

Boolean Algebra and Logic Gates

Md. Zubair Alam Emon

<https://youtu.be/zDlpGo1bRwo>

Boolean Algebra

- In 1854 George Boole introduced a systematic treatment of logic and developed for this purpose an algebraic system now called *Boolean Algebra*
- In 1938 Shannon introduced a 2-value Boolean algebra called *switching algebra*, in which he demonstrated that properties of bi-stable electrical switching circuits can be represented by this algebra.

Boolean Algebra Theorems

- $x + 0 = x$
 - $x + x' = 1$
 - $x + x = x$
 - $x + 1 = 1$
 - $(x')' = x$
 - $x + y = y + x$
 - $x + (y + z) = (x + y) + z$
 - $x(y + z) = x.y + x.z$
 - $(x + y)' = x'y'$
 - $x + xy = x$
- $x . 1 = x$
 - $x . x' = 0$
 - $x . x = x$
 - $x . 0 = 0$
 - $x.y = y.x$
 - $x(yz) = (xy)z$
 - $x + yz = (x + y)(x + z)$
 - $(xy)' = x' + y'$
 - $x(x + y) = x$

Boolean Functions

- Boolean function expresses the logical relationship between binary variables.
- It is evaluated by determining the binary value of the expression for all possible values of the variables.
- It can be represented in a **truth table**.
- A Boolean function can be transformed from an algebraic expression into a **circuit diagram** composed of logic gates.

Example

- Check whether these statements are correct?

$$i) x y \bar{z} + \bar{x} y z + x y z + \bar{x} y \bar{z} = y$$

$$ii) (x + \bar{y} + \bar{z})(\bar{x} + \bar{z}) = \bar{x} \bar{y} + \bar{z}$$

$$iii) \bar{A} \bar{C} + A B C + A \bar{C} + A \bar{B} = A + C$$

$$iii) A'C' + ABC + AC' + AB'$$

$$= C'(A+A') + ABC + AB'(C+C')$$

$$= C' + ABC + AB'C + AB'C'$$

$$= C' + AC(B+B') + AB'C' = C'(1+AB') + AC = C' + AC$$

$$= (C'+A)(C'+C) = C'+A \neq A+C \quad \text{So, (iii) is incorrect}$$

Truth Table

- Truth table is a list of **all** combinations of 1's and 0's assigned to binary variables
 - It has a column that shows the value of the function for each binary combination.
- Number of rows in a truth table is 2^n
 - n is the number of variables in binary function
- Binary combinations for the truth table are obtained from binary numbers by counting from 0 through $2^n - 1$

Circuit Diagram

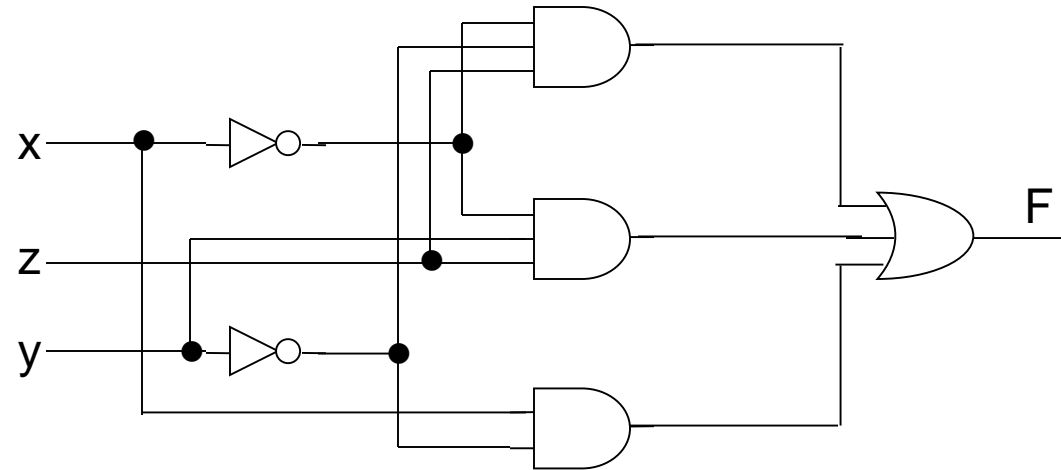
- Variables of the function act as inputs and the value of the function acts as the output of the circuit.
- Particular expression used for the function dictates the interconnection of gates in the circuit diagram.
- Using Boolean algebra rules obtain (if possible) simpler Boolean expression for the function and thus reduce the number of gates in the circuit and the number of inputs to the gate.

Example

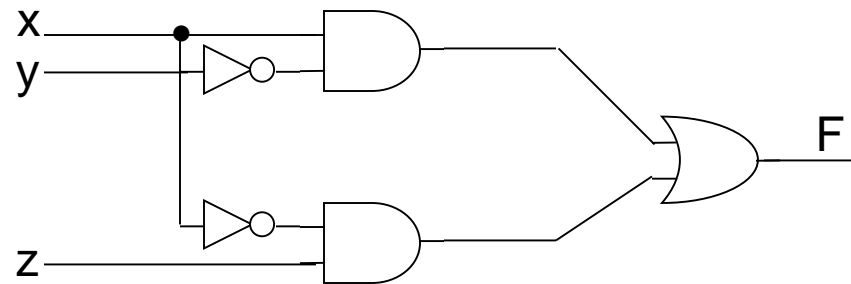
Boolean Function: $F = x'y'z + x'yz + xy'$

Truth Table

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



$$F = x'y'z + x'yz + xy' = x'z(y' + y) + xy' = x'z + xy'$$



DeMorgan's Theorem

- Complement of a function F is F'
- Complement of a function may be derived algebraically through DeMorgan's theorem
- Generalized form of DeMorgan's theorem:
 - The complement of a function is obtained by interchanging AND and OR operations and complementing each literal.
 - $(A + B + C + \dots + F)' = A' B' C' \dots F'$
 - $(A B C \dots F)' = A' + B' + C' + \dots + F'$

Canonical Forms

- Minterm or Standard Product
 - n variables can be combined to form 2^n minterm.
 - Minterm is obtained by ANDing all of the n variables.
 - Variable is primed if corresponding bit is 0
 - Otherwise the variable is unprimed.
- Maxterm or Standard Sum
 - n variables can be combined to form 2^n maxterm.
 - Maxterm is obtained by ORing all of the n variables.
 - Variable is primed if corresponding bit is 1
 - Otherwise the variable is unprimed.
- Maxterm and its corresponding minterm are complement of each other.

Canonical Forms

- A Boolean function can be expressed as a sum (i.e. ORing) of minterms.
 - Only those minterms that produces a 1 in truth table.
 - The function is often written using Σ symbol.
- A Boolean function can also be expressed as a product (i.e. ANDing) of maxterms.
 - Only those maxterms that produces a 0 in truth table.
 - The function is often written using Π symbol.
- Boolean functions expressed as a sum of minterms or product of maxterms are said to be in **canonical** form.

Min Terms and Max Terms

Table 2.3
Minterms and Maxterms for Three Binary Variables

<i>x</i>	<i>y</i>	<i>z</i>	Minterms		Maxterms	
			Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Example: $f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$

$$f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$$

Min Terms and Max Terms

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$f_1' = x'y'z' + x'yz' + x'yz + xy'z + xyz'$$

If we take the complement of f_1' , we obtain the function f_1 :

$$\begin{aligned}f_1 &= (x + y + z)(x + y' + z)(x' + y + z')(x' + y' + z) \\ &= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6\end{aligned}$$

Similarly, it is possible to read the expression for f_2 from the table:

$$\begin{aligned}f_2 &= (x + y + z)(x + y + z')(x + y' + z)(x' + y + z) \\ &= M_0 M_1 M_2 M_4\end{aligned}$$

Example

Express the Boolean function $F = A + B'C$ as a sum of minterms.

- One way to do it, is just put 1 where there is variable missing and use distributive law

$$F = A.1.1 + 1.B'C = A(B+B')(C+C') + (A+A')B'C$$

$\frac{111}{\downarrow}$	$\frac{110}{\downarrow}$	$\frac{101}{\downarrow}$	$\frac{100}{\downarrow}$	$\frac{001}{\downarrow}$	$\rightarrow 1$
$A B C$	$A B \bar{C}$	$A \bar{B} C$	$A \bar{B} \bar{C}$	$\bar{A} \bar{B} C$	

$F(A,B,C) = \sum(1, 4, 5, 6, 7)$

m_1

Canonical Form Conversion

- To convert from one canonical form to another, interchange the symbols Σ and Π and list those numbers missing from the original form.
- To find the missing terms, realize that total number of minterms or maxterms is 2^n
 - n is the number of variables in the function.
- e.g. $F(x, y, z) = \Sigma(1, 3, 6, 7) = \Pi(0, 2, 4, 5)$

<https://youtu.be/mfSxRTyknao>

Example

Express the Boolean function $F = A + B'C$ as a sum of minterms.

- For previous example,

$$A B C + A B \bar{C} + A \bar{B} C + A \bar{B} \bar{C} + \bar{A} \bar{B} C$$

$$F(A, B, C) = \sum(1, 4, 5, 6, 7) \longrightarrow \text{Minterms}$$

$$= \prod(0, 2, 3) \longrightarrow \text{Maxterms}$$

Standard Forms

- Sum of Products (SOP)
 - Boolean expression with AND (i.e. **product**) terms.
 - Each term may have any (1 to n) number of literals.
 - The **sum** denotes the ORing of these terms.
- Product of Sums (POS)
 - Boolean expression containing OR (i.e. **sum**) terms.
 - Each term may have any (1 to n) number of literals
 - The **product** denotes the ANDing of these terms.
- A two-level implementation is preferred because it produces the least amount of delay through the gates when signal propagates from the inputs to the output.

Example

- Say, delay for AND gate and OR gate is respectively 50ps and 70ps. Calculate the time taken for each circuit to reach the output.

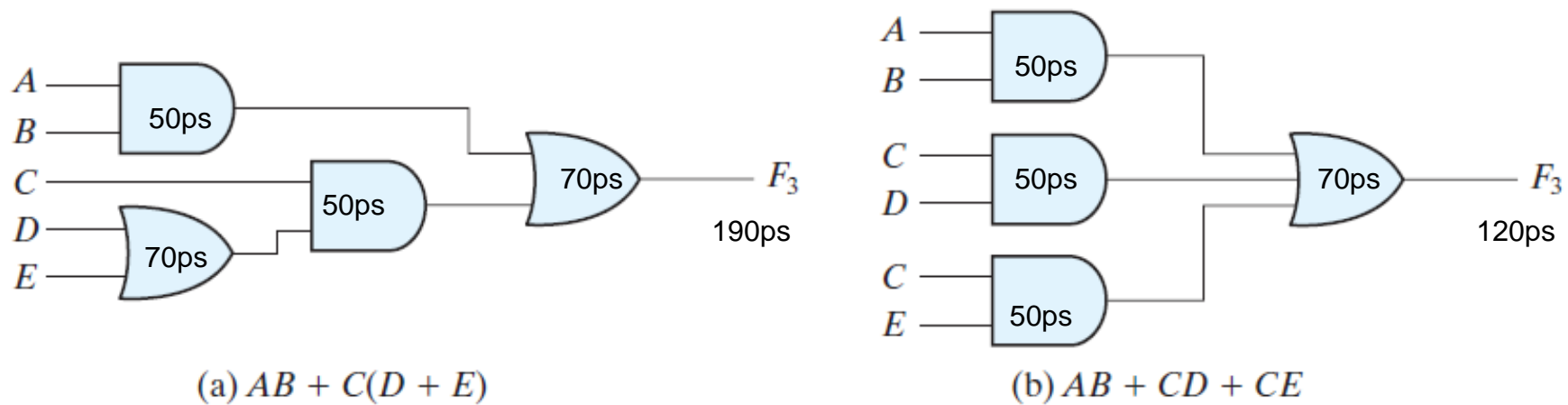


FIGURE 2.4
Three- and two-level implementation







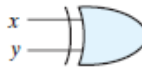
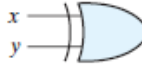
Logic Operations

- There are 2^{2^n} different functions for n binary variables.
- Only a handful are used by the logic designers.
 - AND \rightarrow NAND
 - OR \rightarrow NOR
 - XOR \rightarrow XNOR
 - BUFFER \rightarrow INVERTER

Digital Logic Gates

- AND
- OR
- INVERTER
- BUFFER
- NAND
- NOR
- XOR
- XNOR

<https://youtu.be/voio6SD5xu0>

AND		$F = x \cdot y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inverter		$F = x'$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	F	0	1	1	0									
x	F																	
0	1																	
1	0																	
Buffer		$F = x$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	F	0	0	1	1									
x	F																	
0	0																	
1	1																	
NAND		$F = (xy)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = (x + y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

Extension to Multiple Inputs

- Draw the circuit diagram of:

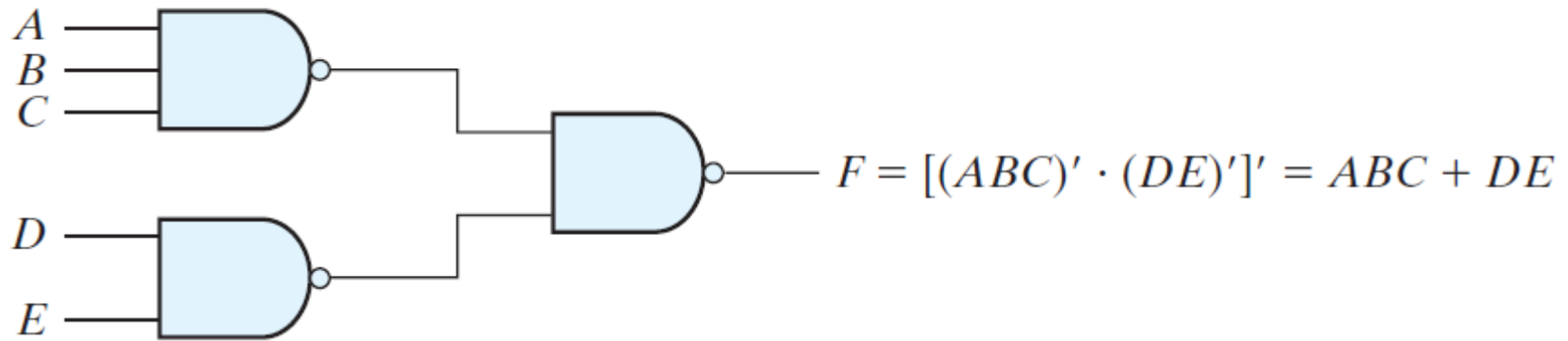
$$F_1(x,y,z) = ((x + y)' + z)'$$

$$F_2(x,y,z) = (x + (y + z)')'$$

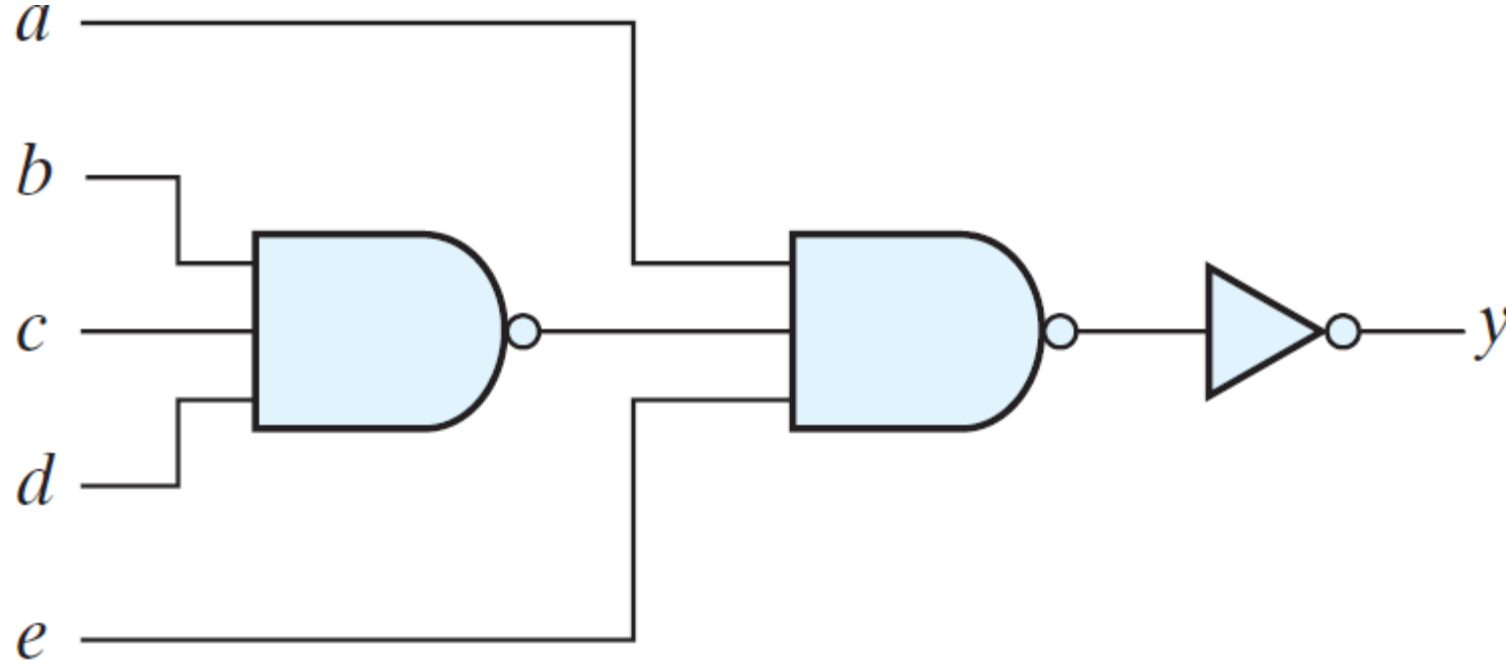
Are these two same ?

- 3-input NOR gate
- 3-input NAND gate
- Cascaded NAND gate
$$F_3 = [(ABC)' \cdot (DE)']'$$
- 3-input XOR gate

Example of Cascaded NAND gate



Example: Construct Boolean expression and truth table of the circuit



$$\begin{aligned}
 y &= ((a(bcd)'e)')' = a(bcd)'e = a(b' + c' + d')e = ab'e + ac'e + \\
 &ad'e = ab'(c+c')(d+d')e + a(b+b')c'(d+d')e + a(b+b')(c+c')d'e \\
 &= ab'cde + ab'cd'e + ab'c'de + ab'c'd'e + \dots \\
 &= \sum(17, 19, 21, 23, 25, 27, 29)
 \end{aligned}$$

Integrated Circuits

- IC is a silicon semiconductor, called **chip**, containing the electronic components for constructing digital gates.
- Digital ICs are often categorized according to their circuit complexity as measured by the number of logic gates in a single package:
 - Small-scale Integration
 - Medium-scale Integration
 - Large-scale Integration
 - Very Large-scale Integration

Digital Logic Families

- Popular logic families:
 - TTL → transistor-transistor logic
 - ECL → emitter-coupled logic
 - MOS → metal-oxide semiconductor
 - CMOS → complementary metal-oxide semiconductor
- Important parameters for evaluating digital logic families:
 - Fan-out
 - number of input pins an output signal can drive
 - Fan-in
 - number of inputs available in a gate
 - Power dissipation
 - Power consumed by a gate from power supply
 - Propagation delay
 - Average time for a signal to go from input to output
 - Noise margin
 - Maximum external noise that does not change output signal

Computer Aided Design

- CAD tools aid in the development of digital hardware by automating the design process, starting from design spec to fabrication
- Some CAD tools are used for schematic capture
- HDL is used to describe digital hardware
 - Represents logic diagrams and other digital info in textual form.
 - Used to simulate the system before its construction to check the functionality
 - Used to verify the operation before it is submitted to fabrication.

HW

- 2.2~2.9, 2.11~2.15, 2.17~2.24, 2.27~2.28