

DIGITAL ELECTRONICS



Dr. Pradip Kumar Sahu
Associate Professor,
Department of Information Technology,
VSSUT, Burla

Lecture of Module 2

Logic Gates

Overview

- ▶ **Introduction**
- ▶ **Logical Operators**
- ▶ **Basic Gates**
- ▶ **Universal Gates**
- ▶ **Realization of Basic Gates using Universal Gates**
- ▶ **Other Logic Gates**

Introduction

- ▶ Binary variables take on one of two values.
- ▶ Logical operators operate on binary values and binary variables.
- ▶ Basic logical operators are the logic functions - AND, OR and NOT.
- ▶ Logic gates implement logic functions.
- ▶ Boolean Algebra: a useful mathematical system for specifying and transforming logic functions.
- ▶ Boolean algebra is a foundation for designing and analyzing digital systems.

Binary Variables

- ▶ **Binary values have different names:**
 - ▶ **True/False**
 - ▶ **On/Off**
 - ▶ **Yes/No**
 - ▶ **1/0**
- ▶ **We use 1 and 0 to denote the two values.**
- ▶ **Variable identifier examples:**
 - ▶ **A, B, x, y, z, or X_1 , X_2 etc.**

Logical Operations

- ▶ The three basic logical operations are:
 - ▶ AND
 - ▶ OR
 - ▶ NOT
- ▶ AND is denoted by a dot (\cdot)
- ▶ OR is denoted by a plus ($+$)
- ▶ NOT is denoted by an over bar ($\bar{}$), a single quote mark ($'$) after, or (\sim) before the variable

Operator

- Operators operate on binary values and binary variables.
- Operations are defined on the values "0" and "1" for each operator:

AND

$$0 \cdot 0 = 0$$

$$0 \cdot 1 = 0$$

$$1 \cdot 0 = 0$$

$$1 \cdot 1 = 1$$

OR

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 1$$

NOT

$$\bar{0} = 1$$

$$\bar{1} = 0$$

Truth Table

- ▶ *Truth table* - a tabular listing of the values of a function for all possible combinations of values on its arguments.
- ▶ Example: Truth tables for the basic logic operations:

AND		
X	Y	$Z = X \cdot Y$
0	0	0
0	1	0
1	0	0
1	1	1

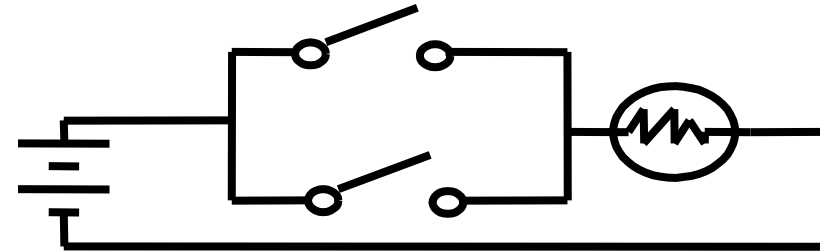
OR		
X	Y	$Z = X + Y$
0	0	0
0	1	1
1	0	1
1	1	1

NOT	
X	$Z = \bar{X}$
0	1
1	0

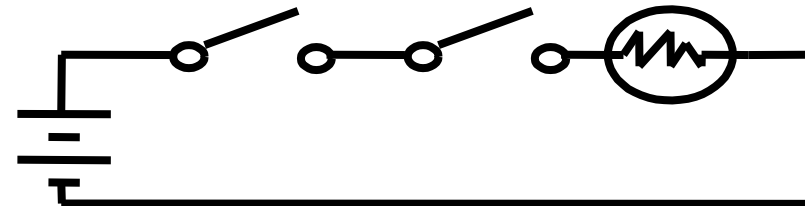
Logic Function Implementation

- ▶ Using Switches
 - ▶ For inputs:
 - ▶ logic 1 is switch closed
 - ▶ logic 0 is switch open
 - ▶ For outputs:
 - ▶ logic 1 is light on
 - ▶ logic 0 is light off.

Switches in parallel => OR



Switches in series => AND



Logic Gates

- ▶ In the earliest computers, switches were opened and closed by magnetic fields produced by energizing coils in *relays*. The switches in turn opened and closed the current paths.
- ▶ Later, *vacuum tubes* that open and close current paths electronically replaced relays.
- ▶ Today, *transistors* are used as electronic switches that open and close current paths.
- ▶ NOT, AND and OR Gates (Basic gates)
- ▶ NAND and NOR Gates (Universal logic gates)

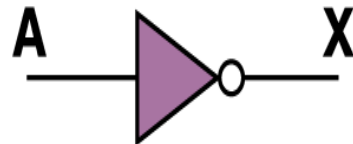
NOT Gate

A NOT gate accepts one input signal (0 or 1) and returns the opposite signal as output.

Boolean Expression

$$X = A'$$

Logic Diagram Symbol



Truth Table

A	X
0	1
1	0

AND Gate

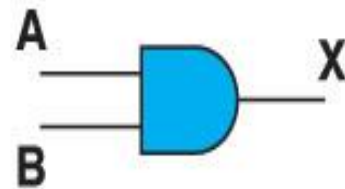
If all inputs are 1, the output is 1; otherwise, the output is 0

Or if any input is 0, output is 0

Boolean Expression

$$X = A \cdot B$$

Logic Diagram Symbol



Truth Table

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

OR Gate

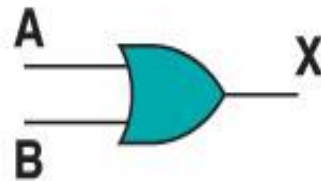
If all inputs are 0, the output is 0; otherwise, the output is 1

Or if any input is 1, output will be 1

Boolean Expression

$$X = A + B$$

Logic Diagram Symbol



Truth Table

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

Universal Gates

- ❑ Universal Logic Gate: Any basic gate or logic function can be realized using this gate.
- ❑ Two universal logic gates -
 - ❖ NAND
 - ❖ NOR

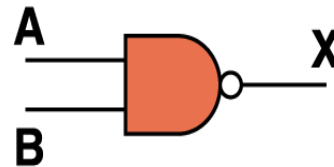
NAND Gate

If all inputs are 1, the output is 0; otherwise, the output is 1
Or if any input is 0, output will be 1

Boolean Expression

$$X = (A \cdot B)'$$

Logic Diagram Symbol



Truth Table

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

NOR Gate

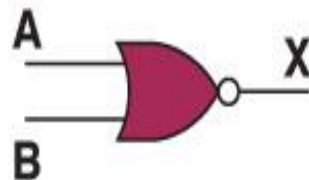
If all inputs are 0, the output is 1; otherwise, the output is 0

Or if any input is 1, output will be 0

Boolean Expression

$$X = (A + B)'$$

Logic Diagram Symbol

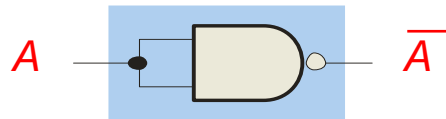


Truth Table

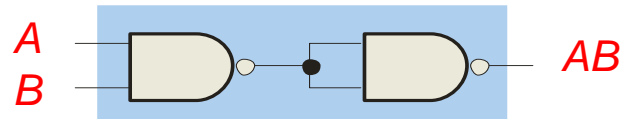
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Realization

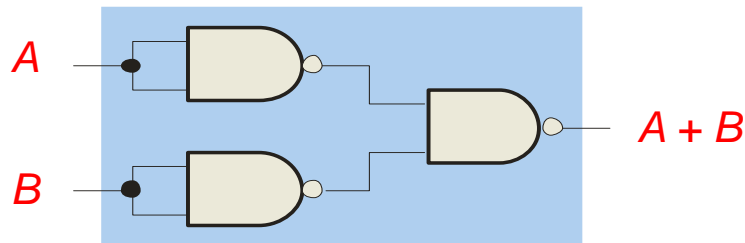
NAND gates are sometimes called **universal** gates because they can be used to produce the other basic Boolean functions.



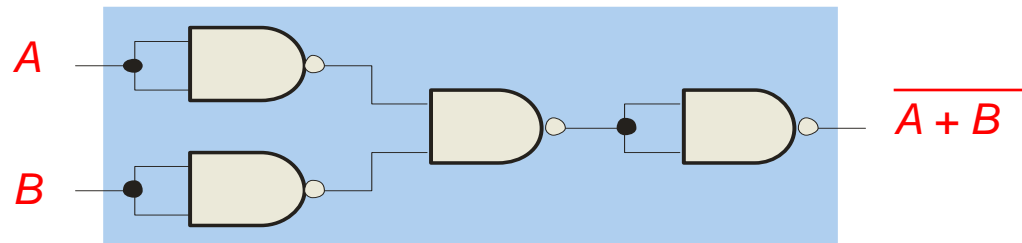
Inverter



AND gate



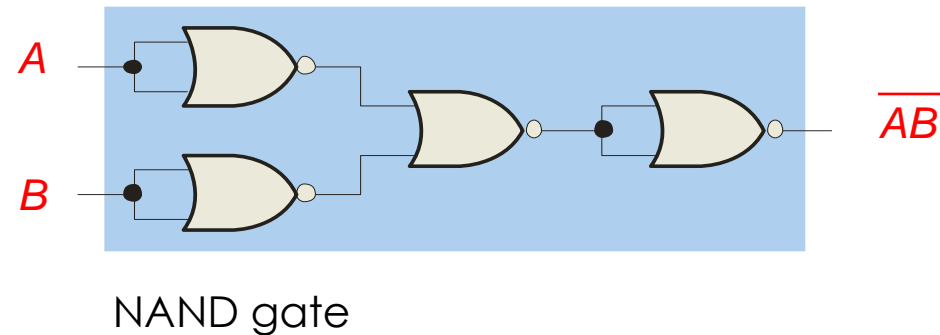
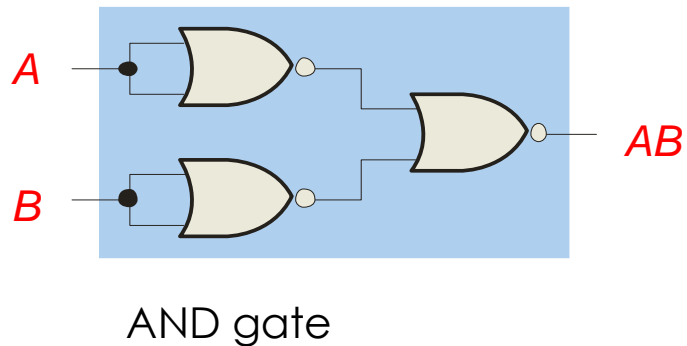
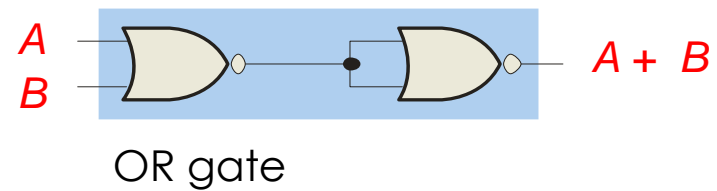
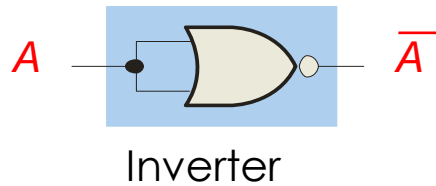
OR gate



NOR gate

Realization

NOR gates are also **universal** gates and can form all of the basic gates.



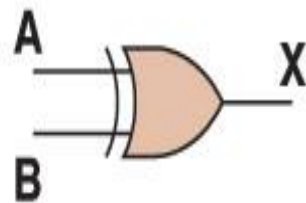
XOR Gate

If odd numbers of inputs are 1, the output is 1; otherwise, the output is 0

Boolean Expression

$$X = A \oplus B$$

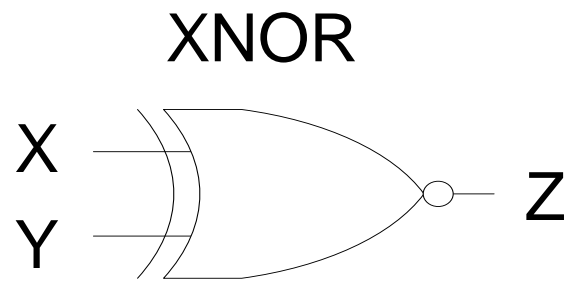
Logic Diagram Symbol



Truth Table

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

X-NOR Gate



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

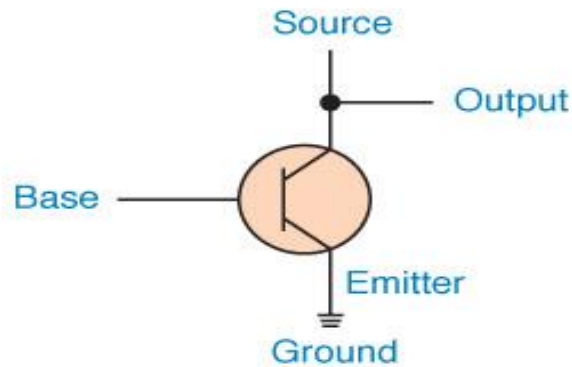
Constructing Gates

Transistor

A device that acts either as a wire that conducts electricity or as a resistor that blocks the flow of electricity, depending on the voltage level of an input signal.

A transistor has no moving parts, yet acts like a switch.

It is made of a **semiconductor** material, which is neither a particularly good conductor of electricity nor a particularly good insulator.

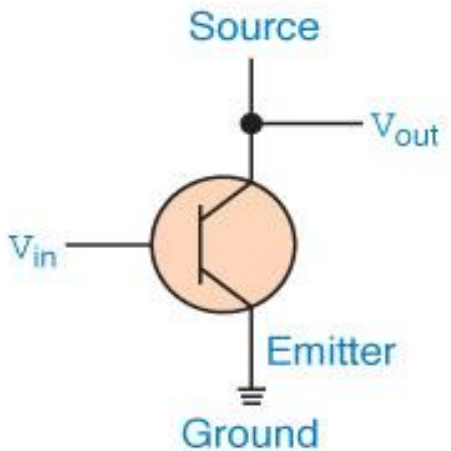
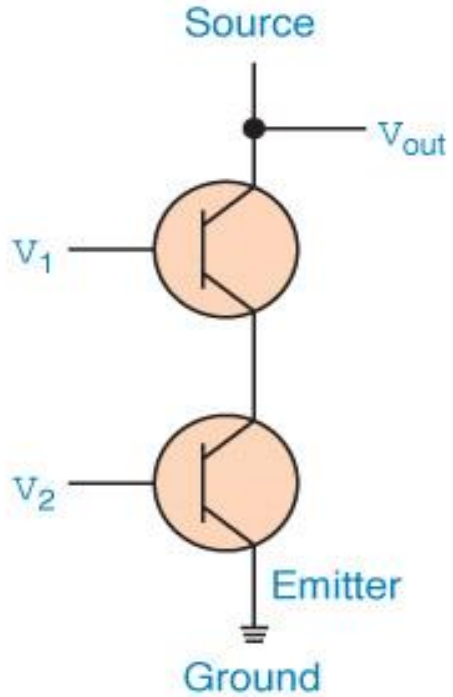
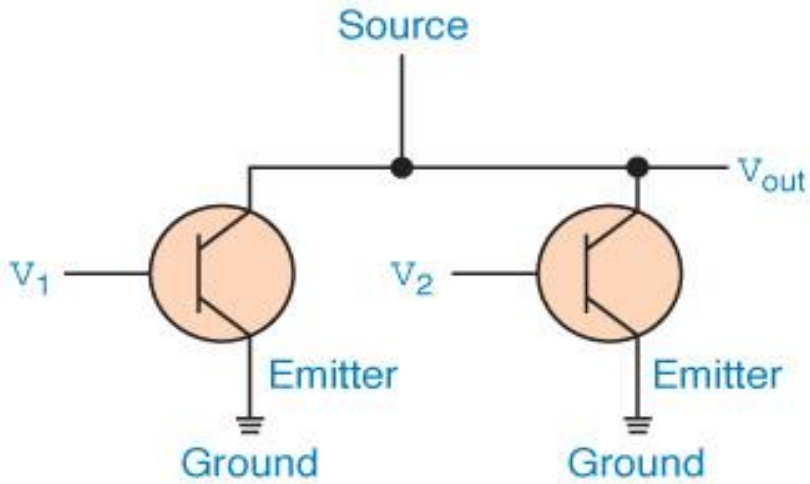


A transistor has three terminals

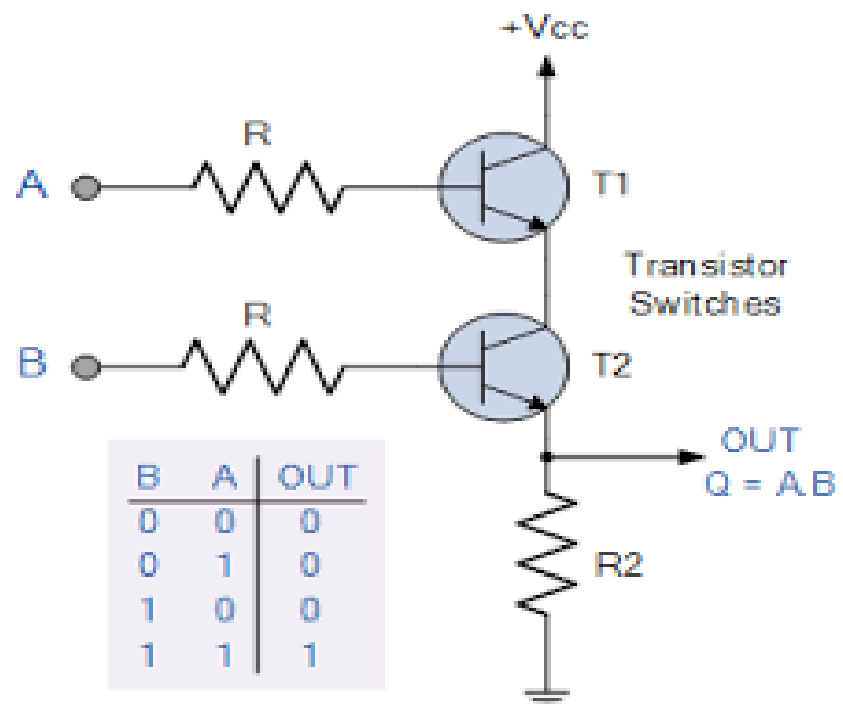
A source

A base

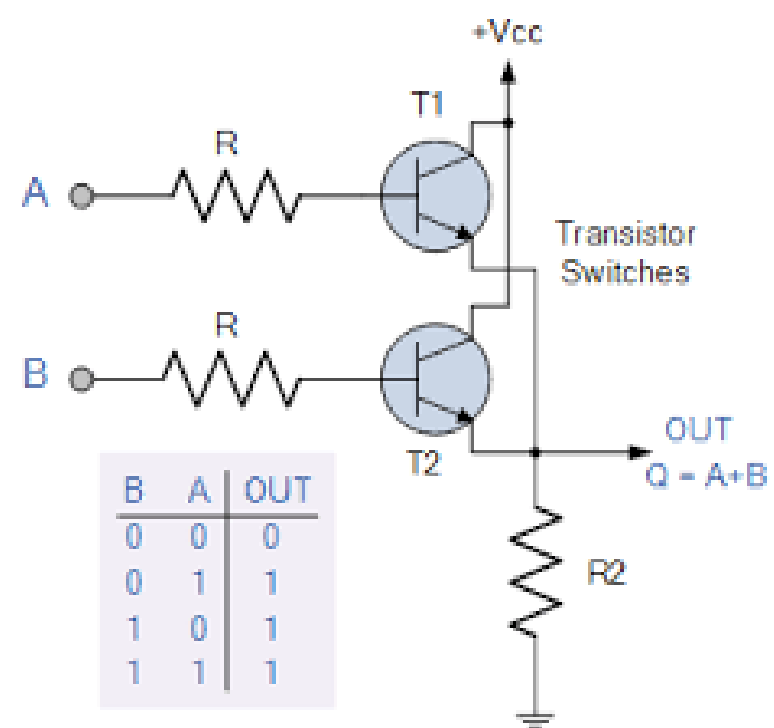
An emitter, typically connected to a ground wire

NOT gate	NAND gate	NOR gate
 <p>Source</p> <p>V_{in}</p> <p>V_{out}</p> <p>Emitter</p> <p>Ground</p>	 <p>Source</p> <p>V_{out}</p> <p>V_1</p> <p>V_2</p> <p>Emitter</p> <p>Ground</p>	 <p>Source</p> <p>V_{out}</p> <p>V_1</p> <p>V_2</p> <p>Emitter</p> <p>Ground</p>

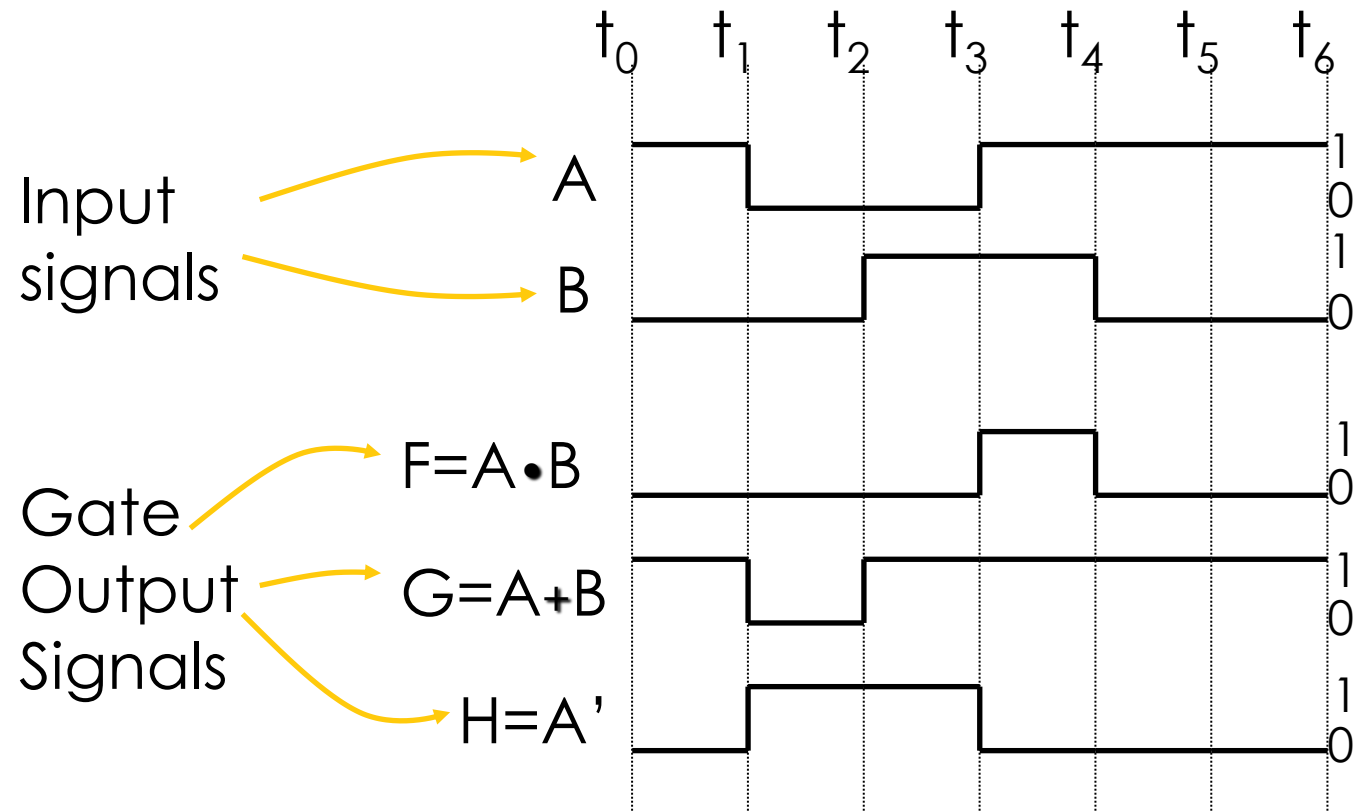
AND Gate



OR Gate



Timing Diagram

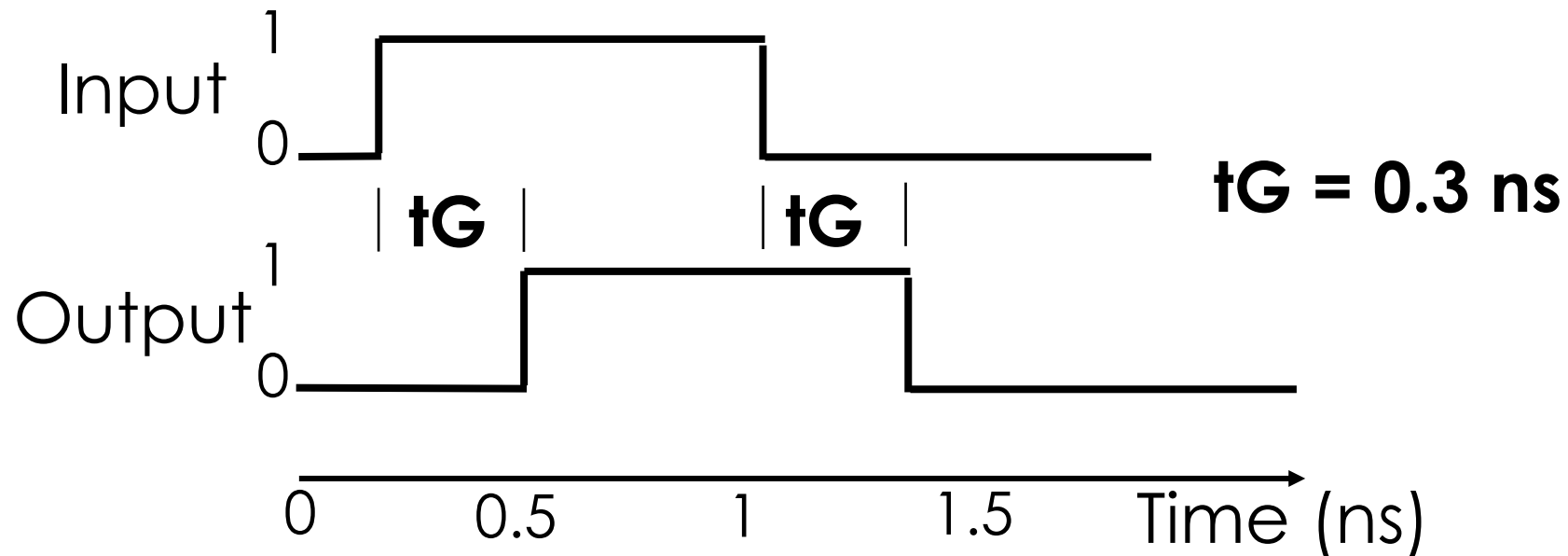


Transitions

Basic Assumption:
Zero time for signals to propagate through gates

Gate Delay

- ▶ In actual physical gates, if one or more input changes causes the output to change, the output change does not occur instantaneously.
- ▶ The delay between an input change(s) and the resulting output change is the *gate delay* denoted by t_G :



Lecture of Module 2

Boolean Algebra

Overview

- ▶ Introduction
- ▶ Boolean Algebra
- ▶ Properties
- ▶ Algebraic Manipulation
- ▶ De-Morgan Theorem
- ▶ Complementation
- ▶ Truth Table

Introduction

- ▶ Understand the relationship between Boolean logic and digital computer circuits.
- ▶ Learn how to design simple logic circuits.
- ▶ Understand how digital circuits work together to form complex computer systems.
- ▶ In the latter part of the nineteenth century, **George Boole** suggested that logical thought could be represented through mathematical equations.
- ▶ Computers, as we know them today, are implementations of Boole's *Laws of Thought*.
- ▶ In this chapter, you will learn the simplicity that constitutes the essence of the machine (Boolean Algebra).

Boolean algebra

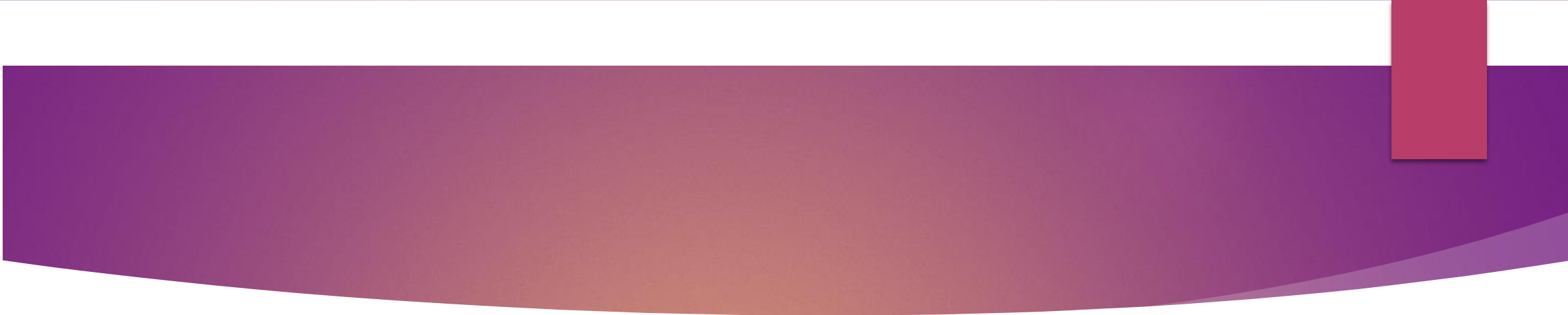
- ▶ Boolean algebra is a mathematical system for the manipulation of variables that can have one of two values.
 - ▶ In formal logic, these values are “true” and “false”.
 - ▶ In digital systems, these values are “on” and “off,” 1 and 0, or “high” and “low”.
- ▶ Boolean expressions are created by performing operations on Boolean variables.
 - ▶ Common Boolean operators include AND, OR, NOT, XOR, NAND and NOR.

- ▶ A Boolean operator can be completely described using a truth table.
- ▶ The truth table for the Boolean operators AND, OR and NOT are shown at the right.
- ▶ The AND operator is known as a Boolean product.
- ▶ The OR operator is the Boolean sum.
- ▶ The NOT operation is most often designated by an over - bar. It is sometimes indicated by a prime mark (') or an “elbow” (\neg). It is Boolean complementation.

X AND Y		
X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

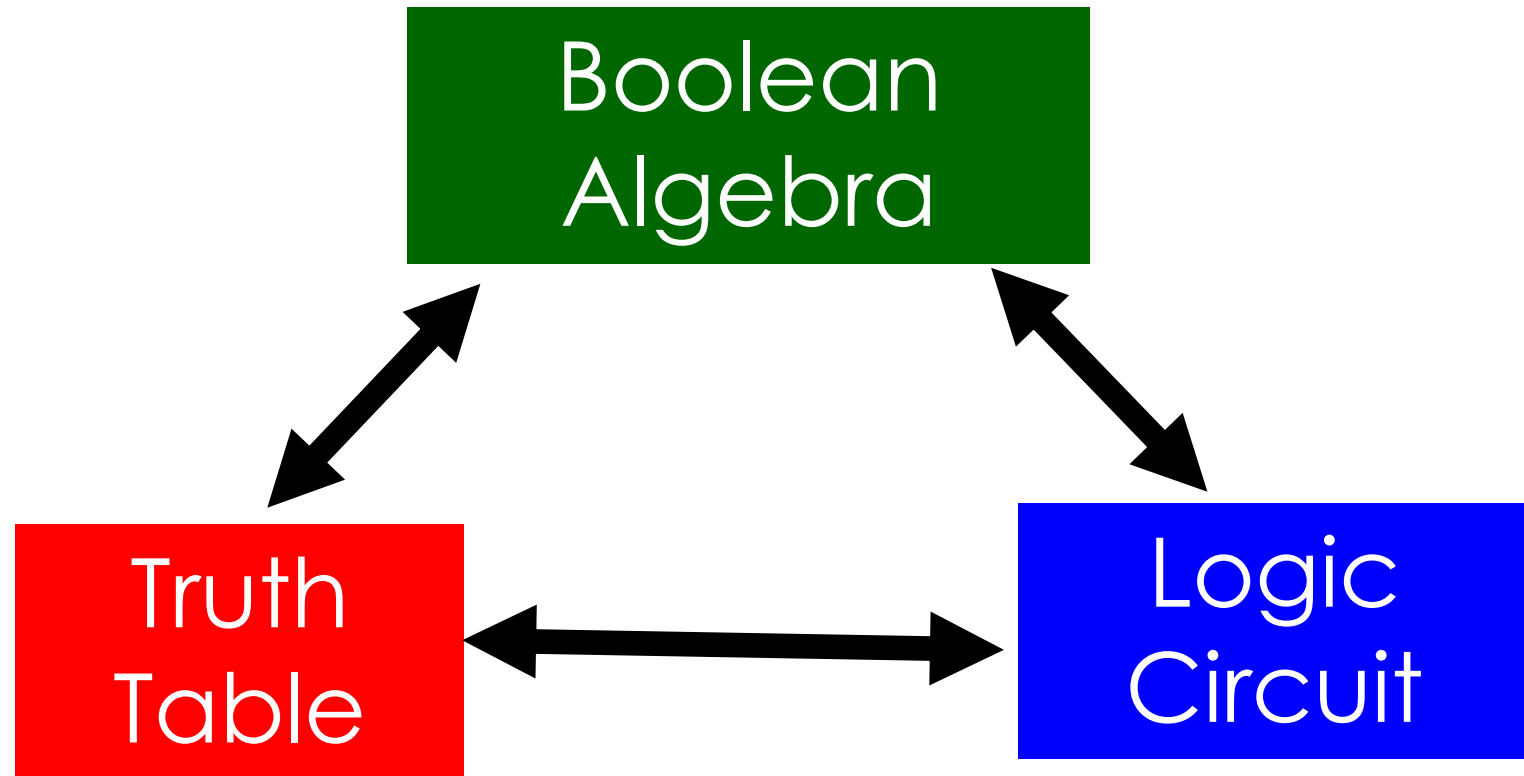
X OR Y		
X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

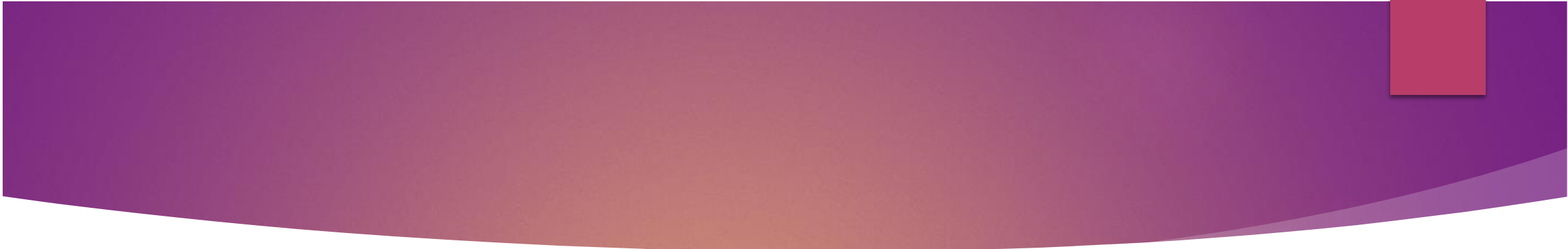
NOT X	
X	\overline{X}
0	1
1	0

- 
- ▶ A Boolean function has:
 - At least one Boolean variable,
 - At least one Boolean operator, and
 - At least one input from the set $\{0,1\}$
 - ▶ It produces an output that is also a member of the set $\{0,1\}$

So, the binary numbering system is so handy in digital systems

Conceptually



- 
- ▶ Digital computers contain circuits that implement Boolean functions.
 - ▶ The **simpler** that we can make a Boolean function, the **smaller** the circuit that will result.
 - ▶ Simpler circuits are cheaper to build, consume less power, and run faster than complex circuits.
 - ▶ With this in mind, we always want to reduce our Boolean functions to their simplest form.
 - ▶ There are a number of Boolean identities that help us to do this.

Properties of Boolean Algebra

- Most Boolean identities have an AND (product) form as well as an OR (sum) form.

Identity Name	AND Form	OR Form
Identity Law	$1x = x$	$0 + x = x$
Null Law	$0x = 0$	$1 + x = 1$
Idempotent Law	$xx = x$	$x + x = x$
Inverse Law	$x\bar{x} = 0$	$x + \bar{x} = 1$

- Next group of Boolean identities should be familiar to you from your study of algebra:

Identity Name	AND Form	OR Form
Commutative Law	$xy = yx$	$x+y = y+x$
Associative Law	$(xy)z = x(yz)$	$(x+y)+z = x+(y+z)$
Distributive Law	$x+yz = (x+y)(x+z)$	$x(y+z) = xy+xz$

- ▶ Last group of Boolean identities are perhaps the most useful.
- ▶ If you have studied set theory or formal logic, these laws are also familiar to you.

Identity Name	AND Form	OR Form
Absorption Law	$x(x+y) = x$	$x + xy = x$
DeMorgan's Law	$\overline{(xy)} = \bar{x} + \bar{y}$	$\overline{(x+y)} = \bar{x}\bar{y}$
Double Complement Law	$\overline{(\bar{x})} = x$	

- We can use Boolean identities to simplify the function:

$$F(X, Y, Z) = (X + Y)(X + \bar{Y})(\overline{XZ})$$

as follows:

$$\begin{aligned} & (X + Y)(X + \bar{Y})(\overline{XZ}) \\ & (X + Y)(X + \bar{Y})(\bar{X} + Z) \\ & (XX + X\bar{Y} + XY + Y\bar{Y})(\bar{X} + Z) \\ & ((X + Y\bar{Y}) + X(Y + \bar{Y}))(\bar{X} + Z) \\ & ((X + 0) + X(1))(\bar{X} + Z) \\ & X(\bar{X} + Z) \\ & X\bar{X} + XZ \\ & 0 + XZ \\ & XZ \end{aligned}$$

Idempotent Law (Rewriting)
DeMorgan's Law
Distributive Law
Commutative & Distributive Laws
Inverse Law
Idempotent Law
Distributive Law
Inverse Law
Idempotent Law

Duality Property of Boolean Algebra

- One expression can be obtained from another expression by replacing the every 1 with 0, every 0 with 1, every (+) with (.), every (.) with (+).
- Any pair of expression satisfying this property is called Dual Expression.
- With respect to duality, Identities 1 – 8 have the following relationship:

1. $X + 0 = X$

2. $X \cdot 1 = X$

(dual of 1)

3. $X + 1 = 1$

4. $X \cdot 0 = 0$

(dual of 3)

5. $X + X = X$

6. $X \cdot X = X$

(dual of 5)

7. $X + X' = 1$

8. $X \cdot X' = 0$

(dual of 7)

9. $X + Y = Y + X$

10. $X \cdot Y = Y \cdot X$

(dual of 9)

11. $X \cdot Y + Z = X \cdot Y + X \cdot Z$

12. $X + Y \cdot Z = (X + Y) \cdot (X + Z)$

(dual of 11)

Algebraic Manipulation

- ▶ Boolean algebra is a useful tool for simplifying digital circuits.
- ▶ Why do it? Simpler can mean cheaper, smaller, faster.
- ▶ Example: Simplify $F = x'yz + x'yz' + xz$.
$$\begin{aligned} F &= x'yz + x'yz' + xz \\ &= x'y(z+z') + xz \\ &= x'y \cdot 1 + xz \\ &= x'y + xz \end{aligned}$$
- ▶ Example: Prove $x'y'z' + x'yz' + xyz' = x'z' + yz'$
- ▶ Proof: $x'y'z' + x'yz' + xyz'$
$$\begin{aligned} &= x'y'z' + x'yz' + x'yz' + xyz' \\ &= x'z'(y'+y) + yz'(x'+x) \\ &= x'z' \cdot 1 + yz' \cdot 1 \\ &= x'z' + yz' \end{aligned}$$

Complementation

- ▶ Sometimes it is more economical to build a circuit using the complement of a function (and complementing its result) than it is to implement the function directly.
- ▶ DeMorgan's law provides an easy way of finding the complement of a Boolean function.
- ▶ DeMorgan's law states:

$$\overline{(xy)} = \bar{x} + \bar{y} \quad \text{and} \quad \overline{(x+y)} = \bar{x}\bar{y}$$

► Find the complement of $F(x, y, z) = x y' z' + x' y z$

$$\begin{aligned} \text{► } G = F' &= (xy'z' + x'yz)' \\ &= (xy'z')' \cdot (x'yz)' \quad \text{DeMorgan} \\ &= (x'+y+z) \cdot (x+y'+z') \quad \text{DeMorgan again} \end{aligned}$$

► Note: The complement of a function can also be derived by finding the function's *dual*, and then complementing all of the literals

Truth Table

- ▶ Enumerates all possible combinations of variable values and the corresponding function value
- ▶ Truth tables for some arbitrary functions
 $F_1(x,y,z)$, $F_2(x,y,z)$, and $F_3(x,y,z)$ are shown to the right.
- ▶ Truth table: a unique representation of a Boolean function
- ▶ If two functions have identical truth tables, the functions are equivalent (and vice-versa).
- ▶ The size of a truth table grows exponentially with the number of variables involved. This motivates the use of Boolean Algebra.

x	y	z		F_1	F_2	F_3
0	0	0		0	1	1
0	0	1		0	0	1
0	1	0		0	0	1
0	1	1		0	1	1
1	0	0		0	1	0
1	0	1		0	1	0
1	1	0		0	0	0
1	1	1		1	0	1

Lecture Module 2

Standard SOP and POS

Overview

- ▶ Introduction
- ▶ SOP and POS
- ▶ Minterms and Maxterms
- ▶ Canonical Forms
- ▶ Conversion Between Canonical Forms
- ▶ Standard Forms

Introduction

- ▶ Through our exercises in simplifying Boolean expressions, we see that there are numerous ways of stating the same Boolean expression.
 - ▶ These “synonymous” forms are *logically equivalent*.
 - ▶ Logically equivalent expressions have identical truth tables.
- ▶ In order to eliminate as much confusion as possible, designers express Boolean functions in *standardized* or *canonical* form.

SOP and POS

- ▶ There are two canonical forms for Boolean expressions: Sum-Of-Products (SOP) and Product-Of-Sums (POS).
 - ▶ Recall the Boolean product is the AND operation and the Boolean sum is the OR operation.
- ▶ In the Sum-Of-Products form, ANDed variables are ORed together.
 - ▶ For example: $F(x, y, z) = xy + xz + yz$
- ▶ In the Product-Of-Sums form, ORed variables are ANDed together:
 - ▶ For example: $F(x, y, z) = (x+y)(x+z)(y+z)$

Definitions

- ▶ *Literal*: A variable or its complement
- ▶ *Product term*: literals connected by \cdot
- ▶ *Sum term*: literals connected by $+$
- ▶ *Minterm*: a product term in which all the variables appear exactly once, either complemented or un-complemented
- ▶ *Maxterm*: a sum term in which all the variables appear exactly once, either complemented or un-complemented

Truth Table notation for Minterms and Maxterms

- ▶ Minterms and Maxterms are easy to denote using a truth table.
- ▶ Example:
Assume 3 variables x,y,z (order is fixed)
- ▶ Any Boolean function $F()$ can be expressed as a *unique* **sum** of **minterms** and a unique **product** of **maxterms** (under a fixed variable ordering).
- ▶ In other words, every function $F()$ has two canonical forms:
 - ▶ Canonical Sum-Of-Products (sum of minterms)
 - ▶ Canonical Product-Of-Sums (product of maxterms)

x	y	z	Minterm	Maxterm
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	$xyz = m_7$	$x'+y'+z' = M_7$

Canonical Forms

► Canonical Sum-Of-Products:

The minterms included are those m_j such that $F() = 1$ in row j of the truth table for $F()$.

► Canonical Product-Of-Sums:

The maxterms included are those M_j such that $F() = 0$ in row j of the truth table for $F()$.

- $f_1(a,b,c) = \sum m(1,2,4,6)$, where \sum indicates that this is a sum-of-products form, and $m(1,2,4,6)$ indicates that the minterms to be included are m_1 , m_2 , m_4 , and m_6 .
- $f_1(a,b,c) = \prod M(0,3,5,7)$, where \prod indicates that this is a product-of-sums form, and $M(0,3,5,7)$ indicates that the maxterms to be included are M_0 , M_3 , M_5 , and M_7 .
- Since $m_j = M_j'$ for any j ,
$$\sum m(1,2,4,6) = \prod M(0,3,5,7) = f_1(a,b,c)$$

Conversion Between Canonical Forms

- ▶ Replace \sum with \prod (or *vice versa*) and replace those j 's that appeared in the original form with those that do not.

- ▶ Example:

$$f_1(a,b,c) = a'b'c + a'bc' + ab'c' + abc'$$

$$= m_1 + m_2 + m_4 + m_6$$

$$= \sum(1,2,4,6)$$

$$= \prod(0,3,5,7)$$

$$= (a+b+c) \cdot (a+b'+c') \cdot (a'+b+c') \cdot (a'+b'+c')$$

$$F = \overline{X}\overline{Y}\overline{Z} + \overline{X}Y\overline{Z} + X\overline{Y}Z + XYZ = m_0 + m_2 + m_5 + m_7 = \sum m(0, 2, 5, 7)$$

$$\overline{F} = \overline{X}\overline{Y}Z + \overline{X}YZ + X\overline{Y}\overline{Z} + XY\overline{Z} = m_1 + m_3 + m_4 + m_6 = \sum m(1, 3, 4, 6)$$

$$\overline{F} = m_1 + m_3 + m_4 + m_6$$

$$\Rightarrow F = \overline{m_1 + m_3 + m_4 + m_6} = \overline{m_1} \cdot \overline{m_3} \cdot \overline{m_4} \cdot \overline{m_6}$$

$$\begin{aligned} \Rightarrow F &= M_1 \cdot M_3 \cdot M_4 \cdot M_6 = (X + Y + \overline{Z})(X + \overline{Y} + \overline{Z})(\overline{X} + Y + Z)(\overline{X} + \overline{Y} + Z) \\ &= \prod M(1, 3, 4, 6) \end{aligned}$$

Standard Forms

- Standard forms are “*like*” canonical forms, except that not all variables need appear in the individual product (SOP) or sum (POS) terms.
- Example:
$$f_1(a,b,c) = a'b'c + bc' + ac'$$

is a *standard* sum-of-products form
- $$f_1(a,b,c) = (a+b+c) \cdot (b'+c') \cdot (a'+c')$$

is a *standard* product-of-sums form.

Conversion of SOP from standard to canonical form

- ▶ Expand *non-canonical* terms by inserting equivalent of 1 in each missing variable x:

$$(x + x') = 1$$

- ▶ Remove duplicate minterms

- ▶
$$\begin{aligned} f_1(a,b,c) &= a'b'c + bc' + ac' \\ &= a'b'c + (a+a')bc' + a(b+b')c' \\ &= a'b'c + abc' + a'bc' + abc' + ab'c' \\ &= a'b'c + abc' + a'bc' + ab'c' \end{aligned}$$

Conversion of POS from standard to canonical form

- ▶ Expand non-canonical terms by adding 0 in terms of missing variables (*e.g.*, $xx' = 0$) and using the distributive law
- ▶ Remove duplicate maxterms
- ▶
$$\begin{aligned} f_1(a,b,c) &= (a+b+c) \cdot (b'+c') \cdot (a'+c') \\ &= (a+b+c) \cdot (\textcolor{violet}{a}a' + b' + c') \cdot (a' + \textcolor{violet}{b}b' + c') \\ &= (a+b+c) \cdot (a+b'+c') \cdot (\textcolor{red}{a'} + \textcolor{red}{b'} + \textcolor{red}{c'}) \cdot (a' + b + c') \cdot (\textcolor{red}{a'} + \textcolor{red}{b'} + \textcolor{red}{c'}) \\ &= (a+b+c) \cdot (a+b'+c') \cdot (a' + b' + c') \cdot (a' + b + c') \end{aligned}$$

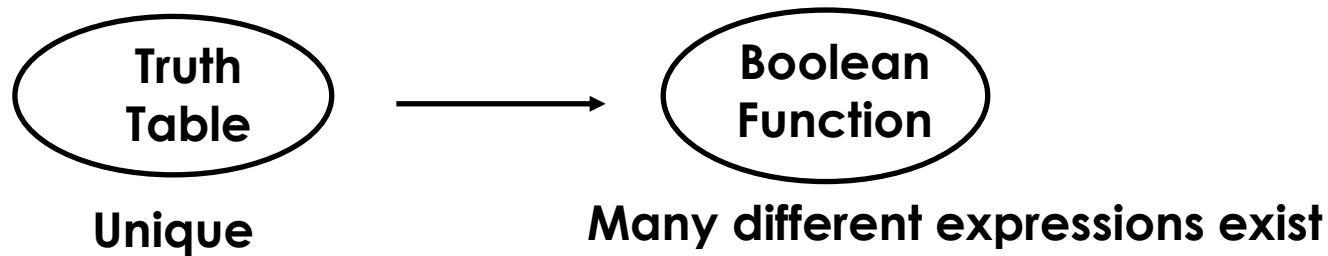
Lecture of Module 2

Minimization Techniques

Overview

- ▶ Introduction
- ▶ Karnaugh Map (K-Map)
- ▶ Simplification Rules
- ▶ K-Map Simplification for Two Variables
- ▶ K-Map Simplification for Three Variables
- ▶ K-Map Simplification for Four Variables
- ▶ Don't Care Conditions
- ▶ Redundancy
- ▶ Design of Combinational Circuits

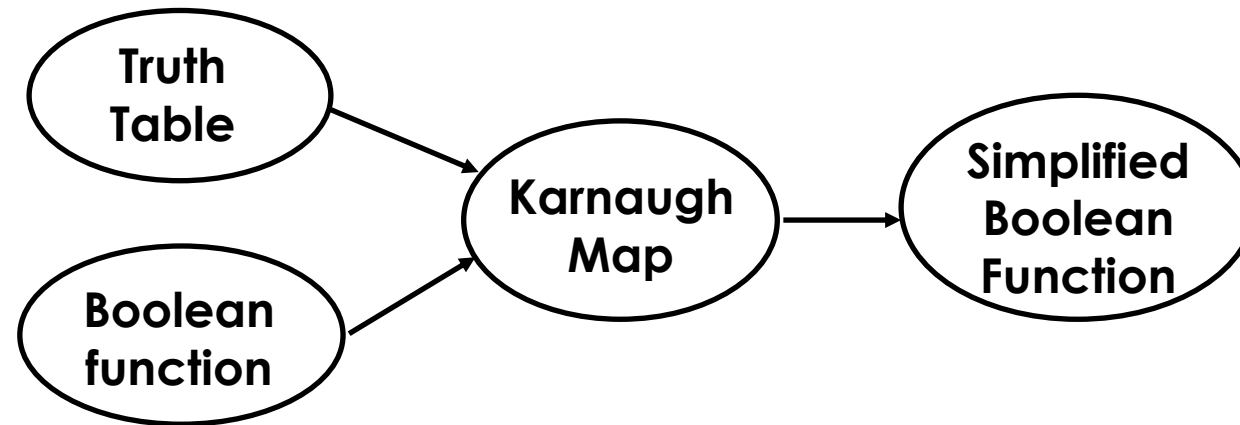
Introduction



Simplification from Boolean function

- Finding an equivalent expression that is least expensive to implement
- For a simple function, it is possible to obtain a simple expression for low cost implementation
- But, with complex functions, it is a very difficult for implementation

Karnaugh Map (K-map) is a simple procedure for simplification of Boolean expressions.



Karnaugh Map (K-Map)

- ▶ Karnaugh maps (K-maps) are *graphical* representations of Boolean functions.
- ▶ One *map cell* corresponds to a row in the truth table.
- ▶ Also, one map cell corresponds to a minterm or a maxterm in the Boolean expression
- ▶ Each term is identified by a decimal number whose binary representation is identical to the binary interpretation of the input values of the term.

	C'D'	C'D	CD	CD'
A'B'	0	1	3	2
A'B	4	5	7	6
AB	12	13	15	14
AB'	8	9	11	10

K-Map Simplification for Two Variables

- ▶ Of course, the Minterm function that we derived from our Truth Table was not in simplest terms.
 - ▶ That's what we started with in this example.
- ▶ We can, however, reduce our complicated expression to its simplest terms by finding adjacent 1s in the K-map that can be collected into groups that are powers of two.
 - In our example, we have two such groups.
 - Can you find them?

x \ y	0	1
0	0	1
1	1	1

K-Map Rules

The rules of K-map simplification are:

- Groupings can contain only 1s; no 0s.
- **The number of 1s in a group must be a power of 2 – even if it contains a single 1.**
- Nearby 1s are to be grouped.
- Corner 1s are to be grouped.
- Group that wraps around the sides of a K-map.
- Diagonal groups are not allowed.
- The groups must be made as large as possible.
- Groups can overlap.

		Y	
		0	1
X	0	0	1
	1	1	1

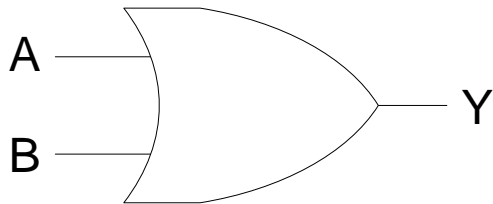
K-Map Rules

- ▶ The best way of selecting two groups of 1s from our simple K-map is shown.
- ▶ We see that both groups are powers of two and that the groups overlap.

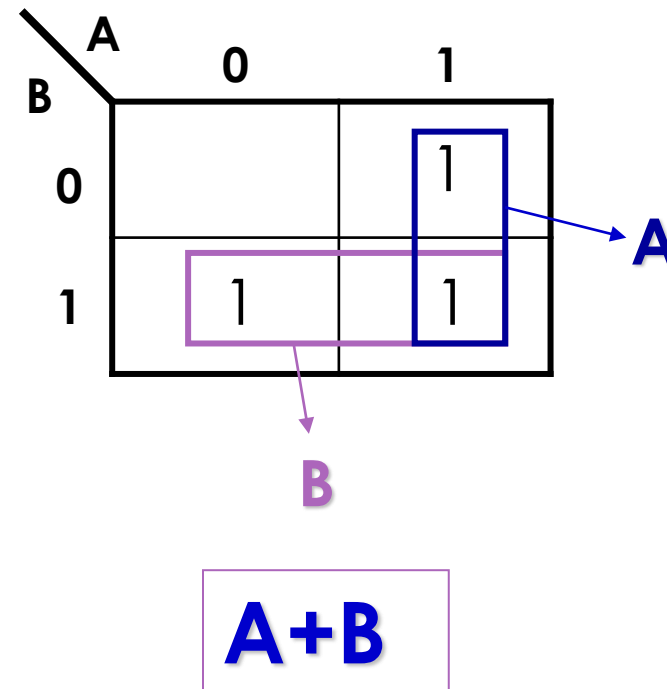
		Y	
		0	1
X	0	0	1
	1	1	1

K-Map Simplification for Two Variables

2-variable Karnaugh maps are trivial but can be used to introduce the methods you need to learn. The map for a 2-input OR gate looks like this:



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



K-Map Simplification for Three Variables

- ▶ A K-map for three variables is constructed as shown in the diagram below.
- ▶ We have placed each Minterm in the cell that will hold its value.
 - ▶ Notice that the values for the yz combination at the top of the matrix form a pattern that is not a normal binary sequence.

x	yz			
	00	01	11	10
0	$\bar{x}\bar{y}\bar{z}$	$\bar{x}\bar{y}z$	$\bar{x}yz$	$\bar{x}y\bar{z}$
1	$x\bar{y}\bar{z}$	$x\bar{y}z$	xyz	$xy\bar{z}$

- Consider the function:

$$F(X, Y, Z) = X'Y'Z + X'YZ + XY'Z + XYZ$$

- Its K-map is given below.
 - What is the largest group of 1s that is a power of 2?

x \ yz	yz			
	00	01	11	10
0	0	1	1	0
1	0	1	1	0

- ▶ This grouping tells us that changes in the variables x and y have no influence upon the value of the function: They are irrelevant.
- ▶ This means that the function, $F(X, Y, Z) = X'Y'Z + X'YZ + XY'Z + XYZ$ reduces to $F = Z$.

You could verify this reduction with Boolean Algebra

X \ YZ	YZ			
	00	01	11	10
0	0	1	1	0
1	0	1	1	0

- Now for a more complicated K-map. Consider the function:

$$F(X, Y, Z) = \bar{X}\bar{Y}\bar{Z} + \bar{X}\bar{Y}Z + \bar{X}YZ + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XY\bar{Z}$$

- Its K-map is shown below. There are (only) two groupings of 1s.
 - Can you find them?

X \ YZ	YZ			
	00	01	11	10
0	1	1	1	1
1	1	0	0	1

- In this K-map, we see an example of a group that wraps around the sides of a K-map.

x \ yz	yz			
	00	01	11	10
0	1	1	1	1
1	1	0	0	1

$$f = \sum(0,4) = \overline{B} \overline{C}$$

A \ BC				
	00	01	11	10
0	1	0	0	0
1	1	0	0	0

$$f = \sum(4,5) = A \overline{B}$$

A \ BC				
	00	01	11	10
0	0	0	0	0
1	1	1	0	0

$$f = \sum(0,1,4,5) = \overline{B}$$

A \ BC				
	00	01	11	10
0	1	1	0	0
1	1	1	0	0

$$f = \sum(0,1,2,3) = \overline{A}$$

A \ BC				
	00	01	11	10
0	1	1	1	1
1	0	0	0	0

$$f = \sum(1,3) = A' C$$

~~$f = \sum(0,4) = \overline{B} \overline{C}$~~

A \ BC				
	00	01	11	10
0	0	1	1	0
1	0	0	0	0

$$f = \sum(4,6) = A \overline{C}$$

A \ BC				
	00	01	11	10
0	0	0	0	0
1	1	0	0	1

$$f = \sum(0,2) = \overline{A} \overline{C}$$

A \ BC				
	00	01	11	10
0	1	0	0	1
1	0	0	0	0

$$f = \sum(0,2,4,6) = \overline{C}$$

A \ BC				
	00	01	11	10
0	1	0	0	1
1	1	0	0	1

K-Map Simplification for Four Variables

- ▶ The K-map can be extended to accommodate the 16 Minterms that are produced by a four-input function.
- ▶ This is the format for a 16-minterm K-map.

WX \ YZ	YZ			
	00	01	11	10
00	$\bar{W}\bar{X}\bar{Y}\bar{Z}$	$\bar{W}\bar{X}\bar{Y}Z$	$\bar{W}\bar{X}Y\bar{Z}$	$\bar{W}\bar{X}YZ$
01	$\bar{W}X\bar{Y}\bar{Z}$	$\bar{W}X\bar{Y}Z$	$\bar{W}XY\bar{Z}$	$\bar{W}XYZ$
11	$WX\bar{Y}\bar{Z}$	$WX\bar{Y}Z$	$WXY\bar{Z}$	$WXYZ$
10	$W\bar{X}\bar{Y}\bar{Z}$	$W\bar{X}\bar{Y}Z$	$W\bar{X}Y\bar{Z}$	$W\bar{X}YZ$

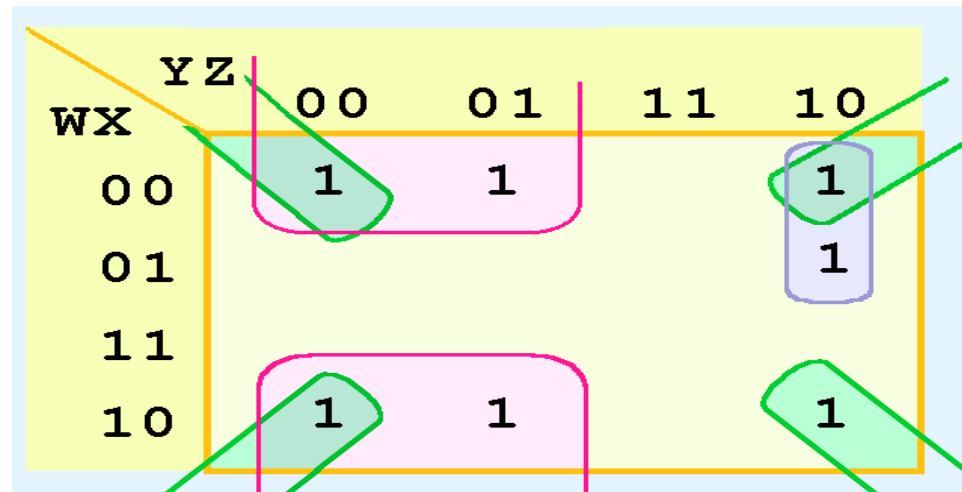
- We have populated the K-map shown below with the nonzero minterms from the function:

$$F(W, X, Y, Z) = \bar{W}\bar{X}\bar{Y}\bar{Z} + \bar{W}\bar{X}\bar{Y}Z + \bar{W}\bar{X}Y\bar{Z} \\ + \bar{W}XY\bar{Z} + W\bar{X}\bar{Y}\bar{Z} + W\bar{X}\bar{Y}Z + W\bar{X}Y\bar{Z}$$

- Can you identify (only) three groups in this K-map?

WX \ YZ	YZ			
	00	01	11	10
00	1	1		1
01				1
11				
10	1	1		1

- ▶ Our three groups consist of:
 - ▶ A purple group entirely within the K-map at the right.
 - ▶ A pink group that wraps the top and bottom.
 - ▶ A green group that spans the corners.
- ▶ Thus we have three terms in our final function:



- It is possible to have a choice as to how to pick groups within a K-map, while keeping the groups as large as possible.
- The (different) functions that result from the groupings below are logically equivalent.

		YZ			
		00	01	11	10
WX	00	1		1	
	01	1		1	1
	11	1			
	10	1			

		YZ			
		00	01	11	10
WX	00	1		1	
	01	1		1	1
	11	1			
	10	1			

AB \ CD	CD			
	00	01	11	10
00	1	0	0	0
01	0	0	0	0
11	0	0	0	0
10	1	0	0	0

$$f = \sum(0,8) = \bar{B} \cdot \bar{C} \cdot \bar{D}$$

AB \ CD	CD			
	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	0	1	0	0
10	0	0	0	0

$$f = \sum(5,13) = B \cdot \bar{C} \cdot D$$

AB \ CD	CD			
	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	1	1	0
10	0	0	0	0

$$f = \sum(13,15) = A \cdot B \cdot D$$

AB \ CD	CD			
	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	0	0	0	0
10	0	0	0	0

$$f = \sum(4,6) = \bar{A} \cdot B \cdot \bar{D}$$

AB \ CD	CD			
	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	0	0

$$f = \sum(2,3,6,7) = \bar{A} \cdot C$$

AB \ CD	CD			
	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	1	0	0	1
10	0	0	0	0

$$f = \sum(4,6,12,14) = B \cdot \bar{D}$$

AB \ CD	CD			
	00	01	11	10
00	0	0	1	1
01	0	0	0	0
11	0	0	0	0
10	0	0	1	1

$$f = \sum(2,3,10,11) = \bar{B} \cdot C$$

AB \ CD	CD			
	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

$$f = \sum(0,2,8,10) = \bar{B} \cdot \bar{D}$$

AB \ CD	CD			
	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	0	0	0	0

$$f = \sum (4, 5, 6, 7) = \bar{A} \bullet B$$

AB \ CD	CD			
	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	0	1	0
10	0	0	1	0

$$f = \sum (3, 7, 11, 15) = C \bullet D$$

AB \ CD	CD			
	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

$$f = \sum (0, 3, 5, 6, 9, 10, 12, 15)$$

$$f = A \otimes B \otimes C \otimes D$$

AB \ CD	CD			
	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

$$f = \sum (1, 2, 4, 7, 8, 11, 13, 14)$$

$$f = A \oplus B \oplus C \oplus D$$

AB \ CD	CD			
	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

$$f = \sum (1, 3, 5, 7, 9, 11, 13, 15)$$

$$f = D$$

AB \ CD	CD			
	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	1	0	0	1
10	1	0	0	1

$$f = \sum (0, 2, 4, 6, 8, 10, 12, 14)$$

$$f = \bar{D}$$

AB \ CD	CD			
	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	0	0	0	0

$$f = \sum (4, 5, 6, 7, 12, 13, 14, 15)$$

$$f = B$$

AB \ CD	CD			
	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	0	0	0
10	1	1	1	1

$$f = \sum (0, 1, 2, 3, 8, 9, 10, 11)$$

$$f = \bar{B}$$

Don't Care Conditions

- ▶ Real circuits don't always need to have an output defined for every possible input.
 - ▶ For example, some calculator displays consist of 7-segment LEDs. These LEDs can display 2^7 patterns but all patterns are not used.
- ▶ If a circuit is designed so that a particular set of inputs can never happen, we call this set of inputs a *don't care* condition.
- ▶ They are very helpful to us in K-map circuit simplification.

- ▶ In a K-map, a don't care condition is identified by an X in the cell of the minterm(s) for the don't care inputs, as shown below.
- ▶ In performing the simplification, we are free to include or ignore the X 's when creating our groups.

WX \ YZ	YZ			
	00	01	11	10
00	X	1	1	X
01		X	1	
11	X		1	
10			1	

- ▶ In one grouping in the K-map below, we have the function:
- ▶ $F = W'X' + YZ$

		YZ			
		00	01	11	10
WX	00	X	1	1	X
	01		X	1	
	11	X		1	
	10			1	

- A different grouping gives us the function:

$$F(W, X, Y, Z) = \bar{W}Z + YZ$$

WX \ YZ	YZ			
	00	01	11	10
00	×	1	1	×
01		×	1	
11	×		1	
10			1	

- The truth table of:

$$F(W, X, Y, Z) = W'X' + YZ$$

differs from the truth table of:

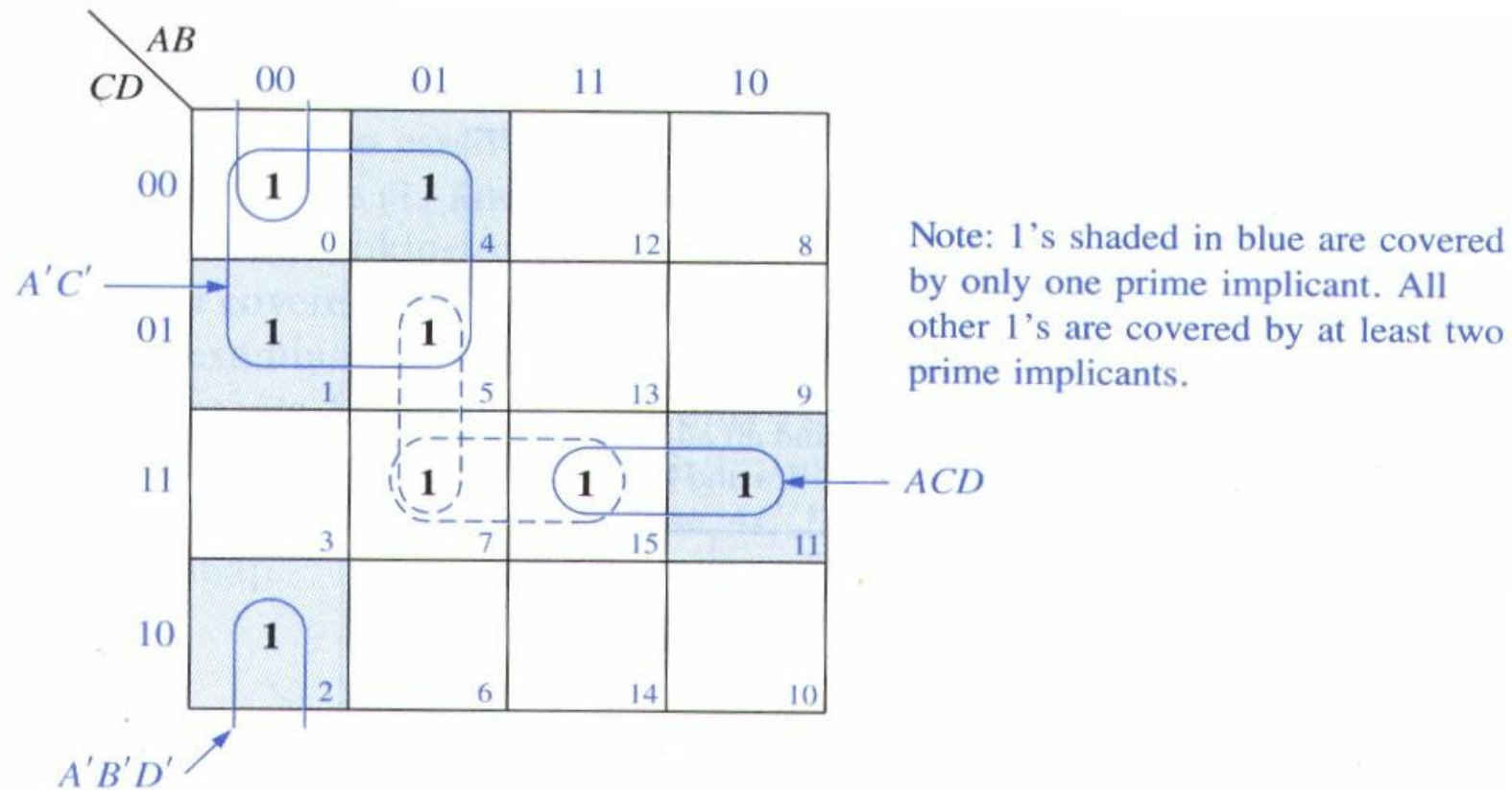
$$\mathbf{F(W, X, Y, Z) = \bar{W}Z + YZ}$$

- However, the values for which they differ, are the inputs for which we have don't care conditions.

WX \ YZ	YZ			
	00	01	11	10
00	×	1	1	×
01		×	1	
11	×		1	
10			1	

WX \ YZ	YZ			
	00	01	11	10
00	×	1	1	×
01		×	1	
11	×		1	
10			1	

100




Design of combinational digital circuits


- ▶ Steps to design a combinational digital circuit:
 - ▶ From the problem statement derive the truth table
 - ▶ From the truth table derive the un-simplified logic expression
 - ▶ Simplify the logic expression
 - ▶ From the simplified expression draw the logic circuit
- ▶ Example: Design a 3-input (A,B,C) digital circuit that will give at its output (X) a logic 1 only if the binary number formed at the input has more ones than zeros.

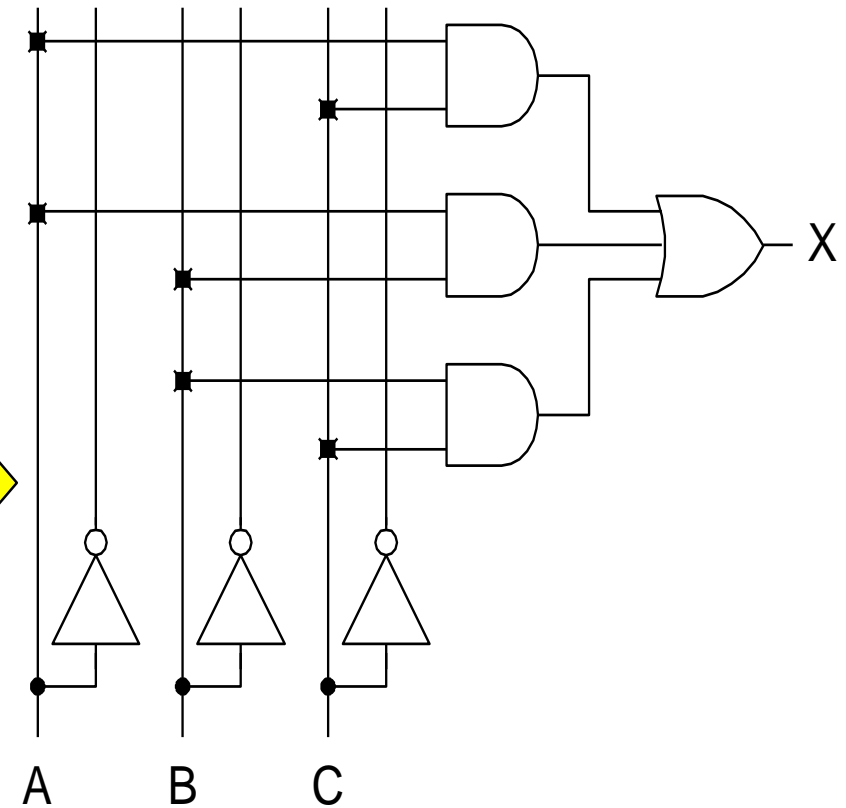
	Inputs			Output
	A	B	C	X
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1


 $X = \sum (3, 5, 6, 7)$



A \ BC				
	00	01	11	10
0	0	0	1	0
1	0	1	1	1


 $X = AC + AB + BC$



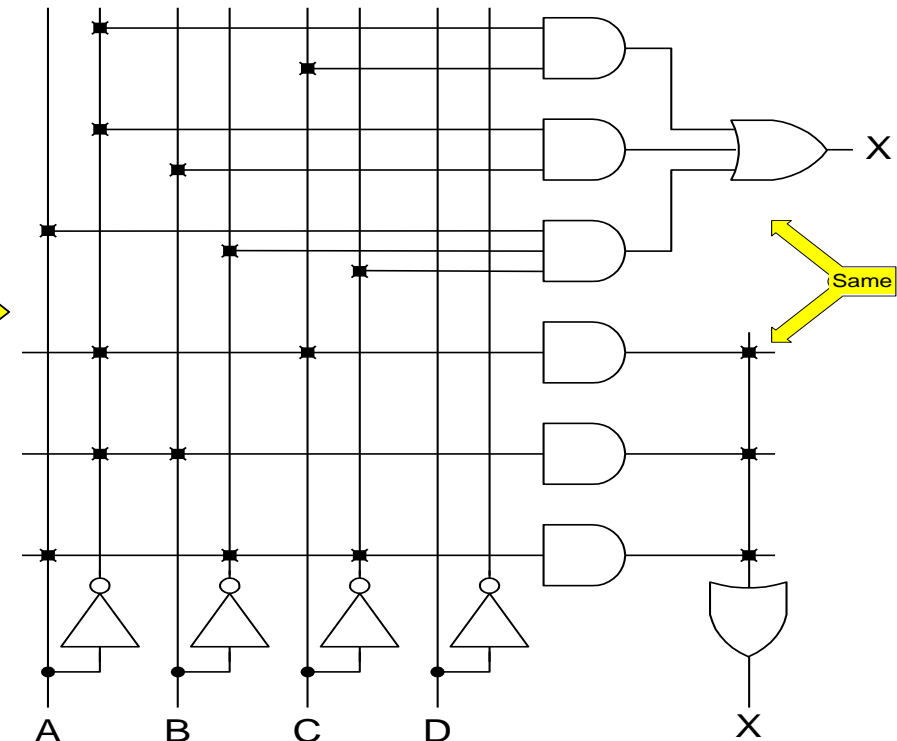
- Example: Design a 4-input (A,B,C,D) digital circuit that will give at its output (X) a logic 1 only if the binary number formed at the input is between 2 and 9 (including).

	Inputs				Output
	A	B	C	D	
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

→ $X = \sum (2,3,4,5,6,7,8,9)$

AB \ CD				
	00	01	11	10
00	0	0	1	1
01	1	1	1	1
11	0	0	0	0
10	1	1	0	0

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



Conclusion

- ▶ K-maps provide an easy graphical method of simplifying Boolean expressions.
- ▶ A K-map is a matrix consisting of the outputs of the minterms of a Boolean function.
- ▶ In this section, we have discussed 2- 3- and 4-input K-maps. This method can be extended to any number of inputs through the use of multiple tables.



Recapping the rules of K-map simplification:

- Groupings can contain only 1s; no 0s.
- Groups can be formed only at right angles; diagonal groups are not allowed.
- The number of 1s in a group must be a power of 2 – even if it contains a single 1.
- The groups must be made as large as possible.
- Groups can overlap and wrap around the sides of the K-map.
- Use don't care conditions when you can.
- Redundancy must be reduced