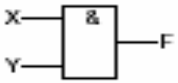
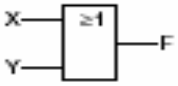
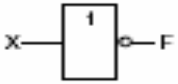
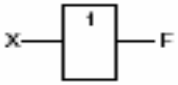
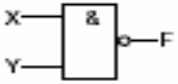
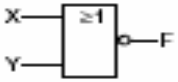
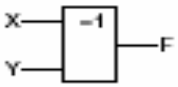
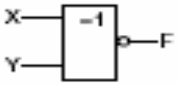
Logic Circuits and Logic Functions

Instructor: Saraju P. Mohanty

- Variables and Functions
- Logic Gates and Networks
- Boolean Algebra
- Synthesis using, AND, OR, NOT, NAND, NOR
- Karnaugh Map
- Strategy for Minimization
- Multiple-Output Circuits
- Analysis of Circuits

Note: The slides are from text or reference book authors or publishers.

IEEE Standard Graphics Symbols

Name	Commonly used	IEEE	Algebraic equation	Truth table
AND			$F = XY$	X Y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR			$F = X + Y$	X Y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT (inverter)			$F = \overline{X}$	X | F 0 | 1 1 | 0
Buffer			$F = X$	X | F 0 | 0 1 | 1
NAND			$F = \overline{X \cdot Y}$	X Y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR			$F = \overline{X + Y}$	X Y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive–OR (XOR)			$F = X\overline{Y} + \overline{X}Y$ $= X \oplus Y$	X Y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive–NOR (XNOR)			$F = XY + \overline{X}\overline{Y}$ $= \overline{X \oplus Y}$	X Y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Digital Logic Gates

Variables, Functions, Gates and Networks

- A logic expression describes the output as a logic function of input variables.

  Example: z = f(x1,x2) = x1 + x2

- NOT, complement, or inversion represent the same thing.

  Example: z' = f'(x1,x2) = (x1+x2)'

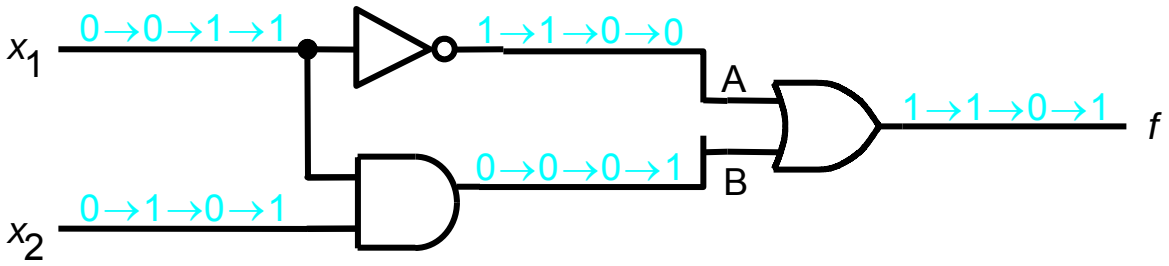
- Logic operations are implemented using logic gates. Each gate has more than one input (s).

- Logic circuits are implemented by a network of logic gates.

Analysis Vs Synthesis of Logic Networks

- Two basic issues of digital system design:
  - Analysis
  - Synthesis
- Analysis: Given an existing network determine its behavior, the function it performs.
- Synthesis: Design a network that implements a desired functional behavior.
- Analysis process is more straightforward and simple than synthesis process.

Analysis of Logic Networks : Example

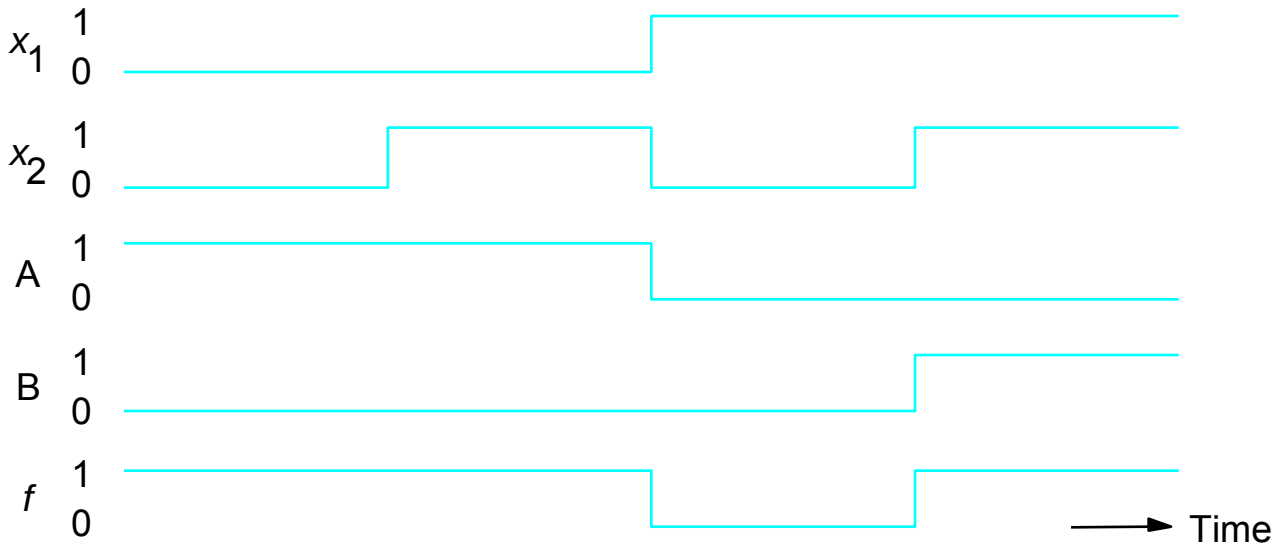


(a) Network that implements $f = \bar{x}_1 + x_1 \cdot x_2$

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	1
1	0	0
1	1	1

A	B
1	0
1	0
0	0
0	1

(b) Truth table



(c) Timing diagram

Boolean Algebra

- To describe the operational properties of digital circuits mathematical notation specifying each gate operation is introduced, which is a class of mathematical system called “Boolean Algebra”.
- For simplicity, the algebra dealing with binary variables and logic properties is Boolean Algebra.
- A “Boolean Function” has binary variables, an equal sign and algebraic expression.
- Boolean Function can be represented as a truth table as well.
- For *n* variable function the number of rows of the truth table will be 2^n.

Boolean Algebra : Boolean Function ……..

- For a given value of binary variables the Boolean function can be equal to either “0” or “1”.
- Example : F = X + Y’Z

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

Truth Table for the Function $F = X + \overline{Y}Z$

- The two parts of the algebraic expression, such as “X” and “Y’Z” are called terms of the function.

Combinational Logic Circuit ??

- In logic circuits the variables are given as input and the functional value is taken as output.
- The gates are interconnected by wires that carry logic signals.
- Thus, the variables are combined by the logic operations, which gives rise to the name “combinational logic”.
- NOTE: Sequential variables are stored over time and as well as combined.

Boolean Algebra : Some Identities

- Nine identities involve single variable.
- Seven identities involve two or more than two variables.
- The duals of an algebraic expression is obtained by interchanging the OR and AND operations, and replacing "1"s by "0"s and "0"s by "1"s.

1.	$X+0=X$	2.	$X\cdot 1=X$	
3.	$X+1=1$	4.	$X\cdot 0=0$	
5.	$X+X=X$	6.	$X\cdot X=X$	
7.	$X+\overline{X}=1$	8.	$X\cdot\overline{X}=0$	
9.	$\overline{\overline{X}}=X$			
10.	$X+Y=Y+X$	11.	$XY=YX$	Commutative
12.	$X+(Y+Z)=(X+Y)+Z$	13.	$X(YZ)=(XY)Z$	Associative
14.	$X(Y+Z)=XY+XZ$	15.	$X+YZ=(X+Y)(X+Z)$	Distributive
16.	$\overline{X+Y}=\overline{X}\cdot\overline{Y}$	17.	$\overline{X\cdot Y}=\overline{X}+\overline{Y}$	DeMorgan's

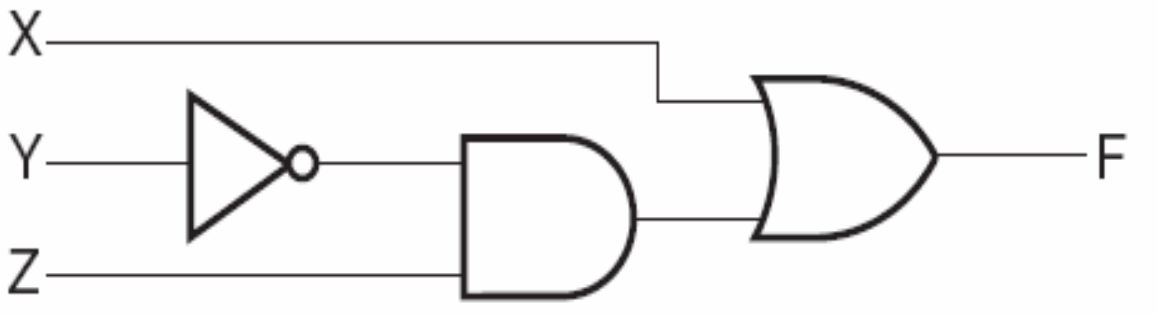
Basic Identities of Boolean Algebra

Left and right column expressions are duals of each other, for example 10 and 11 are duals each other.

Boolean Algebra: Algebraic Manipulations

•When Boolean expressions is implemented with logic gates, each term requires a gate, and each variable within the term becomes an input to the gate.

•Example: Lets consider the implementation of X+Y'Z

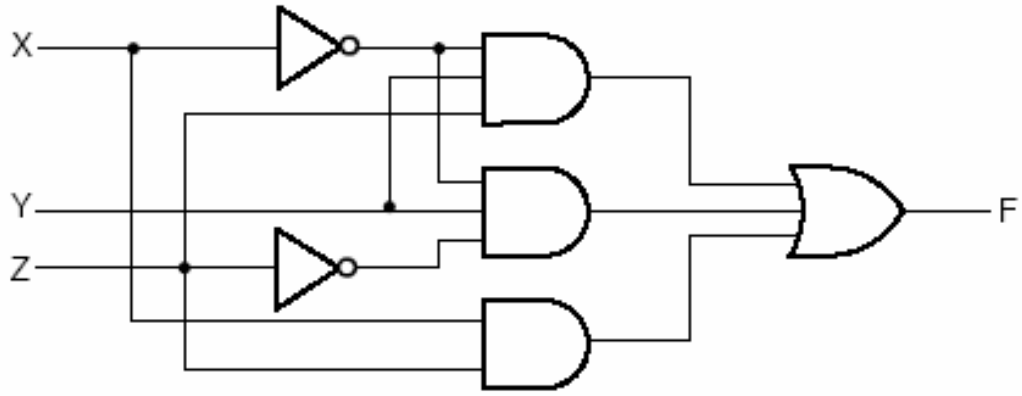


Logic Circuit Diagram for $F = X + \overline{Y}Z$

•Algebraic manipulations are useful for simplifying digital circuits.

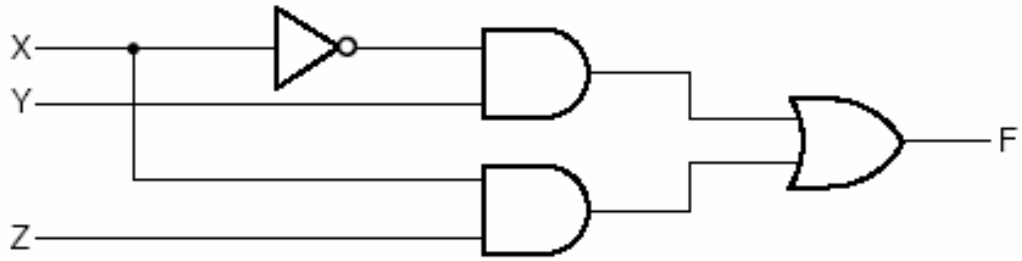
Boolean Algebra: Terms and Literals

Literal is a single variable within a term that may or may not be complemented.



(a) $F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$

3 terms and 8 literals



(b) $F = \overline{X}Y + XZ$

2 terms and 4 literals

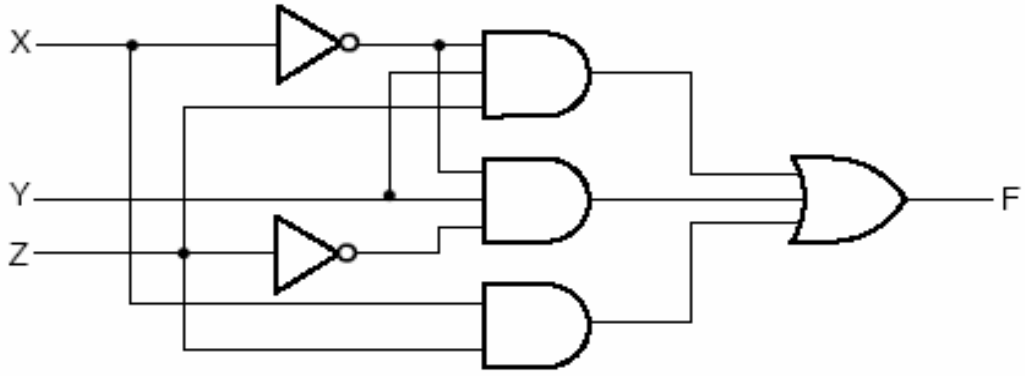
Implementation of Boolean Function with Gates

Boolean Algebra: Simplifications

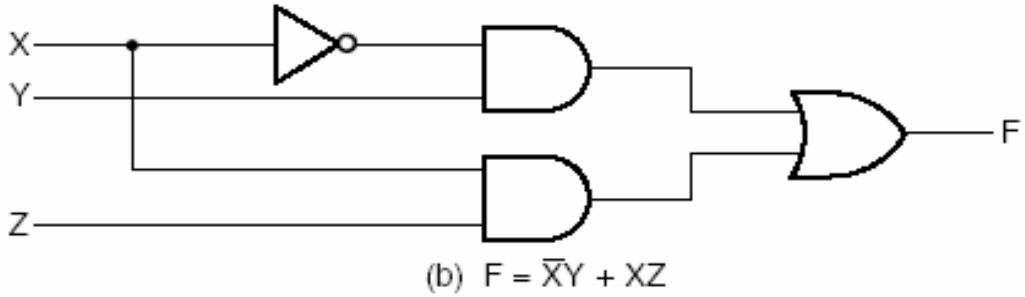
Both (a) and (b) give the same result F:

$$F = X'YZ+X'YZ'+XZ = X'Y(Z+Z')+XZ = X'Y+XZ$$

But, (b) needs fewer number of gates, so is area efficient.



(a) $F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$



(b) $F = \overline{X}Y + XZ$

Implementation of Boolean Function with Gates

Boolean Algebra: Complement of a Function

•Complement of a function is obtained by interchanging "0"s to "1"s and "1"s to "0"s.

•It can be obtained algebraically by applying DeMorgan's theorem.

•The general DeMorgan's theorem: Complement of a function can be obtained by interchanging AND and OR operations and complementing each variable and constant.

Boolean Algebra : DeMorgan's Theorem

A) X	Y	X + Y	$\overline{X+Y}$	B) X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X} \cdot \overline{Y}$
0	0	0	1	0	0	1	1	1
0	1	1	0	0	1	1	0	0
1	0	1	0	1	0	0	1	0
1	1	1	0	1	1	0	0	0

Truth Tables to Verify DeMorgan's Theorem

DeMorgan's Theorem for *n* variables

- $(X_1+X_2+X_3+\ldots\ldots+X_n)' = X_1'X_2'\ldots..X_n'$
- $(X_1X_2X_3\ldots\ldots X_n)' = X_1'+X_2'+\ldots..+X_n'$

Boolean Algebra: Complementing Example

Find complement of the function, F = X'YZ' + X'Y'Z

Method 1: (Using DeMorgan's theorem)

F' = (X'YZ' + X'Y'Z)' = (X'YZ')' (X'Y'Z)'

= (X +Y'+Z) (X+Y+Z')

Method 2: (Using Duality)

We have, F = X'YZ' + X'Y'Z = (X'YZ') + (X'Y'Z) (use parentheses to avoid confusion)

Dual of F = (X'+Y+Z') (X'+Y'+Z)

Complementing each literal : (X+Y'+Z) (X+Y+Z') = F'

Standard Forms of Boolean Functions

- Standard forms help in simplifying the Boolean expressions and frequently results in more desirable logic circuits.
- Standard forms contain product terms and sum terms.
  - product terms (example X'YZ)
  - sum terms (example X+Y+Z')
- In Boolean algebra, product means logical AND operation and sum means logical OR operation.
- Two standard forms Standard forms of Boolean functions are:
  - Sum-of-Products (SOP)
  - Product-of-Sums (POS)
- Product terms are called minterms and sum terms are called maxterms.

Standard Forms: Minterms

- A product term in which all the variables appear exactly once, either complemented or uncomplemented is called a minterm.
- Property: It represents exactly one combination of the binary variables in a truth table. It has the value *1* for that combination and *0* for all others.
- For *n* variables there are 2^n distinct minterms.
- Example: Given two variables X, Y, the four minterms are X'Y', X'Y, XY' and XY.
- The symbol for a minterm is m_j ,where *j* denotes the decimal equivalent for which the minterm has the value of 1.

Standard Forms: Minterms (for 3 variables)

X	Y	Z	Product Term	Symbol	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
0	0	0	$\overline{X}\,\overline{Y}\,\overline{Z}$	m_0	1	0	0	0	0	0	0	0
0	0	1	$\overline{X}\,\overline{Y}Z$	m_1	0	1	0	0	0	0	0	0
0	1	0	$\overline{X}Y\overline{Z}$	m_2	0	0	1	0	0	0	0	0
0	1	1	$\overline{X}YZ$	m_3	0	0	0	1	0	0	0	0
1	0	0	$X\overline{Y}\,\overline{Z}$	m_4	0	0	0	0	1	0	0	0
1	0	1	$X\overline{Y}Z$	m_5	0	0	0	0	0	1	0	0
1	1	0	$XY\overline{Z}$	m_6	0	0	0	0	0	0	1	0
1	1	1	XYZ	m_7	0	0	0	0	0	0	0	1

Minterms for Three Variables

The binary numbers from 000 to 111 are listed under variables. For each binary combination there is a related minterm. Each term is the product of exactly 3 literals. A literal is a complemented variable if the corresponding bit of the related binary combination is 0 and it is uncomplemented variable if it is 1. The table is shown above can be extended for any *n* variables.

Standard Forms: Maxterms

- A sum term that contains all the variables in complemented or uncomplemented form is called a maxterm.
- For *n* variables there are 2^n distinct maxterms.
- Each maxterm is the logical sum of the variables, with each variable being complemented if the corresponding bit of the binary number is 1 and uncomplemented if it is 0.
- The symbol for a maxterm is M_j where j denotes the decimal equivalent of the binary combination for which the maxterm has the value 0.
- Example: Given two variables X, Y, the four maxterms are (X+Y), (X+Y'), (X'+Y), and (X'+Y').

Standard Forms: Maxterms (for 3 variables)

X	Y	Z	Sum Term	Symbol	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
0	0	0	$X+Y+Z$	M_0	0	1	1	1	1	1	1	1
0	0	1	$X+Y+\overline{Z}$	M_1	1	0	1	1	1	1	1	1
0	1	0	$X+\overline{Y}+Z$	M_2	1	1	0	1	1	1	1	1
0	1	1	$X+\overline{Y}+\overline{Z}$	M_3	1	1	1	0	1	1	1	1
1	0	0	$\overline{X}+Y+Z$	M_4	1	1	1	1	0	1	1	1
1	0	1	$\overline{X}+Y+\overline{Z}$	M_5	1	1	1	1	1	0	1	1
1	1	0	$\overline{X}+\overline{Y}+Z$	M_6	1	1	1	1	1	1	0	1
1	1	1	$\overline{X}+\overline{Y}+\overline{Z}$	M_7	1	1	1	1	1	1	1	0

Maxterms for Three Variables

The binary numbers from 000 to 111 are listed under variables. For each binary combination there is a related maxterm. Each term is the sum of exactly 3 literals. A literal is a complemented variable if the corresponding bit of the related binary combination is 1 and it is uncomplemented variable if it is 0. The table is shown above can be extended for any *n* variables.

Note: $M_j = m_j'$

Standard Forms: Sum-of-Minterms

A function of three variables

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

A Boolean function can be expressed algebraically by logically summing all the minterms that produce a "1" in the function. This expression is called a "sum of minterms". The function shown in the left side, can be expressed as follows.

$$F = m_0 + m_2 + m_5 + m_7$$

$$= \Sigma\, m(0,2,5,7)$$

The symbol Σ denotes logical sum (OR).

Standard Forms: Product-of-Maxterms

A function of three variables.

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

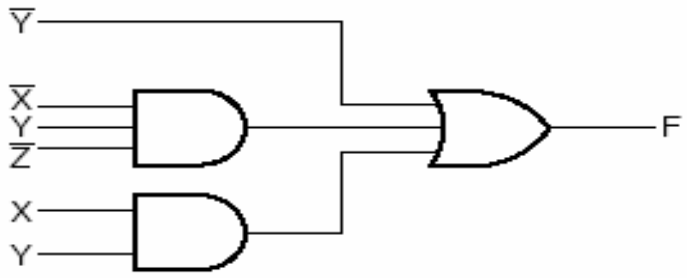
A Boolean function can be expressed algebraically by finding logical product of all the maxterms that produce a "0" in the function. This expression is called a "product of maxterms". The function shown in the left side, can be expressed as follows.

$$F = M_1 \cdot M_3 \cdot M_4 \cdot M_6$$

$$= \Pi\, M(1,3,4,6)$$

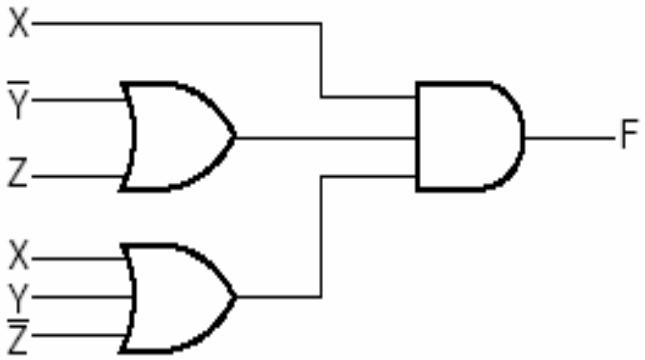
The symbol Π denotes logical product (AND).

Standard Forms: SOP, POS



Sum-of-Products Implementation

Sum-of-Products (SOP) is the simplified form of the Sum-of-Minterms. An SOP expression is equal to 1 only if one or more of the product terms in the expression is equal to one.



Product-of-Sums Implementation

Product-of-Sums (POS) is the simplified form of Product-of-Maxterms. A POS expression is equal to 0 only if one or more of the sum terms in the expression is equal to zero.

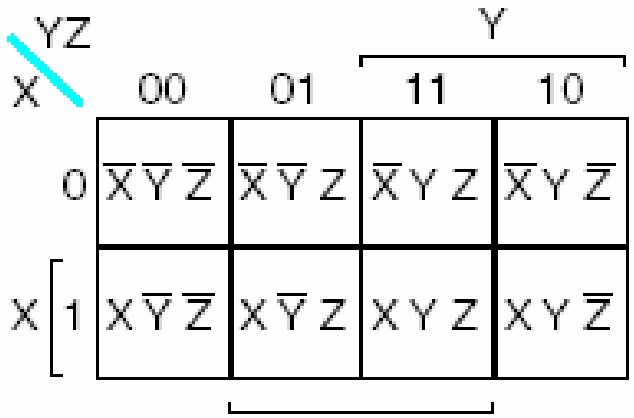
Map Simplification

- We need to express a function in simplified form to reduce the number of gates needed for its implementation, thus reducing the overall area of the chip to be fabricated.
- A function has an unique truth table representation, but it can have various algebraic representations.
- Boolean expressions can be simplified by algebraic manipulations, but this is a complicated procedure.
- The map procedure is much simpler.
- The map is known as Karnaugh map or K-map.
- The K-map is a diagram made up of squares, with each square representing one minterm (or a maxterm) of the function.

Map Simplification : Three Variable Map

There are eight minterms for a Boolean function with three variables. Hence, the K-map consists of eight squares, one for each minterm.

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6



Example: $F(X,Y,Z) = \Sigma\, m(2,3,4,5)$

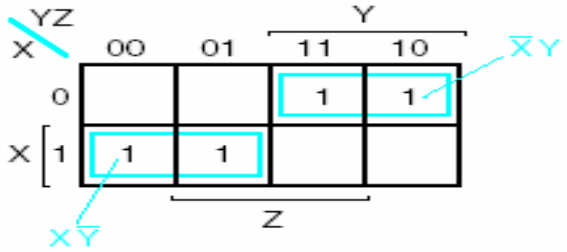


Fig. 2-11 Map for Example 2-3

After simplification we obtain,
$F(X,Y,Z) = X'Y + XY'$

Map Manipulations

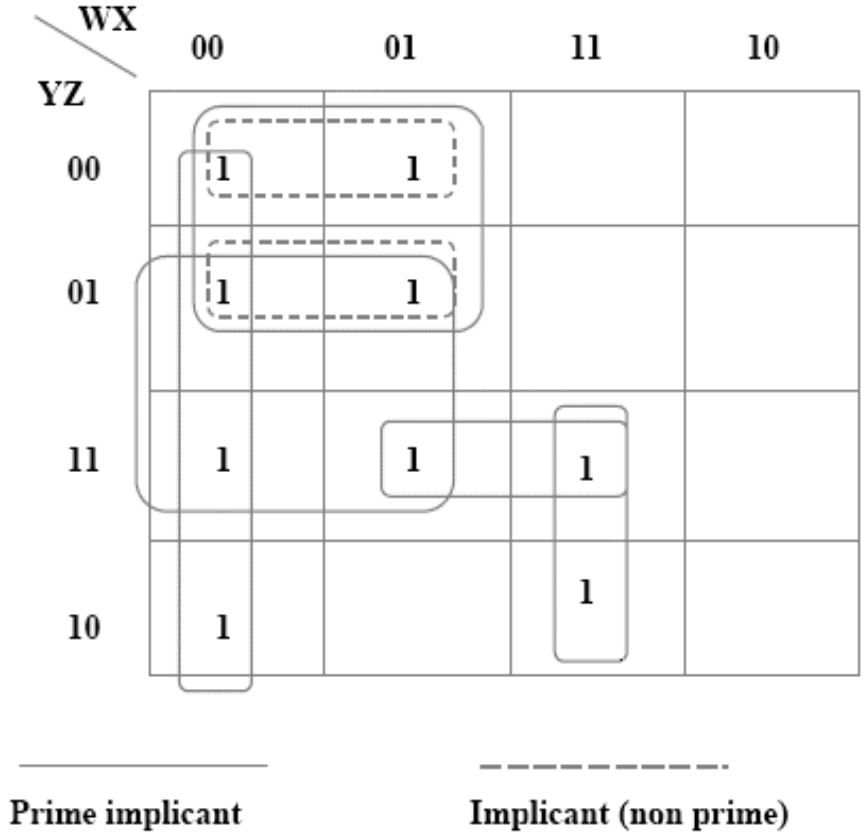
- While combining the squares in K-maps it is necessary to ensure that all the minterms are included.
- To minimize the number of terms in the simplified function any redundant term has to be avoided. So, the procedure of merging the squares need to be systematic.
- Three terms introduced to systematize the combining procedure are listed below.
  - Implicant
  - Prime Implicant
  - Essential Prime Implicant

Map Manipulations: Terminology

- An *implicant* is a product term (product of one or more literals) that could be used to cover minterms of the function. All the rectangles on a map made up of squares containing 1's correspond to implicants.
- A *prime implicant* is an implicant that is not a part of any other implicant of the function. Equivalently, prime implicant is a set of squares that is not a subset of any set containing larger number of squares.
- An *essential prime implicant* is a prime implicant that covers at least one minterm that is not covered by any other prime implicant.
- NOTE: Some of the implicants are prime implicants and some of the prime implicants are essential prime implicants.

Map Manipulations : Terminology Example

$F = \Sigma_{W,X,,Y,Z}\ (0,1,2,3,4,5,7,14,15)$



Prime implicants:
- W'X'
- W'Y'
- W'Z
- XYZ
- WXY

Essential prime implicants:
- W'X'
- W'Y'
- WXY

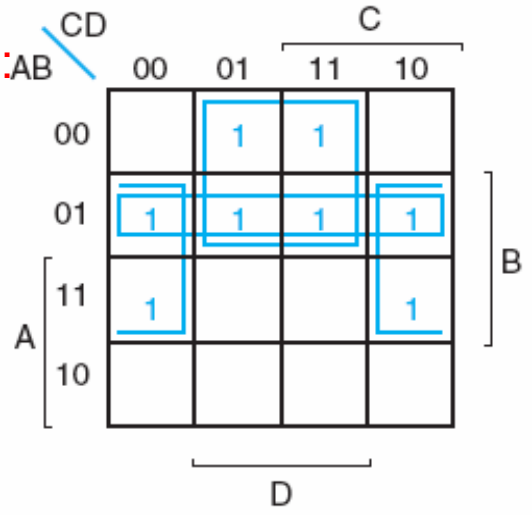
Note:
Implicants W'Y'Z' , W'Y'Z
- are not prime
- are contained in larger block W'Y'

NOTE: Implicants are, all single minterms + all groups of two / four / sixteen minterms.

Map Manipulations: Simplification using Prime Implicants

Rule: Determine all prime implicants. The simplified function is the logical sum of all the essential prime implicants and other prime implicants needed to include the remaining minterms not included in the essential prime implicant.

Example:



Prime Implicants for Example $\bar{A}D$, $B\bar{D}$, and $\bar{A}B$

The terms A'D and BD' are essential prime implicants since minterms 1 and 3 are covered by A'D only, and minterms 12 and 14 are covered by BD' only. As the minterms 4,5,6 and 7 are already included in A'D and BD', the remaining prime implicant A'B is not an essential prime implicant. The simplified expression is **F = A'D + BD'**.

NOTE: In this example, we do not need A'B since essential prime implicants cover all the minterms.

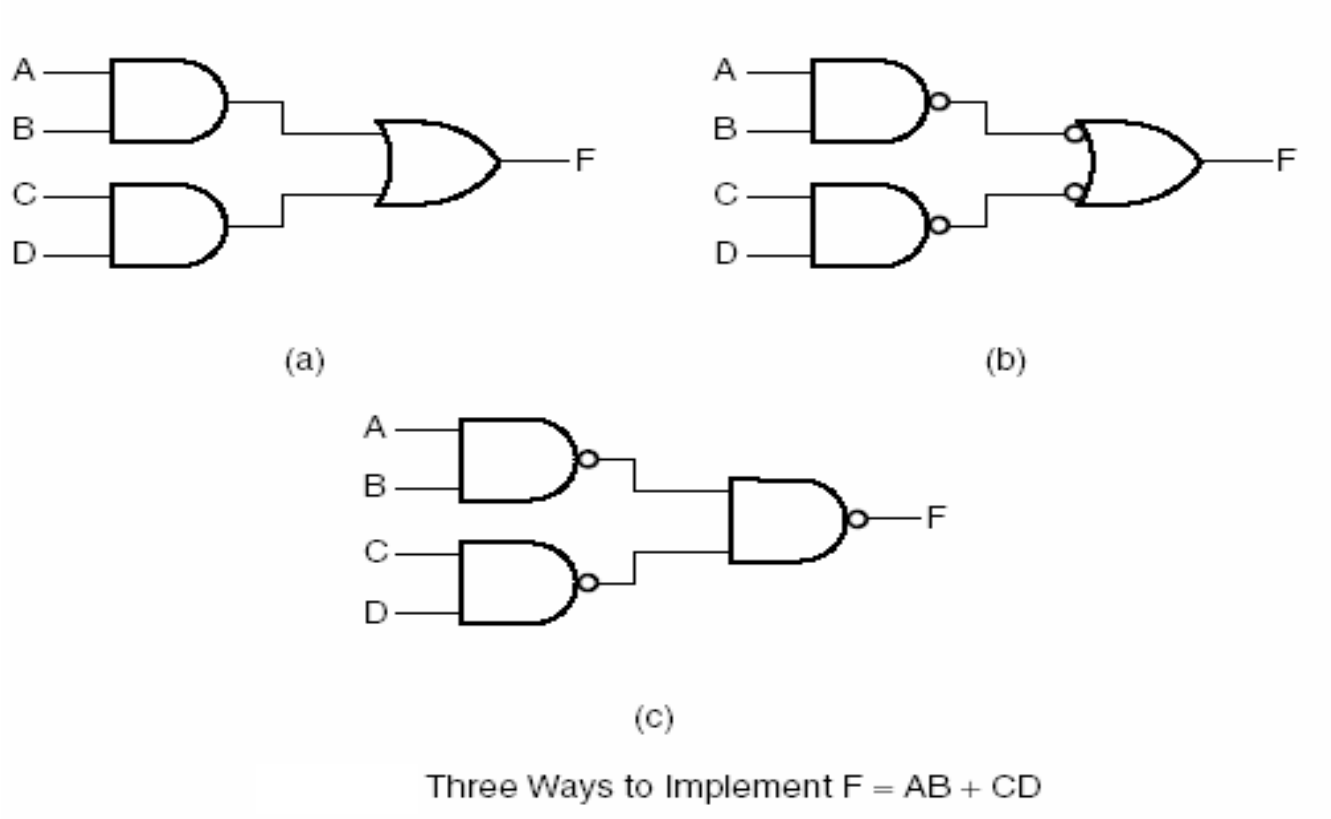
What Gates to be used in design ??

- OR, AND, NOT are basic gates and can implement all logic functions. NAND and NOR are universal gates and can also implement all types of logic functions.
- What is a good choice ??

  The deciding factors are :
  - Cost
  - Possibility of extending the gate to more than two inputs
  - ..and so on.
- NAND and NOR are more popular choices.

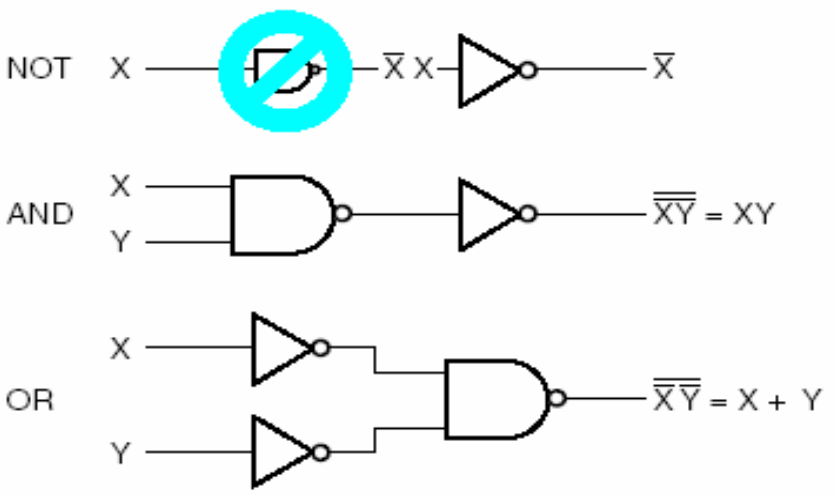
Two-Level Circuits Implementation

Sum-of-products form : AND gate is first level and OR gate is second level. SOP is more convenient for NAND implementations.

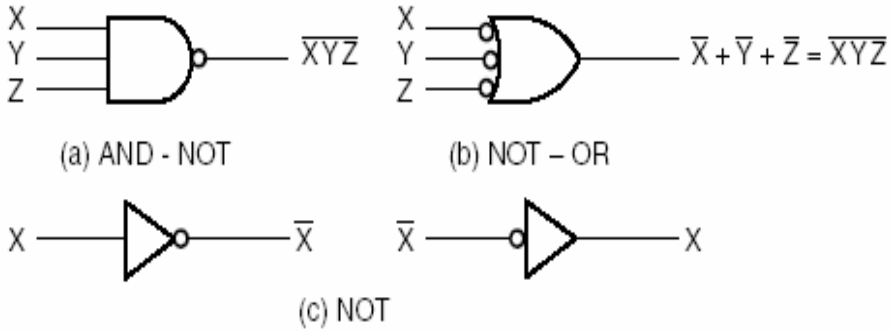


Three Ways to Implement F = AB + CD

NAND Circuits → AND, OR, NOT



Logical Operations with NAND Gates



Alternative Graphics Symbols for NAND and NOT Gates

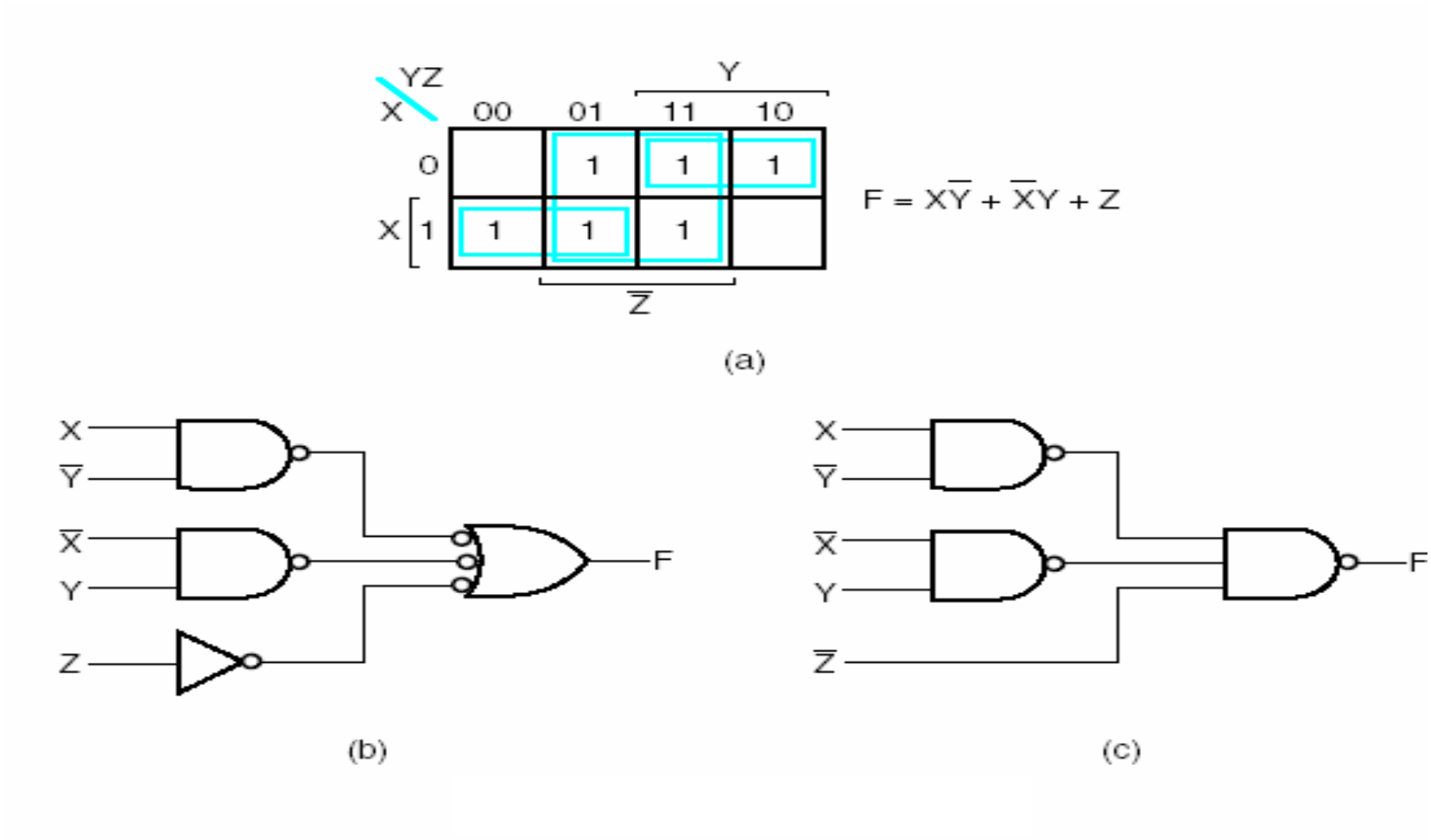
NOTE: This will be helpful for logic conversion.

Two-Level NAND Implementation: General Procedure

- Simplify the function and express it in SOP form.
- Draw a NAND gate for each product term of the expression that has at least two literals. These literals become the inputs for the NAND gates. This group of gates is the 1st level.
- Draw a single gate using AND-NOT or the NOT-OR configuration at the 2nd level with inputs coming from the output of the 1st-level.
- A product term with single literal requires a NOT at the first level. If the single literal is a complement then it can be directly connected as the input of the 2nd-level NAND.

Two-Level NAND Implementation: Example

Example : Implement $F(X,Y,Z) = \Sigma m(1,2,3,4,5,7)$ using NAND gates.

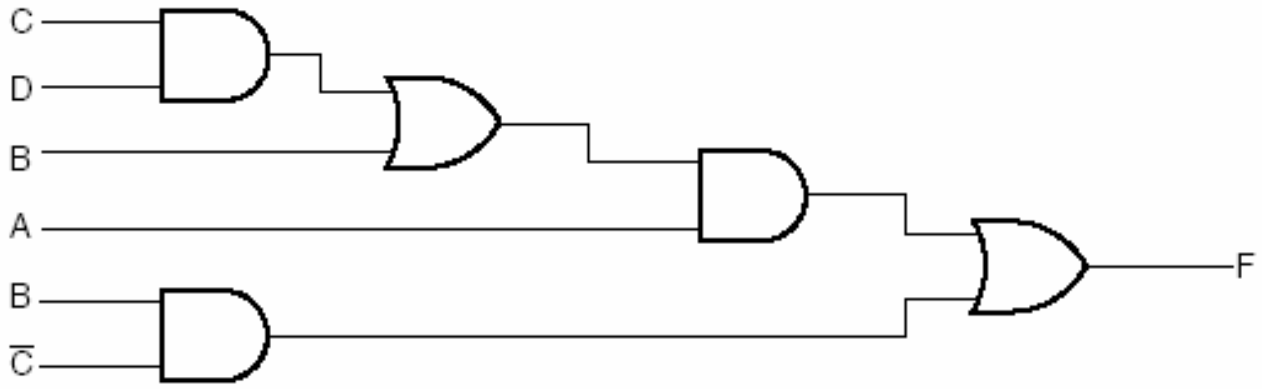


Multilevel NAND Implementation: General Procedure

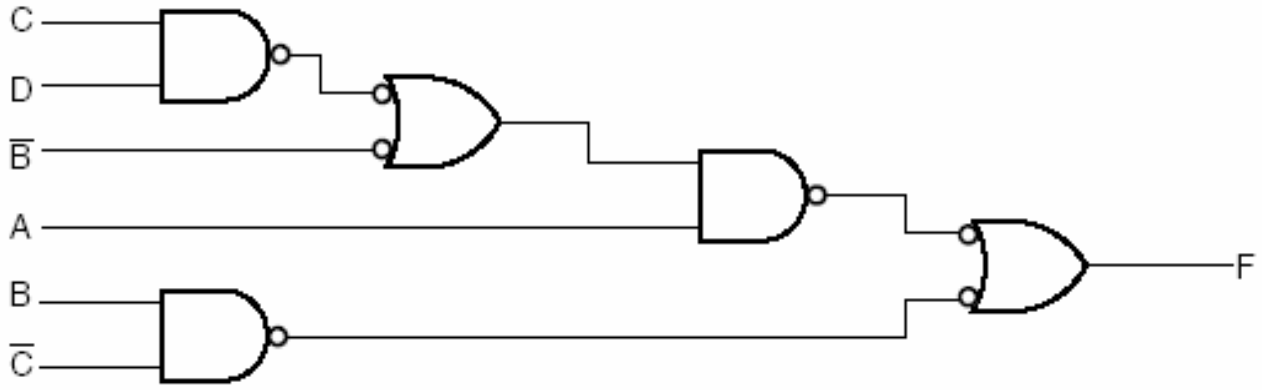
- Simplify the function and express it in sum-of-products.
- Implement the function using AND, OR, and NOT gates.
- Use the following steps to convert from AND-OR implementation to NAND-NAND implementation.
- Convert all AND gates to NAND gates with AND-NOT graphic symbols.
- Convert all OR gates to NAND gates with NOT-OR graphic symbols.
- Check all the bubbles in the diagram. For every bubble that is not counteracted by another bubble along the same line, insert a NOT gate or complement the input literal from its original appearance.

NOTE: We can develop similar procedure for NOR.

Multilevel NAND Implementation: Example



(a) AND – OR gates

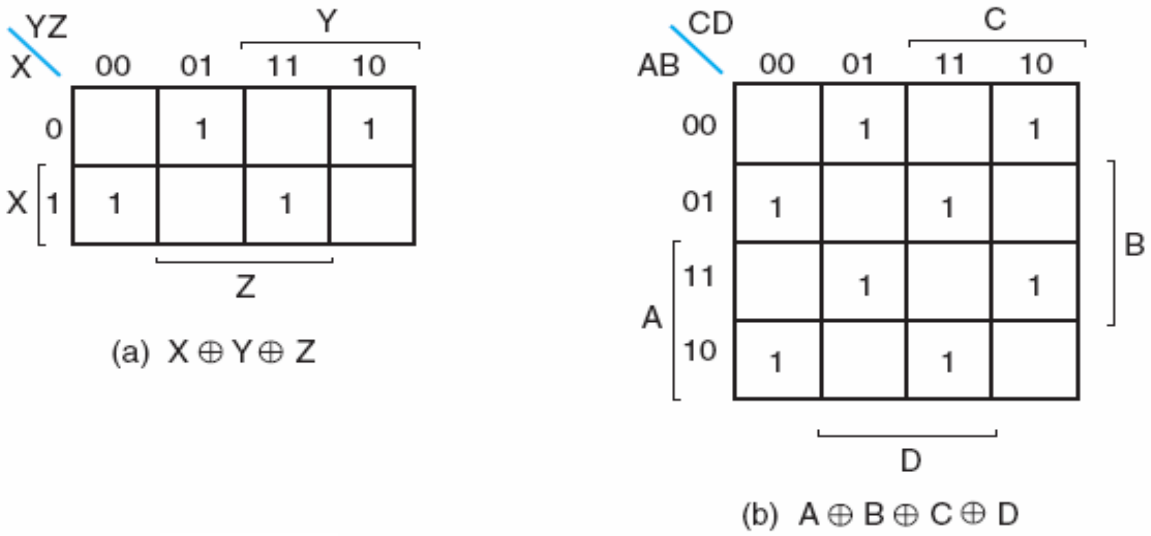


(b) NAND gates

Implementing $F = A\,(CD + B) + B\overline{C}$

Exclusive-OR : ODD Function

- XOR performs the function F=XY'+X'Y.
- The term "exclusive-OR" is more appropriate for two-variable case.
- In general sense the XOR function is known as odd function.
- Why odd function ? The binary values of all the minterms have odd number of 1's.



Maps for Multiple-Variable Odd Functions

XOR : Parity Generation and Checking

- As discussed before, parity bit is an extra bit added to the sequence to make it either even or odd parity.
- Used for error detection during transmission of binary information.
- Odd (and even) functions are used for parity bit generation and checking.
- Parity Generator is a circuit that generates parity bit at the transmitter.
- Parity Checker is a circuit that checks parity bit at the receiver.

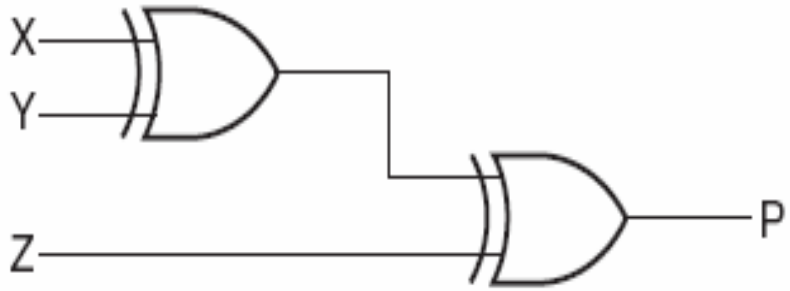
Parity Generator Circuit using XOR

Design of a even parity generator is discussed. The input message is 3-bits. The parity bit generated as 1 if the number of 1's in the 3-bit message is odd, else the generated bit is 0.

Similarly, the design of a parity checker can be considered.

Three-Bit Message			Parity Bit
X	Y	Z	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

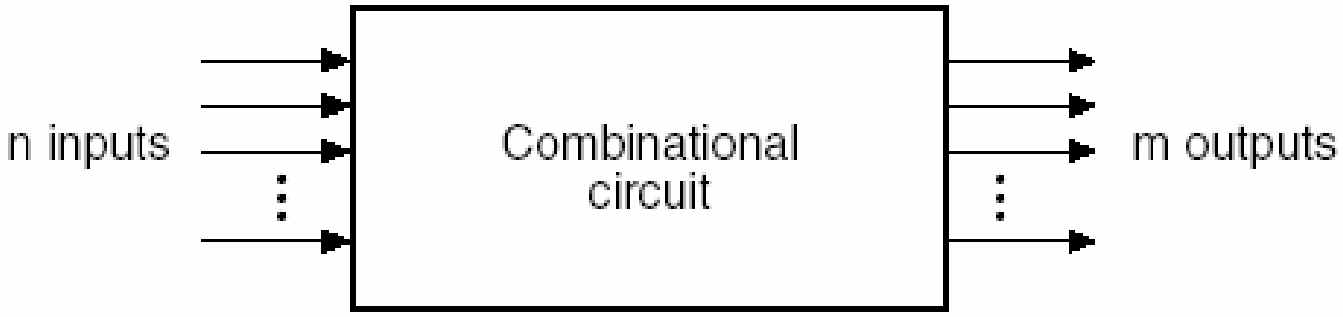
Truth Table for an Even Parity Generator



(a) $P = X \oplus Y \oplus Z$

Combinational Circuit ??

Consists of input variables, output variables, logic gates and interconnections. For n input variables → 2^n possible input implementations. Thus, it can be specified by a truth table. Also by m ($m \leq 2^n$) Boolean functions, one for each possible output variable.



Block Diagram of Combinational Circuit

A combinational circuit has no feedback paths and no storage elements.

Analysis of Combinational Circuits

Functional verification of the logic circuit.
Three Ways

- **Derivation of Boolean Functions:**
  - Label all gate outputs that are function only of input variables or their complements. Determine the Boolean functions for each gate output.
  - Label gates being functions of input variables and previously labeled gates. Find the Boolean functions for the outputs of these gates.
  - Repeat the previous process until circuit outputs are obtained in terms of input variables.

- **Derivation of the truth table:**
  - For n inputs list the binary numbers in a table from 0 to 2^n-1.
  - Break circuit into small single-output blocks by labeling each block output with an arbitrary symbol.
  - Obtain the truth table for the blocks depending on input variables only
  - Proceed until columns for all circuit outputs are determined.

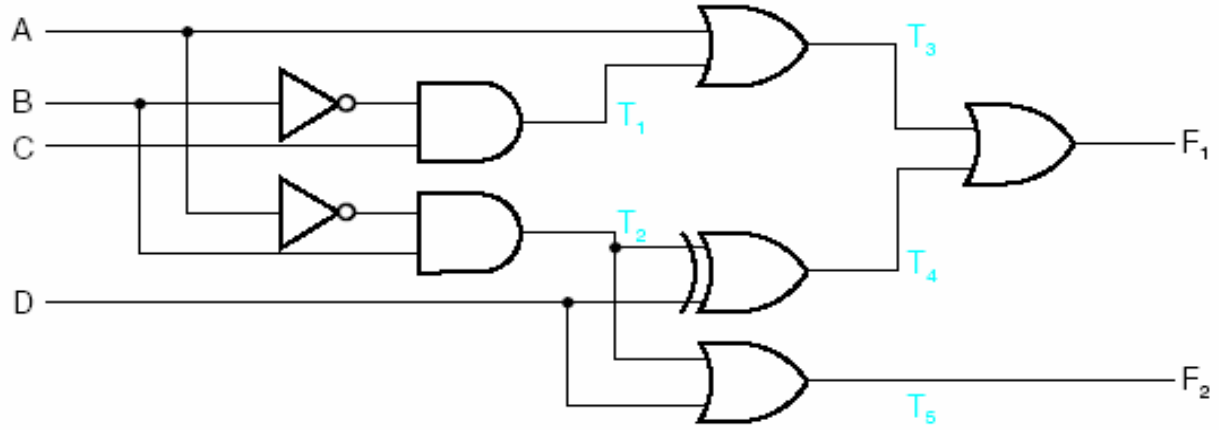
- **Logic Simulation:**
  - Enter circuit as a netlist or schematic
  - Specify inputs as a file contents the simulator can read, or by entering inputs interactively into the simulator.
  - Proceed with simulation.

Combinational Analysis : Boolean functions

Gate output that are functions of input only : T_1=B'C and T_2=A'B

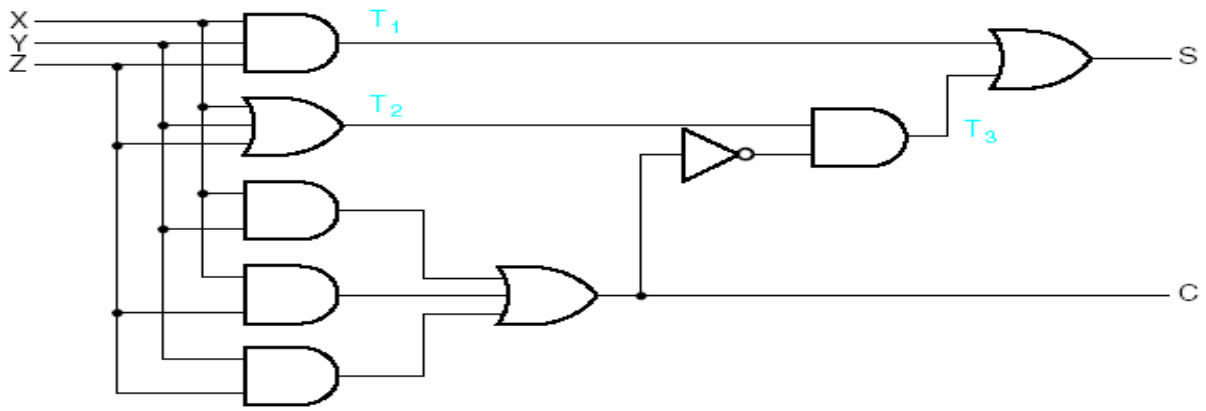
Other gates Outputs of : T_3=A+T_1=A+B'C, T_4=T_2XOR D, and T_5=T_2+D

Outputs : F_2=T_5=A'B+D and F_1=T_3+T_4 = A+B'C+BD'+B'D



Logic Diagram for Analysis Example

Combinational Analysis : Truth Table

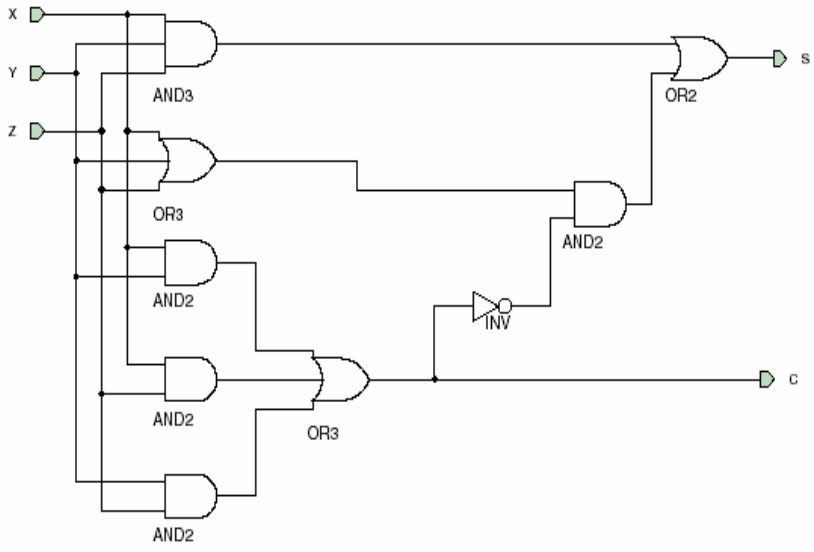


Logic Diagram for Binary Adder

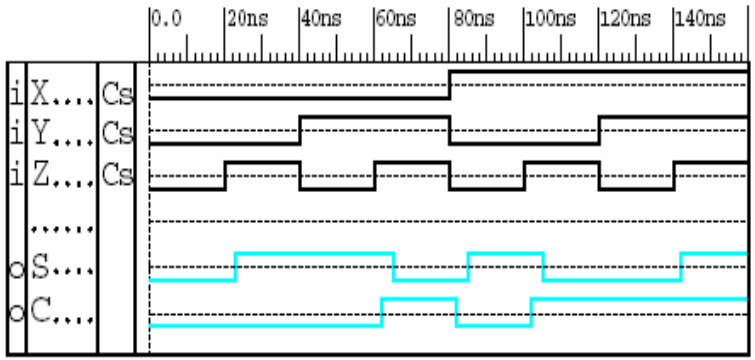
X	Y	Z	C	$\overline{C}$	T_1	T_2	T_3	S
0	0	0	0	1	0	0	0	0
0	0	1	0	1	0	1	1	1
0	1	0	0	1	0	1	1	1
0	1	1	1	0	0	1	0	0
1	0	0	0	1	0	1	1	1
1	0	1	1	0	0	1	0	0
1	1	0	1	0	0	1	0	0
1	1	1	1	0	1	1	0	1

Truth Table for Binary Adder

Combinational Analysis : Simulation



Xilinx Foundation Schematic for Binary Adder



Waveforms for the Binary Adder Schematic

To verify the correctness check the output waveform values against the truth table of full adder.

Combinational Circuits :Design Procedure

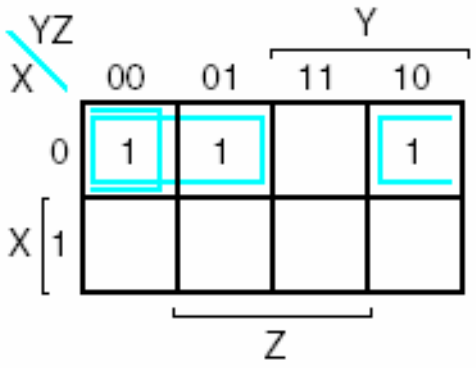
- Step 1: From circuit specifications determine required number of inputs and outputs, assign letter symbol.
- Step 2: Derive truth table that defines required relationship between inputs and outputs.
- Step 3: Obtain simplified Boolean functions for each output as a function of input variables.
- Step 4: Draw the logic diagram.
- Step 5: Verify design correctness.

Combinational Design Example

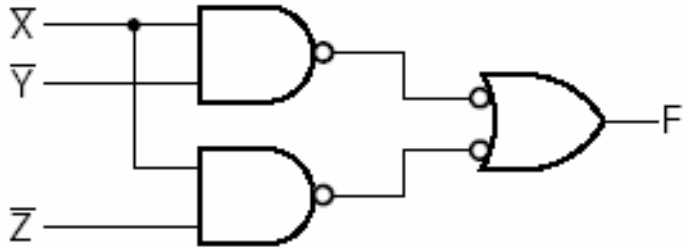
Design a combinatorial circuit with 3 inputs and 1 output, where the output must be logic 1 when the binary value of the inputs is less than 011 (3) and logic 0 otherwise. Use only NAND gates. (Input: X, Y, Z. Output: F)

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

(a) Truth table



(b) Map $F = \overline{X}\overline{Y} + \overline{X}Z$



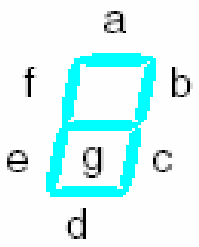
(c) Logic diagram

Combinational Design Example : Code Converter

- For a combinatorial circuit with two or more outputs, each output must be expressed separately as a function of all the input variables.
- Example of a multi-output circuit is a code converter that translates info from one binary code to another. Inputs provide the elements bit combination as specified by the first code, the outputs generate the corresponding bit combination of the second code.
- Illustration via an example:
  - BCD to the seven signals required to drive a seven-segment light-emitting diode (LED) display.

Comb Example: BCD-to-Seven-Segment Decoder

It accepts a decimal digit in BCD and generates the appropriate outputs for the selection of segments that display the decimal digit.

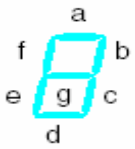


(a) Segment designation

(b) Numeric designation for display

Seven-Segment Display

Comb Example: BCD-to-Seven-Segment Decoder …



(a) Segment designation

(b) Numeric designation for display

Seven-Segment Display

BCD Input				Seven-Segment Decoder						
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
All other inputs				0	0	0	0	0	0	0

Truth Table for BCD–to–Seven-Segment Decoder

Comb Example: BCD-to-Seven-Segment Decoder …

- One possible way of simplification results in the following Boolean expressions:

a= A'C+A'BD+B'C'D'+AB'C'

b= A'B'+A'C'D'+A'CD+AB'C'

c= A'B+A'D+B'C'D'+AB'C'

d= A'CD'+A'B'C+B'C'D'+AB'C'+A'BC'D

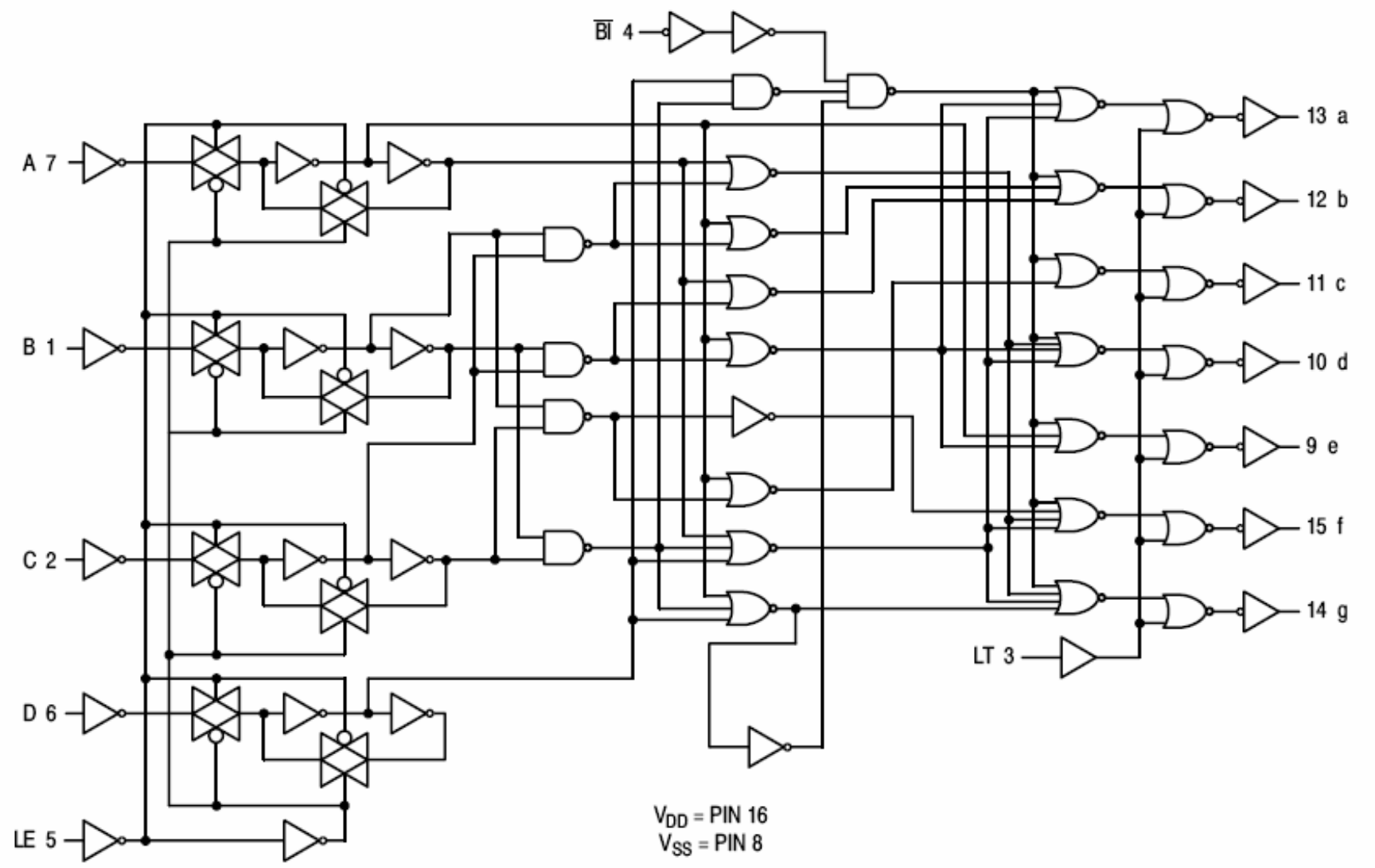
e= A'CD'+B'C'D'

f= A'BC'+A'C'D'+A'BD'+AB'C'

g= A'CD'+A'B'C+A'BC'+AB'C'

- 14 AND gates 7 OR gates. 27 product terms, 6 common terms.

Comb Example: BCD-to-Seven-Segment Decoder …



Comb Example: BCD-to-Seven-Segment Decoder …

•The circuit shown in the previous slide is logic diagram of MC14511B from http://onsemi.com shown in the figure below.

•Lamp test (LT), blanking (BI), and latch enable (LE) inputs are used to test the display, to turn–off the display, and to store a BCD code, respectively.
•Applications: It can be used with LED, incandescent, fluorescent, gas discharge, or liquid crystal.