Department of Computer Science & Engineering (CSE)

# Course Title: Digital Logic Design

Shahriar Rahman Khan
Lecturer
Dept. of CSE, MIST

Course Code: CSE 103
Credit Hr: 3.00
Contact Hr: 3.00

# Overview

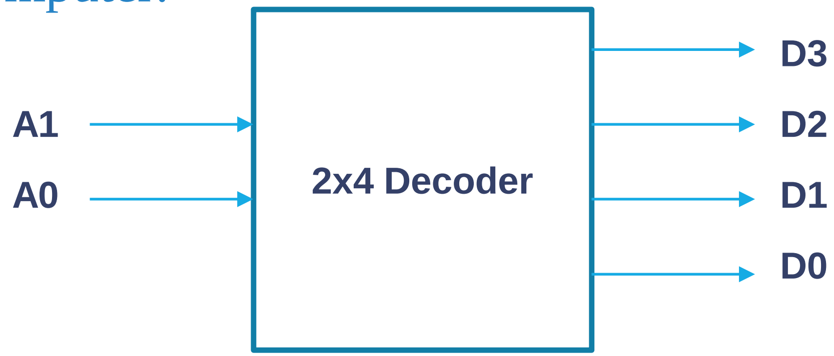
- What is MSI and PLD?
- Binary parallel Adder
- Binary Adder-Subtractors
- Carry Propagation
- BCD Adder
- Magnitude Comparator
- Decoders and De-multiplexers
- Encoders and Multiplexers
- Priority Encoders

# Decoder

- We know, discrete quantities of information are represented in digital systems with binary codes.
- A binary code of n bits is capable of representing up to 2^n distinct elements of the coded information. Let you can represent 8 distinct elements of coded information using 3 bits as 2^3 = 8.
- A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n output lines.

# Decoder

- A computer is connected with 4 output devices, let's say printers. Now a computer operator will select a printer among the 4 printers. To do that, he has to give input to his computer.

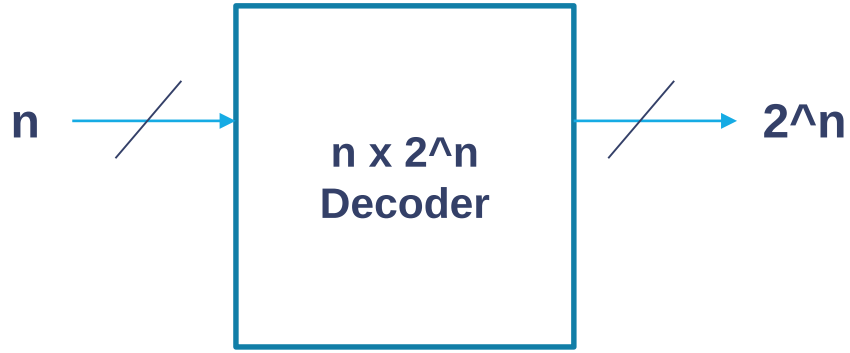



- How many bits required to select any 1 printer among the 4 printers?

| A1 | A0 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |

# Decoder

- A combinational circuit that produces 2^n output lines, in response of n input lines




- Configuration of a Decoder: No of input lines x No of output lines
- So, generalized configuration of a Decoder: If there are n input lines then, Decoder would be n x 2^n Decoder.
- If, the n bit decoded information has unused don't care combinations, then the decoder output will have less than 2^n output lines. Let, m = no of output lines, where m<=2^n.
- So, a decoder is also called n-to-m-line decoder. Like, 3-to-8-line decoder.

# Use of Decoder

- The purpose of a decoder is to generate $2^n$ (or fewer) minterms of n input variables.
- It is used to convert binary data to other codes. Example: binary to octal code conversion, BCD to decimal code conversion, binary to hexadecimal code conversion.
- Decoders are used to input data to a specified output line as is done in addressing core memory where input data is to be stored in a specified memory location.
- It may also be used for data distribution i.e. De-multiplexing.

# 3-to-8-line Decoder

- In the 3-to-8-line decoder circuit, the 3 inputs are decoded into 8 outputs, where each output representing 1 of the minterms of the 3 input variables.

- A particular application of this circuit is binary-to-octal conversion.
- The input variables may represent a binary number, where the outputs will then represent the 8 digits in the octal number system.

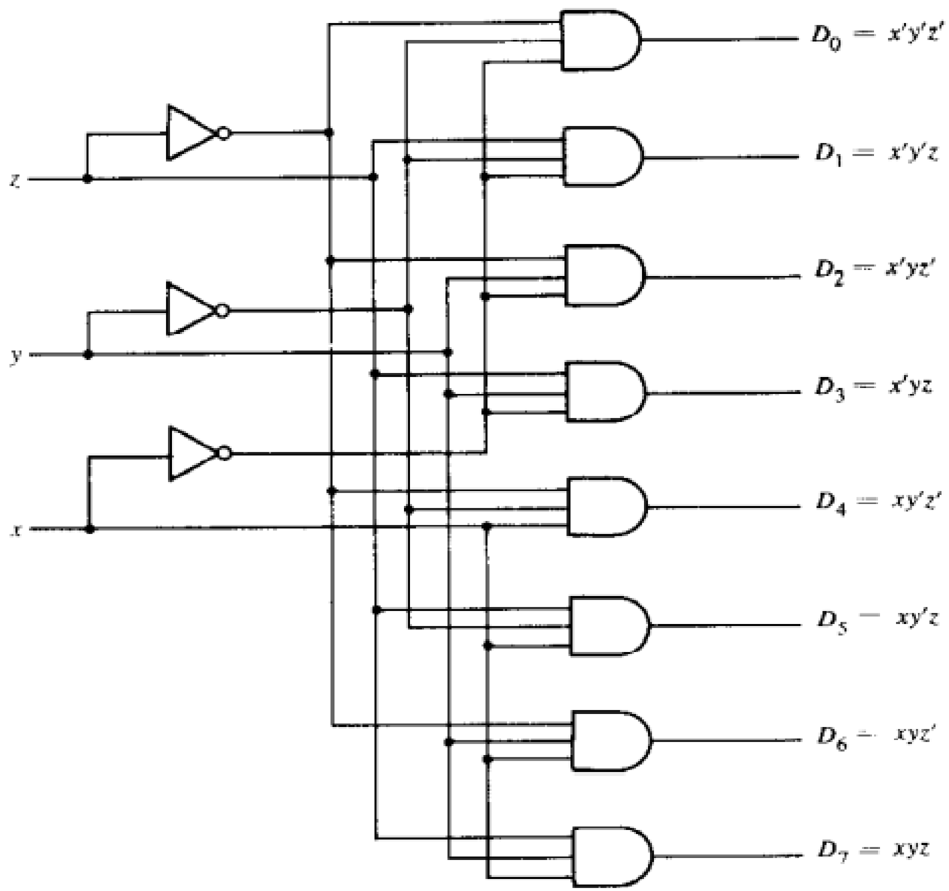
# 3-to-8-line Decoder

**Truth Table of a 3-to-8-Line Decoder**

| Inputs | | | Outputs | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $x$ | $y$ | $z$ | $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ |
| 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 |

- If you observe the truth table, you can see the output variables are mutually exclusive because only one output can be equal to 1 at any one time.
- We can represent any function using a decoder.

# 3-to-8-line Decoder



# BCD-to-Decimal Decoder

- In the BCD-to-Decimal code conversion, 4-to-10-line decoder circuit is used, the 4 inputs are decoded into 10 (0-9) outputs, where each output representing 1 of the minterms of the 4 input variables.

- The input variables may represent a binary number, where the outputs will then represent the 10 digits in the decimal number system.

# BCD-to-Decimal Decoder

| BCD | | | | Decimal | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $A$ | $B$ | $C$ | $D$ | $D_9$ | $D_8$ | $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
| 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 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 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 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# BCD-to-Decimal Decoder

| CD / AB | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| **00** | D0 | D1 | D3 | D2 |
| **01** | D4 | D5 | D7 | D6 |
| **11** | X | X | X | X |
| **10** | D8 | D9 | X | X |

Instead of drawing 10 k-maps, we are drawing only one and write each of the output variables.

- D0 = A'B'C'D'
- D1 = A'B'C'D
- D2 = B'CD'
- D3 = B'CD
- D4 = BC'D'
- D5 = BC'D
- D6 = BCD'
- D7 = BCD
- D8 = AD'
- D9 = AD

# BCD-to-Decimal Decoder

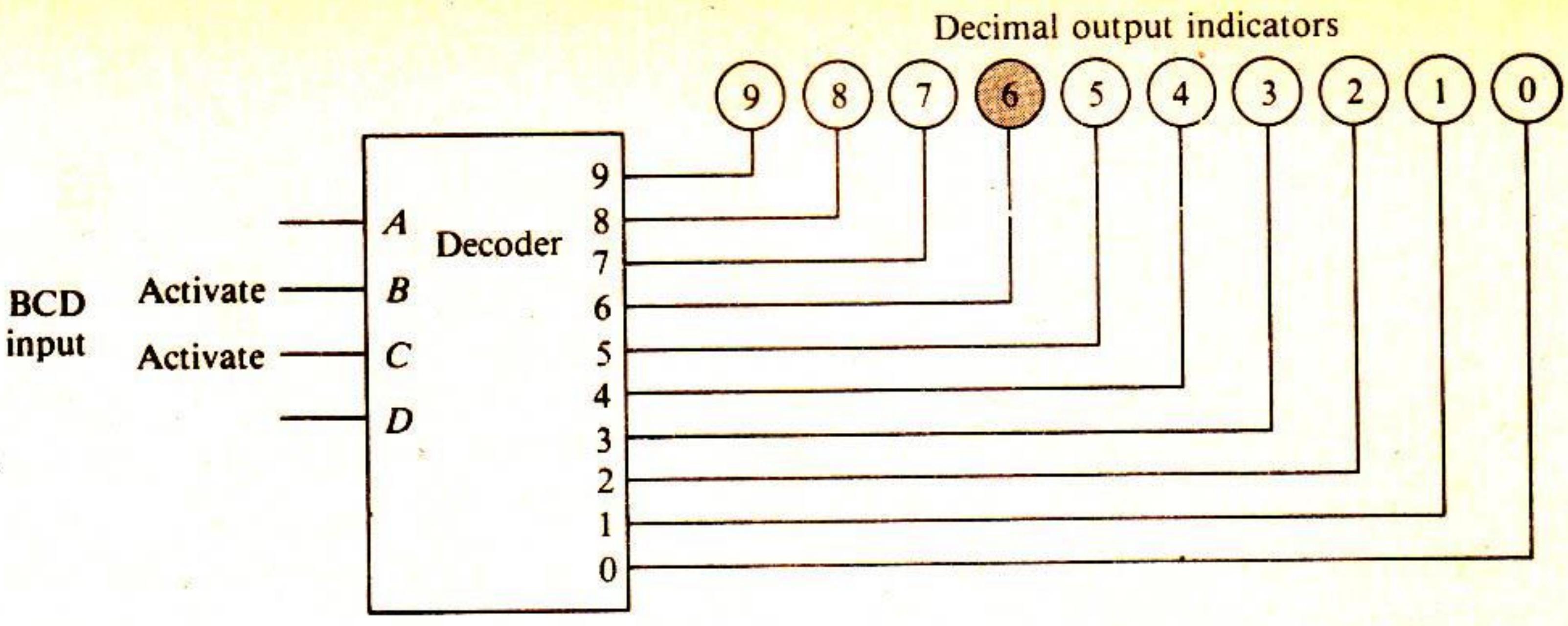



**Fig. 6-6** Logic symbol for a BCD-to-decimal decoder

# Combinational Logic Implementation using Decoder

- As we know, a decoder provides the 2^n minterms of n input variables.
- Also, we know, any Boolean function can be expressed in sum of minterms canonical form.
- So one can use a decoder to generate the minterms and an external OR gate to form the sum.
- The Boolean functions expressed in sum of minterms, can be obtained by forming the truth table or by expanding the functions to their sum of minterms.
- Let, $F(A,B,C) = AB' + BC + A'C'$

# Full-Adder using Decoder

You remember this combinational circuit named Full-Adder from chap 4. As this is a combinational circuit that adds three 1 bit binary digits, so there will be 3 input variables and 2 output variables.

| x | y | z | C | S |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 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 | 1 | 1 |

# Full-Adder using Decoder

| | | | |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 |

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

# Full-Adder using Decoder

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

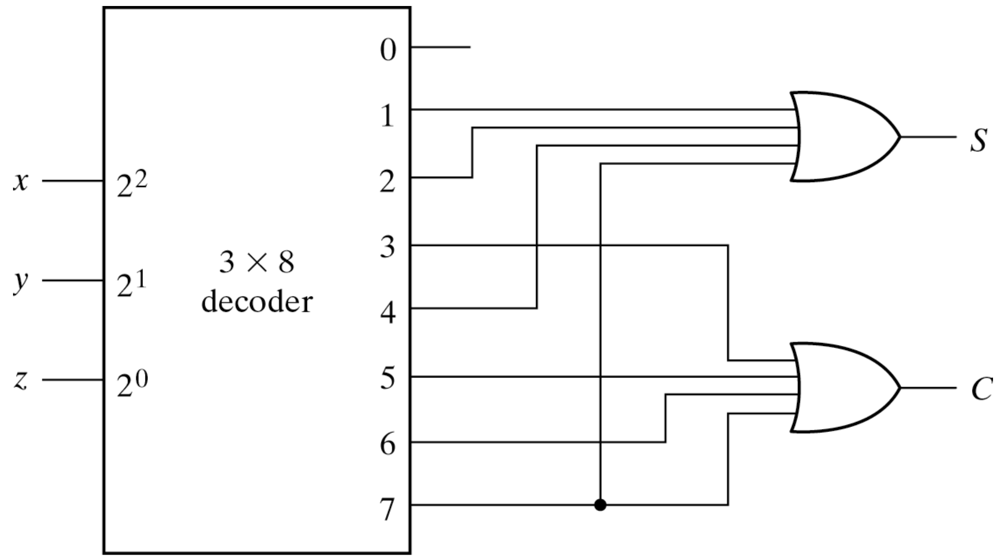



Fig. 4-21 Implementation of a Full Adder with a Decoder

# Combinational Logic Implementation using Decoder

- Now, if there is a function with a long list of minterms, then it will be required an OR gate with a large number of inputs.
- A function F having a list of k minterms can be expressed in it's complemented form F'.

- Let $F = \Sigma(0,1,5,6,7)$
- $F = m_0 + m_1 + m_5 + m_6 + m_7$
- $F' = m_2 + m_3 + m_4$
- $F = (F')'$
- $F = (m_2 + m_3 + m_4)'$
- $NOR(m_2, m_3, m_4)$

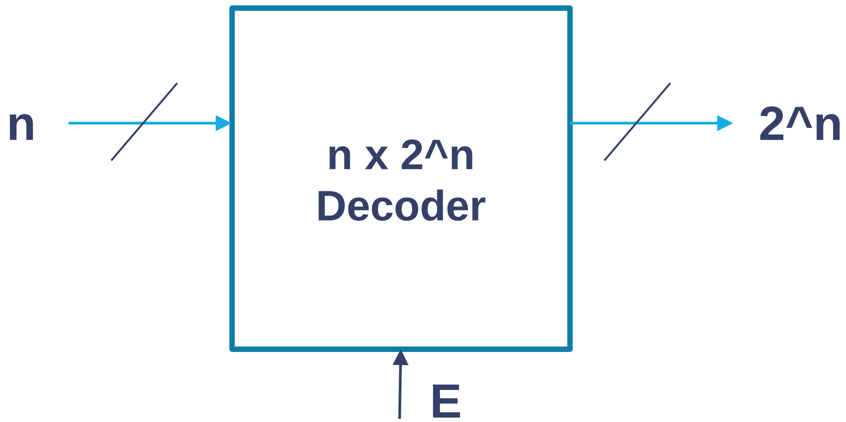
- Hence, the decoder method can be used

To implement any combinational circuit.

| x | y | z | F |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

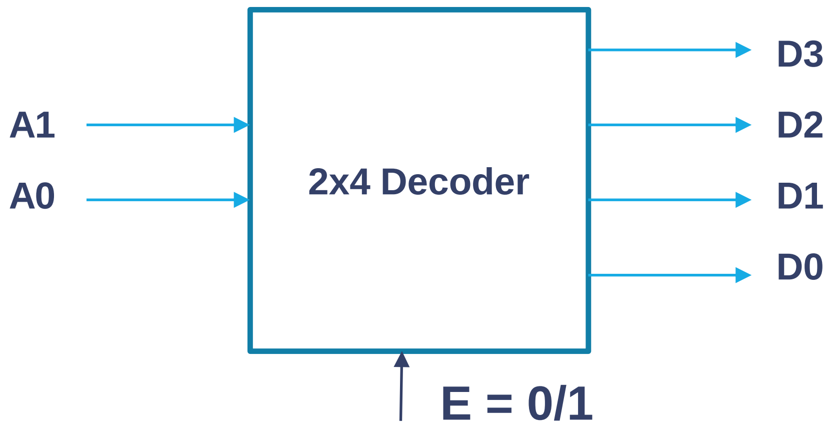
# Decoder with Enable Pin

- Now, we introduce Enable pin E in a decoder. On the value of E, the decoder activates.
- There can be two types of activation: 1-activation and 0-activation.
- In 1-activation, the decoder activates, when E=1 and decoder will be disabled, when E=0.
- In 0-activation, the decoder activates, when E=0 and decoder will be disabled, when E=1.



# Decoder with 1-activation

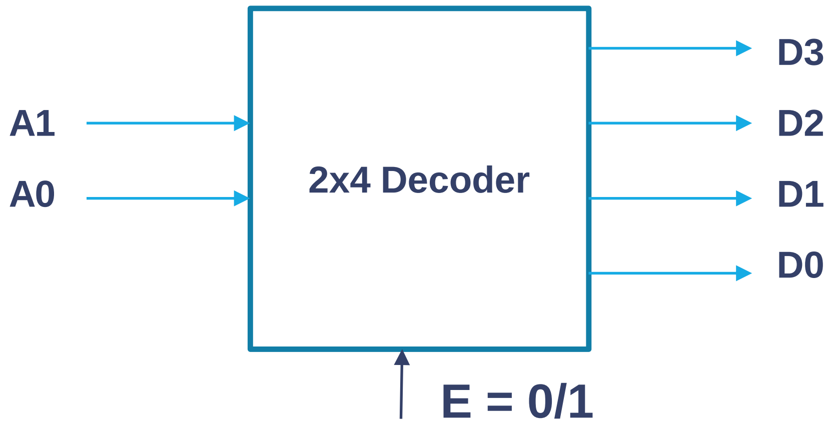
- In 1-activation, the decoder activates, when E=1 and decoder will be disabled, when E=0.




| E | A1 | A0 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|
| 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 |

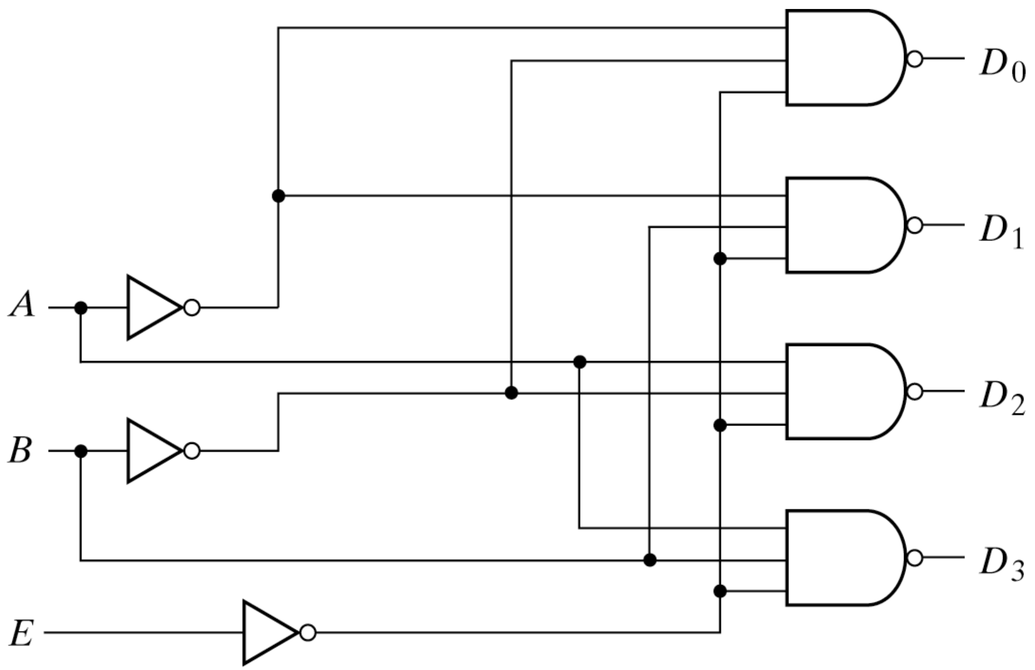
# Decoder with 0-activation

- In 0-activation, the decoder activates, when E=0 and decoder will be disabled, when E=1. 0-activation decoder is used much more than 1-activation. Because, you can implement the outputs using NAND gates and NAND gate is universal.




| E | A1 | A0 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|
| 1 | X | X | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 |

# Decoder with 0-activation




(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 |

(b) Truth table

Fig. 4-19 2-to-4-Line Decoder with Enable Input

# Decoder with 0-activation

| E \ A0A1 | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

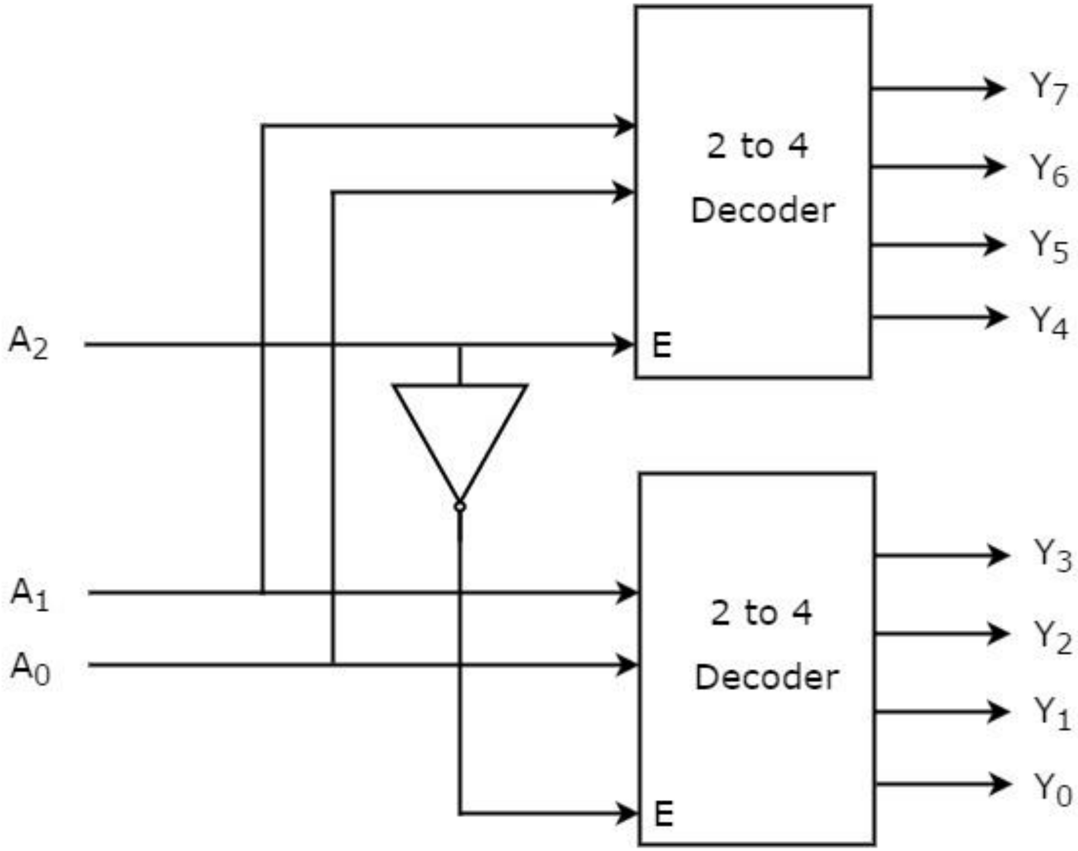
- D0’ = E’A0’A1’
- D0 = (E’A0’A1’)’
- Same for D1, D2, and D3
- This can be implemented using NAND gate, that’s why in the logic diagram, NAND gate is used. Also, as NAND gate is universal, that’s why 0-activation is much more used than 1-activation

# 3x8 Decoder using 2x4 Decoder

- Now to implement a 3x8 decoder using 2x4 decoder, we can use the enable pin E.

| | | | Decoder-2 | | | | Decoder-1 | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **A2 (E)** | **A1** | **A0** | **D7** | **D6** | **D5** | **D4** | **D3** | **D2** | **D1** | **D0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# 3x8 Decoder using 2x4 Decoder

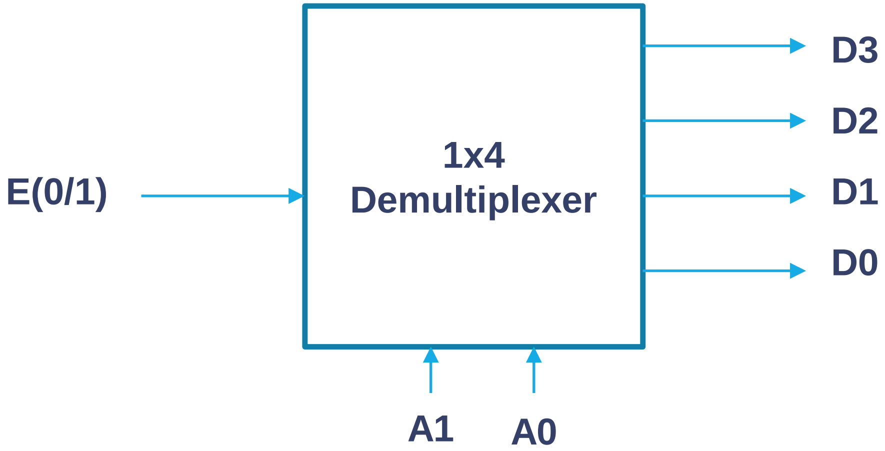


# De-multiplexer

- Now, a decoder with an enable input can function as a de-multiplexer.
- A de-multiplexer is a circuit that receives information on a single line and transmits this information on one of $2^n$ possible output lines.
- The selection of a specific output line is controlled by the n selection lines.

# De-multiplexer

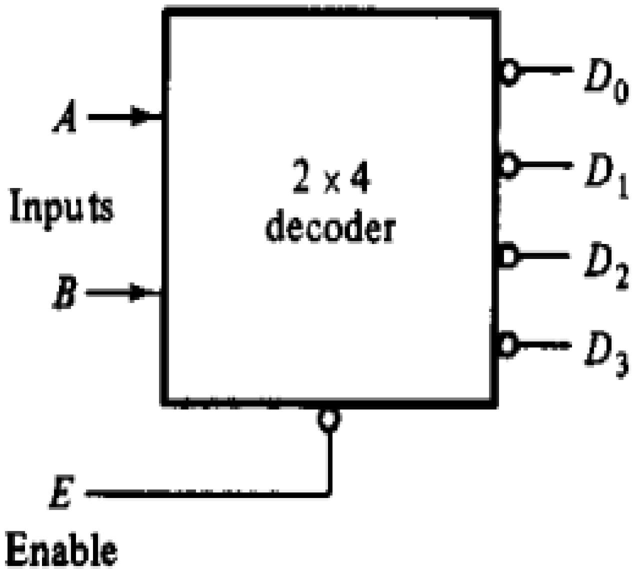
- Remember the example, A computer is connected with 4 output devices, let's say printers. Now a computer operator will select a printer among the 4 printers. To do that, he has to give input to his computer. So far we have done this using decoder.
- So, we have selected the printer, now we need to print a document. How to pass the document or data to the printer? We do this using data line E.
- So we can say that, a decoder with an enable input can function as a de-multiplexer. This is also called decoder/de-multiplexer.



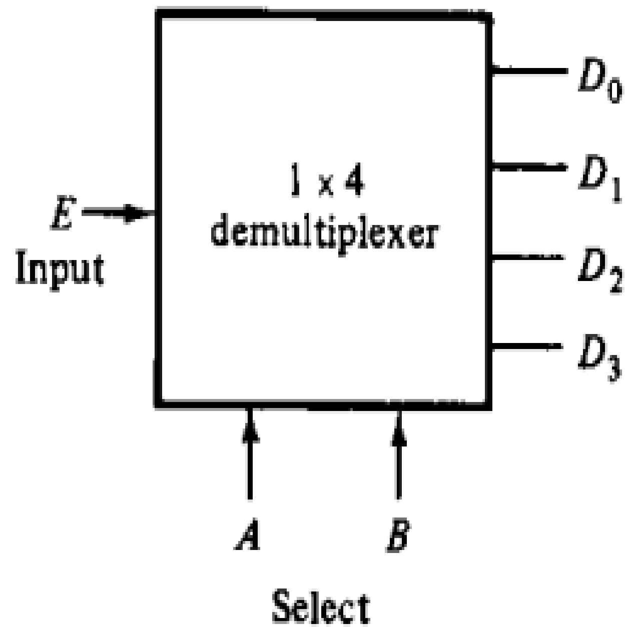
# Decoder as De-multiplexer

- So, a decoder with an enable input can function as a de-multiplexer.
- A de-multiplexer is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines.
- The selection of a specific output line is controlled by the n selection lines.
- Everything is like decoder. When you select A1A0=00, then D0 printer becomes activated and the data E will pass to the output and D0 printer will print it.
- When you select A1A0=10, then D2 printer becomes activated and the data E will pass to the output and D2 printer will print it.

# Decoder as De-multiplexer




(a) Decoder with enable




(b) Demultiplexer

# De-multiplexer

| E | A1 | A0 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 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 |

| A1 | A0 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | E |
| 0 | 1 | 0 | 0 | E | 0 |
| 1 | 0 | 0 | E | 0 | 0 |
| 1 | 1 | E | 0 | 0 | 0 |

# De-multiplexer

| A1 | A0 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | E |
| 0 | 1 | 0 | 0 | E | 0 |
| 1 | 0 | 0 | E | 0 | 0 |
| 1 | 1 | E | 0 | 0 | 0 |

Now, in the truth table have we ever seen that output is a variable?
No!
So, what we will do is, we will write the functions like this:

- D0 = A1’A0’E
- Here, D0 will be 1, when E=1. We are considering only that case, because always remember, a Boolean function will give output 1.

- D1 = A1’A0E
- D2 = A1A0’E
- D3 = A1A0E

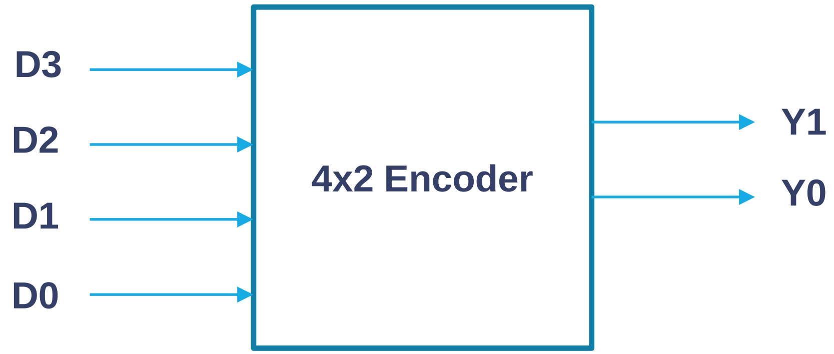
# De-multiplexer

- So, if no of outputs, $n=2^m$
- Then, m=no of selection lines.
- Here, n=4, so m=2

- Types: 1:2 DEMUX, 1:4 DEMUX, 1:8 MUX, 1:16 DEMUX, 1:32 DEMUX

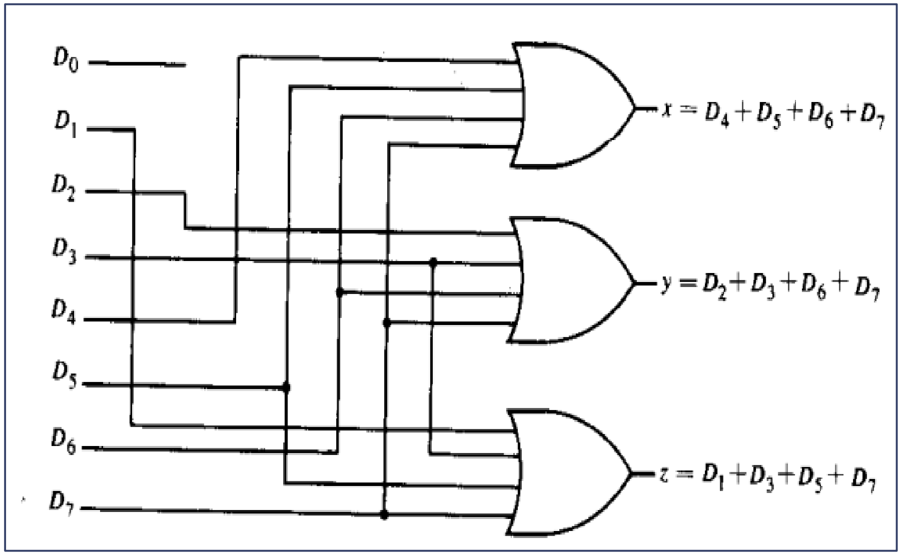
# Encoder

- An encoder is a digital circuit that performs the inverse operation of a decoder.
- It has 2^n (or fewer) input lines and n output lines.
- It generates binary code corresponding to the input lines.
- Example, Octal-to-Binary code converter.

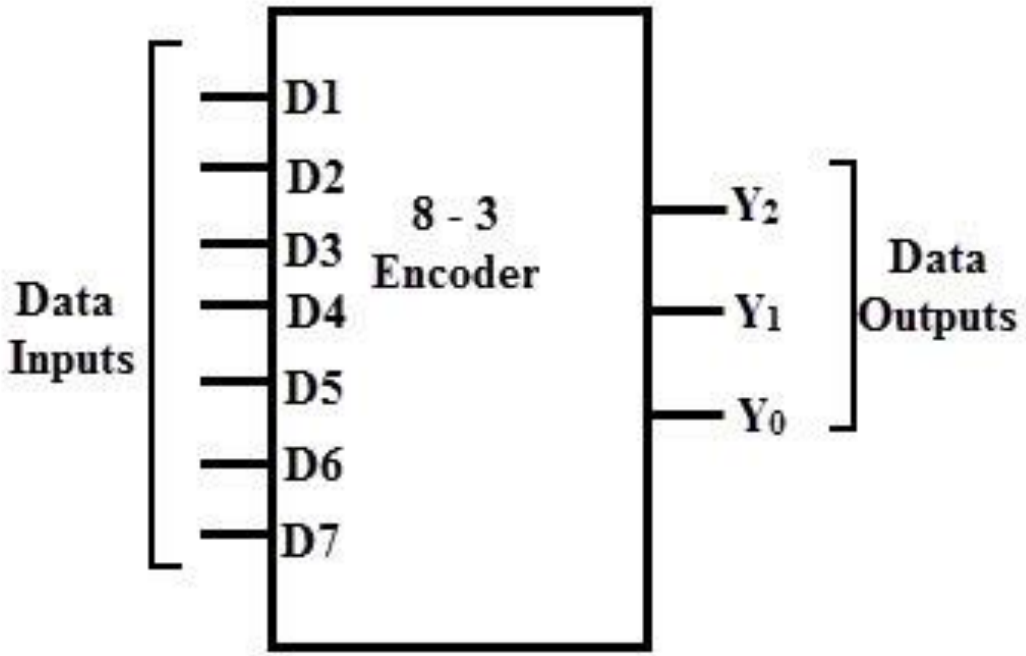


# Octal-to-Binary Encoder

| Inputs | | | | | | | | Outputs | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ | $x$ | $y$ | $z$ |
| 1 | 0 | 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 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

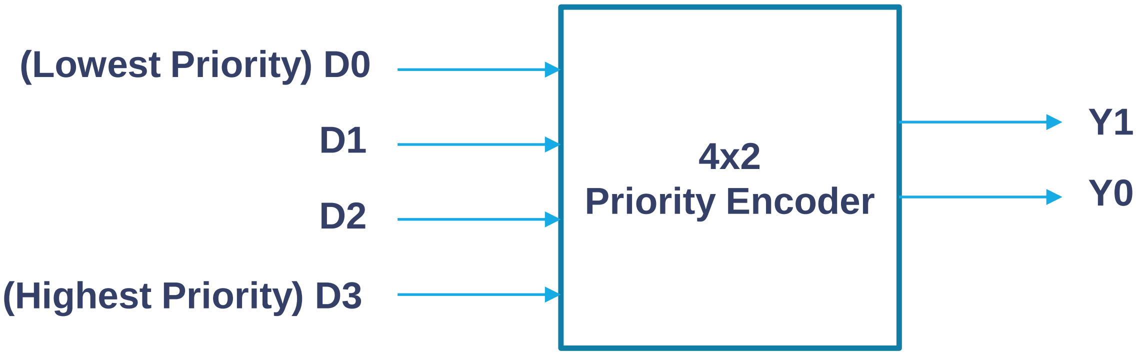


# Octal-to-Binary Encoder



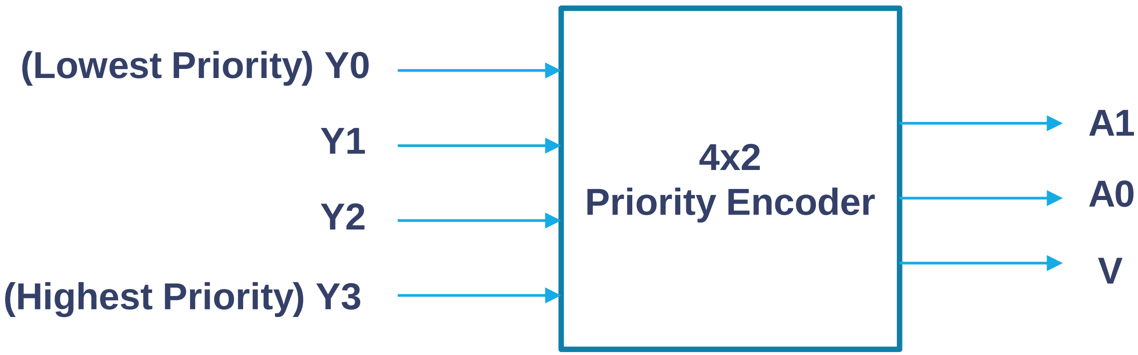
# Priority Encoder

- A priority encoder is a digital circuit that includes the priority function.
- The operation of the priority encoder is such that, if two or more inputs are equal to 1 at the same time, then the input having the highest priority will take precedence.

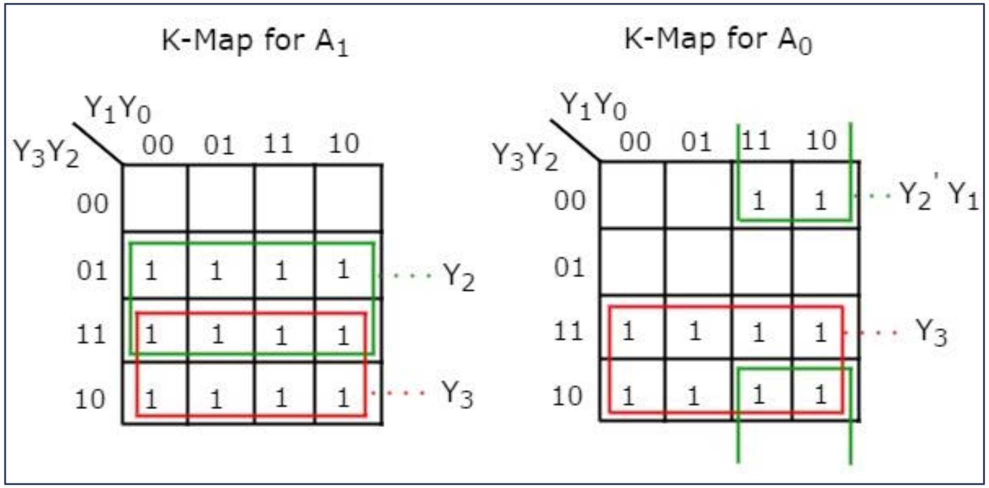



| D3 | D2 | D1 | D0 | A1 | A0 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | X | X |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | X | 0 | 1 |
| 0 | 1 | X | X | 1 | 0 |
| 1 | X | X | X | 1 | 1 |

# Priority Encoder

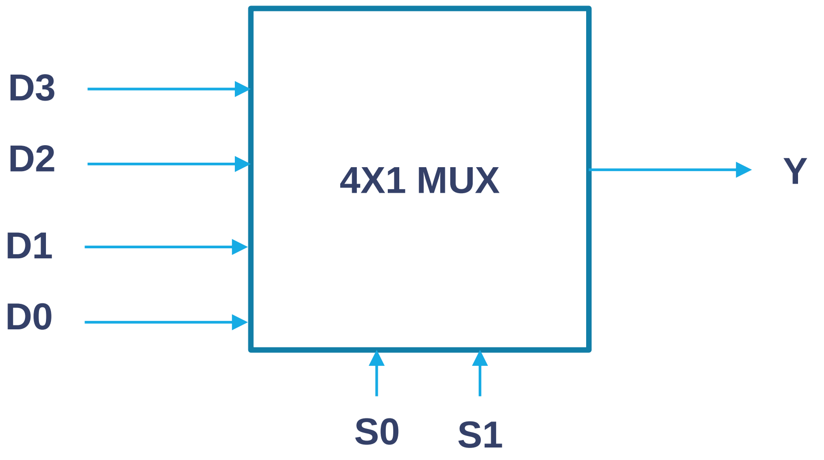



| Y3 | Y2 | Y1 | Y0 | A1 | A0 | V |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | X | X | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | X | 0 | 1 | 1 |
| 0 | 1 | X | X | 1 | 0 | 1 |
| 1 | X | X | X | 1 | 1 | 1 |

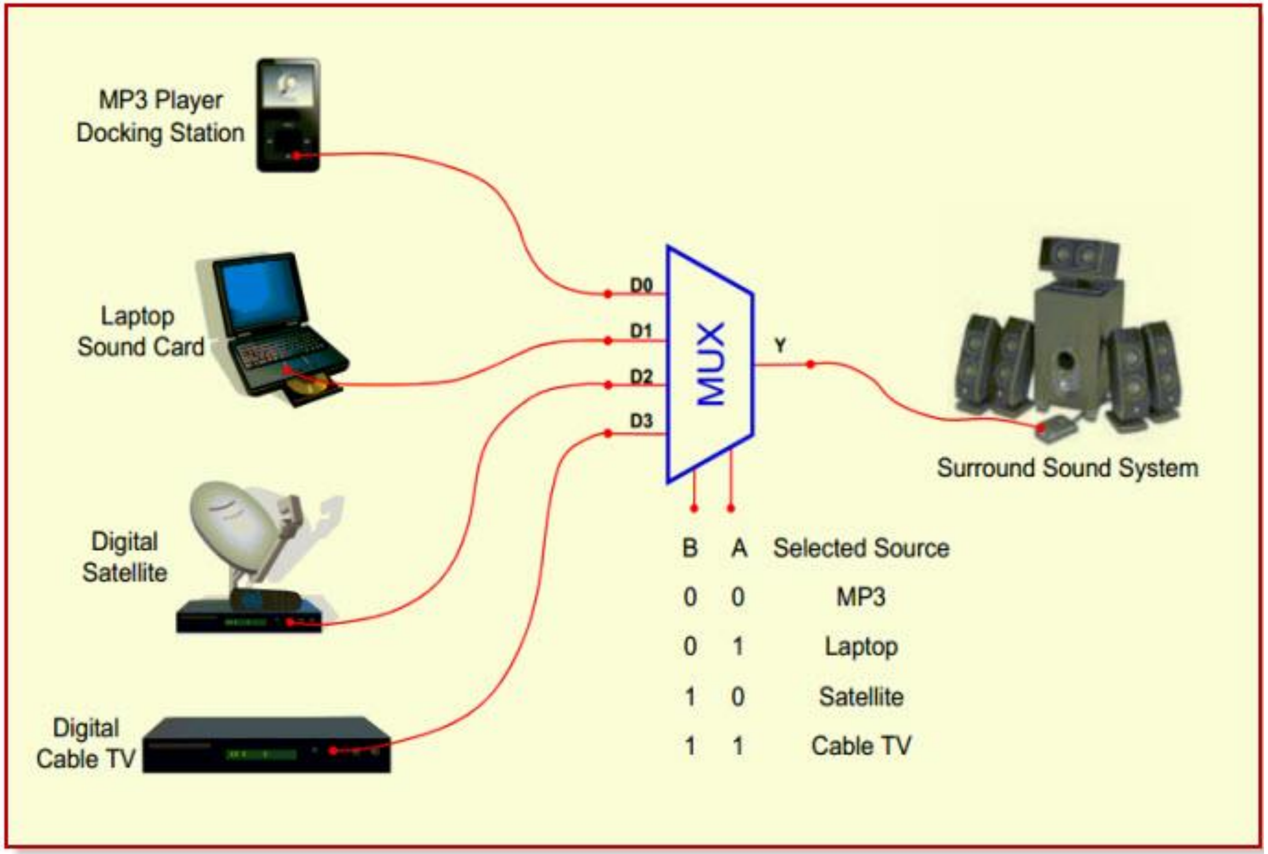


# Multiplexer

- Multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to o/p line. It's also called MUX.
- It is simply a Data Selector.
- How the data is selected?
- There are two selector variables, let S0 and S1.
- MUX will always have 1 output line Y.
- This is a 4x1 MUX.
- It can also be called as 4:1 MUX.
- So, if no of inputs, n=2^m
- Then, m=no of selector variables.
- Here, n=4, so m=2
- Types: 2:1 MUX, 4:1 MUX,
- 8:1 MUX, 16:1 MUX, 32:1 MUX

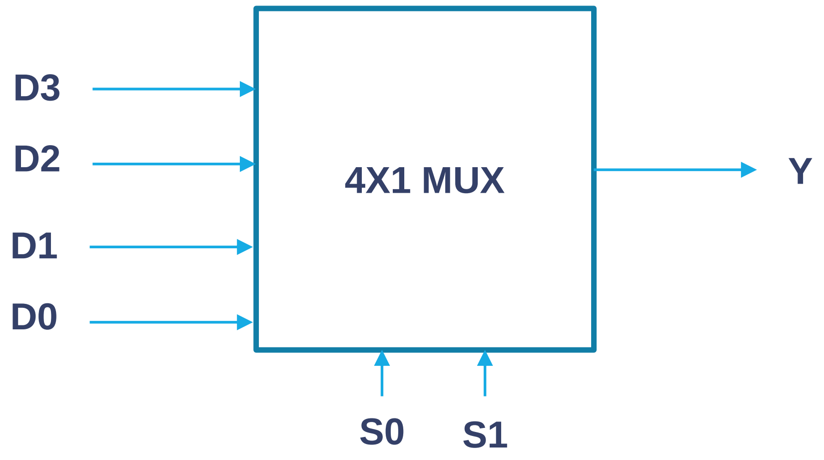


# Multiplexer



# Advantages of Multiplexer

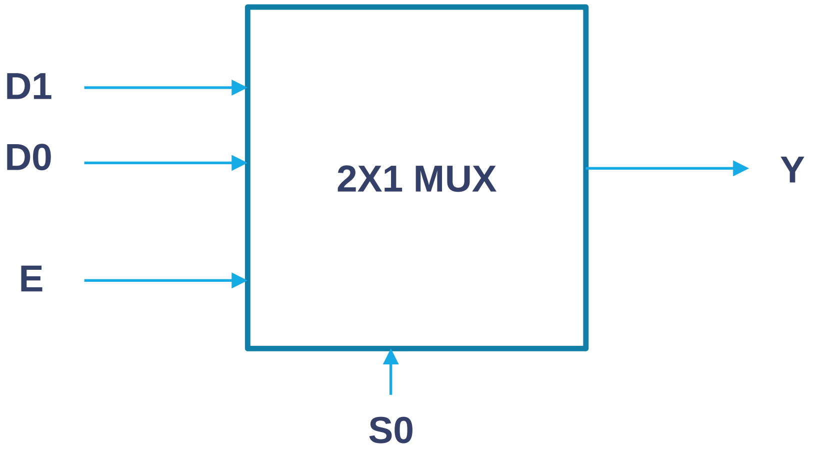
- Multiplexer is a MSI circuit.
- No of wires are reduced as no of gates are reduced. You can simply connect a no of gates by using a single MUX.
- Reduces circuit complexity and cost as no of gates and wires are reduced.
- Implementation of various circuits using MUX.



# 2x1 Multiplexer

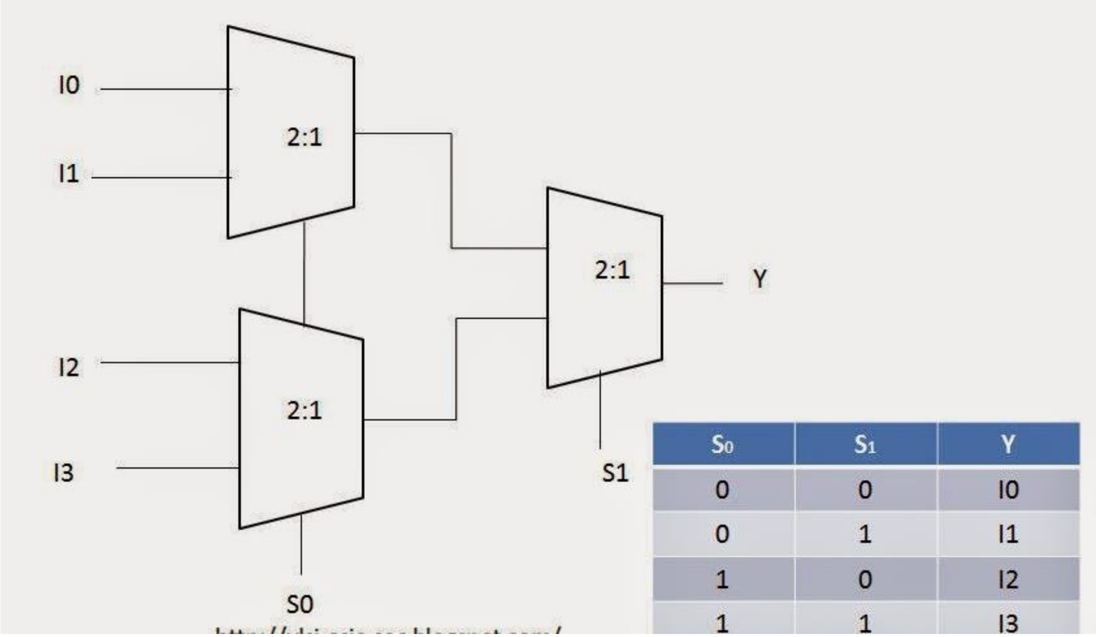
- E is the enable pin. If E = 0, then whatever the selection pin is, the output Y = 0. That's why S0 = don't care.
- If E = 1, then Y = input, on the basis of selection pin S0.

| E | S0 | Y |
|---|---|---|
| 0 | X | 0 |
| 1 | 0 | D0 |
| 1 | 1 | D1 |



# 4x1 Mux using 2x1 Mux

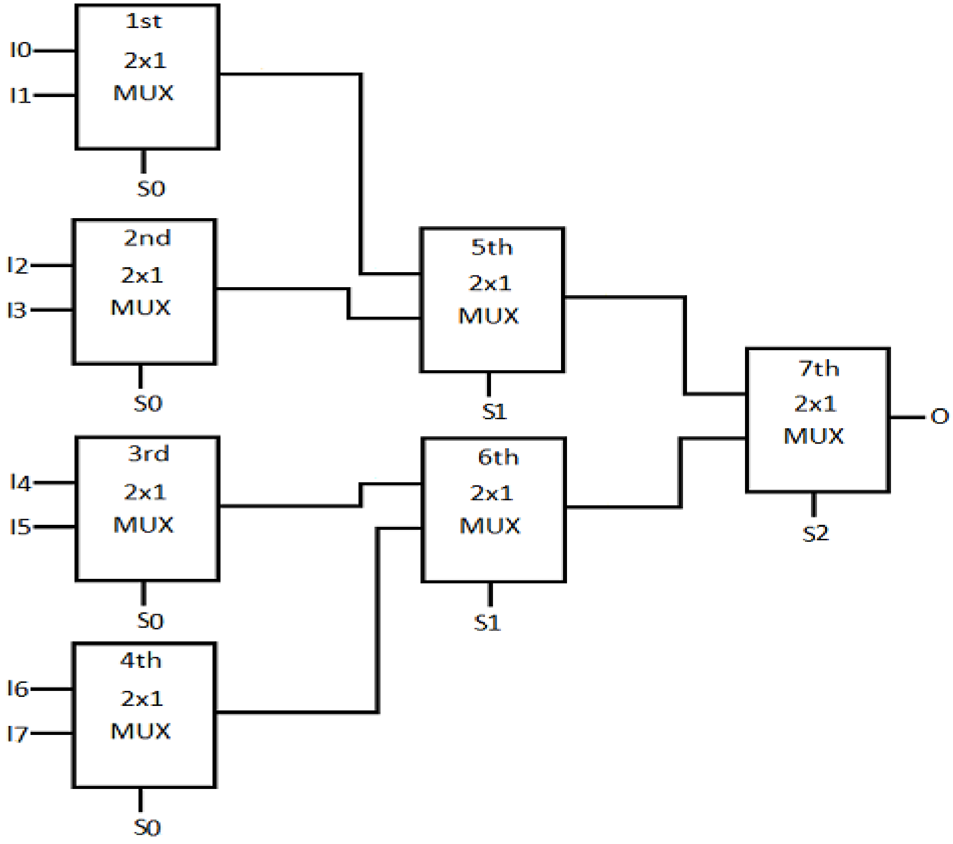
- 4/2 = 2
- 2/2 = 1
- We need total 3 (2x1) MUX




| $S_0$ | $S_1$ | Y |
|---|---|---|
| 0 | 0 | I0 |
| 0 | 1 | I1 |
| 1 | 0 | I2 |
| 1 | 1 | I3 |

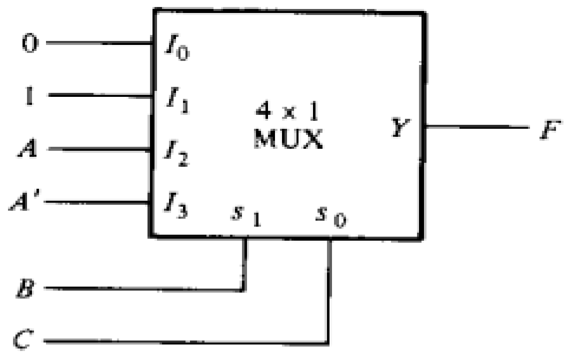
# 8x1 Mux using 2x1 Mux

- 8/2 = 4
- 4/2 = 2
- 2/2 = 1
- We need total 7 (2x1) MUX



# Boolean Function Implementation using MUX

$$F(A, B, C) = \Sigma(1, 3, 5, 6)$$




(a) Multiplexer implementation

| Minterm | A | B | C | F |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 1 |
| 7 | 1 | 1 | 1 | 0 |

(b) Truth table

| | $I_0$ | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|---|
| $A'$ | 0 | 1 | 2 | 3 |
| $A$ | 4 | 5 | 6 | 7 |
| | 0 | 1 | $A$ | $A'$ |

(c) Implementation table

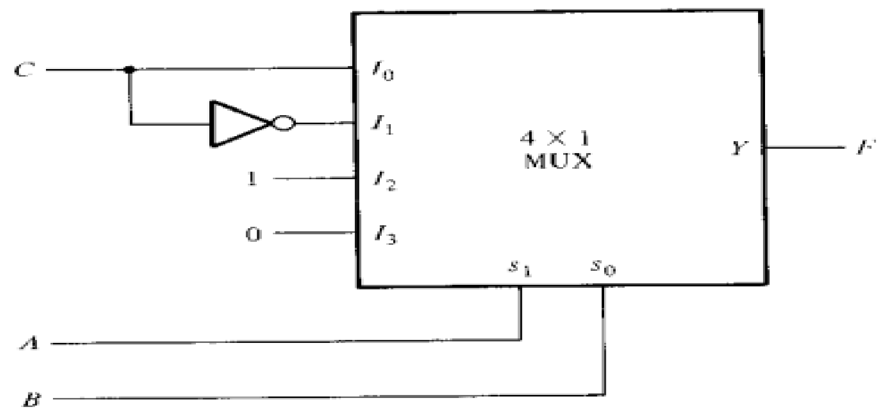
**FIGURE 5-18**

# Boolean Function Implementation using MUX

- F(A,B,C)=Σ(1,2,4,5)

| A | B | C | F | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | F = C |
| 0 | 0 | 1 | 1 | |
| 0 | 1 | 0 | 1 | F = C′ |
| 0 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 1 | F = 1 |
| 1 | 0 | 1 | 1 | |
| 1 | 1 | 0 | 0 | F = 0 |
| 1 | 1 | 1 | 0 | |

(a) Truth table




(b) Multiplexer implementation

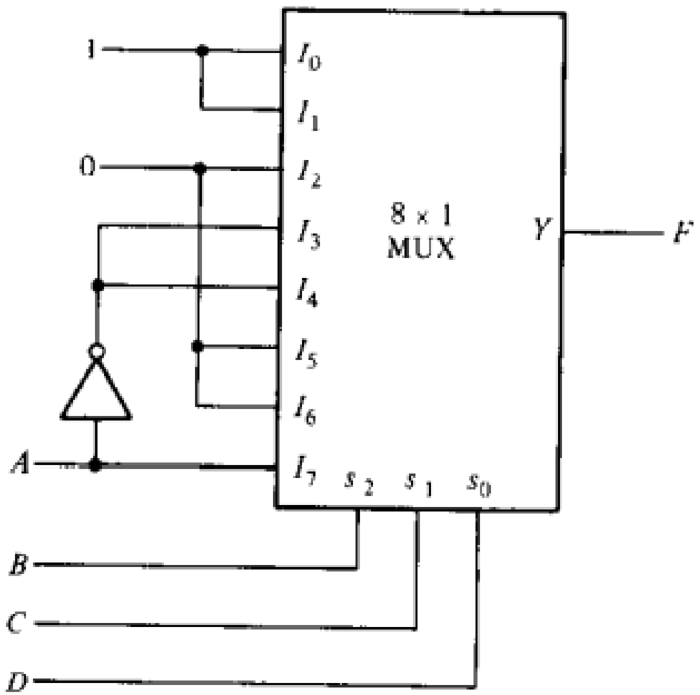
| | $I_0$ | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|---|
| C′ | 0 | 2 | 4 | 6 |
| C | 1 | 3 | 5 | 7 |
| | C | C′ | 1 | 0 |

(c) Implementation table

# Boolean Function Implementation using MUX

- F=(A,B,C,D) = Σ(0,1,3,4,8,9,15)

| | $I_0$ | $I_1$ | $I_2$ | $I_3$ | $I_4$ | $I_5$ | $I_6$ | $I_7$ |
|---|---|---|---|---|---|---|---|---|
| $A'$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| $A$ | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| | 1 | 1 | 0 | $A'$ | $A'$ | 0 | 0 | $A$ |




GOOD NEWS!
THE CLASS IS OVER…
THANK YOU!