PATRIoT Lab

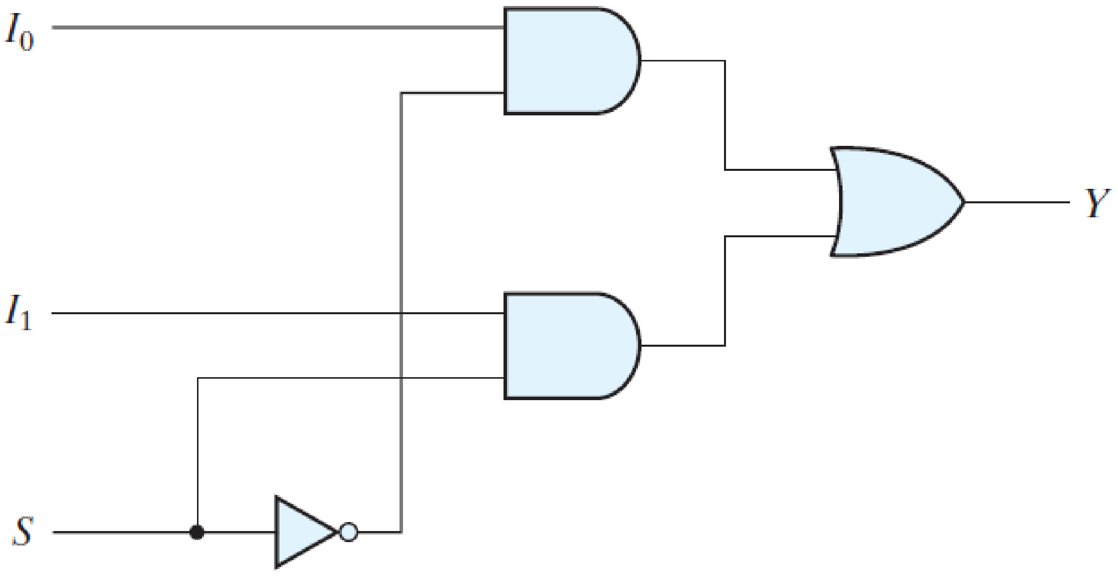
Lecture 12 – Combinational logic circuits

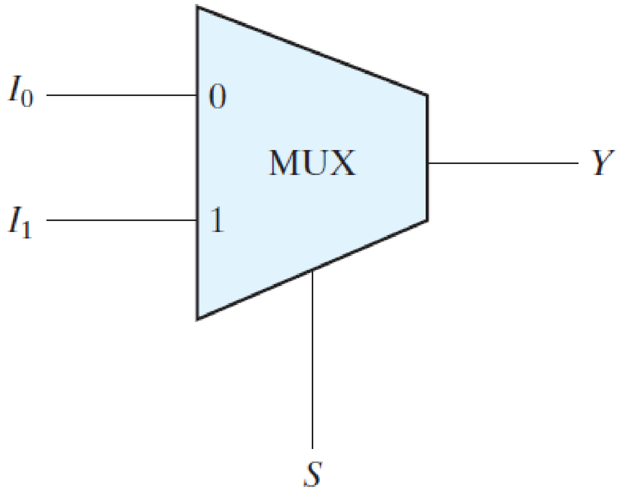
Dr. Aftab M. Hussain,

Assistant Professor, PATRIoT Lab, CVEST

Multiplexer

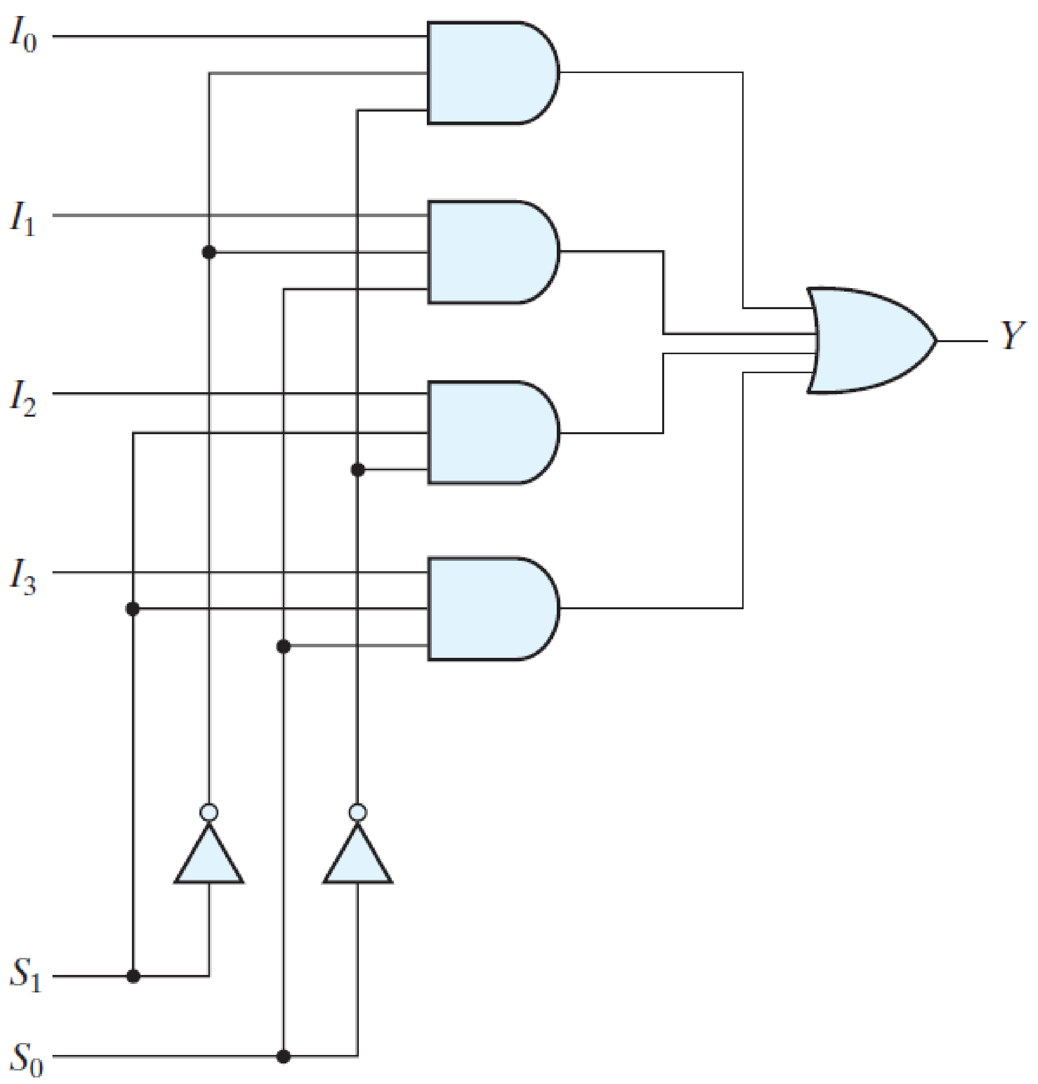
- A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line
- The selection of a particular input line is controlled by a set of selection lines
- Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected
- A two-to-one-line multiplexer connects one of two 1-bit sources to a common destination
- The multiplexer acts like an electronic switch that selects one of two sources
- The block diagram of a multiplexer (also called MUX) is sometimes depicted by a wedge-shaped symbol





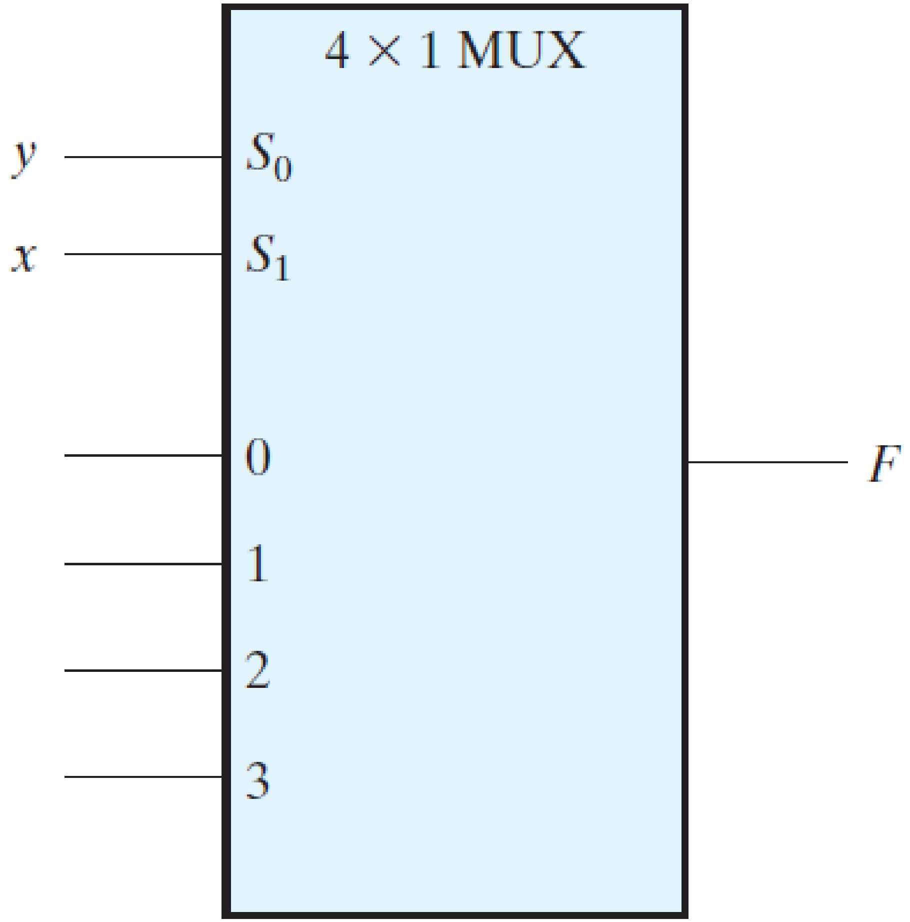
Multiplexer

- Similarly, we can make a four-to-one-line multiplexer
- Each of the four inputs, I_0 through I_3, is applied to one input of an AND gate
- The outputs of the AND gates are applied to a single OR gate that provides the one-line output
- A multiplexer is also called a *data selector*, since it selects one of many inputs and steers the binary information to the output line
- In general, a 2^n-to-1-line multiplexer is constructed ANDing each input line with 2^n minterms
- The outputs of the AND gates are applied to a single OR gate



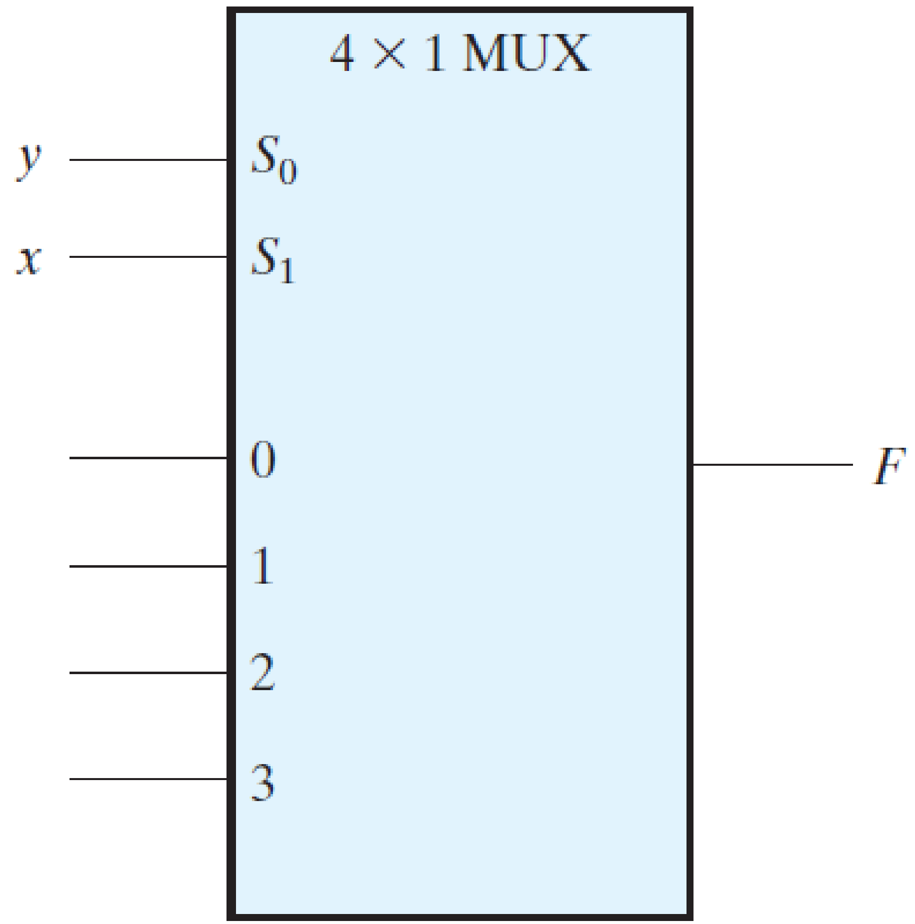
Multiplexer

- We can use MUXes to implement Boolean functions
- The minterms of a function are generated in a multiplexer by the circuit associated with the selection inputs
- The individual minterms can be selected by the data inputs, thereby providing a method of implementing a Boolean function of *n* variables with a multiplexer that has *n* selection inputs and 2^n data inputs, one for each minterm
- Example: implement EXOR



Multiplexer

- There is a neat trick to obtain a more efficient method for implementing a Boolean function of *n* variables with a multiplexer that has *n* - 1 selection inputs
- The first *n* - 1 variables of the function are connected to the selection inputs of the multiplexer
- The remaining single variable of the function is used for the data inputs
- Say we have a three variable function F(x,y,z)
- We take a 4to1 MUX (with two input select lines) and each data input of the multiplexer will be *z*, *z'*, 1, or 0



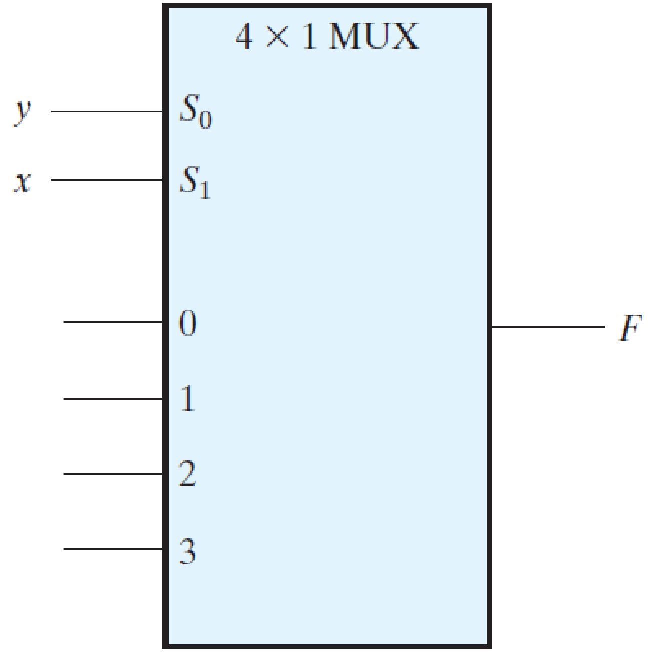
Multiplexer

- Consider the function:

$$F(x, y, z) = \sum(1,4,5,6)$$

- This function of three variables can be implemented with a four-to-one-line multiplexer
- The two variables *x* and *y* are applied to the selection lines in that order; *x* is connected to the S_1 input and *y* to the S_0 input
- The values for the data input lines are determined from the truth table of the function
- When *xy* = 00, output *F* is equal to *z* because *F* = 0 when *z* = 0 and *F* = 1 when *z* = 1
- This requires that variable *z* be applied to data input 0
- In a similar fashion, we can determine the required input to data lines 1, 2, and 3 from the value of *F* when *xy* = 01, 10, and 11, respectively

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



Multiplexer

- The general procedure for implementing any Boolean function of *n* variables with a multiplexer with *n* - 1 selection inputs and 2^{n-1} data inputs follows from the previous example

1. To begin with, Boolean function is listed in a truth table
2. Then the most significant *n* - 1 variables in the table are applied to the selection inputs of the multiplexer
3. For each combination of the selection variables, we evaluate the output as a function of the last variable
4. This function can be 0, 1, the variable, or the complement of the variable
5. These values are then applied to the data inputs in the proper order

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

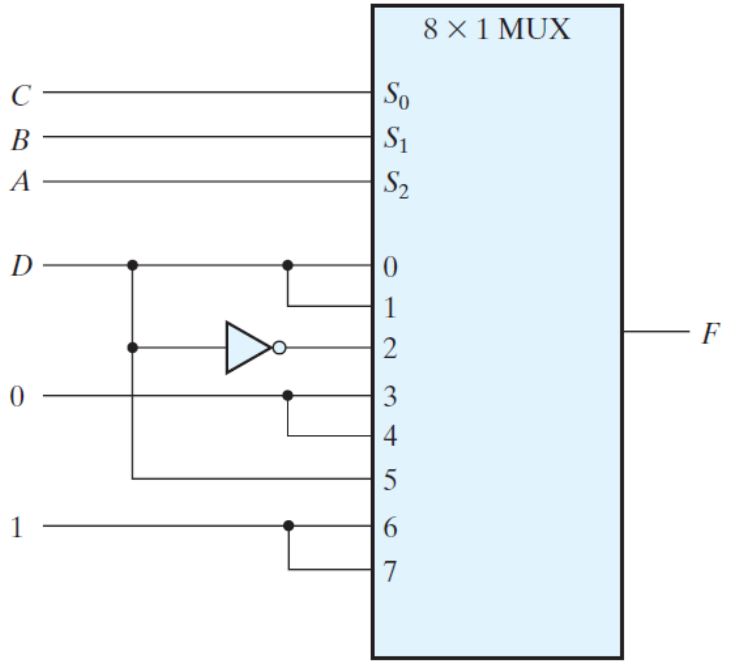
Multiplexer

- The general procedure for implementing any Boolean function of *n* variables with a multiplexer with *n* - 1 selection inputs and 2^{n-1} data inputs follows from the previous example

1. To begin with, Boolean function is listed in a truth table
2. Then the most significant *n* - 1 variables in the table are applied to the selection inputs of the multiplexer
3. For each combination of the selection variables, we evaluate the output as a function of the last variable
4. This function can be 0, 1, the variable, or the complement of the variable
5. These values are then applied to the data inputs in the proper order

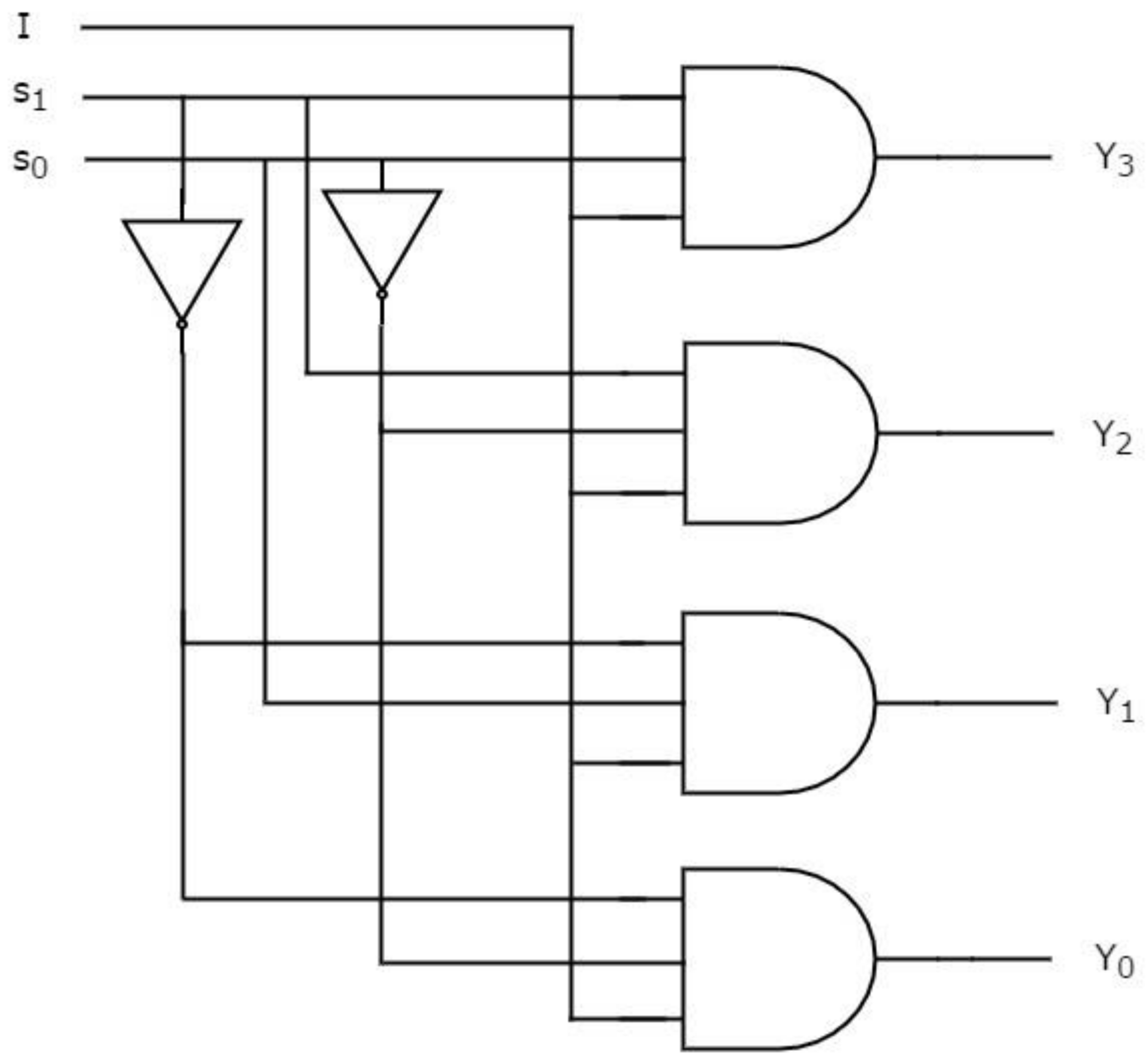
$$F(A,B,C,D) = \sum(1, 3, 4, 11, 12, 13, 14, 15)$$

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



Demultiplexer

- Demultiplexers do the exact opposite of MUX operation – take a single line input and direct it to an output line depending on the select line input
- 1to2^n Demux will have n select lines
- We again make minterms from the available inputs and AND it with the single data line
- Based on the input signals the particular output is connected to the data line and all other outputs are zero

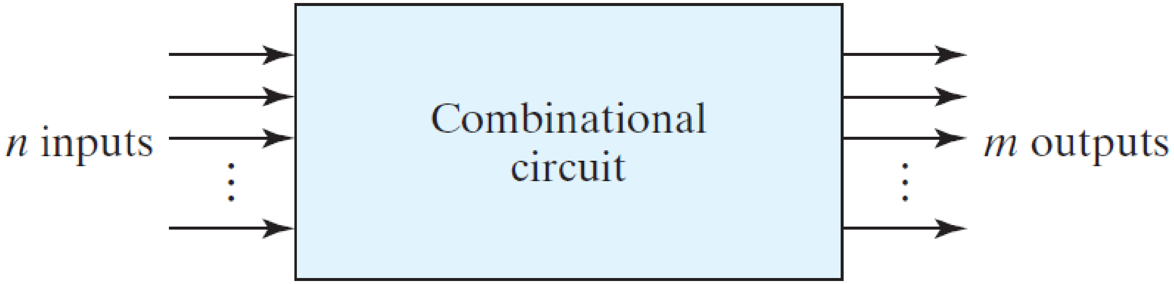


Combinational circuits

- 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

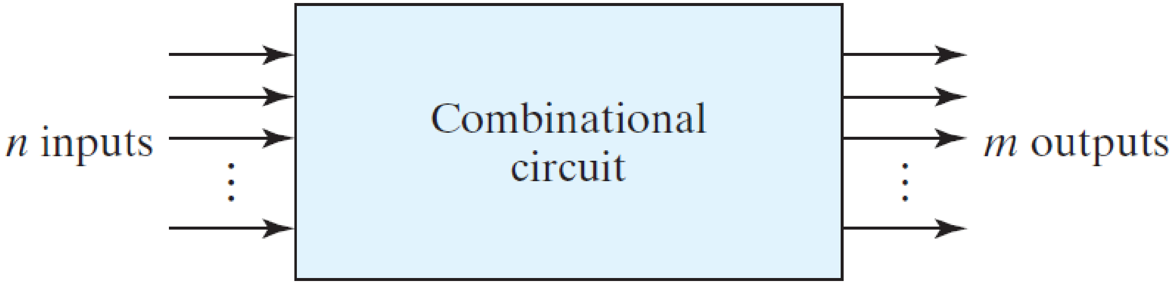
Combinational circuits

- 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 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:

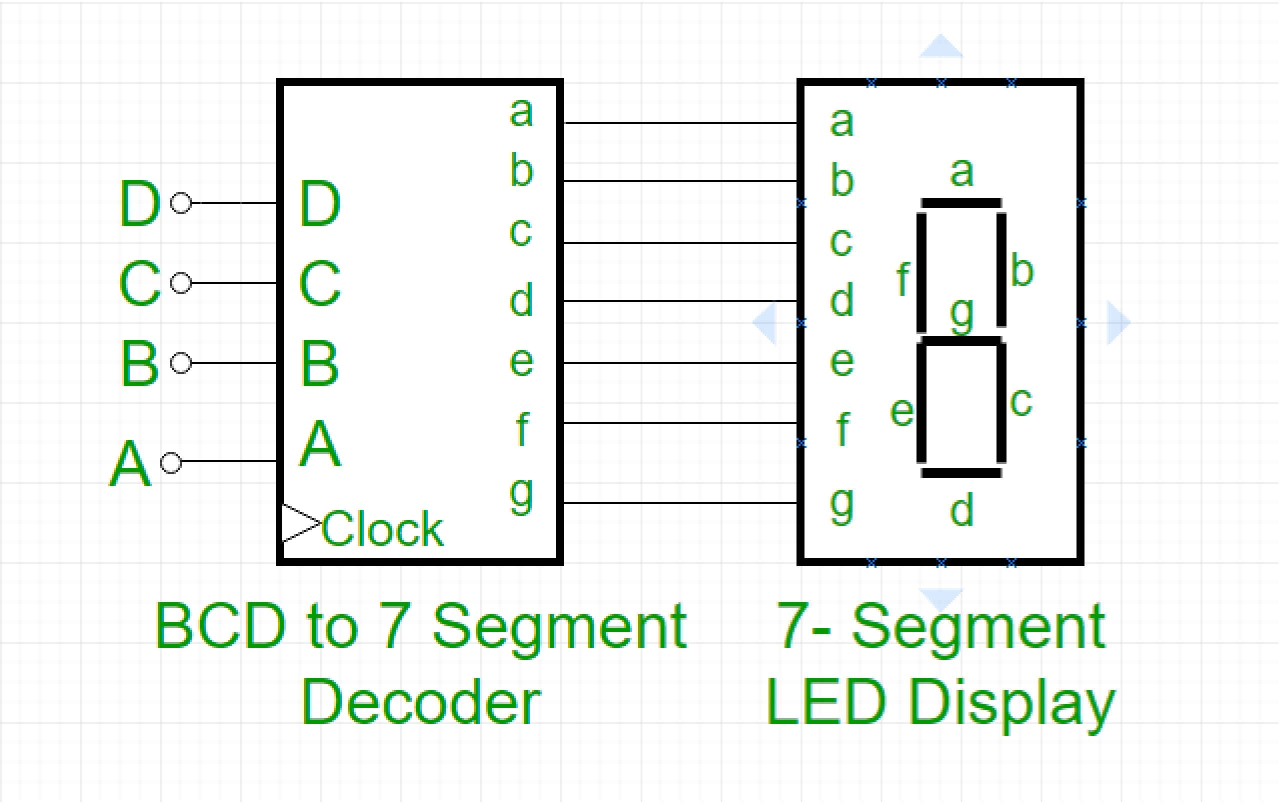
1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each
2. Derive the truth table that defines the required relationship between inputs and outputs
3. Make the K-map, if necessary
4. Obtain the simplified Boolean functions for each output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

- Here we go…

The 7-segment decoder

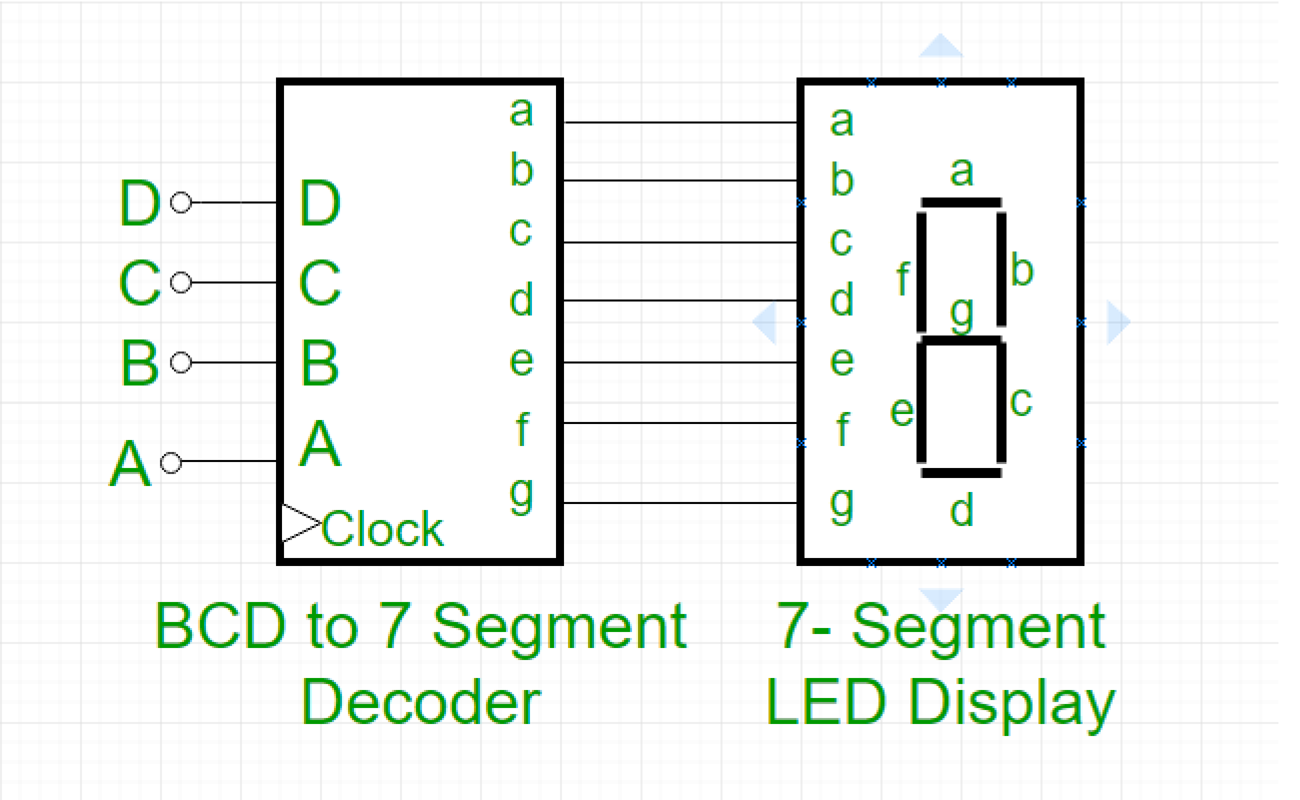
7-segment decoder

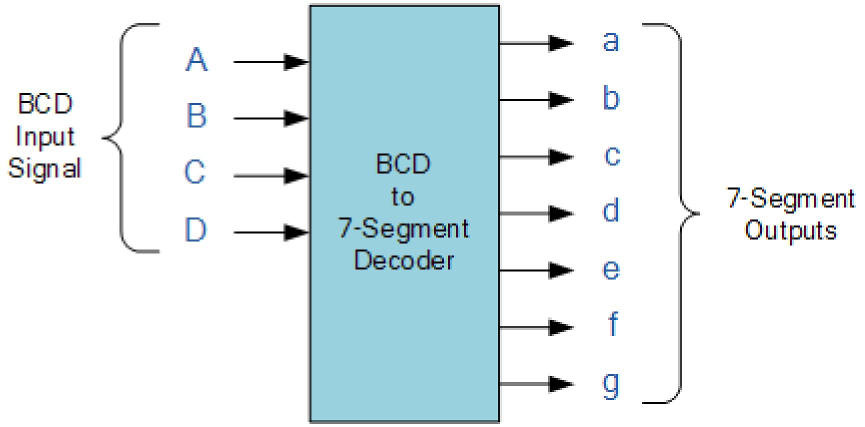
- We discussed many ways of representing symbols using binary variables – signed magnitude, 2's complement, BCD, ASCII etc.
- Going from one representation may be considered as an encoding/decoding operation
- Consider the practical problem of displaying binary numbers using a 7-segment LED display
-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



7-segment decoder

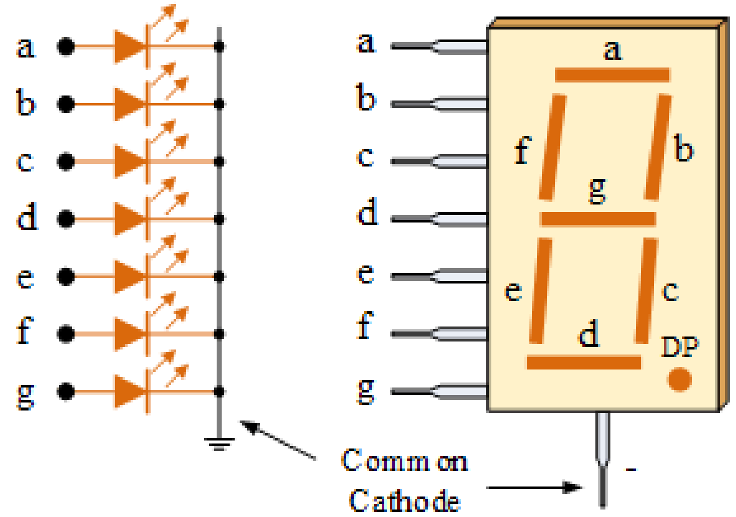
- First, we realize that there are 4 inputs and 7 outputs!
- Thus, the truth-table will be a 16 row table with 7 different columns for 7 outputs
- Hence, there will be 7 different functions we will be implementing to make this decoder
- The other thing we need to realize is whether a particular output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

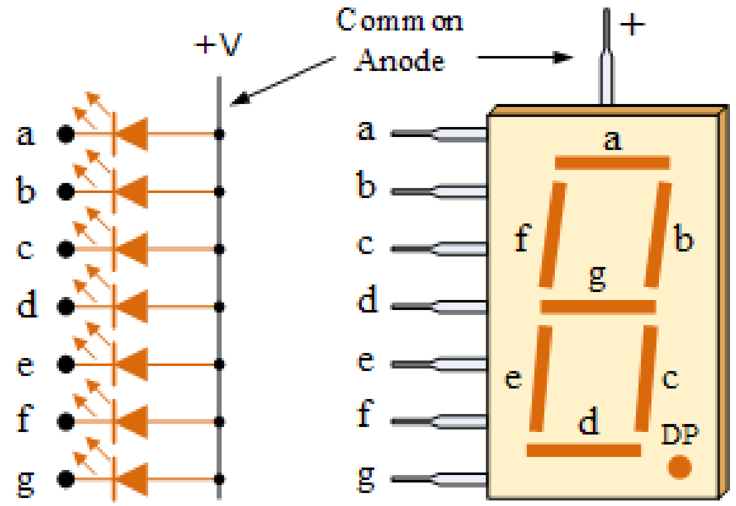




7-segment decoder

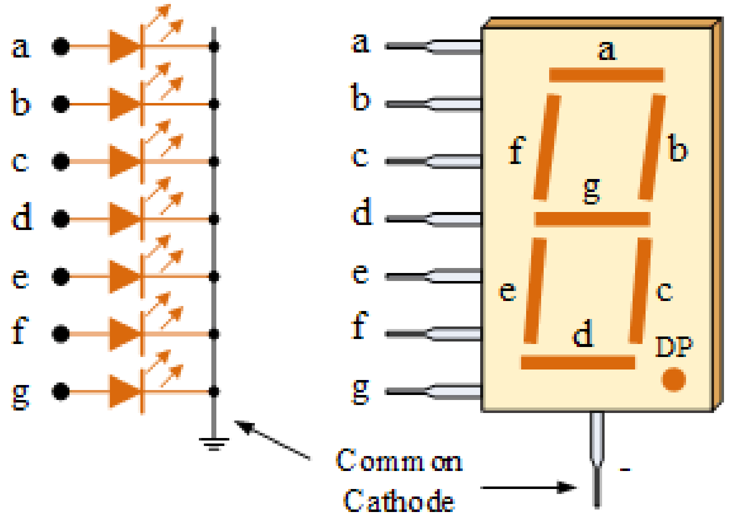
- The other thing we need to realize is whether a particular output should be HIGH or LOW for the LED to glow?
- To know the answer to this, we see that there are two types of 7-segment LED displays – common anode and common cathode
- The common cathode connects all LED cathodes to a common ground, thus,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





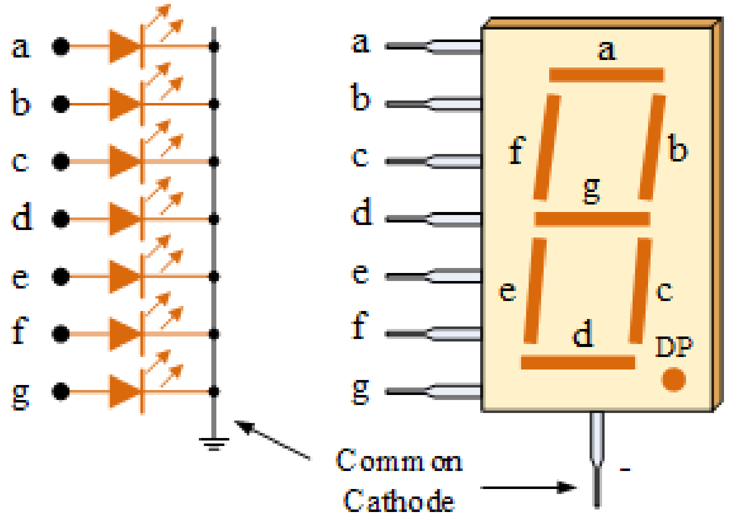
7-segment decoder

- Let us choose common cathode LED display to make the function
- Thus, we can make the truth-table



7-segment 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