EC2.101 – Digital Systems and Microcontrollers

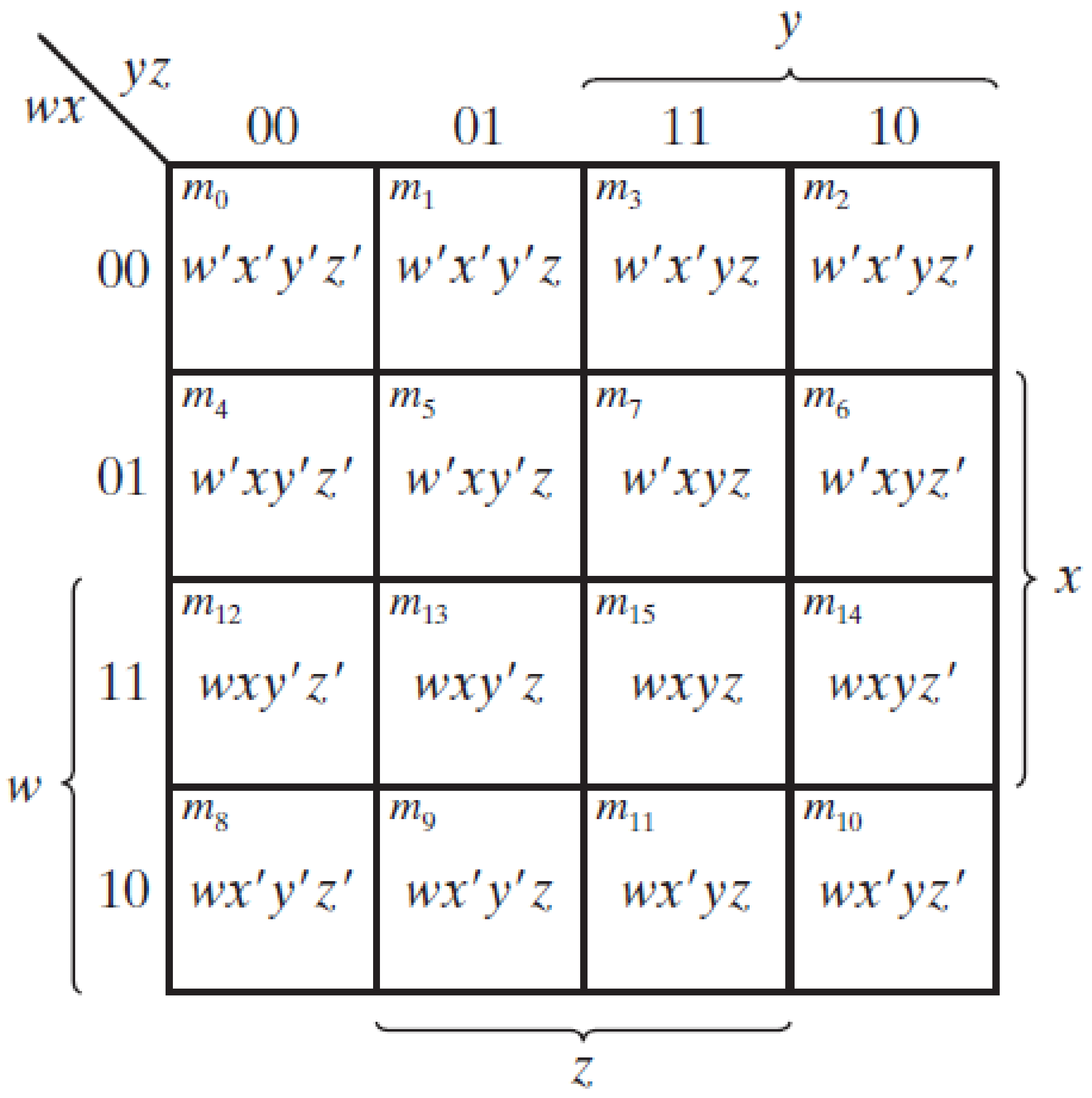
Lecture 11 – K-maps & Logic implementation

Dr. Aftab M. Hussain,

Associate Professor, PATRIoT Lab, CVEST

4 variable K-map

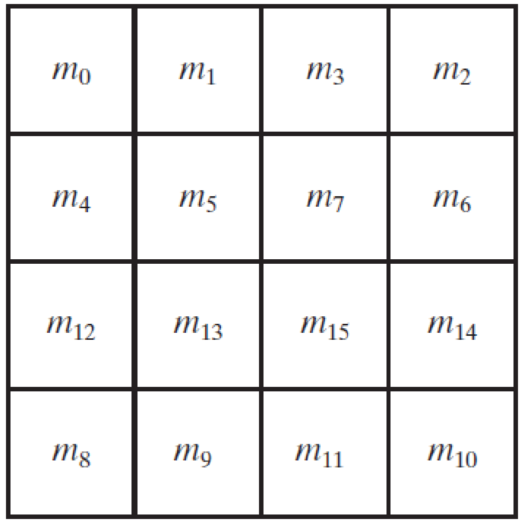
m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}



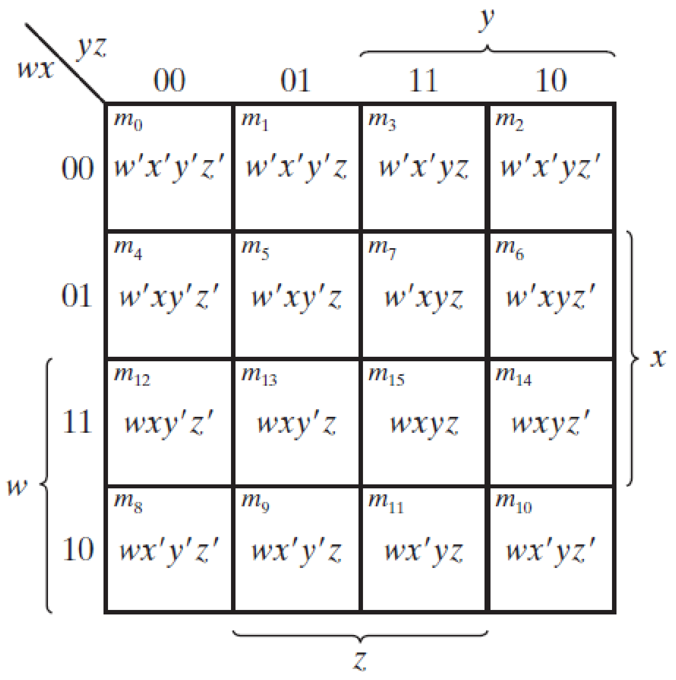
$wx \backslash yz$	00	01	11	10
00	m_0 $w'x'y'z'$	m_1 $w'x'y'z$	m_3 $w'x'yz$	m_2 $w'x'yz'$
01	m_4 $w'xy'z'$	m_5 $w'xy'z$	m_7 $w'xyz$	m_6 $w'xyz'$
11	m_{12} $wxy'z'$	m_{13} $wxy'z$	m_{15} $wxyz$	m_{14} $wxyz'$
10	m_8 $wx'y'z'$	m_9 $wx'y'z$	m_{11} $wx'yz$	m_{10} $wx'yz'$

4 variable K-map

1. We look for a cluster of adjacent squares with the function value being 1 (or true):
   1. A cluster of 16 squares (meaning the entire function is 1)
   2. A cluster of 8 squares (meaning one literal)
   3. A cluster of 4 squares (meaning two literals ANDed)
   4. A cluster of 2 squares (meaning three literals ANDed)
   5. A single square with all the four variables ANDed (a minterm)
2. We OR all the expressions related to the clusters



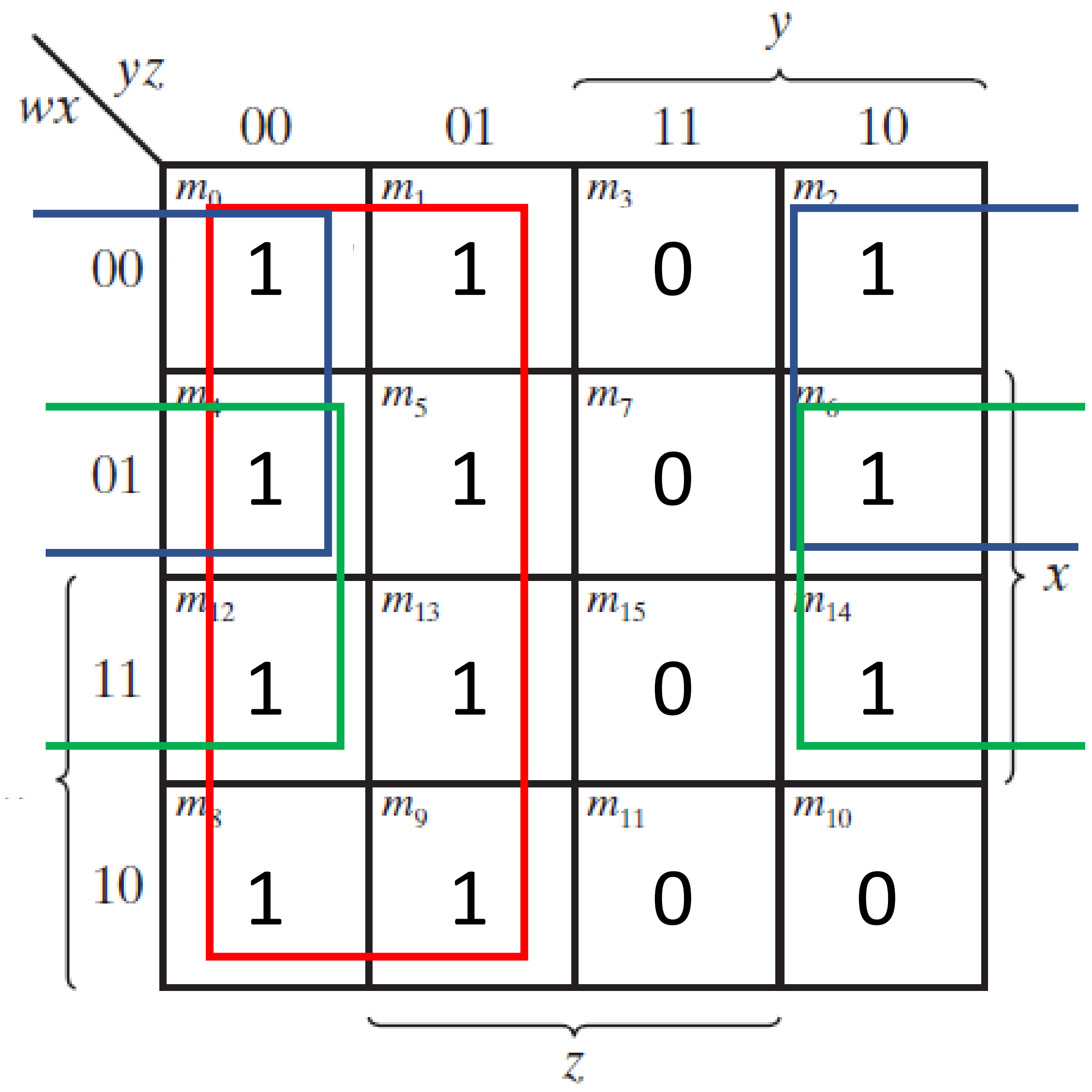
m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}



$wx \backslash yz$	00	01	11	10
00	m_0 $w'x'y'z'$	m_1 $w'x'y'z$	m_3 $w'x'yz$	m_2 $w'x'yz'$
01	m_4 $w'xy'z'$	m_5 $w'xy'z$	m_7 $w'xyz$	m_6 $w'xyz'$
11	m_{12} $wxy'z'$	m_{13} $wxy'z$	m_{15} $wxyz$	m_{14} $wxyz'$
10	m_8 $wx'y'z'$	m_9 $wx'y'z$	m_{11} $wx'yz$	m_{10} $wx'yz'$

4 variable K-map

- Simplify the function $F\ (w, x, y, z)\ = \sum(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$
- Cluster of 16? No
- Cluster of 8?
- It represents y’
- Cluster of 4?
- This represents w’z’
- This represents xz’
- Thus, the function is $F\ (w, x, y, z) = y' + w'z' + xz'$



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

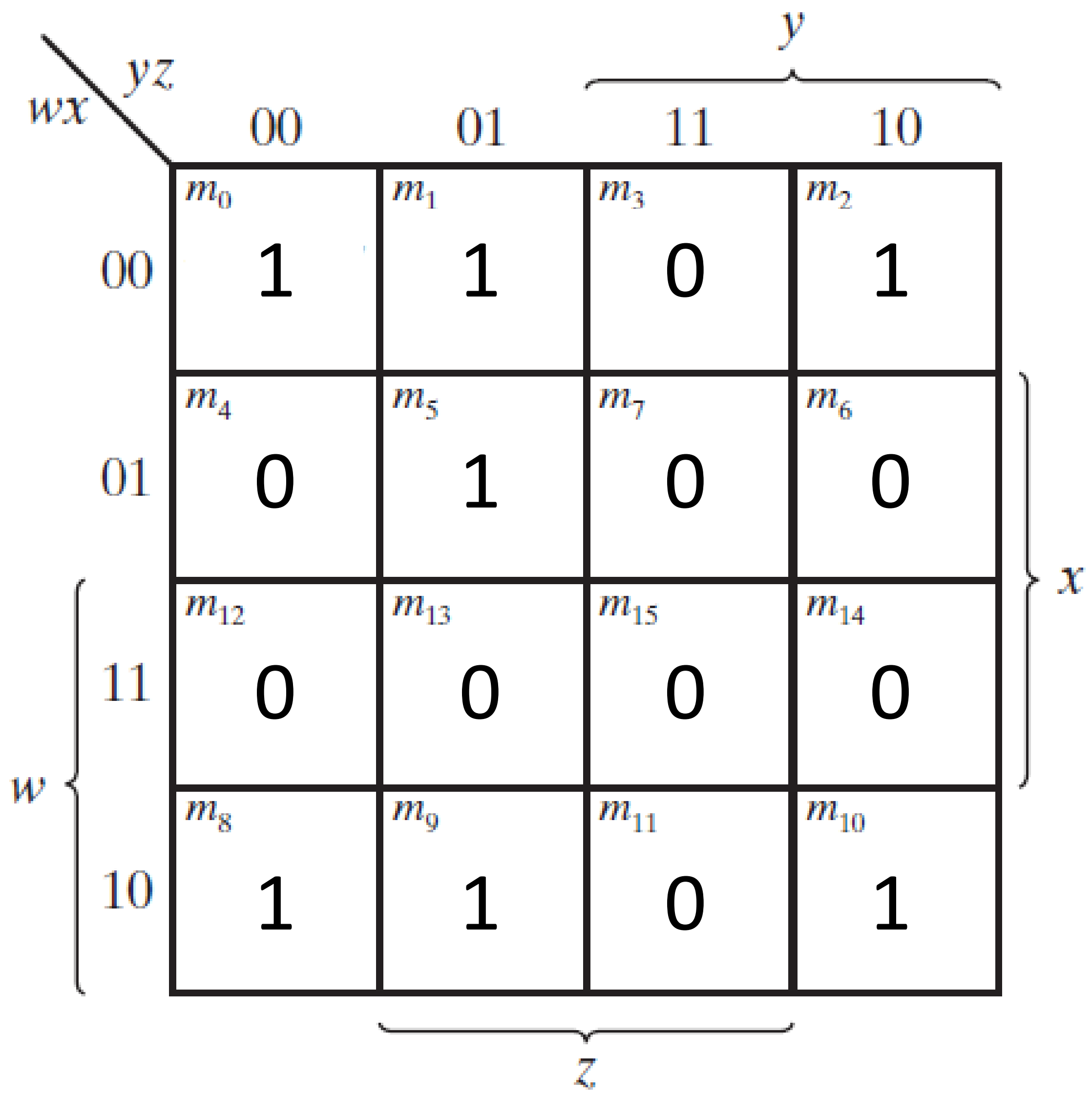
Product of Sums

- The minimized Boolean functions derived from the K-map in the previous lecture were expressed in sum-of-products form
- With a minor modification, the product-of-sums form can be obtained
- The 1's placed in the squares of the map represent the minterms of the function
- From this observation, we see that the complement of a function is represented in the map by the squares not marked by 1's
- If we mark the empty squares by 0's and combine them into valid adjacent squares, we obtain a simplified sum-of-products expression of the complement of the function (i.e., of *F'*)
- The complement of *F'* gives us back the function *F* in product-of-sums form (a consequence of DeMorgan's theorem)

Product of Sums

- Simplify the following Boolean function into (a) sum-of-products form and (b) product-of-sums form: $F\ (w, x, y, z)\ = \sum(0, 1, 2, 5, 8, 9, 10)$
- Two clusters of four squares: $x'z'$ and $x'y'$
- One cluster of two squares: $w'y'z$
- Thus, the function is:

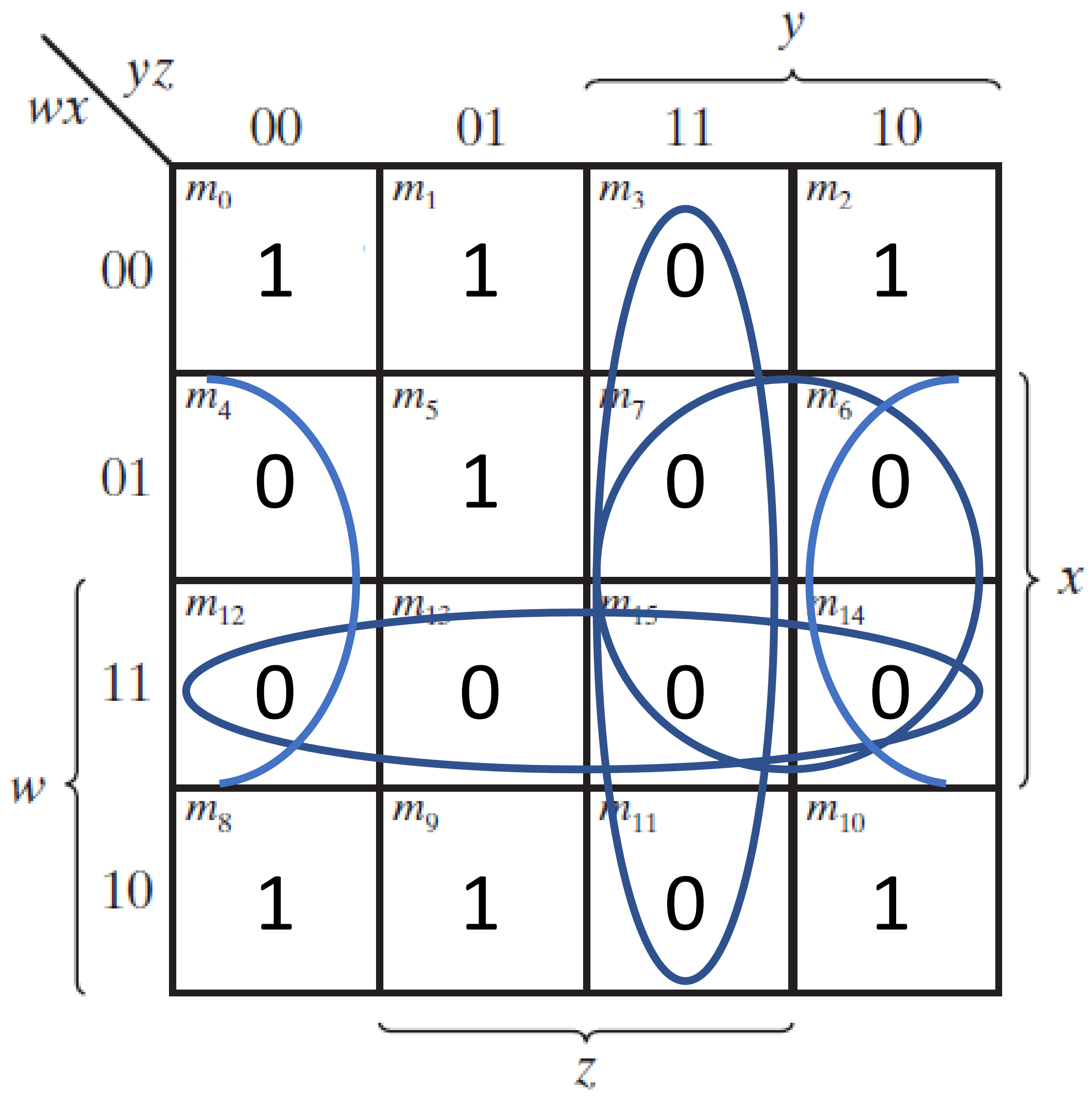
$$F = x'z' + x'y' + w'y'z$$



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

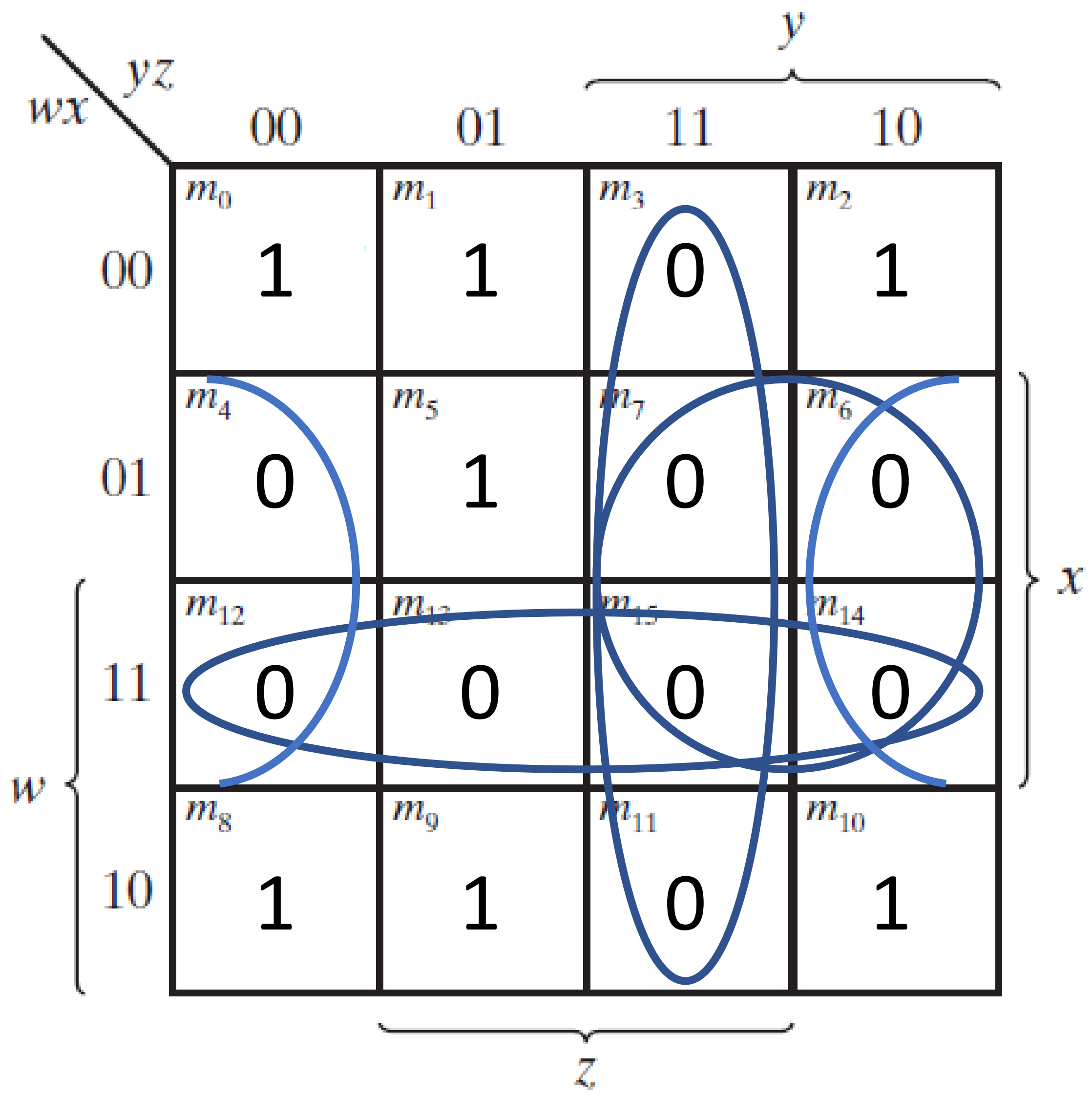
Product of Sums

- The original function is:
$$F = x'z' + x'y' + w'y'z$$
- Now, let us see the complement of the function: F'
- This can be obtained from clustering all the 0s together
- We have four clusters of four (prime implicants)
- Off these, the essential prime implicants are: yz, wx, xz'
- Thus, $F' = wx + yz + xz'$



Product of Sums

- The original function is:
  $$F = x'z' + x'y' + w'y'z$$
- Thus, the complement function is $F' = wx + yz + xz'$
- Now, we can obtain F back from F' using the DeMorgan's theorems
- Thus,
  $$F = (w' + x')(y' + z')(x' + z)$$
- Hence, the actual simplest implementation of a function can be product of sum



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

Logic implementations

Timing diagrams

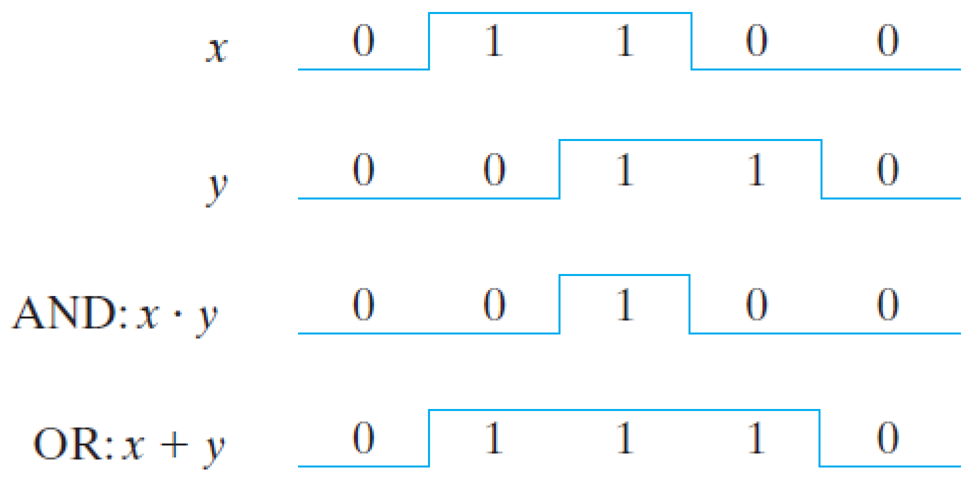
- With logic functions, it is always nice to visualize the various conditions giving a particular output
- One way of visualizing is the truth table, another way can be the timing diagram of a particular function
- Timing diagrams are useful to indicate the sequence of events in large combinational circuits

AND

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1



Timing diagrams

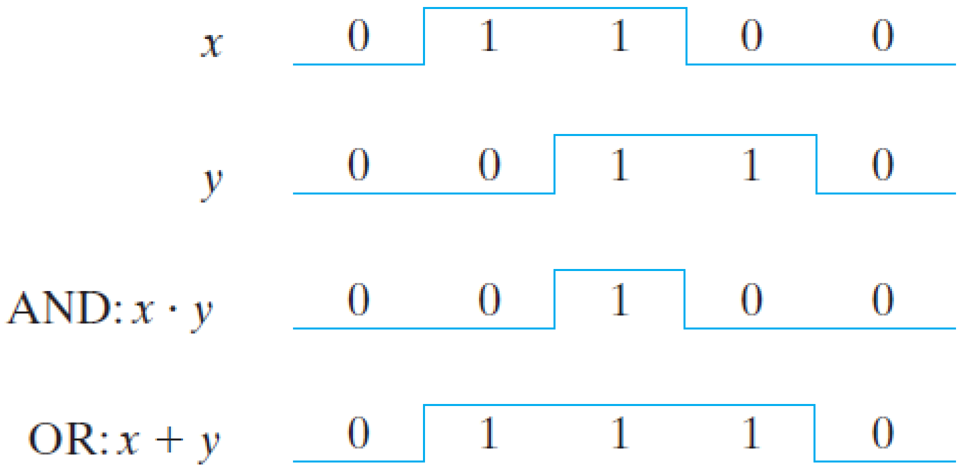
- The timing diagrams illustrate the idealized response of each gate to the four input signal combinations
- The horizontal axis of the timing diagram represents the time, and the vertical axis shows the signal as it changes between the two possible voltage levels
- In reality, the transitions between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

AND

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

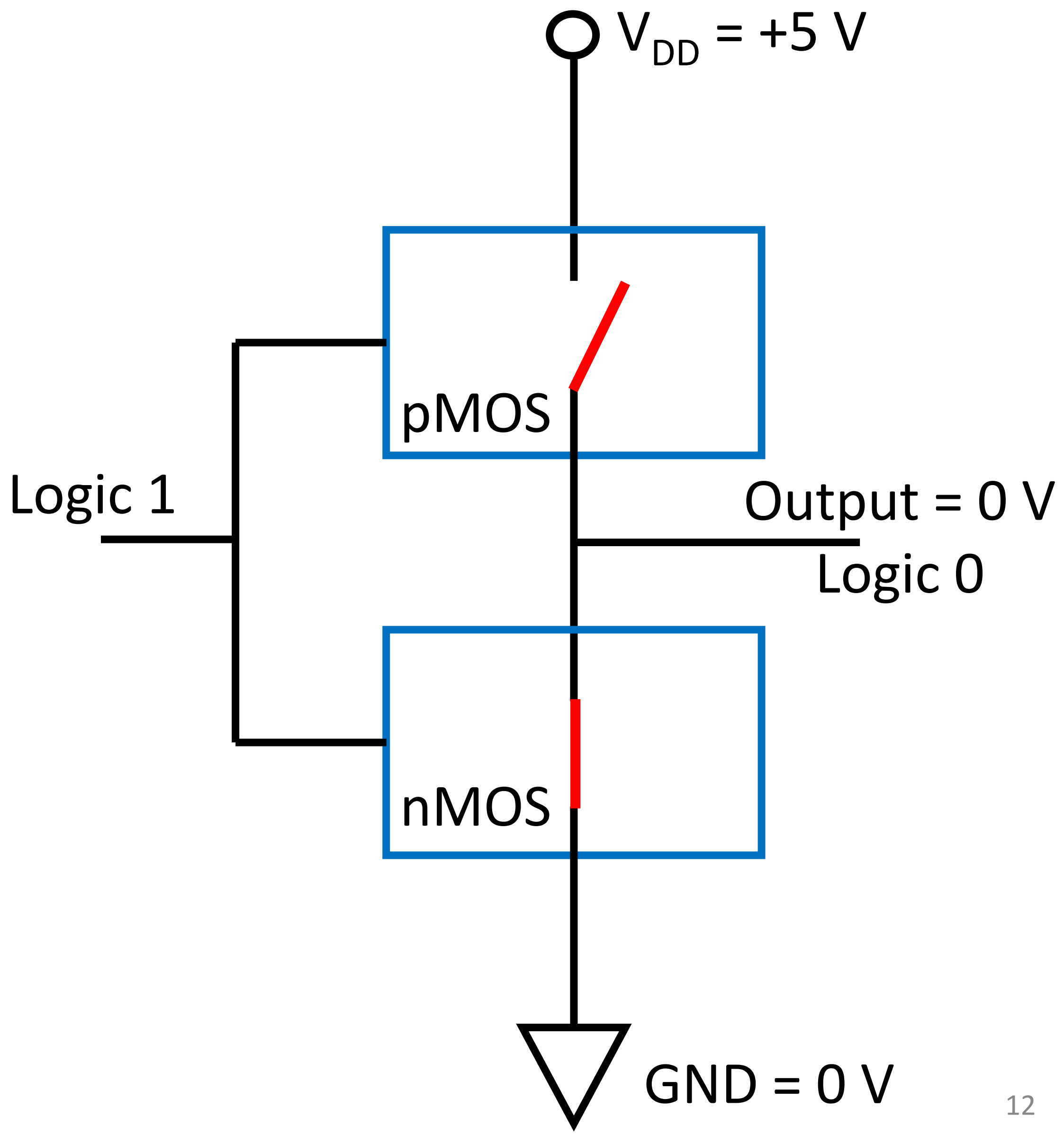
OR

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1



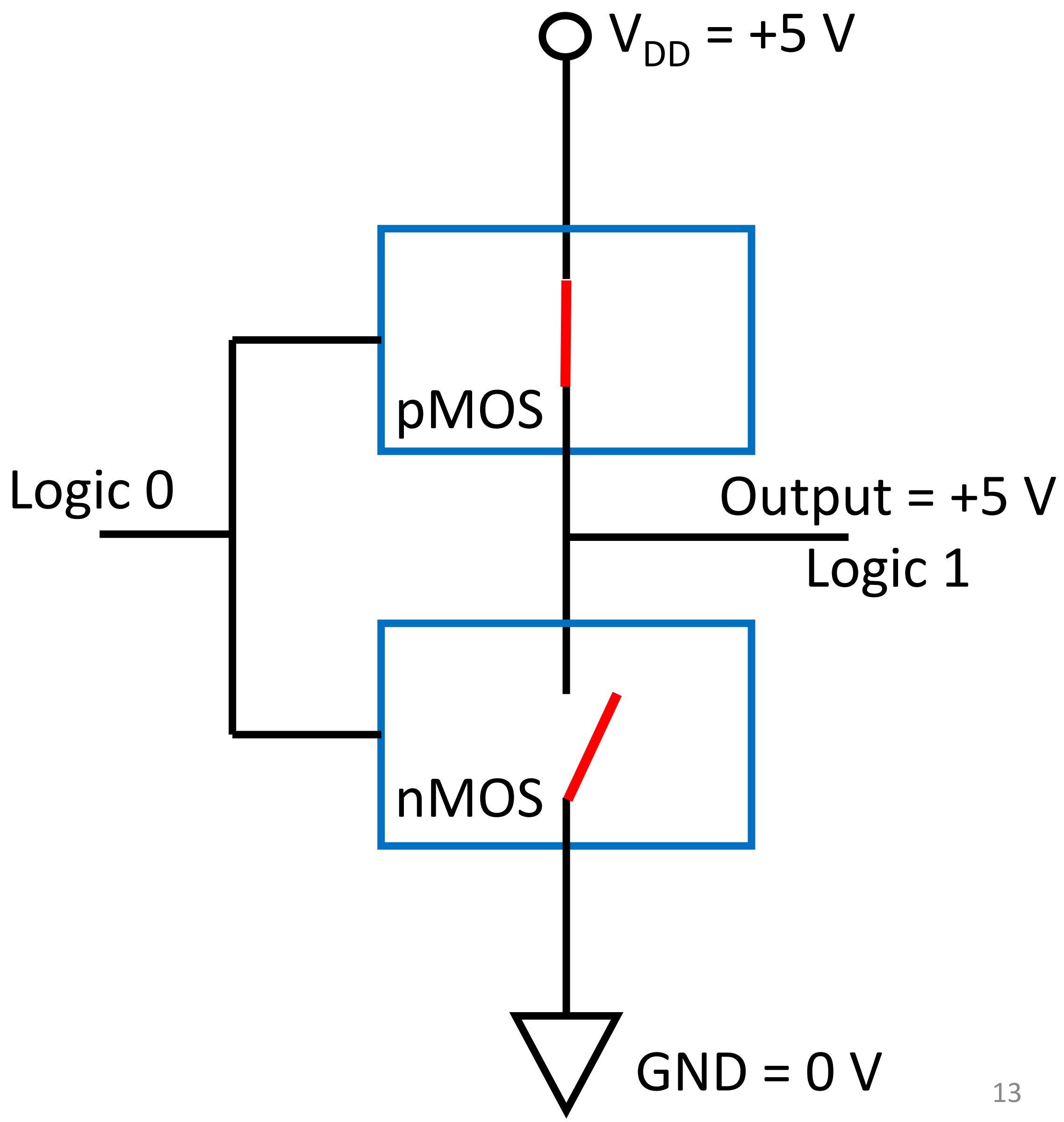
Implementing logic gates

- 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

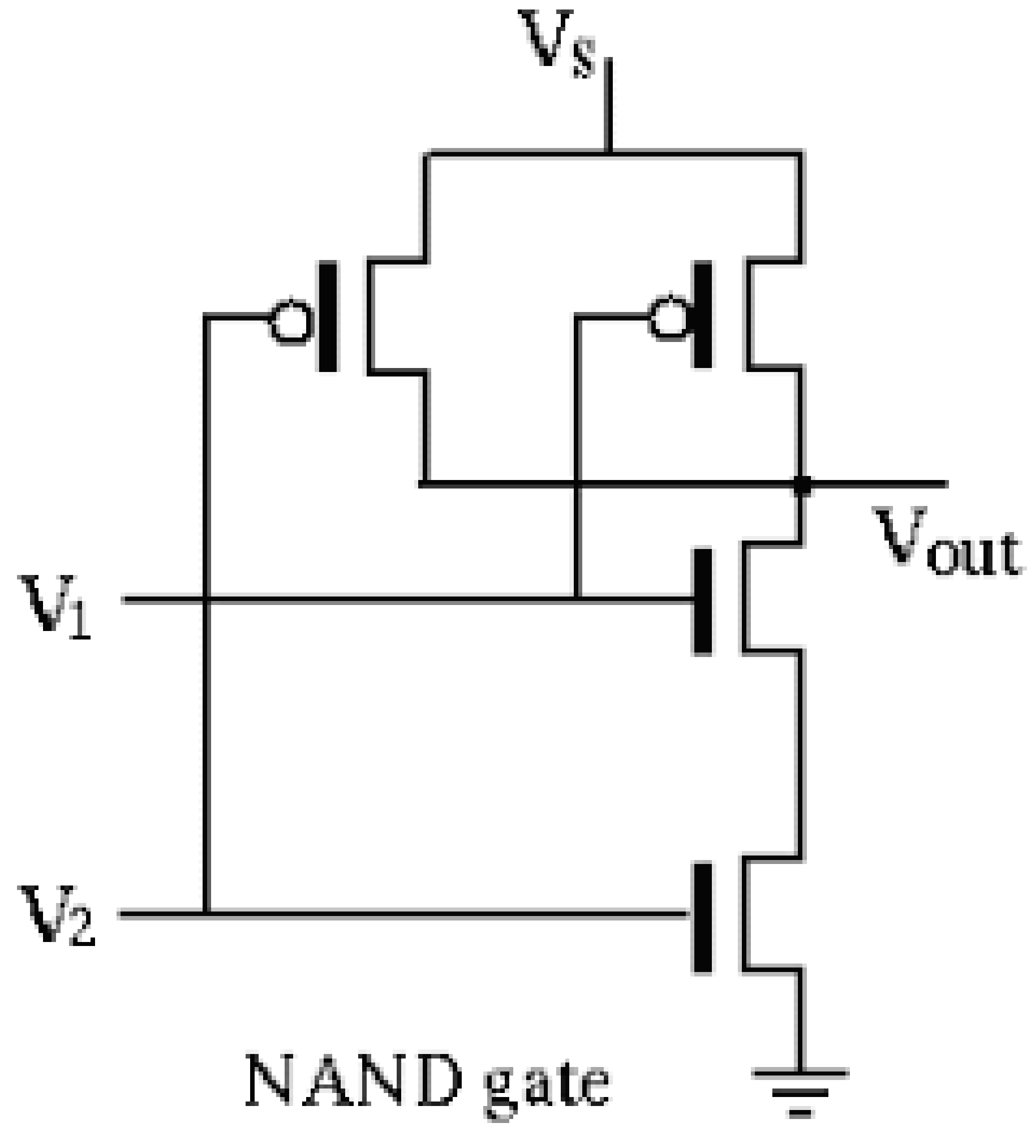


Implementing logic gates

- 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

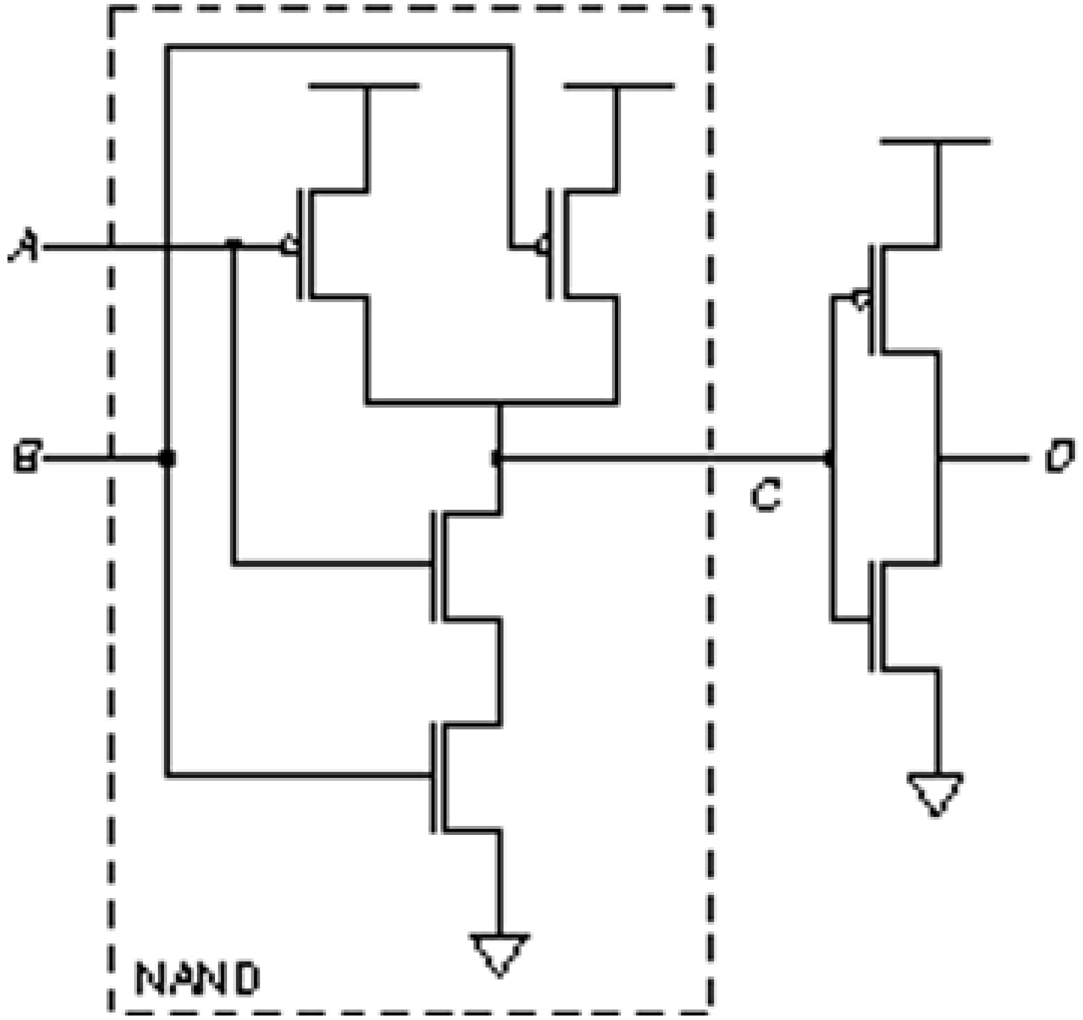


Implementing logic gates

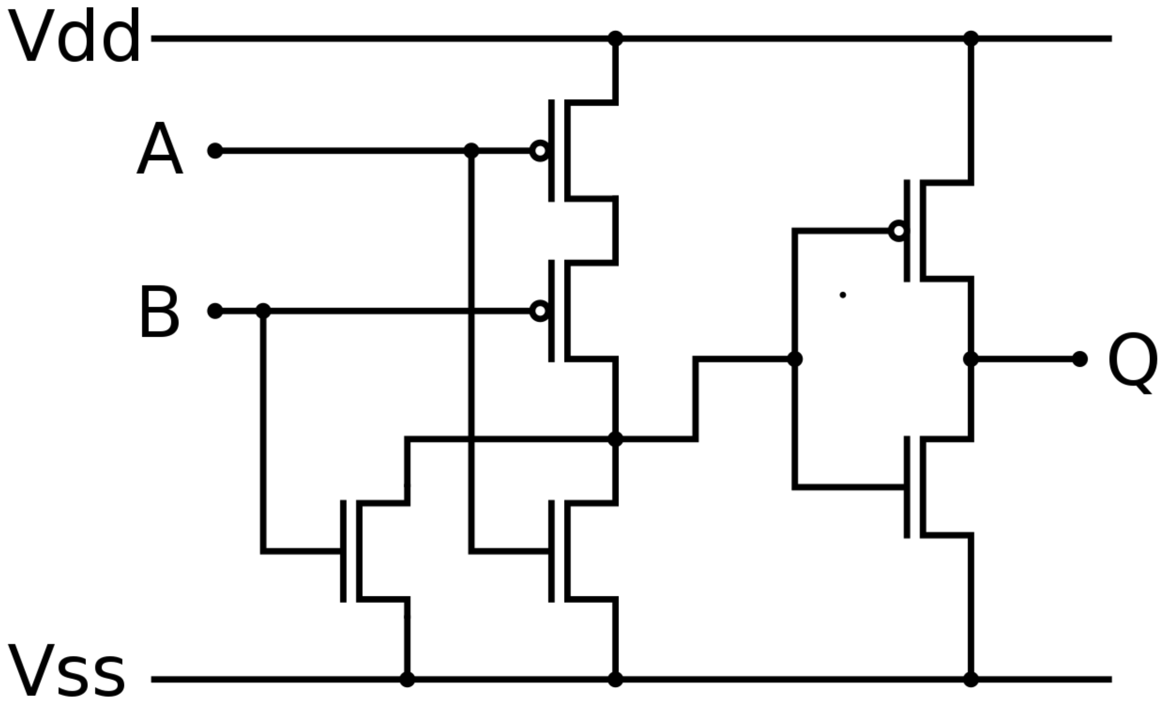


Implementing logic gates

AND gate

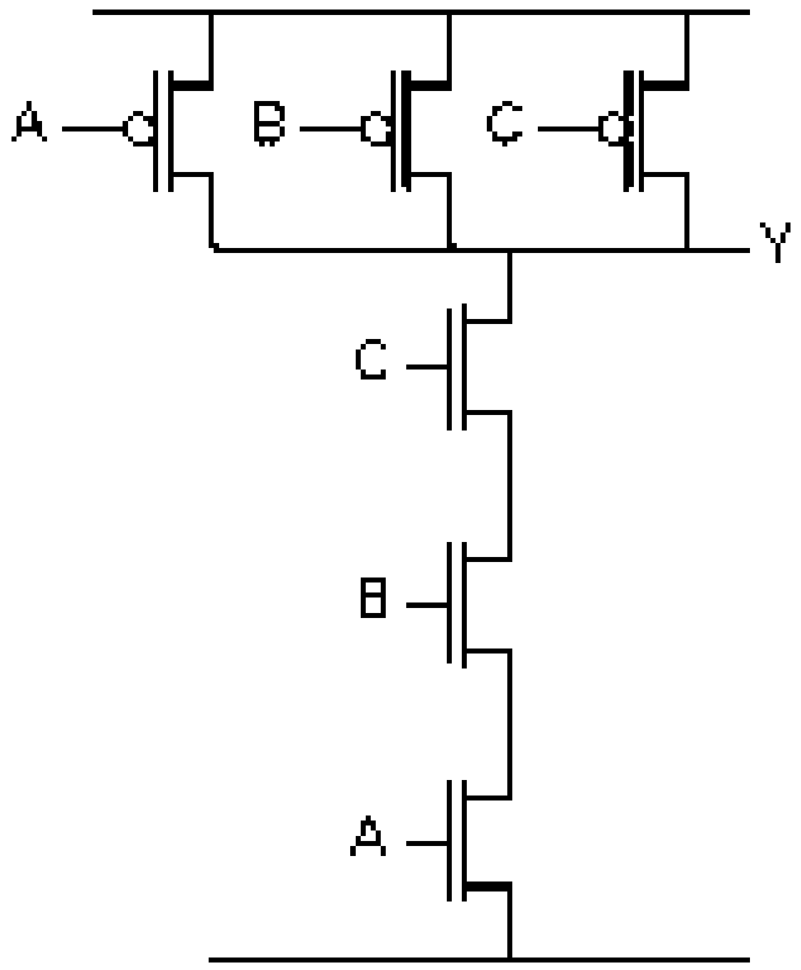


OR gate

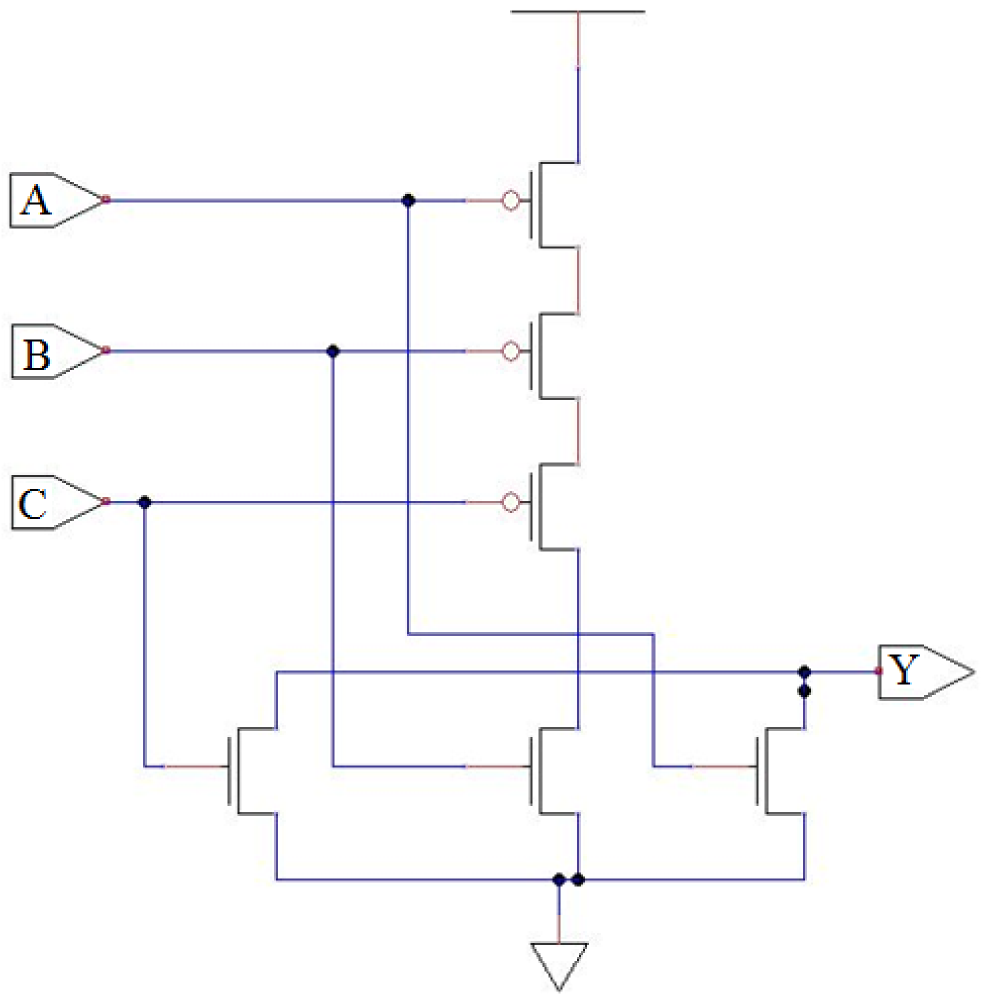


Implementing logic gates

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