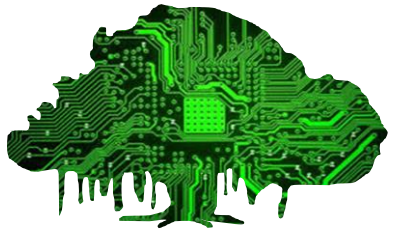


Lecture 8 – Combinational logic circuits

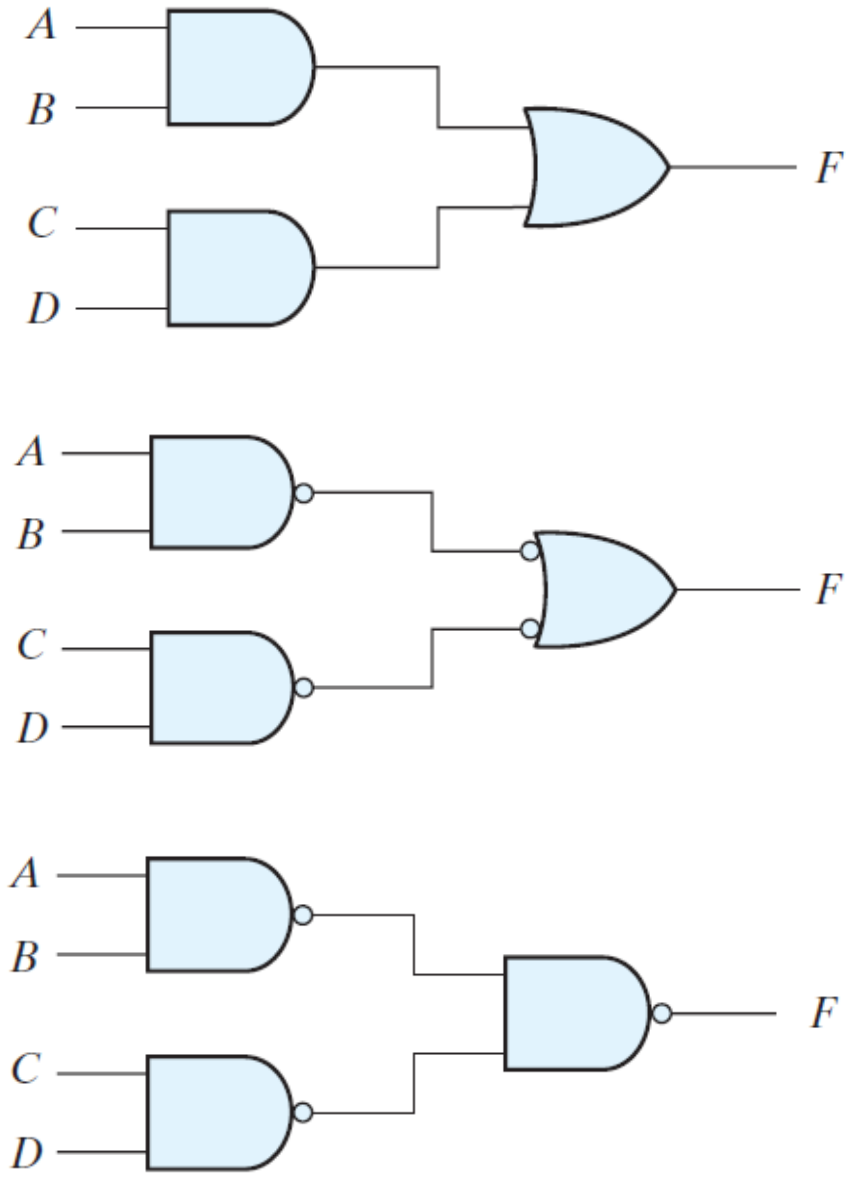
Dr. Aftab M. Hussain,

Assistant Professor, PATRIoT Lab, CVEST

Chapter 4

NAND implementations

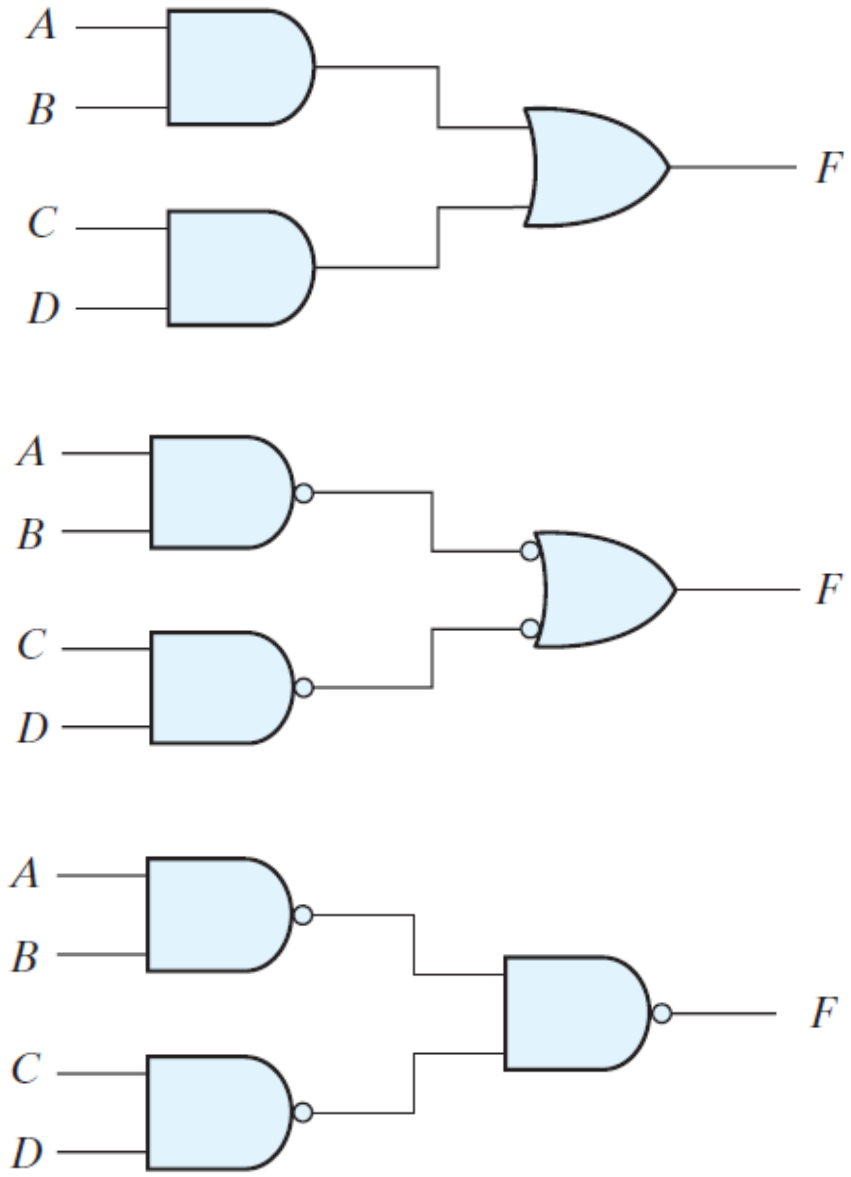
- Remember that NAND and NOR are universal gates – thus, we should be able to make any implementation using NAND and NOR gates
- Consider a simple AND-OR implementation (sum of products)
- We can put two complementary operation in the middle of the wire
- The first layer AND gates become NAND and the second layer OR gate can be made NAND using the DeMorgan theorem



NAND implementations

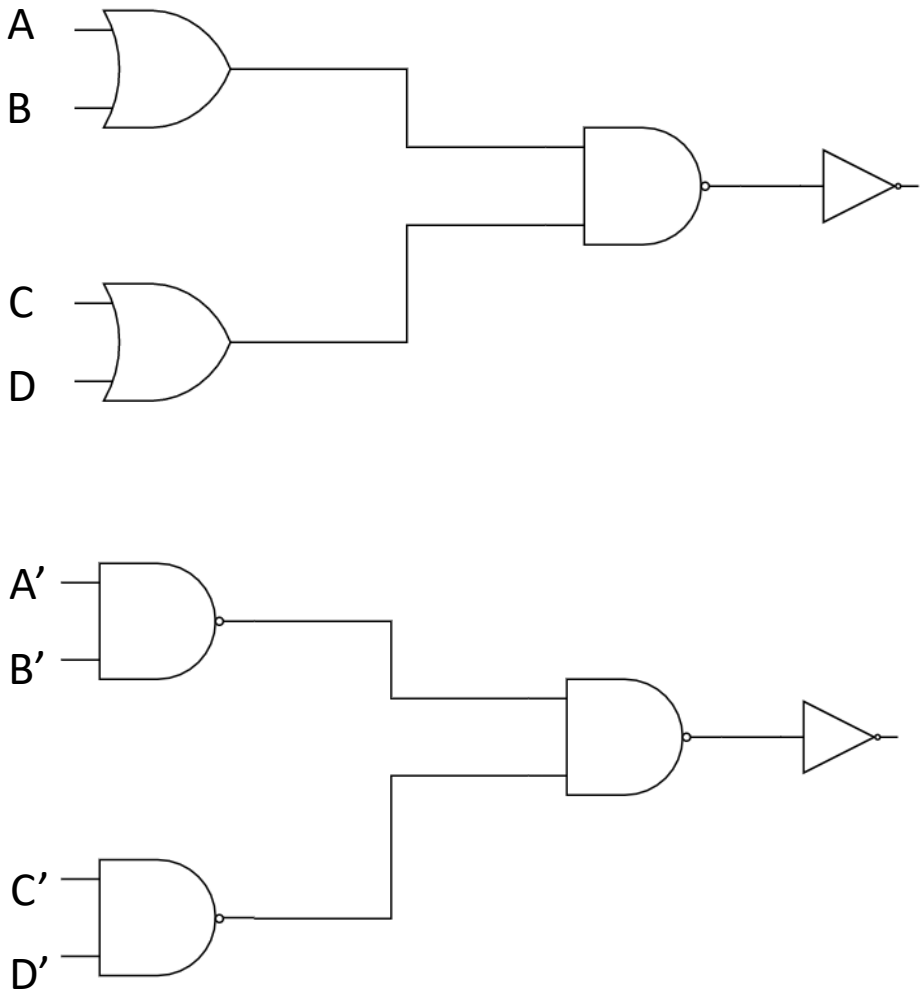
- The procedure for obtaining the NAND logic diagram from a Boolean function is as follows:

1. Simplify the function and express it in sum-of-products form
2. Draw a NAND gate for each product term of the expression that has at least two literals. The inputs to each NAND gate are the literals of the term. This procedure produces a group of first-level gates
3. Draw a single NAND gate with inputs coming from outputs of first-level gates.
4. A term with a single literal requires an inverter in the first level. However, if the single literal is complemented, it can be connected directly to an input of the second level NAND gate



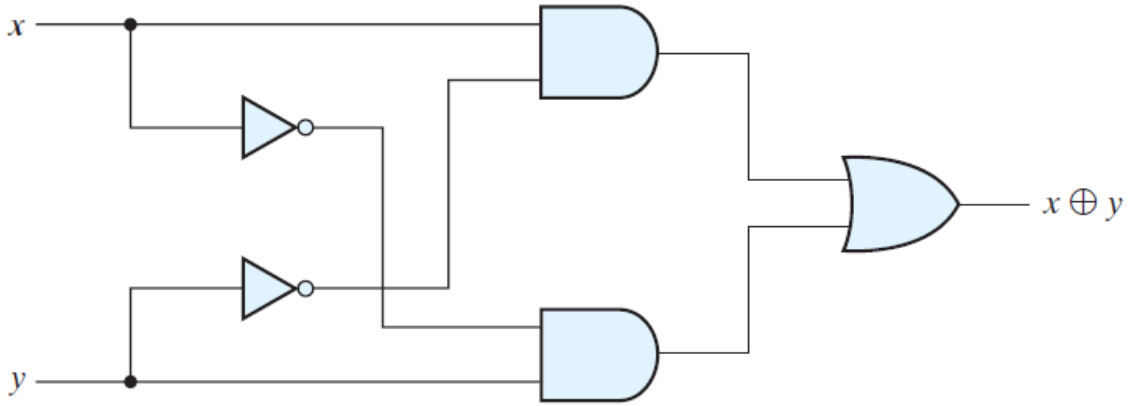
NAND implementations

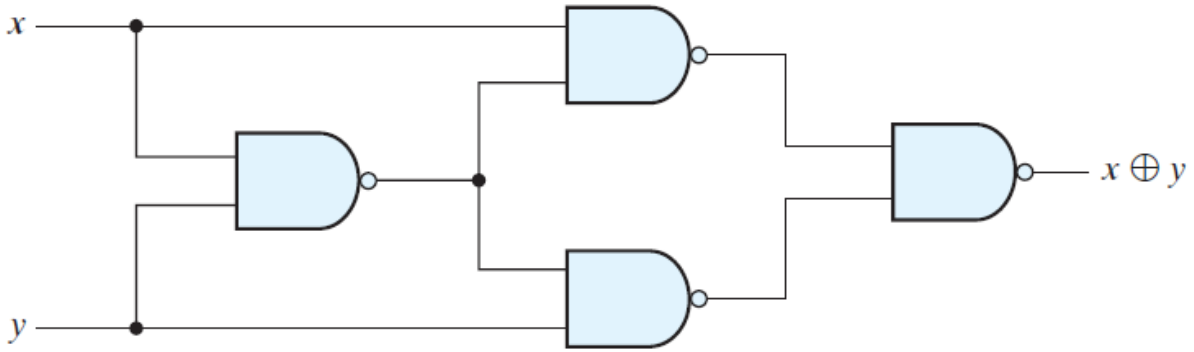
- In case of OR-AND implementation (product of sum)
- We have to complement the output to get a NAND gate at the second level
- Then we complement the inputs we are providing and obtain NAND gates at the first level
- The NOR operation is the dual of the NAND operation
- Therefore, all procedures and rules for NOR logic are the duals of the corresponding procedures and rules developed for NAND logic



ExOR gate

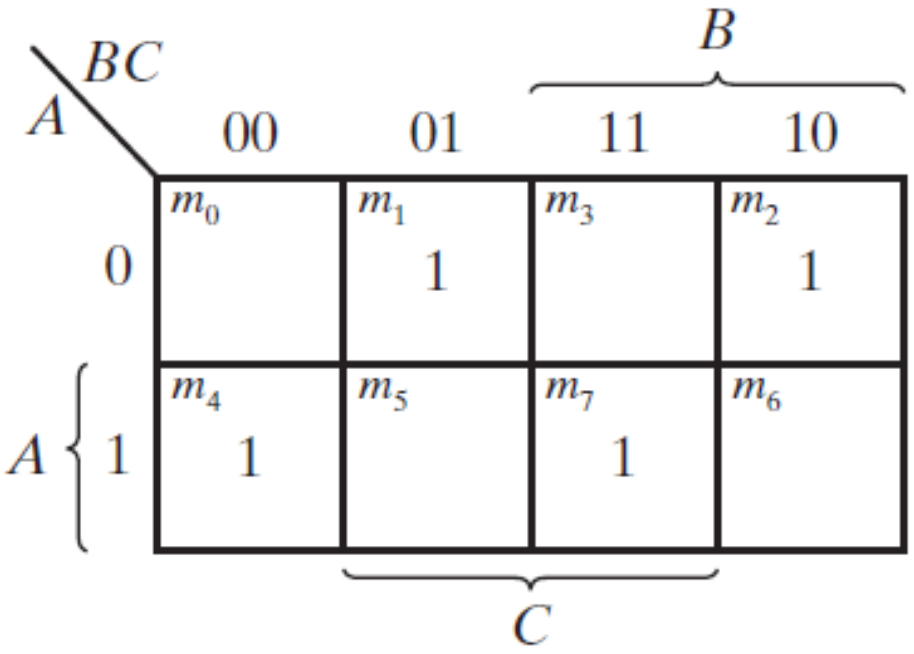
- ExOR is weird because there is no easy way to make an ExOR gate other than simply implementing its logic function

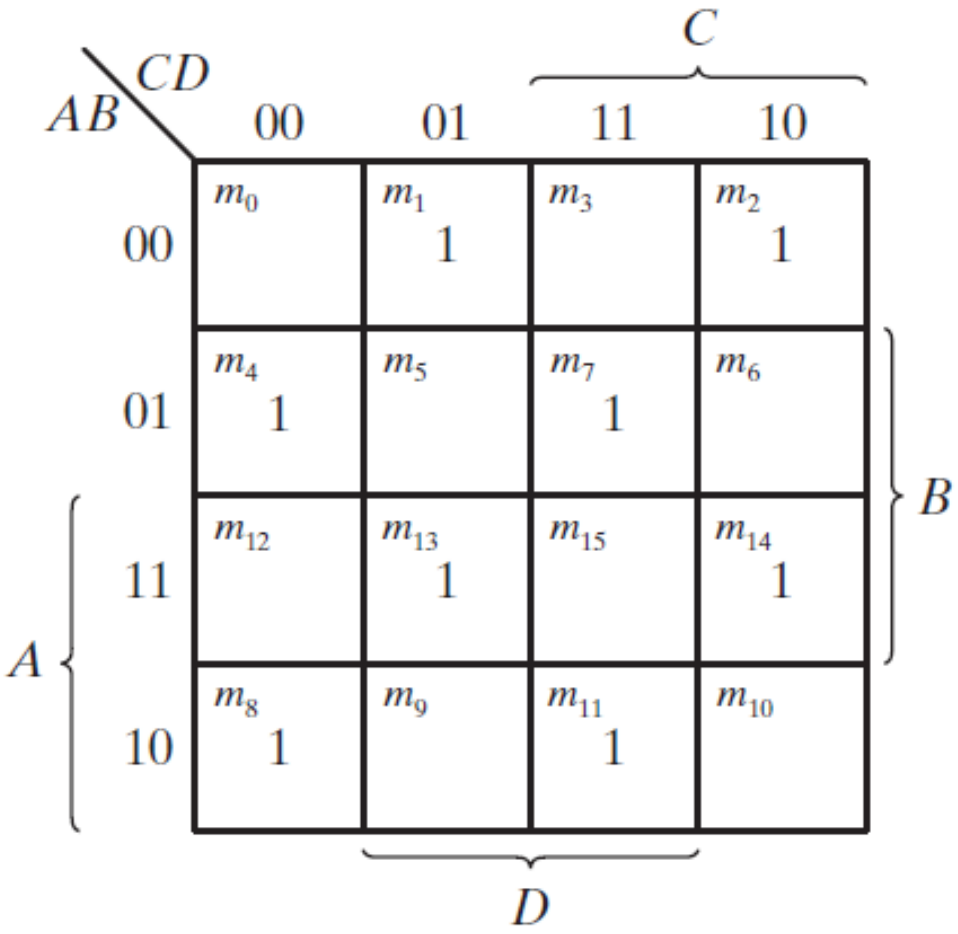




ExOR gate

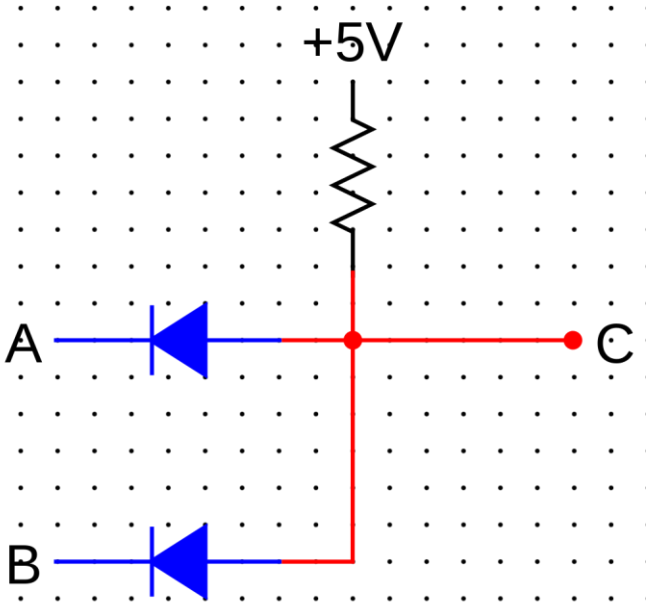
- ExOR is weird because there is no easy way to make an ExOR gate other than simply implementing its logic function



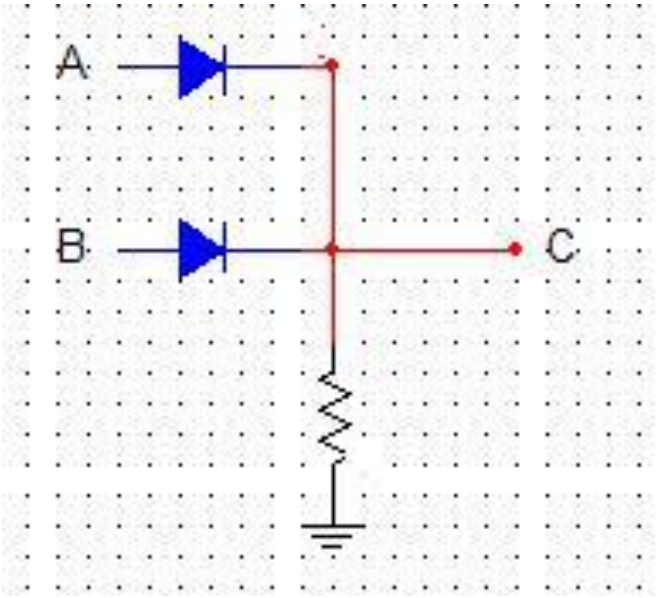


Wired AND and Wired OR

- This is a way of making AND and OR logic gates without the use of complex components like transistors
- A wired logic connection is a logic gate that implements Boolean algebra (logic) using only components such as diodes and resistors
- Diodes in parallel can be configured to obtain wired AND and wired OR by using a “pull-up” resistor or a “pull-down” resistor



Wired AND

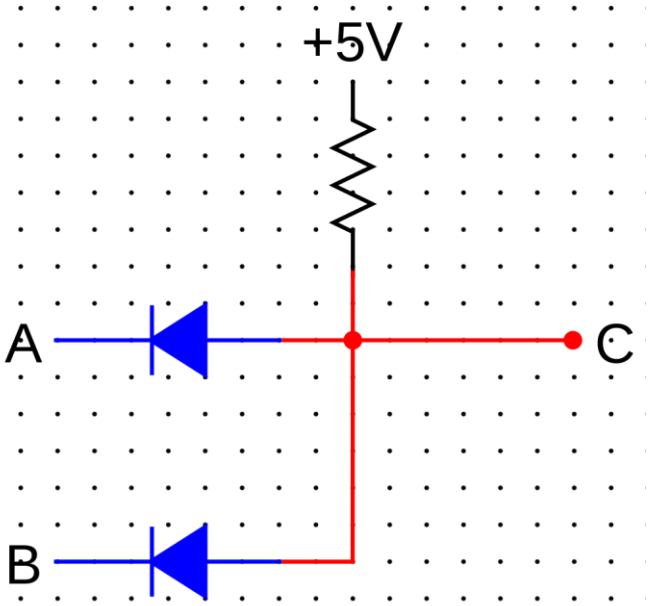


Wired OR

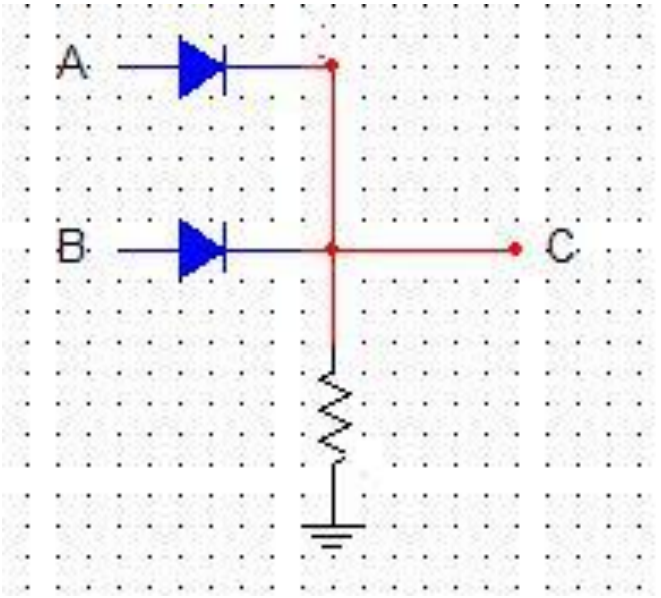
Wired AND and Wired OR

- Advantages
  - Simple to implement
  - Low-cost

- Problems:
  - Wired AND cannot drive the voltage to LOW and Wired OR cannot drive the voltage to HIGH
  - Very high power for certain states
  - May not be compatible in case of multiple levels of logic



Wired AND



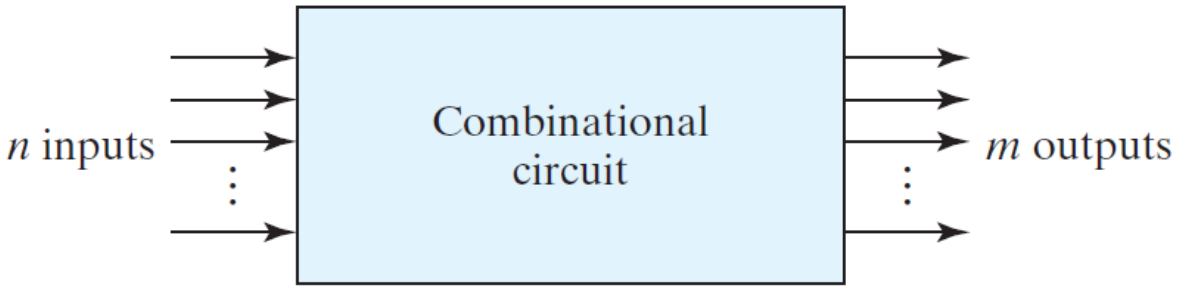
Wired OR

Combinational circuits

- Logic circuits for digital systems may be combinational or sequential
- A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs
- A combinational circuit performs an operation that can be specified logically by a set of Boolean functions
- In contrast, sequential circuits employ storage elements in addition to logic gates
- Their outputs are a function of the inputs and the state of the storage elements
- Because the state of the storage elements is a function of previous inputs, the outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs, and the circuit behavior must be specified by a time sequence of inputs and internal states

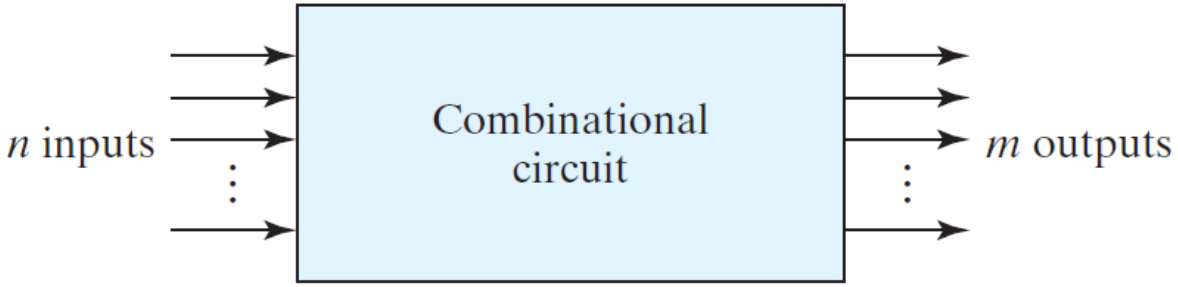
Combinational circuits

- A combinational circuit consists of an interconnection of logic gates
- Combinational logic gates react to the values of the signals at their inputs and produce the value of the output signal, transforming binary information from the given input data to a required output data
- Each input and output variable exists physically as an analog signal whose values are interpreted to be a binary signal that represents logic 1 and logic 0



Combinational circuits

- For n input variables, there are 2^n possible combinations of the binary inputs
- For each possible input combination, there is one possible value for each output variable
- Thus, a combinational circuit can be specified with a truth table that lists the output values for each combination of input variables
- A combinational circuit also can be described by m Boolean functions, one for each output variable
- Each output function is expressed in terms of the n input variables

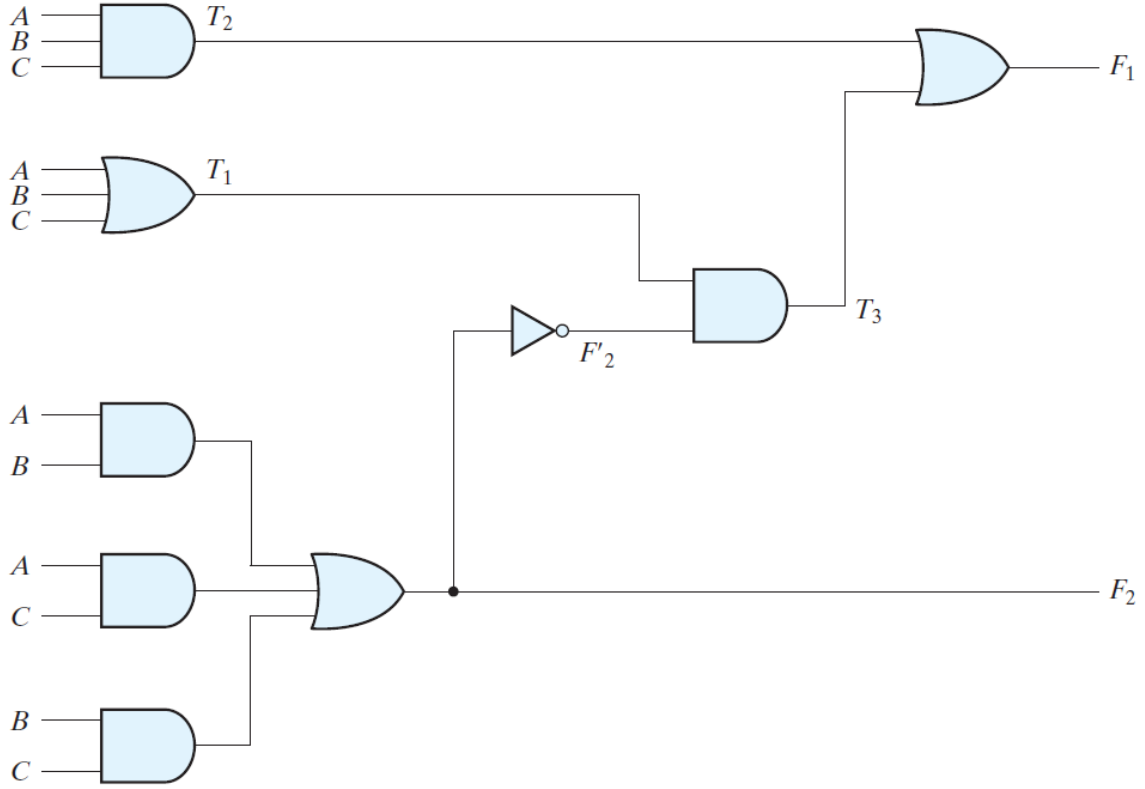


Circuit analysis

- In some cases, we might need to verify whether a particular circuit implements a given logic function
- To analyze a logic diagram:

1. Label all gate outputs that are a function of input variables with arbitrary symbols. Determine the Boolean functions for each gate output
2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates
3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained
4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables

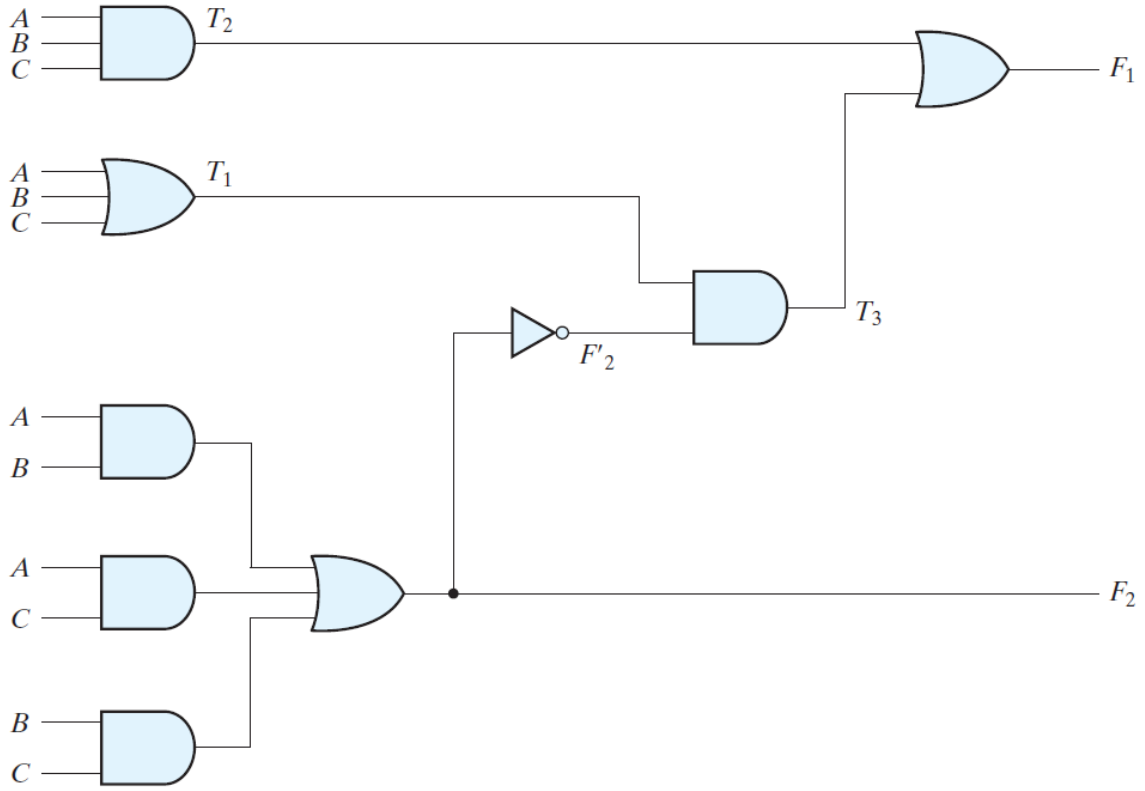
- This circuit is for the Full adder, with F1 as the sum bit and F2 as the carry bit



Circuit analysis

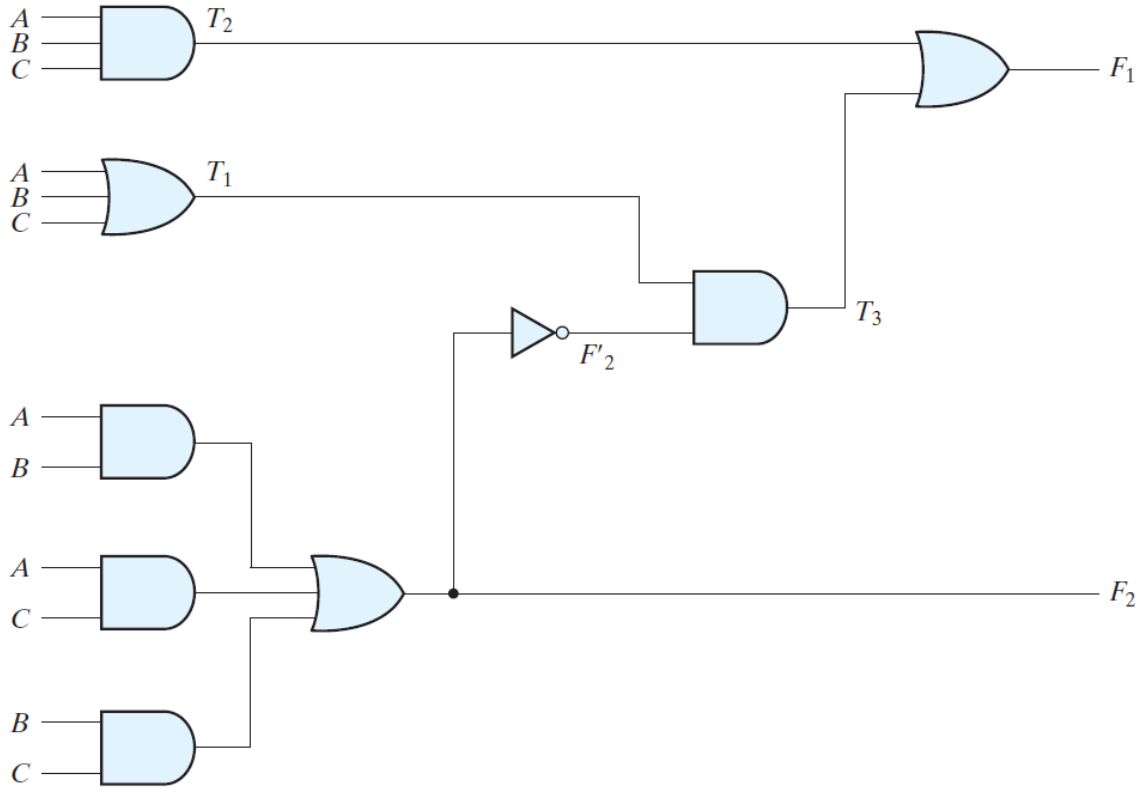
- To obtain the truth table directly from the logic diagram without going through the derivations of the Boolean functions, we proceed as follows:

1. Determine the number of input variables in the circuit. For *n* inputs, form the 2^n possible input combinations and list the binary numbers from 0 to $(2^n - 1)$ in a table
2. Label the outputs of selected gates with arbitrary symbols
3. Obtain the truth table for the outputs of those gates which are a function of the input variables only
4. Proceed to obtain the truth table for the outputs of those gates which are a function of previously defined values until the columns for all outputs are determined



Circuit analysis

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1



Circuit design

- The design of combinational circuits starts from the specification of the design objective and culminates in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be obtained
- The procedure involves the following steps:

1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each
2. Derive the truth table that defines the required relationship between inputs and outputs
3. Make the K-map, if necessary
4. Obtain the simplified Boolean functions for each output as a function of the input variables
5. Draw the logic diagram and verify the correctness of the design (manually or by simulation)

Circuit design

- The output binary functions listed in the truth table are simplified by any available method, such as algebraic manipulation, the map method, or a computer-based simplification program
- Frequently, there is a variety of simplified expressions from which to choose
- A practical designer must consider such constraints as the number of gates, number of inputs to a gate, propagation time of the signal through the gates, number of interconnections, limitations of the driving capability of each gate (i.e., the number of gates to which the output of the circuit may be connected), and various other criteria that must be taken into consideration when designing integrated circuits
- Since the importance of each constraint is dictated by the particular application, it is difficult to make a general statement about what constitutes an acceptable implementation
- In most cases, the simplification begins by satisfying an elementary objective, such as producing the simplified Boolean functions in a standard form
- Then the simplification proceeds with further steps to meet other performance criteria

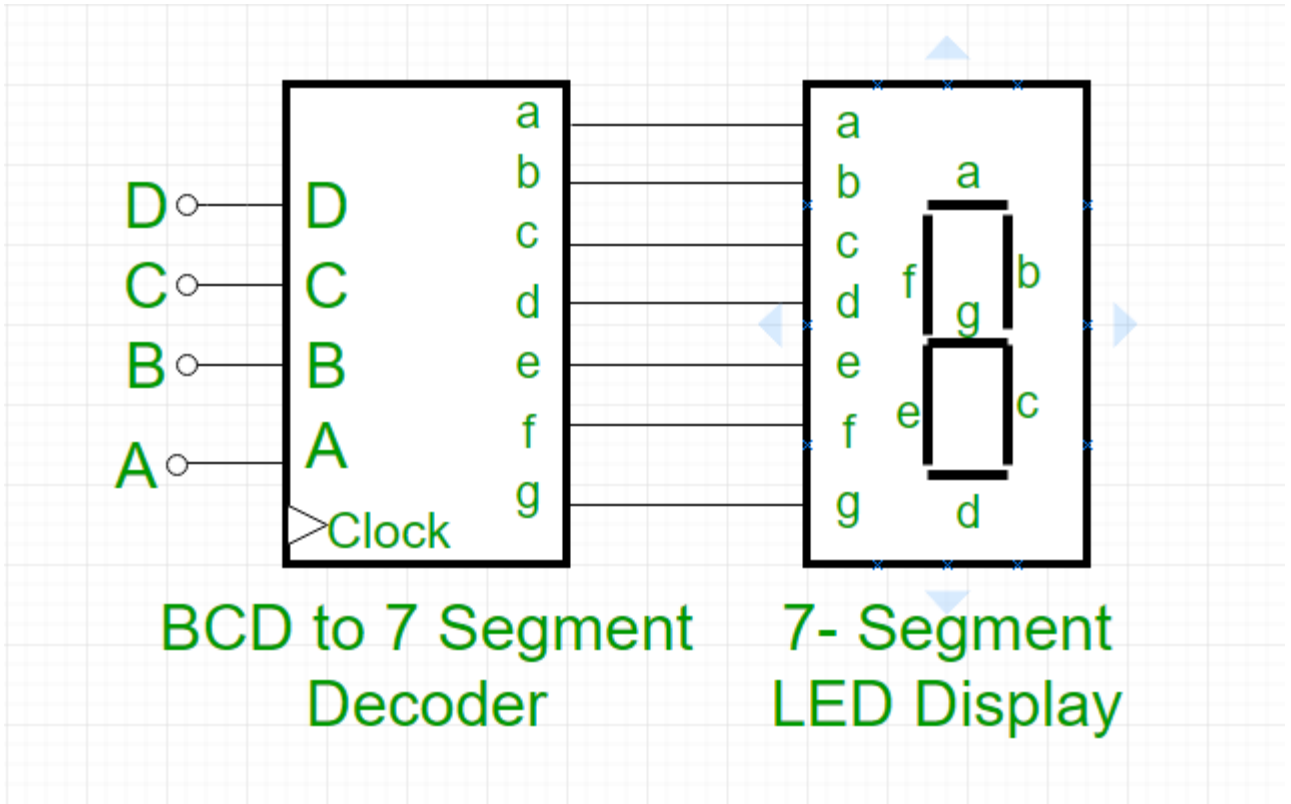
Circuit design

- Say with everything else kept constant, we are most worried about the silicon area within which our design will fit
- To minimize the silicon real-estate needed, we have to reduce the number of transistors we use for a particular design – transistors are electronic switches that form the backbone of most of the modern day electronics
- A simple guide to remember:

Gate	Inputs	No of Transistors
NOT	1	2
Buffer	1	2
NAND	N	2N
NOR	N	2N
AND	N	2N+2
OR	N	2N+2

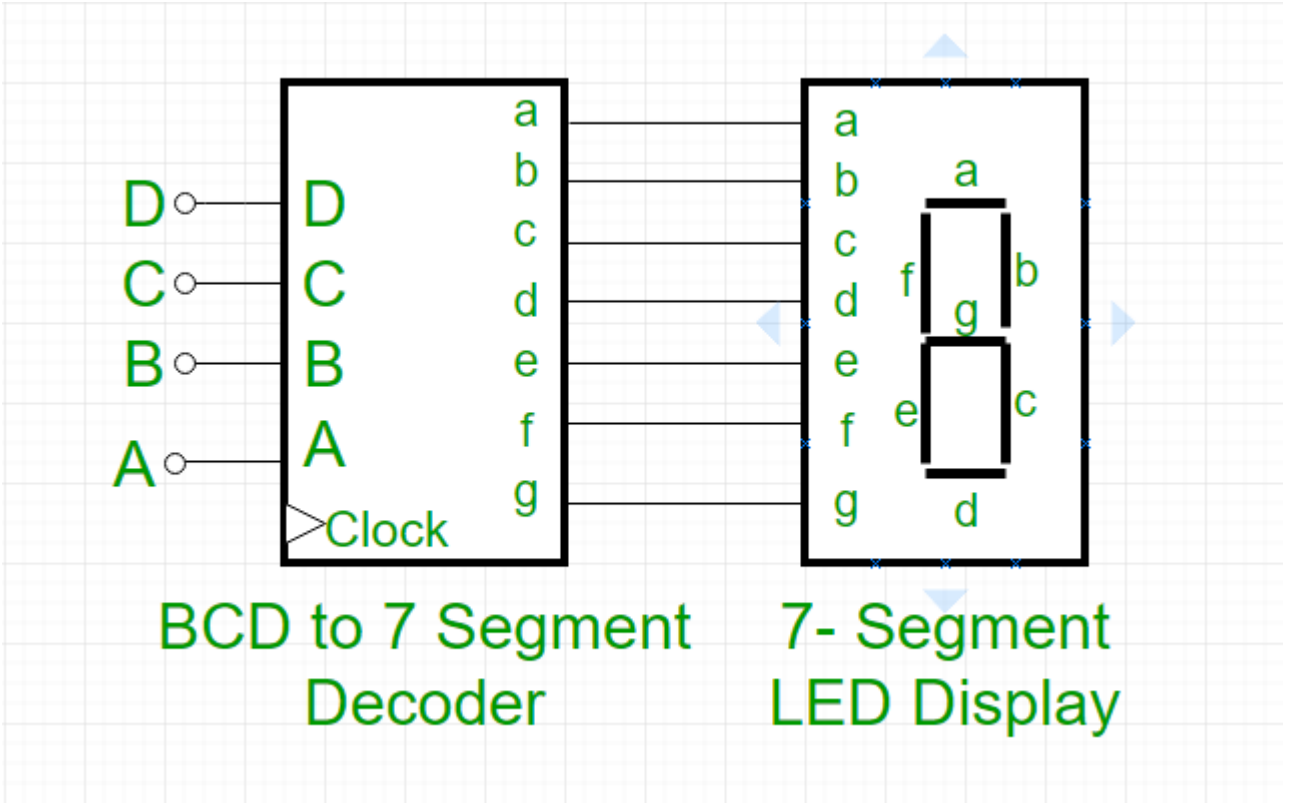
Encoder/decoder

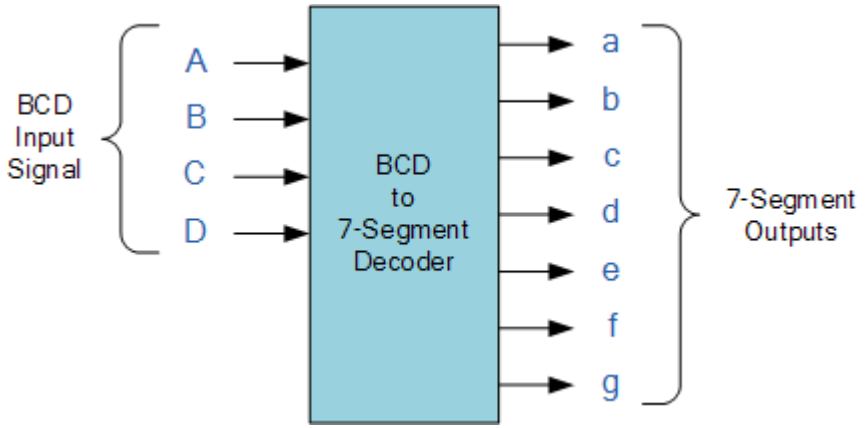
- We discussed many ways of representing symbols using binary variables – signed magnitude, 2's complement, BCD, ASCII etc.
- Going from one representation may be considered as an encoding/decoding operation
- Consider the practical problem of displaying binary numbers using a 7-segment LED display
- For this, we need to "decode" the binary information into the display inputs



Encoder/decoder

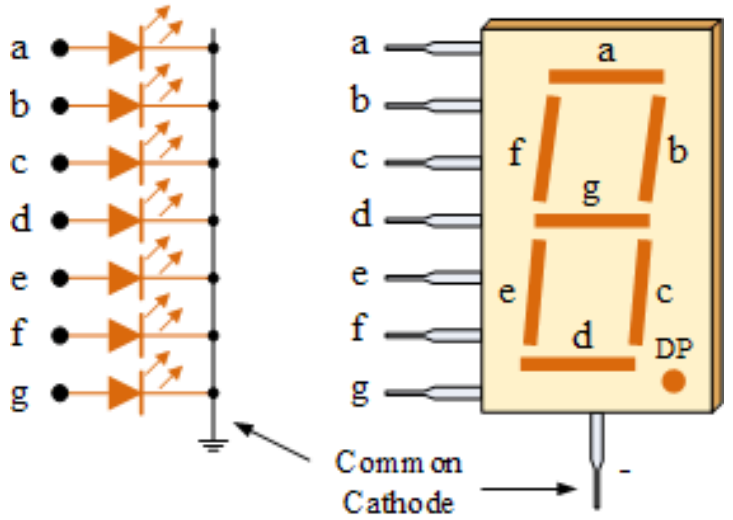
- First, we realize that there are 4 inputs and 7 outputs!
- Thus, the truth-table will be a 16 row table with 7 different columns for 7 outputs
- Hence, there will be 7 different functions we will be implementing to make this decoder
- The other thing we need to realize is whether a particular output should be HIGH or LOW for the LED to glow?

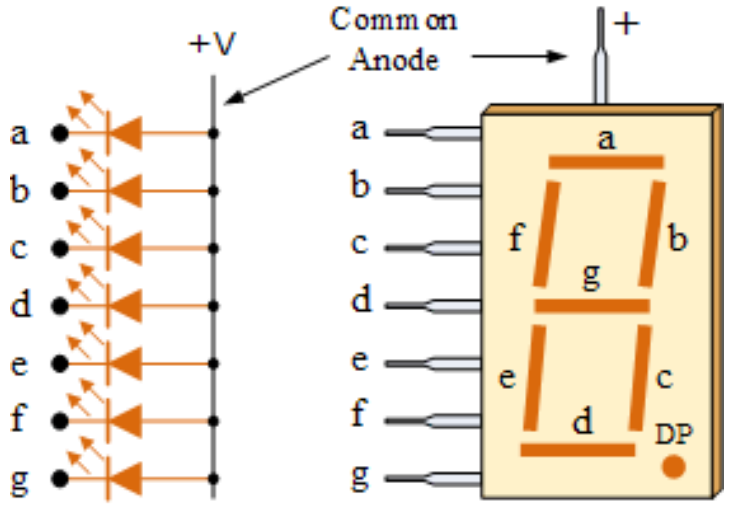




Encoder/decoder

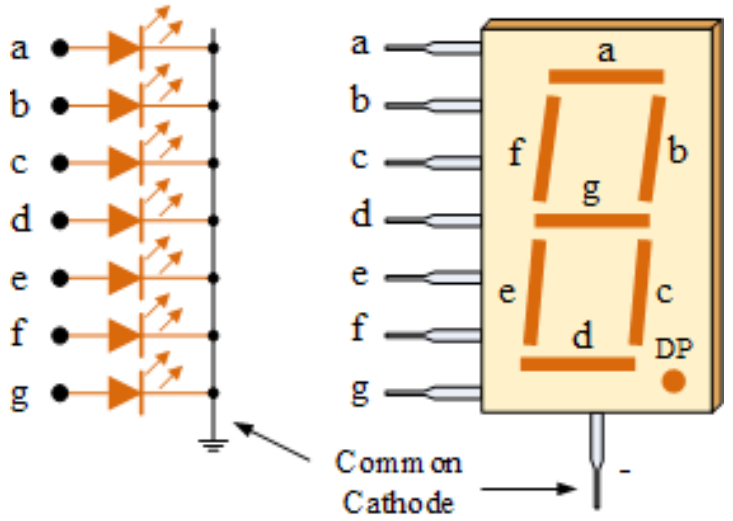
- The other thing we need to realize is whether a particular output should be HIGH or LOW for the LED to glow?
- To know the answer to this, we see that there are two types of 7-segment LED displays – common anode and common cathode
- The common cathode connects all LED cathodes to a common ground, thus, when input is high LED glows
- Conversely, the common anode connects all LED anodes to $+V_{cc}$, thus, when input is low, LED glows





Encoder/decoder

- Let us choose common cathode LED display to make the function
- Thus, we can make the truth-table
- Obviously, the BCD system only goes from 0 to 9, while we have 16 rows for 4 inputs
- In the other rows, we fill all the outputs as don't care, because we are sure that these are not going to be input anyway (trust the engineer before you)
- Thus, we have six don't care conditions



A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	1	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

Encoder/decoder

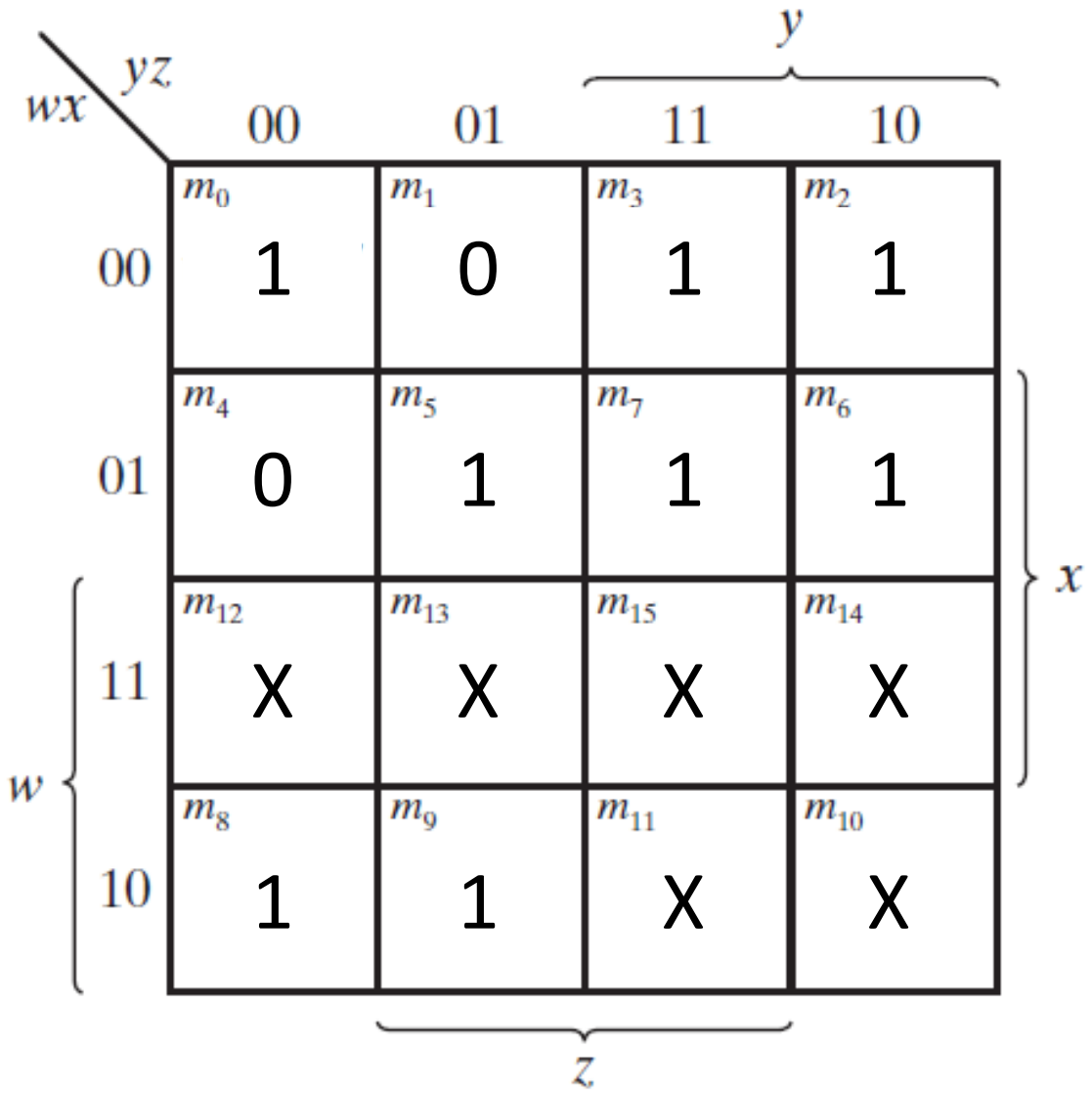
- Thus, the K-map for output a will look like this
- We have two clusters of 8: y and w
- Two clusters of four: xz and $x'z'$
- Thus, the logic function for a can be:

$$F_a = y + w + xz + x'z'$$

- Or we can have PoS as:
- One cluster of two: $xy'z'$
- One min-term: $w'x'y'z$
- Thus,

$$F_a = (x' + y + z)(w + x + y + z')$$

- This can be done for all the other outputs



Encoder/decoder