PATRIoT lab

Lecture 11 – Logic implementation

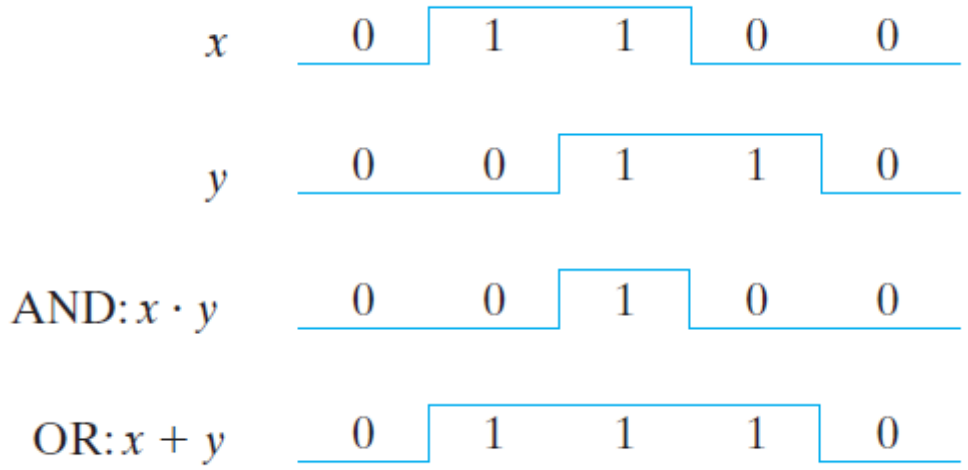
Dr. Aftab M. Hussain,

Assistant Professor, PATRIoT Lab, CVEST

Timing diagrams

- With logic functions, it is always nice to visualize the various conditions giving a particular output
- One way of visualizing is the truth table, another way can be the timing diagram of a particular function
- Timing diagrams are useful to indicate the sequence of events in large combinational circuits

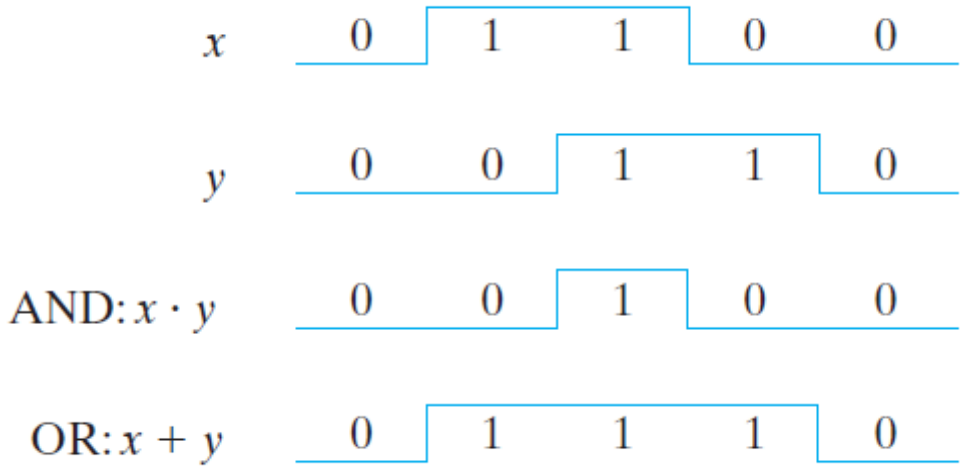
AND			OR		
x	y	$x \cdot y$	x	y	$x + y$
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	1



Timing diagrams

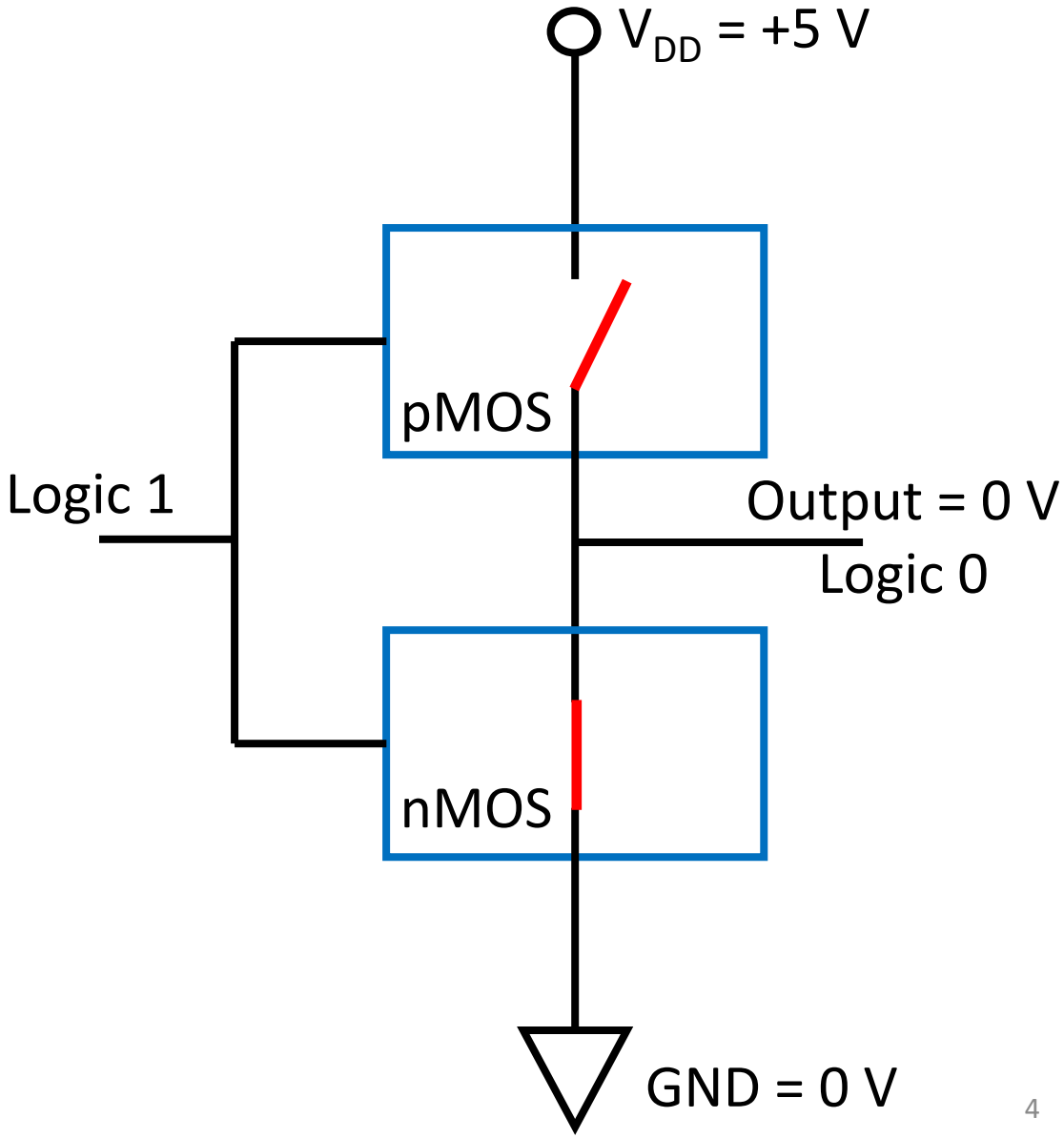
- The timing diagrams illustrate the idealized response of each gate to the four input signal combinations
- The horizontal axis of the timing diagram represents the time, and the vertical axis shows the signal as it changes between the two possible voltage levels
- In reality, the transitions between logic values occur quickly, but not instantaneously
- The low level represents logic 0, the high level logic 1

AND			OR		
x	y	$x \cdot y$	x	y	$x + y$
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	1



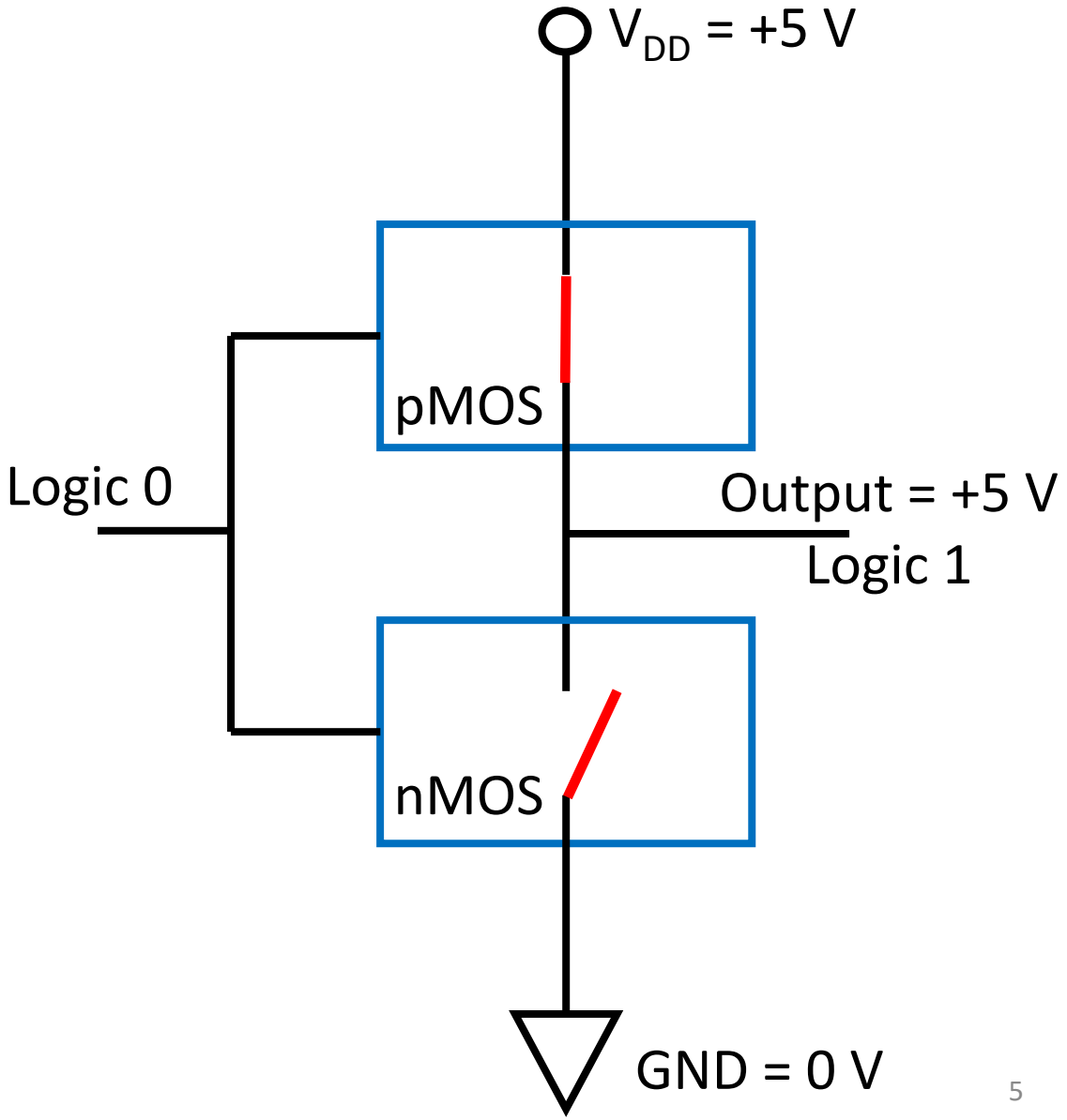
Implementing logic gates

- Typically, logic gates are implemented using transistors in CMOS architecture, where they act as switches, connecting VDD or GND to the output
- The switches are themselves driven using the logic states as inputs:
  - nMOS is on for logic 1 input and off for logic 0
  - pMOS is on for logic 0 input and off for logic 1
- When GND is connected, the output goes to logic 0
- When VDD is connected, the output goes to logic 1

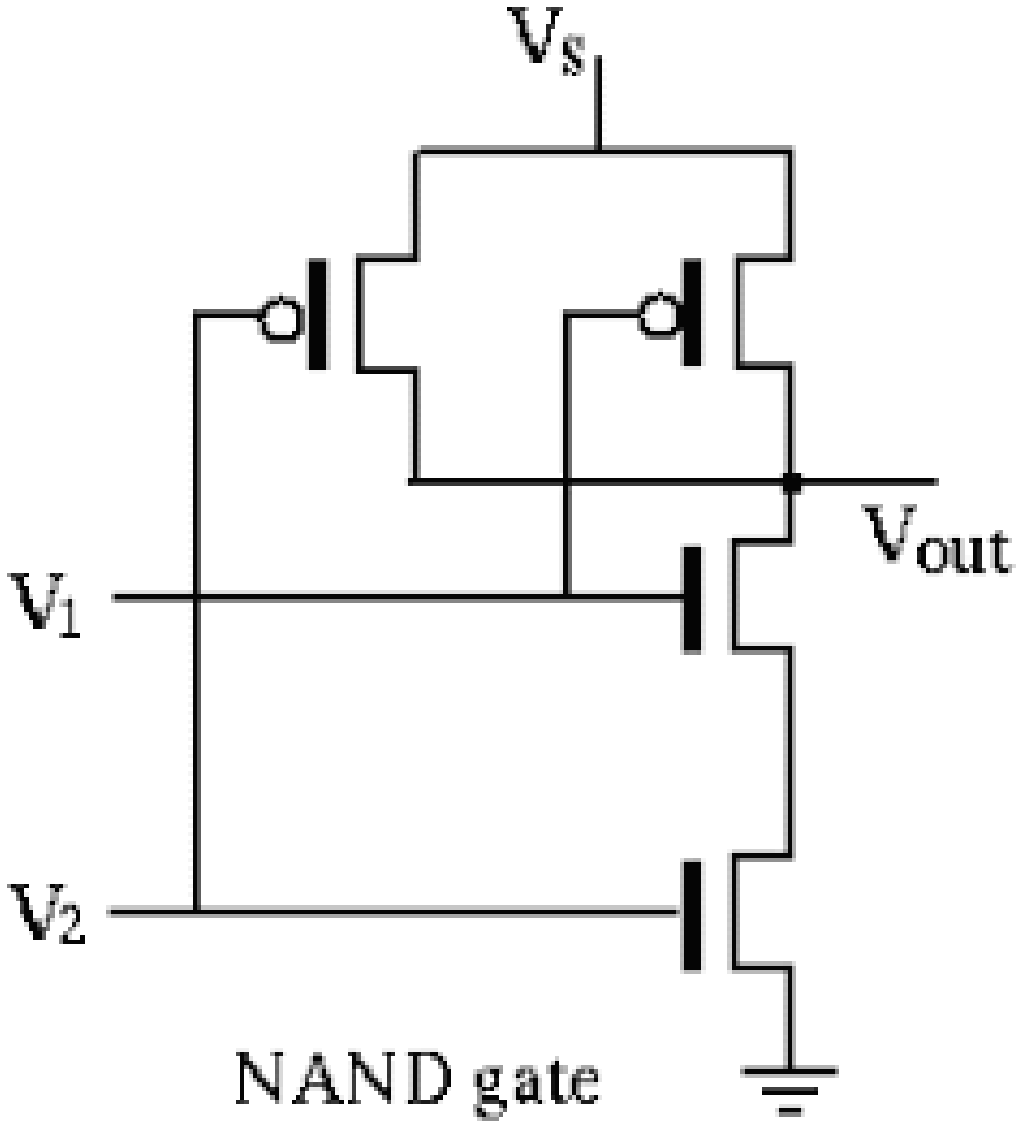


Implementing logic gates

- Typically, logic gates are implemented using transistors in CMOS architecture, where they act as switches, connecting VDD or GND to the output
- The switches are themselves driven using the logic states as inputs:
  - nMOS is on for logic 1 input and off for logic 0
  - pMOS is on for logic 0 input and off for logic 1
- When GND is connected, the output goes to logic 0
- When VDD is connected, the output goes to logic 1

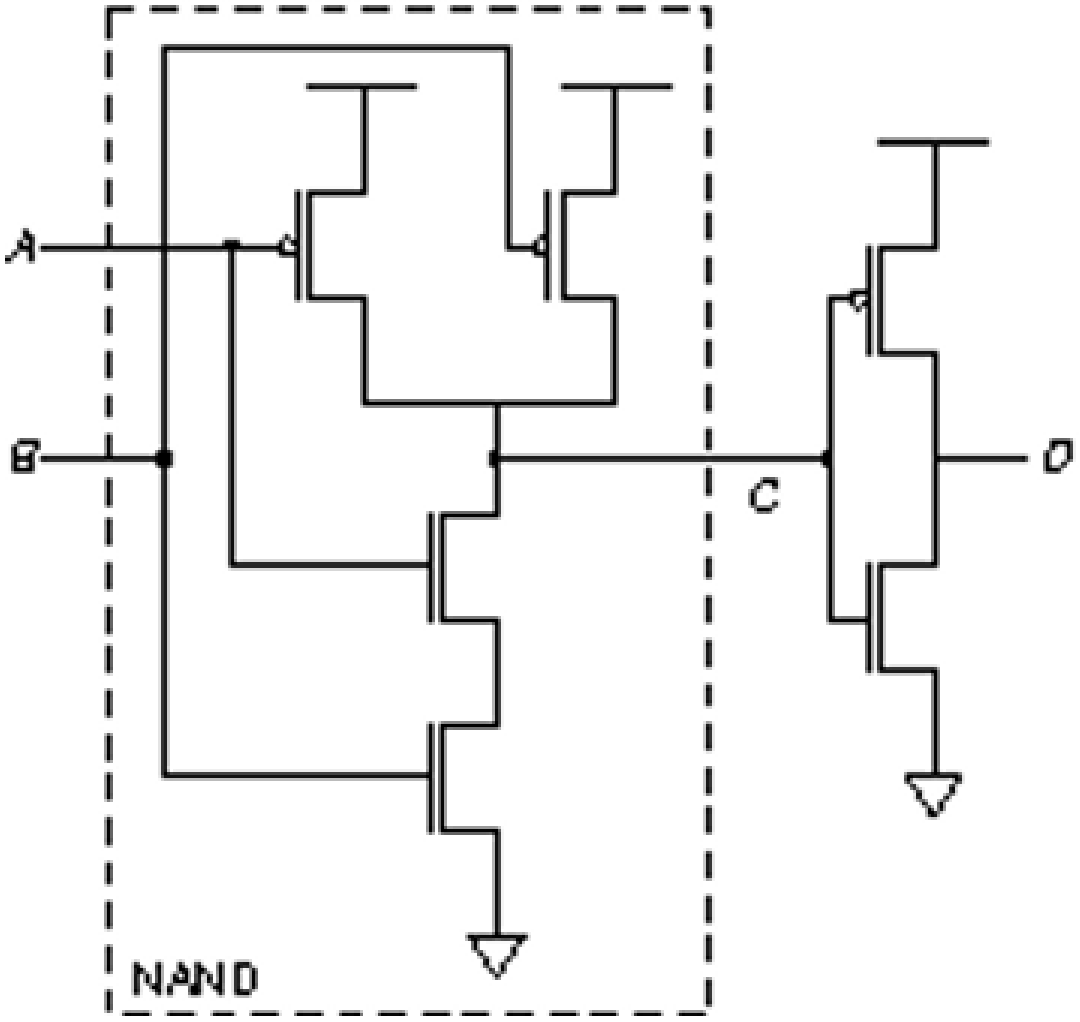


Implementing logic gates

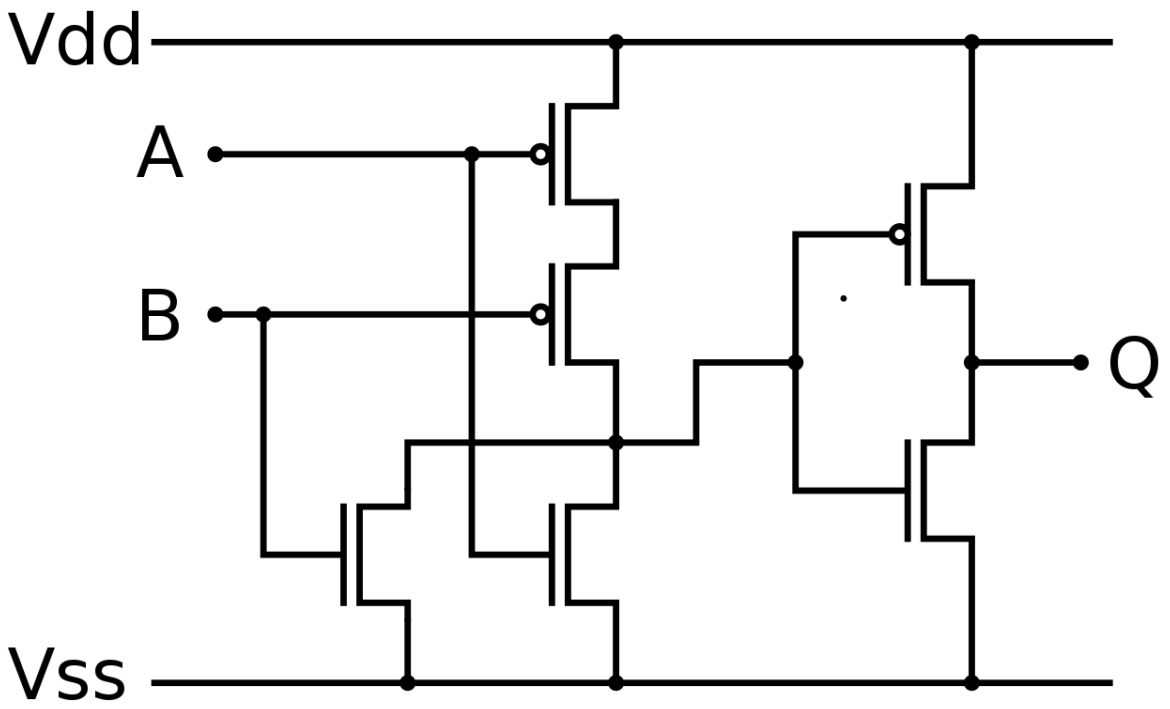


Implementing logic gates

AND gate

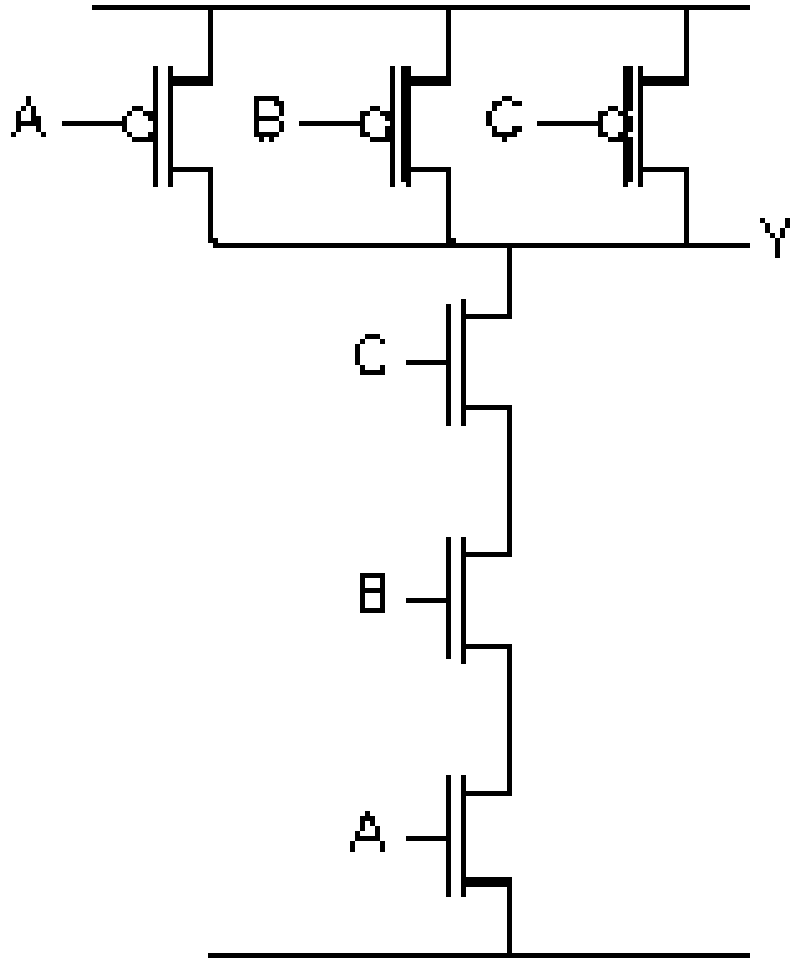


OR gate

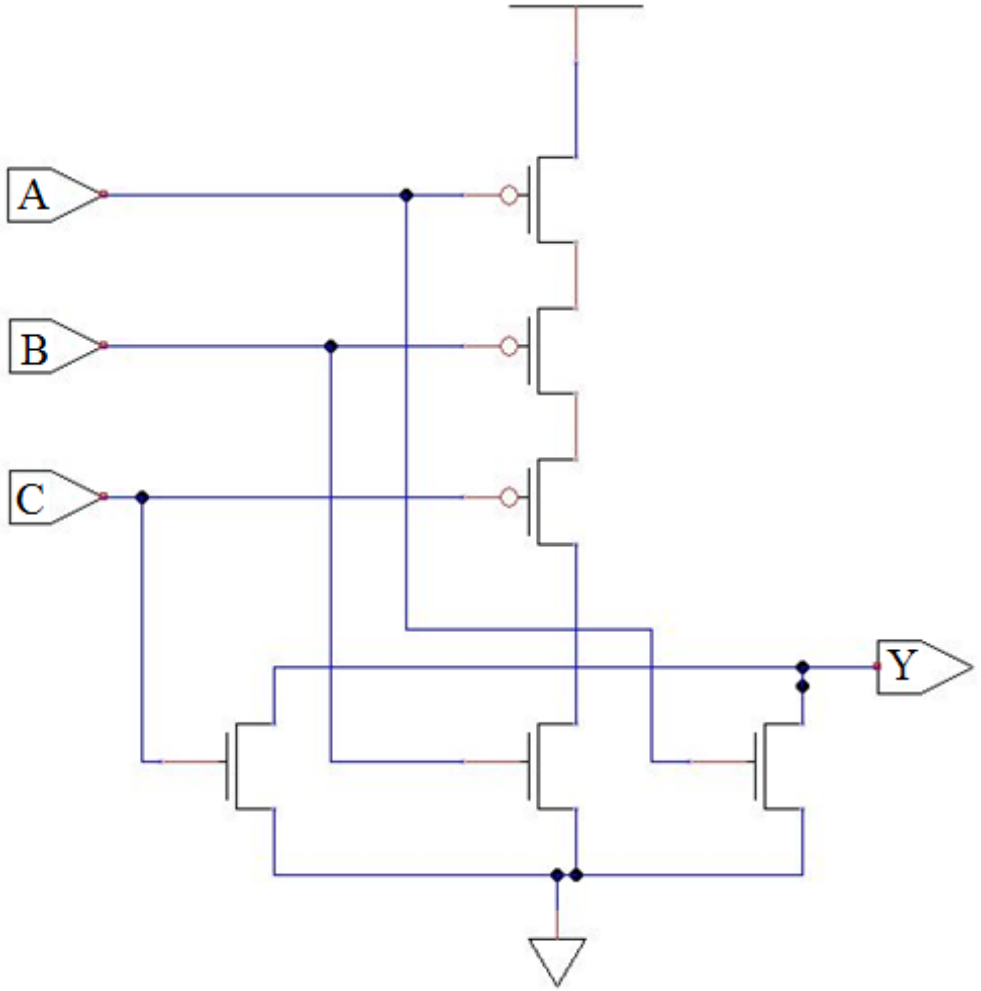


Implementing logic gates

3-input NAND gate



3-input NOR gate



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:

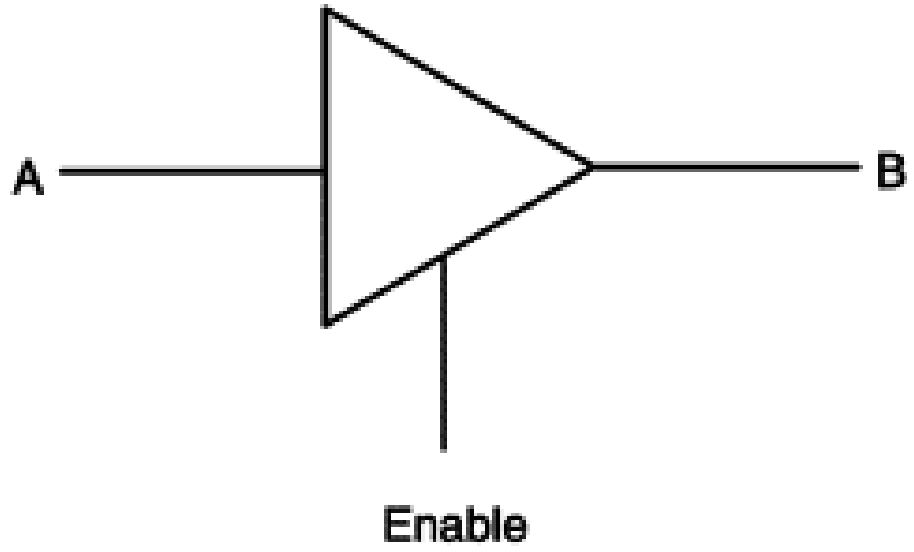
Gate	Inputs	No of Transistors
NOT	1	2
NAND	N	2N
NOR	N	2N
AND	N	2N+2
OR	N	2N+2

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

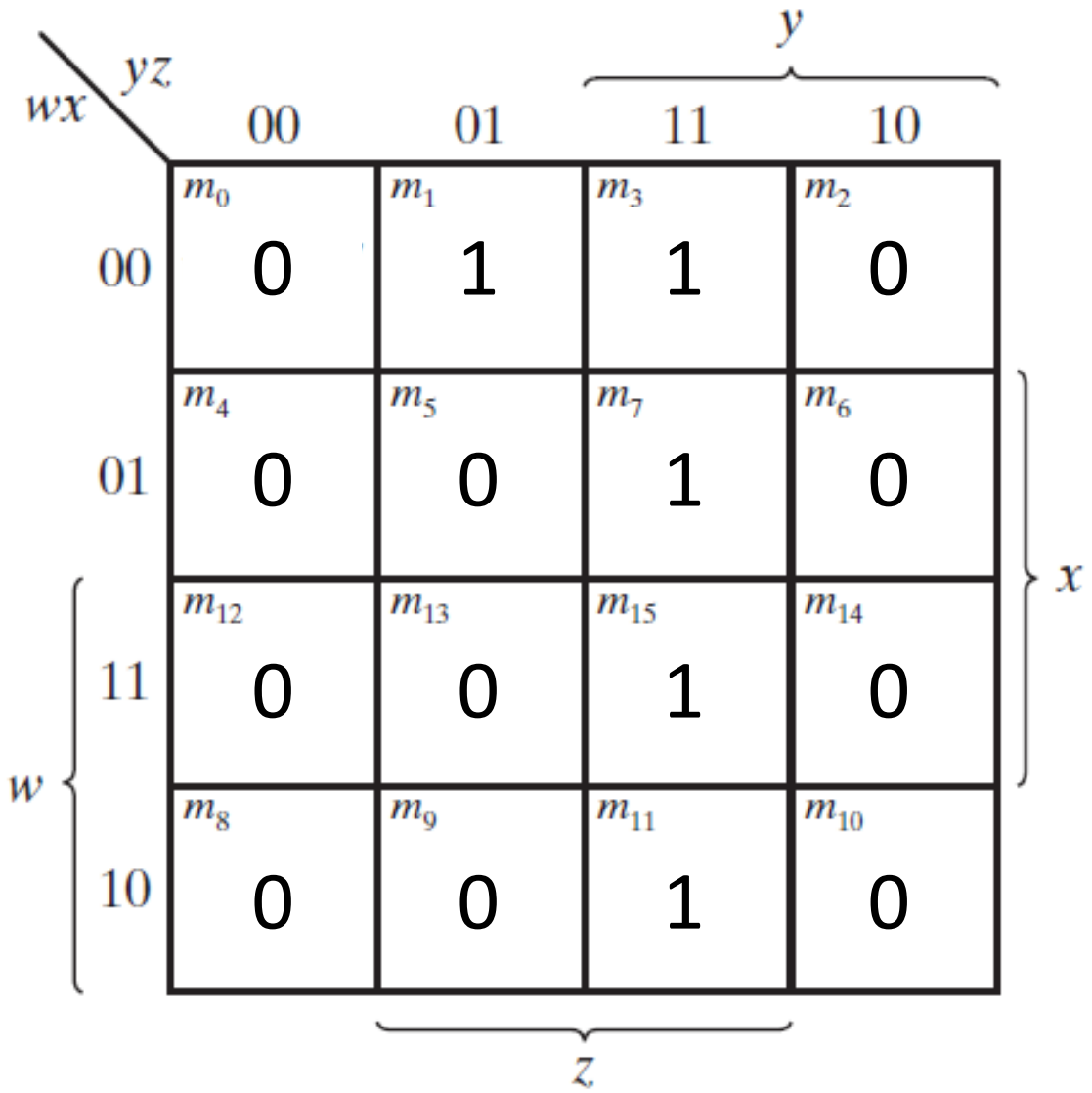
- Simplify the Boolean function

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

- 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

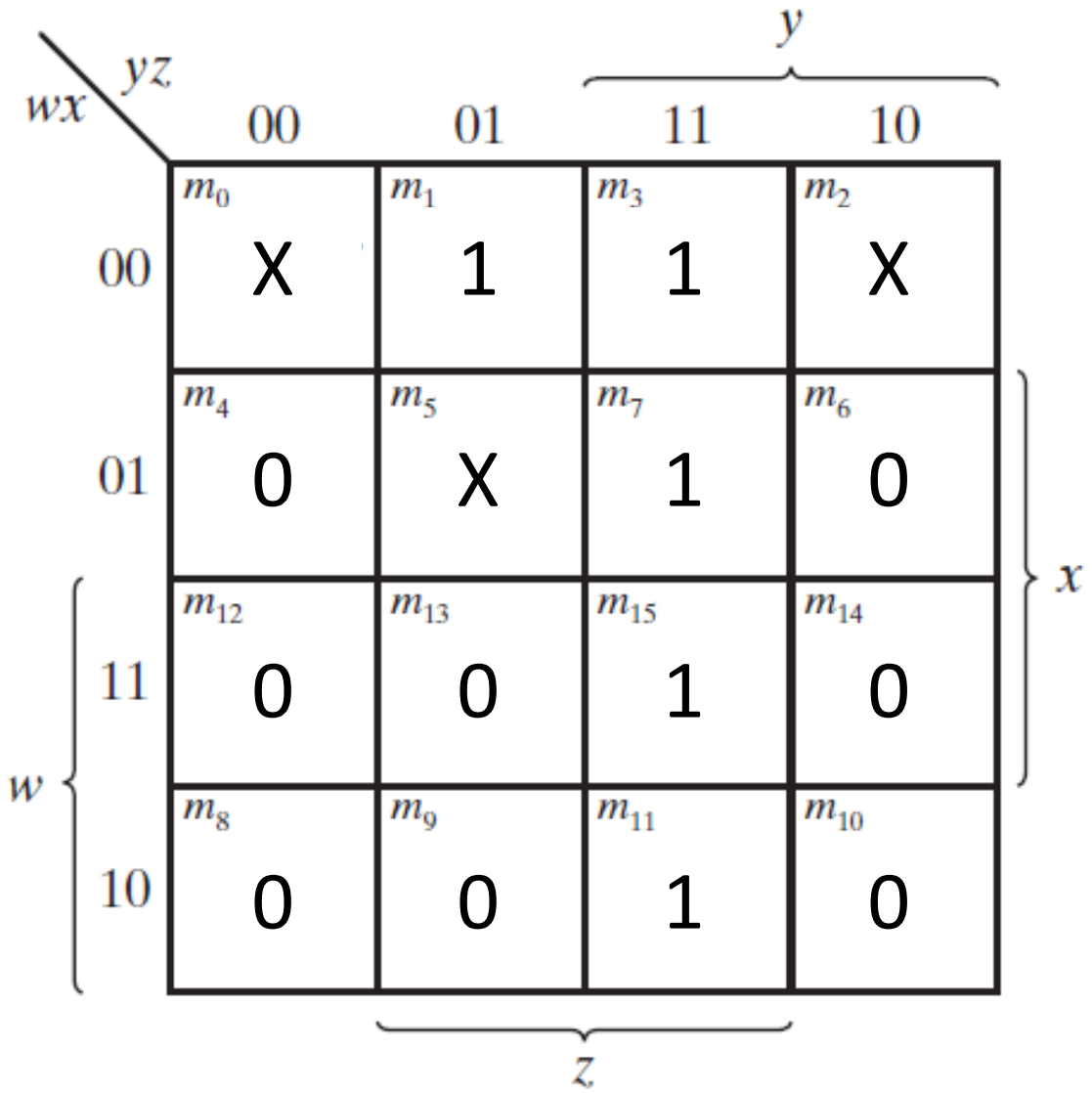
- Now, consider the same function

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

- which has the don't-care conditions

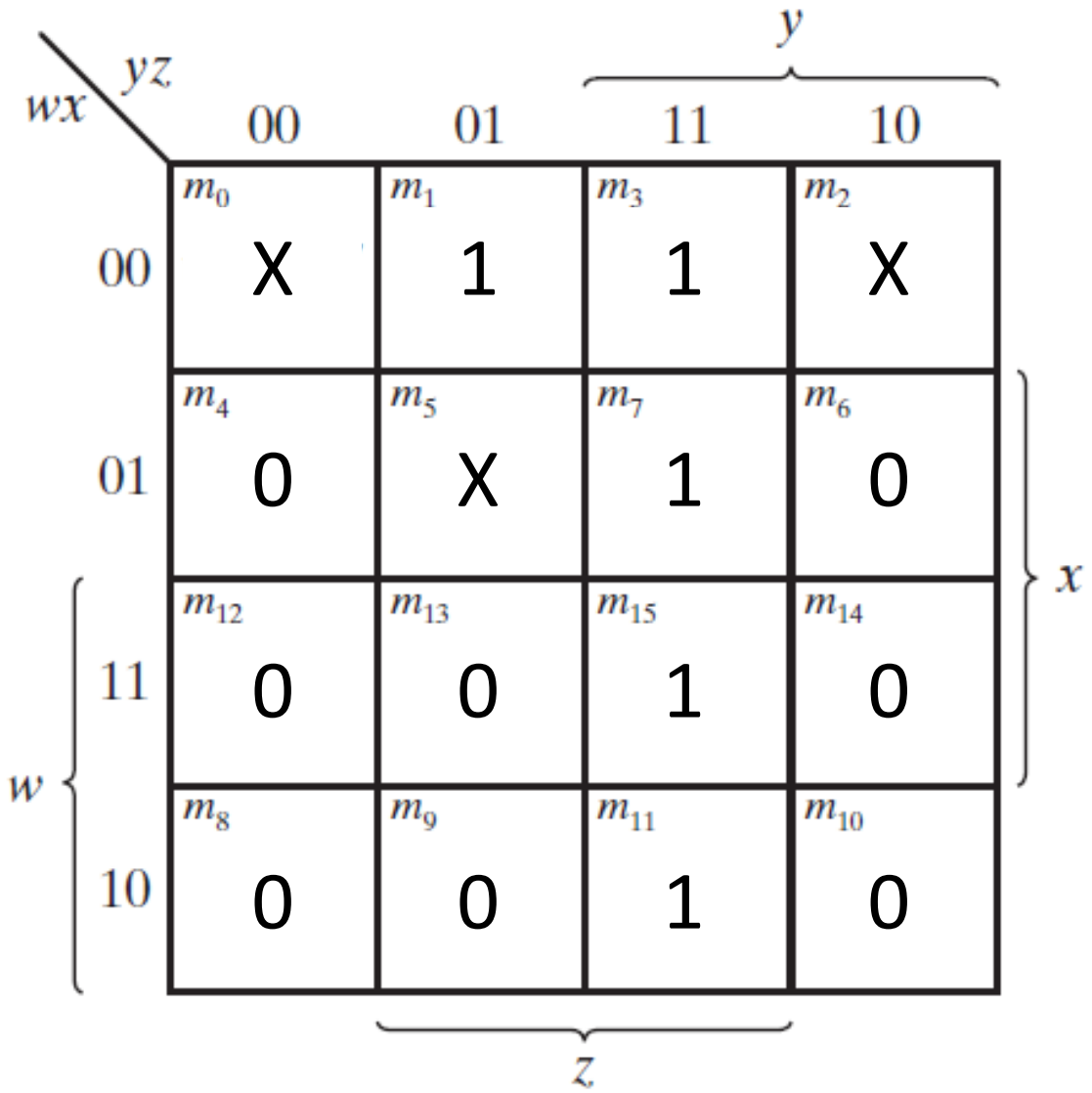
$$d\ (w, x, y, z)\ = \sum (0, 2, 5)$$

- 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 $F = yz + w'x'$



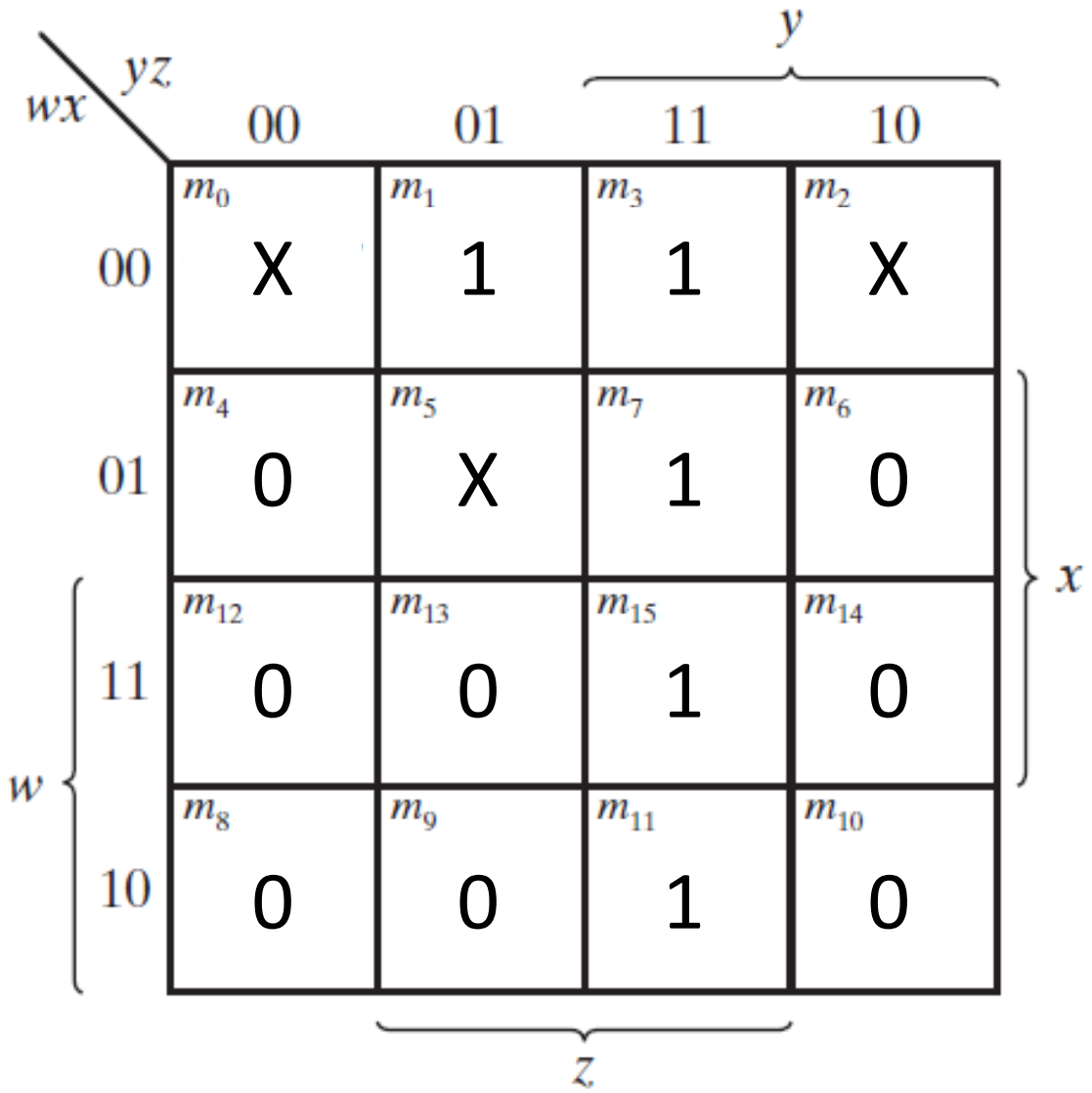
The Other states

- 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



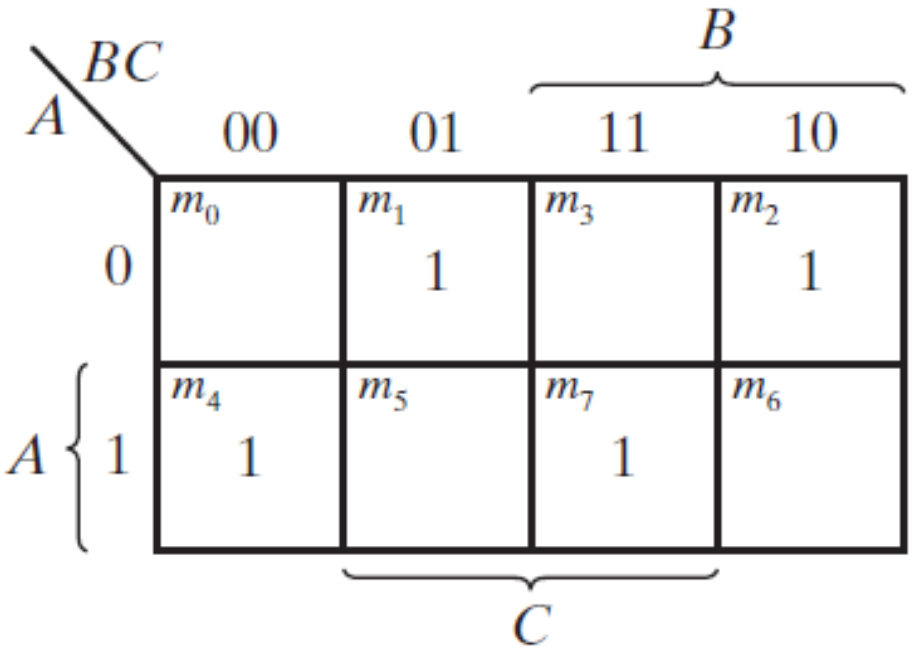
The Other states

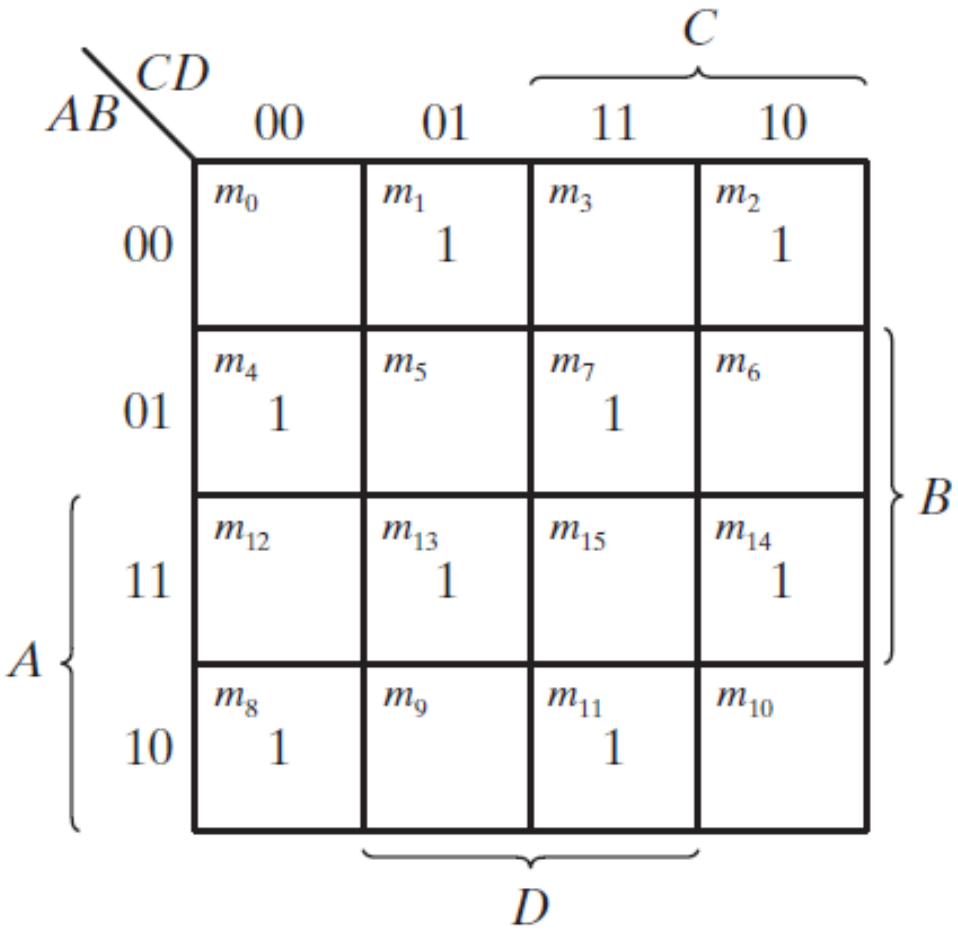
- The key question is: Are these the simplest possible representations for the given function? $F = yz + w'z$ and $F = yz + w'x'$
- What if we look at the product of sum simplification?
- We can get a cluster of 8 squares: z'
- And a cluster of four squares: wy'
- Thus, the function can be represented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



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