RV Institute of Technology and Management®

Go, change the world

Combinational Logic and Sequential Logic
BCS302

Module 2

Prof Padmasree N.
Assistant Professor,
Dept. of CSE, RVITM, Bengaluru
padmasree.rvitm@rvei.edu.in

Prof . Pushpa G
Assistant Professor,
Dept. of ISE, RVITM, Bengaluru
pushpag.rvitm@rvei.edu.in

Prof . Krishnamurthy H
Assistant Professor,
Dept. of ISE, RVITM, Bengaluru
pushpag.rvitm@rvei.edu.in

Referred Books/Sources

- Charles H Roth and Larry L Kinney, Fundamental of Logic Design, Cengage Learning, 2017.
- Donald P Leach, Albert Paul Malvino & Goutam Saha: Digital Principles and Applications, 7 Edition, Tata McGraw Hill, 2015.
- A Verilog HDL Primer by J. Bhasker Bk &Hardcover Published by Star Galaxy Press.
- Verilog HDL: A Guide to Digital Design and Synthesis by Samir Palnitkar Published by Prentice Hall Publication date: March 1996.

Module-2

Combinational circuit design and simulation using gates

&

Multiplexers, Adders,Encoders,Introduction to VHDL, latches and Flipflops

Objectives

1. Draw a timing diagram for a combinational circuit with gate delays.
2. Define static 0- and 1-hazards and dynamic hazards.
3. Given a combinational circuit, find all of the static 0- and 1-hazards. For each hazard, specify the order in which the gate outputs must switch in order for the hazard to actually produce a false output. Given a switching function, realize it using a two-level circuit which is free of static and dynamic hazards (for single input variable changes).
4. Design a multiple-output NAND or NOR circuit using gates with limited fan-in.
5. Design and analyze combinational and sequential logic circuits, multiplexers, decoders, encoders, adders, subtractors, and other basic building blocks.

Combinational circuit design and simulation using gates

Review of Combinational Circuit Design

- The first step in the design of a combinational switching circuit is usually to set up a truth table which specifies the output(s) as a function of the input variables.
  - For n input variables this table will have 2n rows.
- Next step is to derive simplified algebraic expressions for the output functions using Karnaugh maps, the Quine-McCluskey method, or a similar procedure.
- The simplified algebraic expressions are then manipulated into the proper form, depending on the type of gates to be used in realizing the circuit.
- The number of levels in a gate circuit is equal to the maximum number of gates through which a signal must pass when going between the input and output terminals.

Review of Combinational Circuit Design

- The minimum sum of products (or product of sums) leads directly to a minimum two-level gate circuit.
- However, in some applications it is desirable to increase the number of levels by factoring (or multiplying out) because this may lead to a reduction in the number of gates or gate inputs.
- When a circuit has two or more outputs, common terms in the output functions can often be used to reduce the total number of gates or gate inputs.
  - If each function is minimized separately, this does not always lead to a minimum multiple-output circuit.
- For a two-level circuit, Karnaugh maps of the output functions can be used to find the common terms.
- All of the terms in the minimum multiple-output circuit will not necessarily be prime implicants of the individual functions.
- When designing circuits with three or more levels, looking for common terms on the Karnaugh maps may be of little value.
- In this case, the designer will often minimize the functions separately and, then, use ingenuity to factor the expressions in such a way to create common terms.

Review of Combinational Circuit Design

- Minimum two-level AND-OR, NAND-NAND, OR-NAND, and NOR-OR circuits can be realized using the minimum sum of products as a starting point.
- Minimum two-level OR-AND, NOR-NOR, AND-NOR, and NAND-AND circuits can be realized using the minimum product of sums as a starting point.
- Design of multi-level multiple-output NAND-gate circuits is most easily accomplished by first designing a circuit of AND and OR gates.
- Usually, the best starting point is the minimum sum-of-products expressions for the output functions.
- These expressions are then factored in various ways until an economical circuit of the desired form can be found.
- **If this circuit has an OR gate at each output and is arranged so that an AND-gate (or OR-gate) output is never connected to the same type of gate, a direct conversion to a NAND-gate circuit is possible.**
- Conversion is accomplished by replacing all of the AND and OR gates with NAND gates and then inverting any literals which appear as inputs to the first, third, fifth, . . . levels (output gates are the first level).

Review of Combinational Circuit Design

- Similarly, design of multi-level, multiple-output NOR-gate circuits is most easily accomplished by first designing a circuit of AND and OR gates.
- In this case the best starting point is usually the minimum sum-of-products expressions for the complements of the output functions.
- After factoring these expressions to the desired form, they are then complemented to get expressions for the output functions, and the corresponding circuit of AND and OR gates is drawn.

Design of Circuits with Limited Gate Fan-In

- In practical logic design problems, the maximum number of inputs on each gate **(or the fan-in)** is limited.
- Depending on the type of gates used, this limit may be two, three, four, eight, or some other number.
- If a two-level realization of a circuit requires more gate inputs than allowed, factoring the logic expression to obtain a multi-level realization is necessary.

Design of Circuits with Limited Gate Fan-In

- Example:
- Realize f(a, b, c, d) = Σm(0, 3, 4, 5, 8, 9, 10, 14, 15) using three-input NOR gates

map of f:

cd \ ab	00	01	11	10
00	1	1	0	1
01	0	1	0	1
11	1	0	1	0
10	0	0	1	1

- f′ = a′b′c′d + ab′cd + abc′ + a′bc + a′cd′

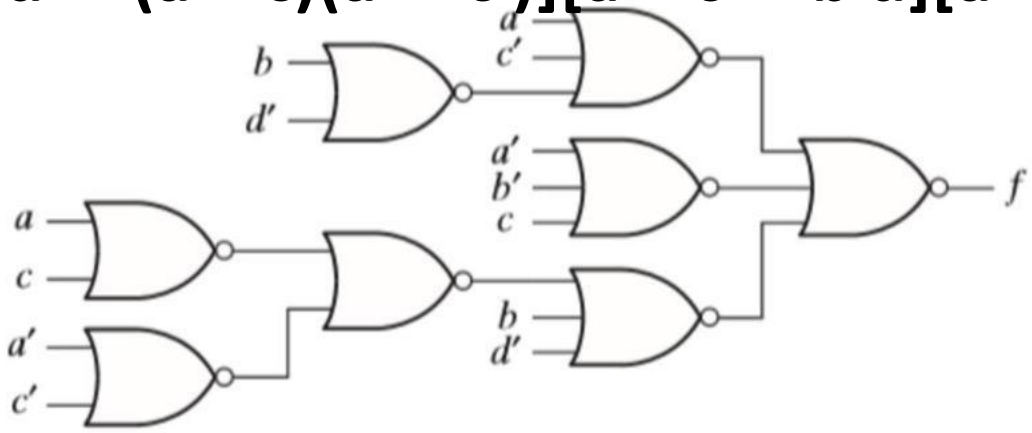
Design of Circuits with Limited Gate Fan-In

$$f' = a'b'c'd + ab'cd + abc' + a'bc + a'cd'$$

- A two-level realization requires two four-input gates and one five-input gate. The expression for f ′ is factored to reduce the maximum number of gate inputs to three and, then, it is complemented:

$$f' = b'd(a'c' + ac) + a'c(b + d') + abc'$$

$$f = [b + d' + (a + c)(a' + c')][a + c' + b'd][a' + b' + c]$$

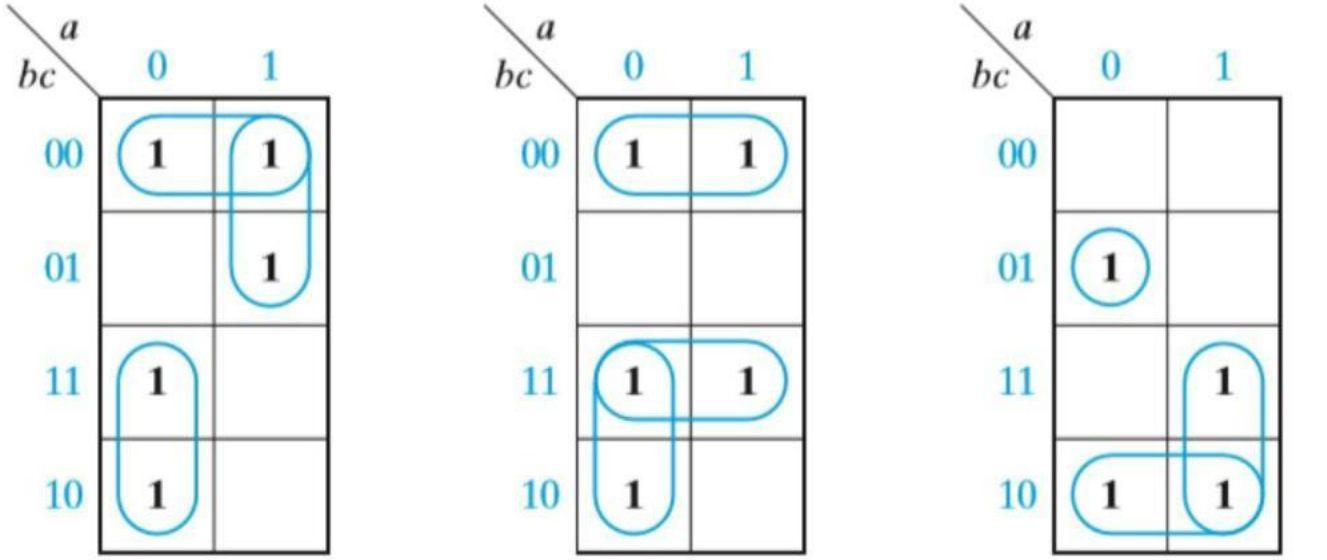


Design of Circuits with Limited Gate Fan-In

- The techniques for designing two-level, multiple-output circuits are not very effective for designing multiple-output circuits with more than two levels.
- Even if the two-level expressions had common terms, most of these common terms would be lost when the expressions were factored.
- Therefore, when designing multiple-output circuits with more than two levels, it is usually best to minimize each function separately.
- The resulting two-level expressions must then be factored to increase the number of levels.
- This factoring should be done in such a way as to introduce common terms wherever possible.

Design of Circuits with Limited Gate Fan-In

- Example
- Realize the functions given below, using only two-input NAND gates and inverters. If we minimize each function separately, the result is



$f_1 = \Sigma\, m(0, 2, 3, 4, 5)$ $f_2 = \Sigma\, m(0, 2, 3, 4, 7)$ $f_3 = \Sigma\, m(1, 2, 6, 7)$

Design of Circuits with Limited Gate Fan-In

f1 = b′c′ + ab′ + a′b

f2 = b′c′ + bc + a′b

f3 = a′b′c + ab + bc′

- Each function requires a three-input OR gate, so we will factor to reduce the number of gate inputs:

  f1 = b′(a + c′) + a′b

  f2 = b(a′ + c) + b′c′ or f2 = (b′ + c)(b + c′) + a′b

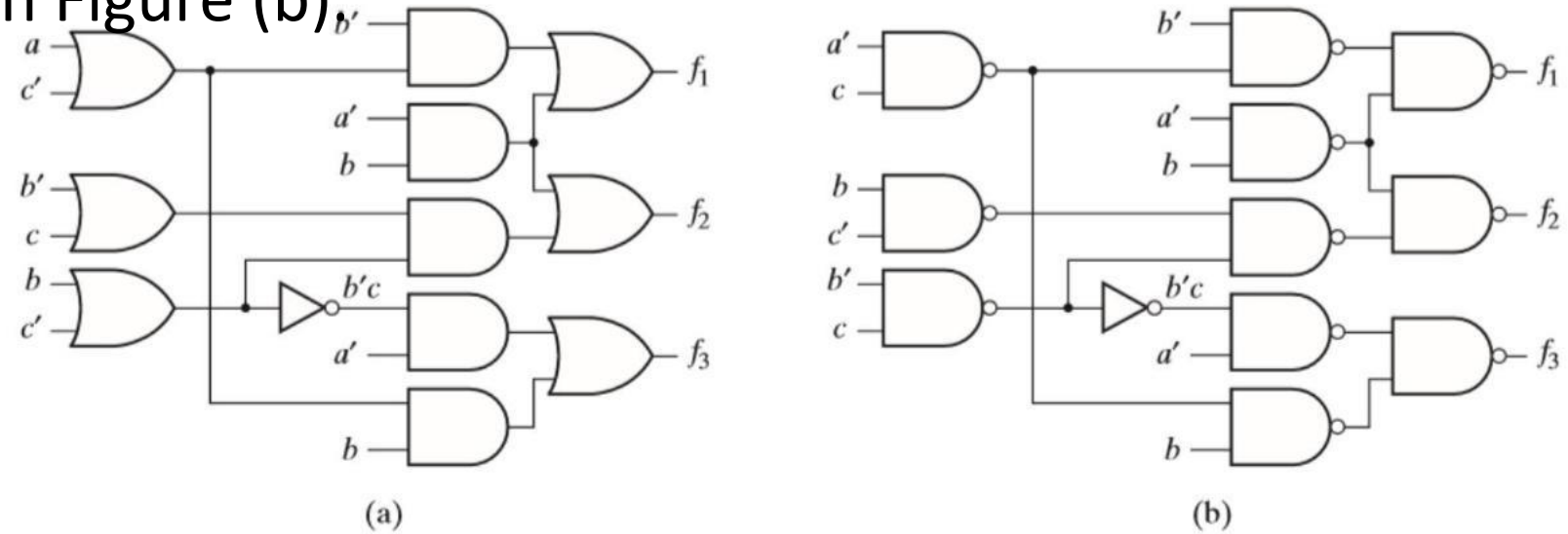
  f3 = a′b′c + b(a + c′)

- The second expression for f2 has a term common to f1, so we will choose the second expression.
- We can eliminate the remaining three-input gate from f3 by noting that

  a′b′c = a′(b′c) = a′(b + c′)′

Design of Circuits with Limited Gate Fan-In

- Figure (a) shows the resulting circuit, using common terms a'b and a + c'.
- Because each output gate is an OR, the conversion to NAND gates, as shown in Figure (b).

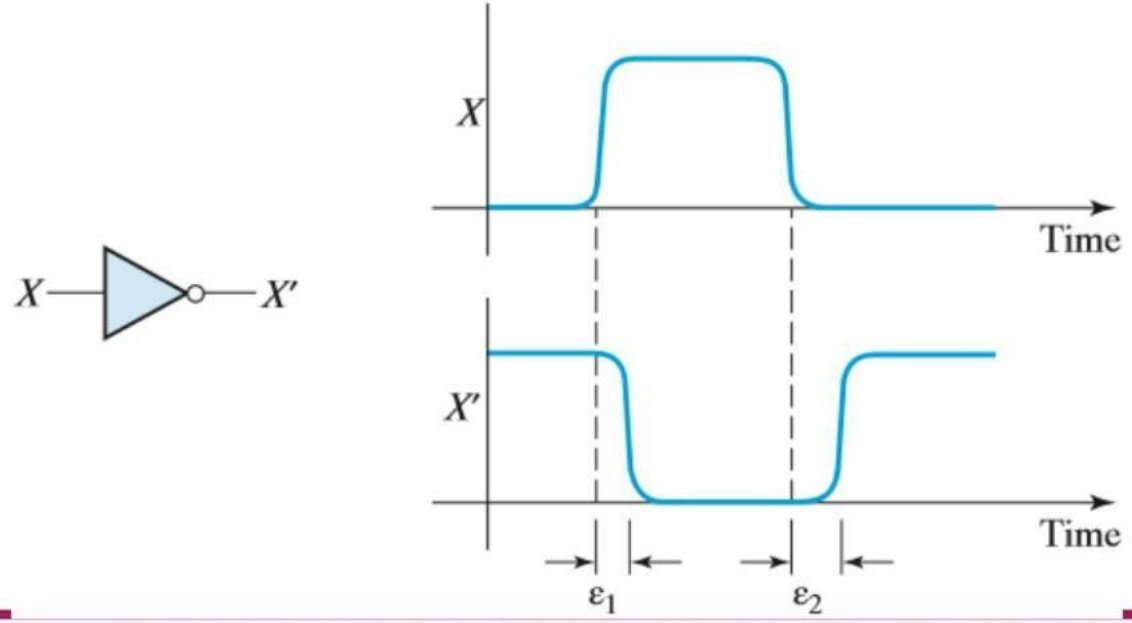


Gate Delays and Timing Diagrams

- When the input to a logic gate is changed, the output will not change instantaneously.
- The transistors or other switching elements within the gate take a finite time to react to a change in input, so that the change in the gate output is delayed with respect to the input change, this delay is called **propagation delay**.

Gate Delays and Timing Diagrams

- Figure shows possible input and output waveforms for an inverter.
  - If the change in output is delayed by time, ε , with respect to the input, we say that this gate has a **propagation delay** of ε.
  - In practice, the propagation delay for a 0 to 1 output change may be different than the delay for a 1 to 0 change.
  - Propagation delays for integrated circuit gates may be as short as a few nanoseconds (Negligible).

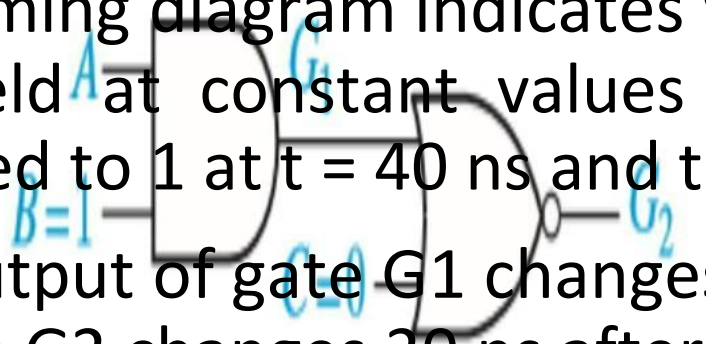


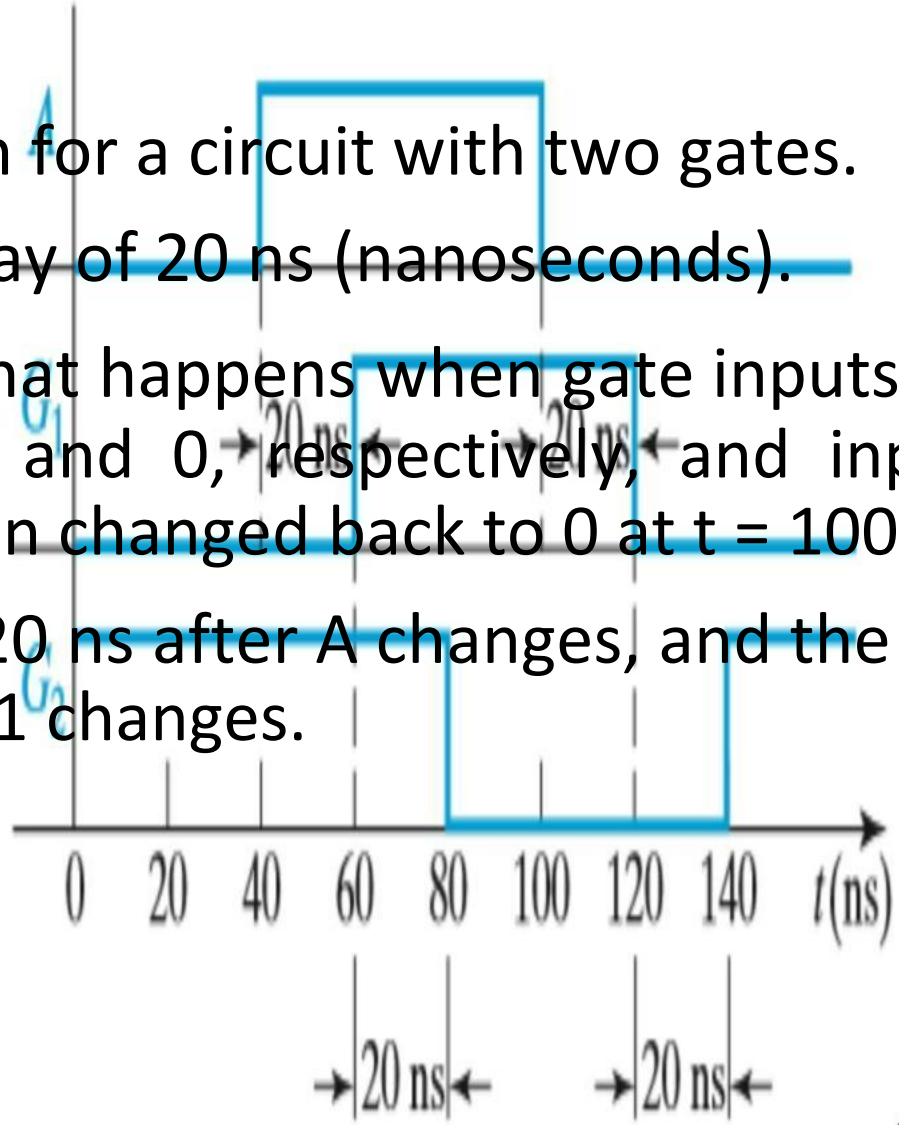
Gate Delays and Timing Diagrams

- **Timing diagrams are frequently used in the analysis of sequential circuits with respect to time** (like memory circuits E.g. Flip-Flop, RAM).
- These diagrams show various signals in the circuit as a function of time.
- Several variables are usually plotted with the same time scale so that the times at which these variables change with respect to each other can easily be observed.

Gate Delays and Timing Diagrams

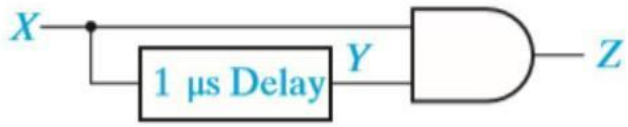
- Figure shows the timing diagram for a circuit with two gates.
- Each gate has a propagation delay of 20 ns (nanoseconds).
- This timing diagram indicates what happens when gate inputs B and C are held at constant values 1 and 0, respectively, and input A is changed to 1 at t = 40 ns and then changed back to 0 at t = 100 ns.
- The output of gate G1 changes 20 ns after A changes, and the output of gate G2 changes 20 ns after G1 changes.

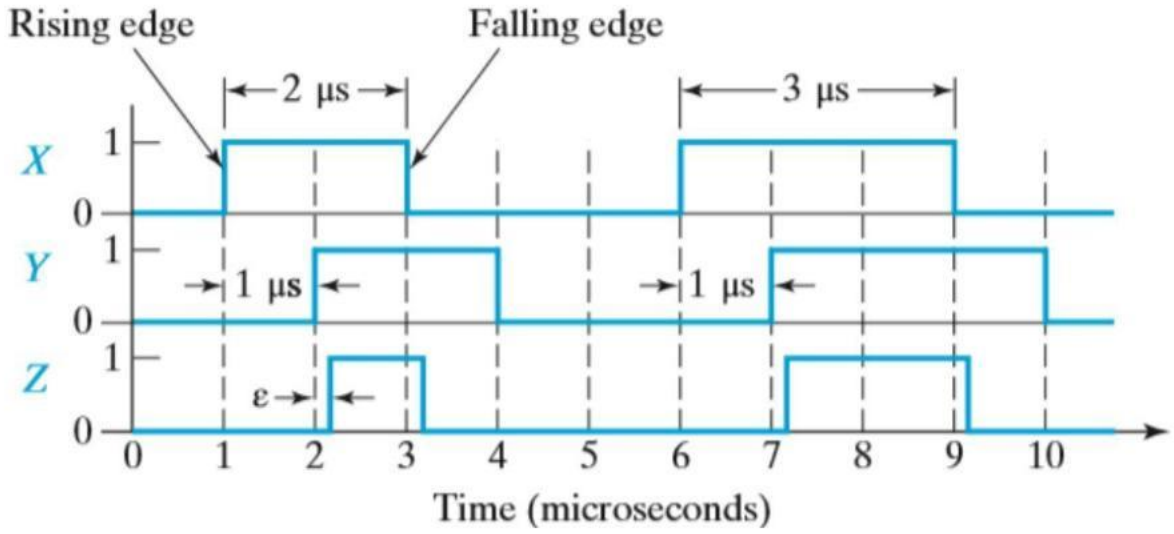




Gate Delays and Timing Diagrams

- Figure shows a timing diagram for a circuit with an added delay element.
- **The input X consists of two pulses, the first of which is 2 microseconds (2×10^{-6} second) wide and the second is 3 microseconds wide.**
- The delay element has an **output Y** which is the same as the input except that it is delayed by **1 microsecond.**





Gate Delays and Timing Diagrams

- That is, Y changes to a 1 value 1 microsecond after the rising edge of the X pulse and returns to 0 1 microsecond after the falling edge of the X pulse.
- The output (Z) of the AND gate should be 1 during the time interval in which both X and Y are 1.
- If we assume a small **propagation delay in the AND gate (ε),** then Z will be as shown in Figure

Hazards in Combinational Logic

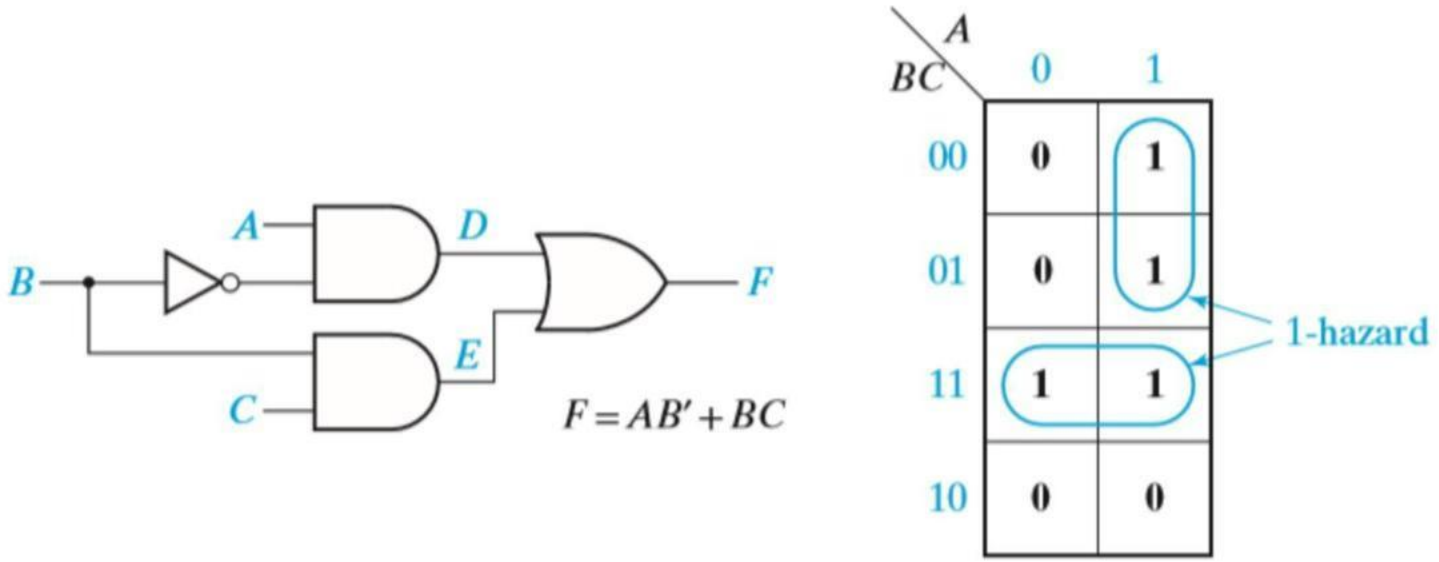
- When the input to a combinational circuit changes, unwanted switching transients may appear in the output.
- These transients occur when different paths from input to output have different propagation delays.

Hazards in Combinational Logic

- If, in response to any single input change and for some combination of propagation delays, a circuit output may momentarily go to 0 when it should remain a constant 1, we say that the circuit has a **static 1-hazard**.
- Similarly, if the output may momentarily go to 1 when it should remain a 0, we say that the circuit has a **static 0-hazard.**
- If, when the output is supposed to change from 0 to 1 (or 1 to 0), the output may change three or more times, we say that the circuit has a **dynamic hazard.**

(a) Static 1-hazard (b) Static 0-hazard (c) Dynamic hazards

Hazards in Combinational Logic Static 1-hazard

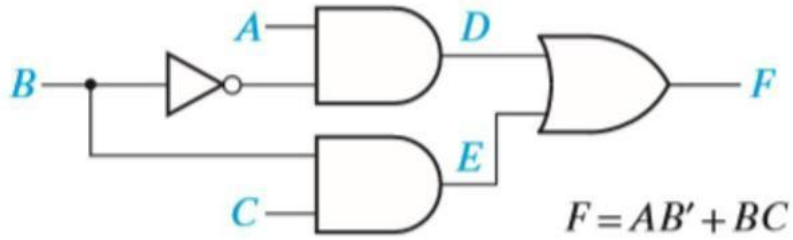


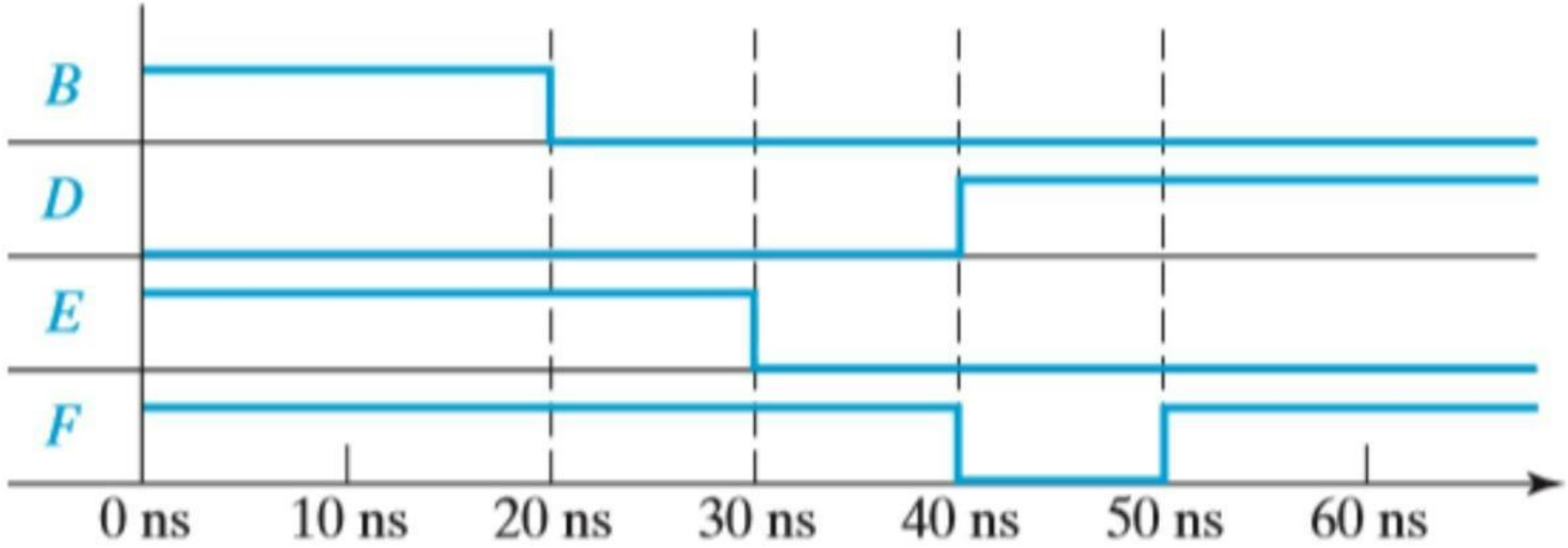
(a) Circuit with a static 1-hazard

- If A = C = 1, then F = B + B′ = 1, so the F output should remain a constant 1 when B changes from 1 to 0.
- If each gate has a propagation delay of 10 ns

Hazards in Combinational Logic Static 1-hazard

- **Static 1-hazard**





(b) Timing chart

Hazards in Combinational Logic Static 1-hazard

- E will go to 0 before D goes to 1, resulting in a momentary 0 (a glitch caused by the 1-hazard) appearing at the output F.
- Note that right after B changes to 0, both the inverter input (B) and output (B’) are 0 until the propagation delay has elapsed.
- During this period, both terms in the equation for F are 0, so F momentarily goes to 0.

Hazards in Combinational Logic Static 1-hazard

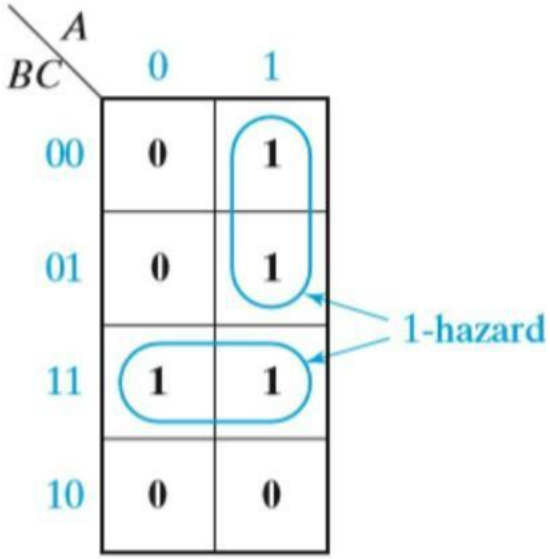
- Note that hazards are properties of the circuit and are independent of the delays existing in the circuit.
- If the circuit is free of hazards, then for any combination of delays that might exist in the circuit and for any single input change, the output will not contain a transient.
- If a circuit contains a hazard, then there is some combination of delays and some input change for which the circuit output contains a transient.

Hazards in Combinational Logic Static 1-hazard

- How to remove Hazard
- Assume the inverter has a delay of 2 ns rather than 10 ns.
- Then the D and E changes reaching the output OR gate are 2 ns apart, in which case the OR gate may not generate the 0 glitch.
- **Hazards can be detected using a Karnaugh map**

Hazards in Combinational Logic Static 1-hazard

- As seen on the map, no loop covers both minterms ABC and AB'C.
- So if A = C = 1 and B changes, both terms can momentarily go to 0, resulting in a glitch in F.



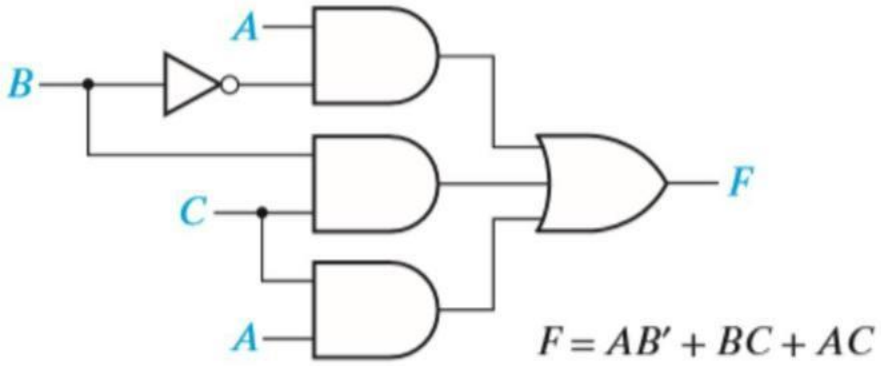
Hazards in Combinational Logic Static 1-hazard

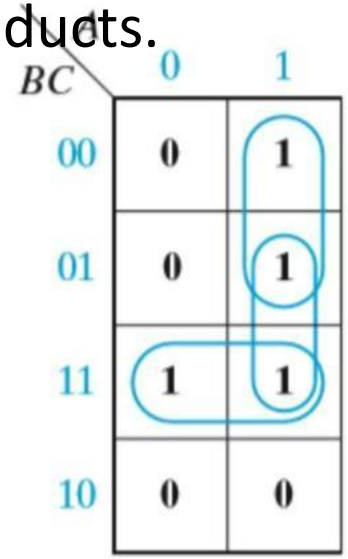
- **Detect hazards in a two-level AND-OR circuit, using the following procedure:**

1. Write down the sum-of-products expression for the circuit.
2. Plot each term on the map and loop it.
3. If any two adjacent 1's are not covered by the same loop, a 1-hazard exists for the transition between the two 1's. For an n-variable map, this transition occurs when one variable changes and the other n − 1 variables are held constant.

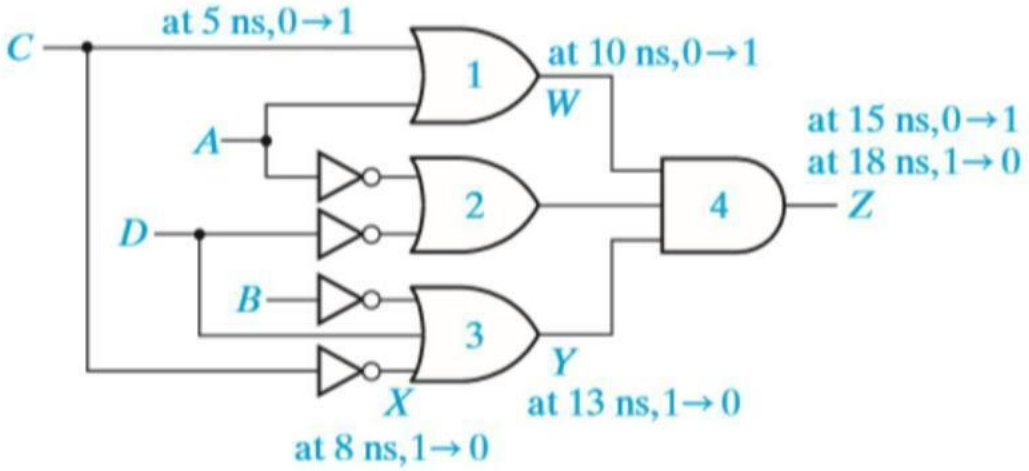
Hazards in Combinational Logic Static 1-hazard

- If we add a loop to the redundant map and, then, add the corresponding gate to the circuit, this eliminates the hazard.
- The term AC remains 1 while B is changing, so no glitch can appear in the output.
  - Note that F is no longer a minimum sum of products.

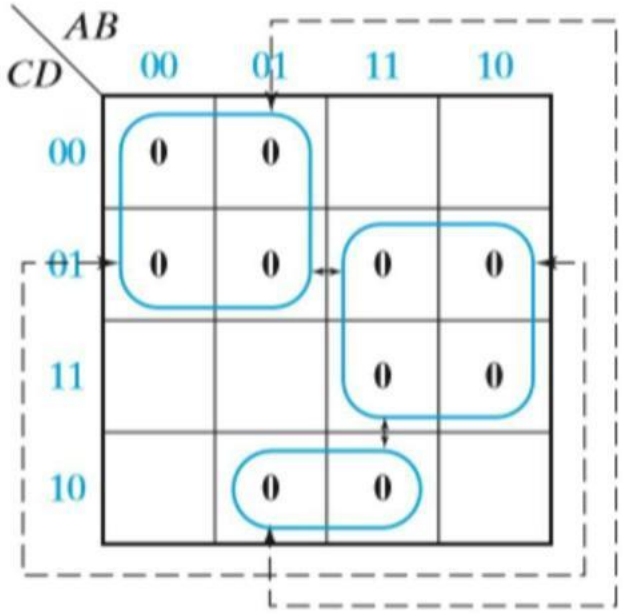




Hazards in Combinational Logic Static 0-hazard



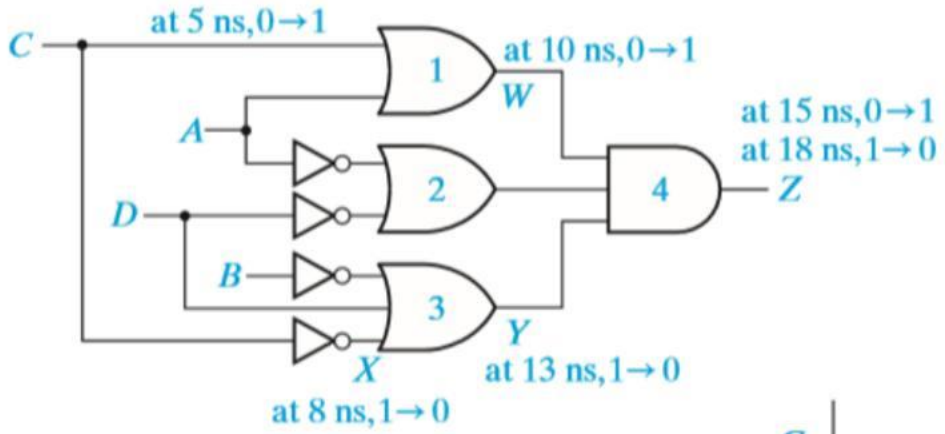
(a) Circuit with a static 0-hazard



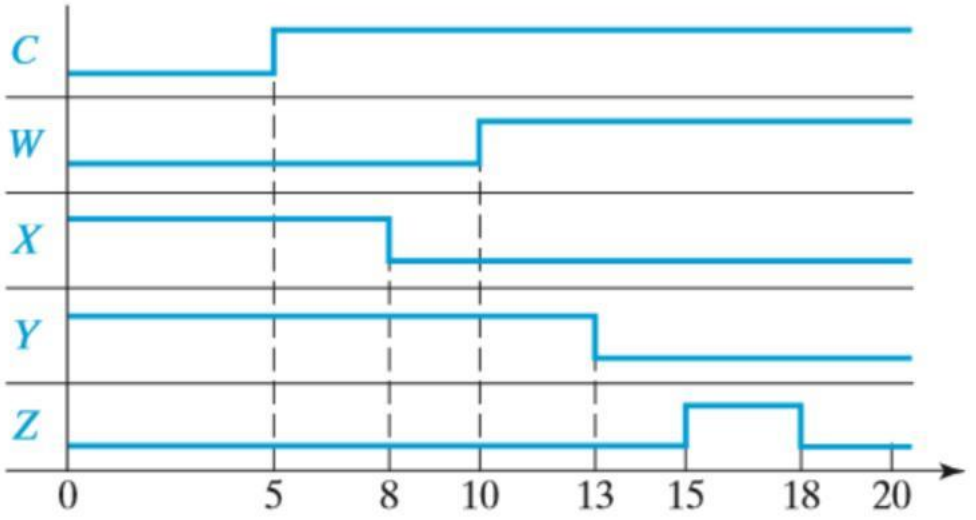
(b) Karnaugh map for circuit of (a)

- F = (A + C)(A′ + D′)(B′ + C′ + D)

Hazards in Combinational Logic Static 0-hazard



(a) Circuit with a static 0-hazard



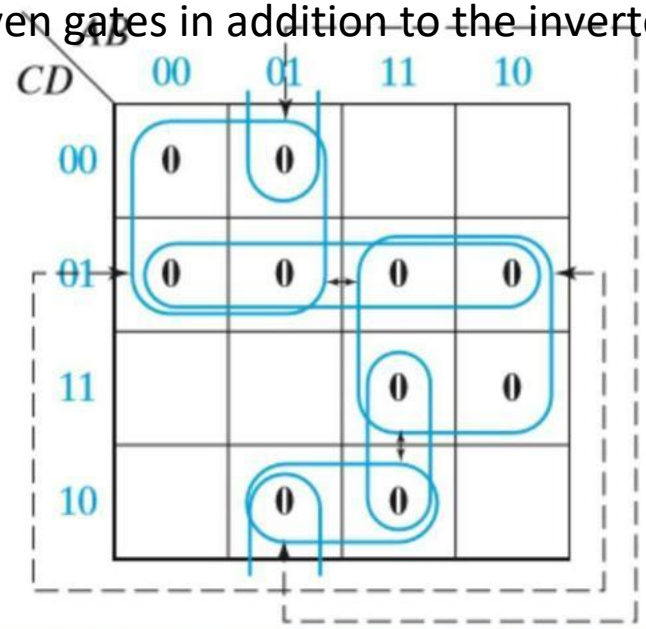
(c) Timing diagram illustrating 0-hazard of (a)

Hazards in Combinational Logic Static 0-hazard

- We can eliminate the 0-hazards by looping additional prime implicants that cover the adjacent 0's that are not already covered by a common loop.
- This requires three additional loops as shown in Figure.
- The resulting equation is

$F = (A + C)(A' + D')(B' + C' + D)(C + D')(A + B' + D)(A' + B' + C')$

- and the resulting circuit requires seven gates in addition to the inverters.



Hazards in Combinational Logic Dynamic hazard

- Hazards in circuits with more than two levels can be determined by deriving either a SOP or POS expression for the circuit that represents a two-level circuit containing the same hazards as the original circuit.
- The SOP or POS expression is derived in the normal manner except that the complementation laws are not used, i.e., $xx' = 0$ and $x + x' = 1$ are not used.

Hazards in Combinational Logic Dynamic hazard

- So, the resulting SOP (POS) expression may contain products (sums) of the form **$xx'\alpha\ (x + x' + \beta)$.**
  - Where **α** is a product of literals or it may be null; **β** is a sum of literals or it may be empty.
- The complementation laws are not used because we are analyzing the circuit behavior resulting from an input change.
- **As that input change propagates through the circuit, at a given point in time a line tending toward the value x may not have the value that is the complement of a line tending toward the value x'.**
- **In the SOP expression, a product of the form $xx'\alpha$ represents a pseudo gate that may temporarily have the output value 1 as x changes and if α = 1.**

Hazards in Combinational Logic Dynamic hazard

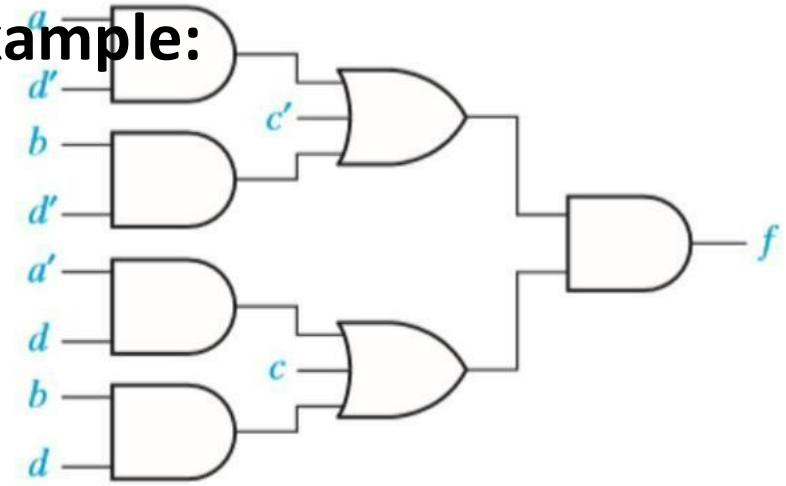
- Given the SOP expression, with a two-level AND-OR circuit, for that:
- **A static 1-hazard exists if the products are mapped on a Karnaugh map and if two 1's are adjacent on the map and not included in one of the products.**
- **A static 0-hazard exists if there are two adjacent 0's on the Karnaugh map for which α = 1 and the two input combinations differ just in the value of x.**
- The circuit can have a **static 0-hazard or a dynamic hazard only if the SOP expression contains a term of the form xx'α.**

Hazards in Combinational Logic Dynamic hazard

- **A dynamic hazard exists** if there is a term of the form xx'α and two conditions are satisfied:
- (1) There are adjacent input combinations on the Karnaugh map differing in the value of x, with α = 1 and with opposite function values.
- (2) for these input combinations the change in **x propagates over at least three paths through the circuit.**

Hazards in Combinational Logic Dynamic hazard

- **Example:**



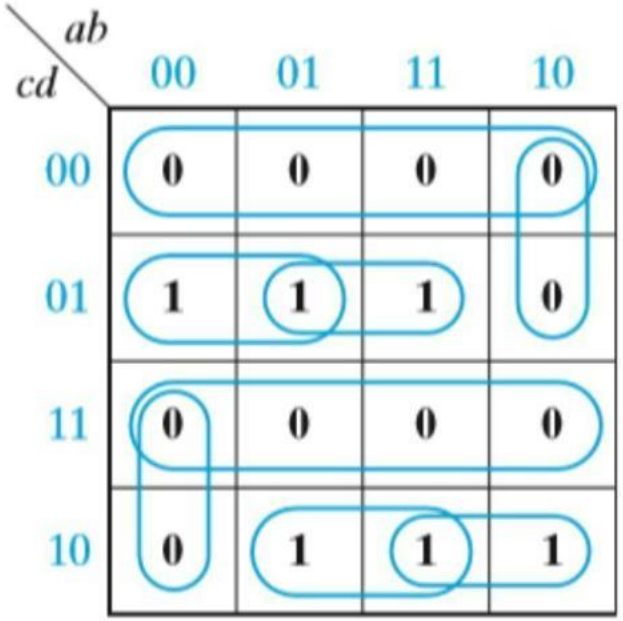
POS Expression

f= (c′ + ad′ + bd′)(c + a′d + bd)

= cc′ + acd′ + bcd′ + a′c′d + aa′dd′ + a′bdd′ + bc′d + abdd′ + bdd′ = cc′ + acd′ + bcd′ + a′c′d + aa′dd′ + bc′d + bdd′

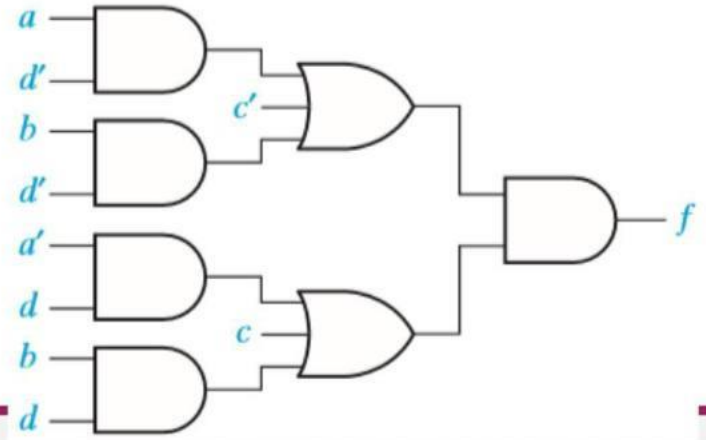
Hazards in Combinational Logic Dynamic hazard

- The Karnaugh map for previous slide example is shown as the circled 1's .
- The circuit does not contain any static 1-hazards because each pair of adjacent 1's are covered by one of the product terms.



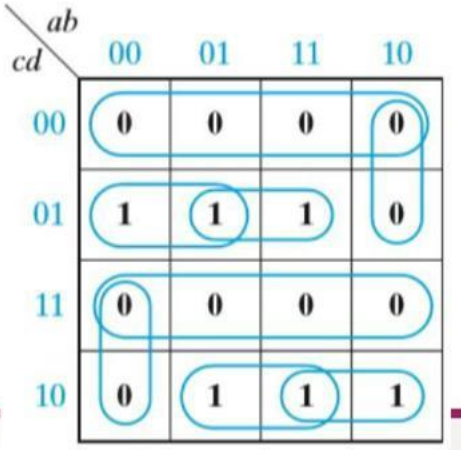
Hazards in Combinational Logic Dynamic hazard

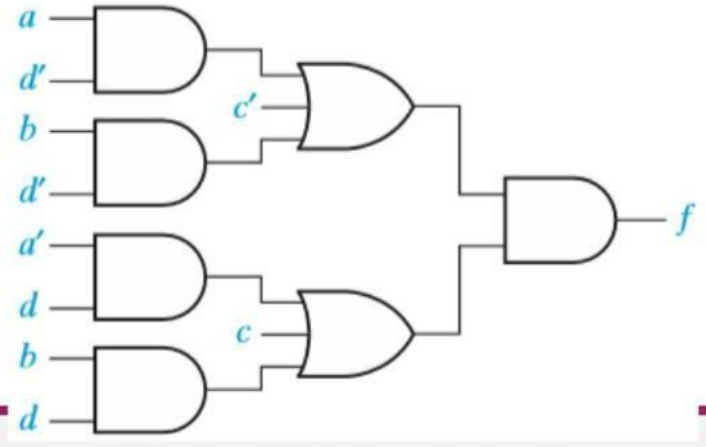
- Will check The terms **cc' and bdd'** may cause either static 0- or dynamic hazards or both.
- **First for c changing and the second for d changing.**
  - The term aa'dd' cannot cause either hazard because, for example, if a changes the dd' part of the product forces it to 0.
- **With a = 0, b = 0, and d = 0 and c changing,** the circuit output is 0 before and after the change, and because the cc' term can cause the output to temporarily become 1, this transition is a static 0-hazard.
  - Similarly, a change in c, with a = 1, b = 0, and d = 1, is a static 0-hazard.
- **The cc' term cannot cause a dynamic hazard because there are only two physical paths from input c to the circuit output.**



Hazards in Combinational Logic Dynamic hazard

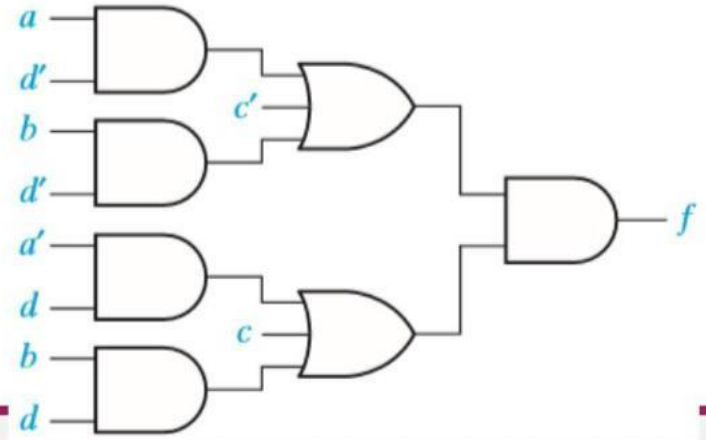
- Lets check the term bdd’ can cause a static 0- or dynamic hazard only if b = 1.
- From the Karnaugh map, it is seen that, with **b = 1 and d changing,** the circuit output changes for any combination of **a** and **c**, so the **only possibility is that of a dynamic hazard.**
- There are **four physical paths from d** to the circuit output, so a dynamic hazard exists if a d change can propagate over at least three of those paths.





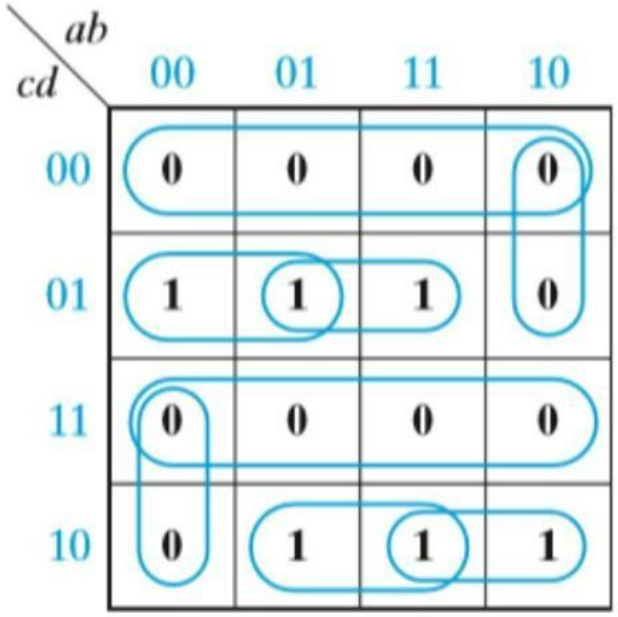
Hazards in Combinational Logic Dynamic hazard

- However, this cannot happen because, with c = 0, propagation over the upper two paths is blocked at the upper OR gate because c′ = 1 forces the OR gate output to be 1,
- With c = 1 propagation over the lower two paths is blocked at the lower OR gate.
- The circuit does not contain a dynamic hazard.



Hazards in Combinational Logic Dynamic hazard

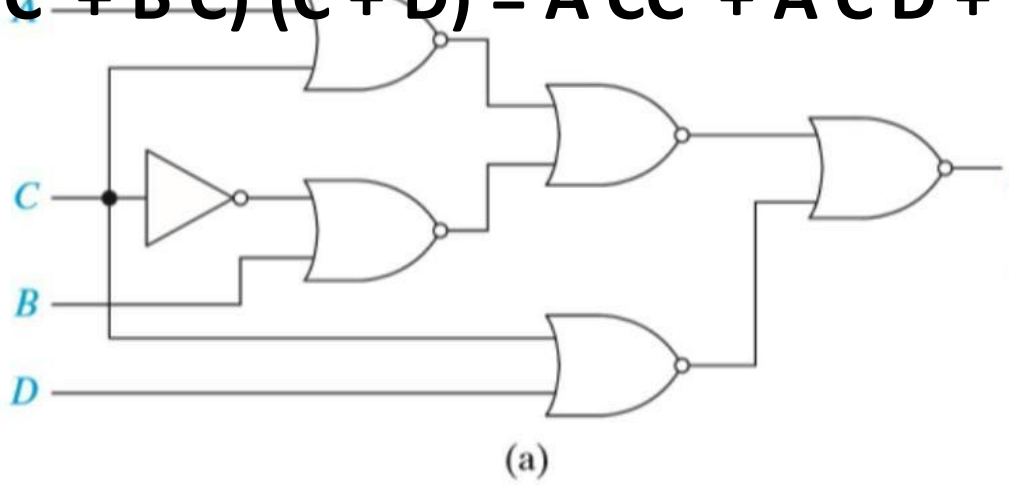
- **Another approach to finding the hazards is as follows:**



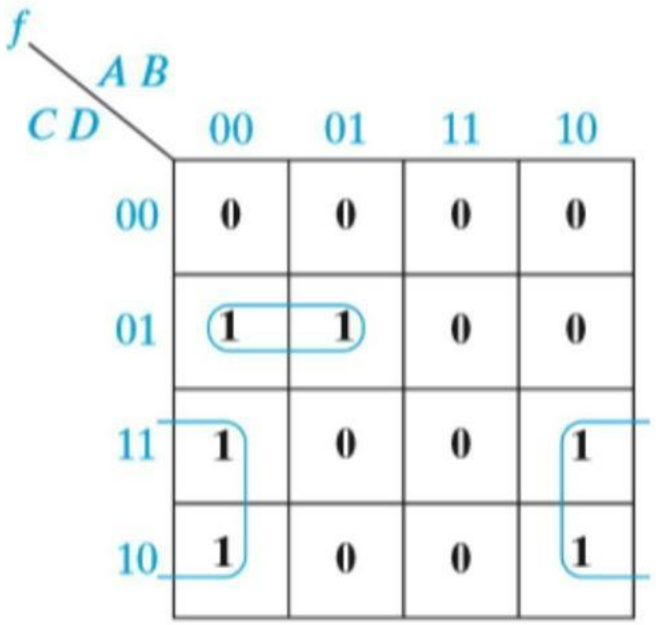
- **POS expression of K-Map is**
- **f = (c' + a + b)(c' + d')(c + a' + b)(c + d)**
- Plotting the 0's of f from this expression on a Karnaugh map reveals that there are 0-hazards when a = b = d = 0 and c changes, and also when b = 0, a = d = 1, and c changes.
- An expression of the form x + x' does not appear in any sum term of f, and this indicates that there are no 1-hazards or dynamic hazards.

Hazards in Combinational Logic Dynamic hazard

- Consider the example to find static and dynamic hazards
- **f = (A'C' + B'C) (C + D) = A'CC' + A'C'D + B'C**



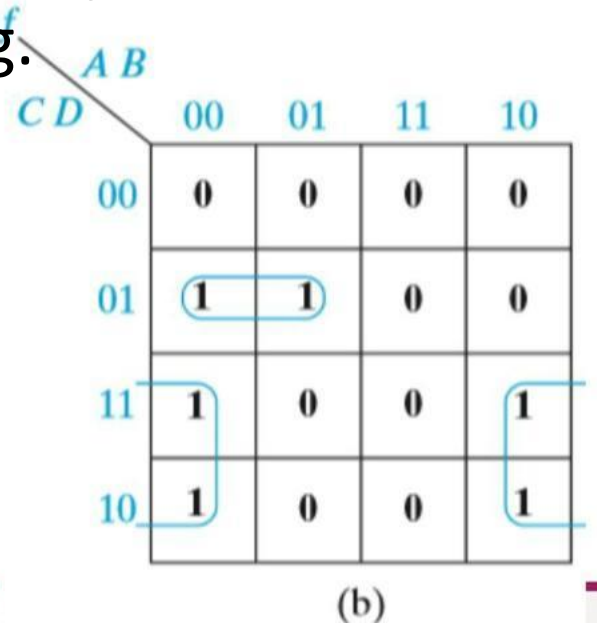
(a)



(b)

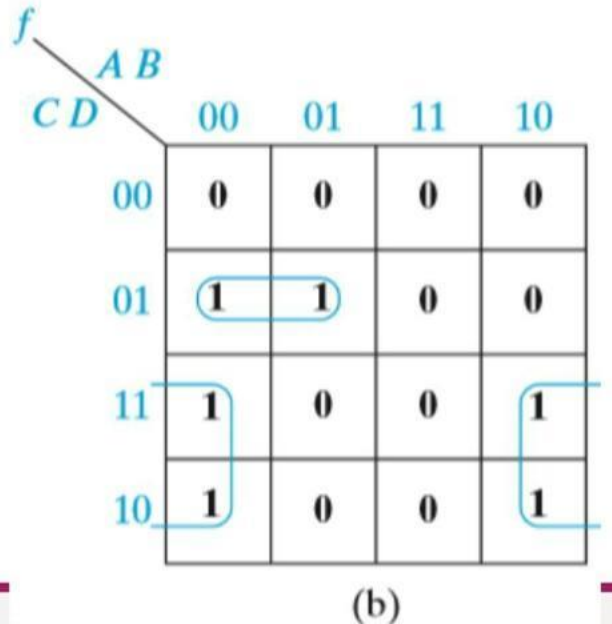
Hazards in Combinational Logic Dynamic hazard

- The Karnaugh map for f in Figure (b) shows that f = 1 for the input combinations (A, B, C, D) = (0, 0, 0, 1) and (0, 0, 1, 1) and neither product of f covers these two minterms; hence, these two input combinations imply a static 1-hazard for C changing.
- The product A'CC' in f indicates the possibility of a static 0-hazard and a dynamic hazard for A = 0 and C changing.



Hazards in Combinational Logic Dynamic hazard

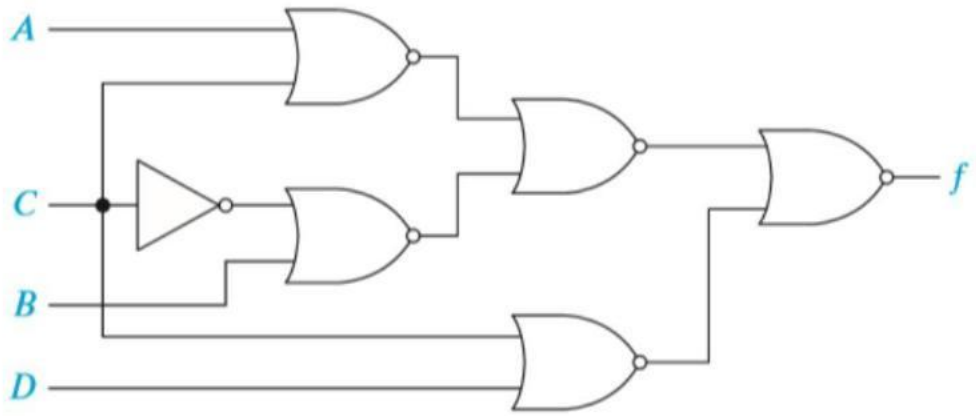
- The Karnaugh map shows that when f = 0, the two input combinations (0, 1, 0, 0) and (0, 1, 1, 0) meet these conditions and, hence, they imply a static 0-hazard.



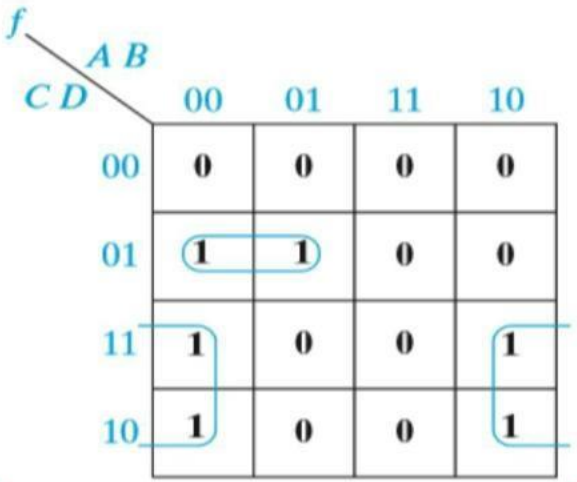
(b)

Hazards in Combinational Logic Dynamic hazard

- The Karnaugh map shows two pairs of input combinations with f changing for **A = 0 and C changing** namely, **(0, 0, 0, 0), (0, 0, 1, 0), and (0, 1, 0, 1), (0, 1, 1, 1).**
- In order for these to be dynamic hazards, the C change must propagate over three or more paths to the output.
- The circuit shows that propagation over the three paths requires B = 0 and D = 0 as well as A = 0; thus, a dynamic hazard only occurs for (0, 0, 0, 0) and (0, 0, 1, 0).
- For (0, 1, 0, 1) and (0, 1, 1, 1), the C change only propagates over one path, and f can only change once.



(a)



(b)

Hazards in Combinational Logic Dynamic hazard

- To design a circuit which is free of static and dynamic hazards, the following procedure may be used:
  1. Find a sum-of-products expression (F^t) for the output in which every pair of adjacent 1's is covered by a 1-term. (The sum of all prime implicants will always satisfy this condition.) A two-level AND-OR circuit based on this F^t will be free of 1-, 0-, and dynamic hazards.
  2. If a different form of the circuit is desired, manipulate F^t to the desired form by simple factoring, DeMorgan's laws, etc. Treat each x_i and x'_i as independent variables to prevent introduction of hazards.

Hazards in Combinational Logic Dynamic hazard

- The hazards and the possibility of resulting glitches in this section has assumed that only a single input can change at a time and that no other input will change until the circuit has stabilized.
- If more than one input can change at one time, then nearly all circuits will contain hazards, and they cannot be eliminated by modifying the circuit implementation.

Simulation and Testing of Logic Circuits

- Verifying the logic design to correct and debugging the design if necessary.
- Logic circuits may be tested either by actually building them or by simulating them on a computer.
- Simulation is generally easier, faster, and more economical.
- As logic circuits become more and more complex, it is very important to simulate a design before actually building it
  - E.g IC or Chip Design
- Simulation is done for several reasons,
  1. verification that the design is logically correct
  2. verification that the timing of the logic signals is correct
  3. simulation of faulty components in the circuit as an aid to finding tests for the circuit.

Simulation and Testing of Logic Circuits

- To use a computer program for simulating logic circuits,
  - you must first specify the circuit components and connections; then, specify the circuit inputs; and, finally, observe the circuit outputs.
- The circuit description may be input into a simulator in the form of a list of connections between the gates and other logic elements in the circuit, or the description may be in the form of a logic diagram drawn on a computer screen.

Simulation and Testing of Logic Circuits

- A simple simulator for combinational logic works as follows:

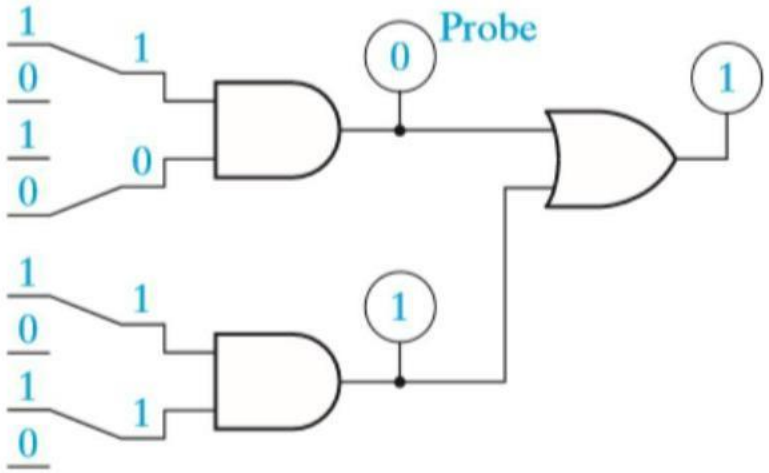
1. The circuit inputs are applied to the first set of gates in the circuit, and the outputs of those gates are calculated.
2. The outputs of the gates which changed in the previous step are fed into the next level of gate inputs. If the input to any gate has changed, then the output of that gate is calculated.
3. Step 2 is repeated until no more changes in gate inputs occur. The circuit is then in a steady-state condition, and the outputs may be read.
4. Steps 1 through 3 are repeated every time a circuit input changes.

Simulation and Testing of Logic Circuits

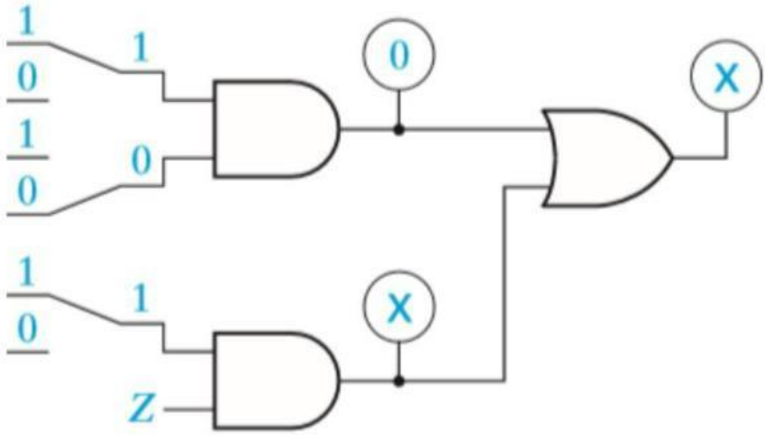
- The two logic values**, 0 and 1**, are not sufficient for simulating logic circuits.
- At times, the value of a gate input or output may be unknown, and we will represent this **unknown value by X**.
- At other times we may have no logic signal at an input, as in the case of an open circuit when an input is not connected to any output. We use the **logic value Z to represent an open circuit, or high impedance (hi-Z) connection**.
- The discussion that follows assumes we are using a four-valued logic simulator with logic values 0, 1, X (unknown), and Z (hi-Z).

Simulation and Testing of Logic Circuits

- Example



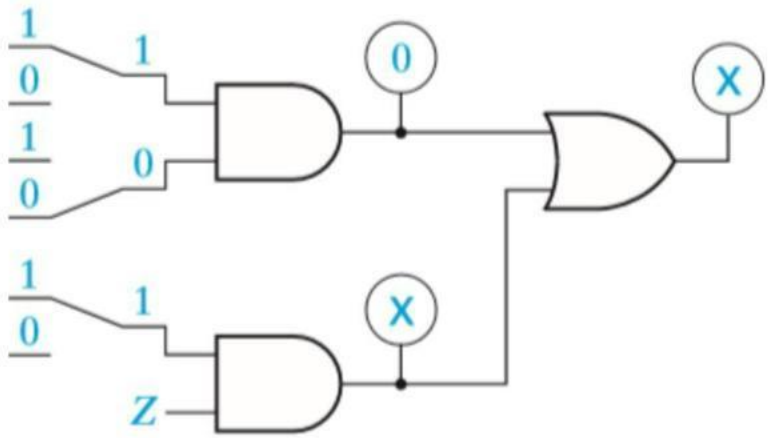
(a) Simulation screen showing switches



(b) Simulation screen with missing gate input

Simulation and Testing of Logic Circuits

- In Figure (b), one gate has no connection to one of its inputs.
- Because that gate has a 1 input and a hi-Z input, we do not know what the hardware will do, and the gate output is unknown.
- This is indicated by an X in the probe



(b) Simulation screen with missing gate input

Simulation and Testing of Logic Circuits

- AND and OR Functions for Four-Valued Simulation
- For an AND gate,
  - if one of the inputs is 0, the output is always 0 regardless of the other input.
  - If one input is 1 and the other input is X (we do not know what the other input is), then the output is X (we do not know what the output is).
  - If one input is 1 and the other input is Z (it has no logic signal), then the output is X (we do not know what the hardware will do).
- For an OR gate,
  - if one of the inputs is 1, the output is 1 regardless of the other input.
  - If one input is 0 and the other input is X or Z, the output is unknown.

·	0	1	X	Z
0	0	0	0	0
1	0	1	X	X
X	0	X	X	X
Z	0	X	X	X

AND Gate

+	0	1	X	Z
0	0	1	X	X
1	1	1	1	1
X	X	1	X	X
Z	X	1	X	X

OR Gate

Simulation and Testing of Logic Circuits

- If a circuit output is wrong for some set of input values, this may be due to several possible causes:

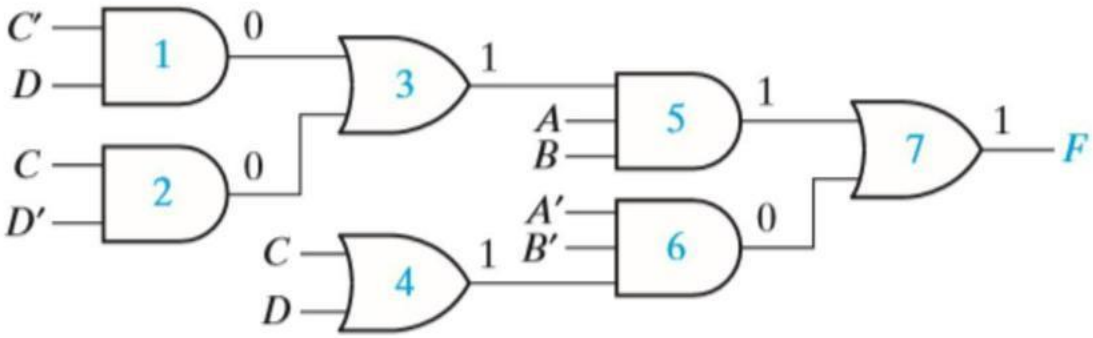
1. Incorrect design
2. Gates connected wrong
3. Wrong input signals to the circuit If the circuit is built in lab, other possible causes include
4. Defective gates
5. Defective connecting wires

Simulation and Testing of Logic Circuits

- Logic Circuit with Incorrect Output then how to troubleshoot
  - If the output gate has the wrong output and its inputs are correct, this indicates that the gate is defective.
  - If one of the inputs is wrong, then either the gate is connected wrong, the gate driving this input has the wrong output, or the input connection is defective.

Simulation and Testing of Logic Circuits

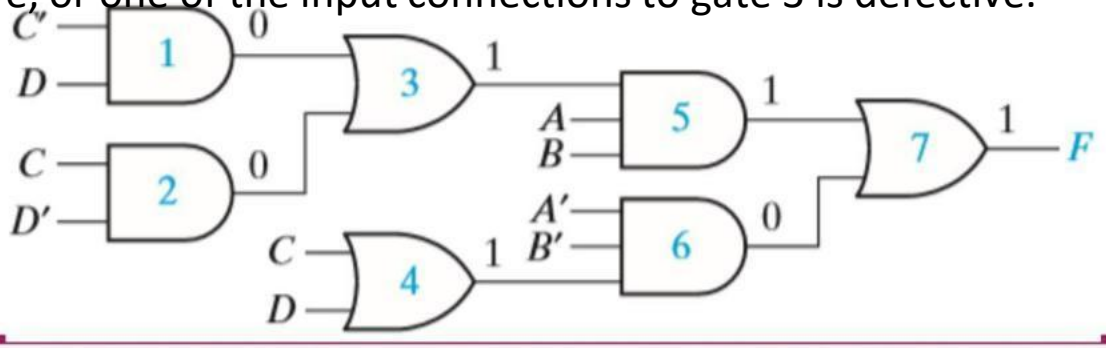
- The function F = AB(C'D + CD') + A'B'(C + D) is realized by the circuit of



Simulation and Testing of Logic Circuits

- For A = B = C = D = 1, the output F = 1 has the wrong value.
- The reason for the incorrect value of F can be determined as follows:

1. The output of gate 7 (F) is wrong, but this wrong output is consistent with the inputs to gate 7, that is, 1 + 0 = 1. Therefore, one of the inputs to gate 7 must be wrong.
2. In order for gate 7 to have the correct output (F = 0), both inputs must be 0. Therefore, the output of gate 5 is wrong. However, the output of gate 5 is consistent with its inputs because 1 · 1 · 1 = 1. Therefore, one of the inputs to gate 5 must be wrong.
3. Either the output of gate 3 is wrong, or the A or B input to gate 5 is wrong. Because C'D + CD' = 0, the output of gate 3 is wrong.
4. The output of gate 3 is not consistent with the outputs of gates 1 and 2 because 0 + 0 ≠ 1. Therefore, either one of the inputs to gate 3 is connected wrong, gate 3 is defective, or one of the input connections to gate 3 is defective.



Multiplexers, Decoders and Programmable Logic Devices

Objective

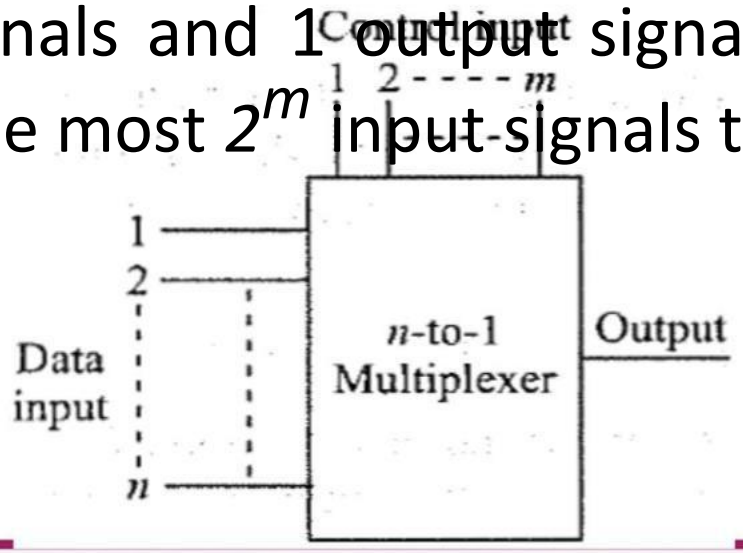
1. Explain the function of a multiplexer. Implement a multiplexer using gates.
2. Explain the operation of three-state buffers. Determine the resulting output when three-state buffer outputs are connected together. Use three-state buffers to multiplex signals onto a bus.
3. Explain the operation of a decoder and encoder. Use a decoder with added gates to implement a set of logic functions. Implement a decoder or priority encoder using gates.

Inroduction

- **Use of more complex integrated circuits (ICs) in logic design**
- Integrated circuits may be classified
  - Small-scale integration (SSI), Medium-scale integration (MSI), Large-scale integration (LSI), or Very-large-scale integration (VLSI), **depending on the number of gates in each integrated circuit package** and the type of function performed.
- **SSI** functions include NAND, NOR, AND, and OR gates, inverters, and flip-flops. SSI integrated circuit packages typically contain one to four gates, six inverters, or one or two flip-flops.
- **MSI integrated circuits, such as adders, multiplexers, decoders, registers, and counters, perform more complex functions. Such integrated circuits typically contain the equivalent of 12 to 100 gates in one package.**
- **More complex functions such as memories and microprocessors are classified as LSI or VLSI integrated circuits.**
- **LSI** integrated circuit generally contains 100 to a few thousand gates in a single package, and a
- **VLSI** integrated circuit contains several thousand gates or more.

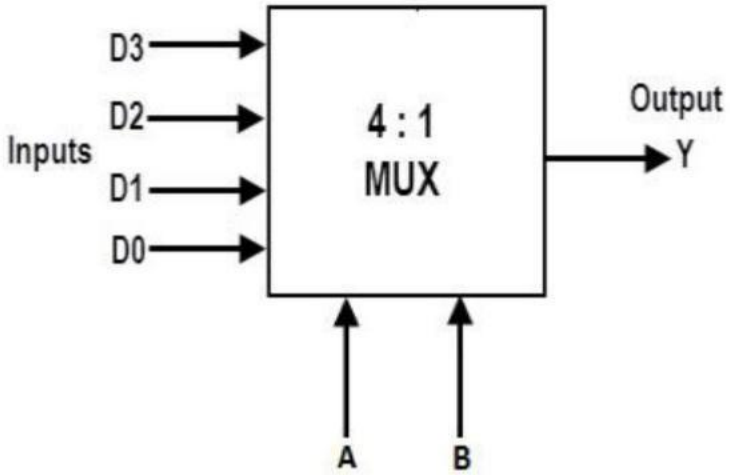
MULTIPLEXERS-1

- *Multiplex* means *many into one.*
- A multiplexer (also called data selector) is a circuit with many inputs but only one output. By applying control signals (Select Input), can steer any input to the output.
- Below shows the block diagram of MUX. The circuit has *n* input signals, *m* control signals and 1 output signal. Note that, *m* control signals can select at the most 2^m input-signals thus $n \leq 2^m$.



MULTIPLEXERS-2

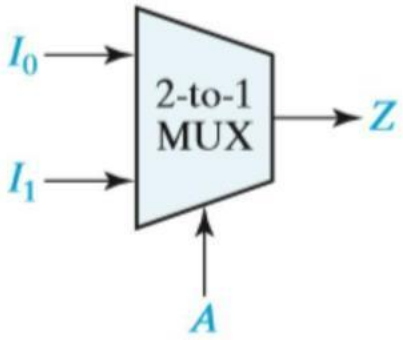
- The block diagram of a 4-to-1 multiplexer is shown below and its truth table.
- Depending on control inputs *A,* B one of the four inputs *Do* to *D3* is steered to output *Y.*

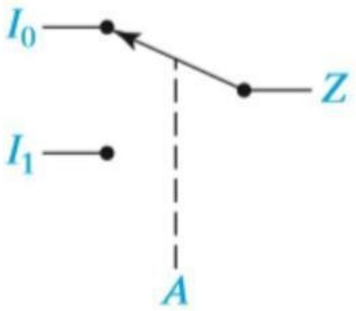


A	*B*	*Y*
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

MULTIPLEXERS-2

- 2-to-1 Multiplexer and Switch Analog
- When the control input A is 0, the switch is in the upper position and the MUX output is Z = I_0;
- When A is 1, the switch is in the lower position and the MUX output is Z = I_1.
- In other words, a MUX acts like a switch that selects one of the data inputs (I_0 or I_1) and transmits it to the output.

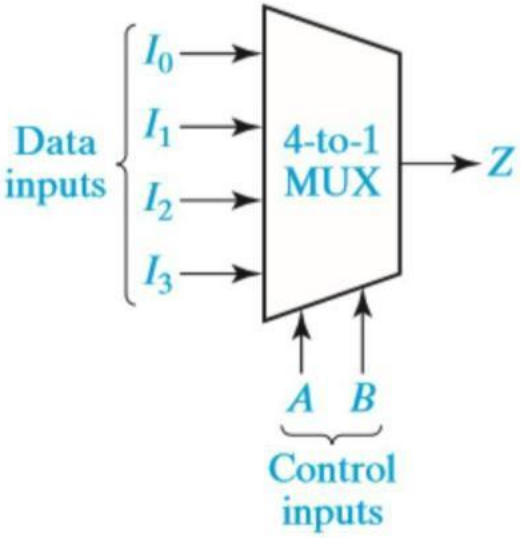




- $Z=A'I_0+AI_1$

MULTIPLEXERS-3

- 4-to-1 MUX logic circuit
- Logic equation of this circuit is a SOP representation.



- $Z = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$

MULTIPLEXERS-3

- 4-to-1 MUX logic circuit
- Logic equation of this circuit is a SOP representation.

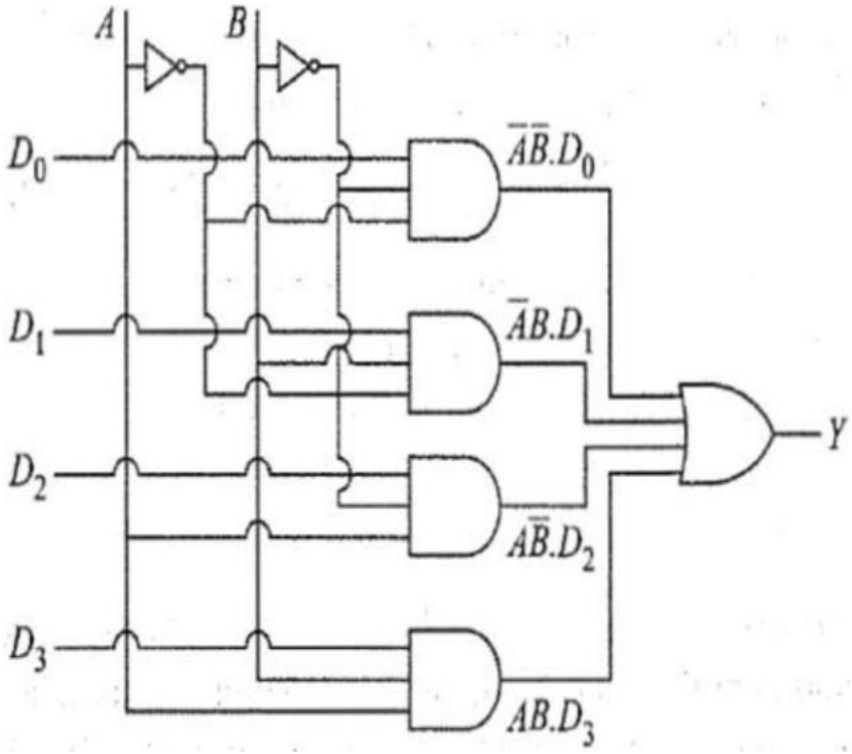
$$Y = A'B'.D_0 + A'B.D_1 + AB'.D_2 + AB.D_3$$

If $A = 0, B = 0,$

$$Y = 0'0'.D_0 + 0'.0.D_1 + 0.0'.D_2 + 0.0.D_3$$

$$Y = 1.1.D_0 + 1.0.D_1 + 0.1.D_2 + 0.0.D_3$$

$$Y = D_0$$



A	B	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

MULTIPLEXERS-4

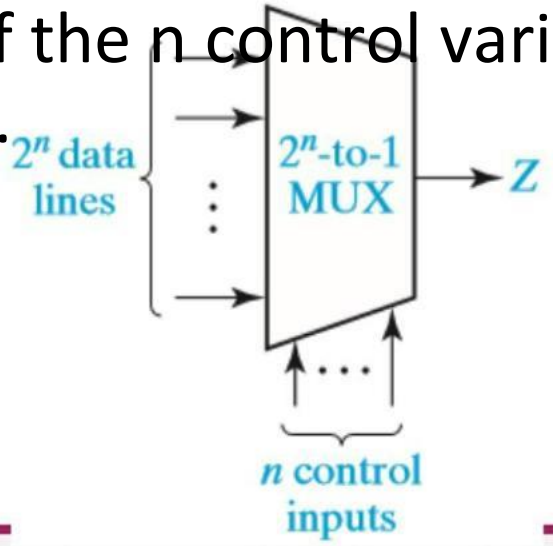
- In other words, for *AB* = 00, the first AND gate to which *D0* is connected remains active and equal to *Do* and all other AND gate are inactive with output held at logic 0.
- Thus, multiplexer output *Y* is same as *D0.* If *D0 =0,* Y=0 and if D0 = 1, *Y*= 1.
- Commercial multiplexers ICs come in integer power of 2, e.g. 2-to-1, 4-to-1, 8-to- 1, 16-to-1 multiplexers.
- Write the 2 to 1 MUX block diagram, equation and it truth table.

MULTIPLEXERS-4

- In general, a multiplexer with n control inputs can be used to select any one of 2^n data inputs.
- The general equation for the output of a MUX with n control inputs and 2^n data inputs is

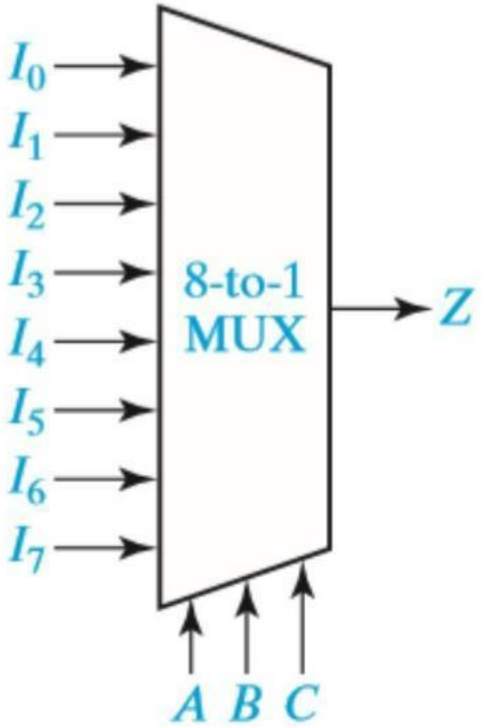
$$Z = \sum_{k=0}^{2^n} m_k I_k$$

- where m_k is a minterm of the n control variables and I_k is the corresponding data input.



MULTIPLEXERS-4

- 8-to-1 multiplexer

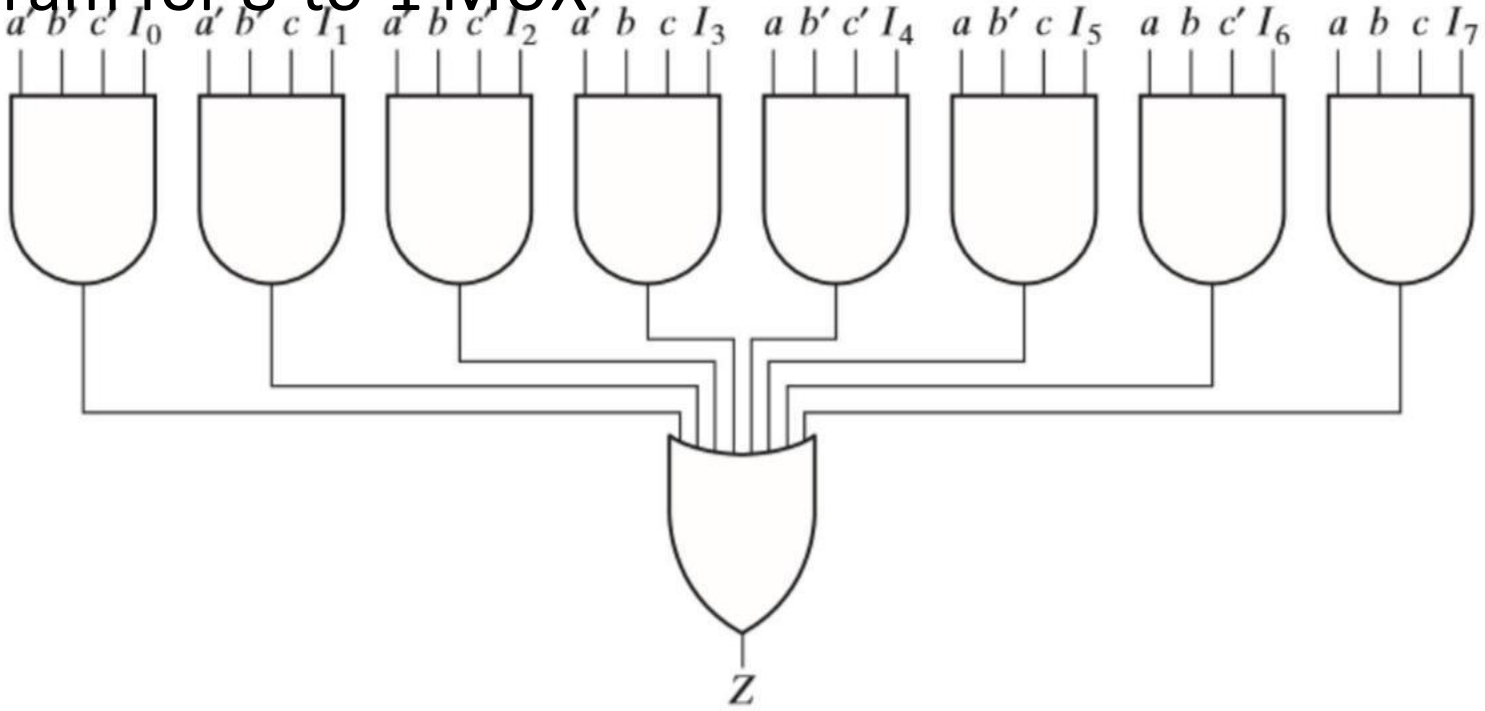


Select/Control Input			Output
A	B	C	Z
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

- $Z = A'B'C'I_0 + A'B'CI_1 + A'BC'I_2 + A'BCI_3 + AB'C'I_4 + AB'CI_5 + ABC'I_6 + ABCI$

MULTIPLEXERS-4

- Logic Diagram for 8-to-1 MUX

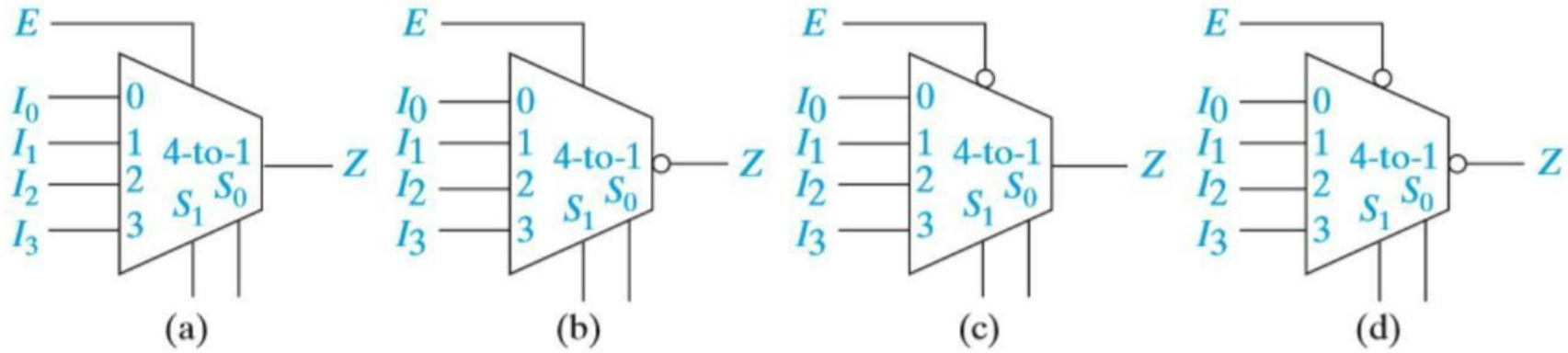


MULTIPLEXERS-4

- **Invert the data inputs**
  - If the OR gate in Figure is replaced by a NOR gate, then the 8-to-1 MUX inverts the selected input.
- **Another type of multiplexer has an additional input called an enable.**
  - The 8-to-1 MUX in Figure can be modified to include an enable by changing the AND gates to five-input gates.
  - The enable signal E is connected to the fifth input of each of the AND gates. Then, if E = 0, Z = 0 independent of the gate inputs I_i and the select inputs a, b, and c.
  - However, if E = 1, then the MUX functions as an ordinary 8-to-1 multiplexer.
  - The terminology used for the MUX output, i.e., active high and active low, can be used for the enable as well.

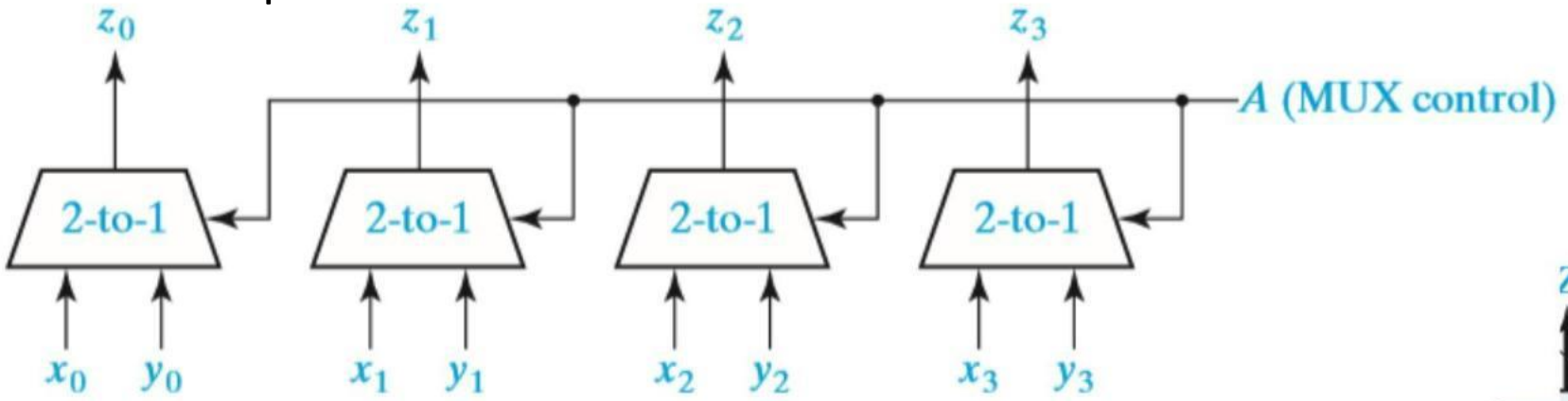
MULTIPLEXERS-4

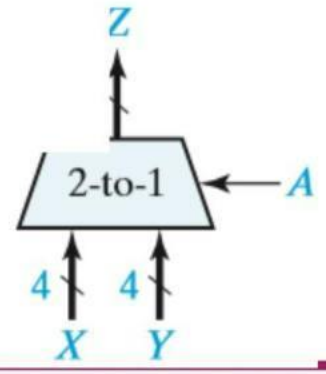
- **Active-High, Active-Low Enable and Output Combinations**



MULTIPLEXERS-4

- **Quad Multiplexer Used to Select Data**
- A quadruple 2-to-1 MUX is used to select one of two 4-bit data words.
  - If the control is A = 0, the values of x0, x1, x2, and x3 will appear at the z0, z1, z2, and z3 outputs
  - If A = 1, the values of y0, y1, y2, and y3 will appear at the outputs.



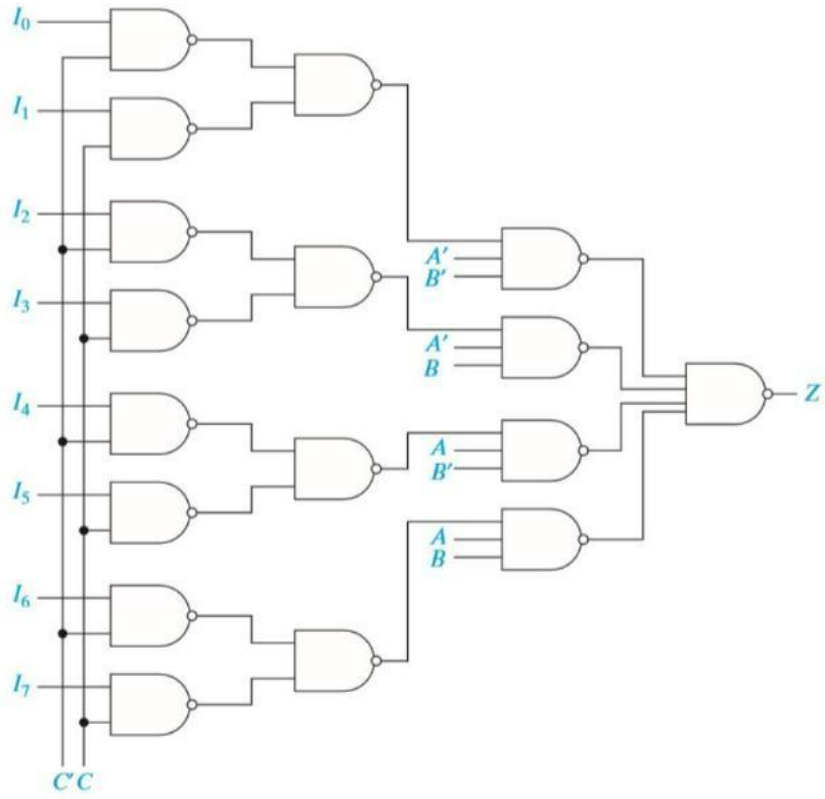


MULTIPLEXERS-4

- Implement using NAND gate

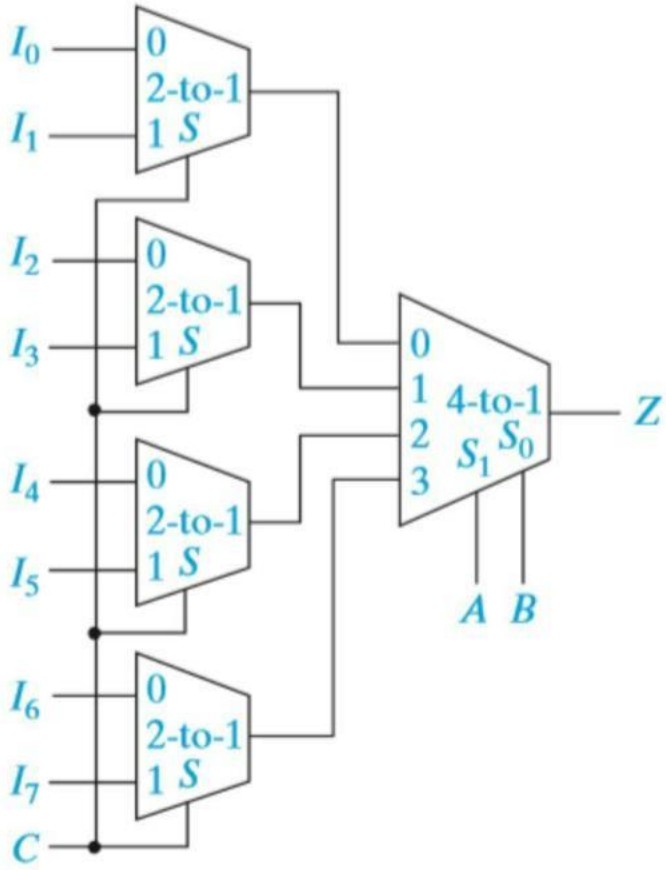
- Z = A'B'C'I0 + A'B'CI1 + A'BC'I2 + A'BCI3 + AB'C'I4 + AB'CI5 + ABC'I6 + ABCI7
- Implement using NAND gate
- Z = A'B'(C'I0 + CI1) + A'B(C'I2 + CI3) + AB'(C'I4 + CI5) + AB(C'I6+CI7)

-



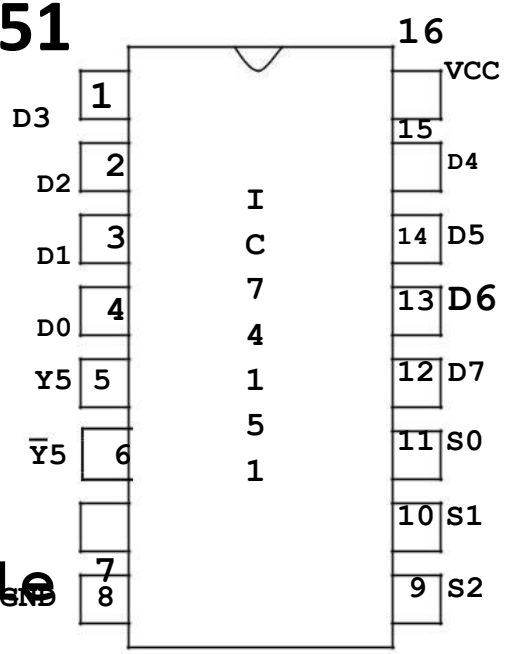
MULTIPLEXERS-4

- Note that the data inputs are connected to four 2-to-1 MUXs with C as the select line, and the outputs of these 2-to-1 MUXs are connected to a 4-to-1 MUX with A and B as the select lines. Figure shows this in block diagram form.



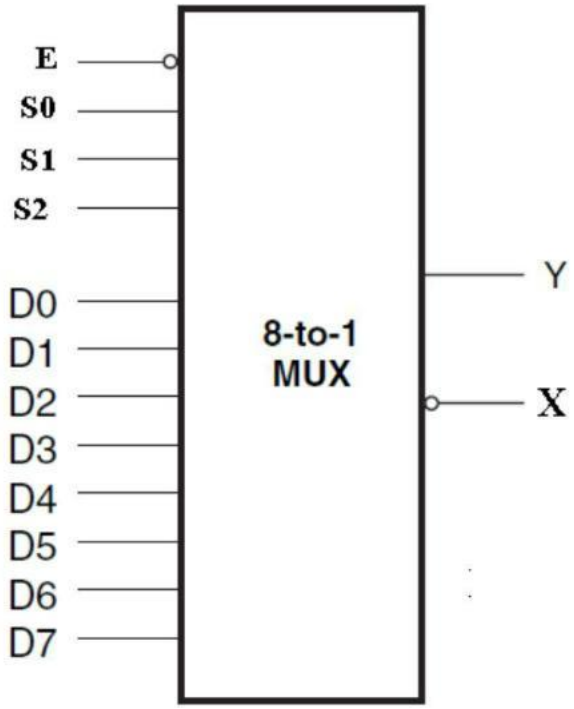
MULTIPLEXERS-6

- **IC 74151**



(0)

-Enable



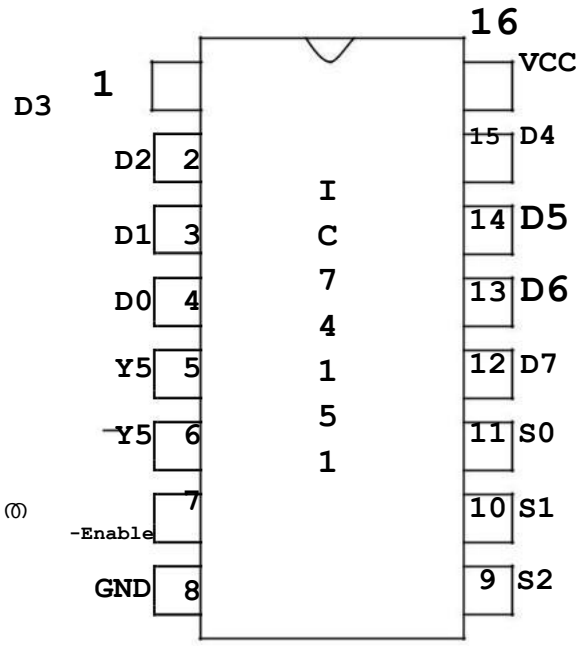
Inputs				Output	
Select			Enable		
S2	S1	S0	E	Y	X
X	X	X	H	L	H
L	L	L	L	D0	$\overline{D0}$
L	L	H	L	D1	$\overline{D1}$
L	H	L	L	D2	$\overline{D2}$
L	H	H	L	D3	$\overline{D3}$
H	L	L	L	D4	$\overline{D4}$
H	L	H	L	D5	$\overline{D5}$
H	H	L	L	D6	$\overline{D6}$
H	H	H	L	D7	$\overline{D7}$

E : ENABLE input
S0, S1, S2 : Select inputs
D0-D7 : Data inputs
Y, X : outputs

MULTIPLEXERS-7

- **Lab Experiment**
- Given Logic expression is *f*(a,b,c,d)=Σm(2,3,4,5,13,15)+d(8,9,10,11)
  Use the Entered Variable Map (EVM) technique.

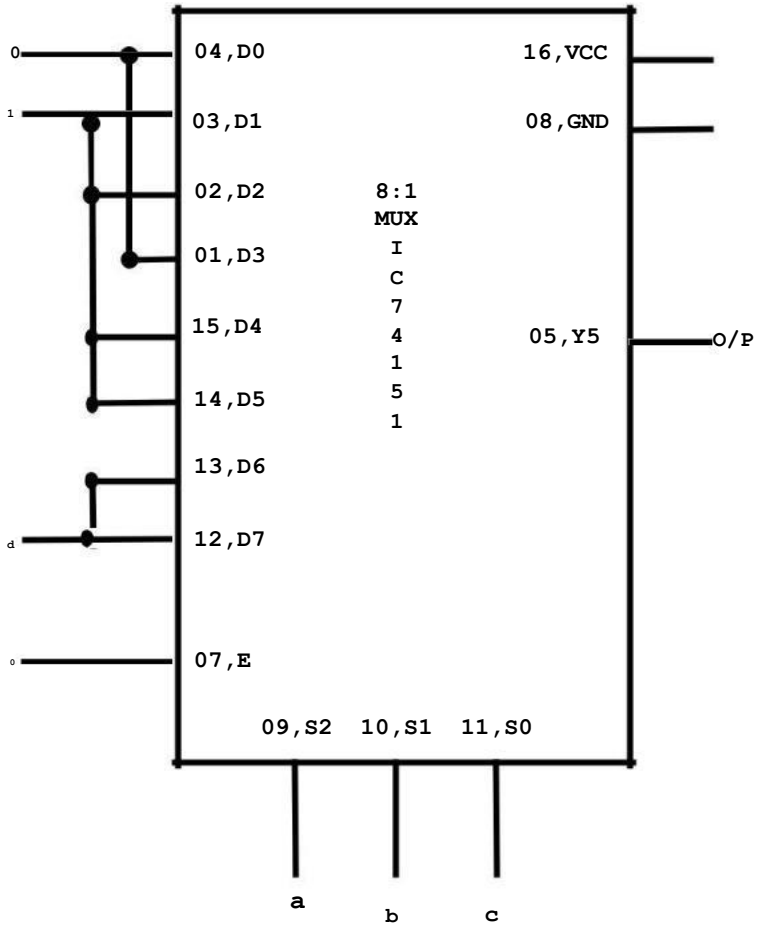
		Select Inputs					Output Y5
Minterm in decimals		a	b	c	d	f	EVM Entry
0	0 1	0 0	0 0	0 0	0 1	0 0	0 (D0)
1	2 3	0 0	0 0	1 1	0 1	1 1	1 (D1)
2	4 5	0 0	1 1	0 0	0 1	1 1	1 (D2)
3	6 7	0 0	1 1	1 1	0 1	0 0	0 (D3)
4	8 9	1 1	0 0	0 0	0 1	X X	X (D4)
5	10 11	1 1	0 0	1 1	0 1	X X	X (D5)
6	12 13	1 1	1 1	0 0	0 1	0 1	d (D6)
7	14 15	1 1	1 1	1 1	0 1	0 1	d (D7)



MULTIPLEXERS-8

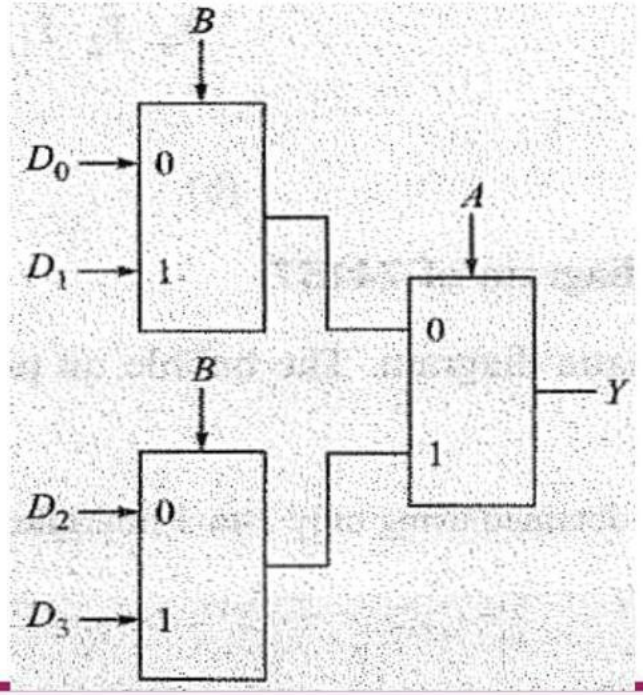
- Final Circuit diagram

		Select Inputs					Output Y5
Minterm in decimals		a	b	c	d	f	EVM Entry
0	0	0	0	0	0	0	0 (D0)
	1	0	0	0	1	0	
1	2	0	0	1	0	1	1 (D1)
	3	0	0	1	1	1	
2	4	0	1	0	0	1	1 (D2)
	5	0	1	0	1	1	
3	6	0	1	1	0	0	0 (D3)
	7	0	1	1	1	0	
4	8	1	0	0	0	X	X (D4)
	9	1	0	0	1	X	
5	10	1	0	1	0	X	X (D5)
	11	1	0	1	1	X	
6	12	1	1	0	0	0	d (D6)
	13	1	1	0	1	1	
7	14	1	1	1	0	0	d (D7)
	15	1	1	1	1	1	



MULTIPLEXERS-9

- Show how 4-to-1 multiplexer can be obtained using only 2-to-1 multiplexer.
- Logic equation for 2-to-1Multiplexer: Y= A’D0 + AD1
- Logic equation for 4-to-1 Multiplexer:
  Y = A'B' D0+ A'BD1 + AB' D2 + ABD3
- This can be rewritten as, Y= A'(B'D0 + BD1) +A (B'D2 + BD3)



MULTIPLEXERS-10

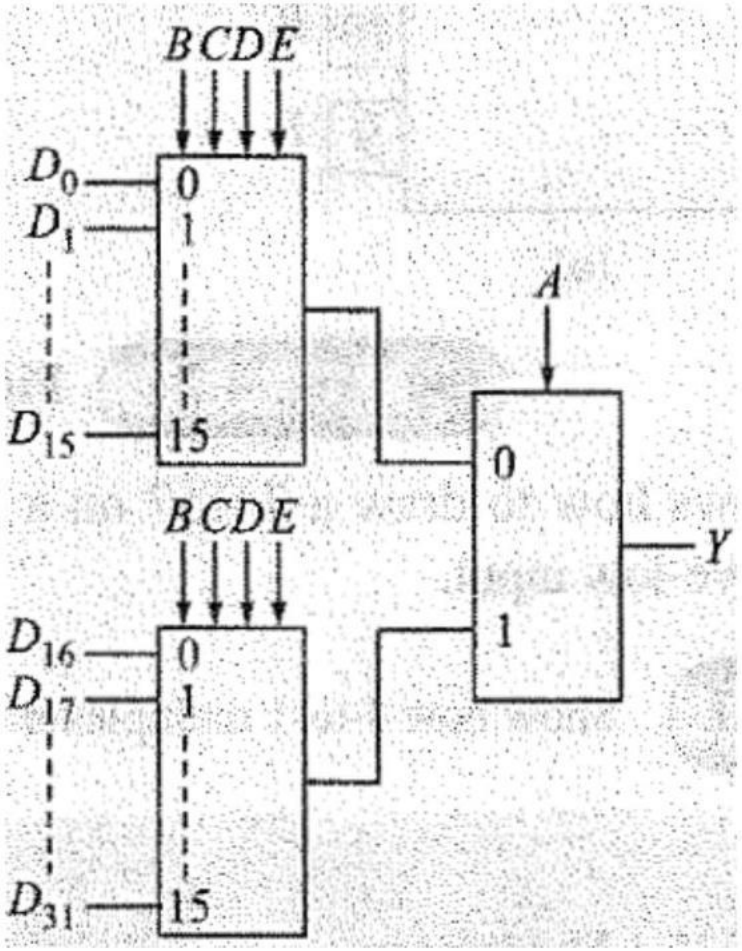
- Realize *Y=A'B + B'C' + ABC* using an 8-to-1 multiplexer.
  - First we express *Y* as a function of minterms of three variables. Thus
  - *Y = A'B + B'C' + ABC*
  - *Y =A'B(C' + C)+B'C'(A'* + A)+ *ABC* *[As,X+X'=* I]
  - *Y = A'B'C' + A'BC' + A'BC + AB'C' + ABC*
  - Comparing this with equation of 8 to 1 multiplexer, we find by substituting D0 = *D2 = D3 =D4 = D7* = 1 and *D1 = D5 = D6* = 0.

MULTIPLEXERS-11

- Can it be realized *Y =A'B + B'C' + ABC* equation with a 4-to-1 multiplexer?
  - The 4-to-1 multiplexer generates 4 minterms for different combinations of *AB.* We rewrite given logic equation in such a way that all these terms are present in the equation.
  - *Y =A'B+B'C' +ABC*
  - *Y =A'B+ B'C'(A' +A)+ ABC* *[As,X +X' =* I]
  - *Y =A'B'.C' + A'B.1 +AB'.C' + AB.C*
  - Compare above with equation of a 4-to-1 multiplexer. We see *D0= C',* D1 = 1, *D2 = C'* and D3 = C generate the given logic function.

MULTIPLEXERS-12

- Design a 32-to-1 multiplexer using two 16-to-1 multiplexers and one 2-to-1 multiplexer.
  - A 32-to-1 multiplexer requires $\log_2 32$ =5 select lines say, *ABCDE*. The lower 4 select lines *BCDE choose* 16-to-1 multiplexer outputs. The 2-to-1 multiplexer chooses one of the output of two 16-to-1 multiplexers depending on what appears in the 5th select line, *A*.

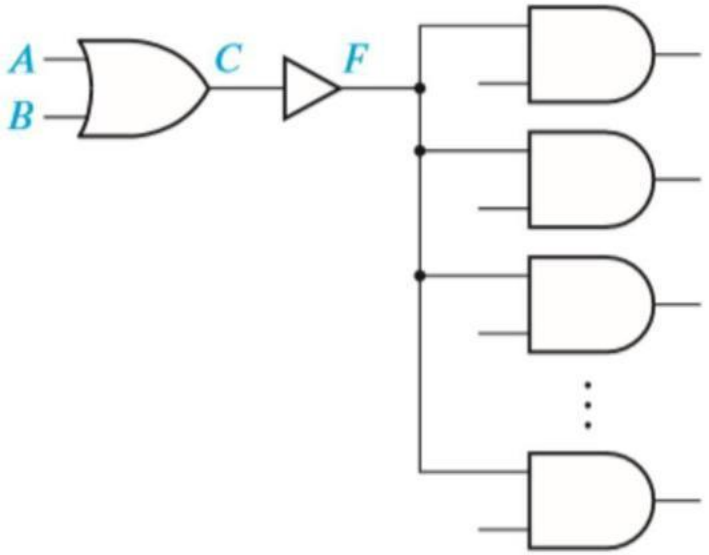


Three-State Buffers

- A gate output can only be connected to a limited number of other device inputs without degrading the performance of a digital system.
- A simple buffer may be used to increase the driving capability of a gate output.

Three-State Buffers

- A buffer connected between a gate output and several gate inputs.
- Because no bubble is present at the buffer output, this is a noninverting buffer, and the logic values of the buffer input and output are the same, that is, F=C.



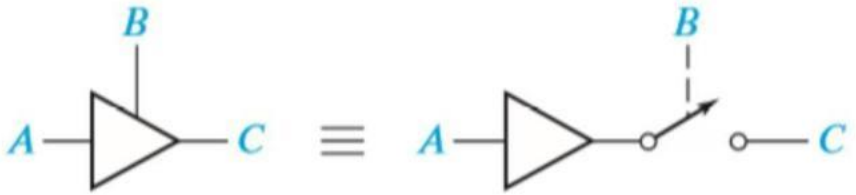
Three-State Buffers

- Normally, a logic circuit will not operate correctly if the outputs of two or more gates or other logic devices are directly connected to each other.
  - For example, if one gate has a 0 output (a low voltage) and another has a 1 output (a high voltage), when the Gate outputs are connected together the resulting output voltage may be some intermediate value that does not clearly represent either a 0 or a 1.
  - In some cases, damage to the gates
- **Use of three-state logic permits the outputs of two or more gates or other logic devices to be connected together.**

Three-State Buffers

- **Three-state buffers are also called tri-state buffers.**

- **When B= 1 then C = A**
- **When B= 0 then C = Z (Hi-Z high-impedance)**



Three-State Buffers

- **Four Kinds of Three-State Buffers**

B	A	C
0	0	Z
0	1	Z
1	0	0
1	1	1

(a)

B	A	C
0	0	Z
0	1	Z
1	0	1
1	1	0

(b)

B	A	C
0	0	0
0	1	1
1	0	Z
1	1	Z

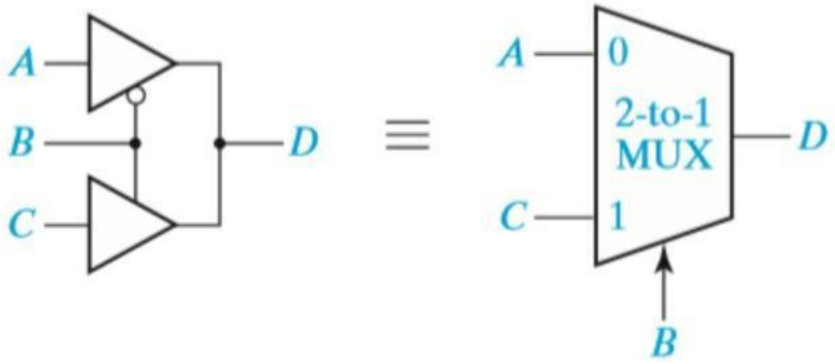
(c)

B	A	C
0	0	1
0	1	0
1	0	Z
1	1	Z

(d)

Three-State Buffers

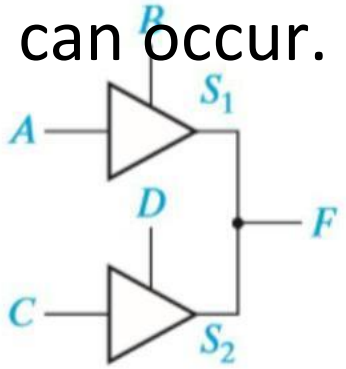
- **Data Selection Using Three-State Buffers**



- **D=B'A+BC**

Three-State Buffers

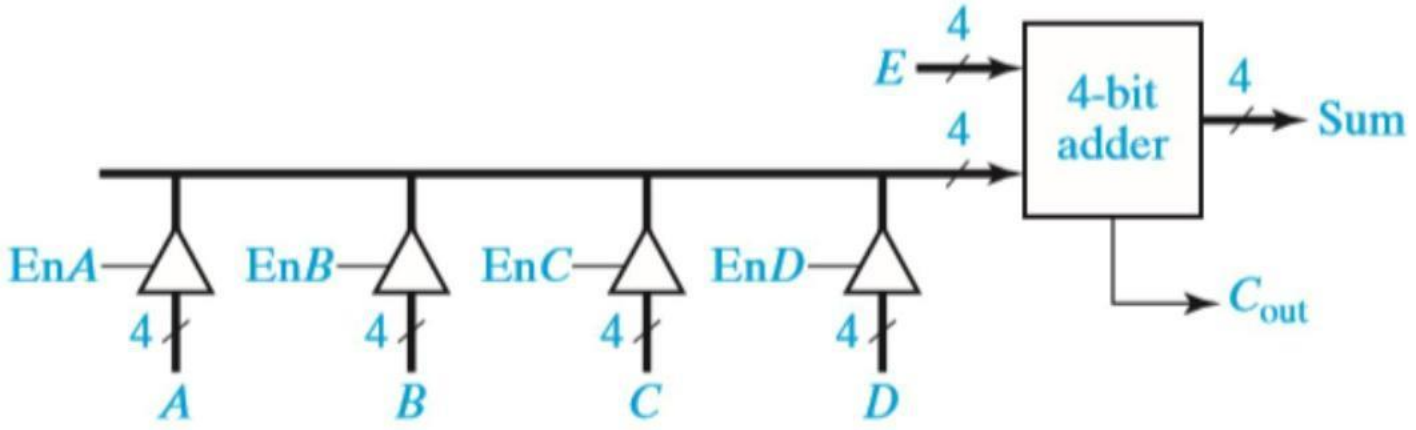
- **Circuit with Two Three-State Buffers**
- When we connect two three-state buffer outputs together, as shown in Figure, if one of the buffers is disabled (output = Z), the combined output F is the same as the other buffer output.
- If both buffers are disabled, the output is Z. If both buffers are enabled, a conflict can occur.



S_1 \ S_2	X	0	1	Z
X	X	X	X	X
0	X	0	X	0
1	X	X	1	1
Z	X	0	1	Z

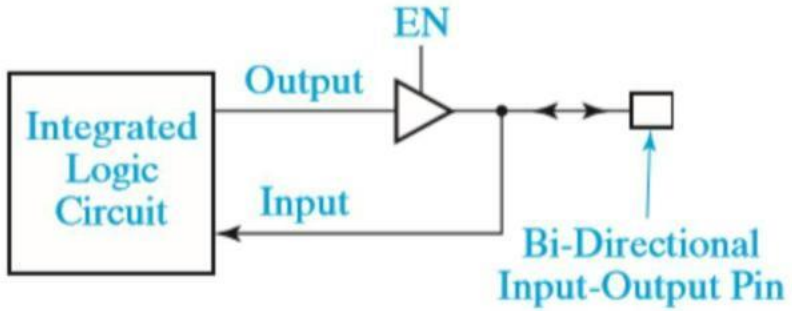
Three-State Buffers

- A multiplexer may be used to select one of several sources to drive a device input.
- For example, if an adder input must come from four different sources, a 4-to-1 MUX may be used to select one of the four sources.
- An alternative is to set up a three-state bus, using three-state buffers to select one of the sources.



Three-State Buffers

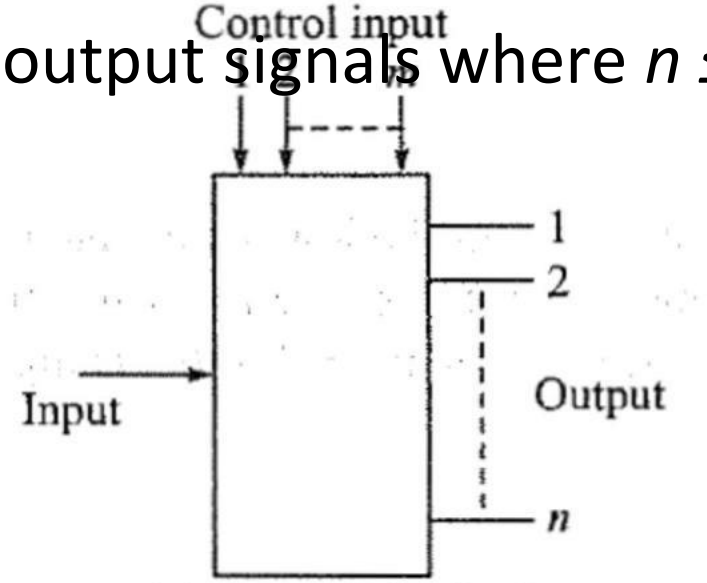
- **Integrated Circuit with Bi-Directional Input-Output Pin**
- Bi-directional means that the same pin can be used as an input pin and as an output pin, but not both at the same time.
- To accomplish this, the circuit output is connected to the pin through a three-state buffer, as shown in Figure.



- When the buffer is enabled, the pin is driven with the output signal. When the buffer is disabled, an external source can drive the input pin.

DEMULTIPLEXERS-1

- Demultiplex means one into many.
- A *demultiplexer* is a logic circuit with one input and many outputs. By applying control signals, can steer the input signal to one of the output lines. The circuit has 1 input signal, *m* control or select signals and *n* output signals where $n \leq 2^m$.

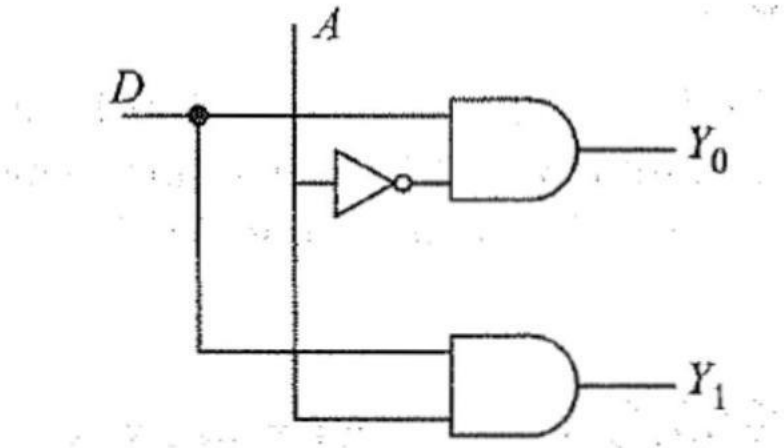


Demultiplexer block diagram

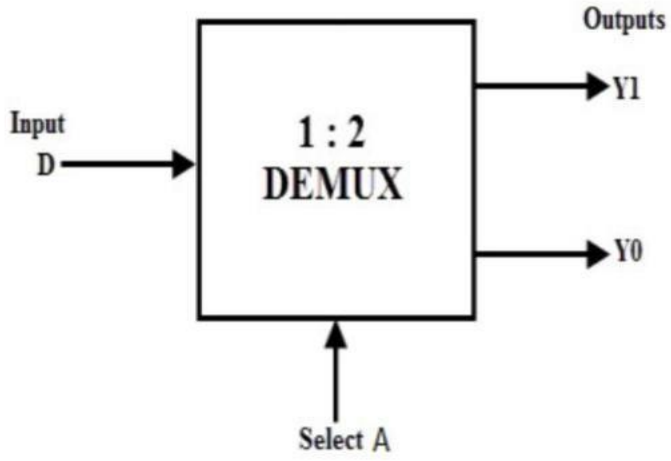
DEMULTIPLEXERS-2

- For 1 to 2 DMUX
- The logic equation
- Y0=DA’ Y1=DA

Input	Select Input	Output	
D	A	Y1	Y0
D	0	0	D
D	1	D	0

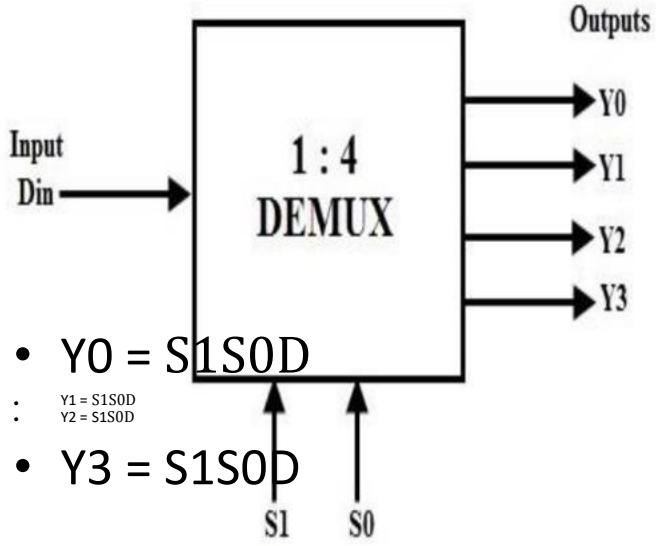


Logic circuit of 1-to-2 demultiplexer



DEMULTIPLEXERS-3

- Write block diagram, truth table and logic equation of 1 to 4 DMUX.



Data Input	Select Inputs		Outputs			
D	S_1	S_0	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	D
D	0	1	0	0	D	0
D	1	0	0	D	0	0
D	1	1	D	0	0	0

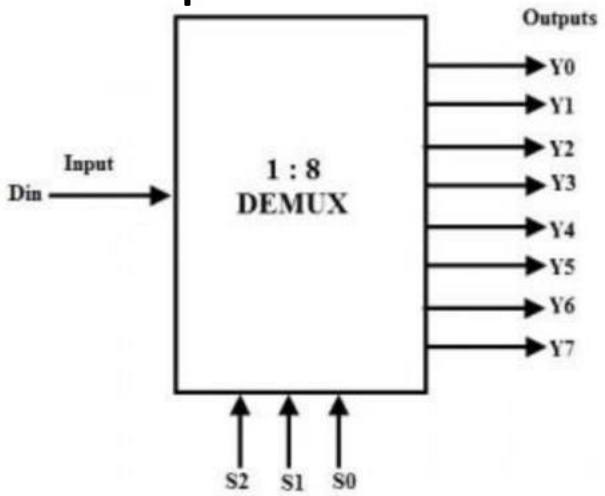
- Y0 = S1S0D

Y1 = S1S0D
Y2 = S1S0D

- Y3 = S1S0D

DEMULTIPLEXERS-4

- Write block diagram, truth table and log equation of 1 to 8 DMUX



Data Input	Select Inputs			Outputs							
D	S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	0	0	0	0	0	D
D	0	0	1	0	0	0	0	0	0	D	0
D	0	1	0	0	0	0	0	0	D	0	0
D	0	1	1	0	0	0	0	D	0	0	0
D	1	0	0	0	0	0	D	0	0	0	0
D	1	0	1	0	0	D	0	0	0	0	0
D	1	1	0	0	D	0	0	0	0	0	0
D	1	1	1	D	0	0	0	0	0	0	0

$Y0 = D\,\overline{S2}\,\overline{S1}\,\overline{S0}$

$Y1 = D\,\overline{S2}\,\overline{S1}\,S0$

$Y2 = D\,\overline{S2}\,S1\,\overline{S0}$

$Y3 = D\,\overline{S2}\,S1\,S0$

$Y4 = D\,S2\,\overline{S1}\,\overline{S0}$

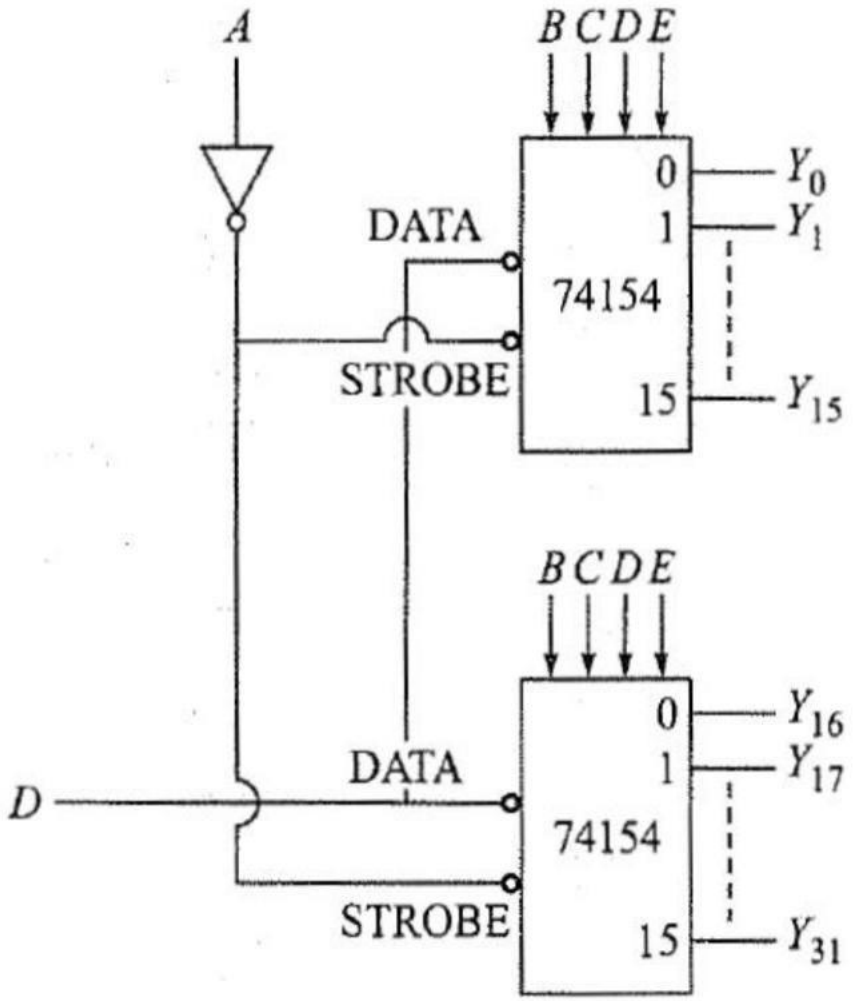
$Y5 = D\,S2\,\overline{S1}\,S0$

$Y6 = D\,S2\,S1\,\overline{S0}$

$Y7 = D\,S2\,S1\,S0$

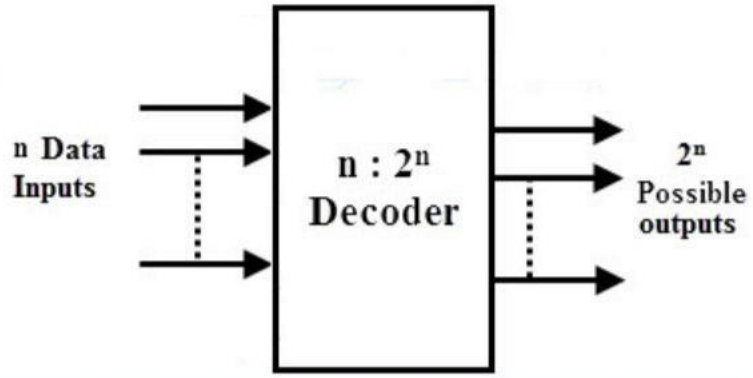
DEMULTIPLEXERS-5

- Show how two 1-to-16 demultiplexers can be connected to get a 1-to-32 demultiplexer.
  - A 1-to-32 demultiplexer has *5* select veriable ABCDE. Four of them (BCDE) are fed to two 1-to-16 DMUX. Fifth A used to select the one of these DMUX



DECODER-1

- **A *decoder* is similar to a demultiplexer, with one exception-there is no data input.**
- *Also called binary-to-decimal decoder.*
- The name decoder means translating of coded information from one format into another.
- A binary decoder is a multi-input, multi-output combinational circuit that converts a binary code of n input lines into a one out of 2^n output code.
- Depending on the number of input lines, the inputs of a binary code can be 2-bit or 3-bit or 4-bit codes. Upon the availability of 2^n lines, it activates the one of its output by deactivating (making logic 0) all other input whenever it receives n inputs.

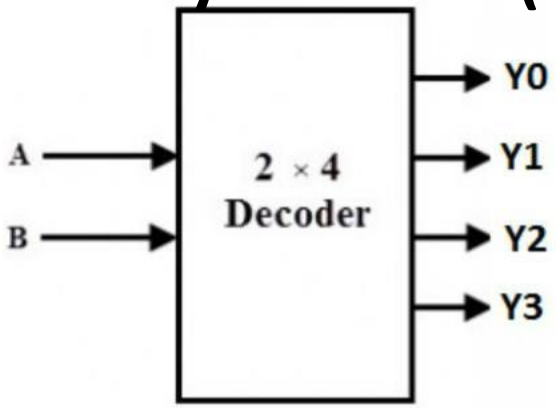


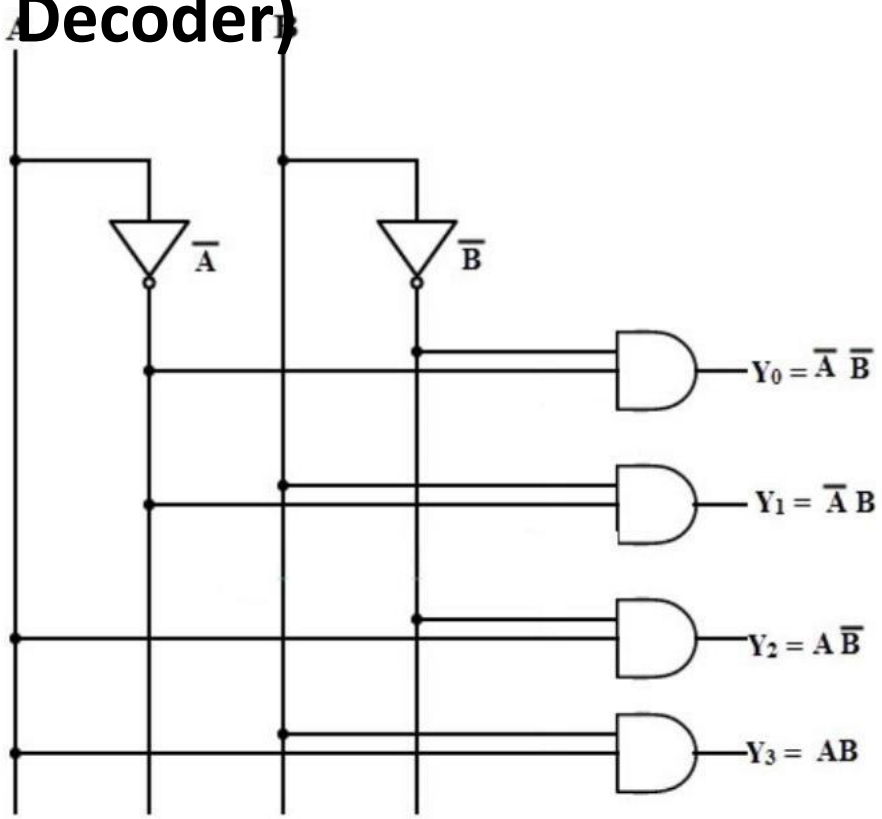
DECODER-2

- The most commonly used practical binary decoders are 2-to-4 decoder, 3-to-8 decoder and 4-to-16 line binary decoder.
- 2-to-4 decoder also called 1 of 4
- 3-to-8 decoder also called 1 of 8
- 4-to-16 line binary decoder also called 1 of 16

DECODER-3

- **2-to-4 Binary Decoder (1 of 4 Decoder)**

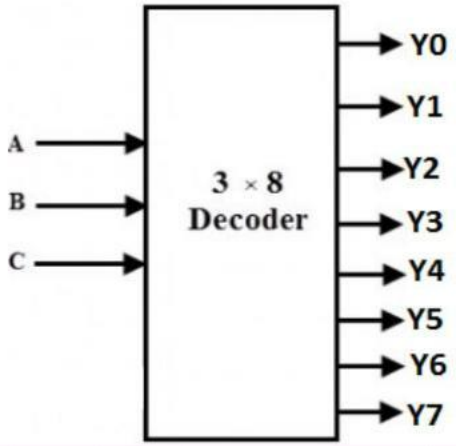




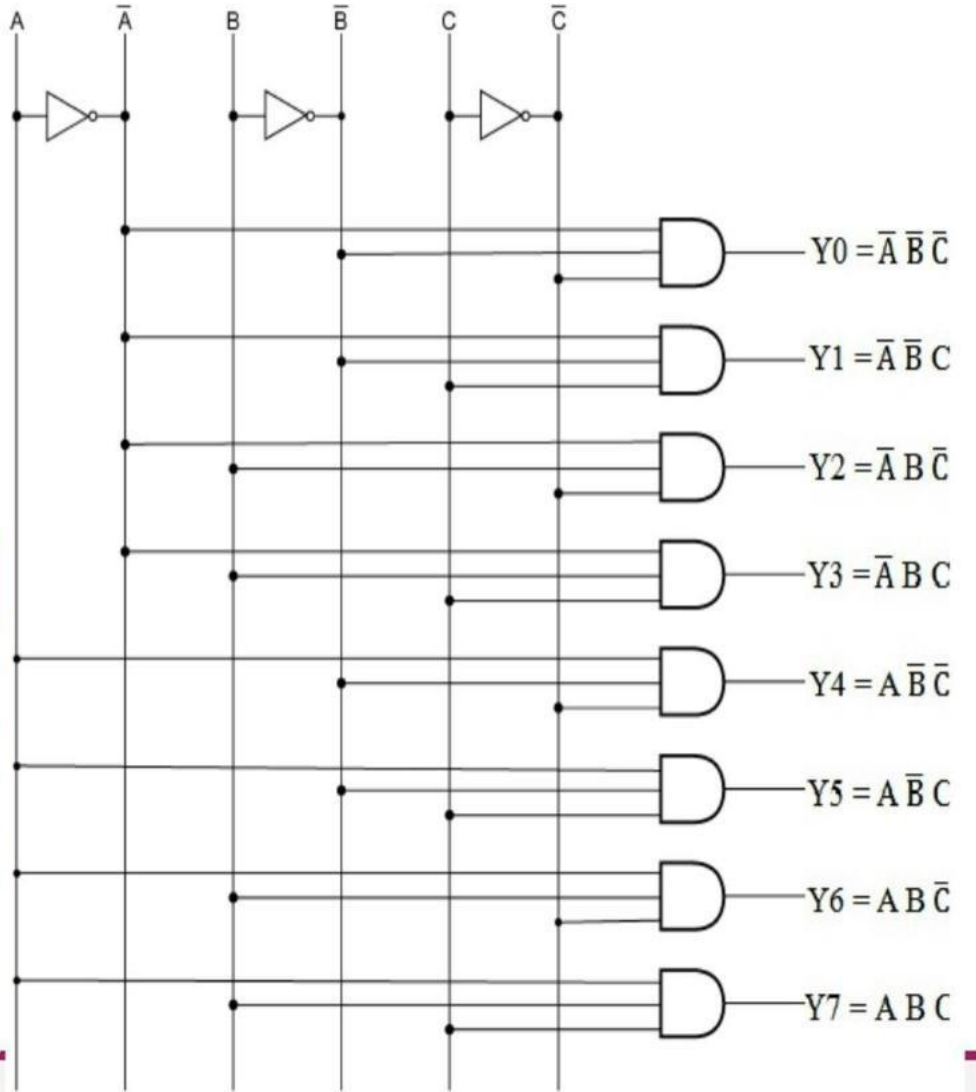
Input		Output			
A	B	Y0	Y1	Y2	Y3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

DECODER-4

- **3-to-8 Binary Decoder (1 of 8 Decoder)**



A	B	C	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
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

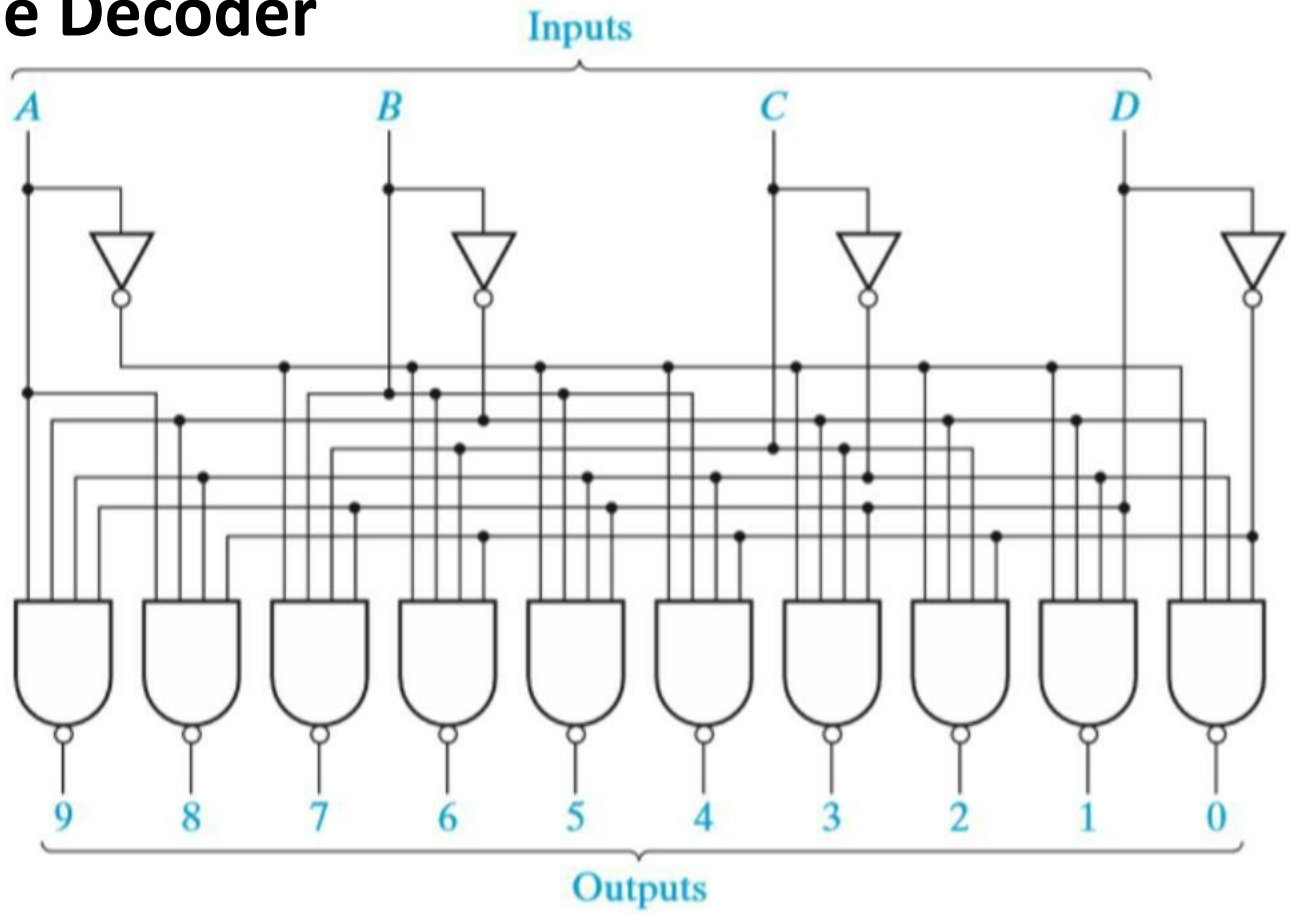


DECODER-5

- This decoder generates all of the minterms of the three input variables. Exactly one of the output lines will be 1 for each combination of the values of the input variables.

DECODER-5

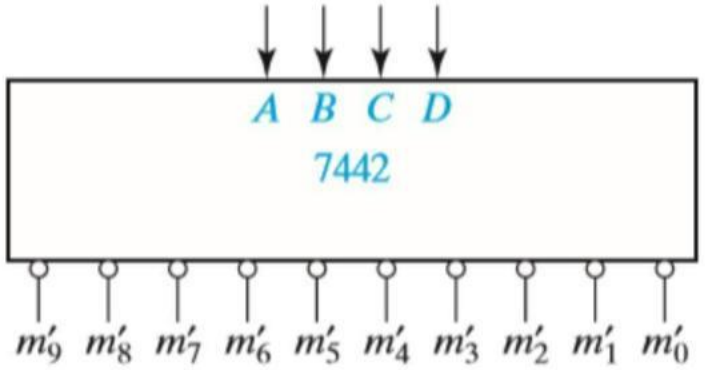
- **A 4-to-10 Line Decoder**



(a) Logic diagram

DECODER-5

- **A 4-to-10 Line Decoder**



(b) Block diagram

BCD Input				Decimal Output									
A	B	C	D	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0
1	0	1	0	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

(c) Truth Table

DECODER-5

- **A 4-to-10 Line Decoder**
- This decoder has inverted outputs (indicated by the small circles).
- For each combination of the values of the inputs, exactly one of the output lines will be 0.
  - When a binary-coded-decimal (BCD) digit is used as an input to this decoder, one of the output lines will go low to indicate which of the 10 decimal digits is present.

DECODER-6

- In general, an n-to-2^n line decoder generates all 2^n minterms (or maxterms) of the n input variables. The outputs are defined by the equations

$$y_i = m_i = M_i', \quad i = 0 \text{ to } 2n - 1 \qquad \text{(noninverted outputs)}$$

or

$$y_i = m_i' = M_i, \quad i = 0 \text{ to } 2n - 1 \qquad \text{(inverted outputs)}$$

  - where mi is a minterm of the n input variables and Mi is a maxterm.

DECODER-6

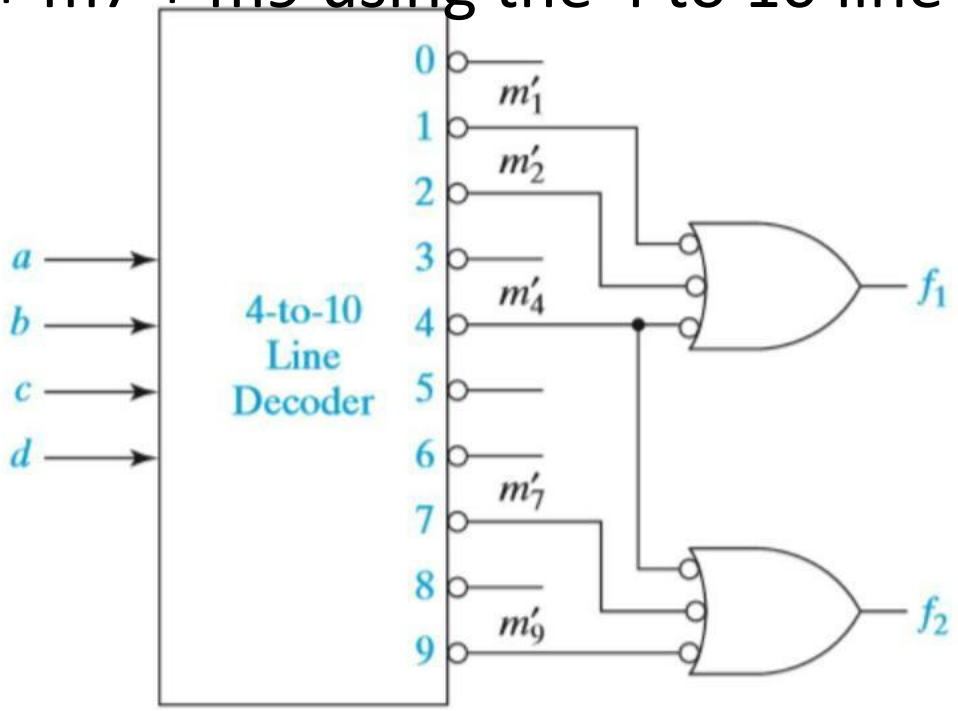
- **Applications of Decoders**
- Decoders are greatly used in applications where the particular output or group of outputs to be activated only on the occurrence of a specific combination of input levels.
- **Binary to Decimal Decoder**
- Decoders are used to get the decimal digit corresponding to a specific input combination. A BCD number needs 4 binary digits to represent the 0 to 9 decimal digits, thus it consists of 4 input lines. It consists of 10 output lines corresponding to 0 to 9 decimal digits. (1 of 10 line decoder)
- **Address Decoders**
- Amongst its many uses, a decoder is widely used to decode the particular memory location in the computer memory system. Decoders accept the address code generated by the CPU which is a combination of address bits for a specific location in the memory. In a memory system, there are several memory ICs are combined and each one has their unique address to distinguish from other memory locations. In such cases a decoder built in the memory ICs circuitry, is used to select a memory IC in response to a range of addresses by decoding the most significant bits of the systems address, thereby a particular memory location or IC is selected.
- **Instruction Decoder**
- Another application of the decoder can be found in the control unit of the central processing unit. This decoder is used to decode the program instructions in order to activate the specific control lines such that different operations in the ALU of the CPU are carried out.

DECODER-6

- Because an n-input decoder generates all of the minterms of n variables, n- variable functions can be realized by ORing together selected minterm outputs from a decoder.
- If the decoder outputs are inverted, then NAND gates can be used to generate the functions.

DECODER-6

- Realize f1(a, b, c, d) = m1 + m2 + m4 and
- f2(a, b, c, d) = m4 + m7 + m9 using the 4 to 10 line decoder.



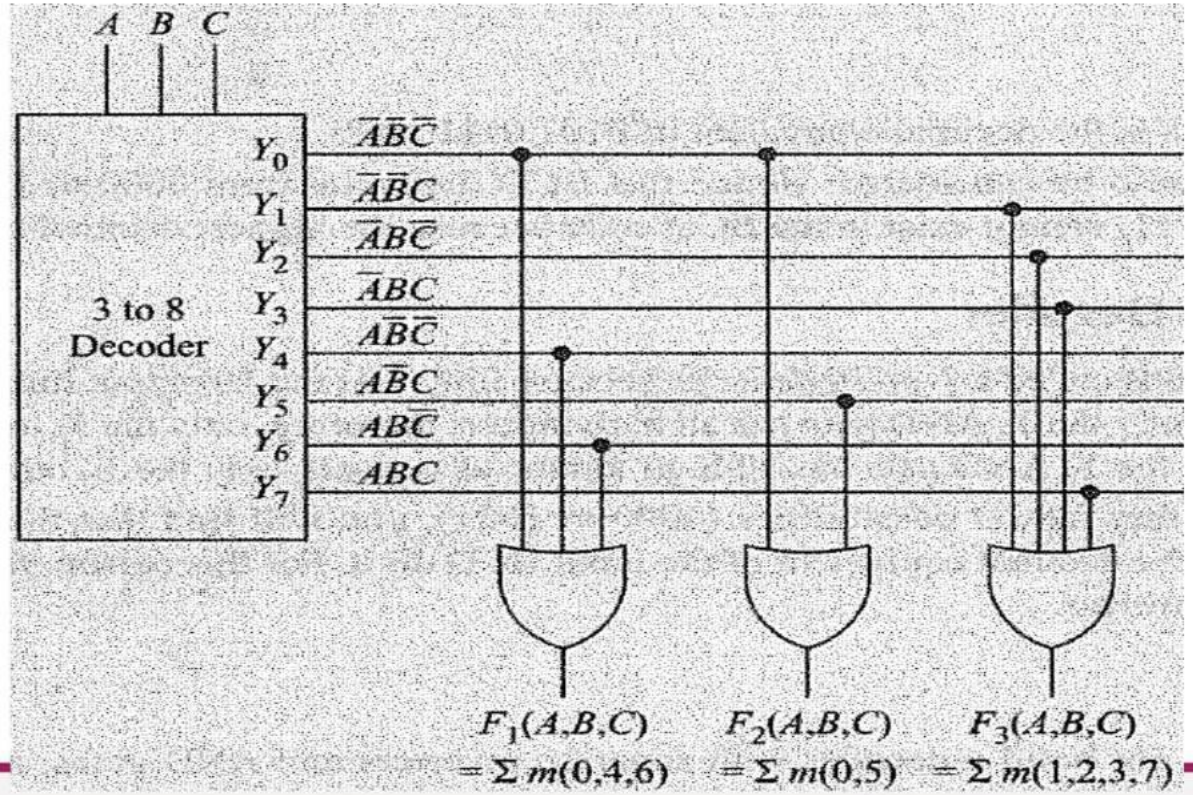
DECODER-7

- Show how using a 3-to-8 decoder and multi-input OR gates following Boolean expressions can be realized simultaneously.

  F1 (A, B, C) = ∑m(0, 4, 6);

  F2(A, B, C) = ∑m(0, 5);

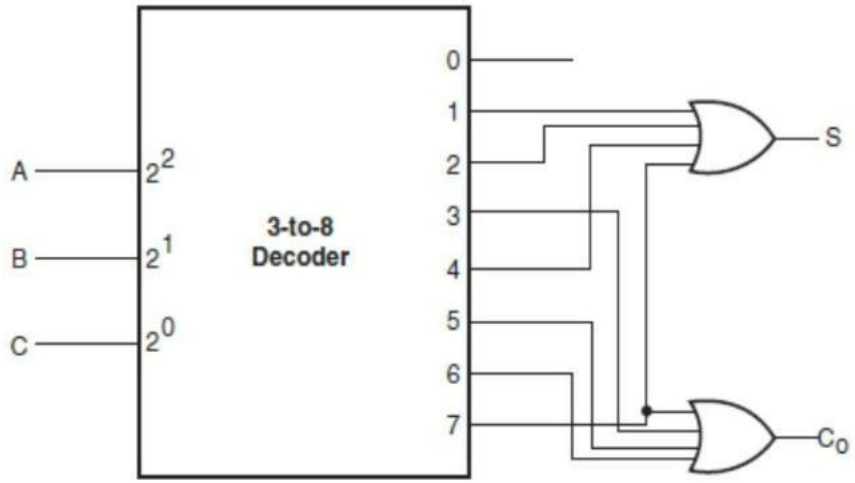
  F3(A, B, C) = ∑m(1, 2, 3, 7)



DECODER-8

- Implement a full adder circuit using a 3-to-8 line decoder.
- Sum output S = ∑m(1 2 4 7)
- Carry ~~output Co= ∑m(3 5 6 7)~~

A	B	C	S	C_o
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



DECODER-9

- **BCD-TO-DECIMAL DECODERS (IC 7445)**

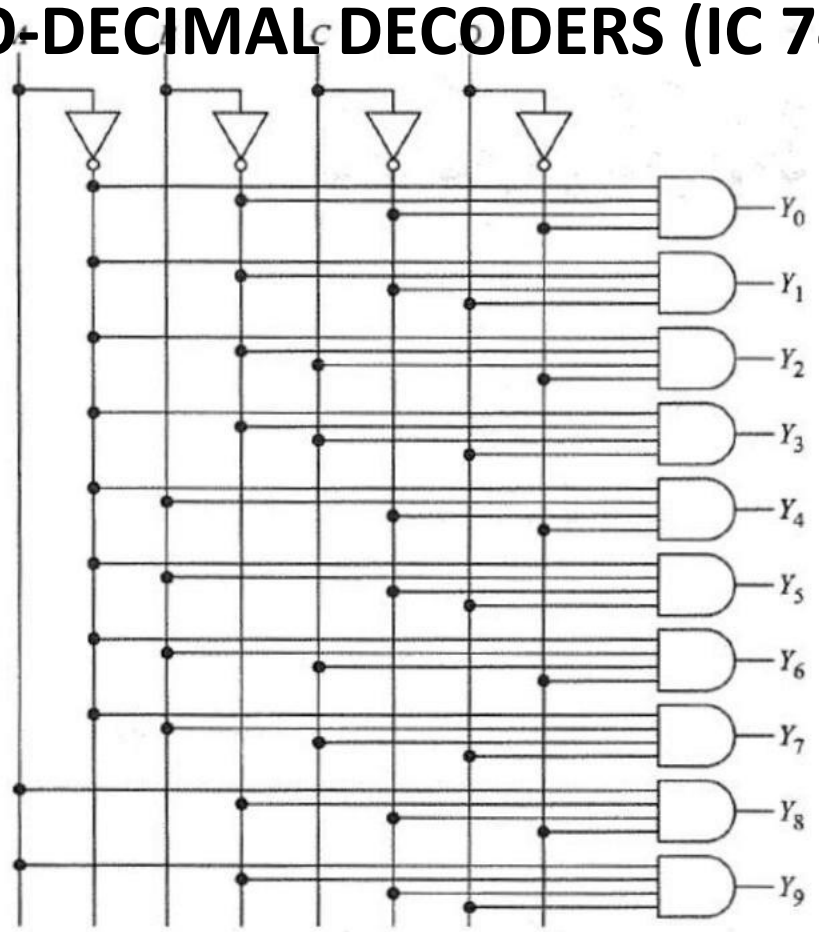
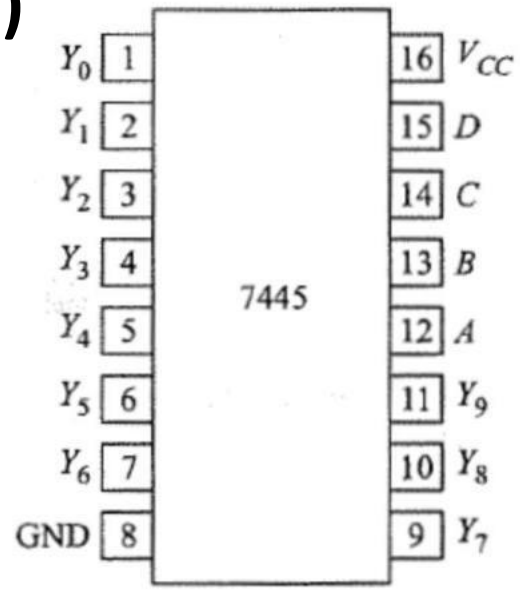


Fig. 4.18 1-of-10 decoder



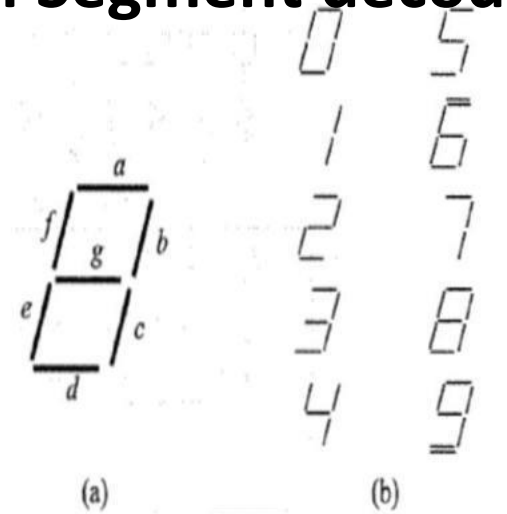
DECODER-10

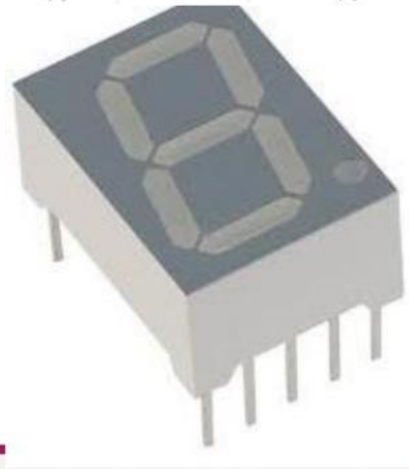
- Truth Table of 1 of 10 decoder

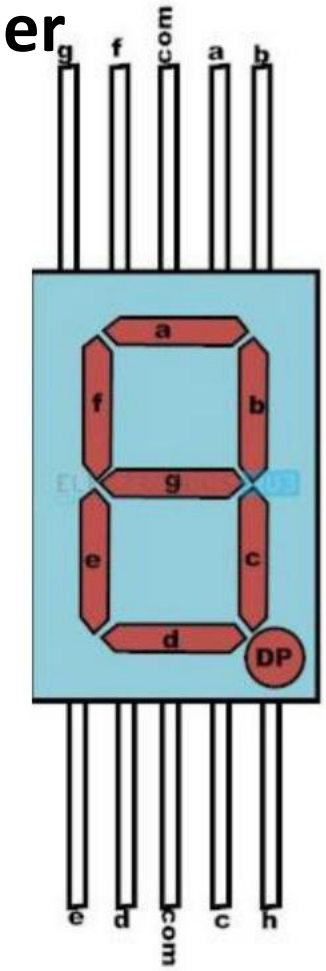
	Inputs				Outputs									
No.	A	B	C	D	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9
0	L	L	L	L	L	H	H	H	H	H	H	H	H	H
1	L	L	L	H	H	L	H	H	H	H	H	H	H	H
2	L	L	H	L	H	H	L	H	H	H	H	H	H	H
3	L	L	H	H	H	H	H	L	H	H	H	H	H	H
4	L	H	L	L	H	H	H	H	L	H	H	H	H	H
5	L	H	L	H	H	H	H	H	H	L	H	H	H	H
6	L	H	H	L	H	H	H	H	H	H	L	H	H	H
7	L	H	H	H	H	H	H	H	H	H	H	L	H	H
8	H	L	L	L	H	H	H	H	H	H	H	H	L	H
9	H	L	L	H	H	H	H	H	H	H	H	H	H	L
	H	L	H	L	H	H	H	H	H	H	H	H	H	H
	H	L	H	H	H	H	H	H	H	H	H	H	H	H
	H	H	L	L	H	H	H	H	H	H	H	H	H	H
	H	H	L	H	H	H	H	H	H	H	H	H	H	H
	H	H	H	L	H	H	H	H	H	H	H	H	H	H
	H	H	H	H	H	H	H	H	H	H	H	H	H	H

DECODER-12

- **Seven Segment decoder**







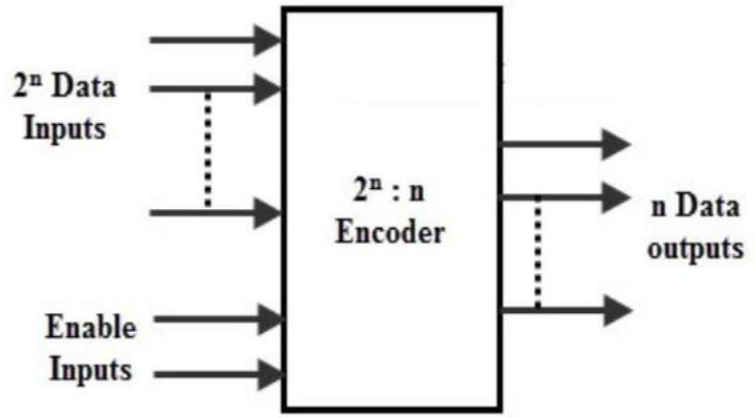
Segments Inputs							7 Segment Display Output
a	b	c	d	e	f	g	
1	1	1	1	1	1	0	0
0	1	1	0	0	0	0	1
1	1	0	1	1	0	1	2
1	1	1	1	0	0	1	3
0	1	1	0	0	1	1	4
1	0	1	1	0	1	1	5
1	0	1	1	1	1	1	6
1	1	1	0	0	0	0	7
1	1	1	1	1	1	1	8
1	1	1	1	0	0	1	9

ENCODERS-1

- An encoder converts an active input signal into a coded output signal.
- An encoder is a device which converts familiar numbers or characters or symbols into a coded format. It accepts the alphabetic characters and decimal numbers as inputs and produces the outputs as a coded representation of the inputs.
- It is a combinational circuit that performs the opposite function of a decoder.
- These are mainly used to reduce the number of bits needed to represent given information.

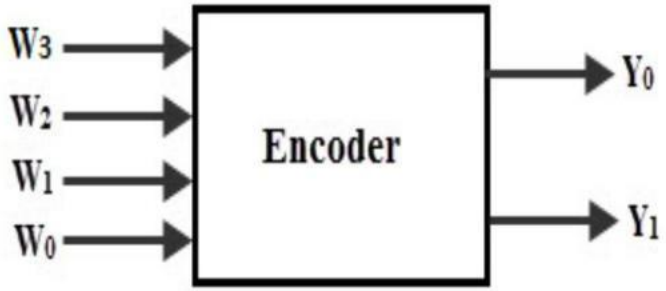
ENCODERS-2

- Depending on the number of input lines, digital or binary encoders produce the output codes in the form of 2 or 3 or 4 bit codes.
- **An *encoder* is a multiplexer without its single output line.**
- It is a combinational logic function that has 2^n (or fewer) input lines and n output lines, which correspond to n selection lines in a multiplexer.
- The n output lines generate the binary code for the possible 2^n input lines.

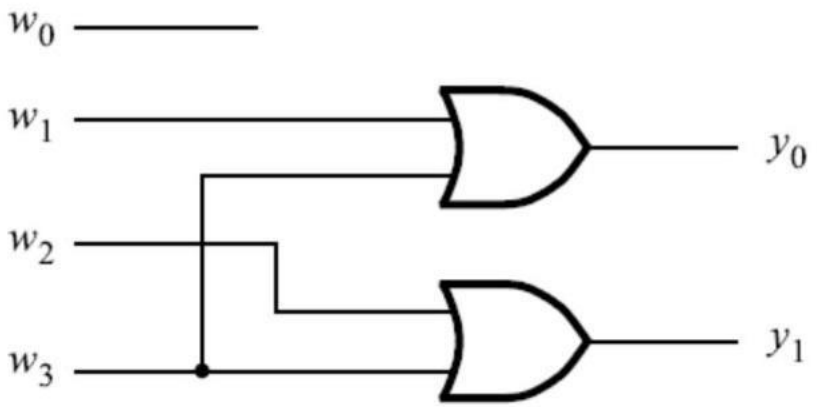


ENCODERS-3

- **4 – to – 2 Bit Binary Encoder**

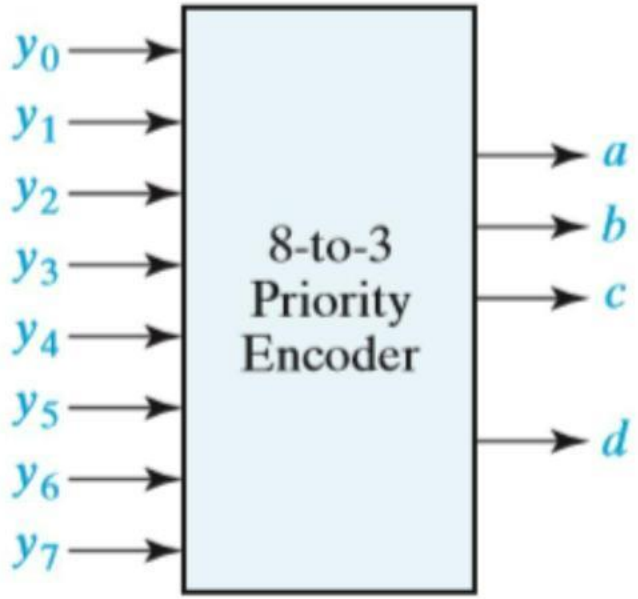


w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1



ENCODERS-3

- **An 8-to-3 Priority Encoder**



y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	a	b	c	d
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
X	1	0	0	0	0	0	0	0	0	1	1
X	X	1	0	0	0	0	0	0	1	0	1
X	X	X	1	0	0	0	0	0	1	1	1
X	X	X	X	1	0	0	0	1	0	0	1
X	X	X	X	X	1	0	0	1	0	1	1
X	X	X	X	X	X	1	0	1	1	0	1
X	X	X	X	X	X	X	1	1	1	1	1

ENCODERS-3

- **An 8-to-3 Priority Encoder**
- 8-to-3 priority encoder with inputs y0 through y7.
- If input yi is 1 and the other inputs are 0, then the abc outputs represent a binary number equal to i.
  - For example, if y3 = 1, then abc = 011.
- If more than one input can be 1 at the same time, the output can be defined using a priority scheme.
- The truth table uses the following scheme:
  - If more than one input is 1, the highest numbered input determines the output.
    - For example, if inputs y1, y4, and y5 are 1, the output is abc = 101.
  - The X's in the table are don't-cares; for example, if y5 is 1, we do not care what inputs y0 through y4 are.
  - Output d is 1 if any input is 1, otherwise, d is 0.
    - This signal is needed to distinguish the case of all 0 inputs from the case where only y0 is 1.

ENCODERS-3

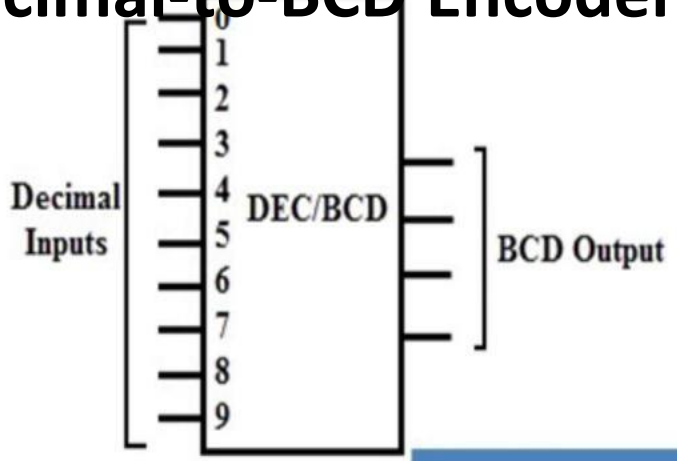
- Complete the following table for a 4-to-2 priority encoder:

y0	Y1	y2	y3	A	B	C

- What will a,b, and c be if y0 y1 y2 y3 is 0101?

ENCODERS-4

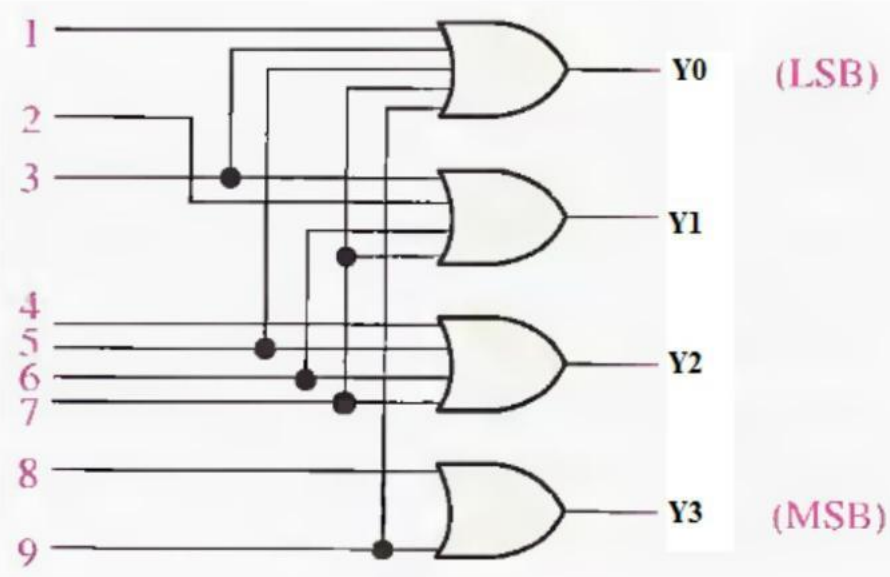
- **Decimal-to-BCD Encoder**



Input										Output			
D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Y_3	Y_2	Y_1	Y_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

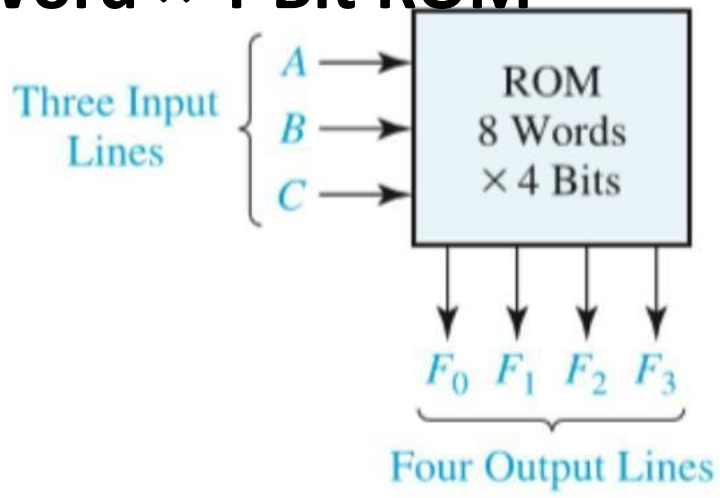
ENCODERS-4

- **Decimal-to-BCD Encoder**
- Y3=D8+D9
- Y2=D4+D5+D6+D7
- Y1=D2+D3+D6+D7
- Y0=D1+D3+D5+D7+D9



Read-Only Memories

- **An 8-Word × 4-Bit ROM**



(a) Block diagram

A B C	F_0 F_1 F_2 F_3
0 0 0	1 0 1 0
0 0 1	1 0 1 0
0 1 0	0 1 1 1
0 1 1	0 1 0 1
1 0 0	1 1 0 0
1 0 1	0 0 0 1
1 1 0	1 1 1 1
1 1 1	0 1 0 1

Typical Data Stored in ROM (2^3 words of 4 bits each)

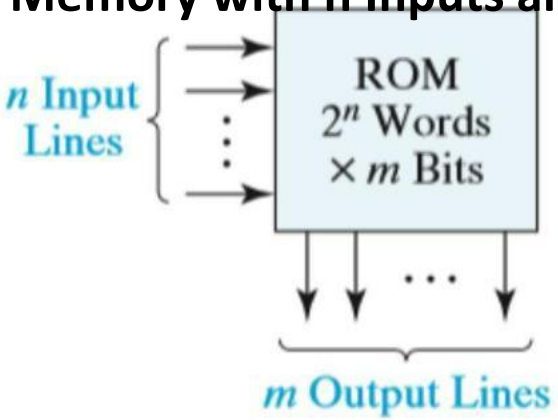
(b) Truth table for ROM

Read-Only Memories

- For each combination of input values on the three input lines, the corresponding pattern of 0's and 1's appears on the ROM output lines.
  - For example, if the combination ABC = 010 is applied to the input lines, the pattern $F_0F_1F_2F_3$ = 0111 appears on the output lines.
- **Each of the output patterns that is stored in the ROM is called a word.**
  - Because the ROM has three input lines, we have 2^3 = eight different combinations of input values. Each input combination serves as an address which can select one of the eight words stored in the memory.
  - Because there are four output lines, each word is four bits long, and the size of this ROM is 8 words × 4 bits.

Read-Only Memories

- **Read-Only Memory with n Inputs and m Outputs**



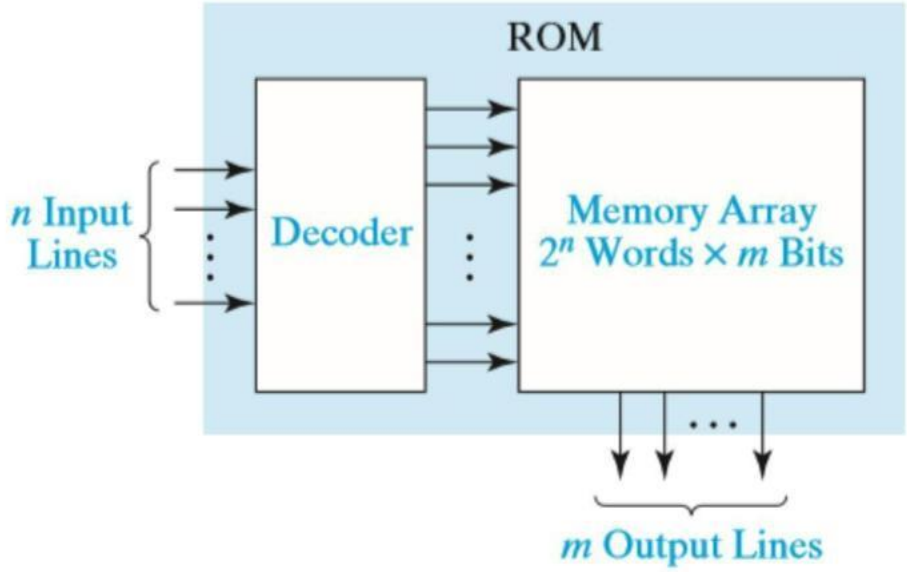
n Input Variables	*m* Output Variables
00 · · · 00	100 · · · 110
00 · · · 01	010 · · · 111
00 · · · 10	101 · · · 101
00 · · · 11	110 · · · 010
⋮	⋮
11 · · · 00	001 · · · 011
11 · · · 01	110 · · · 110
11 · · · 10	011 · · · 000
11 · · · 11	111 · · · 101

Typical Data Array Stored in ROM (2^n words of m bits each)

- **A ROM which has n input lines and m output lines contains an array of 2^n words, and each word is m bits long. The input lines serve as an address to select one of the 2^n words.**

Read-Only Memories

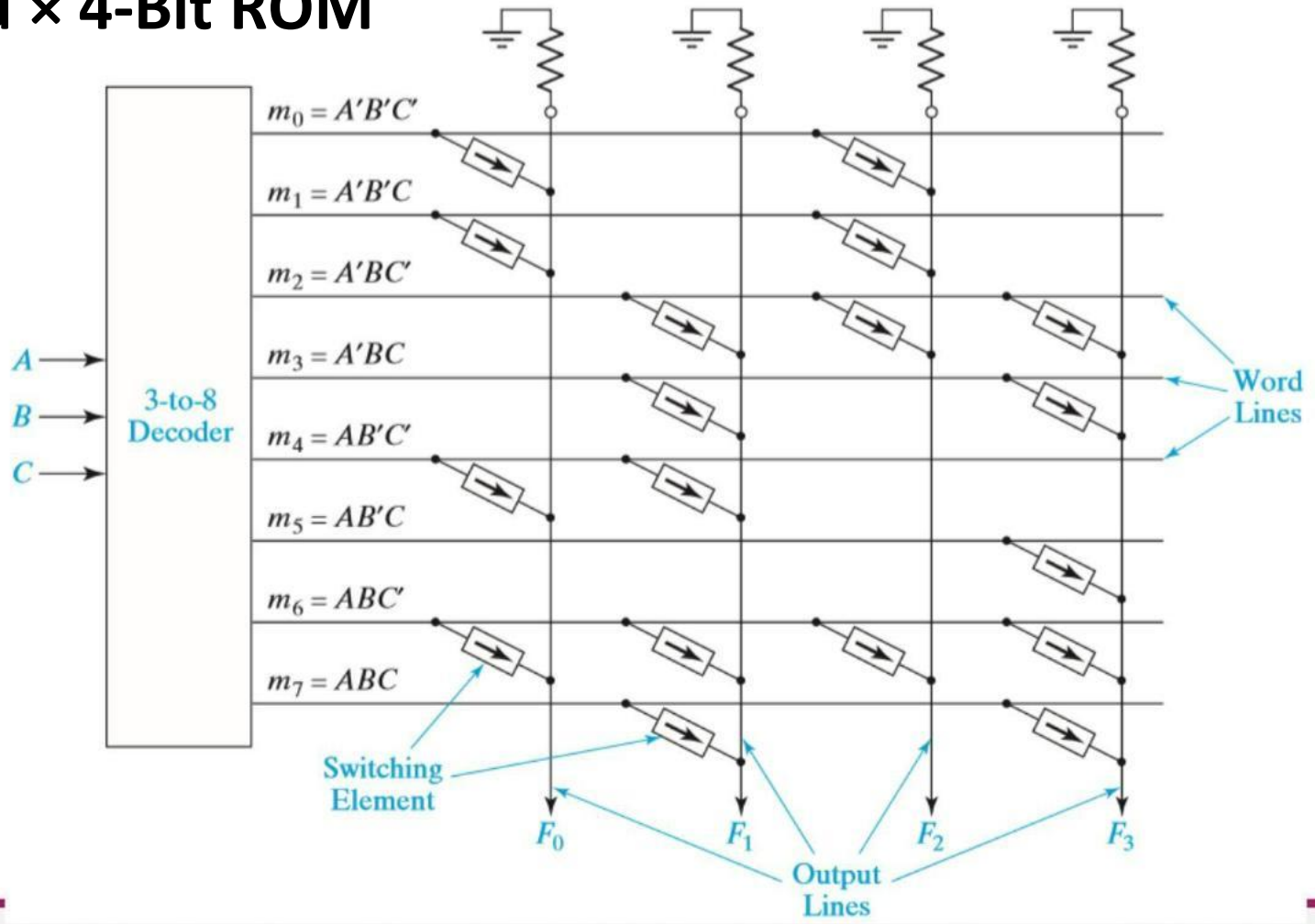
- **Basic ROM Structure**



- When a pattern of n 0's and 1's is applied to the decoder inputs, exactly one of the 2^n decoder outputs is 1.
- This decoder output line selects one of the words in the memory array, and the bit pattern stored in this word is transferred to the memory output lines.

Read-Only Memories

- **An 8-Word × 4-Bit ROM**

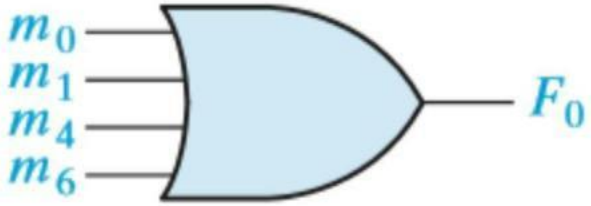


Read-Only Memories

- **An 8-Word × 4-Bit ROM**
  - The decoder generates the eight minterms of the three input variables.
  - The memory array forms the four output functions by ORing together selected minterms.
  - A switching element is placed at the intersection of a word line and an output line if the corresponding minterm is to be included in the output function; otherwise, the switching element is omitted (or not connected).
  - If a switching element connects an output line to a word line which is 1, the output line will be 1.
  - Otherwise, the pull-down resistors which cause the output line to be 0.

Read-Only Memories

- **An 8-Word × 4-Bit ROM**
  - So the switching elements which are connected in this way in the memory array effectively form an OR gate for each of the output functions.
    - For example, m0, m1, m4, and m6 are ORed together to form F0. Figure shows the equivalent OR gate.



Read-Only Memories

- **An 8-Word × 4-Bit ROM**
  - In general, those minterms which are connected to output line F by switching elements are ORed together to form the output Fi.
  - Thus, the ROM generates the following functions:
  - F0 =Σ m(0, 1, 4, 6) = A'B' + AC'
  - F1 =Σ m(2, 3, 4, 6, 7) = B + AC'
  - F2 =Σ m(0, 1, 2, 6) = A'B' + BC'
  - F3 =Σ m(2, 3, 5, 6, 7) = AC + B
    -

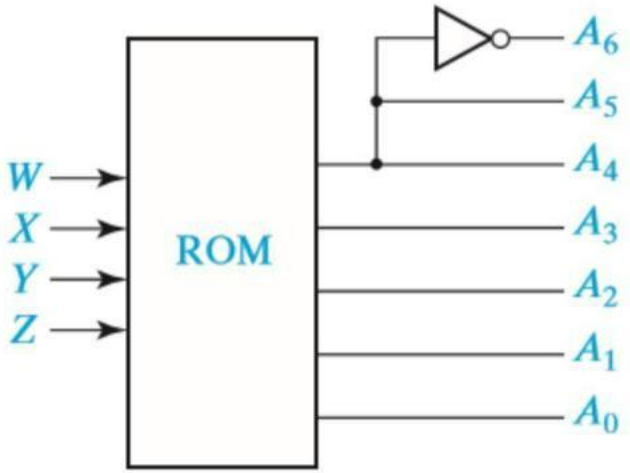
Read-Only Memories

- **Multiple-output combinational circuits can easily be realized using ROMs.**
- **Example:**
  - Realize a code converter that converts a 4-bit binary number to a hexadecimal digit and outputs the 7-bit ASCII code.

Read-Only Memories

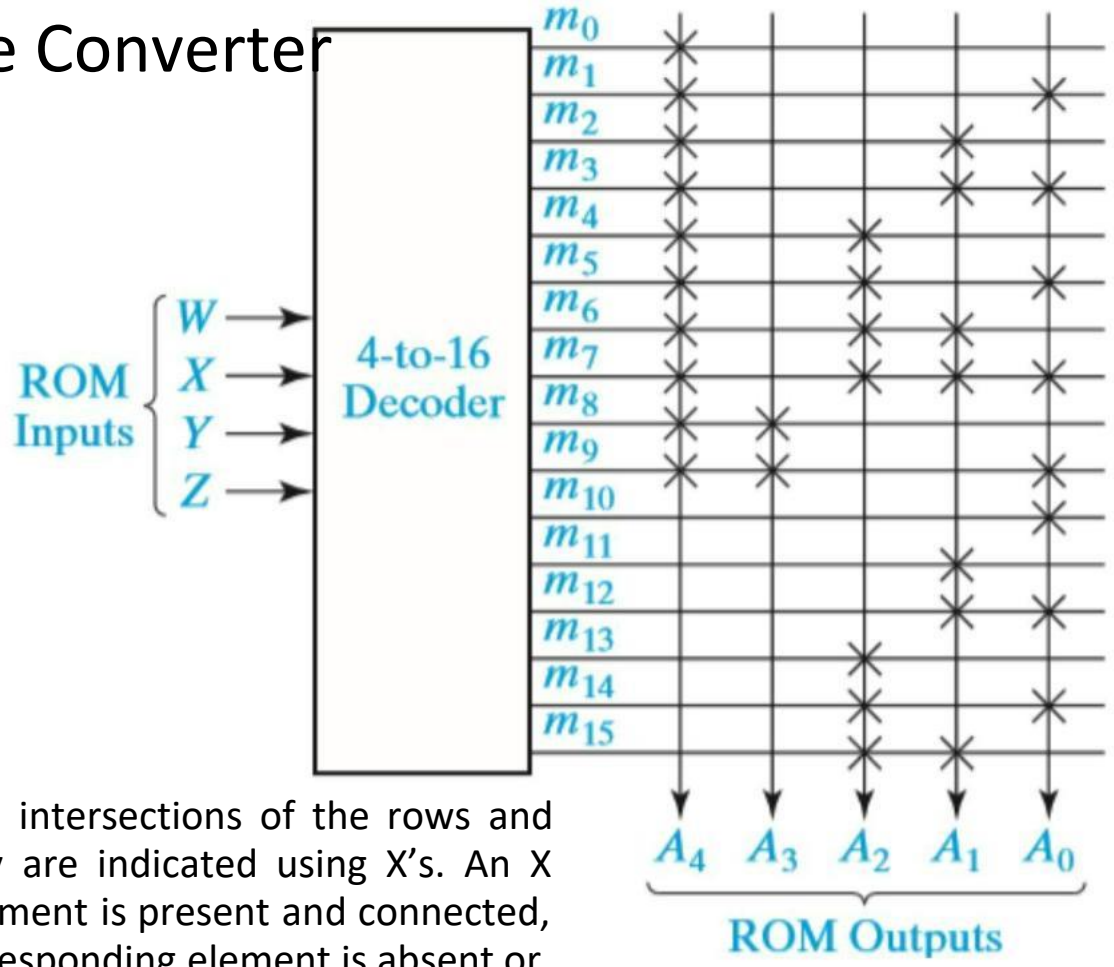
- Truth table and logic circuit for the converter

Input W X Y Z	Hex Digit	ASCII Code for Hex Digit A_6 A_5 A_4 A_3 A_2 A_1 A_0
0 0 0 0	0	0 1 1 0 0 0 0
0 0 0 1	1	0 1 1 0 0 0 1
0 0 1 0	2	0 1 1 0 0 1 0
0 0 1 1	3	0 1 1 0 0 1 1
0 1 0 0	4	0 1 1 0 1 0 0
0 1 0 1	5	0 1 1 0 1 0 1
0 1 1 0	6	0 1 1 0 1 1 0
0 1 1 1	7	0 1 1 0 1 1 1
1 0 0 0	8	0 1 1 1 0 0 0
1 0 0 1	9	0 1 1 1 0 0 1
1 0 1 0	A	1 0 0 0 0 0 1
1 0 1 1	B	1 0 0 0 0 1 0
1 1 0 0	C	1 0 0 0 0 1 1
1 1 0 1	D	1 0 0 0 1 0 0
1 1 1 0	E	1 0 0 0 1 0 1
1 1 1 1	F	1 0 0 0 1 1 0



Read-Only Memories

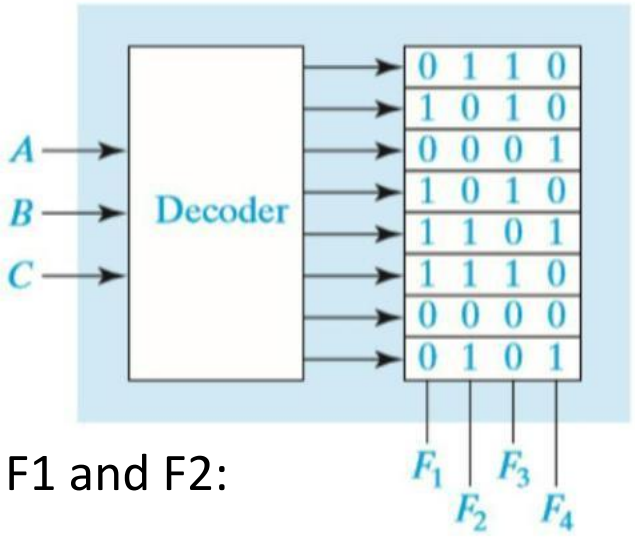
- ROM Realization of Code Converter



The switching elements at the intersections of the rows and columns of the memory array are indicated using X's. An X indicates that the switching element is present and connected, and no X indicates that the corresponding element is absent or not connected.

Read-Only Memories

- The following diagram shows the pattern of 0's and 1's stored in a ROM with eight words and four bits per word. What will be the values of F1, F2, F3, and F4 if A = 0 and B = C = 1?



- Give the minterm expansions for F1 and F2:
- F1=
- F2=