PATRIOT Lab

Lecture 12 – Combinational logic circuits

Dr. Aftab M. Hussain,

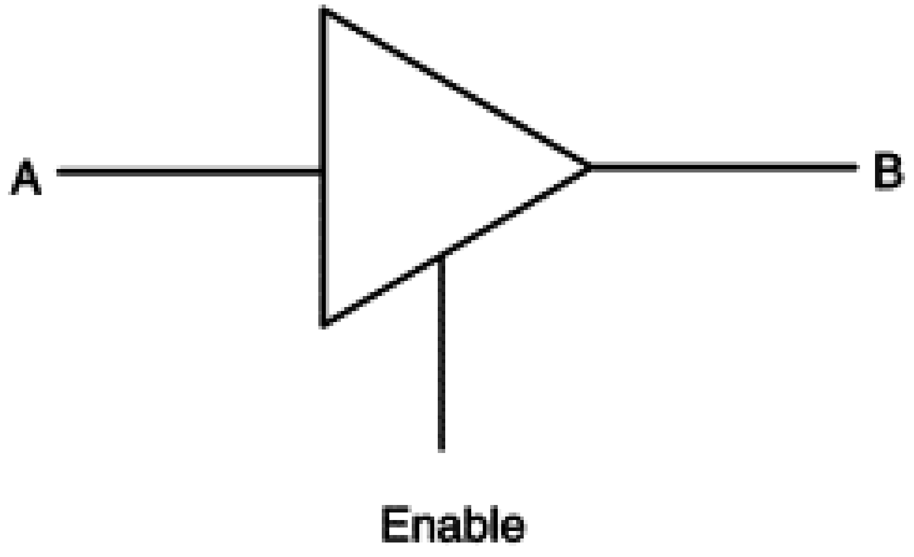
Assistant Professor, PATRIoT Lab, CVEST

The Other states

- This is REALLY important
- Apart from the two states of 0 and 1, there is a third state called the high impedance state denoted by Z
- When we say a particular pin is at HIGH or LOW state, we are assuming a driver behind it, i.e., the pin is *driven* to HIGH or LOW value
- This can be done through a transistor connecting the pin to either ground or Vcc
- However, if we do not connect a pin to either of HIGH or LOW, the pin is said to be in high impedance state or in Z state
- This concept is used extensively in the digital logic world to control buses

The Other states

- Most logic gates only output HIGH or LOW, the third state is generally obtained using tristate buffers
- These are used just before the bus connections



Enable	A	B
0	0	Z
0	1	Z
1	0	0
1	1	1

The Other states

- Another choice is that we can have either 0 or 1 (not both together, of course!)
- This is called the don’t care state – or a condition in the logic function that follows that we do not care what the output in a particular case is, i.e., for a particular set of inputs
- This is represented as X
- This can be either 0 or 1 and both are equally acceptable while forming logic circuits for a given function

The Other states

- Consider a simple two variable statement: If A, then what is B?
- One way we can interpret this: we need to know the value of B when A is TRUE, but when A is FALSE, we DON'T CARE!
- If this is the case, we can make the truth table for the function as shown
- In this case, because X can take either 0 or 1 value, we can simply make the desired function using an AND gate or as transfer of B

A	B	F
0	0	X
0	1	X
1	0	0
1	1	1

The Other states

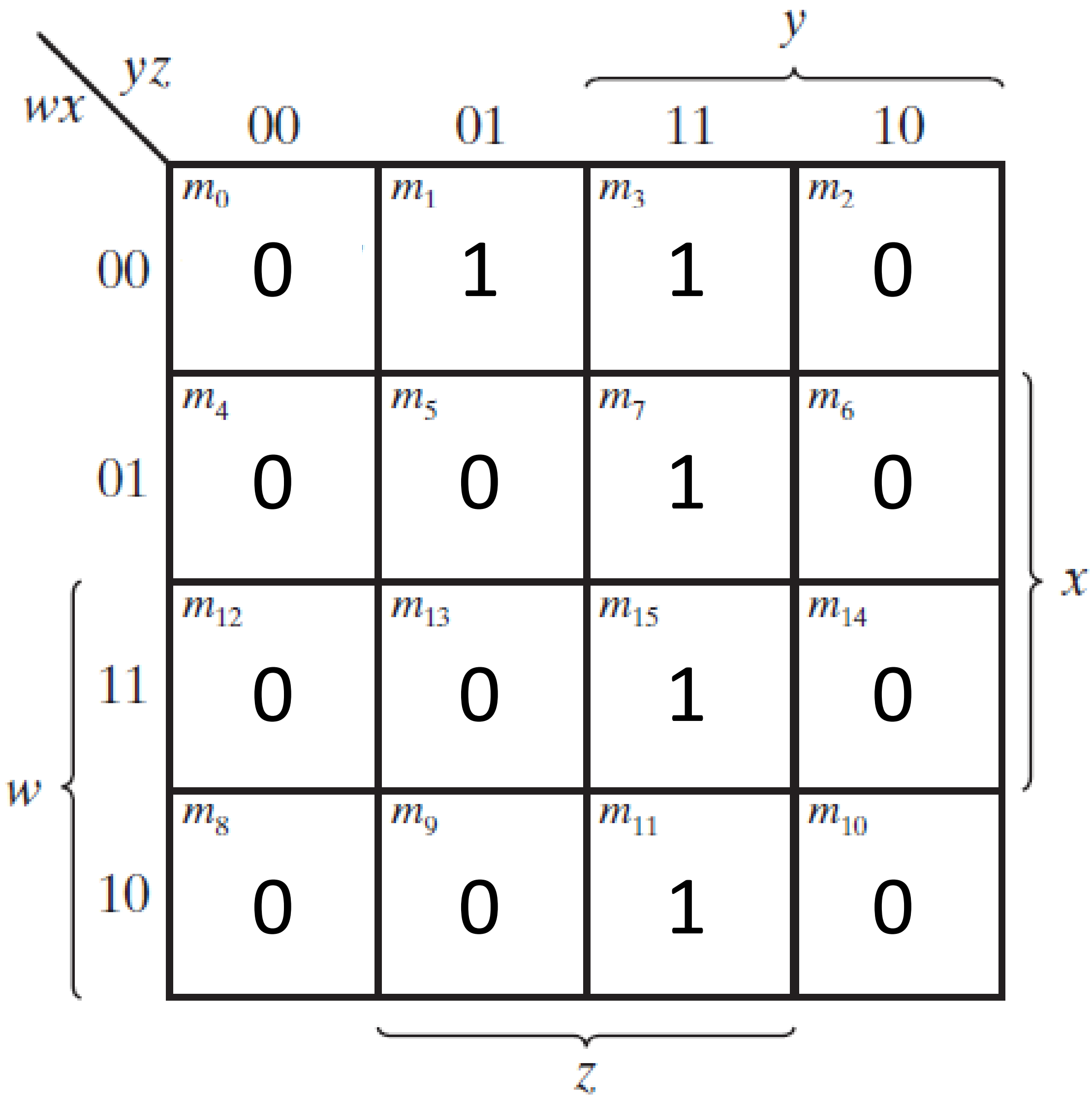
- Simplify the Boolean function

  $$F\ (w, x, y, z)\ = \sum(1, 3, 7, 11, 15)$$

- We have one cluster of four: yz and one cluster of two: $w'x'z$

- Thus, the function is

  $$F = yz + w'x'z$$



The Other states

- Now, consider the same function

$$F\ (w, x, y, z)\ = \sum(1, 3, 7, 11, 15)$$

- which has the don't-care conditions

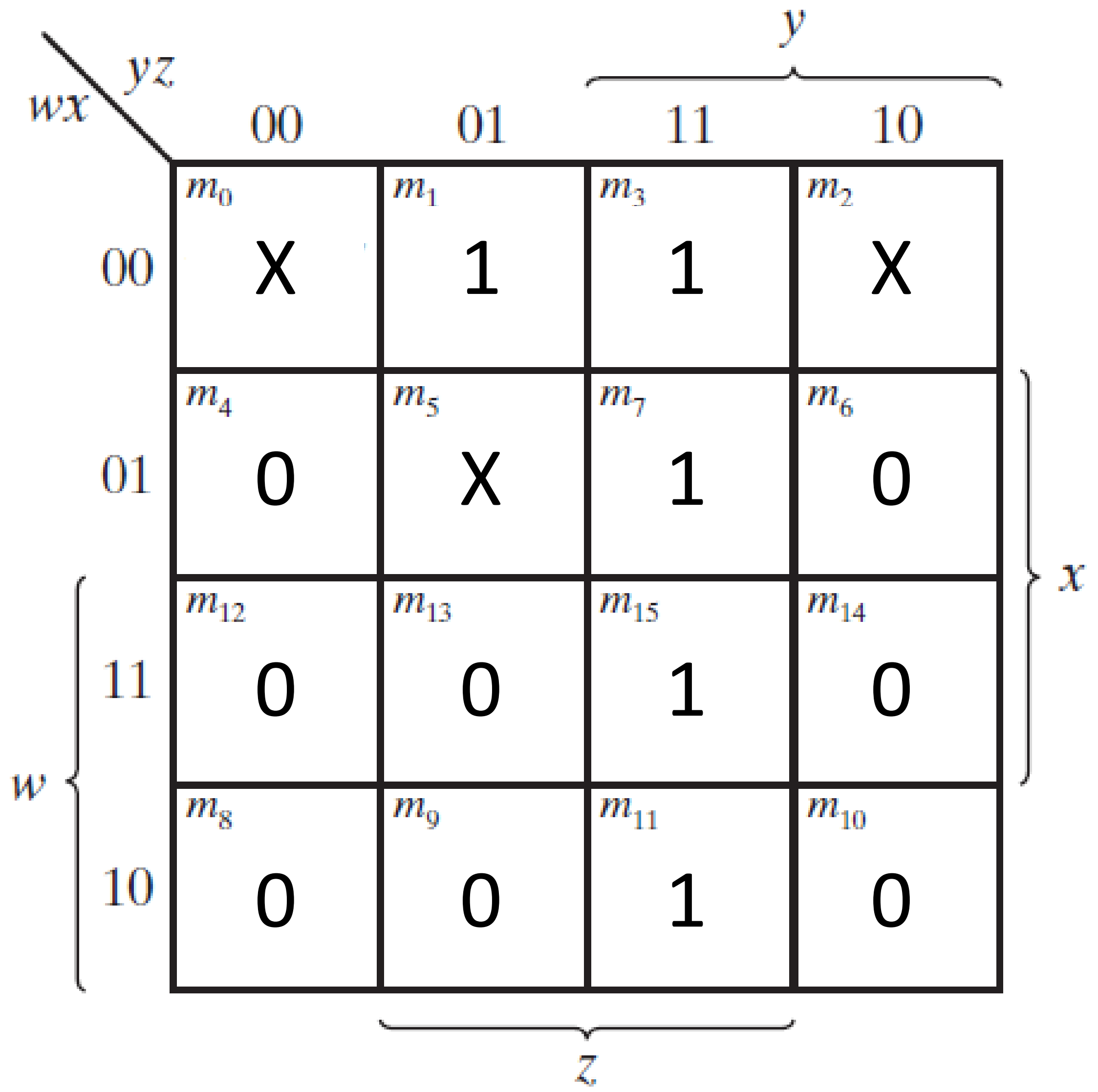
$$d\ (w, x, y, z)\ = \sum(0, 2, 5)$$

- Now, we have two clusters of four squares: yz and $w'z$
- This is because m_5 can be 1, and it is ok if m_0 and m_2 are 0
- Thus, the function is $F = yz + w'z$
- The function can also be simplified as $F = yz + w'x'$

$wx \backslash yz$	00	01	11	10
00	m_0 X	m_1 1	m_3 1	m_2 X
01	m_4 0	m_5 X	m_7 1	m_6 0
11	m_{12} 0	m_{13} 0	m_{15} 1	m_{14} 0
10	m_8 0	m_9 0	m_{11} 1	m_{10} 0

The Other states

- Functions $F = yz + w'z$ and $F = yz + w'x'$ are different functions
- However, both expressions include minterms 1, 3, 7, 11, and 15 that make the function *F* equal to 1
- The don't-care minterms 0, 2, and 5 are treated differently in each expression
- The first expression includes minterms 0 and 2 with the 1's and leaves minterm 5 with the 0's
- The second expression includes minterm 5 with the 1's and leaves minterms 0 and 2 with the 0's
- The two expressions represent two functions that are not equal but follow the logic statement



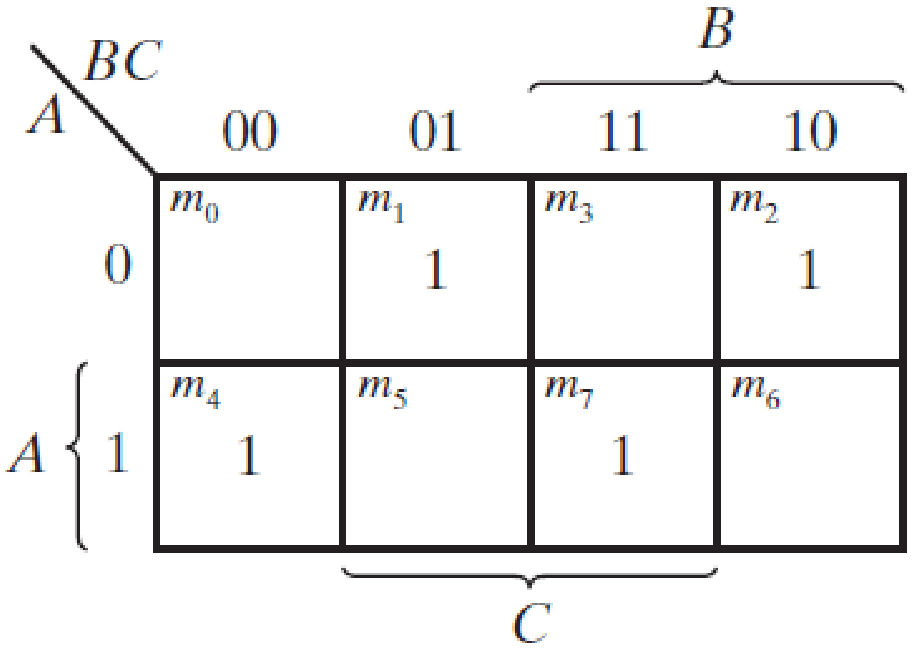
The Other states

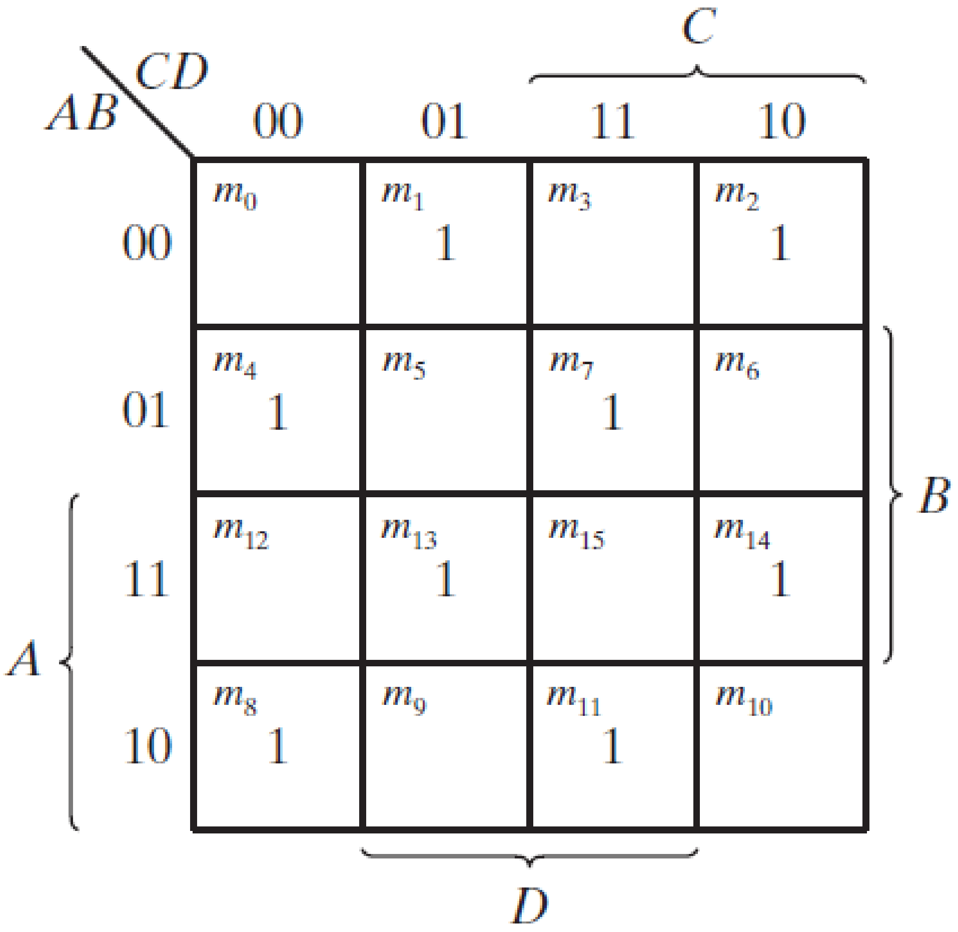
- The key question is: Are these the simplest possible representations for the given function? $F = yz + w'z$ and $F = yz + w'x'$
- What if we look at the product of sum simplification?
- We can get a cluster of 8 squares: z'
- And a cluster of four squares: wy'
- Thus, the function can be represented as $F = z(w' + y)$

$wx \backslash yz$	00	01	11	10
00	m_0 X	m_1 1	m_3 1	m_2 X
01	m_4 0	m_5 X	m_7 1	m_6 0
11	m_{12} 0	m_{13} 0	m_{15} 1	m_{14} 0
10	m_8 0	m_9 0	m_{11} 1	m_{10} 0

ExOR gate

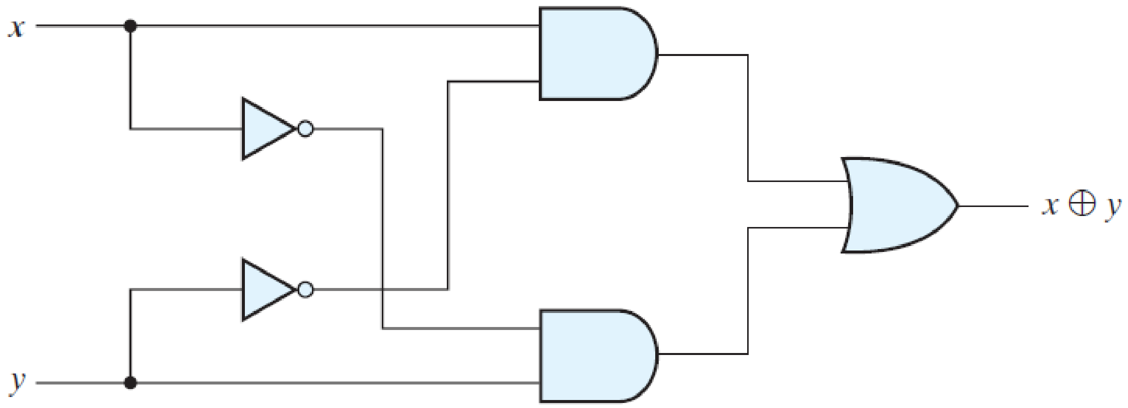
- ExOR is weird because there is no easy way to simply the function using K-maps

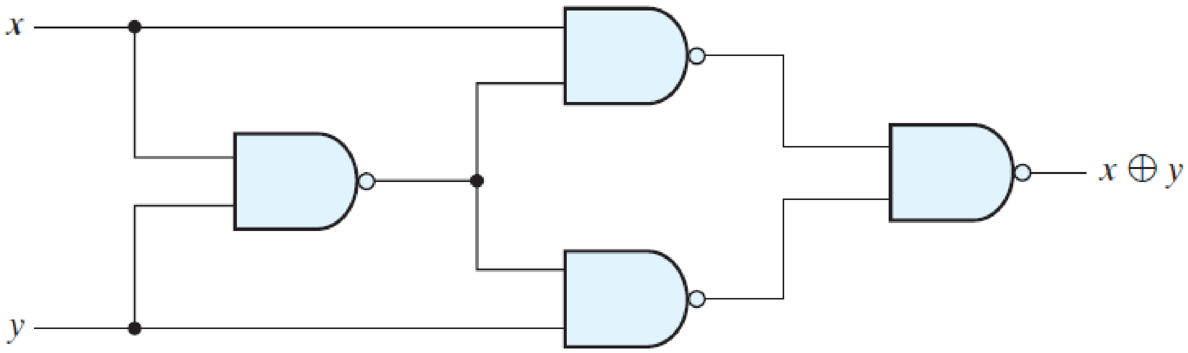




ExOR gate

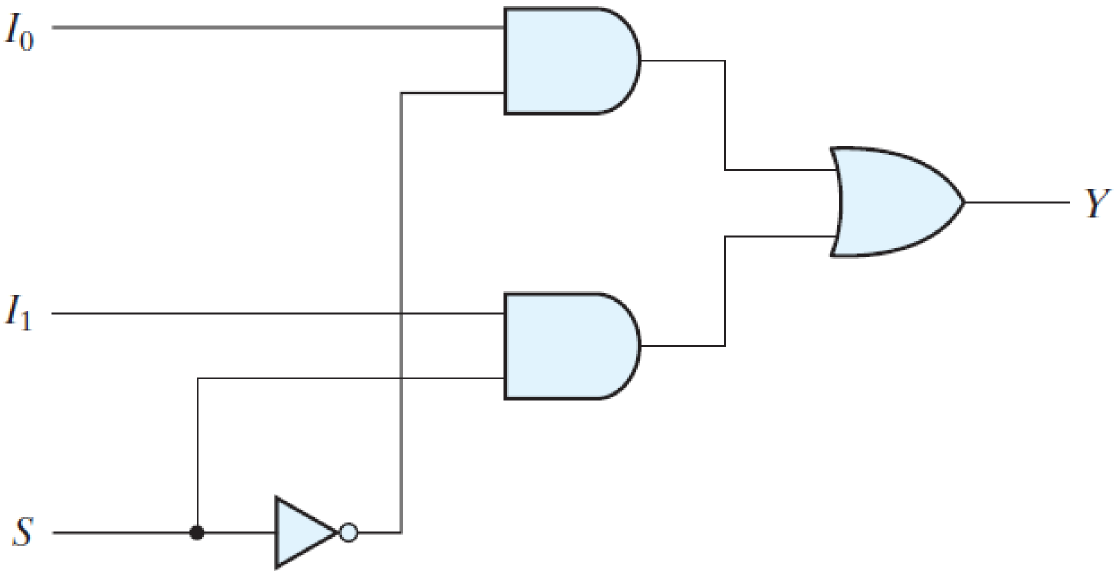
- ExOR is weird because there is no easy way to simply the function using K-maps

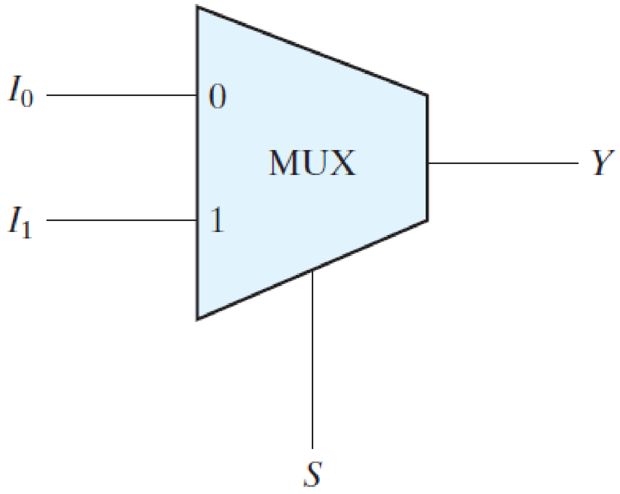




Multiplexer

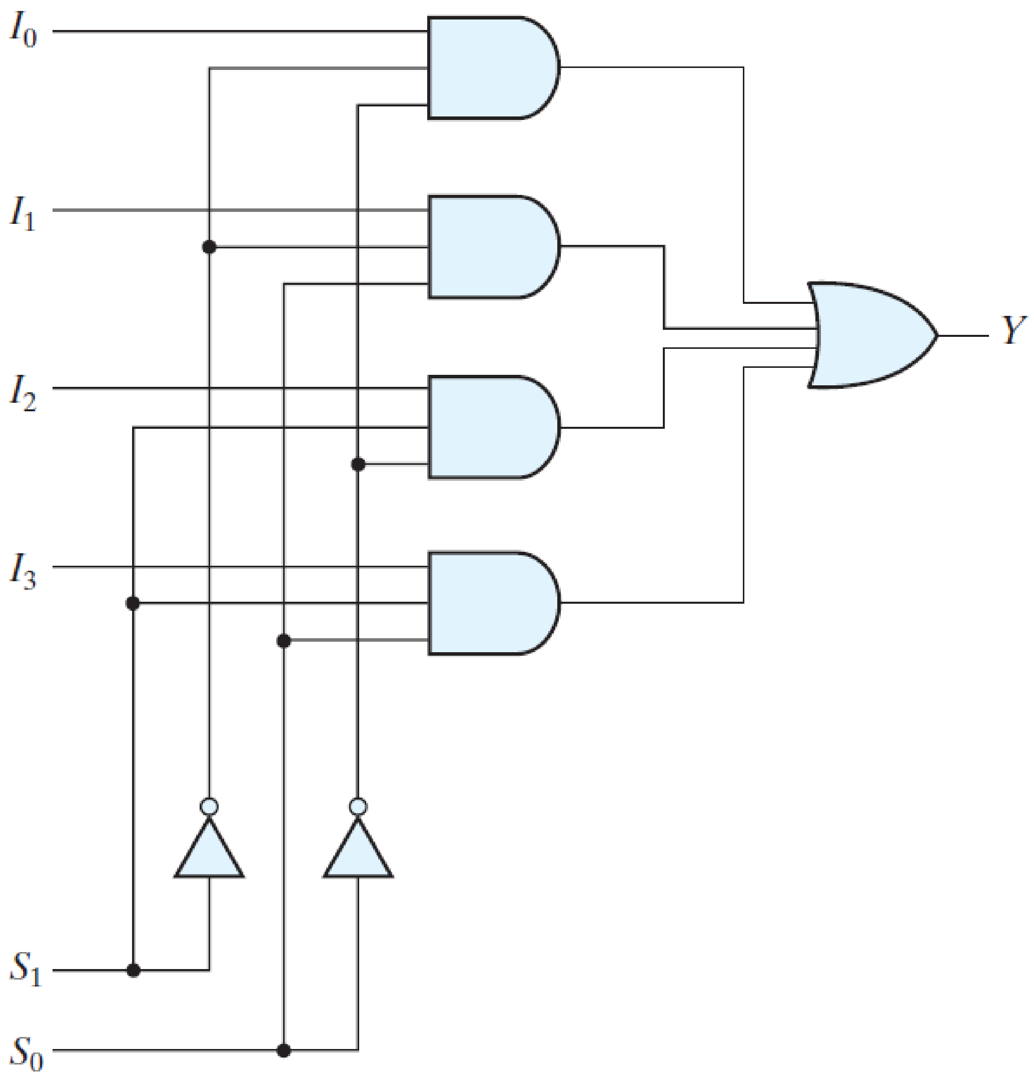
- A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line
- The selection of a particular input line is controlled by a set of selection lines
- Normally, there are 2^n input lines and n selection lines whose bit combinations 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





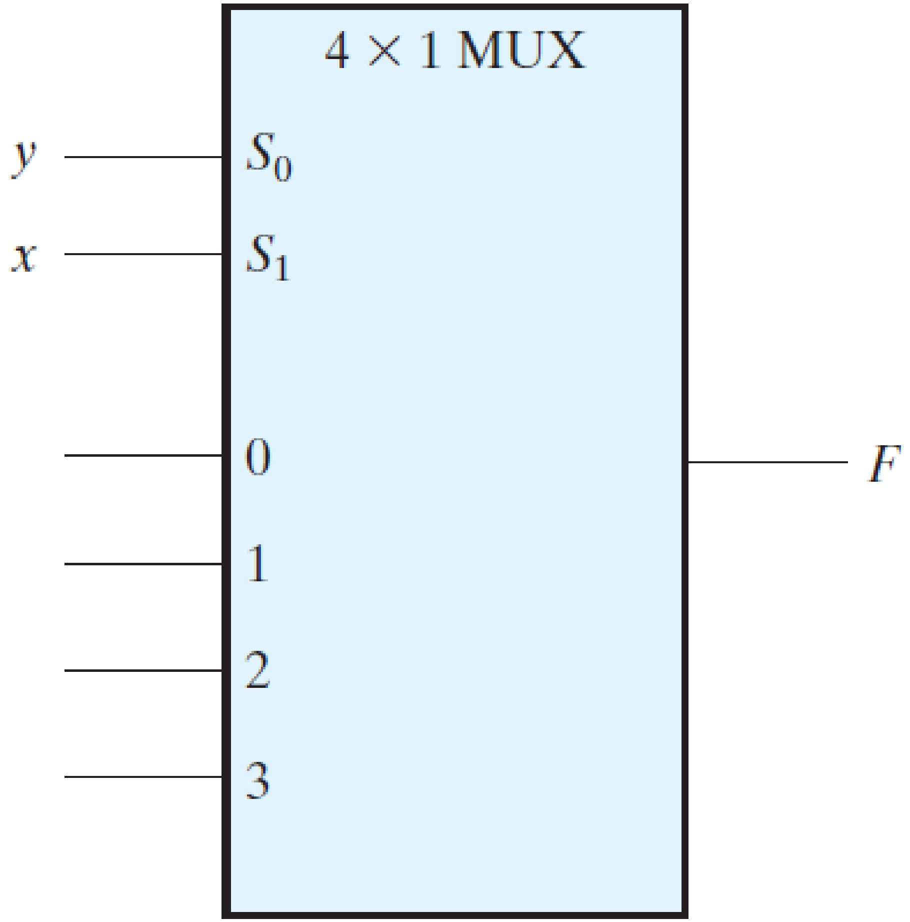
Multiplexer

- 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



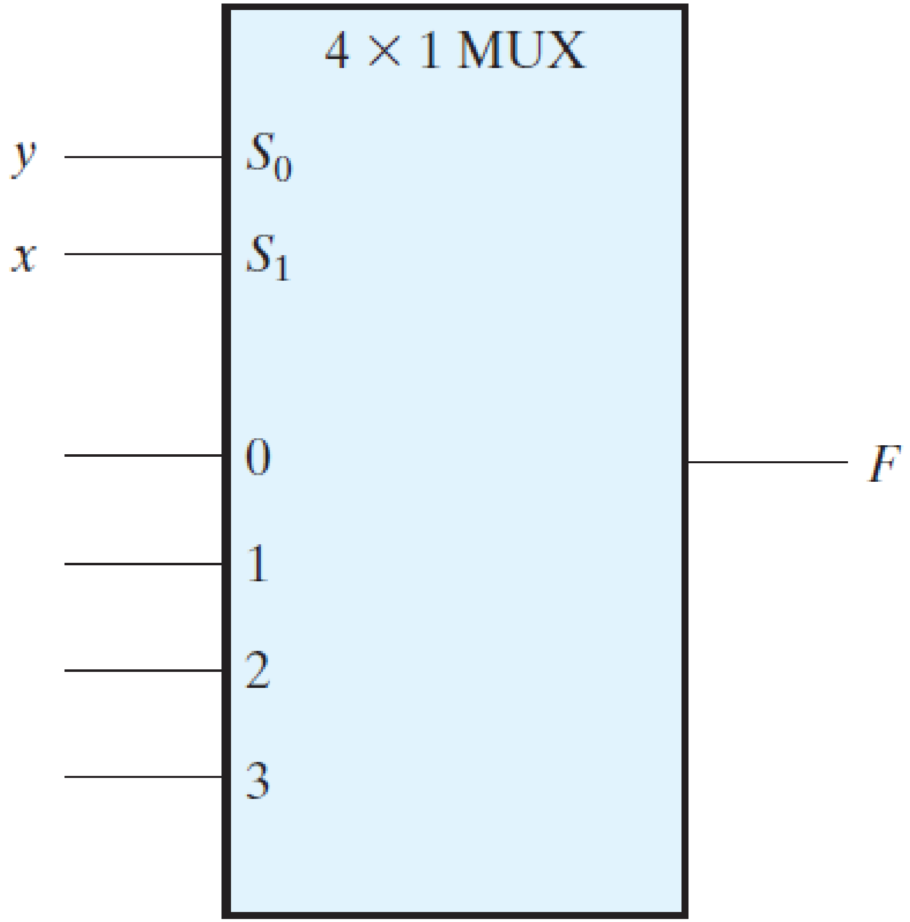
Multiplexer

- 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



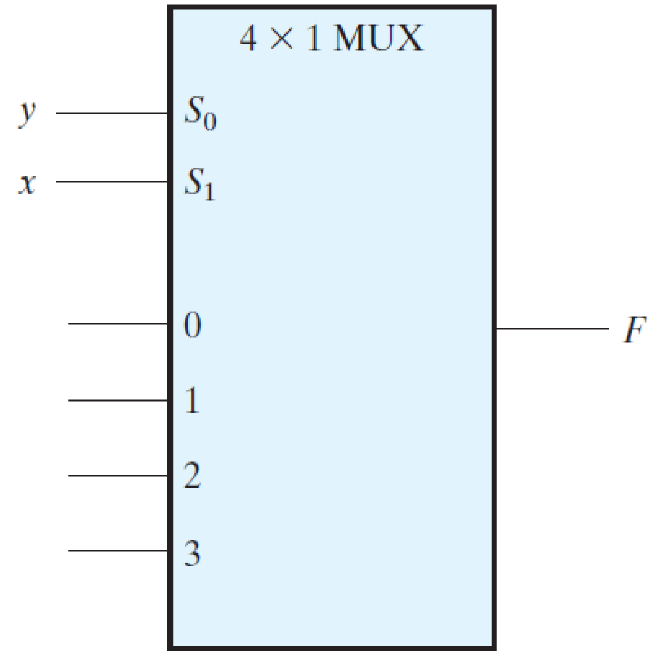
Multiplexer

- 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

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

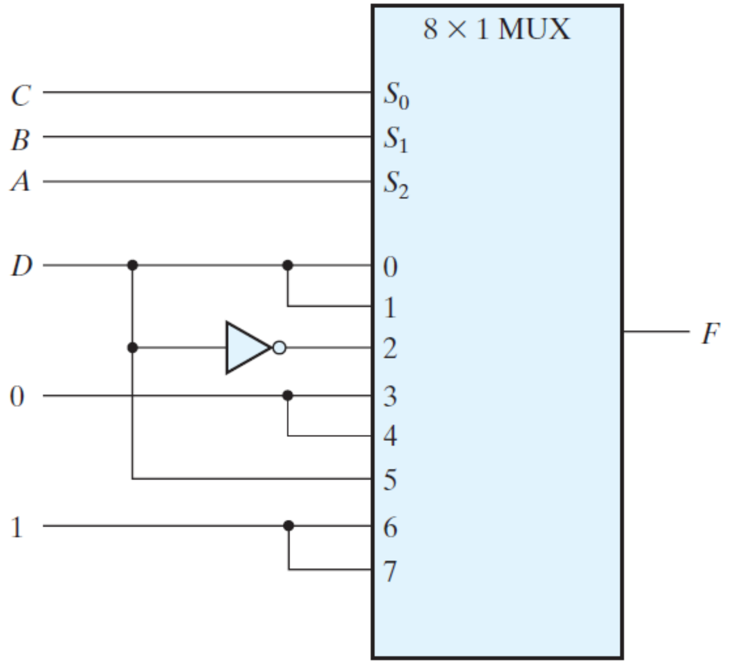
Multiplexer

- 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