Logic and Computer Design Fundamentals

Chapter 2 – Combinational Logic Circuits

Part 2 – Circuit Optimization

Yueming Wang (王跃明), Professor

ymingwang@zju.edu.cn

2018

College of Computer Science, Zhejiang University

Qiushi Academy for Advanced Studies, Zhejiang University

Overview

- **Part 1 – Gate Circuits and Boolean Equations**
  - Binary Logic and Gates
  - Boolean Algebra
  - Standard Forms
- **Part 2 – Circuit Optimization**
  - Two-Level Optimization
  - Map Manipulation
  - Practical Optimization
  - Multi-Level Circuit Optimization
- **Part 3 – Additional Gates and Circuits**
  - Other Gate Types
  - Exclusive-OR Operator and Gates
  - High-Impedance Outputs

Circuit Optimization

- Goal: To obtain the simplest implementation for a given function
- Optimization is a more formal approach to simplification that is performed using a specific procedure or algorithm
- Optimization requires a cost criterion to measure the simplicity of a circuit
- Distinct cost criteria we will use:
  - Literal cost (L)
  - Gate input cost (G)
  - Gate input cost with NOTs (GN)

Literal Cost

- **Literal – a variable or its complement**
- **Literal cost – the number of literal appearances in a Boolean expression corresponding to the logic circuit diagram**
- **Examples:**
  - $F = BD + A\overline{B}C + A\overline{C}\,\overline{D}$ L = 8
  - $F = BD + A\overline{B}C + A\overline{B}\,\overline{D} + AB\overline{C}$ L = 11
  - $F = (A + B)(A + D)(B + C + \overline{D})(\overline{B} + \overline{C} + D)$ L = 10
  - **Which solution is best?** The first solution is best

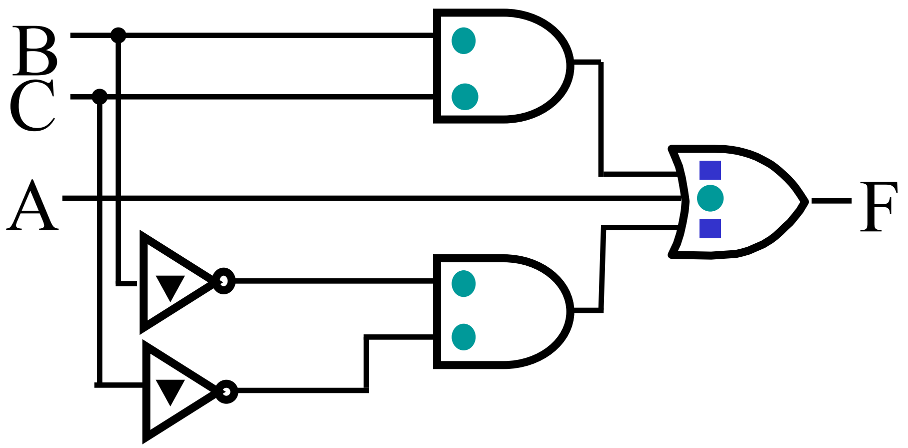
Gate Input Cost

- Gate input costs - the number of inputs to the gates in the implementation corresponding exactly to the given equation or equations. (G - inverters not counted, GN - inverters counted)
- For SOP and POS equations, it can be obtained from the equation(s) by finding the sum of:
  - all literal appearances
  - the number of terms excluding single literal terms, (G) and
  - optionally, the number of distinct complemented single literals (GN).
- Example:
  - $F = BD + A\overline{B}C + A\overline{C}\,\overline{D}$ $\quad G = 11, GN = 14$
  - $F = BD + A\overline{B}C + A\overline{B}\,\overline{D} + AB\overline{C}$ $\quad G = 15, GN = 18$
  - $F = (A + \overline{B})(A + D)(B + C + \overline{D})(\overline{B} + \overline{C} + D)$ $\quad G = 14, GN = 17$
  - Which solution is best? The 1st solution is best

Cost Criteria (continued)

- Example 1: $GN = G + 2 = 9$
- $F = A + B\,C + \overline{B}\,\overline{C}$ $L = 5$

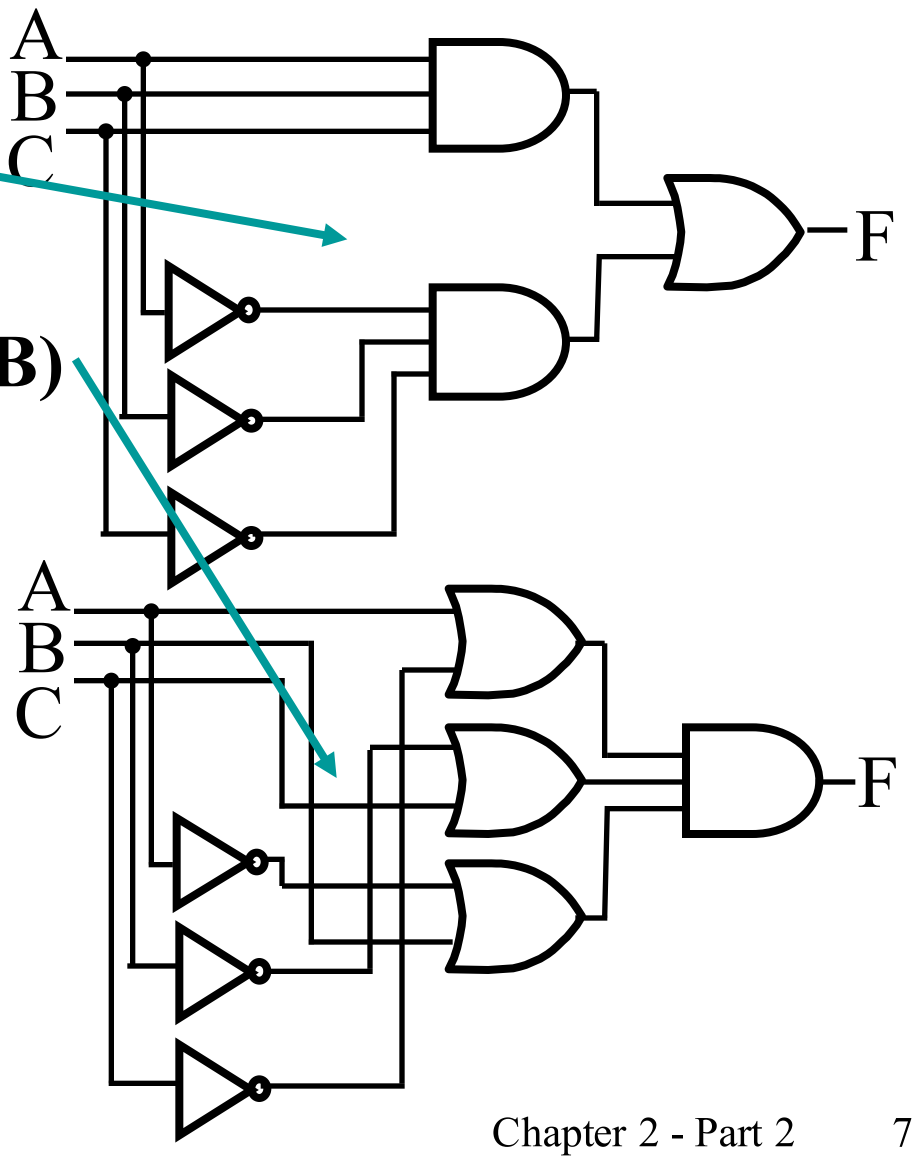
$G = L + 2 = 7$



- L (literal count) counts the AND inputs and the single literal OR input.
- G (gate input count) adds the remaining OR gate inputs
- GN(gate input count with NOTs) adds the inverter inputs

Cost Criteria (continued)

- **Example 2:**
- **$F = A\,B\,C + \overline{A}\,\overline{B}\,\overline{C}$**
- **L = 6 G = 8 GN = 11**
- **$F = (A + \overline{C})(\overline{B} + C)(\overline{A} + B)$**
- **L = 6 G = 9 GN = 12**
- **<u>Same</u> function and <u>same</u> literal cost**
- **But first circuit has <u>better</u> gate input count and <u>better</u> gate input count with NOTs**
- **Select it!**



Boolean Function Optimization

- **Minimizing the gate input (or literal) cost of a (a set of) Boolean equation(s) reduces circuit cost.**
- **We choose gate input cost.**
- **Boolean Algebra and graphical techniques are tools to minimize cost criteria values.**
- **Some important questions:**
  - **When do we stop trying to reduce the cost?**
  - **Do we know when we have a minimum cost?**
- **Treat optimum or near-optimum cost functions for two-level (SOP and POS) circuits first.**
- **Introduce a graphical technique using Karnaugh maps (K-maps, for short)**

Karnaugh Maps (K-map)

- **A K-map is a collection of squares**
  - **Each square represents a minterm**
  - **The collection of squares is a graphical representation of a Boolean function**
  - **Adjacent squares differ in the value of one variable**
  - **Alternative algebraic expressions for the same function are derived by recognizing patterns of squares**
- **The K-map can be viewed as**
  - **A reorganized version of the truth table**
  - **A topologically-warped Venn diagram as used to visualize sets in algebra of sets**

Two Variable Maps

- **A 2-variable Karnaugh Map:**
  - **Note that minterm m0 and minterm m1 are "adjacent" and differ in the value of the variable y**
  - **Similarly, minterm m0 and minterm m2 differ in the x variable.**
  - **Also, m1 and m3 differ in the x variable as well.**
  - **Finally, m2 and m3 differ in the value of the variable y**

	$y = 0$	$y = 1$
$x = 0$	$m_0 = \overline{x}\,\overline{y}$	$m_1 = \overline{x}\,y$
$x = 1$	$m_2 = x\,\overline{y}$	$m_3 = x\,y$

K-Map and Truth Tables

- The K-Map is just a different form of the truth table.
- Example – Two variable function:
  - We choose a,b,c and d from the set {0,1} to implement a particular function, F(x,y).

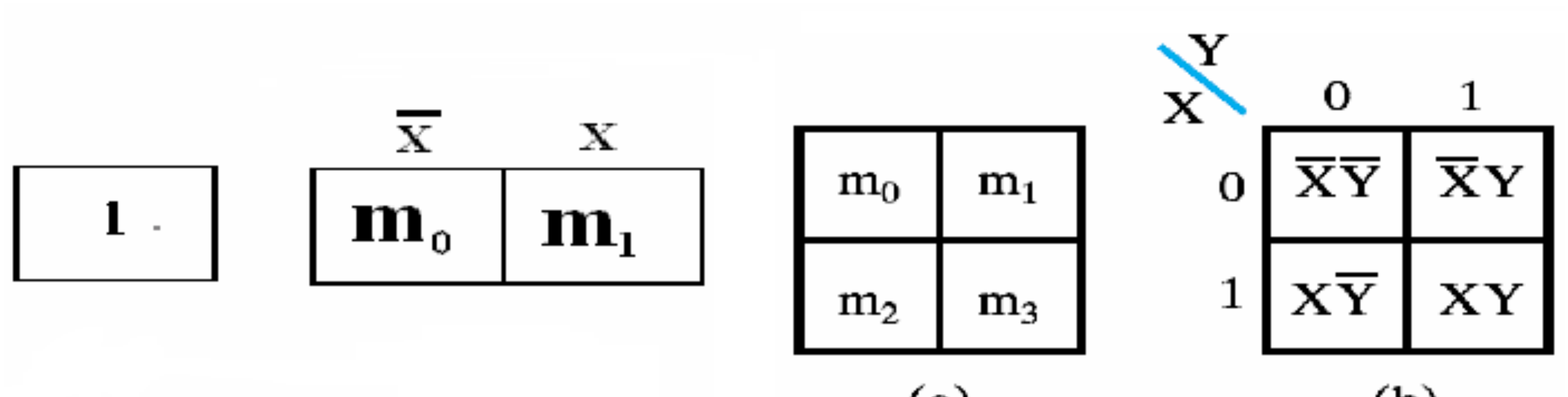
Function Table

Input Values (x,y)	Function Value F(x,y)
0 0	a
0 1	b
1 0	c
1 1	d

K-Map

	y = 0	y = 1
x = 0	a	b
x = 1	c	d

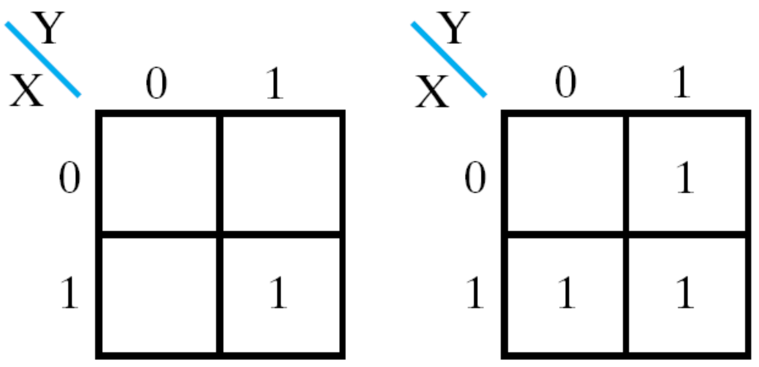
Examples



(a) (b)

Const、one variable、two variable K-maps

$$F(X,Y) = m_1 + m_2 + m_3 = \overline{X}Y + X\overline{Y} + XY = X + Y$$

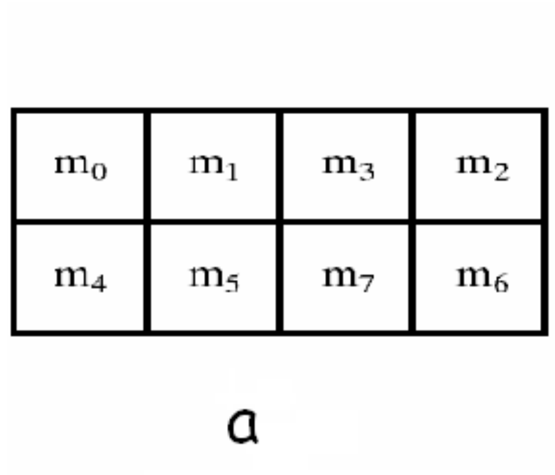


(a) XY (b) X + Y

Present function with K-maps

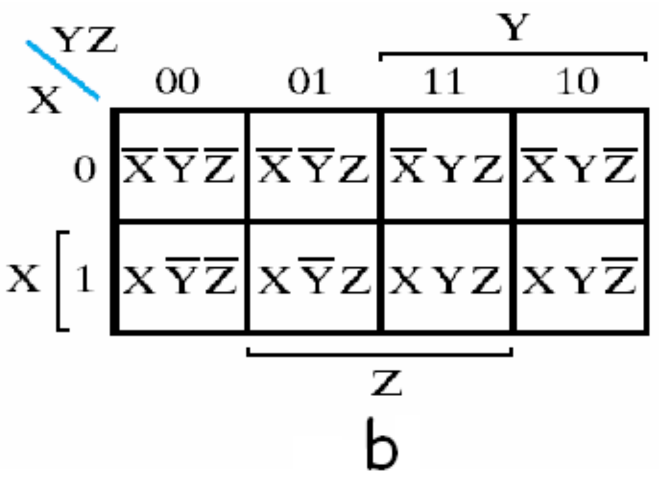
Three variable maps

Has 2^3 minterms， and 8 squares.



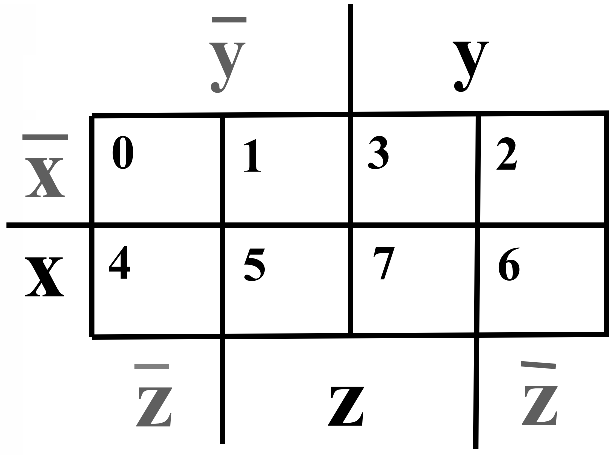
m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

a

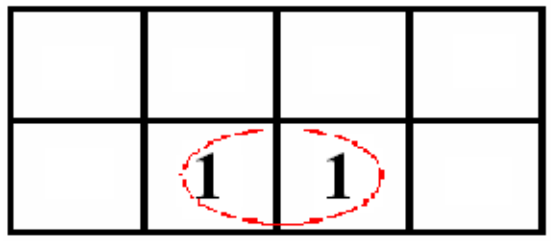


$$\begin{array}{c|cccc} X \backslash YZ & 00 & 01 & 11 & 10 \\ \hline 0 & \overline{X}\,\overline{Y}\,\overline{Z} & \overline{X}\,\overline{Y}Z & \overline{X}YZ & \overline{X}Y\overline{Z} \\ 1 & X\overline{Y}\,\overline{Z} & X\overline{Y}Z & XYZ & XY\overline{Z} \end{array}$$

b



$$\begin{array}{c|cccc} & \overline{y} & & y & \\ \hline \overline{x} & 0 & 1 & 3 & 2 \\ x & 4 & 5 & 7 & 6 \\ \hline & \overline{z} & z & & \overline{z} \end{array}$$



	1	1	

Example:

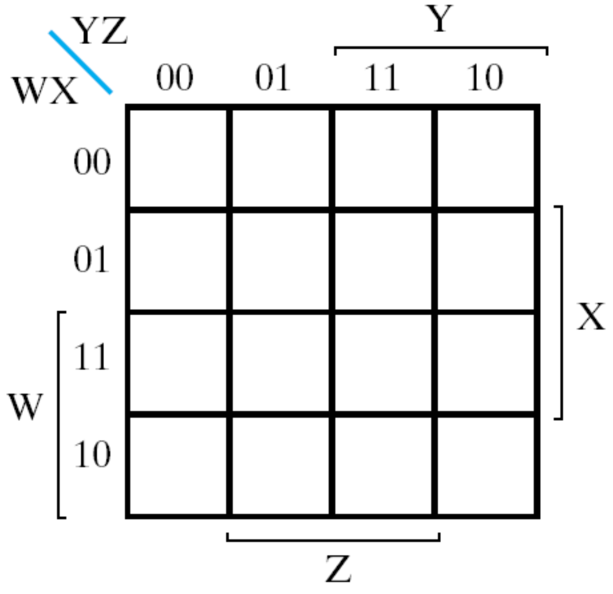
$$F\left(X,Y,Z\right)=m_5+m_7=X\overline{Y}Z+XYZ=XZ\left(\overline{Y}+Y\right)=XZ$$

Four variable maps

Has 2^4 minterms， and 16 squares.

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

(a)



(b)

WX \ YZ	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

Five variable maps

Has 2^5 minterms， and 32 squares.

VW \ XYZ	000	001	011	010	110	111	101	100
00	0	1	3	2	6	7	5	4
01	8	9	11	10	14	15	13	12
11	24	25	27	26	30	31	29	28
10	16	17	19	18	22	23	21	20

Example: Combining Squares

- **Example: F(x,y) = x**

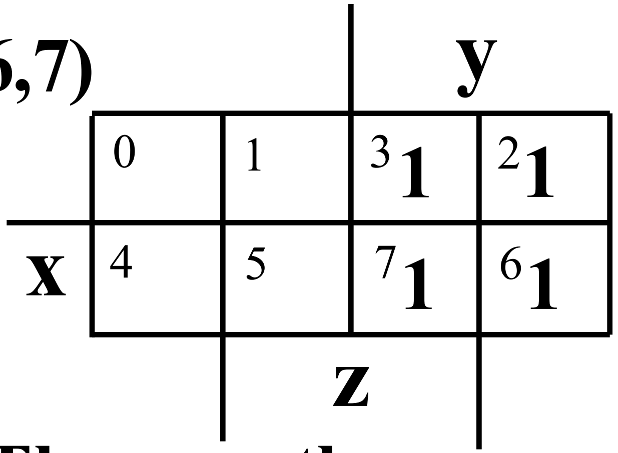
F = x	y = 0	y = 1
x = 0	0	0
x = 1	1	1

- **For function F(x,y), the two adjacent cells containing 1's can be combined using the Minimization Theorem:**

$$F(x,y) = x\,\overline{y} + x\,y = x$$

Example: Combining Squares

- Example: Let $F = \Sigma m(2,3,6,7)$

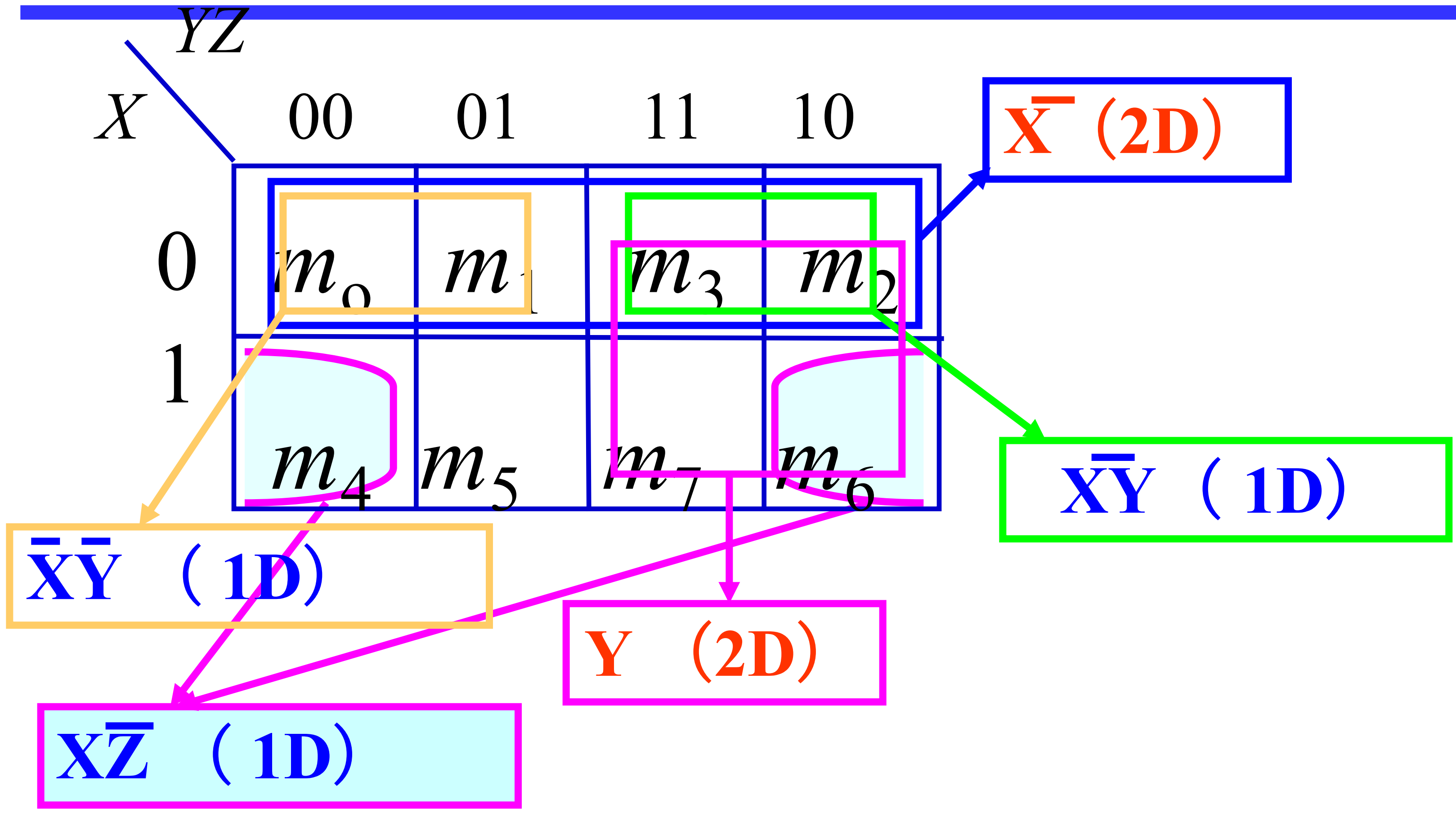


- Applying the Minimization Theorem three times:

$$\begin{aligned} F(x,y,z) &= \overline{x}\,y\,z + x\,y\,z + \overline{x}\,y\,\overline{z} + x\,y\,\overline{z} \\ &= yz + y\overline{z} \\ &= y \end{aligned}$$

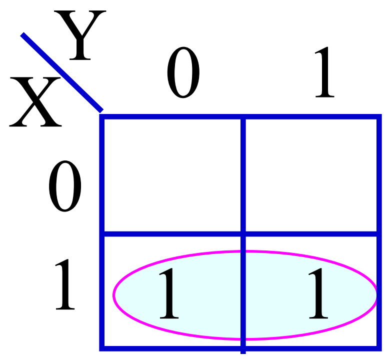
- Thus the four terms that form a 2 $\times$ 2 square correspond to the term "y".

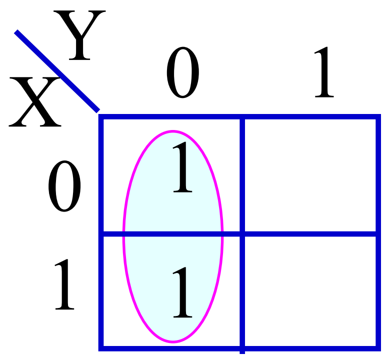
Dimension of Squares

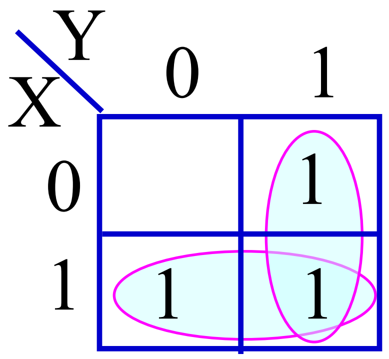


One variable can be removed with 1D square, and two variables can be removed with 2D square

Example: Combining Squares

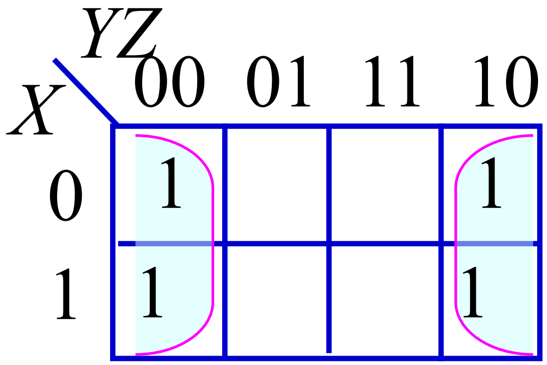


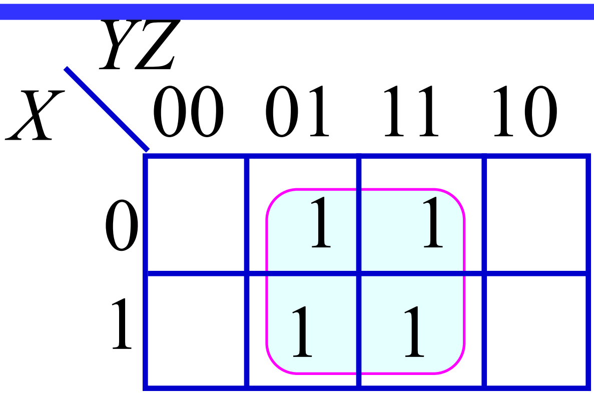


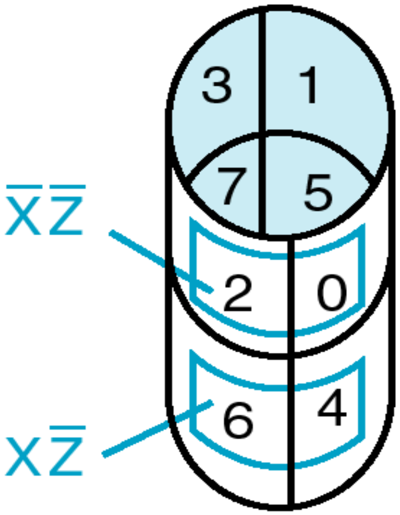


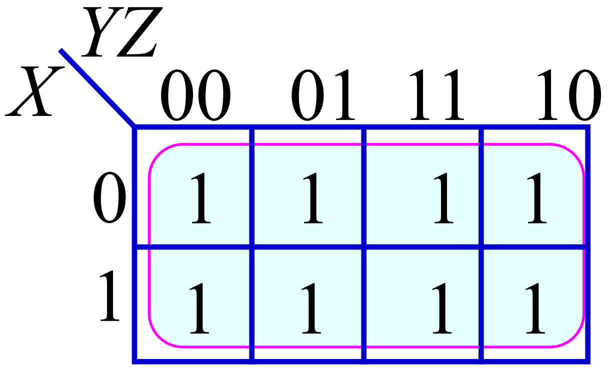
Combining Squares in two variable maps

Example: Combining Squares



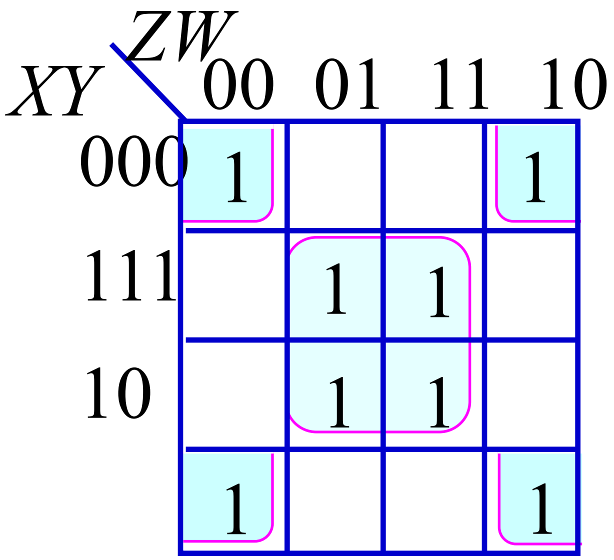


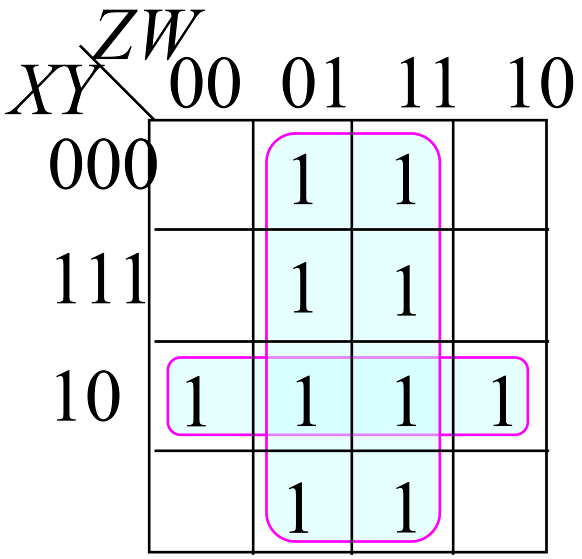


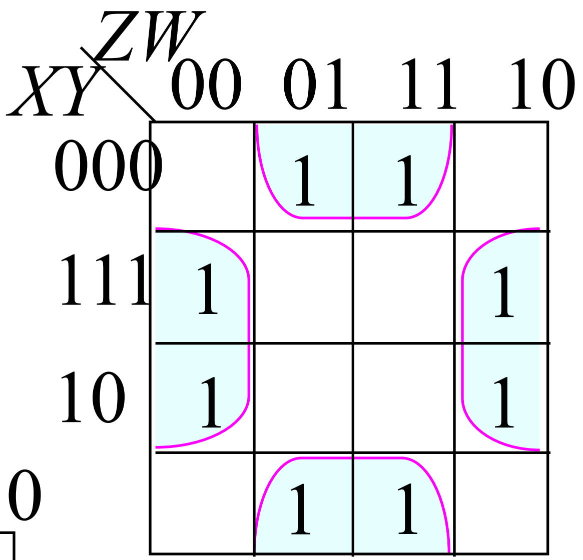


Combining Squares in three variable maps

Example: Combining Squares







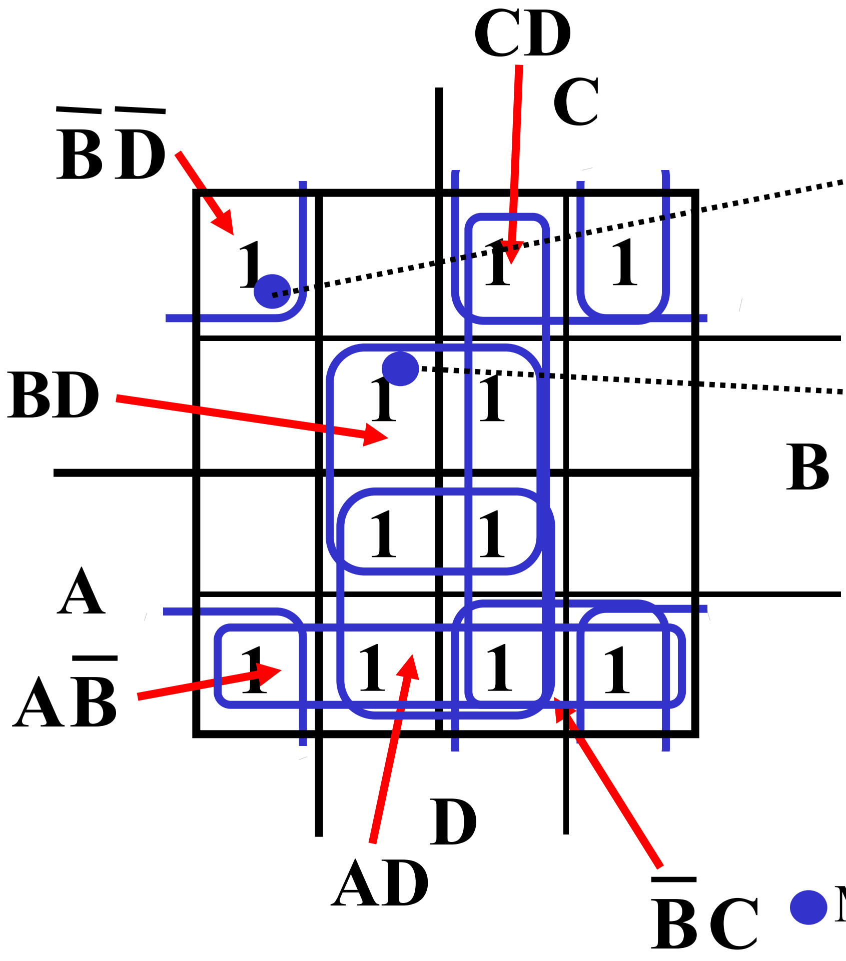
Combining Squares in four variable maps

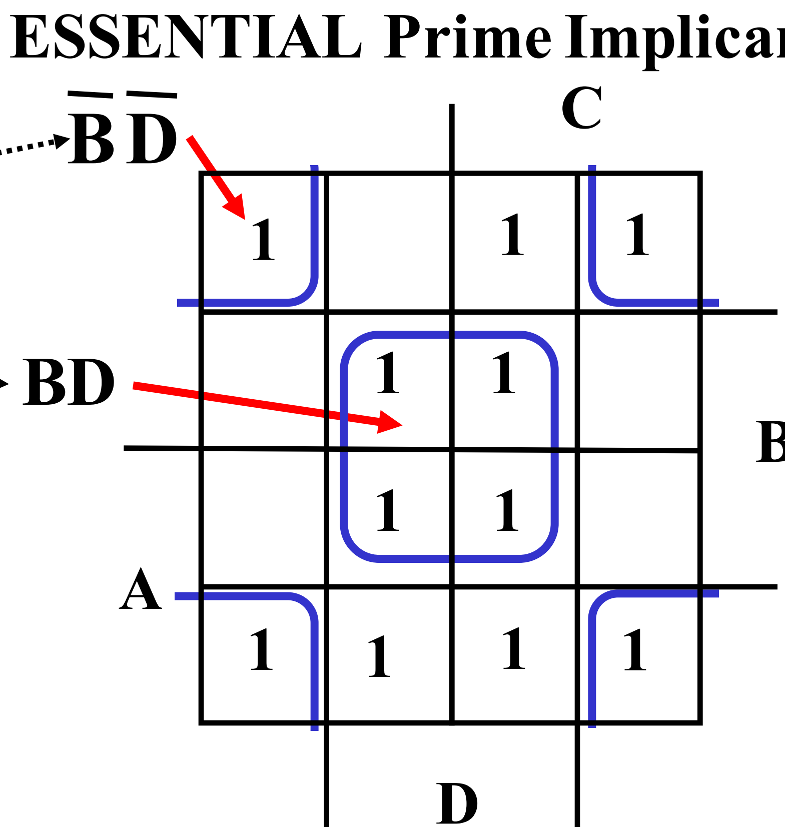
Systematic Simplification

- A *Prime Implicant* is a product term obtained by combining the maximum possible number of adjacent squares in the map into a rectangle with the number of squares a power of 2.
- A prime implicant is called an *Essential Prime Implicant* if it is the <u>only</u> prime implicant that covers (includes) one or more minterms.
- Prime Implicants and Essential Prime Implicants can be determined by inspection of a K-Map.
- A set of prime implicants *"covers all minterms"* if, for each minterm of the function, at least one prime implicant in the set of prime implicants includes the minterm.

Example of Prime Implicants

- Find ALL Prime Implicants





● Minterms covered by single prime implicant

Optimization Algorithm

- Find all prime implicants.
- Include all essential prime implicants in the solution
- Select a minimum cost set of non-essential prime implicants to cover all minterms not yet covered:
  - Obtaining an optimum solution: See Reading Supplement - More on Optimization
  - Obtaining a good simplified solution: Use the Selection Rule

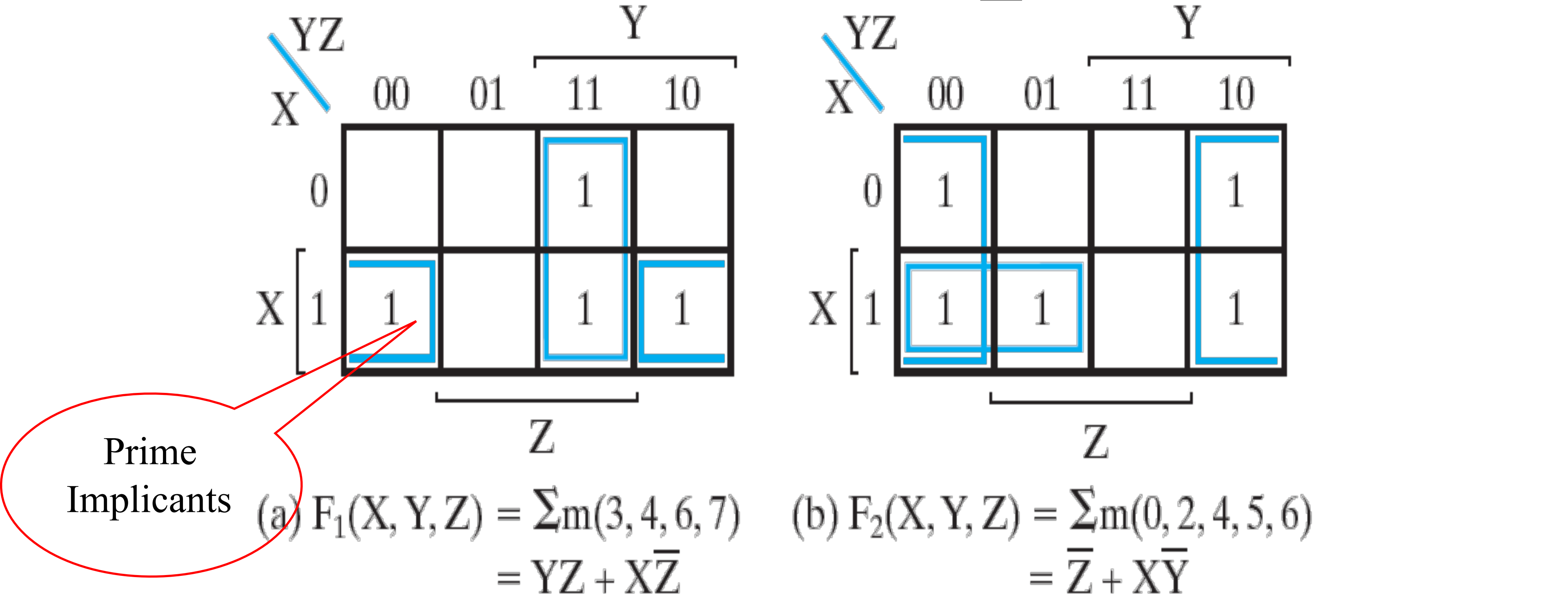
Prime Implicant Selection Rule

- **Minimize the overlap among prime implicants as much as possible. In particular, in the final solution, make sure that each prime implicant selected includes at least one minterm not included in any other prime implicant selected.**

K-Map Simplification

$$F_1(X,Y,Z) = m_3 + m_4 + m_6 + m_7 = \sum m(3,4,6,7)$$

$$F_2(X,Y,Z) = m_0 + m_2 + m_4 + m_5 + m_6 = \sum m(0,2,4,5,6)$$

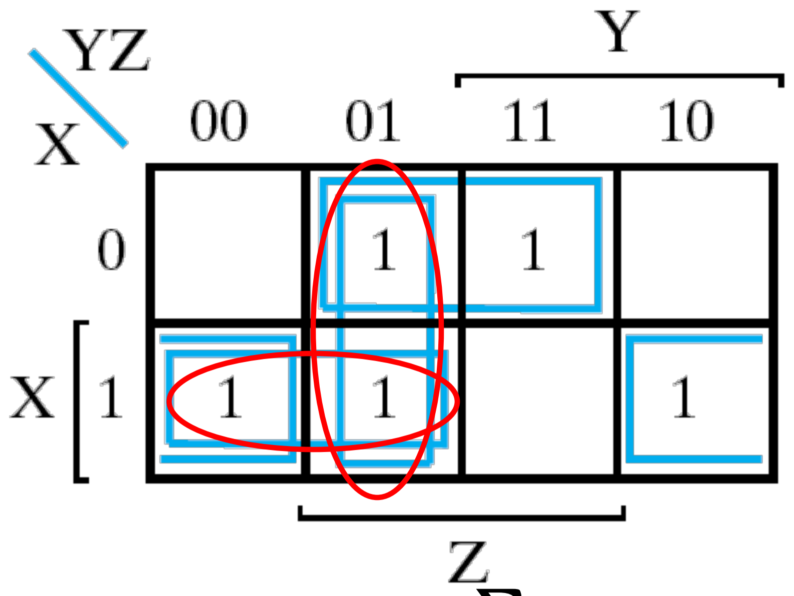


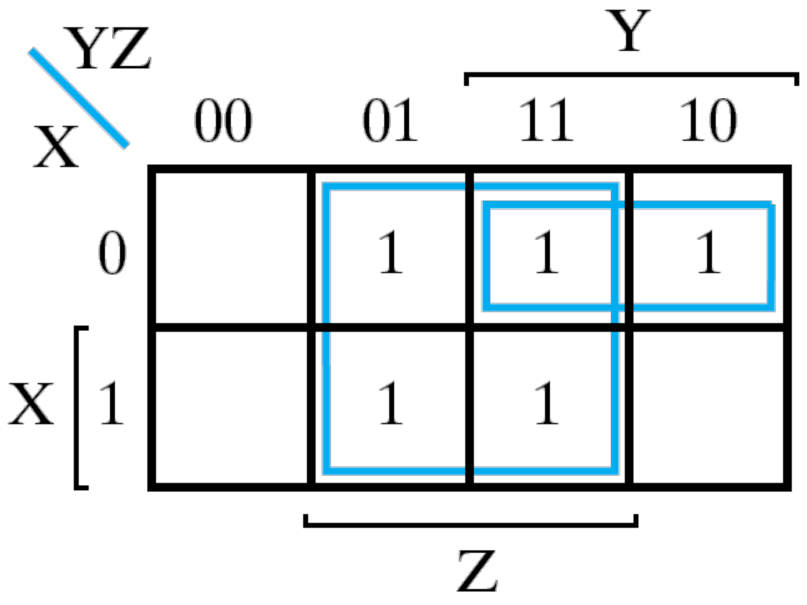
YZ
X
Y
00
01
11
10
0
1
X
Z
Prime Implicants
(a) $F_1(X,Y,Z) = \Sigma m(3,4,6,7) = YZ + X\overline{Z}$
(b) $F_2(X,Y,Z) = \Sigma m(0,2,4,5,6) = \overline{Z} + X\overline{Y}$

K-Map Simplification

$$F_1 = F(X,Y,Z) = \sum m(1,3,4,5,6)$$

$$F_2 = F(X,Y,Z) = \sum m(1,2,3,5,7)$$





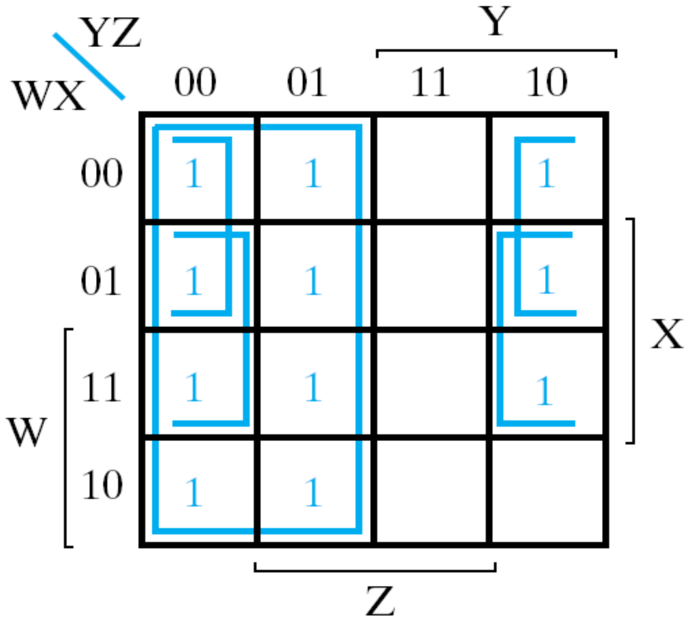
$$F(X,Y,Z) = \sum m(1,3,4,5,6)$$
$$= \overline{X}Z + X\overline{Y} + X\overline{Z}$$
$$= \overline{X}Z + \overline{Y}Z + X\overline{Z}$$

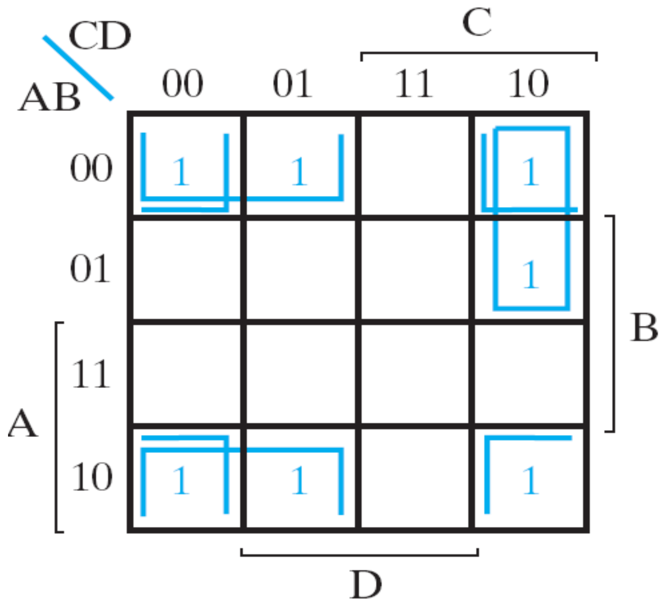
$$F(X,Y,Z) = \sum m(1,2,3,5,7)$$
$$= Z + \overline{X}Y$$

K-Map Simplification

$$F(W,X,Y,Z) = \sum m(0,1,2,4,5,6,8,9,12,13,14)$$

$$F(A,B,C,D) = \overline{A}\,\overline{B}\,\overline{C} + \overline{B}C\overline{D} + A\overline{B}\,\overline{C} + \overline{A}BC\overline{D}$$

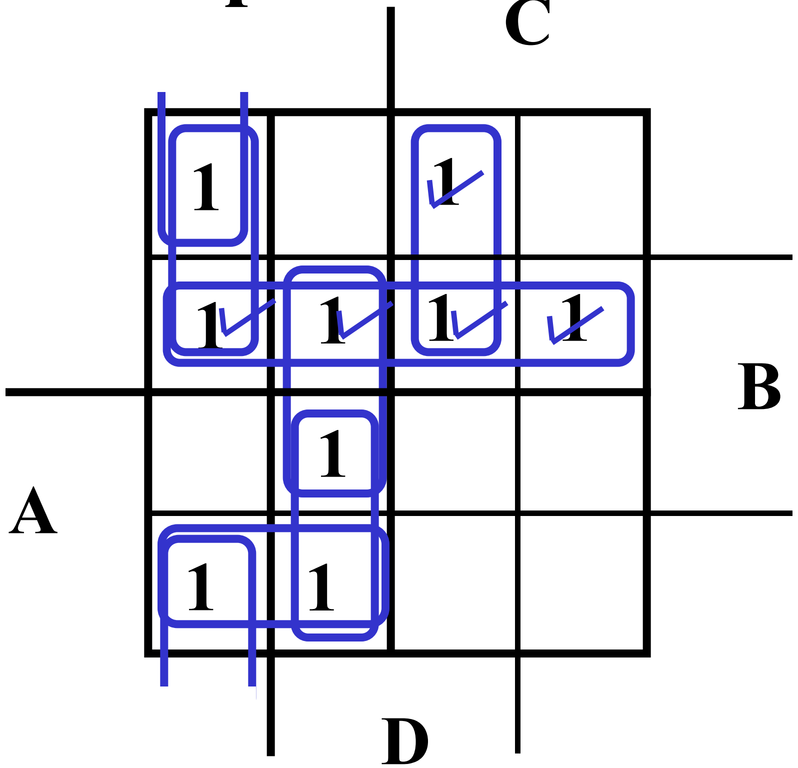


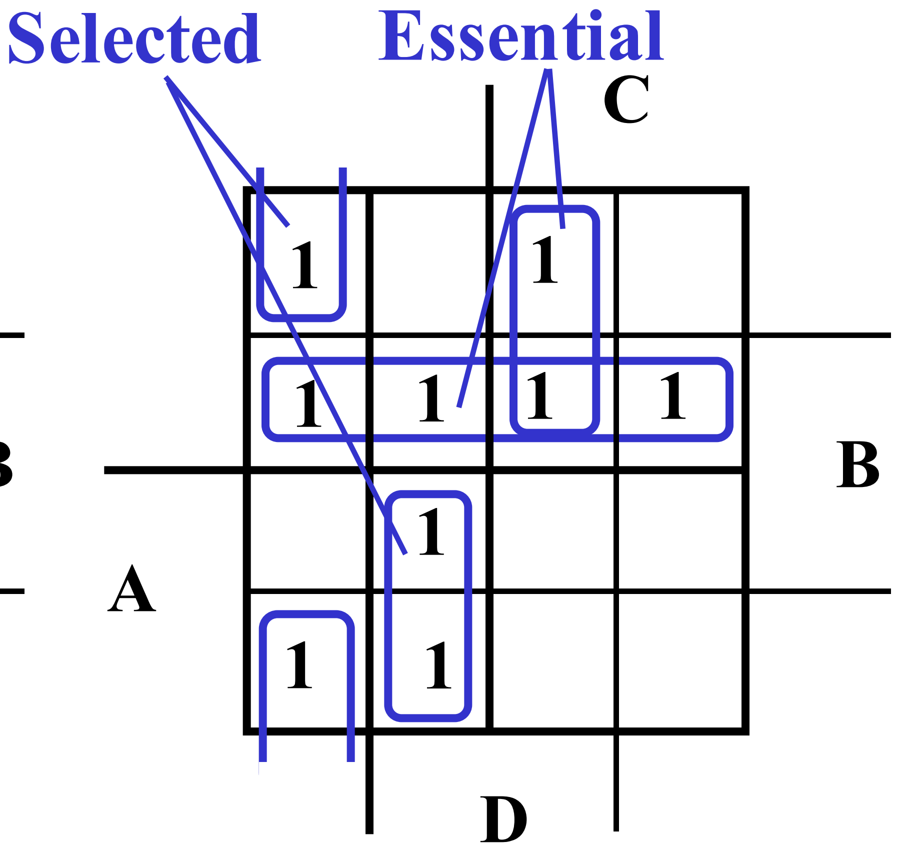


$$F(W,X,Y,Z) = \overline{Y} + \overline{W}\,\overline{Z} + X\overline{Z} \qquad F(A,B,C,D) = \overline{B}\,\overline{D} + \overline{B}\,\overline{C} + \overline{A}C\overline{D}$$

Selection Rule Example

- Simplify F(A, B, C, D) given on the K-map.

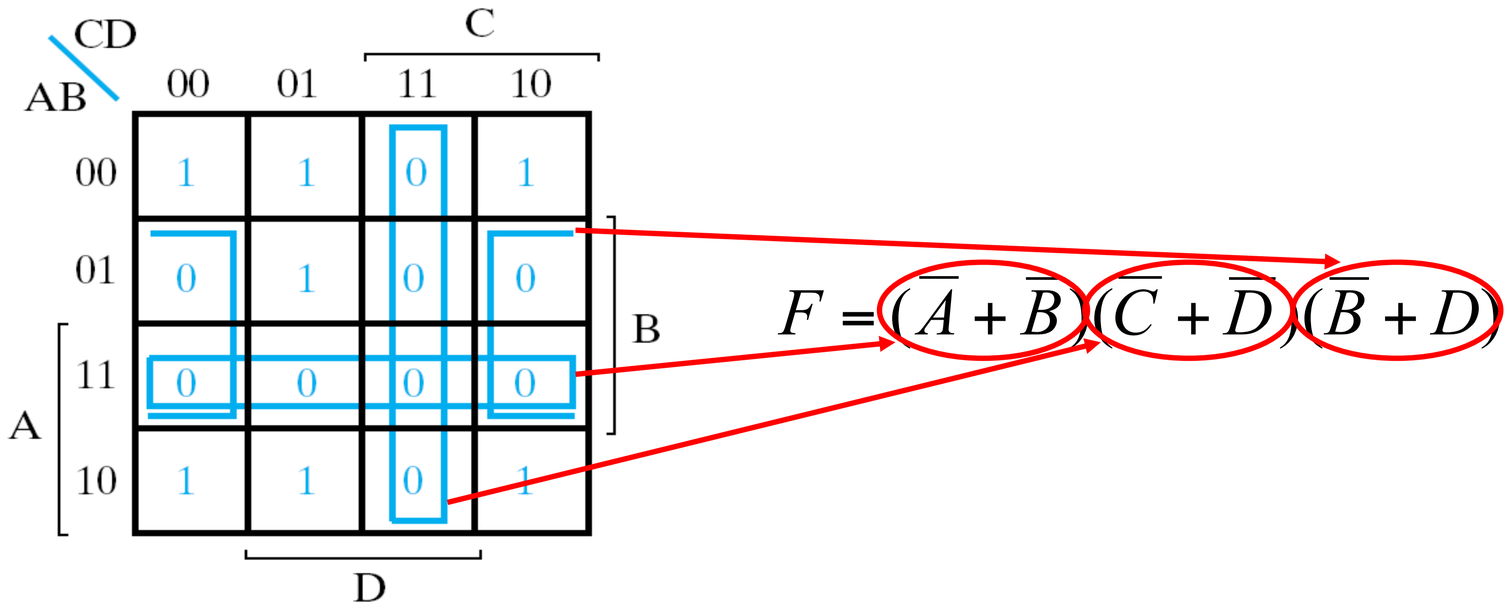




✓ Minterms covered by essential prime implicants

K-Map Simplification with maxterms

Equation: $F(A,B,C,D)=\sum m(0,1,2,5,8,9,10)$


CD
AB
C
00
01
11
10
00
01
11
10
1
1
0
1
0
1
0
0
0
0
0
0
1
1
0
1
B
A
D
$F=(\overline{A}+\overline{B})(\overline{C}+\overline{D})(\overline{B}+D)$

Simplification with Maxterm: complemented function

Equation: $F = \left(\overline{A} + \overline{B} + C\right)\left(B + D\right)$

Complemented function : $\overline{F} = AB\overline{C} + \overline{B}\,\overline{D}$

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

Combine 1: $F = \overline{A}B + \overline{B}D + BC$

Combine 0: $\overline{F} = \overline{B}\,\overline{D} + AB\overline{C}$

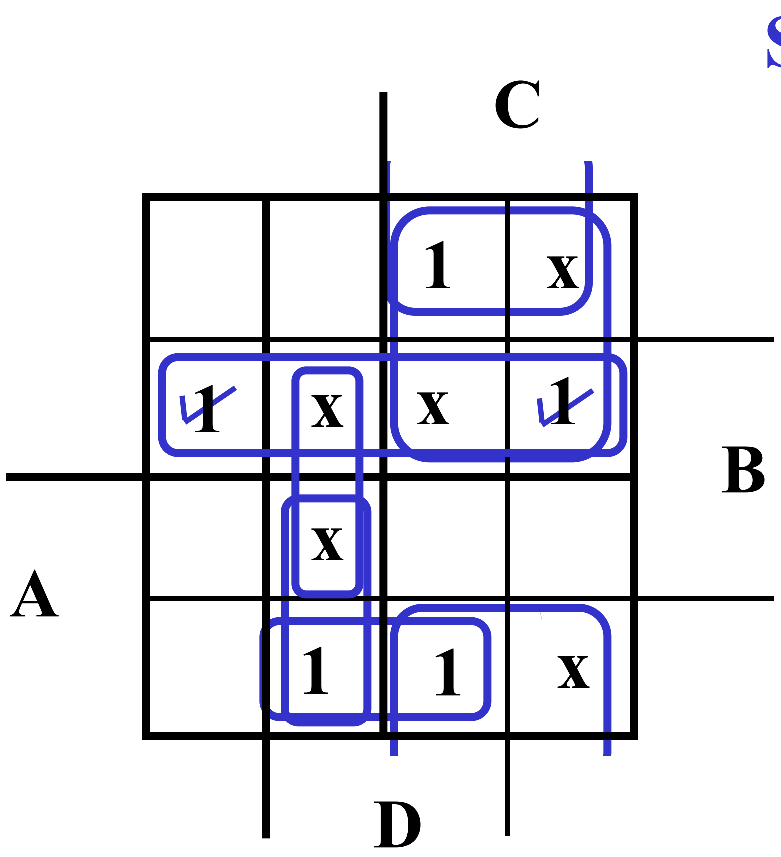
So, $F = \left(\overline{A} + \overline{B} + C\right)\left(B + D\right)$

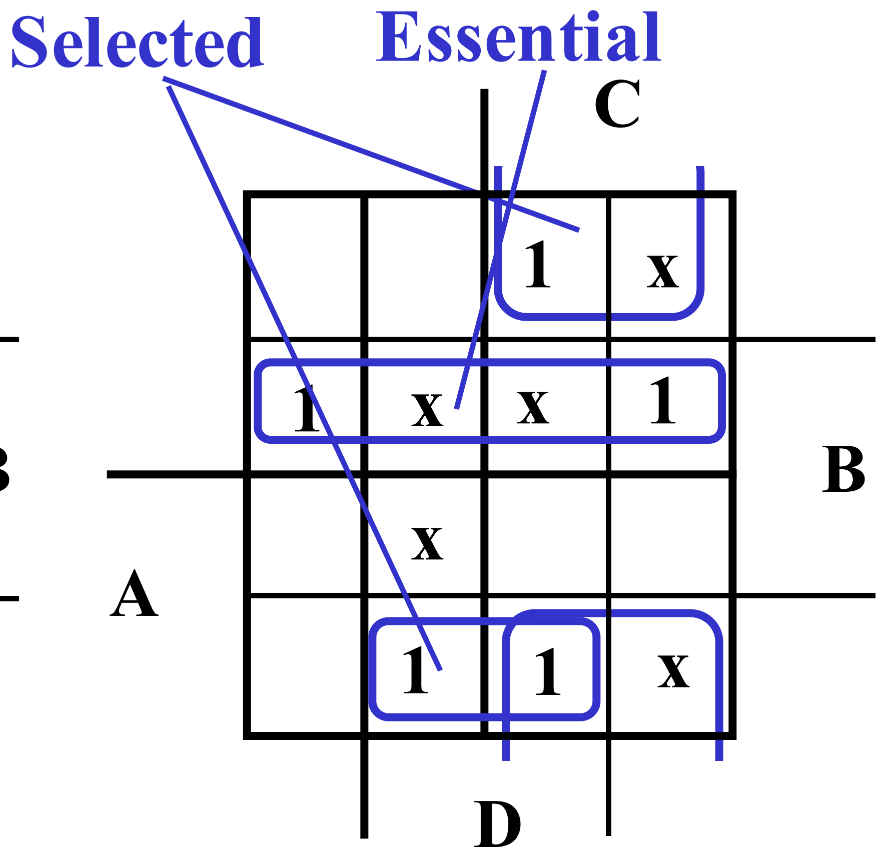
Don't Cares in K-Maps

- **Sometimes a function table or map contains entries for which it is known:**
  - **the input values for the minterm will never occur, or**
  - **The output value for the minterm is not used**
- **In these cases, the output value need not be defined**
- **Instead, the output value is defined as a "don't care"**
- **By placing "don't cares" (an "x" entry) in the function table or map, the cost of the logic circuit may be lowered.**
- **Example 1: A logic function having the binary codes for the BCD digits as its inputs. Only the codes for 0 through 9 are used. The six codes, 1010 through 1111 <u>never occur</u>, so the output values for these codes are "x" to represent "don't cares."**

Optimization with Don't Cares

- Simplify F(A, B, C, D) given on the K-map.





✓ Minterms covered by essential prime implicants

Product of Sums Example

- **Find the optimum POS solution:**

  $F(A, B, C, D) = \Sigma_m(3,9,11,12,13,14,15) + \Sigma d\ (1,4,6)$

  - **Hint: Use $\overline{F}$ and complement it to get the result.**

$F' = B'D' + A'B$

$F = (B + D)(A + B')$

Multiple-Level Optimization*

- Multiple-level circuits - circuits that are not two-level (with or without input and/or output inverters)
- Multiple-level circuits can have reduced gate input cost compared to two-level (SOP and POS) circuits
- Multiple-level optimization is performed by applying transformations to circuits represented by equations while evaluating cost

Examples

- **Algebraic Factoring**

  $F = \overline{A}\,\overline{C}\,\overline{D} + \overline{A}\,B\,\overline{C} + ABC + AC\overline{D}$ G = 16

  - **Factoring:**

  $F = \overline{A}\,(\overline{C}\,\overline{D} + B\overline{C}) + A\,(BC + C\overline{D}\,)$ G = 18

  - **Factoring again:**

  $F = \overline{A}\,\overline{C}(\,B + \overline{D}\,) + AC\,(B + \overline{D}\,)$ G = 12

  - **Factoring again:**

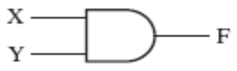
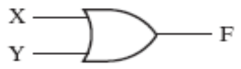
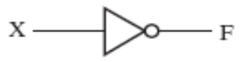
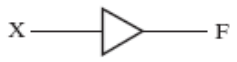
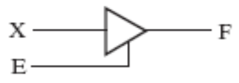
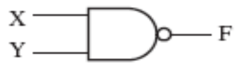
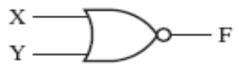
  $F = (\overline{A}\,\overline{C} + AC)\,(B + \overline{D})$ G = 10

Other Gate Types

- Why?
  - Implementation feasibility and low cost
  - Power in implementing Boolean functions
  - Convenient conceptual representation
- Gate classifications
  - Primitive gate - a gate that can be described using a single primitive operation type (AND or OR) plus an optional inversion(s).
  - Complex gate - a gate that requires more than one primitive operation type for its description

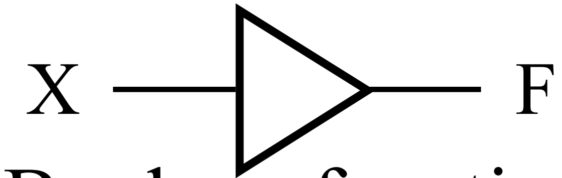
Primitive Gates

- **Symbols & truth tables for primitive gate**

Name	Distinctive shape	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
3-State Buffer			E X | F 0 0 | Hi-Z 0 1 | Hi-Z 1 0 | 0 1 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

Buffer

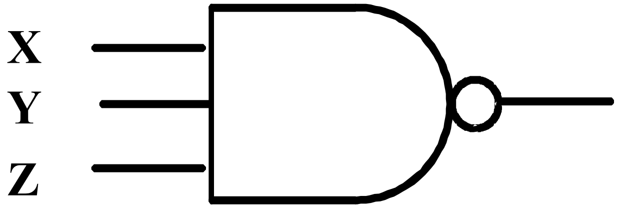
- A buffer is a gate with the function F = X:



- In terms of Boolean function, a buffer is the same as a connection!
- So why use it?
  - A buffer is an electronic amplifier used to improve circuit voltage levels and increase the speed of circuit operation.

NAND Gate

- The basic NAND gate has the following symbol, illustrated for three inputs:
  - AND-Invert (NAND)

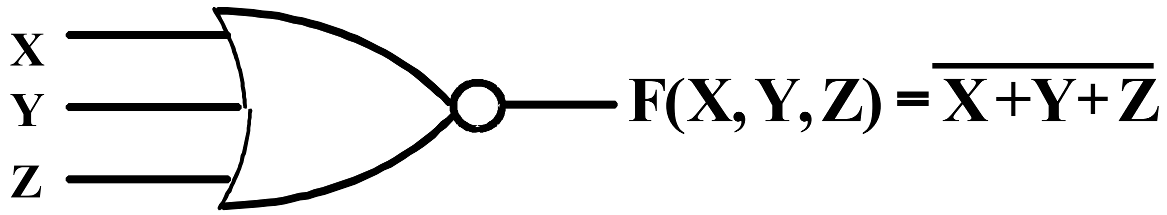


$F(X,Y,Z) = \overline{X \cdot Y \cdot Z}$

- NAND represents <u>NOT</u> <u>AND</u>, i. e., the AND function with a NOT applied. The symbol shown is an AND-Invert.   The small circle ("bubble") represents the invert function.

NOR Gate

- The basic NOR gate has the following symbol, illustrated for three inputs:
  - OR-Invert (NOR)



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

- NOR represents NOT - OR, i. e., the OR function with a NOT applied. The symbol shown is an OR-Invert. The small circle (“bubble”) represents the invert function.

The 3-State Buffer

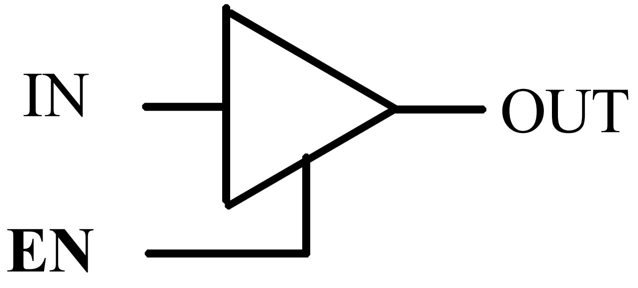
- Three-state logic adds a third logic value, Hi-Impedance (Hi-Z), giving three states: 0, 1, and Hi-Z on the outputs.
- What is a Hi-Z value?
  - The Hi-Z value behaves as an open circuit
  - This means that, looking back into the circuit, the output appears to be disconnected.
  - It is as if a switch between the internal circuitry and the output has been opened.

The 3-State Buffer

- For the symbol and truth table, IN is the data input, and EN, the control input.
- For EN = 0, regardless of the value on IN (denoted by X), the output value is Hi-Z.
- For EN = 1, the output value follows the input value.
- Variations:
  - Data input, IN, can be inverted
  - Control input, EN, can be inverted

  by addition of "bubbles" to signals.

Symbol

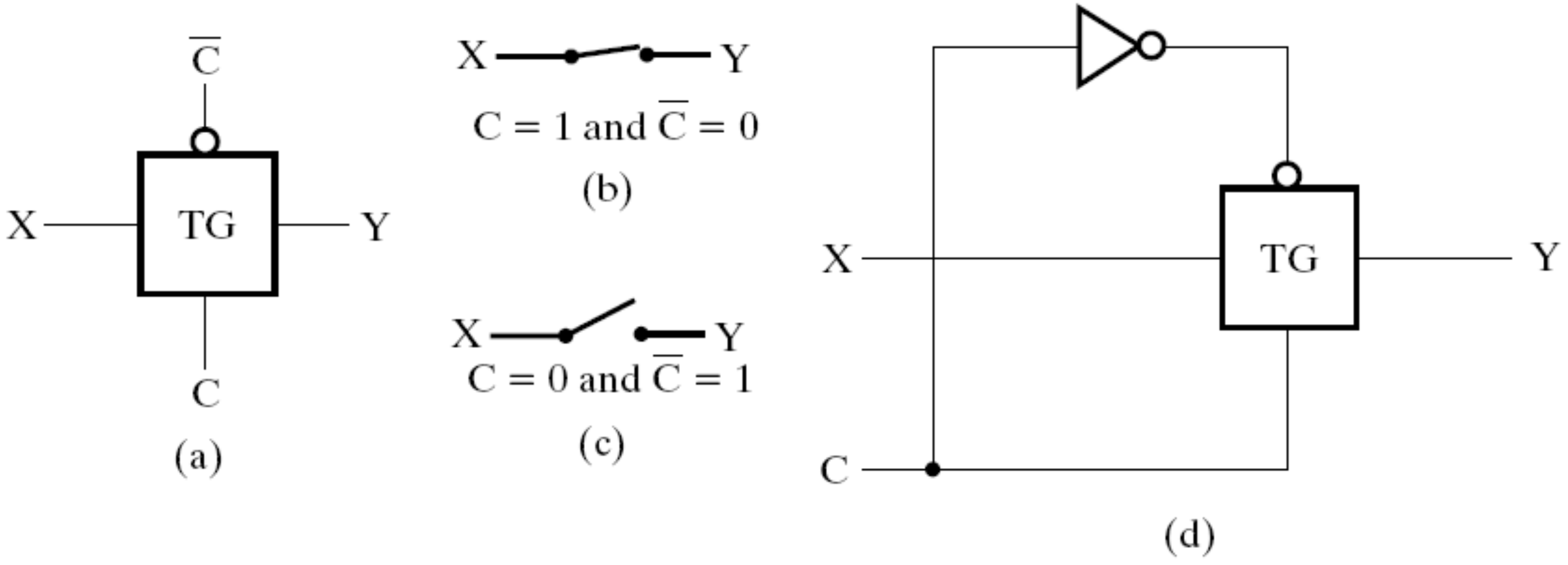


Truth Table

EN	IN	OUT
0	X	Hi-Z
1	0	0
1	1	1

Transmission Gate

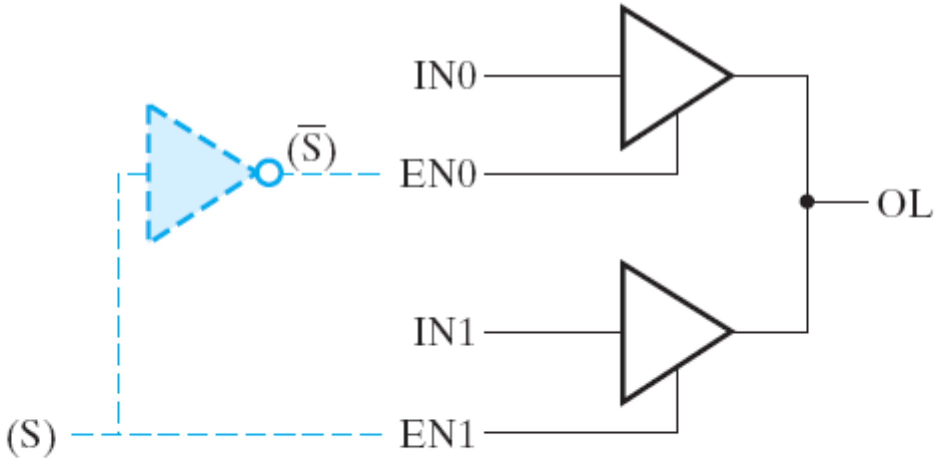
- A transmission gate can be regarded as a switch.



$\overline{C}$
X
TG
Y
C
(a)
X
Y
$C = 1$ and $\overline{C} = 0$
(b)
X
Y
$C = 0$ and $\overline{C} = 1$
(c)
X
TG
Y
C
(d)

The 3-State Buffer

- Multiplexed output line
  - Formed by 3-state buffers



(a) Logic Diagram

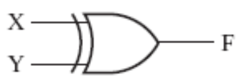
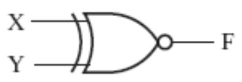
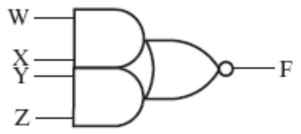
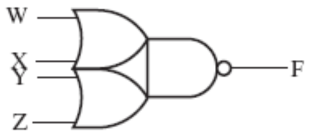
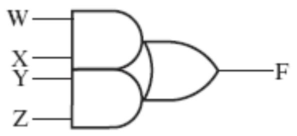
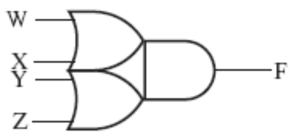
EN1	EN0	IN1	IN0	OL
0	0	X	X	Hi-Z
(S) 0	($\overline{S}$) 1	X	0	0
0	1	X	1	1
1	0	0	X	0
1	0	1	X	1
1	1	0	0	0
1	1	1	1	1
1	1	0	1	
1	1	1	0	

(b) Truth table

- Since EN0 = $\overline{S}$ and EN1 = S, one of the two buffer outputs is always Hi-Z plus the first row of the table never occurs.

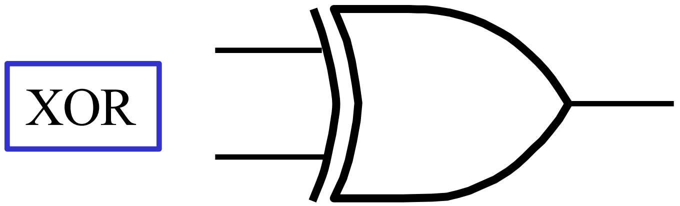
Complex Gates

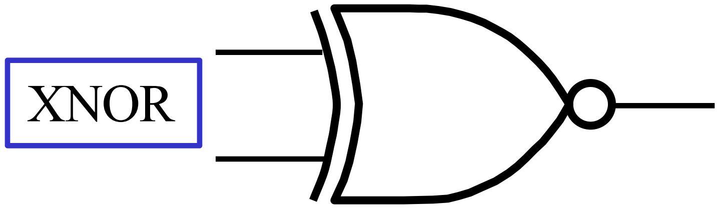
- **Symbols & truth tables for complex gates.**

Name	Distinctive shape symbol	Algebraic equation	Truth table
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
AND-OR-INVERT (AOI)		$F = \overline{WX + YZ}$	
OR-AND -INVERT (OAI)		$F = \overline{(W + X)(Y + Z)}$	
AND-OR (AO)		$F = WX + YZ$	
OR-AND (OA)		$F = (W + X)(Y + Z)$	

Exclusive OR/ Exclusive NOR

- The *eXclusive OR* (*XOR*) function is an important Boolean function used extensively in logic circuits.
- The *eXclusive NOR* function is the complement of the XOR function
- Definitions
  - The XOR function is: $\mathbf{X \oplus Y = X\overline{Y} + \overline{X}Y}$
  - The eXclusive NOR (XNOR) function, otherwise known as *equivalence* is: $\mathbf{\overline{X \oplus Y} = XY + \overline{X}\,\overline{Y}}$
- Symbols





Truth Tables for XOR/XNOR

- Operator Rules: XOR

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

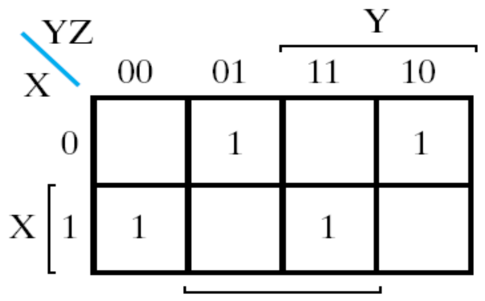
XNOR

X	Y	$\overline{(X \oplus Y)}$ or $X \equiv Y$
0	0	1
0	1	0
1	0	0
1	1	1

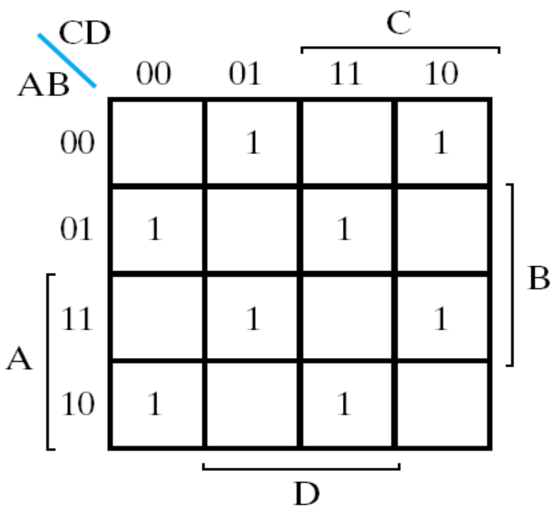
- The XOR function means:

  X OR Y, but NOT BOTH
- Why is the XNOR function also known as the *equivalence* function, denoted by the operator $\equiv$?

XOR/XNOR (Continued)

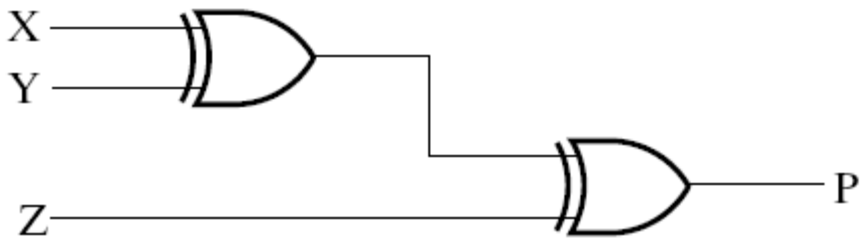


(a) $X \oplus Y \oplus Z$

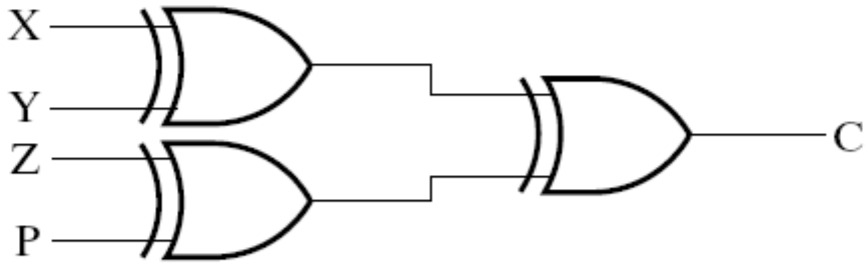


(b) $A \oplus B \oplus C \oplus D$

K-maps with odd variables



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



(b) $C = X \oplus Y \oplus Z \oplus P$

odd functions with odd inputs

Parity Generators and Checkers

Home Assignment

2-1a; 2-2a, c; 2-3a, c; 2-6b, d; 2-10a, c; 2-11a, c, d; 2-12b; 2-15; 2-17; 2-19a; 2-22a; 2-25b; 2-29; 2-30; 2-31;