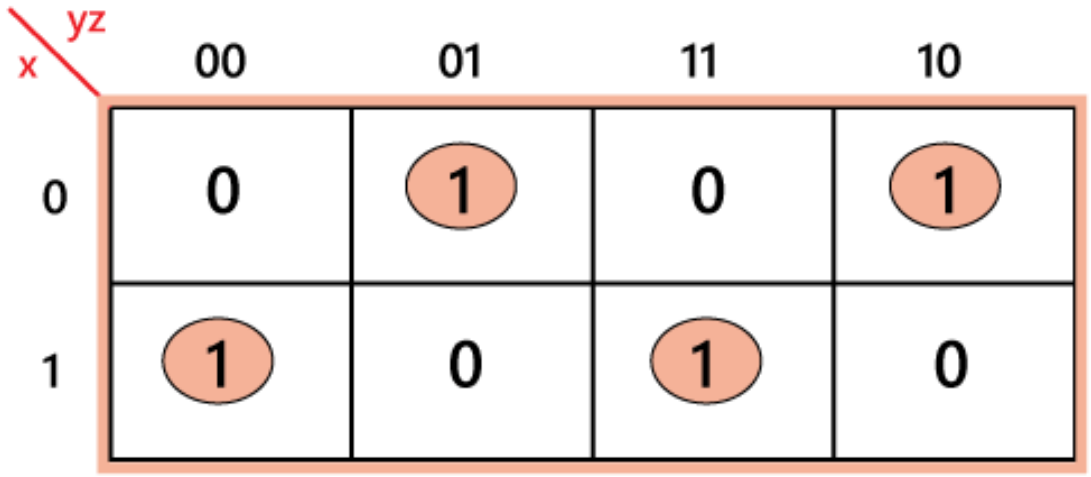


Fig: K-map for Difference

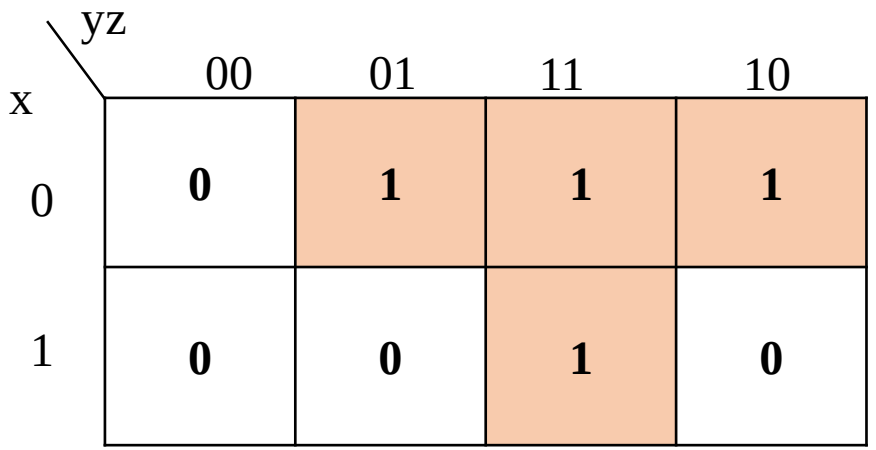
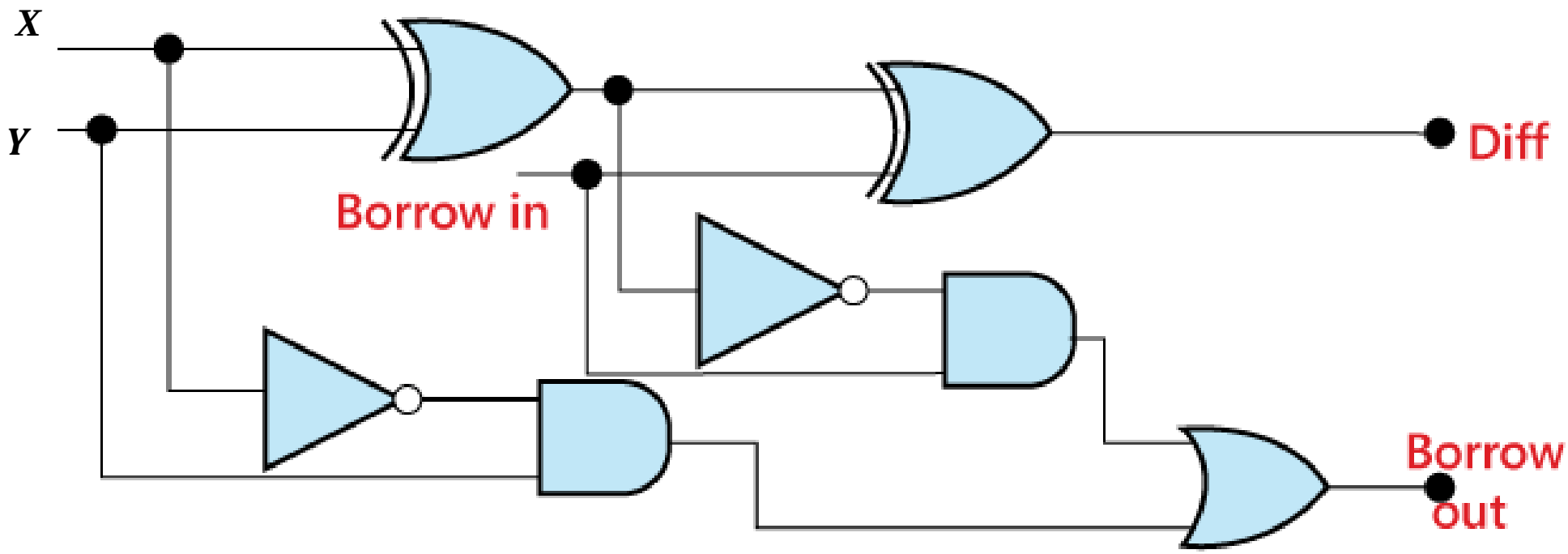


Fig: K-map for Borrow

- ***This approach uses two half-subtractors and a OR gate.***



Z=Borrow in

Binary Adder

- A logical circuit which is used to perform the addition operation of two binary number of any length.
- Formed with the help of the full-adder circuit.
- The full-adders are cascaded in series, and the output carry of the first adder will be treated as the input carry of the next full-adder.
- **N-bit parallel adder:**
  - To add two n-bit binary numbers rather than only single-bit binary numbers.
  - Need to use n-bit parallel adder.
  - In order to get n-bit parallel adder, we cascade the n number of full adders.

4-bit Binary Adder

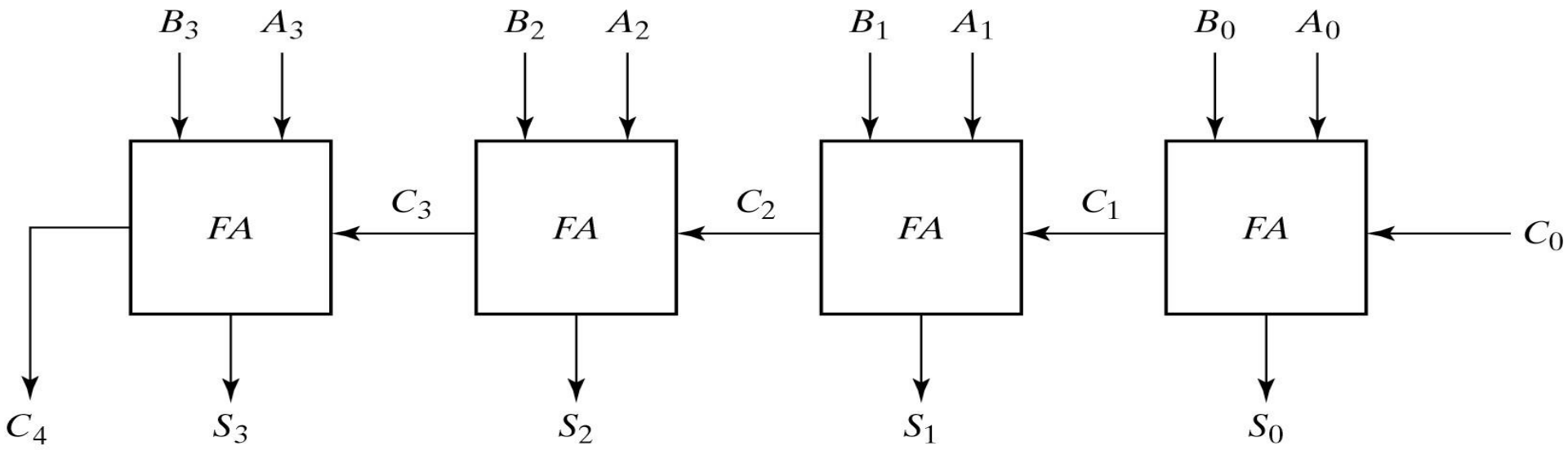
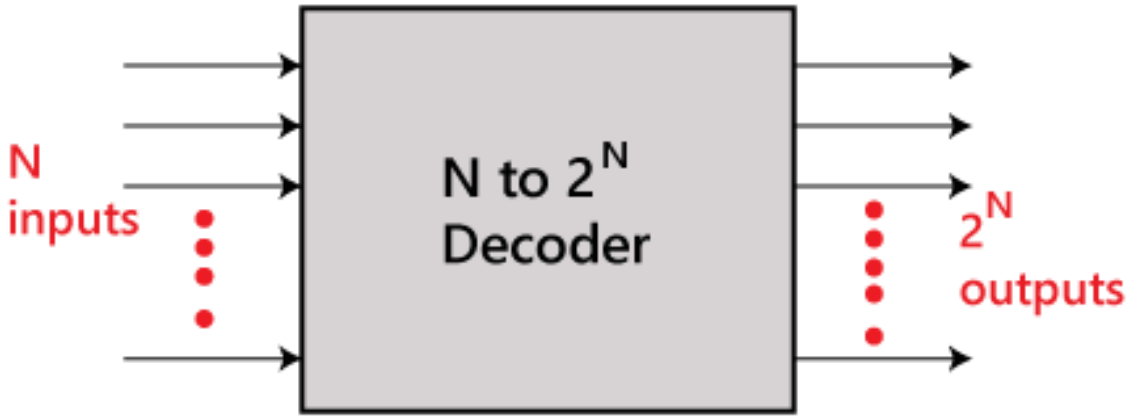


Fig. 4-9 4-Bit Adder

- '**A**' and '**B**' – augend and addend.
- **C0, C1, C2,** and **C3** - carry inputs which are connected together as a chain using Full Adder.
- **C4** → $\mathbf{C_{out}}$
- **S0, S1, S2,** and **S3** - sum outputs that produce the sum of augend and addend bits.

Decoder

- Combinational circuit that changes the ***N-bit*** binary input information into ***2^N output lines***.



- At a time, only one input line is activated for simplicity.
- The produced 2^N-bit output code is equivalent to the input binary information.
- Process is called ***Decoding***.
- Opposite of ***Decoding*** is ***Encoding***.

1. 2-to-4 line decoder:

- Two inputs- A_0 and A_1
- Four outputs- Y_0,Y_1, Y_2, and Y_3
- Single enable line- E

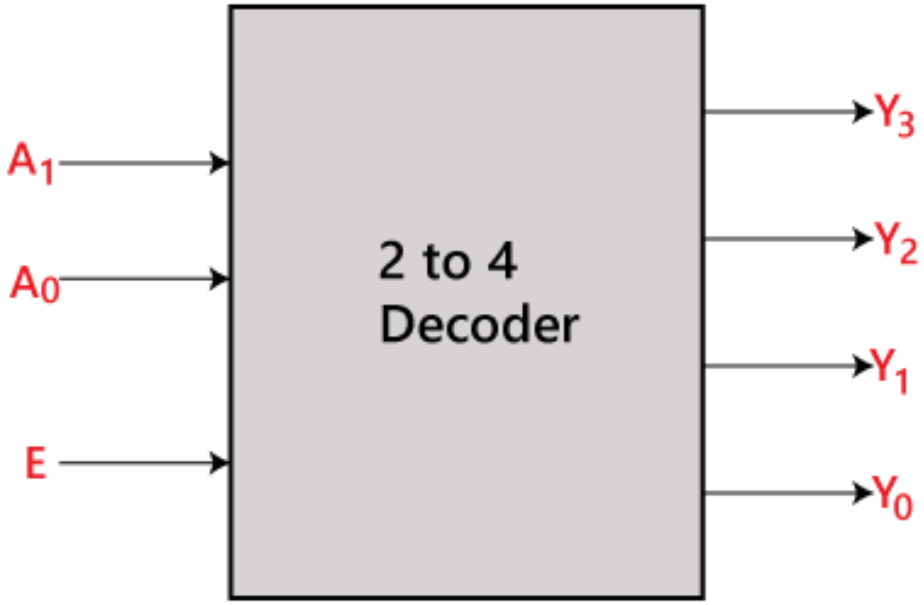


Fig: Block diagram of 2 to 4 Decoder

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

Table: Truth Table foe 2 to 4 Decoder

The logical expression of the term Y0, Y0, Y2, and Y3 is as follows:

$$Y_3=E.A_1.A_0$$
$$Y_2=E.A_1.A_0'$$
$$Y_1=E.A_1'.A_0$$
$$Y0=E.A_1'.A_0'$$

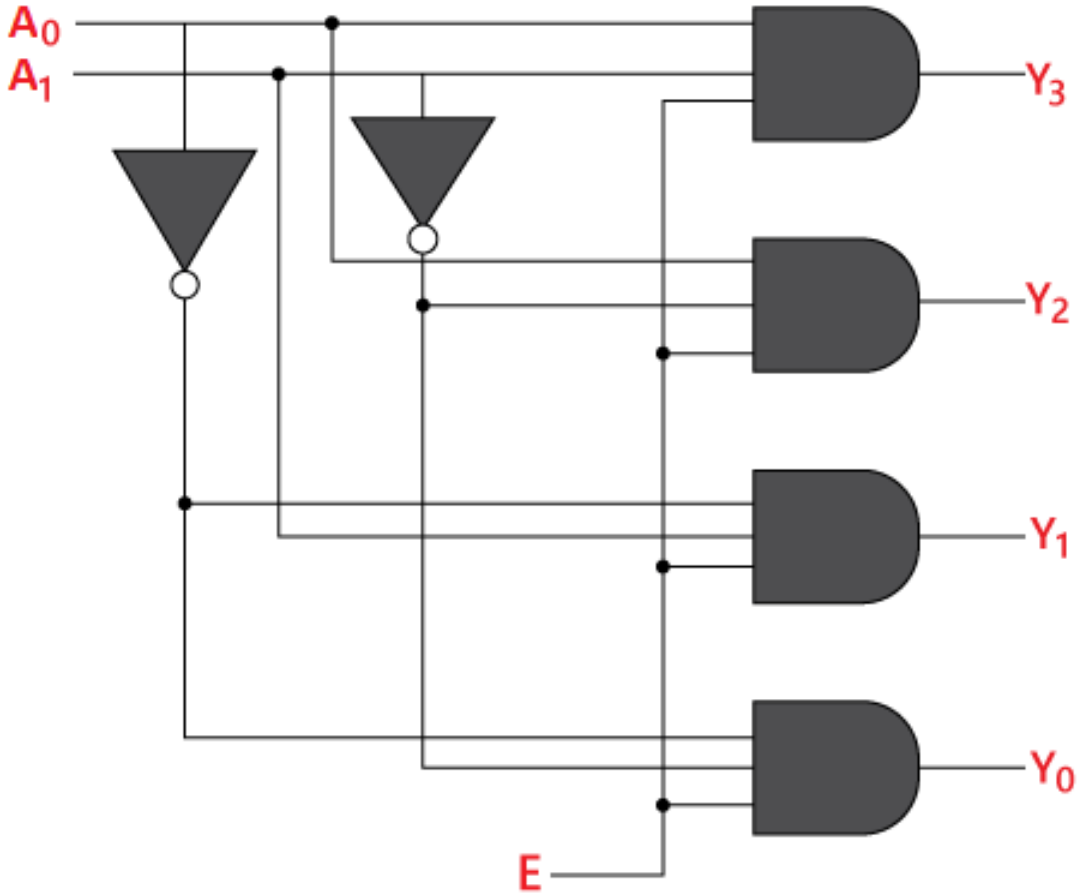


Fig: Logical circuit diagram of 2 to 4 decoder

2. 3-to-8 line Decoder

- Three inputs- A_0, A_1 and A_2
- Eight outputs- $Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_7$
- Single enable line- E

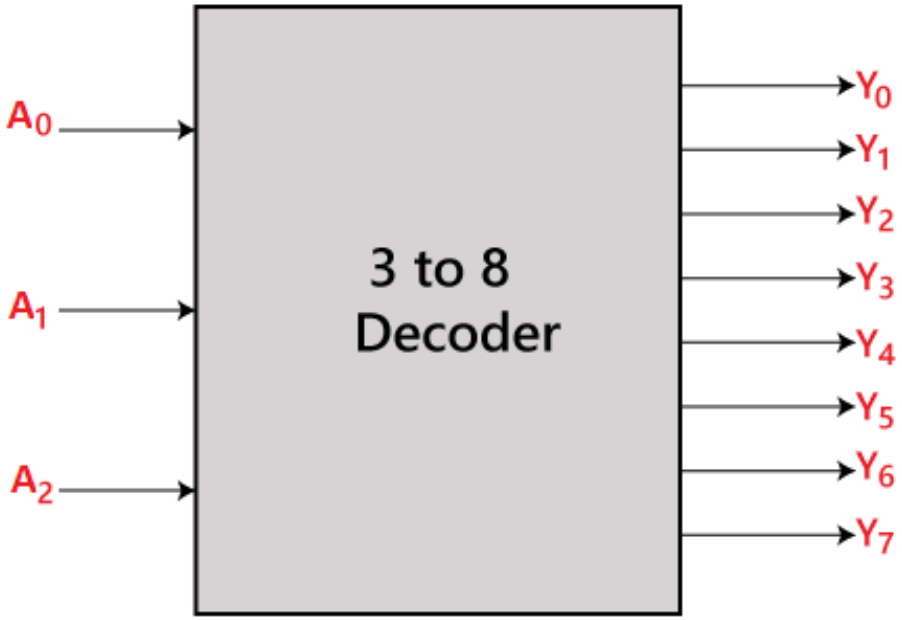


Fig: Block diagram of 3 to 8 Decoder

Enable	INPUTS			Outputs							
E	A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Table: Truth Table foe 3 to 8 Decoder

The logical expression of the term Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7 is as follows:

$$Y_0 = A_0'.A_1'.A_2'$$
$$Y_1 = A_0.A_1'.A_2'$$
$$Y_2 = A_0'.A_1.A_2'$$
$$Y_3 = A_0.A_1.A_2'$$
$$Y_4 = A_0'.A_1'.A_2$$
$$Y_5 = A_0.A_1'.A_2$$
$$Y_6 = A_0'.A_1.A_2$$
$$Y_7 = A_0.A_1.A_2$$

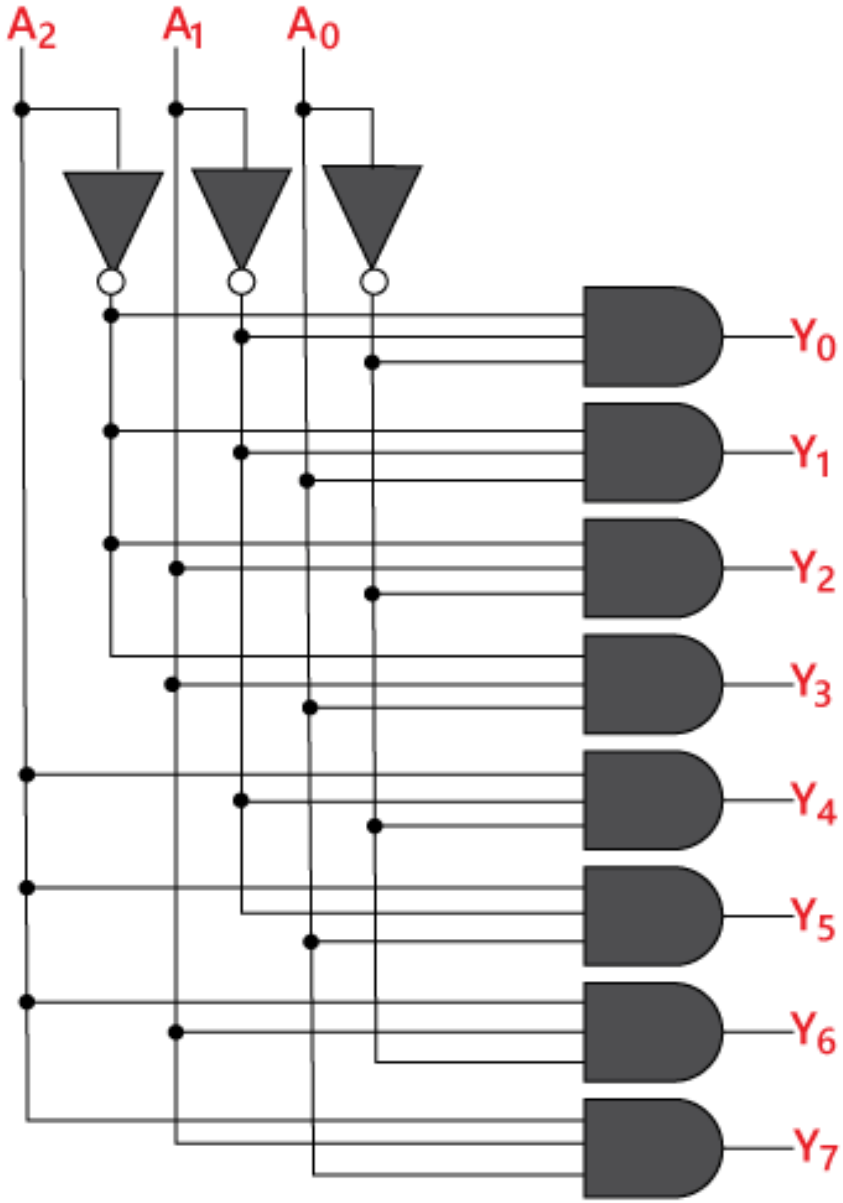


Fig: Logical circuit diagram of 3 to 8 decoder

3. 4 to 16 line Decoder

- Four inputs- A_0, A_1, A_2 and A_3
- Sixteen outputs- Y_0,Y_1, Y_2, ….Y_{16}
- Can be constructed using either 2 to 4 decoder or 3 to 8 decoder.
- The required number of lower-order decoders is given by,

  Required number of lower order decoders=m_2/m_1

  For, $m_1 = 8$, $m_2 = 16$

  Required number of 3 to 8 decoders=16/8=2

- **Operation:**
  a. *When **A_3=0**; the lower decoder will be activated (enabled) and upper decoder will be deactivated. At that time the lower decoder will generate the Minterms from 0000 to 0111 i.e. m0 to m7.*
  b. *When **A_3=1**; the lower decoder will be deactivated and upper decoder will be activated. At that time the upper decoder will generate the Minterms from 1000 to 1111 i.e. m8 to m15.*

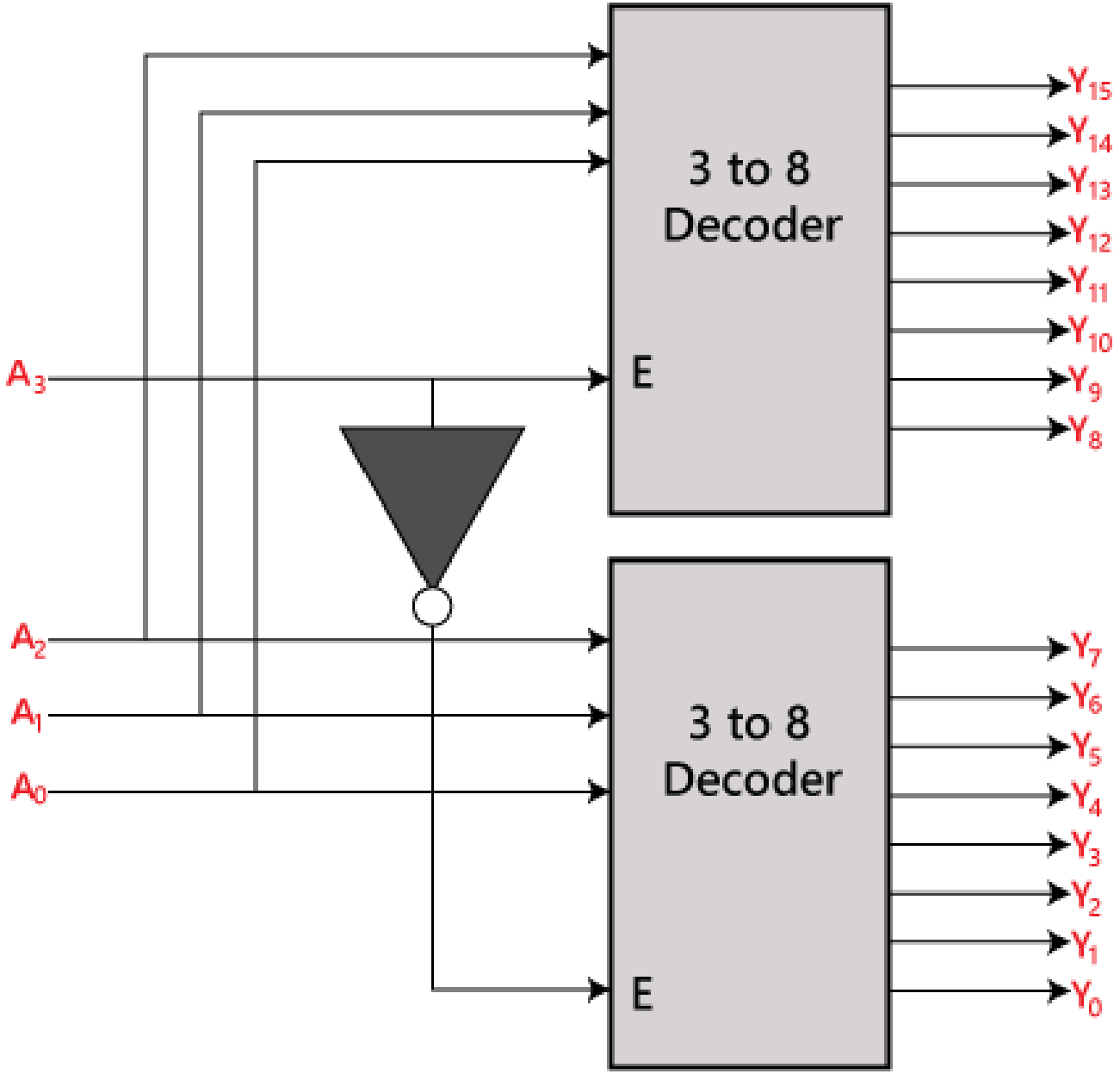
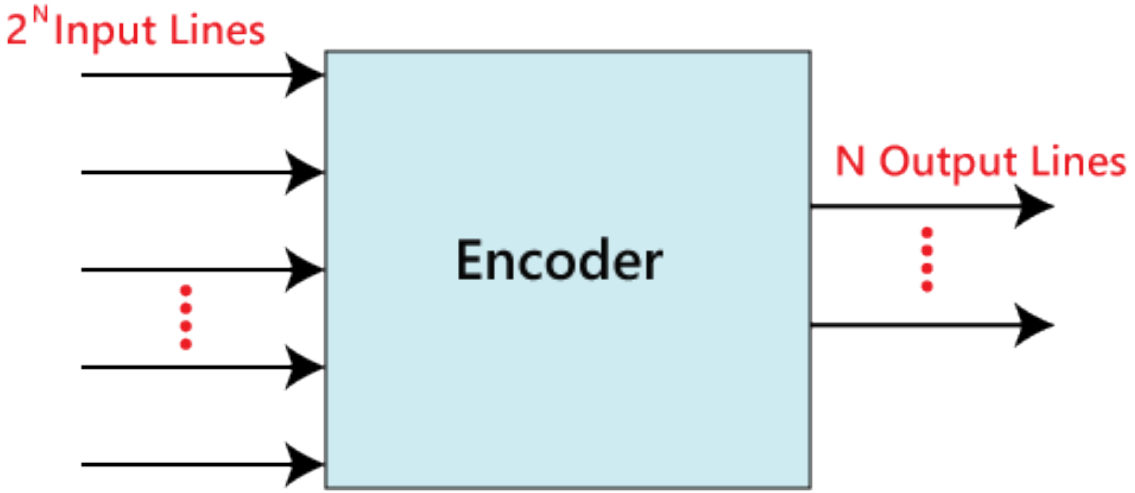


Fig: Logical circuit diagram of 4 to 16 decoder using 3 to 8 decoder

INPUTS				OUTPUTS															
A_3	A_2	A_1	A_0	Y_{15}	Y_{14}	Y_{13}	Y_{12}	Y_{11}	Y_{10}	Y_9	Y_8	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Encoder

- Reverse operation of decoder.
- Has ***2^n input*** lines and ***n output*** lines.
- The output is correspondence to the input line.
- Encodes the information from 2^n input into an n-bit code.



1. 4 to 2 line Encoder:

- Four inputs, ***i.e.,*** Y_0, Y_1, Y_2, ***and*** Y_3, and two outputs, ***i.e.,*** A_0 ***and*** A_1.
- One input-line is set to true at a time to get the respective binary code in the output side.

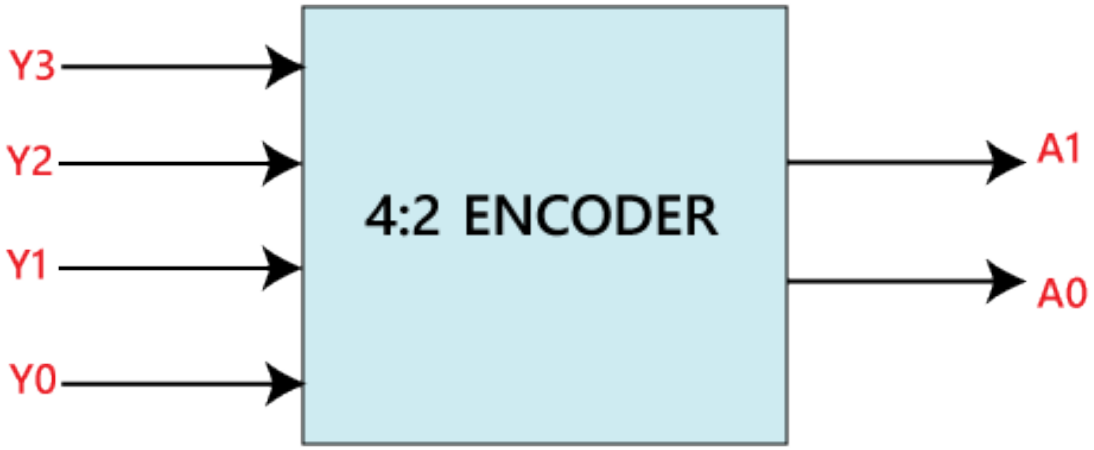


Fig: Block diagram of 4 to 2 Encoder

Inputs				Outputs	
Y3	**Y2**	**Y1**	**Y0**	**A1**	**A0**
0	**0**	**0**	**1**	**0**	**0**
0	**0**	**1**	**0**	**0**	**1**
0	**1**	**0**	**0**	**1**	**0**
1	**0**	**0**	**0**	**1**	**1**

Table: Truth Table foe 4 to 2 Encoder

The logical expression of the term A0 and A1 is as follows:

$$A_1 = Y_3 + Y_2$$

$$A_0 = Y_3 + Y_1$$

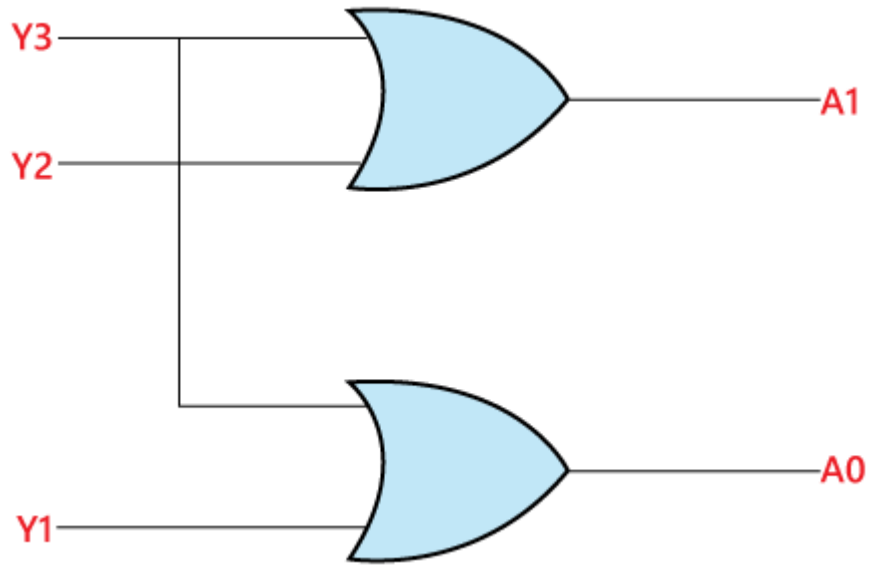


Fig: Logical circuit diagram of 4 to 2 Encoder

Limitation: only one input can be active at a time.
Solution: use parity encoder.

2. Priority Encoder:

- A parity encoder is such that if two or more inputs are active at same time, the input having highest priority will take precedence.

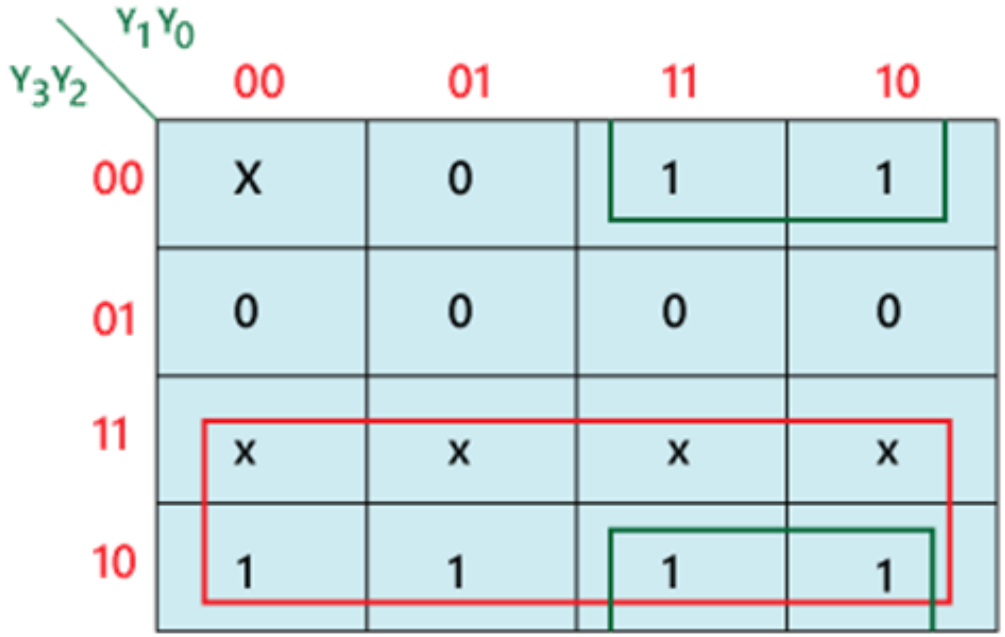
4 to 2 line Priority Encoder:

- The **x’**s designates don’t care condition.
- The Y_3 has high and Y_0 has low priority.
- The logical expression of the term A_0 and A_1 can be found using **K-map** as:

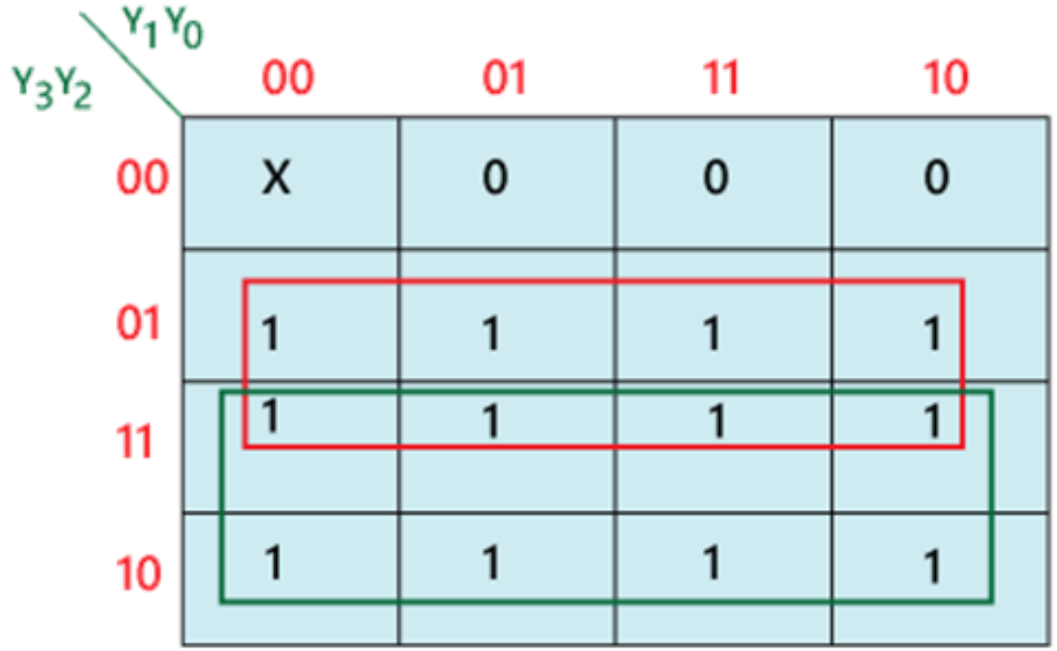
$$A_1 = Y_3 + Y_2$$
$$A_0 = Y_3 + Y_2' . Y_1$$

Inputs				Outputs	
Y3	**Y2**	**Y1**	**Y0**	**A1**	**A0**
0	**0**	**0**	**1**	**0**	**0**
0	**0**	**1**	**x**	**0**	**1**
0	**1**	**x**	**x**	**1**	**0**
1	**x**	**x**	**x**	**1**	**1**

Table: Truth Table foe 4 to 2 Priority Encoder



A0=Y3+Y2'.Y1



A1=Y2+Y3

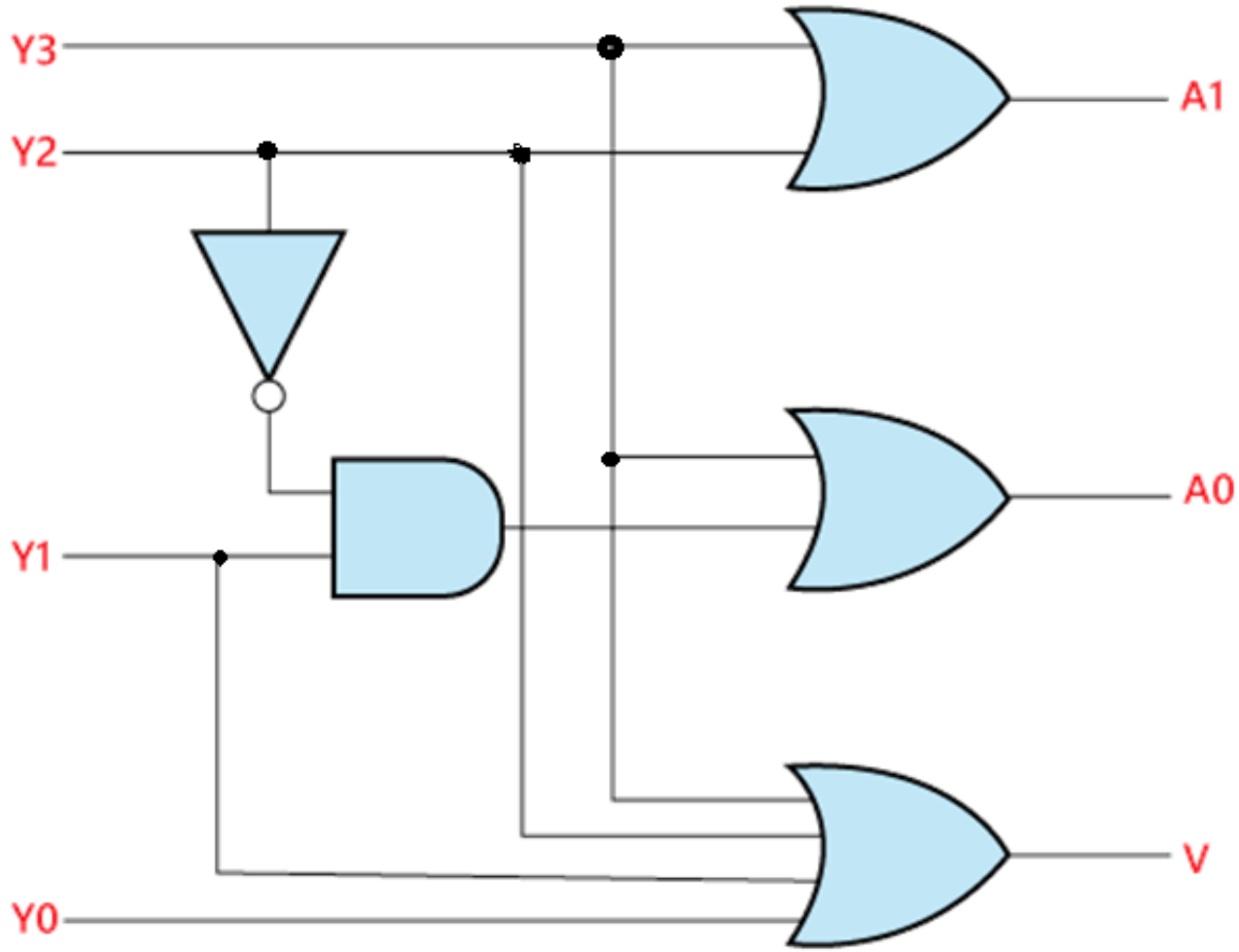


Fig: Logical circuit diagram of 4 to 2 Priority Encoder

3. 8 to 3 line Encoder:

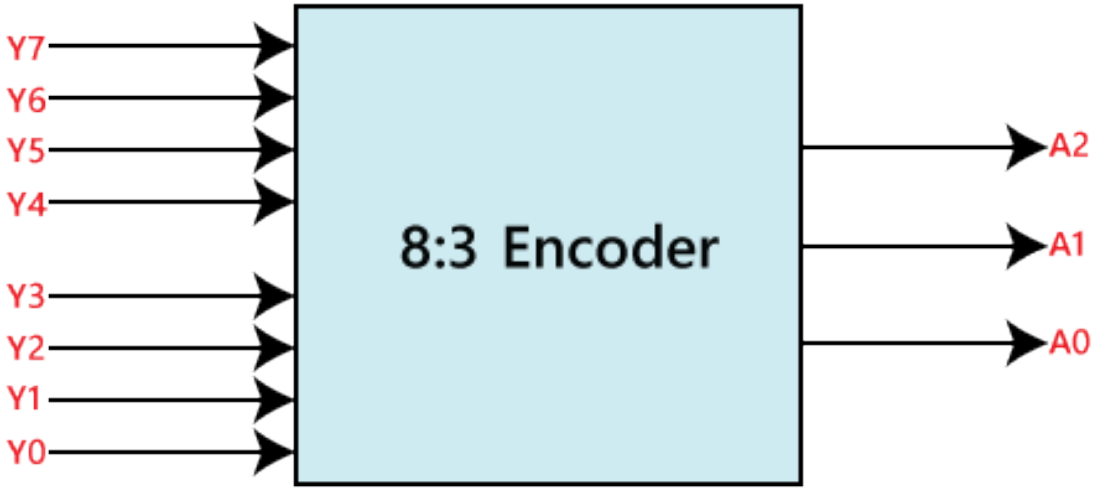


Fig: Block diagram of 8 to 3 Encoder

The logical expression of the term A0, A1, and A2 are as follows:

$$A_2=Y_4+Y_5+Y_6+Y_7$$

$$A_1=Y_2+Y_3+Y_6+Y_7$$

$$A_0=Y_7+Y_5+Y_3+Y_1$$

INPUTS								OUTPUTS		
Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Table: Truth Table foe 8 to 3 Encoder

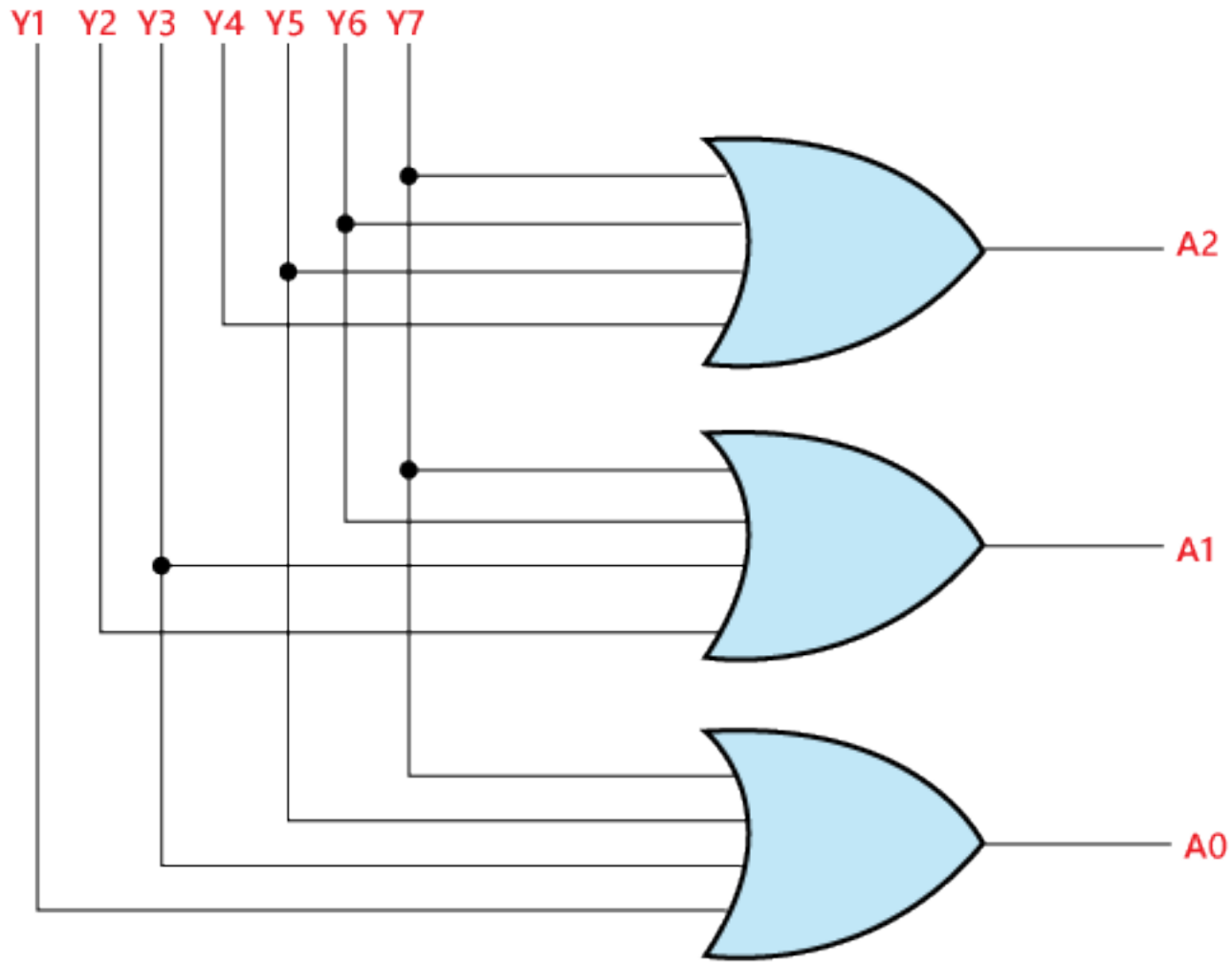


Fig: Logical circuit diagram of 8 to 3 Encoder

4. Decimal to BCD Encoder

- Also known as **10 to 4 line encoder**.
- Total of ten inputs, i.e., Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_7, Y_8, and Y_9 and four outputs, i.e., A_0, A1, A_2, and A_3.
- In 10-input lines, one input-line is set to true at a time to get the respective **BCD code** in the output side.

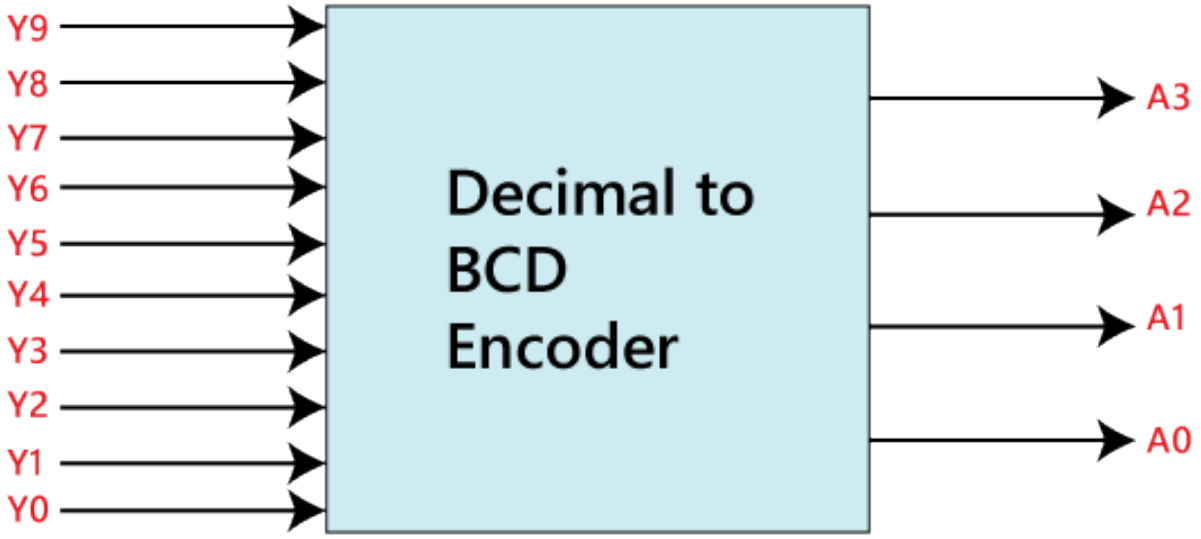


Fig: Block diagram of Decimal to BCD Encoder

The logical expression of the term A_0, A_1, A_2, and A_3 is as follows:

A3 = Y9 + Y8

A2 = Y7 + Y6 + Y5 +Y4

A1 = Y7 + Y6 + Y3 +Y2

A0 = Y9 + Y7 +Y5 +Y3 + Y1

INPUTS										OUTPUTS			
Y_9	Y_8	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_3	A_2	A_1	A_0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

Table: Truth Table foe Decimal to BCD Encoder

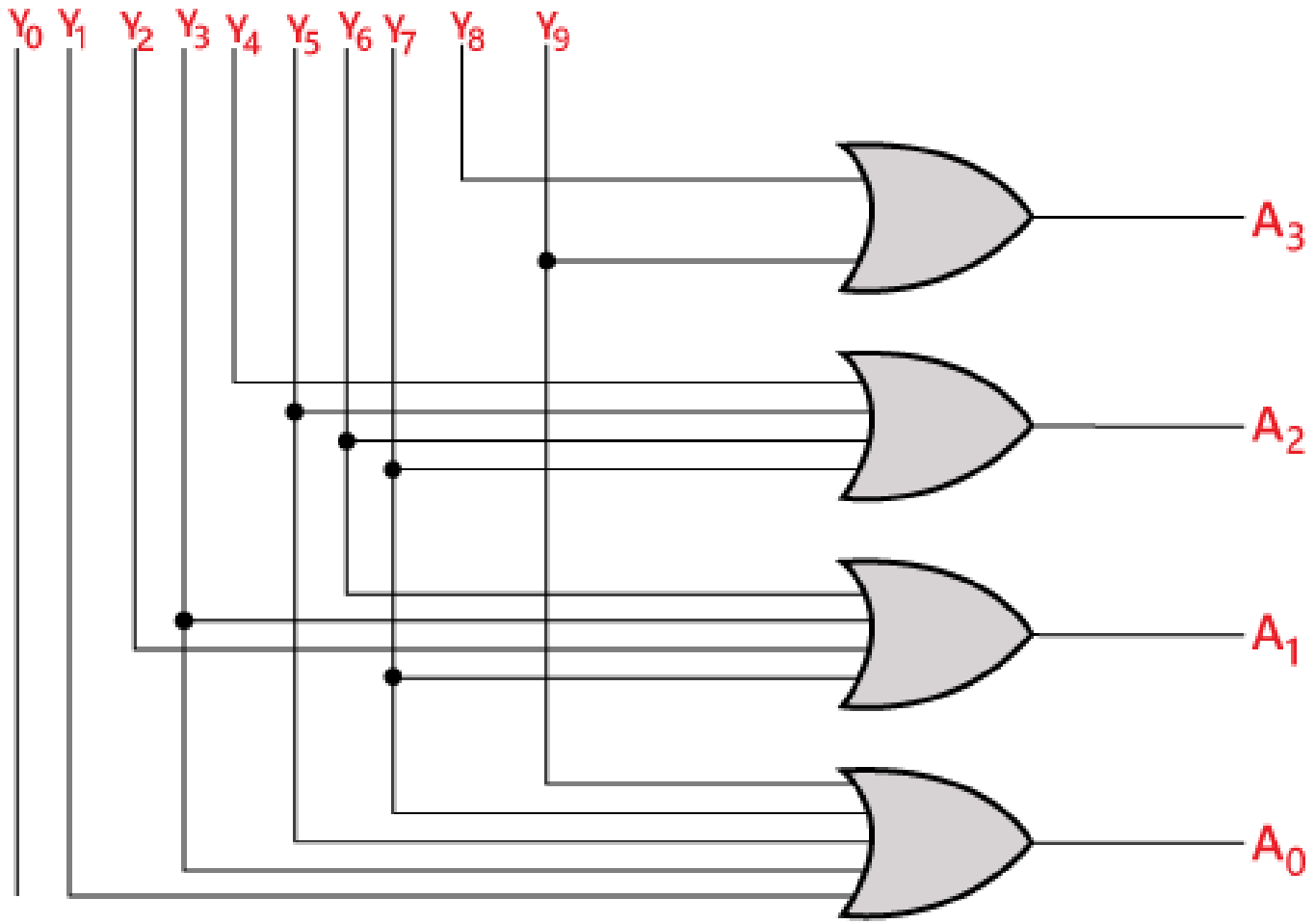


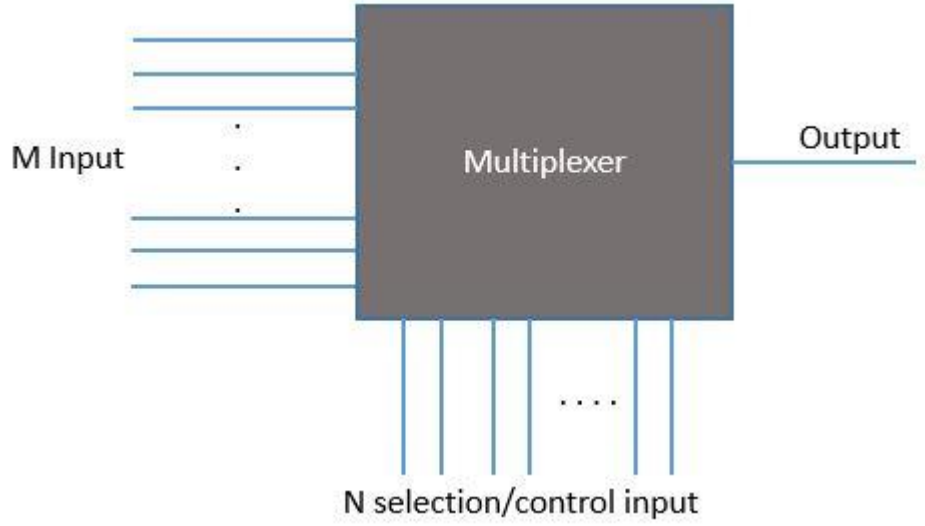
Fig: Logical circuit diagram of Decimal to BCD Encoder

Multiplexers and De-multiplexers

- A combinational circuit that has ***2^n input lines*** and a ***single output line***.
- Simply, multi-input and single-output combinational circuit.
- On the basis of the values of the **selection lines**, one of these **data inputs will be connected to the output.**
- ***n selection lines*** and ***2^n input lines***.
- A multiplexer is also treated as **mux**.
- ***De-multiplexer*** is a circuit that performs function exactly ***reverse of multiplexer***.
- Generally, both used together, because ***communication systems are bi-directional***.

1. Multiplexer/Data selector

- Combinational circuit that select binary information from one of many line and direct it to a single output line.
- Example: ***multi-position switch***
- In Digital application it is called ***digital multiplexer.***
- Number of selection lines (S)= $\boldsymbol{log_2 M}$***, where M= no of inputs***.



2×1 Multiplexer:

- only two inputs, i.e., A_0 and A_1, 1 selection line, i.e., S_0 and single outputs, i.e., Y.
- On the basis of selection S_0 the input are selected one at a time.

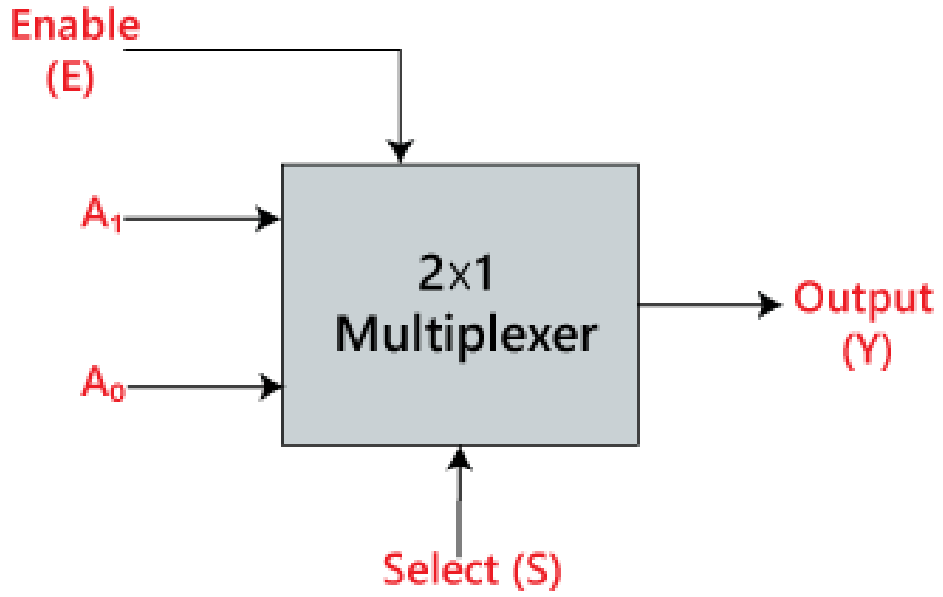


Fig: Block diagram of 2 x 1 MUX

Inputs		Selector	Output
A_1	A_0	S_0	Y
-	A_0	0	A_0
A_1	-	1	A_1

Table: Truth Table

The logical expression of the term Y is as follows:

$$Y=S_0'.A_0+S_0.A_1$$

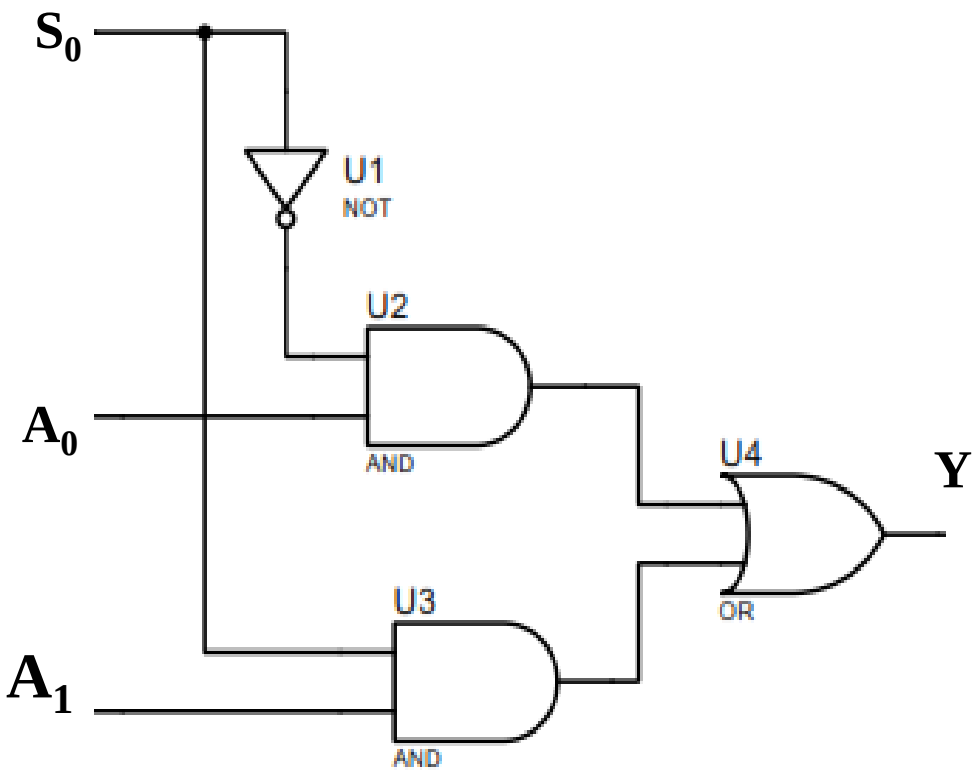


Fig: Logical circuit diagram of 2x1 MUX

4×1 Multiplexer:

- Four inputs, i.E., A_0, A_1, A_2, and A_3.
- Selection lines, i.E., S_0 and S_1.
- Single output, i.E., Y.

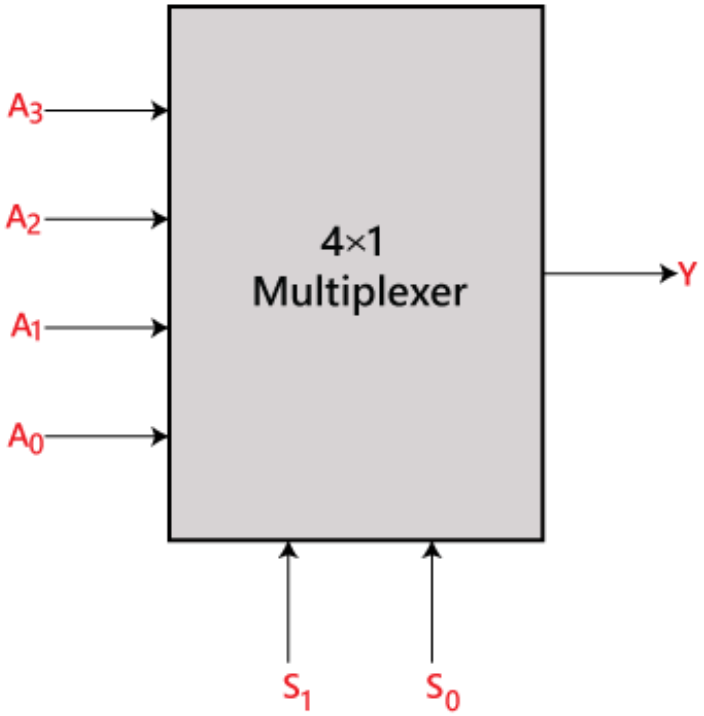


Fig: Block diagram of 4 x 1 MUX

Inputs				Selector		Output
A_3	A_2	A_1	A_0	S_1	S_0	Y
-	-	-	A_0	0	0	A_0
-	-	A_1	-	0	1	A_1
-	A_2	-	-	1	0	A_2
A_3	-	-	-	1	1	A_3

Table: Truth Table

The logical expression of the term Y is as follows:

$$Y=S_1' S_0' A_0+S_1' S_0 A_1+S_1 S_0' A_2+S_1 S_0 A_3$$

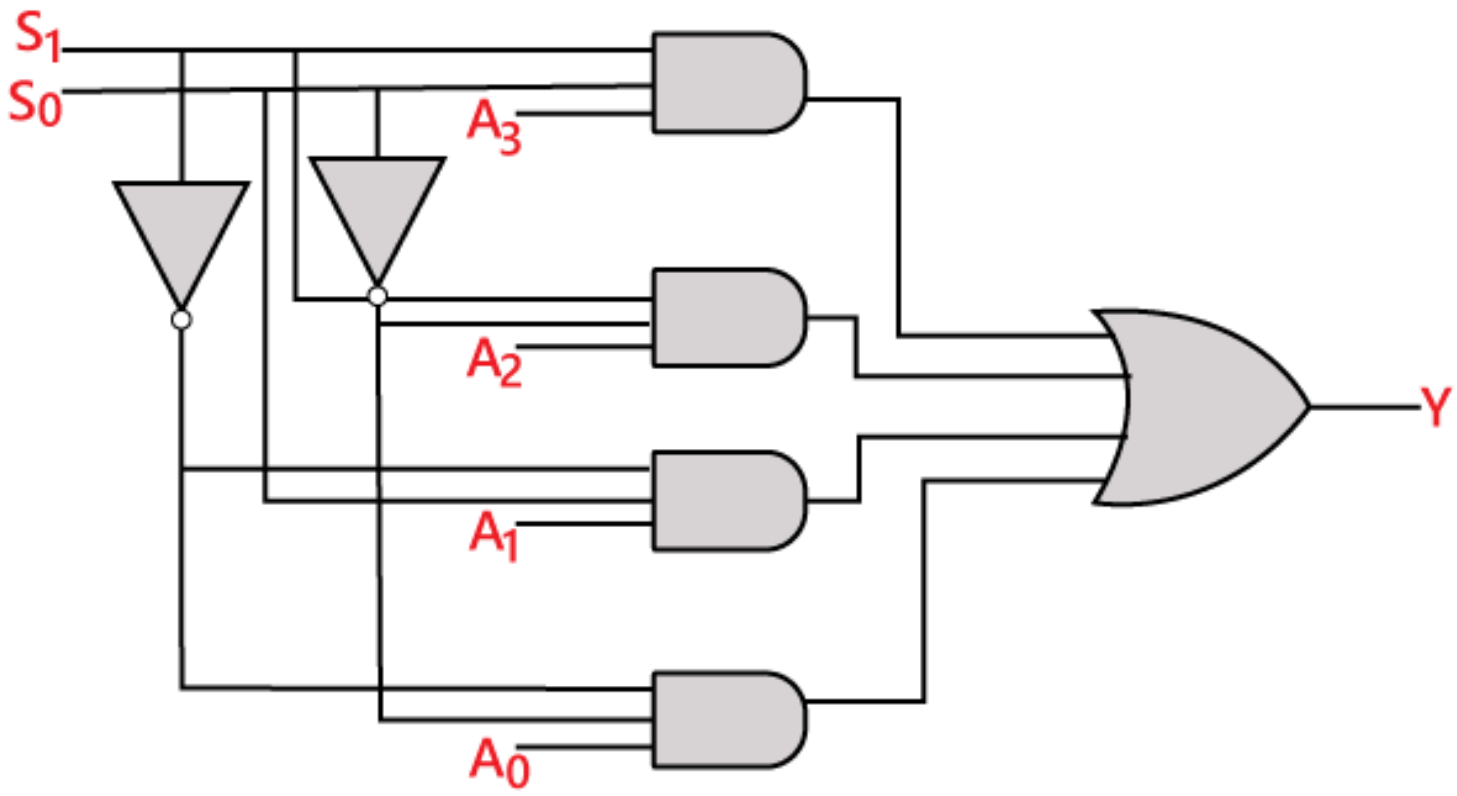


Fig: Logical circuit diagram of 4x1 MUX

8 to 1 Multiplexer

- Eight inputs, i.e., A_0, A_1, A_2, A_3, A_4, A_5, A_6, and A_7
- 3 selection lines, i.e., S_0, S_1 and S_2
- Single output, i.e., Y.

The logical expression of the term Y is as follows:

$$Y=S_0'.S_1'.S_2'.A_0+S_0.S_1'.S_2'.A_1+S_0'.S_1.S_2'.A_2 +S_0.S_1.S_2'.A_3+S_0'.S_1'.S_2\ A_4+S_0.S_1'.S_2\ A_5+S_0' .S_1.S_2\ .A_6+S_0.S_1.S_3.A_7$$

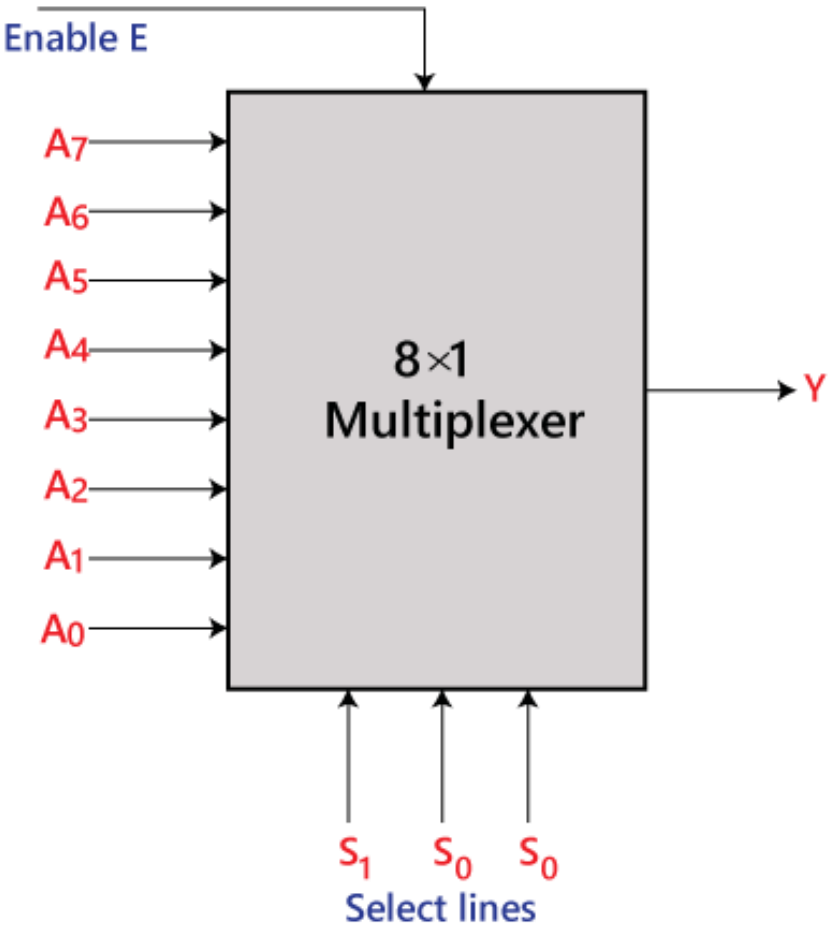


Fig: Block diagram of 8 x 1 MUX

Inputs								Selector			Output
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	S_2	S_1	S_0	Y
-	-	-	-	-	-	-	A_0	0	0	0	A_0
-	-	-	-	-	-	A_1	-	0	0	1	A_1
-	-	-	-	-	A_2	-	-	0	1	0	A_2
-	-	-	-	A_3	-	-	-	0	1	1	A_3
-	-	-	A_4	-	-	-	-	1	0	0	A_4
-	-	A_5	-	-	-	-	-	1	0	1	A_5
-	A_6	-	-	-	-	-	-	1	1	0	A_6
A_7	-	-	-	-	-	-	-	1	1	1	A_7

Table: Truth Table For 8x1 MUX

8 ×1 multiplexer using 4×1 multiplexer

- Can be implemented using lower order Multiplexer.
- To implement the 8×1 multiplexer, we need two 4×1 multiplexers.
- The 4×1 multiplexer has 2 selection lines, 4 inputs, and 1 output.
- For getting 8 data inputs, we need two 4×1 multiplexers.
- The two 4×1 multiplexer produces one/one output. So, in order to get the final output, we need a OR gate/2×1 multiplexer.

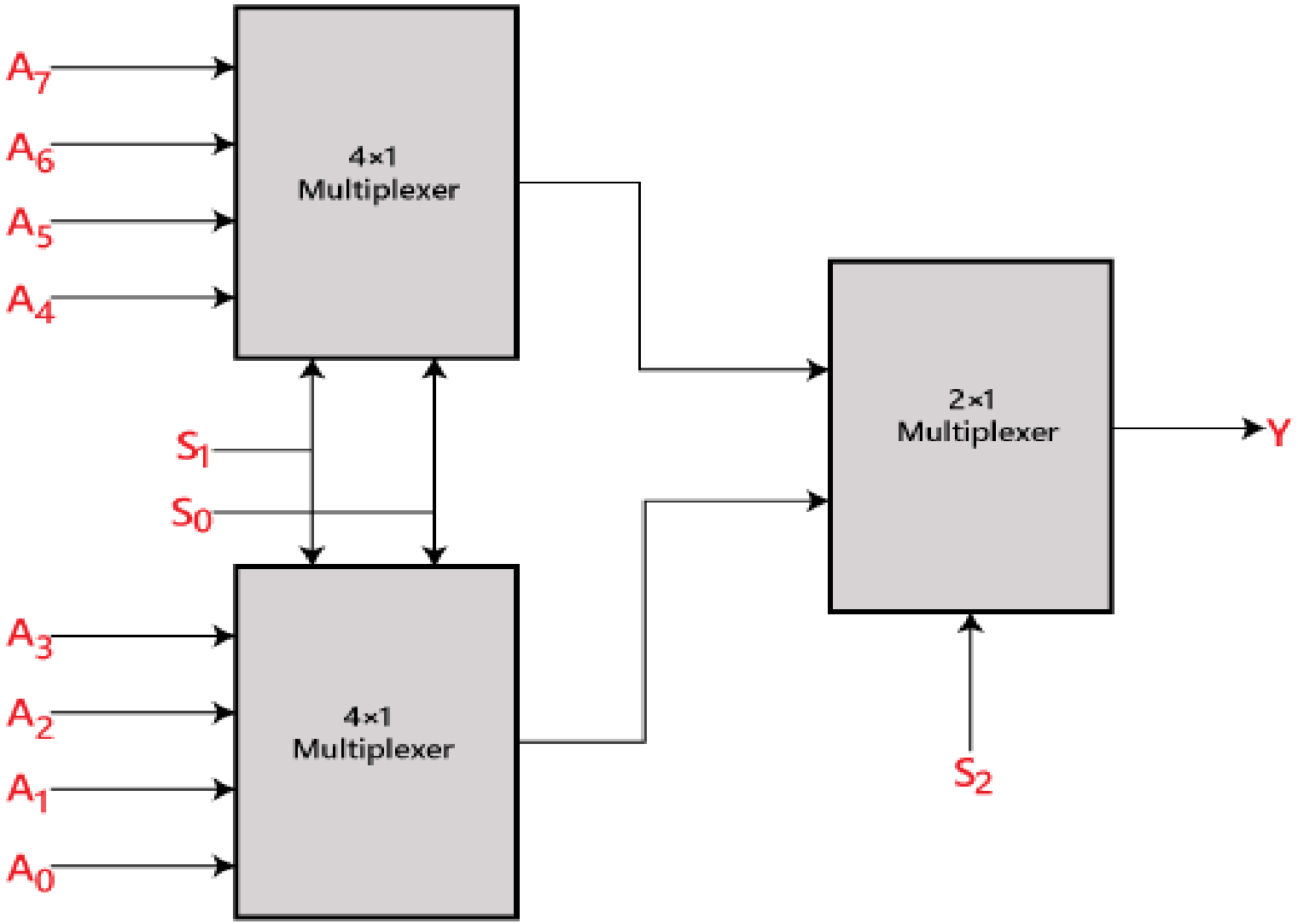


Fig: Block diagram of 8 x 1 MUX using 4x1 MUX

Inputs								Selector			Output
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	S_2	S_1	S_0	Y
-	-	-	-	-	-	-	A_0	0	0	0	A_0
-	-	-	-	-	-	A_1	-	0	0	1	A_1
-	-	-	-	-	A_2	-	-	0	1	0	A_2
-	-	-	-	A_3	-	-	-	0	1	1	A_3
-	-	-	A_4	-	-	-	-	1	0	0	A_4
-	-	A_5	-	-	-	-	-	1	0	1	A_5
-	A_6	-	-	-	-	-	-	1	1	0	A_6
A_7	-	-	-	-	-	-	-	1	1	1	A_7

Table: Truth Table For 8x1 MUX using 4x1

Case (a): *When S_2=0 the Lower 4x1 MUX will operate as 2x1 mux selects the lower one and Upper one will be disabled at that time.*

Case (b): *When S_2=1 the Upper 4x1 MUX will operate and Lower one will be disabled.*

Applications

- Communication system
- Telephone network
- Computer memory
- Satellite communication
- Television channel selection

Advantages:

- Reduces the no. of wires
- Reduces circuit complexity
- Easy selection of source

De-multiplexers/data distributers

- Single input
- 2^n possible output lines
- Selection is controlled by selection line ***S***.
- Just reverse process of Multiplexer
- ***A decoder with enabled input can be function as De-MUX.***

1×2 De-multiplexer

- Only two outputs, i.e., Y_0, *and* Y_1
- 1 selection lines, i.e., S_0 and
- Single input, i.e., ***A***.

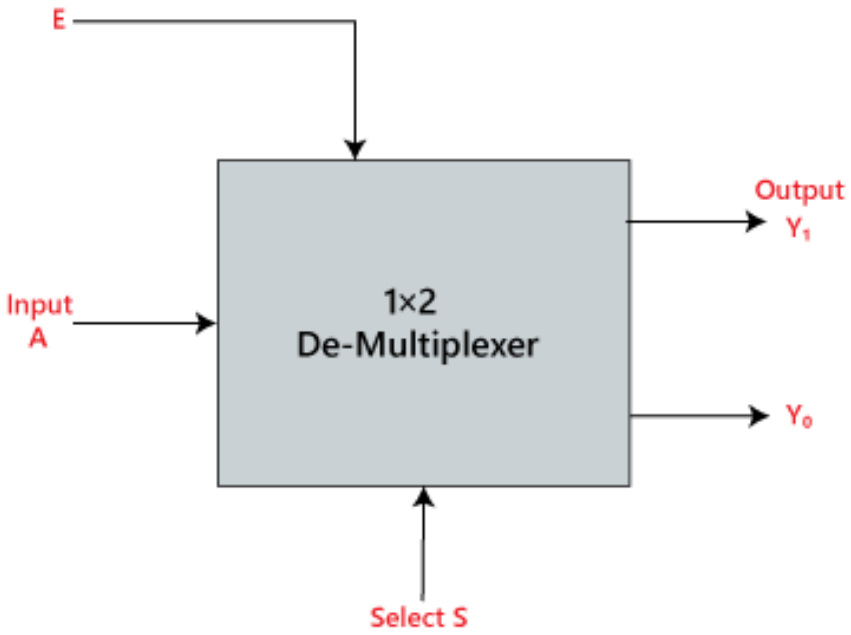


Fig: Block diagram of 1x 2 De-MUX

INPUTS	Output	
S_0	Y_1	Y_0
0	0	A
1	A	0

Table: Truth Table For 1x2 De-MUX

The logical expression of the term Y is as follows:

$$Y_0 = S_0' . A$$

$$Y_1 = S_0 . A$$

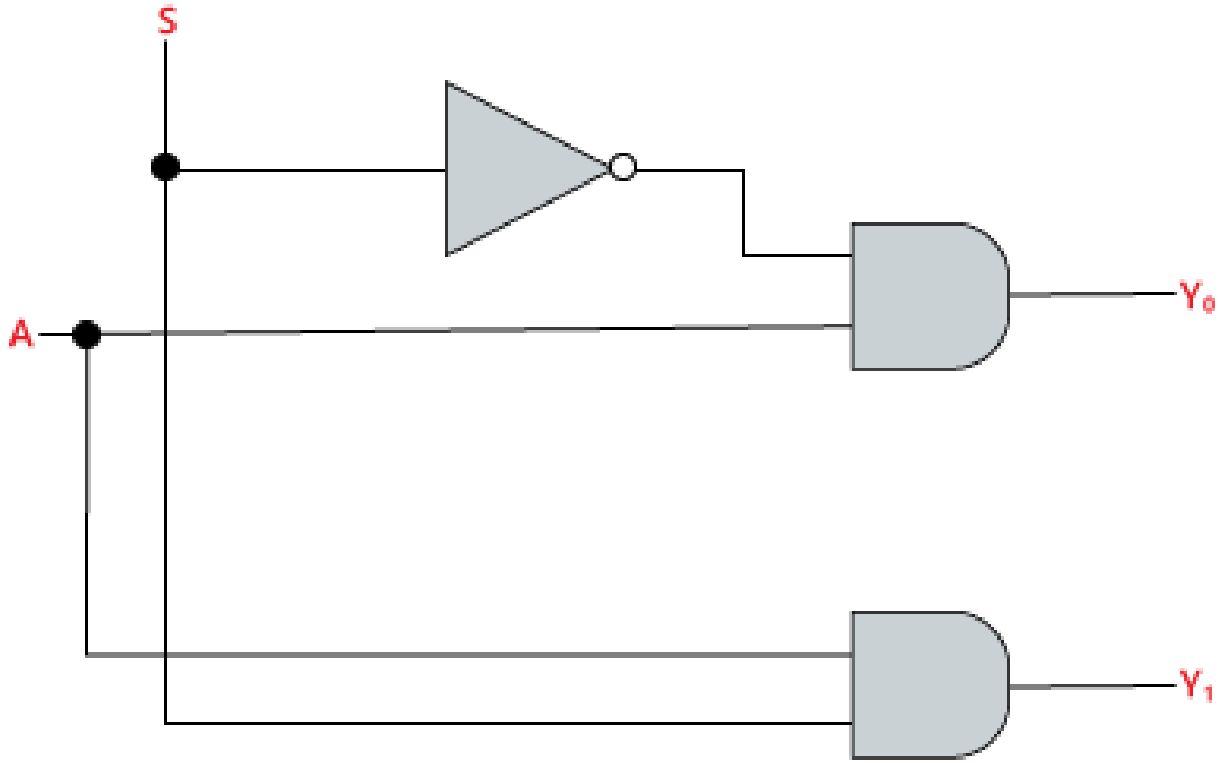


Fig: Circuit diagram of 1x 2 De-MUX

1×4 De-multiplexer:

- Single input, ***i.e., A***
- Four outputs, ***i.e., Y_0, Y_1, Y_2, and Y_3***
- 2 selection lines, ***i.e., S_0 and S_1***

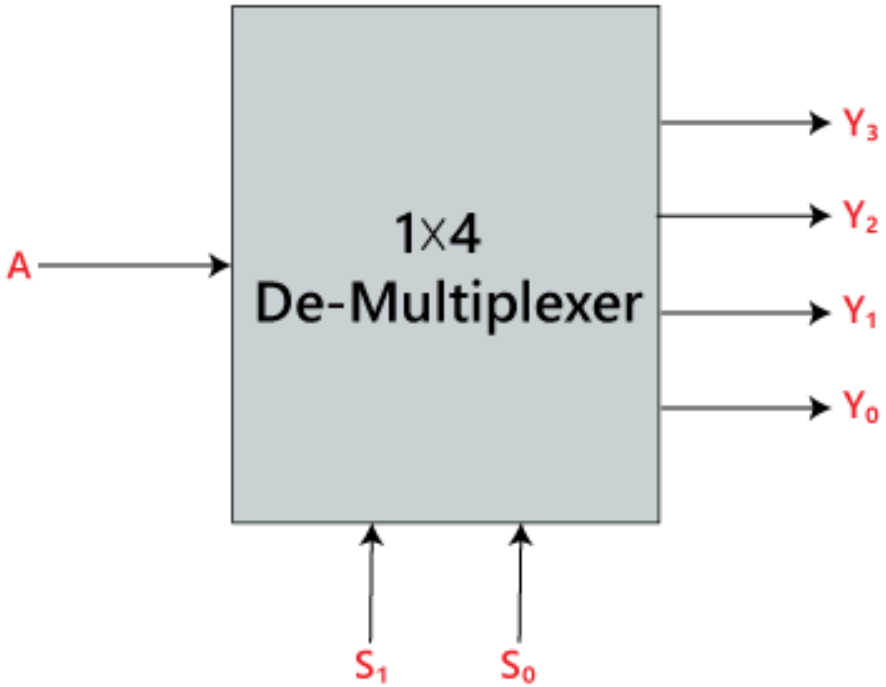


Fig: Block diagram of 1x 4 De-MUX

INPUTS		Output			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	A
0	1	0	0	A	0
1	0	0	A	0	0
1	1	A	0	0	0

Table: Truth Table For 1x4 De-MUX

The logical expression of the term Y is as follows:

$$Y_0 = S_1' S_0' A$$
$$y_1 = S_1' S_0 A$$
$$y_2 = S_1 S_0' A$$
$$y_3 = S_1 S_0 A$$

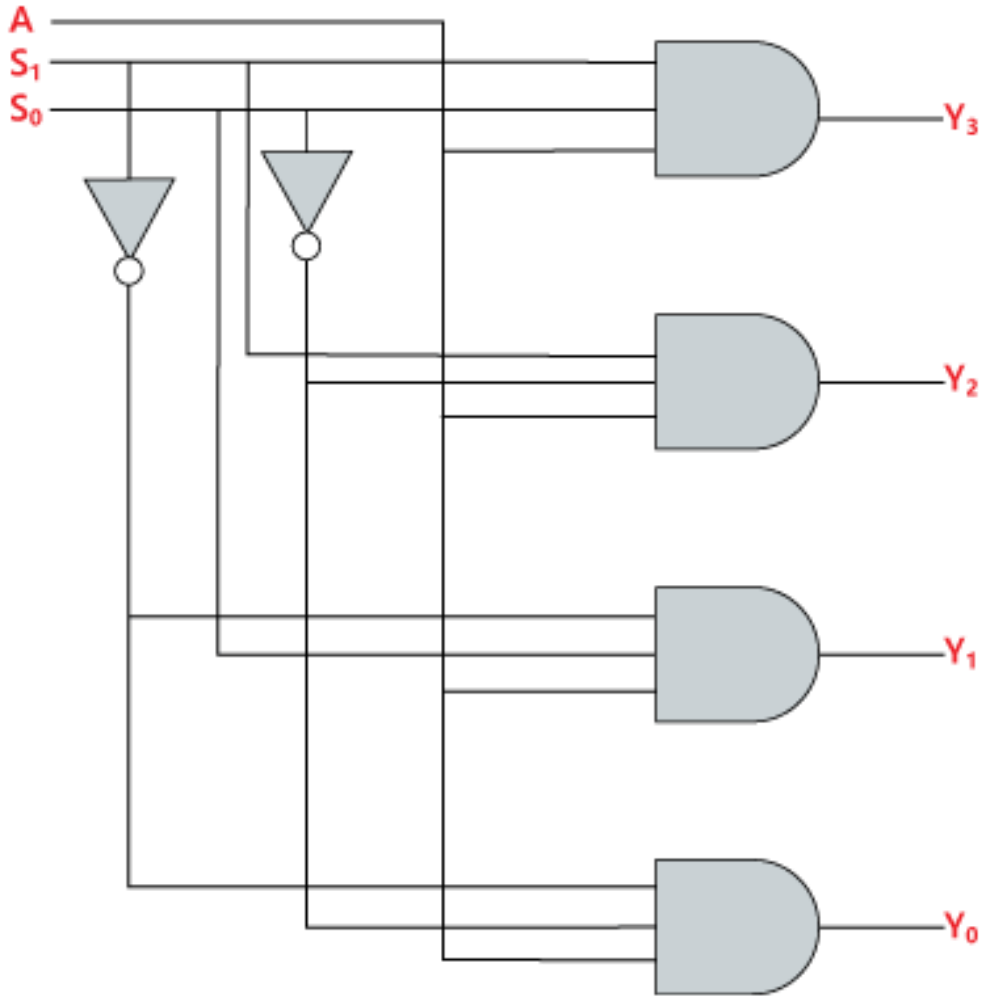


Fig: Circuit diagram of 1x 4 De-MUX

1×8 De-multiplexer

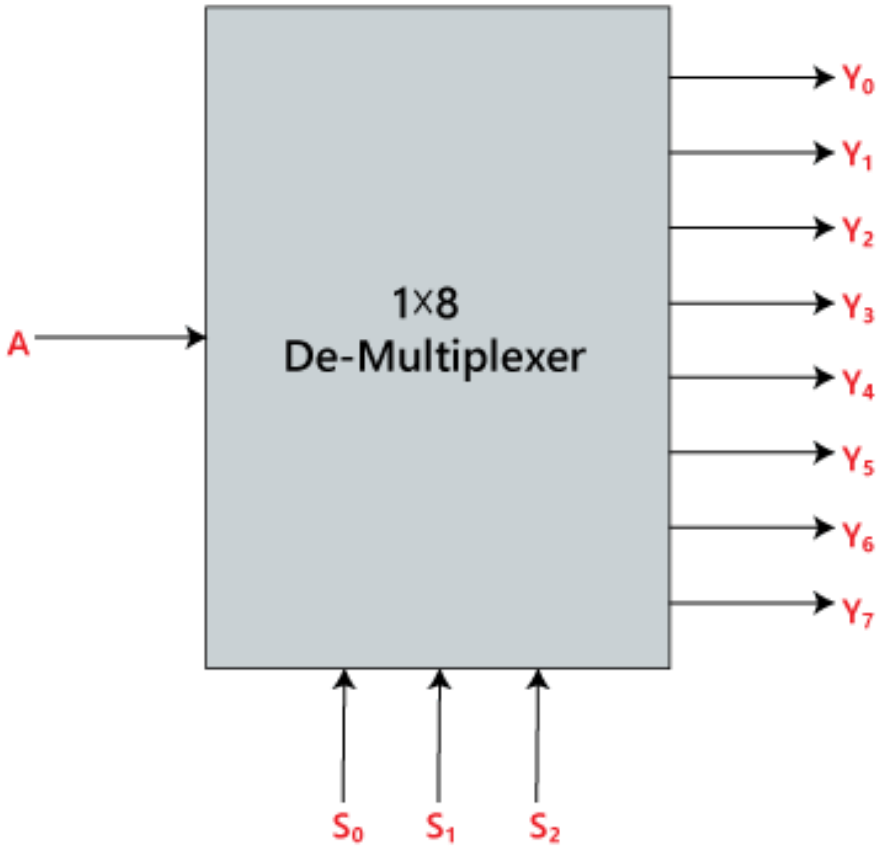


Fig: Block diagram of 1x 8 De-MUX

The logical expression of the term Y is as follows:

$Y_0 = S_0'.S_1'.S_2'.A$

$Y_1 = S_0.S_1'.S_2'.A$

$Y_2 = S_0'.S_1.S_2'.A$

$Y_3 = S_0.S_1.S_2'.A$

$Y_4 = S_0'.S_1'.S_2\,A$

$Y_5 = S_0.S_1'.S_2\,A$

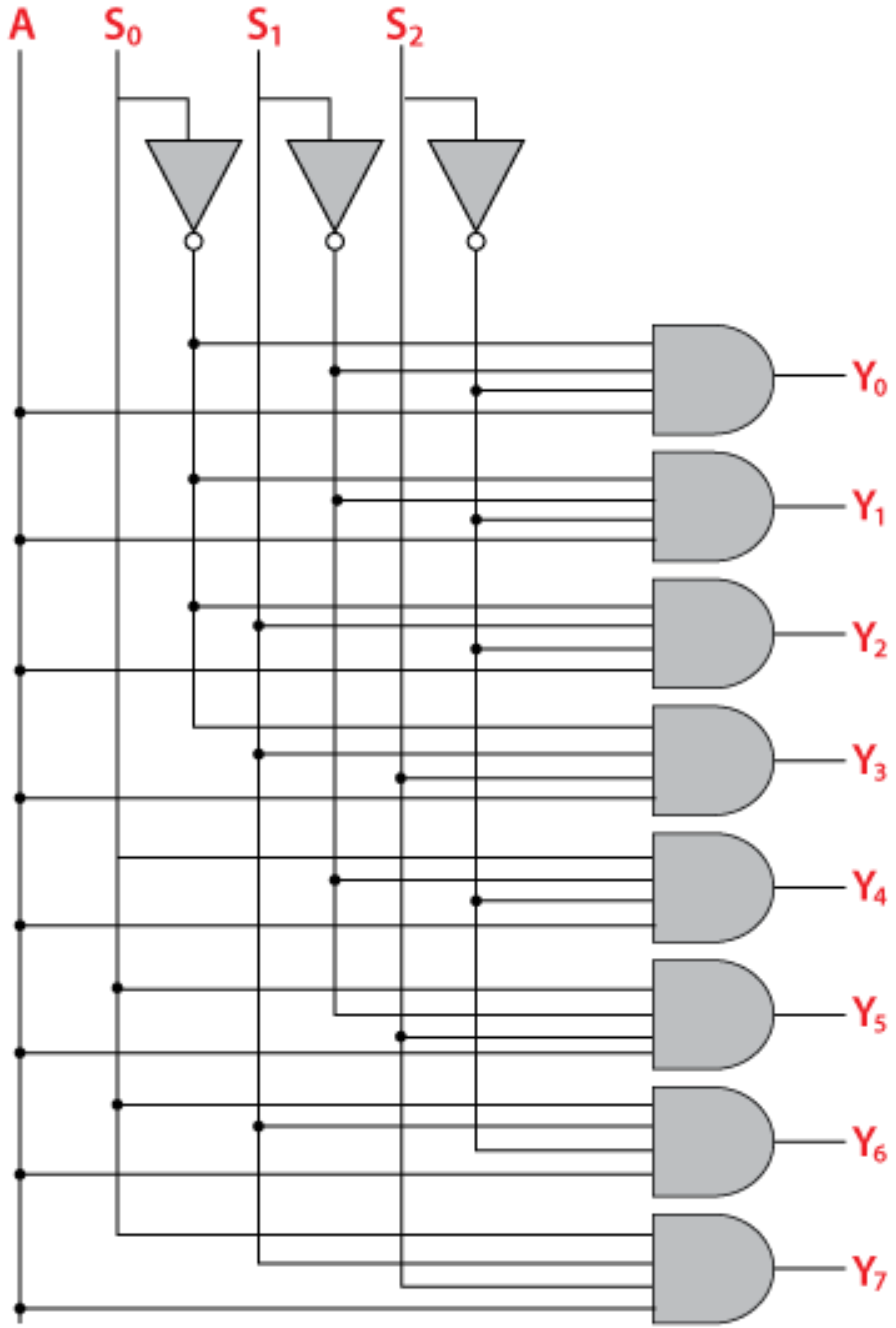
$Y_6 = S_0'.S_1.S_2\,A$

$Y_7 = S_0.S_1.S_3.A$

INPUTS			Output							
S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	A
0	0	1	0	0	0	0	0	0	A	0
0	1	0	0	0	0	0	0	A	0	0
0	1	1	0	0	0	0	A	0	0	0
1	0	0	0	0	0	A	0	0	0	0
1	0	1	0	0	A	0	0	0	0	0
1	1	0	0	A	0	0	0	0	0	0
1	1	1	A	0	0	0	0	0	0	0

Table: Truth Table For 1x8 De-MUX

Fig: Circuit diagram of 1x 8 De-MUX



1×8 De-multiplexer using 1×4 and 1×2 de-multiplexer

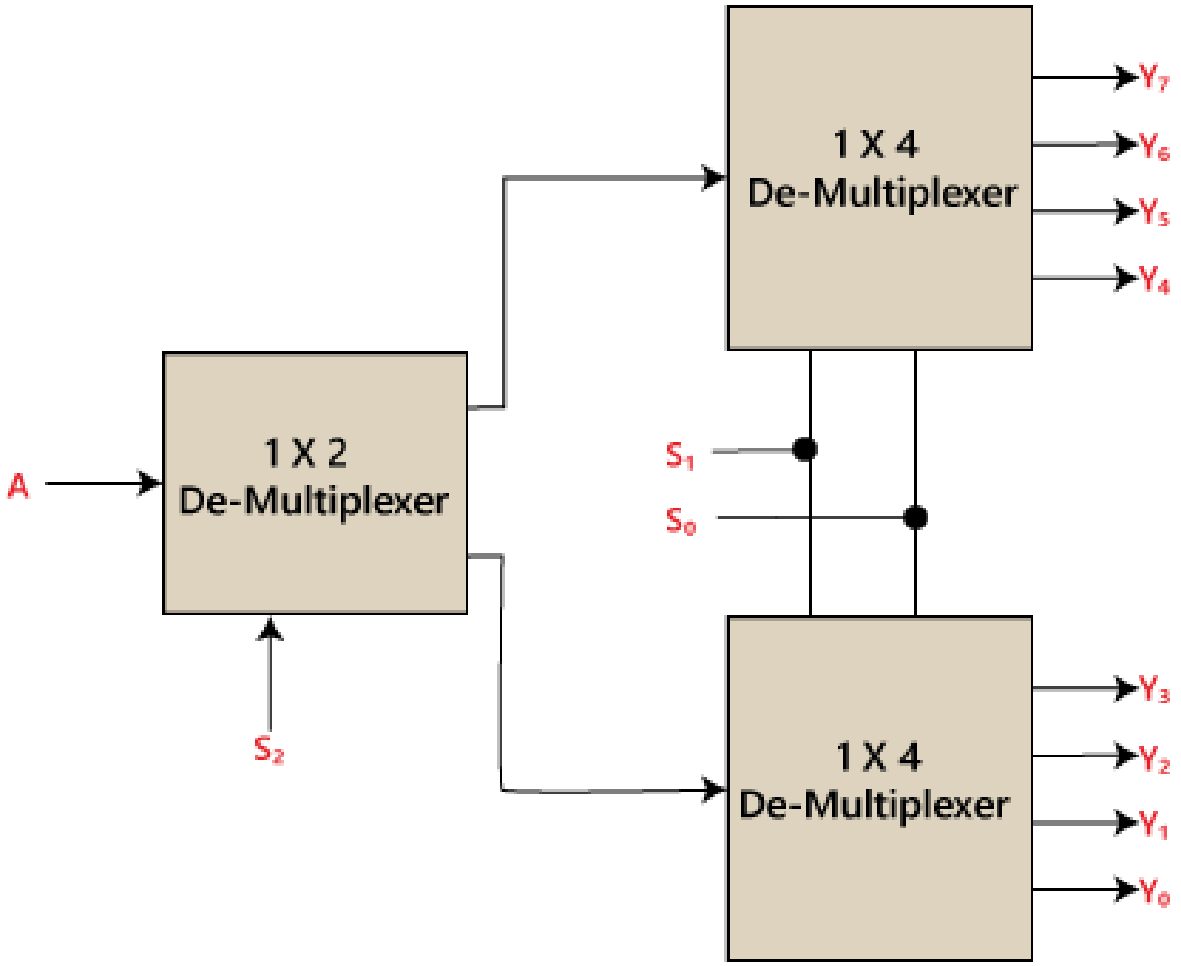


Fig: Block diagram of 1x 8 De-MUX using 1x4 De-MUX