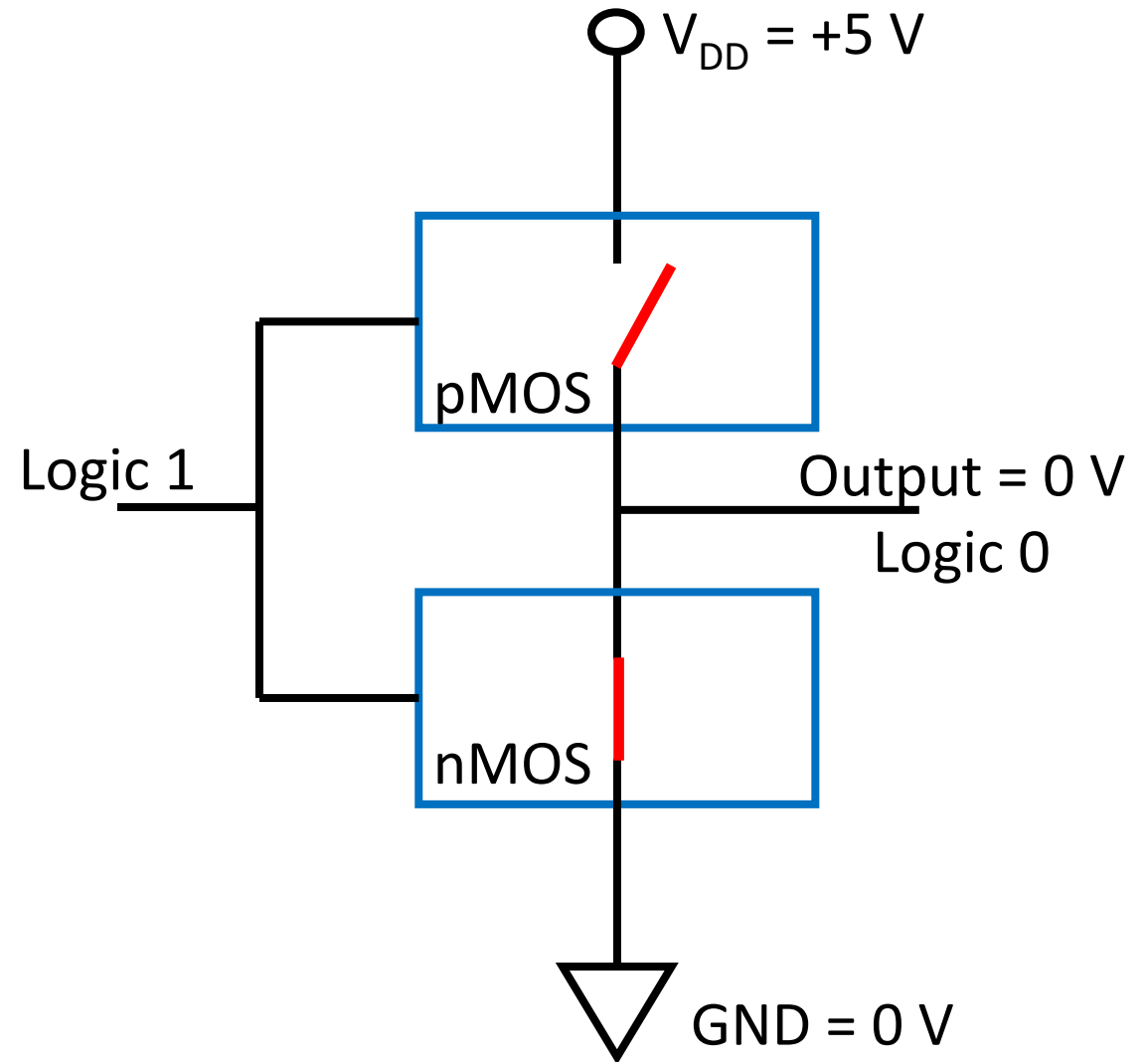


Lecture 11 – Logic implementation

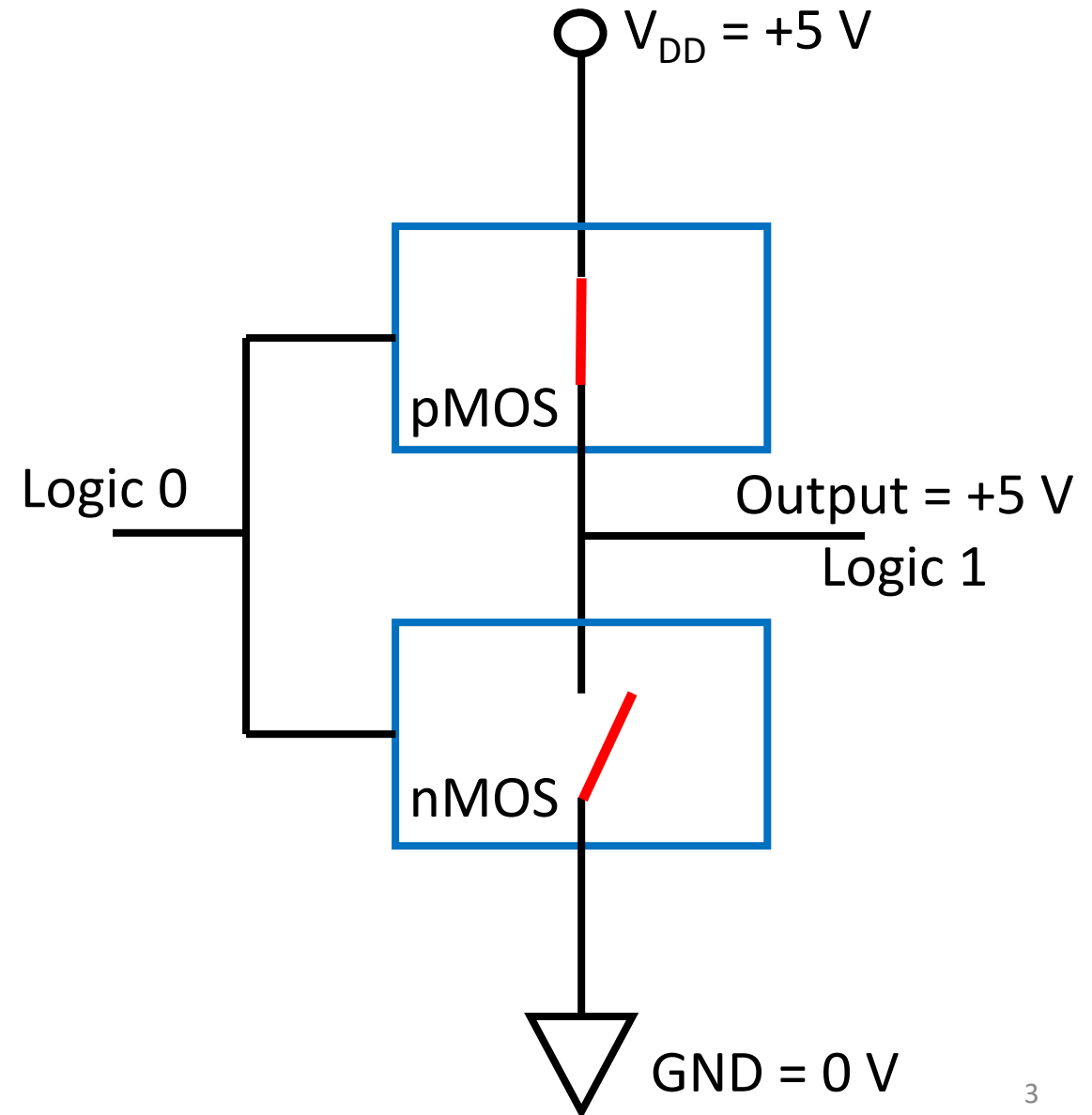
Implementation of logic gates

- Typically, logic gates are implemented using transistors in CMOS architecture, where they act as switches, connecting V_{DD} or GND to the output
 - pMOS – p channel *metal-oxide semiconductor*
 - nMOS – n channel *metal-oxide semiconductor*
 - CMOS – complementary *metal-oxide semiconductor*
- 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



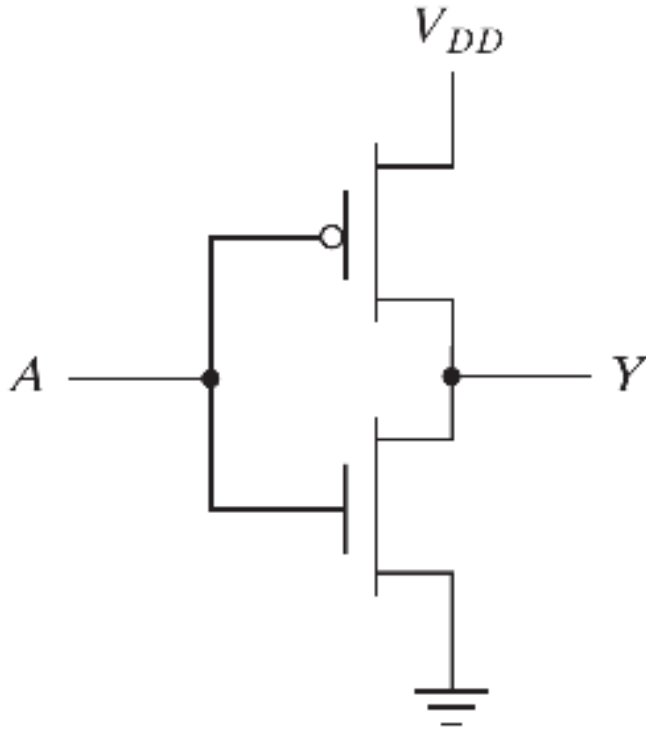
Implementation of logic gates

- Typically, logic gates are implemented using transistors in CMOS architecture, where they act as switches, connecting V_{DD} or GND to the output
 - pMOS – p channel *metal-oxide semiconductor*
 - nMOS – n channel *metal-oxide semiconductor*
 - CMOS – complementary *metal-oxide semiconductor*
- 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

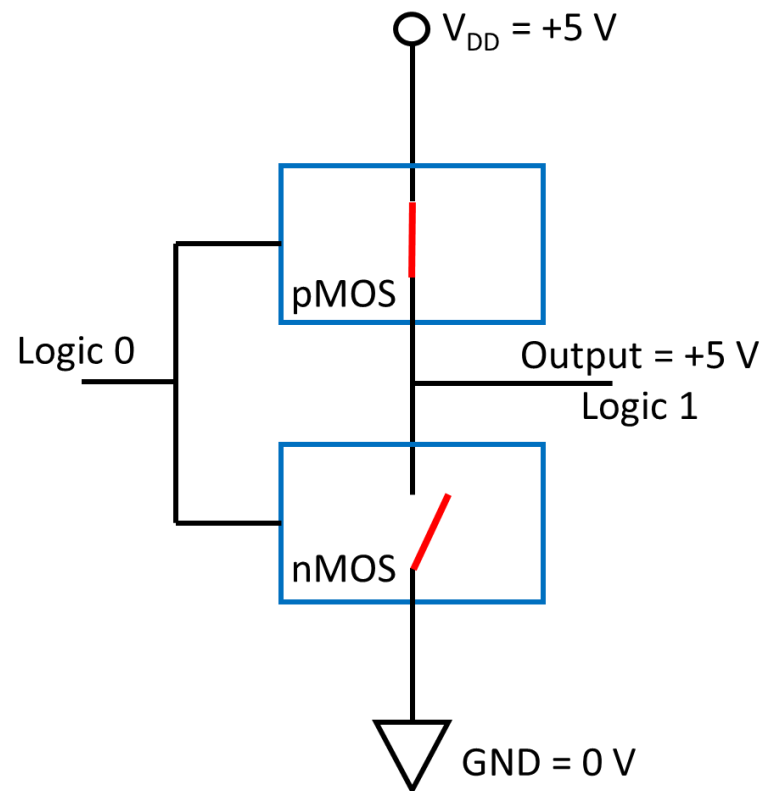


Implementation of logic gates

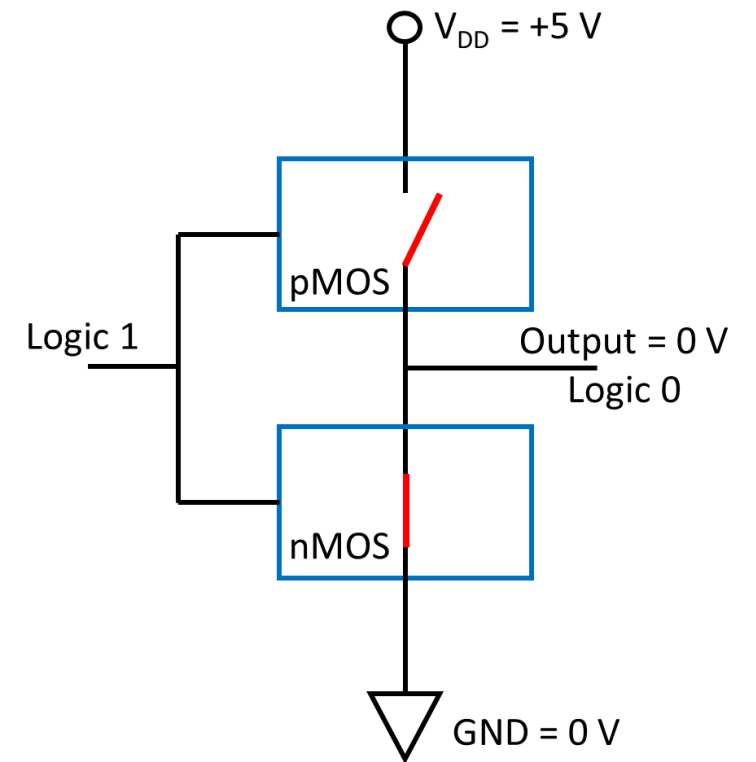
CMOS Inverter



Logical model

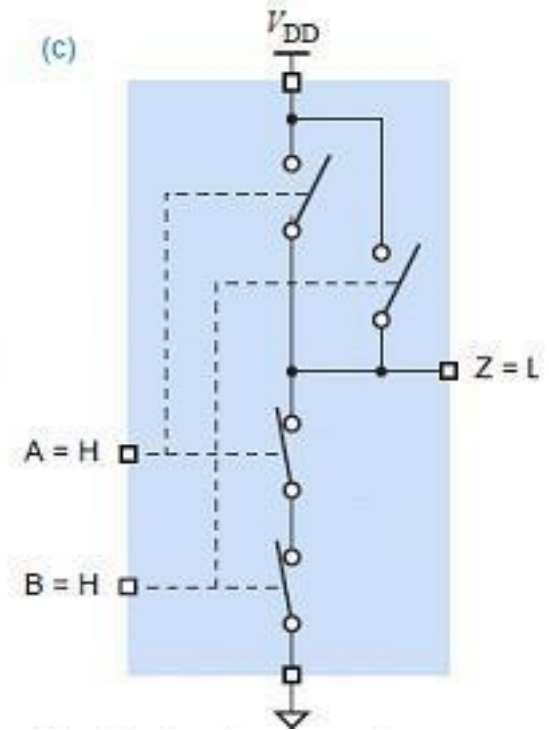
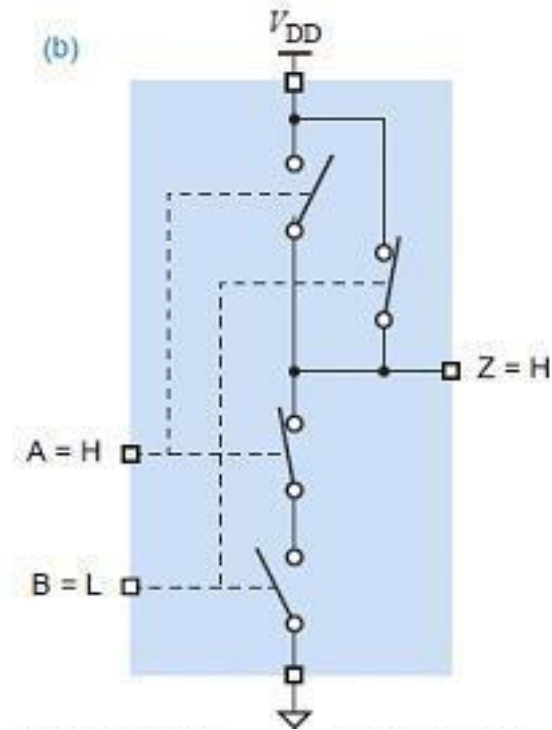
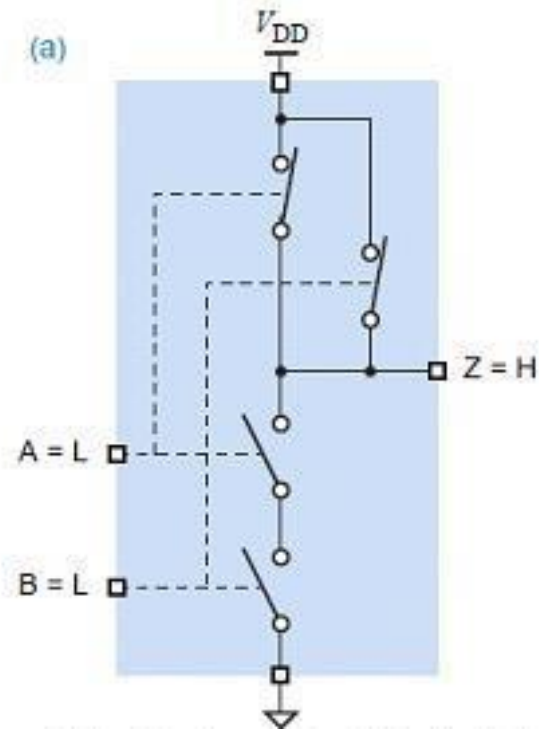
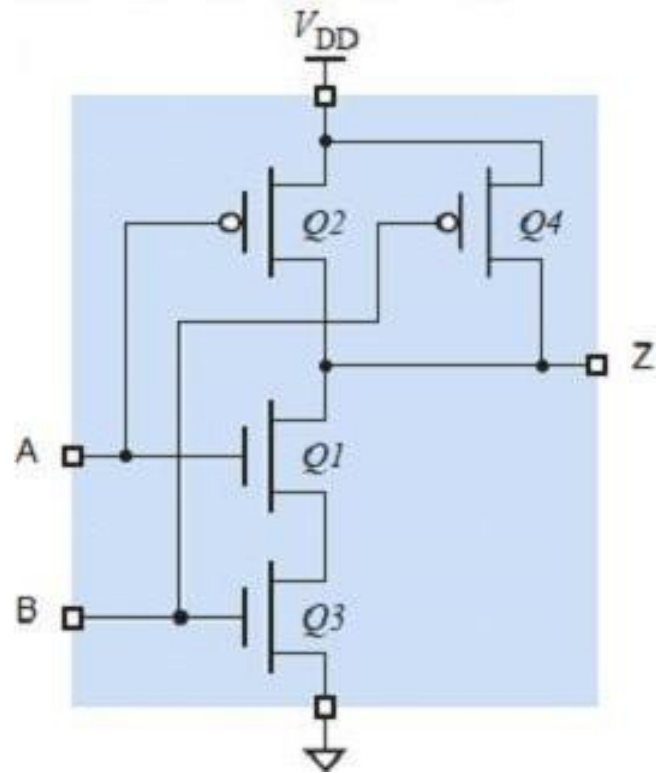


4



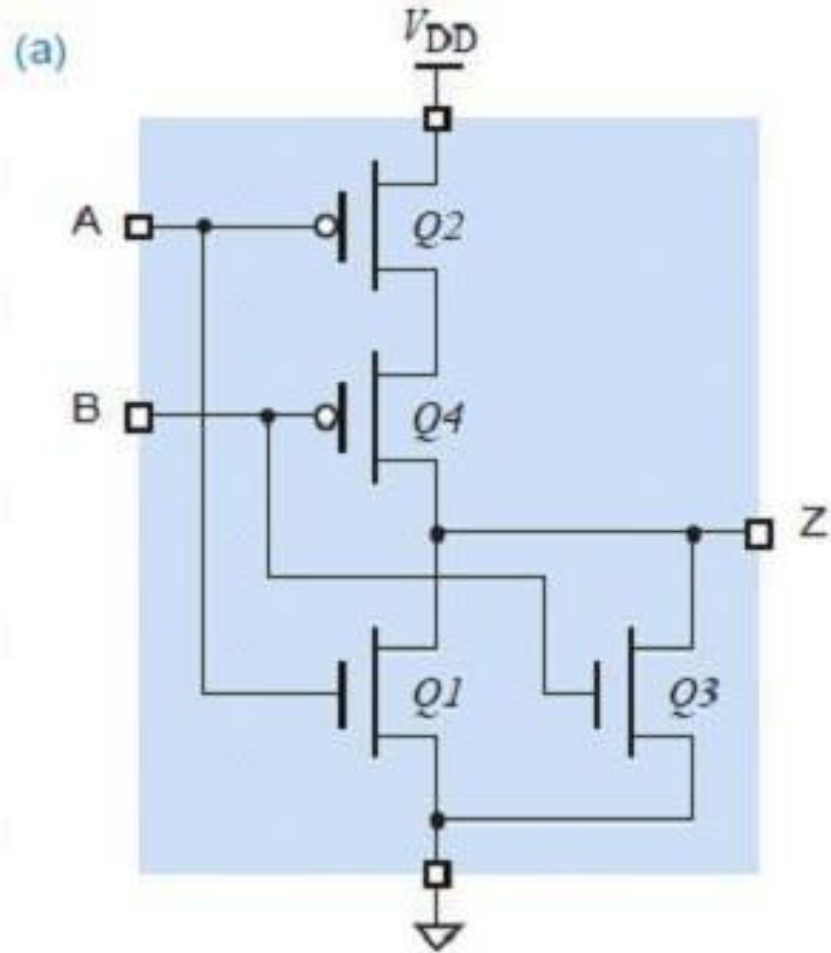
Switch model

Implementation of logic gates



CMOS NAND Gate

Implementation of logic gates



(b)

A	B	$Q1$	$Q2$	$Q3$	$Q4$	Z
L	L	off	on	off	on	H
L	H	off	on	on	off	L
H	L	on	off	off	on	L
H	H	on	off	on	off	L

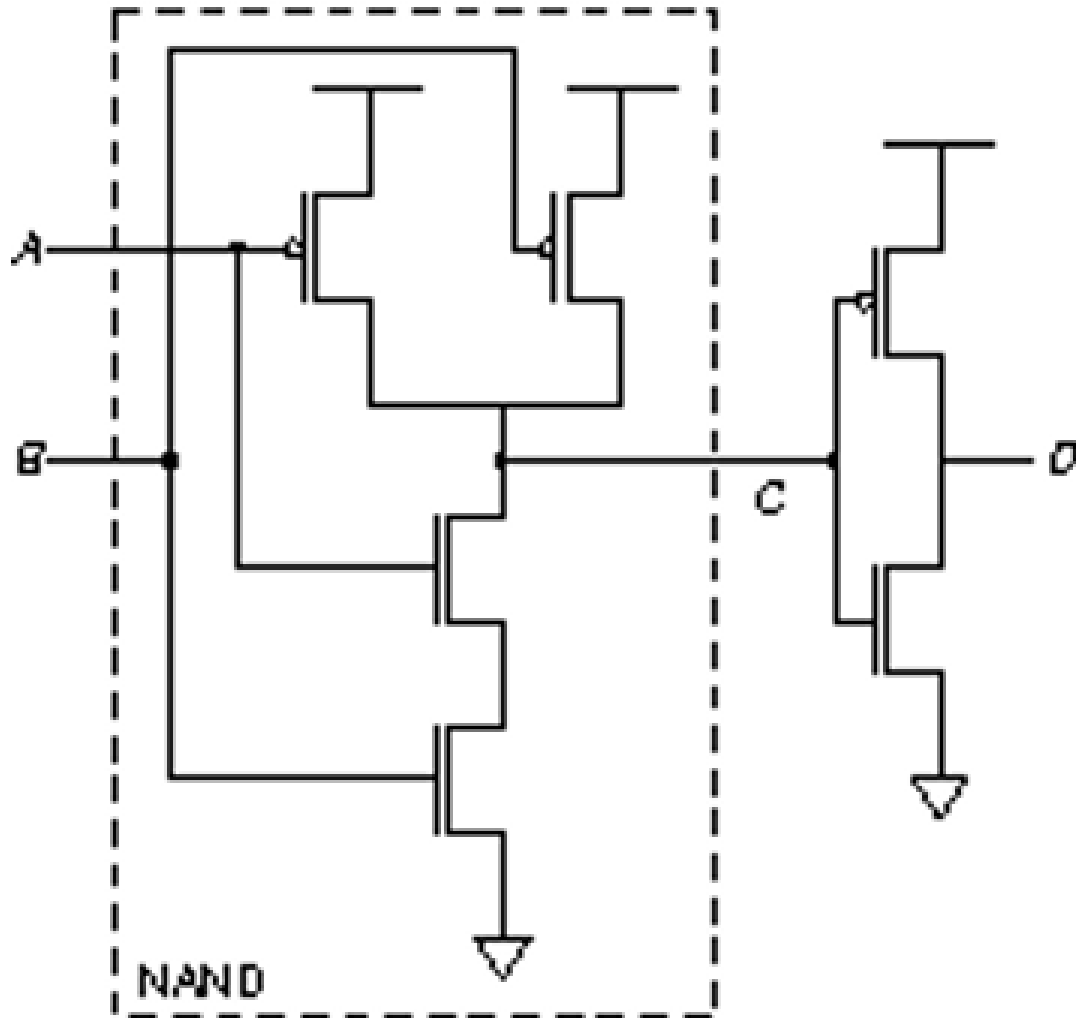
(c)



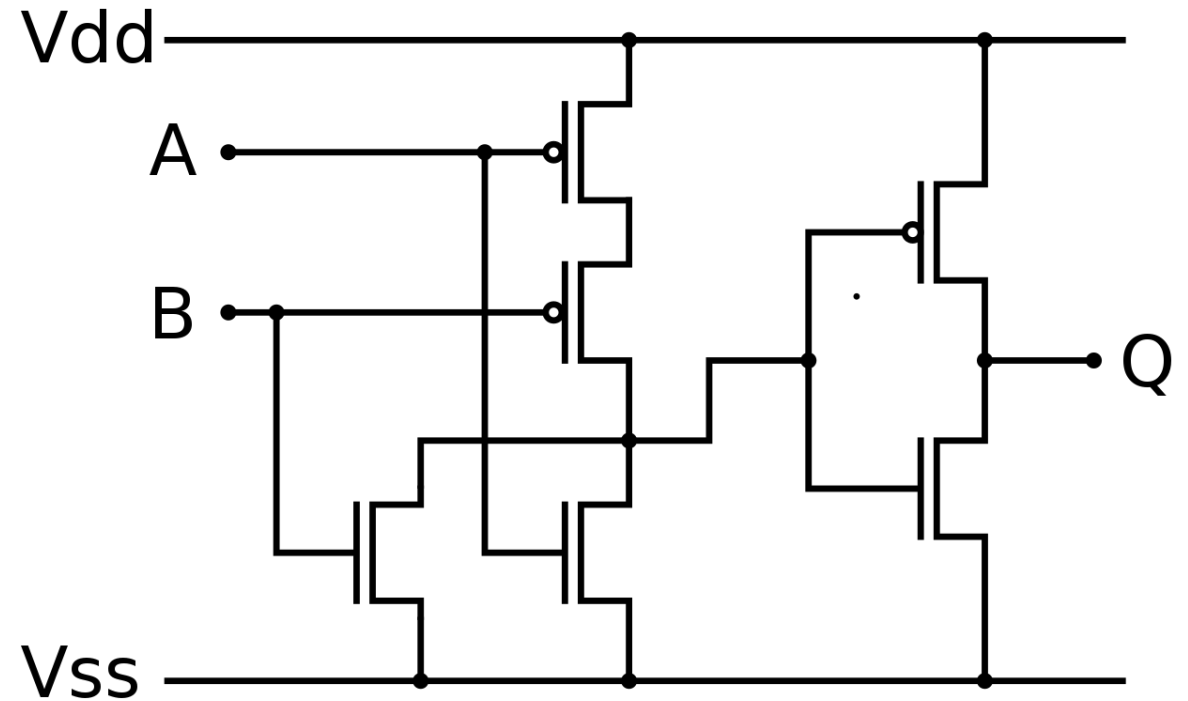
CMOS NOR gate

Implementation of logic gates

AND gate

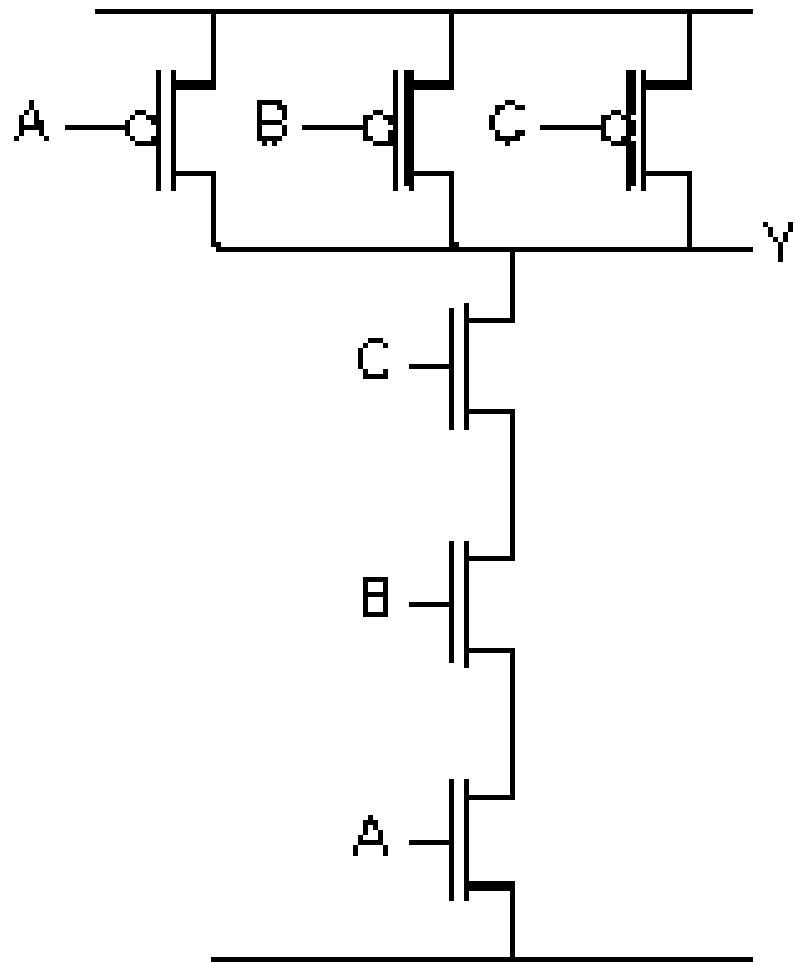


OR gate

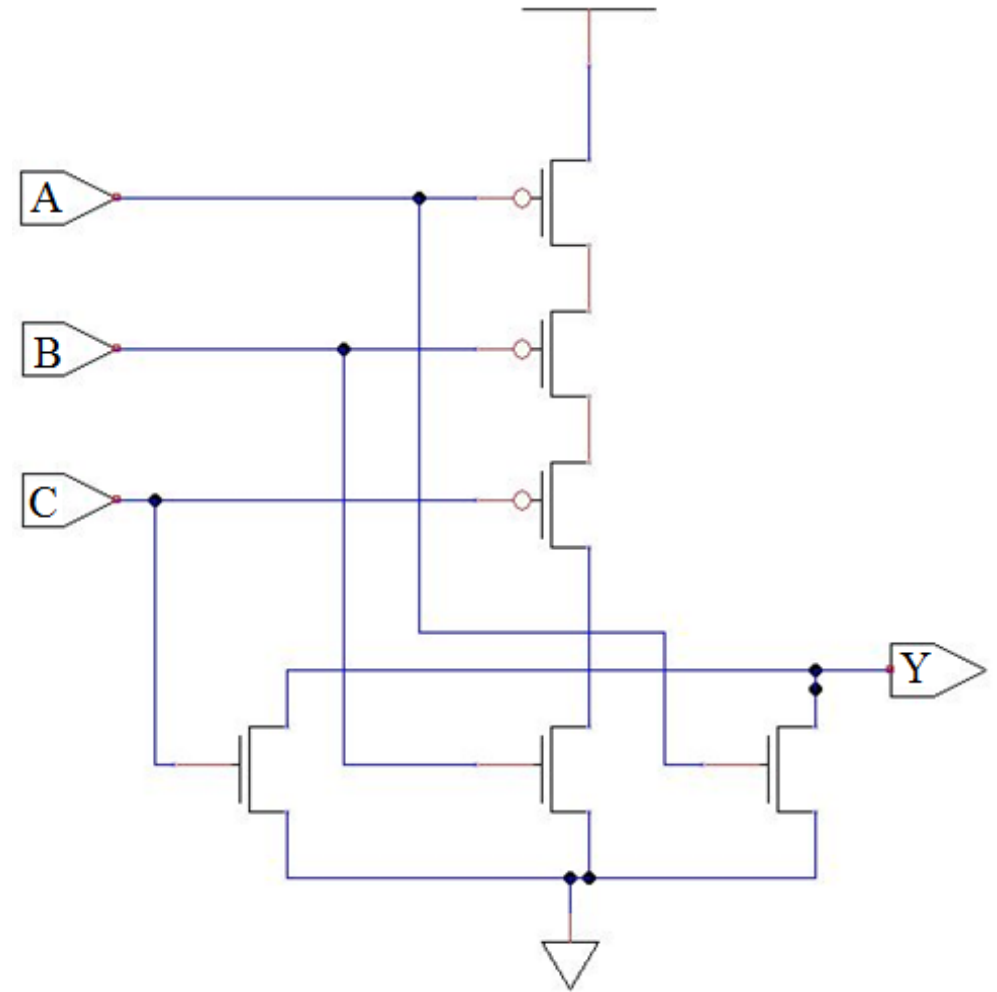


Implementation of logic gates

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 designer must consider such constraints - *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

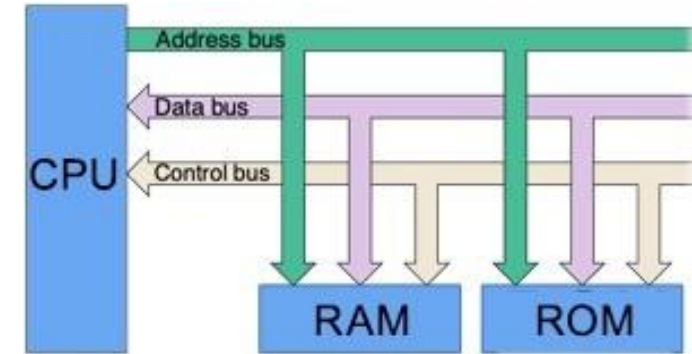
Circuit design

- 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	$2N+2$

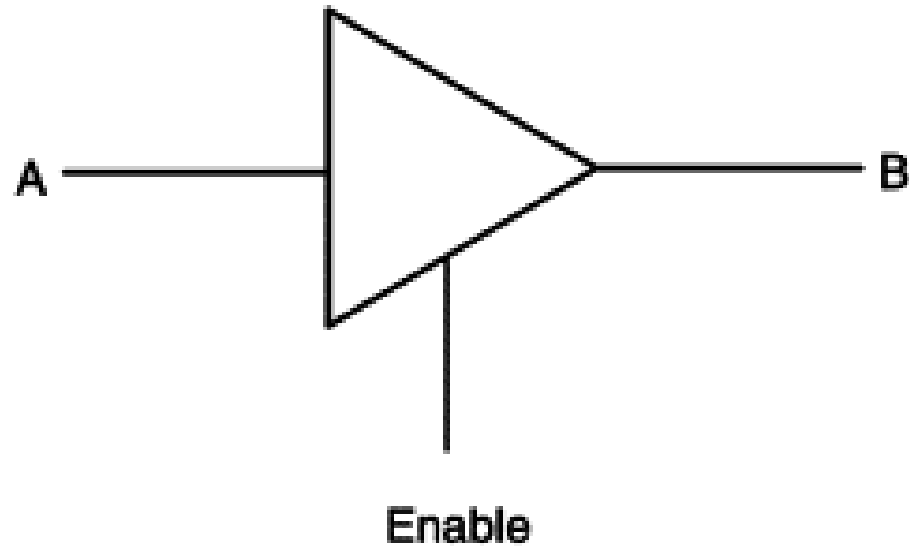
The Other states

- 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

- **Don't care state** : we do not care what the output in a particular case is, i.e., for a particular set of inputs - output can be either 1 or 0
- 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 case where 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

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

$wx \backslash yz$	y			
	00	01	11	10
00	m_0 0	m_1 1	m_3 1	m_2 0
01	m_4 0	m_5 0	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

- 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 $F = yz + w'x'$

		y			
		00	01	11	10
wx	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

		y			
		yz		11	10
w	x	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
w	10	m_8 0	m_9 0	m_{11} 1	m_{10} 0

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

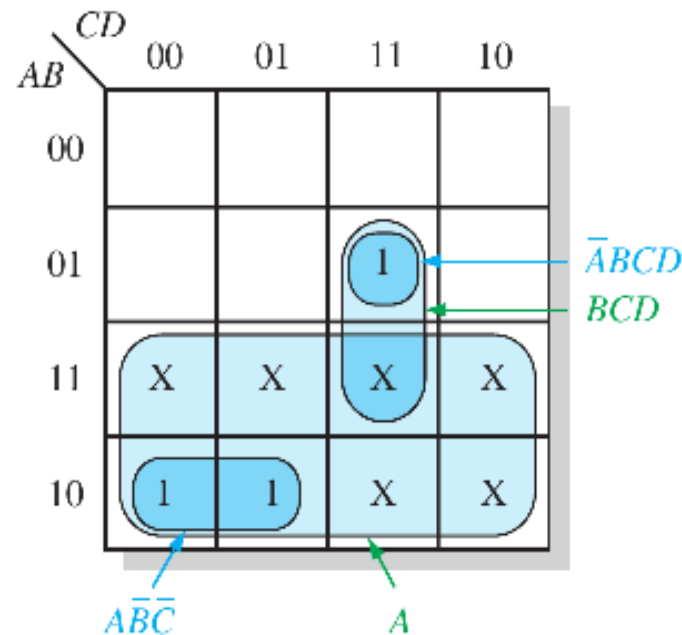
$$F = (z' + wy')' = z(w' + y)$$

$wx \backslash yz$		y			
		00	01	11	10
w	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
		z			

The Other states

Inputs				Output
<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>Y</i>
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Don't cares



Without "don't cares" $Y = \bar{A}\bar{B}C\bar{D} + \bar{A}BCD$

With "don't cares" $Y = A + BCD$